Algorithm 1 Functions for Algorithm

function POLYGONRADIUS(coordinate, distance)
    Generate Polygon out of coordinate point
    return polygon with radius of distance
end function
function TIMERANGE(timestamp1,timestamp2)
    Calculate the difference between timestamps
    return hours range
end function
function SEQUENCETEST(iterator)
    Detect separate calls in same port. Test if sequence exist and assign unique number to sequence of True values between False values
    return set of unique sequence group values
end function
function TIME-POSITIONINTERPOLATION(Dataframe)
    Detects discontinuity in the expected sequence of timestamps and interpolates the empty positions from the previous and next available positions
    Parameters:
    timerange←10. Minutes, starting from the first position in the dataframe
    return adjusted dataframe with interpolated positions
end function
function DBSCAN(positions iterator)
    Creates clusters based on DBSCAN. -1 is assigned to noise positions and a unique number to every cluster
    Parameters:
    minPts←3.Minimum number of positions to generate a cluster
    ϵ←0.00195. Decimal degrees maximum distance. From 0.68 kts upper bound or 0.117NM allowance for 10 minutes movement.
    return set of either unique clusterID or noise value
end function
function VESSELPOLYGONGENERATOR(positions iterator)
    Generates linestrings from the sequence of ordered positions (lon,lat). Then creates a polygon as the radius buffered from the linestring
    Parameters:
    radius←0.00045. Decimal degrees, Radius of 50 meters away from linestring
    return ocean-going vessel polygon
end function
function MERGESERVICE(dataframe)
    Shift method that calculates the time difference between the last timestamp of a service ID and the first timestamp of the next service ID. If the difference is less than a threshold then the services merges and the first group service ID is absorbed by the following group
    Parameters:
    time←24. Hours, considers a barge refueling cargo in the middle of the bunkering operation. The refueling of the barge keeps counting as service as the vessel has to wait.
    return dataframe with merged services
end function
function MERGEDATAFRAMES(dataframe1,dataframe2,leftIndex,rightIndex)
    Merges dataframes by leftIndex=rigtIndex
    Parameters:
    return merged dataframe with algorithm results and vessels characteristics. Named as dataframe1.
end function

Algorithm 2 Bunker barge prospects recognition

Inputs:
A dataframe with columns [portName,coordinate,berthPolygon], Ports;
A dataframe with columns [vesselsID,LengthOverAll,TypeOfVessel], VesselsSpecs;
A dataframe with columns [vesselID,positions,time]; AIS
An empty dataframe with columns [bargeID,portOfService]; BargesInfo
Default Parameters:
loa ←150, meters of length overall of ship
localThreshold ←0.70, minimum stay at local waters
minimumObs ←100, minimum observations to have a valid sample
set polygonGeometry as new column in Ports
for coordinate in Ports **do**
    polygon ← **call** POLYGONRADIUS(coordinate, polygon)
    store polygon to Ports.polygonGeometry
end for
for polygon in Ports.polygonGeometry **do**
    for testPolygon in Ports.polygonGeometry **do**
        if polygon overlaps testPolygon **then**
            newPolygon ← **merge** polygon and testPolygon
            remove polygon and testPolygon
            store newPolygon to Ports.polygonGeometry
        end if
    end for
end for
set inLocalWaters as new column in AIS
for vesselID in VesselsSpecs **do**
    if vesselID in VesselsSpecs.LengthOverAll $\leq$ loa AND VesselsSpecs.TypeOfVessel=tanker **then**
        filter position of vesselID in AIS
        for polygon in Ports **do**
            for position in AIS **do**
                bool,polygonName ← **call** POINTINPOLYGON(position,polygonGeometry)
                if bool = TRUE **then**
                    store [TRUE to AIS.inLocalWaters ; polygonName to AIS.portVisited]
                else
                    store [FALSE to AIS.inLocalWaters ; NULL to AIS.portVisited]
                end if
            end for
            totalRows ← **count** total rows in AIS
            filter positions in AIS WHERE AIS.inLocalWaters = TRUE
            totalFilteredRows ← **count** total filtered rows in AIS
            if totalRows$\geq$ minimumObs AND totalFilteredRows/totalRows $\geq$ localThreshold **then**
                store [vesselID in BargeInfo.bargeID ; AIS.portVisited in BargeInfo.portOfService]
            end if
        end for
    end if
end for
return BargesInfo

Algorithm 3 Stopped vessel

Inputs:
A dataframe with columns [portName,coordinate,polygonGeometry,berthPolygon], Ports;
A dataframe with columns [vesselID,positions,time,speed], AIS;
An empty dataframe with columns[vesselID, initialStopTime, finalStopTime, atBerth, callID, clusterID, stopPosition, portVisited, stopID], Stops;
A set of
Default Parameters:
maxSpeed ← 3, knots of vessel entering or leaving cluster
minStay ← 1, hour. Minimum stay to be classified as stopped.
laidUp ← 1084, hours. Laid-up vessel with more than 45 days in port
noisePts ← -1. Assigment by DBSCAN to noise positions(not clustered positions)
listOfVessels← set of unique values of AIS.vesselID
listOfCallID ← empty set
listOfClusterID ← empty set
set inLocalWaters, callID, clusterID, portVisited as new columns in AIS
set counter to 0
for uniqueVesselID in listOfVessels **do**
  filter AIS WHERE AIS.vesselID=uniqueVesselID. Sort ascending by AIS.time
  for polygonGeometry in Ports **do**
    for positions in AIS **do**
      bool, polygonName ← **call** POINTINPOLYGON(positions,polygonGeometry)
      if bool=TRUE **then**
        store [TRUE in AIS.inLocalWaters ; polygonName in AIS.portVisited]
      else
        store [FALSE in AIS.inLocalWaters ; NULL in AIS.portVisited]
      end if
    end for
  end for
  store [**call** SEQUENCEGENERATOR(AIS.inLocalWaters)] in AIS.callID
  filter AIS WHERE AIS.inLocalWaters=TRUE AND AIS.speed≤maxSpeed
  store unique values of AIS.callID in listOfCallID
  for uniqueCallID in listOfCallID **do**
    filter AIS WHERE AIS.callID=uniqueCallID
    timeDifference ← **call** TIMERANGE(AIS.time[-1], AIS.time[0])
    if timeDifference≥minStay AND timeDifference≤laidUp **then**
      call TIME-POSITIONINTERPOLATION(AIS)
      store [**call** DBSCAN(AIS.positions)] in AIS.clusterID
      filter AIS WHERE AIS.clusterID≠noisePts
      store unique values of AIS.clusterID in listOfClusterID
      for uniqueClusterID in listOfClusterID **do**
        add 1 to counter
        filter AIS WHERE AIS.clusterID=uniqueClusterID
        store [uniqueVesselID in Stops.vesselID ; AIS.time[0] in Stops.initialStopTime ; AIS.time[-1] in Stops.finalStopTime ; uniqueClusterID in Stops.clusterID; uniqueCallID in Stops.callID ; AIS.portVisited[0] in Stops.portVisited ; uniqueCallID in Stops.callID ; AIS.positions[0] in Stops.stopPosition, counter in Stops.stopID]
        for berthPolygon in Ports **do**
          bool, ← **call** POINTINPOLYGON(AIS.positions[0],berthPolygon)
          if bool=TRUE **then**
            store TRUE in Stops.atBerth
          else
            store FALSE in Stops.atBerth
          end if
        end for
      end for
    end if
  end for
end for
return Stops

Algorithm 4 Bunkering recognition

Inputs:
A dataframe with columns [vesselID,positions,time], AIS;
A dataframe with columns[vesselID, initialStopTime, finalStopTime, atBerth, callID, clusterID, stopPosition, portVisited, stopID], Stops;
A dataframe with columns[bargeID,portOfService]; BargesInfo;
A dataframe with columns [vesselID,grossTonnage,TypeOfVessel,yearOfBuilt], VesselsSpecs;
An empty dataframe with columns [vesselID,positions,time], VesselAIS;
An empty dataframe with columns [vesselID,positions,time, alongsideVessel, serviceID], BargeAIS;
An empty dataframe with columns [bargeID, initialServiceTime, finalServiceTime, vesselID, initialStopTime, finalStopTime, atBerth, callID, clusterID, stopPosition, portVisited, stopID], Results;
Default Parameters:
minStay ← 1, hour. Minimum stay of barge inside ocean-going vessel polygon
listOfStops ← set of values of AIS.stopID
listOfBarges ← empty set
listOfServices ← empty set
for uniqueStop in listOfStops **do**
    filter Stops.stopID=uniqueStop
    VesselAIS←**filter** AIS.vesselID=Stops.vesselID AND AIS.time≥Stops.initialStopTime AND AIS.time≤Stops.finalStopTime
    vesselPolygon ← **call** VESSELPOLYGONGENERATOR(VesselAIS.positions)
    filter BargesInfo.portOfService=Stops.portVisited
    store unique values of BargesInfo.bargeID in listOfBarges
    for uniqueBargeID in listOfBarges **do**
        BargeAIS←**filter** AIS.vesselID=uniqueBargeID AND AIS.time≥Stops.initialStopTime AND AIS.time≤Stops.finalStopTime
        for positions in BargeAIS.positions **do**
            bool, ← **call** POINTINPOLYGON(positions,vesselPolygon)
            if bool=TRUE **then**
                store TRUE in BargeAIS.alongsideVessel
            else
                store FALSE in BargeAIS.alongsideVessel
            end if
        end for
        store [**call** SEQUENCEGENERATOR(BargeAIS.alongsideVessel)] in BargeAIS.serviceID
        filter BargeAIS WHERE BargeAIS.alongsideVessel=TRUE
        call MERGESERVICE(BargeAIS)
        store unique values of BargeAIS.serviceID in listOfServices
        for service in listOfServices **do**
            filter BargeAIS.serviceID=service
            timeDifference ← **call** TIMERANGE(BargeAIS.time[-1], BargeAIS.time[0])
            if timeDifference≥minStay AND BargeAIS.vesselID≠Stops.vesselID **then**
                store [BargeAIS.vesselID in Results.bargeID ; BargeAIS.time[0] in Results.initialServiceTime ; Barge AIS.time[-1] in Results.finalServiceTime ; Stops.vesselID in Results.vesselID ; Stops.initialStopTime in Results.initialStopTime ; Stops.finalStopTime in Results.finalStopTime ; Stops.atBerth in Results.atBerth ; Stops.callID in Results.callID ; Stops.clusterID in Results.clusterID ; Stops.stopPosition in Results.stopPosition ; Stops.portVisited in Results.portVisited ; uniqueStop in Results.stopID]
            end if
        end for
    end for
end for
call MERGEDATAFRAME(Results,VesselsSpecs,vesselID,vesselID)
call MERGEDATAFRAME(Results,VesselsSpecs,bargeID,vesselID)
return Results