Geospatial and Time Series Data Analysis

A tutorial for Data Science
Course IFT6758

Spatial Data Analysis

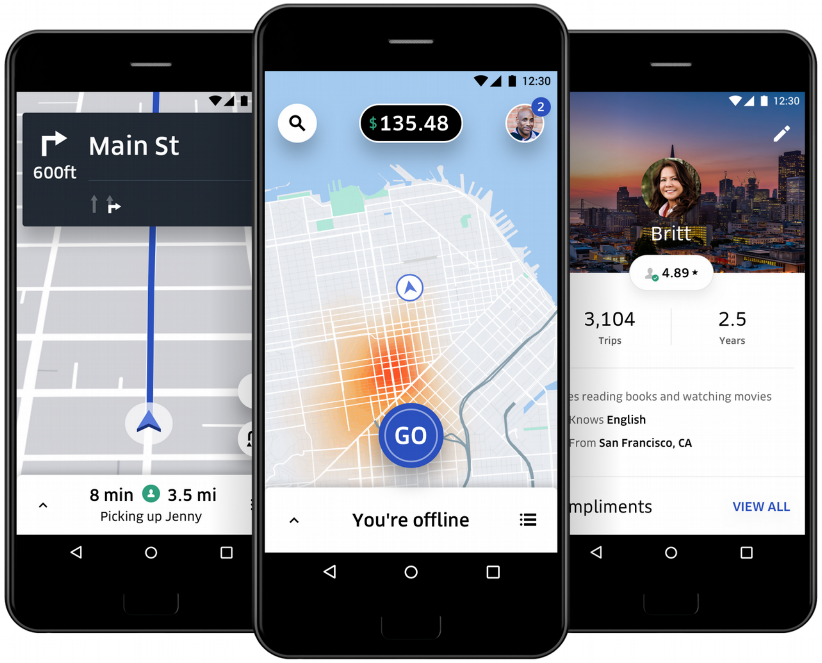
- **Spatial data:** Information about *locations* and *shapes* of objects in a geographic coordinate system

Data analysis for traffic management

- Data collection from mobile devices
- Understanding mobility patterns
- Predicting traffic flow
- Optimizing traffic signal control (traffic lights)

Spatial Data Analysis

- **Spatial data:** Information about *locations* and *shapes* of objects in a geographic coordinate system



Predicting passenger demand

- Data is collected after each trip
- Recommendations to drivers
- Pricing based on predicted demand
- Similar types of analysis are possible for public transit

Spatial Data Analysis

- **Spatial data:** Information about *locations* and *shapes* of objects in a geographic coordinate system

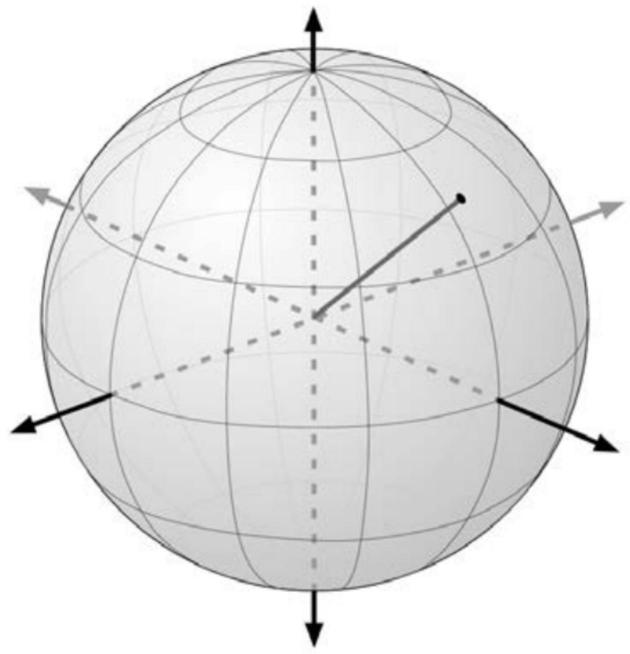
Urban planning

- Different types of data with geographical attributes (census data, traffic data, etc.)
- Long-term predictions
- Zoning (clustering)

Understanding Geospatial Data

- Location: a point on earth is specified by its *latitude* and *longitude*

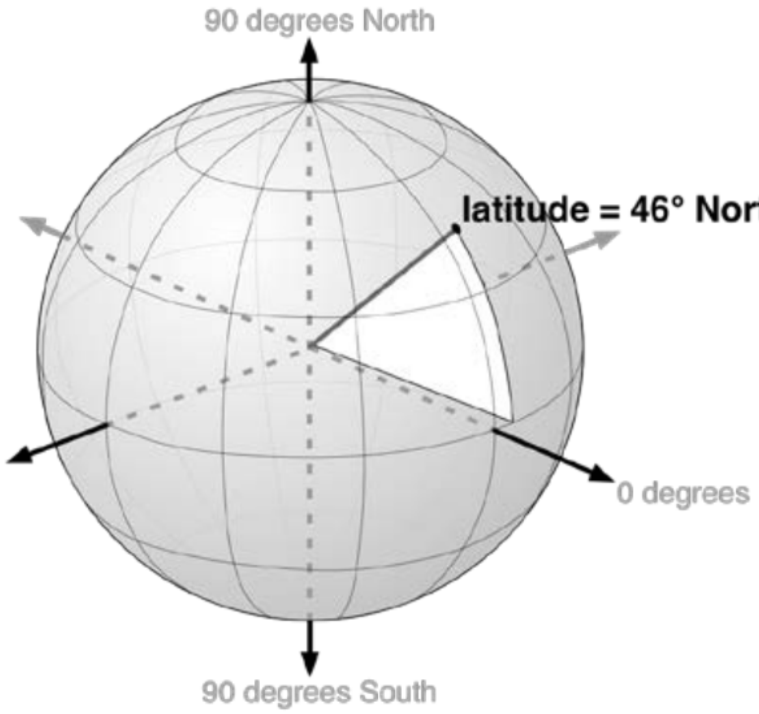
  Imagine a line segment between the center of the Earth and a location



Understanding Geospatial Data

- Location: a point on earth is specified by its *latitude* and *longitude*

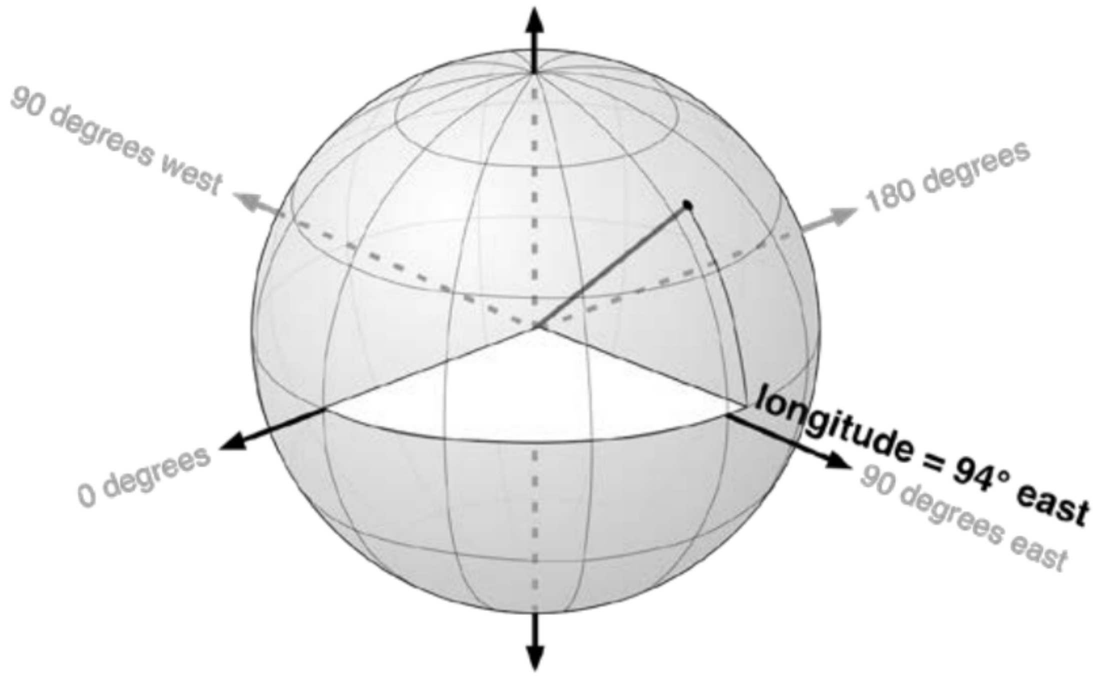
  Latitude is the angle in the north-south direction



Understanding Geospatial Data

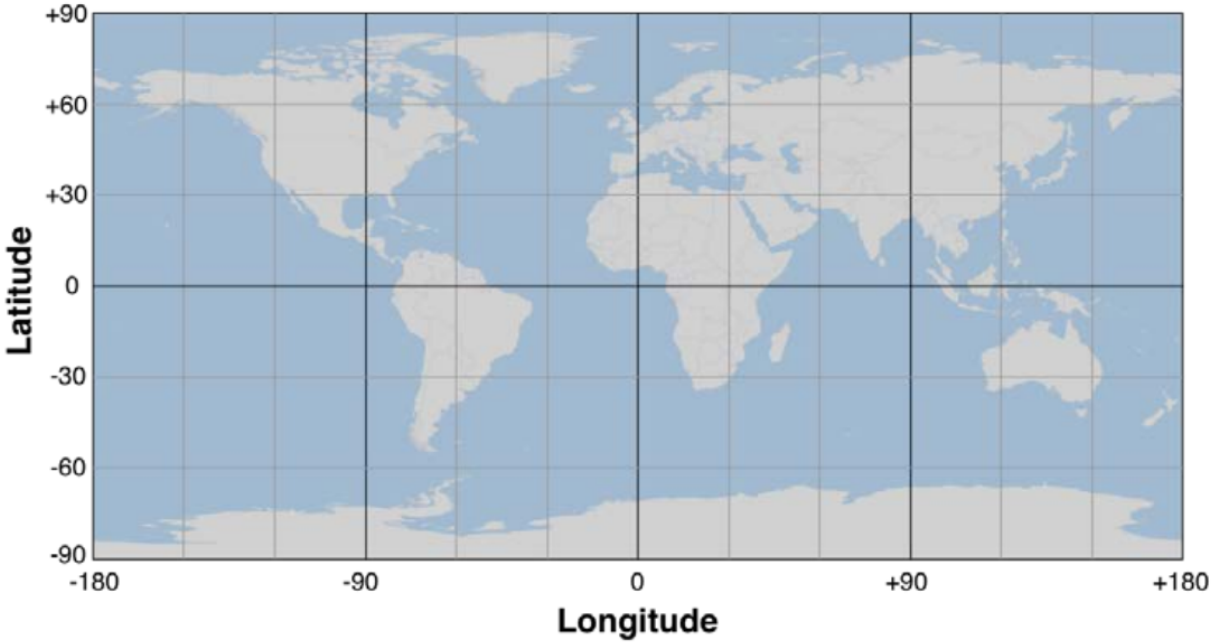
- Location: a point on earth is specified by its *latitude* and *longitude*

  Longitude is the angle in the east-west direction



Understanding Geospatial Data

- Location: a point on earth is specified by its *latitude* and *longitude*



- degrees, minutes, seconds
  $176°14'4''$
- degrees and decimal minutes
  $176°14.066'$
- decimal degrees
  $176.234436°$

Example Task

- You are given the records of taxi trips in San Francisco

Example Task

- You are given the records of taxi trips in San Francisco

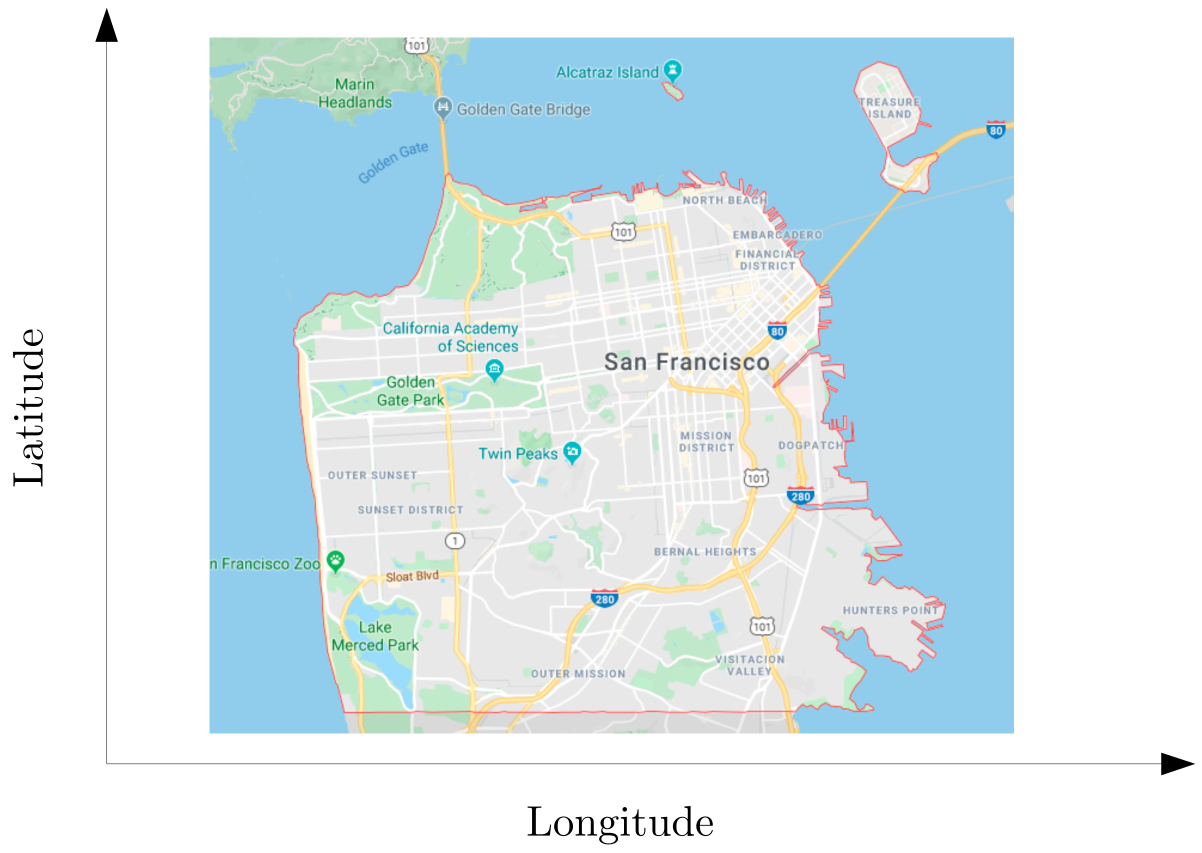
Car ID	Departure			Arrival		
	Date-Time	Latitude	Longitude	Date-Time	Latitude	Longitude
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...

- The task is to predict the passenger demand

  How many requests there will be for trips departing from zone A next Thursday at 10 AM?

Visulalizing Spatial Data

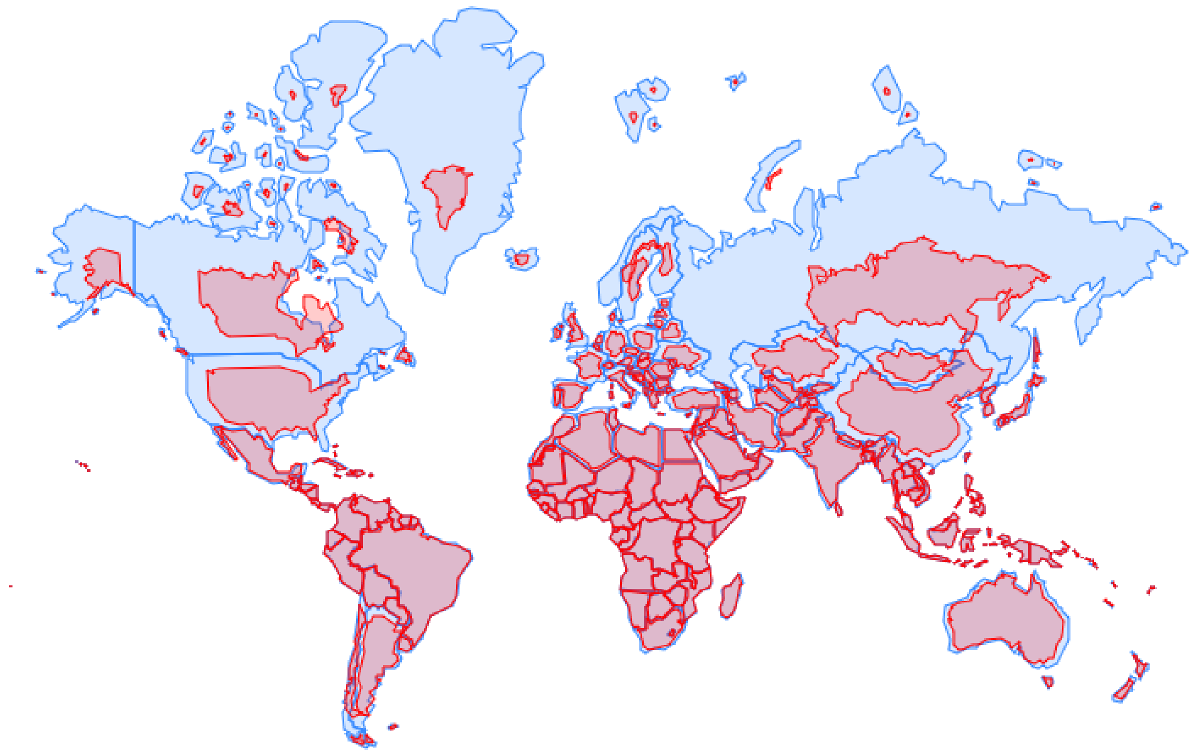
You want to visualize all points of departure and arrival at a certain time.



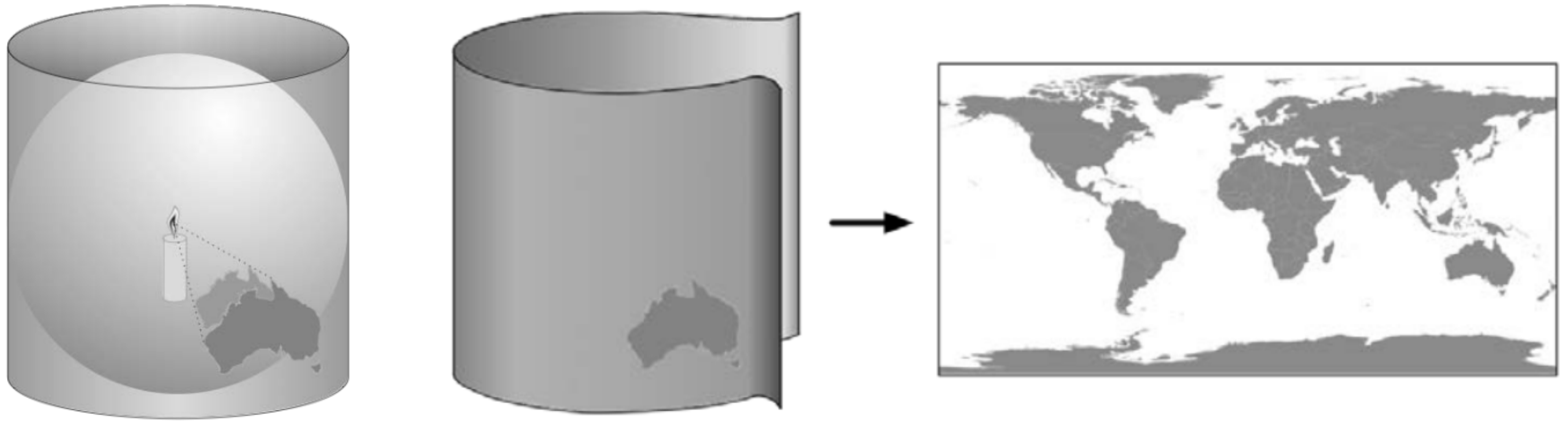
Can't we just use latitude and longitude as coordinates in a two-dimensional plane?

Map Projections

We cannot just use latitude and longitude as coordinates in a two-dimensional plane!



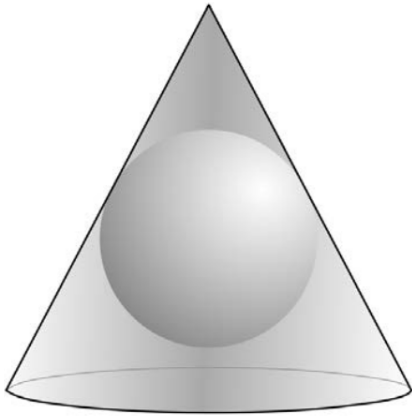
Cylindrical Projections

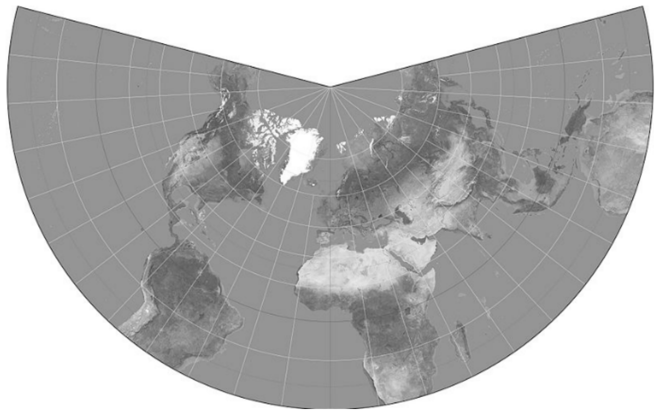


Example cylindrical projections

- Mercator projection
- Equal-area cylindrical projection
- Universal transverse Mercator projection

Conic Projections

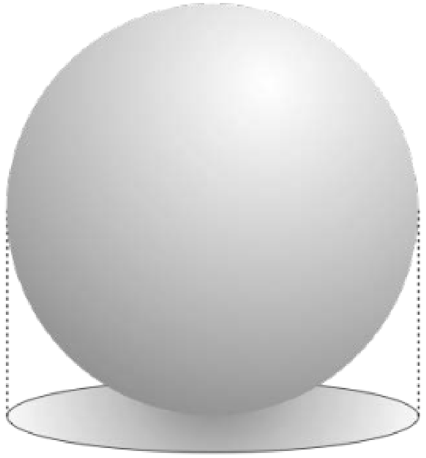


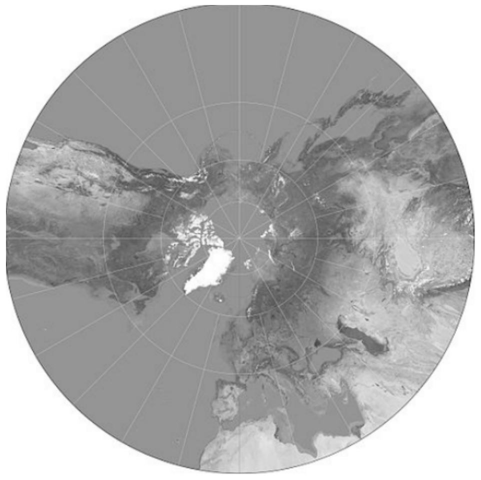


Example conic projections

- Albers equal-area projection
- Lambert conformal conic projection
- Equidistant projection

Azimuthal Projections





Example Azimuthal projections

- Gnomonic projection
- Lambert equal-area azimuthal projection
- Orthographic projection

Datums

A *datum* is a mathematical model of Earth

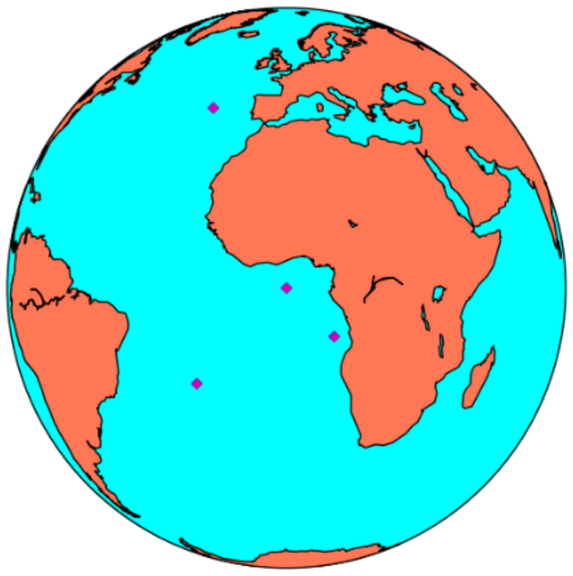
- **NAD 27** (North America)
- **NAD 83** (US, Canada, Mexico, and Central America)
- **WGS 84** (Global, used by GPS satellites)

Representing the data in the wrong datum (without converting) can lead to errors.

Visualizing Spatial Data

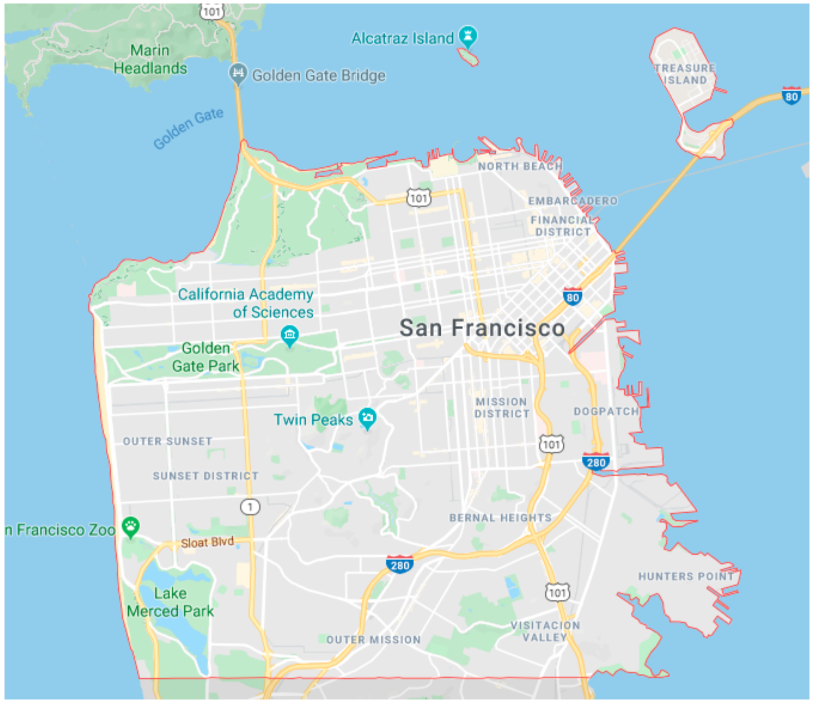
- Pick a projection and plot the base map in that projection
- Transfer the data to that projection and plot them like usual

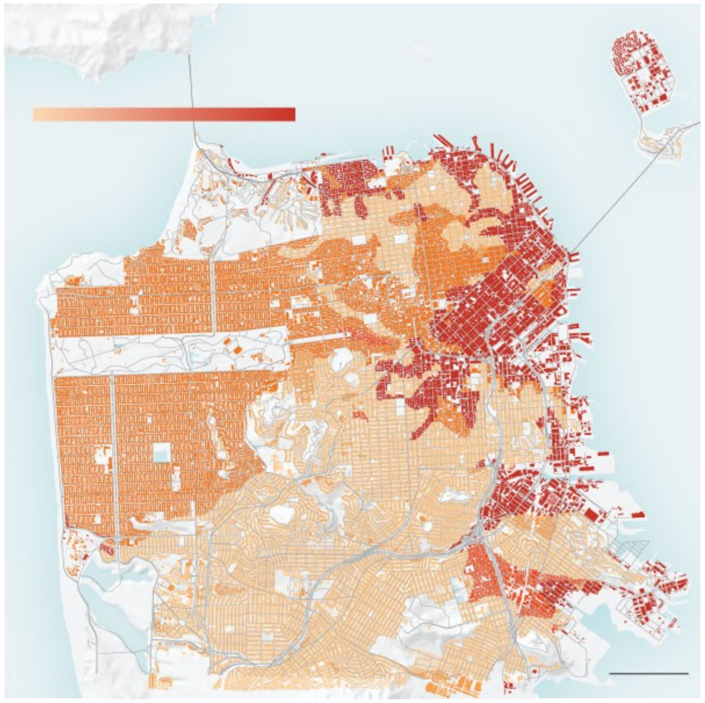
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap(projection='ortho',
              lat_0=0, lon_0=0)

map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='coral',lake_color='aqua')
map.drawcoastlines()

lons = [0, 10, -20, -20]
lats = [0, -10, 40, -20]

x, y = map(lons, lats)

map.scatter(x, y, marker='D',color='m')
```



Visualizing Spatial Data

- You want to see how the departure/arrivals are scattered in different areas of the city.
- Zones of city are not natural features. You have to load this data separately.

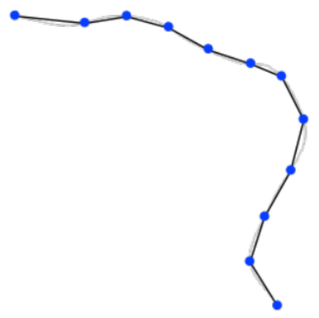


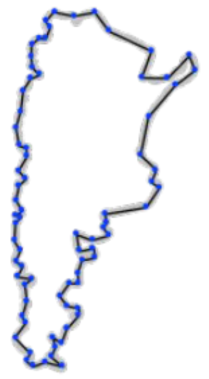


Shapes

Geospatial data often represents shapes in the form of points, paths and surfaces

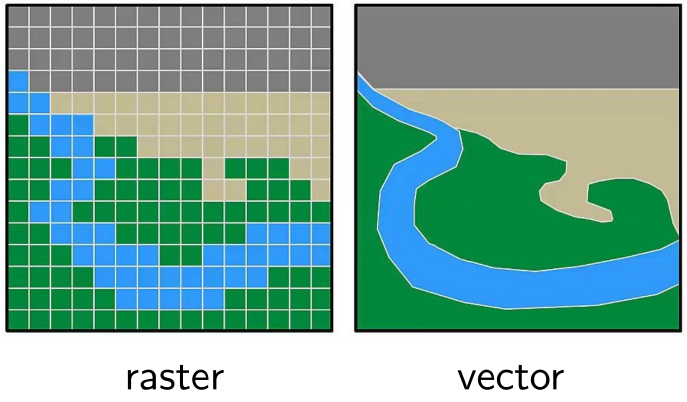
- A **Point** is a coordinate
- A **LineString** consists of a series of segments
- A **Polygon** is a closed LineString


LineString


Polygon

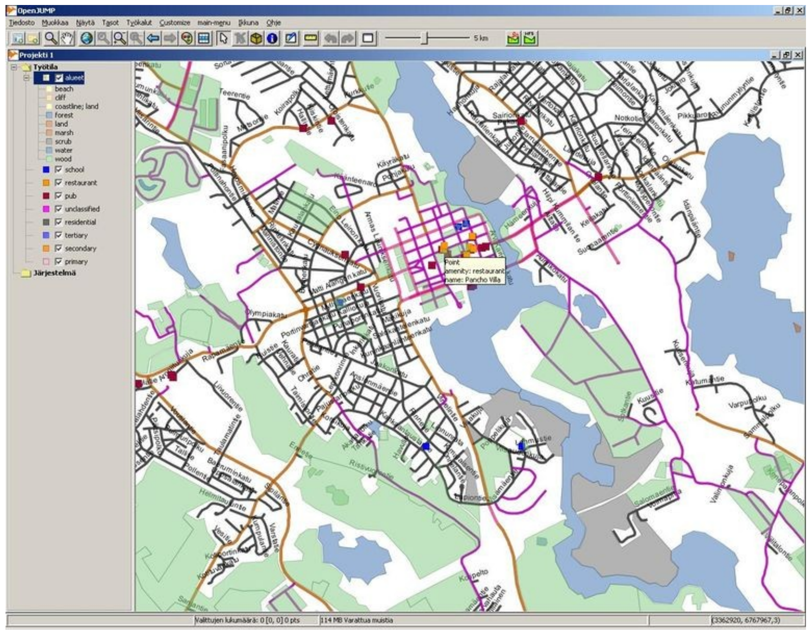
GIS data formats

- GIS data files usually represent:
  - Geospatial features *(points, lines, shapes, ...)*
  - Attributes *(population, name, ...)*
  - Meta-data *(datum, projection, ...)*
  - Display information *(color, line styles, ...)*
  - ...
- Types of GIS data:
  - **Raster** format data
  - **Vector** format data



Shapefiles

- Shapefiles are one of the most common vector data formats
- A collection of multiple files
  - `.shp` (spatial information)
  - `.prj` (projection information)
  - `.dbf` (database of attributes)
  - `.shx` (index for fast access)
  - ...



GIS viewer *OpenJump*

Example Task (continued)

We want to predict the trips departing from a zone at a certain date/time.

We can use different types of features:

- Features related to time
  - Day of week, weekday/weekend
  - Time of day
  - Season
  - Holiday or not

Example Task (continued)

We want to predict the trips departing from a zone at a certain date/time.

We can use different types of features:

- Features related to location
  - ID of the departure zone
  - Population of departure zone
  - Is the departure zone a business area?
  - Is the departure zone a recreation area?

Such information can be extracted from proxies like census data

Example Task (continued)

We want to predict the trips departing from a zone at a certain date/time.

We can use different types of features:

- Features related to weather
  - temperature
  - humidity
  - percipitation
  - ...

Example Task (continued)

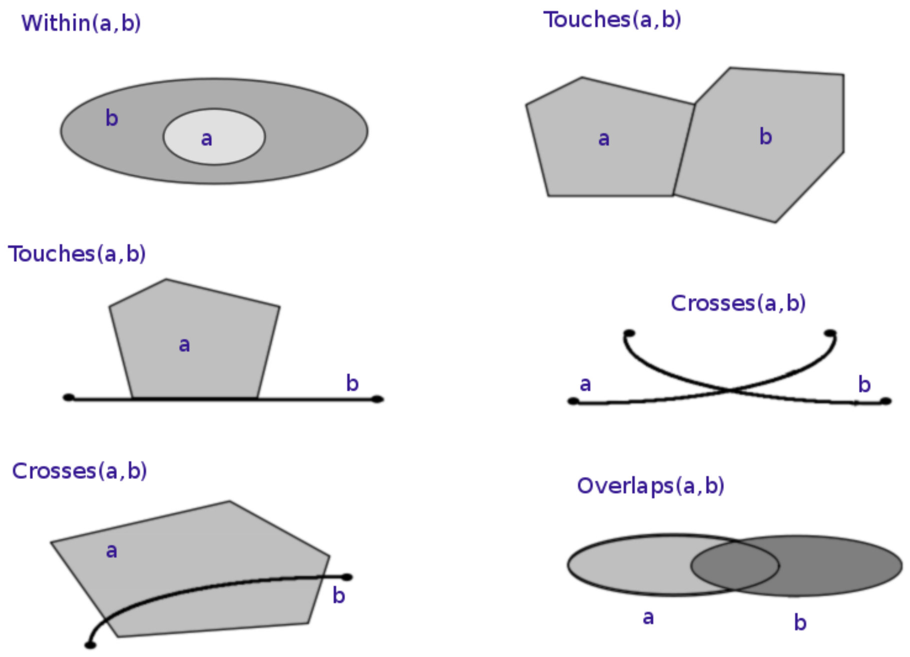
We want to predict the trips departing from a zone at a certain date/time.

- The output value is the count of trips made from a region at a certain time
- For each date/time/region, we should count the number of trips in the data
- We have the departure coordinate in our data, and the shapes of city zones from the shapefiles
- How can we check wether a point falls within a polygon?

Spatial Relations

The spatial relations are defined for any two spatial objects that can be points, lines, or polygons:

- Equals
- Disjoint (no point in common)
- Intersects (not disjoint)
- Touches (at least one boundary point in common, but no shared interior points)
- Contains
- Within (same as Contains, opposite order of argument)



```
from shapely import Point, Polygon

point = Point(0.5, 0.5)
square = Polygon([(0, 0), (0, 1), (1, 1), (1, 0)])
print(point.within(square))
```

```
True
```

Example Task (continued)

For each date/time/region, we should count the number of trips in the data

- Load the departure points as latitude/longitude pairs from trip data
- Load the city zone polygons from the city shapefiles
- Loop over departure points and polygons and see if the `within` relation holds
- This can be very slow!

Example Task (continued)

For each date/time/region, we should count the number of trips in the data

- Load the departure points into the geospatial database
- Load the city zone polygons into the goespatial database
- Perform a spatial join
- Aggregate the counts

```
import geopandas as gpd

// load the departure points ...

// load the zones
zones = gpd.read_file('zones.shp')
points_and_zones = gpd.sjoin(points, zones,
                             op='within')
```

Example Task (continued)

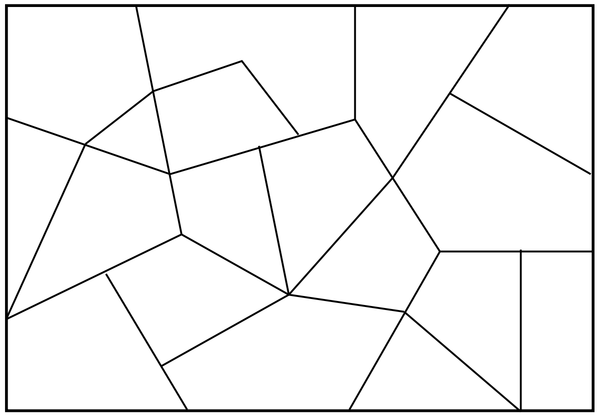
The city is partitioned into zones according to urban planning considerations.

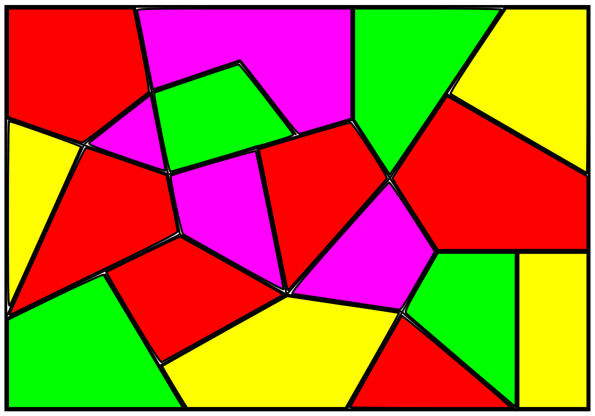
You want to define your own partitioning of the city (group the neighborhoods with similar demand patterns together)

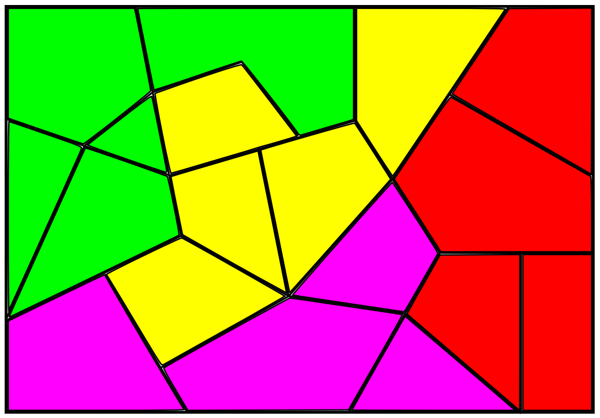
- Start with small areas as building blocks
- Create a feature vector representing trip demand patterns for all these small areas
- Cluster these small areas into larger zones

Spatially Constrained Clustering

- The clustered areas should be contiguous
- We need special algorithms that take contiguity constraints into account


areas


invalid clustering


valid clustering

Spatially Constrained Clustering

- One possible method: *local search*
  - Start with an initial grouping into contiguous clusters
  - Evaluate all possible ways of moving an area to an adjacent cluster
  - Perform the move that improves the clustering criterion the most
  - Continue until there is no improving move
- Equip the local search with meta-heuristics: *tabu search*, *simulated annealing*
- Python Implementations are available in `ClusterPy` library

Clustering Temporal Data

Temporal Data: Data that represents a state in time

Examples:

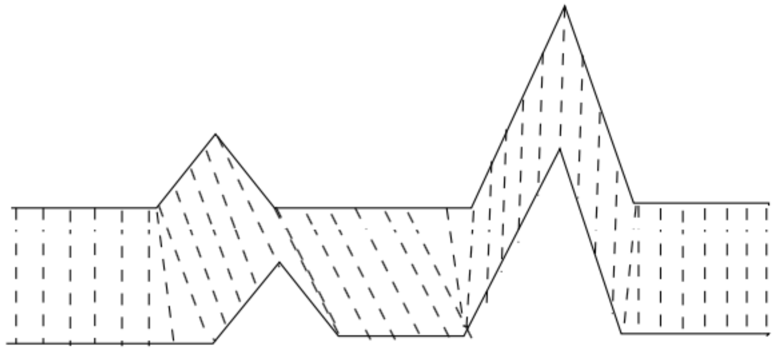
- Power consumption of a household
- Trips of a passenger
- Prices in stock market

The general idea for clustering temporal data:

- Define a *distance metric* to measure the similarity of sequences
- Use a standard clustering algorithm that accepts custom similarity measures

Dynamic Time Warping

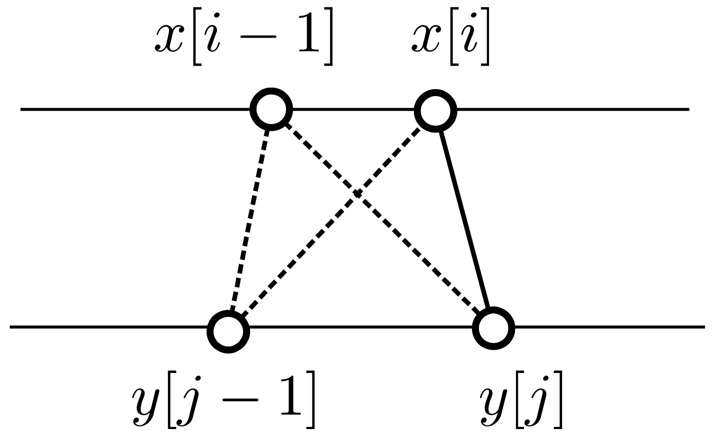
- An algorithm for measuring the distance between two temporal sequences which may vary in speed
- The two first and two last entries are matched together
- We decide how to match the other entries of the two sequences
- The matching is made such that the total distance is minimized



Dynamic Time Warping

- The matching is made such that the total distance is minimized
- Sequences x and y have n and m entries
- For each pair $x[i]$ and $y[j]$ find the best matching up to that pair
- This can be done recursively:

$$\begin{aligned} DTW[i,j] = d(x[i],y[j]) + min(&DTW[i-1,j]+ \\ &DTW[i,j-1]+ \\ &DTW[i-1,j-1]) \end{aligned}$$



Dynamic Time Warping

```
function DTWDISTANCE($x$, $y$)
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $m$ do
            $DTW[i, j] \leftarrow \infty$
        end for
    end for
    $DTW[0, 0] \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $m$ do
            $cost \leftarrow d(x[i], y[j])$
            $DTW[i, j] \leftarrow cost + \min(DTW[i-1, j],$
                                   $DTW[i, j-1],$
                                   $DTW[i-1, j-1])$
        end for
    end for
    return $DTW[n, m]$
end function
```

Summary

- Coordinate Systems
- Projections
- GIS formats (shapefiles)
- Spatial Relations
- Geospatial Databases
- Spatial Clustering
- Temporal Clustering