Viewing categorical AIS tracks

To help understand the AIS data, it can be useful to color-code each location by a category, because the behavior of vessels in a given category might differ from vessels in other categories. Here we'll show how to color code by vessel type using Datashader.

In [1]:
```
import pandas as pd
import numpy as np
import panel as pn
import colorcet as cc
import datashader as ds
import holoviews as hv
from holoviews.util.transform import lon_lat_to_easting_northing as ll2en
from holoviews.operation.datashader import rasterize, datashade, dynspread
hv.extension('bokeh')
```

In [2]:
```
vessel_types=pd.read_csv("metadata/AIS_categories.csv")
vessel_types.head(40).tail()
```

Out[2]:

	num	desc	category	category_desc
35	35	Military ops	6	Military
36	36	Sailing	7	Sailing
37	37	Pleasure Craft	8	Pleasure
38	38	Reserved	0	Unknown
39	39	Reserved	0	Unknown

For plotting, we'll expand the integer values to string labels using online lists of 100+ AIS Vessel Types (https://api.vtexplorer.com/docs/ref-aistypes.html), and further collapse into a smaller number of vessel categories:

In [3]:
```
def vessel_category(val):
    i = int(val)
    cat = int(vessel_types.iloc[i].category) if i in vessel_types.index else 0
    return cat if cat in [0, 2,3,19,12,18] else 21 # limit to most common types

def category_desc(val):
    return vessel_types[vessel_types.category==val].iloc[0].category_desc
```

In [4]:
```
groups = {vessel_category(i):category_desc(vessel_category(i)) for i in vessel_types.num.unique()}
groups
```

Out[4]:
```
{0: 'Unknown',
 21: 'Other',
 2: 'Fishing',
 3: 'Towing',
 12: 'Tug',
 18: 'Passenger',
 19: 'Cargo'}
```

Load AIS pings and Vessel information

In [5]:
```
%%time
basedir = './data/vessel data/Cleaned AIS/Zone10_2014_01/'
broadcast = pd.read_csv(basedir+'Broadcast.csv', parse_dates=[1])
vessel_info =  pd.read_csv(basedir+'Vessel.csv')
vessel_info['vessel_type']= vessel_info['vessel_type'].fillna(0).astype(int) # NaN values are not available (0)
broadcast.head()
```

```
CPU times: user 15 s, sys: 1.81 s, total: 16.8 s
Wall time: 16.8 s
```

Out[5]:

	mmsi_id	date_time	lat	lon	speed_over_ground	course_over_ground	voyage_id	heading	status
0	366025993	2013-12-31 23:57:44	47.581332	-122.361145	0.0	39.599998	1	511	0
1	367160890	2013-12-31 23:57:44	45.835737	-123.990592	6.7	355.399990	2	359	15
2	366490600	2013-12-31 23:57:44	47.631067	-122.382117	0.0	192.100010	3	180	7
3	338000406	2013-12-31 23:57:44	48.123443	-123.444115	0.0	14.200000	4	511	0
4	367840001	2013-12-31 23:57:44	48.121267	-122.726412	11.4	55.400002	5	57	0

Assign broader categories

For each MMSID, looks up the broad category and stores it in a new column `category` .

In [6]:
```
vessel_mapping = {k:v for k,v in zip(vessel_info['mmsi_id'],
                                     vessel_info['vessel_type'].apply(vessel_category))}
categories = broadcast['mmsi_id'].apply(lambda x: vessel_mapping.get(x, 0))
broadcast['category'] = categories
```

Define color key and legend

In [7]:
```
def rgb_to_hex(rgb):
    return '#%02x%02x%02x' % rgb

color_key = {list(groups.keys())[ind]:tuple(int(el*255.) for el in val) for ind,val in
             enumerate(cc.glasbey_bw_minc_20_minl_30[:(len(groups))][::-1])}
```

In [8]:
```
color_names = {groups[k]:rgb_to_hex(v) for k,v in color_key.items()}
color_points = hv.NdOverlay({k: hv.Points([0,0], label=str(k)).opts(color=v, size=0)
                             for k, v in color_names.items()})
```

Project into Web Mercator for plotting:

In [9]:
```
%%time
broadcast.loc[:,'x'], broadcast.loc[:,'y'] = ll2en(broadcast.lon,broadcast.lat)
```

```
CPU times: user 2.32 s, sys: 1.43 s, total: 3.75 s
Wall time: 3.75 s
```

Datashaded, categorical AIS plot (Zone 10)

We can now plot the data colored by category, with a color key.

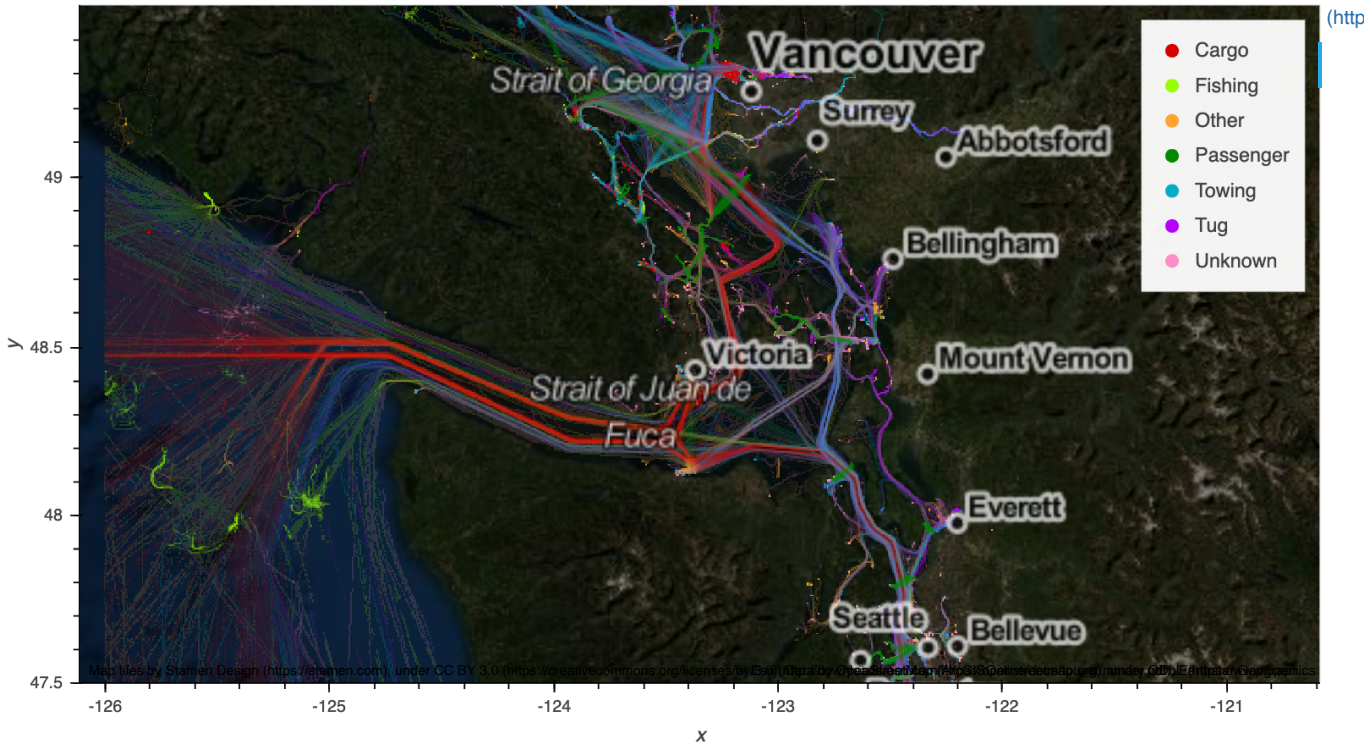
In [10]:
```
x_range, y_range = ll2en([-126,-120.7], [47.5,49.5])
bounds = dict(x=tuple(x_range), y=tuple(y_range))

points = hv.Points(broadcast, ['x', 'y'], ['category']).redim.range(**bounds)
points = dynspread(datashade(points, color_key=color_key, aggregator=ds.count_cat('category')))

tiles  = hv.element.tiles.ESRI().opts(alpha=0.5, bgcolor="black", width=900, height=500)
labels = hv.element.tiles.StamenLabels().opts(alpha=0.7, level='glyph')

tiles * points * labels * color_points
```

Out[10]:



Clearly, the ship's behavior is highly dependent on category, with very different patterns of motion between these categories (and presumably the other categories not shown). E.g. passenger vessels tend to travel *across* waterways, while towing and cargo vessels travel *along* them. Fishing vessels, as one would expect, travel out to open water and then cover a wide area around their initial destination. Zooming and panning (using the Bokeh (https://docs.bokeh.org/en/latest/docs/user_guide/tools.html) tools at the right) reveal other patterns at different locations and scales.

Selecting specific voyages

To help understand how individual tracks relate to others, we can use the x,y location of a tap to query the dataset for a set of voyages that cross that region, then highlight them compared to the main plot.

First, we will create a spatially indexed dataframe to allow spatial searching using SpatialPandas (https://github.com/holoviz/spatialpandas), which may take some time for large datasets.

In [11]:
```
from spatialpandas.geometry import PointArray
from spatialpandas import GeoDataFrame
```

In [12]:
```
%%time
sdf = GeoDataFrame({'geometry':PointArray((broadcast.lon, broadcast.lat)),
                    'x':broadcast.x, 'y':broadcast.y,
                    'date_time': broadcast.date_time,
                    'mmsi_id':broadcast.mmsi_id})
```

```
CPU times: user 271 ms, sys: 368 ms, total: 639 ms
Wall time: 638 ms
```

Next, let's make a function that returns a connected set of ping locations, given an x,y coordinate:

In [13]:
```
def highlight_tracks(x,y, delta = 0.02, max_vessels=1):
    path_data = []
    if None not in [x,y]:
        lon, lat = hv.util.transform.easting_northing_to_lon_lat(x,y)
        selection = sdf.cx[lon-delta:lon+delta, lat-delta:lat+delta]

        if len(selection) > 0:
            marked_mmsids = list(selection['mmsi_id'].unique()[:max_vessels])
            for mmsid in marked_mmsids:
                pathdf = sdf[sdf['mmsi_id']==mmsid]
                pathdf.sort_values(by='date_time')
                coords = list(zip(pathdf['x'], pathdf['y']))
                path_data.append(coords)
    return hv.Path(path_data).opts(color='white')
```

In [14]:
```
points = hv.Points(broadcast, ['x', 'y'], ['category']).redim.range(**bounds)
points = dynspread(datashade(points, color_key=color_key, aggregator=ds.count_cat('category')))
track  = hv.DynamicMap(highlight_tracks, streams=[hv.streams.Tap()])

#tiles * points * track
```

We could view the result above by uncommenting the last line, but let's just go ahead and make a little app so that we can let the user decide whether to have labels visible:

In [15]:
```
def labels(enable=True):
    return hv.element.tiles.StamenLabels().opts(level='glyph', alpha=0.7 if enable else 0)

show_labels = pn.widgets.Checkbox(name="Show labels", value=True)
overlay = tiles * points * track * hv.DynamicMap(pn.bind(labels, enable=show_labels)) * color_points

pn.Column("# Categorical plot of AIS data by type",
          "Zoom or pan to explore the data, then click to select "
          "and highlight connected vessel tracks in a region. ",
          "You may need to zoom in before a track is selectable.",
          show_labels, overlay).servable()
```

Out[15]:

Categorical plot of AIS data by type

Zoom or pan to explore the data, then click to select and highlight connected vessel tracks in a region.

You may need to zoom in before a track is selectable.

Show labels