# Home Range Analysis for 20 Mink in Illinois Using Minimum Convex Polygons

**by Hannah White, whit2973@umn.edu (mailto:whit2973@umn.edu)**
*Geocomputing, Spring 2019*

Professor -- Dr. Eric Shook
TA -- Erik Thorkelson

American Mink http://www.havahart.com/minks-facts/ (http://www.havahart.com/minks-facts/)

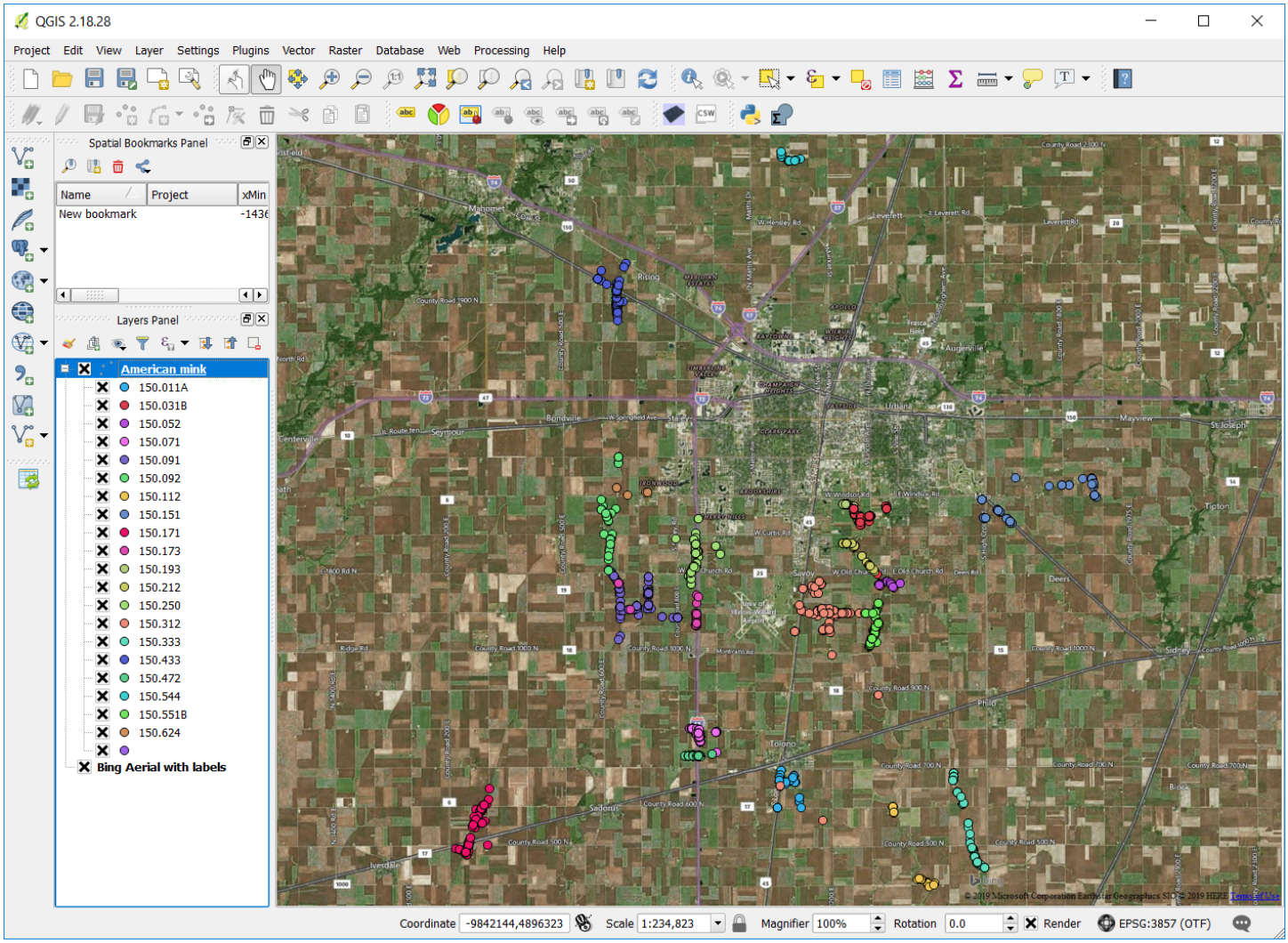
## Introduction

The American Mink (*Neovison vison*) is a solitary species, spending most of its time in or near aquatic habitat. In central Illinois, where the land cover has largely been converted to agriculture, mink persist in the drainage ditches between crop fields. As such, their home ranges tend to be linear and distinct from one another, with little overlap among individuals.

Downloaded from Movebank, which is an online repository housing open-source wildlife tracking data, the dataset for this analysis contains a series of coordinate locations for 20 radio-tagged mink. The screenshot below provides an overview of the shapefile in QGIS, from which the attribute table was exported as a csv for the following analysis.

**Aim of Analysis:**

1. Calculate the size of each home range.
2. Display individual home ranges as minimum convex polygons ('convex hulls').
3. Calculate summary statistics for all home ranges.
4. Visualize the distribution of home range sizes with a boxplot.




## Step 1. Import All Modules Needed for Analysis

In [1]:
```
import pprint

import pandas as pd
from pandas import DataFrame

import folium

import scipy
from scipy.spatial import ConvexHull

import numpy as np

import matplotlib.pyplot as plt
```

## Step 2. Visualize the Dataset

In [2]:

```
# read in the csv file as a pandas dataframe
minkdf = pd.read_csv("E:\MGIS\Python\Indiv Project\minklocs_UTM.csv")

# preview the data (column names and first 5 rows)
print(minkdf.head())


# determine the central point of the data, which will be used to center the folium map
centerloc = (minkdf['latitude'].mean(), minkdf['longitude'].mean())

# background satellite imagery for the folium map
tileURL = 'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}'

# create folium map centered over Champaign-Urbana, Illinois
minkmap = folium.Map(location=centerloc, tiles=tileURL, attr= 'Esri World Imagery')


# each mink has a unique tagID
# create a list of all mink locations and their associated tagID
locations = minkdf[["latitude", "longitude", "tagID"]]
locationlist = locations.values.tolist()


# define function to assign each mink its own color on the folium map
def prettymink(uniqID):
    # create list of the 20 unique tagIDs
    tagIDs = minkdf['tagID'].unique()
    # create list of 20 color options compatible with folium
    coloropts= ['darkgreen', 'blue', 'green', 'purple', 'lightred',
                'orange', 'darkred', 'beige', 'darkblue', 'red',
                'cadetblue', 'darkpurple', 'white', 'pink', 'lightblue',
                'lightgreen', 'gray', 'red', 'black', 'lightgray']
    # merge the lists together such that [[tagID1, color], [tagID2, color], ... [tagID20, color]]
    minkcolors = np.column_stack((tagIDs, coloropts))

    # iterate through each pair in the minkcolors array
    for index in minkcolors:
        # check to see if the user's input (uniqID) matches the current tagID
        if uniqID == index[0]:
            # when it finds the matching tagID, output the associated color
            return index[1]

# add all mink locations to the map
# point[0] is latitude, point[1] is longitude, point[2] is tagID
# icon color is specified by the prettymink function above
# icon 'none' just means no symbol on the map pin
for point in locationlist:
    folium.Marker((point[0], point[1]), icon=folium.Icon(color=prettymink(point[2]), icon='none')).add_to(minkmap)

# I tried adding popups or tooltips for each marker to show tagID, but it made the output render blank with no error
# for example, tooltip = point[2] or popup = 'test' (the code runs, but shows nothing)

minkmap
```

```
      eventID timestamp  longitude   latitude    tagID  assignedID  easting  \
0  1006443547   15:00.0 -88.343113  40.156697  150.624           1   385613
1  1006443548   00:00.0 -88.343113  40.156697  150.624           1   385613
2  1006443549   41:00.0 -88.343113  40.156697  150.624           1   385613
3  1006443550   05:00.0 -88.343113  40.156697  150.624           1   385613
4  1006443551   01:00.0 -88.343113  40.156697  150.624           1   385613

   northing
0   4446014
1   4446014
2   4446014
3   4446014
4   4446014
```

Out[2]:


### Step 3. Get a Feel for the Convex Hull Function

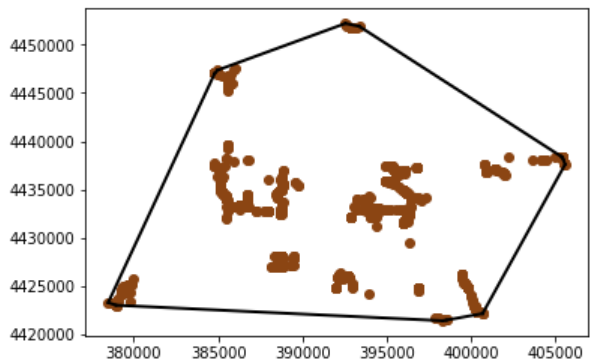
Let's start by plotting all the data points & testing out capabilities of the convex hull function.

The csv we're working with is an exported attribute table from a point shapefile of mink locations.

Since we are interested in calculating area, we will use UTMs instead of lat/long.

In [3]:
```
# create a numpy array of coordinate pairs using UTM eastings and northings from csv
# skip the first row containing headers
minklocs = np.loadtxt("E:\MGIS\Python\Indiv Project\minklocs_UTM.csv", delimiter=',', usecols=(6,7), skiprows=1)

# get a glimpse of the array format
print("minklocs:\n", minklocs)

# define a minimum convex polygon around all points in the minklocs array
hull = ConvexHull(minklocs)

# UTM units are meters, so the calculated area will be m^2
print("Hull Area =", round(hull.area, 2), "square meters")

# plot the points
# the colon placeholder means use all indices, in this case "for all sets of points in the array"
# the first brackets specify the x coord using the 0th index
# the second brackets specify the y coord using the 1th index
# "o" specifies a circle marker
plt.plot(minklocs[:,0], minklocs[:,1], "o", color="saddlebrown")

# print to better understand the structure of vertices & simplices
print("Hull vertices:", hull.vertices)
print("Hull simplices:\n", hull.simplices)

# plot the minimum convex polygon
# simplices are similar to vertices
# vertices are all the points forming the convex hull
# with our ConvexHull function, vertices are stored as indices referencing specific points in the minklocs array
# in a similar fashion, simplices reference the same hull points as start-end points of the hull's edge segments
# so each simplex has 2 indices for the start & end points
# "lw" specifies line weight
for simplex in hull.simplices:
    plt.plot(minklocs[simplex, 0], minklocs[simplex, 1], color="black", lw=2)
```

```
minklocs:
 [[ 385613. 4446014.]
 [ 385613. 4446014.]
 [ 385613. 4446014.]
 ...
 [ 388728. 4432618.]
 [ 388726. 4432571.]
 [ 388822. 4433676.]]
Hull Area = 92285.68 square meters
Hull vertices: [ 892  600  599  955  948 1272 1273  192  171  361  357  381  376  876]
Hull simplices:
 [[ 600  892]
 [ 876  892]
 [ 171  361]
 [ 171  192]
 [ 599  955]
 [ 599  600]
 [ 376  876]
 [ 948  955]
 [ 948 1272]
 [1273  192]
 [1273 1272]
 [ 357  361]
 [ 381  376]
 [ 381  357]]
```




## Step 4. Group the Point Locations for Each Mink

We will use a dictionary to hold the data, with 20 keys (1 for each individual mink). Each key will have a list of values representing coordinate pairs.
**dictionary = {mink1 : (easting, northing), (easting northing), (etc) ... mink20 : (easting, northing), (easting, northing), (etc)}**

In [4]:
```
# as a refresher, print the column names
# coordinates are in UTMs, stored as 'easting' and 'northing' pairs
# each of the 20 mink in this dataset has a unique identifier, stored under 'tagID'
print("COLUMN NAMES:", minkdf.columns)

# initiate a dictionary for mink locations grouped by unique tagID's
# end-goal will be dict = {tagID : (easting, northing), ...}
minkdict = {}

# for loop to populate dictionary of mink locations
# "for each unique tagID, add all corresponding eastings/northings into a list"
# format will be dict = {tagID : (easting, northing), ...}
for tagID in minkdf['tagID'].unique():
        minkdict[tagID] = [(minkdf['easting'][j], minkdf['northing'][j]) for j in minkdf[minkdf['tagID']==tagID].index]

pprint.pprint(minkdict)
```

```
COLUMN NAMES: Index(['eventID', 'timestamp', 'longitude', 'latitude', 'tagID', 'assignedID',
       'easting', 'northing'],
      dtype='object')
{'150.011A': [(392144, 4425728),
              (392144, 4425728),
              (392022, 4424819),
              (392144, 4425728),
              (392019, 4424820),
              (392019, 4424820),
              (392019, 4424820),
              (392145, 4425731),
              (392145, 4425731),
              (392145, 4425731),
              (392019, 4424820),
              (392145, 4425731),
              (392164, 4426092),
              (392019, 4424820),
              (392164, 4426092),
              (392164, 4426092),
```

## Step 5. Calculate Home Range Estimates

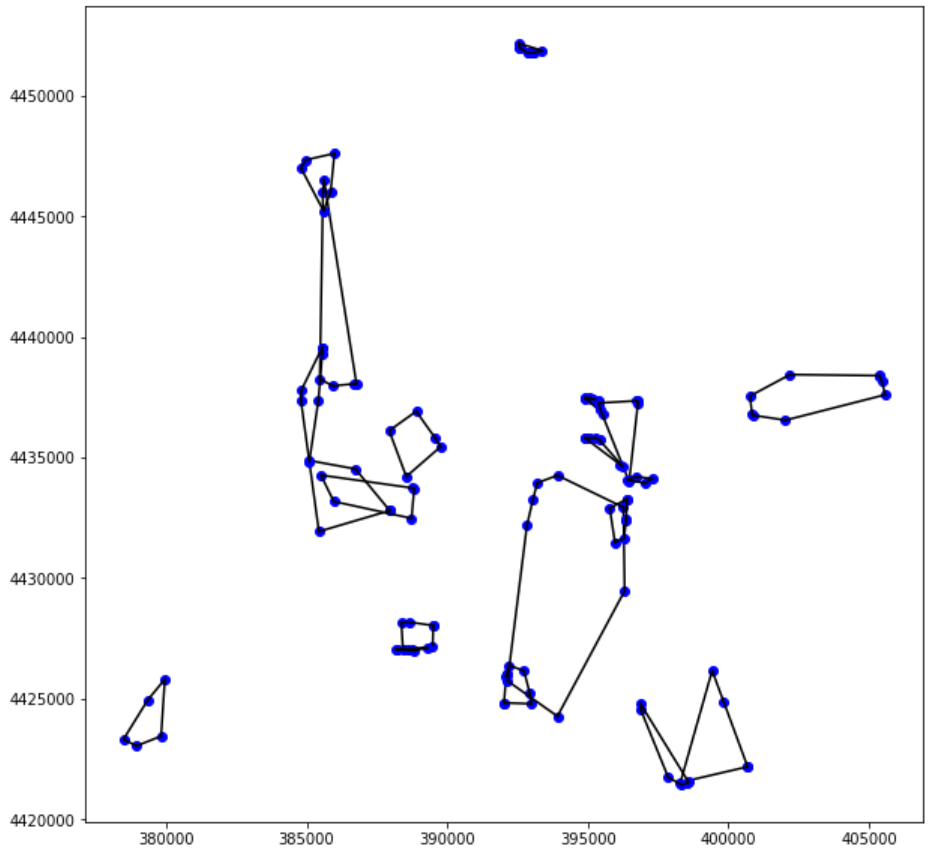
Again, we will use a dictionary to hold the data, such that **dictionary = {mink1 : home range, ... mink20 : home range}**

In [5]:
```
# calculate an estimate of home range size for each mink
# home range size will be the area, in square meters, of each convex hull
# end-goal will be homeranges = {'tagID' : <area>, ...}
homeranges = {}

# populate dictionary with a home range estimate for each of the 20 mink
# format will be homeranges = {'tagID' : <area>, ...}
for key in minkdict:
    thehull = ConvexHull(minkdict[key])
    homeranges[key] = thehull.area

pprint.pprint(homeranges)
```

```
{'150.011A': 4511.575260739793,
 '150.031B': 8168.685321584607,
 '150.052': 1875.6903949300663,
 '150.071': 4176.124734708104,
 '150.091': 9411.575233908967,
 '150.092': 9649.497872849004,
 '150.112': 7707.909325019264,
 '150.151': 11365.01968989633,
 '150.171': 6760.844247320943,
 '150.173': 8628.727223939803,
 '150.193': 1948.9625636989656,
 '150.212': 3711.9007087495715,
 '150.25': 6690.970115401573,
 '150.312': 23258.36718270441,
 '150.333': 11431.731818636135,
 '150.433': 5818.126811752945,
 '150.472': 2260.7692447885333,
 '150.544': 1927.6841931101812,
 '150.551B': 4135.16915118299,
 '150.624': 18195.601121203978}
```

## Step 6. Display the Individual Home Ranges

In [7]:
```
# for each unique mink, create a convex hull using XY coords for that mink
for key in minkdict:
    thehull = ConvexHull(minkdict[key])

    # plot the vertices of each home range hull
    # 'bo' specifies blue circle markers
    for vertex in thehull.vertices:
        plt.plot(minkdict[key][vertex][0], minkdict[key][vertex][1], 'bo')

    # plot the edges of each home range hull
    # 'k-' specifies black line symbology
    # had to cast each dictionary value set from a list to a numpy array
    # otherwise I would get an error saying "list indices must be integers not tuples"
    # this is because a simplex has two indices, and each coordinate pair in the dictionary is a tuple
    for simplex in thehull.simplices:
        plt.plot(np.asarray(minkdict[key])[simplex, 0], np.asarray(minkdict[key])[simplex, 1], 'k-')

# increase figure size (the default is small, like we saw in Step 3 above)
plt.rcParams["figure.figsize"] = (10,10)
# show all vertices and simplices comprising each convex hull (home range)
plt.show()
```




### Step 7. Calculate Summary Statistics

Of the 20 home ranges, what was the average, minimum, & maximum home range size.

In [8]:
```
# read in the 'homeranges' dictionary as a pandas dataframe
# 'orient by index' assigns the keys to rows rather than columns
# the columns parameter is just a string of what we'd like the header to say
homerangestats = pd.DataFrame.from_dict(homeranges, orient='index', columns=['Home Ranges (sq m)'])

# calculate the summary stats and print each accordingly
print("MEAN = ", homerangestats.mean(axis=0))
print("MIN = ", homerangestats.min(axis=0))
print("MAX = ", homerangestats.max(axis=0))

# display the dataframe
display(homerangestats)
```

| | Home Ranges (sq m) |
|---|---|
| **150.624** | 18195.601121 |
| **150.333** | 11431.731819 |
| **150.071** | 4176.124735 |
| **150.151** | 11365.019690 |
| **150.433** | 5818.126812 |
| **150.091** | 9411.575234 |
| **150.25** | 6690.970115 |
| **150.544** | 1927.684193 |
| **150.011A** | 4511.575261 |
| **150.171** | 6760.844247 |
| **150.312** | 23258.367183 |

## Step 8. Visualize Home Range Distribution

Using a box plot, we can see that--aside from 2 outliers--the data is pretty normally distributed.

The 2 outliers are likely errors resulting from the method we used (convex hull) to estimate home range sizes. It only takes one point to greatly inflate home range estimates using this method. Minimum convex polygons work best when points are centralized in one area. It does not work well for irregular home ranges, which may have been the case with these 2 mink following agricultural drainage ditches.

In [10]:
```
# create a boxplot
homerangestats.boxplot(column='Home Ranges (sq m)', fontsize=15)
```