In [1]:
```
import re
import os
import warnings
import pandas as pd
import panel as pn
import numpy as np
import datetime as dt
import param
import holoviews as hv
from colorcet import fire
from panel.template import DarkTheme
from io import StringIO
from holoviews.operation.datashader import rasterize
from bokeh.models.tools import HoverTool
from scripts.sathelpers import SatelliteDataStore
hv.extension('bokeh')
```

In [2]:
```
# Filter warnings in hit intersection code
warnings.filterwarnings('ignore', category=RuntimeWarning)
```

Set up some CSS used later

In [3]:
```
css = """
.bk.bk-data-table {
  color: black;
}
.slick-header-columns {
  color: white;
  font-weight: bold;
}
"""
pn.config.raw_css.append(css)
```

In [4]:
```
# Set some configuration variables
# In general, these should be explicit paths with no variables or homedir (
AIS_DIR = "data/vessel data/Cleaned AIS"
SAT_DIR = "data/satellite data/index_active"

if not os.path.isdir(AIS_DIR) or not os.path.isdir(SAT_DIR):
    raise IOError("Invalid source data directory")
```

Step 0. Configure the input parameters

In [5]:
```
# Based on the year of interest, also define the AIS file to look at
AIS_FILENAME = "ais_2015.h5"
```

Step 1. Load the satellite data

In [6]:
```
satdata = SatelliteDataStore(SAT_DIR)
```

In [7]:
```
df = pd.read_csv("./metadata/UCS-Satellite-Database-8-1-2020.txt", sep='\t'
df = df.dropna(axis='columns',how='all')
df.head(3)
```

Out[7]:

	Name of Satellite, Alternate Names	Current Official Name of Satellite	Country/Org of UN Registry	Country of Operator/Owner	Operator/Owner	Users	Purpose	
0	1HOPSAT-TD (1st-generation High Optical Perfor...	1HOPSAT	NR (3/20)	USA	Hera Systems	Commercial	Earth Observation	[
1	3Cat-1	3Cat-1	NR	Spain	Universitat Politècnica de Catalunya	Civil	Technology Development	
2	Aalto-1	Aalto-1	Finland	Finland	Aalto University	Civil	Technology Development	

3 rows × 41 columns

In [8]:
```
norad_names = dict(zip(df['Name of Satellite, Alternate Names'], df['NORAD
available_norad_ids = satdata.get_norad_ids()
norad_names.pop([el for el in list(norad_names.keys()) if type(el) != str][
norad_names = {k:int(v) for k,v in norad_names.items() if int(v) in availab
```

Step 2. Load the AIS data

Since the example in this notebook is from the period of time of 2009, we just need to load its AIS tracks.

In [9]:
```
ais = pd.read_hdf(os.path.join(AIS_DIR, AIS_FILENAME))
ais.sort_values(by="date_time", inplace=True)
ais.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3187260 entries, 309064 to 3186559
Data columns (total 4 columns):
 #   Column     Dtype
---  ------     -----
 0   mmsi_id    int64
 1   date_time  datetime64[ns]
 2   lat        float32
 3   lon        float32
dtypes: datetime64[ns](1), float32(2), int64(1)
memory usage: 97.3 MB
```

Step 3. Compute the visible points

In [10]:
```
from scripts import intersect; intersect.PRINT_INFO=False
```

Step 4. Visualize the results

Start by loading vessel metadata:

In [11]:
```
vessel_categories = pd.read_csv("./metadata/AIS_categories.csv")
vessel_df = pd.read_csv("./data/vessel data/ConsolidatedAIS/Vessel.csv")
vessel_info_dict = {row['mmsi_id']:{'vessel_name':row['vessel_name'], 'leng
'vessel_type': vessel_categories[vessel_categories['num']==(0 if np.isnan(r
                                  else int(row['vessel_type']))].iloc[0]['de
                                     'width':row['width']} for i, row in ves
```

Utility functions

In [12]:
```
def modulo_lon(val):
    return (val+180) % 360 - 180

def get_track(lat, lon, lat_clip=85.5):
    "Turn track of latitudes and longitudes into NaN-separated Curve"
    mask = np.abs(lat) > lat_clip
    lat[mask] = np.float('nan')
    lon[mask] = np.float('nan')
    lon = np.array([modulo_lon(el) for el in lon])

    eastings, northings = hv.util.transform.lon_lat_to_easting_northing(lon
    # Heuristic to insert NaNs to break up Curve (prevent wrapping issues a
    inds = np.where(np.abs(np.diff(eastings)) > 2e7)[0] # Big delta to spli
    inds += 1
    eastings  = np.insert(eastings,  inds, [float('nan') for i in range(len
    northings = np.insert(northings, inds, [float('nan') for i in range(len
    return hv.Curve((eastings, northings))


def grouby_mmsid(hits):
    "Apply a groupby, reindex on sorted datetimes"
    group = {}
    for mmsi_id, df in hits.groupby('mmsi_id'):
        df['timestamp'] = pd.to_datetime(df['date_time'])
        # Assuming sorted avoiding .sort_values(by='timestamp')
        group[mmsi_id]= df.drop_duplicates().set_index('timestamp')
    return group

table_cols = ['vessel_name', 'mmsi_id', 'vessel_type', 'start_lat', 'end_la
              'start_lat', 'start_lon', 'length', 'width']
def viewable_vessel_df(hits_mmsid_groupby, vessel_info_dict, ):
    data = []
    for mmsi_id, df in hits_mmsid_groupby.items():
        start, end = df.iloc[0], df.iloc[-1]
        start_lat, end_lat = start['lat'], end['lat']
        start_lon, end_lon = start['lon'], end['lon']
        vessel_record = vessel_info_dict.get(mmsi_id,
                                             dict({k:'' for k in table_cols
        vessel_info = {k: '' if (isinstance(v, float) and np.isnan(v)) else
                       for k,v in vessel_record.items()}
        data.append({'mmsi_id':mmsi_id,'vessel_name':vessel_info['vessel_na
                     'vessel_type':vessel_info['vessel_type'],
                     'start_lat':start_lat, 'end_lat':end_lat,
                     'start_lon':start_lon, 'end_lon':end_lon,
                     'length':vessel_info['length'], 'width':vessel_info['w

    return pd.DataFrame(data).sort_values(by='mmsi_id')

def get_vessels(hits_mmsid_groupby, start_date, end_date, lat_limit=85.5):
    "Mark the vessels in the AIS data at the midpoint between start and end
    sdate = dt.datetime(start_date.year, start_date.month, start_date.day)
    edate = dt.datetime(end_date.year, end_date.month, end_date.day)
    middate = sdate + (edate - sdate) / 2
    lats, lons, lengths, widths, vessel_names = [], [], [], [], []
    for mmsi_id, df in hits_mmsid_groupby.items():
        idx = df.index.get_loc(middate, method='nearest')
```

```
        vinfo = vessel_info_dict.get(mmsi_id, {})
        vessel_names.append(vinfo.get('vessel_name', 'Unknown'))
        lengths.append(vinfo.get('length', 'Unknown'))
        widths.append(vinfo.get('width', 'Unknown'))
        lat = float(df.iloc[idx]['lat'])
        lats.append(lat if abs(lat) < lat_limit else float('nan'))
        lons.append(float(df.iloc[idx]['lon']) if abs(lat) < lat_limit else

    eastings, northings = hv.util.transform.lon_lat_to_easting_northing(np.
    tooltips = [("name", "@name"), ("latitude", "@lat"), ("longitude", "@lo
                ("length", "@length"), ("width", "@width")]
    return hv.Points((eastings, northings, vessel_names, lengths, widths, l
                      vdims=['name', 'length', 'width', 'lat', 'lon']).opts(
                                                              tools=[HoverToo
```

DynamicMap callback:

In [13]:
```
def rasterize_hits(name_dict, start_dict, end_dict, start_hours_dict, end_h
                   checkbox_dict, plot_size_dict, rangexy_dict):
    "DynamicMap callback plotting rasterized hits, satellite track and vess
    name, start_date, end_date = name_dict['value'], start_dict['value'], e
    start_hours, end_hours = start_hours_dict['value'], end_hours_dict['val
    full_range = checkbox_dict['value']
    norad_id = int(norad_names[name])
    start_time = pd.Timestamp(year=start_date.year, month=start_date.month,
                              hour = start_hours.hour, minute=start_hours.m

    end_time = pd.Timestamp(year=end_date.year, month=end_date.month, day=e
                            hour = end_hours.hour, minute=end_hours.minute,
    if full_range:
        start_time, end_time = satdata.get_timespan(norad_id)
    try:
        (times, lats, lons, alts) = satdata.get_precomputed_tracks(norad_id
    except:
        print('Exception in get_precomputed_tracks: %s' % str(e))
        return hv.Overlay([])

    # Need longitudes in (-180,180) format, not 0-360
    mask = lons > 180.0
    lons[mask] -= 360

    try:
        sat = pd.DataFrame({"date_time": times.astype("<M8[s]"),"lat": lats
        hits = intersect.compute_hits(sat, ais, start_time=start_time, end_
    except Exception as e:
        print('Exception in compute_hits: %s' % str(e))
        return hv.Overlay()

    hits_mmsid_groupby = grouby_mmsid(hits)
    hit_vessel_info = viewable_vessel_df(hits_mmsid_groupby, vessel_info_di
    drilldown.selection = hit_vessel_info
    mask = (np.abs(hits['lat']) < 85)
    eastings, northings = hv.util.transform.lon_lat_to_easting_northing(hit
    rasterim = rasterize(hv.Points(pd.DataFrame({'northing':northings[mask]
                    'easting':eastings[mask]}), ['easting', 'northing']),
                             width = int(plot_size_dict['width']), height =
                             x_range=rangexy_dict['x_range'], y_range=range
                            ).opts(cmap=fire[180:], width=700, height=500,

    elements = [rasterim]
    if not full_range:
        elements += [get_track(lats, lons).opts(color='red'),
                     get_vessels(hits_mmsid_groupby, start_date, end_date)]
    return hv.Overlay(elements)
```

Declaring panel widgets

Satellite selector widgets:

In [14]:
```
satellites = list(norad_names.keys())
constellations = sorted(list(set([re.sub(r'[ -].*', '', str(s)) for s in sa
constellation = pn.widgets.Select(options=constellations, name="Constellati
constellation.value = 'International'
satellite = pn.widgets.Select(options=[s for s in satellites
                                       if re.match(constellation.value, str
                                sizing_mode='stretch_width', name="Satellite"

@pn.depends(constellation.param.value, watch=True)
def update_satellite_options(constellation):
    satellite.options = [s for s in satellites if re.match(constellation, s
    satellite.param.value = satellite.options[0] if satellite.options else
```

The drilldown table and download CSV callback:

In [15]:
```
empty_df = pd.DataFrame({el:[] for el in table_cols})

class Drilldown(param.Parameterized):
    selection = param.DataFrame(empty_df)

    @param.depends('selection')
    def update_table(self, *args, **kwargs):
        return pn.widgets.DataFrame(self.selection, show_index=False,
                                    autosize_mode='fit_columns', height=400

    def csv_download(self):
        sio = StringIO()
        self.selection.to_csv(sio)
        sio.seek(0)
        return sio

drilldown = Drilldown()
```

In [16]:
```
download_button = pn.widgets.FileDownload(
    callback=drilldown.csv_download, filename='hits.csv', sizing_mode='stre
```

Date and checkbox widgets:

In [17]:
```
start_date = pn.widgets.DatePicker(name='Start Date', value=dt.date(2015, 1
end_date = pn.widgets.DatePicker(name='End Date', value=dt.date(2015, 1, 4)
full_range = pn.widgets.Checkbox(name='Full date range', sizing_mode='stret
map_opacity = pn.widgets.FloatSlider(name='Map opacity', value=0.7, start=0
```

Time widgets:

In [18]:
```
zero_hours = dt.datetime(2020, 1, 1, 0, 0, 0, 0)
twelve_hours = dt.datetime(2020, 1, 1, 12, 0, 0, 0)
start_time = pn.widgets.DatetimeInput(value=zero_hours, format="%H:%M",
                                      width=80, name='Start Time', align='e
end_time = pn.widgets.DatetimeInput(value=twelve_hours, format="%H:%M",
                                    width=80, name='End Time', align='end',
```

Setting up callback to disable date pickers when 'full date range' checkbox active:

In [19]:
```
@pn.depends(full_range.param.value, watch=True)
def disable_callback(full_range):
    start_date.disabled = full_range
    end_date.disabled = full_range
```

Declaring HoloViews elements

In [20]:
```
tiles = hv.element.tiles.ESRI().redim(x='easting', y='northing').opts(bgcol
hits_dmap = hv.DynamicMap(rasterize_hits,
                          streams=[satellite.param.value,  start_date.param
                                   start_time.param.value, end_time.param.v
                                   hv.streams.PlotSize(width=700, height=50
                     positional_stream_args=True)
```

Declaring Panel dashboard

In [21]:
```
logo_width=50
logos = pn.Row(pn.pane.Markdown('# AIS Visibility Dashboard', width=400),
               pn.pane.SVG('./anaconda.svg', width=logo_width, height=logo_
               pn.pane.SVG('./ATT_logo.svg', width=logo_width, height=logo_
```

In [22]:
```
instructions = """
Select a date/time range and a satellite, and this dashboard will show you
of that satellite over the time range, plus the vessels visible from that s
Zoom around Alaska to see the vessels in detail, after selecting
the Scroll Zoom tool on the plot.
"""
```

In [23]:
```
viz = pn.Column(pn.pane.Markdown(instructions, width=800),
                pn.Row(tiles.opts(padding=0) * hits_dmap.opts(padding=0), d
all_widgets = pn.Column(full_range, pn.Row(start_date, start_time, sizing_m
                        pn.Row(end_date, end_time, sizing_mode='stretch_wid
                        satellite, map_opacity, download_button, sizing_mod
```

In [32]:
```
pn.Column(all_widgets, viz)
```

Out[32]:

☐ Full date range

Start Date

2015-01-01

End Date

2015-01-04

Constellation

International

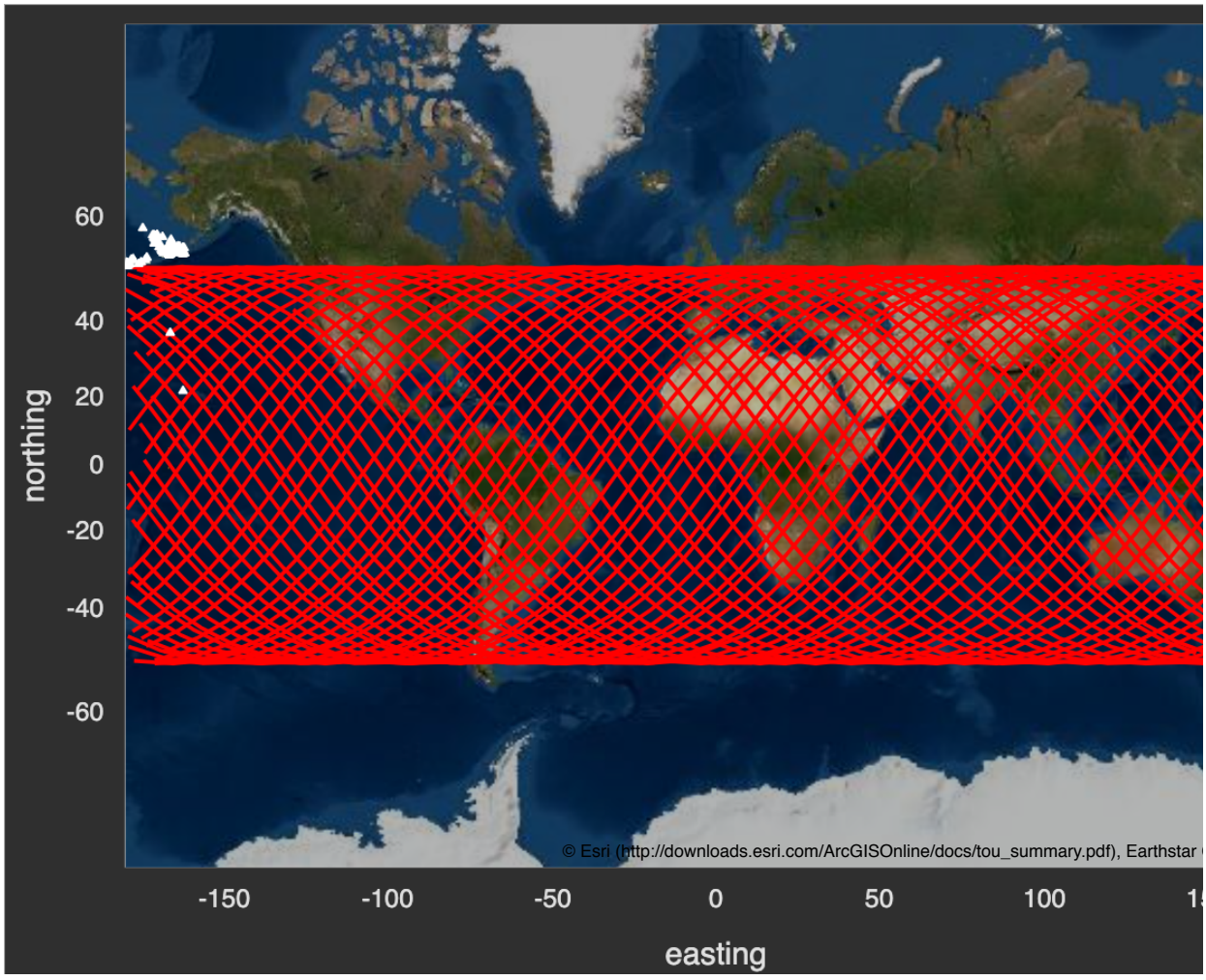
Satellite

International Space Station (ISS [first element Zarya])

Map opacity: **0.70**

Download hits.csv

Select a date/time range and a satellite, and this dashboard will show you the track of that satellite over the visible from that satellite. Zoom around Alaska to see the vessels in detail, after selecting the Scroll Zoom to



In [29]:
```
template = pn.template.MaterialTemplate(title='AIS Visibility Dashboard', t
template.sidebar.append(all_widgets)
template.main.append(viz)
template.servable();
```