GG4257 - Urban Analytics: A Toolkit for Sustainable Urban Development

Lab Assignment No 1 : Handling and GeoVisualisation of Urban Data

Cover Page

Student ID: 200024980

Module Code: GG4257

Module Title: Urban Analytics: A Toolkit for Sustainable Urban Development

Assignment: Labs 1-4

Degree Programme: International Relations and Geography

Deadline Date: Thursday, February 22nd, 2024

In submitting this assignment I hereby confirm that

I have read the University's statement on Good Academic Practice; that the following work is my own work; and that significant academic debts and borrowings have been properly acknowledged and referencd.

Introduction

This section introduces how to replicate the code and how to get the required data.

All the data required to run this notebook locally will be placed in a Google Drive folder. In addition, screenshots of essential maps and graphs will be included in this folder and accessed throughout the notebook to save processing time.

Moreover, the notebook will also be published on my Github Repo under the Assignment 1 folder. In case you would like to run this locally, I will also upload the environment file (.yml) to the repo, as well as the link to the Google Drive folder in the read me file.

In the script below, all the paths to the data assume that the file is in a folder called 'data'.

Google Drive Folder with Data

My Github Repo

Lab No 1: Python Recap: 9 Challenges

Challenge 1

Create a table with required columns and rows to represent the type of roads in the UK. The table should include headers and hyperlinks.

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
# Using a pandas DataFrame
road_type = ["Rural", "Secondary", "Primary", "Highway"] # Defining the name
road_num = [1, 2, 3, 4] # Providing roads with a unique ID
road_name = ["Largo Road", "Lamond Drive", "South Castle", "A9"] # Naming th
course_dict = {"Type": road_type, "Classification": road_num, "Name": road_r
df = pd.DataFrame(course_dict)
print (df)
```

	Type	Classification	Name
0	Rural	1	Largo Road
1	Secondary	2	Lamond Drive
2	Primary	3	South Castle
3	Highway	4	A9

Using Markdown Syntax

Number	Road Type	Road Name
1	Rural	Largo Road
2	Secondary	Lamond Drive

Number	Road Type	Road Name
3	Primary	South Castle
4	Tertiary	A9

Challenge 2

In the following cell, create the code to calculate the average of three numbers and print the result. There is no need for functions. Comment your code and the process.

In [ ]:
```
# Defining the numbers as variables
num1 = 7
num2 = 4
num3 = 17.5
#Defining a variable for the average
avg = (num1+num2+num3)/3
print(f"The average of 7, 4, and 17.5 is {avg}")
```

Challenge 3

Write a condition to check if a number is even.

In [ ]:
```
k = 6 # Feel free to change this number to test the code.
#You can use the % to ask how many times a number is divisible by another nu
if k%2 == 0:
    print(f" {k} is an even number.")
else:
    print(f"{k} is not an even number.")
```

Challenge 4

Write the python code to find the factorial of a number using a loop. Thefactorial of a number (integer positive) is the sum of multiplication of all the integers smaller than that positive integer. For example, factorial of **5** is:

5 * 4 * 3 * 2 * 1 which equals to 120.

How you can do that in a python script?, please try your best without using google.

In [ ]:
```
number = 9 # Defining the number
sum = 1 #Sum needs to be one to ensure that the factorial starts from 2.
for i in range(2, number+1): # 'i' is the loop variable
    sum *= i # Here, *= multiplies the current value of sum by the value of
print(sum)
```

Challenge 5:

Help me to write the python code, I can use to provide the grades based on the marks according to the following table:

Mark	Grade
81-100	20
61-80	15
41-60	10
20-40	5

In [ ]:
```
mark = int(input("Enter the mark:")) # We can use this to prompt the grader
if mark >= 81:
    print ("The grade is 20.")
elif 61<= mark <= 81:
    print("The grade is 15.")
elif 41 <= mark <= 60:
    print("The grade is 10.")
else:
    print("The grade is 5.")
```

Challenge 6:

- Write and use a function to convert Fahrenheit to Celsius. Use the input parameter to let the user enter a number in Fahrenheit and then get the correspondant value in Celsius
- Go to https://www.w3schools.com/python/exercise.asp?filename=exercise_functions1 and complete the 6 excersices. Functions are a very important part of Python so we will use extra time to practice that.

In [ ]:
```
tempf = int(input("What is the temperature in Fahrenheit?")) # Again, we can

# From a quick google search, the formula for calculating F to C is C = 5/9(
#Below, I'm defining the function as conversion, then adding in the variable
def conversion(tempf):
    return (5/9)*(tempf-32)
tempc = conversion(tempf)# I'm making a new variable that is the result of t
print(f"{tempf} degrees Fahrenheit is equal to {tempc} degrees Celsius.")
```

Challenge 7

Calculate the mean and standard deviation of elements in a NumPy array. The outcome should have something like `print(f"Mean: {mean_value}, Standard Deviation: {std_deviation}")`

In [ ]:
```
# First, I'm going to create a small random NumPy array using the random.ran
arr1 = np.random.randint(0, 500, size=(4,4))
```

```
print (arr1)
#Now, I'm going to create a mean value variable and use NumPy's mean functic
#then divide them by the number of values in arr1
mean_value = np.mean(arr1)

#Here, I'm going to use the NumPy standard deviation function
std_deviation = np.std(arr1)

print(f"Mean: {mean_value}, Standard Deviation: {std_deviation}")
```

Challenge 8:

You have the following dataframes 'sales_data' and 'movie_ratings'

In [ ]:
```
import pandas as pd

sales_data_dict = {
    'Date': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04'],
    'Region': ['North', 'South', 'East', 'West'],
    'Product': ['A', 'B', 'A', 'C'],
    'Quantity': [15, 8, 12, 10],
    'Revenue': [1500, 1200, 1800, 900]
}

sales_data = pd.DataFrame(sales_data_dict)


movie_ratings_dict = {
    'Title': ['Movie1', 'Movie2', 'Movie3', 'Movie4'],
    'Genre': ['Action', 'Drama', 'Comedy', 'Thriller'],
    'Year': [2015, 2010, 2018, 2011],
    'Rating': [8.5, 7.8, 9.2, 8.0]
}

movie_ratings = pd.DataFrame(movie_ratings_dict)

print(sales_data)
print(movie_ratings)
```

8.1 Display the first 5 rows of the DataFrame 'sales_data'

In [ ]:
```
#Displaying the first five rows using .head
sales_data.head(5)
```

8.2 Use loc to filter and display rows where the "Quantity" is greater than 10 and the "Region" is "North".

In [ ]:
```
#Using 'loc' to filter and display rows. To filter by a string ("North"), I
filtered_sales_data = sales_data.loc[(sales_data["Quantity"] > 10) & (sales_
print (filtered_sales_data)
```

8.3 Use iloc to display the values in the first row and the first three columns.

In [ ]:
```
#Using 'iloc' to display the values in the first row and first three columns
# The colon can be used to specify all the other rows before 3 to avoid havi
first_values = sales_data.iloc[0, :3]
first_values
```

8.4 Calculate the total sales for each region and display the result.

Assuming that total sales means the total revenue generated by each region.

In [ ]:
```
# Calculating the total sales for each region
# First, I'm going to filter the sales data set by each region using loc.
north_sales = sales_data.loc[sales_data["Region"] == "North"]
south_sales = sales_data.loc[sales_data["Region"] == "South"]
east_sales = sales_data.loc[sales_data["Region"] == "East"]
west_sales = sales_data.loc[sales_data["Region"] == "West"]
# Next, I'm going to use the sum () tool to find the sum of the revenue colu
north_revenue = north_sales["Revenue"].sum()
south_revenue = south_sales["Revenue"].sum()
east_revenue = east_sales["Revenue"].sum()
west_revenue = west_sales["Revenue"].sum()
# Number of units sold?
print (f"North: {north_revenue}, South: {south_revenue}, East: {east_revenue
```

In [ ]:
```
# This process was long and labor intensive. From this documentation: https:
# I can use groupby to print the total sales by region and revenue and then
total_sales_per_region = sales_data.groupby('Region')['Revenue'].sum()
print(total_sales_per_region)
```

8.5 Display the last 3 rows of the DataFrame 'movie_ratings'.

In [ ]:
```
# I can use 'tail' to do so.
movie_ratings.tail(3)
```

8.6 Calculate and display the average rating for each genre.

In [ ]:
```
# I could do this in the same way as above with the revenue, by creating sep
# to find the .groupby tool, which will allow me to group genre and ratings,
avg_ratings_by_genre = movie_ratings.groupby("Genre")["Rating"].mean()
avg_ratings_by_genre
```

Challenge 9:

- Recreate the explore map
- Get the histogram of the raster you plot in the exercise

In [ ]:
```
import geopandas as gpd
import rasterio
from rasterio.plot import show

shapefile_path = 'data/Road_Safety_Accidents.shp'
gdf = gpd.read_file(shapefile_path)
```

In [ ]:
```
raster_path = 'data/elev.tif' # These are both lines that we ran during the
raster = rasterio.open(raster_path)
```

In [ ]:
```
# To get a sense of the data, I need to use .columns
gdf.columns
```

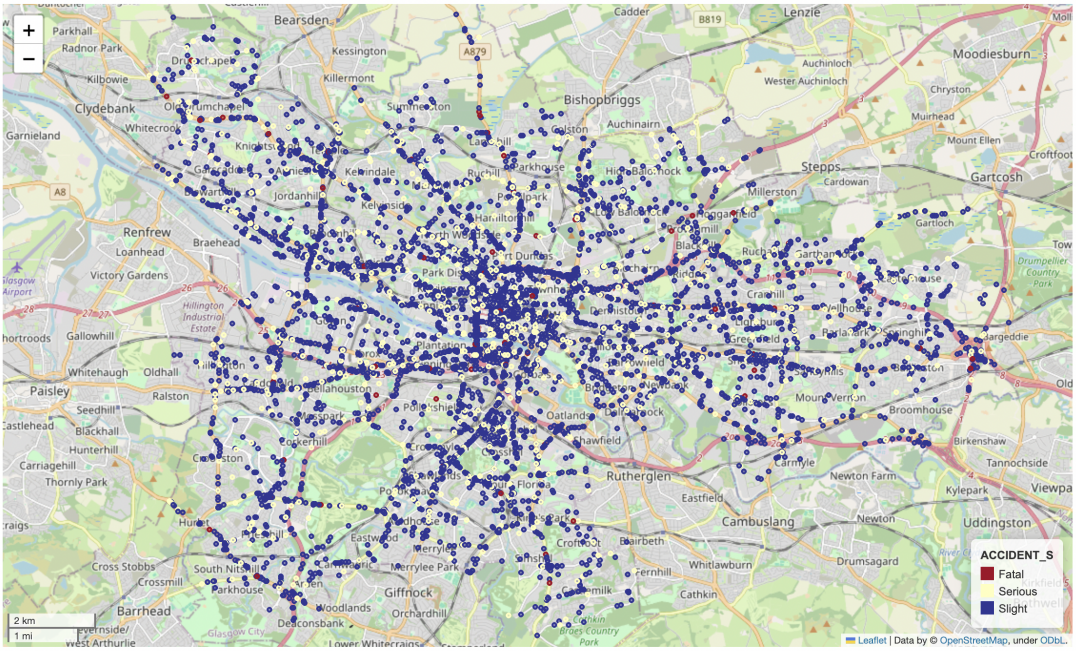
9.1 Explore with Colors

In [ ]:
```
# Adding a CRS so that the baselayer loads in
gdf.crs=('EPSG:4326')
```

In [ ]:
```
# The code below will assign the accident seriousness column a color based c
gdf.explore(column='ACCIDENT_S', cmap='RdYlBu', legend=True) # I also added
```

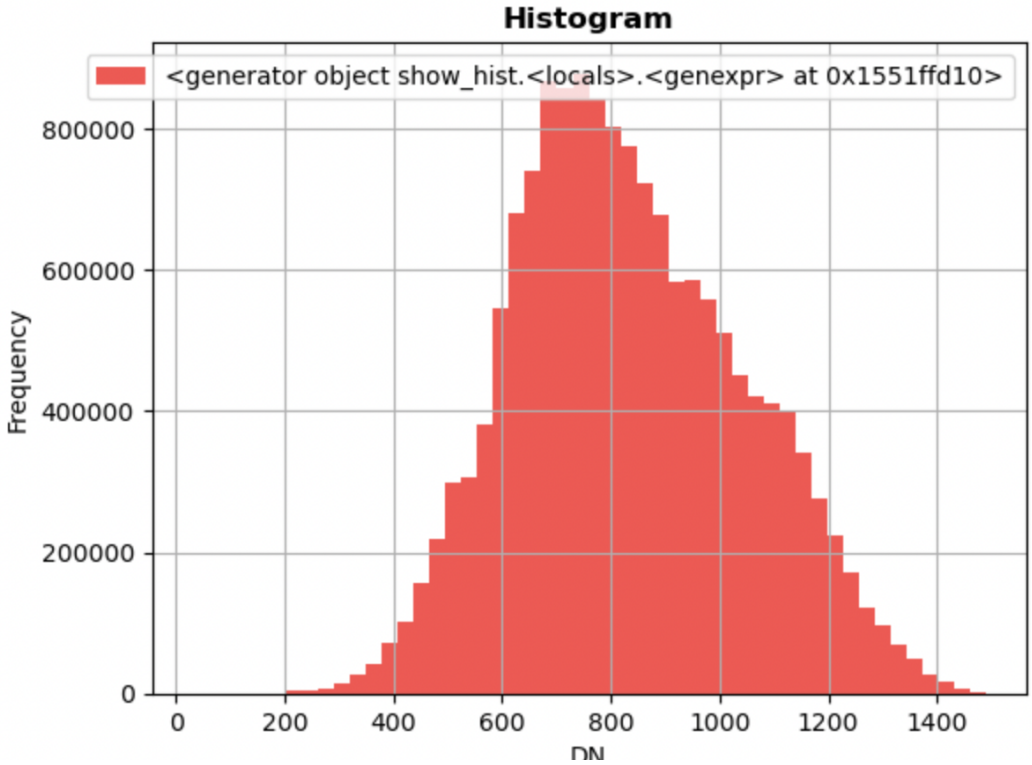
[37]:
```
# The code below will assign the accident seriousness column a color based on the GeoDataFrame documentation.
gdf.explore(column='ACCIDENT_S', cmap='RdYlBu', legend=True) # I also added a legend
```

[37]:



9.2 Histogram

In [ ]:
```
# In order to plot the histogram, we need show_hist from rasterio.plot
from matplotlib import pyplot
from rasterio.plot import show_hist
# Here, I've added some more information to title the histogram and make it
show_hist(raster, bins=50, lw=0.0, stacked=False, alpha=0.7, histtype="stepf
```



Lab No 2: Data Manipulation and Working with Web Services: 2 Challenges

Challenge 1

1.1 Using a Dictionary, create a dataframe (table), with at least 4 columns and more than 100 rows.

In [ ]:
```
#From the pandas documentation, I can use random.randint() to make random in
age = range(0, 100) #creating a list of consecutive ages from 0 to 100
region = ["North", "South", "East", "West"] #Creating regions
a = np.repeat(region, 25) #this uses numpy's repeat() function to repeat val
chips = np.random.randint(0, 50, 100) # 0 and 50 mean that the numbers of co
cookies = np.random.randint(0, 67, 100)
apples = np.random.randint(0, 6, 100)
candy = np.random.randint(0, 20, 100)
# Creating the df
sweets = pd.DataFrame({'age':age,'region':a, 'cookies':cookies, 'chips':chip
sweets
```

1.2 Using the appropriate method, create a new DataFrame containing only the first 30 rows and the first 3 columns of the original DataFrame. Name this new DataFrame subset_df.

In [ ]:
```
#Using 'iloc' to display the values in the first 30 rows and first three col
subset_df = sweets.iloc[:30, :3]
print(subset_df)
```

1.3 Using the appropriate method, filter the rows from the original dataframe where a numerical attribute(column) is greater than a particular numerical value, and find another categorical attribute that is equal to a specific string or text. Name this new DataFrame filtered_df.

In [ ]:
```
# I can use .loc to filter rows. I can also use == to specify a specific str
filtered_df = sweets.loc[(sweets["cookies"] > 50) & (sweets["region"] == "No
print (filtered_df)
```

1.4 Check this website https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.html and apply the methods, mean, standard deviation, group_by to run fundamental statistical analysis of your created data frame.

In [ ]:
```
# I can use .mean. However, because I have specified the region, I needed to
sweets.mean(numeric_only=True) # Sweets is the original df.
```

In [ ]:
```
# Standard Dev
sweets.std(numeric_only=True)
```

In [ ]:
```
# I can use groupby to compare categorical and numerical data in a comparisc
# reset_index is important because it avoids making the resulting dataframe
Sum_of_cookies_by_region = sweets.groupby('region')['cookies'].mean().reset_
Sum_of_cookies_by_region
```

Challenge 2

Part No 1:

1. Using the same workflow previously described, now calculate the clustered areas for the GeoPandasDataFrame `gdf_bikes_end`
2. Make sure you don't have any NaN in your columns, add a CRS, clean up the unnecessary attributes, calculate the cluster values, and plot a map of 4 calculated clusters for the return locations.

Part No 2:

1. Using the Glasglow Open Data API (Transit) https://developer.glasgow.gov.uk/api-details#api=traffic&operation=traffic-sensor-locations fetch all the sensor locations in the city.
2. Map the sensor

3. Find the WorkingZones and Calculate/Map the areas with more and fewer sensors distributed in the city.
4. You will need:
    - Get two separate Geopandas DataFrames, one for the traffic sensors and another one for the WorkingZones.
    - Using `sJoin` (Spatial Join) https://geopandas.org/en/stable/docs/reference/api/geopandas.sjoin.html calculate the overlay of sensors and polygons.
    - Using group_by https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html to count the number of sensors per WorkingZone
    - Make sure you add the counts into the WorkingZone polygons of Glasgow so you can create a map of Zones with more and fewer traffic sensors.
    - Of course, you will need extra steps where you manipulate the data and extract what you need, for instance, clipping the Working Zones only for Glasgow.
5. Make sure you comment on your code and describe how you are manipulating the data.

Part 1: Calculating the clustered areas for the GeoPandasDataFrame "gdf_bikes_end"

1.1 Getting the data from an API

In [ ]:
```
import requests
import pandas as pd
import geopandas as gpd

# Let's describe the url
url_bikes = "https://api.glasgow.gov.uk/mobility/v1/get_rentals?startDate=20
# Making the query to the web server, using the Get method from the requests
response = requests.get(url_bikes)
response
```

In [ ]:
```
data = response.json()
rental_data = data['data']
rental_data # Great! Looks like this data is not nested
```

1.2 Conversion to a pandas dataframe

In [ ]:
```
rental_pd = pd.DataFrame(rental_data)
rental_pd.head()
```

In [ ]:
```
# Cleaning the data by checking for NaN in the coordinates column
nan_in_column_Lat = rental_pd['endPlaceLat'].isna().any()
nan_in_column_Long = rental_pd['endPlaceLong'].isna().any()
```

```
print(nan_in_column_Lat,nan_in_column_Long) # Looks like there are some NaN

# Using .dropna
clean_rental_pd = rental_pd.dropna(subset=['startPlaceLat', 'startPlaceLong'
clean_rental_pd.info()
```

1.3 Creating a GeoDataFrame of the End Points of Bike Journeys

In [ ]:
```
import geopandas as gpd
gdf_bikes_end = gpd.GeoDataFrame(clean_rental_pd, geometry=gpd.points_from_x
# I've generated a point geometry from the end lat and long data. I've also
```

In [ ]:
```
gdf_bikes_end.explore()
```

1.4 Cleaning the Unnecessary Columns and the Dtypes

In [ ]:
```
# We can use keep_cols to cleanup unnecessary columns by choosing which ones
keep_cols = [
    "endDate",
    "endPlaceId",
    "endPlaceName",
    "durationSeconds",
    "isInvalid",
    "price",
    "isEbike",
    'endPlaceLong',
    'endPlaceLat',
    "geometry", # Because we've converted it to a geodataframe based on the
]
gdf_bikes_end = gdf_bikes_end[keep_cols]
gdf_bikes_end.head()
```

In [ ]:
```
# Seeing the types of data
gdf_bikes_end.dtypes
```

In [ ]:
```
#The first three variables are objects, when they should be int, str, and da
gdf_bikes_end.endPlaceId = gdf_bikes_end.endPlaceId.astype(int)
gdf_bikes_end.endPlaceName = gdf_bikes_end.endPlaceName.astype(str)
gdf_bikes_end['endDate'] = pd.to_datetime(gdf_bikes_end['endDate'], format='
```

In [ ]:
```
# Before we progress, we need to double check there are no NaN values
nan_in_column_Lat = gdf_bikes_end['endPlaceLat'].isna().any()
nan_in_column_Long = gdf_bikes_end['endPlaceLong'].isna().any()

print(nan_in_column_Lat,nan_in_column_Long)
```

1.5 KMeans and Leafmap: Clusters

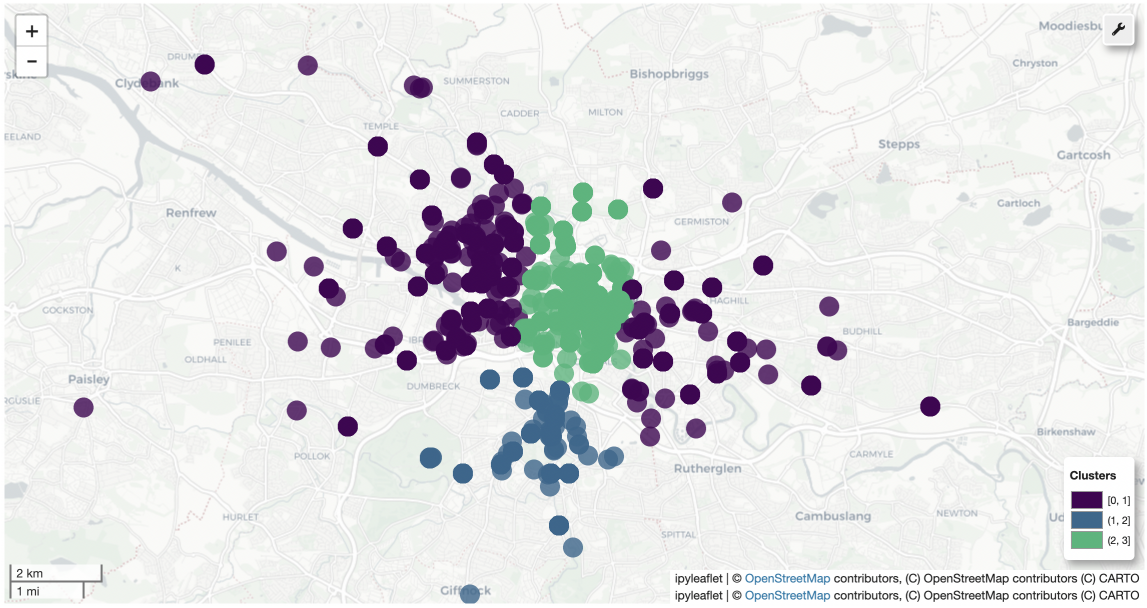
In [ ]:
```
# Now we can use KMeans to calculate the clusters and plot a map of 4 cluste
# First we need to get the lat and long values in a way KMeans can recognise
```

```
from sklearn.cluster import KMeans
num_clusters = 4 #Defining the number of clusters I want

kmeans_collection = KMeans(n_clusters=num_clusters, random_state=42)
gdf_bikes_end['kmeans_cluster'] = kmeans_collection.fit_predict(gdf_bikes_en
```

In [ ]:

```
import leafmap # Leafmap is a package for interactive mapping and geospatial
m = leafmap.Map() # Calling the .Map tool

m = leafmap.Map(center=(55.860166, -4.257505), #These coordinates will cente
                zoom=12, # This defines the scope of the map when initally l
                draw_control=False,
                measure_control=False,
                fullscreen_control=False,
                attribution_control=True,
               )

m.add_basemap("CartoDB.Positron") # Adding CartoDB as the basemap
m.add_data(
    gdf_bikes_end,
    column='kmeans_cluster',
    legend_title='Clusters',
    cmap='viridis',
    k=4,
)

#Plotting the map
m # Now, we have four different clusters of the end journeys of rental bikes
```

Here's the screenshot of the clusters of end journeys



Part 2: Mapping Traffic Sensors in Glasgow

2.1 Getting all the sensor locations in Glasgow

The city of Glasgow provides a real time traffic API allowing for queries of current traffic flows, congestion, and location queries as well as the ability to query over a given time period. They do this with sensors. The challenge is to map the number of sensors by Working Zones in Glasgow to understand which zones have more sensors than others.

In [ ]:
```
#Just running all of the libraries
import pandas as pd
import numpy as np
import leafmap
from sklearn.cluster import KMeans
import urllib.request
#%matplotlib inline
from shapely.geometry import Point
import requests
import pandas as pd
import geopandas as gpd
```

In [ ]:
```
# Describing the url
url_sensor = "https://api.glasgow.gov.uk/traffic/v1/movement/sites"
# Making the query to the web server, using the Get method from the requests
response = requests.get(url_sensor)
response
```

2.2 Mapping the sensors

Now, the data is in a json format. It needs to be converted to a geopandas dataframe.

In [ ]:
```
#Making sure the json is something readable
data = response.json()
data
# I copied this into my browser and used a json viewer extension.
# It looks like it's under the siteID column and then further nested under t
```

In [ ]:
```
 # As we can see from the above, data is still nested under the from and to
# I just need the from column to be expanded. I can do this using iterative
# Firstly, I need to create empty columns to store the data under the from c
siteId_list = []
lat_list = []
long_list = []
description_list = []

# Now, I need to create a for loop to go over each of the entries in the jso
for i in data: # Defining where I want to entries to come from
    siteId = i['siteId']
    lat = i['from']['lat'] # I need to first go into the from column and the
    long = i['from']['long'] # Same again.
    description = i['from']['description']

# Now, I need to add these values to the empty lists I created earlier
    siteId_list.append(siteId)
    lat_list.append(lat)
    long_list.append(long)
```

```
    description_list.append(description)

# So at this point, my values are added to the empty lists. Therefore, I am
sensors_df = pd.DataFrame({ # I have to define the values to go in each colu
    'siteId': siteId_list,
    'lat': lat_list,
    'long': long_list,
    'description': description_list,
})
```

In [ ]:
```
sensors_df.head()
```

In [ ]:
```
# Need to check for NaN values in the lat and long columns
nan_in_column_lat = sensors_df['lat'].isna().any()
nan_in_column_long = sensors_df['long'].isna().any()
print(nan_in_column_lat,nan_in_column_lat)
```

In [ ]:
```
gdf_sensors = gpd.GeoDataFrame(sensors_df, geometry=gpd.points_from_xy(sensc
# gdf_sensors.explore() # Feel free to remove the hashtag to view this.
# It seems as though there are some errors in the coordinate data that are r
# once we clip the bounds of the data to the city of Glasgow.
```

2.3 Getting the Working Zones in Scotland

In [ ]:
```
import geopandas as gpd

# Reading the working zones shapefile.
shapefile_path = "data/WorkplaceZones2011Scotland/WorkplaceZones2011Scotlanc
gdf_wz = gpd.read_file(shapefile_path)
```

In [ ]:
```
#gdf_wz.explore() # This may take some time to load. Feel free to remove the
```

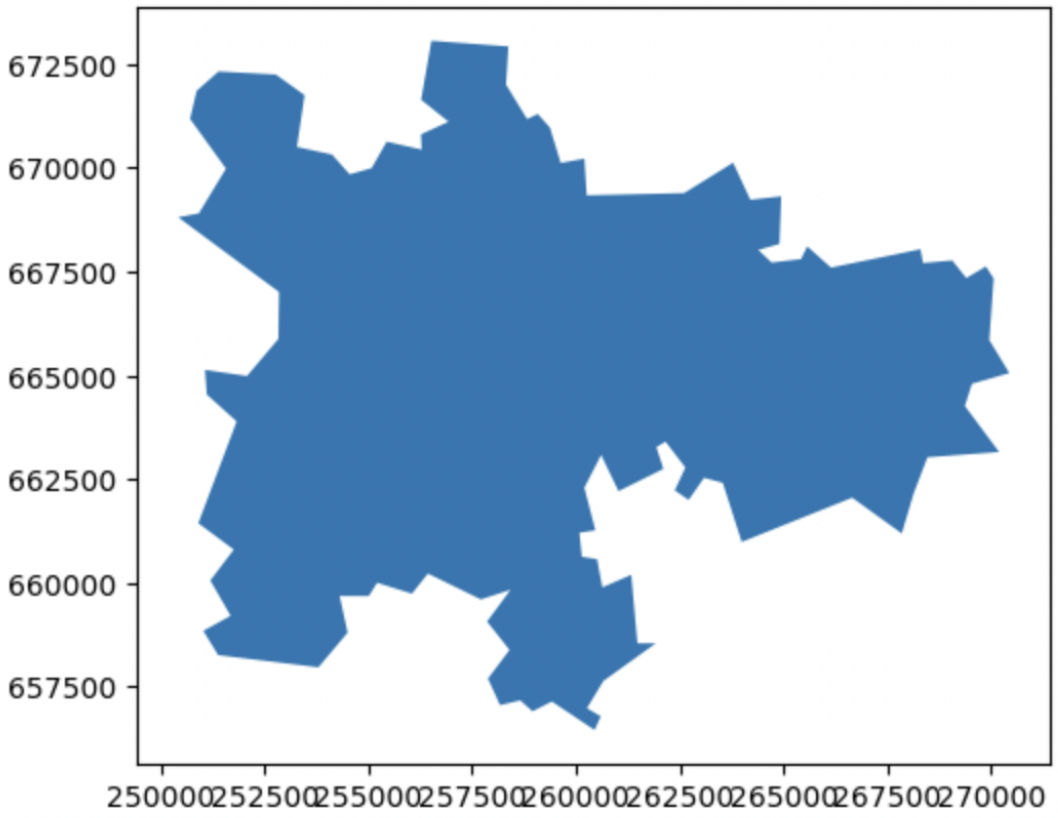
This map is displaying the working zones for the whole of Scotland. Now, I need isolate the working zones in Glasgow. I first need to define the boundaries of Glasgow in order to do so.

In [ ]:
```
# From the data provided, here is a map of all the Local Planning Authoritie
UK_LPA = "data/LAP_2021/LPA_MAY_2021_UK_BUC_V2.shp"
gdf_UK_LPA = gpd.read_file(UK_LPA) # Reading it into a geodataframe from the
```

In [ ]:
```
# gdf_UK_LPA.explore() # Same issue as above; these are all the LPAs for the
```

In [ ]:
```
# From the explore function above, we can see that the LPA21NM column with t
# Therefore, the line below will plot just the city based on the Local Plann
gdf_UK_LPA[gdf_UK_LPA.LPA21NM == "Glasgow City LPA"].plot()
```

```
<Axes: >
```



Here is the screenshot of the plot of Glasgow's LPA.

In [ ]:
```
# Now, I need to make a subset of the LPA shapefile that only includes the g
glasgow_lpa_gdf = gdf_UK_LPA[gdf_UK_LPA.LPA21NM == "Glasgow City LPA"]
```

In [ ]:
```
glasgow_lpa_gdf.explore() # This is the outline of glasgow city.
```

In [ ]:
```
# At this point, we have the glasgow city boundary limits, as defined by gla
# which is defined as gdf_wz.
# Based on the geopandas documentation:(https://geopandas.org/en/stable/docs
glasgow_wz = gpd.clip(gdf_wz, glasgow_lpa_gdf)
```

In [ ]:
```
glasgow_wz.explore() # This map is all the working zones within Glasgow
```

2.4 Using sJoin to calculate the overlay of sensors and polygons

In [ ]:
```
# This line is just to make sure the crs of the two shapefiles is aligned. E
glasgow_wz = glasgow_wz.to_crs(gdf_sensors.crs)
```

In [ ]:
```
#From the online documentation, 'gdf_sensors.sindex.valid_query_predicates'
glasgow_wz.sindex.valid_query_predicates
```

In [ ]:
```
joined_data = gpd.sjoin(glasgow_wz, gdf_sensors, how='left', predicate='inte
```

```
# Fill missing values in the index_right that was just created with 0 - to k
joined_data['index_right'].fillna(0, inplace=True)
# Now joined_data will contain all polygons from glasgow_wz with an index_ri
joined_data
```

In [ ]:
```
joined_data.explore() # Quickly viewing the data to make sure that all of Gl
```

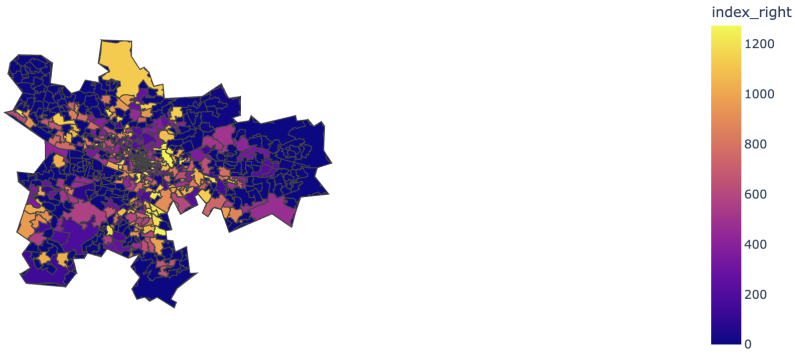
In [ ]:
```
sensors_in_wz_grouped = joined_data.groupby('WZCD').size().reset_index(name=
sensors_in_wz_grouped # This has counted the number of sensors for each WZ.
```

In [ ]:
```
# At this point, we now need to merge the counts back into the original data
sensors_in_wz = pd.merge(joined_data, sensors_in_wz_grouped, left_on="WZCD",
```

2.5 Creating a Choropleth Map of Bike End Journeys

In [ ]:
```
# I can use plotly to create a choropleth map
import plotly.express as px
fig = px.choropleth(sensors_in_wz,
                    geojson=sensors_in_wz.geometry,
                    locations=sensors_in_wz.index,
                    color="index_right",
                    projection="mercator",
                   )
fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Here is a screenshot of the choropleth map



This graph shows the working zones in Glasgow that have the highest and lowest number of end journeys for rental bikes (represented as yellow and blue respectively). It looks as though the most common areas to end a bike journey are in the city centre as well as a working zone in the north. This might reflect commuters returning home using the rental bike scheme, for example. Residential areas on the outskirts of the city, however, tend not to be areas where people end bike journeys, which might reflect private car use or other methods of public transportation.

Lab No 3: Geovisualization Techniques - Data Viz - Part 1: 4 Challenges

Challenge 1

What happens if you have non-numerical attributes?

Extending a `data_description` function to only accept numerical columns and calculate mean and counts. The outcome should be a table with Mean and Counts per Column.

In [ ]:
```
import pandas as pd

listings =pd.read_csv("data/listings.csv")
# Subset data
subset_listings = listings[['id',
                            'neighbourhood_cleansed',
                            'latitude',
                            'longitude',
                            'property_type',
                            'room_type',
                            'bedrooms',
                            'price',
                            'number_of_reviews']]

# We need to do some cleaning...There are some weird characters $ and , then
# I use multiple functions in one line, but the key is using .loc and replac
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFram
subset_listings.loc[subset_listings.index, 'price'] = subset_listings['price

# Remove any records that are not complete
subset_listings = subset_listings.dropna()
```

In [ ]:
```
subset_listings.dtypes
```

In [ ]:
```
# Looks like price is a string, not a number.
subset_listings.price = subset_listings.price.astype(int)
```

1.1 Function for Non-Numerical Descriptive Statistics

In [ ]:
```
import pandas as pd

def data_description(x): # X is a stand-in for the dataframe
    stats = {} # Creating an empty dictionary to store the statistics for th
    non_numerical_columns = x.select_dtypes(exclude='number').columns # Usin
    for col in non_numerical_columns: #Creating the for loop to go over each
        stats[col] = {'Mode': x[col].mode(), 'Counts': x[col].count()} # For
    return stats # Returning the dictionary with stats for non-numerical col
```

```
result = data_description(subset_listings)
result # However, this is a dictionary, not a pandas dataframe.
```

In [ ]:
```
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.from_dict.ht
# From the documentation provided above, I can use DataFrame.from_dict to co
pd.DataFrame.from_dict(result, orient='index')
```

	Mode	Counts
neighbourhood_cleansed	0 De Baarsjes - Oud-West Name: neighbourhoo...	13837
property_type	0 Apartment Name: property_type, dtype: object	13837
room_type	0 Entire home/apt Name: room_type, dtype: o...	13837

1.2 Functions for Numerical Descriptive Statistics

Same process for non-numerical values:

In [ ]:
```
def data_description(x): # X is a stand-in for the dataframe
    stats = {} # Creating an empty dictionary to store the statistics for th
    numerical_columns = x.select_dtypes(exclude='object').columns # Using .s
    for col in numerical_columns: #Creating the for loop to go over each col
        stats[col] = {'Mean': x[col].mean(), 'Counts': x[col].count()}
    return stats # Returning the dictionary with stats for numerical columns

result = data_description(subset_listings)# However, as above, this is a dic
pd.DataFrame.from_dict(result, orient='index').round(2)
# I can round the values to two decimal points based on this documentation:
```

	Mean	Counts
id	7418371.14	13837
latitude	52.37	13837
longitude	4.89	13837
bedrooms	1.43	13837
price	132.83	13837
number_of_reviews	15.42	13837

Challenge 2

Now is your turn to find, read, process and then make a comprehensive descriptive statistics analysis based on the previous resources and others you might need to look at. Create insightful visualizations combining both maps and charts to convey meaningful information about a chosen city.

1. Define a problem within the urban environment and choose a dataset related to urban life or city dynamics. This could include data on crime rates, housing prices, transportation, demographics, or any other urban-related dataset.
2. Get the data ideally using an **API, or web services**. But it's fine if you need to download the data. Describe why you had to use the traditional method.
3. Work with the data cleaning and pre-processing, check for missing values, convert data types, and perform any other necessary preprocessing steps. You have the code for that in this and previous labs.
4. Use Pandas to calculate descriptive statistics such as `mean`, `median`, `standard deviation`, and other relevant measures. Explore `correlations` between different variables (Include at least one `univariate` and `bivariate` plots)
5. Create at least one (or more) interactive map to visualize spatial aspects of the data. For example, plot crime rates across different neighbourhoods or visualize housing prices.
6. Complement with additional charts (line charts, bar charts, etc.) to extend the map and highlight key trends or patterns in the data.

7. As always, document well what you are doing and how you use `descriptive statistics and map visualizations` to extract insights from the data.

> You could try explaining any observed patterns, trends, or correlations, whether they are spatial or non-spatial. **Keep in mind** the defined problem and whether your analysis provides the required insights (Your conclusion could be that you need more data or another type of analysis)

2.1 Defining a problem: Crime Rates and Transportation Hubs in Washington D.C.

Washington D.C. has the highest crime rate per 100,000 people in the United States.

Additionally, many people in the city are reliant on transportation (particularly the underground train, or the Metro) to commute to work. Other studies have linked crime rates to transportation hubs, as linked below.

1. The spatial relationship between crime and public transportation: A geospatial analysis of Salt Lake City's TRAX system
2. Crime and Public Transportation: A Case Study of Ottawa's O-Train System

Irvin-Erikson and La Vigne, in a spatio-temporal analysis of crime around Washington's metro stations, found that crimes at stations are affected by the immediate and larger environment in which the station is housed in. Different crimes are more likely to happen at stations with certain characteristics, such as parking availability and remoteness at particular times.

Therefore, further exploring the relationship between transportation hubs and crime rates would provide insights into how crime is spatially distributed, particularly in a place like DC that has seen rapid gentrification and an influx of white, upper-middle class workers into the city for government and related work. Therefore, this analysis will focus on the relationship between crime incidents in 2022 near Metro Stations in DC. It will determine the correlation between size of the station and the number of crimes reported within a certain buffer distance, and then assess the stations that have the highest densities of crimes within 200m.

2.2 Getting the data

The city of Washington D.C. publishes APIs and data here: https://opendata.dc.gov/. However, using an API requires an authenticated app token, which I do not have access to. This meant I could only access the first 1000 rows of data even though there were 27,000 total rows of data. Therefore, I had to download the data into a csv. file to access this. Even so, I've included both processes below; the first is reading the data from an API (and then transforming into a pd), and the second is from the downloaded data in

csv form. If you'd like to test the code with a small amount of data, simply using the API will work too.

1. From an API

In [ ]:
```
import requests
import pandas as pd
import geopandas as gpd

# Describing the url for the Crime Incidents in 2022 dataset
url_crime = "https://maps2.dcgis.dc.gov/dcgis/rest/services/FEEDS/MPD/MapSer
# Making the query to the web server, using the Get method from the requests
response = requests.get(url_crime)
response
```

In [ ]:
```
data = response.json()
data # I put this into my browser's json viewer. Data is under the features
```

In [ ]:
```
# Similar to the metadata /data structure, the data here that I need is unde
crime_data = data['features']
crime_data # Now, I just need to call properties to then access the lat, lon
```

Based on the same process of creating loops to iterate over each list earlier, I can use loops to change the structure of the data.

In [ ]:
```
# Firstly, I need to create empty columns to store the data under the from c
SHIFT_list = []
OFFENSE_list = []
WARD_list = []
lat_list = []
long_list =[]
```

In [ ]:
```
# Now, I need to create a 'for' loop to go over each of the entries in the j
for index in crime_data: # Defining where I want to entries to come from
    OFFENSE = index['attributes']['OFFENSE']
    WARD = index['attributes']['WARD']
    SHIFT = index['attributes']['SHIFT']
    lat = index['attributes']['LATITUDE']
    long = index['attributes']['LONGITUDE']

# Now, I need to add these values to the empty lists I created earlier
    SHIFT_list.append(SHIFT)
    OFFENSE_list.append(OFFENSE)
    WARD_list.append(WARD)
    lat_list.append(lat)
    long_list.append(long)

# So at this point, my values are added to the empty lists. Therefore, I am
crime_pd_from_json = pd.DataFrame({ # I have to define the values to go in e
    'SHIFT': SHIFT_list,
    'OFFENSE': OFFENSE_list,
    'WARD': WARD_list,
    'lat': lat_list,
```

```
    'long': long_list,
})
```

In [ ]:
```
crime_pd_from_json.info()
```

As you can see, there are only 1000 entries. Therefore, I had to download the data into a csv and load it the traditional way. I've placed the csv in the previously linked google drive folder with data, but you can also download it on the portal itself.

2. From a CSV

In [ ]:
```
# Here's the traditional CSV method.
crime_pd = pd.read_csv("data/Crime_Incidents_in_2022.csv")
```

In [ ]:
```
crime_pd.info() # As you can see, there are over 27000 entries.
```

In [ ]:
```
crime_pd.head()
```

In [ ]:
```
# I can use keep_cols to define the columns that are necessary. For these pu
keep_cols = [
    "WARD",
    "SHIFT",
    "OFFENSE",
    "X",
    "Y",
]
crime_pd = crime_pd[keep_cols]
crime_pd.head()
```

2.3 Data Cleaning and Pre-Processing

1. Transforming to a gdf

In [ ]:
```
import geopandas as gpd
crime_gdf = gpd.GeoDataFrame(crime_pd, geometry=gpd.points_from_xy(crime_pd[
crime_gdf = crime_gdf.drop('X', axis=1)
crime_gdf = crime_gdf.drop('Y', axis=1)# We should also drop the now unneces
crime_gdf.info()

crime_gdf.head()
```

2. Checking for missing values

In [ ]:
```
# The line below will return if there are any NaN values.
nan_in_columns = crime_gdf.isna().any()
print(nan_in_columns)
```

In [ ]:
```
# We can drop these NaN values
crime_gdf = crime_gdf.dropna(subset=['WARD'])
```

```
nan_in_columns = crime_gdf.isna().any()
print (nan_in_columns)
```

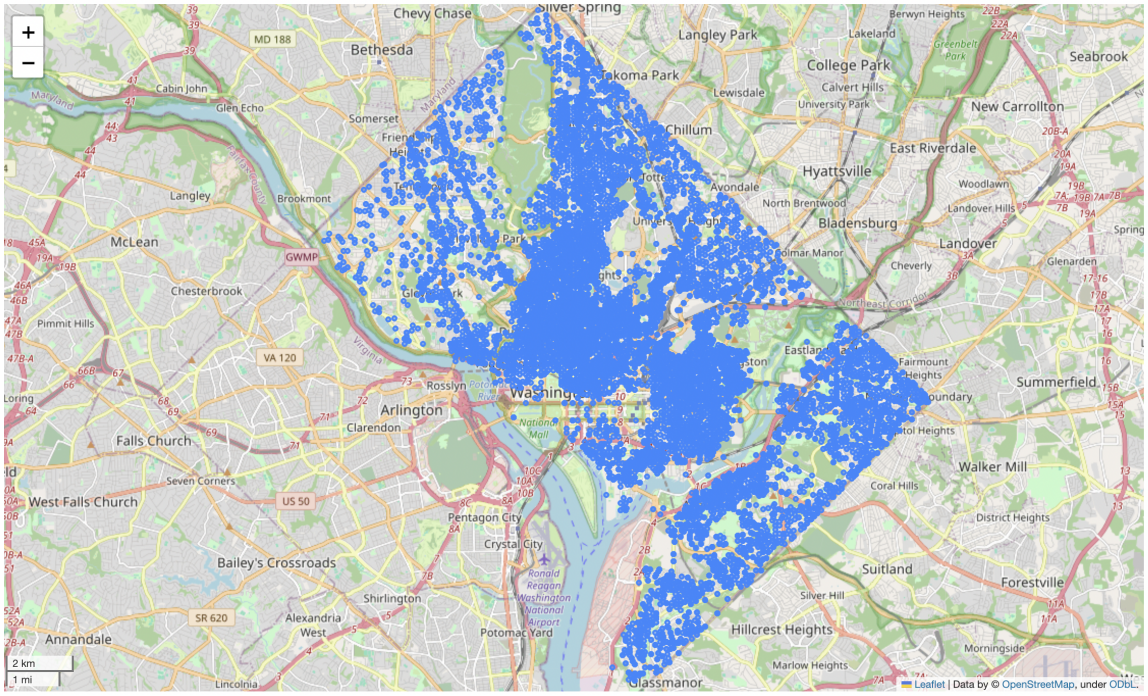
3. Converting data types

In [ ]:
```
crime_gdf.dtypes
```

In [ ]:
```
# the lat and long correctly are floats. However, the WARD should be a numbe
crime_gdf.WARD = crime_gdf.WARD.astype(int)
```

4. Visualising the crime data

In [ ]:
```
crime_gdf.explore()
```



2.4 Data Analysis for Crimes in DC

1. Univariate Analysis

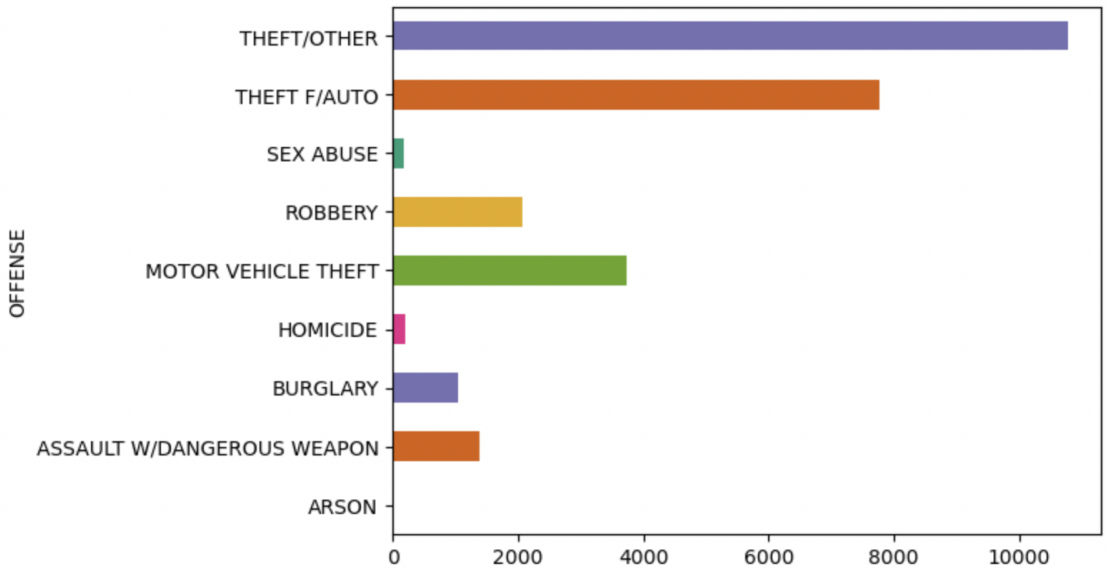
There are three ways to perform univariate analysis:

- Summary Statistics
- Frequency Distributions Table
- Charts (Boxplot, Histogram, Barplot, Pie Chart)

In [ ]:
```
# This will show us the total number of values for each offense
offense_counts = crime_gdf['OFFENSE'].value_counts()
print(offense_counts)
```

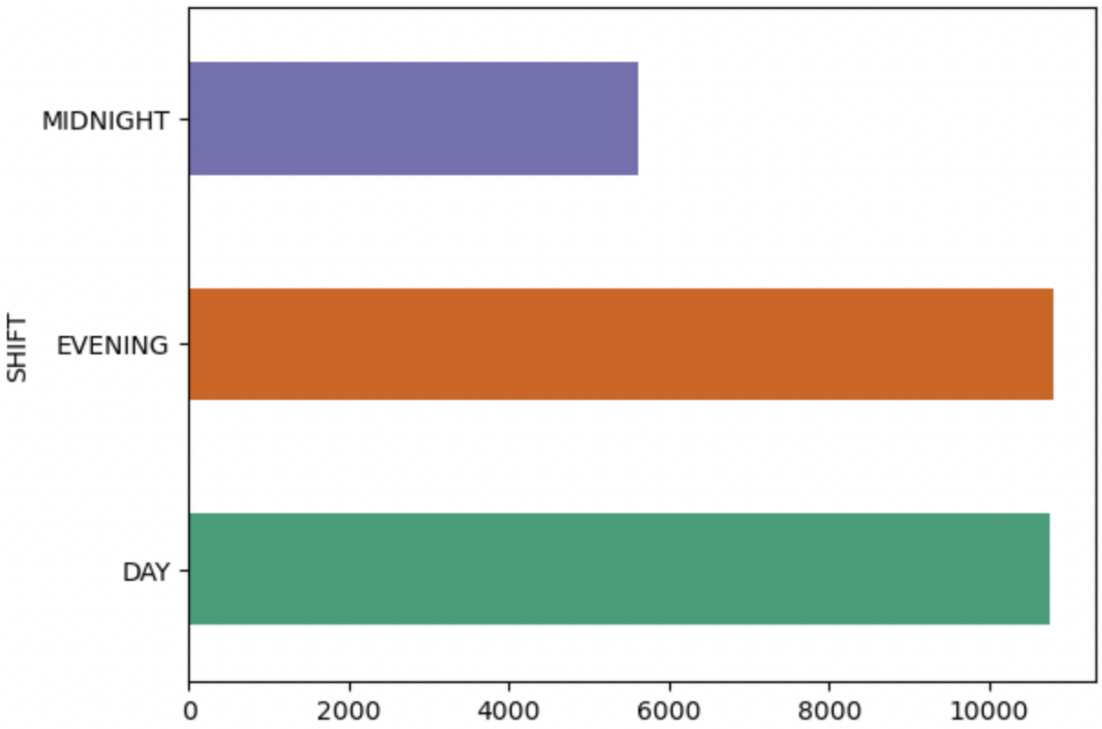
In [ ]:
```
# We can plot this on a histogram using seaborn
import seaborn as sns
crime_gdf.groupby('OFFENSE').size().plot(kind='barh', color=sns.palettes.mpl
```

```
<Axes: ylabel='OFFENSE'>
```



In [ ]:
```
# About half as many arrests were made on the midnight shift.
shift_stats = crime_gdf['SHIFT'].value_counts() #Using value counts
print(shift_stats)
crime_gdf.groupby('SHIFT').size().plot(kind='barh', color=sns.palettes.mpl_p
```

```
<Axes: ylabel='SHIFT'>
```



2. Bivariate Analysis

Usually means boxplot(categorical vs numerical), scatterplot(numerical vs numerical), or contingency table(categorical vs categorical). In this case, the crime data provides numbers of offenses by type and by shift, as well as their location within a ward. Considering these are categorical variables, a contingency variable is sufficient.

In [ ]:
```
# Here for the complete documentation https://pandas.pydata.org/pandas-docs/
x_tab_offense_shift = pd.crosstab(crime_gdf['OFFENSE'], crime_gdf['SHIFT'])
x_tab_offense_shift
```

SHIFT	DAY	EVENING	MIDNIGHT
OFFENSE			
ARSON	3	1	0
ASSAULT W/DANGEROUS WEAPON	269	534	578
BURGLARY	507	266	275
HOMICIDE	0	0	203
MOTOR VEHICLE THEFT	1279	1619	840
ROBBERY	415	821	833
SEX ABUSE	55	79	49
THEFT F/AUTO	3493	2977	1292
THEFT/OTHER	4736	4493	1536

In [ ]:
```
x_tab_offense_ward = pd.crosstab(crime_gdf['OFFENSE'], crime_gdf['WARD'])
x_tab_offense_ward
# Here, we can see that ward 2 had the highest number of thefts in 2022. War
```

WARD	1	2	3	4	5	6	7	8
OFFENSE								
ARSON	0	0	0	0	2	1	0	1
ASSAULT W/DANGEROUS WEAPON	93	98	25	86	239	100	320	420
BURGLARY	102	198	59	86	213	102	140	148
HOMICIDE	14	11	2	9	33	9	47	78
MOTOR VEHICLE THEFT	455	394	87	291	699	595	707	510
ROBBERY	298	234	37	151	352	260	403	334
SEX ABUSE	16	26	11	13	36	21	30	30
THEFT F/AUTO	1402	1335	486	1039	1325	991	718	466
THEFT/OTHER	1572	2500	678	871	1598	1604	1126	816

2.5 Interactive Map: Metro Stations

At this point, we have all the data points for all arrests made in DC in 2022. I'd like to see the number of arrests that happen within a buffer zone of 200 meters from Metro Stations, which is the underground train, as well as potentially the time in which those arrests occured. This means cleaning and processing the data using the same methods as the crime data.

1. Getting the Metro Station Data

In [ ]:
```
# Pulling the Metro Station Entrance Data from OpenData DC. It's fine to use

import requests
import pandas as pd
import geopandas as gpd

# Describing the url for the the Metro Stations dataset
url_metro = "https://maps2.dcgis.dc.gov/dcgis/rest/services/DCGIS_DATA/Trans
# Making the query to the web server, using the Get method from the requests
response = requests.get(url_metro)
response
```

In [ ]:
```
data = response.json()
data
```

In [ ]:
```
#By pulling the data into a json broswer viewer, the data is under features.
metro_data = data['features']
metro_data
```

In [ ]:
```
# As we can see from the above, data is still nested under the attributes an
# I just need the attributes and the geometry columns for the x,y points to
```

```
# Firstly, I need to create empty columns to store the data under the attrib
NAME_list = []
LINE_list = []
long_list = []
lat_list = []
# Now, I need to create a for loop to go over each of the entries in the jso
for index in metro_data: # Defining where I want to entries to come from
    NAME = index['attributes']['NAME'] # Here, we have to go into the attrib
    LINE = index['attributes']['LINE'] # Same again.
    long = index['geometry']['x'] # Geometry is it's own separate structure.
    lat = index['geometry']['y']
# Now, I need to add these values to the empty lists I created earlier
    NAME_list.append(NAME)
    LINE_list.append(LINE)
    long_list.append(long)
    lat_list.append(lat)

# So at this point, my values are added to the empty lists. Therefore, I am
metro_pd = pd.DataFrame({ # I have to define the values to go in each column
    'NAME': NAME_list,
    'lat': lat_list,
    'long': long_list,
    'LINE': LINE_list,
})
```

In [ ]:
```
metro_pd
#The 'LINE' column refers to the name of the metro route, such as the Elizab
# More lines means a larger station, or at least a station with more interse
```

In [ ]:
```
# Lets check the data types.
metro_pd.dtypes
# This looks fine to me; the lat and long are floats, which is necessary for
```

In [ ]:
```
# Transforming to a GeoDataFrame and dropping the unnecessary columns
metro_gdf = gpd.GeoDataFrame(metro_pd, geometry=gpd.points_from_xy(metro_pd[
metro_gdf = metro_gdf.drop('lat', axis=1)
metro_gdf = metro_gdf.drop('long', axis=1)# We should also drop the now unne
metro_gdf.info()

metro_gdf.head()
```

2. Adding a Station Size Column

In [ ]:
```
# I'd like a new column that has the number of lines as an integer. Because
# https://www.w3schools.com/python/ref_string_split.asp
# I need the value of the new column to reflect the length of the split stri
def number_lines_function(line):
    return len(line.split(',')) # Defining the split along the commas
# Now that I have this function, I can use .apply to apply the function to t
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html
metro_gdf['station_size'] = metro_gdf['LINE'].apply(number_lines_function) #
# Display the updated DataFrame
metro_gdf
```

In [ ]:
```
# The line below will return if there are any NaN values. Luckily, it looks
nan_in_columns = metro_gdf.isna().any()
print(nan_in_columns)
```

In [ ]:
```
print(metro_gdf.crs)
print(crime_gdf.crs)
```

In [ ]:
```
metro_gdf.explore() # This shows all the locations of the stations with line
```

3. Checking the CRS

In [ ]:
```
# I found that the best CRS for Washington D.C. is EPSG:'26985', based on th
projected_metro_gdf = metro_gdf.to_crs('EPSG:26985')
projected_crime_gdf = crime_gdf.to_crs('EPSG:26985')
print(projected_metro_gdf.crs)
print(projected_crime_gdf.crs)
```

4. Buffer Creation

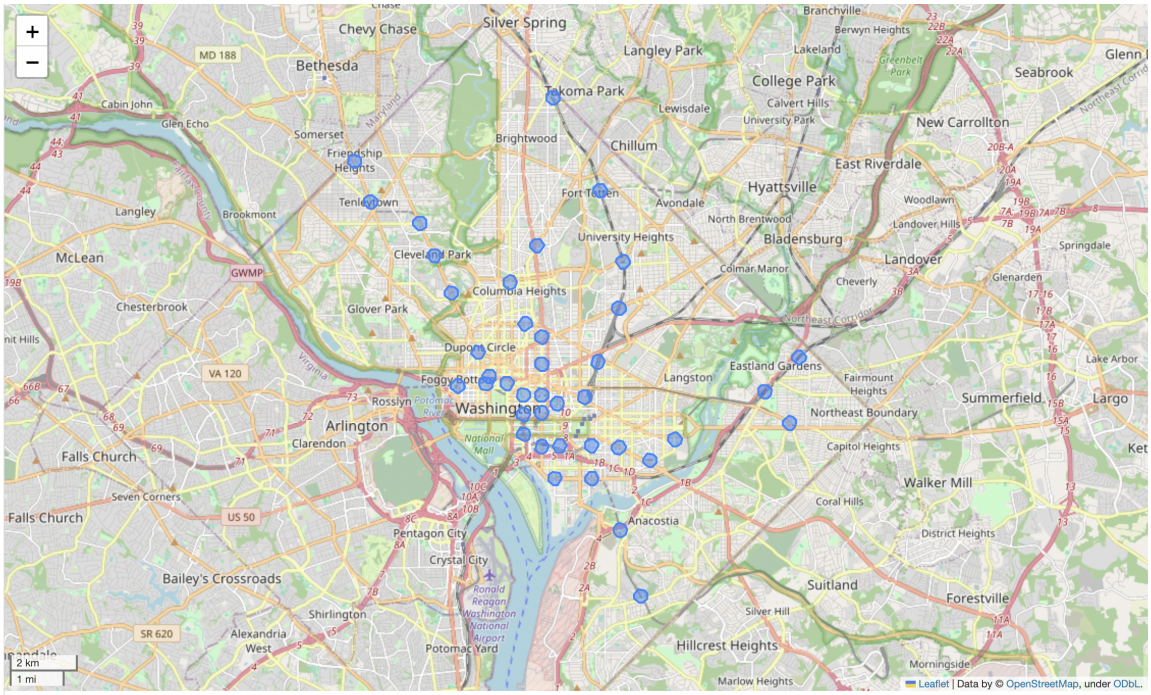
In [ ]:
```
# At this point, I can create the buffers.
metro_buffers = projected_metro_gdf.buffer(200)
```

In [ ]:
```
metro_buffers_gdf = gpd.GeoDataFrame(geometry=metro_buffers) # Making sure i
```

In [ ]:
```
metro_buffers_gdf['buffer_id'] = range(len(metro_buffers_gdf)) # Creating an
metro_buffers_gdf.head()
```

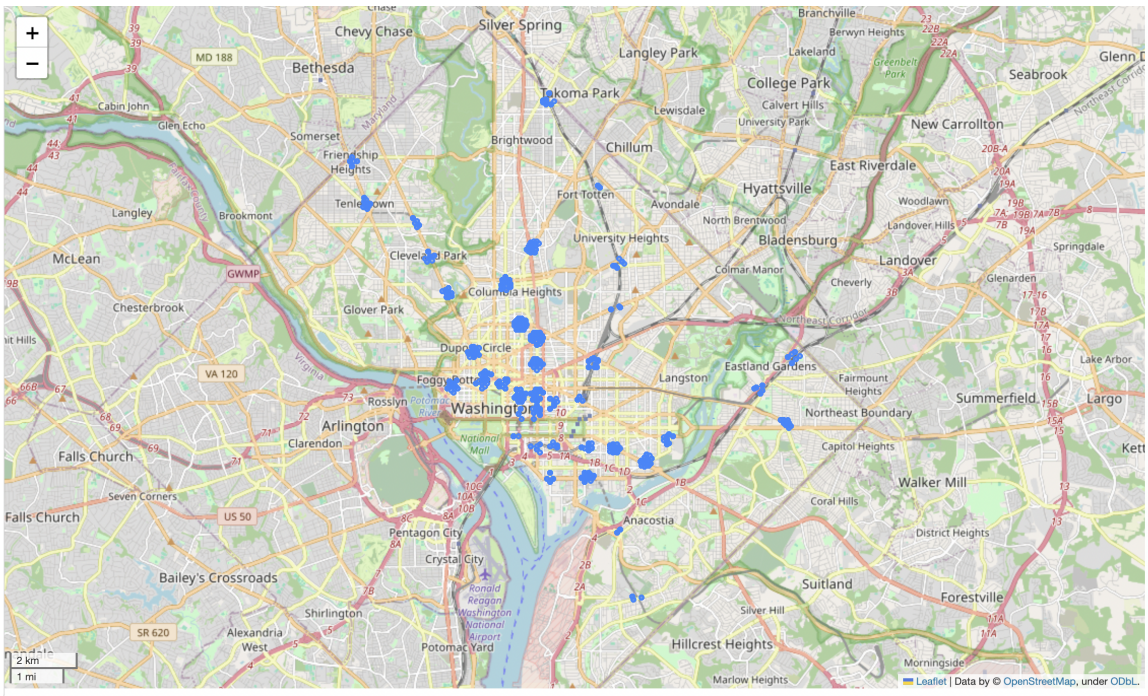
In [ ]:
```
metro_buffers_gdf['station_size'] = projected_metro_gdf['station_size']
# This line gives the metro_buffers_gdf a column called station size based o
```

In [ ]:
```
metro_buffers_gdf.explore()
```



5. Sjoin

In [ ]:
```
# Now that we have the buffers, we can use sjoin to add the points that fall
spatial_result =  projected_crime_gdf.sjoin(metro_buffers_gdf, how="inner",
spatial_result.explore()
```



In [ ]:
```
spatial_result.head()
```

In [ ]:
```
# I want to know the relationship between the number of crimes in each buffe
# I can use group_by to find the number of points in the spatial result gdf
crime_numbers = spatial_result.groupby('index_right').size().reset_index(nam
```

```
# Merging the crime numbers with the original metro_buffers_gdf dataset
crimes_in_buffers = metro_buffers_gdf.merge(crime_numbers, left_index=True,
# I used the documentation from here: https://geopandas.org/en/stable/docs/u

crimes_in_buffers.explore()
```

2.6 Additional Analysis

1. Offenses by Station Size

In [ ]:
```
offenses_per_station_size = spatial_result.groupby('station_size').size().re
offenses_per_station_size
```

	station_size	offenses_per_buffer
0	1	1300
1	2	960
2	3	653
3	4	306
4	5	7

Here, we can see that most crimes happen in small stations; however, this is likely a result of the fact that there are simply more stations with one line. It might also reflect that smaller stations are less well staffed by Metro Police officers.

In [ ]:
```
# Next step here is to calculate the average number of crimes for each size
average_crime_by_size = crimes_in_buffers.groupby('station_size')['crime_cou

print(average_crime_by_size)
```

It looks like there is little relationship between the average number of crimes and the size of the station. To make sure, we can run a correlation test.

2. Correlation Test

From this geopandas documentation, I can use .corr to calculate the relationship between the number of crimes and the size of the station.

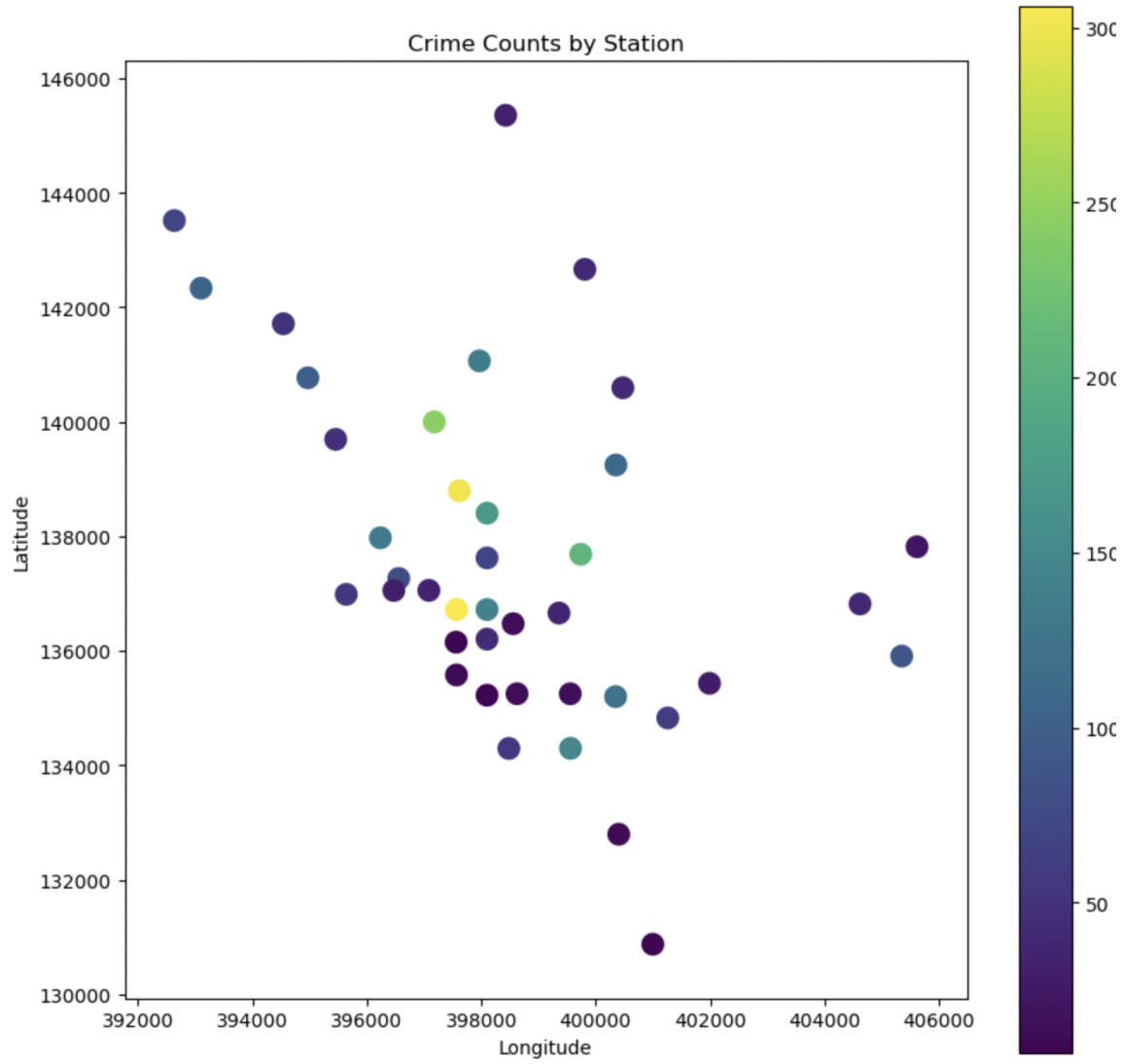
In [ ]:
```
# Calculating correlation between 'crime_count' and 'station_size'
correlation = crimes_in_buffers['crime_count'].corr(crimes_in_buffers['stati

# Print correlation value
print("Correlation between crime count and station size:", correlation)
```

It looks as though the correlation between point count and station size is negligible. However, this might be due to the assumption that more lines = more people, when in reality, the best way to understand that is to count riders. Moreover, there are very few stations that have more than 1 or 2 lines, and that also is skewing the result. Therefore, a next step from here to gain an understanding of the relationship between crime and Metro Stations is instead to plot the number of crimes on a choropleth map.

3. Chloropleth Map

In [ ]:
```
crimes_in_buffers.explore()
```

In [ ]:
```
import matplotlib.pyplot as plt
#I can use matplotlib tool to make a chloropleth map.

# Plotting the buffers with color based on the crime_count variable
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
crimes_in_buffers.plot(column='crime_count', cmap='viridis', ax=ax, legend=T
plt.title('Crime Counts by Station') # Adding labels
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



4. Implications and Further Studies

This chloropleth map has implications for the governance and organisation of resources for the transportation authority. It suggests that the station size is not a good predictor of crime, and that instead, there are several stations that are hotspots for crime. This is evidenced by the close proximity of a yellow point next to a green one, suggesting that resources should be devoted towards addressing and understanding the high crime rates in these two yellow stations. Additionally, an explanation for the relatively low crime rates in the largest stations could be greater police presence near government buildings, which the center of D.C. largely consists of.

This aligns with the literature on crime rates in transportation nodes in D.C.; Irvin-Erikson and La Vigne note that passenger numbers, the frequency of rail service, and the design or security characteristics of the station are hugely impactful on the rates of crimes in and around these stations. Station size is a small factor in this, which likely explains the lack of a direct correlation.

Next steps for this data would include analysing the differences in crime rates based on the time of day as well as the type of crime. Moreover, a crucial limitation of the buffer creation methodology is that it does not create buffers based on travel time. A better way to understand the relatioship between crimes and trainstations is to subset all the crimes that occurred within a 10 minute walking distance, which would require a shift in priorities for exploring this research question.

Nevertheless, the choropleth map and the data reinforce the conclusion that a multi-stakeholder approach to the complex crime problem in stations is needed. Approaches to understanding cuty dynamics and the spatial relationship between crime and transportation hubs could additionally contribute to efforts to address Sustainable Development Goals 10 (Reducing Inequalities) and 11 (Sustainable Cities and Communities).

Challenge 3

1. Go to https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95/about_data
2. Get the data for Motor Vehicle Collisions - Crashes Jan 2024. The dataset contains 2.06 M of records.
3. Use the API endpoint to map the data (e.g. https://data.cityofnewyork.us/resource/h9gi-nx95.json)
4. Customize the map by representing the data by `number_of_persons_killed` and `number_of_cyclist_killed`
5. Finally, calculate descriptive statistics for at least two attributes, such as `mean`, `standard deviation`, and other relevant measures 6. Justify/Describe the attribute selection.
6. Plot correlations between the chosen attributes and create `univariate` and/or `multivariate` charts to justify your insights.

> Please take note that the dataset includes various numerical values. Hence, each student's attribute selection, justification, charts, and maps are expected to vary.

1. Getting the Data

Here is the link to the dataset:

> https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95/about_data

The data on motor vehicle crashes in New York is provided through an API.

> Here is the link to the API: https://data.cityofnewyork.us/resource/h9gi-nx95.json

However, as with lots of large datasets, we will require a token to access this. I created an account on the city of new york open data website. Once you've made an account, click on the edit button next to your profile. On the left hand side of the page, navigate to "Developer Settings". Then click "Create New App Token" and enter in basic details (name and description). You can then copy and paste this token into the code below.

To follow the official documentation on doing so, see this website.

In [ ]:
```
# Based on the documentation from the website provided, here is the code for
import pandas as pd
from sodapy import Socrata

# Unauthenticated client only works with public data sets. Motor Vehicle Col
client = Socrata("data.cityofnewyork.us", "2ibzJzKj3i6OoHoAmTm1Ui3I8") # Rep

# First 2 million results, returned as JSON from API / converted to Python l
results = client.get("h9gi-nx95", limit=2000000) # This limit defines the am

# Convert to pandas DataFrame
mvc_df = pd.DataFrame.from_records(results)
len(mvc_df)
```

In [ ]:
```
mvc_df.columns
```

2. Using the API endpoint to map the data

1. Subsetting the Data

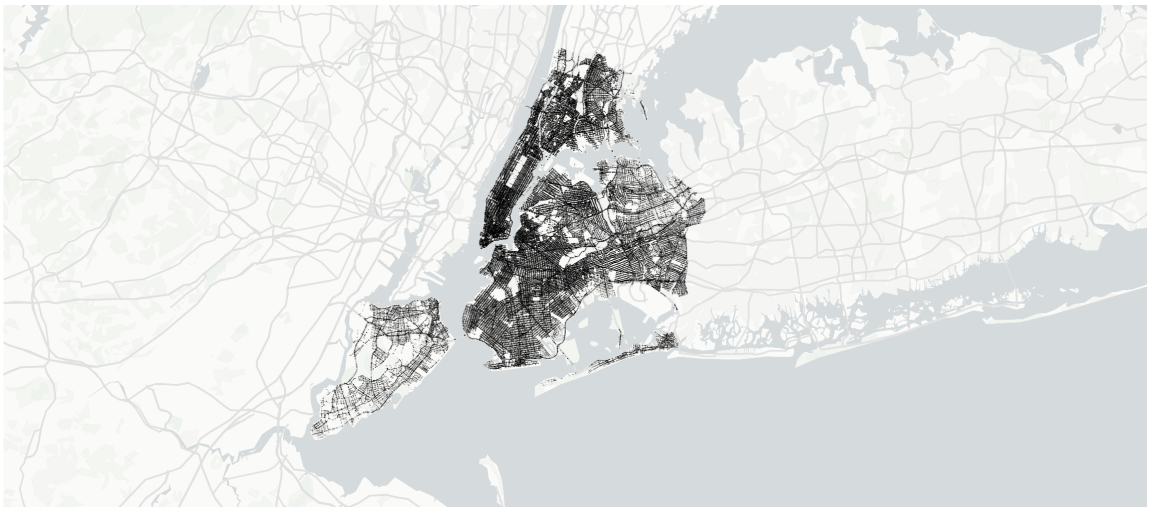
In [ ]:
```
subset_crashes = mvc_df[['number_of_persons_injured', # Let's first subset a
                         'longitude',
                         'latitude',
                         'number_of_persons_killed',
                         'number_of_pedestrians_injured',
                         'number_of_cyclist_injured',
                         'number_of_cyclist_killed',
                         'number_of_motorist_injured',
                         'number_of_motorist_killed',
                            ]]

#Now, I'm going to drop any NaN values from the columns:
subset_crashes = subset_crashes.dropna()

subset_crashes.head()
```

2. Creating A GeoDataFrame

In [ ]:
```
import geopandas as gpd
geometry = gpd.points_from_xy(subset_crashes['longitude'], subset_crashes['l
geo_crashes = gpd.GeoDataFrame(subset_crashes, geometry=geometry, crs='EPSG:
geo_crashes = geo_crashes.drop('longitude', axis=1)# Lets make sure to drop
geo_crashes = geo_crashes.drop('latitude', axis=1)
geo_crashes.head()
```

3. Map Visualisation Using Lonboard

In [ ]:
```
from lonboard import viz
viz(geo_crashes) # The lonboard library does not provide direct parameters f
```



3. Subsetting the map by persons and cyclists killed

Customizing the map by representing the data by number_of_persons_killed and number_of_cyclist_killed

In [ ]:
```
# Because we want to see just the persons and cyclists killed, let's filter
crash_deaths = mvc_df[[
                            'number_of_persons_killed',
                            'number_of_cyclist_killed',
                            'longitude',
                            'latitude',
                           ]]
crash_deaths = crash_deaths.dropna()
geometry = gpd.points_from_xy(crash_deaths['longitude'], crash_deaths['latit
crash_deaths_geo = gpd.GeoDataFrame(crash_deaths, geometry=geometry, crs='EP
crash_deaths_geo = crash_deaths_geo.drop('longitude', axis=1)# Lets make sur
crash_deaths_geo = crash_deaths_geo.drop('latitude', axis=1)
crash_deaths_geo.head()
```

In [ ]:
```
# Lets check the dtypes
crash_deaths_geo.dtypes
```

In [ ]:
```
# Persons and cyclists killed should be int, not objects or str.
crash_deaths_geo.number_of_persons_killed = crash_deaths_geo.number_of_persc
```

```
crash_deaths_geo.number_of_cyclist_killed = crash_deaths_geo.number_of_cycli
```

In [ ]:
```
# At this point, we have points for all the crashes, even if there were no c
# That means we need to filter the data again:
crash_deaths_geo = crash_deaths_geo[(crash_deaths_geo['number_of_persons_kil
```

In [ ]:
```
crash_deaths_geo.shape # Looks like there are only over a hundred or so deat
```

Because there are so few deaths from crashes, using lonboard or another tool that's built for millions of data points is likely not applicable.

In [ ]:
```
crash_deaths_geo.explore()
```

4. Descriptive Statistics for Two Attributes

I want to calculate the descriptive statistics for the crash_time and the number of persons injured. My hypothesis is that hours around twilight or at night are far more likely to result in pedestrian injuries. In a sense, I expect a relationship between adverse road conditions and severity of incident. I am also expecting that hours with greater numbers of vehicle travel will result in more crashes. To understand this, I am going to run some basic descriptive statistics. This refers to the mean, standard deviation, and other relevant measures.

In [ ]:
```
mvc_df.head() # The original dataframe, as defined earlier.
```

1. Subsetting a gdf with crash_time and injuries

In [ ]:
```
import geopandas as gpd
injuries_time = mvc_df[[
                            'crash_time',
                            'number_of_persons_injured',
                            'longitude',
                            'latitude',
                           ]]
injuries_time = injuries_time.dropna()
geometry = gpd.points_from_xy(injuries_time['longitude'], injuries_time['lat
injuries_time = gpd.GeoDataFrame(injuries_time, geometry=geometry, crs='EPSG
injuries_time = injuries_time.drop('longitude', axis=1)# Lets make sure to d
injuries_time = injuries_time.drop('latitude', axis=1)
injuries_time.head()
```

In [ ]:
```
# Let's check the dtypes
injuries_time.dtypes
```

In [ ]:
```
# Crash_time should be a datatime, which injuries should be an int
injuries_time.number_of_persons_injured = injuries_time.number_of_persons_in
injuries_time['crash_time'] = pd.to_datetime(injuries_time['crash_time'], fo
```

In [ ]:
```
injuries_time.head()
```

In [ ]:
```
# I am only interested in the hours that injuries occured. From this documen
# https://pandas.pydata.org/docs/reference/api/pandas.Series.dt.hour.html
injuries_time['crash_hour'] = injuries_time['crash_time'].dt.hour
# Let's remove the crash_time column and just use the crash_hour column.
injuries_time = injuries_time.drop('crash_time', axis=1)
injuries_time
```

2. Descriptive Stats for Hour of Crash

In [ ]:
```
descriptive_stats = injuries_time.describe()
descriptive_stats
```

In [ ]:
```
 # There are too many decimal points. From this documentation, I can round t
descriptive_stats.round(2)
```

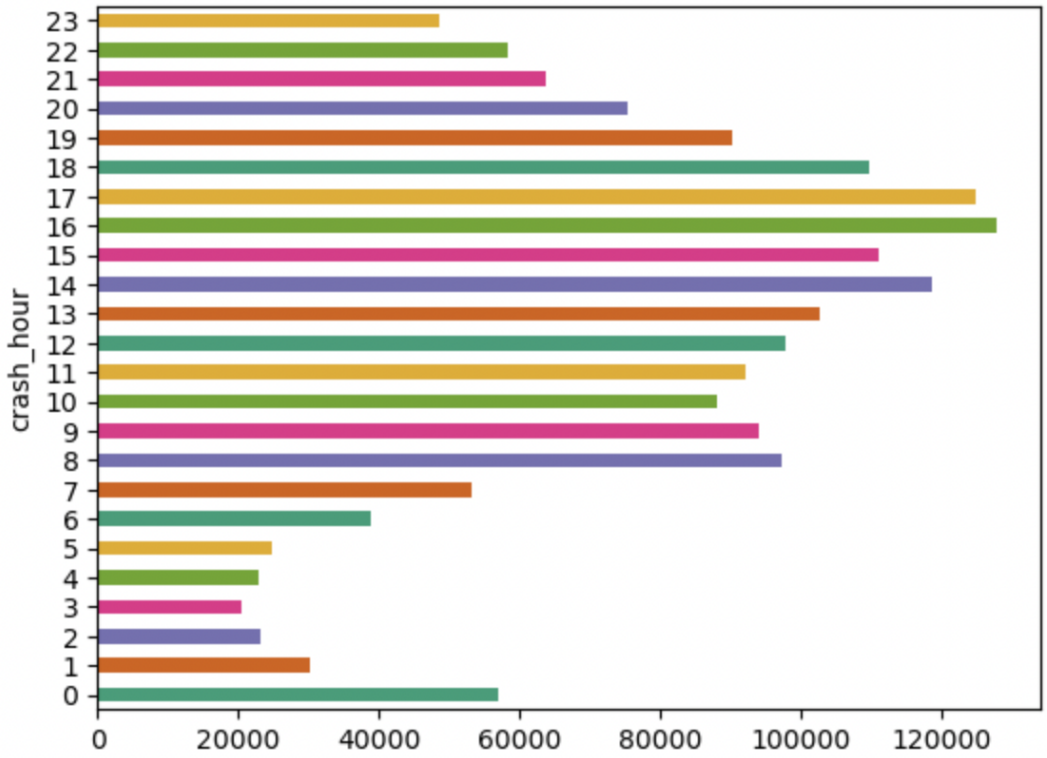
In [ ]:
```
#Using .mean
mean_hour = injuries_time['crash_hour'].mean()
mean_persons_injured = injuries_time['number_of_persons_injured'].mean()
print(mean_persons_injured) # On average, approximately a third of all accid
print(mean_hour) # 13:00 to 14:00 is the mean time of an accident.
```

5. Justify/Describe the attribute selection.

The hour of the crash provides insight into the temporal distribution of crashes. On the other hand, the number of persons injured can serve as a proxy for the severity of the crash, and thus selecting both these attributes will provide insight into the relationship between the time and the severity of crash. In turn, this might inform policy makers on the times that crashes result in the worst human health impact, which could offer insights on when to employ more ambulances or traffic police, for example.

In [ ]:
```
# Let's use seaborn to create a histogram of all the crashes by the hour the
import seaborn as sns
injuries_time.groupby('crash_hour').size().plot(kind='barh', color=sns.palet
```

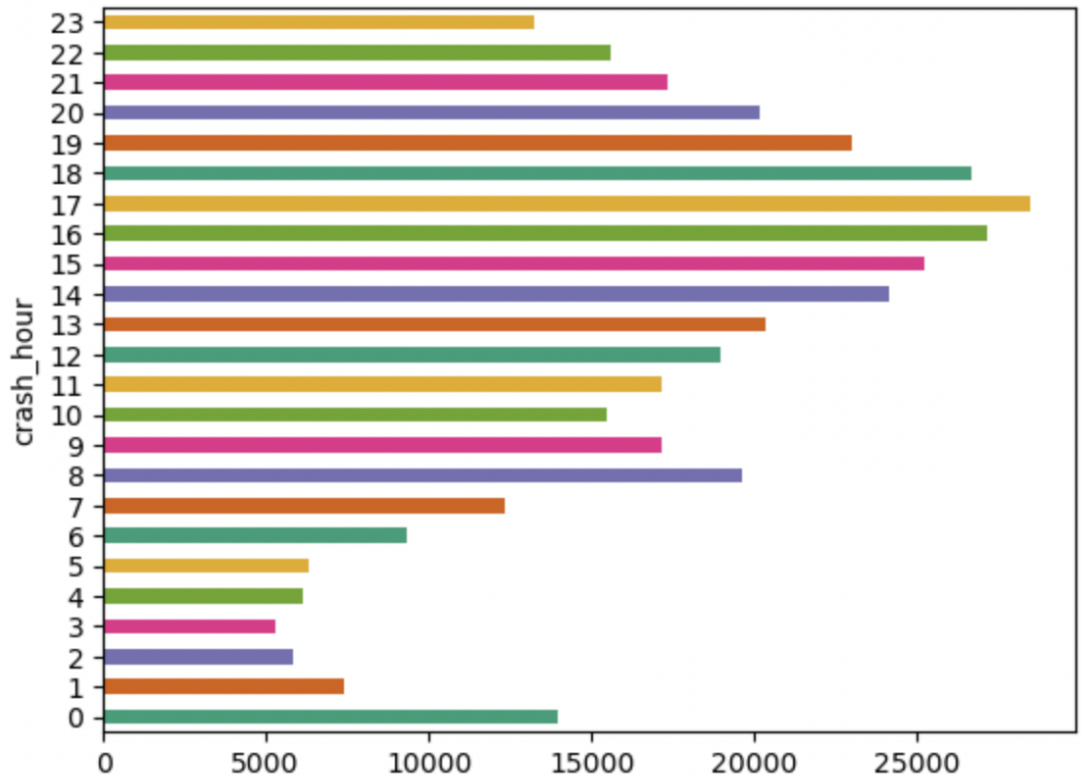
```
<Axes: ylabel='crash_hour'>
```



In [ ]:
```
# Let's make a subset of the data that only includes crashes that resulted i
crashes_resulting_in_injuries = injuries_time.loc[injuries_time['number_of_p
print(f"The total number of injuries is {len(crashes_resulting_in_injuries)}
crashes_resulting_in_injuries.groupby('crash_hour')['number_of_persons_injur
# Looks like 17:00 is the hour in which the most injuries occur. This aligns
```

In [ ]:
```
# Let's use seaborn to create a histogram of all the injuries by the hour th
import seaborn as sns
crashes_resulting_in_injuries.groupby('crash_hour').size().plot(kind='barh',
```

```
<Axes: ylabel='crash_hour'>
```



I expected that the number of crashes would change throughout the day. The data presented above support this. Injuries are highest around 17:00-18:00 and lowest at 03:00-04:00. There is also a jump in injuries between 08:00 and 09:00, which coincides with the morning rush hour. Crashes, on the other hand, peak between 16:00 and 17:00. This suggests that more injuries occur later during the rush hour, whereas crashes tend to happen earlier. In turn, this information be used could inform policymakers and emergency health service providers.

Challenge 4

1. Lonboard and Datashader Differences

What are the differences between working with Lonboard and Datashader? Which one provides the most exciting functionality, and how do the outcomes from both of them vary?

Lonboard is a library that enables interactive visualisations such as choropleths and heatmaps in Jupyter. It is an quick way to create interactive data visualisations because it uses a binary pipeline that avoids text encoding through GEOJsons (an example is ipyleaflets, which can be quite slow). In this sense, the interactive outputs of lonboard can be applied to front-end contexts such as HTML or JavaScript because of this speed.

However, this comes at a cost to detail. When the user pans or zooms the interactive map, Lonboard aggregates data points to accommodate for performance, which degrades the applicability of the library.

Datashader, on the other hand, is a library that rasterises and aggregates datasets in order to visualise them. Essentially, it enables the processing of incredibly large datasets by turning the points into aggregates or images to render them. In terms of ease of use and customisation, Datashader has more customisation options compared to lonboard, but it is less streamlined. However, there are more exciting possibilities to use Datashader to map millions of data points in fields beyond geography, such as economics or ecology.

2. Dataset Finding

The dataset that I picked is the NYPD Calls for Service Year to Date. This data represents all the calls from residents of NYC to the 911 Emergency Dispatcher in 2023. It geolocates these calls and adds a description of what the caller is reporting. It also includes the date of the incident, the time of the dispatch and arrival, and whether the crime is currently occuring (ie. the cip_jobs column).

3. Potential Problem

A potential problem for this dataset is for the NYPD to determine where calls are mostly coming from across the city and whether they are reporting on a crime in progress. This might allow the agency to allocate resources; for example, if most calls that report on a crime-in-progress occur in one neighborhood or business district, then the NYPD could station more police officers or conduct effective public education campaigns on bystander syndrome and so on.

Moreover, as Wirsching suggests in this paper, understanding where 911 calls are coming from and response times can be used to quantitiatively inform research on the relationship between political alignment and municipal policing practices.

4. Loading the Dataset

Here is the API for the calls for service dataset in New York City.

> https://data.cityofnewyork.us/resource/n2zq-pubd.json

Similar to the process on App Tokens and authentication earlier, to access this data you need to create a new token and replace it with the one in the code.

In [ ]:
```
#This code is based on the documentation from the website provided by socrat
import pandas as pd
from sodapy import Socrata
```

```
client = Socrata("data.cityofnewyork.us", "VExzd4RP8BY5ZdhJWYiYTAHQI") # Rep

# 5,000,000 results, returned as JSON from API / converted to Python list of
# dictionaries by sodapy.
results = client.get("n2zq-pubd", limit=5000000) # While the dataset has ove
# As a note, this took a significant amount of time to load, so feel free to

# Convert to pandas DataFrame
calls_df = pd.DataFrame.from_records(results)
calls_df.head()
```

1. Dropping unnecessary columns

In [ ]:
```
keep_cols = [
    "typ_desc", # string description of the call
    "cip_jobs", # Whether the crime is in progress, and if it is, whether it
    "latitude",
    "longitude",
]
calls_df = calls_df[keep_cols]
```

2. Changing the dtypes

In [ ]:
```
calls_df.dtypes
```

In [ ]:
```
# The latitude and longitude should be a float variable. cip_jobs is split i
calls_df.latitude = calls_df.latitude.astype(float)
calls_df.longitude = calls_df.longitude.astype(float)
calls_df.dtypes
```

3. Geodataframe and CRS

One issue with Datashader is that it changes the geometry of the data as it creates a raster of the points on the grid. In order to ensure that the points have the correct CRS for the datashader tool, I need to ensure that the longitude and latitude are in the correct projection. One way to do this is to set the crs of calls_df, convert it to a geodataframe, and then convert these to easting and northing in order to match the projection of New York City.

In [ ]:
```
# Creating a geodataframe
import geopandas as gpd
geometry = gpd.points_from_xy(calls_df['longitude'], calls_df['latitude']) #
calls_gdf = gpd.GeoDataFrame(calls_df, crs='EPSG:4326', geometry=geometry) #
```

In [ ]:
```
# Changing the CRS: I need to use a spherical mercator projection, which plo
calls_gdf = calls_gdf.to_crs('EPSG:3857')
```

In [ ]:
```
# At this point, I need to convert the correct coordinates into separate col
calls_gdf['longitude_projected'] = calls_gdf.geometry.x
```

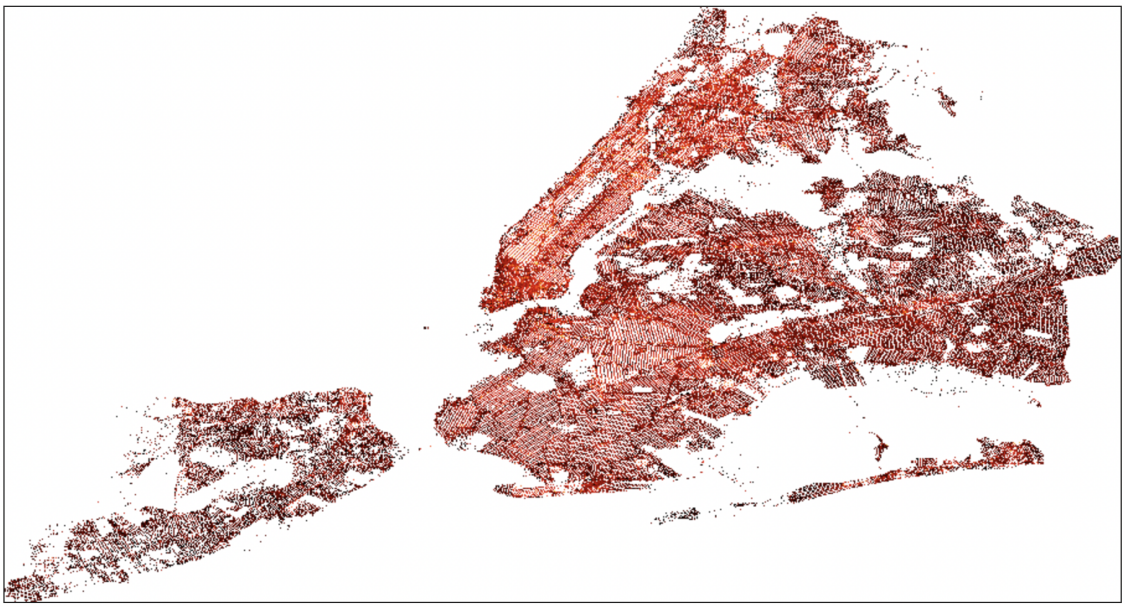
```
calls_gdf['latitude_projected'] = calls_gdf.geometry.y
calls_gdf.head()
```

In [ ]:
```
# Dropping unnecessary columns
calls_gdf = calls_gdf.drop('longitude', axis=1)# Lets make sure to drop the
calls_gdf = calls_gdf.drop('latitude', axis=1)
calls_gdf = calls_gdf.drop('geometry', axis=1)
```

4. Datashader: Key Variables of Interest

1. Mapping all calls with datashader

In [ ]:
```
# From the datashader documentation, here's a way to use datashader without
# This is the map of every single call placed in NYC.
import datashader as ds, pandas as pd, colorcet
cvs = ds.Canvas(plot_width=900, plot_height=480)
agg = cvs.points(calls_df, 'longitude', 'latitude') # Using calls_df here be
img = ds.tf.shade(agg, cmap=colorcet.fire, how='log')
img
```

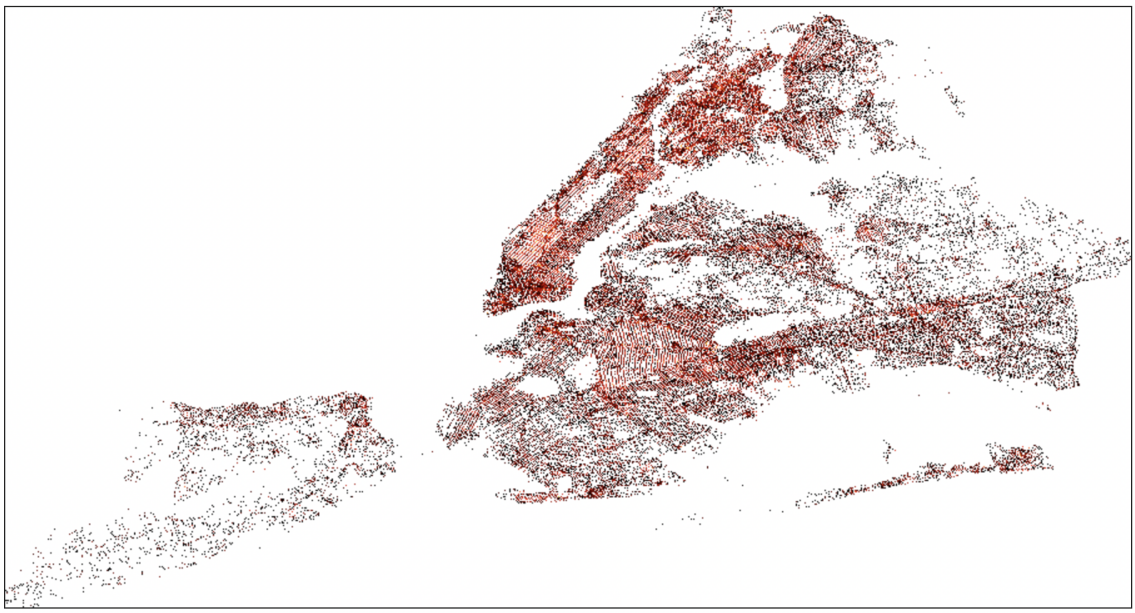


2. Mapping Crimes in Progress Calls with Datashader

However, I want to see the map of calls for service placed relating to a crime in progress. Therefore, I just need to use .loc to subset the original calls_df to include only calls where cip_jobs == Critical and == Non Critical

In [ ]:
```
critical_calls_gdf = calls_gdf[calls_gdf['cip_jobs'] == 'Critical'] # Using
non_critical_calls_gdf = calls_gdf[calls_gdf['cip_jobs'] == 'Non Critical']
# using pd.concat to add them together. this is now all the calls that refer
CIP_calls_gdf = pd.concat([critical_calls_gdf, non_critical_calls_gdf])
print(len(CIP_calls_gdf))
```

In [ ]:
```
# Using datashader to plot the crimes in progress calls
import datashader as ds, pandas as pd, colorcet
cvs = ds.Canvas(plot_width=900, plot_height=480) # Creating a canvas with a
agg = cvs.points(CIP_calls_gdf, 'longitude_projected', 'latitude_projected')
img = ds.tf.shade(agg, cmap=colorcet.fire, how='log') # This applies a 'fire
img
```

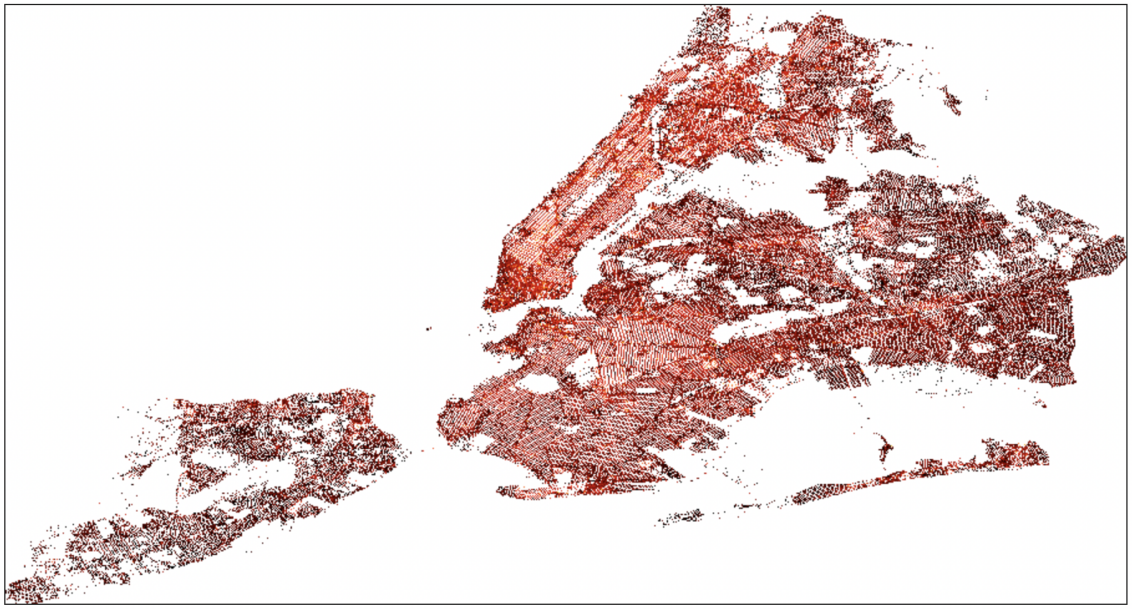


3. Mapping the non crimes-in-progress calls with datashader

In order to compare the calls that refer to a crime in progress and the calls that don't, I need to define the non_CIP calls.

In [ ]:
```
# Plotting the calls that do not refer to a crime in progress
non_CIP_calls_gdf = calls_gdf[calls_gdf['cip_jobs'] == 'Non CIP']
```

In [ ]:
```
# Using datashader to plot the non crimes in progress calls
import datashader as ds, pandas as pd, colorcet
cvs = ds.Canvas(plot_width=900, plot_height=480)
agg = cvs.points(non_CIP_calls_gdf, 'longitude_projected', 'latitude_project
img = ds.tf.shade(agg, cmap=colorcet.fire, how='log')
img
```

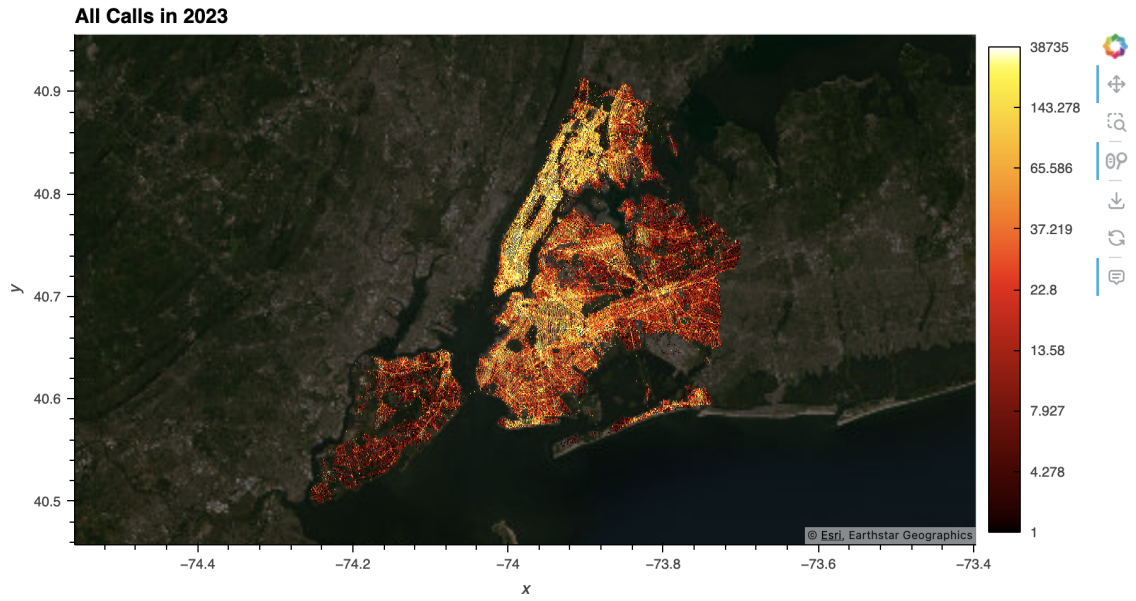


5. Using hvplot

In [ ]:
```
# Ensuring all the libraries are loaded.
import holoviews as hv, pandas as pd, colorcet as cc
from holoviews.element.tiles import EsriImagery
from holoviews.operation.datashader import datashade
hv.extension('bokeh')
```

1. Mapping All Calls with hvplot

In [ ]:
```
import hvplot.pandas #This adds hvplot support for a pandas dataframe
map_tiles = EsriImagery().opts(alpha=0.5, width=900, height=480, bgcolor='bl
plot = calls_gdf.hvplot(
    'longitude_projected', #Notice the longitude projected; this is because
    'latitude_projected',
    kind='scatter', # This sets the type of plot as a scatter
    rasterize=True, # This increases the performance by rasterising the poin
    cmap=cc.fire, # Defining the same colormap as before with datashader
    cnorm='eq_hist' # Finally, this normalises the colormap
).opts(title='All Calls in 2023') # This addition to add a title is from the
map_tiles * plot
```

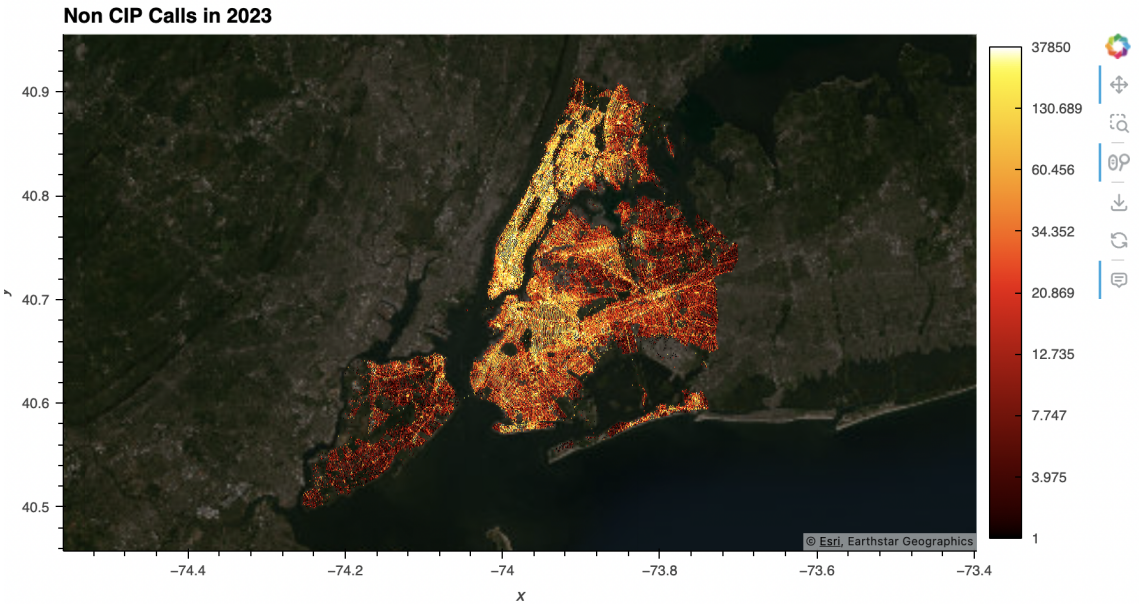


2. Crime-in-progress Calls with hvplot

In [ ]:
```
map_tiles = EsriImagery().opts(alpha=0.5, width=900, height=480, bgcolor='bl
plot = CIP_calls_gdf.hvplot(
    'longitude_projected',
    'latitude_projected',
    kind='scatter',
    rasterize=True,
    cmap=cc.fire,
    cnorm='eq_hist'
).opts(title='CIP Calls in 2023')
map_tiles * plot
```

3. Non Crimes in Progress Calls

In [ ]:
```
map_tiles = EsriImagery().opts(alpha=0.5, width=900, height=480, bgcolor='bl
plot = non_CIP_calls_gdf.hvplot(
    'longitude_projected',
    'latitude_projected',
    kind='scatter',
    rasterize=True,
    cmap=cc.fire,
    cnorm='eq_hist'
).opts(title='Non CIP Calls in 2023')
map_tiles * plot
```



6. Challenges and Discussion

One major challenge that affected the hvplot was the baselayer below the points was the CRS. Using WGS84 as a CRS inaccurately plotted the location of the points in relation to the baselayer. This is likely due to the fact that the dataset itself uses longitude and latitude points specifically based off of the most accurate CRS for NYC, which is EPSG: 3857. In this sense, I had to recognise the incorrect CRS and then ensure that all the dataframes operated under the same crs.

Beyond this, the two hvplot results were difficult to compare. In large part, this is due to the similar colors of the CIP calls and the non-CIPs. A potential way to make the distributiuon of crimes in progress clearer would be to use different color ramps for CIP calls and non-CIP calls and then to plot them on the same map. However, the functionality for doing this in hvplot with the method above is limited because of the rasterisation function. Further libraries maybe have to be used in order to overlay the data in a way that does not obscure data due to large cell grid sizes; for example, if crimes in progress are green and crimes not in progress are red, then there will have to be a calculation within that grid cell to decide which color to use. This may require

further libraries and processing to be employed so that the data does not become obscured.

Nevertheless, the data does reveal that there is a higher concentration of calls relating to crimes in progress along major roads in Manhattan, as seen by the strong vertical placements of the points. It also seems as though there is a relationship between crime-in-progress calls and major ropadways, which might reflect the increased visibility and traffic that occurs on major roadways, meaning higher chances that someone will call. Moreover, residential areas in Brooklyn and Staten Island have far fewer calls relating to crimes in progress compared to central New York.

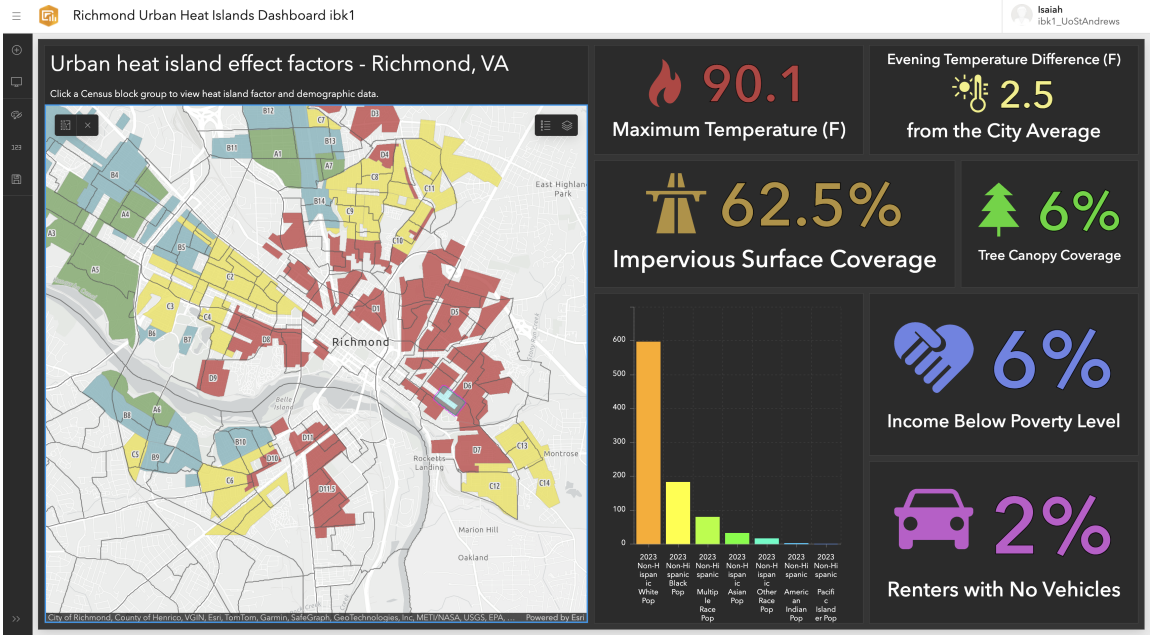
Lab No 4: Geovisualization Techniques - Part 2, Apps: 2 Challenges

Challenge 2

To see the screenshot of the dashboard on Urban Heat Islands in Richmond, VA, please see Lab_4 folder in my Github Repo or the Google Drive folder. If you downloaded the folder from Google Drive, the following markdown should also print the screenshot.

Link to my ArcGIS Dashboard

Here is the dashboard:



Challenge 3

1. Go to this portal https://www.spatialdata.gov.scot/geonetwork/srv/eng/catalog.search#/home
2. Get the Scottish Index of Multiple Deprivation (SIMD) 2020 dataset and extract the data only for the city of Edinburgh.
3. Create two static choropleth maps (e.g. `matplotlib`). These maps should represent an attribute you find interesting in the SIMD dataset. Using two different classifier methods, you need to show how the maps appear different even though the data and attributes are the same. Include a clear description of your choice and the difference in the classification method for the attribute chosen (e.g. Plotting histograms with breakpoints(bins). You can find a complete list of classifiers at https://pysal.org/mapclassify/api.html.
4. Finally, create other two interactive maps (e.g. `choropleth_mapbox`) - one for Glasgow and one for Edinburgh - to represent the difference in deprivation for both cities. Pick any of the available attributes.

> As always include the appropriate descriptions and code comments where you narrate how you are processing the data. And the insights you get from the results.

3.1 Finding SIMD Data for 2020

Here is the data source from the Scottish Government.

In [ ]:
```
# From the portal, I downloaded all of the data on SIMD in 2020
import plotly.express as px
import geopandas as gpd

shapefile_path = 'data/SG_SIMD_2020.shp' # I will upload the zip file to my
gdf = gpd.read_file(shapefile_path)
gdf.head()
```

3.2 Extracting the SIMD Data for Edinburgh

In [ ]:
```
# It looks like LAName includes the names. I can use value.counts to list al
LAName_counts = gdf['LAName'].value_counts()
print(LAName_counts)
```

In [ ]:
```
# City of Edinburgh is the value to sort by in order to filter the gdf.
edinburgh_gdf = gdf[gdf['LAName'] == 'City of Edinburgh'] # == is to specify
edinburgh_gdf.head()
```

3.3 Creating Two Static Choropleth Maps with matplotlib

These maps should represent an attribute you find interesting in the SIMD dataset. Using two different classifier methods, you need to show how the maps appear different even though the data and attributes are the same. Include a clear description of your choice

and the difference in the classification method for the attribute chosen (e.g. Plotting histograms with breakpoints(bins). You can find a complete list of classifiers at https://pysal.org/mapclassify/api.html.

In [ ]:
```
# Just to see what is in the dataset.
edinburgh_gdf.columns
```

I'm interested in the number of houses that are overcrowded in Edinburgh, which is defined by "HouseNumOC".

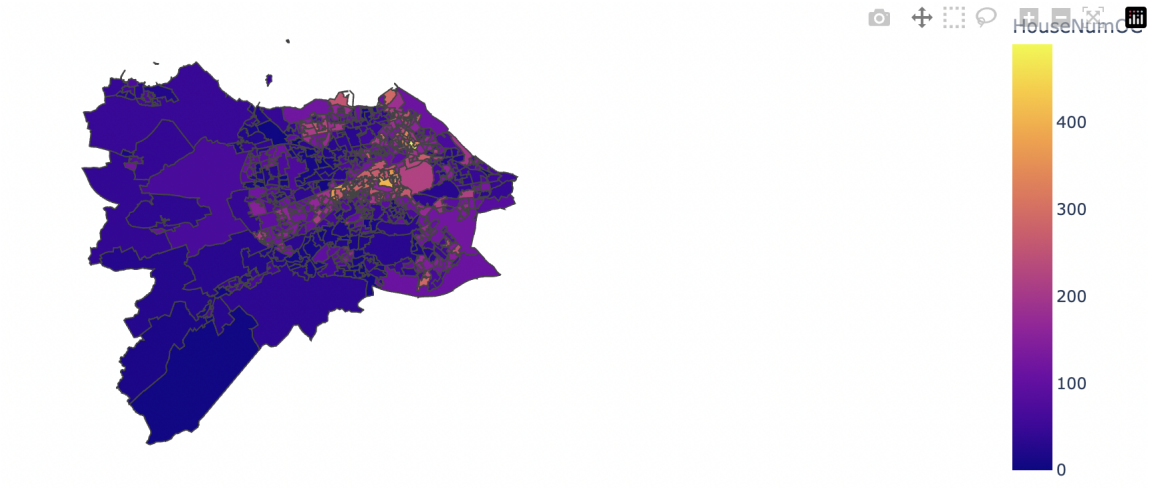
In [ ]:
```
# Therefore, I'm going to subset the data by HouseNumOC as well as the geome
keep_cols = [
    "DataZone",
    "HouseNumOC",
    "geometry",
]
oc_gdf = edinburgh_gdf[keep_cols] # Is there a way to export this as a csv t
oc_gdf.explore() # Just to gain a sense of the range of values that we are w
```

In [ ]:
```
# Let's check the crs
print(oc_gdf.crs)
```

In [ ]:
```
# Let's reproject this to WGS84
projected_oc_gdf = oc_gdf.to_crs('EPSG:4326')
```

In [ ]:
```
# In order to plot the choropleth, we need to set the index.
projected_oc_gdf.set_index('DataZone', inplace=True)
```

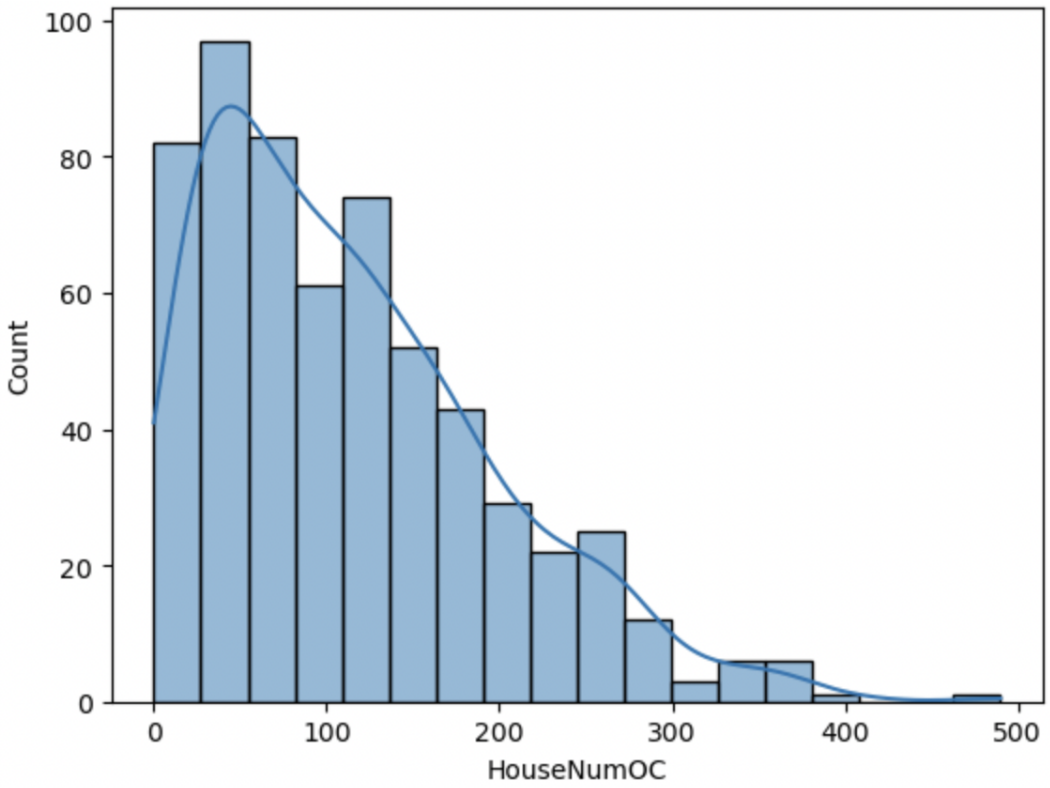
In [ ]:
```
fig = px.choropleth(projected_oc_gdf,
                    geojson=projected_oc_gdf.geometry,
                    locations=projected_oc_gdf.index,
                    color="HouseNumOC",
                    projection="mercator", # This is important because it en
                    #the geographical features are displayed in a way that i
                   )
fig.update_geos(fitbounds="locations", visible=False) # Ensuring the bounds
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



1. Classifier Methods using PYSAL

In [ ]:
```
#Importing libraries
import mapclassify as mc
import matplotlib.pyplot as plt
import folium
import seaborn as sns
```

In [ ]:
```
# Let's make a histogram of the distribution
sns.histplot(data=projected_oc_gdf, x="HouseNumOC", kde=True)
plt.show()
# It looks like the data is positively skewed, with most data in the 0-100 r
```



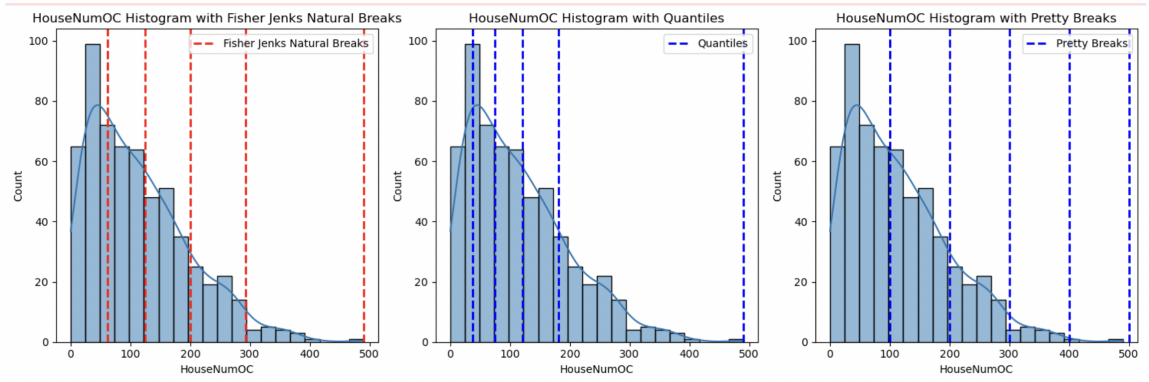
In [ ]:
```
# Defining the number of classes for classification
num_classes = 5

# Using FisherJenks classification: emphasizes differences between classes
classifier_fj = mc.FisherJenks(projected_oc_gdf['HouseNumOC'], k=num_classes
print(classifier_fj)
print(min(classifier_fj.bins), max(classifier_fj.bins))
print(classifier_fj.bins) # bins object stores the break points when conside

# Using Quantiles classification: means each class has the same number of fe
classifier_qt = mc.Quantiles(projected_oc_gdf['HouseNumOC'], k=num_classes)
print(classifier_qt)
print(min(classifier_qt.bins), max(classifier_qt.bins))
print(classifier_qt.bins)

# Using Pretty Breaks classification: makes classes based on rounded numbers
classifier_pb = mc.PrettyBreaks(projected_oc_gdf['HouseNumOC'], k=num_classe
print(classifier_pb)
print(min(classifier_pb.bins), max(classifier_pb.bins))
print(classifier_pb.bins)
```

2. Plotting Different Classifier Methods

Let's see which of the three (Pretty Breaks, Fisher Jenks, or Quantiles) are the most appropiate for the data. I can do this by plotting a line along each of the classifcation breaks.

In [ ]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 5)) # Changed to 1, 3 to accommo

# 2. The ax property allows us to plot them all in one line. This one is for
sns.histplot(data=projected_oc_gdf, x="HouseNumOC", ax=axes[0], kde=True, bi
axes[0].axvline(classifier_fj.bins[0], color='red', linestyle='dashed', line
for bin_value in classifier_fj.bins:
    axes[0].axvline(bin_value, color='red', linestyle='dashed', linewidth=2)
axes[0].set_title("HouseNumOC Histogram with Fisher Jenks Natural Breaks")
axes[0].legend()

# 3. The Quantiles plot using ax=axes[1]
sns.histplot(data=projected_oc_gdf, x="HouseNumOC", ax=axes[1], kde=True, bi
axes[1].axvline(classifier_qt.bins[0], color='blue', linestyle='dashed', lin
for bin_value in classifier_qt.bins:
    axes[1].axvline(bin_value, color='blue', linestyle='dashed', linewidth=2
axes[1].set_title("HouseNumOC Histogram with Quantiles")
axes[1].legend()

#4. Final Pretty Breaks plot using ax=axes[2]
sns.histplot(data=projected_oc_gdf, x="HouseNumOC", ax=axes[2], kde=True, bi
axes[2].axvline(classifier_pb.bins[0], color='blue', linestyle='dashed', lin
for bin_value in classifier_pb.bins:
    axes[2].axvline(bin_value, color='blue', linestyle='dashed', linewidth=2
axes[2].set_title("HouseNumOC Histogram with Pretty Breaks")
axes[2].legend()

# 5. Adjusting the plot.
```

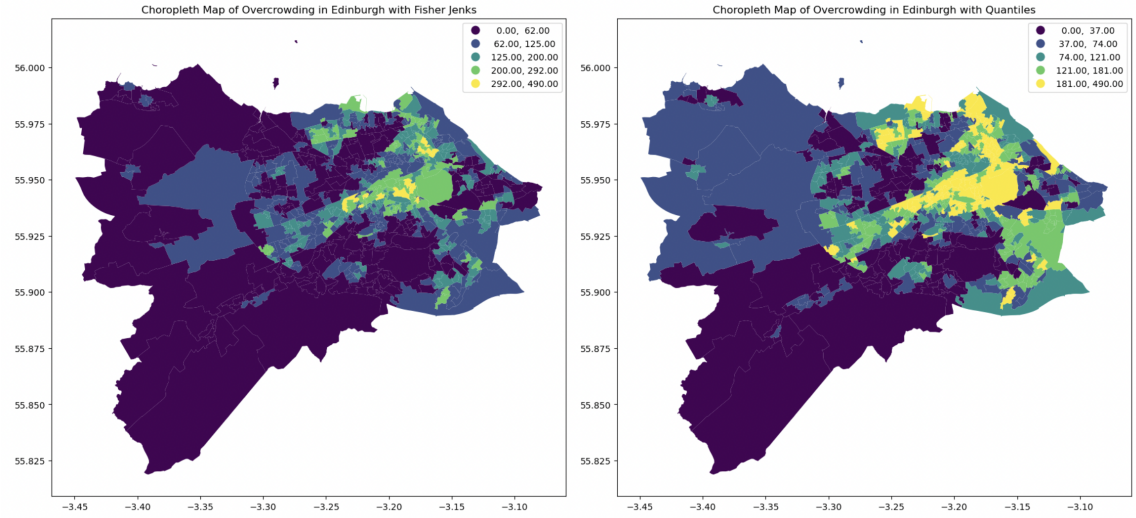
```
plt.tight_layout()
plt.show()
```



Looking at these three classifications, both Fisher Jenks and Quantiles have a wide break on the tail of the distribution. Pretty Breaks is similar to Equal Intervals; it does not discriminate based on the distribution. In order to showcase the areas that are experiencing overcrowding for policy makers, Fisher Jenks or Quantiles is more appropriate because it would highlight the overcrowded areas based on their proportionality to the rest of the data.

My assumption is that Quantiles will be more appropriate for visualizing this data because the most intense color will reflect the fifth most overcrowded area, which could effetively used to allocate resources and funds.

In [ ]:
```
# Let's map the data using Fisher Jenks and Quantiles in a subplot
fig, axs = plt.subplots(1, 2, figsize=(18, 8))

projected_oc_gdf.plot(column='HouseNumOC', ax=axs[0],
         legend=True, cmap='viridis',
         scheme='UserDefined', # So that I can define the classifcation myse
         classification_kwds={'bins': classifier_fj.bins} # Fisher Jenks as
        )

axs[0].set_title("Choropleth Map of Overcrowding in Edinburgh with Fisher Je

projected_oc_gdf.plot(column='HouseNumOC', ax=axs[1],
         legend=True, cmap='viridis',
         scheme='UserDefined',
         classification_kwds={'bins': classifier_qt.bins}) # Quantiles as de

axs[1].set_title("Choropleth Map of Overcrowding in Edinburgh with Quantiles

plt.tight_layout()
plt.show()
```



3. Insights

The Fisher Jenks choropleth map identifies the areas that are experiencing extreme levels of overcrowding, which could provide justification for interventions if resources are limited. In addition, the areas on the outskirts of the city are more homogenous. On the other hand, the Quantiles choropleth map indicates that a fifth of all overcrowded homes are in the center because the yellow range is over 100 value points greater. It also suggests that southern areas on the outskirts of the city are less overcrowded than the areas in the north. All in all , depending on the audience of the data, either classification method could be justified. Nevertheless, it seems as though Fisher Jenks more clearly shows the areas that have the most overcrowded households on the extreme end of the distribution.

3.4 Interactive Maps for Glasgow and for Edinburgh

In [ ]:
```
# Here's the glasgow SIMD data with the following code.
import plotly.express as px
import geopandas as gpd

shapefile_path = 'data/SIMD_2020_GlasgowCity.shp'
gdf = gpd.read_file(shapefile_path)
```

In [ ]:
```
gdf.set_index('DataZone', inplace=True)
gdf.head()
```

In [ ]:
```
gdf.columns
```

In [ ]:
```
# Let's filter the data to the HlthDrugSR; this refers to the Number of Hosp
# Using keep_col
keep_cols = [
    "HlthDrugSR",
    "geometry",
```

```
]
glas_od_gdf = gdf[keep_cols]
```

1. Descriptive Statistics for Glasgow Data

We need to run the descriptive statistics for the attribute HlthDrugSR to understand the range of values.

In [ ]:
```
glas_od_gdf.describe()
```

2. Descriptive Statistics for Edinburgh Data

In [ ]:
```
keep_cols_1 = [
    "HlthDrugSR",
    "geometry",
]
edi_od_gdf = edinburgh_gdf[keep_cols_1]
```

In [ ]:
```
edi_od_gdf.describe()
```

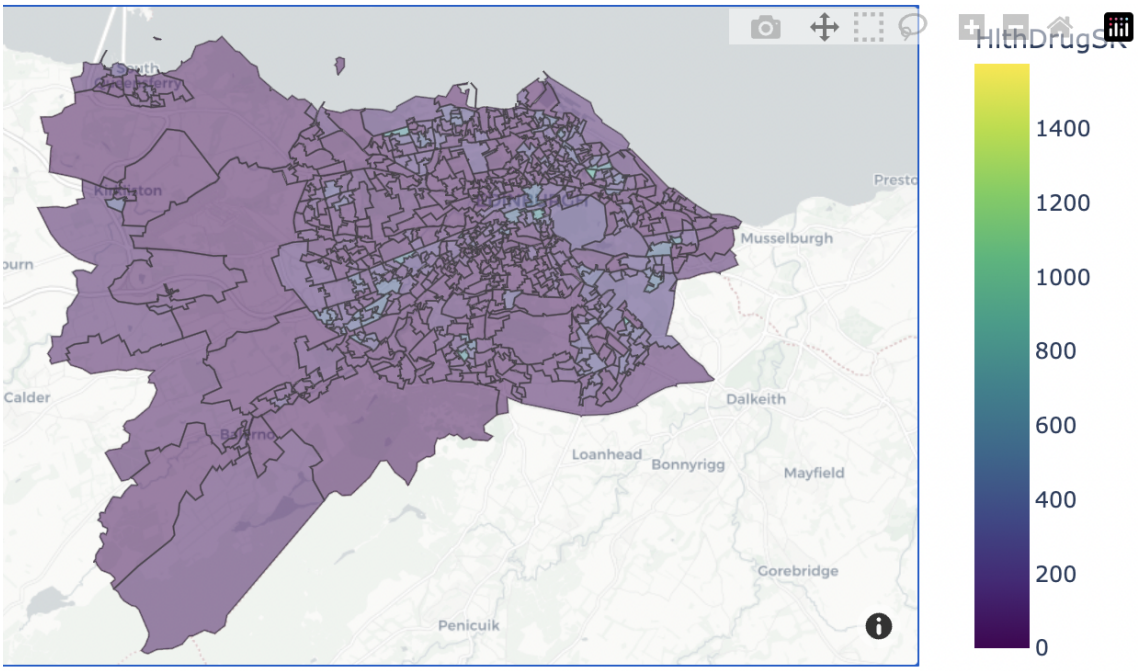
It looks like the range of the values for this attribute is 0 to 1575 for Glasgow and 0 to 863 for Edinburgh. In order to show the differences in Healthcare Stays Related to Drug Use, an appropriate range to take would be 0 to 1575 to include all the data.

3. Setting the CRS

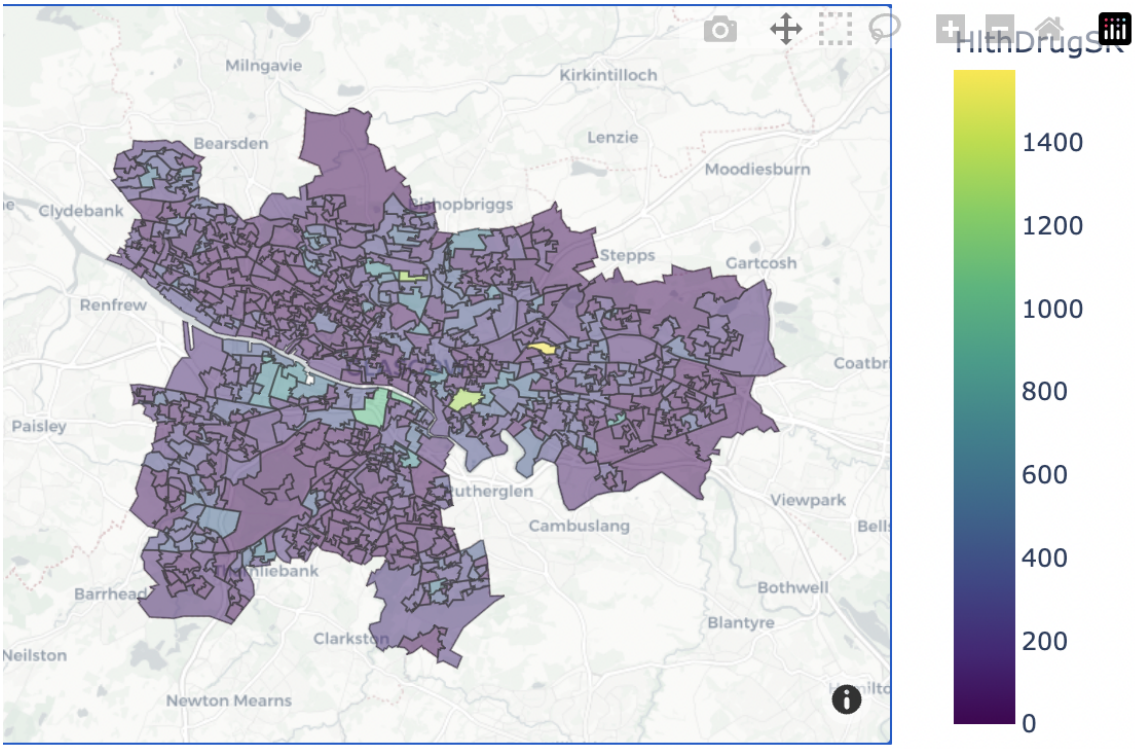
In [ ]:
```
# Important to check the crs first.
edi_od_gdf = edi_od_gdf.to_crs('EPSG:4326') # Just ensuring that the CRS is
print(edi_od_gdf.crs)
print(glas_od_gdf.crs)
```

4. Interactive Maps Using Mapbox

In [ ]:
```
# Plotting the number of health stays related to drug use in Edinburgh by
fig = px.choropleth_mapbox(edi_od_gdf,
                           geojson=edi_od_gdf.geometry,
                           locations=edi_od_gdf.index,
                           color="HlthDrugSR",
                           color_continuous_scale="Viridis",
                           range_color=(0, 1575), # Although the data for ed
                                                  # so that the colors are c
                           opacity=0.5,
                           center={"lat": 55.954390, "lon": -3.187273}, # Re
                           mapbox_style="carto-positron",
                           zoom=9.5)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



In [ ]:
```
fig = px.choropleth_mapbox(glas_od_gdf,
                           geojson=glas_od_gdf.geometry,
                           locations=glas_od_gdf.index,
                           color="HlthDrugSR",
                           color_continuous_scale="Viridis", # Viridis will
                           range_color=(0, 1575),
                           opacity=0.5,
                           center={"lat": 55.866193, "lon": -4.258246}, # Ch
                           mapbox_style="carto-positron",
                           zoom=9.5)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



In both Edinburgh and Glasgow, there are specific datazones that are hotspots for hospital stays related to drug use. However, the number of drug related stays in Glasgow is nearly two times higher than in Edinburgh. While this might reflect different social characteristics of the two cities, it should be noted that Glasgow is slightly larger than Edinburgh in population, which could explain some of this difference.