L1: Geospatial Data

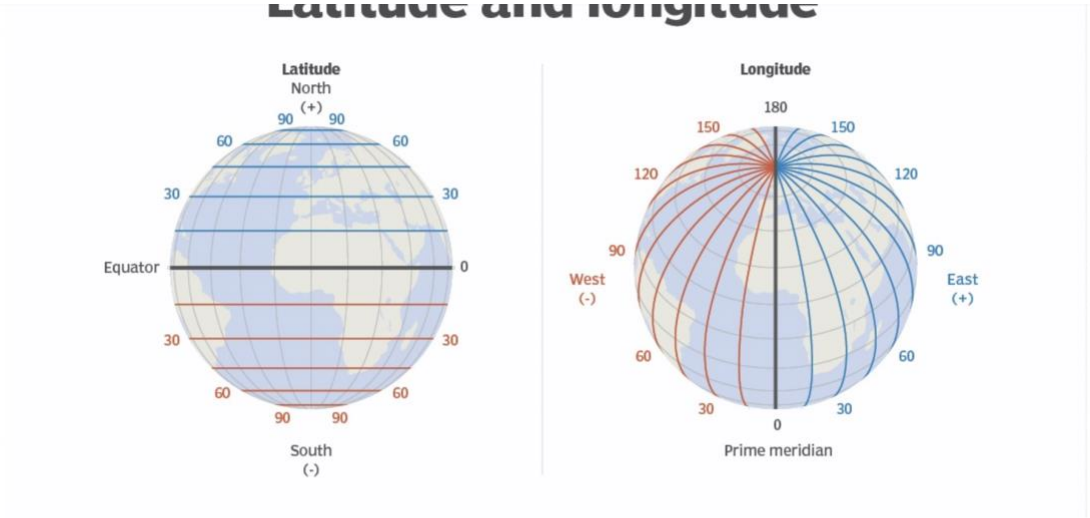
2. Geospatial Data

2.1. What is Geospatial Data?

Geospatial Data is a type of dataset containing information about features on Earth's surface. What is a feature? Things or events we can observe on Earth's surface are called features. For example, a tree, a building, a road, and a river are examples of features. These examples are static features that do not move or change very quickly. Features can also be dynamic, like a car's travel route or the path of a hurricane.
There are two components of feature information in a geospatial dataset:**feature position** and **feature attributes**.

Feature position tells us where this feature is located on a coordinate system on Earth's surface. Depending on data types within the geospatial dataset, there are different methods of presenting feature position information. We'll discuss this more later on in this lesson.
Feature Attributes tells us other non-geolocation information, such as a building's height, year built etc. Another example can be a hurricane's speed and pressure at certain time.

When discussing feature position, we talked about representing a location on a coordinate system for Earth surface. In the next section, we are going to learn more about the coordinate systems used in geospatial data.



Source: TechTarget
When referencing a location on a CRS, the general consensus is to start first with latitude and then longitude in the form of (latitude, longitude). There are two ways to present latitude-longitude information. The first method is degrees, minutes, and seconds (DMS). For example, the coordinates using DMS for New York are (40° 43' 50.1960'' N, 73° 56' 6.8712'' W). The ° denotes degree. The ' denotes minute, and the '' denotes second. The second method is decimal degrees (DD). The coordinates using DD for New York are (40.730610, -73.935242).

There are two main differences between DMS and DD. The first difference is that DMS and DD use different symbols to represent the direction information of a location. DMS uses characters to describe which hemisphere the location is in (N vs S; W vs E). DD uses + and - signs to describe the hemisphere location. In the DMS method, New York's direction coordinates are (N, W). In the DD method, New York's direction coordinates are (+,-). We can use Figure 1 to understand how the DD method assigns +/- signs. On the left part of the figure, we can see that if a location is in the Northern Hemisphere, the latitude sign is +. If the location is in the south, the latitude sign is -. The concept is the same when applied to longitude.
The second difference is that DMS is in degrees and DD is in decimals. We can easily convert between the two methods. To convert DMS to DD, first we keep the degree number. Then, we divide the minute number by 60 and divide the second number by 3600. We then add the results of the two divisions and add them back to the degree number. The final number will be the number in DD. Sometimes, which hemisphere the data is in is indicated by the variable name or header, and the values of data points are all positive in the table. In this case, we need to manually adjust the values of the data points to properly represent the hemisphere location in the DD method.
In Python, we will use the DD method to present coordinate information. You can easily find the DD coordinates for a place in Google Maps.

4. Types of Geospatial Data

In the last section, we introduced the latitude-longitude CRS. In the feature position section, we talked about how different geodata types will use different methods to present location information. There are two main geodata types: raster data and vector data.

Raster data divides a feature into a grid system. Each grid unit is called a pixel or a cell. Raster data stores feature attributes based on each pixel. In the satellite imagery lesson, we will talk more about raster data.

Vector data stores feature attributes based on feature shape. A feature can have a point shape, a line shape, or a polygon shape. In this lesson, we'll focus on vector data.

Before moving on to the next topic of vector data, let's take a look at Figure 2 to get a visual understanding of the difference between the two geodata types.

Figure 2. Vector Data vs Raster Data

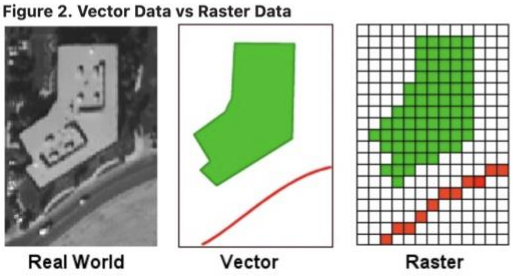
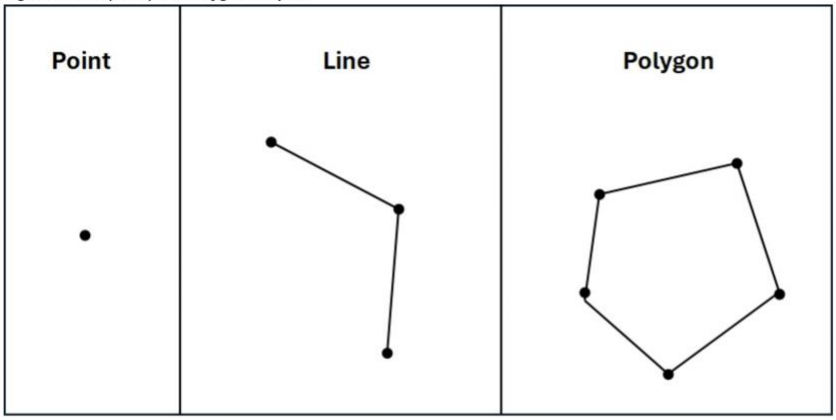


Figure 3. Point, Line, and Polygon Shapes



The above three shapes in Figure 3 consist of points and lines. The points are called vertices. The location of each vertex can be represented by a latitude-longitude coordinate pair. By having shape information and vertex coordinate information, we can identify the geospatial location of a feature on a latitude-longitude coordinate system. In vector data, we store the shape information and coordinate information in a variable called **geometry**. It is this geometry variable that differentiates geospatial data from traditional tabular data.

In terms of vector geospatial data format, there are two common data formats: **GeoJSON** and **Shapefile**. The required readings will give a very short introduction to these two data formats.

6. Geospatial Data Source and Analysis Software

There are several ways to collect geospatial data. Some of the sources that collect data include the Global Positioning System (GPS), wearable devices, mobile phones, remote sensing (satellite), and internet of things (IoT) to name a few.

After getting geospatial data, researchers usually use a **geographic information system (GIS)** to read and analyze the data. There are many popular paid or free GISs, like ArcGIS or QGIS. These types of software usually have built-in functions to read different types of geospatial data and visualize functions to present the data.

7. Application of Geospatial Data

In this section, we are going to use Python to demonstrate how to pull geospatial data from online data resource, clean the data to be ready for analysis and visualize the data.

We are going to use Hurricane Irene's path along the east coast of the United States as an example. In late August of 2011, Hurricane Irene caused severe damage to this region of the United States. For example, lower New York City lost electricity for one week as a result of the storm.

In this application, we will pull the hurricane's path and wind data from the National Hurricane Center (NHC) of the National Oceanic and Atmospheric Administration (NOAA). We will pull the data and draw the hurricane's path on a map. In this demonstration, we are also going to illustrate how to overlay different information on a map. We will overlay U.S. state borders on the map to show which states were affected by this hurricane.

The main Python package to use for geospatial data analysis is geopandas. Geopandas is the geospatial data version of pandas. It can handle the various types of geospatial data we mentioned in previous sections. We then will use the folium package to visualize the hurricane's path. Let's get started.

[8]:
```
# Convert PDF file to csv file, and then a pandas dataframe
tabula.convert_into(irene_pdf, "irene.csv", output_format="csv", stream=True, pages = 9)
irene_1 = pd.read_csv("irene.csv")
irene_1
```

```
Got stderr: Jan 15, 2025 2:29:33 PM org.apache.pdfbox.pdmodel.font.PDTrueTypeFont <init>
WARNING: Using fallback font 'LiberationSans' for 'TimesNewRomanPSMT'
Jan 15, 2025 2:29:33 PM org.apache.pdfbox.pdmodel.font.PDTrueTypeFont <init>
WARNING: Using fallback font 'LiberationSans' for 'TimesNewRomanPS-BoldMT'
Jan 15, 2025 2:29:34 PM org.apache.pdfbox.pdmodel.font.PDTrueTypeFont <init>
WARNING: Using fallback font 'LiberationSans' for 'TimesNewRomanPSMT'
Jan 15, 2025 2:29:34 PM org.apache.pdfbox.pdmodel.font.PDTrueTypeFont <init>
WARNING: Using fallback font 'LiberationSans' for 'TimesNewRomanPS-BoldMT'
```

[8]:

	Date/Time	Latitude	Longitude	Pressure	Wind Speed	Unnamed: 5
0	(UTC)	(°N)	(°W)	(mb)	(kt)	Stage
1	21 / 0000	15.0	59.0	1006	45	tropical storm
2	21 / 0600	16.0	60.6	1006	45	"
3	21 / 1200	16.8	62.2	1005	45	"
4	21 / 1800	17.5	63.7	999	50	"
5	22 / 0000	17.9	65.0	993	60	"
6	22 / 0600	18.2	65.9	990	65	hurricane
7	22 / 1200	18.9	67.0	989	70	"
8	22 / 1800	19.3	68.0	988	75	"
9	23 / 0000	19.7	68.8	981	80	"
10	23 / 0600	20.1	69.7	978	80	"
11	23 / 1200	20.4	70.6	978	80	"

7.3 Creating the Date-Time Variable

Next, we can see that the date/time variable does not contain month and year information. Therefore, we're going to create a new date/time variable to provide complete date/time information.

[11]:
```
# Create a new Date Time variable to contain month and year information
irene_1[["Date","Time"]] = irene_1["Date/Time"].str.split(" / ", expand = True)
irene_1["Date_Time"] = "08/" + irene_1["Date"] + "/2011/" + irene_1["Time"]
irene_1.set_index("Date_Time")
irene_1
```

[11]:

	Date/Time	Latitude	Longitude	Pressure	Wind Speed	Unnamed: 5	Date	Time	Date_Time
1	21 / 0000	15.0	59.0	1006.0	45.0	tropical storm	21	0000	08/21/2011/0000
2	21 / 0600	16.0	60.6	1006.0	45.0	"	21	0600	08/21/2011/0600
3	21 / 1200	16.8	62.2	1005.0	45.0	"	21	1200	08/21/2011/1200
4	21 / 1800	17.5	63.7	999.0	50.0	"	21	1800	08/21/2011/1800
5	22 / 0000	17.9	65.0	993.0	60.0	"	22	0000	08/22/2011/0000
6	22 / 0600	18.2	65.9	990.0	65.0	hurricane	22	0600	08/22/2011/0600
7	22 / 1200	18.9	67.0	989.0	70.0	"	22	1200	08/22/2011/1200
8	22 / 1800	19.3	68.0	988.0	75.0	"	22	1800	08/22/2011/1800
9	23 / 0000	19.7	68.8	981.0	80.0	"	23	0000	08/23/2011/0000
10	23 / 0600	20.1	69.7	978.0	80.0	"	23	0600	08/23/2011/0600
11	23 / 1200	20.4	70.6	978.0	80.0	"	23	1200	08/23/2011/1200
12	23 / 1800	20.7	71.2	977.0	80.0	"	23	1800	08/23/2011/1800

7.4 Ensuring Spatial Data is Correct

We also notice that the data values for longitude in the dataset are all positive. However, the direction for longitude is "W." The numbers for longitude should all be negative based on Figure 1 in section 3. Hence, we need to add a negative sign in front of all numbers for the longitude variable to correctly reflect the location.

[15]:
```
# Adjust Longitude value to correctly reflect the geolocation
irene_1['Longitude'] = 0 - irene_1['Longitude']
```

[17]:
```
# Select the variables we are interest for next steps
irene_2 = irene_1[["Date_Time","Longitude","Latitude","Wind Speed"]]
irene_2
```

[17]:

	Date_Time	Longitude	Latitude	Wind Speed
1	08/21/2011/0000	-59.0	15.0	45.0
2	08/21/2011/0600	-60.6	16.0	45.0
3	08/21/2011/1200	-62.2	16.8	45.0
4	08/21/2011/1800	-63.7	17.5	50.0

7.5 Converting to a Geodataframe

Now we have a good dataframe for Hurricane Irene's path. To convert this dataframe to a geospatial dataframe, we need to create a geometry variable, as explained in the previous section. We will use a method from geopandas to create this variable. One of the parameters in the following code is "EPSG:4326", which is the code name for the latitude-longitude system we are familiar with. Remember there are several CRS systems available, but we'll proceed with this commonly used one.

[18]:
```
# Create a geolocation variable
geometry = gpd.points_from_xy(irene_2.Longitude, irene_2.Latitude, crs="EPSG:4326")
```

Now let's convert the current pandas dataframe to a geodataframe.

[19]:
```
# Convert the current dataframe to a geodataframe with geometry variable
irene_3 = gpd.GeoDataFrame(
    irene_2, geometry=geometry, crs="EPSG:4326"
)
```

Great! Now we have a geodataframe. Let's check out the first five entries in this dataframe.

[20]:
```
irene_3.head()
```

[20]:

	Date_Time	Longitude	Latitude	Wind Speed	geometry
1	08/21/2011/0000	-59.0	15.0	45.0	POINT (-59 15)
2	08/21/2011/0600	-60.6	16.0	45.0	POINT (-60.6 16)
3	08/21/2011/1200	-62.2	16.8	45.0	POINT (-62.2 16.8)
4	08/21/2011/1800	-63.7	17.5	50.0	POINT (-63.7 17.5)
5	08/22/2011/0000	-65.0	17.9	60.0	POINT (-65 17.9)

[21]:
```
type(irene_3)
```

[21]:
```
geopandas.geodataframe.GeoDataFrame
```

And let's check our geometry variable.

[22]:
```
type(irene_3['geometry'])
```

[22]:
```
geopandas.geoseries.GeoSeries
```

The geodataframe basically behaves like the pandas dataframe. Therefore, we can apply the same methods and analysis from pandas to geopandas. Here is one example.

[23]:
```
print("Mean wind speed of Hurricane Irene is {} knots and it can go up to {} knots maximum".format(round(irene_2['Wind S
                                                                                                    irene_2['Wind Speed'].max())+".
```

```
Mean wind speed of Hurricane Irene is 69.6154 knots and it can go up to 105.0 knots maximum.
```

7.6 Data for U.S. State Borders

Before we draw Hurricane Irene's path on a map, we would like to add a layer with U.S. state borders to the map. With the state border visualization along with Hurricane Irene's path, we can learn which states were affected by this hurricane. We will pull the state border file from the United States Census Bureau's website. This is a zipped shapefile. We first need to import a package to unzip the file.

[24]:
```
# Import a file unzip package
from zipfile import ZipFile
```

[25]:
```
# Retrieve US State Shapfile from United States Census Bureau
us_state_url = "http://www2.census.gov/geo/tiger/TIGER2012/STATE/tl_2012_us_state.zip"

us_state_shape_zip, _ = urllib.request.urlretrieve(us_state_url)
```

[26]:
```
# Unzip the zipped shapefile and assign it a new file name "us_state_shape"
with ZipFile(us_state_shape_zip, 'r') as zObject:

    zObject.extractall("us_state_shape")
```

Once we unzip the file, we will use the read_file method from geopandas to read this file as a geodataframe for further data processing.

[27]:
```
# Read in our shapefile to a geodataframe
us_state_shape_g = gpd.read_file("us_state_shape")
```

[28]:
```
# Check the metadata of the new geodataframe
us_state_shape_g.info()
```

[34]:
```
# Draw Hurricane Irene's path and other infomation to a map

# First, create a basemap
map = folium.Map(location=[30,-102], zoom_start=4, control_scale=True)

# Then add the first layer of US state borders to the map
folium.GeoJson(us_state_shape_g).add_to(map)

# Then add the hurricane travel path to the map. We use a red dot to represent the hurricane's location at a specific d
folium.GeoJson(irene_3,
               marker=folium.Circle(radius=2000, fill_color="red", fill_opacity=0.4, color="red", weight=5),
              tooltip=folium.GeoJsonTooltip(fields=["Date_Time","Wind Speed"]),
              popup=folium.GeoJsonPopup(fields=["Date_Time","Wind Speed"]),).add_to(map)

map
```

[34]:

Voila! We just created a map overlayed with U.S. state borders and Hurricane Irene's path. In the upper left corner of the map, there is an icon you can use to zoom in and out. We see U.S. state borders in solid blue lines on the map. Hurricane Irene's path is represented by a series of red dots on the map. When you hover your cursor over one of the red dots, an information box will show up, providing date/time and wind speed information. With this map, we can see Hurricane Irene went through most of the northern states along the east coast of the U.S. Wind speed was at its strongest when the hurricane was passing through between the Dominican Republic and the Bahamas. The wind speed slowed down significantly after landfall.

8. Conclusion ¶

In this lesson, we first explained what geospatial data is. We introduced the concept of using a coordinate reference system (CRS) for providing geoposition information on Earth's surface. We also explained vector data and its common data formats. Then, we demonstrated a simple geospatial data application using Python. Through this application, we learned how to process different types of geospatial data. We also learned how to overlay processed data onto a map for data visualization.

L2: Satellite Imagery

1. Introduction

In this lesson, we are going to introduce a type of data that we have not touched upon yet: **satellite imagery**. Traditionally, satellite images are the type of data used in natural sciences or the defense sector. However, in recent years, more and more financial analysts have been using satellite images to conduct analyses, hoping to gain extra insights on top of traditional financial analysis to make better financial decisions. The content in this lesson draws from the information in your required readings. We'll first introduce the basics of satellite imagery. Then, we will briefly talk about some applications of satellite images in financial analysis. Finally, we will use Google Earth Engine (GEE) to access some satellite images and provide examples of how we can use satellite images for analysis.

2. Financial Application of Satellite Imagery

In this section, we are going to talk about how to incorporate satellite images as part of data for financial analysis.
One of the most famous cases of using satellite images for financial analysis is to use satellite images to count the number of cars in and out of Walmart stores' parking lots over a period of time. Financial analysts use this piece of information as a proxy of foot traffic to gauge the sales of the store during a specific period of time. This leads to the first application of using satellite images.

2.1 Monitoring Economic Activities of a Location

The Walmart parking lot example is a classic application of satellite imagery. Another example is to use satellite images to monitor container ship traffic at a major canal or a major port to identify possible supply chain obstacles. We can also use satellite images to monitor factory sites to evaluate potential production levels of a certain time. We can also monitor a mining site to track commodity production.

2.2 Tracking Crop Yield

We can use satellite images to study if a given crop, like wheat or corn, is healthy or not. From this information, financial analysts can estimate if

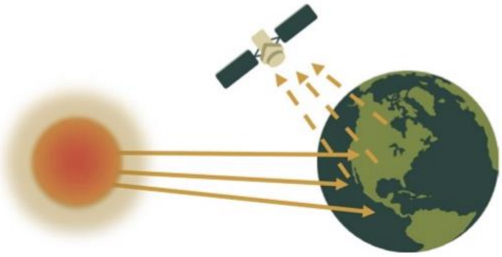
3. The Basics of Satellite Imagery

3.1 Remote Sensing

Before talking about satellite images, we need to introduce a few concepts. The first is remote sensing. **Remote sensing** is the activity of using Earth observation satellites or airplanes equipped with sensors to obtain images and other information about the Earth's surface from above.

The sensors on a satellite measure **electromagnetic radiation (EMR)**, which is first emitted by the sun. Once it reaches Earth's surface, it is then reflected back to the sensors on the satellite. Figure 1 below demonstrates the path of EMR movement.

Figure 1: Illustration of How a Satellite Measures EMR Reflected from Earth's Surface



3.2 Electromagnetic Spectrum

EMR is a type of energy that travels in waves. However, these waves are not homogeneous; they have different wavelengths. The wavelength of a wave is measured as the distance between two wave tops. If one wave has a shorter wavelength than the other waves, then this wave has a higher frequency because the wave moves up and down and then up again faster.

We group waves with similar wavelengths into a band called a **spectrum**. Figure 2 below shows how we group waves into different spectrums or bands by their wavelengths.

Figure 2: Electromagnetic Radiation Spectrum

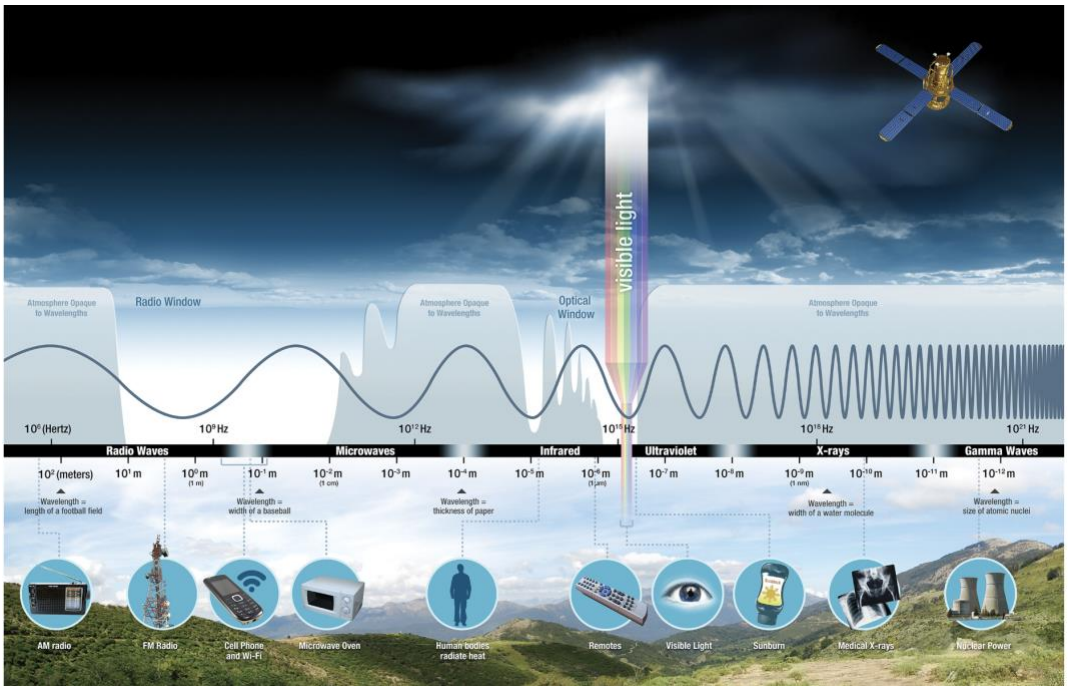


Figure 2 represents the whole spectrum of electromagnetic radiation. However, in general, Earth observation satellites don't collect data across the whole sprectrum. Satellites usually only collect information from infrared bands and bands visible to the human eye. The information from these bands provides enough data for us to investigate activities happening on Earth's surface.

All electromagnetic radiation is light, but only certain wavelengths can be detected by the human eye. This is known as the visible light spectrum. All the colors that humans can see are combinations of three primary colors: red, green, and blue (RGB). Each of these primary colors represents a band.

Photos taken from cameras are also a combination of these three primary colors. These combinations can reveal different levels of information on Earth, and they are very insightful for researchers. We will only use a few combinations in this lesson. For those who are interested in learning more about combinations, there is plenty of information on the internet.
Satellites that collect information from multiple bands are called multispectral satellites. Each satellite can construct its own band system. For example, a popular satellite for which the public can access satellite images for free is the United States' Landsat satellites. Figure 3 below shows the band structure for the U.S. Landsat 8 satellite.

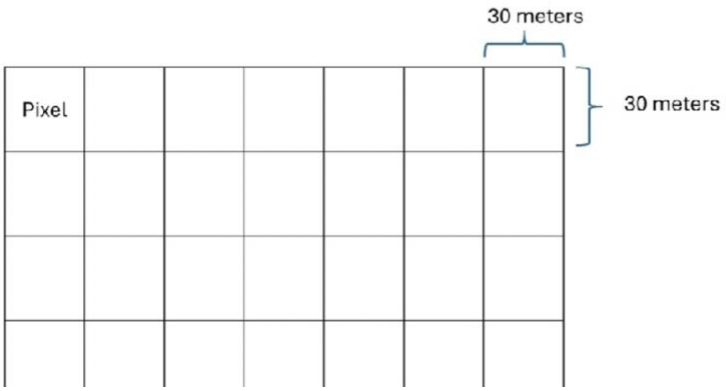
Figure 3: Band Structure for U.S. Landsat 8 Satellite

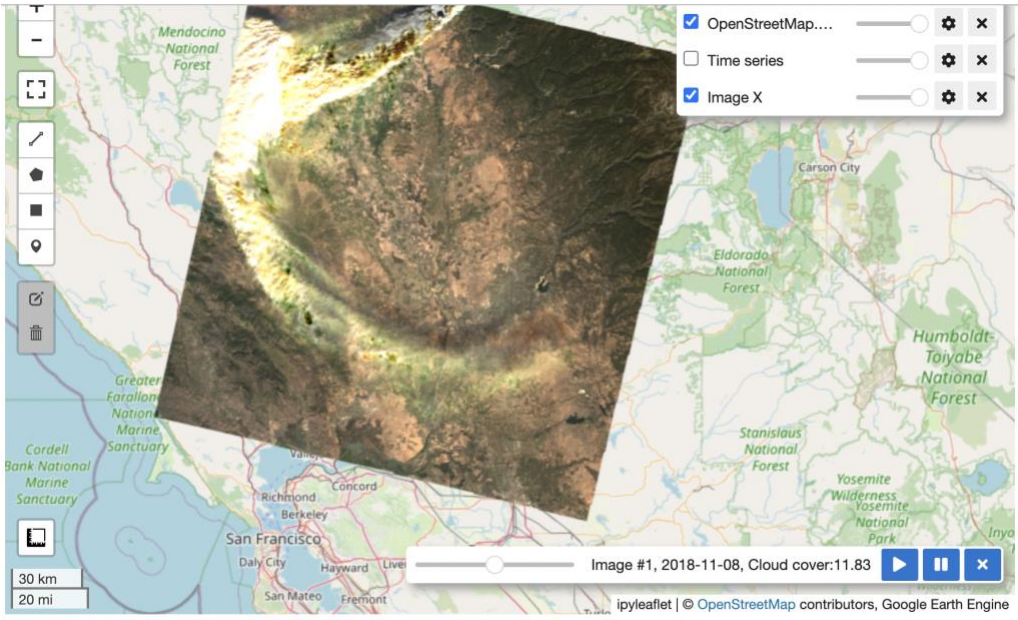
	Bands	Wavelength (micrometers)	Resolution (meters)
Landsat 8 Operational Land Imager (OLI) and Thermal Infrared Sensor	Band 1 - Coastal aerosol	0.43 - 0.45	30
	Band 2 - Blue	0.45 - 0.51	30
	Band 3 - Green	0.53 - 0.59	30
	Band 4 - Red	0.64 - 0.67	30
	Band 5 - Near Infrared (NIR)	0.85 - 0.88	30
	Band 6 - SWIR 1	1.57 - 1.65	30
	Band 7 - SWIR 2	2.11 - 2.29	30

3.3 Spatial Resolution

When a satellite collects information from an observation area, it divides the area into a matrix or a grid. Each cell in the matrix is called a **pixel**. A cell is a square. If the cell is square that is 30 meters x 30 meters, then the resolution is 30 meters. The number represents the size of a pixel. Figure 5 demonstrates this concept.

Figure 5: Spatial Resolution



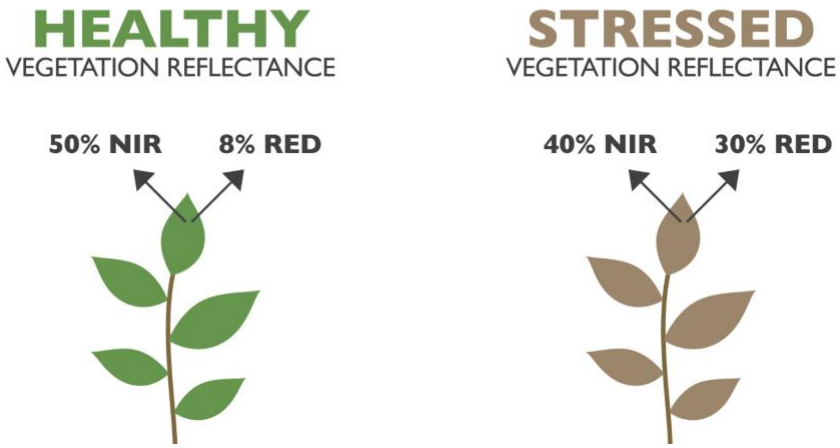


We can see that the upper corner of the middle image does show the fire. However, it is still hard to see the impact of the fire when comparing the first image to the last image. Let's introduce an NDVI index to better study the damage.

5.8 Normalized Difference Vegetation Index (NDVI)

The Normalized Difference Vegetation Index (NDVI) is an index to study plant health. This index is calculated by using reflected light from the red band and near infrared band of satellite images. The higher the index number means the plants are healthier and greener. The lower the index number means the plants are browner and less healthy. The concept is illustrated in Figure 7.

Figure 7. NDVI Illustration



$$NDVI = \frac{NIR - RED}{NIR + RED}$$

In the following section, we will turn our three images into NDVI images. Each pixel will have an NDVI number. A pixel with a high NDVI number will be painted green. A pixel with a low NDVI number will be painted red. If the NDVI number is in between, the pixel will be painted yellow.

First, let's calculate NDVI numbers for each image.

[36]:
```
#Calculate NDVI for each image
img_ndvi_1 = ee.Image(landsat_list_2.get(0)).normalizedDifference(['SR_B5', 'SR_B4'])
img_ndvi_2 = ee.Image(landsat_list_2.get(1)).normalizedDifference(['SR_B5', 'SR_B4'])
img_ndvi_3 = ee.Image(landsat_list_2.get(2)).normalizedDifference(['SR_B5', 'SR_B4'])

landsat_3 = ee.ImageCollection.fromImages([img_ndvi_1, img_ndvi_2, img_ndvi_3])
print('Total number:', landsat_3.size().getInfo())
```

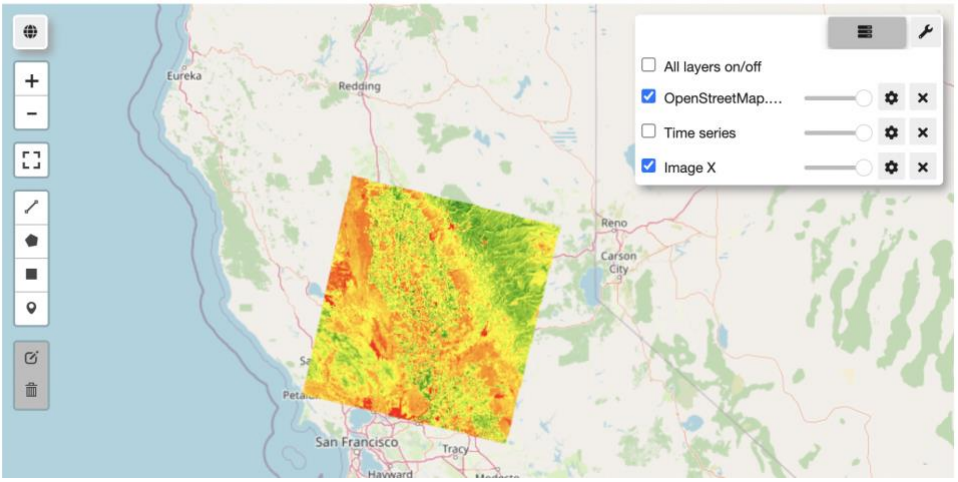
```
Total number: 3
```

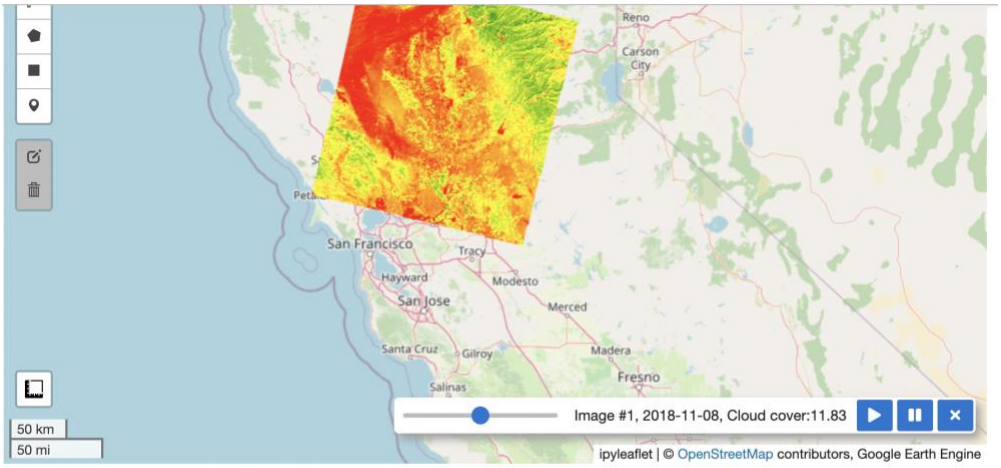
Then, we'll set up the new color palette and NDVI visualization parameters.

[37]:
```
# Create a new list
landsat_list_3 = landsat_3.toList(landsat_3.size())

# ndvi palette: red is low, green is high vegetation
palette = ['red', 'yellow', 'green']

ndvi_parameters = {'min': 0,
                   'max': 0.4,
                   'dimensions': 512,
                   'palette': palette,
                   'region': roi}
```

[38]:
```
image3 = landsat_3.toBands()

Map4 = geemap.Map()
Map4.addLayer(image3,{},"Time series", False)
Map4.setCenter(lon,lat,8)
Map4.add_time_slider(landsat_3, ndvi_parameters, labels = labels_2, time_interval=1)
Map4
```

[38]:





From the above interactive NDVI images, let's focus on the upper middle part of each image. Let's start with image #0.

In image #0, the area contains a mixture of green, yellow, and red. This image was taken before the Camp Fire. In image #1, we can see a big swath of red running across the area from right to left and going down. This image was taken during the Camp Fire. The red area is where the fire was. Now let's look at image #3, which was taken after the Camp Fire. In image #3, we can see a big red area in the upper middle part and some on the upper left side of the image. To summarize, with NDVI images, we can better identify changes in the vegetation on Earth's surface.

6. Conclusion

In this lesson, we first learned the basics of remote sensing, the method to obtain satellite images. We then went through the fundamentals of satellite images, including the electromagnetic spectrum, spatial resolution, and commonly used satellites. We then introduced some use cases for applying satellite images to finance analyses. We finished the lesson by using Google Earth Engine's Python API to pull some satellite images and conducted some analysis. We learned to use NDVI images to investigate the vegetation damage after the 2018 Camp Fire in California.