Table of Contents

In [1]:
```
# not all of these libraries are needed but for now let's load them to be prepared :-)
from ipyleaflet import (Map, GeoData, WidgetControl, GeoJSON, basemaps, LocalTileLayer,
 LayersControl, Icon, Marker,basemap_to_tiles, Choropleth, AntPath,
 MarkerCluster, Heatmap, SearchControl, FullScreenControl, AwesomeIcon,
 ScaleControl, MeasureControl, SplitMapControl, WMSLayer, Polygon, Choropleth)
```

In [2]:
```
# ipywidgets add user interactions to our notebook cells
# read the docs https://ipywidgets.readthedocs.io/en/latest/
from ipywidgets import Text, HTML, IntSlider, ColorPicker, jslink, Layout
from branca.colormap import linear
```

In [3]:
```
# something to look into when wanting to color areas of a city or a region (Choropleth Map
# https://blog.datawrapper.de/choroplethmaps/
#import geopandas as gpd
#import json
```

In [4]:
```
# Pandas = derived from "Python and data analysis"
import pandas as pd
import math
```

1 Reading the data as provided by the gadget (there are always two types of files - tracks and categories of spots)

*** Important ***

> first line in CVS should look like this:
> Date,Time,RawTime,Latitude,Longitude,Altitude,Sats,SatSpeed,Precision

In [33]:
```
categories=pd.read_csv('data/2020-11-06_rate_Wallenstein_Gastro_Corona.csv', sep=',')
tracks=pd.read_csv('data/2020-11-06_track_Wallenstein_Gastro_Corona.csv', sep=',')


'''categories=pd.read_csv('data/2020-10-12_rate_waste.csv', sep=',')
tracks=pd.read_csv('data/2020-10-12_track_waste.csv', sep=',')'''

print('number of track points: ', len (tracks))
tracks.tail(8)
```

```
number of track points:  341
```

Out[33]:

	Date	Time	RawTime	Latitude	Longitude	Altitude	Sats	SatSpeed	Precision
333	61120	16:17:44	15174400	48.232300	16.373714	201.2	5	1.9076	1.51
334	61120	16:17:49	15174900	48.232271	16.373638	201.5	6	3.5373	1.47
335	61120	16:17:54	15175400	48.232249	16.373592	201.9	6	0.6667	1.47
336	61120	16:17:59	15175900	48.232195	16.373600	198.9	6	2.2594	2.03
337	61120	16:18:04	15180400	48.232100	16.373603	197.0	5	2.2594	2.03
338	61120	16:18:09	15180900	48.232035	16.373612	195.8	5	2.2594	1.51
339	61120	16:18:14	15181400	48.231941	16.373633	195.8	5	5.2226	1.51
340	61120	16:18:16	15181600	48.231913	16.373650	195.8	5	4.9078	1.51

2 Aspects of data quality

In [34]:
```
print('Frequency of same time records:      ', list(tracks.groupby(tracks['RawTime']).size
print('Distribution of number of satelites: ', list(tracks.groupby(tracks['Sats']).size().
```

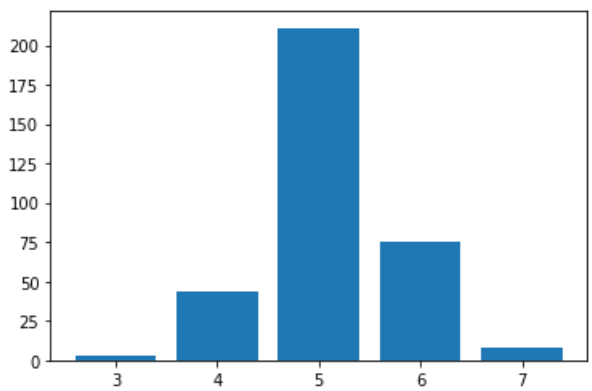
```
Frequency of same time records:       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]
Distribution of number of satelites:  [211, 75, 44, 8, 3]
```

In [35]:
```
satDistribution = tracks.groupby(tracks['Sats']).size().sort_values(ascending=False)
satNum =  list(satDistribution.keys())
satFreq = list(satDistribution)
satDens = list (zip (satNum, satFreq))

print('satellite density (sat-number, frequency): ', satDens)

```

```
satellite density (sat-number, frequency):  [(5, 211), (6, 75), (4, 44), (7, 8), (3, 3)]
```

In [36]:
```
from matplotlib import pyplot as plt
plt.bar(satNum, satFreq)
```

Out[36]:
```
<BarContainer object of 5 artists>
```



3 Distance measurement

In [37]:
```
# Haversine formula example in Python
# Author: Wayne Dyck
# https://en.wikipedia.org/wiki/Haversine_formula

def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371 # km

    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = radius * c

    return d
```

In [38]:
```
tracks_d = tracks.assign (Distance = 0)
tracks_d.head(3)
```

Out[38]:

	Date	Time	RawTime	Latitude	Longitude	Altitude	Sats	SatSpeed	Precision	Distance
0	61120	15:42:31	14423100	48.231356	16.373740	174.5	5	0.5000	2.61	0
1	61120	15:42:36	14423600	48.231421	16.373773	172.8	5	3.3706	2.61	0
2	61120	15:42:41	14424100	48.231450	16.373781	172.2	5	2.8336	2.61	0

In [39]:
```
for i in range (len(tracks_d)-1):
    j = i+1
    tracks_d.loc[j,'Distance'] = round (distance((tracks_d.Latitude[i],
                                                  tracks_d.Longitude[i]),
                                                 (tracks_d.Latitude[j],
                                                  tracks_d.Longitude[j]))*1000, 2)
```

In [40]:
```
tracks_d.tail()
```

Out[40]:

	Date	Time	RawTime	Latitude	Longitude	Altitude	Sats	SatSpeed	Precision	Distance
336	61120	16:17:59	15175900	48.232195	16.373600	198.9	6	2.2594	2.03	6.03
337	61120	16:18:04	15180400	48.232100	16.373603	197.0	5	2.2594	2.03	10.57
338	61120	16:18:09	15180900	48.232035	16.373612	195.8	5	2.2594	1.51	7.26
339	61120	16:18:14	15181400	48.231941	16.373633	195.8	5	5.2226	1.51	10.57
340	61120	16:18:16	15181600	48.231913	16.373650	195.8	5	4.9078	1.51	3.36

In [41]:
```
data_points = len(tracks_d)
print('all data points: ', data_points)
tracks_d = tracks_d[tracks_d['Distance'] > 0]

location_changes =len (tracks_d)
print('location changes: ', location_changes)
print ('reduction by: ', round (100 * (1 - location_changes / data_points), 2), ' %')
```

```
all data points:  341
location changes:  340
reduction by:  0.29  %
```

In [42]:
```
tracks_d.Distance.sum().round(2)
```

Out[42]: 2807.29

In [43]:
```
tracks_d.Distance.mean().round(2)
```

Out[43]: 8.26

In [44]:
```
tracks_d.to_csv('data/2020-11-06_track_dist_Wallenstein_Gastro.csv', index=False)
```

4 Use of categories (1 - 5)

4.0.1 Example - Gastronomy during Covid19

- cat1 - Business as usual (Green)
- cat2 - Limited opening hours (Orange)
- cat3 - Temporarily Closed (Red)
- cat4 - Closed & No Information (Grey)
- cat5 - Out of business (Black)

In [45]:
```
cat=categories.groupby(['Category']).size()
cat

```

Out[45]:
```
Category
1    11
2     2
3     3
4     8
5     1
dtype: int64
```

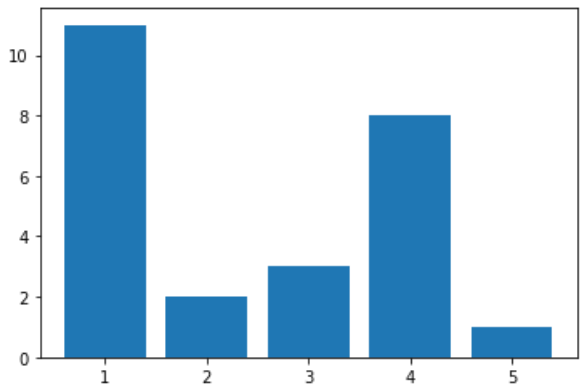
In [46]:
```
catSeries = pd.Series([0,0,0,0,0,0])
catSeries = catSeries [1:6]
```

In [47]:
```
for x in cat.index:
    catSeries [x] = cat [x]
catSeries
```

Out[47]:
```
1    11
2     2
3     3
4     8
5     1
dtype: int64
```

In [48]:
```

plt.bar(catSeries.index, catSeries)
```

Out[48]:
```
<BarContainer object of 5 artists>
```



In [49]:
```
# at this point we add some meaning to the categories
cat1 = categories[ categories['Category'] == 1]
cat2 = categories[ categories['Category'] == 2]
cat3 = categories[ categories['Category'] == 3]
cat4 = categories[ categories['Category'] == 4]
cat5 = categories[ categories['Category'] == 5]

```

In [50]:
```
# a collection of points (GPS coordinates) needs to be provided as a list of lists, this i
def location_converter (df):
    markers = df.loc[:,{'Latitude','Longitude'}] #ouput dataframe
    markers = markers.reindex(columns = ['Latitude','Longitude']) #
    markers = markers.to_records(index=False) #output array
    markers = list (markers) #output list of tuples
    markers = [list(i) for i in markers] # list of lists
    return markers

```

In [51]:
```
# calling the funnnction and checking outpu
cat1_pos = location_converter (cat1)
cat2_pos = location_converter (cat2)
cat3_pos = location_converter (cat3)
cat4_pos = location_converter (cat4)
cat5_pos = location_converter (cat5)

track_pos = location_converter (tracks)
track_pos [0:2]
```

Out[51]:
```
[[48.231356, 16.373739999999998], [48.23142100000001, 16.373773]]
```

In [52]:
```
import os
os.getcwd()
```

Out[52]:
```
'/Users/me/code/notebooks/ipyleaflet/observation-analysis'
```

5 Basic Map Parameters

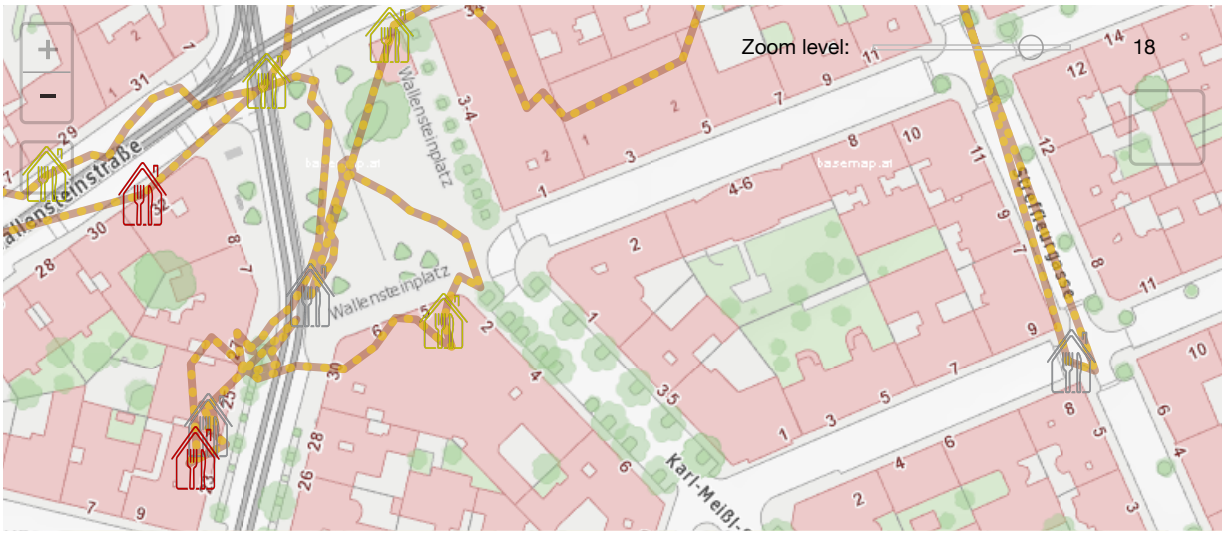
In [53]:
```
# the centre of your map should be about the starting point of your mapping tour
center = [48.231139, 16.374955]
zoom =16

# you can adjust the map size via ipwidgets *Layout* attribute
basemap = basemaps.Esri.WorldStreetMap
layout = Layout(width='100%', height='600px')

"""
alternative options for basemaps are
basemap = basemaps.Stamen.Watercolor
basemap = basemaps.Stamen.Toner
basemap = basemaps.Stamen.Terrain
basemap = basemaps.Esri.WorldStreetMap
etc

"""
```

Out[53]:
```
'\nalternative options for basemaps are \nbasemap = basemaps.Stamen.Watercolor\nbasemap = ba
semaps.Stamen.Toner\nbasemap = basemaps.Stamen.Terrain\nbasemap = basemaps.Esri.WorldStreetM
ap \netc  \n\n'
```

6 Basic Map

In [54]:
```
if m: m.clear_layers()
```

In [55]:
```
m = Map(center=center, zoom=zoom, basemap = basemap, layout=layout)

# add user interaction / user information such as the scale of a map
zoom_slider = IntSlider(description='Zoom level:', min=10, max=20, value=16)
jslink((zoom_slider, 'value'), (m, 'zoom'))

widget_control1 = WidgetControl(widget=zoom_slider, position='topright')
m.add_control(widget_control1)
m.add_control(FullScreenControl())
m.add_control(ScaleControl(position='bottomleft', imperial = False))
m.add_control(LayersControl(position='topright'))

# this adds a nice feature to meassure the length of a path or a polygone area in square m
measure = MeasureControl(
    position='bottomleft',
    active_color = 'orange',
    primary_length_unit = 'meters',
    primary_area_unit = 'sqmeters',
    completed_color = 'blue'
)
m.add_control(measure)
display (m)
```



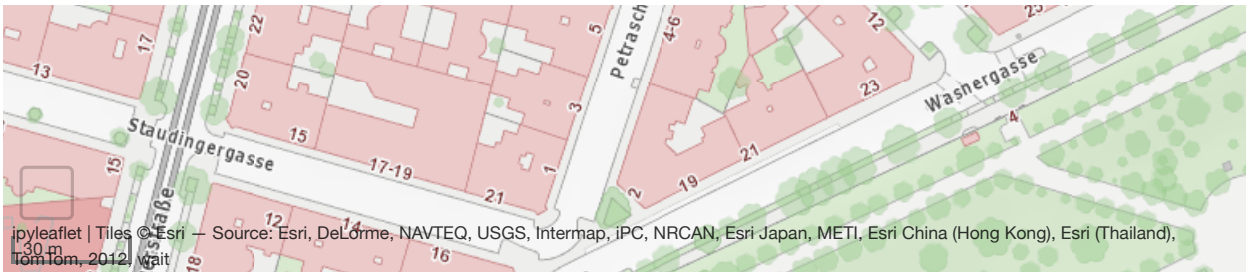
7 Adding categories 1-5 to the map

In [56]:
```
# Mapping categories with Icons


#for jupyter notebook
icon_1 = Icon(icon_url= 'icons/gastro/gastro_green.png', icon_size=[30, 35])
icon_2 = Icon(icon_url= 'icons/gastro/gastro_orange.png', icon_size=[30, 35])
icon_3 = Icon(icon_url= 'icons/gastro/gastro_red.png', icon_size=[30, 40])
icon_4 = Icon(icon_url= 'icons/gastro/gastro_grey.png', icon_size=[30, 40])
icon_5 = Icon(icon_url= 'icons/gastro/gastro_black.png', icon_size=[30, 40])

'''
#accessing icons in jupyter labs, requires a different paths
trash_icon = Icon(icon_url='files/'+os.getcwd().split('/')[-1]+'/icons/trashbin.png', icon
tree_icon = Icon(icon_url='files/'+os.getcwd().split('/')[-1]+'/icons/tree.png', icon_size
'''
#external icons
#icon = Icon(icon_url='https://leafletjs.com/examples/custom-icons/leaf-green.png', icon_s
# Just to see what os.path returns?
'''
try:
    print("File exist: ", os.path.isfile(foot_icon))
except:
    print(foot_icon)
'''
```

Out[56]:
```
'\ntry: \n    print("File exist: ", os.path.isfile(foot_icon))\nexcept:\n    print(foot_ico
n)\n'
```

In [57]:

```
#trash_icon = AwesomeIcon (name='trash', marker_color='white', icon_color='black', spin=Tr
#trash_icon = Icon(icon_url='https://leafletjs.com/examples/custom-icons/leaf-red.png', ic

for i in range(len(cat1_pos)):
    cat1_marker  = Marker(location=cat1_pos[i], icon = icon_1)
    m.add_layer(cat1_marker);

for i in range(len(cat2_pos)):
    cat2_marker  = Marker(location=cat2_pos[i], icon = icon_2)
    m.add_layer(cat2_marker);

for i in range(len(cat3_pos)):
    cat3_marker  = Marker(location=cat3_pos[i], icon = icon_3)
    m.add_layer(cat3_marker);

for i in range(len(cat4_pos)):
    cat4_marker  = Marker(location=cat4_pos[i], icon = icon_4)
    m.add_layer(cat4_marker);

for i in range(len(cat5_pos)):
    cat5_marker  = Marker(location=cat5_pos[i], icon = icon_5)
    m.add_layer(cat5_marker);



 # creating the path of the mapping exercise

'''
# https://www.rapidtables.com/web/color/Gold_Color.html
# grey antz path
    color='#D3D3D3',
    pulse_color='#A9A9A9',

# golden antv path
    color='#FAFAD2',
    pulse_color='#FFD700',
'''

ant_path = AntPath (
    locations=track_pos,
    dash_array=[1, 10],
    delay=2000,
    color='#A0522D',
    pulse_color='#FFD700',
    name='Trail')

m.add_layer(ant_path)

display (m)
```

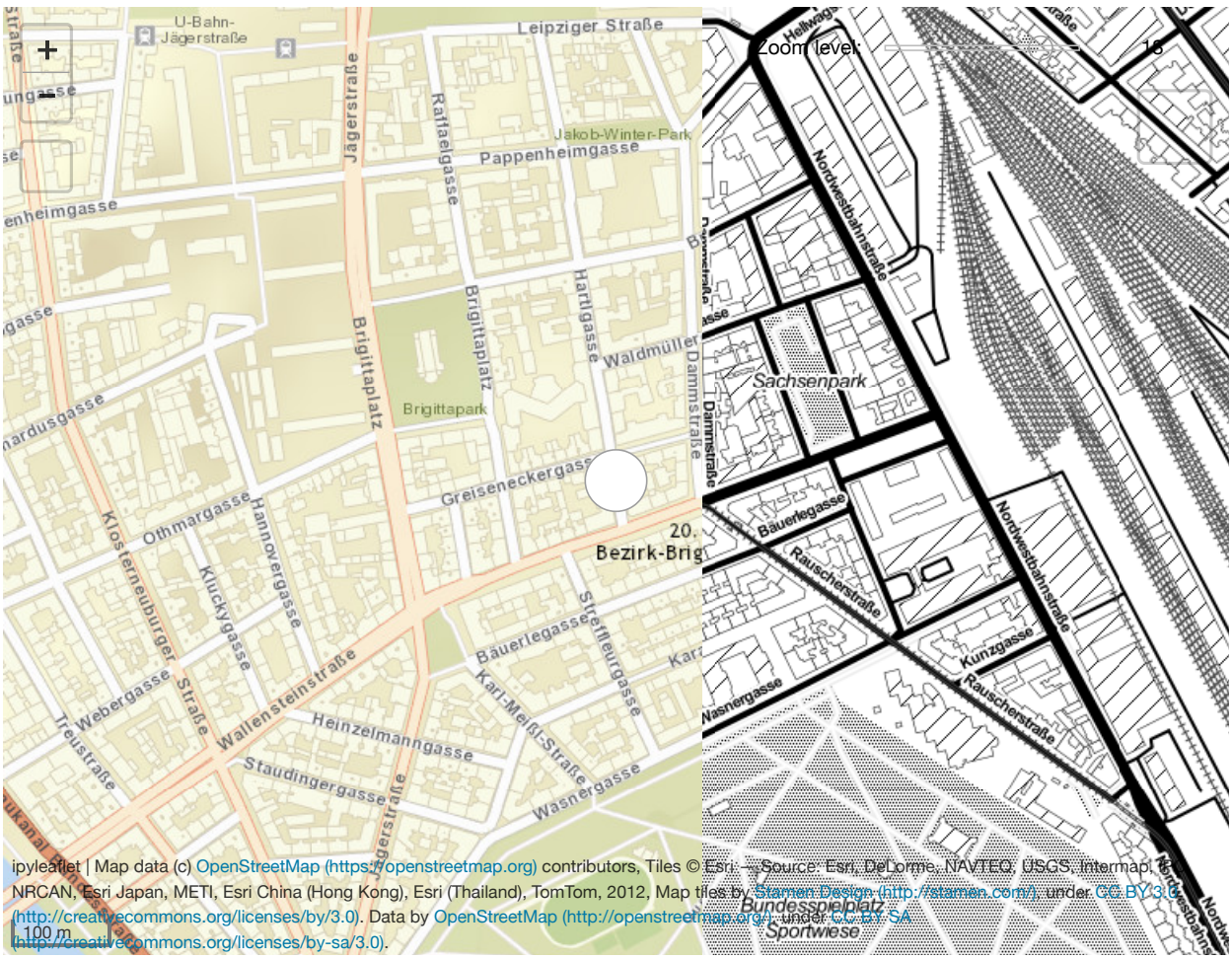


In [ ]:

```
# creating the path of the mapping exercise

ant_path = AntPath (
    locations=track_pos,
    dash_array=[1, 10],
    delay=2000,
    color='#7590ba',
    pulse_color='#3f6fba',
    name='Trail')

m.add_layer(ant_path)

display (m)
```

8 Create a Split Map

In [58]:
```
basemap = basemaps.Esri.WorldStreetMap
split_map = Map(center=center, zoom=zoom, layout=layout)

# create right and left layers
left_layer  = basemap_to_tiles(basemap=basemap)
right_layer = basemap_to_tiles(basemap=basemaps.Stamen.Toner)

# create split control
control = SplitMapControl(left_layer=left_layer, right_layer=right_layer)

#add control to map
split_map.add_control(control)

# display map

zoom_slider = IntSlider(description='Zoom level:', min=10, max=20, value=16)
jslink((zoom_slider, 'value'), (m, 'zoom'))

widget_control1 = WidgetControl(widget=zoom_slider, position='topright')
split_map.add_control(widget_control1)
split_map.add_control(FullScreenControl())
split_map.add_control(ScaleControl(position='bottomleft', imperial = False))
split_map.add_control(LayersControl(position='topright'))

display(split_map)
```

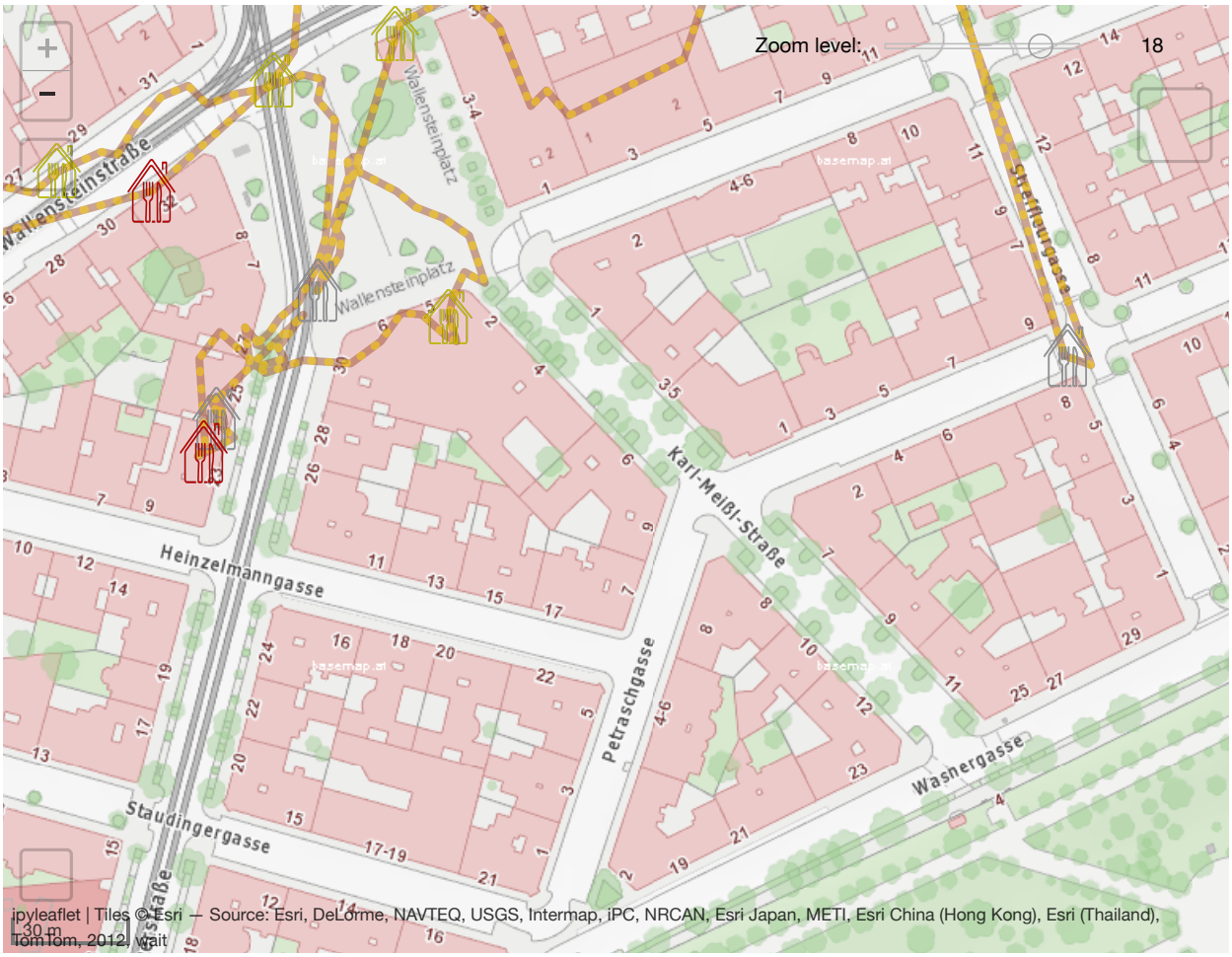


9 The resulting HTML file should be visible in any browser (however, icons will be missing - fixible)

In [59]:
```
m.save('my_map.html', title='My Map')
```

10 Integrating a different basemap with more details

In [60]:
```
from ipyleaflet import Map, WMSLayer, basemaps
#wmts = "http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png"

wms = WMSLayer(
    url='http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png',
    format='image/png',
    transparent=True,
    attribution='wait'
)

m.add_layer(wms)

m
```



11 For later: Experimenting with coloring regions or neighborhoods

In [ ]:
```
import csv
from collections import defaultdict
```

In [ ]:
```
#the syntax is: mydict[key] = "value"
#mydict ["iphone 5S"] = 2013

def parse_csv_by_field(filename, fieldnames):
    print(fieldnames)
    d = defaultdict(list)
    with open(filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile, fieldnames)
        next(reader)  # remove header
        for row in reader:
            d[row ['bundesland']] = int (row ['measurement'])
    return dict(d)


area_data = parse_csv_by_field('data/area_data.csv', ['bundesland','measurement'])
area_data


```

In [ ]:
```
m.clear_layers()
```

```
geo_json_borders

{'type': 'FeatureCollection',
 'name': 'gemeinden_999_geo',
 'crs': {'type': 'name',
  'properties': {'name': 'urn:ogc:def:crs:OGC:1.3:CRS84'}},
 'features': [{'type': 'Feature',
   'properties': {'name': 'Pöttsching', 'iso': '10609'},
   'geometry': {'type': 'MultiPolygon',
    'coordinates': [[[[16.404354111718263, 47.79918128500937],
       [16.400857594414486, 47.79178318259396],
       [16.370098559225617, 47.75647909430695],
       [16.36178609891293, 47.750404442983026],
       [16.337313248332276, 47.775956948979676],
```

In [ ]:
```
import geopandas as gpd
import json
states = gpd.read_file('geojson/laender.json')
print(states.head())
```

```
borders1 = 'geojson/bezirke_vienna.json'
borders2 = 'geojson/gemeinden_999_geo.json'
borders3 = 'geojson/laender.json'

with open(borders3) as f:
    geo_json_borders = json.load(f)

wms = WMSLayer(

url='http://maps.wien.gv.at/basemap/geolandbasemap/normal/google3857/{z}/{y}/{x}.png',
    format='image/png',
    transparent=True,
    attribution='wait'
)


m = Map(center=center, zoom=12, layout=Layout(width='100%', height='600px'))

'''
border_layer = GeoJSON(data=geo_json_borders,
                                  style = {'color': 'red',
                                           'opacity': 1.0,
                                           'weight': 2.9,
                                           'fill': 'blue',
                                           'fillOpacity': 0.2})
'''

layer = Choropleth(
    geo_data=geo_json_borders,
    choro_data=area_data,
    key_on= 'iso',
    colormap=linear.YlOrRd_04,
    border_color='black',
```

```
    style={'fillOpacity': 0.8, 'dashArray': '5, 5'})


m.add_layer(wms)
m.add_layer(border_layer)

m
```

In [ ]:
```
geo_json_borders ['features'] [0] ['properties'] ['name']
```

In [ ]:
```

```