Geospatial Data and Graphing

San Francisco District Attorney's Office

Siddharth Chattoraj

August 22, 2023

Contents

1 Spatial Joins

1.1 Files

Shapefiles — These are the files that you would most commonly use. Each link contains a downloadable zip file with 4 files. The .shp is the actual shapefile, but you must include all 4 files from the zip file in the same directory since the .shp file relies on the other three.

San Francisco Neighborhoods Shapefile
San Francisco Zip Codes Shapefile
San Francisco Supervisor Districts Shapefile
San Francisco Police Districts Shapefile

GeoJSON Files — Sometimes a coding language or graphing tool will require a GeoJSON file. Relevant GeoJSON files can be downloaded below.

San Francisco Neighborhoods GeoJSON
San Francisco Zip Codes GeoJSON
San Francisco Supervisor Districts GeoJSON
San Francisco Police Districts GeoJSON

Data Sources — The sources below are helpful resources to find or analyze new data.

San Francisco Neighborhood Population Data
San Francisco Open Data Portal
Census Data Portal

1.2 Location Mapping and Spatial Joins

Data For Example Below: Incident Locations.csv

Steps:

1. Read .csv file.
2. Convert longitude (x) and latitude (y) coordinates to geometry vectors using 4326 (standard) coordinate reference system.
3. Import zip code shapefile and convert coordinate reference system of the polygon zip code areas to 4326.
4. Conduct a spatial join by mapping longitude and latitude coordinate geometry vectors to zip code polygons.
5. Repeat process for supervisor district, neighborhoods, and police districts.
6. Remove duplicate columns that occur as a result of multiple spatial joins.
7. Save .csv file.

```
install.packages("sf")
install.packages("dplyr")
install.packages("readr")

library(sf)
library(dplyr)
library(readr)

# Read locations data
incident_locations <- read_csv("Data/Incident Locations.csv")
incident_locations$INCIDENT_NO <- as.character(incident_locations$INCIDENT_NO)
```

```r
# Create geometry column
incident_locations <- st_as_sf(incident_locations, coords = c("LON", "LAT"), crs =
                                          4326)

# Import San Francisco Zip Code Shapefile
gdf_zips <- st_read("Data/Zip Codes/geo_export_68affbe4-ce61-4d6c-b707-ce1a0ebed55d.
                                          shp")

# Check and set CRS if they are different
if (st_crs(incident_locations) != st_crs(gdf_zips)) {
  gdf_zips <- st_transform(gdf_zips, crs = st_crs(incident_locations))
}

# Spatial Join
joined_data1 <- st_join(incident_locations, gdf_zips, join = st_intersects)

# Import San Francisco Supervisor Districts Shapefile
gdf_sup <- st_read("Data/Supervisor Districts/geo_export_323aac1a-7340-4c45-a771-
                                          a9d2084355b0.shp")

# Check and set CRS if they are different
if (st_crs(incident_locations) != st_crs(gdf_sup)) {
  gdf_sup <- st_transform(gdf_sup, crs = st_crs(incident_locations))
}

# Spatial Join
joined_data2 <- st_join(incident_locations, gdf_sup, join = st_intersects)

# Import San Francisco Neighborhoods Shapefile
gdf_nhoods <- st_read("Data/Neighborhoods/geo_export_d783cca7-4b0f-4bf8-9c5a-
                                          4018d6431a44.shp")

# Check and set CRS if they are different
if (st_crs(incident_locations) != st_crs(gdf_nhoods)) {
  gdf_nhoods <- st_transform(gdf_nhoods, crs = st_crs(incident_locations))
}

# Spatial Join
joined_data3 <- st_join(incident_locations, gdf_nhoods, join = st_intersects)

# Import San Francisco Police Districts Shapefile
gdf_police_districts <- st_read("Data/Police Districts/geo_export_d75d86f0-92e8-499f-
                                          b896-07c12a786b06.shp")

# Check and set CRS if they are different
if (st_crs(incident_locations) != st_crs(gdf_police_districts)) {
  gdf_police_districts <- st_transform(gdf_police_districts, crs = st_crs(
                                            incident_locations))
}

# Spatial Join
joined_data4 <- st_join(incident_locations, gdf_police_districts, join = st_intersects
                                          )

# Identify overlapping columns
overlapping_columns <- intersect(colnames(joined_data1), colnames(joined_data2))
overlapping_columns <- union(overlapping_columns, intersect(colnames(joined_data1),
                                          colnames(joined_data3)))
overlapping_columns <- union(overlapping_columns, intersect(colnames(joined_data1),
                                          colnames(joined_data4)))

# Exclude overlapping columns from subsequent joins
non_overlapping_joined_data2 <- joined_data2[, !(colnames(joined_data2) %in%
                                          overlapping_columns)]
non_overlapping_joined_data3 <- joined_data3[, !(colnames(joined_data3) %in%
                                          overlapping_columns)]
non_overlapping_joined_data4 <- joined_data4[, !(colnames(joined_data4) %in%
                                          overlapping_columns)]

# Combine the Spatial Data Frames
combined_data <- cbind(joined_data1, non_overlapping_joined_data2,
```

```
                                          non_overlapping_joined_data3,
                                          non_overlapping_joined_data4)

# Drop unnecessary columns
combined_data <- combined_data[, !(colnames(combined_data) %in% c('geometry.1', '
                                          geometry.2', 'geometry.3', 'objectid', '
                                          st_length_', 'company', 'shape_le_1', '
                                          shape_leng', 'index_right', 'multigeom', '
                                          id', 'date_dat_2', 'pop10_sqmi', 'pop2010',
                                           'sqmi', 'st_area_sh', 'state', 'zip_code',
                                           'time_dat_2', 'sup_dist_n', 'sup_dist_2',
                                          'sup_dist_p', 'po_name', 'date_data_', '
                                          time_data_'))]

# Write to CSV
write_csv(combined_data, "Outputs/combined_data.csv")

View(combined_data)
```

1.3 Map Plots With Python Plotly

Data For Example Below: Sharepoint → DAT-CSU-Exec DA Stat → Documents → Rebooking → Report → Outputs → rebooking_incidents.csv

Steps:

1. Set time, location, and date and filter the data frame based on those variables
2. Calculate the midpoint of longitude (x) and latitude (y) coordinates in order to center the map on the selected supervisor district.
3. Set density scale. Radius refers to how big of an area is covered when counting the total number of occurrences in a specific region. Smaller radii result in smaller clusters, which is better for smaller amount of data or viewing specific details. Larger radii result in larger clusters, which is better for larger amounts of data or overarching data analysis.
4. Set density formula: It calculates the sum of squared distances between given latitude and longitude coordinates and a midpoint, and then ot checks if this sum is less than or equal to the square of a given radius. This type of formula is used to determine if points are within a certain distance (radius) from a central point (midpoint).

   $$\sum \left((\textbf{latitude} - \textbf{latitude\_midpoint})^2 + (\textbf{longitude} - \textbf{longitude\_midpoint})^2 \right) \leq \textbf{radius}^2$$

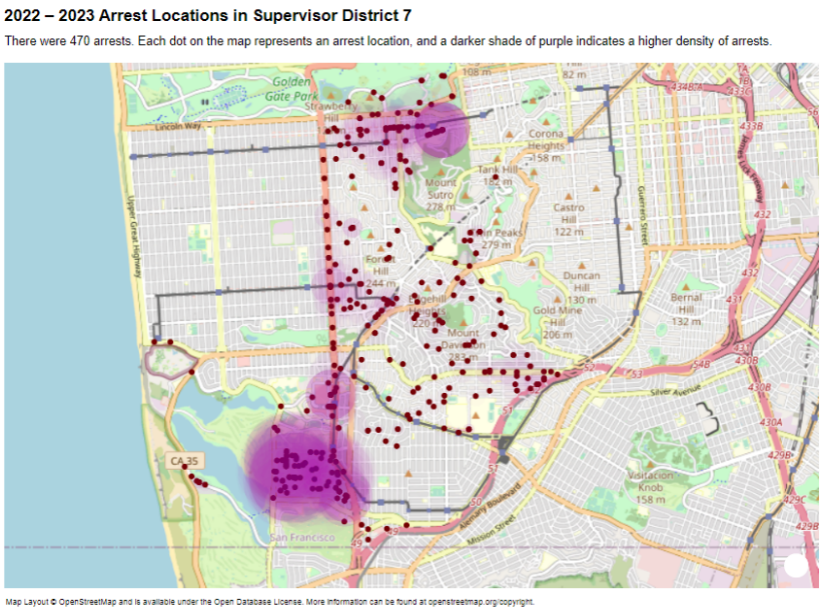
5. Create a scatter plot map of the locations and bubbles for the overlaid density graph.
6. Set graph layout information and use OpenStreetMap as the map background. Per their terms, a copyright notice must be listed that both gives credit to OpenStreetMap for the map background and has a link back to their copyright policy.
7. Save graph!

```
# Set time, location, and date
timeframe_description = "2022 - 2023"
sup_dist = 7
start_date = "2022-01-01"
end_date = "2023-06-30"

# Filter the DataFrame
locations = rebooking_incidents[
    (rebooking_incidents.sup_dist == sup_dist) &
    (rebooking_incidents.OCCUR_FROM_DATE_TM >= start_date) &
    (rebooking_incidents.OCCUR_FROM_DATE_TM <= end_date)
]
locations = locations[['Longitude', 'Latitude']]
locations.reset_index(drop=True, inplace=True)
```

```
# Calculate the physical midpoint of the locations
mid_lat = np.mean(locations['Latitude'])
mid_lon = np.mean(locations['Longitude'])

# Define the color scale for the density (you can calculate this based on your data)
color_scale = ['#730000'] * len(locations)

# Define the radius for the density bubbles (adjust this as per your requirement)
radius = 0.005  # For example, set a radius of 1 degree

# Calculate the density for each location
density = []
for i, location in locations.iterrows():
    lat = location['Latitude']
    lon = location['Longitude']
    num_points_within_radius = sum(
        ((locations['Latitude'] - lat) ** 2 + (locations['Longitude'] - lon) ** 2) <=
                                                    radius ** 2
    )
    density.append(num_points_within_radius)

# Create a scatter plot of the points
scatter_map = go.Scattermapbox(
    lat=locations['Latitude'],
    lon=locations['Longitude'],
    mode='markers',
    marker=dict(
        size=6.5,
        color=color_scale,
    ),
    text='Density:',  # You can customize this to show additional information.
)

# Create bubbles for the density
density_bubbles = go.Scattermapbox(
    lat=locations['Latitude'],
    lon=locations['Longitude'],
    mode='markers',
    marker=dict(
        size=density,
        sizemode='diameter',
        sizeref=max(density) / 100,  # Adjust the reference size for the bubbles
        opacity=0.02,  # Opacity
        color = "purple"
    ),
    hovertext=density,  # You can customize this to show additional information.
)

# Zoom Size
if (sup_dist == 4) or (sup_dist == 7) or (sup_dist == 8) or (sup_dist == 10):
    zoom_size = 12
elif sup_dist == 9:
    zoom_size = 11.75
else:
    zoom_size = 13

# Subtitle Text
subtitle_text = f"There were {len(locations)} arrests. \
Each dot on the map represents an arrest location, and a darker shade of purple
                                        indicates a higher density of arrests."

# Title Text
if sup_dist == 6:
    title_text = f"<b>{timeframe_description} Arrest Locations in Supervisor District
                                            {sup_dist} (Not Pictured: Treasure
                                            Island/Yerba Buena Island)"
else:
    title_text = f"<b>{timeframe_description} Arrest Locations in Supervisor District
                                            {sup_dist}"
```

```
# Define the layout for the map
layout = go.Layout(
    mapbox_style="open-street-map",
    hovermode='closest',
    mapbox=dict(
        center=dict(
            lat=mid_lat,
            lon=mid_lon,
        ),
        zoom=zoom_size,
    ),
    height=575,  # Increase the height of the graph
    width=870,  # Increase the width of the graph
    title=dict(
        text=title_text,
        font=dict(
            family="Arial",
            size=16,
            color="black",
        ),
        xanchor="left",  # Left-aligned
        x=0.0825,  # Left-aligned
        y=0.9875
    ),
    annotations=[
        dict(
            text=subtitle_text,
            showarrow=False,
            font=dict(
                family="Arial",
                size=12,
                color="black",
            ),
            x=0.08,  # Left-aligned
            y=1.06,  # Adjust the position of the subtitle
            xref="paper",
            yref="paper",
        ),
        dict(
            text=f"Map Layout    OpenStreetMap and is available under the Open
                                                    Database License. More
                                                    information can be found at \
openstreetmap.org/copyright.",
            showarrow=False,
            font=dict(
                family="Arial",
                size=9.5,
                color="black",
            ),
            x=0.08,  # Left-aligned
            y=-0.03,  # Adjust the position of the subtitle
            xref="paper",
            yref="paper",
        )
    ],
    plot_bgcolor='rgba(0, 0, 0, 0)',  # Transparent background
    paper_bgcolor='rgba(0, 0, 0, 0)'  # Transparent background
)

# Create the figure and add the scatter plot and density bubbles to it
mappy = go.Figure(data=[scatter_map, density_bubbles], layout=layout)

mappy.update_layout(margin=dict(l=0, r=0, t=60, b=0),
                    showlegend=False)

# Write to JSON
mappy.write_json('Sup JSON/map.json')

# Show the map
mappy.show()
```

Graph — Plotly (Python):



1.4 Alternative Example (R): SF Population and Neighborhood Mapping

Using left-joins and merges to Map Geographical Data Based on Categorization — Census tracts are already mapped by neighborhood in the Analysis Neighborhoods Data

Neighborhood Data: Analysis_Neighborhoods_-_2020_census_tracts_assigned_to_neighborhoods.csv
Population Data: ACSDP5Y2021.DP05-2023-07-05T211846.csv

Steps:

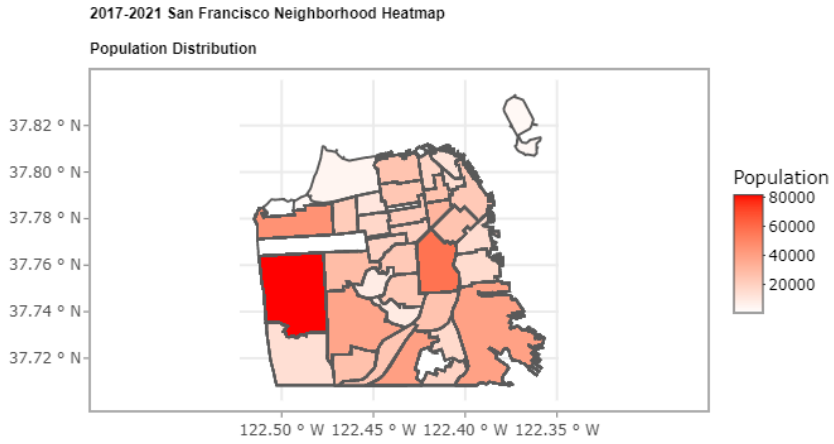
1. Read and format neighborhood location data and population data. Pull relevant census tract population data from the large .csv file.
2. Left-join the population data with the neighborhood data.
3. Merge the data frame created in Step 2 with the San Francisco Neighborhoods shapefile on the neighborhood name column.
4. Calculate each neighborhoods' total population via aggregation and convert to a simple features (sf) object for graphing.
5. Create a population heatmap using ggplot2.
6. To improve the design of the heatmap, convert the ggplot2 heatmap to a Plotly heatmap and edit relevant layout settings.
7. Display the graph and population statistics!

```
# San Francisco Neighborhood Populations

# Packages
library(plotly)
library(rgdal)
library(stringr)
library(sf)
library(ggplot2)

# Read the neighborhood data
```

```r
neighborhoods <- read.csv("Data/Analysis_Neighborhoods_-
                                          2020_census_tracts_assigned_to_neighborhoods
                                          .csv")
neighborhoods$name <- as.numeric(neighborhoods$name) # Convert Census Tract Values to
                                          Float

# Population Data
pop_copy <- read.csv("Data/ACSDP5Y2021.DP05-2023-07-05T211846.csv", stringsAsFactors =
                                           FALSE)
pop <- as.data.frame(t(pop_copy), stringsAsFactors = FALSE)
colnames(pop) <- pop[1, ]
pop <- pop[-1, ]
pop <- pop[, -1]  # Delete the first column
pop$label <- row.names(pop)
row.names(pop) <- NULL
pop <- pop[, c("label", names(pop)[-ncol(pop)])]

pattern <- "Census\\.Tract\\.\\d+(\\.\\d+)?\\.\\.San\\.Francisco\\.County\\.\\.
                                          California\\.\\.Estimate"
pop <- pop[grepl(pattern, pop$label), ]

pop <- pop %>%
  tibble::rownames_to_column(var = "index")
pop <- pop[, -1]  # Delete the first column

pop <- pop[, 1:2]
colnames(pop)[2] <- "population"

pop$population <- gsub(",", "", pop$population)
pop$population <- as.integer(pop$population)
pop$label <- str_extract(pop$label, "\\d+(\\.\\d+)?")
pop$label <- as.numeric(pop$label)

# Left-join to join population with neighborhoods
neighborhoods_pop <- merge(pop, neighborhoods, by.x = "label", by.y = "name", all.x =
                                          TRUE)

# Pull relevant columns
neighborhoods_pop_df <- neighborhoods_pop[, c("label", "population", "the_geom", "
                                          neighborhoods_analysis_boundaries")]
neighborhoods_pop_df <- na.omit(neighborhoods_pop_df)

# Read shapefile of San Francisco Neighborhoods
gdf <- st_read(dsn = "Data/geo_export_d783cca7-4b0f-4bf8-9c5a-4018d6431a44.shp")

# Merge the datasets based on the neighborhood column
merged_df <- merge(gdf, neighborhoods_pop_df, by.x = "nhood", by.y = "
                                          neighborhoods_analysis_boundaries")

# Calculate the cumulative population for each neighborhood
cumulative_population <- aggregate(merged_df$population, by = list(merged_df$nhood),
                                          FUN = sum)
colnames(cumulative_population) <- c("Neighborhood", "Population")
cumulative_population <- cumulative_population %>%
  arrange(desc(Population))

# Create a new dataframe with the cumulative population values
gdf_heatmap <- data.frame(nhood = merged_df$nhood, geometry = merged_df$geometry)
gdf_heatmap <- merge(gdf_heatmap, cumulative_population, by.x = "nhood", by.y = "
                                          Neighborhood", all.x = TRUE)

# Total Population of San Francisco
cat("San Francisco Total Population:", sum(cumulative_population$Population), "\n")

# Convert the gdf_heatmap data frame to an sf object
gdf_heatmap_sf <- st_as_sf(gdf_heatmap)

# Create the ggplot heatmap
heatmap_plot <- ggplot() +
  geom_sf(data = gdf_heatmap_sf, aes(fill = Population, text = paste("Neighborhood: ",
                                             nhood, "<br>Population: ", Population)))
```

```
                                                    +
  scale_fill_gradient(low = "white", high = "red") +
  labs(fill = "Population") +
  ggtitle("<b><span style='font-size:12px'>2017-2021 San Francisco Neighborhood
                                                    Heatmap\nPopulation Distribution") +
  theme_bw() +
  theme(plot.title = element_text(family = "Arial", size = rel(1.5)))

# Convert ggplot to plotly
heatmap_plotly <- ggplotly(heatmap_plot, tooltip = "text")

# Specify the latitude and longitude range for zooming
lat_range <- c(37.75, 37.8)
lon_range <- c(-122.45, -122.4)

# Modify plotly layout settings
heatmap_plotly <- heatmap_plotly %>% layout(
  xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = TRUE, linecolor =
                                                    "transparent"),
  yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = TRUE, linecolor =
                                                    "transparent"),
  margin = list(l = 20, r = 20, t = 80, b = 20),
  plot_bgcolor = "white"
)

# Display the interactive heatmap plot
heatmap_plotly

# Total Population of San Francisco
cat("San Francisco Total Population:", sum(cumulative_population$Population), "\n")
write.csv(cumulative_population, file = "Outputs/sf_population.csv", row.names = FALSE
                                                    )
```

Graph — Plotly (R):



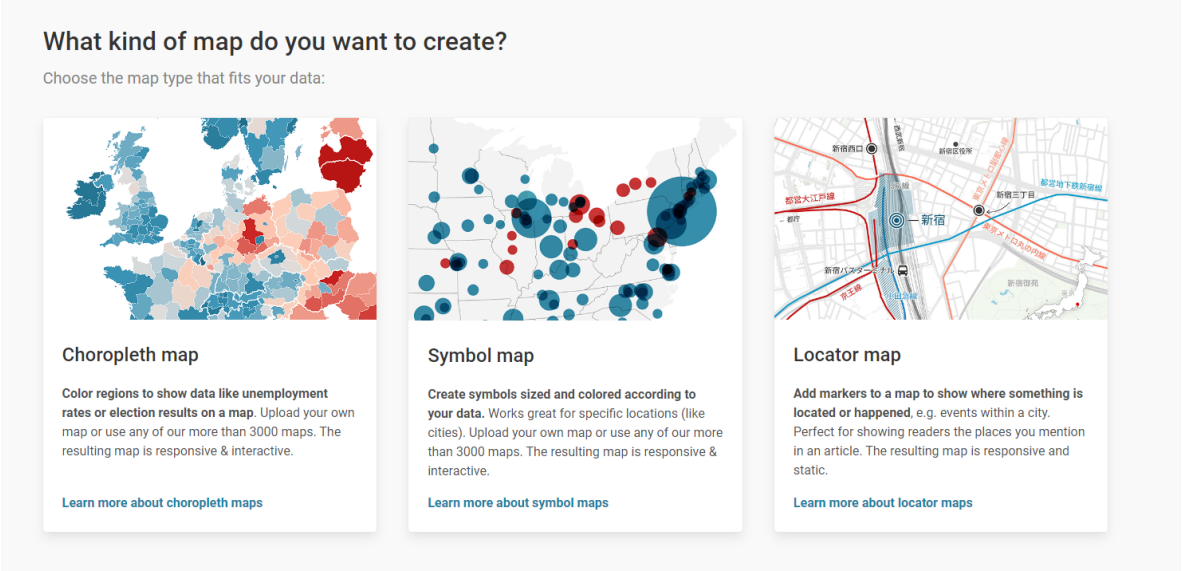
2 Geospatial Mapping With Datawrapper

2.1 Introduction

Visit Datawrapper and either create a chart first or create an account first. You will eventually have to create an account for maximum features, but the account is completely free.

To create a map — Note: This process is slightly different than for a normal chart — press the green "Start creating" button, then move your cursor to the top right corner and select "+ Create new", and then select "Map".

There are 3 types of maps available: choropleth, symbol, and locator. Choropleth maps are best for displaying and comparing the density of regional distributions. Symbol maps are best for displaying and comparing the density of individual locations. Locator maps are best for depicting a single location in detail.

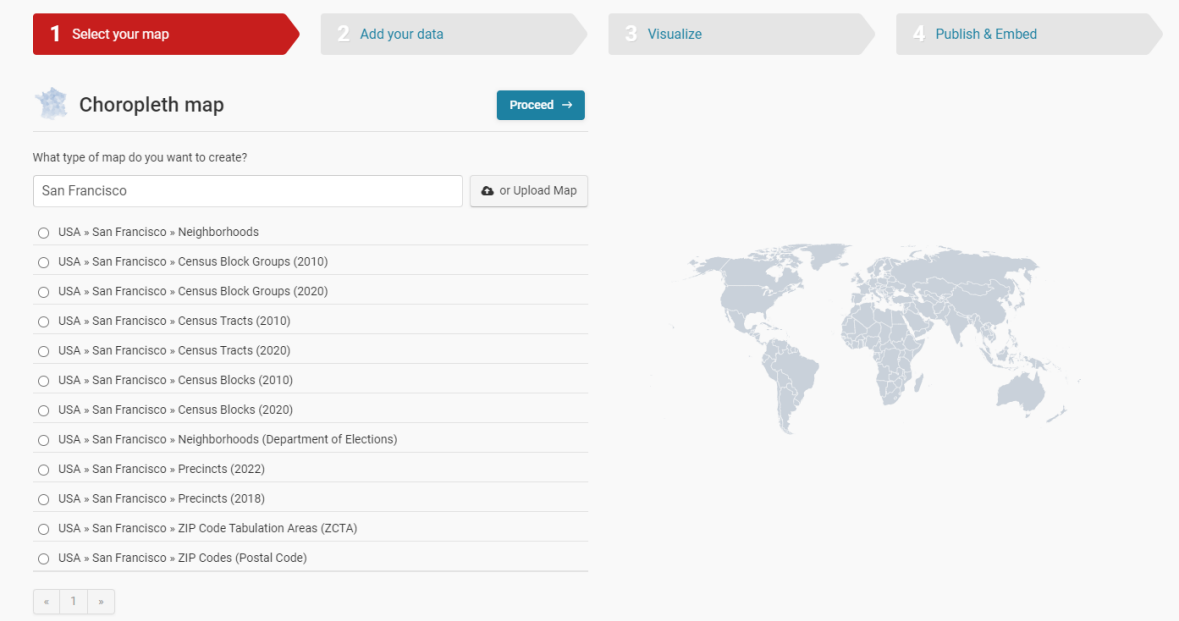


2.2 Choropleth and Symbol Maps

Steps:

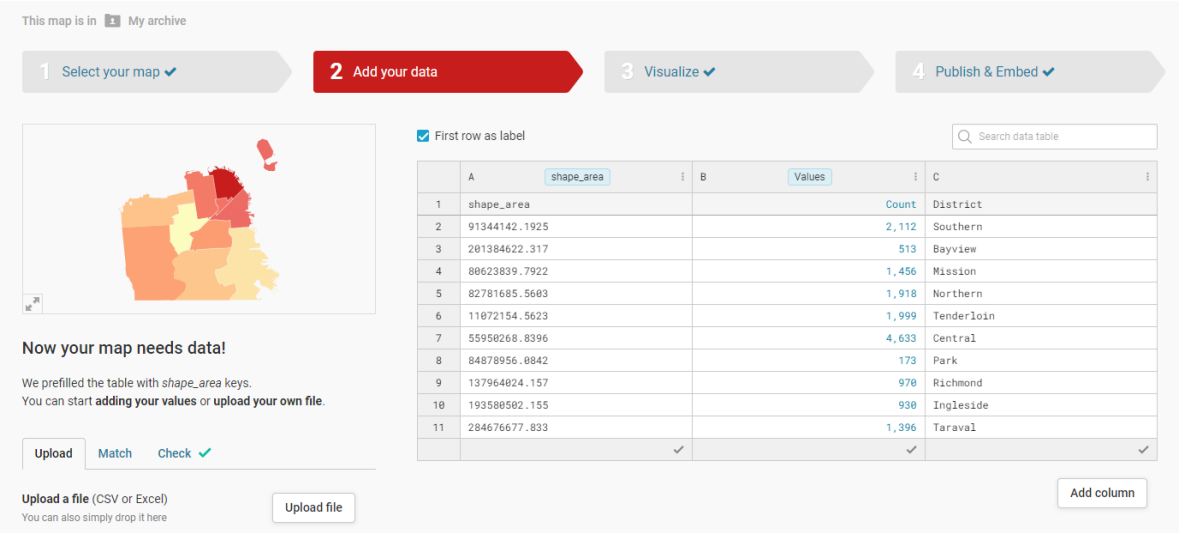
1. Select Your Map

   Choose your map or upload your own using a GeoJSON File (subsection 1.1).



2. Add your data

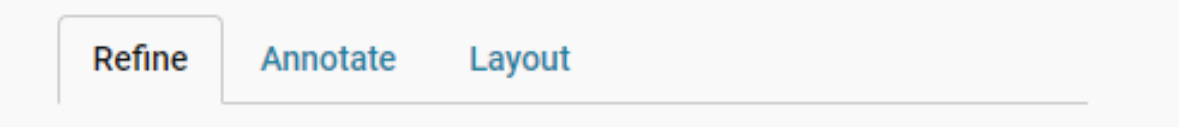
Upload your data as a .csv file, .xlsx file, or as a Google sheet that anyone can edit. Depending on the data, Datawrapper may ask you for shape area to complete the map. If you need this information, it is located in a column in the .shp file — you would need to extract and download it.



	A shape_area	B Values	C
1	shape_area	Count	District
2	91344142.1925	2,112	Southern
3	201384622.317	513	Bayview
4	80623839.7922	1,456	Mission
5	82781685.5603	1,918	Northern
6	11072154.5623	1,999	Tenderloin
7	55950268.8396	4,633	Central
8	84878956.0842	173	Park
9	137964024.157	970	Richmond
10	193580502.155	930	Ingleside
11	284676677.833	1,396	Taraval

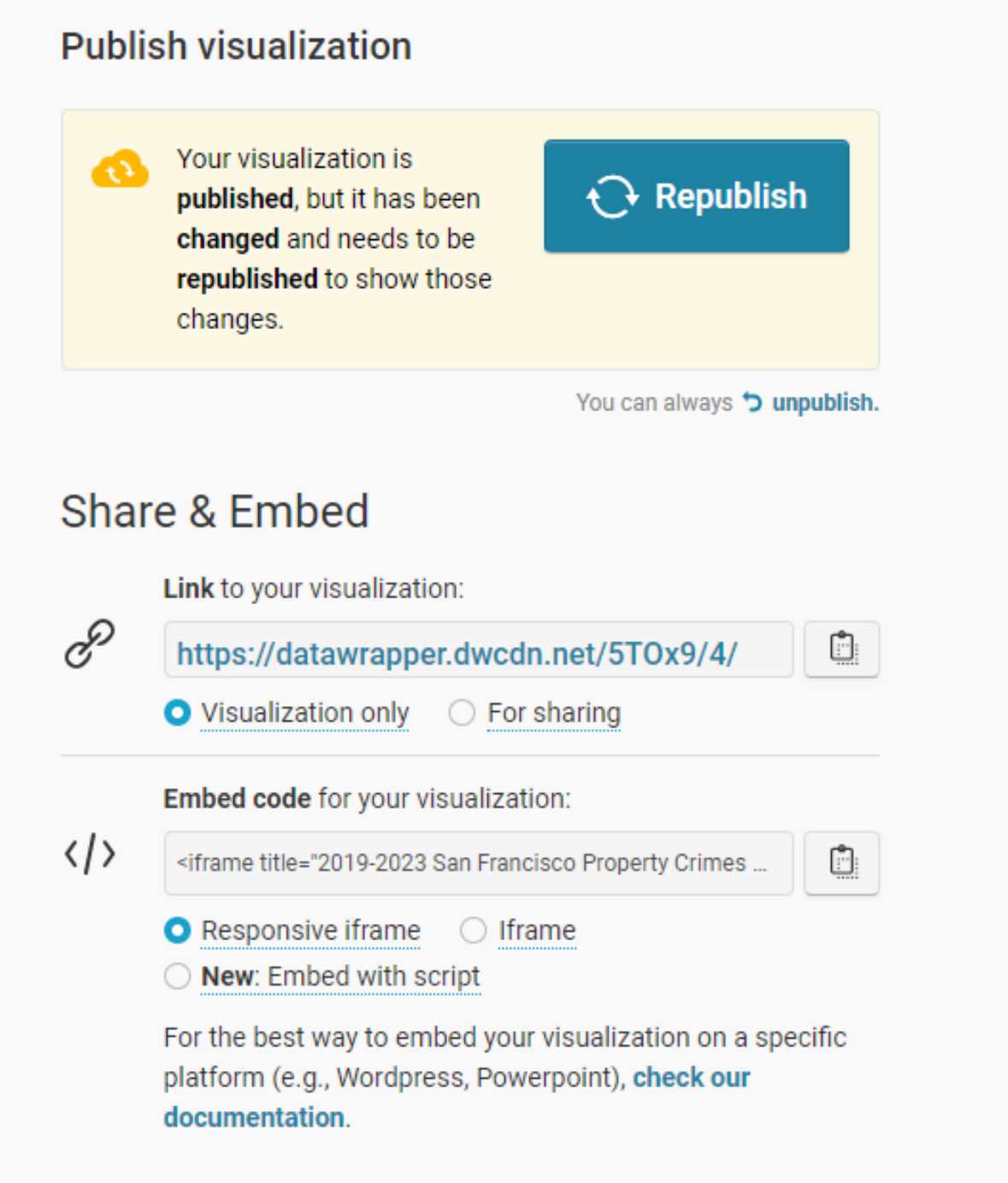
3. Visualize

In this section, there are 3 tabs: Refine, Annotate, and Layout. In the Refine tab for choropleth maps, you can change the color scheme, the formatting of labels and legends, and the appearance of the map. In the Refine tab for symbol maps, you can change the symbol shape and size, the symbol color scheme, and the appearance of the map. In the Annotate section, you can add and edit the title, subtitle, chart author, map labels, map annotations, and tooltips (hover features). In the Layout tab, you can edit the thematic features of the chart.



4. Publish & Embed (Need account for full features)

Publish your visualization to save your changes. You can access your visualization at any time via the dashboard. You can embed your visualization in another website or document by using the embed link or iframe code present. If you would like to export the visualization as an image, you can edit the size, whether or not the title/footer is included, and the background (transparent or regular). You can also make a copy of the visualization to if you would like to edit a new version.



2.3 Locator Maps

Datawrapper has created a comprehensive guide to creating locator maps.

3 ArcGIS Pro

3.1 Geocoding (Translating Addresses to Locations)

Required Elements:

- House Number
- Street Address (Abbreviations like St or Ave are fine)
- City
- State

Example: 2526 Hyde Street, San Francisco, CA

Steps:

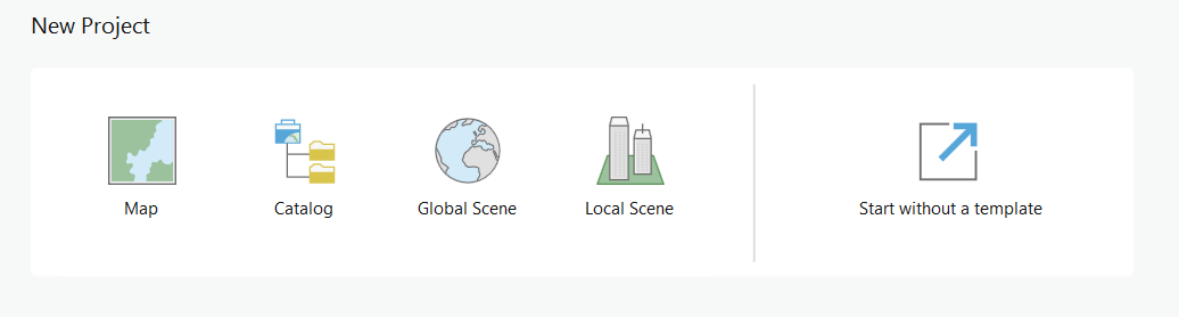
1. Download a .csv file to your computer that has a column containing the required concatenated elements.
2. Open ArcGIS Pro and click "Map" under "New Project" and then press "OK".
3. Click on "Add Data" in the "Map" toolbar in the ribbon and then select your file from step 1.
4. Right-click on your file, which should be located in the Contents pane on the left. Select "Geocode Table".
5. Answer the 6 questions about the data to the best of your ability. Make sure to select "Address" in the "Address or Place" place in Step 3. If your address data is concatenated, you don't need to fill out anything else in Step 3. If not, fill out all the other fields you know. When you are done with all 6 steps, press "Estimate Credits" at the top of the catalog pane, and then click "Run" at the bottom of the pane.
6. Press "Yes" to rematch process, which gives you the chance to fix any errors if need be.
7. Assuming there are no other errors, you should be able to view your map. You can make change to the symbology per the instructions listed in section 2.

3.2 Dot Plots and Density Maps

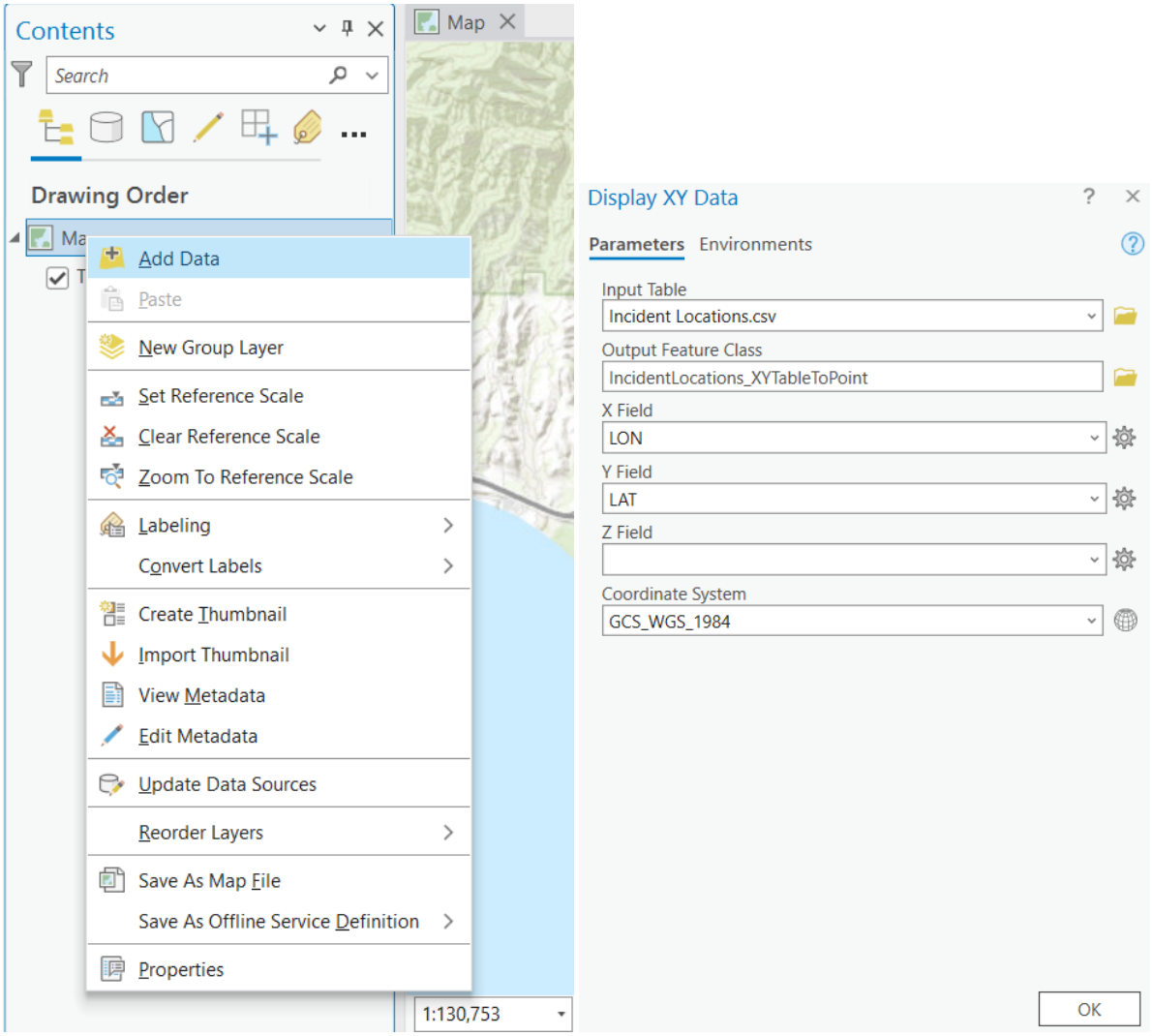
When graphing in ArcGIS, there are two main panes: Contents (left) and Catalog Pane (right). Occasionally, you may want to close one for better viewing. To re-add them, go to the ribbon and press "View" and then either press "Reset Panes" (default version) or press "Catalog Pane" and "Contents". If you want to create a new map tab within a project, go to the ribbon and press "Insert" and then press "New Map".

Steps:

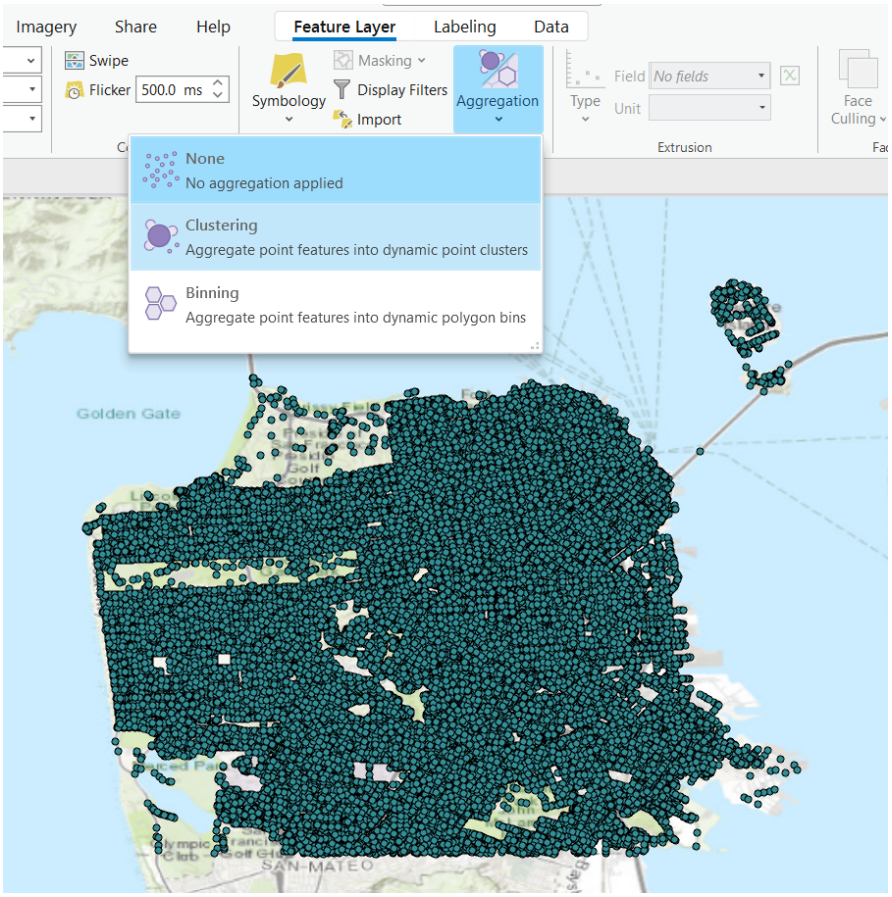
1. Open ArcGIS Pro and create a new project. Select "Map" when presented with a list of different types of maps and scenes to choose.



2. The scale of the map is in the bottom left corner, and it can be adjusted by altering the ratio or by scrolling in and out on the map. To add points, in the Contents pane on the left, right-click on Map and press "Add Data". After you add the data, a file will pop up in the Contents Pane. Right-click that file and press "Display XY Data". Check to make sure the fields are correct (longitude is X and latitude is Y), and leave the Coordinate System box as is.



3. A dot plot of your locations will appear. You may use this plot, or you can add a density cluster feature by pressing "Feature Layer" in the ribbon, then selecting "Aggregation" in the Drawing tab, and then pressing "Clustering". **The clustering feature will remove the existing dot map. You will need to repeat Step 2 — but you must rename Output Feature Class to something else — to add the dots to the cluster map. Ensure that the dots file is below the cluster file in the contents pane so that the dots do not obstruct the clusters.**

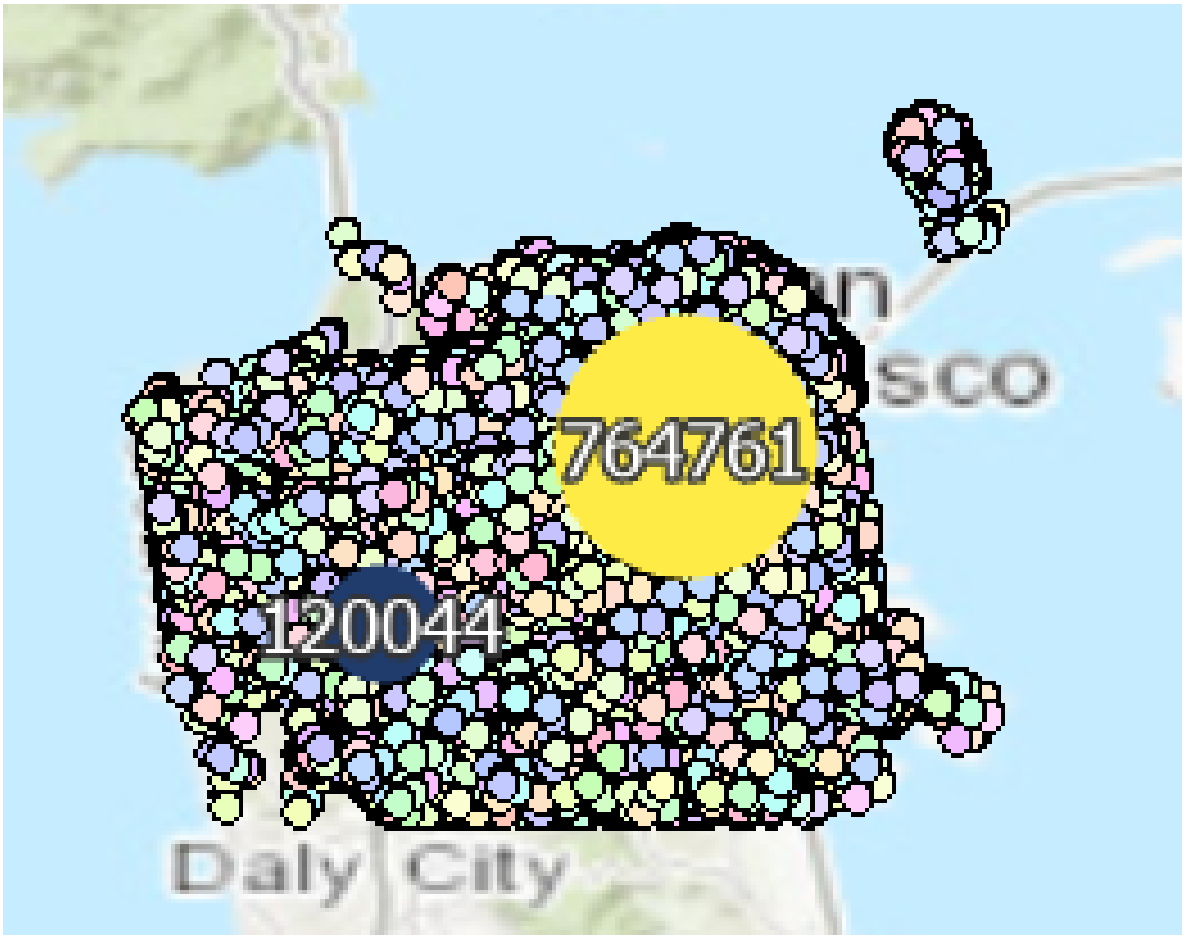


4. You can change the color of all of the dots or clusters by **right-clicking** the corresponding dot or cluster in the Contents pane. The recommended color scheme for crime data is No color features, Dark Umber (2nd column, 6th row in color palette) or Poinsettia Red (2nd column, 4th row in color palette) clusters and Medium Coral Light (2nd column, 2nd row in color palette) dots set to 50% (if small number of data) or 75% (if large number of data) transparency — select the color and then select "Color Properties" to edit.

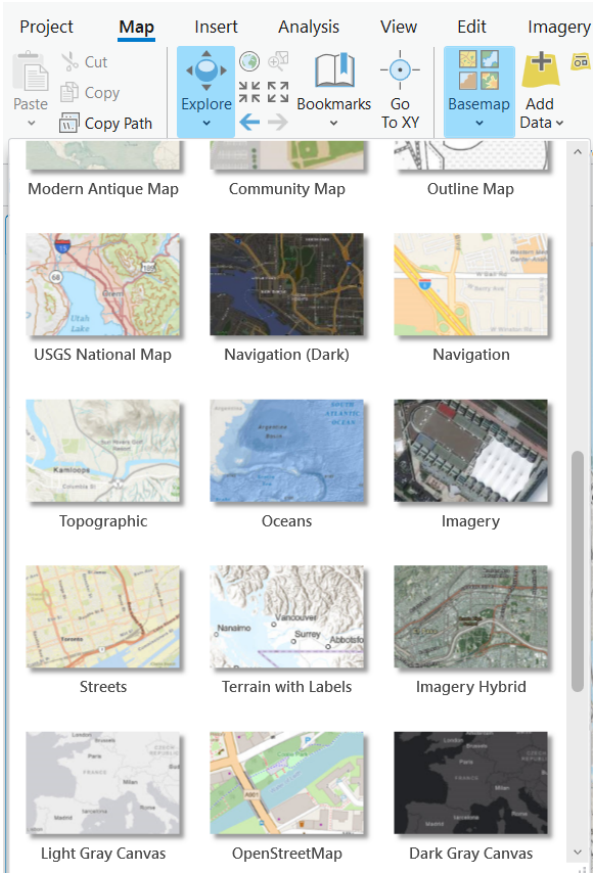
5. You can change the shape and symbol of all of the dots or clusters by **left-clicking** the corresponding dot or cluster in the Contents pane. The Gallery tab in the Symbology pane that

opens has a plethora of shapes and symbols. If you would like to change the size of the cluster, click the “Clustering” tab in the ribbon and select “Symbology. Then alter the minimum and maximum size in the Clusters tab of the Symbology pane that opens on the right.

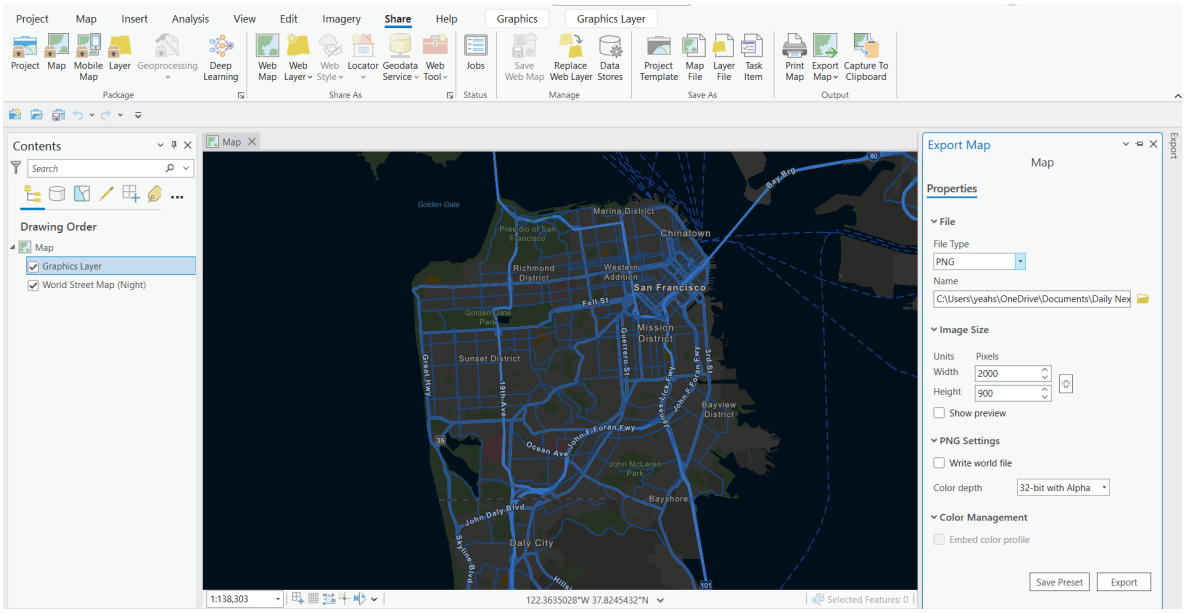
If you would like to make the clusters multiple colors to represent different attributes, follow the same steps as changing the size of the cluster but click the two squares logo in the Clusters tab of the Symbology pane and choose the corresponding Summary Field. If you would like to make the dots multiple colors to represent different attributes, left-click the dot in the Contents pane and then press ”Feature Layer“ in the ribbon, then select ”Symbology“ in the Drawing tab, and then press ”Unique Values“. Choose the appropriate fields in the first tab of the Symbology pane that will open on the right. This is best for a field with a small number of categories (ex: Felony vs. Misdemeanor or Tenderloin vs. South of Market) — the categories must be present in the data for the feature to work. If you have a large number of categories, then there is a large number of colors that will be drawn, creating a map that is not as clear (see picture below).



6. You can change the map layout by pressing ”Map“ in the ribbon and then pressing ”New Map“ and then choosing a different map background. OpenStreetMap and Streets (Night) are really good maps for visualizations. The NAIP Imagery Hybrid map is really good for showing data in a physical location.



7. Export the map by pressing "Share" in the ribbon, then pressing "Export Map", and then choosing Flattened PDF (images – most common) or Vector PDF (vector graphics). Change the file type to PNG, select your export directory, and ensure that you preview the size (you may have to go back and forth between "Width" and "Height" to prevent certain areas from being cut off. Change the scale in the bottom left corner if you wish. Press "Export" once done to export the map.



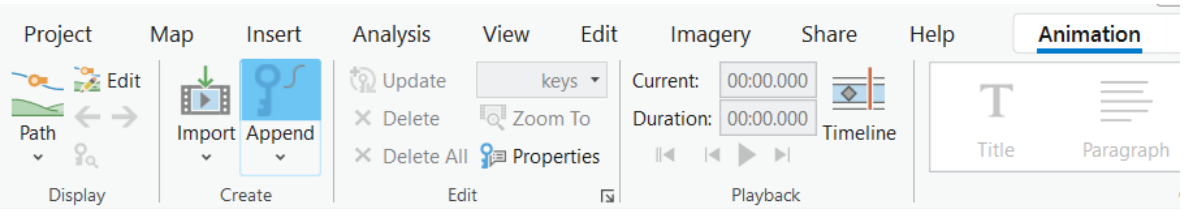
3.3 Animations and Timelapses

When making your map (subsection 3.2), ensure there is a column of temporal data to go along with the coordinates. After your map — whether it is a dot map, cluster map, heatmap, a mix, or most other types of map — is finished you can create the animation/timelapse and screen record (**Windows** — Windows Key + Alt + R/**Mac** — Shift + Command + 5 → Click Record) or export (Animation → Export → Movie).

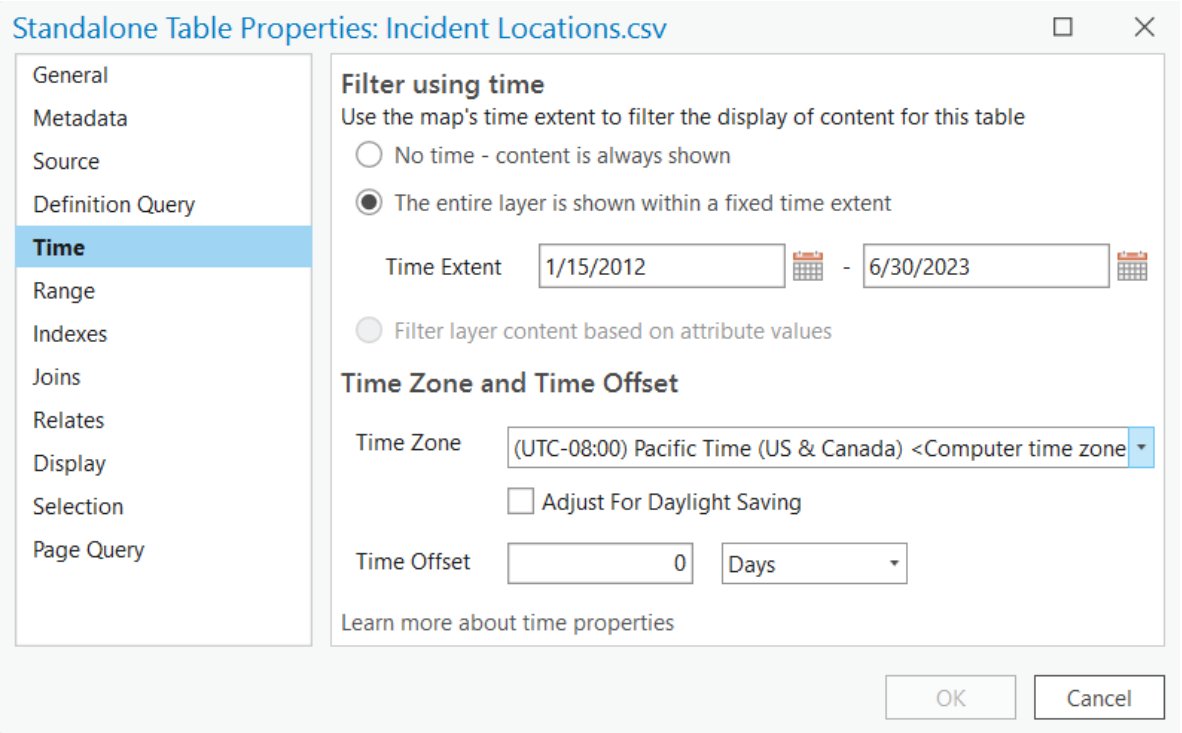
ArcGIS has created documentation explaining how to visualize temporal data. The documentation is detailed and has step-by-step instructions on how to create a variety of visualizations. A short-form summary of timelapses is below.

Steps:

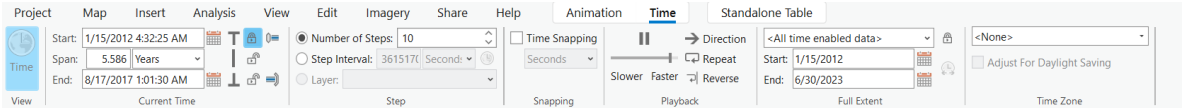
1. Once your map is created, type "Animation" in the Command Search bar at the top center of the screen. Select "Add (Create Animation)". You should now be in the "Animation" tab in the ribbon and you can close the keyframe bar that appears at the bottom.
2. If you would just like to move the camera around your map, move around the map and press "Append" until you are done. You can press the play button to watch it.



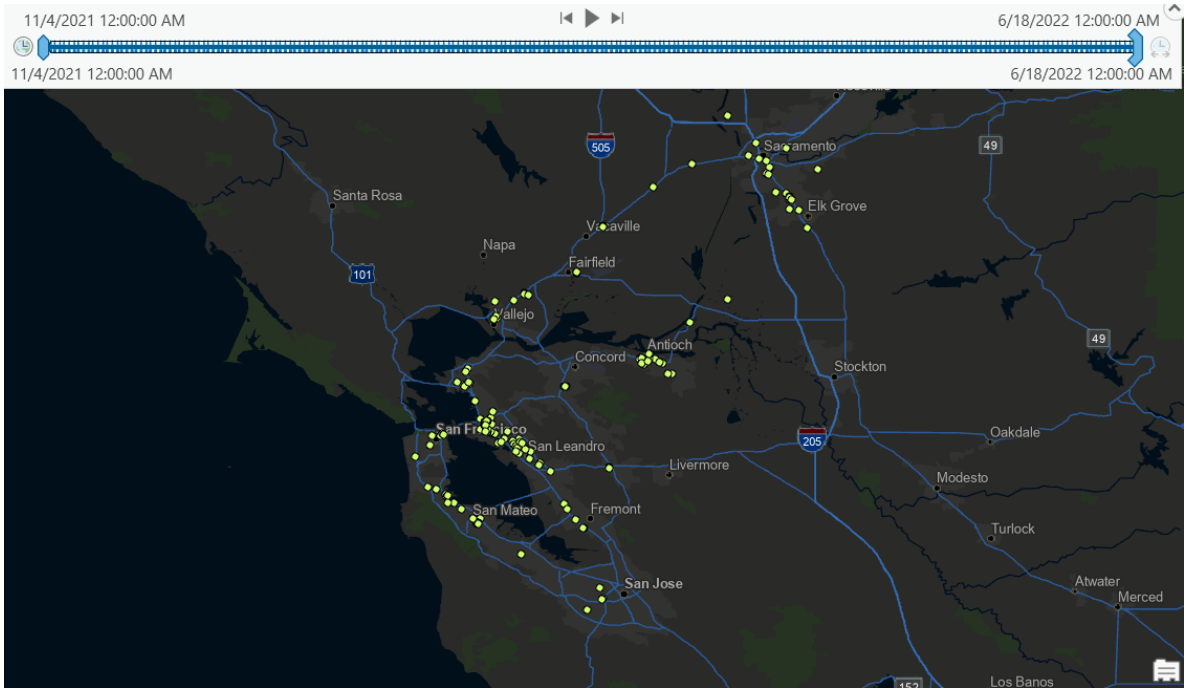
3. If you would like to make a timelapse, **double left-click** the data in the Contents panel, choose either the second or the third "Filter Using Time" bubble based on the type of data you have, and set the time extent to match your temporal data's range. Then visit the "Time" tab in the ribbon.



4. The timelapse will play if you press play in the timelapse bar that appears on the screen. but it is not cumulative. It will show the total number of occurrences within the "Span" set in the top left of the ribbon based on the "Number of Steps", "Start", and "End" selections. To create a timelapse that aggregates all data (cumulative over time, such as January – February, January – March, January – April, etc.), press the lock icon next to the start date in the upper left of the ribbon.



5. Screen record timelapse or append sections of the timelapse to create an animation!



4 Microsoft Excel

4.1 Geographical Time-lapse

This feature is very quick to use, but it only works if the data is inputted correctly. If Microsoft Excel's 3D Maps feature cannot identify your temporal data as a date or time or both, even if it as formatted as such with Excel, the feature will not work.

1. Insert time data and longitude and latitude coordinates in Excel. Add any categories you would like. Ensure that the time information is of type date or time. If not, select and right-click the column, press "Format Cells", and choose your preferred date/time format.
2. In the ribbon, click "Insert", and then click "3D Maps". If the feature is grayed out, it will not work. Once a "tour" (a map) is created, return to the Excel sheet, select your data, click "3D Maps", and then click "Add selected data to 3D Maps"
3. Click "Refresh Data" in the 3D Map window and a pane on the right should appear. Fill in the boxes with the corresponding information. Rename the layer name (highlighted) by clicking the pencil button.
4. You can add any additional features you would like, including another layer, map labels, a different layout (press "Themes" in the ribbon to choose), and more. Hovering over the different options on the ribbon at the top of the 3D Maps Feature provides information on what is available.
5. Once you are finished, press "Create Video" in the ribbon, and the video will export!