Assignment 3

Submitted by:

Name	Banner ID
Gaganpreet Singh	B00819217
Shruthi Kalasapura Ramesh	B00822766

The Code and the ouput files for this assignment are present on [GitLab (https://git.cs.dal.ca/singh1/a3-data-science/blob/master/)](https://git.cs.dal.ca/singh1/a3-data-science/blob/master/)

Importing libraries and provided dataset.

In [1]:
```
import os
import geopandas as gpd
import numpy as np
import pandas as pd
from shapely.geometry import Point
import shapely
import missingno as msn
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import matplotlib.cm as cm
from matplotlib.colors import Normalize
import pandas
```

In [2]:
```
df=pd.read_csv('/Users/gaganpree99/Desktop/DataScience/A3/AISData.csv')

df.rename(columns={'location.coordinates.0':'x','location.coordinates.1'
:'y'},
          inplace=True)

gdf = gpd.GeoDataFrame(df.drop(['x', 'y'], axis=1),
                          crs={'init': 'epsg:4326'},
                          geometry=[shapely.geometry.Point(xy) for xy in zi
p(df.x, df.y)])
```

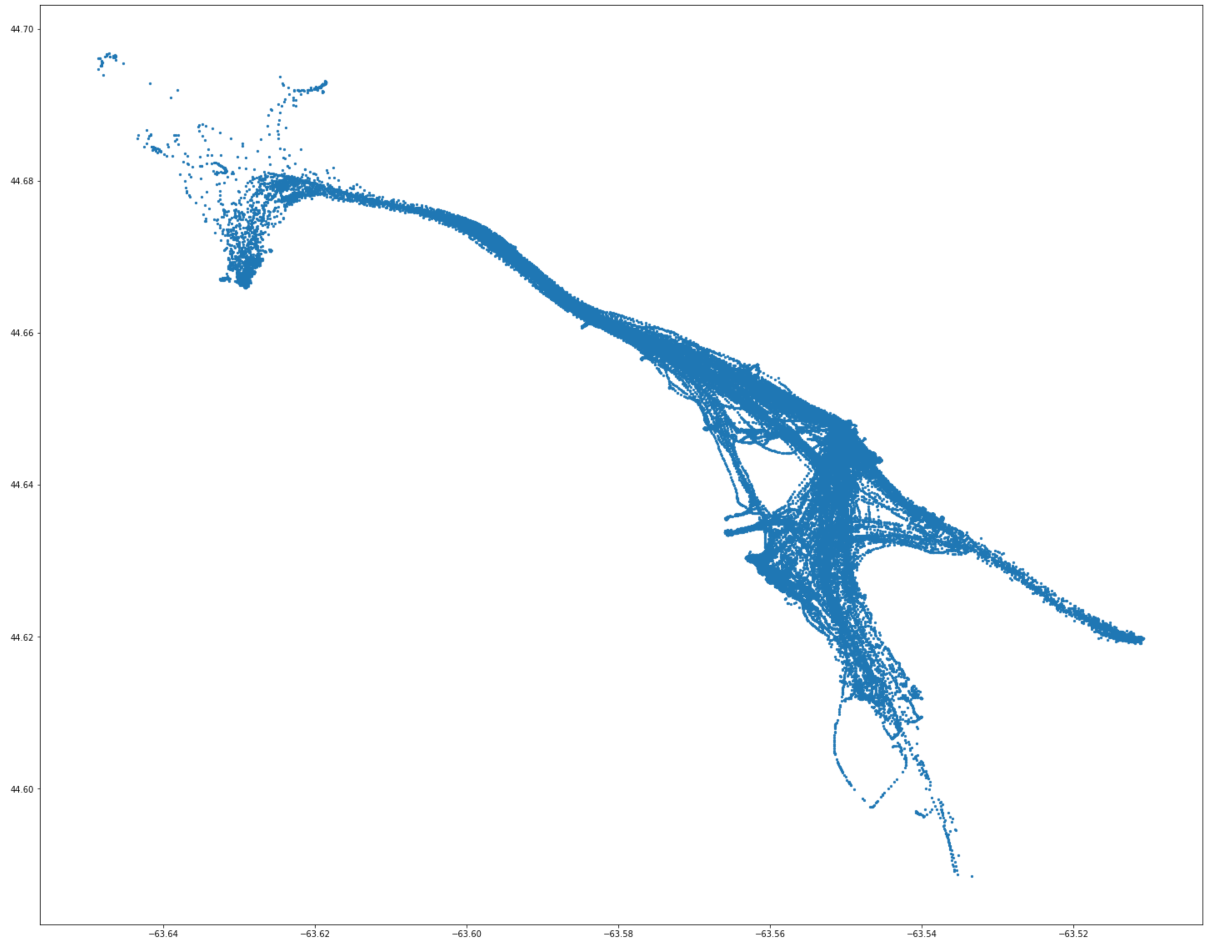
Analyzing GeoDataframe for the AIS data

```
In [3]: gdf.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 766671 entries, 0 to 766670
Data columns (total 7 columns):
Unnamed: 0           766671 non-null int64
event_time           766671 non-null object
position_accuracy    697295 non-null float64
mmsi                 766671 non-null int64
sog                  697295 non-null float64
cog                  697295 non-null float64
geometry             766671 non-null object
dtypes: float64(3), int64(2), object(2)
memory usage: 40.9+ MB
```

Plot for AIS Data:

```
In [4]: ax=gdf.plot(figsize=(25,25),markersize=5)
```



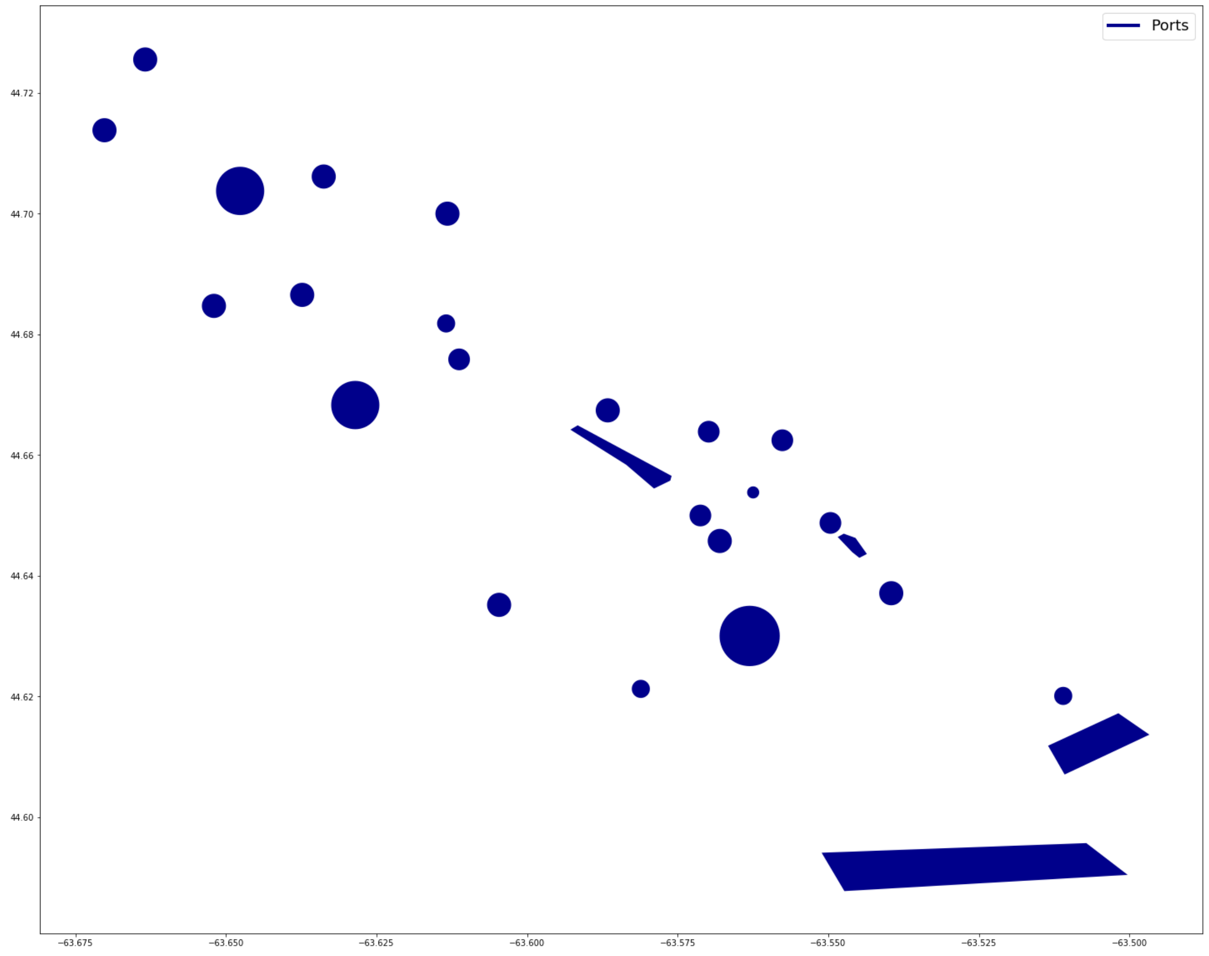
Analyzing the shapefile for Canadian ports

In [5]:
```
import fiona
import pprint
with fiona.open('/Users/gaganpree99/Desktop/DataScience/A3/Nima_Ports/as
signment3shapefile.shp') as src:
    pprint.pprint(src[:3])
```

```
[{'geometry': {'coordinates': [[(-63.5916030406951 9, 44.66492922546069
6),
                                (-63.576014041900635, 44.65655014555536
4),
                                (-63.57622861862184, 44.6557716996118
5),
                                (-63.57893228530884, 44.6544590023915
9),
                                (-63.58354568481445, 44.6583970049068
5),
                                (-63.59281539916991, 44.6642119381484
5),
                                (-63.59160304069519, 44.66492922546069
6)]],
               'type': 'Polygon'},
  'id': '0',
  'properties': OrderedDict([('field_1', 0),
                             ('port_name', 'pointpolygon'),
                             ('size', 0.0)]),
  'type': 'Feature'},
 {'geometry': {'coordinates': [[(-63.569431, 44.649993),
                                (-63.56943966749199, 44.6498165691474
1),
                                (-63.56946558649528, 44.6496418374203
8),
                                (-63.56950850739569, 44.6494704875809
4),
                                (-63.569568016841494, 44.6493041698217
5),
                                (-63.56964354172419, 44.6491444858737
1),
                                (-63.56973435469786, 44.64899297358057
6),
                                (-63.569839581183956, 44.6488510920885
1),
                                (-63.56995820779387, 44.6487202077938
6),
                                (-63.57008909208851, 44.6486015811839
5),
                                (-63.57023097358057, 44.6484963546978
5),
                                (-63.57038248587371, 44.6484055417241
7),
                                (-63.570542169821756, 44.6483300168414
8),
                                (-63.570708487580944, 44.6482705073956
9),
                                (-63.570879837420385, 44.6482275864952
7),
                                (-63.57105456914741, 44.6482016674919
9),
                                (-63.571231, 44.648193),
                                (-63.5714074308526, 44.64820166749199),
                                (-63.57158216257964, 44.6482275864952
7),
                                (-63.571753512419065, 44.6482705073956
9),
                                        (-63.57191983017827, 44.6483300168414
8),
                                        (-63.57207951412629, 44.6484055417241
7),
                                        (-63.57223102641944, 44.6484963546978
5),
                                        (-63.5723729079115, 44.64860158118395),
                                        (-63.572503792206135, 44.6487202077938
6),
                                        (-63.57262241881607, 44.6488510920885
1),
                                        (-63.57272764530215, 44.64899297358057
6),
                                        (-63.57281845827583, 44.6491444858737
1),
                                        (-63.57289398315852, 44.6493041698217
5),
                                        (-63.57295349260432, 44.6494704875809
4),
                                        (-63.57299641350473, 44.6496418374203
8),
                                        (-63.57302233250802, 44.6498165691474
1),
                                        (-63.573031, 44.649993),
                                        (-63.57302233250802, 44.6501694308525
9),
                                        (-63.57299641350473, 44.6503441625796
3),
                                        (-63.57295349260432, 44.6505155124190
7),
                                        (-63.57289398315852, 44.6506818301782
6),
                                        (-63.57281845827583, 44.6508415141262
9),
                                        (-63.57272764530215, 44.6509930264194
4),
                                        (-63.57262241881607, 44.6511349079115),
                                        (-63.572503792206135, 44.6512657922061
4),
                                        (-63.5723729079115, 44.65138441881606),
                                        (-63.57223102641944, 44.6514896453021
5),
                                        (-63.57207951412629, 44.6515804582758
3),
                                        (-63.57191983017827, 44.6516559831585
2),
                                        (-63.571753512419065, 44.6517154926043
2),
                                        (-63.57158216257964, 44.6517584135047
3),
                                        (-63.5714074308526, 44.65178433250802),
                                        (-63.571231, 44.651793),
                                        (-63.57105456914741, 44.6517843325080
2),
                                        (-63.570879837420385, 44.6517584135047
3),
                                        (-63.570708487580944, 44.6517154926043
2),
                                (-63.570542169821756, 44.6516559831585
2),
                                (-63.57038248587371, 44.6515804582758
3),
                                (-63.57023097358057, 44.6514896453021
5),
                                (-63.57008909208851, 44.6513844188160
6),
                                (-63.56995820779387, 44.6512657922061
4),
                                (-63.569839581183956, 44.651134907911
5),
                                (-63.56973435469786, 44.6509930264194
4),
                                (-63.56964354172419, 44.6508415141262
9),
                                (-63.569568016841494, 44.6506818301782
6),
                                (-63.56950850739569, 44.6505155124190
7),
                                (-63.56946558649528, 44.6503441625796
3),
                                (-63.56943966749199, 44.6501694308525
9),
                                (-63.569431, 44.649993)]],
               'type': 'Polygon'},
  'id': '1',
  'properties': OrderedDict([('field_1', 1),
                             ('port_name', 'port1'),
                             ('size', 0.0018)]),
  'type': 'Feature'},
 {'geometry': {'coordinates': [[(-63.60949000000001, 44.675853),
                                (-63.60949866749199, 44.6756765691474
1),
                                (-63.60952458649529, 44.6755018374203
8),
                                (-63.60956750739569, 44.6753304875809
4),
                                (-63.60962701684149, 44.6751641698217
5),
                                (-63.60970254172419, 44.6750044858737
1),
                                (-63.60979335469786, 44.6748529735805
7),
                                (-63.60989858118394, 44.6747110920885
1),
                                (-63.610017207793874, 44.6745802077938
7),
                                (-63.61014809208851, 44.6744615811839
5),
                                (-63.61028997358057, 44.6743563546978
6),
                                (-63.61044148587371, 44.6742655417241
7),
                                (-63.61060116982175, 44.6741900168414
9),
                                 (-63.61076748758094, 44.6741305073956
9),
                                 (-63.61093883742037, 44.6740875864952
8),
                                 (-63.61111356914741, 44.6740616674919
9),
                                 (-63.61129, 44.674053),
                                 (-63.61146643085259, 44.6740616674919
9),
                                 (-63.61164116257964, 44.6740875864952
8),
                                 (-63.61181251241905, 44.6741305073956
9),
                                 (-63.611978830178266, 44.6741900168414
9),
                                 (-63.612138514126286, 44.6742655417241
7),
                                 (-63.61229002641944, 44.6743563546978
6),
                                 (-63.6124319079115, 44.67446158118395),
                                 (-63.61256279220614, 44.6745802077938
7),
                                 (-63.61268141881607, 44.6747110920885
1),
                                 (-63.61278664530215, 44.6748529735805
7),
                                 (-63.61287745827583, 44.6750044858737
1),
                                 (-63.61295298315853, 44.6751641698217
5),
                                 (-63.61301249260433, 44.6753304875809
4),
                                 (-63.61305541350473, 44.6755018374203
8),
                                 (-63.613081332508O2, 44.6756765691474
1),
                                 (-63.61309000000001, 44.675853),
                                 (-63.61308133250802, 44.6760294308525
9),
                                 (-63.61305541350473, 44.6762041625796
3),
                                 (-63.61301249260433, 44.6763755124190
6),
                                 (-63.61295298315853, 44.67654183017826
6),
                                 (-63.61287745827583, 44.6767015141262
9),
                                 (-63.61278664530215, 44.6768530264194
4),
                                 (-63.61268141881607, 44.6769949079115),
                                 (-63.61256279220614, 44.6771257922061
4),
                                 (-63.6124319079115, 44.67724441881606),
                                 (-63.61229002641944, 44.6773496453021
5),
                                 (-63.612138514126286, 44.6774404582758
3),
                                          (-63.611978830178266, 44.6775159831585
2),
                                          (-63.61181251241905, 44.6775754926043
2),
                                          (-63.61164116257964, 44.6776184135047
3),
                                          (-63.61146643085259, 44.6776443325080
2),
                                          (-63.61129, 44.67765300000001),
                                          (-63.61111356914741, 44.6776443325080
2),
                                          (-63.61093883742037, 44.6776184135047
3),
                                          (-63.61076748758094, 44.6775754926043
2),
                                          (-63.61060116982175, 44.6775159831585
2),
                                          (-63.61044148587371, 44.6774404582758
3),
                                          (-63.61028997358057, 44.6773496453021
5),
                                          (-63.61014809208851, 44.6772444188160
6),
                                          (-63.610017207793874, 44.6771257922061
4),
                                          (-63.60989858118394, 44.6769949079115),
                                          (-63.60979335469786, 44.6768530264194
4),
                                          (-63.60970254172419, 44.6767015141262
9),
                                          (-63.60962701684149, 44.67654183017826
6),
                                          (-63.60956750739569, 44.6763755124190
6),
                                          (-63.60952458649529, 44.6762041625796
3),
                                          (-63.60949866749199, 44.6760294308525
9),
                                          (-63.60949000000001, 44.675853)]],
                  'type': 'Polygon'},
   'id': '2',
   'properties': OrderedDict([('field_1', 2),
                              ('port_name', 'port2'),
                              ('size', 0.0018)]),
   'type': 'Feature'}]
```

Ploting Canadian ports

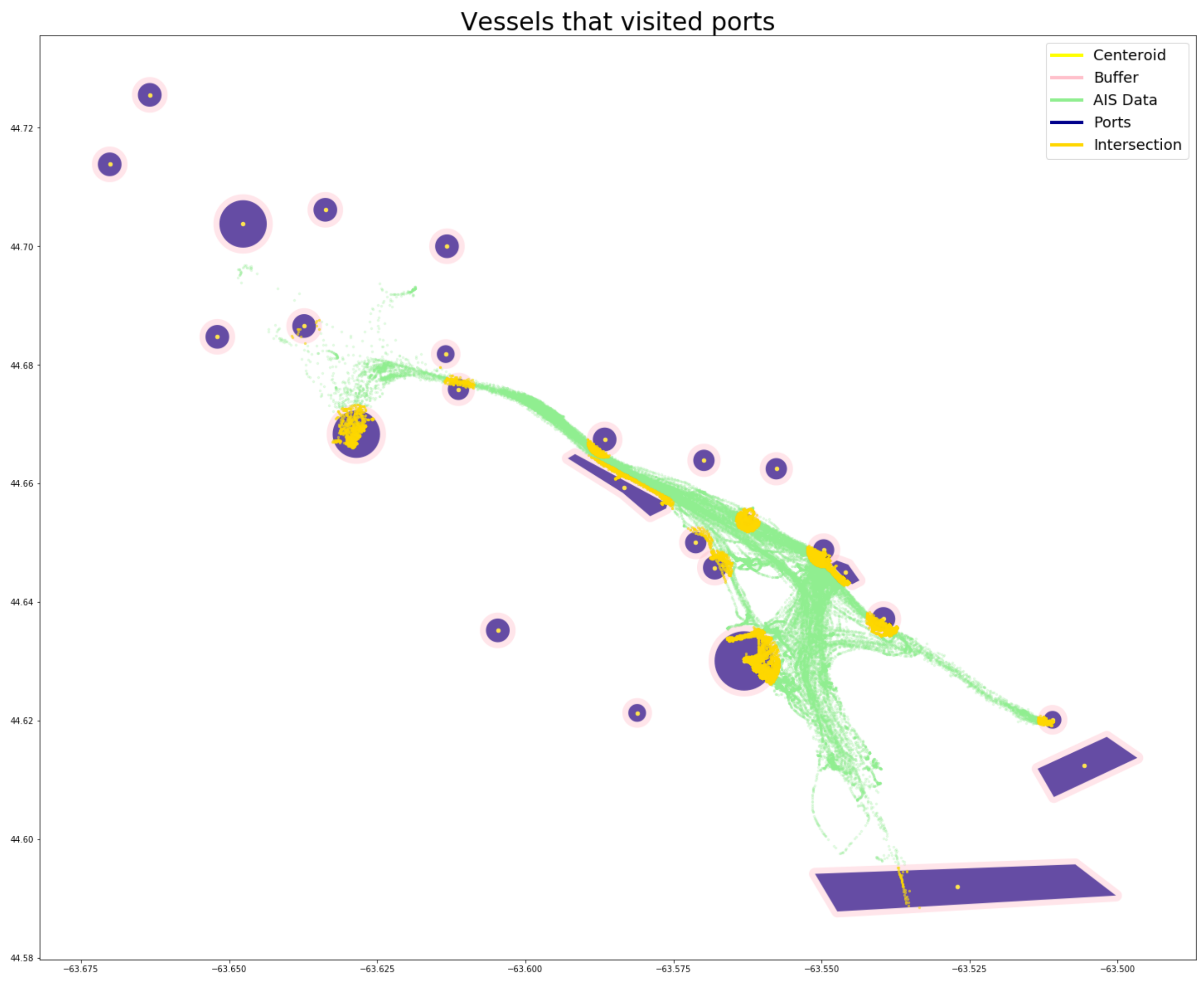
In [6]:
```
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

ports = gpd.read_file('/Users/gaganpree99/Desktop/DataScience/A3/Nima_Po
rts/assignment3shapefile.shp')
# ports.crs = {'init' :'epsg:4326'}
ports.crs = {'init': 'epsg:3006'}
byportName=ports.set_index(['port_name'])
ax=byportName.plot(figsize=(25,25),color='darkblue')

legend_elements = [Line2D([0], [0], color='darkblue', lw=4, label='Port
s')]

ax.legend(handles=legend_elements,prop=dict(size=18))
plt.show()
```



1. Find all the vessels that visited ports in the provided shapefile.

In [7]:
```
centeroids = []
points = []

for x in ports['port_name']:
#   print(x)
    centeroids.append(byportName.loc[[x],:].centroid)

# # To create buffer around centeroids
# for center in centeroids:
#     p = center.geometry.buffer(0.005)
#     points.append(p[0])

# # To create buffer around shape of port
for shape in ports['port_name']:
    p = byportName.loc[[shape],:].geometry.buffer(0.001)
    points.append(p[0])
```

In [8]:
```
ax = byportName.loc[ports['port_name'],:].plot(color='darkblue', figsize
=(25,25),markersize=5)

for center in centeroids:
    gpd.GeoSeries(center).plot(ax=ax, color='yellow', markersize=20)

gdf.plot(ax=ax, color='lightgreen', alpha=0.2, markersize=5)

for shape in points:
    gpd.GeoSeries(shape).plot(ax=ax, color='pink', markersize=5, alpha =
0.4)
    gdf.loc[gdf.within(shape),:].plot(ax=ax, color='gold', markersize=5,
alpha=0.5)

ax.set_title("Vessels that visited ports", fontdict={'fontsize': 30, 'fo
ntweight': 'medium'})
legend_elements = [Line2D([0], [0], color='yellow', lw=4, label='Centero
id'),
                   Line2D([0], [0], color='pink', lw=4, label='Buffer'),
                   Line2D([0], [0], color='lightgreen', lw=4, label='AIS
Data'),
                   Line2D([0], [0], color='darkblue', lw=4, label='Port
s'),
                   Line2D([0], [0], color='gold', lw=4, label='Intersect
ion')
                   ]

ax.legend(handles=legend_elements,prop=dict(size=18))
plt.show()
```



The graph shown above shows the intersection of AIS Data with ports in Gold color. A buffer of '0.001' is drawn around the shape of each port in pink color.

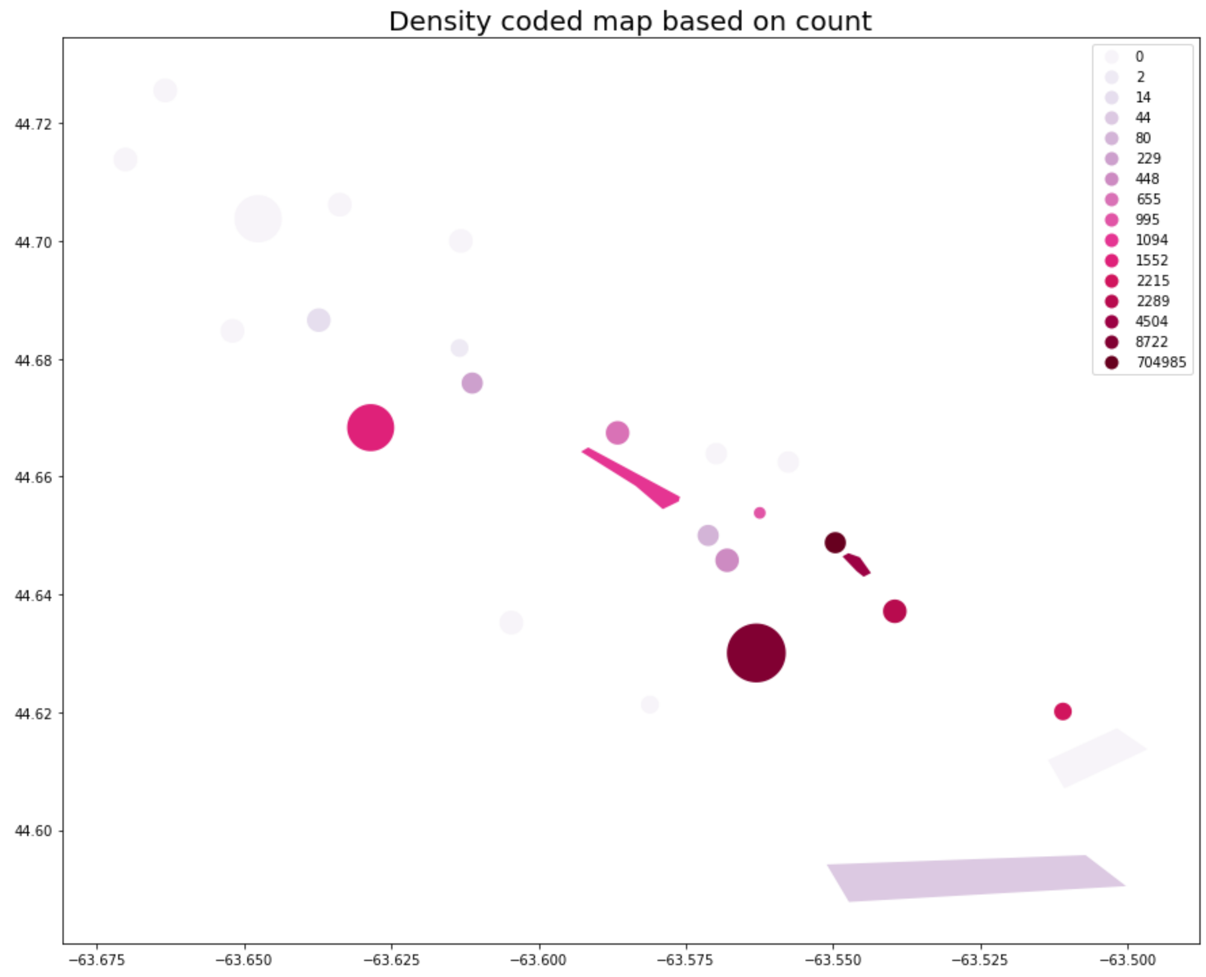
---

2. Show the density of each port using a colour-coded map.

In [10]:
```python
import matplotlib.cm as cm
from matplotlib.colors import Normalize

densityDataframe = gpd.GeoDataFrame(columns=["PortName", "Count", "Densi
ty", "geometry"])

for areaName in set(byportName.index.values):
    area=byportName.loc[byportName.index == areaName,:]
    bufferedArea=area.geometry.buffer(0.001)[0]
    intersections=gdf.loc[gdf.within(bufferedArea),:]
    count = intersections.shape[0]
    density = count/(gdf.shape[0])

    densityDataframe = densityDataframe.append({
        "PortName": areaName,
        "Count": count,
        "Density": density,
        "geometry": area.geometry[0]
    }, ignore_index=True)
```

In [11]:
```python
fig, ax = plt.subplots(1, figsize= (15,15))
densityDataframe.plot(ax = ax, column = 'Count', legend=True, cmap='PuR
d',label='curve1')
ax.set_title("Density coded map based on count", fontdict={'fontsize': 2
0, 'fontweight': 'medium'})
```

Out[11]:
```
Text(0.5, 1, 'Density coded map based on count')
```



The above plot shows the color coded map for the provided AIS dataset.

As we can see from this plot, the density of the ports is in extreme ends. Either it is too low or it is too high.

3. Now divide the AIS data into data frames with a one-hour interval. Repeat steps 1 and 2 forall of the sub-dataframes.

Splitting Data on hourly interval.

In [12]:
```python
sortBasedOnTime = gdf.sort_values(by="event_time").reset_index(drop=True
)
```

```
In [13]: list = []
         S = pd.to_datetime(sortBasedOnTime.event_time)
         for i, g in sortBasedOnTime.groupby([(S - S[0]).astype('timedelta64[1h]'
         )]):
                 list.append(g.reset_index(drop=True))
```

A function to create Directory for storing plots

```
In [15]: def createDirectory(output_path):
             import shutil, os
             try:
                 shutil.rmtree(output_path)
                 os.mkdir(output_path)
             except OSError:
                 os.mkdir(output_path)
                 print ("Successfully created the directory %s " % output_path)
             else:
                 print ("Successfully created the directory %s " % output_path)
```

Repeating Question 1 on the new dataset created. The plots for this have been saved into 'maps_1/' directory.

In [16]:
```
output_path = 'maps_1'
createDirectory(output_path)

filenames = []
count = 0
for data in list:

    ax=byportName.loc[ports['port_name'],:].plot(figsize=(15,15))
    fig=data.plot(ax=ax, color='lightgreen', markersize=5)

    for center in centeroids:
        gpd.GeoSeries(center).plot(ax=ax, color='yellow', markersize=20)

    for shape in points:
        gpd.GeoSeries(shape).plot(ax=ax, color='pink', markersize=5, alp
ha = 0.4)
        data.loc[data.within(shape),:].plot(ax=ax, color='gold', markers
ize=5, alpha=0.5)


    legend_elements = [Line2D([0], [0], color='yellow', lw=4, label='Cen
teroid'),
                       Line2D([0], [0], color='pink', lw=4, label='Buffer'),
                       Line2D([0], [0], color='lightgreen', lw=4, label='AIS
Data'),
                       Line2D([0], [0], color='darkblue', lw=4, label='Port
s'),
                       Line2D([0], [0], color='gold', lw=4, label='Intersect
ion')
                       ]

    ax.legend(handles=legend_elements,prop=dict(size=18))

    # add a title
    fig.set_title('Vessels that visited ports per hour', \
              fontdict={'fontsize': '25', 'fontweight' : '3'})

    # this will save the figure as a high-res png in the output path. yo
u can also save as svg if you prefer.
    filenames.append(str(count) +'.jpg')
    filepath = os.path.join(output_path, filenames[count])
    chart = fig.get_figure()
    chart.savefig(filepath, dpi=75)
    count = count + 1
    plt.close("all")
```

```
Successfully created the directory maps_1
```

Repeating Question 2 on the new dataset created. The plots for this have been saved into 'maps_2/' directory.

```
In [17]: output_path = 'maps_2'
         createDirectory(output_path)

         filenames = []
         temp = 0
         for data in list:
             densityTemp = gpd.GeoDataFrame(columns=["PortName", "Count", "Densit
         y", "geometry"])

             ax=byportName.loc[ports['port_name'],:].plot(color='black', figsize=
         (15,15))
             for areaName in set(byportName.index.values):
                 area=byportName.loc[byportName.index == areaName,:]
                 bufferedArea=area.geometry.buffer(0.001)[0]
                 intersections=data.loc[data.within(bufferedArea),:]
                 count = intersections.shape[0]
                 density = count*1.0/gdf.shape[0]

                 densityTemp = densityTemp.append({
                     "PortName": areaName,
                     "Count": count,
                     "Density": density,
                     "geometry": area.geometry[0]
                 }, ignore_index=True)

             fig = densityTemp.plot(ax = ax, column = 'Count', legend=True, cmap=
         'Paired')


             # add a title
             fig.set_title('Density Coded Map/hour', \
                           fontdict={'fontsize': '25', 'fontweight' : '3'})

             # this will save the figure as a high-res png in the output path. yo
         u can also save as svg if you prefer.
             filenames.append(str(temp) +'.jpg')
             filepath = os.path.join(output_path, filenames[temp])
             chart = fig.get_figure()
             chart.savefig(filepath, dpi=75)
             temp = temp + 1
             plt.close("all")
```

```
Successfully created the directory maps_2
```

Function to create animation for the created plots:

```
In [22]: from os import listdir
         from os.path import isfile, join
         import imageio, os

         def creatAnimation(output_path):
             onlyfiles = [f for f in listdir(output_path) if isfile(join(output_p
         ath, f))]
             images = []
             for filename in onlyfiles:
                 filepath = os.path.join(output_path, filename)
                 images.append(imageio.imread(filepath))
             pathName = os.path.join(output_path + "/movie.gif")
             imageio.mimsave(pathName, images)
```

```
In [23]: creatAnimation("maps_1")
         creatAnimation("maps_2")
```

Animation for Q1	Animation for Q2
Animation for Q1	Animation for Q2

The animations for both Question 1 and Question 2 are shown in the figure above. The first part shows the Vessels visiting the ports in 1 hour interval and the second part shows the Density Coded map for the same interval.

NOTE: If you are not able to visualize the animations in this file. You can directly navigate to the folder `maps_1` and `maps_2` and check the `movie.gif` file there.

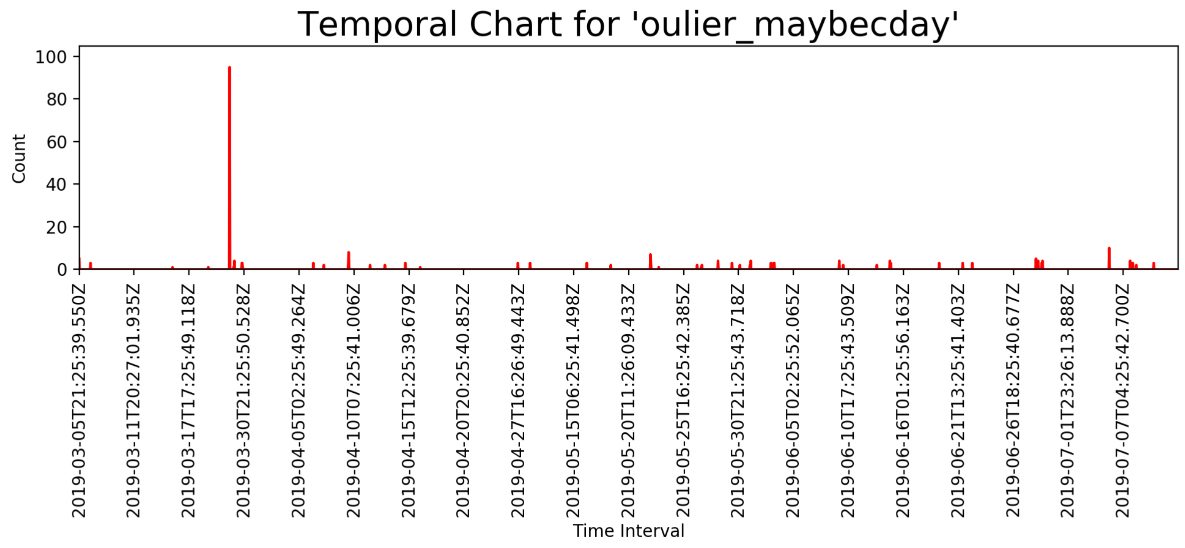
or You can download the animations from GitLab: maps_1/movie.gif (https://git.cs.dal.ca/singh1/a3-data-science/blob/master/maps_1/movie.gif) and GitLab: maps_2/movie.gif (https://git.cs.dal.ca/singh1/a3-data-science/blob/master/maps_2/movie.gif)

4. Select any port you like. Create a temporal chart for the density of messages in that port.

The port we have selected is 'oulier_maybecday'.

The following code plots the temporal chart for the density of port at a specific hour.

In [41]:
```python
from matplotlib import rcParams
import time
rcParams.update({'figure.autolayout': True})
%matplotlib notebook

xdata = []
xlabel = []
ydata = []
plt.show()

axes = plt.gca()

axes.set_xlim(0, len(list))


line, = axes.plot(xdata, ydata, 'r-')
# plt.figure(figsize=(9,5))


areaName = 'oulier_maybecday'
area = byportName.loc[byportName.index == areaName,:]
aa=area.centroid.geometry.buffer(0.001)[0]
t  = 0

for timeInterval in list:
    r=timeInterval.loc[timeInterval.within(aa),:]
    count = r.shape[0]
    if count > 0:
        density = count*1.0/timeInterval.shape[0]
    else:
        density = 0

    xdata.append(t)
    ydata.append(count)
    line.set_xdata(xdata)
    line.set_ydata(ydata)

    if ((t%(125)) == 0):
        xlabel.append(timeInterval['event_time'][0])

    plt.draw()
    plt.pause(1e-17)
    time.sleep(0.1)
    #print(areaName,timeInterval['event_time'][0], count, density)
    t = t + 1

axes.set_ylim(0,max(ydata)+10)
plt.xticks(xdata, xlabel, rotation=90)
plt.title("Temporal Chart for 'oulier_maybecday'", fontdict={'fontsize':
20, 'fontweight': 'medium'})
plt.locator_params(axis='x', tight=True, nbins=20)
plt.xlabel('Time Interval')
plt.ylabel('Count')
plt.show(),
```



```
Out[41]: (None,)
```

The above plot shows the temporal chart for port 'oulier_maybecday'.

We can zoom-in to study the plots in details.

5. Use concept drift methods on step 4 and find out if there is any drift in the data that can be detected.

a. Adaptive Windowing method for concept drift detection (ADWIN)

```
In [144]: from skmultiflow.drift_detection.adwin import ADWIN
          adwin = ADWIN(delta=5)

          for i in range(len(ydata)):
              adwin.add_element(ydata[i])
              if adwin.detected_change():
                  print('Change detected in data: ' + str(ydata[i]) + ', at: ' + s
          tr(list[i]['event_time'][0]))
```

```
Change detected in data: 0, at: 2019-03-09T06:25:58.744Z
Change detected in data: 0, at: 2019-03-31T05:25:40.688Z
```

Even after changing the value of input parameter i.e. `delta` we are not able to see the drifts as expected from temporal chart plotted in previous question.

b. Drift Detection Method (DDM)

```
In [127]: from skmultiflow.drift_detection import DDM
          ddm = DDM(min_num_instances=15, warning_level=2.0, out_control_level=4.0
          )

          for i in range(len(ydata)):
              ddm.add_element(ydata[i])
          #     if ddm.detected_warning_zone():
          #         print('Warning zone has been detected in data: ' + str(ydata
          [i]) + ', at: ' + str(list[i]['event_time'][0]))
              if ddm.detected_change():
                  print('Change has been detected in data: ' + str(ydata[i]) + ',
           at: ' + str(list[i]['event_time'][0]))
```

```
Change has been detected in data: 95, at: 2019-03-29T15:25:39.701Z
Change has been detected in data: 1, at: 2019-03-30T21:25:50.528Z
Change has been detected in data: 3, at: 2019-04-06T15:28:00.286Z
Change has been detected in data: 2, at: 2019-04-07T15:25:40.206Z
Change has been detected in data: 3, at: 2019-04-09T23:25:40.686Z
Change has been detected in data: 4, at: 2019-06-01T13:26:23.460Z
Change has been detected in data: 3, at: 2019-06-03T11:26:13.665Z
```

In DDM, on tweaking the values of input parameters we could see the drifts that can be verified through the temporal chart shown in question 4.

Hence, DDM gives the desired results.

c. Early Drift Detection Method (EDDM)

```
In [106]: from skmultiflow.drift_detection import EDDM
          eddm = EDDM()

          for i in range(len(ydata)):
              eddm.add_element(ydata[i])
              if eddm.detected_warning_zone():
                  print('Warning zone has been detected in data: ' + str(ydata[i])
          + ' - of index: ' + str(i))
              if eddm.detected_change():
                  print('Change has been detected in data: ' + str(ydata[i]) + ' -
          of index: ' + str(i))
```

EDDM detects no change on the datastream. Also, there are no input parameters that can be tweaked to get desired results.

d. Page-Hinkley method for concept drift detection

In [123]:
```
from skmultiflow.drift_detection import PageHinkley
ph = PageHinkley(min_instances=15, delta=0.005, threshold=10, alpha=0.99
99)

for i in range(len(ydata)):
    ph.add_element(ydata[i])
    if ph.detected_change():
        print('Change has been detected in data: ' + str(ydata[i]) + ' -
of index: ' + str(list[i]['event_time'][0]))
```

```
Change has been detected in data: 95 - of index: 2019-03-29T15:25:39.70
1Z
Change has been detected in data: 8 - of index: 2019-04-10T00:30:48.105
Z
Change has been detected in data: 1 - of index: 2019-05-23T18:25:42.865
Z
Change has been detected in data: 4 - of index: 2019-06-01T13:26:23.460
Z
Change has been detected in data: 3 - of index: 2019-06-30T04:25:41.876
Z
Change has been detected in data: 3 - of index: 2019-07-08T21:26:32.642
Z
```

On tuning the input parameters for Page-Hinkley method, we get the drifts that can be verified through temporal chart plotted in question 4.

After executing all the methods for drift detection, we observed that `Drift Detection Method` and `Page-Hinkley Method` are suitable for our use case as they are able to detect the Drifts in the given data stream.

However, if we were to compare the above 2 methods, `Drift Detection Method` has an upper edge over `Page-Hinkley Method` as it can detect warnings as well, which is not possible in case of `Page-Hinkley Method`.

---

6. Cluster the ports based on their message density using DBSCAN and categorize the ports based on traffic (message density).

The following code implements the DBSCAN algorithm based on AIS message density. We have tweaked the input parameters to get optimum results.

The value of `eps` has been set to 600 and the values of `min_samples` has been set to 2.

```
In [87]: from sklearn.cluster import DBSCAN
         clustering = DBSCAN(eps=600, min_samples=2).fit(densityDataframe[['Coun
         t']])
         clustering
```

```
Out[87]: DBSCAN(algorithm='auto', eps=600, leaf_size=30, metric='euclidean',
                metric_params=None, min_samples=2, n_jobs=None, p=None)
```

Finding the number of clusters created.

```
In [88]: num_clusters = len(set(clustering.labels_))
         num_clusters
```

```
Out[88]: 3
```

Finding the Labels for the ports and assigning them to the dataframe.

```
In [89]: cluster_labels = clustering.labels_
         cluster_labels
```

```
Out[89]: array([ 0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1, -1,  0,  1,  0,
         0,
                 0,  0, -1,  0,  0,  0,  0,  0,  0])
```

```
In [90]: densityDataframe['DBSCANlabel'] = cluster_labels
```

Let us have a look at the geodataframe that has the `DBSCANlabel` for each port:

```
In [141]: densityDataframe
```

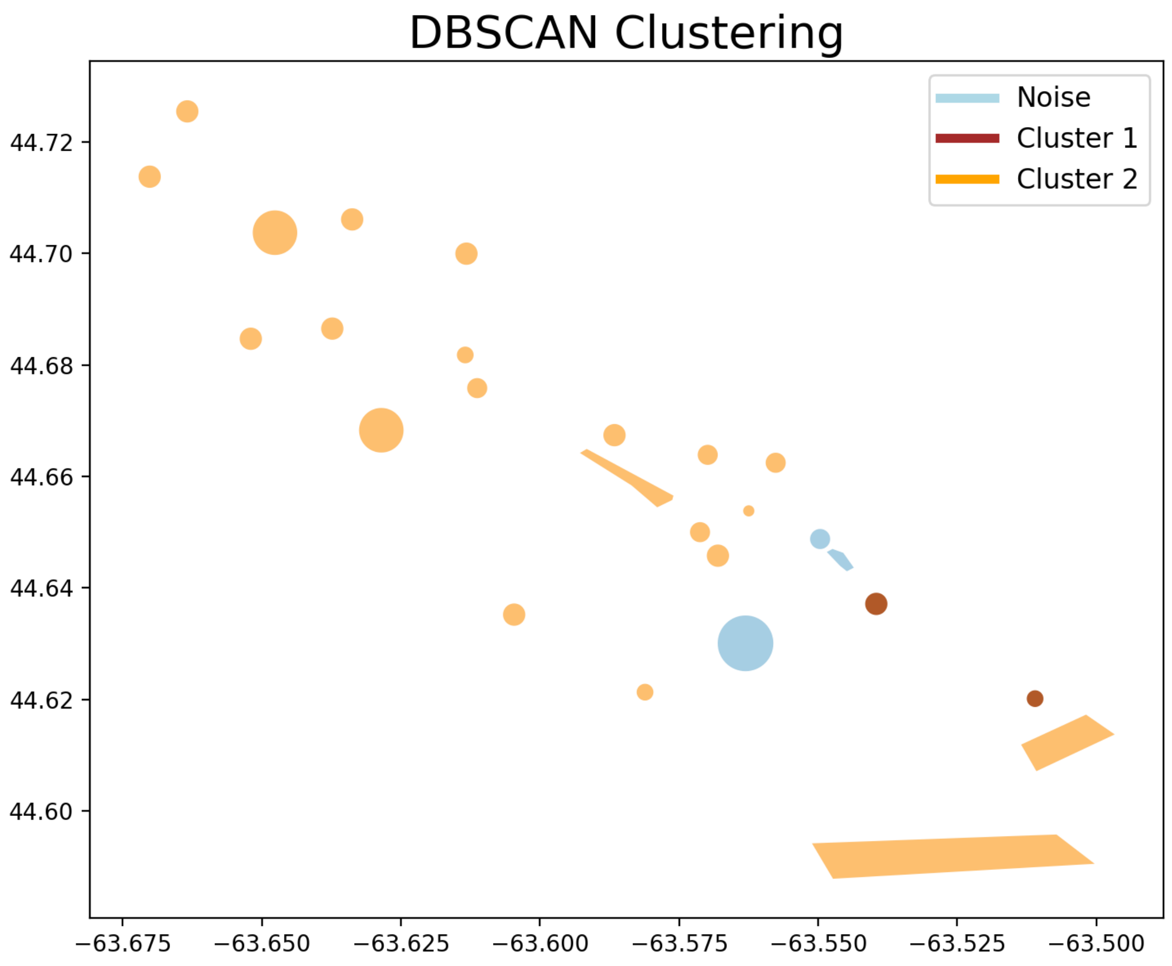
```
Out[141]:
```

	PortName	Count	Density	geometry	DBSCANlabel
0	port1	80	0.000104	POLYGON ((-63.569431 44.649993, -63.5694396674...	0
1	po002	2	0.000003	POLYGON ((-63.611943 44.681805, -63.6119502229...	0
2	NN Jetty	0	0.000000	POLYGON ((-63.631767 44.706138, -63.6317766305...	0
3	pp	0	0.000000	POLYGON ((-63.66140799999999 44.725527, -63.66...	0
4	p009	0	0.000000	POLYGON ((-63.64999499999999 44.684713, -63.65...	0
5	enter2	0	0.000000	POLYGON ((-63.50183486938477 44.61723320107161...	0
6	port2	229	0.000299	POLYGON ((-63.60949000000001 44.675853, -63.60...	0
7	southend container terminal	8722	0.011376	POLYGON ((-63.55805200000001 44.63004, -63.558...	-1
8	port6	0	0.000000	POLYGON ((-63.555828 44.662453, -63.5558366674...	0
9	plll	0	0.000000	POLYGON ((-63.579618 44.621272, -63.5796252229...	0
10	armament	655	0.000854	POLYGON ((-63.584608 44.667408, -63.5846176305...	0
11	po001	2289	0.002986	POLYGON ((-63.537543 44.637118, -63.5375526305...	1
12	ind	4504	0.005875	POLYGON ((-63.54742169380188 44.64697911403847...	-1
13	p003	0	0.000000	POLYGON ((-63.668168 44.713808, -63.6681776305...	0
14	auto_port	2215	0.002889	POLYGON ((-63.50949499999999 44.62011, -63.509...	1
15	pointpolygon	1094	0.001427	POLYGON ((-63.59160304069519 44.6649292254607,...	0
16	Fairview cove	1552	0.002024	POLYGON ((-63.62453000000001 44.668288, -63.62...	0
17	south_enterance	44	0.000057	POLYGON ((-63.50715637207031 44.59572358282151...	0
18	mid bedford	0	0.000000	POLYGON ((-63.64364799999999 44.703755, -63.64...	0
19	port7	704985	0.919540	POLYGON ((-63.547843 44.648763, -63.5478516674...	-1
20	p010	14	0.000018	POLYGON ((-63.63534300000002 44.686538, -63.63...	0
21	Bills island	0	0.000000	POLYGON ((-63.611225 44.699988, -63.6112346305...	0
22	northarm	0	0.000000	POLYGON ((-63.602647 44.635192, -63.6026566305...	0

	PortName	Count	Density	geometry	DBSCANlabel
23	oulier_maybecday	995	0.001298	POLYGON ((-63.56147 44.653817, -63.56147481527...	0
24	waterfront h	448	0.000584	POLYGON ((-63.56601499999999 44.64577199999999...	0
25	port5	0	0.000000	POLYGON ((-63.568048 44.663875, -63.5680566674...	0

Lets plot these clusters:

```
In [140]: fig, ax = plt.subplots(1, figsize= (10,10))
          densityDataframe.plot(ax = ax, column = 'DBSCANlabel', cmap='Paired')
          ax.set_title("DBSCAN Clustering", fontdict={'fontsize': 20, 'fontweight'
          : 'medium'})
          legend_elements = [
                                 Line2D([0], [0], color='lightblue', lw=4, label='Nois
          e'),
                                 Line2D([0], [0], color='brown', lw=4, label='Cluster
           1'),
                                 Line2D([0], [0], color='orange', lw=4, label='Cluster
          2')
                                 ]

          ax.legend(handles=legend_elements,prop=dict(size=12))
```



```
Out[140]: <matplotlib.legend.Legend at 0x13f89f198>
```

The above plot shows the clusters created using DBSCAN based on the AIS message density. The ports have been classified into 2 categories as shown in figure.

3 of the ports have been classified as noise.

References

1. skmultiflow.drift_detection.adwin module — scikit-multiflow 0.1.4 documentation. (2019). Retrieved 5 August 2019, from https://scikit-multiflow.github.io/scikit-multiflow/skmultiflow.drift_detection.adwin.html (https://scikit-multiflow.github.io/scikit-multiflow/skmultiflow.drift_detection.adwin.html)
2. Matplotlib: Python plotting — Matplotlib 3.1.1 documentation. (2019). Retrieved 5 August 2019, from https://matplotlib.org/ (https://matplotlib.org/)
3. GeoPandas 0.5.1 — GeoPandas 0.5.1 documentation. (2019). Retrieved 5 August 2019, from http://geopandas.org/ (http://geopandas.org/)
4. Python Data Analysis Library — pandas: Python Data Analysis Library. (2019). Retrieved 5 August 2019, from https://pandas.pydata.org/ (https://pandas.pydata.org/)