🌳

Exploring City Green Space with Google Earth Engine

Little blurb: A detailed explanation of data analysis, visualization, and using satellite data with Python.

This article helps readers understand how to use Google Earth Engine's Python API to explore satellite imagery and how green space changes in cities.

Note: This is written in the format of a Medium article (including the little blurb, italicized text above, and alternating code and text below), designed for people to explore how to run this work. You can view this Notion page live here which should allow you to see the evolution of the time lapses.

Please find the GitHub repository here.

By 2050, the World Bank projects that 70% of the global population will live in cities. As more of the world's population moves to cities, green space will be crucial for social, mental, economic, and climate impacts in cities. Previous research has shown that urban trees have direct health benefits through the reduction of air pollution and improving mental health, they can reduce violence and aggression, improve academic performance, and have direct economic benefits through energy conservation and food security. With Google Earth Engine, it's possible to obtain satellite imagery from 1984, thereby allowing us to track how a city's green space has evolved over almost 40 years. In this article, you'll learn how to work with Google Earth Engine, how to measure vegetation, and visualize and track the green space changes of cities. Relevant code is embedded inline, or available packaged in a reproducible Jupyter Notebook available on GitHub.

Accessing Satellite Data via Google Earth Engine

No data science project is complete without data, and Google Earth Engine provides access to numerous datasets. With Climate and Weather, Satellite Imagery, Geophysical and more categories of datasets, the applications are endless. Signing up for a Google Earth Engine account essentially only requires a Google account and is very fast. Once your account is approved, you can install the Earth Engine API using `pip`, and then import Earth Engine with `ee`:

```
pip install earthengine-api
import ee
```

Your access to the API also needs to be authenticated, which entails running the following `Authenticate()` command, which will trigger a new window to open. Make sure you log in with the same Google account that was granted access to Earth Engine, and copy your authorization token into the box that will appear after running the code. After authentication, you initialize the library.

```
ee.Authenticate()
ee.Initialize()
```

Importing Packages

Other relevant packages to install include `Folium`, to create interactive maps, `pandas`, `numpy`, and `datetime` for data processing, and `matplotlib` and `IPython` for visualization.

The full list of packages to install is below. Some of these might not be pre-installed, but you can install them by using `pip install [replace with library]`.

```
# Earth Engine
import ee
import geemap

# For data manipulation
from datetime import datetime as dt
import pandas as pd
import numpy as np

# For visualization
import folium
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.colors import ListedColormap
from IPython.display import Image

# For use with APIs
import requests
import json
import urllib
from glob import glob

##TBD
from tqdm import tqdm
```

Understanding Green Space

One of the most common methods for identifying the amount of vegetation and green space in a given area is with a metric called NDVI, or the Normalized Difference Vegetation Index. This metric is calculated mathematically from the visible and near-infrared light reflected by vegetation as given by the following equation:

$$NDVI = (NIR - RED)/(NIR + RED)$$

Where NDVI, as stated previously, is the Normalized Difference Vegetation Index. NIR is the near-infrared light and RED is the visible red light. It measures the ratio of (the reflective difference in the red and the near-infrared light) to (the sum of red and near-infrared reflectance). This metric is useful because green vegetation reflects near-infrared light, while absorbs red light (Hafen, 2021). Thus, if there were lots of green vegetation in an area, we would expect the perfect outcome of NDVI to be:

$$NDVI = (NIR - RED)/(NIR + RED) = (1 - 0)/(1 + 0) = 1$$

The results of this calculation range from -1 to +1. The scale is interpreted as follows:

- Anything below 0 signifies no vegetation
- Anything close to 0 signifies there are no green leaves
- Anything close to 1 (0.8-0.9) signifies the highest possible density of green leaves.

Here, we use NDVI as our method to calculate the change of green space in a city largely because of the industry standard in using this, “nearly all satellite Vegetation Indices employ this difference formula to quantify the density of plant growth on the Earth” according to NASA (Levy and Przyborski, 2000).

Dataset: Landsat Satellite Imagery

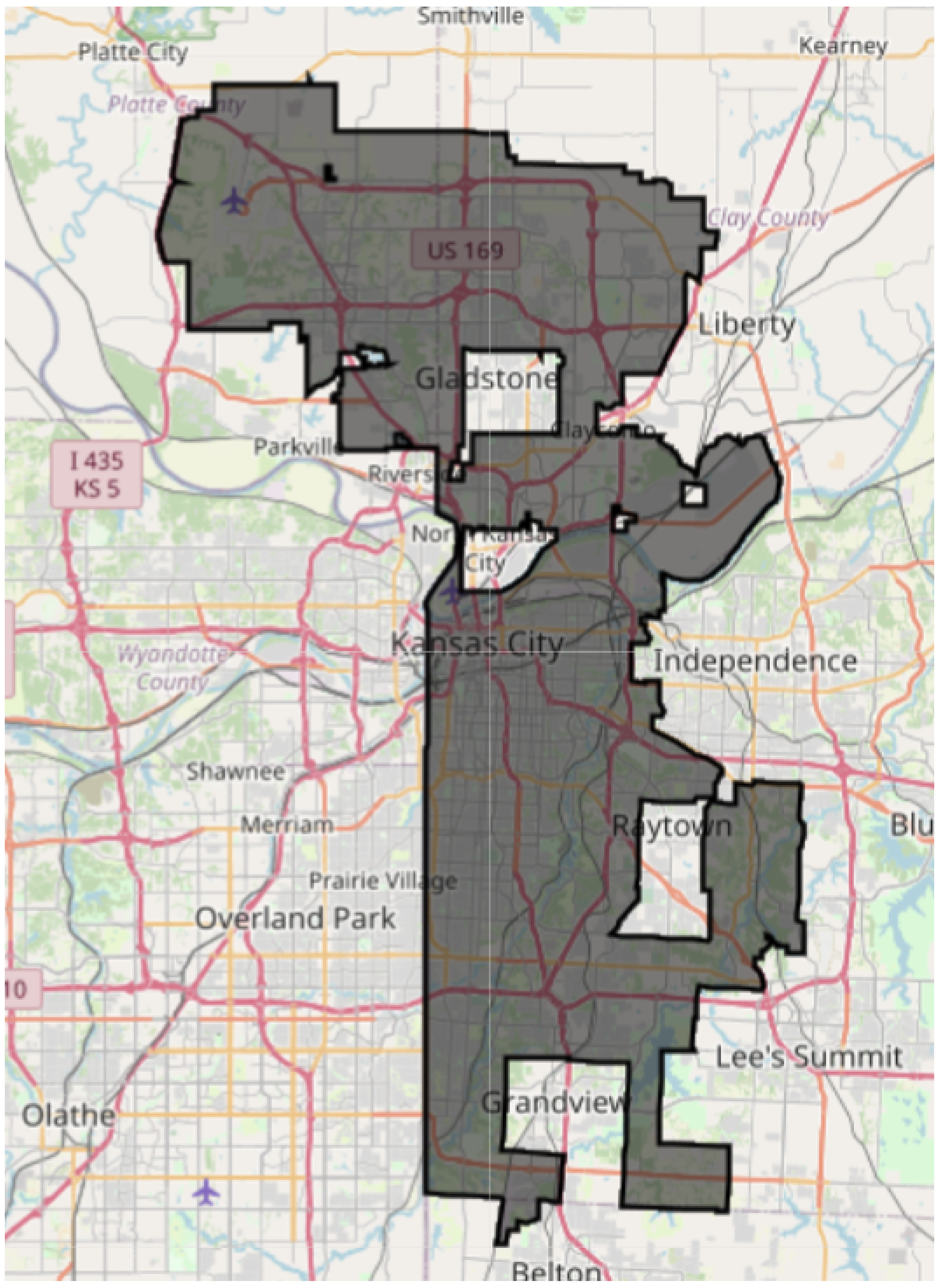
Landsat is a joint program between USGS and NASA to provide satellite images of the Earth’s surface. Through Google Earth Engine, it’s possible to access data dating from 1972 to present day, with the satellites providing images at a 30-meter resolution every 1-2 weeks (Google Earth Engine, 2022). The raw images with details of the reflective light bands are available, as well as data that has precomputed vegetation indices. In this exercise, we use data from Landsat 5 (1984-2012), Landsat 7 (1999-2022) and Landsat 8 (2013-2022), to provide an almost 40 year picture of how green space changes in cities. The Google Earth Engine API offers alternative datasets such as MODIS, which provides similar aerial data but with improved temporal resolution, but worse spatial resolution, at 250 meters. The Landsat dataset's 30 meter resolution offers improved spatial resolution, important for a project like this where we analyze the green space detail of cities.

For easier computation and reusability, we define a class, Landsat, to load the data as we import multiple Landsat datasets to fully cover this time period. The datasets can be loaded as follows:

```
# Load the Landsat Data
class Landsat:
  def __init__(self, name, short_name, start_date, end_date):
    self.name = name
    self.short_name = short_name
    self.start_date = start_date # First available date for Landsat data
    self.end_date = end_date # Last available date for Landsat data

LANDSAT_5 = Landsat("LANDSAT/LT05/C01/T1_8DAY_NDVI", "LANDSAT 5" ,"1984-01-01", "2012-04-30") # 36 years of data
LANDSAT_7 = Landsat("LANDSAT/LE07/C01/T1_8DAY_NDVI", "LANDSAT 7" ,"1999-01-01", "2022-01-01") # 22 years of data
LANDSAT_8 = Landsat("LANDSAT/LC08/C01/T1_8DAY_NDVI", "LANDSAT 8" ,"2013-04-07", "2022-01-01") # 09 years of data
```

Define City Boundaries

A second API is needed to define the area of interest of the Landsat data. Boundaries-io provides a GeoJSON area bounded by US postal codes, with Canadian and United Kingdom options also available. You can sign up for this API through RapidAPI, and at time of writing, make 50 free calls per day, with Freemium options if more is required. An important assumption here is that although the US postal codes provide the official city boundaries, people's understandings of a large metropolitan area (such as Los Angeles or Atlanta) might differ, so some of these results may be surprising. For example, Kansas City is located across two US state boundaries, and therefore truncated postal codes, but functionally operates as a single city.



Example of a city that is officially separated by US state boundaries but that functionally operates as a single city, a potential limitation.

After signing up for the API, save your authentication key to use in code. This API uses the name of the city or county and the 2-letter state code to obtain the map boundaries. Keep reading for the full contents of the class, but to access the API and obtain boundaries for a given city, the following code is used.

```
## ------- City Boundaries & Map Visualizations --------
  def queryBoundaries(self, save_local=False):
    # Define API URL
    url = "https://vanitysoft-boundaries-io-v1.p.rapidapi.com/reaperfire/rest/v1/public/boundary/place/{}/state/{}"\
          .format(urllib.parse.quote(self.name), self.state)

    headers = {
      'x-rapidapi-host': "vanitysoft-boundaries-io-v1.p.rapidapi.com",
      'x-rapidapi-key': "REPLACE YOUR KEY HERE"
      }

    response = requests.request("GET", url, headers=headers)

    _json = json.loads(response.text)

    if save_local:
      # the json file where the output must be stored
      _filename = f"{self.name}-{self.state}.json"
      out_file = open(_filename, "w")
      json.dump(_json, out_file, indent = 3)
      out_file.close()

    self._boundaries = geemap.geojson_to_ee(_json)
    return self._boundaries
```

Visualizing Maps

After the Landsat and postal code boundary data is obtained, it's possible to visualize an interactive map of the city. `Folium` is a library that makes it easy to visualize GeoJSON data in an interactive manner. The code is structured by creating a map given a latitude and longitude central location, and then the postal code boundaries can be superimposed to visually define the area of interest.

```
def drawMapWithBoundaries(self, zoom=12):
    """
      Input:
        zoom (int): Optional argument for initial map zoom
      Output:
        Returns a Folium map centered at self.map_center
    """
    map = folium.Map(location=self.map_center, zoom_start=zoom)
    map.addLayer(self._boundaries)
    return map
```

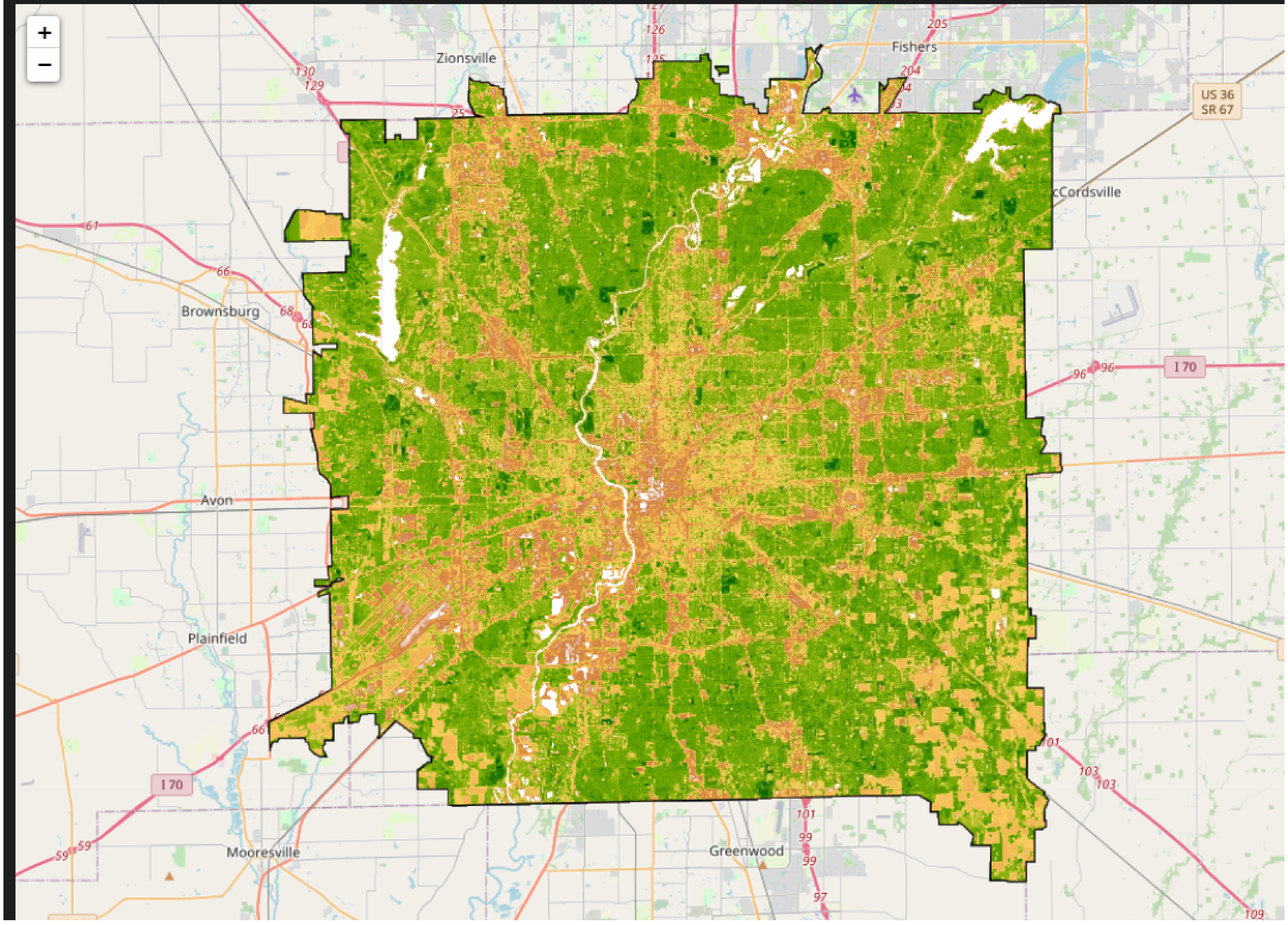
Visualizing NDVI

As the Landsat collection we loaded contains pre-computed NDVI results, these can be visualized on the previously created map simply by defining the date range. A color palette can also be defined, but generally ranges from darker green signifying areas with high amounts of vegetation, to red signifying a lack of vegetation. The NDVI can be limited to the area of interest by using the postal code boundaries defined previously. An example of this can be seen below with Indianapolis.

```
def visualizeNdvi(self, landsat_obj, start=None, end=None):
    """
    Display the 8-day NDVI obtained from a specific Landsat colection.
    Please note: This method will display the first 8-day ndvi available for the range of dates provided.
    Hence, you must change the pair of start and end dates to observe different images.
    Input:
    - landsat_obj (Landsat): Which landsat collection to extract data from
    - start/end (string): Dates to filter for ndvi images (yyyy-mm-dd)
    """
    landsat = ee.ImageCollection(landsat_obj.name)
    start_date = start if start != None else landsat_obj.start_date
    end_date = end if end != None else landsat_obj.end_date

    # Filter area and dates of interest
    landsat_AOI = landsat.filterBounds(self._boundaries).\
                  filterDate(start_date,end_date)

    # Define color pallet for map visualization
    palette = [
      'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901',
      '66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01',
      '012E01', '011D01', '011301']

    ndvi_parameters = {'min': 0,
                       'max': 1,
                       'dimensions': 512,
                       'palette': palette}

    # Obtain the first 8-day NDVI Composite available on the range of dates
    img_date = landsat_AOI.first().getInfo()['id'].split("/")[-1]
    print("Showing the 8-day average NDVI obtained on:", img_date)

    map = self.drawMapWithBoundaries()
    map.addLayer(landsat_AOI.first().clip(self._boundaries), ndvi_parameters)
    return map
```

```
indianapolis.visualizeNdvi(LANDSAT_8, start="2021-03-10", end="2021-03-30")
```

```
Showing the 8-day average NDVI obtained on: 20210314
```

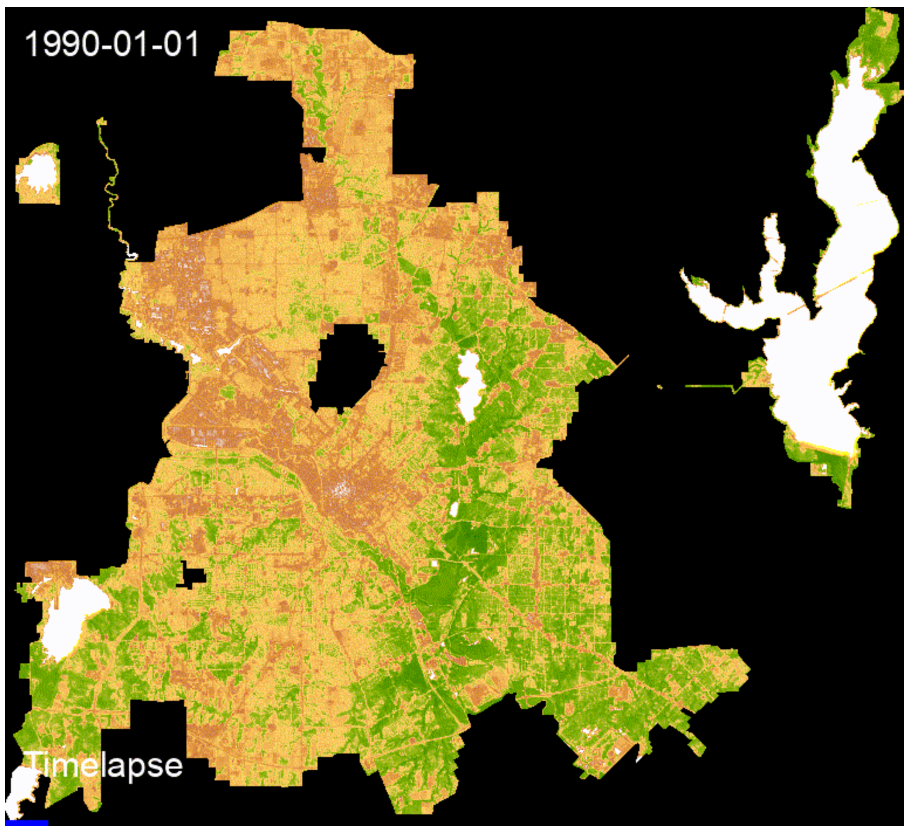


NDVI overlayed on the map of Indianapolis constrained by the city's postal code boundaries.

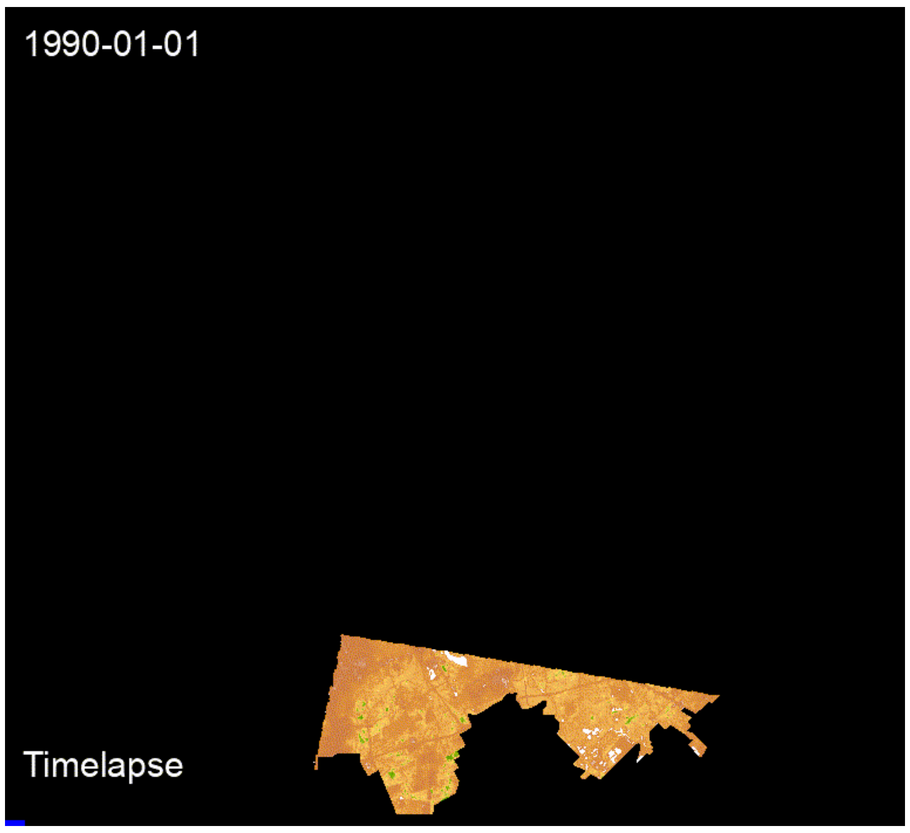
Timelapses

Using an open-source, web-based tool for geospatial visualizations created by Qiusheng Wu, we created multiple time-lapses to visualize NDVI changes for a few cities.

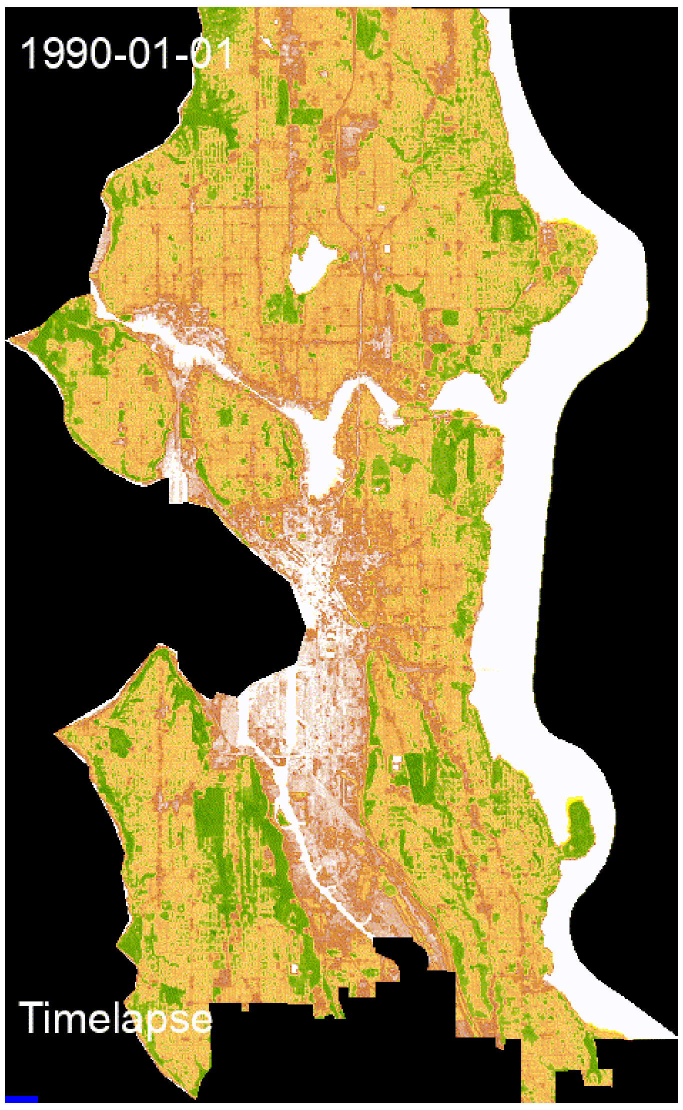
To create the time-lapses, the same Landsat collections used to extract the NDVI data were used. Time-lapses until 2010 were generated using Landsat 5, and those past 2010 with Landsat 7. Lastly, to determine the city boundary, `geojsons` files are generated with the boundary specifications using the previously defined `UsNdviModel` class.



Dallas, TX. Average annual NDVI for 1990-2010



Dallas, TX. Monthly average NDVI for 1990-1993



Seattle, WA. Average annual NDVI between 1990-2010

1994-01-01
Timelapse
Seattle, WA. Avereage monthly NDVI between 994-1997

To visualize all the time-lapses generated, refer to GitHub.

NDVI Computation and Manipulation

After obtaining and visualizing the data, the monthly average NDVI results are saved to a pandas data frame for exploratory data analysis. As we are working with multiple Landsat datasets to expand the time range for analysis, the data frames also need to be merged. Finally, some plotting functions can be defined to visualize these results.

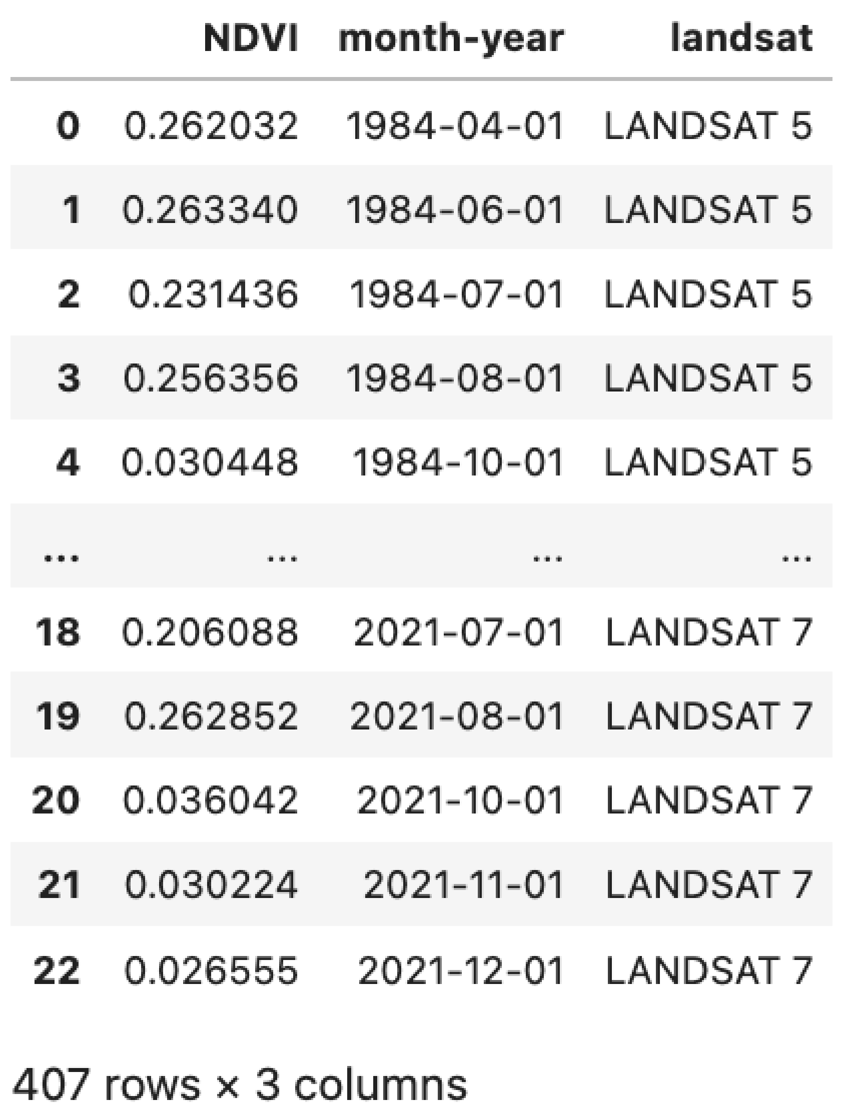
The full US NDVI class which contains code from the previous sections is available below.

```
class UsNdviModel:
  def __init__(self, name, state, map_center=[0,0]):
    """
      name (str): The name of the city/county
      state (str): The 2-letter state code
      map_center (list): A list of lat and long coordinates to draw the map center
    """
    self.name = name
    self.state = state
    self.map_center = map_center
    self.queryBoundaries()

  ########################################################
  ## ------- City Boundaries & Map Visualizations --------
  def queryBoundaries(self, save_local=False):
    # Define API URL
    url = "https://vanitysoft-boundaries-io-v1.p.rapidapi.com/reaperfire/rest/v1/public/boundary/place/{}/state/{}"\
          .format(urllib.parse.quote(self.name), self.state)

    headers = {
      'x-rapidapi-host': "vanitysoft-boundaries-io-v1.p.rapidapi.com",
      'x-rapidapi-key': "{INSERT API KEY HERE}"
      }

    response = requests.request("GET", url, headers=headers)

    _json = json.loads(response.text)

    if save_local:
      # the json file where the output must be stored
      _filename = f"{self.name}-{self.state}.json"
      out_file = open(_filename, "w")
      json.dump(_json, out_file, indent = 3)
      out_file.close()

    self._boundaries = geemap.geojson_to_ee(_json)
    return self._boundaries

  def drawMapWithBoundaries(self, zoom=12):
    """
      Input:
        zoom (int): Optional argument for initial map zoom
      Output:
        Returns a Folium map centered at self.map_center
    """
    map = folium.Map(location=self.map_center, zoom_start=zoom)
    map.addLayer(self._boundaries)
    return map

  def visualizeNdvi(self, landsat_obj, start=None, end=None):
    """
    Display the 8-day NDVI obtained from a specific Landsat colection.
    Please note: This method will display the first 8-day ndvi available for the range of dates provided.
    Hence, you must change the pair of start and end dates to observe different images.
    Input:
    - landsat_obj (Landsat): Which landsat collection to extract data from
    - start/end (string): Dates to filter for ndvi images (yyyy-mm-dd)
    """
    landsat = ee.ImageCollection(landsat_obj.name)
    start_date = start if start != None else landsat_obj.start_date
    end_date = end if end != None else landsat_obj.end_date

    # Filter area and dates of interest
    landsat_AOI = landsat.filterBounds(self._boundaries).\
                  filterDate(start_date,end_date)

    # Define color pallet for map visualization
    palette = [
      'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901',
      '66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01',
      '012E01', '011D01', '011301']

    ndvi_parameters = {'min': 0,
                       'max': 1,
                       'dimensions': 512,
                       'palette': palette}
    # Obtain the first 8-day NDVI Composite available on the range of dates
    img_date = landsat_AOI.first().getInfo()['id'].split("/")[-1]
    print("Showing the 8-day average NDVI obtained on:", img_date)

    map = self.drawMapWithBoundaries()
    map.addLayer(landsat_AOI.first().clip(self._boundaries), ndvi_parameters)
    return map

  #################################################
  # ------ NDVI Computations & Manipulation -------

  def singleLandsatHistoricalNdvi(self, landsat_obj:Landsat, start:str = "", end:str = "", show_logs=False):
    """
      Creates a pandas dataframe with the monthly avg ndvi for a specific Landsat
      Input:
      - landsat_obj (Landsat): Which landsat to extract data from
      - start (str, optional): Date to start filtering data.
      - end (str, optional): Date to end filtering data.
      * obs: if start or end is not provided, the default dates are the Landsat's
            date range
      Outputs:
      - A pandas dataframe with the monthly ndvi
    """

    landsat = ee.ImageCollection(landsat_obj.name)
    start_date = start if start != "" else landsat_obj.start_date
    end_date = end if end != "" else landsat_obj.end_date

    if show_logs:
      print("Gathering NDVI from "+ landsat_obj.short_name + " from {} to {}"
          .format(start_date, end_date))

    start_year, end_year = int(start_date.split('-')[0]), int(end_date.split('-')[0])
    landsat_AOI = landsat.filterBounds(self._boundaries).filterDate(start_date,end_date)


    global COLLECTION, BOUNDARIES, MONTHS
    COLLECTION = landsat_AOI
    BOUNDARIES = self._boundaries

    MONTHS = ee.List.sequence(1,12)
    years = ee.List.sequence(start_year,end_year)

    output = years.map(yearFunc).getInfo()
    df = self.convertToDataFrame(output, landsat_obj.short_name)

    self.df = df
    return df


  def mergeLandsatData(self, show_logs=False):
    """
      Creates a pandas dataframe with the monthly avg ndvi for ALL Landsats
    """
    # Because there is a limit amount of data, we get Landsat data in batches
    # We use LANDSAT 5 and LANDSAT 7 to gather data from 1984 to 2022
    data_frames = []
    start_year, end_year = 1984, 2022
    curr_year = start_year
    yr_batch = 3

    while curr_year < end_year:
      landsat = LANDSAT_5 if curr_year < 1999 else LANDSAT_7
      upper_bound = curr_year + yr_batch
      if upper_bound > end_year:
        upper_bound = end_year

      data_frames.append(self.singleLandsatHistoricalNdvi(landsat, f"{curr_year}-01-01", f"{upper_bound}-01-01", show_logs=show_logs))

      curr_year = upper_bound

    merged_df = pd.concat(data_frames)
    self.merged_df = merged_df.copy()
    return merged_df

  def saveDataFrameToCSV(self, filepath=""):
    if not hasattr(self, "merged_df"):
      self.mergeLandsatData()

    if filepath=="":
      filepath = '{}_{}.csv'.format(self.name, self.state)

    self.merged_df.to_csv(filepath, index=False)

  ################################
  #---------- PLOTS -----------
  def plotNdvi(self, figsize=(30,15)):
    """
    Plot time-series ndvi data from a single LANDSAT collection
    """
    if not hasattr(self, "df"):
      raise Exception("You must run 'singleLandsatHistoricalNdvi' before executing this function")

    plt.figure(figsize=figsize)
    plt.plot(self.df["month-year"], self.df["NDVI"], '-o', label=self.df["landsat"][0])

    ax = plt.gca()
    ax.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    plt.gcf().autofmt_xdate() # Rotation
    plt.legend()
    plt.show()

  def plotMergedNdvi(self, figsize=(20,12)):
    """
    Plot time-series NDVI from multiple LANDSAT collections
    """
    if not hasattr(self, "merged_df"):
      self.mergeLandsatData()

    _df = self.merged_df

    plt.figure(figsize=(20,12))
    landsats = list(np.unique(_df["landsat"]))
    for val in landsats:
      plt.plot(_df.loc[_df["landsat"]==val]["month-year"], _df.loc[_df["landsat"]==val]["NDVI"], label=val)

    year_avg = self.merged_df.groupby(pd.Grouper(key='month-year', freq='1Y')).mean()
    plt.plot(year_avg["NDVI"].keys(), year_avg["NDVI"].values, '-o', label="Yearly average")

    plt.legend()
    plt.show()


  ###############################
  #------- HELPER METHODS -------
  def convertToDataFrame(self, output, landsat_short_name):
    ndvi_list = []
    date_list = []

    for lst in output:
      for dic in lst:
        if "NDVI" in dic and dic["NDVI"] != None:
          ndvi_list.append(dic["NDVI"])
          date_list.append("{}-{}".format(dic["month"], dic["year"]))

    data = {"NDVI":ndvi_list, "month-year":date_list}
    df =  pd.DataFrame(data)
    df["month-year"] = pd.to_datetime(df["month-year"], format="%m-%Y")
    df["landsat"] = landsat_short_name

    return df
```

After defining this class, the NDVI of cities in the US can be obtained by running the following code cell, replacing Houston with the city of interest.

```
houston = UsNdviModel("Houston", "TX", [29.759527, -95.355758])
```

The NDVI of each city can then be saved into a pandas data frame with the associated month, year, and Landsat collection.



	NDVI	month-year	landsat
0	0.262032	1984-04-01	LANDSAT 5
1	0.263340	1984-06-01	LANDSAT 5
2	0.231436	1984-07-01	LANDSAT 5
3	0.256356	1984-08-01	LANDSAT 5
4	0.030448	1984-10-01	LANDSAT 5
...	...	...	...
18	0.206088	2021-07-01	LANDSAT 7
19	0.262852	2021-08-01	LANDSAT 7
20	0.036042	2021-10-01	LANDSAT 7
21	0.030224	2021-11-01	LANDSAT 7
22	0.026555	2021-12-01	LANDSAT 7

407 rows × 3 columns

The monthly NDVI data from 1984 to 2021 of Seattle

To investigate how the green space changed in US State Capitals and major cities, we gather the NDVI data for 89 cities in total. The code to scrape Wikipedia for the most populous cities and to download the NDVI results into a CSV file are available in the GitHub.

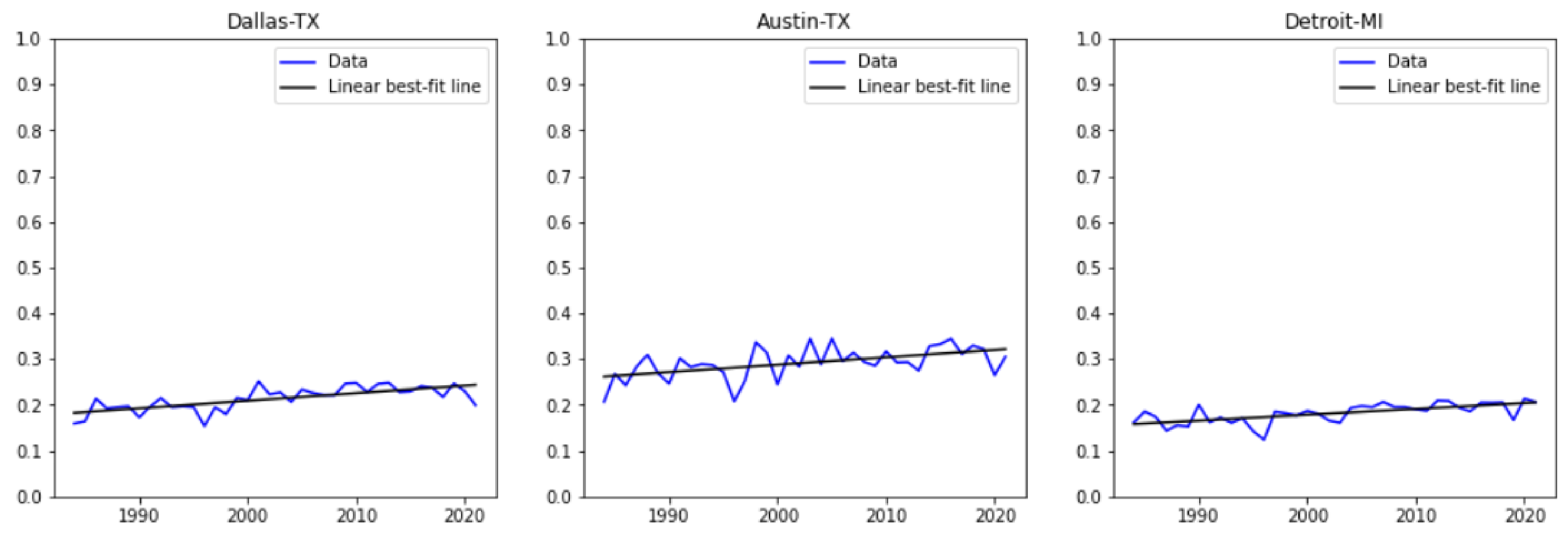
Exploratory Data Analysis (EDA)

Now that the NDVI results for each city are easily accessible, exploratory analysis can be conducted to see whether there are any patterns that emerge from the data.

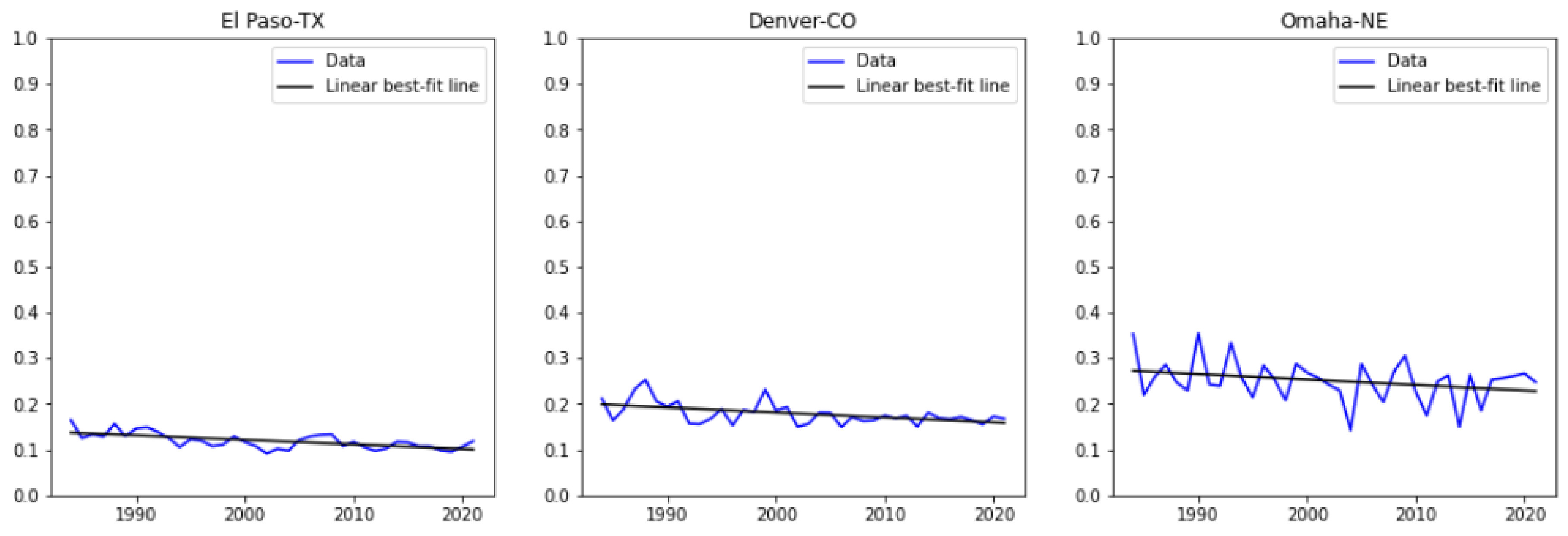
> ** THIS WAS SUPPOSED TO ZACH'S CONTRIBUTION ** Notes: Zach had done a little bit of EDA code for the second assignment, so we just reran that on the new data.

Yearly NDVI averages

For this step of EDA, we averaged NDVI observations per year and fitted a linear model to the data. Below, we display the three cities with steepest positive slope, indicating a general positive trend.



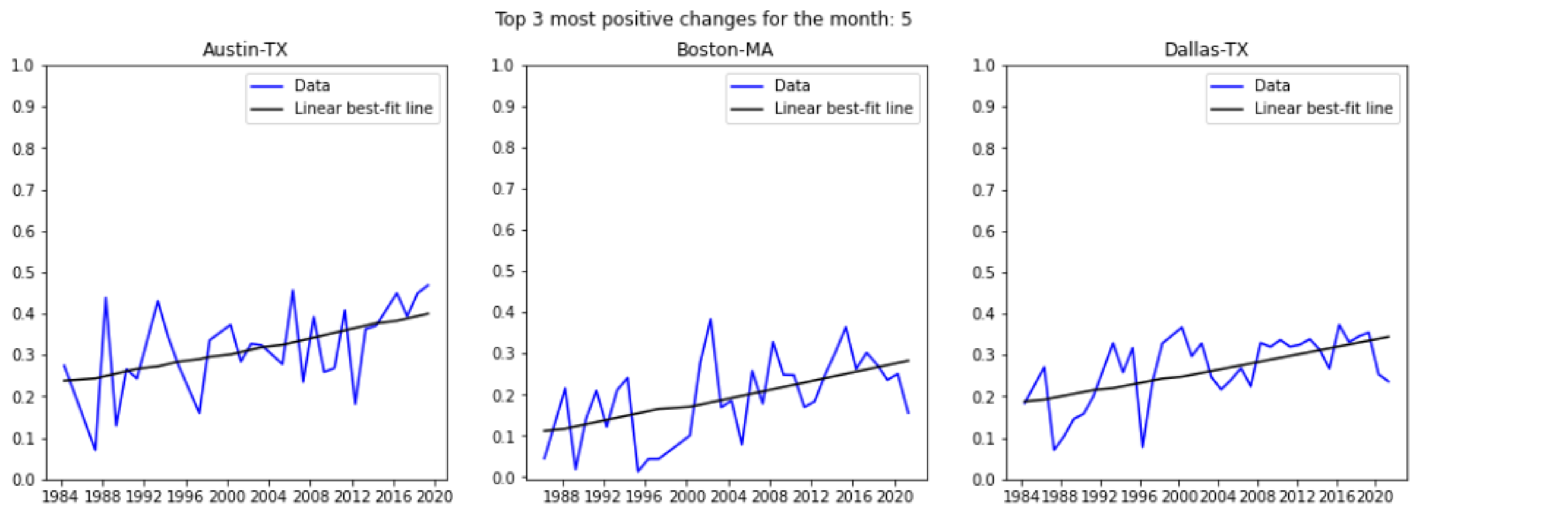
Similarly, the three cities with steepest negative slope, indicating a negative general trend, are shown below.



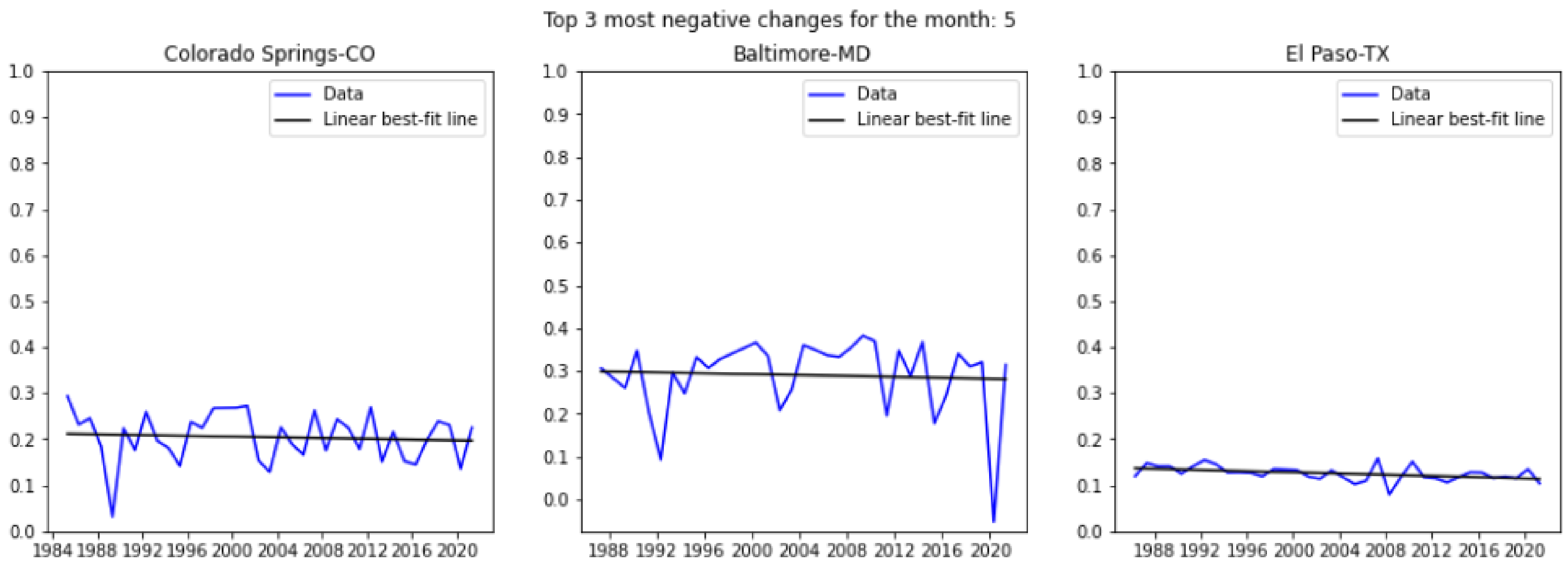
Monthly NDVI averages

In the second step of EDA analysis, we extract the NDVI averages for a specific month across multiple years for each city, to attempt to remove the influence of seasonality, and re-run the same analysis performed above. We found that results change depending on which month we run the analysis, indicating that seasonality plays a major confounding role, specially when comparing cities that experience significantly distinct weathers in the same period of the year.

For instance, if we focus on May only, the top 3 cities with the most significant upward trend are Austin, Boston and Dallas. Two out of three of these cities also appeared on the Yearly NDVI results list, which validates that the results are somewhat consistent, even across the seasonality influence of the yearly data. On the other hand, the 3 cities with the most negative overall trend are Colorado Springs, Baltimore, and El Paso, where only El Paso appears in the Yearly NDVI results as well.

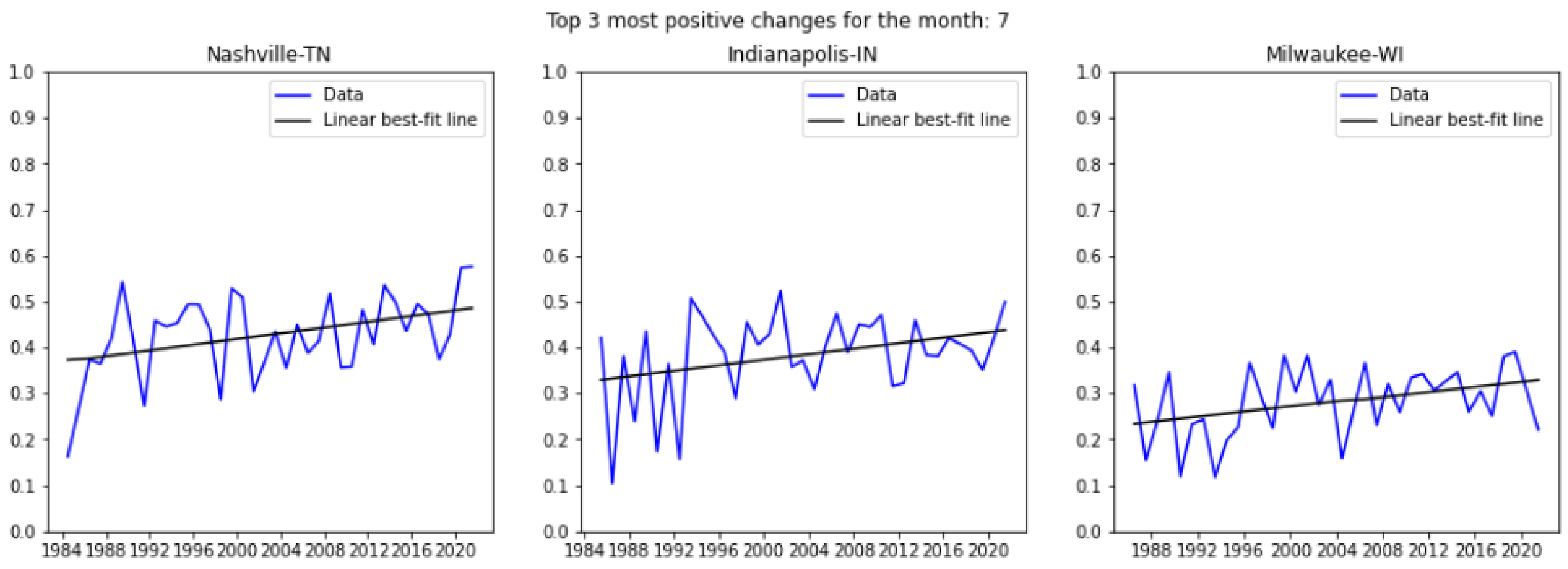


The top 3 cities with the most positive changes for the month of May

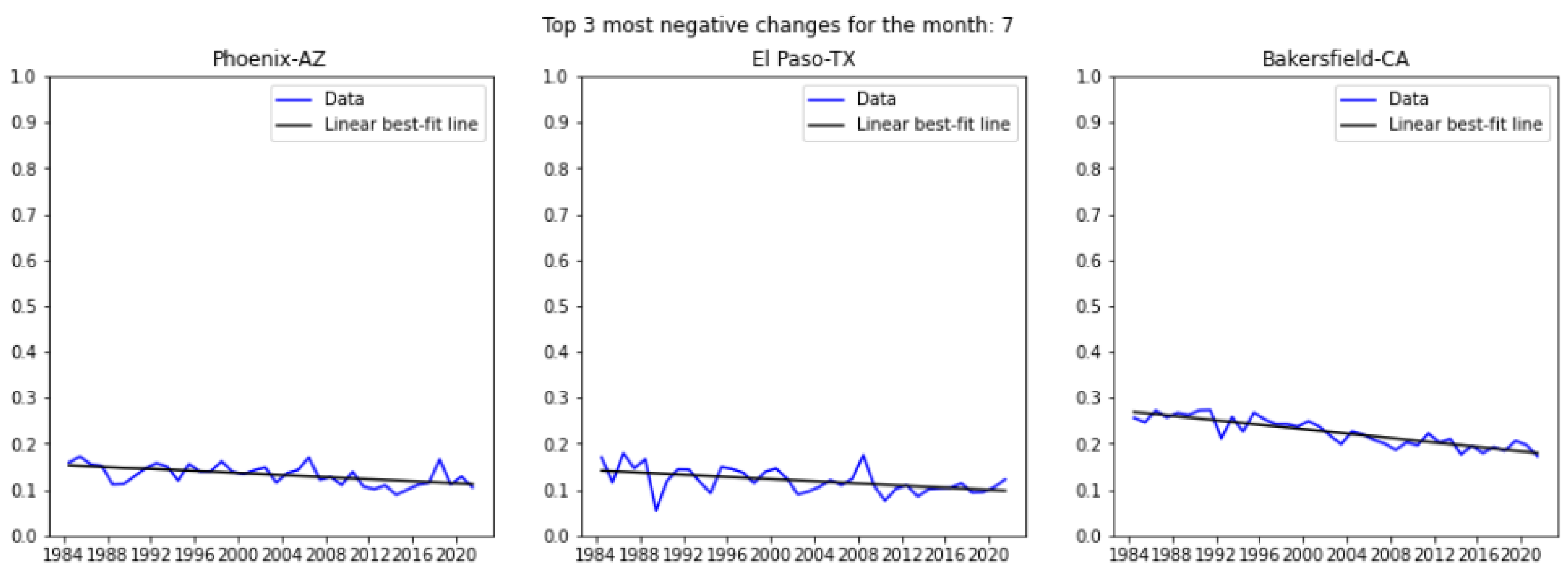


The top 3 cities with the most negative changes for the month of May

Similarly, if we instead take July, we find that Nashville, Indianapolis and Milwaukee are the cities with the most positive overall trend, whereas Phoenix, El Paso, and Bakersfield show the most negative overall trend. Interestingly, only El Paso appears in both the Yearly and July Monthly results list.



The top 3 cities with the most positive change for the month of July



The top 3 cities with the most negative change for the month of July

Continued Reading

We hope you've enjoyed exploring Google Earth Engine and seeing how the green space in US cities has evolved!

For further inspiration, you can explore these papers to see how researchers have been using satellite data and NDVI. Ranging from studying the impact of continuous deforestation in New England to correlating urban greenness to affluence in Johannesburg, there are many possible questions to investigate now that you know how to use Google Earth Engine!

- Impact of continuous deforestation in New England (Olofsson, Holden, Bullock, & Woodcock, 2016).
- Others used different satellite images but similarly applied NDVI as the metric of choice to quantify urban greenness, taking a similar approach to our case study by investigating affluent and poor suburb neighborhoods in Johannesburg (Abutaleb, Freddy Mudede, Nkongolo, & Newete, 2021).
- Similar work can be found classifying satellite images in Kaggle competitions and on Medium for Amazon rainforest deforestation that use Google Earth Engine satellite imagery and NDVI to calculate the deforestation.

Itemized List of Contributions

Isabel:

- Found and figured out how to get access to Google Earth Engine, decided on using the Landsat collection for our satellite images
- Found similar work (see the document I compiled at the beginning to get us all on the same page) and translated documentation from JavaScript to Python for the beginning code to use the API
- Explored a lot of 'Collections' on Google Earth Engine to figure out how to constrain our NDVI for the cities (GAUL, etc.) which failed
- Found the Boundaries.io API which had the US postal codes (Implemented one afternoon upstairs with Lucas)
- Coworked with Lucas to get the basic code that I had found and adapted from Javascript/Medium posts working for Assignment 1
- Wrote the Assumption Explanations and the Method Justification pieces for Assignment 1
- Wrote the majority of the Write Up Draft for Assignment 2 (minus the EDA section)
- Wrote the majority of the Final Project, Lucas and I cobbled the EDA part together towards the end of the day with Zach's code from Assignment 2.

Lucas:

- Explored multiple Landsat Collections available on the Google Earth Engine catalog, including Surface Reflectance Lansats 5, 7, and 8.
- Integrated the Boundaries.io API with Folium maps, and the Earth Engine API in order to collect and visualize NDVI data within a city boundary
- Wrote the significant majority of the LANDSAT_NDVI_Composite notebook, which defines the `UsNdviModel` class shown above. This class is an excellent abstract implementation to obtain and visualize NDVI data for *any* American city.
    - I used this notebook to generate all `csv` files containing NDVI data for 89 US Cities (50 most populous cities + remaining state capitals)
    - This notebook also contains logic to visualize NDVI indexes on Folium maps, and can also be used to export *geojsons* with city boundaries, which can be imported into other tools, or reused.
- Built upon Zach's Assignment 2 EDA notebook to conduct EDA for the 50 most populous US cities. In addition to Zach's work, I also included visualizations for the NDVI changes comparing the same month accross multiple years
- Created time-lapses to visualize NDVI changes for a few cities.
- Oversaw aspects related to project organization, including: directory organization, GitHub's pull requests, and Python virtual environments.

References

Abutaleb, K., Freddy Mudede, M., Nkongolo, N., & Newete, S. W. (2021). Estimating urban greenness index using Remote Sensing Data: A case study of an affluent vs poor suburbs in the City of Johannesburg. *The Egyptian Journal of Remote Sensing and Space Science*, *24*(3), 343–351. https://doi.org/10.1016/j.ejrs.2020.07.002

Earth Observing System. (2019, August 30). NDVI FAQS: Top 23 frequently asked questions about ndvi. EARTH OBSERVING SYSTEM. Retrieved from https://eos.com/blog/ndvi-faq-all-you-need-to-know-about-ndvi/#:~:text=Consider%20using%20EVI%20(Enhanced%20Vegetation,crops%20or%20other%20dense%20crops

Hafen, K. (2021, February 9). Remote Sensing with QGIS: Calculate ndvi. OpenSourceOptions. Retrieved from https://opensourceoptions.com/blog/remote-sensing-with-qgis-calculate-ndvi/

Levy, R. and Przyborski P. (2000, August 30). Measuring Vegetation (NDVI & EVI). Retrieved from https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_2.php

Olofsson, P., Holden, C. E., Bullock, E. L., & Woodcock, C. E. (2016). Time series analysis of satellite data reveals continuous deforestation of New England since the 1980s. *Environmental Research Letters*, *11*(6), 064002. https://doi.org/10.1088/1748-9326/11/6/064002