Urban Greenspace and Asthma Prevalence

Get Started with Big Data Pipelines

Vegetation has the potential to provide many ecosystem services in Urban areas, such as cleaner air and water and flood mitigation. However, the results are mixed on relationships between a simple measurement of vegetation cover (such as average NDVI, a measurement of vegetation health) and human health. We do, however, find relationships between landscape metrics that attempt to quantify the connectivity and structure of greenspace and human health. These types of metrics include mean patch size, edge density, and fragmentation.

Read More
Check out this study by @tsai_relationships_2019 on the relationship between edge density and life expectancy in Baltimore, MD. The authors also discuss the influence of scale (e.g. the resolution of the imagery) on these relationships, which is important for this case study.

In this notebook, you will write code to calculate patch, edge, and fragmentation statistics about urban greenspace in Chicago. These statistics should be reflective of the connectivity and spread of urban greenspace, which are important for ecosystem function and access. You will then use a linear model to identify statistically significant correlations between the distribution of greenspace and health data compiled by the US Center for Disease Control.

Working with larger-than-memory (big) data

For this project, we are going to split up the green space (NDVI) data by census tract, because this matches the human health data from the CDC. If we were interested in the average NDVI at this spatial scale, we could easily use a source of multispectral data like Landsat (30m) or even MODIS (250m) without a noticeable impact on our results. However, because we need to know more about the structure of green space within each tract, we need higher resolution data. For that, we will access the National Agricultural Imagery Program (NAIP) data, which is taken for the continental US every few years at 1m resolution. That’s enough to see individual trees and cars! The main purpose of the NAIP data is, as the name suggests, agriculture. However, it’s also a great source for urban areas where lots is happening on a very small scale.

The NAIP data for the City of Chicago takes up about 20GB of space. This amount of data is likely to crash your kernel if you try to load it all in at once. It also would be inconvenient to store on your harddrive so that you can load it in a bit at a time for analysis. Even if your are using a computer that would be able to handle this amount of

data, imagine if you were analysing the entire United States over multiple years!

To help with this problem, you will use cloud-based tools to calculate your statistics instead of downloading rasters to your computer or container. You can crop the data entirely in the cloud, thereby saving on your memory and internet connection, using `rioxarray`.

Check your work with testing!

This notebook does not have pre-built tests. You will need to write your own test code to make sure everything is working the way that you want. For many operations, this will be as simple as creating a plot to check that all your data lines up spatially the way you were expecting, or printing values as you go. However, if you don't test as you go, you are likely to end up with intractable problems with the final code.

STEP 1: Set up your analysis

Try It
As always, before you get started:

1. Import necessary packages
2. Create **reproducible file paths** for your project file structure.
3. To use cloud-optimized GeoTiffs, we recommend some settings to make sure your code does not get stopped by a momentary connection lapse – see the code cell below.

In [105…

```
# Import libraries
import pandas as pd
import rioxarray as rxr # Work with geospatial raster data
import geopandas as gpd
import pathlib
import os
import hvplot.pandas
import geoviews as gv
import cartopy.crs as ccrs
gv.extension('bokeh')  # Activates GeoViews with the Bokeh backend

# For the NDVI Data
import re # Extract metadata from file names
import zipfile # Work with zip files
from io import BytesIO # Stream binary (zip) files
import numpy as np # Unpack bit-wise Fmask
import requests # Request data over HTTP
import pystac_client
import rioxarray.merge as rxrmerge
import shapely
import xarray as xr
import glob
```

```
from cartopy import crs as ccrs
from scipy.ndimage import convolve
from sklearn.model_selection import KFold
from scipy.ndimage import label
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
```

In [85]:
```
# Create Reproducable File Paths
data_dir = os.path.join(
    # Home directory
    pathlib.Path.home(),
    # Earth analytics data directory
    'earth-analytics',
    'data',
    # Project directory
    'big-data',
)
os.makedirs(data_dir, exist_ok=True)
```

In [86]:
```
# Prevent GDAL from quitting due to momentary disruptions
os.environ["GDAL_HTTP_MAX_RETRY"] = "5"
os.environ["GDAL_HTTP_RETRY_DELAY"] = "1"
```

STEP 2: Create a site map

We'll be using the Center for Disease Control (CDC) Places dataset for human health data to compare with vegetation. CDC Places also provides some modified census tracts, clipped to the city boundary, to go along with the health data. We'll start by downloading the matching geographic data, and then select the City of Chicago.

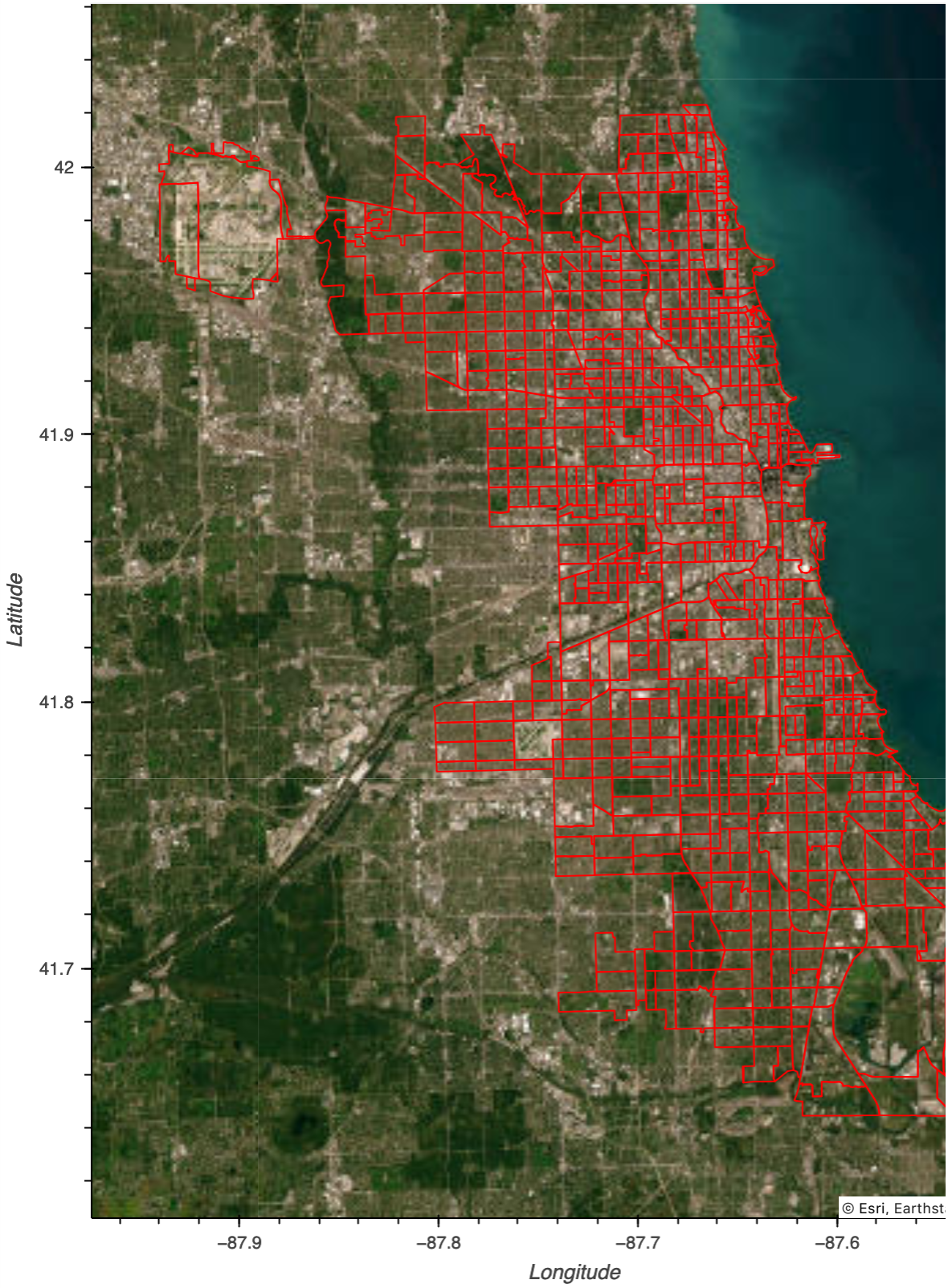
Try It

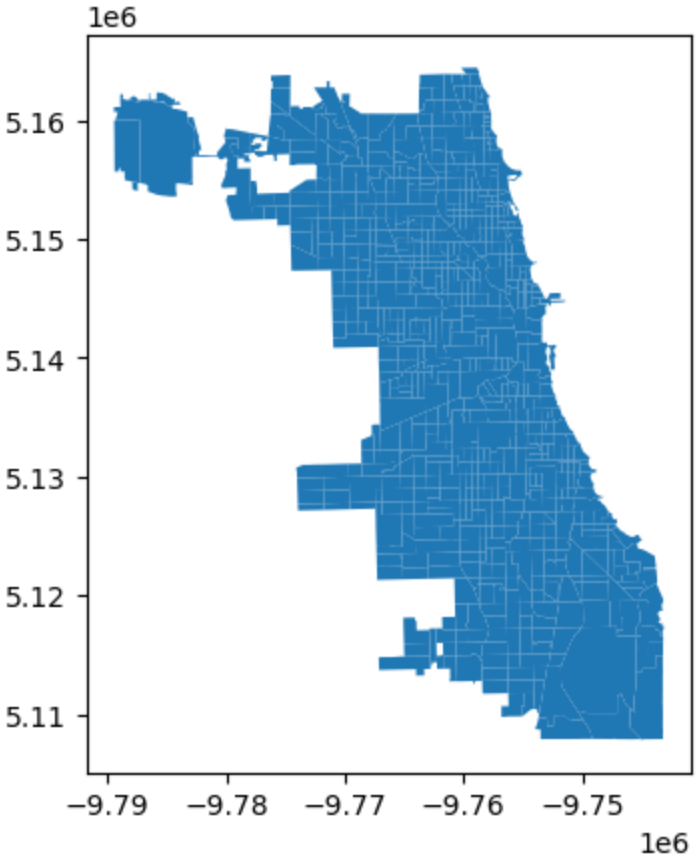
1. Download the Census tract Shapefile that goes along with CDC Places
2. Use a **row filter** to select only the census tracts in Chicago
3. Use a **conditional statement** to cache your download. There is no need to cache the full dataset – stick with your pared down version containing only Chicago.

In [87]:
```
#Create a file path for your CDC Tract Data

cdc_data_dir = os.path.join(data_dir, 'cdc_data')
os.makedirs(cdc_data_dir, exist_ok=True)
cdc_data_path = os.path.join(cdc_data_dir, '*.shp')

# Define info for CDC download, and only download cdc data once
if not os.path.exists(cdc_data_path):
    cdc_data_url = ('https://data.cdc.gov/download/x7zy-2xmx/application%2Fz
    cdc_data_gdf = gpd.read_file(cdc_data_url)
    chicago_gdf = cdc_data_gdf[cdc_data_gdf.PlaceName=='Chicago']
    chicago_gdf.to_file(cdc_data_path, index=False)
```

```
# Load in cdc data
chicago_gdf = gpd.read_file(cdc_data_path)

# Check the data - site plot
chicago_gdf.plot()

(
    chicago_gdf
    .to_crs(ccrs.Mercator())
    .hvplot(
        line_color='red', fill_color=None,
        crs=ccrs.Mercator(), tiles='EsriImagery',
        frame_width=600
    )
)
```

Out[87]:


42
41.9
41.8
41.7
Latitude
−87.9
−87.8
−87.7
−87.6
Longitude
© Esri, Earthst



Reflect and Respond
What do you notice about the City of Chicago from the coarse satellite image? Is green space evenly distributed? What can you learn about Chicago from websites, scientific papers, or other resources that might help explain what you see in the site map?

CITY OF CHICAGO DATA DESCRIPTION AND CITATION HERE

Chicago

The larger Chicago area covers the following counties in Illinois: Cook (FIPS 17030), DeKalb, DuPage, Kane, Kankakee, Kendall, Lake, McHenry, Will) and three counties in northwest Indiana (Lake, LaPorte, Porter). For this analysis, we will be focusing on central Chicago - which is Cook County.

From the maps, it is clear that green space is not equally distributed across the region. 8.5% of the land area of Chicago is park space open to the public. Chicago added 1,300 acres of open space between 1998 and 2010, but still ranks below many major metropolitian cities such as Miami, New York, San Francisco, and others.

According to the City of Chicago, the city has "24 Industrial Corridors, comprising about 12 percent of city land, have boundaries that generally align with railroad embankments, waterways, highways, arterial streets and other manmade and natural buffers that effectively separate interior industrial uses from adjacent residential and commercial activity." Several of these industrial coordiors correlate with areas of lower green space prevalence.

Asthma is more prevalent in some areas of Chicago, particularly in the western and southern parts of the city. These areas include the industrial corridor, west of downtown Chicago, and near the village of Maywood.

Data Discription

Data is drawn from the Center for Disease Control PLACES: Local Data for Better Health data set. The Places Dataset estimates chronic disease and other health-related measures at various geographic levels of the United States using a small area estimation methodology. Data are derived from Behavioral Risk Factors Surveillance System data, Census population data, and American Community Survey data. There are a total of 40 measures genereated in ghe 2024 release. For more information see the CDC Places website.

For this analysis, I will be using data from the city level - specifically the city of Chicago.

For information on how to access this data via the CDC portal, you can follow this tutorial. https://www.cdc.gov/places/tools/explore-places-data-portal.html

Reference: https://www.cdc.gov/places/methodology/index.html

In [88]:
```
chicago_gdf
```

Out[88]:

	place2010	tract2010	ST	PlaceName	plctract10	PlcTrPop10	
0	1714000	17031010100	17	Chicago	1714000-17031010100	4854	((-
1	1714000	17031010201	17	Chicago	1714000-17031010201	6450	((-
2	1714000	17031010202	17	Chicago	1714000-17031010202	2818	((
3	1714000	17031010300	17	Chicago	1714000-17031010300	6236	((- -
4	1714000	17031010400	17	Chicago	1714000-17031010400	5042	((-
...	...	...	...	...	...	...	
804	1714000	17031835700	17	Chicago	1714000-17031835700	0	((
805	1714000	17031980000	17	Chicago	1714000-17031980000	0	((-
806	1714000	17031980100	17	Chicago	1714000-17031980100	0	((-
807	1714000	17043840000	17	Chicago	1714000-17043840000	0	(-
808	1714000	17043840801	17	Chicago	1714000-17043840801	0	((

809 rows × 7 columns

Download census tracts and select your urban area

You can obtain urls for the U.S. Census Tract shapefiles from the TIGER service. You'll

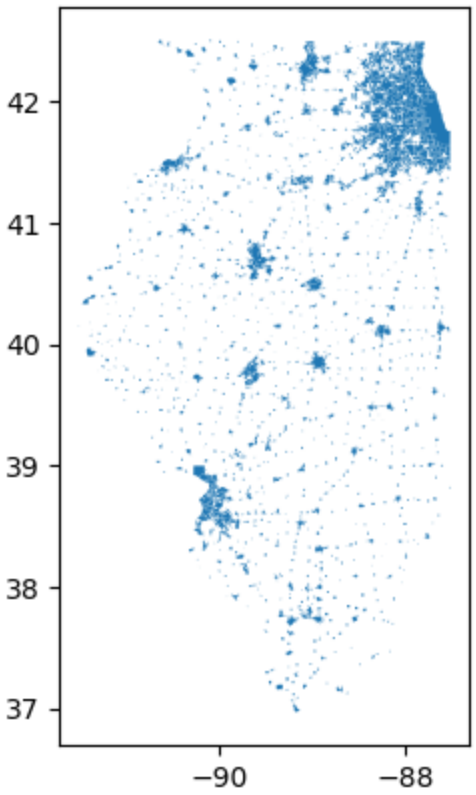
notice that these URLs use the state FIPS, which you can get by looking it up (e.g. here, or by installing and using the `us` package.

Try It

1. Download the Census tract Shapefile for the state of Illinois (IL).
2. Use a **conditional statement** to cache the download
3. Use a **spatial join** to select only the Census tracts that lie at least partially within the City of Chicago boundary.

In [89]:
```
# Define info for big data download (Illinois)
census_data_url=("https://www2.census.gov/geo/tiger/TIGER2024/PLACE/tl_2024_
)

census_data_dir = os.path.join(data_dir, 'census_data')
os.makedirs(census_data_dir, exist_ok=True)
census_data_path = os.path.join(census_data_dir, '*.shp') # Spatial Join

# Only download the census tracts once
if not os.path.exists(census_data_path):
    census_data_gdf = gpd.read_file(census_data_url)
    census_data_gdf.to_file(census_data_path)
    chi_census_gdf = census_data_gdf[census_data_gdf.NAME=='Chicago']
    chi_census_gdf.to_file(census_data_gdf, index=False)


# Load in census tract data
census_data_gdf = gpd.read_file(census_data_path)

# Check the data - site plot - Census tracts with satellite imagery in the b
census_data_gdf.plot()
```

Out[89]: <Axes: >



In [90]:
```
# Print the column names
print(census_data_gdf.columns)
```

```
Index(['STATEFP', 'PLACEFP', 'PLACENS', 'GEOID', 'GEOIDFQ', 'NAME', 'NAMELSA
D',
       'LSAD', 'CLASSFP', 'PCICBSA', 'MTFCC', 'FUNCSTAT', 'ALAND', 'AWATER',
       'INTPTLAT', 'INTPTLON', 'geometry'],
      dtype='object')
```

STEP 3 - Access Asthma and Urban Greenspaces Data

Access human health data

The U.S. Center for Disease Control (CDC) provides a number of health variables through their Places Dataset that might be correlated with urban greenspace. For this assignment, start with adult asthma. Try to limit the data as much as possible for download. Selecting the state and county is a one way to do this.

Try It

1. You can access Places data with an API, but as with many APIs it is easier to test out your search before building a URL. Navigate to the Places Census Tract Data Portal and search for the data you want.
2. The data portal will make an API call for you, but there is a simpler, easier to read and modify way to form an API call. Check out to the socrata documentation to see

how. You can also find some limited examples and a list of available parameters for this API on CDC Places SODA Consumer API Documentation.

3. Once you have formed your query, you may notice that you have exactly 1000 rows. The Places SODA API limits you to 1000 records in a download. Either narrow your search or check out the `&$limit=` parameter to increase the number of rows downloaded. You can find more information on the Paging page of the SODA API documentation
4. You should also clean up this data by renaming the `'data_value'` to something descriptive, and possibly selecting a subset of columns.

In [91]:
```
# Set up a path for the asthma data
cdc_path = os.path.join(data_dir, 'asthma.csv')

# Download asthma data (only once)
if not os.path.exists(cdc_path):
    asthma_url = (
        "https://data.cdc.gov/resource/cwsq-ngmh.csv"
        "?year=2022"
        "&stateabbr=IL"
        "&countyname=Cook"
        "&measureid=CASTHMA"
        "&$limit=1500"
    )
    asthma_df = (
        pd.read_csv(asthma_url)
        .rename(columns={
            'data_value': 'asthma',
            'low_confidence_limit': 'asthma_ci_low',
            'high_confidence_limit': 'asthma_ci_high',
            'locationname': 'tract'})
        [[
            'year', 'tract',
            'asthma', 'asthma_ci_low', 'asthma_ci_high', 'data_value_unit',
            'totalpopulation', 'totalpop18plus'
        ]]
    )
    asthma_df.to_csv(asthma_path, index=False)

# Load in asthma data
asthma_df = pd.read_csv(asthma_path)

# Preview asthma data
asthma_df
```

Out[91]:

	year	tract	asthma	asthma_ci_low	asthma_ci_high	data_value_unit
0	2022	17031031100	8.4	7.5	9.5	%
1	2022	17031031900	8.6	7.7	9.7	%
2	2022	17031062600	8.3	7.3	9.3	%
3	2022	17031070101	8.9	7.9	9.9	%
4	2022	17031081100	9.0	8.0	10.1	%
...	...	...	...	...	...	...
1323	2022	17031834900	13.5	12.1	15.0	%
1324	2022	17031828601	9.8	8.8	11.0	%
1325	2022	17031843700	8.4	7.5	9.5	%
1326	2022	17031829700	10.8	9.6	12.1	%
1327	2022	17031829100	10.2	9.1	11.5	%

1328 rows × 8 columns

Join health data with census tract boundaries

Try It

1. Join the census tract `GeoDataFrame` with the asthma prevalence `DataFrame` using the `.merge()` method.
2. You will need to change the data type of one of the merge columns to match, e.g. using `.astype('int64')`
3. There are a few census tracts in Chicago that do not have data. You should be able to confirm that they are not listed through the interactive Places Data Portal. However, if you have large chunks of the city missing, it may mean that you need to expand the record limit for your download.

In [92]:
```
# Determine Data types
print(census_data_gdf.dtypes)
print(asthma_df.dtypes)
```

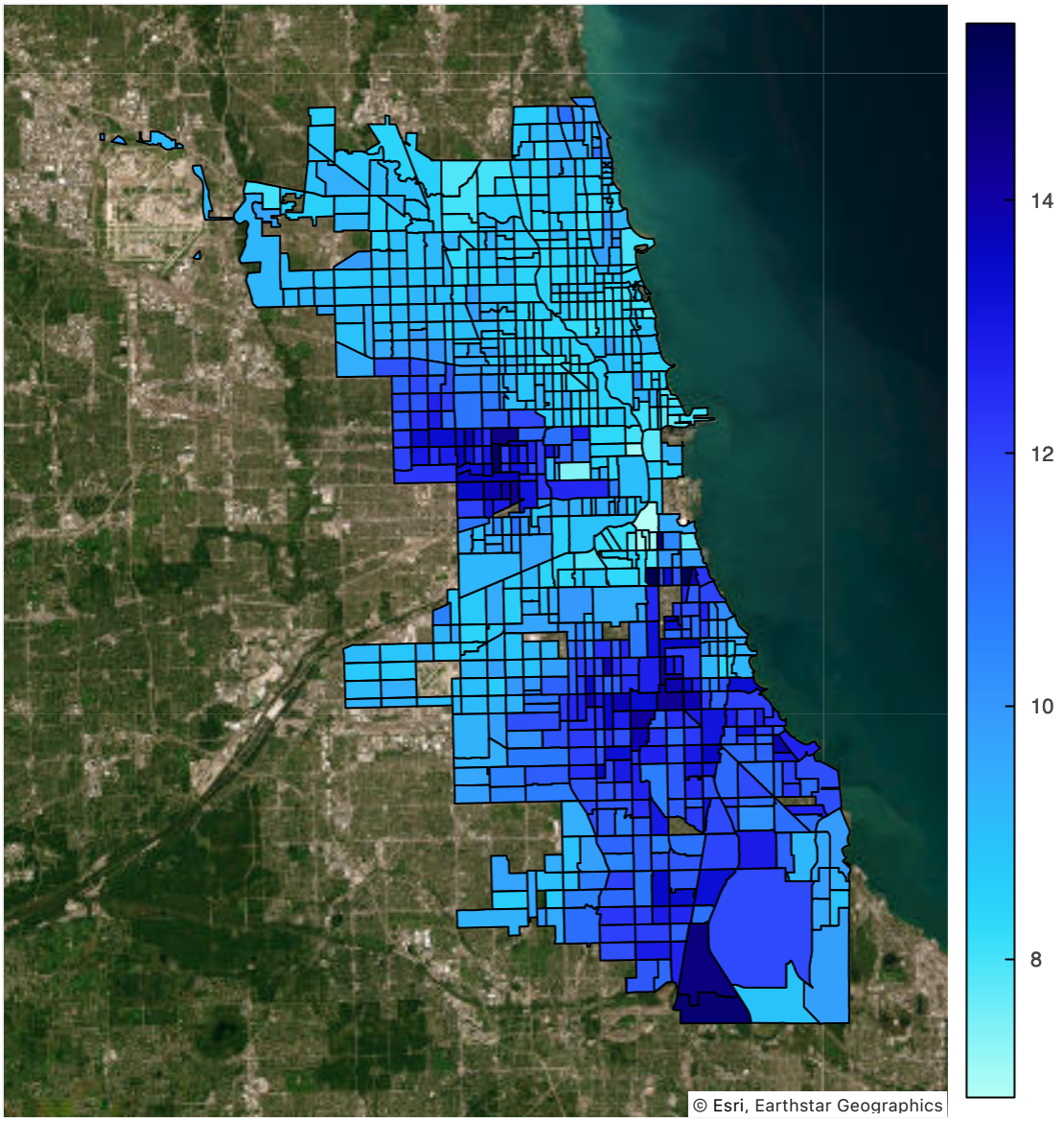
```
STATEFP       object
PLACEFP       object
PLACENS       object
GEOID         object
GEOIDFQ       object
NAME          object
NAMELSAD      object
LSAD          object
CLASSFP       object
PCICBSA       object
MTFCC         object
FUNCSTAT      object
ALAND          int64
AWATER         int64
INTPTLAT      object
INTPTLON      object
geometry    geometry
dtype: object
year                 int64
tract                int64
asthma             float64
asthma_ci_low      float64
asthma_ci_high     float64
data_value_unit     object
totalpopulation      int64
totalpop18plus       int64
dtype: object
```

In [93]:
```
asthma_df.columns
```

Out[93]:
```
Index(['year', 'tract', 'asthma', 'asthma_ci_low', 'asthma_ci_high',
       'data_value_unit', 'totalpopulation', 'totalpop18plus'],
      dtype='object')
```

In [94]:
```
# Change tract identifier datatype for merging
chicago_gdf.tract2010 = chicago_gdf.tract2010.astype('int64')

# Merge census data with geometry
merged_gdf = (
    chicago_gdf
    .merge(asthma_df, left_on='tract2010', right_on='tract', how='inner')
)

# Plot asthma data as chloropleth
(
    gv.tile_sources.EsriImagery
    *
    gv.Polygons(
        merged_gdf.to_crs(ccrs.Mercator()),
        vdims=['asthma', 'tract2010'],
        crs=ccrs.Mercator()
    ).opts(color='asthma', colorbar=True, tools=['hover'])
).opts(width=600, height=600, xaxis=None, yaxis=None)
```

Out[94]:



Reflect and Respond

Write a description and citation for the asthma prevalence data. Do you notice anything about the spatial distribution of asthma in Chicago? From your research on the city, what might be some potential causes of any patterns you see?

CDC ASTHMA PLACES DESCRIPTION AND CITATION

Reflect and Respond

It is clear that there is a geographical component to the distribution of adults who have asthma. Whether this is a by product of another variable or not cannot be determined at this time. Other potential causes could be income disparity, areas with high pollution due

to heavy industry or other reasons, or areas with insufficent access to affordable healthcare. As noted above - there are 24 industrial corridors in Chicago - depending on what type of industry - these corridors could be the focal centers.

While there is some overlap of between the areas the asthma prevalence and industrial coordiors where is no clear correleation. You can find a mape of the industrial areas here. There also does not appear to be any correlation between truck routes or interstates and asthma. All areas with high rates of asthma do have interstates running through or near them, but not all areas near interstated have high rates of asthma.

There is one map that does closely match the distribution of asthma prevalence, and that is a map based on race. A demographic map of Chicago, when overlaid on a map of asthma prevalence, shows a strong correlation between asthma and black communities, with a lesser correlation between asthma and hispanic communities.

Get Data URLs

NAIP data are freely available through the Microsoft Planetary Computer SpatioTemporal Access Catalog (STAC).

Try It
Get started with STAC by accessing the planetary computer catalog with the following code:

```
e84_catalog = pystac_client.Client.open(
    "https://planetarycomputer.microsoft.com/api/stac/v1"
)
```

In [95]:
```
# Connect to the planetary computer catalog
e84_catalog = pystac_client.Client.open(
    "https://planetarycomputer.microsoft.com/api/stac/v1"
)
e84_catalog.title
```

Out[95]:
```
'Microsoft Planetary Computer STAC API'
```

Try It

1. Using a loop, for each Census Tract:
    A. Use the following sample code to search for data, replacing the names with applicable values with descriptive names:

    ```
    search = e84_catalog.search(
        collections=["naip"],
        intersects=shapely.to_geojson(tract_geometry),
        datetime=f"{year}"
    )
    ```

    B. Access the url using `search.assets['image'].href`

2. Accumulate the urls in a `pd.DataFrame` or `dict` for later

3. Occasionally you may find that the STAC service is momentarily unavailable. You should include code that will retry the request up to 5 times when you get the `pystac_client.exceptions.APIError`.

> **Warning**
>
> As always – DO NOT try to write this loop all at once! Stick with one step at a time, making sure to test your work. You also probably want to add a `break` into your loop to stop the loop after a single iteration. This will help prevent long waits during debugging.

In [96]:
```
# Use a loop and for each Census track search for the approrpiate data.
# Convert geometry to lat/lon for STAC
merged_latlon_gdf = merged_gdf.to_crs(4326)

# Define a path to save NDVI stats
ndvi_stats_path = os.path.join(data_dir, 'chicago-ndvi-stats.csv')

# Check for existing data - do not access duplicate tracts
downloaded_tracts = []
if os.path.exists(ndvi_stats_path):
    ndvi_stats_df = pd.read_csv(ndvi_stats_path)
    downloaded_tracts = ndvi_stats_df.tract.values
else:
    print('No census tracts downloaded so far')

# Loop through each census tract
scene_dfs = []
for i, tract_values in tqdm(merged_latlon_gdf.iterrows()):
    tract = tract_values.tract2010
    # Check if statistics are already downloaded for this tract
    if not (tract in downloaded_tracts):
        # Retry up to 5 times in case of a momentary disruption
        i = 0
        retry_limit = 5
        while i < retry_limit:
            # Try accessing the STAC
            try:
                # Search for tiles
                naip_search = e84_catalog.search(
                    collections=["naip"],
                    intersects=shapely.to_geojson(tract_values.geometry),
                    datetime="2021"
                )

                # Build dataframe with tracts and tile urls
                scene_dfs.append(pd.DataFrame(dict(
                    tract=tract,
                    date=[pd.to_datetime(scene.datetime).date()
                          for scene in naip_search.items()],
                    rgbir_href=[scene.assets['image'].href for scene in naip
```

```
                ))

                break
            # Try again in case of an APIError
            except pystac_client.exceptions.APIError:
                print(
                    f'Could not connect with STAC server. '
                    f'Retrying tract {tract}...')
                time.sleep(2)
                i += 1
                continue

# Concatenate the url dataframes
if scene_dfs:
    scene_df = pd.concat(scene_dfs).reset_index(drop=True)
else:
    scene_df = None

# Preview the URL DataFrame
scene_df
```

```
No census tracts downloaded so far
0it [00:00, ?it/s]
```

Out[96]:

	tract	date	rgbir_href
0	17031010100	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
1	17031010201	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
2	17031010201	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
3	17031010202	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
4	17031010300	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
...	...	...	...
1252	17031807900	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
1253	17031807900	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
1254	17031807900	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
1255	17031808100	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...
1256	17031808100	2021-09-08	https://naipeuwest.blob.core.windows.net/naip/...

1257 rows × 3 columns

In [143…

```
# Define the directory
hls_dir = os.path.join(data_dir, 'hls')
os.makedirs(hls_dir, exist_ok=True)

hls_url = (
    "https://github.com/cu-esiil-edu/esiil-learning-portal/releases"
    "/download/data-release/redlining-foundations-data.zip"
)
```

```
if not glob.glob(os.path.join(hls_dir, '*.tif')):
    # Download sample raster data
    hls_response = requests.get(hls_url)

    # Save the raster data (uncompressed)
    with zipfile.ZipFile(BytesIO(hls_response.content)) as hls_zip:
        hls_zip.extractall(hls_dir)
```

Compute NDVI Statistics

Next, calculate some metrics to get at different aspects of the distribution of greenspace in each census tract. These types of statistics are called **fragmentation statistics**, and can be implemented with the `scipy` package. Some examples of fragmentation statistics are:

Percentage vegetation
The percentage of pixels that exceed a vegetation threshold (.12 is common with Landsat) Mean patch size

The average size of **patches**, or contiguous area exceeding the vegetation threshold. Patches can be identified with the `label` function from `scipy.ndimage` Edge density

The proportion of edge pixels among vegetated pixels. Edges can be identified by **convolving** the image with a **kernel** designed to detect pixels that are different from their surroundings.

> **What is convolution?**
>
> If you are familiar with differential equations, convolution is an approximation of the LaPlace transform.
>
> For the purposes of calculating edge density, convolution means that we are taking all the possible 3x3 chunks for our image, and multiplying it by the kernel:
>
> $$\text{Kernel} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
>
> The result is a matrix the same size as the original, minus the outermost edge. If the center pixel is the same as the surroundings, its value in the final matrix will be $-8+1+1+1+1+1+1+1+1=0$. If it is higher than the surroundings, the result will be negative, and if it is lower than the surroundings, the result will be positive. As such, the edge pixels of our patches will be negative.

Try It

1. Select a single row from the census tract `GeoDataFrame` using e.g. `.loc[[0]]`, then select all the rows from your URL `DataFrame` that match the census tract.
2. For each URL in crop, merge, clip, and compute NDVI for that census tract
3. Set a threshold to get a binary mask of vegetation
4. Using the sample code to compute the fragmentation statistics. Feel free to add any other statistics you think are relevant, but make sure to include a fraction vegetation, mean patch size, and edge density. If you are not sure what any line of code is doing, make a plot or print something to find out! You can also ask ChatGPT or the class.

In [144…

```python
# Skip this step if data are already downloaded
if not scene_df is None:
    # Get an example tract
    tract = chicago_gdf.loc[0].tract2010
    ex_scene_gdf = scene_df[scene_df.tract==tract]

    # Loop through all images for tract
    tile_das = []
    for _, href_s in ex_scene_gdf.iterrows():
        # Open vsi connection to data
        tile_da = rxr.open_rasterio(
            href_s.rgbir_href, masked=True).squeeze()

        # Crop data to the bounding box of the census tract
        boundary = (
            merged_gdf
            .set_index('tract2010')
            .loc[[tract]]
            .to_crs(tile_da.rio.crs)
            .geometry
        )
        crop_da = tile_da.rio.clip_box(
            *boundary.envelope.total_bounds,
            auto_expand=True)

        # Clip data to the boundary of the census tract
        clip_da = crop_da.rio.clip(boundary, all_touched=True)

        # Compute NDVI
        ndvi_da = (
            (clip_da.sel(band=4) - clip_da.sel(band=1))
            / (clip_da.sel(band=4) + clip_da.sel(band=1))
        )

        # Accumulate result
        tile_das.append(ndvi_da)

    # Merge data
    scene_da = rxrmerge.merge_arrays(tile_das)

    # Mask vegetation
    veg_mask = (scene_da>.3)
```

```
    # Calculate mean patch size
    labeled_patches, num_patches = label(veg_mask)

    # Count patch pixels, ignoring background at patch 0
    patch_sizes = np.bincount(labeled_patches.ravel())[1:]
    mean_patch_size = patch_sizes.mean()

    # Calculate edge density
    kernel = np.array([
        [1, 1, 1],
        [1, -8, 1],
        [1, 1, 1]])
    edges = convolve(veg_mask, kernel, mode='constant')
    edge_density = np.sum(edges != 0) / veg_mask.size
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[144], line 39
     36     tile_das.append(ndvi_da)
     38 # Merge data
---> 39 scene_da = rxrmerge.merge_arrays(tile_das)
     41 # Mask vegetation
     42 veg_mask = (scene_da>.3)

File ~/miniconda3/envs/earth-analytics-python/lib/python3.11/site-packages/r
ioxarray/merge.py:116, in merge_arrays(dataarrays, bounds, res, nodata, prec
ision, method, crs, parse_coordinates)
    107 input_kwargs = {
    108     "bounds": bounds,
    109     "res": res,
   (...)
    112     "method": method,
    113 }
    115 if crs is None:
--> 116     crs = dataarrays[0].rio.crs
    117 if res is None:
    118     res = tuple(abs(res_val) for res_val in dataarrays[0].rio.resolu
tion())

IndexError: list index out of range
```

Repeat for all tracts

Try It

1. Using a loop, for each Census Tract:

    A. Using a loop, for each data URL:

        a. Use `rioxarray` to open up a connection to the STAC asset, just like you would a file on your computer
        b. Crop and then clip your data to the census tract boundary > HINT: check out the `.clip_box` parameter `auto_expand` and the `clip` parameter

`all_touched` to make sure you don't end up with an empty array

c. Compute NDVI for the tract

B. Merge the NDVI rasters

C. Compute:

a. total number of pixels within the tract

b. fraction of pixels with an NDVI greater than .12 within the tract (and any other statistics you would like to look at)

D. Accumulate the statistics in a file for later

2. Using a conditional statement, ensure that you do not run this computation if you have already saved values. You do not want to run this step many times, or have to restart from scratch! There are many approaches to this, but we actually recommend implementing your caching in the previous cell when you generate your dataframe of URLs, since that step can take a few minutes as well. However, the important thing to cache is the computation.

In [107…

```
# Skip this step if data are already downloaded
if not scene_df is None:
    ndvi_dfs = []
    # Loop through the census tracts with URLs
    for tract, tract_date_gdf in tqdm(scene_df.groupby('tract')):
        # Open all images for tract
        tile_das = []
        for _, href_s in tract_date_gdf.iterrows():
            # Open vsi connection to data
            tile_da = rxr.open_rasterio(
                href_s.rgbir_href, masked=True).squeeze()

            # Clip data
            boundary = (
                merged_gdf
                .set_index('tract2010')
                .loc[[tract]]
                .to_crs(tile_da.rio.crs)
                .geometry
            )
            crop_da = tile_da.rio.clip_box(
                *boundary.envelope.total_bounds,
                auto_expand=True)
            clip_da = crop_da.rio.clip(boundary, all_touched=True)

            # Compute NDVI
            ndvi_da = (
                (clip_da.sel(band=4) - clip_da.sel(band=1))
                / (clip_da.sel(band=4) + clip_da.sel(band=1))
            )

            # Accumulate result
            tile_das.append(ndvi_da)
```

```
        # Merge data
        scene_da = rxrmerge.merge_arrays(tile_das)

        # Mask vegetation
        veg_mask = (scene_da>.3)

        # Calculate statistics and save data to file
        total_pixels = scene_da.notnull().sum()
        veg_pixels = veg_mask.sum()

        # Calculate mean patch size
        labeled_patches, num_patches = label(veg_mask)

        # Count patch pixels, ignoring background at patch 0
        patch_sizes = np.bincount(labeled_patches.ravel())[1:]
        mean_patch_size = patch_sizes.mean()

        # Calculate edge density
        kernel = np.array([
            [1, 1, 1],
            [1, -8, 1],
            [1, 1, 1]])
        edges = convolve(veg_mask, kernel, mode='constant')
        edge_density = np.sum(edges != 0) / veg_mask.size

        # Add a row to the statistics file for this tract
        pd.DataFrame(dict(
            tract=[tract],
            total_pixels=[int(total_pixels)],
            frac_veg=[float(veg_pixels/total_pixels)],
            mean_patch_size=[mean_patch_size],
            edge_density=[edge_density]
        )).to_csv(
            ndvi_stats_path,
            mode='a',
            index=False,
            header=(not os.path.exists(ndvi_stats_path))
        )

# Re-load results from file
ndvi_stats_df = pd.read_csv(ndvi_stats_path)
ndvi_stats_df
```

```
  0%|          | 0/788 [00:00<?, ?it/s]
/var/folders/j7/815c6qns6g39vvr0gw7cftfm0000gn/T/ipykernel_64400/3478176366.py:50: RuntimeWarning: Mean of empty slice.
  mean_patch_size = patch_sizes.mean()
/Users/erinzimmerman/miniconda3/envs/earth-analytics-python/lib/python3.11/site-packages/numpy/_core/_methods.py:138: RuntimeWarning: invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
```

Out[107…

	tract	total_pixels	frac_veg	mean_patch_size	edge_density
0	17031010100	1348088	0.140350	55.225919	0.118612
1	17031010201	1632112	0.200683	57.543394	0.161668
2	17031010202	1146600	0.158785	63.260250	0.123673
3	17031010300	1711710	0.146512	57.113642	0.126384
4	17031010400	3119840	0.096548	52.983817	0.079474
...	...	...	...	...	...
783	17031843500	6963674	0.061033	9.732779	0.104686
784	17031843600	1177024	0.052817	9.177296	0.101217
785	17031843700	7025535	0.023710	7.477533	0.047642
786	17031843800	3786240	0.090268	24.169295	0.105052
787	17031843900	8095350	0.111222	23.985215	0.124282

788 rows × 5 columns

STEP 3 - Explore your data with plots

Chloropleth plots

Before running any statistical models on your data, you should check that your download worked. You should see differences in both median income and mean NDVI across the City.

Try It
Create a plot that contains:

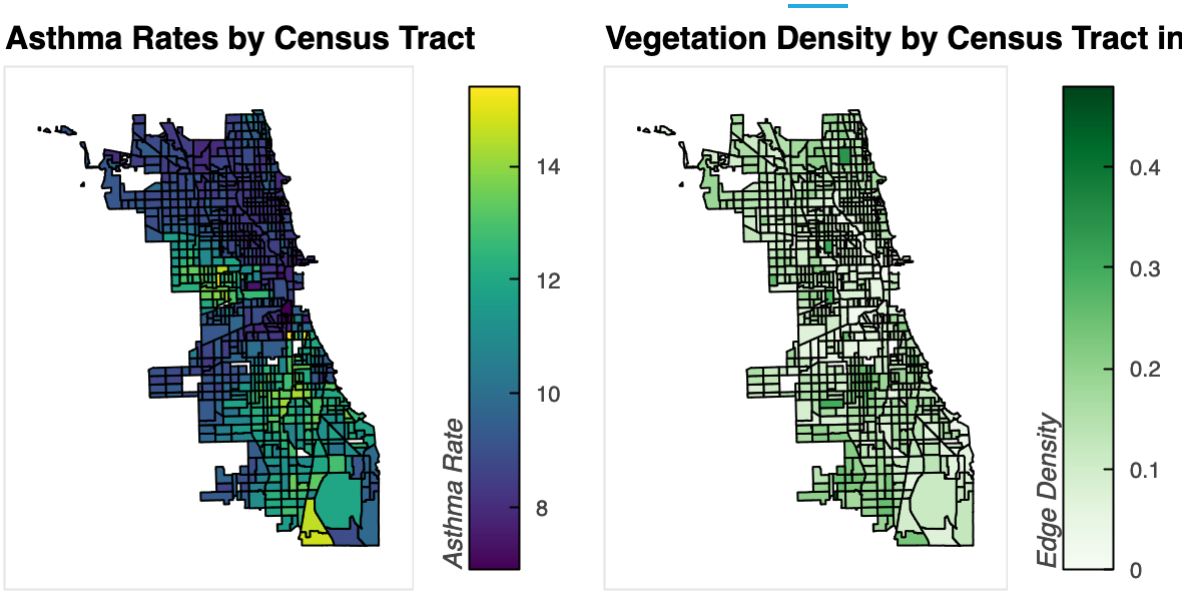
- 2 side-by-side Chloropleth plots
- Asthma prevelence on one and mean NDVI on the other
- Make sure to include a title and labeled color bars

In [124…

```
# Merge census data with geometry
merged_ndvi_gdf = (
    merged_gdf
    .merge(
        ndvi_stats_df,
        left_on='tract2010', right_on='tract', how='inner')
)

# Plot choropleths with vegetation statistics
def plot_chloropleth(gdf, colorbar_opts=None, **opts):
    """Generate a choropleth with the given color column and colorbar label"
    return gv.Polygons(
```

```
        gdf.to_crs(ccrs.Mercator()),
        crs=ccrs.Mercator()
    ).opts(
        xaxis=None,
        yaxis=None,
        colorbar=True,
        colorbar_opts=colorbar_opts,
        xlabel="Longitude",
        ylabel="Latitude",
        **opts
    )

# Generate plots with correct colorbar labels
(
    plot_chloropleth(
        merged_ndvi_gdf,
        color='asthma',
        cmap='viridis',
        title="Asthma Rates by Census Tract",
        colorbar_opts={'title': "Asthma Rate"}  # Fixed here
    )
    +
    plot_chloropleth(
        merged_ndvi_gdf,
        color='edge_density',
        cmap='Greens',
        title="Vegetation Density by Census Tract in Chicago",
        colorbar_opts={'title': "Edge Density"}  # Fixed here
    )
)
```

Out[124…



Do you see any similarities in your plots? Do you think there is a relationship between adult asthma and any of your vegetation statistics in Chicago? Relate your visualization to the research you have done (the context of your analysis) if applicable.

Plot discription and Comparison

While there are some slight similarities between the two plots I don't see any clear correlation between the asthma rates by census tract and the vegitation density by census tract. Some areas actually show a inverse relationship cuhc as areas through the center of the city along Interstate 55.

STEP 4: Explore a linear ordinary least-squares regression

Model description

One way to find if there is a statistically significant relationship between asthma prevalence and greenspace metrics is to run a linear ordinary least squares (OLS) regression and measure how well it is able to predict asthma given your chosen fragmentation statistics.

Before fitting an OLS regression, you should check that your data are appropriate for the model.

Try It
Write a model description for the linear ordinary least-squares regression that touches on:

1. Assumptions made about the data
   - OLS modeling is one of the simplist and most straight forward types of models; however, its usability resets upon certain assumptions about the data. Foremost, the model assumes a linear regression between x and y. Other assumptions are: * Errors: errors are normally distrbiuted and unrelated to each other * Independence: when working with big data you will likely have multiple variables, it is important to consider that these variables do not interact with each other (they are not co-linear) * Stationarity: model parameters do not vary over time. * Completeness: there are no 'no value' observations.
2. What is the objective of this model? What metrics could you use to evaluate the fit?
   - The objective of this model is to explain as much variation as possible, provide information on the relative importance of different X variables, and avoid overfitting.
3. Advantages and potential problems with choosing this model.

- Advantages are that the model is simple and easy to use
- disadvantage are that all of the assumptions need to be met or your results may be 'unstable.'

ADD YOUR CDC PLACES DESCRIPTION AND CITATION HERE

Data preparation

When fitting statistical models, you should make sure that your data meet the model assumptions through a process of selection and/or transformation.

You can select data by:

- Eliminating observations (rows) or variables (columns) that are missing data
- Selecting a model that matches the way in which variables are related to each other (for example, linear models are not good at modeling circles)
- Selecting variables that explain the largest amount of variability in the dependent variable.

You can transform data bt:

- Transforming a variable so that it follows a normal distribution. The `log` transform is the most common to eliminate excessive skew (e.g. make the data symmetrical), but you should select a transform most suited to your data.
- Normalizing or standardizing variables to, for example, eliminate negative numbers or effects caused by variables being in a different range.
- Performing a principle component analysis (PCA) to eliminate multicollinearity among the predictor variables

> **Tip**
>
> Keep in mind that data transforms like a log transform or a PCA must be reversed after modeling for the results to be meaningful.

Try It

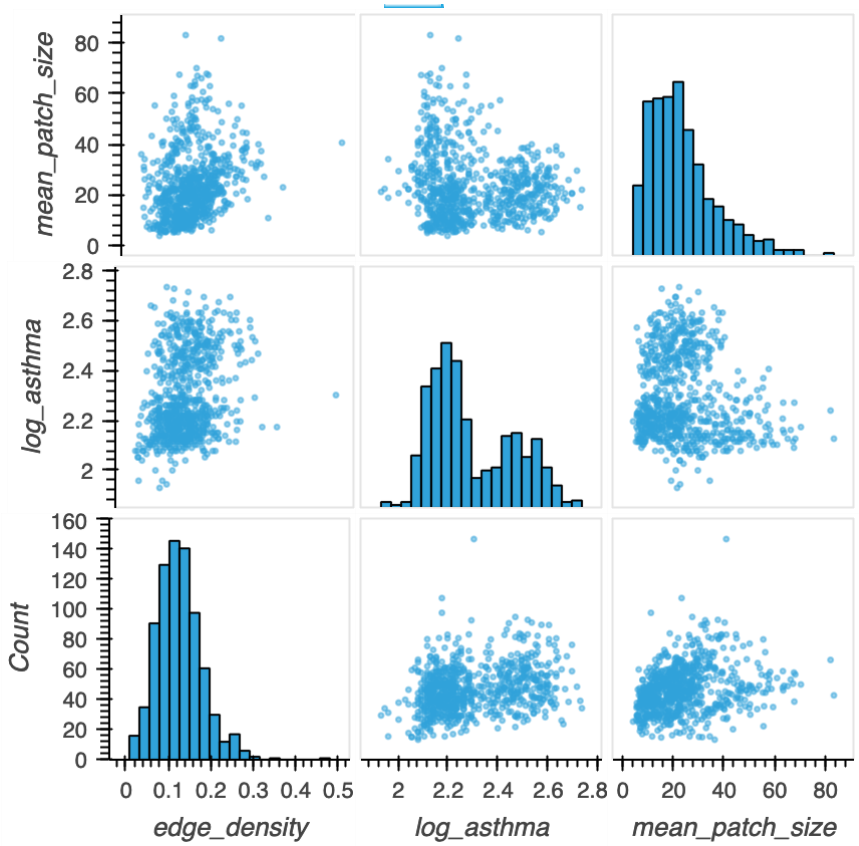
1. Use the `hvplot.scatter_matrix()` function to create an exploratory plot of your data.
2. Make any necessary adjustments to your data to make sure that they meet the assumptions of a linear OLS regression.
3. Check if there are `NaN` values, and if so drop those rows and/or columns. You can use the `.dropna()` method to drop rows with `NaN` values.
4. Explain any data transformations or selections you made and why

In [125…

```
# Variable selection and transformation
model_df = (
```

```
    merged_ndvi_gdf
    .copy()
    [['frac_veg', 'asthma', 'mean_patch_size', 'edge_density', 'geometry']]
    .dropna()
)

model_df['log_asthma'] = np.log(model_df.asthma)

# Plot scatter matrix to identify variables that need transformation
hvplot.scatter_matrix(
    model_df
    [[
        'mean_patch_size',
        'edge_density',
        'log_asthma'
    ]]
    )
```

Out[125…



EXPLAIN YOUR SELECTION AND TRANSFORMATION PROCESS HERE

Fit and Predict

If you have worked with statistical models before, you may notice that the `scikitlearn` library has a slightly different approach than many software packages. For example, `scikitlearn` emphasizes generic model performance measures like cross-validation and importance over coefficient p-values and correlation. The scikitlearn approach is meant to generalize more smoothly to machine learning (ML)

models where the statistical significance is harder to derive mathematically.

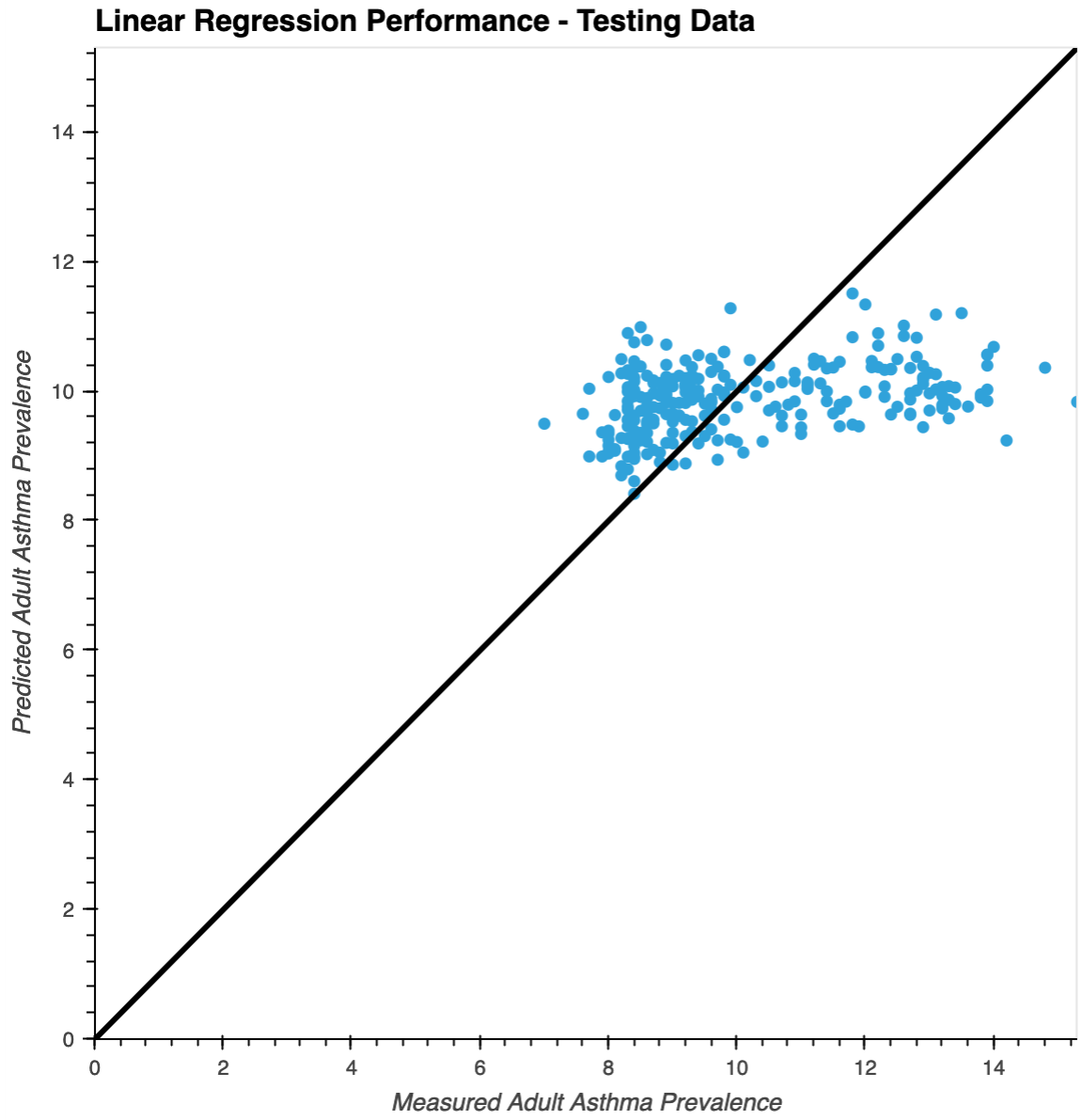
Try It

1. Use the scikitlearn documentation or ChatGPT as a starting point, split your data into training and testing datasets.
2. Fit a linear regression to your training data.
3. Use your fitted model to predict the testing values.
4. Plot the predicted values against the measured values. You can use the following plotting code as a starting point.

In [129…
```
import holoviews as hv
```

In [130…
```
# Select predictor and outcome variables

X = model_df[['edge_density', 'mean_patch_size']]
y = model_df[['log_asthma']]

# Split into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)

# Fit a linear regression
reg = LinearRegression()
reg.fit(X_train, y_train)

# Predict asthma values for the test dataset
y_test['pred_asthma'] = np.exp(reg.predict(X_test))
y_test['asthma'] = np.exp(y_test.log_asthma)

# Plot measured vs. predicted asthma prevalence with a 1-to-1 line
y_max = y_test.asthma.max()
(
    y_test
    .hvplot.scatter(
        x='asthma', y='pred_asthma',
        xlabel='Measured Adult Asthma Prevalence',
        ylabel='Predicted Adult Asthma Prevalence',
        title='Linear Regression Performance - Testing Data'
    )
    .opts(aspect='equal', xlim=(0, y_max), ylim=(0, y_max), height=600, widt
) * hv.Slope(slope=1, y_intercept=0).opts(color='black')
```

Out[130…



Spatial bias

We always need to think about bias, or systematic error, in model results. Every model is going to have some error, but we'd like to see that error evenly distributed. When the error is systematic, it can be an indication that we are missing something important in the model.

In geographic data, it is common for location to be a factor that doesn't get incorporated into models. After all – we generally expect places that are right next to each other to be more similar than places that are far away (this phenomenon is known as *spatial autocorrelation*). However, models like this linear regression don't take location into account at all.

Try It

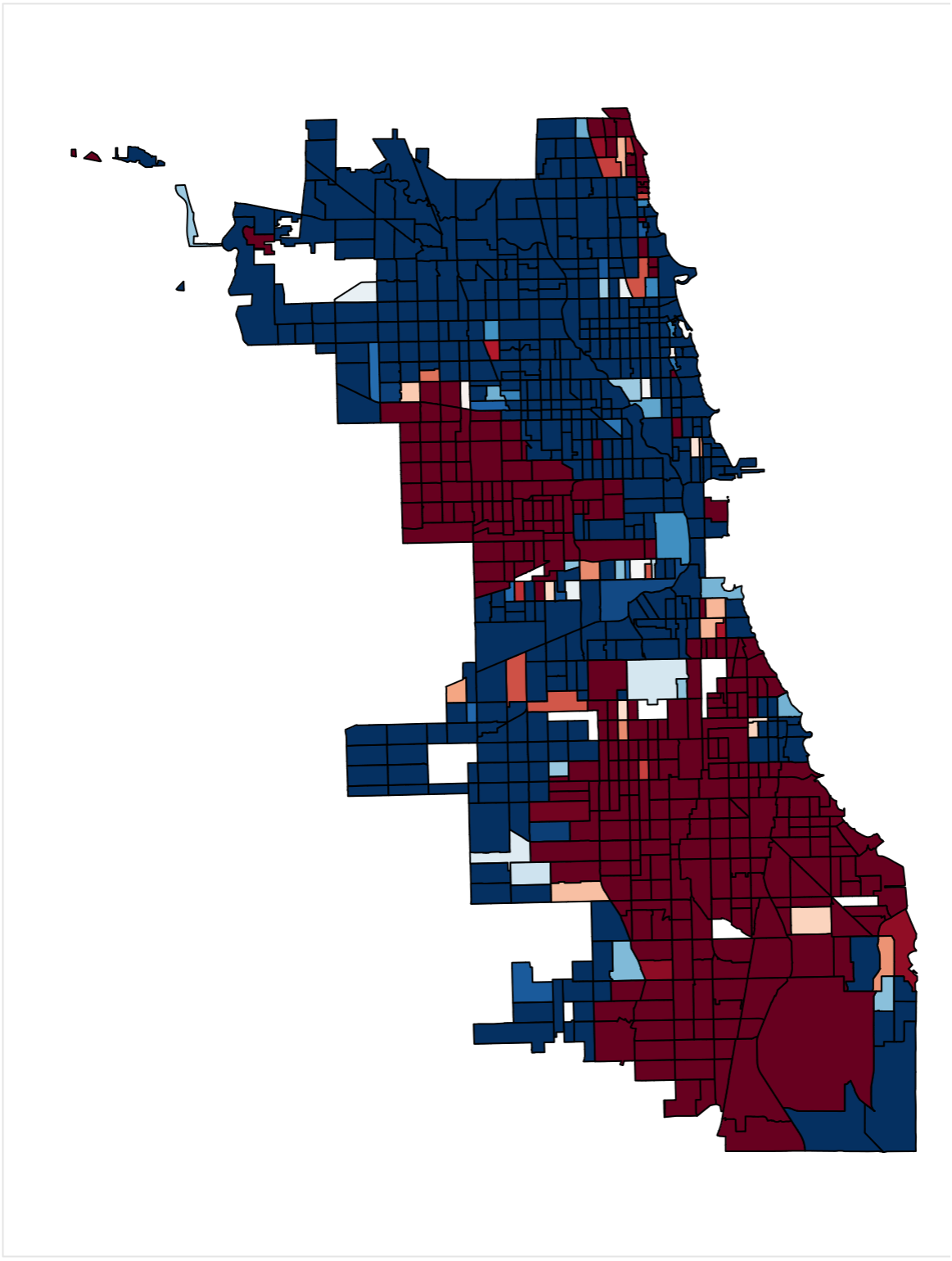
1. Compute the model error (predicted - measured) for all census tracts

2. Plot the error as a chloropleth map with a diverging color scheme
3. Looking at both of your error plots, what do you notice? What are some possible explanations for any bias you see in your model?

In [135…

```
# Compute model error for all census tracts
model_df['pred_asthma'] = np.exp(reg.predict(X))
model_df['err_asthma'] = model_df['pred_asthma'] - model_df['asthma']

(
    plot_chloropleth(model_df, color='err_asthma', cmap='RdBu')
    .redim.range(err_asthma=(-0.3, 0.3))
    .opts(frame_width=600, aspect='equal', colorbar_opts={})
)
```

Out[135...



Reflect and Respond
What do you notice about your model from looking at the error plots? What additional data, transformations, or model type might help improve your results?

DESCRIBE AND INTERPRET YOUR IMAGE HERE There is a striking correlation - though maybe not causation - between the error map and the asthma prevelalence seen on the

previous plots. The relationships between geography and such things as income, demographics or industry may also come into play.