ER131 Final Project Template

Fall 2023

Predicting Sea Turtle Nest Success Rates and Relocation on Cumberland Island, GA

Fister, Alex: Background section, Accessing nesting data, searching for data, Classification, EDA, interpretation and conclusion for classification ~ 8-10 hours (a lot of the nesting data and background info were things I had already from my thesis) Menting, Joshua: Searching for data, writing abstract, project objectives, presentation slides, help on modeling and debugging ~ 9 hours Rosal Calit, Gabriella: Searching for data, Cleaning (especially matching lon and lat), EDA, Help on modeling and debugging ~ 7-10 hours Weinberger, Benjamin: Searching for data, cleaning, organizing the notebook, Regression, EDA, data description, interpretation and conclusion for regression ~ 8-12 hours (not sure how long i spent googling sources)

Abstract (5 points)

Turtle nest-site erosion could influence sea turtle egg and hatchling survival as nesting habitats include beaches on barrier islands, which undergo extreme erosion/accretion at different rates at different times of the year. This could potentially submerge nest sites. Washover events can increase egg mortality by saltwater inundation. Also, temperatures rising from climate change can cause feminization of hatchlings. A solution is nest relocation. The goal, then, is to predict success rates for nests where this data is missing and predict whether these nests would require relocation.

The primary prediction problem was approached by looking at three target variables: hatch success rate (corrected) which was modeled with regression analysis, emergence success rate (corrected) which was also modeled with regression analysis, and relocation which was analyzed using a classification model. The features included erosion, temperature, washovers, date, time, location, and relocation. 4 data sets were cleaned. Regression models included OLS, ridge, and lasso, and classification models included logistic regression, KKN, and decision trees. MSE from k-fold cross-validation and the classification validation score were used for measures of test error.

In all of our regression models, the RMSE found from cross validation was substantial, and adjusting the hyperparameters did not improve the model's prediction. Additionally, our predictions for nests where there was no recorded emergence were unrealistic in their range. In classification we were able to build models that scored above 50% on the validation scores, but these models likely would need more data on distance from nearest predators to be more effective at predicting relocation.

Project Background (5 points)

Green turtles (Chelonia mydas), loggerhead turtles (Caretta caretta) and leatherback turtles (Dermochelys coriacea) nest on US Atlantic coast beaches (Carroll et al. 2022). Turtles hatch on these beaches and inhabit ocean ecosystems until they reach sexual maturity. Nesting sea turtles return to their natal beaches to lay eggs in the same nest-sites they were born in (Varela et al. 2019).

Nest-site erosion could influence sea turtle egg and hatchling survival. Sea turtle nesting habitats include beaches on barrier islands. These areas are constantly undergoing extreme erosion and accretion over a given year, causing dramatic changes in shoreline composition (Lamont and Carthy, 2007). The dynamic nature of these beaches are a challenge for the stability of sea turtle nesting sites. Erosional shorelines have decreased habitat availability at different times of the year (Lamont and Carthy, 2007), which could submerge historical nesting sites. Cumberland Island, Little St. Simons Island, and St. Catherines Island are all barrier islands that include sea turtle nesting sites.

When beaches are flooded, nest washover events could increase egg mortality. Saltwater inundation for prolonged periods of time has been found to decrease egg and hatchling survival rates in all stages of embryonic development (Pike et al. 2015). Islands with severe rates of erosion place nest-sites at a greater risk for washover events.

Light pollution and temperature are other factors influencing survival. Light pollution is positively correlated with coastal urbanization (such as beach houses and restaurants) (Hu et al. 2018). Unfortunately we were unable to find a light pollution dataset that we could feasibly convert into a csv file to analyze, so this was left out of our models. Sea turtle hatchlings rely on moonlight to guide them to the ocean, so anthropogenic light can disorient hatchlings and lead to mortality (Bourgeois et al. 2009). Temperatures of the air and sand determine the sex of sea turtle eggs, so nesting in a warmer climate could cause widespread feminization of hatchlings, or even mortality (Hamann et al. 2007).

A mitigation strategy for egg mortality caused by beach erosion is nest relocation. Sea turtle nest relocation involves moving nests to stretches of beach that are not at risk for flooding (Pfaller et al. 2008). This management strategy is beneficial because it places nests in habitats more favorable for egg and hatchling survival, which nesting sea turtles cannot find using only natal knowledge. Potential drawbacks of nest relocation are mortality from human handling and

```
hotter sand temperature further inland. Sea turtle nesting sites are
categorized as in situ or relocated.


Quantitative analysis would be useful for predicting hatch and
emergence success rates for nests where these data are missing, and
whether these nests require relocation. Data about erosion, washovers,
and temperature can be used to identify sites with high risks of egg
and hatchling mortality. This information could help predict whether
or not a nest will need to be relocated, and which nest-sites can
ensure the highest possible rates of emergence and hatch success.
These predictions could be used as a framework for nest relocation
management strategies.
```

Varela, MR. Patrício, AR. Anderson, K. 2019. Assessing climate change associated sea-level rise impacts on sea turtle nesting beaches using drones, photogrammetry and a novel GPS system. Glob Change Biol. 2019; 25: 753– 762.

Carroll, J. M. Whitesell, M. J. Hunter, E. A. Rostal, D. C. 2022. First time's a charm? Loggerhead neophyte mothers have higher hatch success. Southeastern Naturalist. 21(4): 291-298

Pike, D. Roznik, E. Bell, I. 2015. Nest inundation from sea-level rise threatens sea turtle population viability. R. Soc. open sci., volume 2: 150127 Lamont, M. M. Carthy, R. R. 2007. Response of nesting sea turtles to barrier island dynamics. Chelonian Conservation and Biology, 6(2): 206-212

Bourgeois, E. Gilot-Fromont, A. Viallefont, F. Boussamba, S.L. Deem. 2009. Influence of artificial lights, logs and erosion on leatherback sea turtle hatchling orientation at Pongara National Park. Gabon. Biol. Conser., 142, pp. 85-93

Hu, Z., H. Hu, and Y. Huang. 2018. Association between nighttime artificial light pollution and sea turtle nest density along Florida coast: A geospatial study using VIIRS remote sensing data. Environmental Pollution 239:30–42.

Hamann, M. Limpus, C.J. Read, M.A. 2007. Vulnerability of marine reptiles in the Great Barrier Reef to climate change. J. Johnson, P. Marshall (Eds.), Climate Change and The Great Barrier Reef: A Vulnerability Assessment, Great Barrier Reef Marine Park Authority and Australian Greenhouse Office, Townsville, pp. 667-716

Pfaller JB, Limpus CJ, Bjorndal KA. Nest-site selection in individual loggerhead turtles and consequences for doomed-egg relocation. Conserv Biol. 2009 Feb;23(1):72-80. doi: 10.1111/j.1523-1739.2008.01055.x. Epub 2008 Sep 15. PMID: 18798862.

Project Objective (5 points)

Our first objective in this project is to predict both hatch and emergence success rates for nest observations where emergence was not recorded. This will allow for a better understanding of the typical patterns of hatching and emergence and how they relate to our potential feature variables (this part relates more to inferences). Our second objective is to predict the probability of relocation for nests where there was no opportunity to relocate. This would be a helpful model to aid conservation efforts by defining criteria or relying on modeling to identify nests that should be relocated in order to improve success rates.

Data Description (5 points)

The data on sea turtle nesting on Cumberland Island is collected by the Georgia Department of Natural Resources (GDNR) Sea Turtle Conservation Program. We received somewhat cleaned data from GDNR contacts for three islands in the State of Georgia, and we used the island with the largest number of observations and corresponding weather data. Wather data was collected from NOAA's website, which publishes data collected from weather stations across the country. Vertical land movement, which we used as a proxy for erosion rates, were collected from the USGS website and subset outside of Jupyterhub as the file prompted a warning when an upload was attempted. After the data was subset to only the area of Cumberland Island, it was sucessfully uploaded to the notebook.

1. Structure: The data all came in the form of csv files.
2. Granularity: For the GDNR turtle nesting dataset, each row represents a single nest over the nesting season, with different features of different aspects of the nest and its location, as well as its progress over time. For the monthly weather dataset, weather was aggregated at the monthly scale and each row reported weather observations for each month. For daily weather data, weather data was aggregated by the day. Vertical land movement data was a single value, aggregated from 2007 to 2021 which covers most of our nesting data.
3. Scope: Both types of weather data cover a single point at the weather station on the island. The nesting data covers most of the beach area of the island. Vertical land movement data is approximately 75 m resolution and mm-level precision along the coastline, as was subset to only include the area of the island where turtles nest.
4. Temporality: The nesting data includes dates for when nests are first recorded after being created by the turtles, as well as dates for emergence and inventory. We can see that nesting dates occur only between June and August. Monthly weather data ranges from 2003 to the present, while daily weather data was subset from 2008 to 2023 before being uploaded. Vertical land movement is aggregated over the period from 2007 to 2021, and thus has no temporal values.
5. Faithfulness: The nesting data is the most unreliable out of our datasets. We see that there are strange dates and other inconsistencies in the dataframe which may have come from human error. Somes nests are labeled as "Final Status Unknown". We will predict our target variables for these nests, as there was no data collected.

We will use the 'Relocation' column, Hatch Success Rate and Emergence Success Rate as our target variables.

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline
pd.set_option('display.max_columns', 500)
```

```
cuis = pd.read_csv('CUIS_2008_2023.csv')
dw = pd.read_csv('daily_weather_cuis_08_23.csv')
mw = pd.read_csv('USR0000GSTA.csv')
vlm = pd.read_csv('VerticalLandMotion_Rate_CUIS_GA.csv')

cuis.loc[cuis['Final Status Unknown'] == 1].head()
```

```
        UID       Beach  County  Activity # Activity  Nest # Ref #  \
437    2472  Cumberland  Camden         142        N     103   NaN
1476  63599  Cumberland  Camden        1286       UN     698   NaN
1477  63600  Cumberland  Camden        1287       UN     699   NaN
2212  67209  Cumberland  Camden         114        N     547   NaN
2350  71249  Cumberland  Camden         335        N     216   NaN

     Activity Date  Year  Month  Week  Dayofyear  JulianDate  Latitude
\
437        6/23/09  2009      6  25.0      174.0   2455005.5   30.8715

1476    2012-00-00  2012      0   NaN        NaN         NaN   30.7194

1477    2012-00-00  2012      0   NaN        NaN         NaN   30.7196

2212        6/6/13  2013      6  22.0      157.0   2456449.5   30.8630

2350       6/22/13  2013      6  24.0      173.0   2456465.5   30.7993


      Longitude Relocation  Relocation Latitude  Relocation Longitude
\
437    -81.4185    in situ                  NaN                   NaN

1476   -81.4696    in situ                  NaN                   NaN

1477   -81.4698    in situ                  NaN                   NaN

2212   -81.4210    in situ                  NaN                   NaN

2350   -81.4514    in situ                  NaN                   NaN


     Relocation Location  Washovers  Loss Reports  Prerelocations  \
437                  NaN        NaN             1               0
1476                 NaN        NaN             0               0
1477                 NaN        NaN             0               0
2212                 NaN        NaN             0               0
2350                 NaN        NaN             1               0

      Total Lost Eggs  Total Lost Hatchlings  Lost Nest Emerge Date  \
437               1.0                    NaN        NaN      8/21/09
1476              NaN                    NaN        NaN          NaN
1477              NaN                    NaN        NaN          NaN
```

```
2212              NaN                    NaN        NaN      8/9/13
2350              1.0                    NaN        NaN         NaN

     Inventory Date  Incubation (days)  Clutch Count  Shells>50%  \
437             NaN               59.0           NaN         NaN
1476            NaN                NaN           NaN         NaN
1477            NaN                NaN           NaN         NaN
2212            NaN               64.0           NaN         NaN
2350            NaN                NaN           NaN         NaN

      Unhatched Eggs  Dead Hatchlings  Live Hatchlings  \
437              NaN              NaN              NaN
1476             NaN              NaN              NaN
1477             NaN              NaN              NaN
2212             NaN              NaN              NaN
2350             NaN              NaN              NaN

      Misoriented Hatchlings  Final Status Unknown  Exclude From Calc
\
437                      NaN                   1.0                NaN

1476                     NaN                   1.0                NaN

1477                     NaN                   1.0                NaN

2212                     NaN                   1.0                NaN

2350                     NaN                   1.0                NaN


      Hatch Success  Emergence Success
437             NaN                NaN
1476            NaN                NaN
1477            NaN                NaN
2212            NaN                NaN
2350            NaN                NaN
```

cuis.describe()

```
                 UID    Activity #        Nest #          Year
Month  \
count   10292.000000  10292.000000  10292.000000  10292.000000
10292.000000
mean   202418.435095    653.756607    387.078508   2017.113098
6.053148
std    118608.301828    483.113569    280.782147      4.461502
1.126786
min       388.000000      1.000000      1.000000   2008.000000
0.000000
25%     92605.750000    262.000000    162.000000   2013.000000
6.000000
```

```
50%    224202.500000    554.000000    329.000000   2018.000000
6.000000
75%    312130.250000    946.250000    556.000000   2021.000000
7.000000
max    382668.000000   2066.000000   1254.000000   2023.000000
8.000000

               Week     Dayofyear    JulianDate      Latitude
Longitude  \
count  10095.000000  10095.000000  1.009500e+04  10287.000000
10287.000000
mean      24.646756    172.012580  2.457961e+06     30.857059    -
81.425084
std        3.008382     20.900496  1.628324e+03      0.057945
0.018945
min       16.000000    116.000000  2.454596e+06     30.713190    -
81.472280
25%       22.000000    156.000000  2.456490e+06     30.817470    -
81.442125
50%       25.000000    172.000000  2.458284e+06     30.857840    -
81.422820
75%       27.000000    187.000000  2.459392e+06     30.904760    -
81.404670
max       34.000000    240.000000  2.460166e+06     30.988600    -
81.401700

       Relocation Latitude  Relocation Longitude    Washovers  Loss
Reports  \
count          3054.000000           3054.000000  7145.000000
10292.000000
mean             30.853586            -81.426514     0.333520
1.070443
std               0.056832              0.019307     1.343779
0.495625
min              30.713360            -81.490200     0.000000
0.000000
25%              30.811400            -81.445500     0.000000
1.000000
50%              30.854270            -81.424100     0.000000
1.000000
75%              30.901808            -81.405400     0.000000
1.000000
max              30.956330            -81.402220    50.000000
7.000000

       Prerelocations  Total Lost Eggs  Total Lost Hatchlings  Lost
Nest  \
count         10292.0      9711.000000             351.000000
268.0
mean              0.0         2.015240               7.222222
```

```
1.0
std                 0.0        6.771417                14.307108
0.0
min                 0.0        0.000000                 1.000000
1.0
25%                 0.0        1.000000                 1.000000
1.0
50%                 0.0        1.000000                 1.000000
1.0
75%                 0.0        1.000000                 5.000000
1.0
max                 0.0      117.000000                99.000000
1.0

       Incubation (days)  Clutch Count   Shells>50%  Unhatched Eggs  \
count        6949.000000   9468.000000  9228.000000     9228.000000
mean           58.124478    106.220215    75.870178       26.129714
std             4.955463     22.680940    34.850677       30.591279
min            43.000000      0.000000     0.000000        0.000000
25%            55.000000     92.000000    59.000000        6.000000
50%            58.000000    107.000000    83.000000       13.000000
75%            61.000000    121.000000   100.000000       32.000000
max            80.000000    200.000000   162.000000      172.000000

       Dead Hatchlings  Live Hatchlings  Misoriented Hatchlings  \
count      9228.000000      9228.000000             3503.000000
mean          1.647703         1.367794                1.129603
std           6.750437         8.648449                6.205163
min           0.000000         0.000000                0.000000
25%           0.000000         0.000000                0.000000
50%           0.000000         0.000000                0.000000
75%           1.000000         0.000000                0.000000
max         121.000000       154.000000              100.000000

       Final Status Unknown  Exclude From Calc  Hatch Success  \
count                 805.0                4.0    9485.000000
mean                    1.0                1.0      69.269899
std                     0.0                0.0      30.831031
min                     1.0                1.0       0.000000
25%                     1.0                1.0      58.620700
50%                     1.0                1.0      82.352900
75%                     1.0                1.0      91.538500
max                     1.0                1.0     161.643800

       Emergence Success
count        9485.000000
mean           66.540107
std            31.941269
min           -52.500000
25%            52.671800
```

```
50%             80.172400
75%             90.598300
max            158.904100
```

```python
print(dw['DATE'].min())
print(dw['DATE'].max())
```

```
2008-05-01
2023-10-31
```

```python
print(mw['DATE'].min())
print(mw['DATE'].max())
```

```
2003-05
2023-10
```

```python
vlm.describe()
```

```
         Unnamed: 0  Longitude(deg)   Latitude(deg)  
VLMrate(cm/yr)  \
count  1.231300e+04     12313.000000    12313.000000      12313.000000

mean   2.753392e+06       -81.439538       30.867881          0.014255

std    4.549850e+03         0.018496        0.048872          0.020054

min    2.736587e+06       -81.490000       30.722498         -0.061000

25%    2.750343e+06       -81.453000       30.835706          0.001000

50%    2.753712e+06       -81.439000       30.873744          0.015000

75%    2.756952e+06       -81.427000       30.908699          0.028000

max    2.761310e+06       -81.402000       30.970617          0.088000


        VLMerror(cm/yr)
count      12313.000000
mean           0.053331
std            0.027126
min            0.000000
25%            0.033000
50%            0.051000
75%            0.069000
max            0.188000
```

Data Cleaning (10 points)

In this data cleaning process, we took several steps to enhance the dataset's quality. We eliminated rows with nonsensical dates, such as '2009-00-0,' as they lacked meaningful information and were predominantly filled with NaN values. Converting date-related variables into datetime objects helped us merge datasets based on dates. To determine the hatching date and corresponding night temperature, we calculated the estimated emergence date by adding the mean incubation days to the Activity Date, representing the nest's creation. For temperature analysis, we computed the monthly night temperature using a weighted average of 80% minimum and 20% maximum temperature. This average filled in missing daily night temperatures, crucial for representing the hatching temperature, considering sea turtles hatch at night. Additionally, we introduced features for the final latitude and longitude, accounting for relocated nests. Missing coordinates were imputed with the mean emergence latitude and longitude.

Post data cleaning, we merged sea turtle data with monthly and daily weather data based on dates. Challenges arose when integrating vertical land motion data due to specific latitude and longitude values. To address this, we employed a function to identify the closest coordinates between the two datasets, facilitating a comprehensive overview of the incubation and emergence circumstances.

Our approach aimed to retain as much data as possible for accurate predictions. We excluded data with minimal contribution to models and filled missing values with representative data to enhance the dataset's reliability.

```
# Drop rows with weird dates
cuis_cleaned = cuis.drop(cuis.loc[np.isnan(cuis['Week']) ==
True].index)
cuis_cleaned['Activity Date'] = pd.to_datetime(cuis_cleaned['Activity
Date'])

# Make objects in cuis into dates
cuis_cleaned['Emerge Date'] = pd.to_datetime(cuis_cleaned['Emerge
Date'])
cuis_cleaned['Emerge_month'] = cuis_cleaned['Emerge Date'].dt.month

# Make estimated emergence date column
mean_incub_days = cuis_cleaned["Incubation (days)"].mean().round()
cuis_cleaned['est_incub_(days)'] = cuis_cleaned['Activity Date'] +
pd.DateOffset(days=mean_incub_days)

# make objects in dw and mw into dates
dw['DATE'] = pd.to_datetime(dw['DATE'])
mw['DATE'] = pd.to_datetime(mw['DATE'])
mw['Month'] = mw['DATE'].dt.month
mw['Year'] = mw['DATE'].dt.year

# make a night avg temp in monthly weather data
mw['Monthly_Night_TAVG'] = mw['TMIN']*0.8 + mw['TMAX']*0.2

# set a column of final location in cuis
cuis_cleaned['Emerge_Lat'] = cuis_cleaned['Latitude']
reloc_or_loc = ~np.isnan(cuis_cleaned['Relocation Latitude'])
cuis_cleaned.loc[reloc_or_loc, 'Emerge_Lat'] =
```

```
cuis_cleaned.loc[reloc_or_loc, 'Relocation Latitude']

cuis_cleaned['Emerge_Lon'] = cuis_cleaned['Longitude']
reloc_or_loc = ~np.isnan(cuis_cleaned['Relocation Longitude'])
cuis_cleaned.loc[reloc_or_loc, 'Emerge_Lon'] =
cuis_cleaned.loc[reloc_or_loc, 'Relocation Longitude']
```

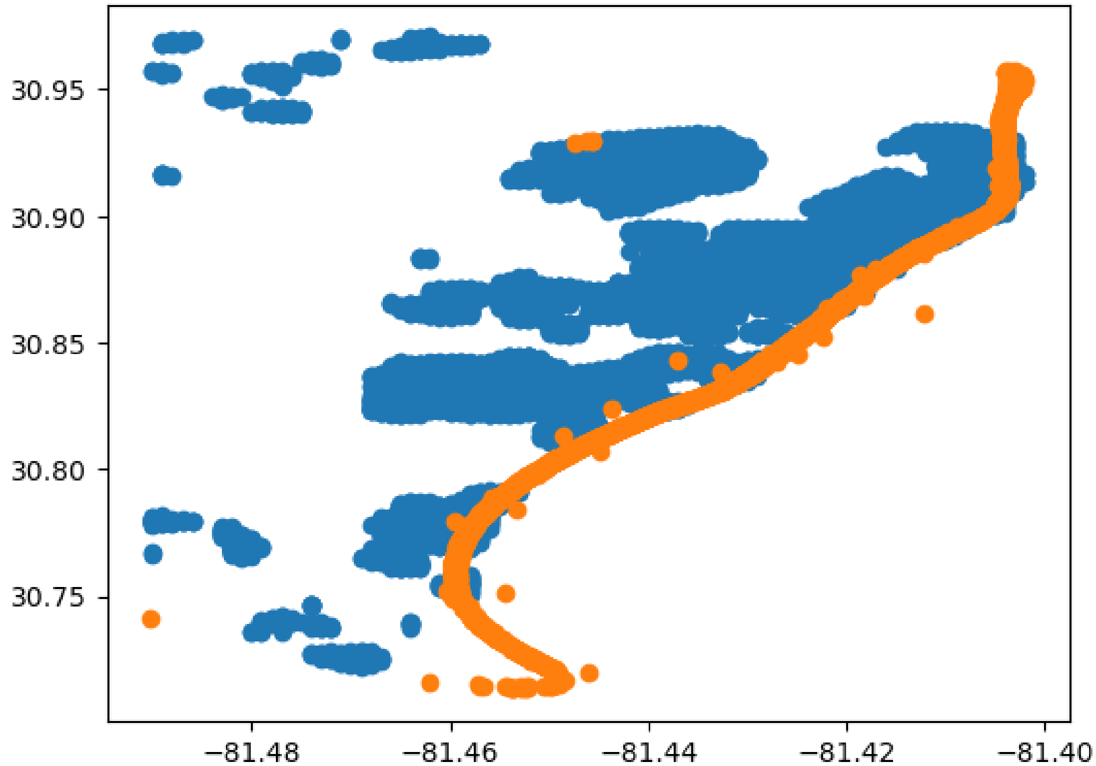
Now, we merge the nesting data with the weather dataframes. We grab monthly data to approximate conditions during incubation, and then match the nightly average temperature to the estimated date of emergence (we calculated using the mean incubation period) to approximate conditions during the actual emergence event.

```
cuis_merged = cuis_cleaned.merge(mw[['Monthly_Night_TAVG', 'Month',
'Year']], how = 'inner', on = ['Month', 'Year'])
cuis_merged = cuis_merged.merge(dw[['DATE', 'NIGHT_TAVG']],
how='left', left_on='est_incub_(days)', right_on='DATE')
```

Now we set up the dataframes to merge vertical land movement with the other data.

```
vlm.rename(columns={' Latitude(deg)': 'Latitude(deg)', '
VLMerror(cm/yr)': 'VLMerror(cm/yr)', ' VLMrate(cm/yr)':
'VLMrate(cm/yr)'}, inplace=True)

plt.scatter(vlm['Longitude(deg)'], vlm['Latitude(deg)'])
plt.scatter(cuis_cleaned['Emerge_Lon'], cuis_cleaned['Emerge_Lat'])
```

```
<matplotlib.collections.PathCollection at 0x7f2ae8159af0>
```




```
#setting missing emerge lat and lon to mean emerge lat and lon
missing_lat_idx = np.isnan(cuis_merged['Emerge_Lat'])
cuis_merged.loc[missing_lat_idx, 'Emerge_Lat'] =
np.mean(cuis_merged['Emerge_Lat'])
cuis_merged.loc[missing_lat_idx, 'Emerge_Lon'] =
np.mean(cuis_merged['Emerge_Lon'])

!pip install haversine

Collecting haversine
  Using cached haversine-2.8.0-py2.py3-none-any.whl (7.7 kB)
Installing collected packages: haversine
Successfully installed haversine-2.8.0

# function to find the closest lat-lon coordinates in vlm to emergence
lat-lon coordinates in cuis
from haversine import haversine
def closest_coor(lat, lon, df):
  dist = []
  coord1 = (lat, lon)
  for _, row in df.iterrows():
    coord2 = (row['Latitude(deg)'], row['Longitude(deg)'])
    dist.append(haversine(coord1, coord2))

  min_dist_idx = np.argmin(dist)
  return df.loc[min_dist_idx, 'Latitude(deg)'], df.loc[min_dist_idx,
```

```
'Longitude(deg)']

#insert closest vlm lat and lon to corresponding emerge lat and lon
closest_coords = []
for _, row in cuis_merged.iterrows():
  closest_coords.append(closest_coor(row['Emerge_Lat'],
row['Emerge_Lon'], vlm))

cuis_merged['vlm lat'], cuis_merged['vlm lon'] = zip(*closest_coords)

cuis_merged = cuis_merged.merge(vlm[['Latitude(deg)',
'Longitude(deg)', 'VLMrate(cm/yr)', 'VLMerror(cm/yr)']], how =
'inner', left_on = ['vlm lat', 'vlm lon'], right_on=['Latitude(deg)',
'Longitude(deg)'])
cuis_merged

cuis_merged.to_csv('cuis_merged.csv', index=False)
```

Since the above code takes a long time to run, we saved the merged dataframe as a csv file to load back in when we were working with the notebook.

Data Summary and Exploratory Data Analysis (10 points)

In this section you should provide a tour through some of the basic trends and patterns in your data. This includes providing initial plots to summarize the data, such as box plots, histograms, trends over time, scatter plots relating one variable or another.

Here, we load in the merged dataframe to avoid the time it takes to run the merge process.

```
cuis_merged = pd.read_csv('cuis_merged.csv')

cuis_merged['Hatch Success'].describe()

count    8599.000000
mean       69.532760
std        30.705003
min         0.000000
25%        59.566650
50%        82.539700
75%        91.472900
max       161.643800
Name: Hatch Success, dtype: float64

cuis_merged['Emergence Success'].describe()

count    8599.000000
mean       67.041573
std        31.591287
min       -52.500000
25%        53.632800
```

```
50%         80.373800
75%         90.598300
max        158.904100
Name: Emergence Success, dtype: float64
```

Here we see that some of our values for Hatch & Emergence Success, which are supposed to be a percentages, are above 100, which is strange. We will recalculate our values of hatch success and emergence success to ensure they are realistic. Hatch Success is the percentage out of the total clutch count that hatch. Emergence success is the percentage of eggs that hatch and make it out of the nest, which is the total hatched eggs minus live and dead hatchlings found in the nest, divided by the total clutch count.

```
cuis_merged['Hatch Success Corrected'] = (cuis_merged['Clutch Count']
- cuis_merged['Unhatched Eggs'])/cuis_merged['Clutch Count']
cuis_merged['Hatch Success Corrected'].describe()
```

```
count    8349.000000
mean        0.755664
std         0.277365
min        -0.386555
25%         0.698925
50%         0.875000
75%         0.939024
max         1.000000
Name: Hatch Success Corrected, dtype: float64
```

```
subtraction_var = cuis_merged['Unhatched Eggs'] + cuis_merged['Live
Hatchlings'] + cuis_merged['Dead Hatchlings']
cuis_merged['Emergence Success Corrected'] = (cuis_merged['Clutch
Count'] - subtraction_var)/cuis_merged['Clutch Count']
cuis_merged['Emergence Success Corrected'].describe()
```

```
count    8349.000000
mean        0.730006
std         0.289359
min        -0.388889
25%         0.646552
50%         0.857143
75%         0.929032
max         1.000000
Name: Emergence Success Corrected, dtype: float64
```

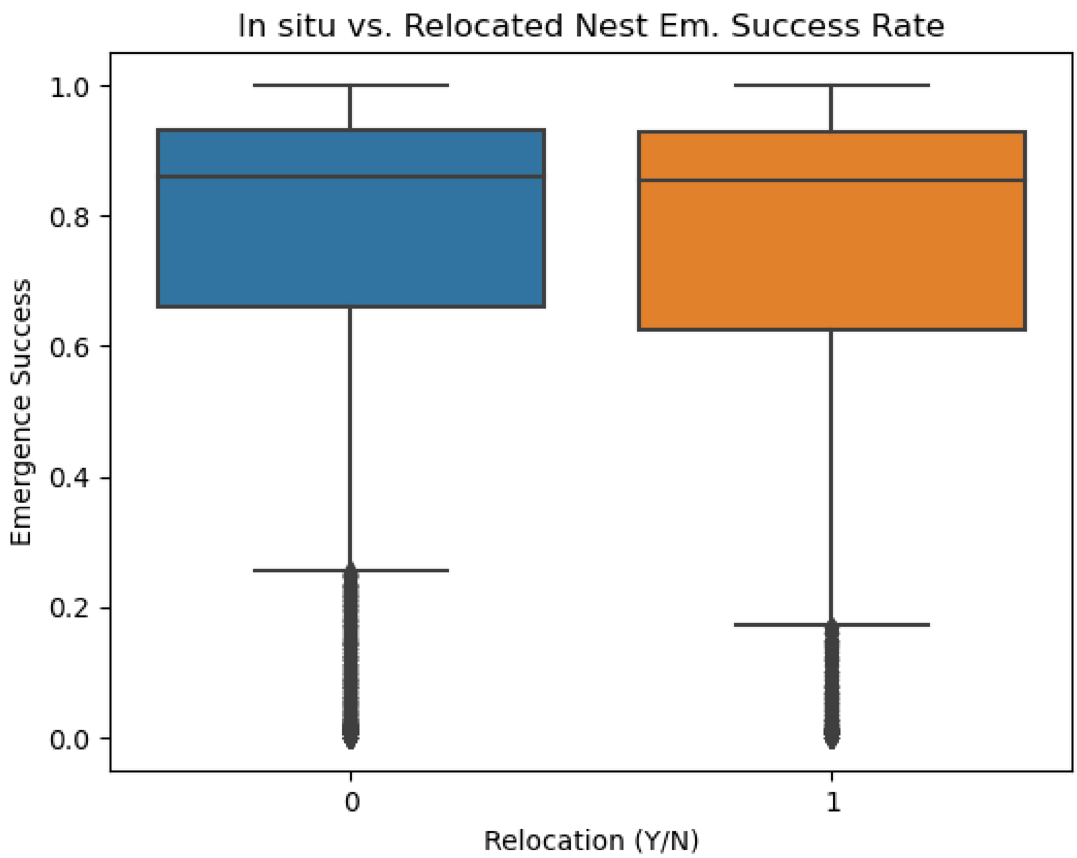
```
cuis_merged_clean =
cuis_merged.drop(cuis_merged.loc[cuis_merged['Hatch Success
Corrected'] < 0].index)
cuis_merged_clean =
cuis_merged.drop(cuis_merged.loc[cuis_merged['Emergence Success
Corrected'] < 0].index)
cuis_merged_clean['Emergence Success Corrected'].describe()
```

```
len(cuis_merged_clean.loc[cuis_merged_clean['Final Status Unknown'] ==
1.0])
```

```
750
```

```
rel = cuis_merged_clean.loc[cuis['Relocation'] == 'relocated']
['Emergence Success Corrected']
ins = cuis_merged_clean.loc[cuis['Relocation'] == 'in situ']
['Emergence Success Corrected']
bxplt_array = [ins, rel]
sns.boxplot(data = bxplt_array).set(xlabel = 'Relocation (Y/N)',
ylabel = 'Emergence Success', title = "In situ vs. Relocated Nest Em.
Success Rate")
```

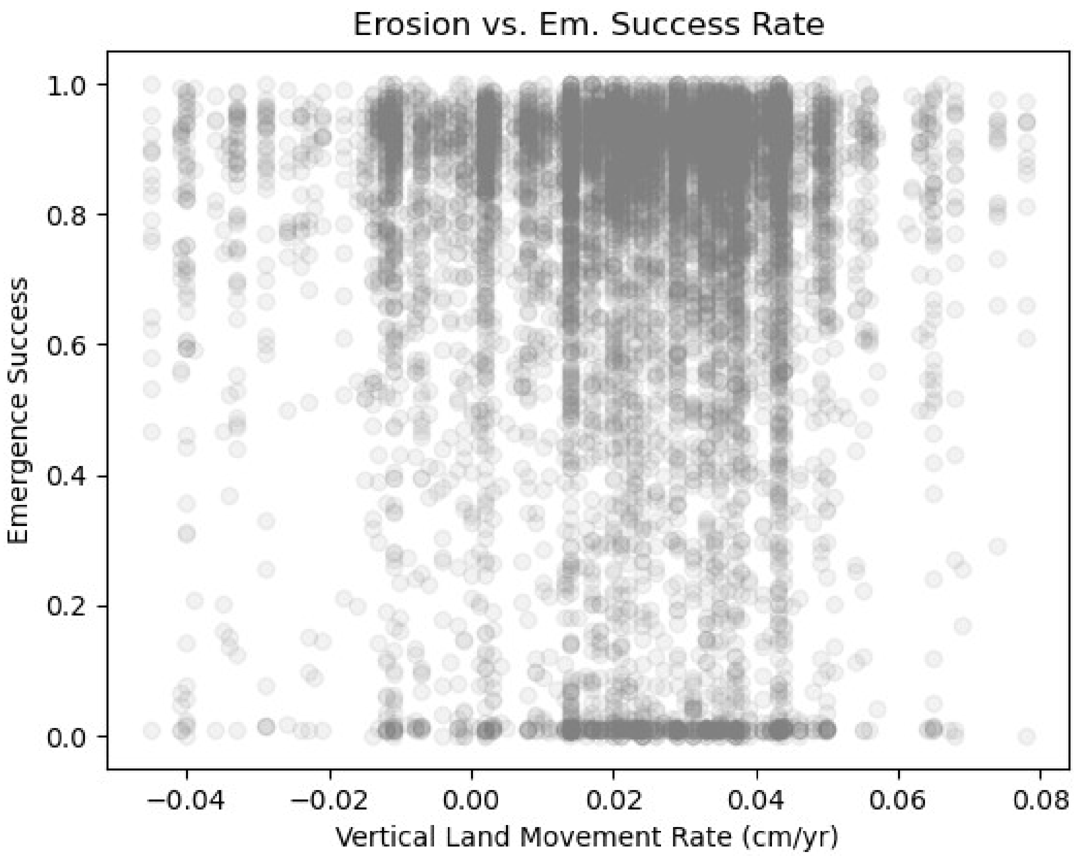
```
[Text(0.5, 0, 'Relocation (Y/N)'),
 Text(0, 0.5, 'Emergence Success'),
 Text(0.5, 1.0, 'In situ vs. Relocated Nest Em. Success Rate')]
```




```
plt.scatter(cuis_merged_clean['VLMrate(cm/yr)'],
cuis_merged_clean['Emergence Success Corrected'], alpha = 0.1, color =
'gray')
plt.xlabel('Vertical Land Movement Rate (cm/yr)')
```

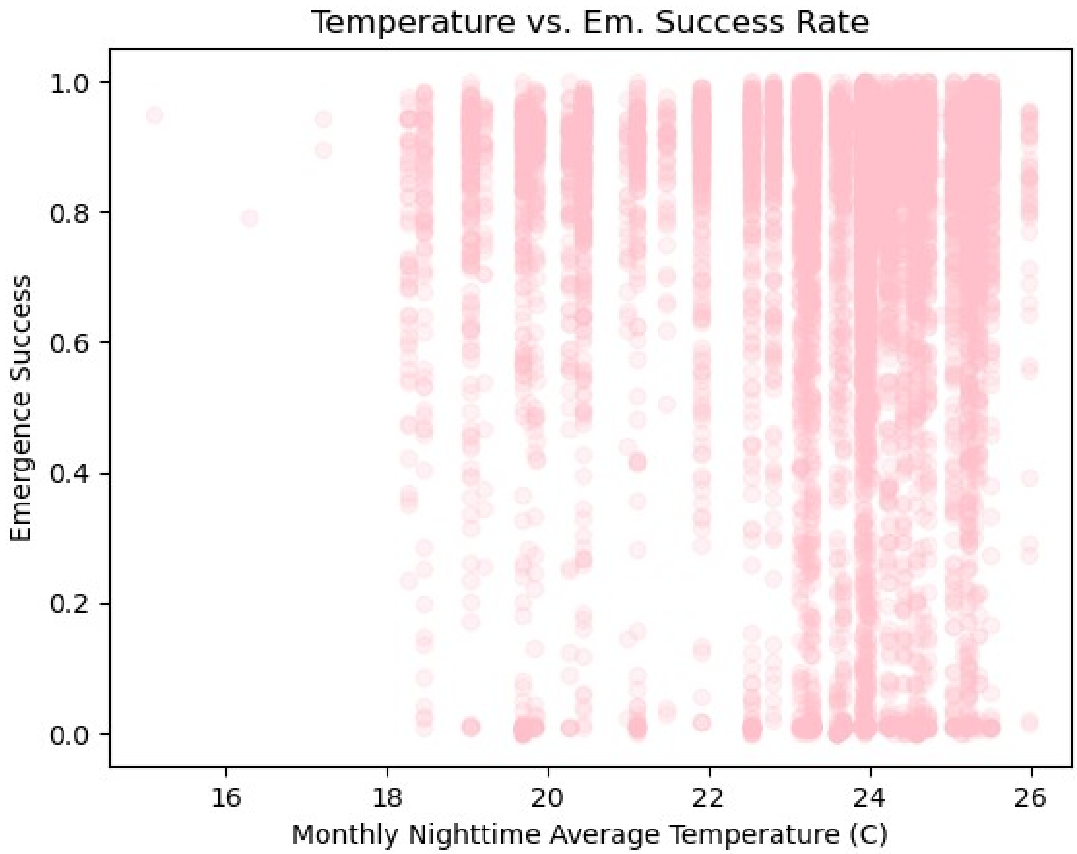
```
plt.ylabel('Emergence Success')
plt.title('Erosion vs. Em. Success Rate')
```

```
Text(0.5, 1.0, 'Erosion vs. Em. Success Rate')
```




```
plt.scatter(cuis_merged_clean['Monthly_Night_TAVG'],
cuis_merged_clean['Emergence Success Corrected'], alpha = 0.2, color =
'pink')
plt.xlabel('Monthly Nighttime Average Temperature (C)')
plt.ylabel('Emergence Success')
plt.title('Temperature vs. Em. Success Rate')
```

```
Text(0.5, 1.0, 'Temperature vs. Em. Success Rate')
```




```
cuis_merged_clean.head()
```

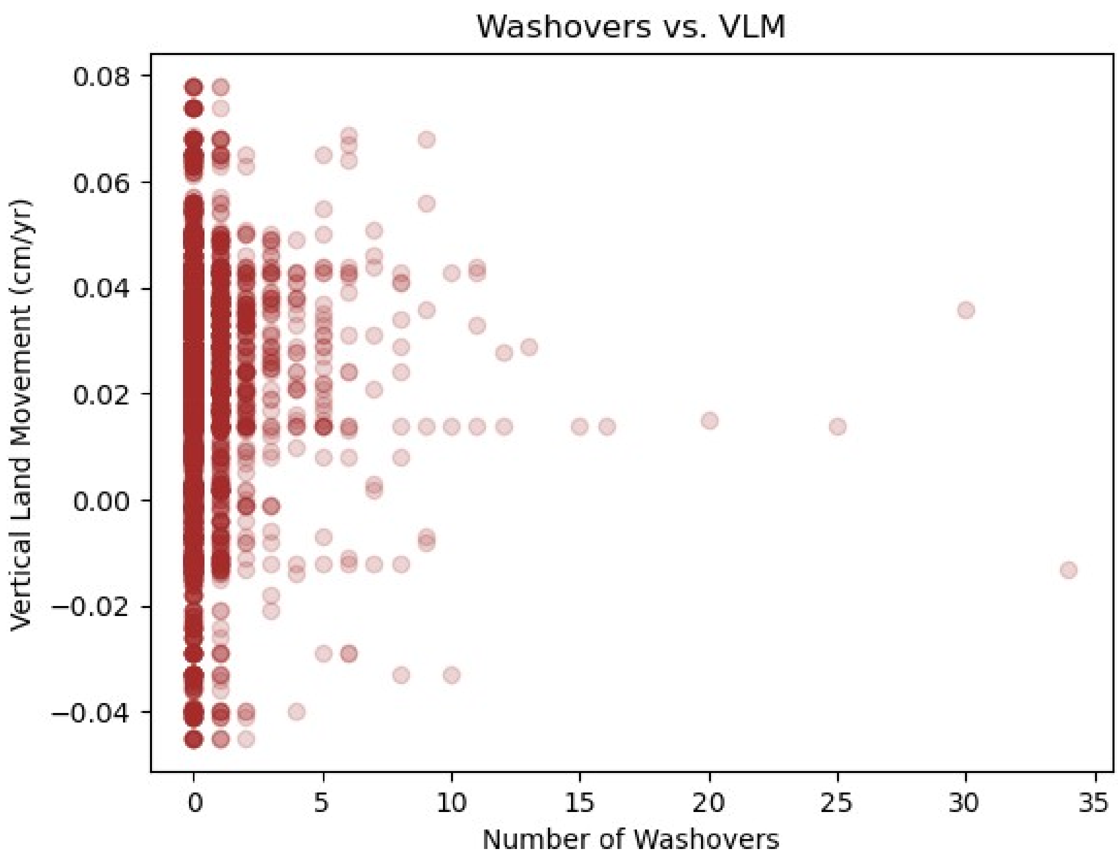
```
      UID       Beach  County  Activity # Activity  Nest # Ref #  \
0   99003  Cumberland  Camden           1        N       1   NaN
1   99965  Cumberland  Camden         148        N      84   NaN
2  100546  Cumberland  Camden         225        N     133   NaN
3  101127  Cumberland  Camden         327        N     190   NaN
4     427  Cumberland  Camden           5        N       5   NaN

  Activity Date  Year  Month  Week  Dayofyear  JulianDate  Latitude  \
0    2008-05-09  2008      5  18.0      130.0   2454595.5  30.81768
1    2008-06-09  2008      6  23.0      161.0   2454626.5  30.81803
2    2008-06-19  2008      6  24.0      171.0   2454636.5  30.81868
3    2008-06-28  2008      6  25.0      180.0   2454645.5  30.81895
4    2009-05-14  2009      5  19.0      134.0   2454965.5  30.81910

   Longitude Relocation  Relocation Latitude  Relocation Longitude  \
0  -81.44213    in situ                  NaN                   NaN
1  -81.44197    in situ                  NaN                   NaN
2  -81.44142    in situ                  NaN                   NaN
3  -81.44138    in situ                  NaN                   NaN
4  -81.44110  relocated              30.8192              -81.4413
```

```
  Relocation Location  Washovers  Loss Reports  Prerelocations  \
0                 NaN        0.0             0               0
1                 NaN        0.0             0               0
2                 NaN        2.0             0               0
3                 NaN        1.0             0               0
4                 NaN        0.0             2               0

   Total Lost Eggs  Total Lost Hatchlings  Lost Nest Emerge Date  \
0              NaN                    NaN        NaN  2008-07-15
1              NaN                    NaN        NaN  2008-08-11
2              NaN                    NaN        NaN         NaN
3              NaN                    NaN        NaN         NaN
4              1.0                    1.0        NaN  2009-07-21

  Inventory Date  Incubation (days)  Clutch Count  Shells>50%
Unhatched Eggs  \
0            NaN               67.0         123.0        89.0
34.0
1            NaN               63.0         150.0       127.0
23.0
2            NaN                NaN          93.0         0.0
93.0
3            NaN                NaN         124.0         0.0
124.0
4        7/29/09               68.0         120.0        73.0
41.0

   Dead Hatchlings  Live Hatchlings  Misoriented Hatchlings  \
0              1.0              0.0                     NaN
1              1.0              0.0                     NaN
2              0.0              0.0                     NaN
3              0.0              0.0                     NaN
4              1.0              0.0                     NaN

   Final Status Unknown  Exclude From Calc  Hatch Success  Emergence
Success  \
0                   NaN                NaN        72.3577
71.5447
1                   NaN                NaN        84.6667
84.0000
2                   NaN                NaN         0.0000
0.0000
3                   NaN                NaN         0.0000
0.0000
4                   NaN                NaN        60.8333
60.0000

   Emerge_month est_incub_(days)  Emerge_Lat  Emerge_Lon
Monthly_Night_TAVG  \
0           7.0       2008-07-06    30.81768   -81.44213
```

```
19.758
1           8.0       2008-08-06    30.81803   -81.44197
23.578
2           NaN       2008-08-16    30.81868   -81.44142
23.578
3           NaN       2008-08-25    30.81895   -81.44138
23.578
4           7.0       2009-07-11    30.81920   -81.44130
20.992

         DATE  NIGHT_TAVG    vlm lat  vlm lon  Latitude(deg)
Longitude(deg)  \
0  2008-07-06        75.2  30.817591  -81.445      30.817591          -
81.445
1  2008-08-06        81.2  30.817591  -81.445      30.817591          -
81.445
2  2008-08-16        76.4  30.817591  -81.445      30.817591          -
81.445
3  2008-08-25        73.8  30.817591  -81.445      30.817591          -
81.445
4  2009-07-11        76.4  30.817591  -81.445      30.817591          -
81.445

   VLMrate(cm/yr)  VLMerror(cm/yr)  Hatch Success Corrected  \
0           0.029            0.093                 0.723577
1           0.029            0.093                 0.846667
2           0.029            0.093                 0.000000
3           0.029            0.093                 0.000000
4           0.029            0.093                 0.658333

   Emergence Success Corrected
0                     0.715447
1                     0.840000
2                     0.000000
3                     0.000000
4                     0.650000
```

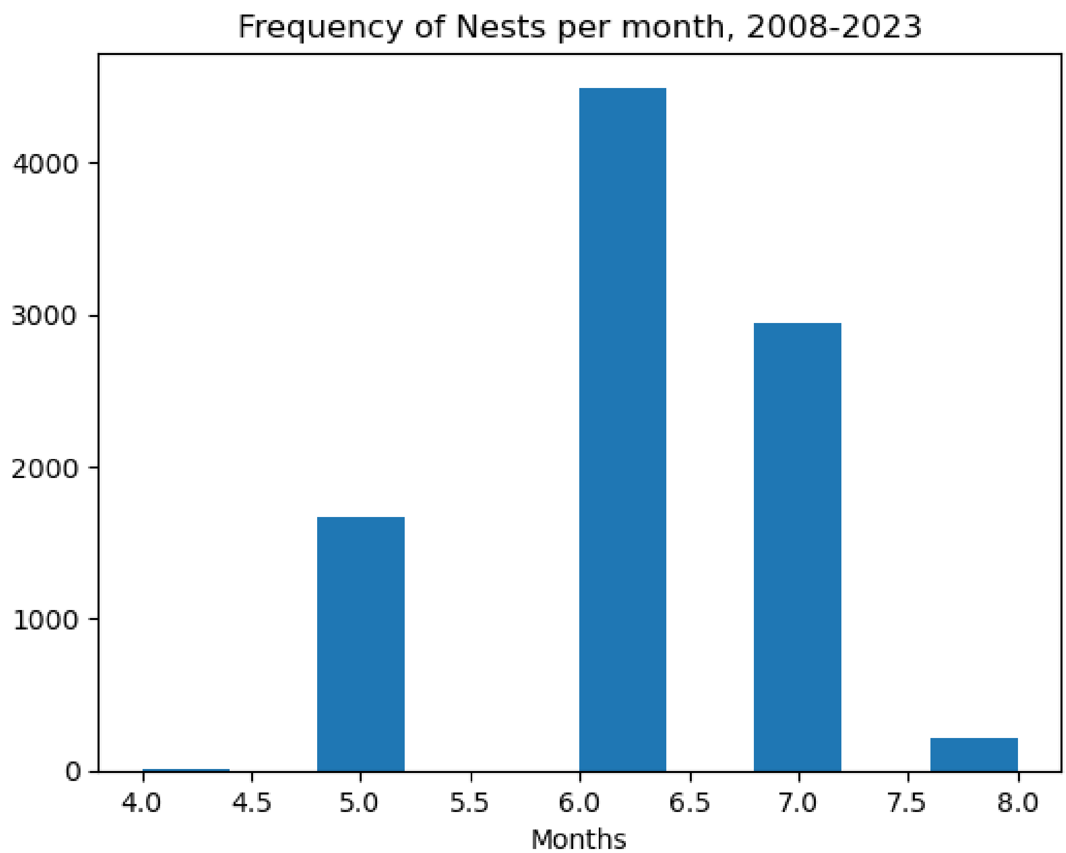
```
plt.scatter(cuis_merged_clean.Washovers,
cuis_merged_clean['VLMrate(cm/yr)'], c='brown', alpha = 0.2)
plt.xlabel('Number of Washovers')
plt.ylabel('Vertical Land Movement (cm/yr)')
plt.title('Washovers vs. VLM')
```

```
Text(0.5, 1.0, 'Washovers vs. VLM')
```




```
plt.hist(cuis_merged_clean['Month'])
plt.title("Frequency of Nests per month, 2008-2023")
plt.xlabel('Months')
```

```
Text(0.5, 0, 'Months')
```



Forecasting and Prediction Modeling (25 points)

This section is where the rubber meets the road. In it you must:

1. Explore at least 3 prediction modeling approaches for each prediction question, ranging from the simple (e.g. linear regression, KNN) to the complex (e.g. SVM, random forests, Lasso).

2. Motivate all your modeling decisions. This includes parameter choices (e.g., how many folds in k-fold cross validation, what time window you use for averaging your data) as well as model form (e.g., If you use regression trees, why? If you include nonlinear features in a regression model, why?).
3. Carefully describe your cross validation and model selection process. You should partition your data into training and testing data sets. The training data set is what you use for cross-validation (i.e. you sample from within it to create folds, etc.). The testing data set is held to the very end of your efforts, and used to compare qualitatively different models (e.g. OLS vs random forests).
4. Very carefully document your workflow. We will be reading a lot of projects, so we need you to explain each basic step in your analysis.

5. Seek opportunities to write functions allow you to avoid doing things over and over, and that make your code more succinct and readable.

We convert the categorical "Relocation" variable into a numeric (0/1) variable in order to include it in our regression.

```
cuis_merged_clean['Relocation'] =
cuis_merged_clean.Relocation.eq('relocated').mul(1)
```

Here, we split our data into the testing and training datasets that will be used to train and validate our model, and to predict on.

```
cuis_merged_clean_training =
cuis_merged_clean.drop(cuis_merged_clean.loc[cuis_merged_clean['Final
Status Unknown'] == 1.0].index)

cuis_merged_clean_test =
cuis_merged_clean.loc[cuis_merged_clean['Final Status Unknown'] ==
1.0]

X_to_predict = cuis_merged_clean_test[['Year', 'Month', 'Emerge_Lat',
'Emerge_Lon', 'VLMrate(cm/yr)', 'Monthly_Night_TAVG', 'NIGHT_TAVG',
'Relocation']]

cuis_merged_clean_training =
cuis_merged_clean_training.drop(cuis_merged_clean_training.loc[np.isna
n(cuis_merged_clean_training['Hatch Success Corrected']) ==
True].index)
len(cuis_merged_clean_training)

8325
```

Regression

Here we are selecting the relevant features to include in our modeling of Hatch Success (corrected).

```
X = cuis_merged_clean_training[['Year', 'Month', 'Emerge_Lat',
'Emerge_Lon', 'VLMrate(cm/yr)', 'Monthly_Night_TAVG', 'NIGHT_TAVG',
'Relocation']] #specify the features
y = cuis_merged_clean_training['Hatch Success Corrected'] #specify the
target variable
```

Here we use k-fold cross validation with 10 folds. We decided on 10 folds as we have over 8000 observations to train our data on, so splitting the data 10 times will lead to large sample sizes each time without being too computationally intensive. We wrote a function to perform this validation so the code is more readable.

```
from sklearn.model_selection import KFold
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
```

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

def kfold_crossval_model(SKL_model, X, y, alpha, n_folds):
    """
    Objective: This function will perform k-fold cross validation and
calculate MSE for each fold, using multiple models.

    Outputs: array of RMSE values across folds and mean RMSE value
from all folds
    """

    # YOUR CODE HERE

    # Initialize the model, differentiating between models that
require and do not require the alpha parameter
    if SKL_model == LinearRegression:
        model = SKL_model()
    else:
        model = SKL_model(alpha = alpha)

    # Fit the model on the training data and get the mean squared
error on the test data
    kf = KFold(n_splits = n_folds, shuffle = True, random_state = 10)

    fold_rmse = [] # initiate a list to hold the MSE associated with
each fold

    for t_index, v_index in kf.split(X):
        # Subset X and y into training and validation subsets
        X_fold_train = X.iloc[t_index]
        y_fold_train = y.iloc[t_index]
        X_fold_val = X.iloc[v_index]
        y_fold_val = y.iloc[v_index]

        # Initiate and fit a linear regression model using the
training data
        model.fit(X_fold_train, y_fold_train)

        # Predict the Y-values associated with the validation data
        y_pred = model.predict(X_fold_val)

        # Find the testing MSE and append it to fold_mse
        fold_rmse.append(np.sqrt(mean_squared_error(y_fold_val,
y_pred)))
        mean_rmse = np.mean(fold_rmse)

    return fold_rmse, mean_rmse
```

```
kfold_crossval_model(LinearRegression, X, y, alpha = 0.5, n_folds =
10)
```

```
([0.27751384347077745,
  0.28551730828910354,
  0.27371478016256273,
  0.28357777874371415,
  0.2685591600175452,
  0.2687402384460932,
  0.2636598526948538,
  0.274293035431503,
  0.27121774938774657,
  0.2752282261647822],
 0.2742021972808682)
```

```
lm_hatch = LinearRegression()
lm_hatch.fit(X, y)

print(lm_hatch.coef_)
```

```
[-0.00277961 -0.03747007 -0.22496665  0.41005262 -0.10478683
0.00405657
 -0.00231884  0.05307021]
```

```
lm_hatch_ypred = pd.DataFrame(lm_hatch.predict(X_to_predict))
lm_hatch_ypred.describe()
```

	0
count	750.000000
mean	0.730837
std	0.028735
min	0.649024
25%	0.709826
50%	0.728657
75%	0.749667
max	0.817179

Here we run the same regression, but using the Ridge model. We chose an alpha value of 0.5 to penalize added variables that do not provide information without overly penalizing added features. We determined this by manually changing the alpha parameter and observing the MSE results from k-fold cross validation.

```
kfold_crossval_model(Ridge, X, y, alpha = 0.5, n_folds = 10)
```

```
([0.2775251028052225,
  0.28549920081369046,
  0.27370112426048676,
  0.2835499421610755,
  0.2682378809321699,
  0.26871143512996426,
```

```
  0.2637002049548184,
  0.274330934434998,
  0.2711212972877436,
  0.2751977093332388],
 0.2741574832113408)
```

The MSE's were not very high, and I doubt we could lower them without incorporating better features, so we will train the model on our entire training set. Here we output the values of the coefficients in the model.

```
ridge_hatch = Ridge(alpha = 0.5)
ridge_hatch.fit(X, y)

print(ridge_hatch.coef_)
```

```
[-0.00278971 -0.03744058 -0.12373518  0.10020934 -0.09790349
0.00405168
 -0.00231999  0.05292977]
```

Now we will use the fitted model from the cell above to predict Hatch Success Rate for the nests where this data was unable to be collected.

```
ridge_hatch_ypred = pd.DataFrame(ridge_hatch.predict(X_to_predict))
ridge_hatch_ypred.describe()
```

```
                0
count  750.000000
mean     0.730827
std      0.028487
min      0.650703
25%      0.710077
50%      0.728931
75%      0.749951
max      0.819498
```

Now, we will run Lasso regression via the same methods, except with a much lower alpha value so that our model penalizes added features less heavily, as our features are not great predictors of our target variable (uh oh).

```
kfold_crossval_model(Lasso, X, y, alpha = 0.001, n_folds = 10)
```

```
([0.2777143743519547З,
  0.28519080616760023,
  0.273272534484958,
  0.2837379598193187,
  0.2683177449170203,
  0.2683217715552033,
  0.2639100064664687,
  0.27443810962321374,
```

```
  0.2713777619851771,
  0.27503132488870 7],
 0.2741312394259622)
```

```
lasso_hatch = Lasso(alpha = 0.001)
lasso_hatch.fit(X, y)

print(lasso_hatch.coef_)
```

```
[-0.00270148 -0.02629596 -0.          -0.          -0.           0.
 -0.00199691  0.04860102]
```

As we can see above, the Lasso model determines most of our features are not useful, even at our low alpha level of 0.001. Now we will predict using our lasso model:

```
lasso_hatch_ypred = pd.DataFrame(lasso_hatch.predict(X_to_predict))
lasso_hatch_ypred.describe()
```

```
                0
count  750.000000
mean     0.733377
std      0.025776
min      0.674461
25%      0.714112
50%      0.727452
75%      0.751669
max      0.806263
```

Now, we will change our X and y to predict Emergence success rather than Hatch Success. We will run all the same models and adjust the hyperparameters accordingly.

```
X = cuis_merged_clean_training[['Year', 'Month', 'Emerge_Lat',
'Emerge_Lon', 'VLMrate(cm/yr)', 'Monthly_Night_TAVG', 'NIGHT_TAVG',
'Relocation']] #specify the features
y = cuis_merged_clean_training['Emergence Success Corrected'] #specify
the target variable

kfold_crossval_model(LinearRegression, X, y, alpha = 0.5, n_folds =
10)
```

```
([0.2933984107794873,
  0.2916425384143592,
  0.280714106358638,
  0.29136381008260176,
  0.2803670466089671,
  0.2788502089512199,
  0.2804520082160618,
  0.28351429203821776,
  0.28204863810887665,
```

```
  0.28605895355477995],
 0.28484100131132095)
```

Now, print out the coefficients:

```
lm_em = LinearRegression()
lm_em.fit(X, y)

print(lm_em.coef_)
```

```
[-0.00223498 -0.04225061 -0.33945921  0.81314848 -0.05330132
0.00800946
 -0.00252498  0.04753858]
```

```
lm_em_ypred = pd.DataFrame(lm_em.predict(X_to_predict))
lm_em_ypred.describe()
```

	0
count	750.000000
mean	0.708718
std	0.026451
min	0.628376
25%	0.692284
50%	0.705067
75%	0.726842
max	0.787936

```
kfold_crossval_model(Ridge, X, y, alpha = 0.5, n_folds = 10)
```

```
([0.29347391878952506,
  0.2915907134989155,
  0.2808175936165159,
  0.29132464345351555,
  0.2800018809671525,
  0.278855275026683,
  0.28048902783531626,
  0.2836095928033745,
  0.2819570186938424,
  0.2860536180316038],
 0.2848173282716444)
```

```
ridge_em = Ridge(alpha = 0.5)
ridge_em.fit(X, y)

print(ridge_em.coef_)
```

```
[-0.00224976 -0.04219608 -0.14789595  0.21707904 -0.06135813
0.00800293
 -0.00252704  0.04726698]
```

```
ridge_em_ypred = pd.DataFrame(ridge_em.predict(X_to_predict))
ridge_em_ypred.describe()
```

```
                0
count  750.000000
mean     0.708754
std      0.025796
min      0.631557
25%      0.692331
50%      0.704967
75%      0.725702
max      0.792446
```

```
kfold_crossval_model(Lasso, X, y, alpha = 0.001, n_folds = 10)
```

```
([0.29370793702977904,
  0.29124818075027337,
  0.2805872338049711,
  0.29145357744928746,
  0.2799963995194221,
  0.2786804055469424,
  0.28072539306003197,
  0.2836736235094091,
  0.28210014370709724,
  0.2859283014137956],
 0.28481011957910096)
```

```
lasso_em = Lasso(alpha = 0.001)
lasso_em.fit(X, y)

print(lasso_em.coef_)
```

```
[-0.00216211 -0.03087726 -0.         -0.         -0.
0.00387034
 -0.00220149  0.04280416]
```

```
lasso_em_ypred = pd.DataFrame(lasso_em.predict(X_to_predict))
lasso_em_ypred.describe()
```

```
                0
count  750.000000
mean     0.711454
std      0.022485
min      0.654549
25%      0.696007
50%      0.707957
75%      0.726276
max      0.780550
```

Classification:

We are using classification to predict relocation because it is a categorical variable listing nests as either "in situ" or "relocated." To do this, I created the erosion_reloc dataframe displaying the relevant features. First, we convert relocated values to 1 and in situ values to 0 in order to express each category in binary terms.

```
erosion_reloc = cuis_merged_clean[['Beach', 'Latitude', 'Longitude',
'Relocation','Clutch Count', 'Emerge_Lat', 'Emerge_Lon', 'Washovers',
'VLMrate(cm/yr)', 'VLMerror(cm/yr)']]
```

We get rid of all the NaN values.

```
erosion_reloc =
erosion_reloc.drop(erosion_reloc.loc[np.isnan(erosion_reloc['Latitude'
]) == True].index)
erosion_reloc =
erosion_reloc.drop(erosion_reloc.loc[np.isnan(erosion_reloc['Longitude
']) == True].index)
erosion_reloc = wash_reloc =
erosion_reloc.drop(erosion_reloc.loc[np.isnan(erosion_reloc['Washovers
']) == True].index)
erosion_reloc =
erosion_reloc.drop(erosion_reloc.loc[np.isnan(erosion_reloc['Clutch
Count']) == True].index)
```

We add a column of ones to the dataframe so that statsmodels can run this as an intercept value to construct a logistic curve.

```
erosion_reloc['intercept'] = np.ones(len(erosion_reloc))
erosion_reloc.head()
```

```
        Beach  Latitude  Longitude  Relocation  Clutch Count
Emerge_Lat  \
0  Cumberland  30.81768  -81.44213           0         123.0
30.81768
1  Cumberland  30.81803  -81.44197           0         150.0
30.81803
2  Cumberland  30.81868  -81.44142           0          93.0
30.81868
3  Cumberland  30.81895  -81.44138           0         124.0
30.81895
4  Cumberland  30.81910  -81.44110           1         120.0
30.81920

   Emerge_Lon  Washovers  VLMrate(cm/yr)  VLMerror(cm/yr)  intercept
0   -81.44213        0.0           0.029            0.093        1.0
1   -81.44197        0.0           0.029            0.093        1.0
2   -81.44142        2.0           0.029            0.093        1.0
```

3	-81.44138	1.0	0.029	0.093	1.0
4	-81.44130	0.0	0.029	0.093	1.0

We use the vertical land movement rate column and intercept column as features for the logistic curve.

```
import statsmodels.api as sm
y = erosion_reloc['Relocation']
X = erosion_reloc[['VLMrate(cm/yr)', 'Clutch Count', 'intercept']]
logit = sm.Logit(y, X)
logit_fit = logit.fit()
logit_fit.params
```

```
Optimization terminated successfully.
         Current function value: 0.627246
         Iterations 5
```

```
VLMrate(cm/yr)   -0.918683
Clutch Count      0.005224
intercept        -1.279521
dtype: float64
```

Using these parameters and the features, we return a vector of probabilities using the following

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

formula:

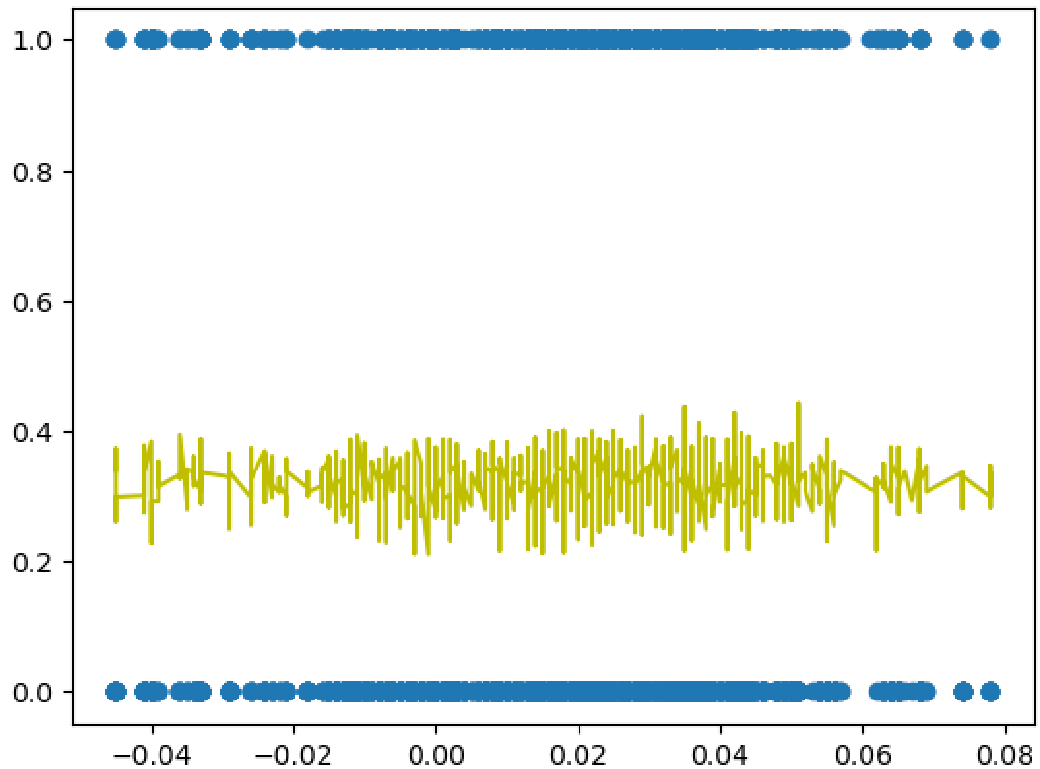
```
x_weight = 0
for i in range(len(logit_fit.params)):
    x_weight = x_weight + logit_fit.params[i] * X.iloc[:, i]
probs = np.exp(x_weight) / (1+np.exp(x_weight))
probs.values
```

```
array([0.33991701, 0.37223796, 0.30568492, ..., 0.28191125,
0.2942106 ,
       0.3348878 ])
```

The plot below represents the logistic curve for vertical land movement rate and relocation.

```
X_plot = erosion_reloc[['VLMrate(cm/yr)', 'intercept']].sort_values(by
= 'VLMrate(cm/yr)')
plt.scatter(erosion_reloc['VLMrate(cm/yr)'],
erosion_reloc['Relocation'])
y = probs.values
plt.plot(X_plot["VLMrate(cm/yr)"], y, c = 'y')
```

```
[<matplotlib.lines.Line2D at 0x7f2a4f388e80>]
```



Now we try a logistic curve with washovers and intercept as features.

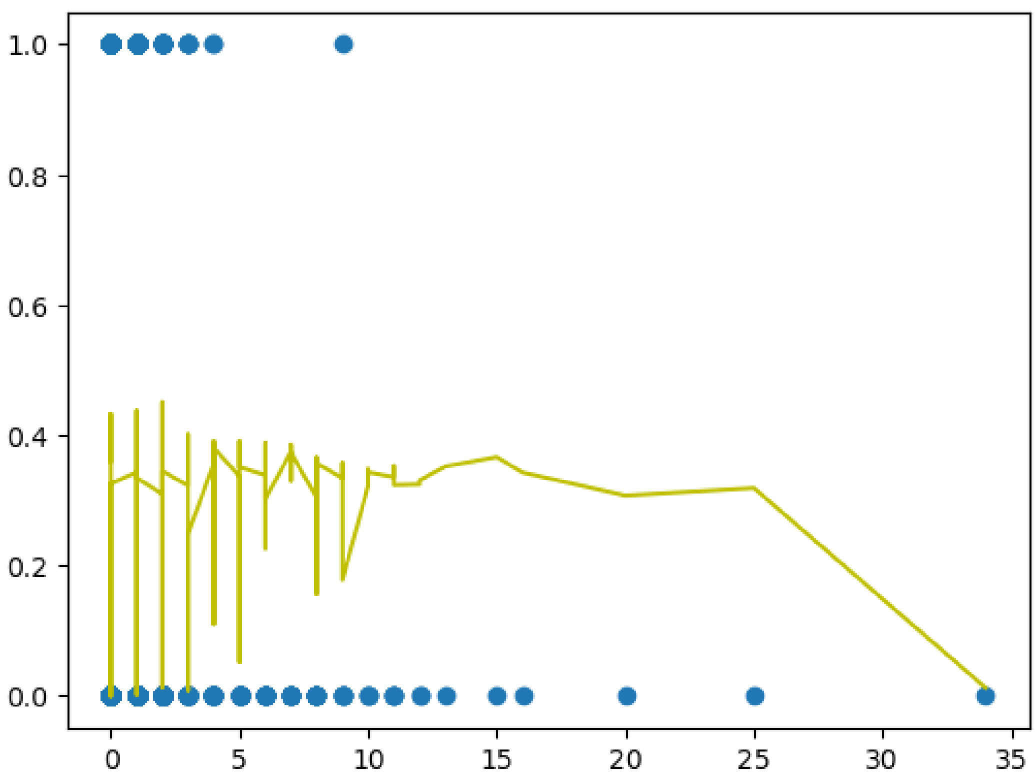
```
y = erosion_reloc['Relocation']
X = erosion_reloc[['Washovers', 'VLMrate(cm/yr)', 'Clutch Count',
'intercept']]
logit = sm.Logit(y, X)
logit_fit = logit.fit()
logit_fit.params
```

```
Optimization terminated successfully.
         Current function value: 0.617823
         Iterations 7
```

```
Washovers        -0.468888
VLMrate(cm/yr)   -0.789337
Clutch Count      0.004566
intercept        -1.112248
dtype: float64
```

```
x_w = 0
for i in range(len(logit_fit.params)):
    x_w = x_w + logit_fit.params[i] * X.iloc[:, i]
```

```
p = np.exp(x_w) / (1+np.exp(x_w))
p.values

array([0.36043617, 0.38931829, 0.16134708, ..., 0.30772966,
0.31907074,
       0.01281136])

X_plot = erosion_reloc[['Washovers', 'intercept']].sort_values(by =
'Washovers')
plt.scatter(erosion_reloc[['Washovers']], erosion_reloc['Relocation'])
y = p.values
plt.plot(X_plot["Washovers"], y, c = 'y')

[<matplotlib.lines.Line2D at 0x7f2a4e1d0160>]
```



We will test out different decision trees by fitting them to the training data and then scoring them with the training and validation data

```
from sklearn.model_selection import train_test_split

target = erosion_reloc['Relocation']
features = erosion_reloc[['Washovers', 'VLMrate(cm/yr)', 'Clutch
Count', 'intercept']]
X, X_test, y, y_test = train_test_split(features, target, test_size =
0.2, train_size = 0.8)
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size =
0.25, train_size = 0.75)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn import tree

features = erosion_reloc[['VLMrate(cm/yr)', 'Washovers', 'Longitude',
'Latitude']]
target = erosion_reloc['Relocation']

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
print("Number of features: {}".format(tree.tree_.n_features))
print("Number of nodes (internal and terminal):
{}".format(tree.tree_.node_count), "\n")
train_score = tree.score(X_train, y_train)
val_score = tree.score(X_val, y_val)
print('Train Score: ', train_score)
print('Validation Score: ', val_score)
```

```
Number of features: 4
Number of nodes (internal and terminal): 3057

Train Score:  0.898273572377158
Validation Score:  0.607484076433121
```

We can use this initial tree to rank the importance of the features. This shows that vertical land movement is the most important feature.

```
pd.DataFrame({'Feature': features.columns, 'Importance':
tree.feature_importances_})
```

	Feature	Importance
0	VLMrate(cm/yr)	0.031279
1	Washovers	0.385747
2	Longitude	0.582974
3	Latitude	0.000000

Now we will compare our initial tree with a bagging classifier using our four features and the default number of estimators.

```
from sklearn.ensemble import BaggingClassifier

bag_tree = BaggingClassifier(n_estimators = 10,max_features = 2)
bag_tree.fit(X_train, y_train)
bag_train_score = bag_tree.score(X_train, y_train)
bag_val_score = bag_tree.score(X_val, y_val)
print('Train Score: ', bag_train_score)
print('Validation Score: ', bag_val_score)
```

```
Train Score:  0.7195219123505976
Validation Score:  0.6735668789808917
```

Let's compare these scores to a random forest ensemble method:

```
from sklearn.ensemble import RandomForestClassifier

rf_tree = RandomForestClassifier(n_estimators = 10)
rf_tree.fit(X_train, y_train)
rf_train_score = rf_tree.score(X_train, y_train)
rf_val_score = rf_tree.score(X_val, y_val)
print('Train Score: ', rf_train_score)
print('Validation Score: ', rf_val_score)
```

```
Train Score:  0.8786188579017264
Validation Score:  0.6106687898089171
```

We can cross-validate to find the best hyperparameters for the random forest using RandomizedSearchCV. This filters through various possible hyperparameters.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {'max_leaf_nodes': randint(3, 100),'min_samples_leaf':
randint(1, 10),'min_samples_split': randint(2, 20)}
rf_tree_rnd_search = RandomizedSearchCV(rf_tree,
param_distributions=param_dist,cv=5, n_iter=5, random_state = 2021)
rf_tree_rnd_search.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=5,
estimator=RandomForestClassifier(n_estimators=10),
                   n_iter=5,
                   param_distributions={'max_leaf_nodes':
<scipy.stats._distn_infrastructure.rv_discrete_frozen object at
0x7f2a4decf910>,
                                        'min_samples_leaf':
<scipy.stats._distn_infrastructure.rv_discrete_frozen object at
0x7f2a4ded25e0>,
                                        'min_samples_split':
<scipy.stats._distn_infrastructure.rv_discrete_frozen object at
0x7f2a4decf3d0>},
                   random_state=2021)
```

This shows the best hyperparameters for the random forest:

```
print(rf_tree_rnd_search.best_score_)
print(rf_tree_rnd_search.best_params_)
```

```
0.6671978751660026
{'max_leaf_nodes': 36, 'min_samples_leaf': 6, 'min_samples_split': 9}
```

We chose to do boosting using GradientBoosterClassifier rather than AdaBoostClassifier because of the outliers we observed in the "Washovers" column, since AdaBoostClassifier does not handle outliers well.

```
from sklearn.ensemble import GradientBoostingClassifier

gb_tree = GradientBoostingClassifier()
gb_tree.fit(X_train, y_train)
gb_train_score = gb_tree.score(X_train, y_train)
gb_val_score = gb_tree.score(X_val, y_val)
print('Train Score: ', gb_train_score)
print('Validation Score: ', gb_val_score)
```

```
Train Score:  0.6905710491367862
Validation Score:  0.6719745222929936
```

I chose two parameters to search over -- max leaf nodes and min samples split. I chose the range that includes the values from the cross-validated best values.

```
param_dist = {'max_leaf_nodes': randint(3, 100),
              'min_samples_split': randint(4, 30)}

rnd_gb_search = 
RandomizedSearchCV(gb_tree,param_distributions=param_dist,
                                cv=5, n_iter=10)

rnd_gb_search.fit(X_train, y_train)

print(rnd_gb_search.best_params_)
```

```
{'max_leaf_nodes': 82, 'min_samples_split': 29}
```

```
gb_tree = GradientBoostingClassifier(max_leaf_nodes = 11, 
min_samples_split = 20)
gb_tree.fit(X_train, y_train)


gb_train_score = gb_tree.score(X_train, y_train)
gb_val_score = gb_tree.score(X_val, y_val)

print('Train Score: ', gb_train_score)
print('Validation Score: ', gb_val_score)
```

```
Train Score:  0.6903054448871182
Validation Score:  0.6711783439490446
```

Out of our models, gradient boosting performs best on the testing data. We can use this model for our confusion matrix.

```
models = [tree, bag_tree, rf_tree, rf_tree_rnd_search, gb_tree]
for i in models:
    print('Test Score: ', i.score(X_test, y_test))
```

```
Test Score:  0.5788216560509554
Test Score:  0.6759554140127388
Test Score:  0.5939490445859873
Test Score:  0.6807324840764332
Test Score:  0.677547770700637
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test , gb_tree.predict(X_test))
```

```
array([[847,  18],
       [387,   4]])
```

Interpretation and Conclusions (20 points)

Regression:

For both hatch and emergence success, we used regression models to predict our success rates (in percentage of total clutch count). Given that we did not have individual data for each egg, we could not determine a binary measure of mortality that we could use a classification model to predict. To evaluate the performance of our regression models, we ran k-fold cross validation on every model (OLS, Ridge, and Lasso) before choosing hyperparameters. The k-fold cross validation function calculates RMSE for each fold. Our average RMSE values were similar across the different models, and were pretty substantial. Adjusting our hyperparameters did not improve model prediction as much as we hoped originally. More specifically, our Lasso model, when given a lambda (alpha) parameter of 1 or higher, dropped all features and assigned them coefficients of zero, predicting the mean value for every observation. This indicates that our feature variables were not the best predictors of success rates, but this could be due to difficulties finding and accessing relevant datasets. When we predicted success rates for observations where there was no measured success rate, we predicted values ranging from 65-85%. This is quite a limited range and speaks to the limitations of our regression models. It is highly unlikely that none of the observations we attempted to predict had success rates below 50% or even close to zero, but our models were unable to capture that variability. This could be because our feature values do not vary significantly either, especially between different nest observations.

Classification:

We used statsmodels to create logistic regression models for washovers and VLM (vertical land movement) rate as predictor variables and relocation (1 is relocated, 0 is in situ) as the response variable. The curve represents the probability of whether or not a nest will be relocated, and plotting it with the two categories allows us to visualize our model's accuracy. Our logistic regression model for VLM rate is not accurate because the yellow regression line does not intersect with any blue points representing in situ and relocated. This could be because the in situ and relocated values overlap on the x axis. Our logistic regression model for washovers has better accuracy because some parts of the line intersect with the points, but most points still do not intersect. Additionally, our training of the model on unbalanced data with far more “in situ” nests than “relocated” nests leads to bias in our models. We constructed several decision trees after splitting the data into training, testing, and validation sets. The features we selected were longitude, latitude, washovers, and VLM rate. The initial tree, random forest, and bagging methods were all very well-fit to the training data, indicating that the features we selected have high variance for those models. Out of the five models, gradient boosting scored the best on the testing data set. Thus, we selected gradient boosting as our optimal model for predicting relocation. This model gives us a precision of 0.7368 and recall of 0.06588. This indicates that when this model predicts that a nest is relocated, it is correct about 74% of the time (however, the small subset of data in the confusion matrix makes this score highly biased). Our recall score indicates that for all nests that are actually relocated, our model has correctly identified 6.5% as being relocated. These show that our model is not very good at assessing relocated nests.

We aimed to accurately predict nest relocation in order to better understand how many nests need to be relocated each year and which features best contribute to the relocation decision. This would have shown how we should allocate nest relocation management on Cumberland Island. Our model’s performance reveals that our feature selection would need a lot of improvements in order to best answer this question. Averaging vertical land movement across long periods of time as a metric for erosion could contribute to the inaccuracy of our model. Additionally, we did not have much washover data for all nests across the island. Light pollution is also a major consideration for sea turtle nest management, yet we were not able to include this in a csv format. Given these constraints to our model, I would recommend that decision makers still consider washovers and land erosion when deciding to relocate a nest, yet also take into account these other factors.