Hillforts Primer

An Analysis of the Atlas of Hillforts of Britain and Ireland

Appendix 2

Mike Middleton
https://orcid.org/0000-0001-5813-6347

Version 1.0, March 2024.
This research was begun in March 2022.

Part 1: Name, Admin & Location Data

Colab Notebook: Live code (Must be logged into Google. Select Google Colaboratory, at the top of the screen, if page opens as raw code)
HTML: Read only

Part 2: Management & Landscape

Colab Notebook: Live code
HTML: Read only

Part 3: Boundary & Dating

Colab Notebook: Live code
HTML: Read only

- Boundary Data
- Dating Data

Part 4: Investigations & Interior

Colab Notebook: Live code
HTML: Read only

Part 5: Entrance, Enclosing & Annex

Colab Notebook: Live code
HTML: Read only

Appendix 1: Hypotheses Testing the Alignment of Hillforts with an Area of 21 Hectares or More

Colab Notebook: Live code
HTML: Read only

Appendix 2: Classification Northwest

Colab Notebook: Live code
HTML: Read only

User Settings

Pre-processed data and images are available for download (without the need to run the code in these files) here: https://github.com/MikeDairsie/Hillforts-Primer.

To review only confirmed hillforts (see Part 1: Status, Data Reliability), download, save images or to change the background image to show the topography, first save a copy of this document into your Google Drive folder. Once saved, change, confirmed_only, download_data, save_images and/or show_topography to **True** in the code blocks below, **Save** and then select **Runtime>Run all** in the main menu above to rerun the code. If selected, running the code will initiate the download and saving of files. Each document will download a number of data packages and you may be prompted to **allow** multiple downloads. Be patient, downloads may take a little time after the document has finished running. Note that each part of the Hillforts Primer is independent and the download, save_image and show_topography variables will need to be enabled in each document, if this functionality is required. Also note that saving images will activate the Google Drive folder and this will request the user to **allow** access. Selecting show_topography will change the background image to a colour topographic map. It should also be noted that, if set to True, this view will only show the distribution of the data selected. It will not show the overall distribution as a grey background layer as is seen when using the simple coastal outlines.

In [1]:
```
confirmed_only = False
```

In [2]:
```
download_data = False
```

In [3]:
```
save_images = False
```

In [4]:
```
show_topography = False
```

In [5]:
```
# Percentage above which Appendix 02 plots will display
# minimum is 10%
show_percentage = 20
```

Bypass Code Setup

The initial sections of all the Hillforts Primer documents set up the coding environment and define functions used to plot, reprocess and save the data. If you would like to bypass the setup, please use the following link:

Go to Review Data Appendix 2.

Reload Data and Python Functions

Source Data

The data was downloaded from The Atlas of Hillforts of Britain and Ireland website as a csv file (comma separated values) and saved onto the author's GitHub repository thus enabling the data to be used by this document.

Lock, G. and Ralston, I. 2017. Atlas of Hillforts of Britain and Ireland. [ONLINE] Available at: https://hillforts.arch.ox.ac.uk
Rest services: https://maps.arch.ox.ac.uk/server/rest/services/hillforts/Atlas_of_Hillforts/MapServer
Licence: https://creativecommons.org/licenses/by-sa/4.0/
Help: https://hillforts.arch.ox.ac.uk/assets/help.pdf
Data Structure: https://maps.arch.ox.ac.uk/assets/data.html
Hillforts: Britain, Ireland and the Nearer Continent (Sample):
https://www.archaeopress.com/ArchaeopressShop/DMS/A72C523E8B6742ED97BA86470E747C69/9781789692266-sample.pdf

Map outlines made with Natural Earth. Free vector and raster map data @ naturalearthdata.com.

Python Modules and Code Setup

In [6]:
```
import sys
print(f'Python: {sys.version}')

import sklearn
print(f'Scikit-Learn: {sklearn.__version__}')

import pandas as pd
print(f'pandas: {pd.__version__}')

import numpy as np
```

```
print(f'numpy: {np.__version__}')

%matplotlib inline
import matplotlib
print(f'matplotlib: {matplotlib.__version__}')
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.patches as mpatches
import matplotlib.patches as patches
from matplotlib.cbook import boxplot_stats
from matplotlib.lines import Line2D
import matplotlib.cm as cm

import seaborn as sns
print(f'seaborn: {sns.__version__}')
sns.set(style="whitegrid")

import scipy
print(f'scipy: {scipy.__version__}')
from scipy import stats
from scipy.stats import gaussian_kde

import os
import collections
import math
import random
import PIL
import urllib
random.seed(42) # A random seed is used to ensure that the random numbers
# created are the same for each run of this document.

from slugify import slugify

# Import Google colab tools to access Drive
from google.colab import drive
```

```
Python: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Scikit-Learn: 1.2.2
pandas: 1.5.3
numpy: 1.25.2
matplotlib: 3.7.1
seaborn: 0.13.1
scipy: 1.11.4
```

Ref: https://www.python.org/

Ref: https://scikit-learn.org/stable/

Ref: https://pandas.pydata.org/docs/

Ref: https://numpy.org/doc/stable/

Ref: https://matplotlib.org/

Ref: https://seaborn.pydata.org/

Ref: https://docs.scipy.org/doc/scipy/index.html

Ref: https://pypi.org/project/python-slugify/

Plot Figures and Maps functions

The following functions will be used to plot data later in the document.

In [7]:
```
def show_records(plt, plot_data):
    text_colour = 'k'
    if show_topography == True:
        text_colour = 'w'
    plt.annotate(str(len(plot_data))+' records', xy=(-1180000, 6420000), \
                 xycoords='data', ha='left', color=text_colour)
```

In [8]:
```
def get_backgrounds():
    if show_topography == True:
        backgrounds = ["hillforts-topo-01.png",
                       "hillforts-topo-north.png",
                       "hillforts-topo-northwest-plus.png",
                       "hillforts-topo-northwest-minus.png",
                       "hillforts-topo-northeast.png",
                       "hillforts-topo-south.png",
                       "hillforts-topo-south-plus.png",
                       "hillforts-topo-ireland.png",
                       "hillforts-topo-ireland-north.png",
                       "hillforts-topo-ireland-south.png"]
    else:
        backgrounds = ["hillforts-outline-01.png",
                       "hillforts-outline-north.png",
                       "hillforts-outline-northwest-plus.png",
                       "hillforts-outline-northwest-minus.png",
```

```
                    "hillforts-outline-northeast.png",
                    "hillforts-outline-south.png",
                    "hillforts-outline-south-plus.png",
                    "hillforts-outline-ireland.png",
                    "hillforts-outline-ireland-north.png",
                    "hillforts-outline-ireland-south.png"]
    return backgrounds
```

In [9]:
```
def get_bounds():
    bounds = [[-1200000,220000,6400000,8700000],
    [-1200000,220000,7000000,8700000],
    [-1200000,-480000,7000000,8200000],
    [-900000,-480000,7100000,8200000],
    [-520000, 0,7000000,8700000],
    [-800000,220000,6400000,7100000],
    [-1200000,220000,6400000,7100000],
    [-1200000,-600000,6650000,7450000],
    [-1200000,-600000,7050000,7450000],
    [-1200000,-600000,6650000,7080000]]
    return bounds
```

In [10]:
```
def show_background(plt, ax, location=""):
    backgrounds = get_backgrounds()
    bounds = get_bounds()
    folder = "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    #"https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/
    #hillforts-topo/"

    if location == "n":
        background = os.path.join(folder, backgrounds[1])
        bounds = bounds[1]
    elif location == "nw+":
        background = os.path.join(folder, backgrounds[2])
        bounds = bounds[2]
    elif location == "nw-":
        background = os.path.join(folder, backgrounds[3])
        bounds = bounds[3]
    elif location == "ne":
        background = os.path.join(folder, backgrounds[4])
        bounds = bounds[4]
    elif location == "s":
        background = os.path.join(folder, backgrounds[5])
        bounds = bounds[5]
    elif location == "s+":
        background = os.path.join(folder, backgrounds[6])
        bounds = bounds[6]
    elif location == "i":
        background = os.path.join(folder, backgrounds[7])
        bounds = bounds[7]
    elif location == "in":
        background = os.path.join(folder, backgrounds[8])
        bounds = bounds[8]
    elif location == "is":
        background = os.path.join(folder, backgrounds[9])
        bounds = bounds[9]
    else:
        background = os.path.join(folder, backgrounds[0])
        bounds = bounds[0]

    img = np.array(PIL.Image.open(urllib.request.urlopen(background)))
    ax.imshow(img, extent=bounds)
```

In [11]:
```
def get_counts(data):
    data_counts = []
    for col in data.columns:
        count = len(data[data[col] == 'Yes'])
        data_counts.append(count)
    return data_counts
```

In [12]:
```
def add_annotation_plot(ax):
    ax.annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
                color='grey', xy=(0.01, 0.01), xycoords='figure fraction', \
                horizontalalignment = 'left')
    ax.annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
                size='small', color='grey', xy=(0.99, 0.01), \
                xycoords='figure fraction', horizontalalignment = 'right')
```

In [13]:
```
def add_annotation_l_xy(ax):
    ax.annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
                color='grey', xy=(0.01, 0.035), xycoords='figure fraction', \
                horizontalalignment = 'left')
    ax.annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
                size='small', color='grey', xy=(0.99, 0.035), \
                xycoords='figure fraction', horizontalalignment = 'right')
```

In [14]:
```
def plot_bar_chart(data, split_pos, x_label, y_label, title):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    x_data = data.columns
    x_data = [x.split("_")[split_pos:] for x in x_data]
    x_data_new = []
    for l in x_data :
        txt =  ""
        for part in l:
            txt += "_" + part
        x_data_new.append(txt[1:])
    y_data = get_counts(data)
    ax.bar(x_data_new,y_data)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    add_annotation_plot(ax)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [15]:
```
def plot_bar_chart_using_two_tables(x_data, y_data, x_label, y_label, title):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    ax.bar(x_data,y_data)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    add_annotation_plot(ax)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [16]:
```
def plot_bar_chart_numeric(data, split_pos, x_label, y_label, title, n_bins):
    new_data = data.copy()
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    data[x_label].plot(kind='hist', bins = n_bins)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    add_annotation_plot(ax)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [17]:
```
def plot_bar_chart_value_counts(data, x_label, y_label, title):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    df = data.value_counts()
    x_data = df.index.values
    y_data = df.values
    ax.bar(x_data,y_data)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    add_annotation_plot(ax)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [18]:
```
def get_bins(data, bins_count):
    data_range = data.max() - data.min()
    print(bins_count)
    if bins_count != None:
        x_bins = [x for x in range(data.min(), data.max(), bins_count)]
        n_bins = len(x_bins)
    else:
        n_bins = int(data_range)
        if n_bins < 10:
            multi = 10
            while n_bins< 10:
                multi *= 10
                n_bins = int(data_range * multi)
        elif n_bins > 100:
            n_bins = int(data_range)/10

    return n_bins
```

In [19]:
```
def plot_histogram(data, x_label, title, bins_count = None):
    n_bins = get_bins(data, bins_count)
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel(x_label)
    ax.set_ylabel('Count')
    plt.ticklabel_format(style='plain')
```

```
    plt.hist(data, bins=n_bins)
    plt.title(get_print_title(title))
    add_annotation_plot(ax)
    save_fig(title)
    plt.show()
```

In [20]:
```
def plot_continuous(data, x_lable, title):
    fig = plt.figure(figsize=(12,8))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel(x_lable)
    plt.plot(data, linewidth=4)
    plt.ticklabel_format(style='plain')
    plt.title(get_print_title(title))
    add_annotation_plot(ax)
    save_fig(title)
    plt.show()
```

In [21]:
```
# box plot
from matplotlib.cbook import boxplot_stats
def plot_data_range(data, feature, o="v"):
    fig = plt.figure(figsize=(12,8))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel(feature)
    add_annotation_plot(ax)
    plt.title(get_print_title(feature + " Range"))
    plt.ticklabel_format(style='plain')
    if o == "v":
        sns.boxplot(data=data, orient="v")
    else:
        sns.boxplot(data=data, orient="h")
    save_fig(feature + " Range")
    plt.show()

    bp = boxplot_stats(data)

    low = bp[0].get('whislo')
    q1 = bp[0].get('q1')
    median =  bp[0].get('med')
    q3 = bp[0].get('q3')
    high = bp[0].get('whishi')

    return [low, q1, median, q3, high]
```

In [22]:
```
def location_XY_plot():
    plt.ticklabel_format(style='plain')
    plt.xlim(-1200000,220000)
    plt.ylim(6400000,8700000)
    add_annotation_l_xy(plt)
```

In [23]:
```
def add_grey(region=''):
    if show_topography == False:
        # plots all the hillforts as a grey background
        loc = location_data.copy()
        if region == 's':
            loc = loc[loc['Location_Y'] < 8000000].copy()
            loc = loc[loc['Location_X'] > -710000].copy()
        elif region == 'ne':
            loc = loc[loc['Location_Y'] < 8000000].copy()
            loc = loc[loc['Location_X'] > -800000].copy()

        plt.scatter(loc['Location_X'], loc['Location_Y'], c='Silver')
```

In [24]:
```
def plot_over_grey_numeric(merged_data, a_type, title, extra="", inner=False, \
                           fringe=False, oxford=False,swindon=False):
    plot_data = merged_data
    fig, ax = plt.subplots(figsize=(14.2 * 0.66, 23.0 * 0.66))
    show_background(plt, ax)
    location_XY_plot()
    add_grey()
    patches = add_oxford_swindon(oxford,swindon)
    plt.scatter(plot_data['Location_X'], plot_data['Location_Y'], c='Red')
    if fringe:
        f_for_legend = add_21Ha_fringe()
        patches.append(f_for_legend)
    if inner:
        i_for_legend = add_21Ha_line()
        patches.append(i_for_legend)
    show_records(plt, plot_data)
    plt.legend(loc='upper left', handles= patches)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [25]:
```
def plot_over_grey_boundary(merged_data, a_type, boundary_type):
    plot_data = merged_data[merged_data[a_type] == boundary_type]
    fig, ax = plt.subplots(figsize=(9.47, 15.33))
    show_background(plt, ax)
    location_XY_plot()
    add_grey(region='')
    plt.scatter(plot_data['Location_X'], plot_data['Location_Y'], c='Red')
    show_records(plt, plot_data)
    plt.title(get_print_title('Boundary_Type: ' + boundary_type))
    save_fig('Boundary_Type_' + boundary_type)
    plt.show()
    print(f'{round((len(plot_data)/len(merged_data)*100), 2)}%')
```

In [26]:
```
def plot_density_over_grey(data, data_type):
    new_data = data.copy()
    new_data = new_data.drop(['Density'], axis=1)
    new_data = add_density(new_data)
    fig, ax = plt.subplots(figsize=((14.2 * 0.66)+2.4, 23.0 * 0.66))
    show_background(plt, ax)
    location_XY_plot()
    add_grey()
    plt.scatter(new_data['Location_X'], new_data['Location_Y'], \
                c=new_data['Density'], cmap=cm.rainbow, s=25)
    plt.colorbar(label='Density')
    plt.title(get_print_title(f'Density - {data_type}'))
    save_fig(f'Density_{data_type}')
    plt.show()
```

In [27]:
```
def add_21Ha_line():
    x_values = \
     [-367969, -344171, -263690, -194654, -130542, -119597, -162994, -265052]#, -304545]
    y_values = \
     [7019842, 6944572, 6850593, 6779602, 6735058, 6710127, 6684152, 6663609]#, 6611780]
    plt.plot(x_values, y_values, 'k', ls='-', lw=15, alpha=0.25, \
             label = '≥ 21 Ha Line')
    add_to_legend = \
    Line2D([0], [0], color='k', lw=15, alpha=0.25, label = '≥ 21 Ha Line')
    return add_to_legend
```

In [28]:
```
def add_21Ha_fringe():
    x_values = \
     [-367969,-126771,29679,-42657,-248650,-304545,-423647,-584307,-367969]
    y_values = \
     [7019842,6847138,6671658,6596650,6554366,6611780,6662041,6752378,7019842]
    plt.plot(x_values, y_values, 'k', ls=':', lw=5, alpha=0.45, \
             label = '≥ 21 Ha Fringe')
    add_to_legend = \
    Line2D([0], [0], color='k', ls=':', lw=5, alpha=0.45, label = '≥ 21 Ha Fringe')
    return add_to_legend
```

In [29]:
```
def add_oxford_swindon(oxford=False,swindon=False):
    # plots a circle over Swindon & Oxford
    radius = 50
    marker_size = (2*radius)**2
    patches = []
    if oxford:
        plt.scatter(-144362,6758380, c='dodgerblue', s=marker_size, alpha=0.50)
        b_patch = mpatches.Patch(color='dodgerblue', label='Oxford orbit')
        patches.append(b_patch)
    if swindon:
        plt.scatter(-197416, 6721977, c='yellow', s=marker_size, alpha=0.50)
        y_patch = mpatches.Patch(color='yellow', label='Swindon orbit')
        patches.append(y_patch)
    return patches
```

In [30]:
```
def plot_over_grey(merged_data, a_type, yes_no, extra="", \
                   inner=False, fringe=False, oxford=False,swindon=False):
    # plots selected data over the grey dots. yes_no controlls filtering
    #the data for a positive or negative values.
    plot_data = merged_data[merged_data[a_type] == yes_no]
    fig, ax = plt.subplots(figsize=(14.2 * 0.66, 23.0 * 0.66))
    show_background(plt, ax)
    location_XY_plot()
    add_grey()
    patches = add_oxford_swindon(oxford,swindon)
    plt.scatter(plot_data['Location_X'], plot_data['Location_Y'], c='Red')
    if fringe:
        f_for_legend = add_21Ha_fringe()
        patches.append(f_for_legend)
    if inner:
        i_for_legend = add_21Ha_line()
        patches.append(i_for_legend)
    show_records(plt, plot_data)
```

```
    plt.legend(loc='upper left', handles= patches)
    plt.title(get_print_title(f'{a_type} {extra}'))
    save_fig(f'{a_type}_{extra}')
    plt.show()
    print(f'{round((len(plot_data)/len(merged_data)*100), 2)}%')
    return plot_data
```

In [31]:
```
def plot_type_values(data, data_type, title):
    new_data = data.copy()
    fig, ax = plt.subplots(figsize=((14.2 * 0.66)+2.4, 23.0 * 0.66))
    show_background(plt, ax)
    location_XY_plot()
    plt.scatter(new_data['Location_X'], new_data['Location_Y'], \
                c=new_data[data_type], cmap=cm.rainbow, s=25)
    plt.colorbar(label=data_type)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [32]:
```
def bespoke_plot(plt, title):
    add_annotation_plot(plt)
    plt.ticklabel_format(style='plain')
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [33]:
```
def get_proportions(date_set):
    total = sum(date_set) - date_set[-1]
    newset = []
    for entry in date_set[:-1]:
        newset.append(round(entry/total,2))
    return newset
```

In [34]:
```
def plot_dates_by_region(nw,ne,ni,si,s, features):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    x_data = nw[features].columns
    x_data = [x.split("_")[2:] for x in x_data][:-1]
    x_data_new = []
    for l in x_data:
        txt =  ""
        for part in l:
            txt += "_" + part
        x_data_new.append(txt[1:])

    set1_name = 'NW'
    set2_name = 'NE'
    set3_name = 'N Ireland'
    set4_name = 'S Ireland'
    set5_name = 'South'
    set1 = get_proportions(get_counts(nw[features]))
    set2 = get_proportions(get_counts(ne[features]))
    set3 = get_proportions(get_counts(ni[features]))
    set4 = get_proportions(get_counts(si[features]))
    set5 = get_proportions(get_counts(s[features]))

    X_axis = np.arange(len(x_data_new))

    budge = 0.25

    plt.bar(X_axis - 0.55 + budge, set1, 0.3, label = set1_name)
    plt.bar(X_axis - 0.4 + budge, set2, 0.3, label = set2_name)
    plt.bar(X_axis - 0.25 + budge, set3, 0.3, label = set3_name)
    plt.bar(X_axis - 0.1 + budge, set4, 0.3, label = set4_name)
    plt.bar(X_axis + 0.05 + budge, set5, 0.3, label = set5_name)

    plt.xticks(X_axis, x_data_new)
    plt.xlabel('Dating')
    plt.ylabel('Proportion of Total Dated Hillforts in Region')
    title = 'Proportions of Dated Hillforts by Region'
    plt.title(title)
    plt.legend()
    add_annotation_plot(ax)
    save_fig(title)
    plt.show()
```

Review Data Functions

The following functions will be used to confirm that features are not lost or forgotten when splitting the data.

In [35]:
```
def test_numeric(data):
    temp_data = data.copy()
```

```
    columns = data.columns
    out_cols = ['Feature','Entries', 'Numeric', 'Non-Numeric', 'Null']
    feat, ent, num, non, nul = [],[],[],[],[]
    for col in columns:
        if temp_data[col].dtype == 'object':
            feat.append(col)
            temp_data[col+'_num'] = temp_data[col].str.isnumeric()
            entries = temp_data[col].notnull().sum()
            true_count = temp_data[col+'_num'][temp_data[col+'_num'] == True].sum()
            null_count = temp_data[col].isna().sum()
            ent.append(entries)
            num.append(true_count)
            non.append(entries-true_count)
            nul.append(null_count)
        else:
            print(f'{col} {temp_data[col].dtype}')
    summary = pd.DataFrame(list(zip(feat, ent, num, non, nul)))
    summary.columns = out_cols
    return summary
```

In [36]:
```
def find_duplicated(numeric_data, text_data, encodeable_data):
    d = False
    all_columns = \
    list(numeric_data.columns) + list(text_data.columns) + \
    list(encodeable_data.columns)
    duplicate = \
     [item for item, count in collections.Counter(all_columns).items() \
      if count > 1]
    if duplicate :
        print(f"There are duplicate features: {duplicate}")
        d = True
    return d
```

In [37]:
```
def test_data_split(main_data, numeric_data, text_data, encodeable_data):
    m = False
    split_features = \
    list(numeric_data.columns) + list(text_data.columns) + \
    list(encodeable_data.columns)
    missing = list(set(main_data)-set(split_features))
    if missing:
        print(f"There are missing features: {missing}")
        m = True
    return m
```

In [38]:
```
def review_data_split(main_data, numeric_data, text_data, \
                      encodeable_data = pd.DataFrame()):
    d = find_duplicated(numeric_data, text_data, encodeable_data)
    m = test_data_split(main_data, numeric_data, text_data, encodeable_data)
    if d != True and m != True:
        print("Data split good.")
```

In [39]:
```
def find_duplicates(data):
    print(f'{data.count() - data.duplicated(keep=False).count()} duplicates.')
```

In [40]:
```
def count_yes(data):
    total = 0
    for col in data.columns:
        count = len(data[data[col] == 'Yes'])
        total+= count
        print(f'{col}: {count}')
    print(f'Total yes count: {total}')
```

Null Value Functions

The following functions will be used to update null values.

In [41]:
```
def fill_nan_with_minus_one(data, feature):
    new_data = data.copy()
    new_data[feature] = data[feature].fillna(-1)
    return new_data
```

In [42]:
```
def fill_nan_with_NA(data, feature):
    new_data = data.copy()
    new_data[feature] = data[feature].fillna("NA")
    return new_data
```

In [43]:
```
def test_numeric_value_in_feature(feature, value):
    test = feature.isin([-1]).sum()
    return test
```

In [44]:
```
def test_catagorical_value_in_feature(dataframe, feature, value):
    test = dataframe[feature][dataframe[feature] == value].count()
    return test
```

In [45]:
```
def test_cat_list_for_NA(dataframe, cat_list):
    for val in cat_list:
        print(val, test_catagorical_value_in_feature(dataframe, val,'NA'))
```

In [46]:
```
def test_num_list_for_minus_one(dataframe, num_list):
    for val in num_list:
        feature = dataframe[val]
        print(val, test_numeric_value_in_feature(feature, -1))
```

In [47]:
```
def update_cat_list_for_NA(dataframe, cat_list):
    new_data = dataframe.copy()
    for val in cat_list:
        new_data = fill_nan_with_NA(new_data, val)
    return new_data
```

In [48]:
```
def update_num_list_for_minus_one(dataframe, cat_list):
    new_data = dataframe.copy()
    for val in cat_list:
        new_data = fill_nan_with_minus_one(new_data, val)
    return new_data
```

Reprocessing Functions

In [49]:
```
def add_density(data):
    new_data = data.copy()
    xy = np.vstack([new_data['Location_X'],new_data['Location_Y']])
    new_data['Density'] = gaussian_kde(xy)(xy)
    return new_data
```

Save Image Functions

In [50]:
```
fig_no = 0
part = 'Appendix02'
IMAGES_PATH = r'/content/drive/My Drive/'
fig_list = pd.DataFrame(columns=['fig_no', 'file_name', 'title'])
topo_txt = ""
if show_topography:
    topo_txt = "-topo"
```

In [51]:
```
def get_file_name(title):
    file_name = slugify(title)
    return file_name
```

In [52]:
```
def get_print_title(title):
    title = title.replace("_", " ")
    title = title.replace("-", " ")
    title = title.replace(",", ";")
    return title
```

In [53]:
```
def format_figno(no):
    length = len(str(no))
    fig_no = ''
    for i in range(3-length):
        fig_no = fig_no + '0'
    fig_no = fig_no + str(no)
    return fig_no
```

In [54]:
```
if save_images == True:
    drive.mount('/content/drive')
    os.getcwd()
else:
    pass
```

In [55]:
```
def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    global fig_no
    global IMAGES_PATH
    if save_images:
        #IMAGES_PATH = r'/content/drive/My Drive/Colab Notebooks/Hillforts_Primer_Images/HP_Appendix_02_images/'
        fig_no+=1
        fig_no_txt = format_figno(fig_no)
        file_name = file_name = get_file_name(f'{part}_{fig_no_txt}')
        file_name = f'hillforts_primer_{file_name}{topo_txt}.{fig_extension}'
        fig_list.loc[len(fig_list)] = [fig_no, file_name, get_print_title(fig_id)]
        path = os.path.join(IMAGES_PATH, file_name)
```

```
        print("Saving figure", file_name)
        plt.tight_layout()
        plt.savefig(path, format=fig_extension, dpi=resolution,  bbox_inches='tight')
    else:
        pass
```

Load Data

The source csv file is loaded and the first two rows are displayed to confirm the load was successful. Note that, to the left, an index has been added automatically. This index will be used frequently when splitting and remerging data extracts.

In [56]:
```
hillforts_csv = r"https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-atlas-source-data-cs
#r"https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/
#hillforts-atlas-source-data-csv/hillforts.csv"
hillforts_data = pd.read_csv(hillforts_csv, index_col=False)
pd.set_option('display.max_columns', None, 'display.max_rows', None)
hillforts_data.head(2)
```

```
<ipython-input-56-03f8272c1daf>:4: DtypeWarning: Columns (10,12,68,83,84,85,86,165,183) have mixed types. Specify dt
ype option on import or set low_memory=False.
  hillforts_data = pd.read_csv(hillforts_csv, index_col=False)
```

Out[56]:

	OBJECTID	Main_Atlas_Number	Main_Country_Code	Main_Country	Main_Title_Name	Main_Site_Name	Main_Alt_Name	Main_Display_N
0	1	1	EN	England	EN0001 Aconbury Camp, Herefordshire	Aconbury Camp	Aconbury Beacon	Aconbury C Hereford (Aconbury Bea
1	2	2	EN	England	EN0002 Bach Camp, Herefordshire	Bach Camp	NaN	Bach C Hereford

Filter confirmed (if selected)

If confirmed_only is set to True in User Settings above, this will filter the source data so that it contains only confirmed forts.

In [57]:
```
if confirmed_only == True:
    hillforts_data = \
    hillforts_data[hillforts_data['Status_Interpretation_Reliability'] == \
                   'Confirmed']
    print(f'Data filtered to contain only {len(hillforts_data)} confirmed hillforts.')
else:
    print(f'Using all {len(hillforts_data)} record in the Hillforts Atlas.')
```

```
Using all 4147 record in the Hillforts Atlas.
```

Download Function

In [58]:
```
from google.colab import files
def download(data_list, filename, hf_data=hillforts_data):
    if download_data == True:
        name_and_number = hf_data[['Main_Atlas_Number','Main_Display_Name']].copy()
        dl = name_and_number.copy()
        for pkg in data_list:
            if filename not in ['england', 'wales','scotland', \
                                'republic-of-ireland','norhtern-ireland', \
                                'isle-of-man', 'roi-ni', 'eng-wal-sco-iom']:
                if pkg.shape[0] == hillforts_data.shape[0]:
                    dl = pd.merge(dl, pkg, left_index=True, right_index=True)
            else:
                dl = data_list[0]
        dl = dl.replace('\r',' ', regex=True)
        dl = dl.replace('\n',' ', regex=True)
        fn = 'hillforts_primer_' + filename
        fn = get_file_name(fn)
        dl.to_csv(fn+'.csv', index=False)
        files.download(fn+'.csv')
    else:
        pass
```

Reload Name and Number

The Main Atlas Number and the Main Display Name are the primary unique reference identifiers in the data. With these, users can identify any record numerically and by name. Throughout this document, the data will be clipped into a number of sub-data packages. Where needed, these data extracts will be combined with Name and Number features to ensure the data can be understood and can, if needed, be concorded.

In [59]:
```
name_and_number_features = ['Main_Atlas_Number','Main_Display_Name']
name_and_number = hillforts_data[name_and_number_features].copy()
name_and_number.head()
```

Out[59]:

	Main_Atlas_Number	Main_Display_Name
0	1	Aconbury Camp, Herefordshire (Aconbury Beacon)
1	2	Bach Camp, Herefordshire
2	3	Backbury Camp, Herefordshire (Ethelbert's Camp)
3	4	Brandon Camp, Herefordshire
4	5	British Camp, Herefordshire (Herefordshire Bea...

Reload Location

In [60]:
```
location_numeric_data_short_features = ['Location_X','Location_Y']
location_numeric_data_short = hillforts_data[location_numeric_data_short_features]
location_numeric_data_short = add_density(location_numeric_data_short)
location_numeric_data_short.head()
location_data = location_numeric_data_short.copy()
location_data.head()
```

Out[60]:

	Location_X	Location_Y	Density
0	-303295	6798973	1.632859e-12
1	-296646	6843289	1.540172e-12
2	-289837	6808611	1.547729e-12
3	-320850	6862993	1.670548e-12
4	-261765	6810587	1.369981e-12

Reload Location Cluster Data Packages

In [61]:
```
cluster_data = \
hillforts_data[['Location_X','Location_Y', 'Main_Country_Code']].copy()
cluster_data['Cluster'] = 'NA'
cluster_data['Cluster'].where(cluster_data['Main_Country_Code'] != \
                              'NI', 'I', inplace=True)
cluster_data['Cluster'].where(cluster_data['Main_Country_Code'] != \
                              'IR', 'I', inplace=True)

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'I') & (cluster_data['Location_Y'] >= 7060000) , \
   'North Ireland', cluster_data['Cluster'])
north_ireland = cluster_data[cluster_data['Cluster'] == 'North Ireland'].copy()

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'I') & (cluster_data['Location_Y'] < 7060000) , \
   'South Ireland', cluster_data['Cluster'])
south_ireland = cluster_data[cluster_data['Cluster'] == 'South Ireland'].copy()

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'NA') & (cluster_data['Location_Y'] < 7070000) , \
   'South', cluster_data['Cluster'])
south = cluster_data[cluster_data['Cluster'] == 'South'].copy()

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'NA') & (cluster_data['Location_Y'] >= 7070000) & \
   (cluster_data['Location_X'] >= -500000), 'Northeast', cluster_data['Cluster'])
north_east = cluster_data[cluster_data['Cluster'] == 'Northeast'].copy()

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'NA') & (cluster_data['Location_Y'] >= 7070000) & \
    (cluster_data['Location_X'] < -500000), 'Northwest', cluster_data['Cluster'])
north_west = cluster_data[cluster_data['Cluster'] == 'Northwest'].copy()

temp_cluster_location_packages = \
 [north_ireland, south_ireland, south, north_east, north_west]

cluster_packages = []
```

```
for pkg in temp_cluster_location_packages:
    pkg = pkg.drop(['Main_Country_Code'], axis=1)
    cluster_packages.append(pkg)

north_ireland, south_ireland, south, north_east, north_west = \
cluster_packages[0], cluster_packages[1], cluster_packages[2], \
cluster_packages[3], cluster_packages[4]
```

Classification Northwest

Reload density function

In [62]:
```
def renew_density(data):
    new_data = data.copy()
    try:
        new_data = new_data.drop(['Density'], axis=1)
    except:
        pass
    try:
        new_data = new_data.drop(['Density_trans'], axis=1)
    except:
        pass
    xy = np.vstack([new_data['Location_X'],new_data['Location_Y']])
    new_data['Density'] = gaussian_kde(xy)(xy)
    new_data['Density_trans'] = stats.boxcox(new_data['Density'], 0.5)
    return new_data
```

Refresh density values

In [63]:
```
north_west = renew_density(north_west)
```

Northwest plot functions

In [64]:
```
def plot_north_west_dnesity_minus(show_eildon, cluster_north_west, \
                                  location_X_cluster_north_west, \
                                  location_Y_cluster_north_west):
    fig, ax = plt.subplots(figsize=((2.772 * 1.75)+2.0, (7.26 * 1.75)))
    show_background(plt, ax, 'nw-')
    plt.ticklabel_format(style='plain')
    plt.xlim(-900000,-480000)
    plt.ylim(7100000,8200000)
    plt.scatter(cluster_north_west['Location_X'], \
                cluster_north_west['Location_Y'], \
                c=cluster_north_west['Density_trans'], cmap=cm.rainbow, s=25)
    rect3, rect4, vline2, xline2 = get_rectangles(location_X_cluster_north_west, \
                                                  location_Y_cluster_north_west, \
                                                  3, 1.5)
    ax.add_patch(rect3)
    ax.add_patch(rect4)
    plt.vlines(x=[vline2[1]], ymin=vline2[0], ymax=vline2[2], colors='purple', \
               ls='--', lw=1.5)
    plt.hlines(y=[xline2[1]], xmin=xline2[0], xmax=xline2[2], colors='purple', \
               ls='--', lw=1.5)
    plt.plot(-609918,7575512 ,'ko')
    if show_eildon:
        ax.annotate('SC2466: Dunadd', xy=(-609918,7575512), xycoords='data', \
                    ha='left', xytext=(-110, -135), textcoords='offset points', \
                    arrowprops=dict(arrowstyle="->", color='k', lw=1, \
                                    connectionstyle="arc3,rad=-0.2"))
    plt.colorbar(label='Density Transformed')
    add_annotation_plot(plt)
    title = 'Density Trans & Boxplot (NW minus Ireland)'
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

In [65]:
```
def plot_data_range_plus(data, feature, o=None):
    if o == None:
        pass
    else:
        fig = plt.figure(figsize=(12,8))
        ax = fig.add_axes([0,0,1,1])
        ax.set_xlabel(feature)
        add_annotation_plot(ax)
        plt.title(get_print_title(feature + " Range"))
        plt.ticklabel_format(style='plain')
```

```
    if o == "v":
        sns.boxplot(data=data, orient="v")
    elif o == None:
        pass
    else:
        sns.boxplot(data=data, orient="h")

    if o == None:
        pass
    else:
        save_fig(feature + " Range")
        plt.show()

    bp = boxplot_stats(data)

    low = bp[0].get('whislo')
    q1 = bp[0].get('q1')
    median =  bp[0].get('med')
    q3 = bp[0].get('q3')
    high = bp[0].get('whishi')

    return [low, q1, median, q3, high]
```

In [66]:
```
def get_rectangles(x_data, y_data, w1, w2):
    x_lo, x_q1, x_median, x_q3, x_hi = x_data[0], x_data[1], x_data[2], \
    x_data[3], x_data[4]
    y_lo, y_q1, y_median, y_q3, y_hi = y_data[0], y_data[1], y_data[2], \
    y_data[3], y_data[4]

    rect = patches.Rectangle((x_lo, y_lo), x_hi-x_lo, y_hi-y_lo, linewidth=w2, \
                             edgecolor='k', facecolor='none')
    rect2 = patches.Rectangle((x_q1, y_q1), x_q3-x_q1, y_q3-y_q1, linewidth=w1, \
                              edgecolor='r', facecolor='none')

    return rect, rect2, [y_q1, x_median, y_q3], [x_q1, y_median, x_q3]
```

In [67]:
```
def plot_north_west_density_for_column(show_eildon, regional_data, cluster, \
                                       column, location_X_cluster, \
                                       location_Y_cluster, show_boxes = False):
    fig, ax = plt.subplots(figsize=((2.772 * 1.75)+2.0, (7.26 * 1.75)))
    show_background(plt, ax, 'nw-')
    plt.ticklabel_format(style='plain')
    plt.xlim(-900000,-480000)
    plt.ylim(7100000,8200000)
    plt.scatter(cluster['Location_X'], \
                cluster['Location_Y'], \
                c=cluster['Density_trans'], cmap=cm.rainbow, s=25)
    if show_boxes:
        rect3, rect4, vline2, xline2 = get_rectangles(location_X_cluster, \
                                                      location_Y_cluster, \
                                                      3, 1.5)
        ax.add_patch(rect3)
        ax.add_patch(rect4)
        plt.vlines(x=[vline2[1]], ymin=vline2[0], ymax=vline2[2], \
                   colors='purple', ls='--', lw=1.5)
        plt.hlines(y=[xline2[1]], xmin=xline2[0], xmax=xline2[2], \
                   colors='purple', ls='--', lw=1.5)
        plt.plot(-609918,7575512 ,'ko')
    if show_eildon:
        ax.annotate('SC2466: Dunadd', xy=(-609918,7575512), xycoords='data', \
                    ha='left', xytext=(-110, -135), textcoords='offset points', \
                    arrowprops=dict(arrowstyle="->", color='k', lw=1, \
                                    connectionstyle="arc3,rad=-0.2"))
    plt.colorbar(label='Density Transformed')
    add_annotation_plot(plt)
    title = column
    text_colour = 'k'
    plt.annotate(str(round((len(cluster) / len(regional_data)) * 100, 1))+ "%", \
                 xy=(-880000, 7150000), xycoords='data', ha='left', \
                 color=text_colour)
    plt.annotate(str(len(cluster))+'/'+str(len(regional_data)), \
                 xy=(-880000, 7120000), xycoords='data', ha='left', \
                 color=text_colour)
    plt.title(get_print_title(title))
    save_fig("NW_" + title)
    plt.show()
```

In [68]:
```
def plot_encodeable(hillforts_data, north_west, encodeable_features, show_percentage):
    encodeable_data = hillforts_data[encodeable_features].copy()
    location_data = pd.merge(north_west, encodeable_data, \
                                  left_index=True, right_index=True)
    print(len(location_data))
    for column in encodeable_features:
        # Filter data
```

```
        cluster = location_data[location_data[column] == "Yes"]
        show_eildon = False
        if show_percentage < 10:
            show_percentage = 10

        if (len(cluster) / len(location_data)) * 100 > show_percentage:
            # refresh density
            cluster = renew_density(cluster)
            location_X_cluster = \
            plot_data_range_plus(cluster['Location_X'], '', None)
            location_Y_cluster = \
            plot_data_range_plus(cluster['Location_Y'], '', None)
            #print(len(cluster))
            plot_north_west_density_for_column(show_eildon, location_data, \
                                               cluster, column, location_X_cluster, \
                                               location_Y_cluster, False)
```

In [69]:
```
def plot_altitude(hillforts_data, north_west, single_data_set, single_column, altitude, show_percentage):
    # Filter data
    location_data = pd.merge(north_west, single_data_set, \
                                left_index=True, right_index=True)
    print(len(location_data))
    new_col_name = "Landscape_Altitude" + "_" + altitude
    location_data.columns = ["Location_X","Location_Y","Cluster","Density","Density_trans", "Location_X_y","Location
    cluster = location_data[location_data[new_col_name] == "Yes"]
    show_eildon = False
    if show_percentage < 10:
        show_percentage = 10

    if (len(cluster) / len(north_west)) * 100 > show_percentage:
        # refresh density
        cluster = renew_density(cluster)
        location_X_cluster = \
        plot_data_range_plus(cluster['Location_X'], '', None)
        location_Y_cluster = \
        plot_data_range_plus(cluster['Location_Y'], '', None)
        #print(len(cluster))
        plot_north_west_density_for_column(show_eildon, north_west, \
                                           cluster, new_col_name, location_X_cluster, \
                                           location_Y_cluster, False)
```

In [70]:
```
def plot_numeric(hillforts_data, north_west, numeric_features, show_percentage):
    numeric_data = hillforts_data[numeric_features].copy()
    location_data = pd.merge(north_west, numeric_data, \
                                left_index=True, right_index=True)
    print(len(location_data))
    for column in numeric_features:
        unique_values = sorted(location_data[column ].dropna().unique())
        print(unique_values)
        for val in unique_values:

            #Filter data
            cluster = location_data[location_data[column] == val]
            new_col_name = column + "_" + str(val)
            cluster = cluster.rename(columns={column: new_col_name})
            show_eildon = False
            if show_percentage < 10:
                show_percentage = 10

            if (len(cluster) / len(location_data)) * 100 > show_percentage:
                # refresh density
                cluster = renew_density(cluster)
                location_X_cluster = \
                plot_data_range_plus(cluster['Location_X'], '', None)
                location_Y_cluster = \
                plot_data_range_plus(cluster['Location_Y'], '', None)
                #print(len(cluster))
                plot_north_west_density_for_column(show_eildon, location_data, \
                                                   cluster, new_col_name, location_X_cluster, \
                                                   location_Y_cluster, False)
```

In [71]:
```
cluster_north_west = add_density(north_west)
cluster_north_west['Density_trans'] = \
stats.boxcox(cluster_north_west['Density'], 0.5)
```

In [72]:
```
location_X_cluster_north_west = \
plot_data_range_plus(north_west['Location_X'], \
                'Location_X - Northwest (minus Ireland)', None)
```

In [73]:
```
location_Y_cluster_north_west = \
plot_data_range_plus(north_west['Location_Y'], \
                'Location_Y - Northwest (minus Ireland)', None)
```

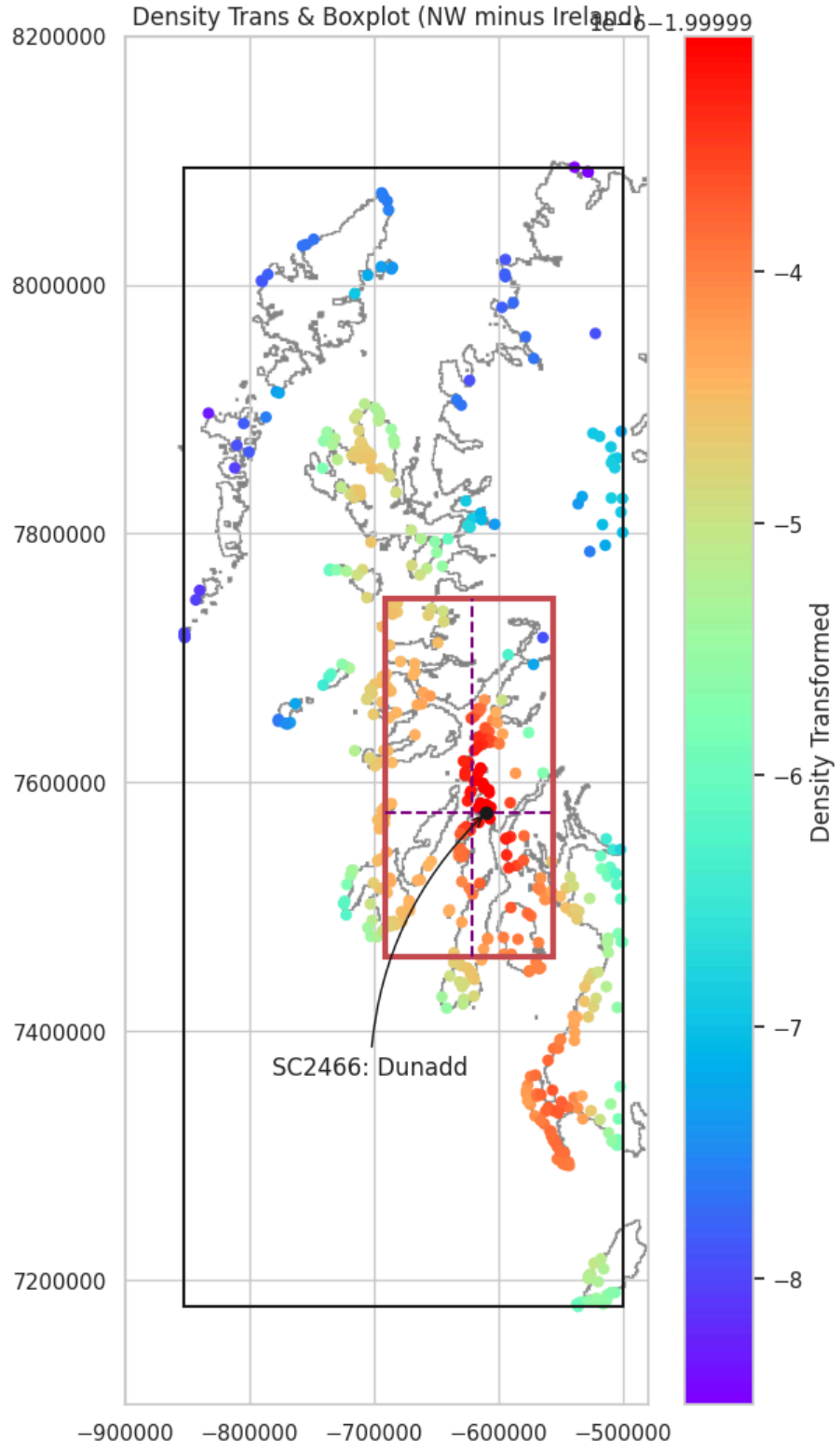
In [74]:
```
show_eildon = True
```

Northwest classification density plots

Images will only be plotted below if the resulting filter of the data contains more sites, as a percentage of all sites in the area, than the show_percentage value set in User Settings above.

Plot Northwest location data

In [75]:
```
plot_north_west_dnesity_minus(show_eildon, north_west, \
                              location_X_cluster_north_west, \
                              location_Y_cluster_north_west)
```



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

In [76]:
```
print(str(len(north_west)) + " Records")
```

```
487 Records
```

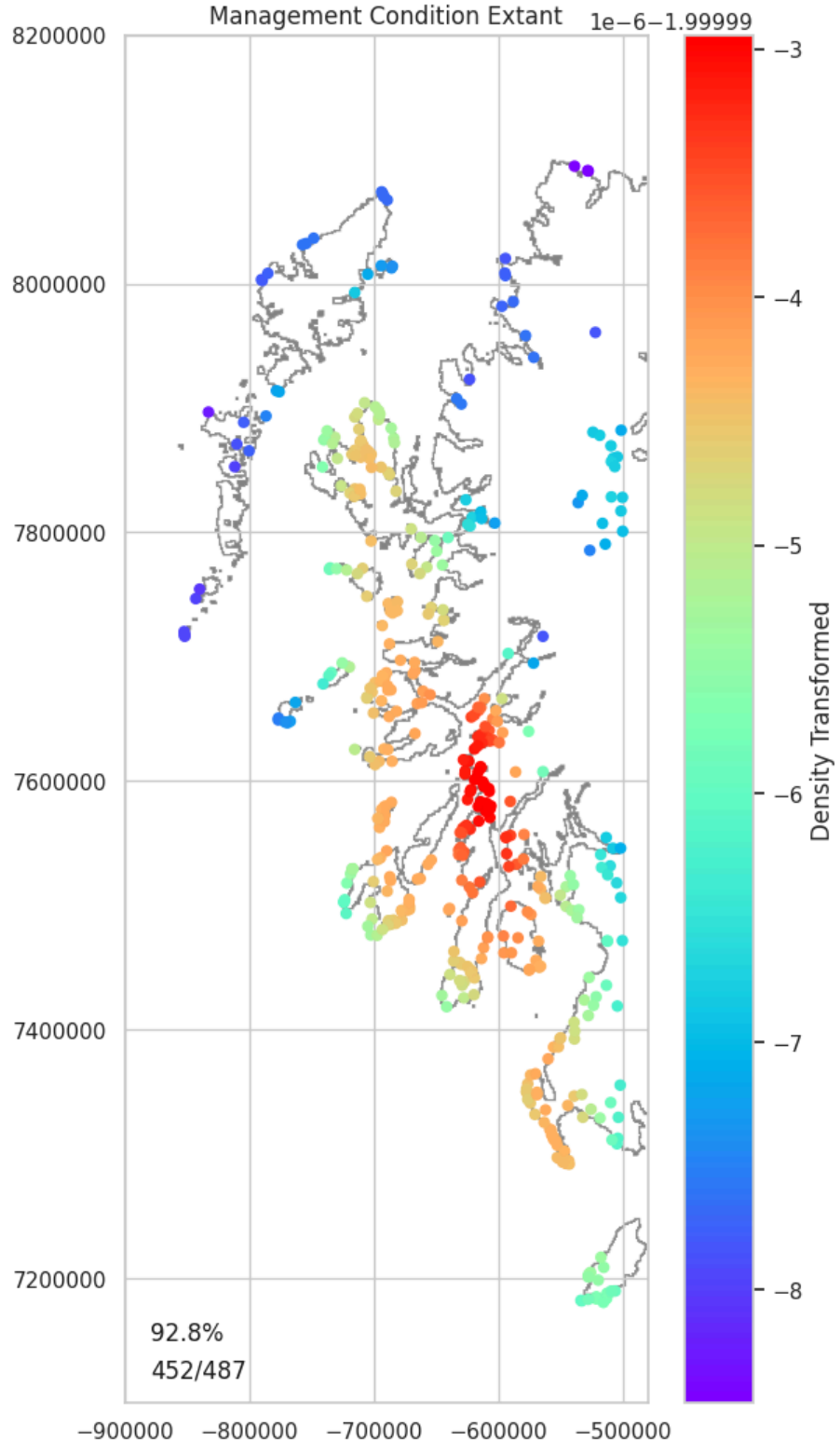
Management Encodeable Data

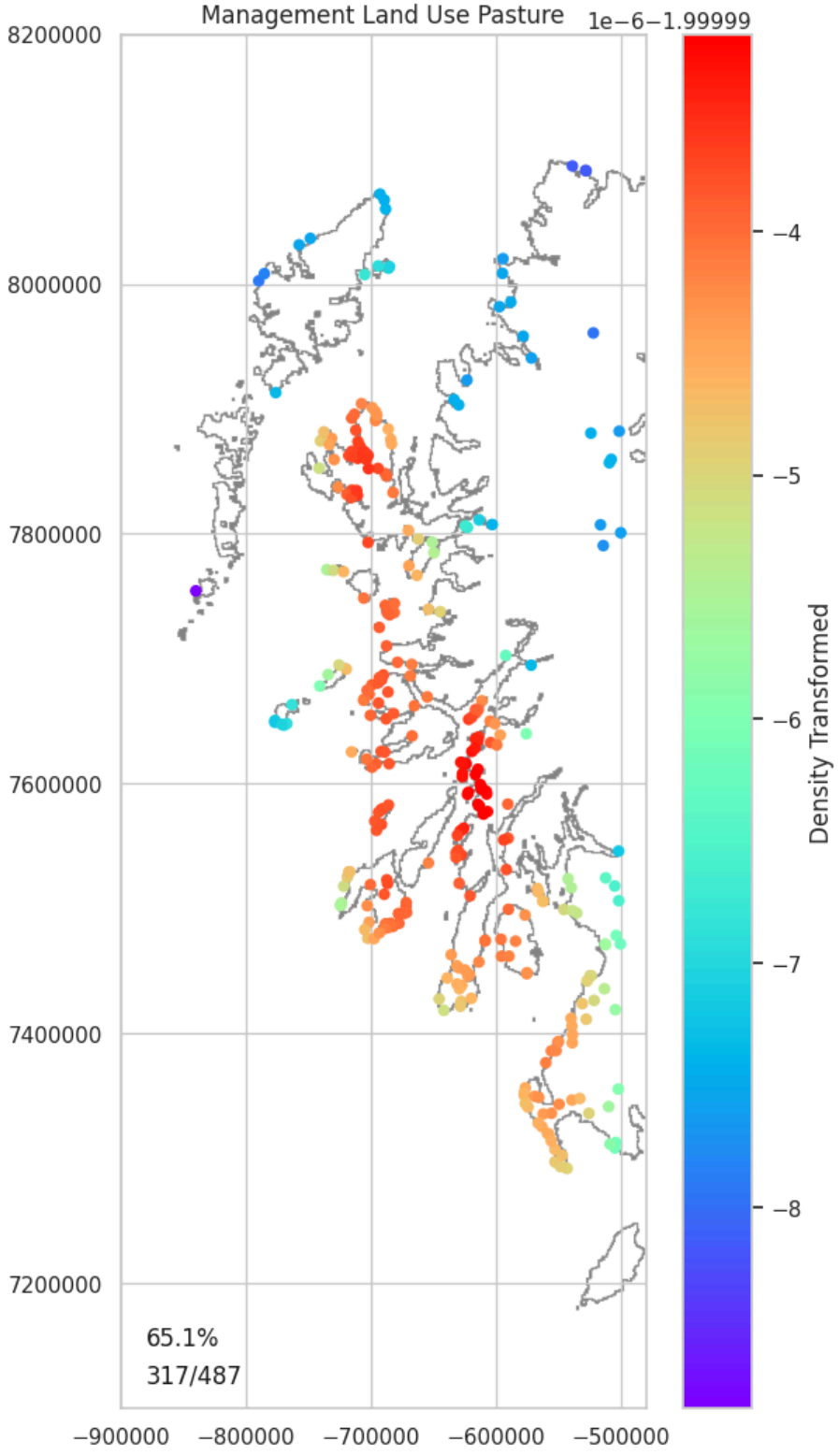
In [77]:
```
management_encodeable_features = [
 'Management_Condition_Extant',
 'Management_Condition_Cropmark',
 'Management_Condition_Destroyed',
 'Management_Land_Use_Woodland',
 'Management_Land_Use_Plantation',
 'Management_Land_Use_Parkland',
```

```
 'Management_Land_Use_Pasture',
 'Management_Land_Use_Arable',
 'Management_Land_Use_Scrub',
 'Management_Land_Use_Outcrop',
 'Management_Land_Use_Moorland',
 'Management_Land_Use_Heath',
 'Management_Land_Use_Urban',
 'Management_Land_Use_Coastal',
 'Management_Land_Use_Other']
```

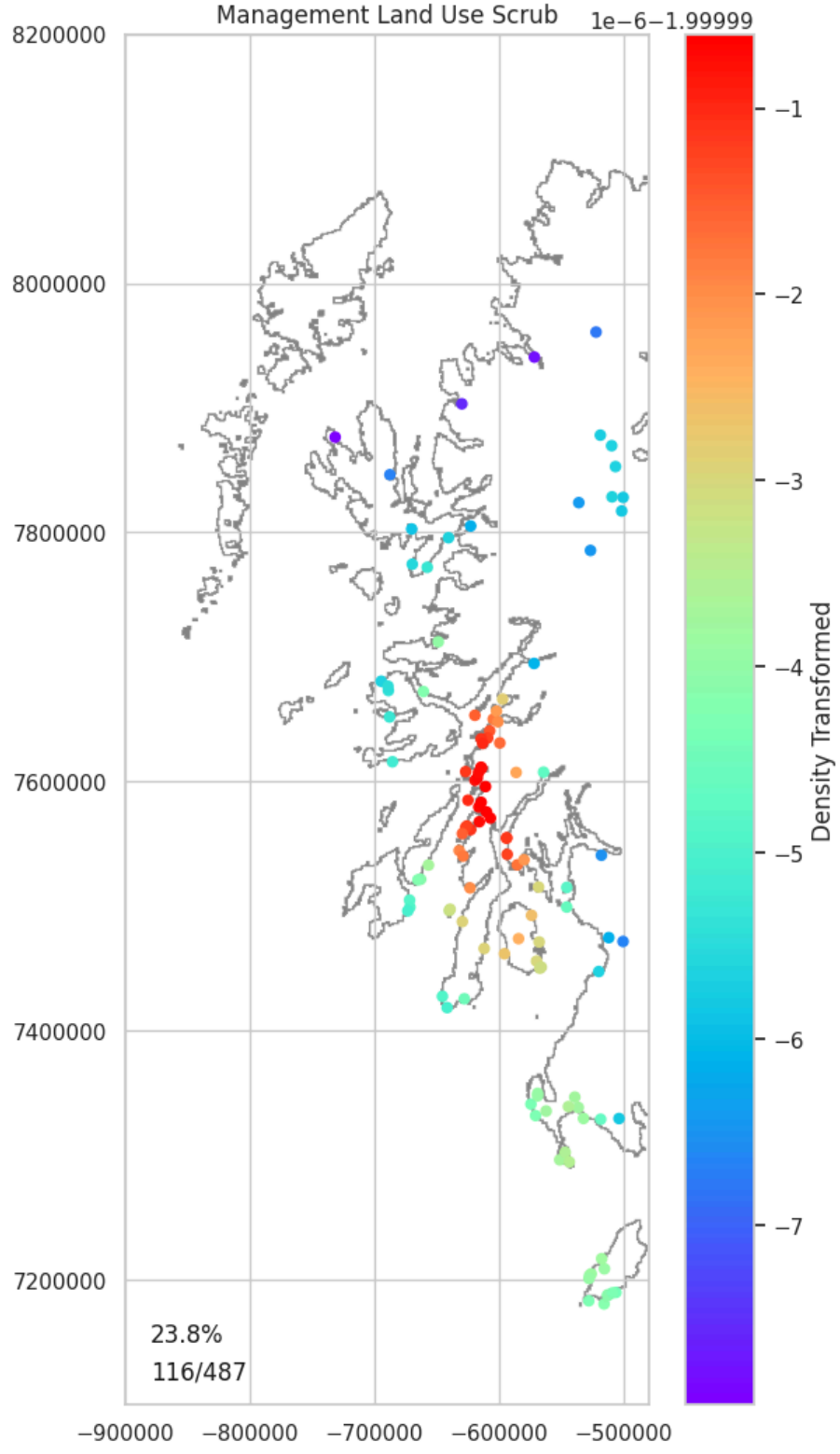
In [78]:
```
plot_encodeable(hillforts_data, north_west, management_encodeable_features, show_percentage)
```

487

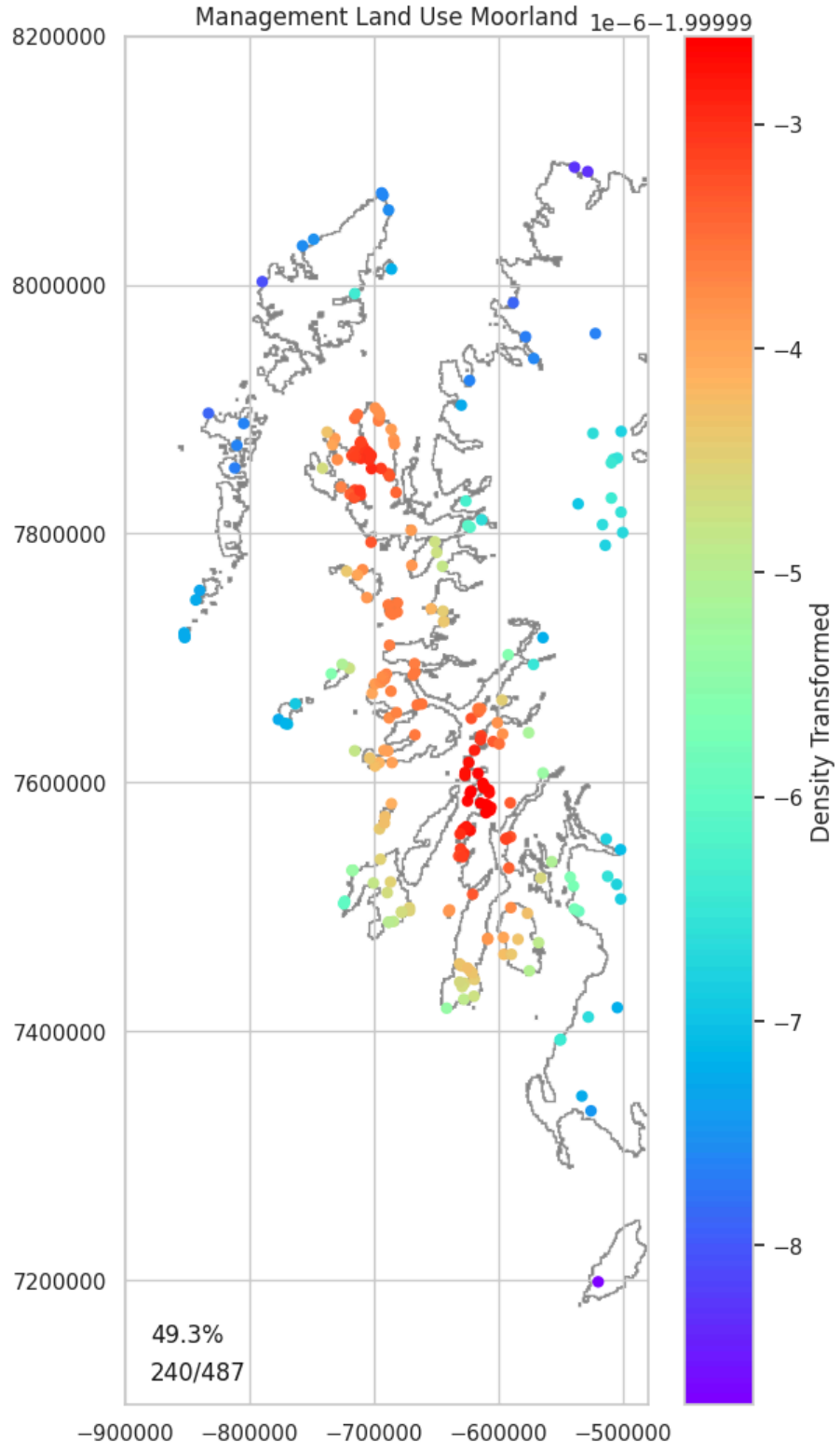




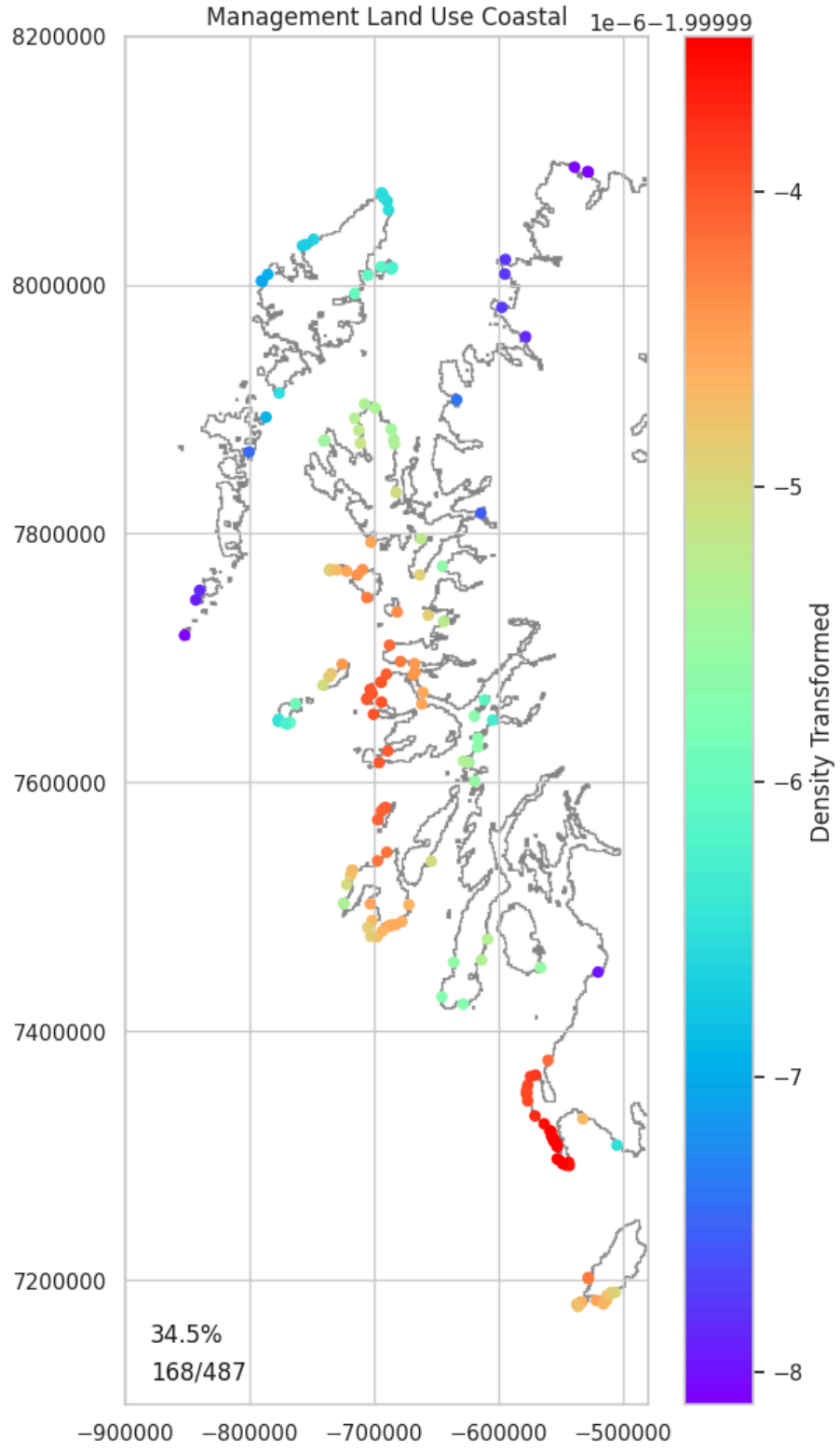
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Landscape Numeric Data

In [79]:
```
landscape_numeric_features = [
 'Landscape_Altitude']
```

In [80]:
```
landscape_numeric_data = \
hillforts_data[landscape_numeric_features].copy()
landscape_numeric_data.head()
```

Out[80]:

	Landscape_Altitude
0	276.0
1	150.0
2	225.0
3	150.0
4	338.0

In [81]:
```
location_landscape_numeric_data = \
pd.merge(location_numeric_data_short, landscape_numeric_data, left_index=True, \
         right_index=True)
```

In [82]:
```
over_300 = \
location_landscape_numeric_data\
 [(location_landscape_numeric_data['Landscape_Altitude'] >= 300) & \
  (location_landscape_numeric_data['Landscape_Altitude'] < 400)].copy()
over_300['Landscape_Altitude'] = "Yes"
```

In [83]:
```
plot_altitude(hillforts_data, north_west, over_300, 'Landscape_Altitude',  "300-400m", show_percentage)
```

```
7
```

In [84]:
```
over_200 = \
location_landscape_numeric_data\
 [(location_landscape_numeric_data['Landscape_Altitude'] >= 200) & \
  (location_landscape_numeric_data['Landscape_Altitude'] < 300)].copy()
over_200['Landscape_Altitude'] = "Yes"
```

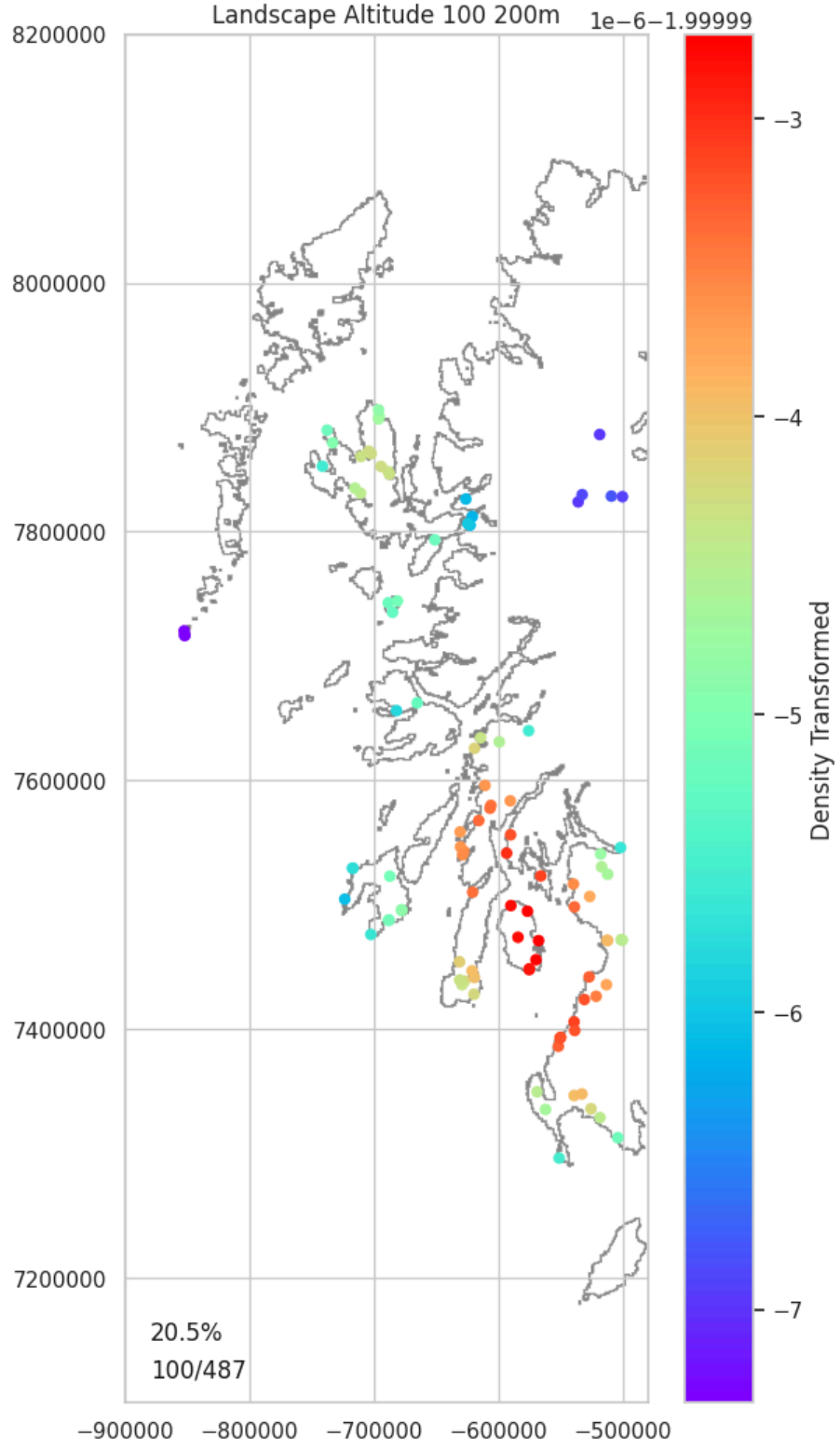
In [85]:
```
plot_altitude(hillforts_data, north_west, over_200, 'Landscape_Altitude', "200-300m", show_percentage)
```

```
23
```

In [86]:
```
over_100 = \
location_landscape_numeric_data\
 [(location_landscape_numeric_data['Landscape_Altitude'] >= 100) & \
  (location_landscape_numeric_data['Landscape_Altitude'] < 200)].copy()
over_100['Landscape_Altitude'] = "Yes"
```

In [87]:
```
plot_altitude(hillforts_data, north_west, over_100, 'Landscape_Altitude',  "100-200m", show_percentage)
```

```
100
```

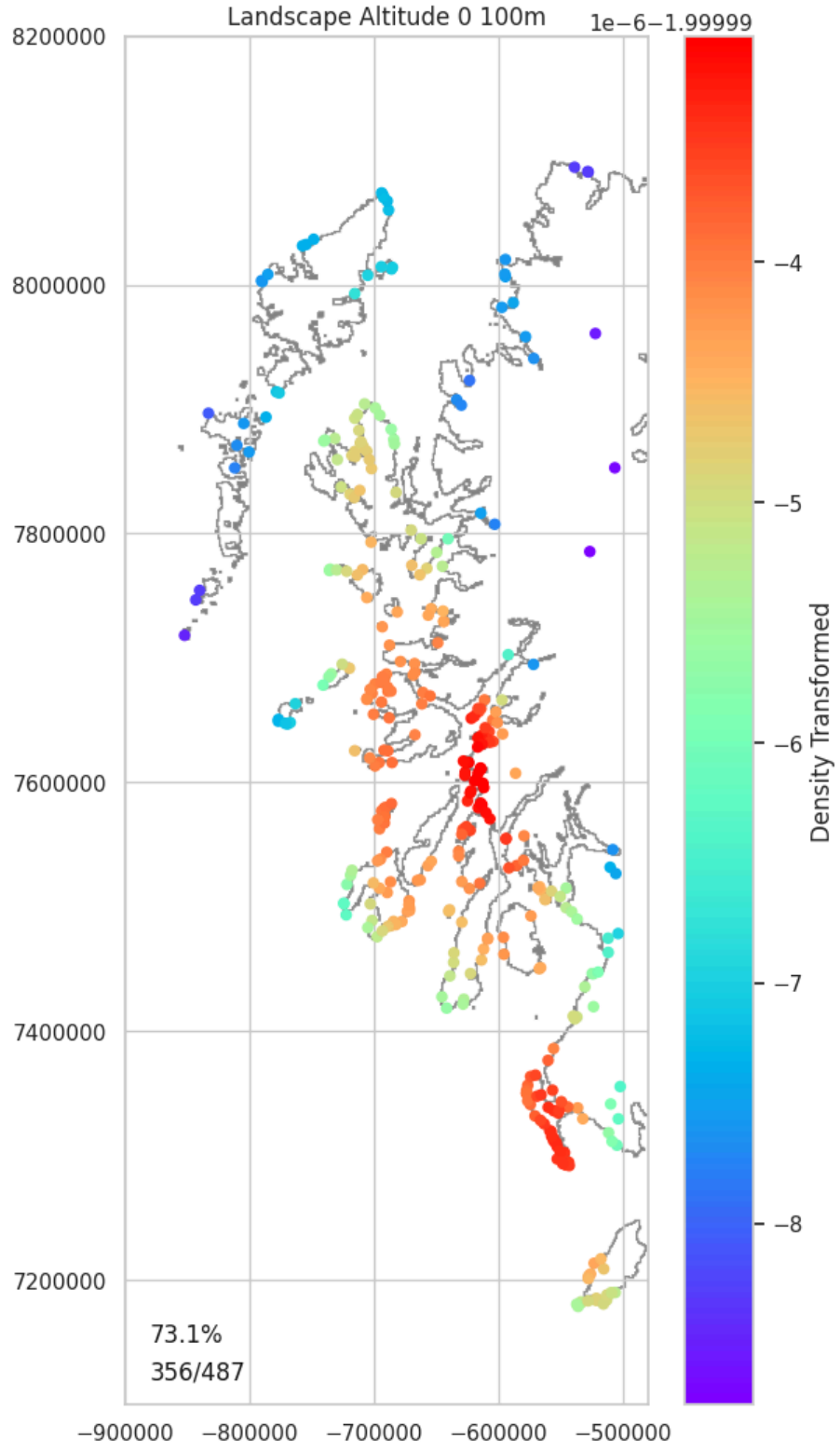


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

In [88]:
```
over_000 = \
location_landscape_numeric_data\
 [(location_landscape_numeric_data['Landscape_Altitude'] >= 0) & \
  (location_landscape_numeric_data['Landscape_Altitude'] < 100)].copy()
over_000['Landscape_Altitude'] = "Yes"
```

In [89]:
```
plot_altitude(hillforts_data, north_west, over_000, 'Landscape_Altitude', "0-100m", show_percentage)
```

```
356
```



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

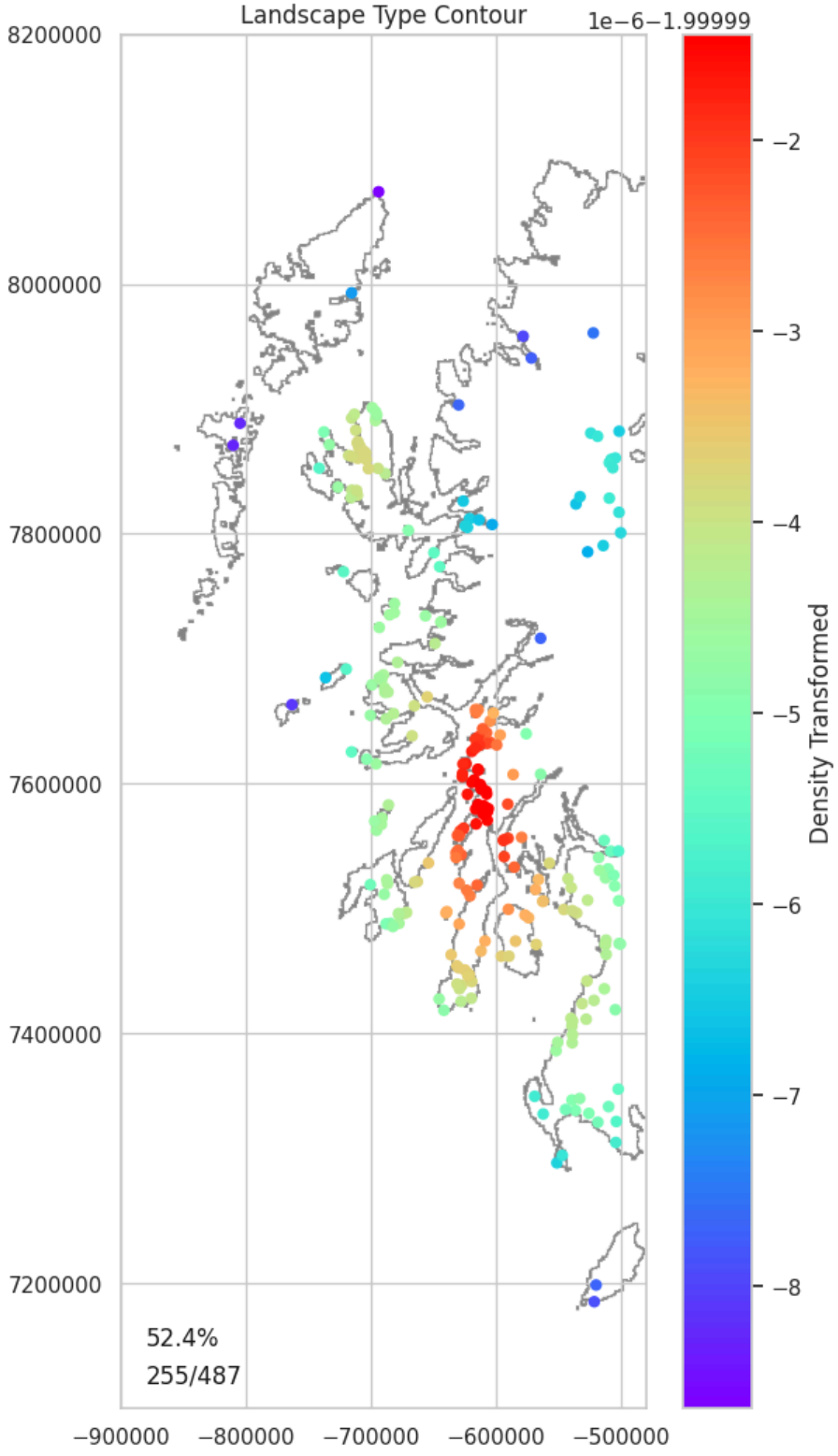
Landscape Encodeable Data

In [90]:
```
landscape_encodeable_features = [
 'Landscape_Type_Contour',
 'Landscape_Type_Partial',
 'Landscape_Type_Promontory',
 'Landscape_Type_Hillslope',
 'Landscape_Type_Level',
 'Landscape_Type_Marsh',
 'Landscape_Type_Multiple',
 'Landscape_Topography_Hilltop',
 'Landscape_Topography_Coastal',
```

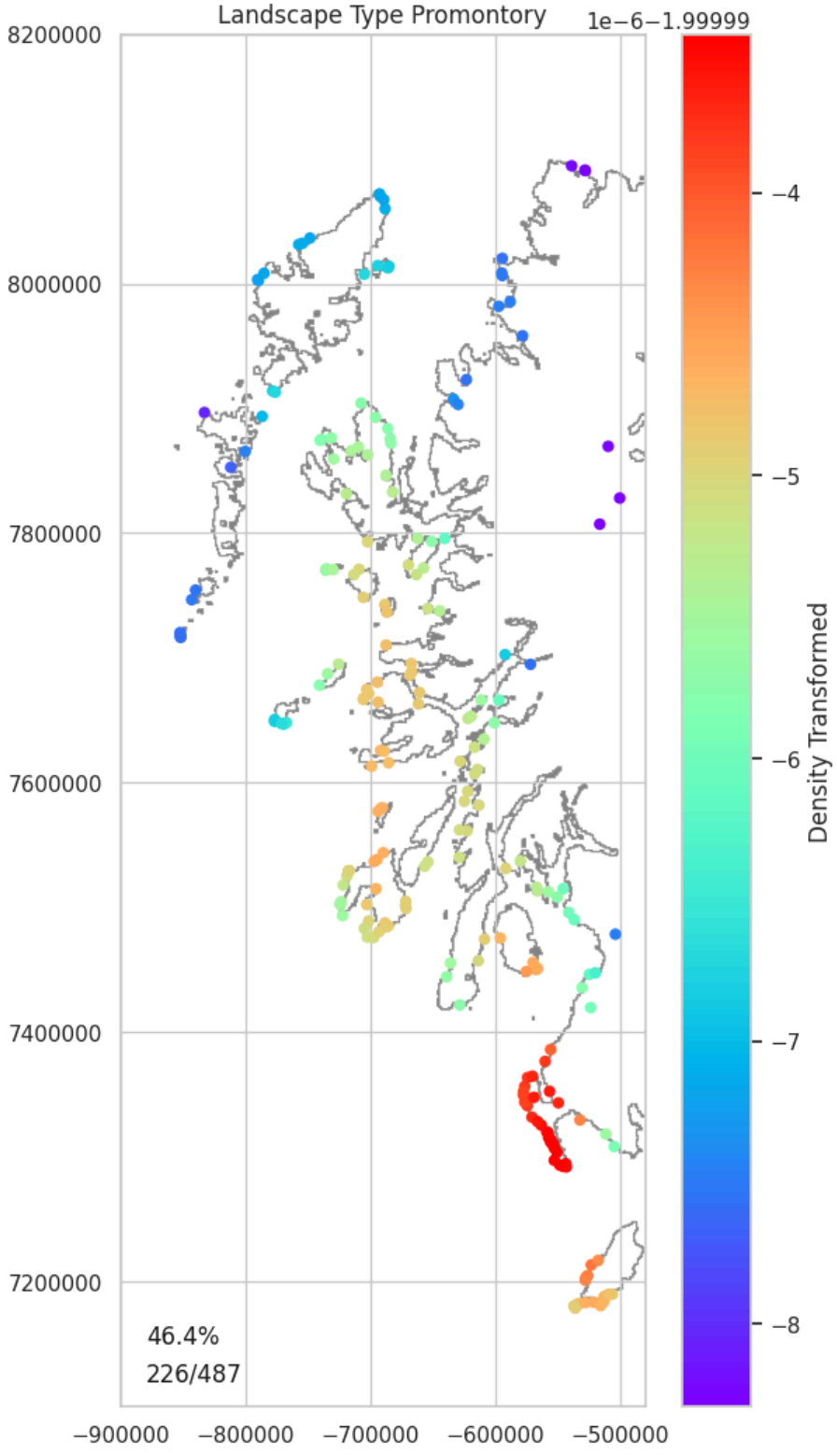
```
 'Landscape_Topography_Inland',
 'Landscape_Topography_Valley',
 'Landscape_Topography_Knoll',
 'Landscape_Topography_Ridge',
 'Landscape_Topography_Scarp',
 'Landscape_Topography_Hillslope',
 'Landscape_Topography_Lowland',
 'Landscape_Topography_Spur',
 'Landscape_Aspect_N',
 'Landscape_Aspect_NE',
 'Landscape_Aspect_E',
 'Landscape_Aspect_SE',
 'Landscape_Aspect_S',
 'Landscape_Aspect_SW',
 'Landscape_Aspect_W',
 'Landscape_Aspect_NW',
 'Landscape_Aspect_Level']
```

In [91]:
```
plot_encodeable(hillforts_data, north_west, landscape_encodeable_features, show_percentage)
```

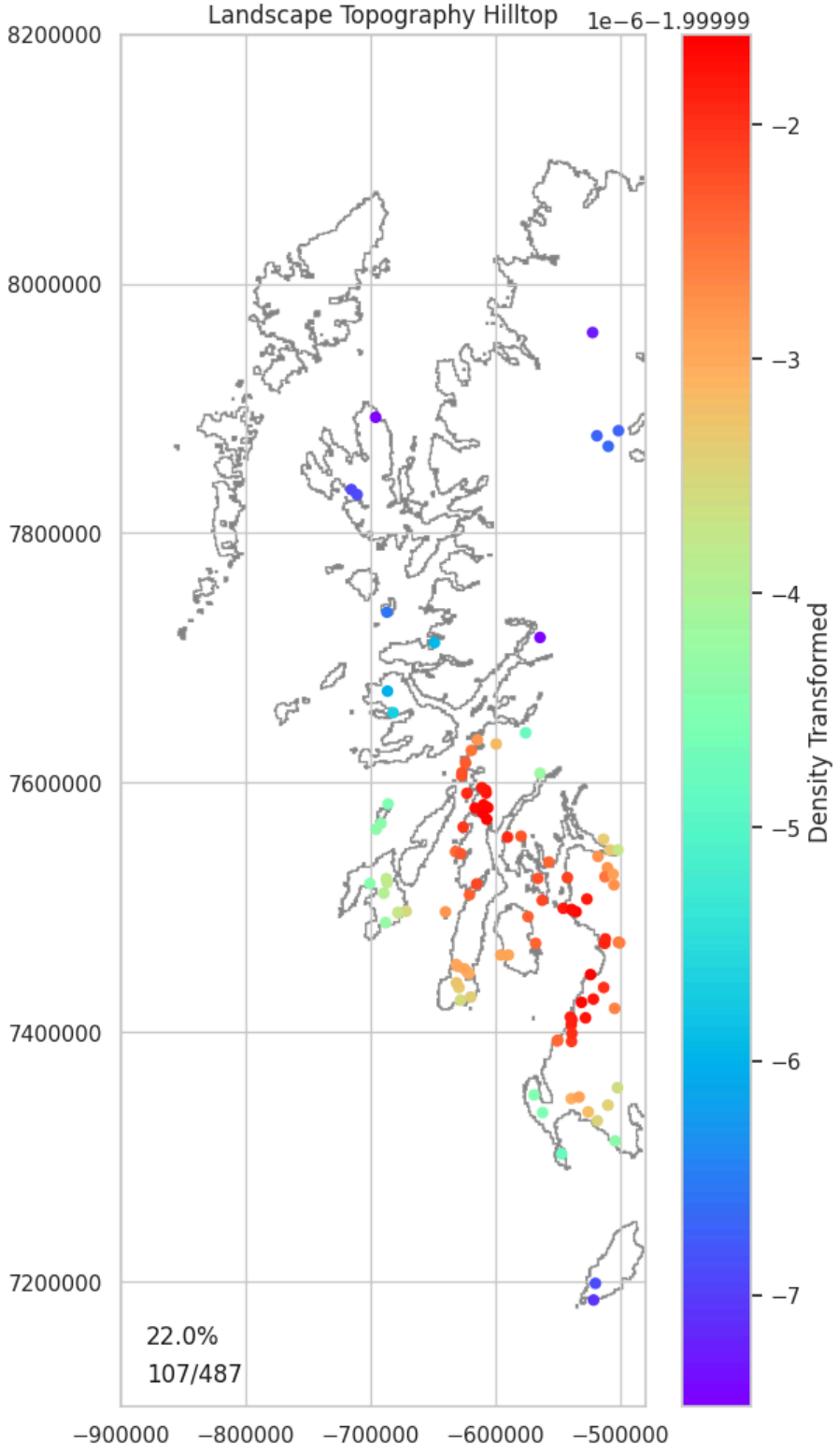
```
487
```



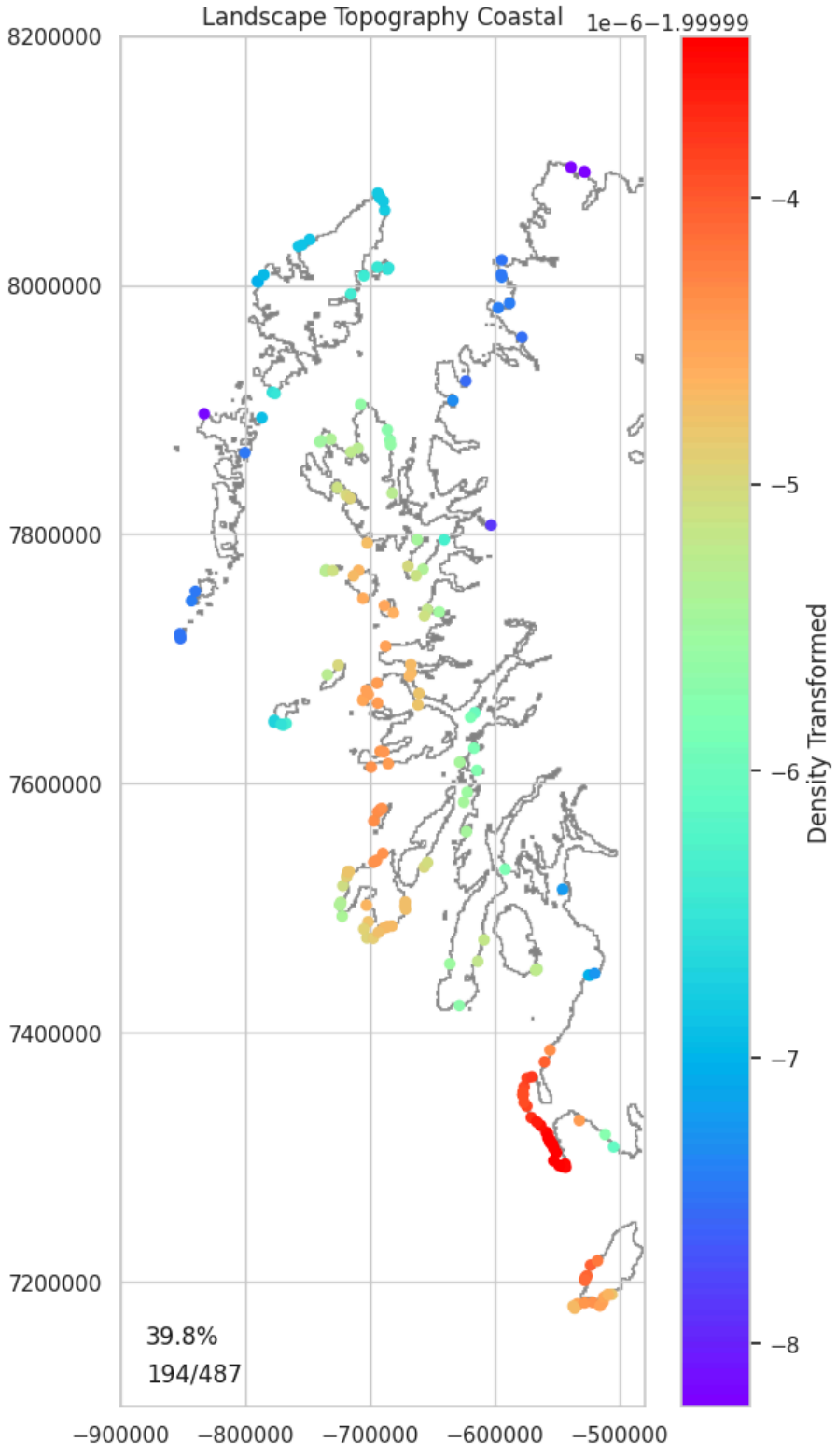
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



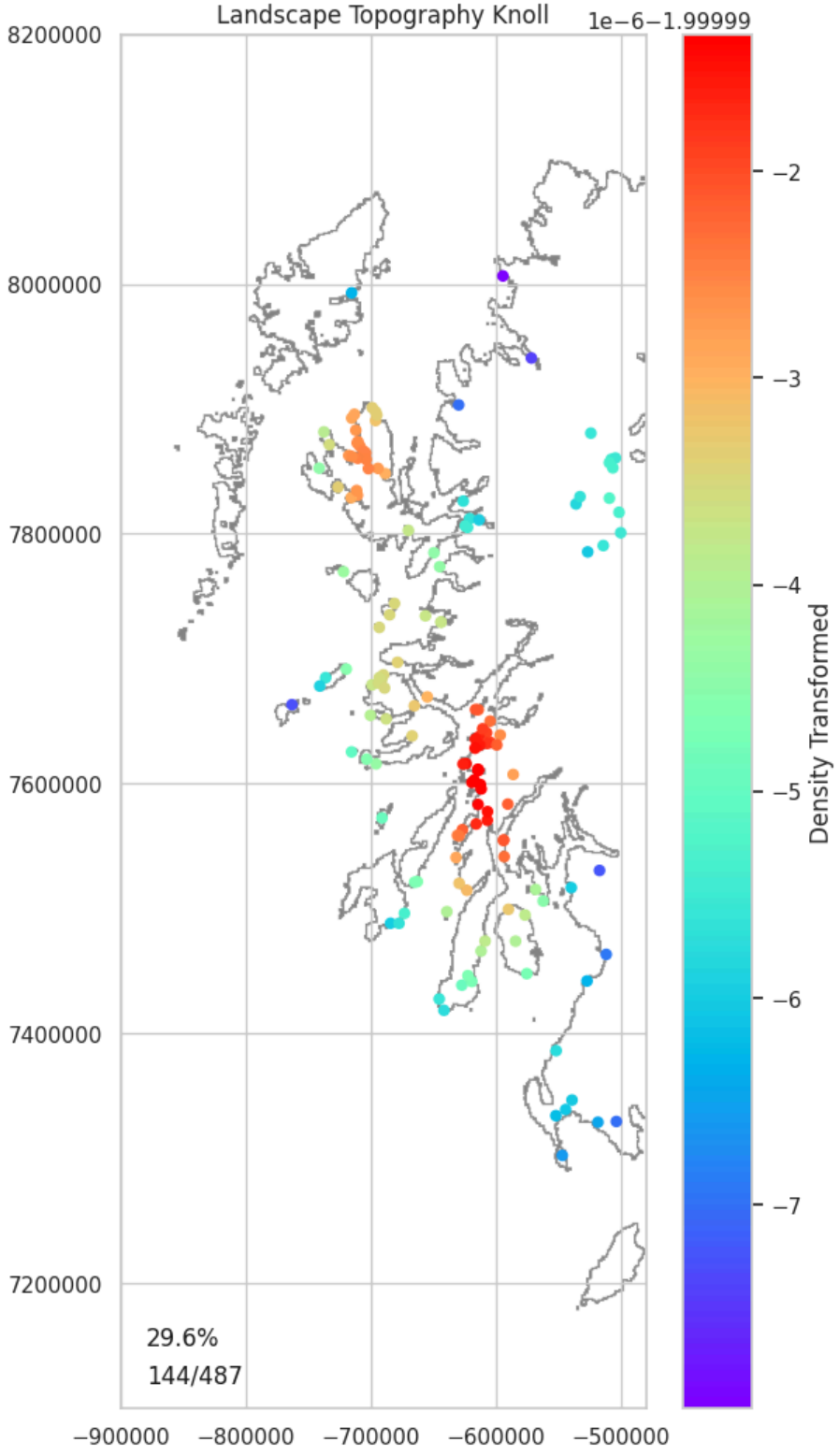
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



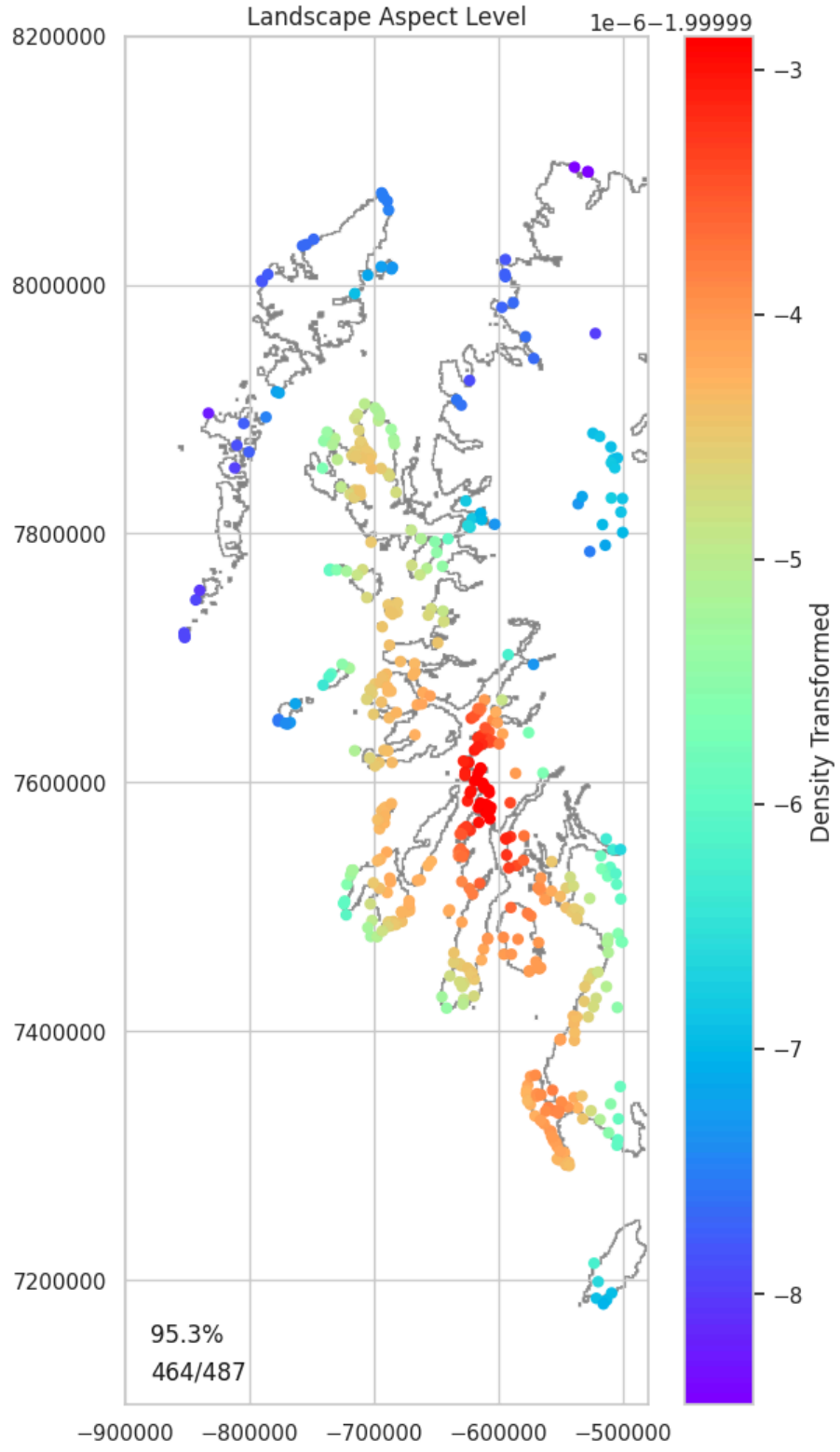
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



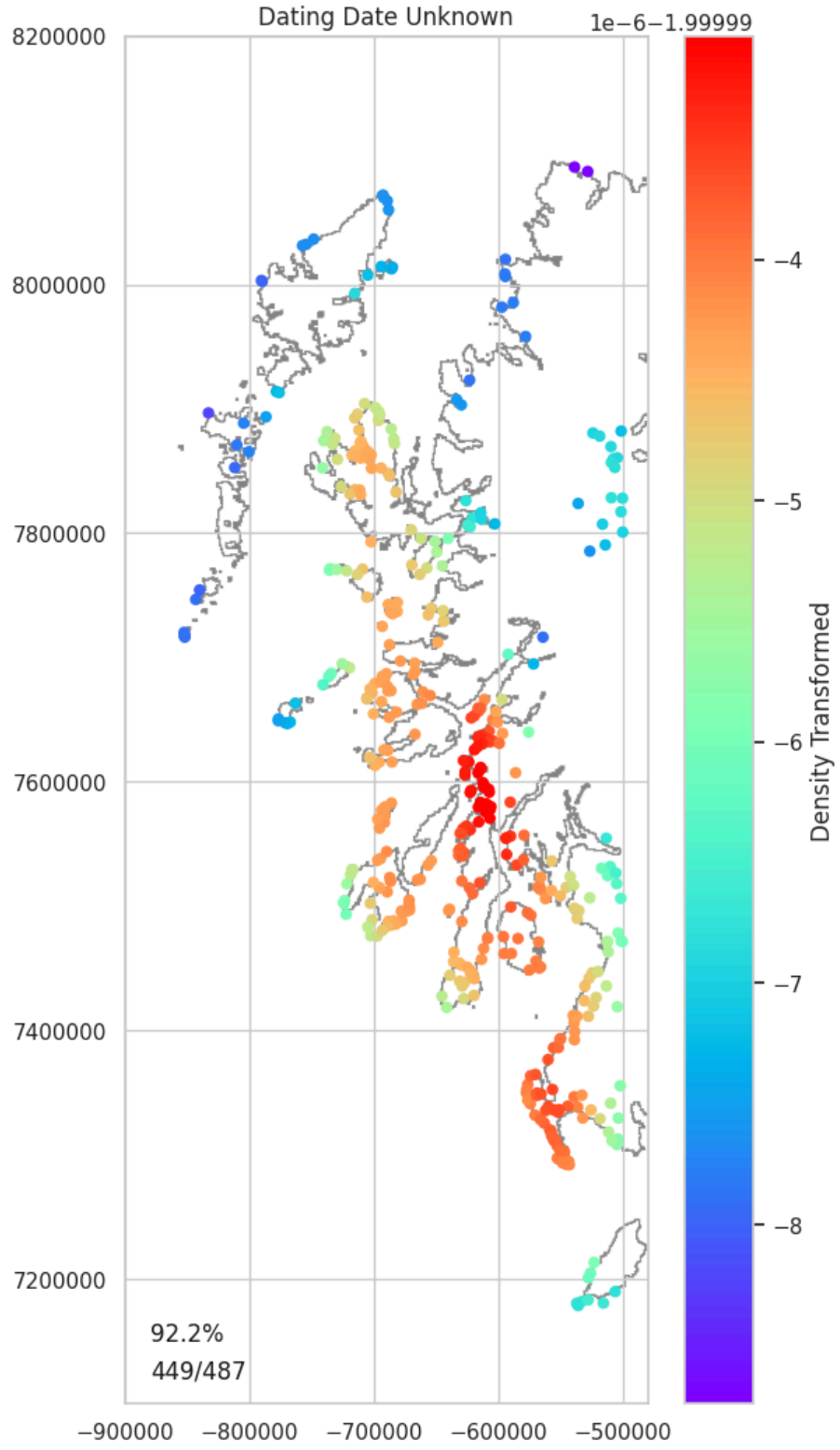
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Dating Encodeable Data

In [92]:
```
dating_encodeable_features = [
 'Dating_Date_Pre_1200BC',
 'Dating_Date_1200BC_800BC',
 'Dating_Date_800BC_400BC',
 'Dating_Date_400BC_AD50',
 'Dating_Date_AD50_AD400',
 'Dating_Date_AD400_AD800',
 'Dating_Date_Post_AD800',
 'Dating_Date_Unknown']
```

In [93]:
```
plot_encodeable(hillforts_data, north_west, dating_encodeable_features, show_percentage)
```

487



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

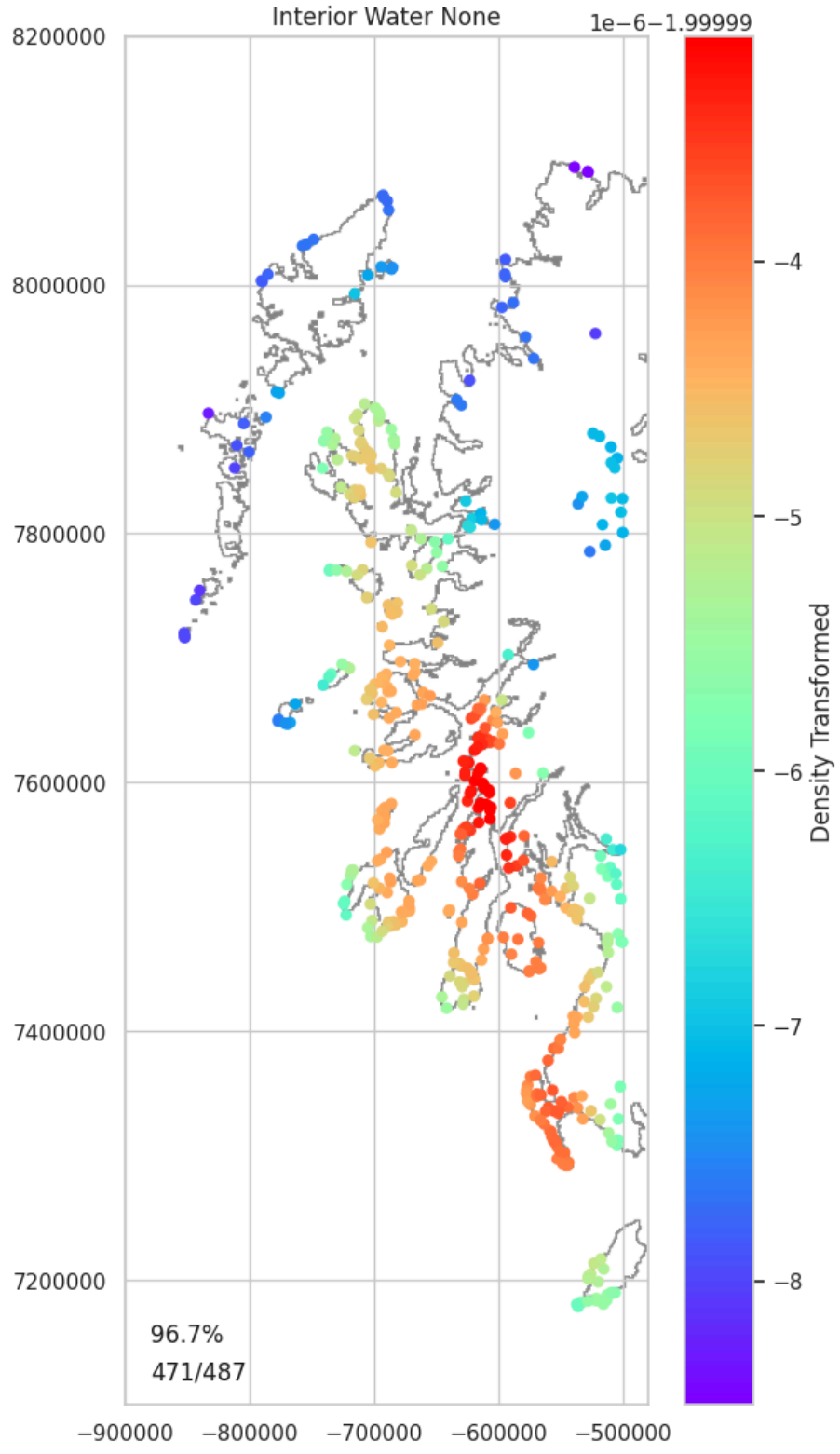
Interior Encoeable Data

In [94]:
```
interior_encodeable_features = [
 'Interior_Water_None',
 'Interior_Water_Spring',
 'Interior_Water_Stream',
 'Interior_Water_Pool',
 'Interior_Water_Flush',
 'Interior_Water_Well',
```

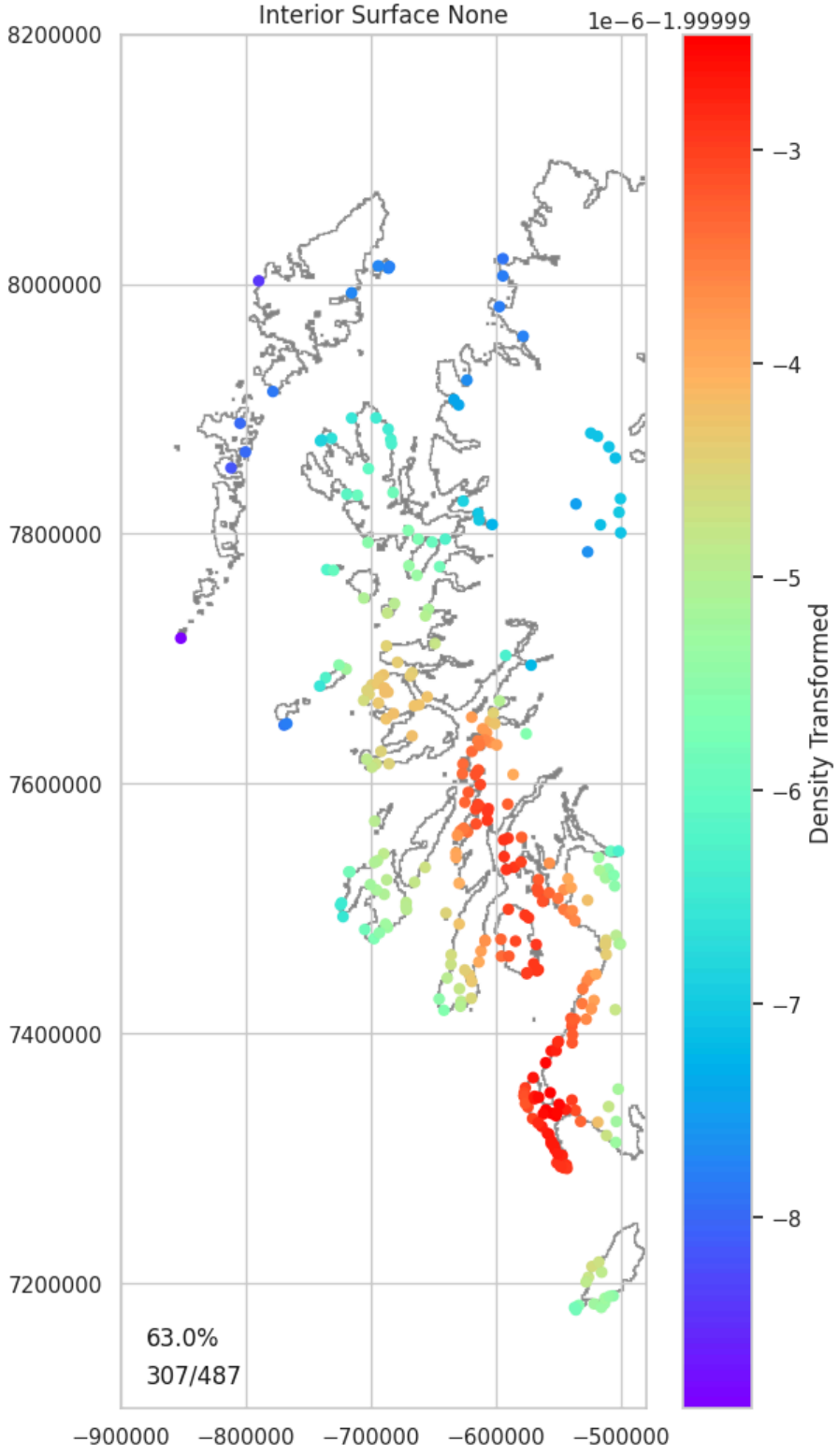
```
 'Interior_Water_Other',
 'Interior_Surface_None',
 'Interior_Surface_Round',
 'Interior_Surface_Rectangular',
 'Interior_Surface_Curvilinear',
 'Interior_Surface_Roundhouse',
 'Interior_Surface_Pit',
 'Interior_Surface_Quarry',
 'Interior_Surface_Other',
 'Interior_Excavation_None',
 'Interior_Excavation_Pit',
 'Interior_Excavation_Posthole',
 'Interior_Excavation_Roundhouse',
 'Interior_Excavation_Rectangular',
 'Interior_Excavation_Road',
 'Interior_Excavation_Quarry',
 'Interior_Excavation_Other',
 'Interior_Excavation_Nothing',
 'Interior_Geophysics_None',
 'Interior_Geophysics_Pit',
 'Interior_Geophysics_Roundhouse',
 'Interior_Geophysics_Rectangular',
 'Interior_Geophysics_Road',
 'Interior_Geophysics_Quarry',
 'Interior_Geophysics_Other',
 'Interior_Geophysics_Nothing']
```

In [95]:
```
plot_encodeable(hillforts_data, north_west, interior_encodeable_features, show_percentage)
```

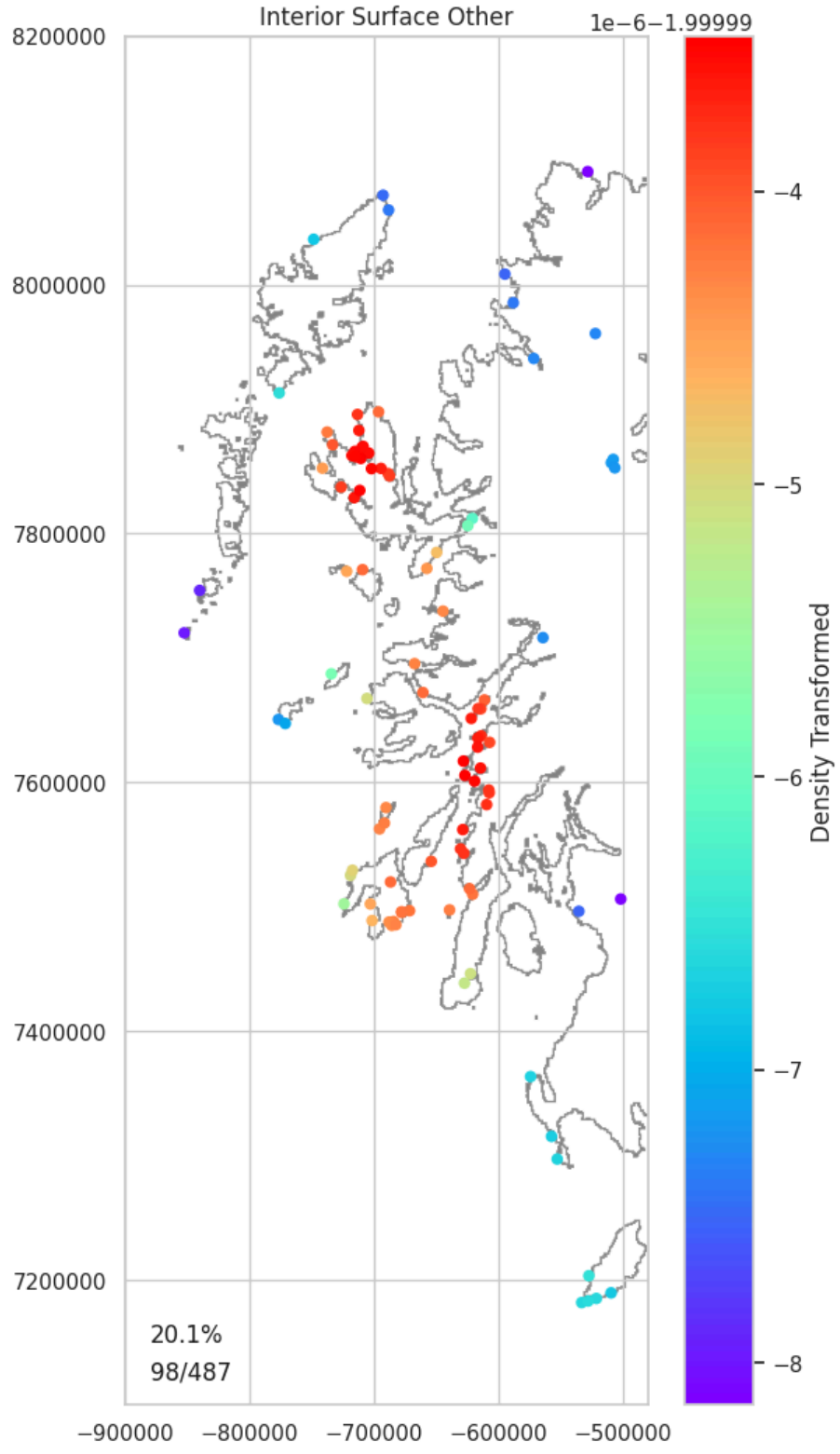
```
487
```



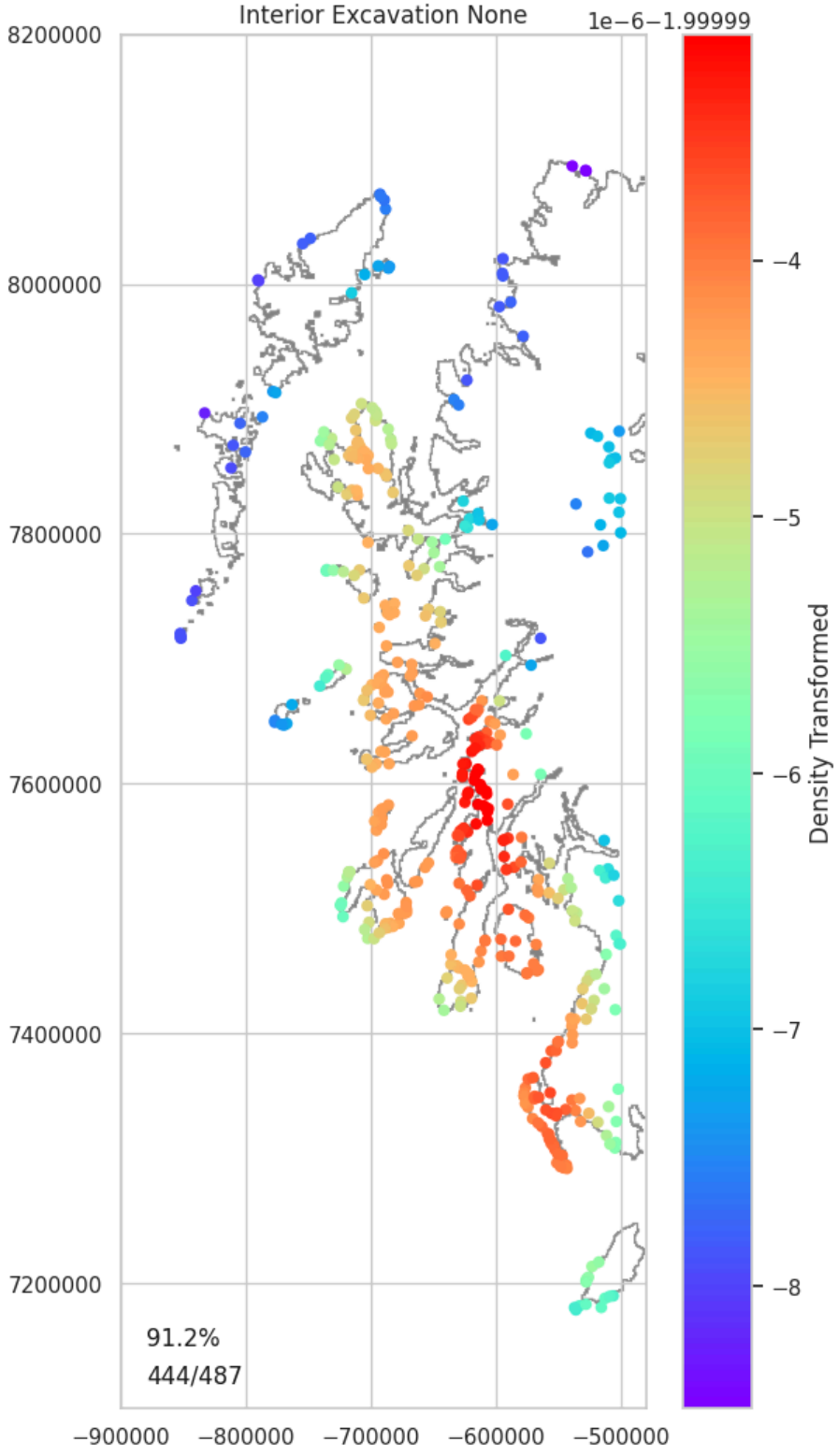
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



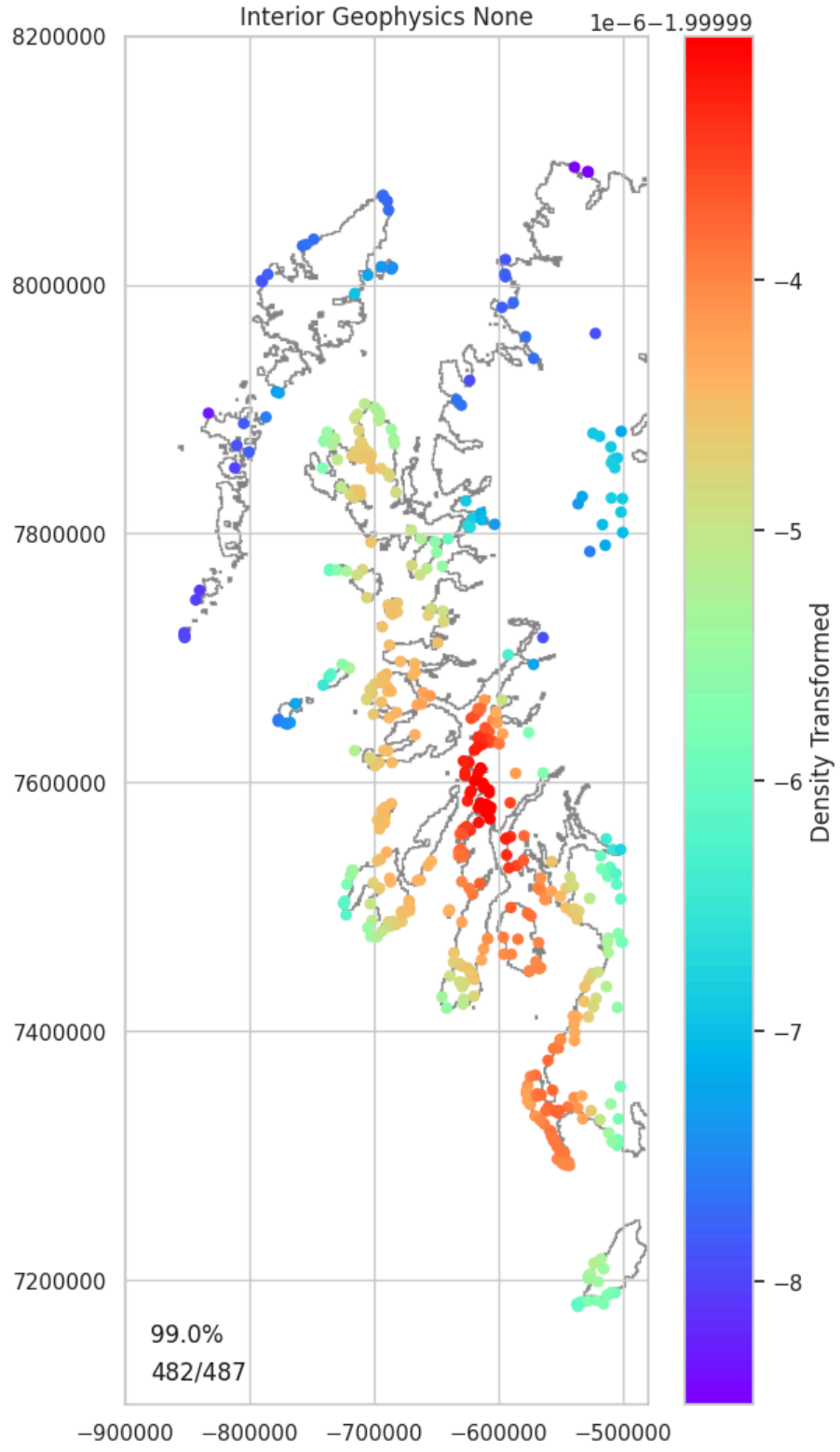
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



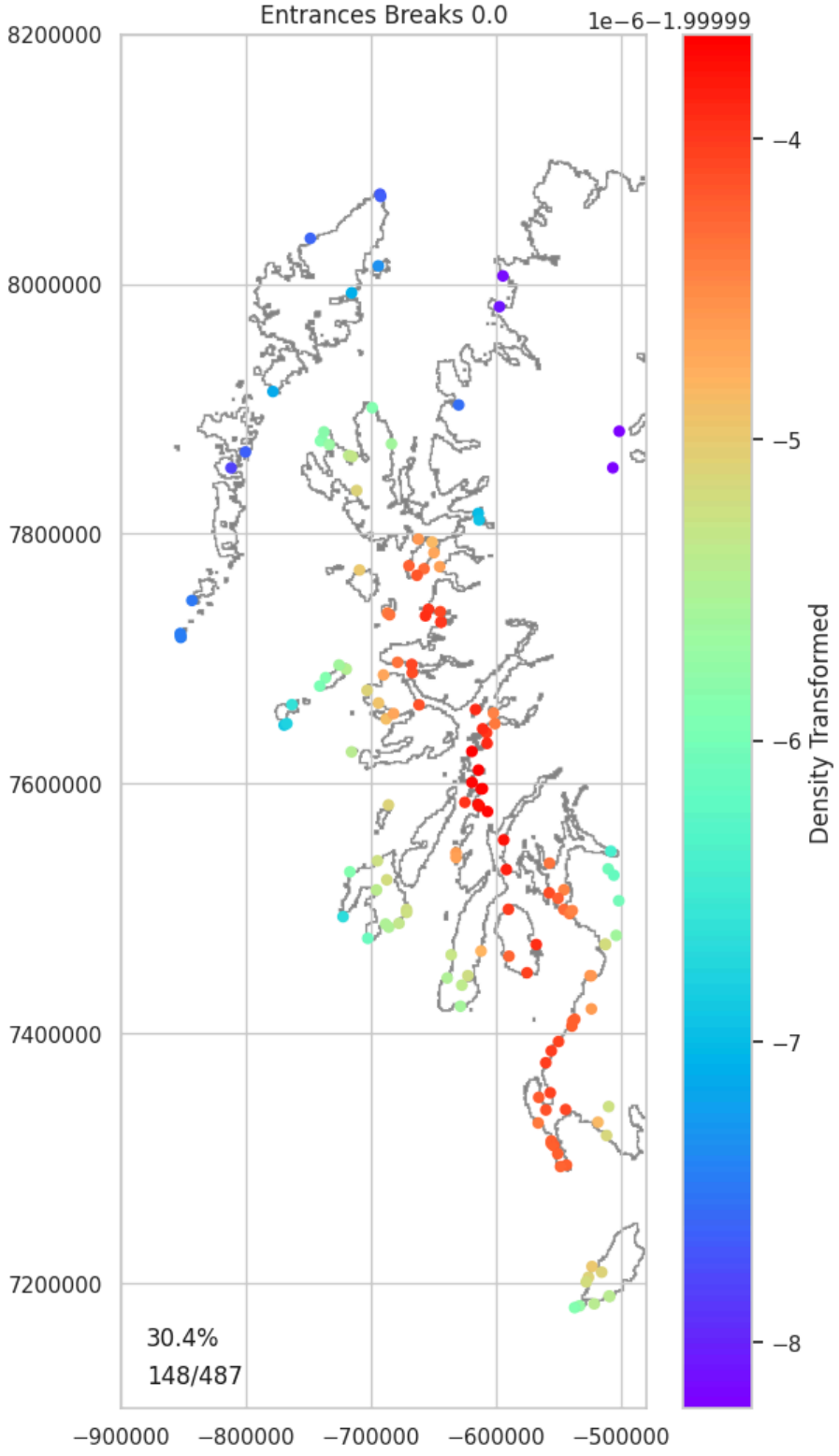
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Entrance Numeric Data

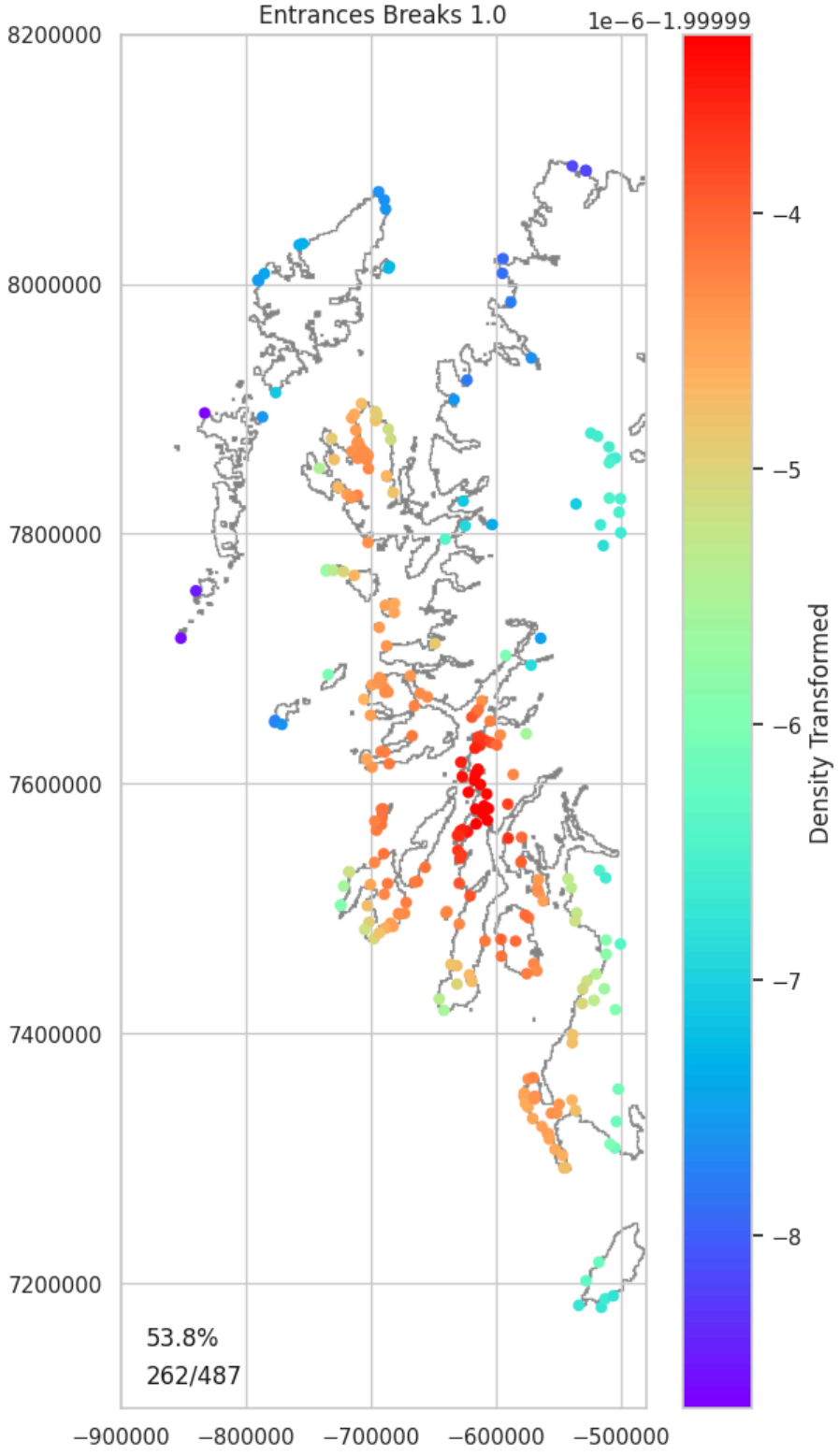
In [96]:
```
entrance_numeric_features = [
 'Entrances_Breaks',
 'Entrances_Original']
```

In [97]:
```
plot_numeric(hillforts_data, north_west, entrance_numeric_features, show_percentage)
```

```
487
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 9.0]
```

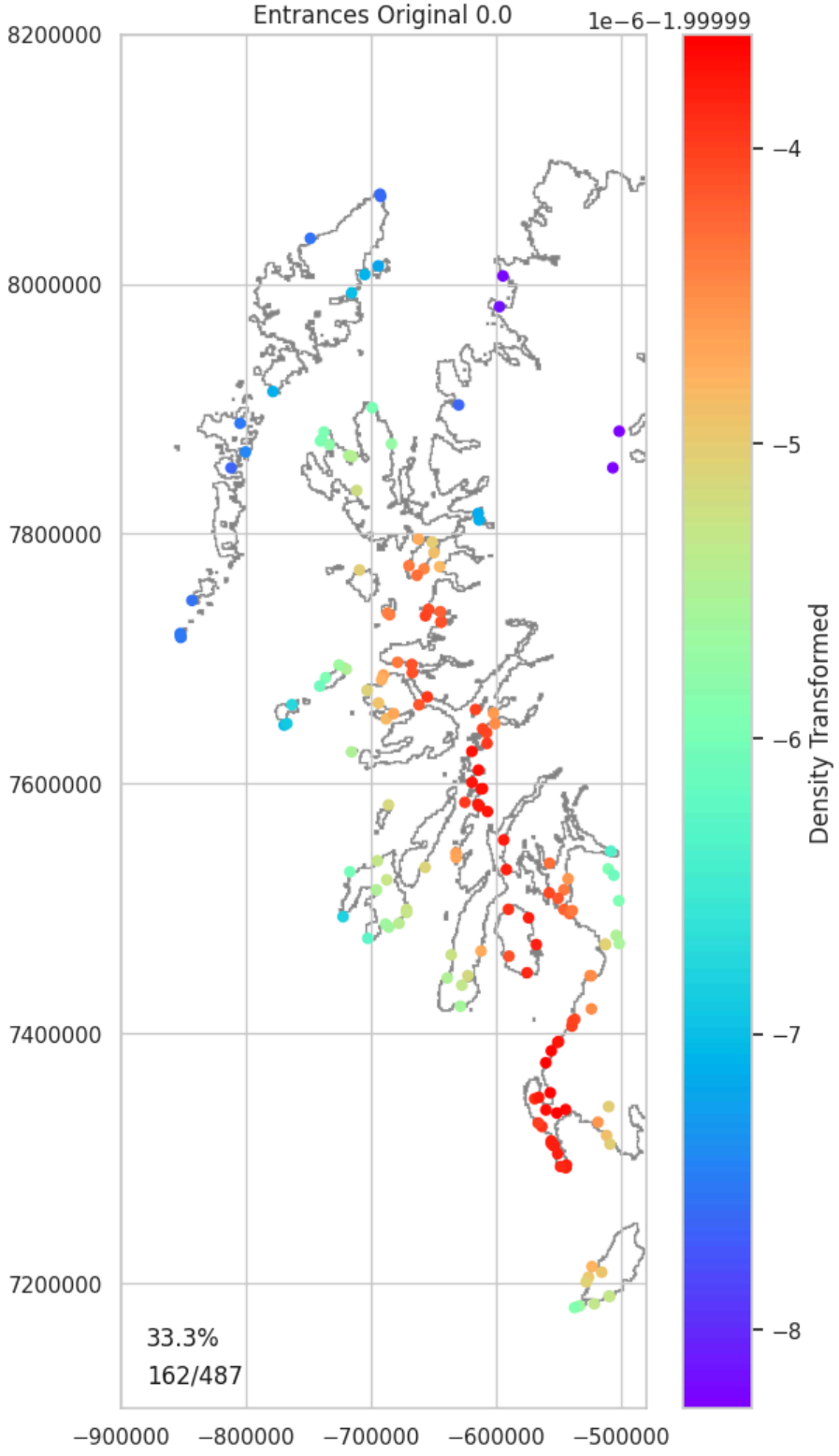


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

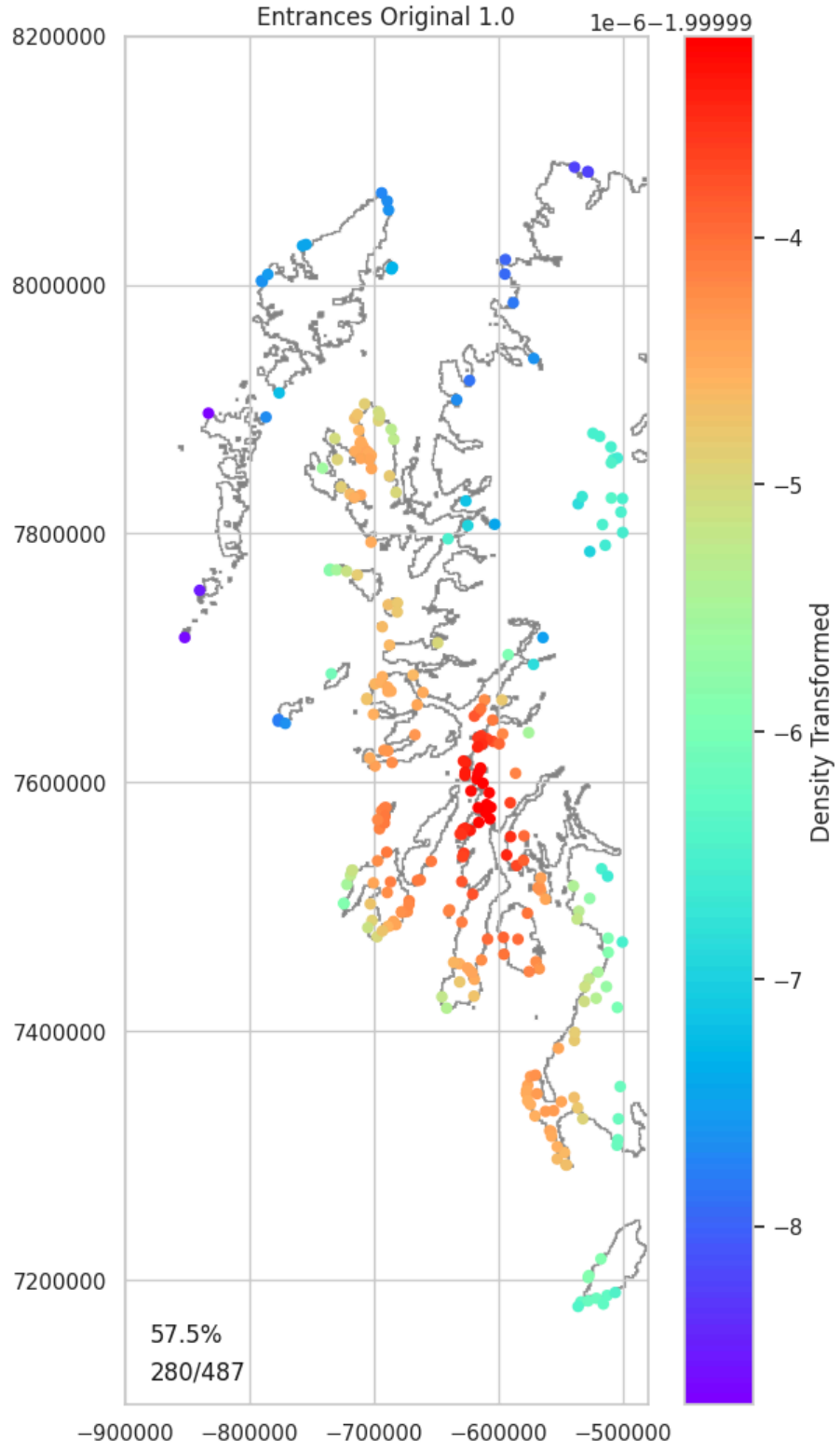


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

```
[0.0, 1.0, 2.0, 3.0, 4.0]
```



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Entrance Encodable Data

In [98]:
```
entrance_encodeable_features = [
 'Entrances_Guard_Chambers',
 'Entrances_Chevaux']
```

In [99]:
```
plot_encodeable(hillforts_data, north_west, entrance_encodeable_features, show_percentage)
```

487

Enclosing Numeric Data

In [100…

```
def plot_enclosing_area_1(hillforts_data, north_west, numeric_features, show_percentage):
    numeric_data = hillforts_data[numeric_features].copy()
    location_data = pd.merge(north_west, numeric_data, \
                                left_index=True, right_index=True)
    print(len(location_data))
    ranges = [[0, 0.13], [0.13, 1], [1,50]]
    for rng in ranges:
        #Filter data
        cluster = location_data[location_data['Enclosing_Area_1'].between(rng[0], rng[1])]
        new_col_name = 'Enclosing_Area_1' + "_" + str(rng[0]) + "_to " + str(rng[1]) + "_Ha"
        cluster = cluster.rename(columns={'Enclosing_Area_1': new_col_name})
        show_eildon = False
        if show_percentage < 10:
            show_percentage = 10

        if (len(cluster) / len(location_data)) * 100 > show_percentage:
            # refresh density
            cluster = renew_density(cluster)
            location_X_cluster = \
            plot_data_range_plus(cluster['Location_X'], '', None)
            location_Y_cluster = \
            plot_data_range_plus(cluster['Location_Y'], '', None)
            #print(len(cluster))
            plot_north_west_density_for_column(show_eildon, location_data, \
                                               cluster, new_col_name, location_X_cluster, \
                                               location_Y_cluster, False)
```

Only Enclosing Area 1 will be used. See Hillforts Primer Part 5.

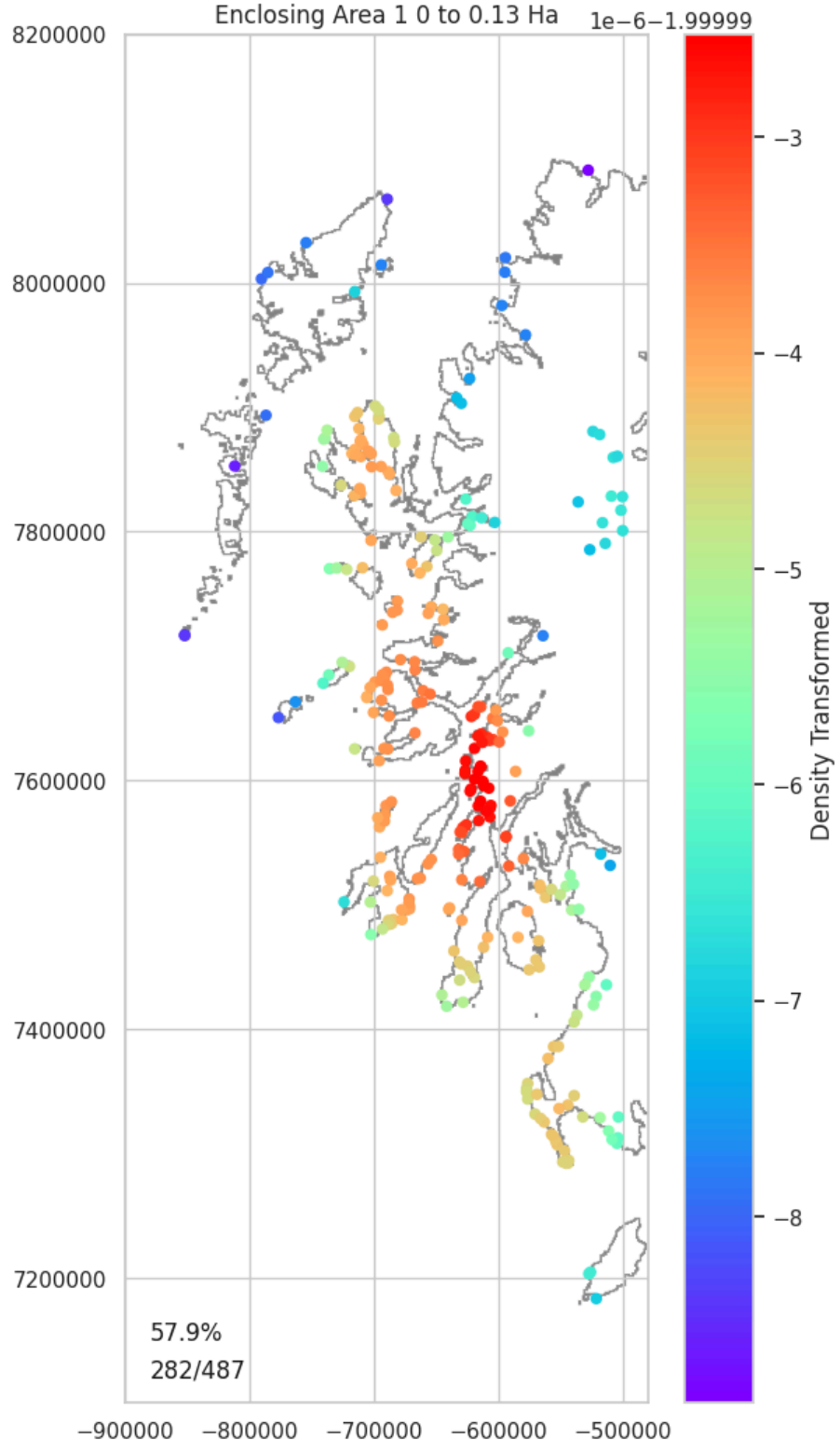
In [101…

```
enclosing_numeric_features_part1 = [
 'Enclosing_Area_1'#,
 #'Enclosing_Area_2',
 #'Enclosing_Area_3',
 #'Enclosing_Area_4',
 #'Enclosing_Enclosed_Area',
 #'Enclosing_Area',
 ]
```

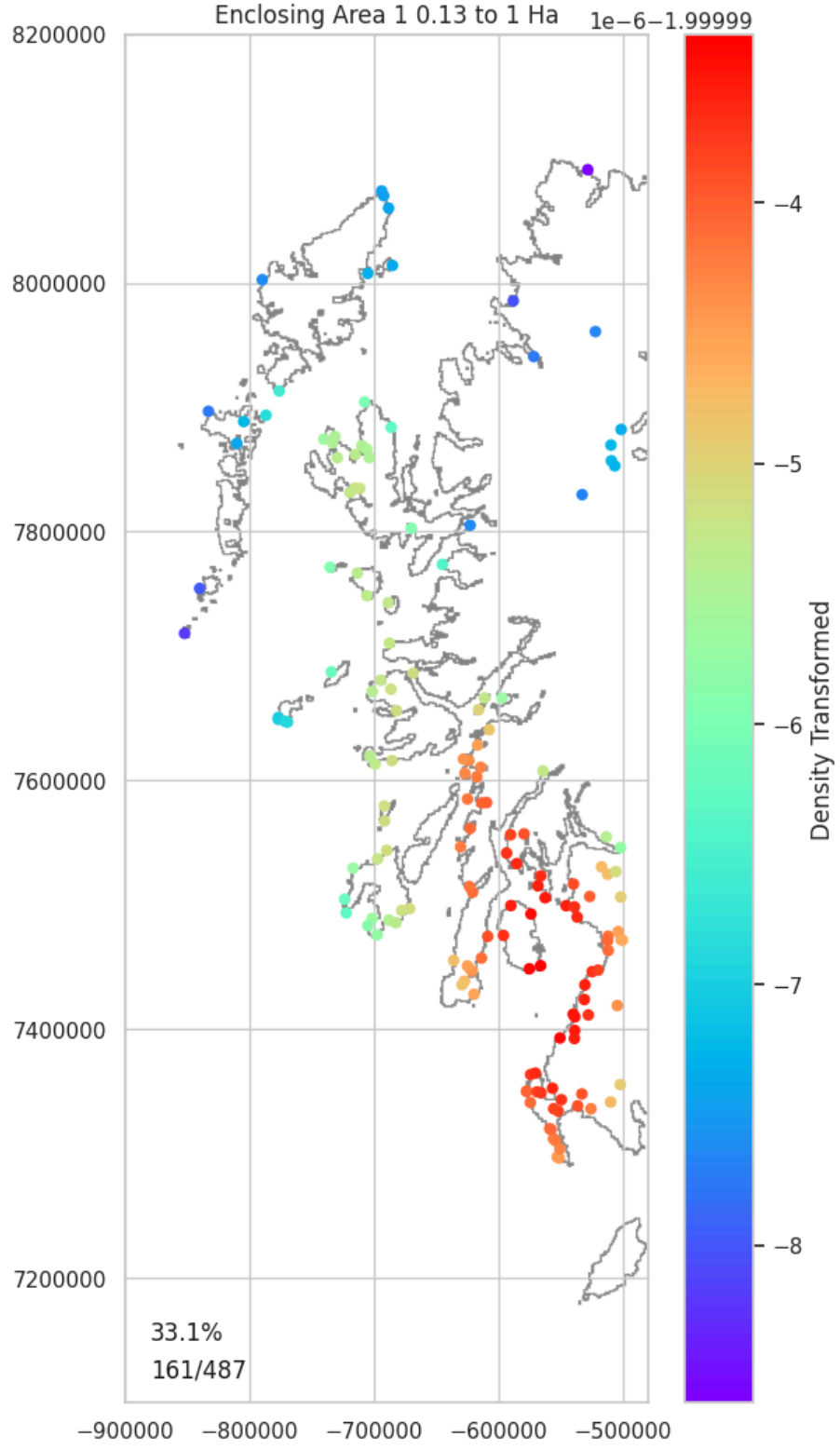
In [102…

```
plot_enclosing_area_1(hillforts_data, north_west, enclosing_numeric_features_part1, show_percentage)
```

```
487
```



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

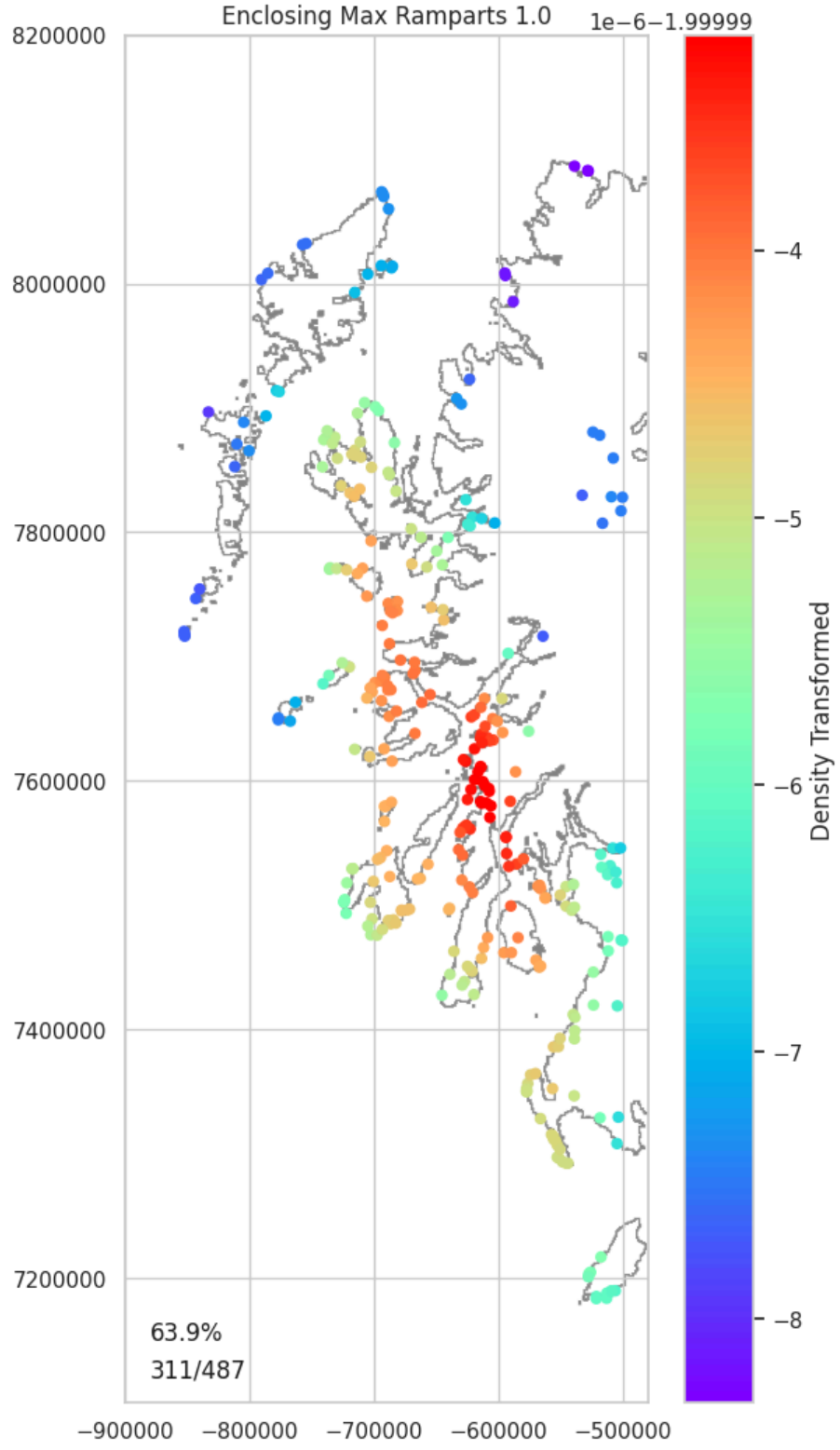
In [103…
```
print(enclosing_numeric_features_part1)
```

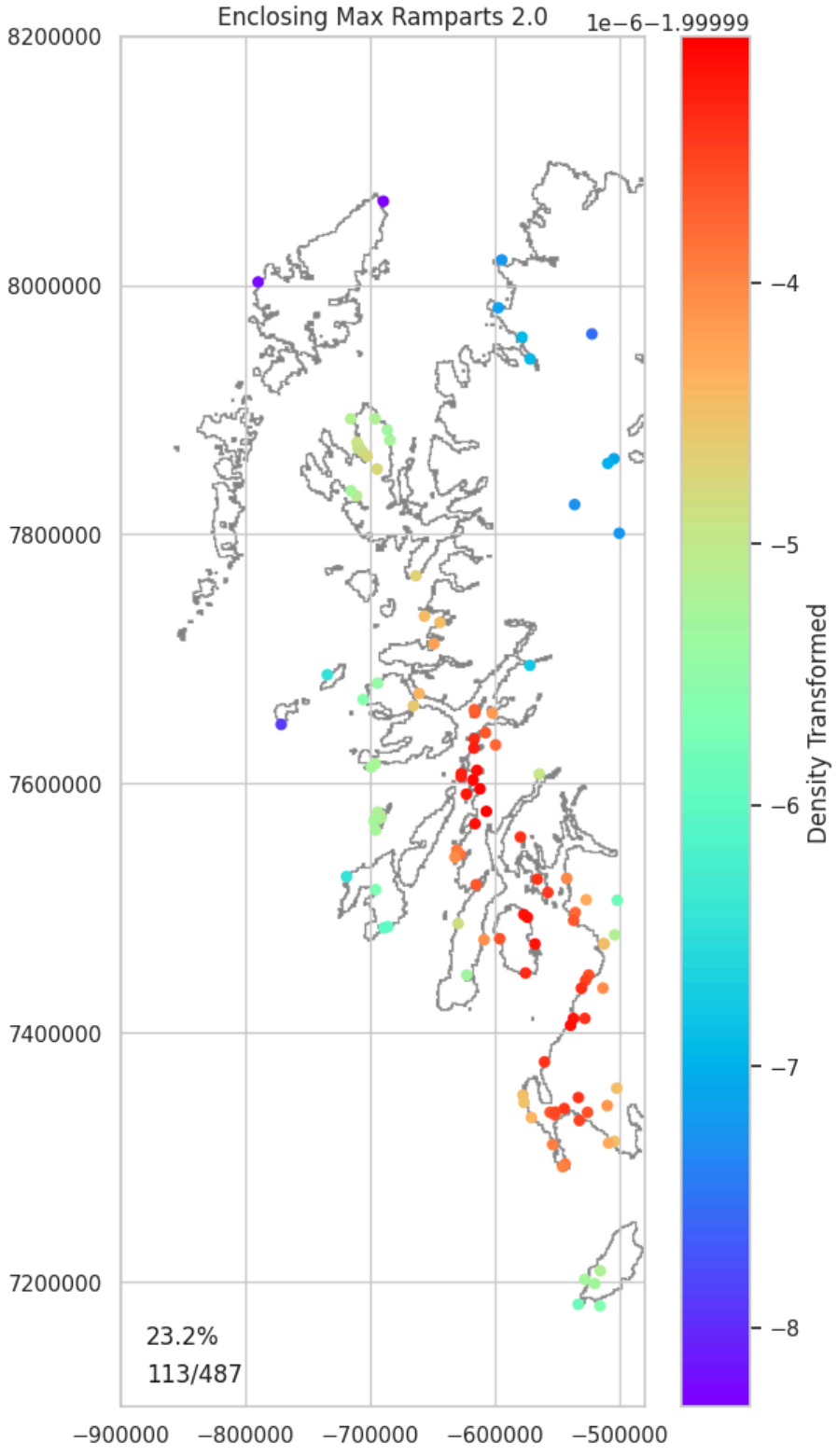
```
['Enclosing_Area_1']
```

In [104…
```
enclosing_numeric_features_part2 = [
 'Enclosing_Max_Ramparts',
 'Enclosing_NE_Quadrant',
 'Enclosing_SE_Quadrant',
 'Enclosing_SW_Quadrant',
 'Enclosing_NW_Quadrant',
 'Enclosing_Ditches_Number']
```

In [105…
```
plot_numeric(hillforts_data, north_west, enclosing_numeric_features_part2, show_percentage)
```

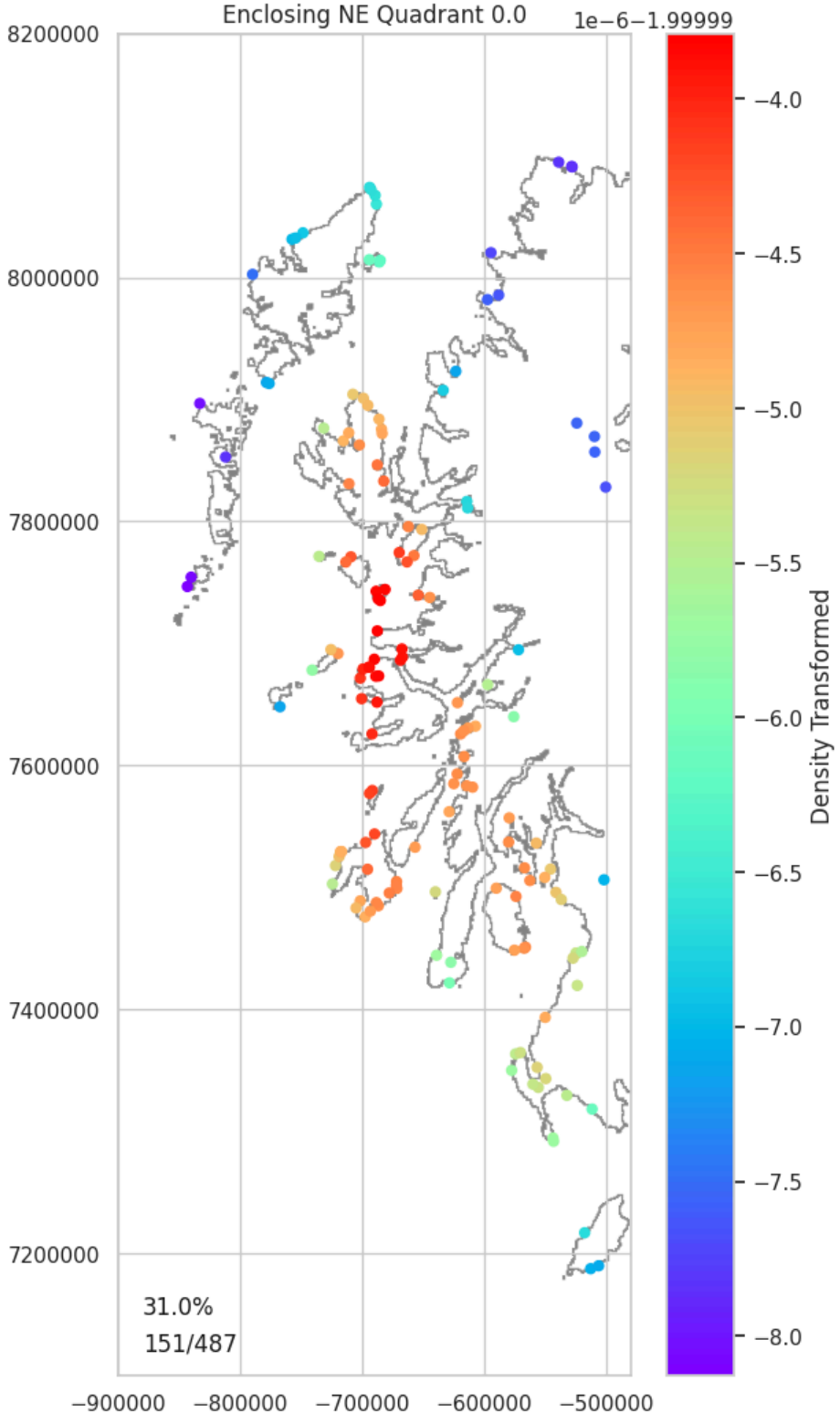
487
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]



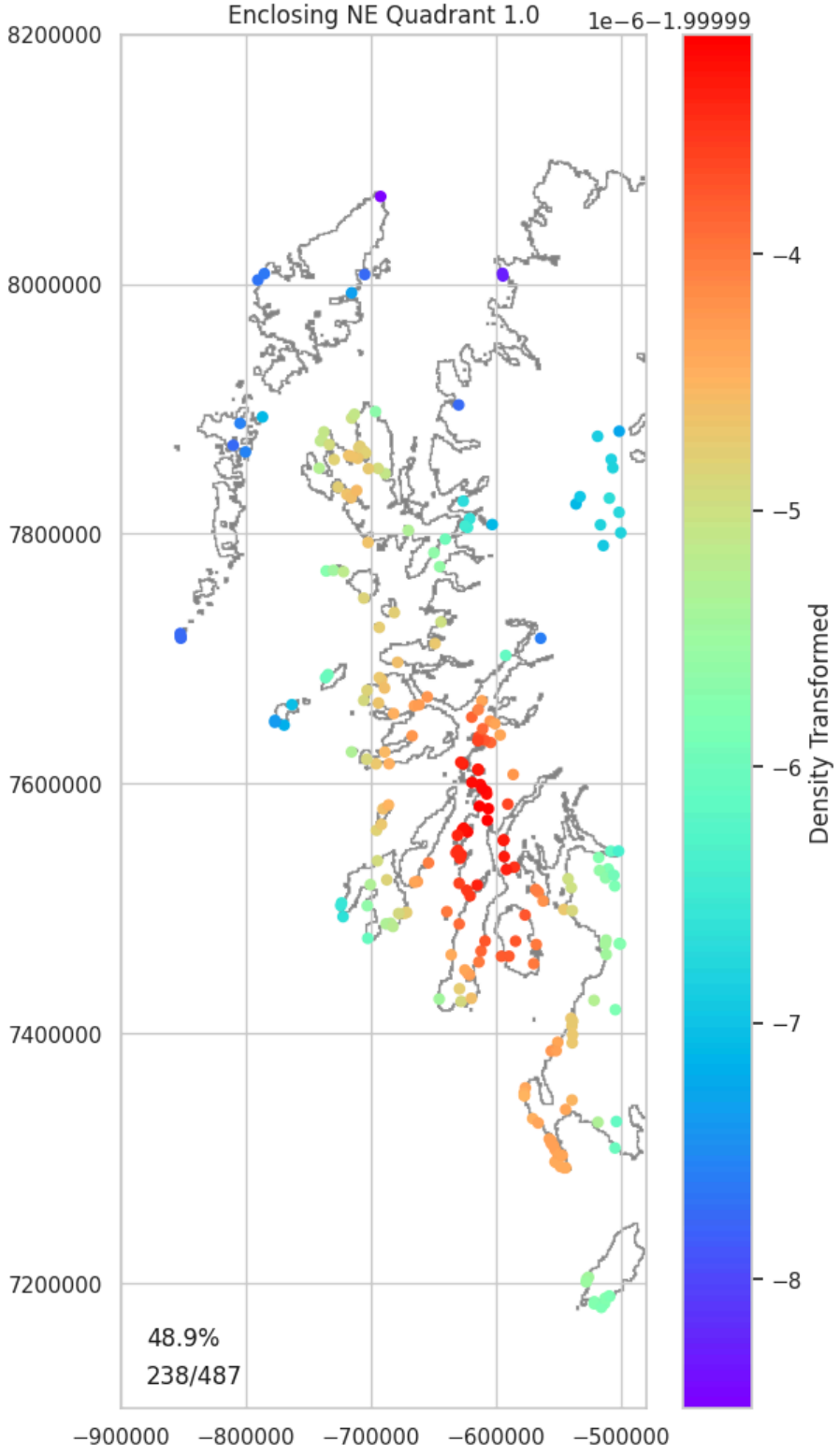


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]

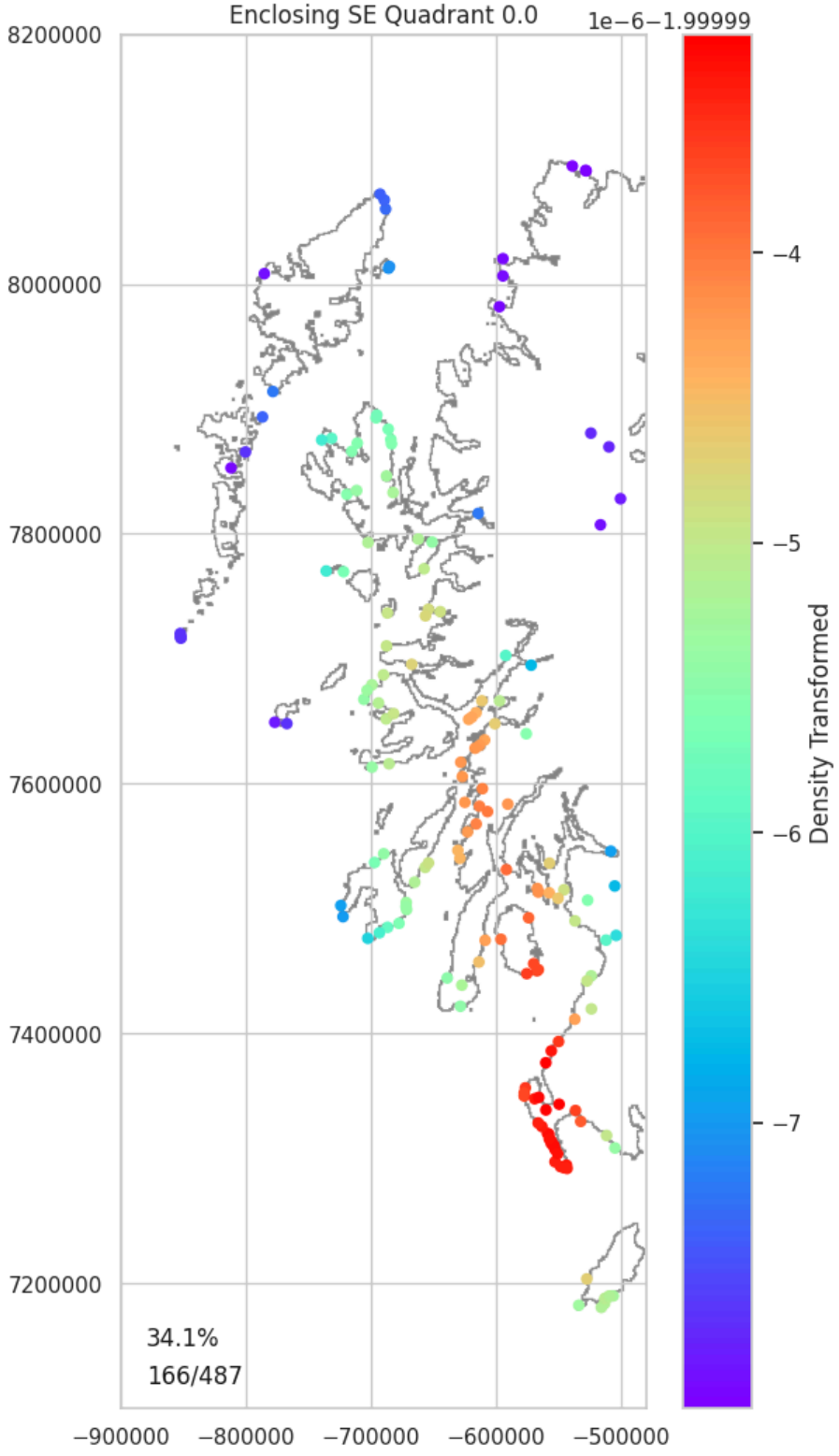


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

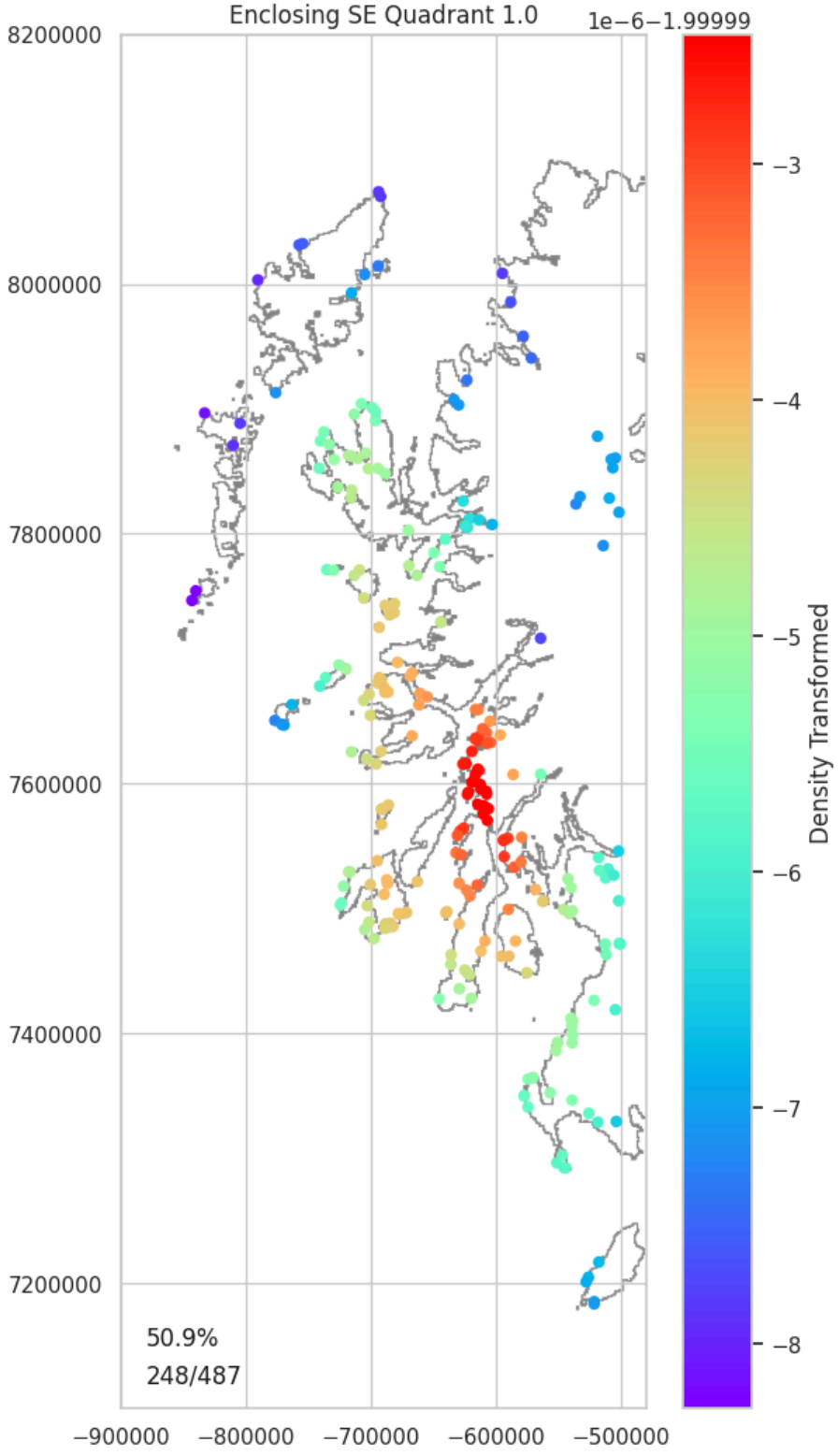


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

[0.0, 1.0, 2.0, 3.0, 4.0]

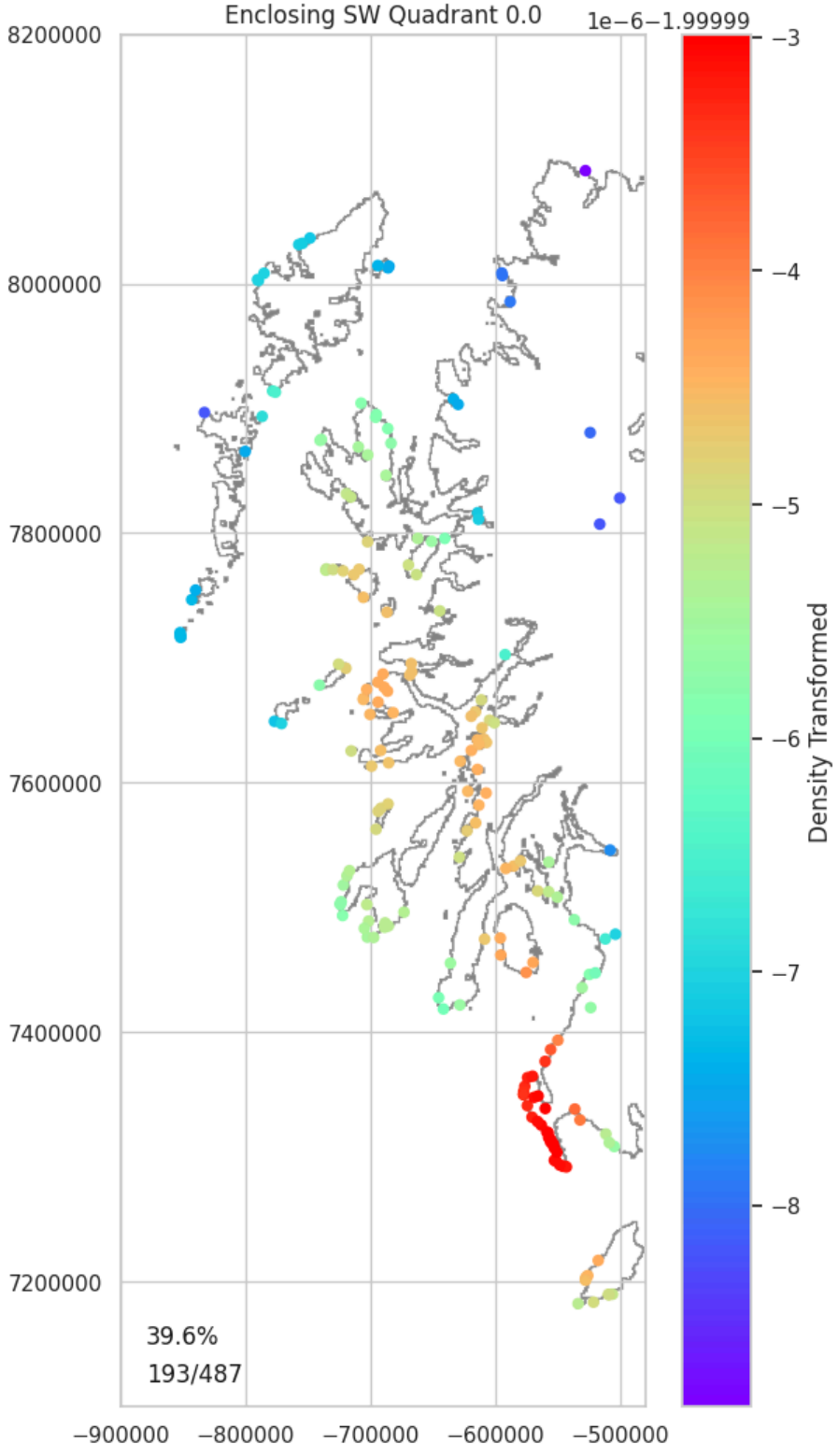


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

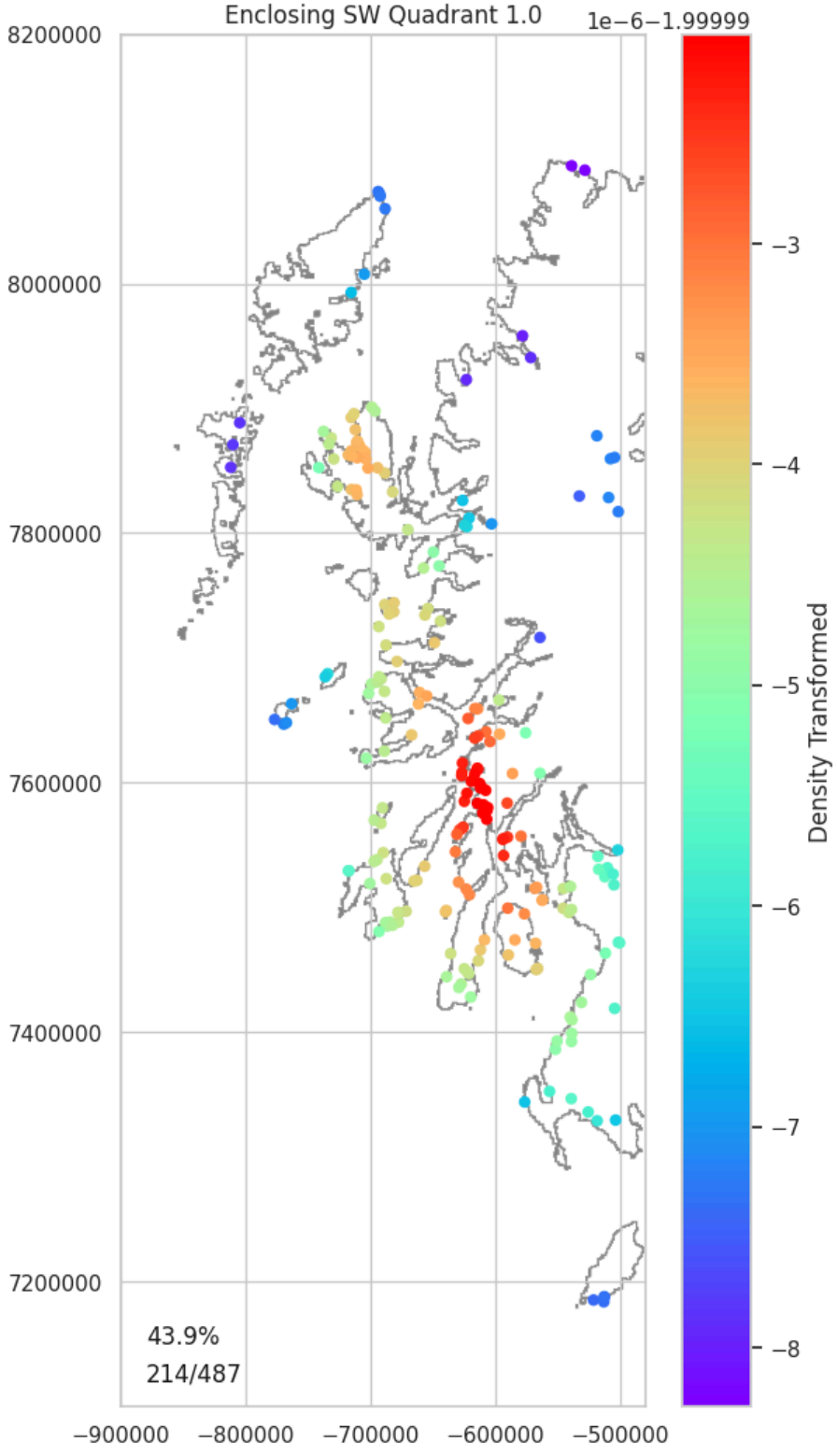


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]

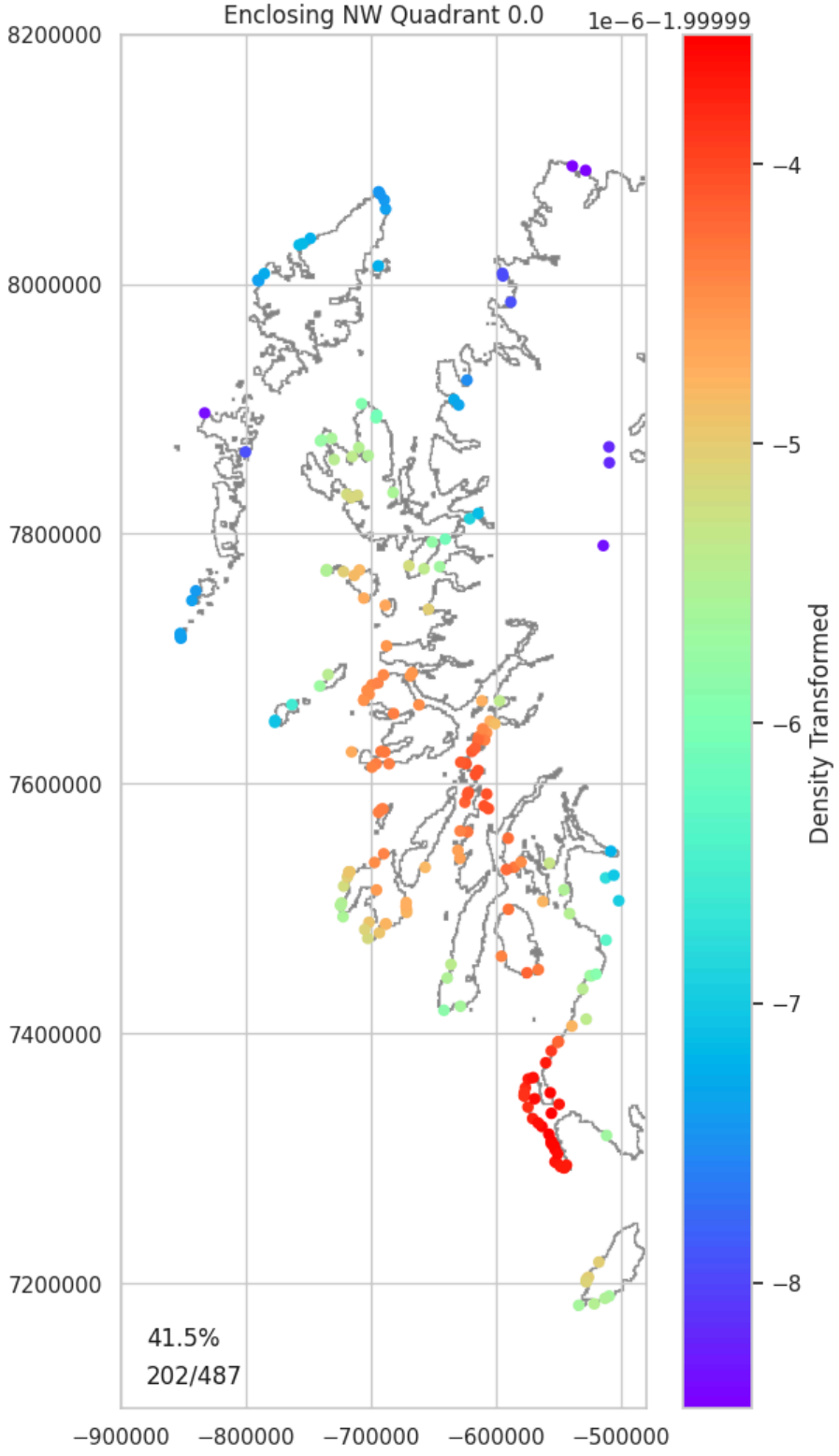


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

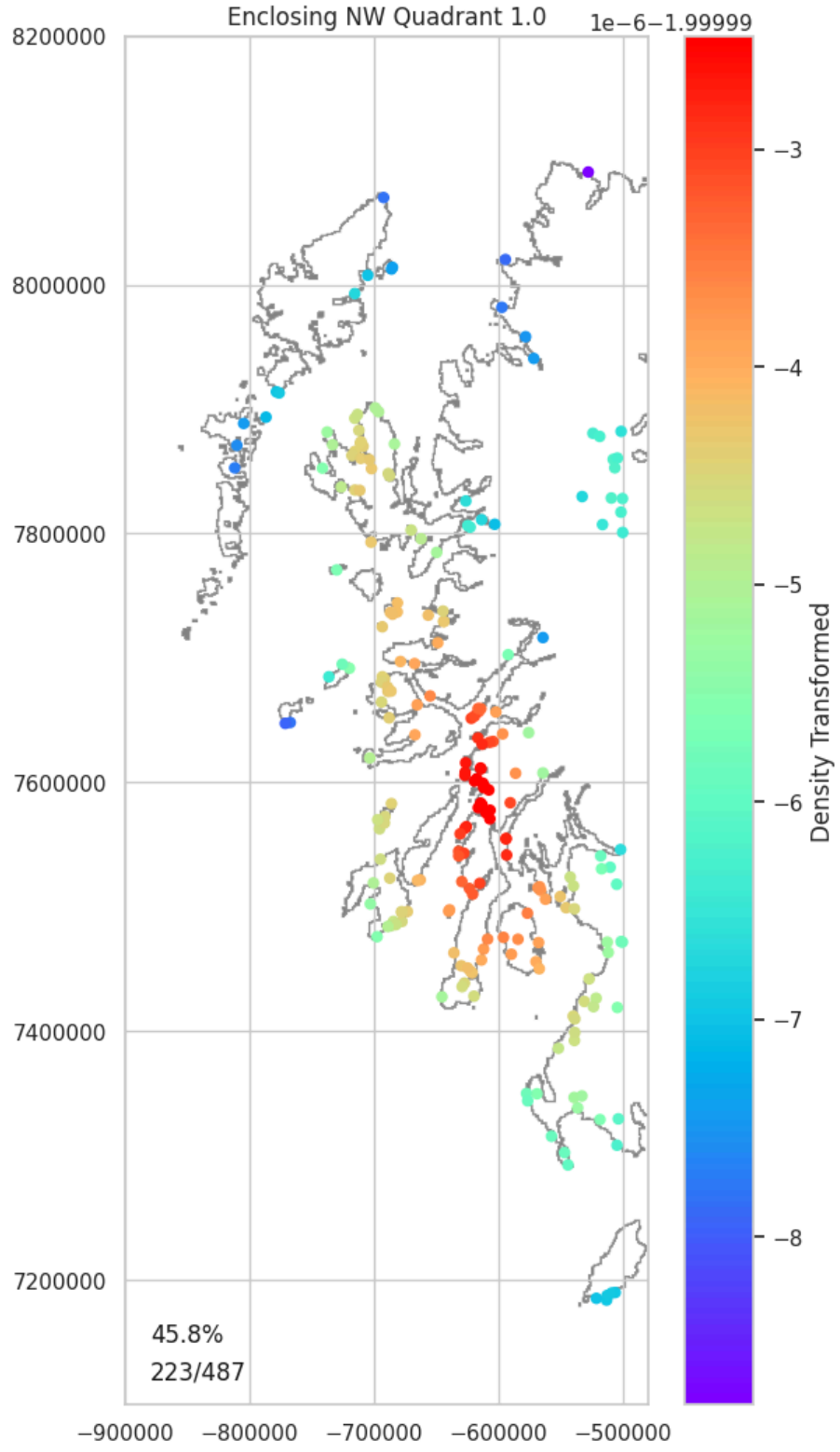


Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

```
[0.0, 1.0, 2.0, 3.0, 4.0]
```



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
```

Enclosing Encodeable Data

In [106…

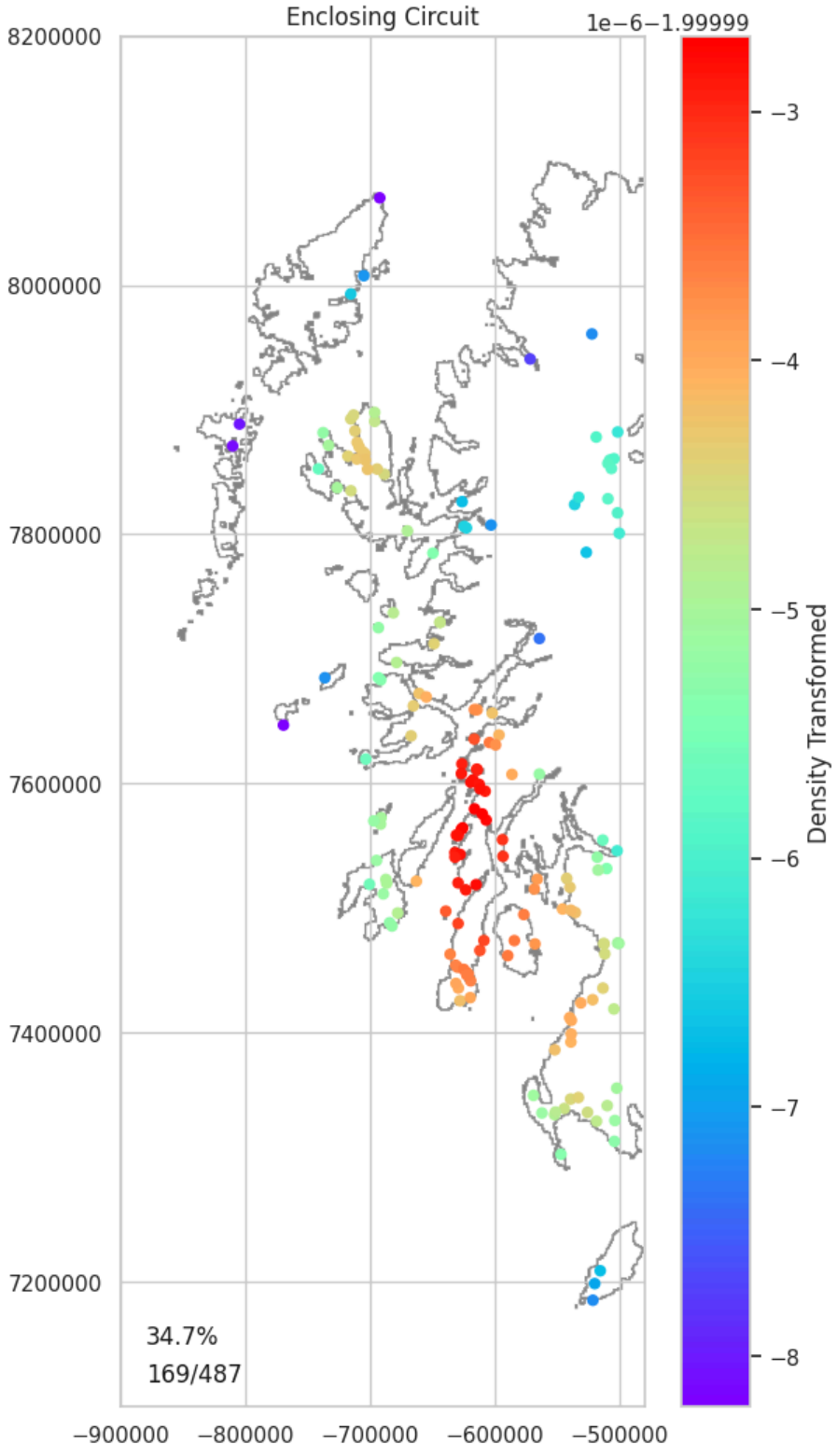
```
enclosing_encodeable_features = [
 'Enclosing_Multiperiod',
 'Enclosing_Circuit',
 'Enclosing_Current_Part_Uni',
 'Enclosing_Current_Uni',
 'Enclosing_Current_Part_Bi',
 'Enclosing_Current_Bi',
 'Enclosing_Current_Part_Multi',
 'Enclosing_Current_Multi',
```

```
 'Enclosing_Current_Unknown',
 'Enclosing_Period_Part_Uni',
 'Enclosing_Period_Uni',
 'Enclosing_Period_Part_Bi',
 'Enclosing_Period_Bi',
 'Enclosing_Period_Part_Multi',
 'Enclosing_Period_Multi',
 'Enclosing_Surface_None',
 'Enclosing_Surface_Bank',
 'Enclosing_Surface_Wall',
 'Enclosing_Surface_Rubble',
 'Enclosing_Surface_Walk',
 'Enclosing_Surface_Timber',
 'Enclosing_Surface_Vitrification',
 'Enclosing_Surface_Burning',
 'Enclosing_Surface_Palisade',
 'Enclosing_Surface_Counter_Scarp',
 'Enclosing_Surface_Berm',
 'Enclosing_Surface_Unfinished',
 'Enclosing_Surface_Other',
 'Enclosing_Excavation_Nothing',
 'Enclosing_Excavation_Bank',
 'Enclosing_Excavation_Wall',
 'Enclosing_Excavation_Murus',
 'Enclosing_Excavation_Timber_Framed',
 'Enclosing_Excavation_Timber_Laced',
 'Enclosing_Excavation_Vitrification',
 'Enclosing_Excavation_Burning',
 'Enclosing_Excavation_Palisade',
 'Enclosing_Excavation_Counter_Scarp',
 'Enclosing_Excavation_Berm',
 'Enclosing_Excavation_Unfinished',
 'Enclosing_Excavation_No_Known',
 'Enclosing_Excavation_Other',
 'Enclosing_Gang_Working',
 'Enclosing_Ditches']
```

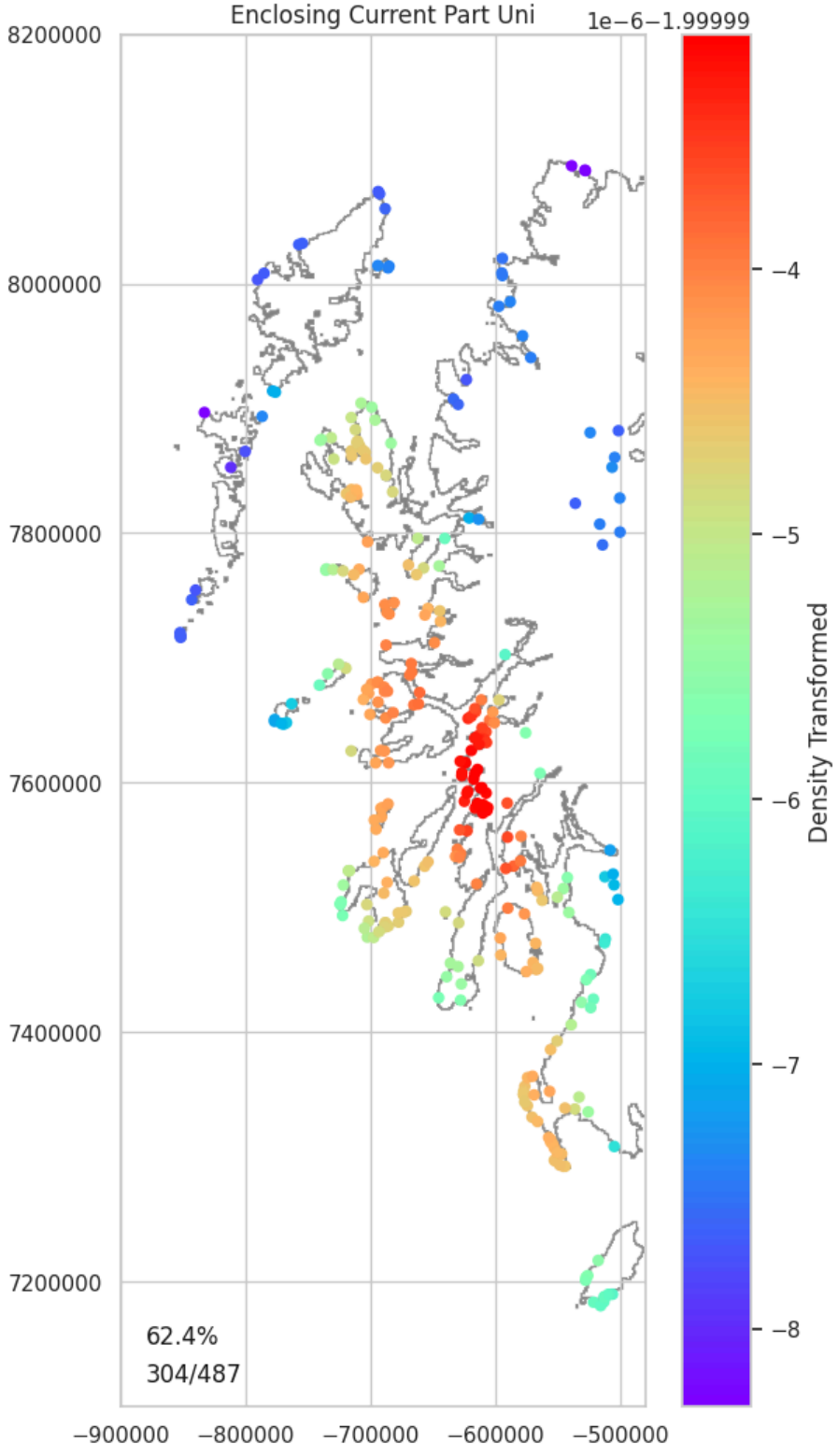
In [107…

```
plot_encodeable(hillforts_data, north_west, enclosing_encodeable_features, show_percentage)
```

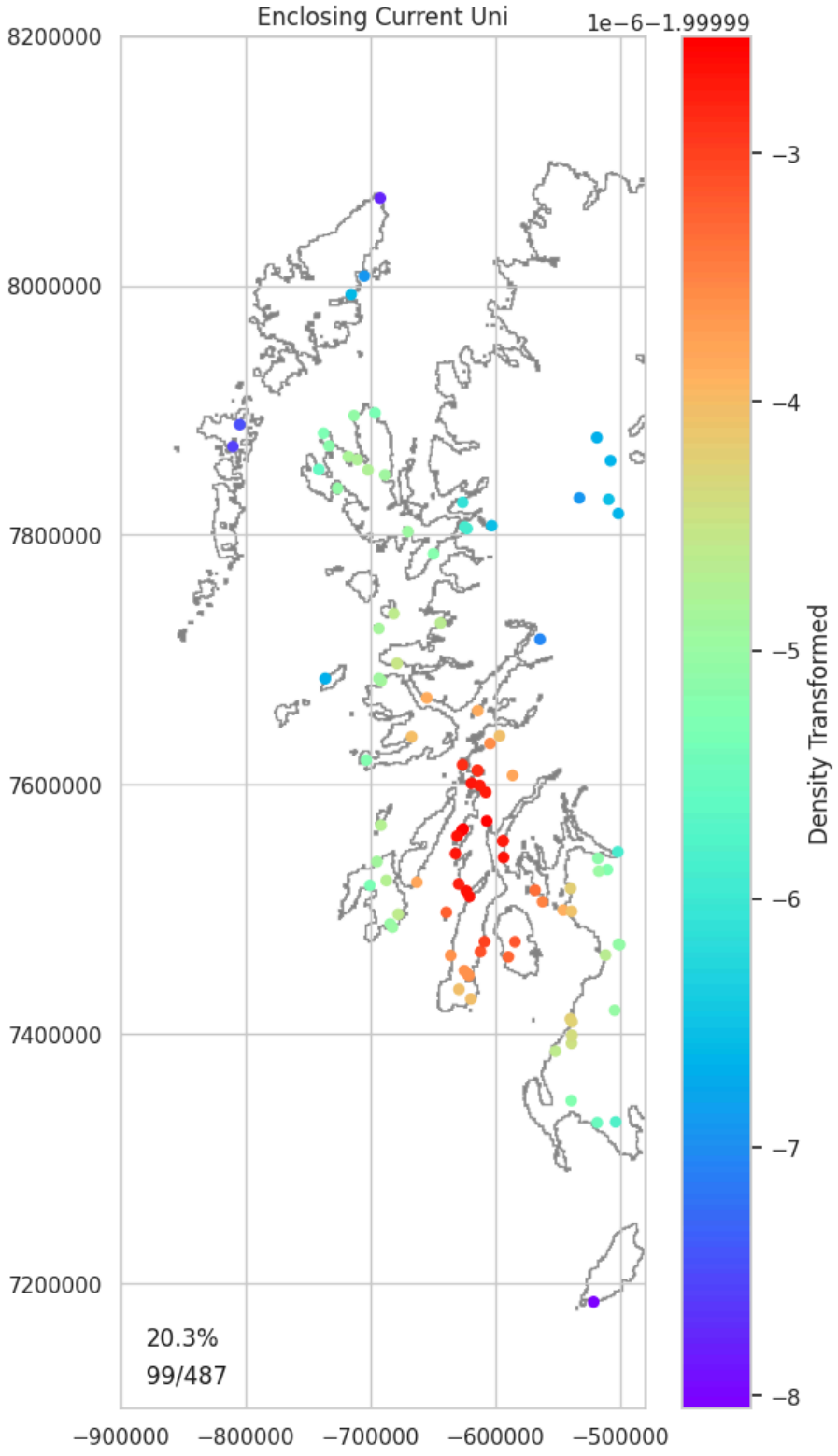
```
487
```



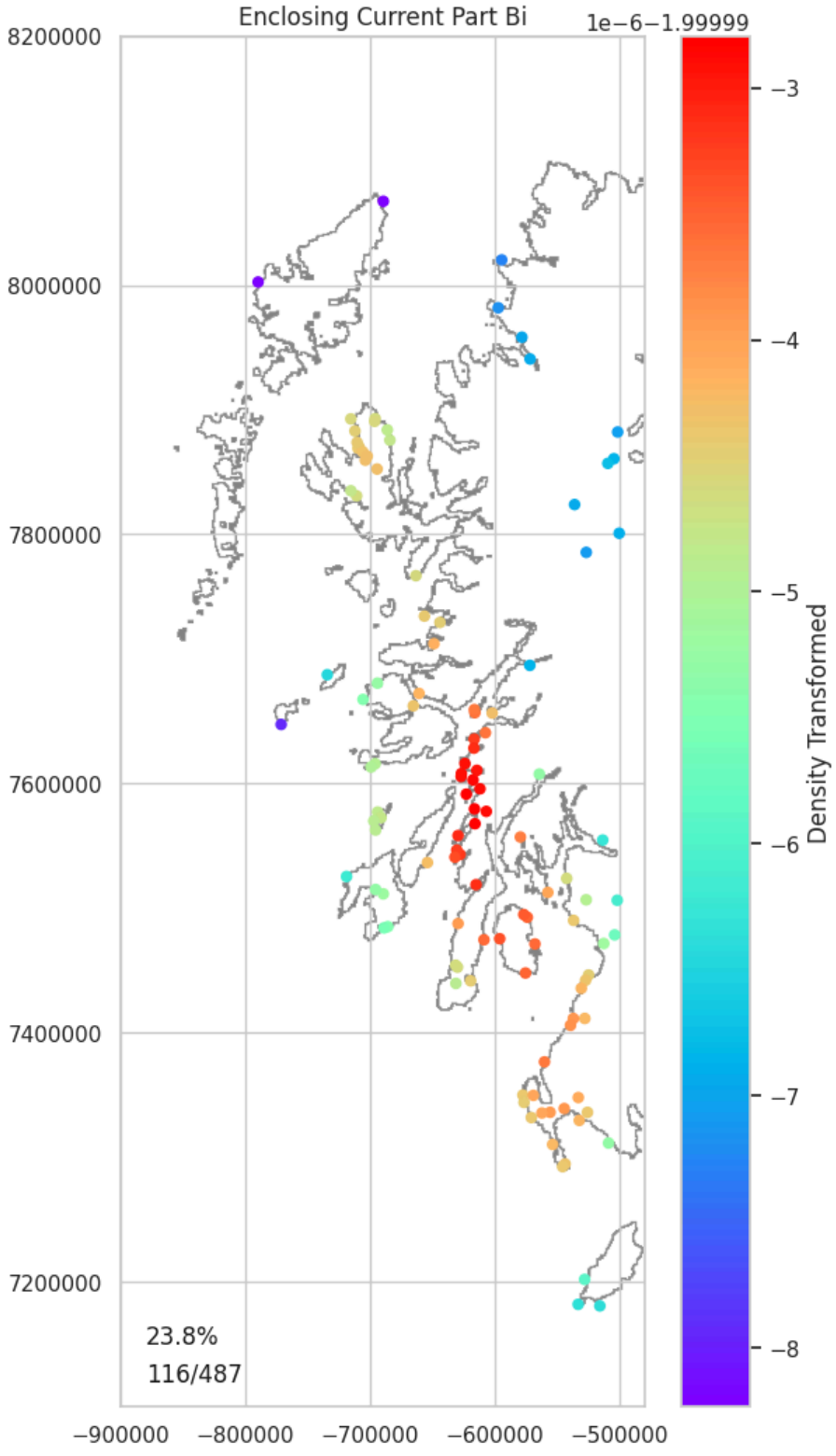
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



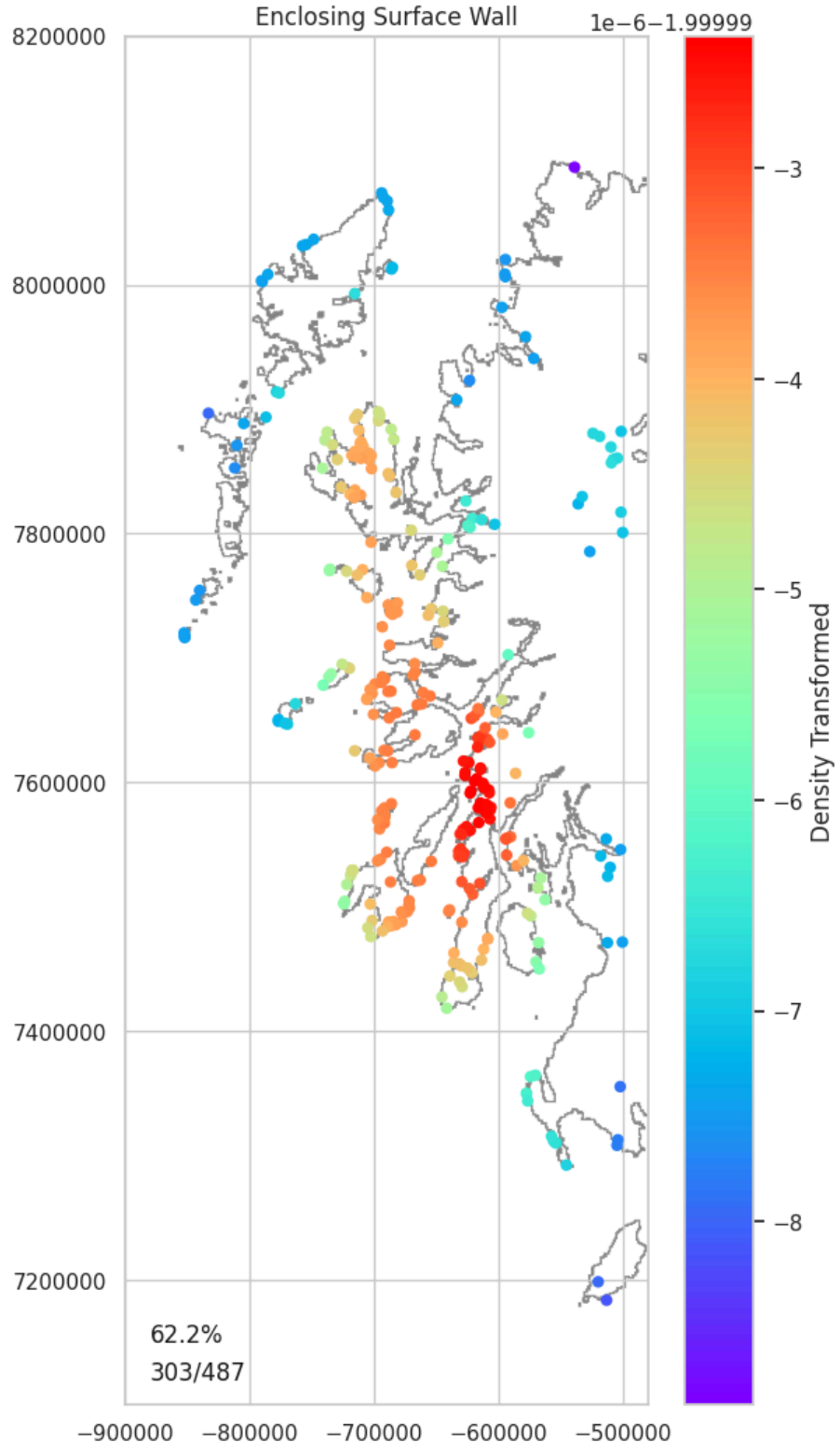
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



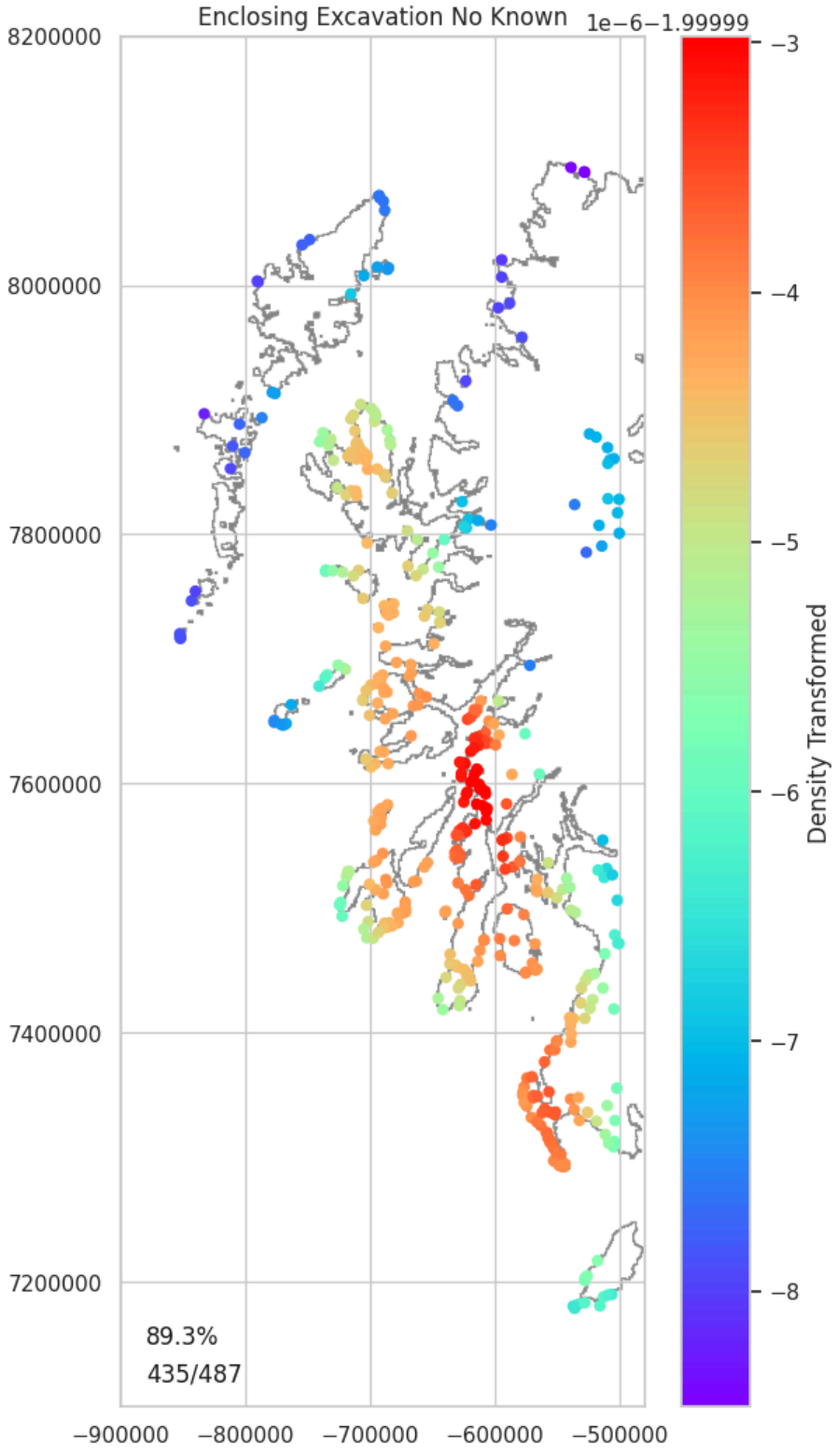
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



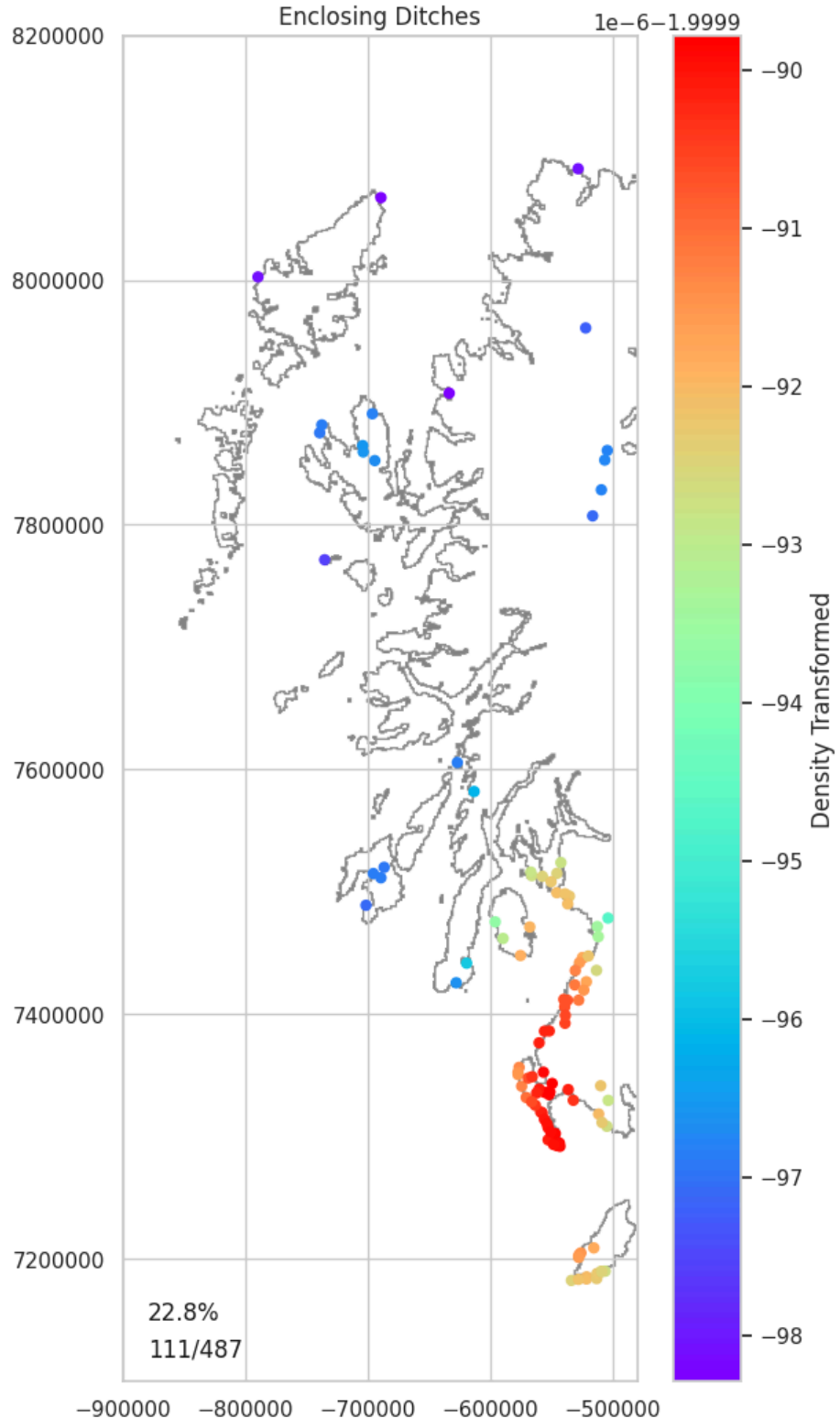
Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk



Middleton, M. 2024, Hillforts Primer Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Annex Encodeable Data

In [108…
```
annex_encodeable_features = ['Annex']
```

In [109…
```
plot_encodeable(hillforts_data, north_west, annex_encodeable_features, show_percentage)
```

487

Dating Data Download Package

If you do not wish to download the data using this document, all the processed data packages, notebooks and images are available here:
https://github.com/MikeDairsie/Hillforts-Primer.

In [110…
```
#download(dating_data_list, 'Dating_package')
```

Save Figure List

In [111…
```
if save_images:
    path = os.path.join(IMAGES_PATH, f"fig_list_{part.lower()}.csv")
    fig_list.to_csv(path, index=False)
```