Hillforts Primer

An Analysis of the Atlas of Hillforts of Britain and Ireland

Part 3

Mike Middleton, March 2022
https://orcid.org/0000-0001-5813-6347

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
HTML: Read only topographic

Part 2: Management & Landscape

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic

Part 3: Boundary & Dating

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic

- Boundary Data
- Dating Data

Part 4: Investigations & Interior

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic

Part 5: Entrance, Enclosing & Annex

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic

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

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic

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 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 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 [ ]:
```
download_data = False
```

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

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

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 Part 3.

Reload Data and Python Functions

Source Data

The Atlas of Hillforts of Britain and Ireland data is made available under the licence, Attribution-ShareAlike 4.0 International (CC BY-SA 4.0). This allows for redistribution, sharing and transformation of the data, as long as the results are credited and made available under the same licence conditions.

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/A72C523E8B6742ED97BA86470E747C6sample.pdf

Python Modules and Code Setup

In [ ]:
```
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 c

from slugify import slugify

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

```
Python: 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
Scikit-Learn: 1.2.2
pandas: 1.5.3
numpy: 1.22.4
matplotlib: 3.7.1
seaborn: 0.12.2
scipy: 1.10.1
```

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 [ ]:
```
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=
```

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

    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 [ ]:
```
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 [ ]:
```
def add_annotation_plot(ax):
    ax.annotate("Middleton, M. 2022, Hillforts Primer", size='small', color='grey'
    ax.annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", size
```

In [ ]:
```
def add_annotation_l_xy(ax):
    ax.annotate("Middleton, M. 2022, Hillforts Primer", size='small', color='grey'
    ax.annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", size
```

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

In [ ]:
```
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 [ ]:
```
def plot_over_grey_numeric(merged_data, a_type, title, extra="", inner=False, fring
    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 [ ]:
```
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 [ ]:
```
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
    plt.colorbar(label='Density')
    plt.title(get_print_title(f'Density - {data_type}'))
    save_fig(f'Density_{data_type}')
    plt.show()
```

In [ ]:
```
def add_21Ha_line():
    x_values = [-367969, -344171, -263690, -194654, -130542, -119597, -162994, -26
    y_values = [7019842, 6944572, 6850593, 6779602, 6735058, 6710127, 6684152, 666
    plt.plot(x_values, y_values, 'k', ls='-', lw=15, alpha=0.25, label = '≥ 21 Ha
    add_to_legend = Line2D([0], [0], color='k', lw=15, alpha=0.25, label = '≥ 21 Ha
    return add_to_legend
```

In [ ]:
```
def add_21Ha_fringe():
    x_values = [-367969,-126771,29679,-42657,-248650,-304545,-423647,-584307,-3679
    y_values = [7019842,6847138,6671658,6596650,6554366,6611780,6662041,6752378,70
    plt.plot(x_values, y_values, 'k', ls=':', lw=5, alpha=0.45, label = '≥ 21 Ha Fr
    add_to_legend = Line2D([0], [0], color='k', ls=':', lw=5, alpha=0.45, label =
    return add_to_legend
```

In [ ]:
```
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 [ ]:
```
def plot_over_grey(merged_data, a_type, yes_no, extra="", inner=False, fringe=Fals
    # plots selected data over the grey dots. yes_no controlls filtering the data
    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 [ ]:
```
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_ty
    plt.colorbar(label=data_type)
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

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

In [ ]:
```
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(e
    missing = list(set(main_data)-set(split_features))
    if missing:
        print(f"There are missing features: {missing}")
        m = True
    return m
```

In [ ]:
```
def review_data_split(main_data, numeric_data, text_data, encodeable_data = pd.Dat
    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 [ ]:
```
def find_duplicates(data):
    print(f'{data.count() - data.duplicated(keep=False).count()} duplicates.')
```

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

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

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

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

In [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
fig_no = 0
part = 'Part03'
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 [ ]:
```
def get_file_name(title):
    file_name = slugify(title)
    return file_name
```

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

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

```
Mounted at /content/drive
```

In [ ]:
```
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_1
        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 [ ]:
```
hillforts_csv = r"https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/ma
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-54-2b53084ab660>:2: DtypeWarning: Columns (10,12,68,83,84,85,86,16
5,183) have mixed types. Specify dtype option on import or set low_memory=False.
  hillforts_data = pd.read_csv(hillforts_csv, index_col=False)
```

Out[ ]:

	OBJECTID	Main_Atlas_Number	Main_Country_Code	Main_Country	Main_Title_Name	Main_Site
0	1	1	EN	England	EN0001 Aconbury Camp, Herefordshire	Aconbur
1	2	2	EN	England	EN0002 Bach Camp, Herefordshire	Bac

Download Function

In [ ]:
```
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
                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 [ ]:
```
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[ ]:

	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 [ ]:
```
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[ ]:

	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 [ ]:
```
cluster_data = hillforts_data[['Location_X','Location_Y', 'Main_Country_Code']].co
cluster_data['Cluster'] = 'NA'
```

```
cluster_data['Cluster'].where(cluster_data['Main_Country_Code'] != 'NI', 'I', inpla
cluster_data['Cluster'].where(cluster_data['Main_Country_Code'] != 'IR', 'I', inpla

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'I') & (cluster_data['Location_Y'] >= 7060000) , 'No
   )
north_ireland = cluster_data[cluster_data['Cluster'] == 'North Ireland'].copy()

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

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

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'NA') & (cluster_data['Location_Y'] >= 7070000) & (
   )
north_east = cluster_data[cluster_data['Cluster'] == 'Northeast'].copy()

cluster_data['Cluster'] = np.where(
   (cluster_data['Cluster'] == 'NA') & (cluster_data['Location_Y'] >= 7070000) & (
   )
north_west = cluster_data[cluster_data['Cluster'] == 'Northwest'].copy()

temp_cluster_location_packages = [north_ireland, south_ireland, south, north_east,

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],
```

Review Data Part 3

Boundary Data

The boundary data contains eight features.

In [ ]:
```
boundary_features = [
 'Boundary_Boundary_Type',
 'Boundary_Boundary_Comments',
 'Boundary_Country_Code_2',
 'Boundary_HER_2',
 'Boundary_HER_PRN_2',
 'Boundary_Current_County_2',
 'Boundary_Historic_County_2',
 'Boundary_Current_Parish_2']

boundary_data = hillforts_data[boundary_features].copy()
boundary_data.head()
```

Out[ ]:

	Boundary_Boundary_Type	Boundary_Boundary_Comments	Boundary_Country_Code_2	Boundary
0	NaN	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	County	NaN	NaN	Worces

The boundary data is partial. Five of the features contain 20 records or less. There is so little data in these features that their distributions are not useful. These five features will be dropped from the reprocessed download.

In [ ]:
```
boundary_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 8 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Boundary_Boundary_Type      428 non-null    object
 1   Boundary_Boundary_Comments  259 non-null    object
 2   Boundary_Country_Code_2     7 non-null      object
 3   Boundary_HER_2              12 non-null     object
 4   Boundary_HER_PRN_2          10 non-null     object
 5   Boundary_Current_County_2   20 non-null     object
 6   Boundary_Historic_County_2  17 non-null     object
 7   Boundary_Current_Parish_2   360 non-null    object
dtypes: object(8)
memory usage: 259.3+ KB
```

Boundary Numeric Data

There is no Boundary numeric data.

In [ ]:
```
boundary_numeric_data = pd.DataFrame()
```

Boundary Text Data

There is a single boundary text feature.

In [ ]:
```
boundary_text_features = [
 'Boundary_Boundary_Comments']

boundary_text_data = boundary_data[boundary_text_features].copy()
boundary_text_data[boundary_text_data['Boundary_Boundary_Comments'].notna()].head(
```

Out[ ]:

	Boundary_Boundary_Comments
9	Part of the site is located in Shropshire and ...
40	Part in Shropshire and part in Wales (Powys).
70	Part of site in Wales (Powys), part in England...
95	Formerly bisected by the historic counties of ...
96	Although situated entirely in Thatcham it lies...

Boundary Text Data - Resolve Null Values

Test for 'NA'.

In [ ]:
```
test_cat_list_for_NA(boundary_text_data, boundary_text_features)
```

```
Boundary_Boundary_Comments 0
```

Fill null values with 'NA'.

In [ ]:
```
boundary_text_data = update_cat_list_for_NA(boundary_text_data, boundary_text_featu
boundary_text_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 1 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Boundary_Boundary_Comments  4147 non-null   object
dtypes: object(1)
memory usage: 32.5+ KB
```

Boundary Encodeable Data

There are seven Boundary encodable features. As mentioned above, five of these features contain so little data they will be dropped from the download. Before being dropped, the data will be plotted.

In [ ]:
```
boundary_encodeable_features = [
 'Boundary_Boundary_Type',
 'Boundary_Country_Code_2',
 'Boundary_HER_2',
 'Boundary_HER_PRN_2',
 'Boundary_Current_County_2',
 'Boundary_Historic_County_2',
 'Boundary_Current_Parish_2']

boundary_encodeable_data = boundary_data[boundary_encodeable_features].copy()
boundary_encodeable_data.head()
```

Out[ ]:

	Boundary_Boundary_Type	Boundary_Country_Code_2	Boundary_HER_2	Boundary_HER_PRN_2	E
0	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	County	NaN	Worcestershire	WSM00932	

In [ ]:
```
boundary_encodeable_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 7 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Boundary_Boundary_Type      428 non-null    object
 1   Boundary_Country_Code_2     7 non-null      object
 2   Boundary_HER_2              12 non-null     object
 3   Boundary_HER_PRN_2          10 non-null     object
 4   Boundary_Current_County_2   20 non-null     object
 5   Boundary_Historic_County_2  17 non-null     object
 6   Boundary_Current_Parish_2   360 non-null    object
dtypes: object(7)
memory usage: 226.9+ KB
```

Boundary Encodeable Data - Resolve Null Values

All features in this dataset contain null values. 'NA' is not currently present in any feature and will be used to replace null values.

In [ ]:
```
test_cat_list_for_NA(boundary_encodeable_data, boundary_encodeable_features)
```

```
Boundary_Boundary_Type 0
Boundary_Country_Code_2 0
Boundary_HER_2 0
Boundary_HER_PRN_2 0
Boundary_Current_County_2 0
Boundary_Historic_County_2 0
Boundary_Current_Parish_2 0
```

Null values updated to 'NA'.

In [ ]:
```
boundary_encodeable_data = update_cat_list_for_NA(boundary_encodeable_data, boundar
```

In [ ]:
```
boundary_encodeable_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 7 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Boundary_Boundary_Type      4147 non-null   object
 1   Boundary_Country_Code_2     4147 non-null   object
 2   Boundary_HER_2              4147 non-null   object
 3   Boundary_HER_PRN_2          4147 non-null   object
 4   Boundary_Current_County_2   4147 non-null   object
 5   Boundary_Historic_County_2  4147 non-null   object
 6   Boundary_Current_Parish_2   4147 non-null   object
dtypes: object(7)
memory usage: 226.9+ KB
```

Boundary Type Plotted

The majority of hillforts (3719/89.68%) have no Boundary Type information.

In [ ]:
```
boundary_encodeable_data['Boundary_Boundary_Type'].value_counts()
```

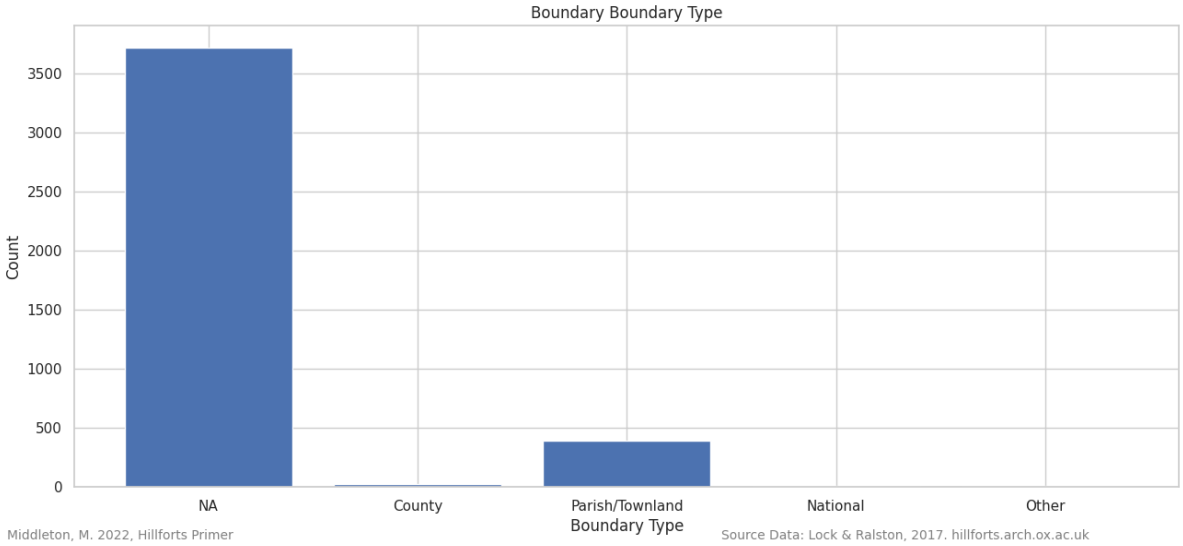
Out[ ]:
```
NA                3719
Parish/Townland    391
County              22
Other                8
National             7
Name: Boundary_Boundary_Type, dtype: int64
```

In [ ]:
```
x_data = []
for bdry in list(pd.unique(boundary_encodeable_data['Boundary_Boundary_Type'])):
    x_data.append(bdry)

y_data = []
for entry in x_data:
    count = len(boundary_encodeable_data[boundary_encodeable_data['Boundary_Boundar
    y_data.append(count)
```

In [ ]:
```
plot_bar_chart_using_two_tables(x_data, y_data, 'Boundary Type', 'Count', 'Boundary
```

```
Saving figure hillforts_primer_part03-001.png
```

```
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



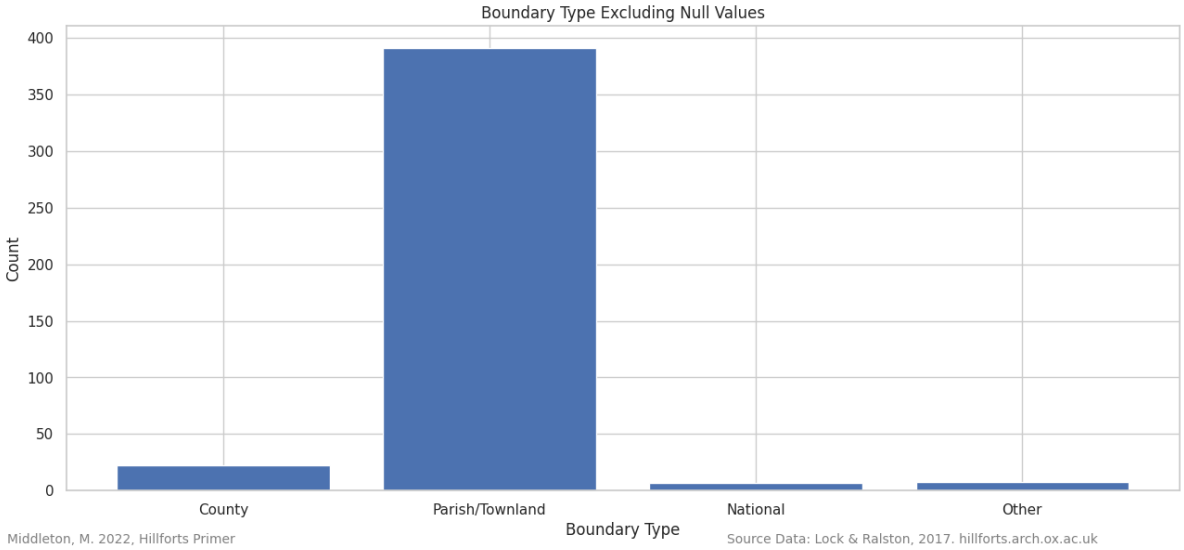
Boundary Data Plotted (Excluding No Boundary information)

Where a Boundary Type has been recorded, Parish/Townland is the most common (391). There are a small number of hillforts on a county boundary (22) and an even smaller number on a national border (7).

In [ ]:
```
x_data_short = x_data[1:]
y_data_short = y_data[1:]
```

In [ ]:
```
plot_bar_chart_using_two_tables(x_data_short, y_data_short, 'Boundary Type', 'Coun
```

```
Saving figure hillforts_primer_part03-002.png
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



Boundary Data Mapped

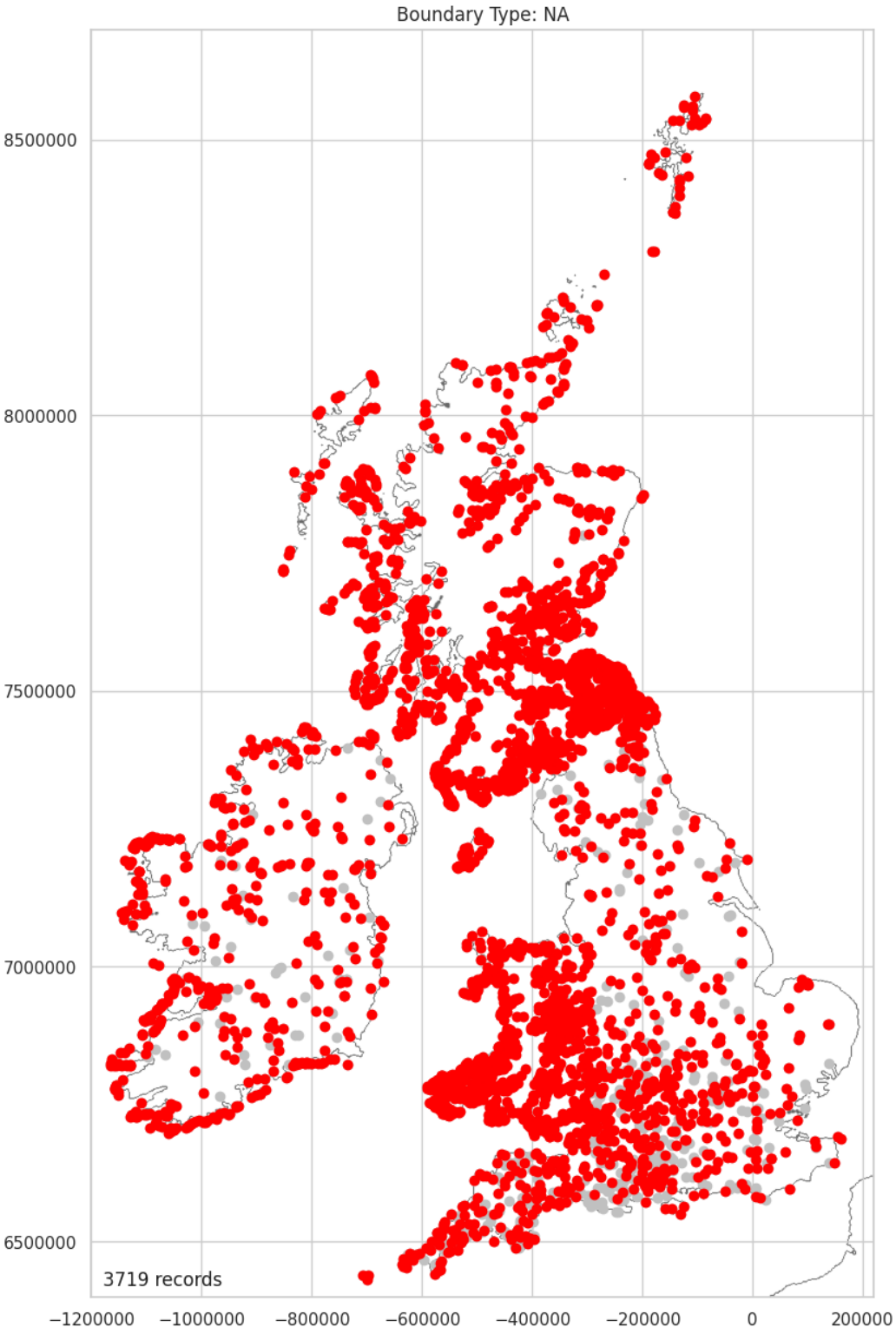
In [ ]:
```
location_boundary_data = pd.merge(location_numeric_data_short, boundary_encodeable_
```

No Boundary Data Mapped

3719 hillforts (89.68%) have no associated boundary information.

In [ ]:
```
plot_over_grey_boundary(location_boundary_data, 'Boundary_Boundary_Type', 'NA')
```

```
Saving figure hillforts_primer_part03-003.png
```



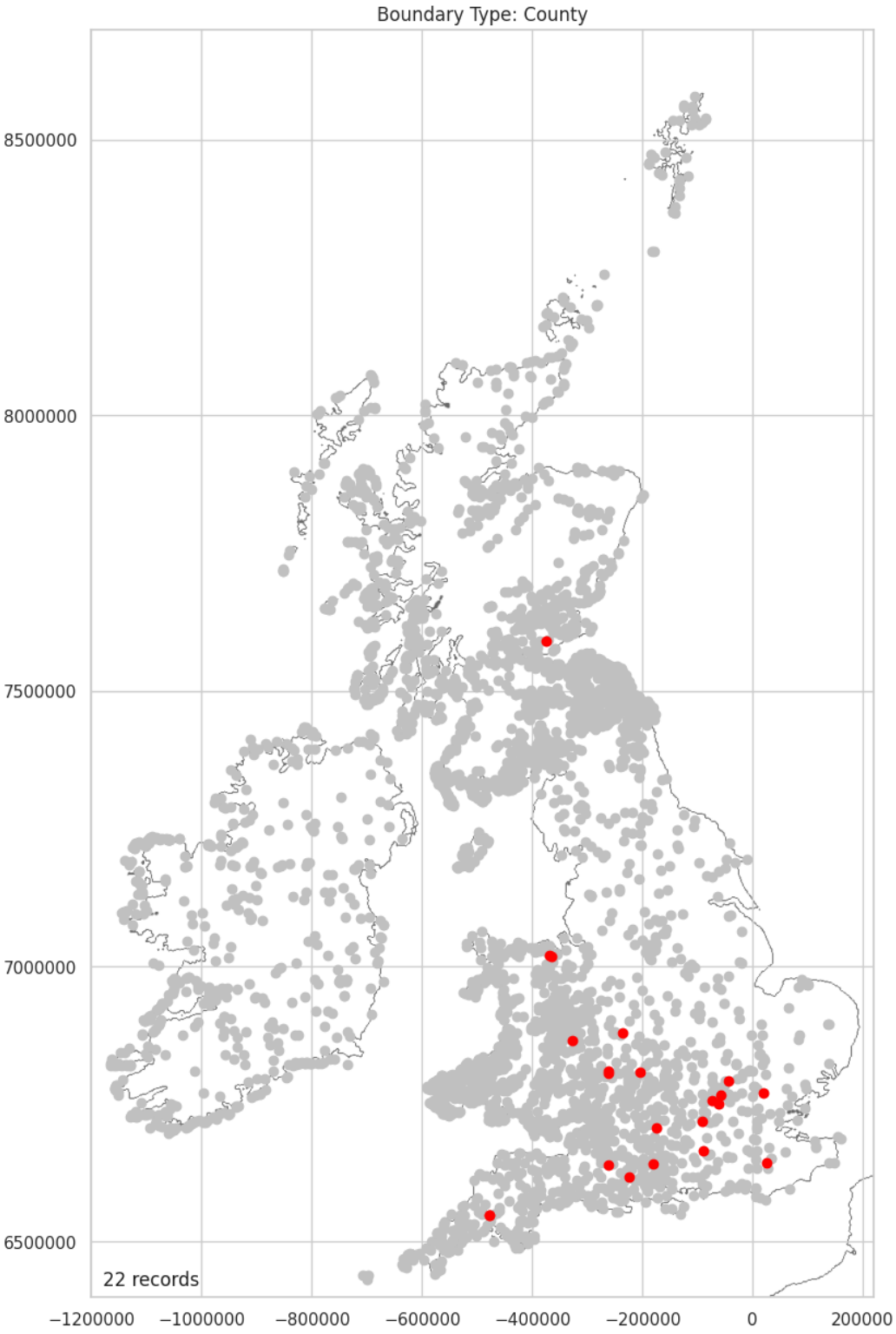
89.68%

County Boundary Mapped

Only 22 hillforts have a relationship to county boundaries.

In [ ]:
```
plot_over_grey_boundary(location_boundary_data, 'Boundary_Boundary_Type', 'County'
```

```
Saving figure hillforts_primer_part03-004.png
```



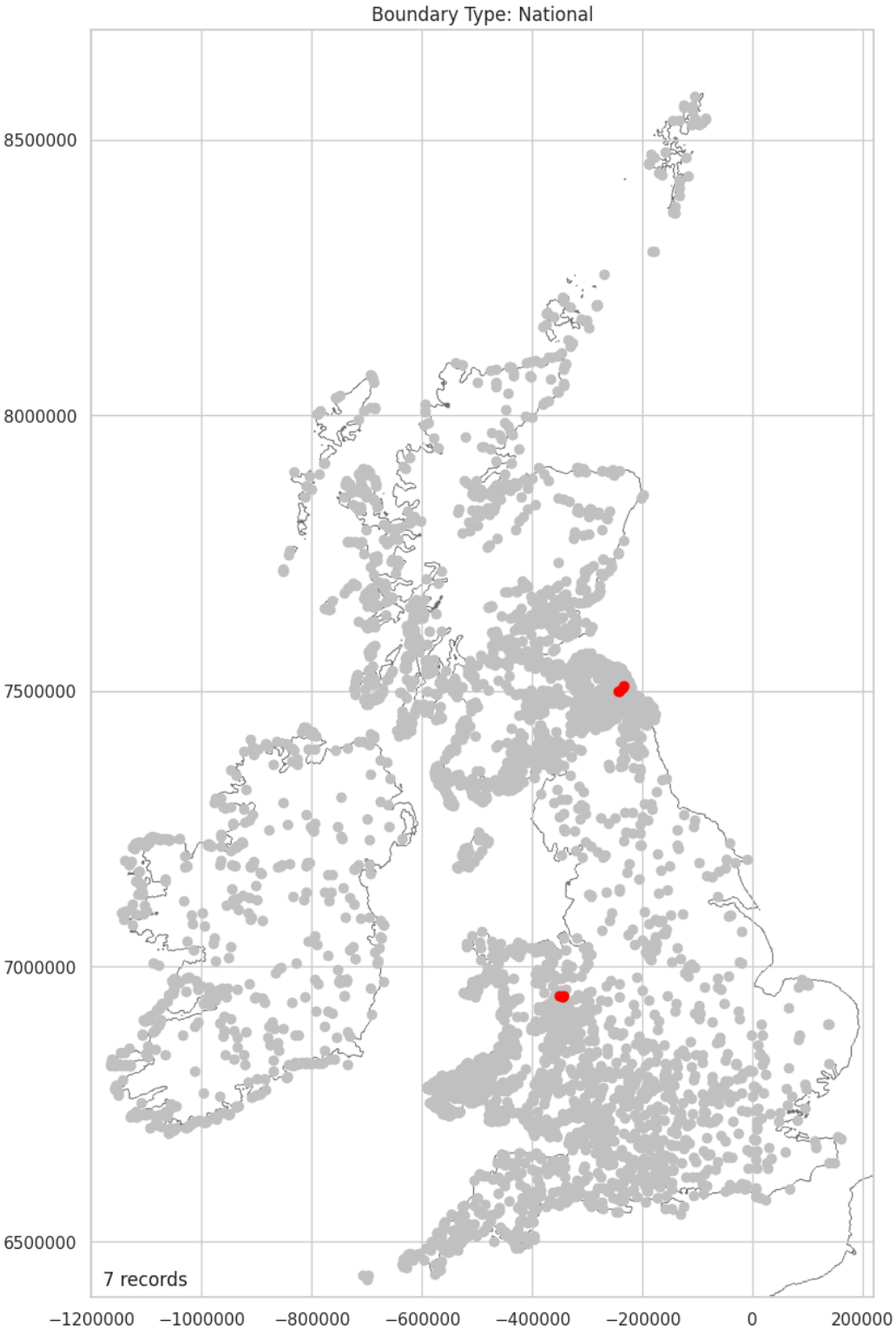
0.53%

National Boundary Mapped

Only seven hillforts have been recorded as built, on what is now, a national boundary.

In [ ]:
```
plot_over_grey_boundary(location_boundary_data, 'Boundary_Boundary_Type', 'Nationa
```

```
Saving figure hillforts_primer_part03-005.png
```



`0.17%`

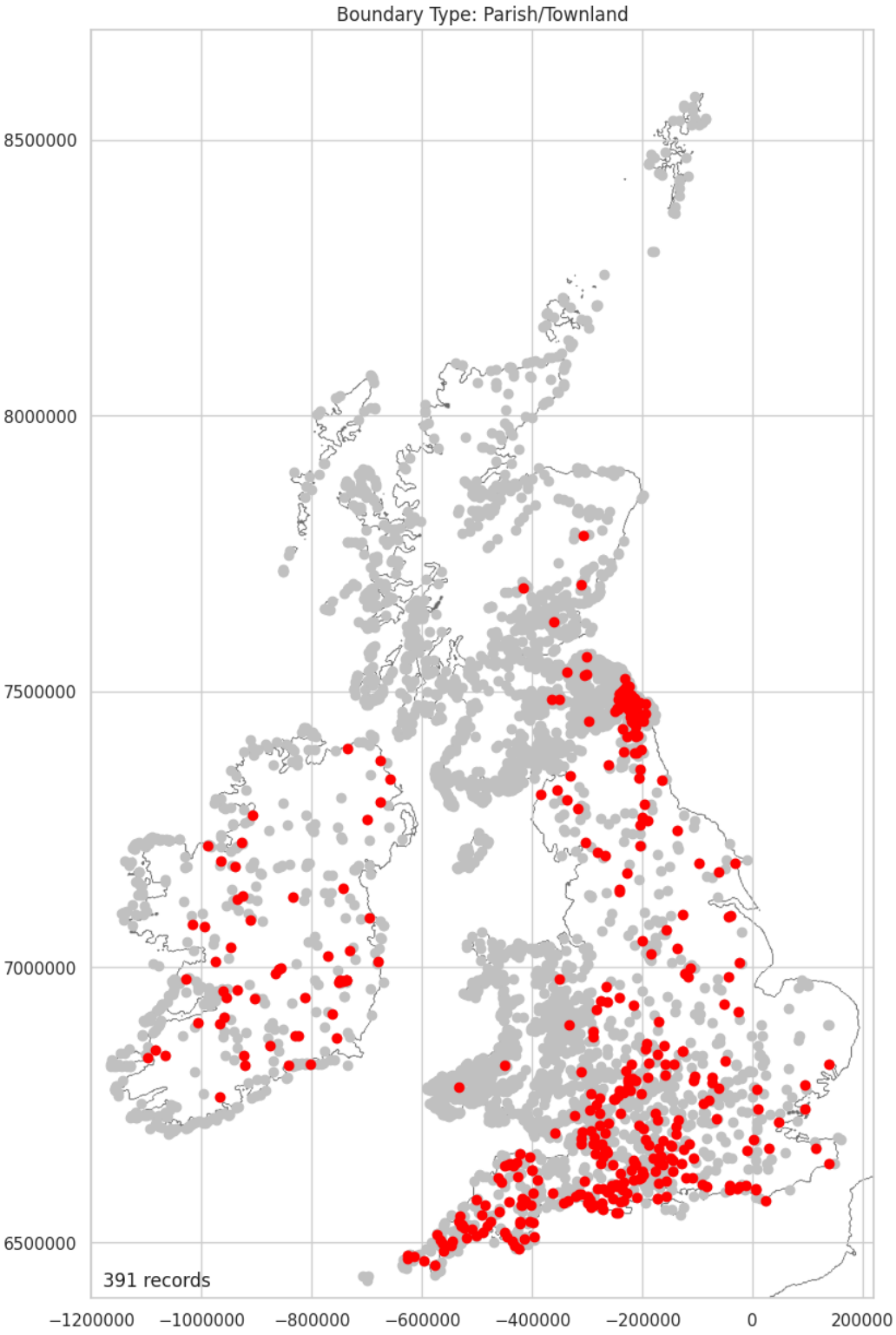
Parish / Townland Boundary Mapped

This distribution contains a survey bias. The data for England and the Republic of Ireland looks to have some coherence but the data for Scotland, Wales and Northern Ireland is patchy. There also seems to be recording bias, caused by localised intense recording,

around Berwick and various locations across the south of England. See: Boundary Current Parish 2 Mapped.

In [ ]:
```
plot_over_grey_boundary(location_boundary_data, 'Boundary_Boundary_Type', 'Parish/
```

```
Saving figure hillforts_primer_part03-006.png
```



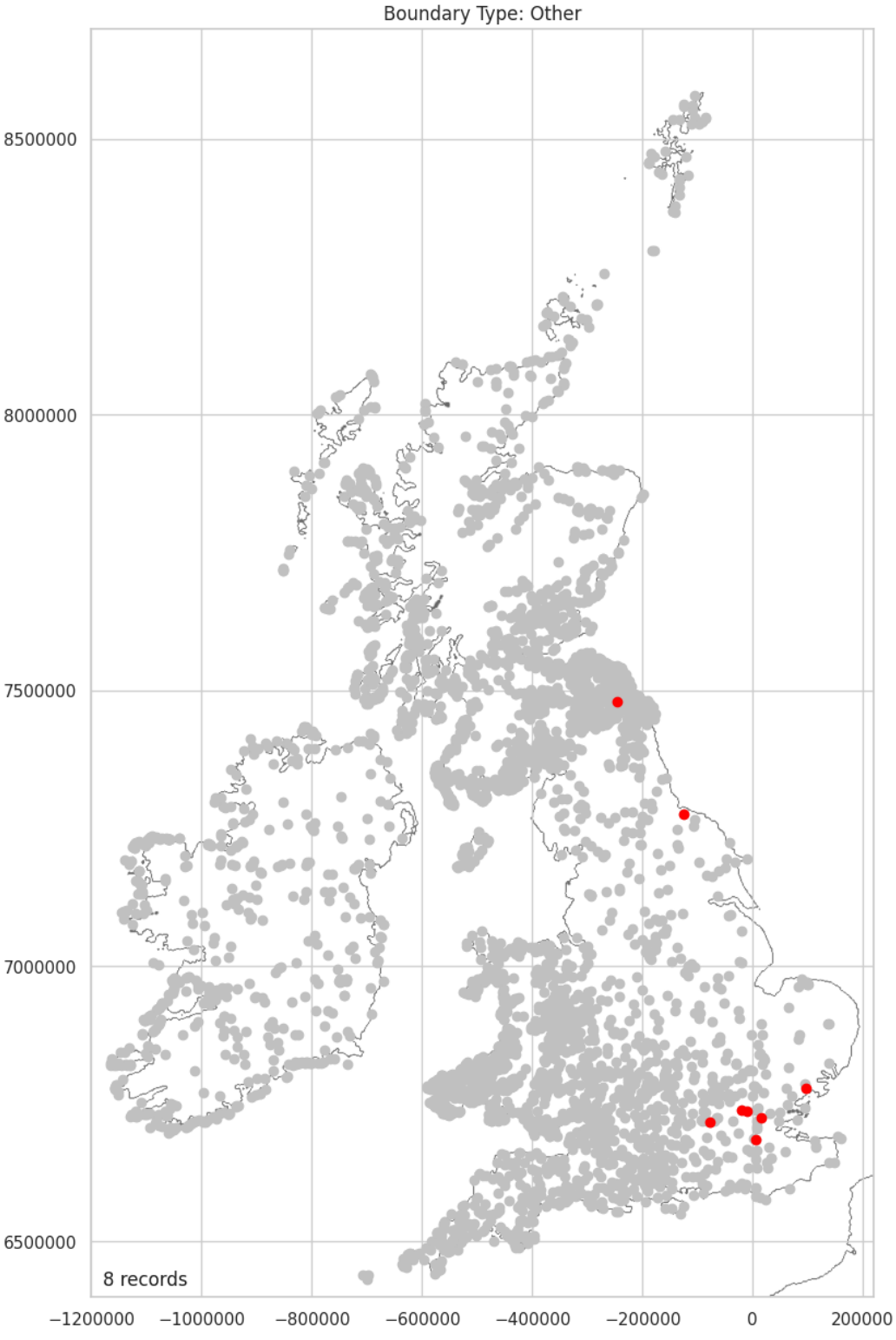

```
9.43%
```

Other Boundary Mapped

Eight hillforts coincide with boundaries classified as other.

In [ ]:
```
plot_over_grey_boundary(location_boundary_data, 'Boundary_Boundary_Type', 'Other')
```

```
Saving figure hillforts_primer_part03-007.png
```




```
0.19%
```

Boundary Country Code 2

Seven hillforts have a Boundary Code 2. There are three codes used which relate to England (EN), Scotland (SC) and Wales (WA).

In [ ]:
```
boundary_encodeable_data['Boundary_Country_Code_2'].value_counts()
```

Out[ ]:
```
NA    4140
EN       4
WA       2
SC       1
Name: Boundary_Country_Code_2, dtype: int64
```

Boundary HER 2

There are 12 hillforts with a Boundary HER 2. There are nine values, of which most have a single entry. All are HER service names.

In [ ]:
```
boundary_encodeable_data['Boundary_HER_2'].value_counts()
```

Out[ ]:
```
NA                      4135
Worcestershire             2
Shropshire                 2
Clwyd Powys                2
West Berkshire             1
Dudley                     1
Hampshire                  1
Fife Council               1
East Sussex                1
Wiltshire and Swindon      1
Name: Boundary_HER_2, dtype: int64
```

Boundary HER PRN 2

There are ten hillforts with a HER PRN 2 value. There are nine unique values, of which most have a single entry. All are HER PRN ID numbers.

In [ ]:
```
boundary_encodeable_data['Boundary_HER_PRN_2'].value_counts()
```

Out[ ]:
```
NA          4137
WSM00932       2
MSA868         1
19259          1
MSA828         1
MWB3075        1
7097           1
3252           1
28636          1
MWI17466       1
Name: Boundary_HER_PRN_2, dtype: int64
```

Boundary Current County 2

There are 20 hillforts with a Boundary Current County 2. There are 15 unique values, of which most have a single entry. All are county names.

In [ ]:
```
boundary_encodeable_data['Boundary_Current_County_2'].value_counts()
```

Out[ ]:
```
NA                          4127
Worcestershire                 3
Shropshire                     2
Powys                          2
Wiltshire                      2
West Berkshire                 1
West Midlands                  1
Somerset                       1
Flintshire                     1
Denbighshire                   1
Hampshire                      1
Fife                           1
Hertfordshire                  1
Windsor and Maidenhead         1
Cornwall                       1
East Sussex                    1
Name: Boundary_Current_County_2, dtype: int64
```

Boundary Historic County 2

There are 17 hillforts with a Boundary Historic County 2. There are 13 unique values, of which most have a single entry. All are historic county names.

In [ ]:
```
boundary_encodeable_data['Boundary_Historic_County_2'].value_counts()
```

Out[ ]:
```
NA                  4130
Worcestershire         3
Shropshire             2
Denbighshire           2
Montgomeryshire        1
Hampshire              1
Berkshire              1
Mointgomeryshire       1
Staffordshire          1
Somerset               1
Fife                   1
Selkirkshire           1
Sussex                 1
Wiltshire              1
Name: Boundary_Historic_County_2, dtype: int64
```

Boundary Current Parish 2

There are 14 hillforts with a Boundary Current Parish 2. There are four unique values.

In [ ]:
```
boundary_encodeable_data['Boundary_Current_Parish_2'].value_counts()[:5]
```

Out[ ]:
```
NA           3787
Branxton        4
Chatton         4
Doddington      3
Lowick          3
Name: Boundary_Current_Parish_2, dtype: int64
```

Boundary Current Parish 2 Mapped

This feature is very similar to, and suffers the same survey bias as, that detailed in Parish / Townland Boundary Mapped.

In [ ]:
```
temp_boundary_data_plus = boundary_encodeable_data.copy()
temp_boundary_data_plus.where(temp_boundary_data_plus['Boundary_Current_Parish_2']
temp_boundary_data_plus.head()
```

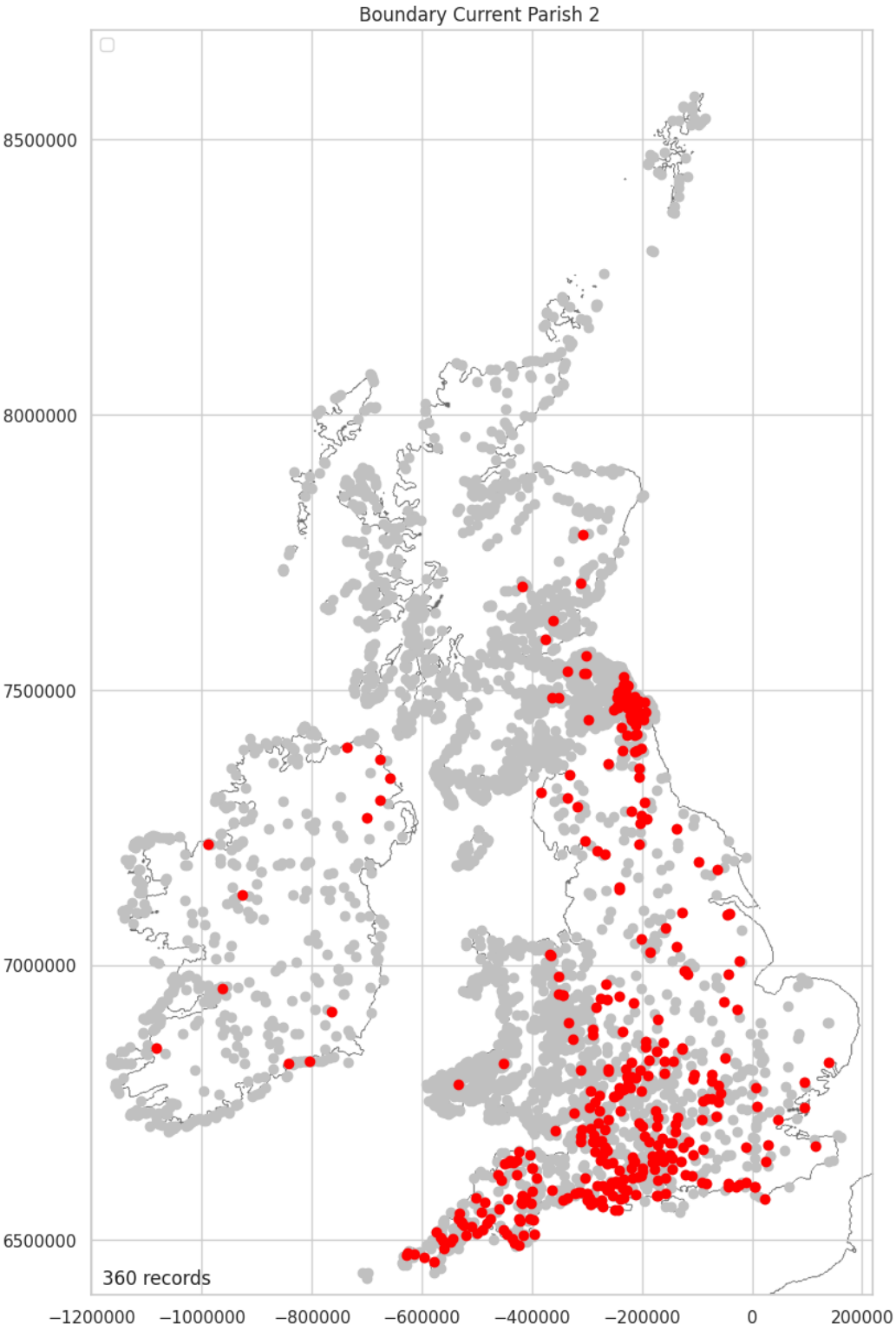
Out[ ]:

	Boundary_Boundary_Type	Boundary_Country_Code_2	Boundary_HER_2	Boundary_HER_PRN_2	E
0	NA	NA	NA	NA	
1	NA	NA	NA	NA	
2	NA	NA	NA	NA	
3	NA	NA	NA	NA	
4	Yes	Yes	Yes	Yes	

In [ ]:
```
temp_location_boundary_data = pd.merge(location_numeric_data_short, temp_boundary_(
```

In [ ]:
```
Boundary_Current_Parish_2_stats = plot_over_grey(temp_location_boundary_data, 'Bou
```

```
Saving figure hillforts_primer_part03-008.png
```



8.68%

Review Boundary Data Split

In [ ]:
```
review_data_split(boundary_data, boundary_numeric_data, boundary_text_data, boundar
```

```
Data split good.
```

Drop Features From Boundary Encodeable Data

Only Boundary Type and Current Parish 2 will be retained in the download data package. The remaining boundary features do not contain sufficient data and would likely be misleading, if used in a machine leaning model.

In [ ]:
```
boundary_encodable_features_short = [
 'Boundary_Boundary_Type',
 'Boundary_Current_Parish_2']
```

In [ ]:
```
boundary_encodable_short = boundary_encodeable_data[boundary_encodable_features_sho
boundary_encodable_short.head()
```

Out[ ]:

	Boundary_Boundary_Type	Boundary_Current_Parish_2
0	NA	NA
1	NA	NA
2	NA	NA
3	NA	NA
4	County	Eastnor (Herefordshire); Little Malvern (Worce...

Boundary Data Package

In [ ]:
```
boundary_data_list = [boundary_numeric_data, boundary_text_data, boundary_encodable
```

Boundary 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 [ ]:
```
download(boundary_data_list, 'Boundary_package')
```

Dating Data

Additional Dating information is held in a seperate Dating Evidence Table. This can be downloaded from the Hillforts Atlas Rest Service API here or this project's data store here. The Dating Evidence Table has not been analysed as part of the Hillforts Primer at this time.

There are 15 Dating features. The first eight record period date ranges. There are two features recording dating evidence of activity prior to construction and two features recording activity post abandon. The remaining thee features record dating reliability, related dating evidence and general dating comments.

In [ ]:
```
dating_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',
 'Dating_Date_Reliability',
 'Dating_Date_Comments',
 'Dating_Pre',
 'Dating_Pre_Comments',
 'Dating_Post',
 'Dating_Post_Comments',
 'Related_Dating_Evidence']

dating_data = hillforts_data[dating_features].copy()
dating_data.head()
```

Out[ ]:

	Dating_Date_Pre_1200BC	Dating_Date_1200BC_800BC	Dating_Date_800BC_400BC	Dating_Date_4(
0	No	No	Yes	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	Yes	

All the period features and the 'pre' and 'post' features contain yes/no responses. These contain no null entries. All the remaining features contain empty records. Reliability and Related dating evidence contain controlled vocabularies (data derived from a pick list).

In [ ]:
```
dating_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Dating_Date_Pre_1200BC   4147 non-null   object
 1   Dating_Date_1200BC_800BC 4147 non-null   object
 2   Dating_Date_800BC_400BC  4147 non-null   object
 3   Dating_Date_400BC_AD50   4147 non-null   object
 4   Dating_Date_AD50_AD400   4147 non-null   object
 5   Dating_Date_AD400_AD800  4147 non-null   object
 6   Dating_Date_Post_AD800   4147 non-null   object
 7   Dating_Date_Unknown      4147 non-null   object
 8   Dating_Date_Reliability  4134 non-null   object
 9   Dating_Date_Comments     4116 non-null   object
 10  Dating_Pre               4147 non-null   object
 11  Dating_Pre_Comments      448 non-null    object
 12  Dating_Post              4147 non-null   object
 13  Dating_Post_Comments     1961 non-null   object
 14  Related_Dating_Evidence  805 non-null    object
dtypes: object(15)
memory usage: 486.1+ KB
```

Dating Numeric Data

There is no numeric Dating Data.

In [ ]:
```
dating_numeric_data = pd.DataFrame()
```

Dating Text Data

There are three Dating text features.

In [ ]:
```
dating_text_features = [
 'Dating_Date_Comments',
 'Dating_Pre_Comments',
 'Dating_Post_Comments']

dating_text_data = dating_data[dating_text_features].copy()
dating_text_data.head()
```

Out[ ]:

	Dating_Date_Comments	Dating_Pre_Comments	Dating_Post_Comments
0	The finding of Iron Age and Roman pottery sugg...	NaN	Evidence of Civil War occupation and possible ...
1	None	NaN	NaN
2	The chance finding of a number of cloudy blue ...	None	None
3	Iron Age to Roman and possible later enclosure...	Possible Bronze Age ring ditch could indicate ...	Possible later Roman or post-Roman enclosure.
4	The earlier enclosure of Phase I could be late...	There is no evidence of pre-hillfort activity,...	The ringwork is thought to be of medieval date.

Dating Text Data - Resolve Null Values

Test for 'NA'.

In [ ]:
```
test_cat_list_for_NA(dating_text_data, dating_text_features)
```

```
Dating_Date_Comments 0
Dating_Pre_Comments 0
Dating_Post_Comments 0
```

Fill null values with 'NA'.

In [ ]:
```
dating_text_data = update_cat_list_for_NA(dating_text_data, dating_text_features)
dating_text_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Dating_Date_Comments  4147 non-null   object
 1   Dating_Pre_Comments   4147 non-null   object
 2   Dating_Post_Comments  4147 non-null   object
dtypes: object(3)
memory usage: 97.3+ KB
```

Dating Encodeable Data

In [ ]:
```
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',
 'Dating_Date_Reliability',
 'Dating_Pre',
 'Dating_Post',
 'Related_Dating_Evidence']

dating_encodeable_data = dating_data[dating_encodeable_features].copy()
dating_encodeable_data.head()
```

Out[ ]:

	Dating_Date_Pre_1200BC	Dating_Date_1200BC_800BC	Dating_Date_800BC_400BC	Dating_Date_4(
0	No	No	Yes	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	Yes	

Review Dating Data Split

In [ ]:
```
review_data_split(dating_data, dating_numeric_data, dating_text_data, dating_encod
```

```
Data split good.
```

Period Data

The majority of hillforts have no date infromation. The maps below show the data contains a recording bias in that the majority of dating information comes from southern England. There is very little dating information outside this area.

In [ ]:
```
date_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']

date_data = dating_encodeable_data[date_features]
date_data.head()
```

Out[ ]:

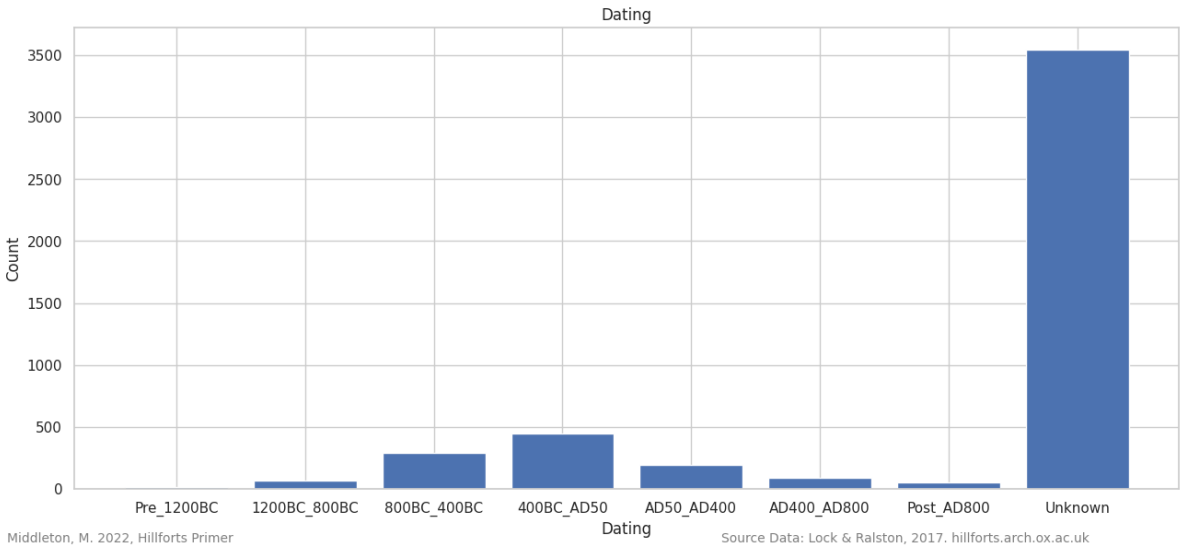
	Dating_Date_Pre_1200BC	Dating_Date_1200BC_800BC	Dating_Date_800BC_400BC	Dating_Date_4(
0	No	No	Yes	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	Yes	

Period Data Plotted

The majority of hillforts are undated.

In [ ]:
```
plot_bar_chart(date_data, 2, 'Dating', 'Count', 'Dating')
```

```
Saving figure hillforts_primer_part03-009.png
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



Date Data (Excluding No Dates) Plotted

For the relativly few hillforts that have dating evidence, the majority are dated to the Iron Age (800BC to AD50) with the largest cluster being the late Iron Age (400BC to AD50). There are relativly few pre Iron Age dates. In contrast, there are a number early medieval dates (AD50 to AD800) with most falling at the lower end between AD50 and AD400.

In [ ]:
```
date_features_minus = [
 'Dating_Date_Pre_1200BC',
```

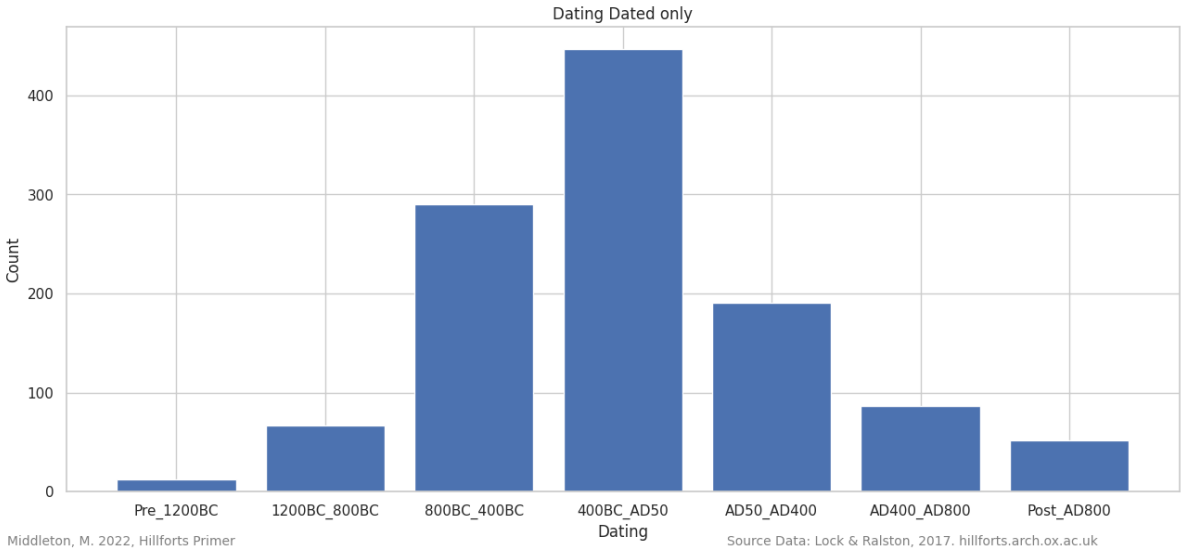
```
 'Dating_Date_1200BC_800BC',
 'Dating_Date_800BC_400BC',
 'Dating_Date_400BC_AD50',
 'Dating_Date_AD50_AD400',
 'Dating_Date_AD400_AD800',
 'Dating_Date_Post_AD800']

date_data_minus = hillforts_data[date_features_minus]
date_data_minus.head()
```

Out[ ]:

	Dating_Date_Pre_1200BC	Dating_Date_1200BC_800BC	Dating_Date_800BC_400BC	Dating_Date_4(
0	No	No	Yes	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	Yes	

In [ ]:
```
plot_bar_chart(date_data_minus, 2, 'Dating', 'Count', 'Dating_Dated_only')
```

```
Saving figure hillforts_primer_part03-010.png
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



Period Data Mapped

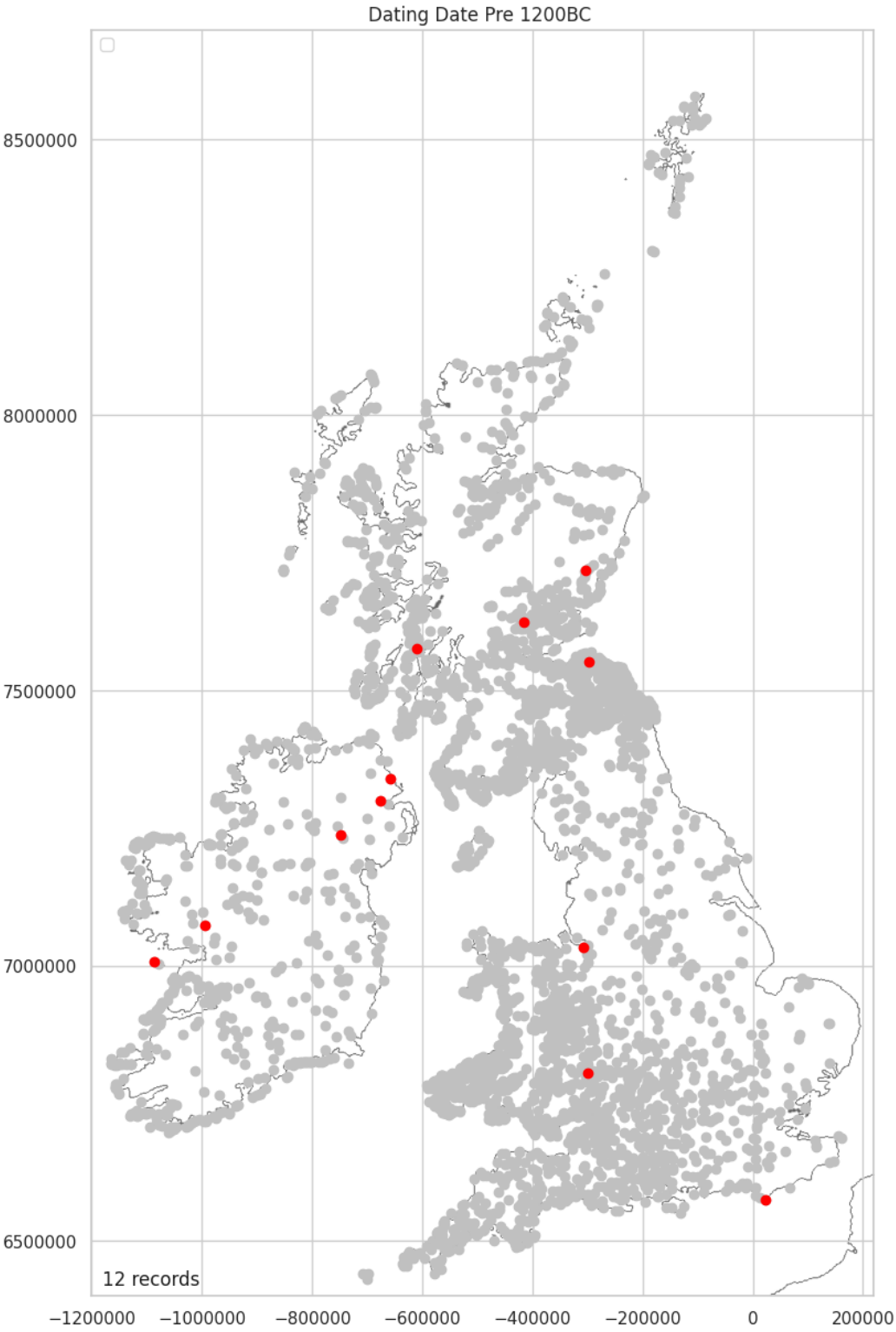
In [ ]:
```
location_date_data = pd.merge(location_numeric_data_short, date_data, left_index=Tr
```

Pre 1200 BC Mapped

There are only 12 hillforts which have produced dates pre 1200BC. A density plot has not been produced as there is insufficient data.

In [ ]:
```
dt_1200 = plot_over_grey(location_date_data, 'Dating_Date_Pre_1200BC', 'Yes')
```

```
Saving figure hillforts_primer_part03-011.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

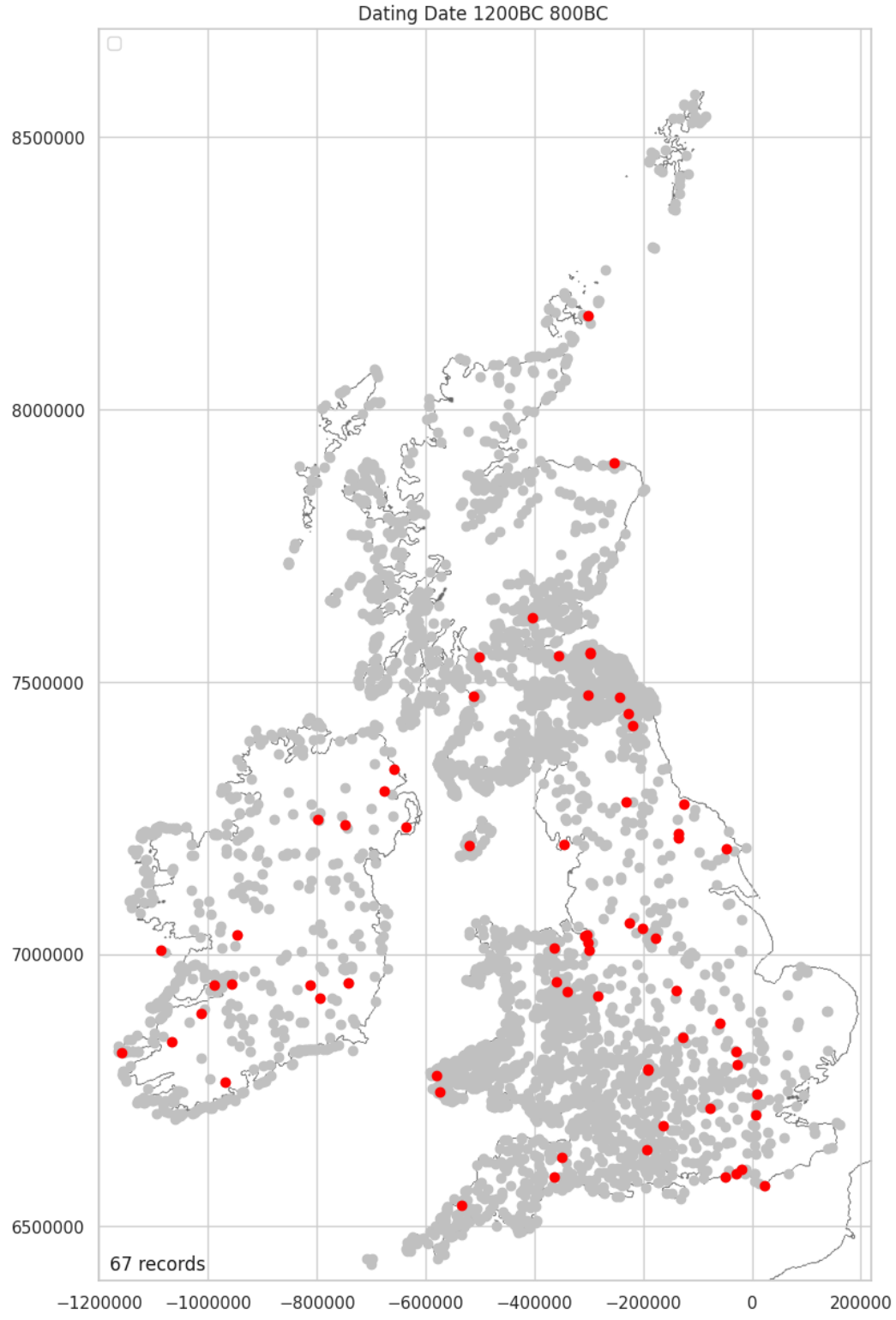
0.29%

1200BC - 800BC Mapped

Only 67 hillforts have a date between 1200 BC and 800 BC. A density plot has not been produced as there is insufficient data.

In [ ]:
```
dt_1200_800 = plot_over_grey(location_date_data, 'Dating_Date_1200BC_800BC', 'Yes'
```

```
Saving figure hillforts_primer_part03-012.png
```



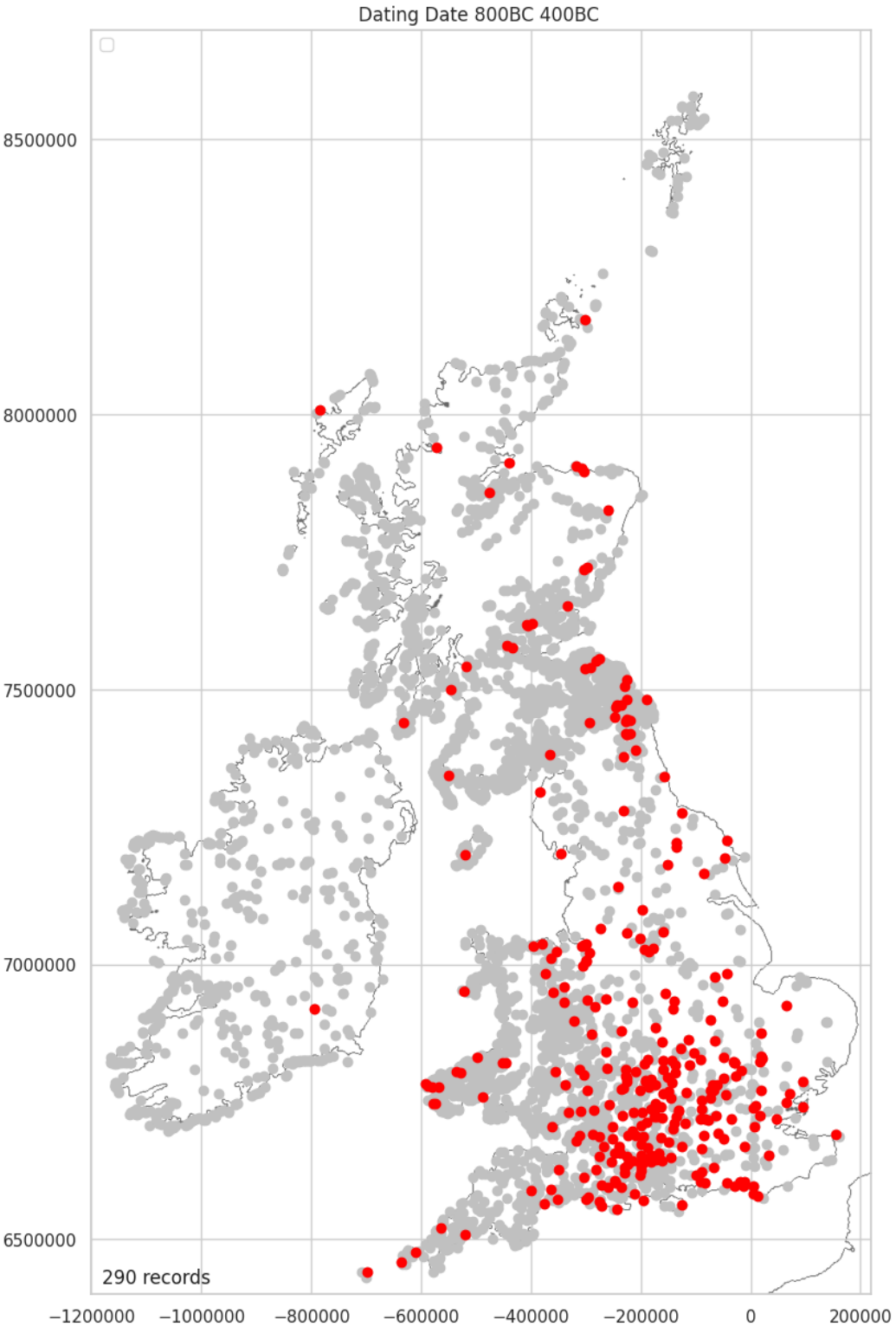

```
1.62%
```

800BC - 400BC Mapped

Hillfots dated 800 BC to 400 BC are located predominatly in south central England. There is a clear bias in this distribution with there being only a single fort in Ireland and very few in Scoland, Wales and south-west England.

In [ ]:
```
dt_800_400 = plot_over_grey(location_date_data, 'Dating_Date_800BC_400BC', 'Yes')
```

```
Saving figure hillforts_primer_part03-013.png
```



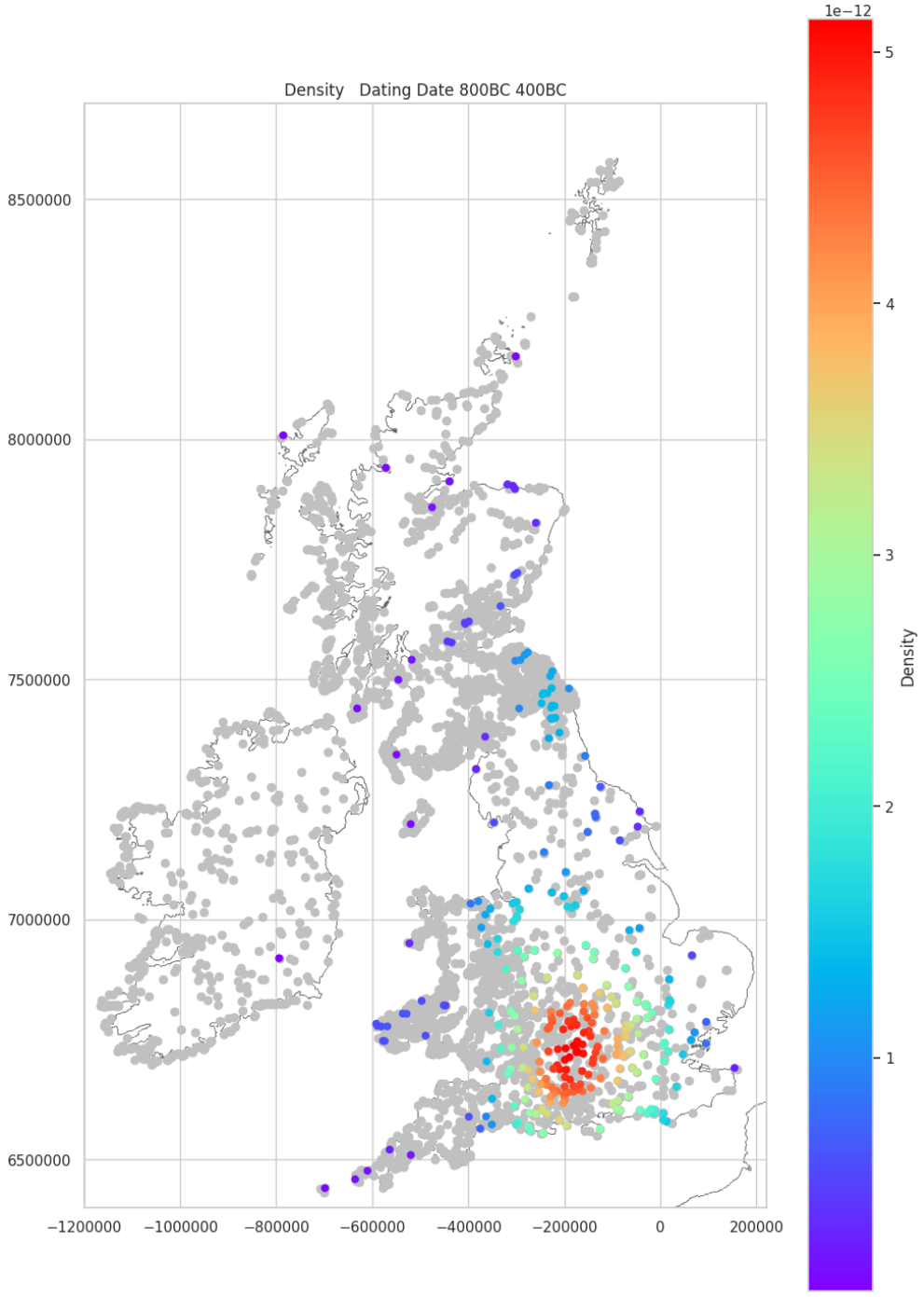
6.99%

800BC - 400BC Density Mapped

Although the density for this distribtion has been produced, it looks to be a highly missleading custer due to a recording bias.

In [ ]:
```
plot_density_over_grey(dt_800_400, 'Dating_Date_800BC_400BC')
```

```
Saving figure hillforts_primer_part03-014.png
```

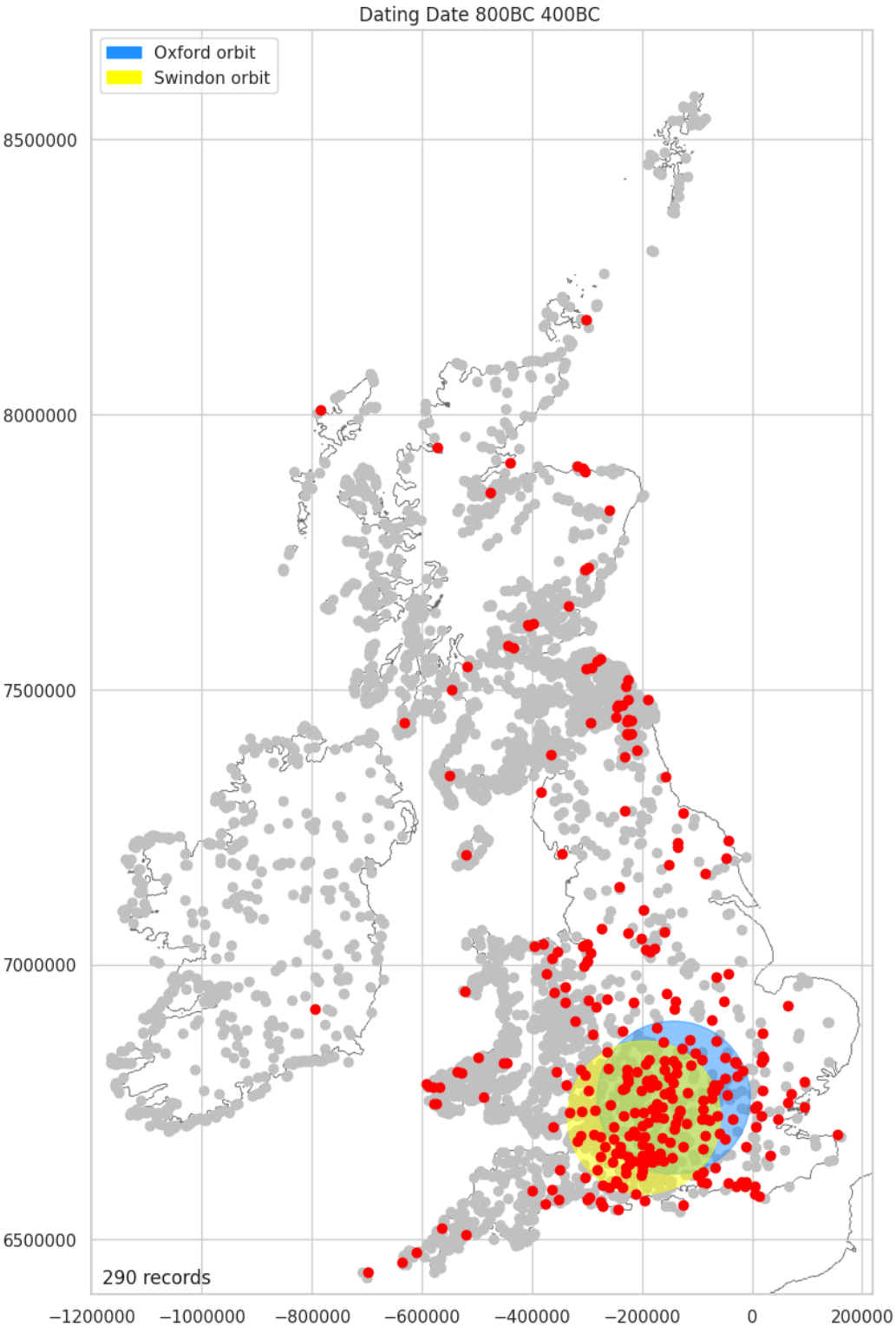


800BC - 400BC Mapped Plus Oxford and Swindon Orbits

Plotting the orbits of Oxford Universiy and the head office of Historic England, there is an almost exact corrilation between the overlaps of these two orbits and the most dense concentration of records in this cluster. This suggests the distribution is more likely to be the result of concentrated sampling in this area rather than being a meaningful distribution relating to this period. What the cluster does show is that, in this concentrated area, many hillforts have dates in this range.

In [ ]:
```
date_800bc_400bc = plot_over_grey(location_date_data, 'Dating_Date_800BC_400BC', '
```

```
Saving figure hillforts_primer_part03-015.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

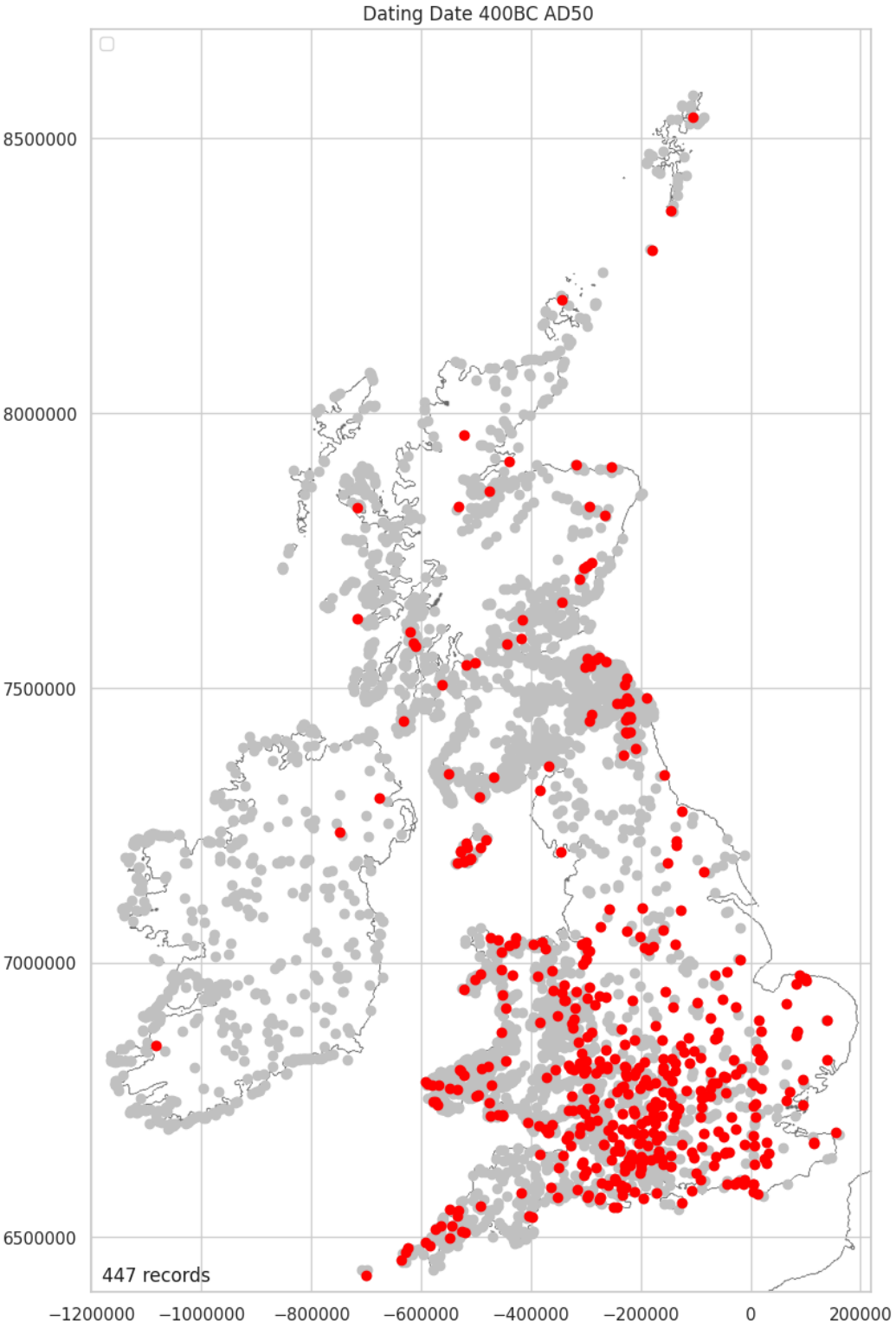
6.99%

400BC - AD50 Mapped

The reccording bias seen in 800 BC to 400 BC is again visable in this cluster and similar voids in the record can be seen acroos the rest of the atlas.

In [ ]:
```
dt_400_50 = plot_over_grey(location_date_data, 'Dating_Date_400BC_AD50', 'Yes')
```

```
Saving figure hillforts_primer_part03-016.png
```



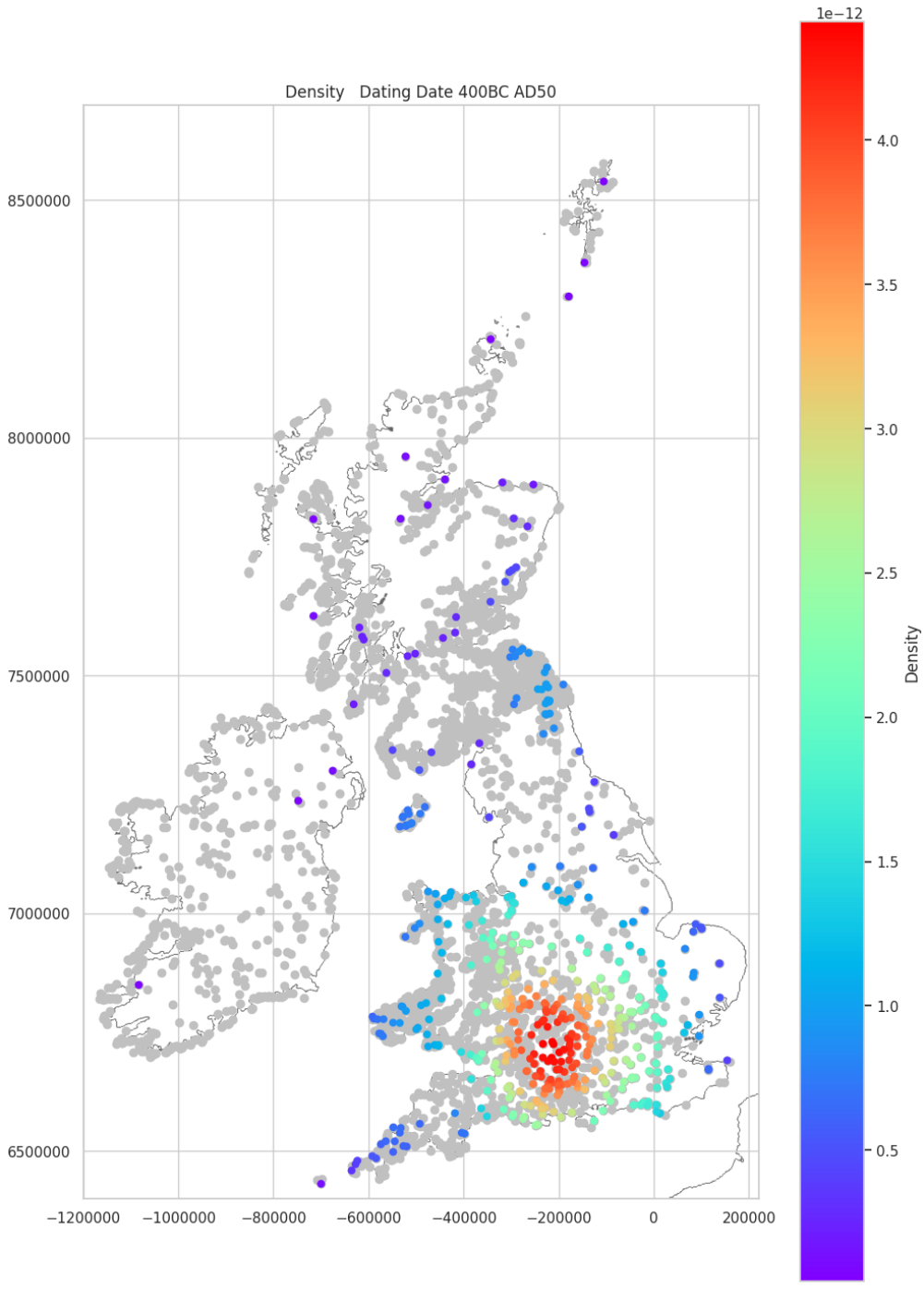

```
10.78%
```

400BC - AD50 Density Mapped

The cluster focus is identical to that seen in 800 BC to 400 BC.

In [ ]:
```
plot_density_over_grey(dt_400_50, 'Dating_Date_400BC_AD50')
```

```
Saving figure hillforts_primer_part03-017.png
```

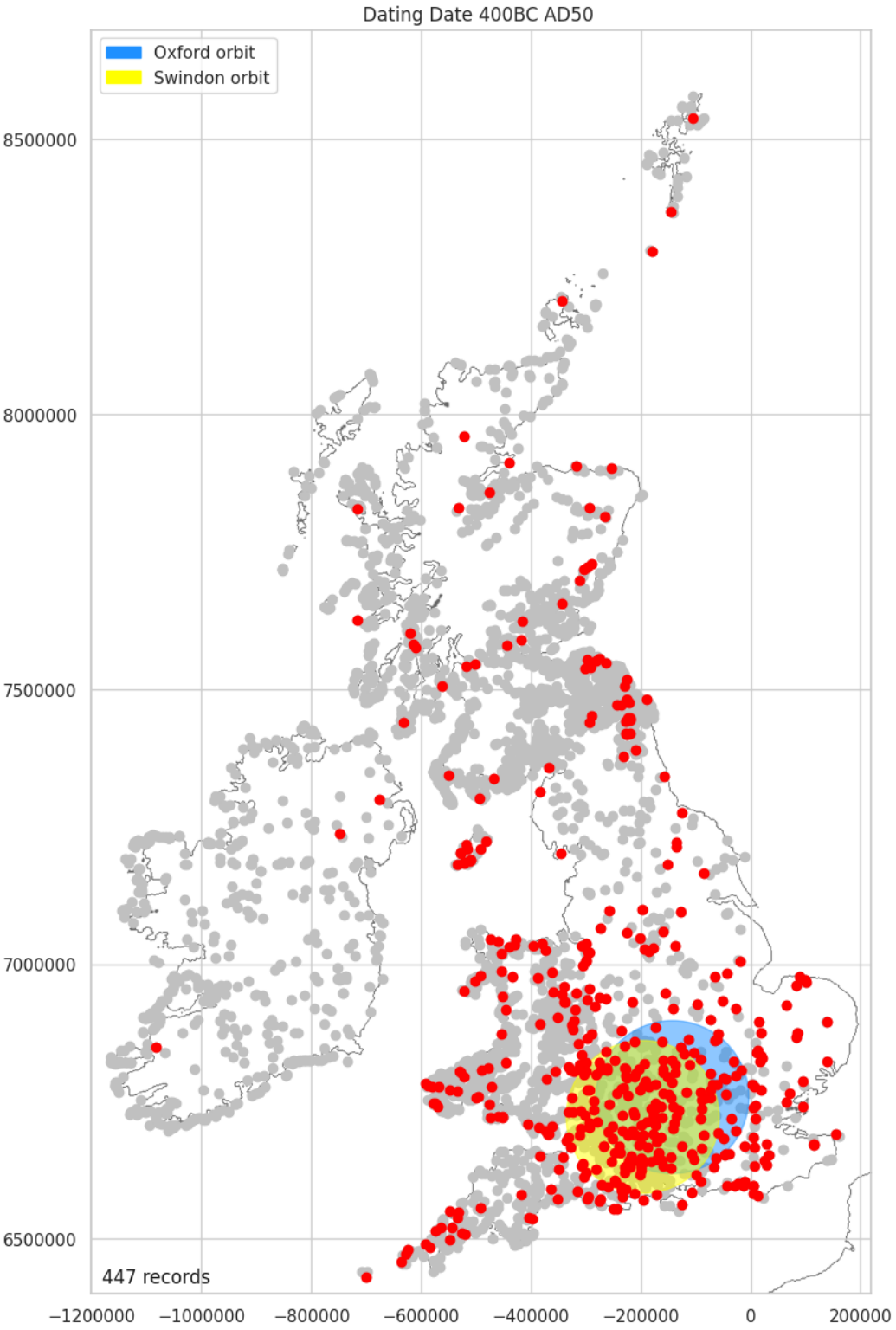


400BC - AD50 Mapped Plus Oxford and Swindon Orbits

The cluster focus is again a result of intensive recording within the orbits of Oxford Universit and Historic England. Within this area it can be said that for forts with dates, hillforts with a 400 BC to AD 50 date are the most common.

In [ ]:
```
date_400bc_50ad = plot_over_grey(location_date_data, 'Dating_Date_400BC_AD50', 'Ye
```

```
Saving figure hillforts_primer_part03-018.png
```




```
10.78%
```

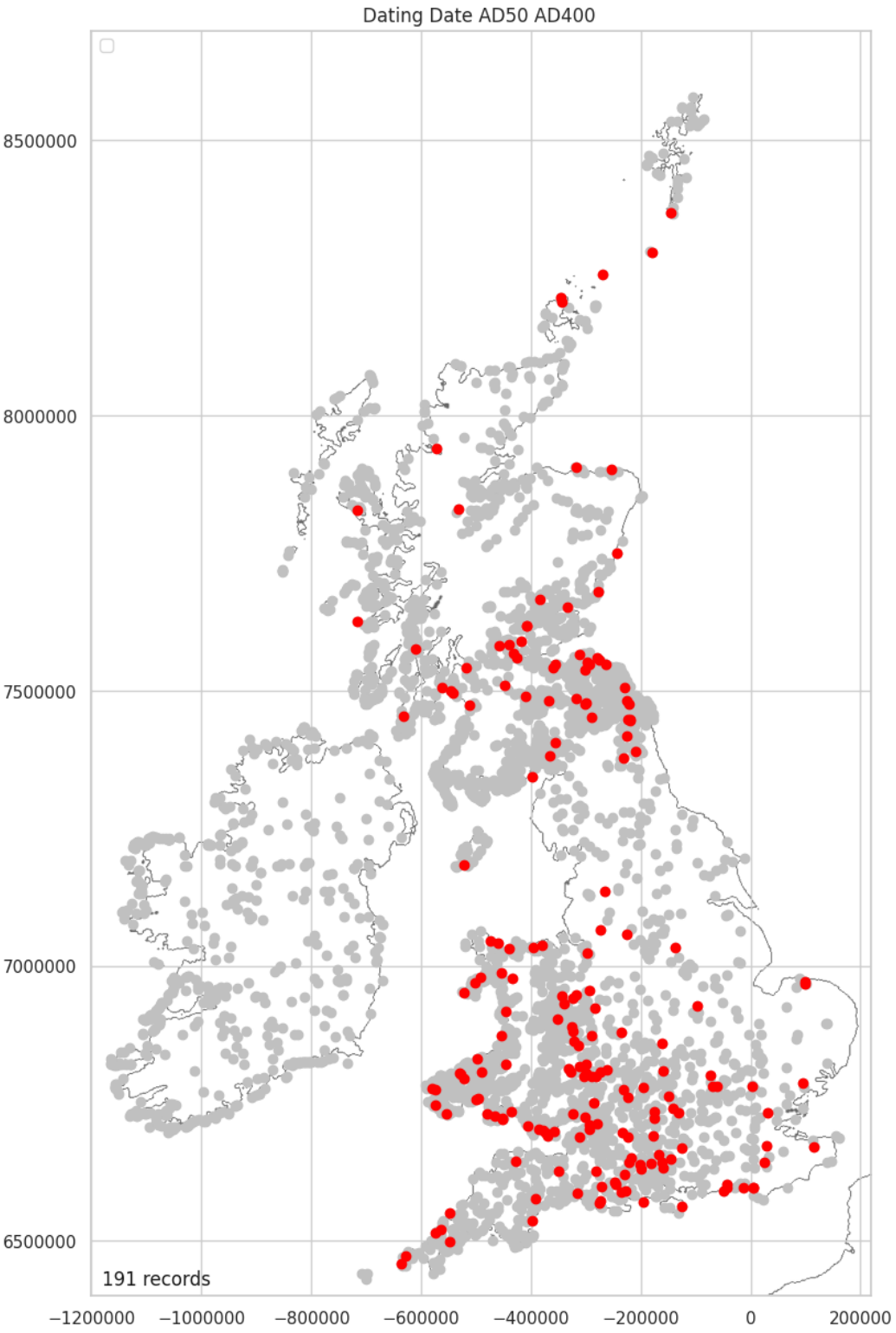
AD50 - AD400 Mapped

The distribution of forts in the AD 50 to AD 400 range will also be bieased decause of the recording focus mentioned above. It is noteable that there are no records recorded for this

period in Ireland. Because of the bias in the dataing records mentioned in 400BC - AD50 Mapped Plus Oxford and Swindon Orbits, a distribution plot for this period has not been produced.

In [ ]:
```
dt_50_400 = plot_over_grey(location_date_data, 'Dating_Date_AD50_AD400', 'Yes')
```

```
Saving figure hillforts_primer_part03-019.png
```



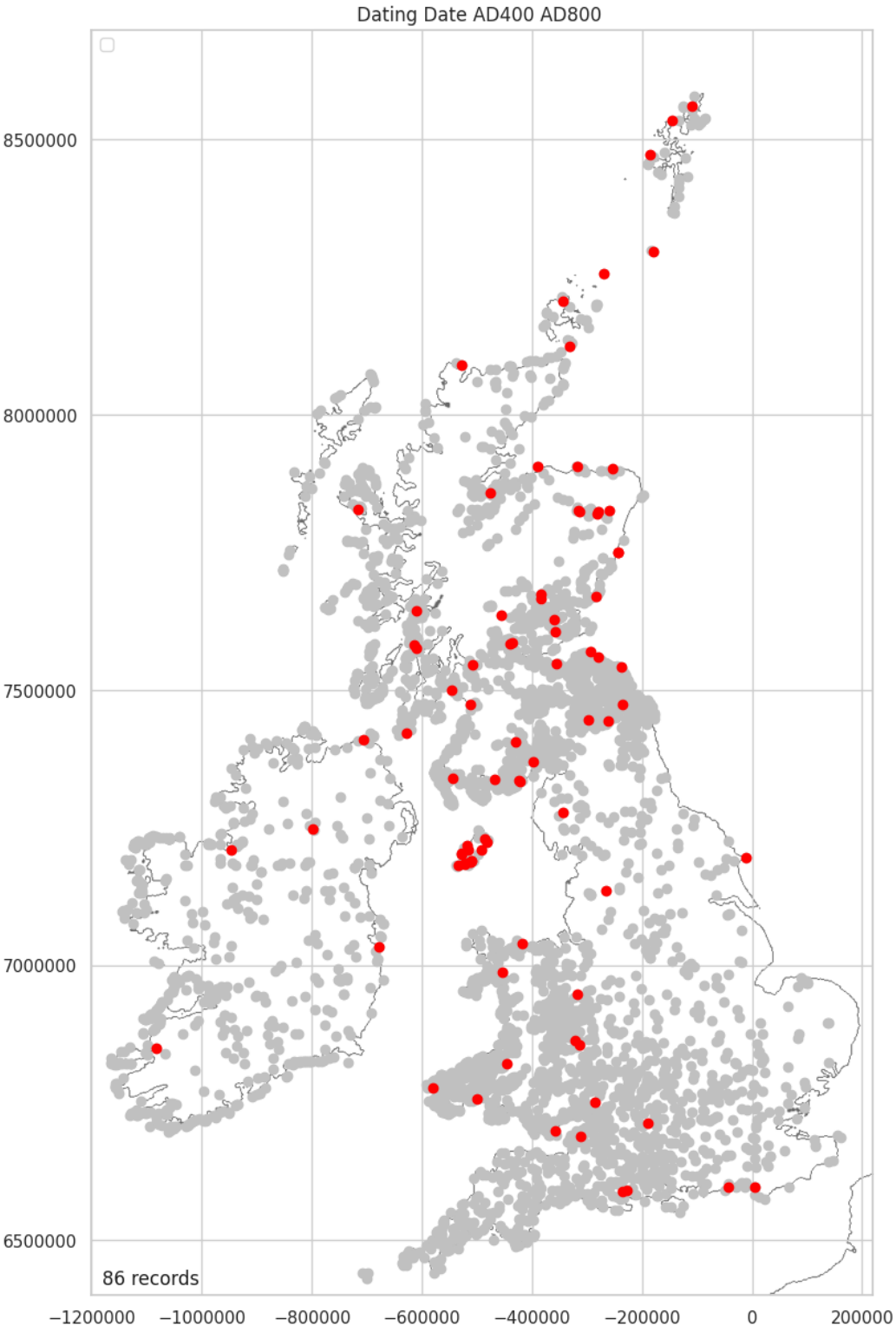

```
4.61%
```

AD400 - AD800 Mapped

A small number of hillforts have dates in the AD 400 to AD 800 rage. Interestingly most of these are outside the bias seen in the dating focus mentioned in 400BC - AD50 Mapped Plus Oxford and Swindon Orbits. This may be an observation that is meaningful for the south of England.

In [ ]:
```
dt_400_800 = plot_over_grey(location_date_data, 'Dating_Date_AD400_AD800', 'Yes')
```

```
Saving figure hillforts_primer_part03-020.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

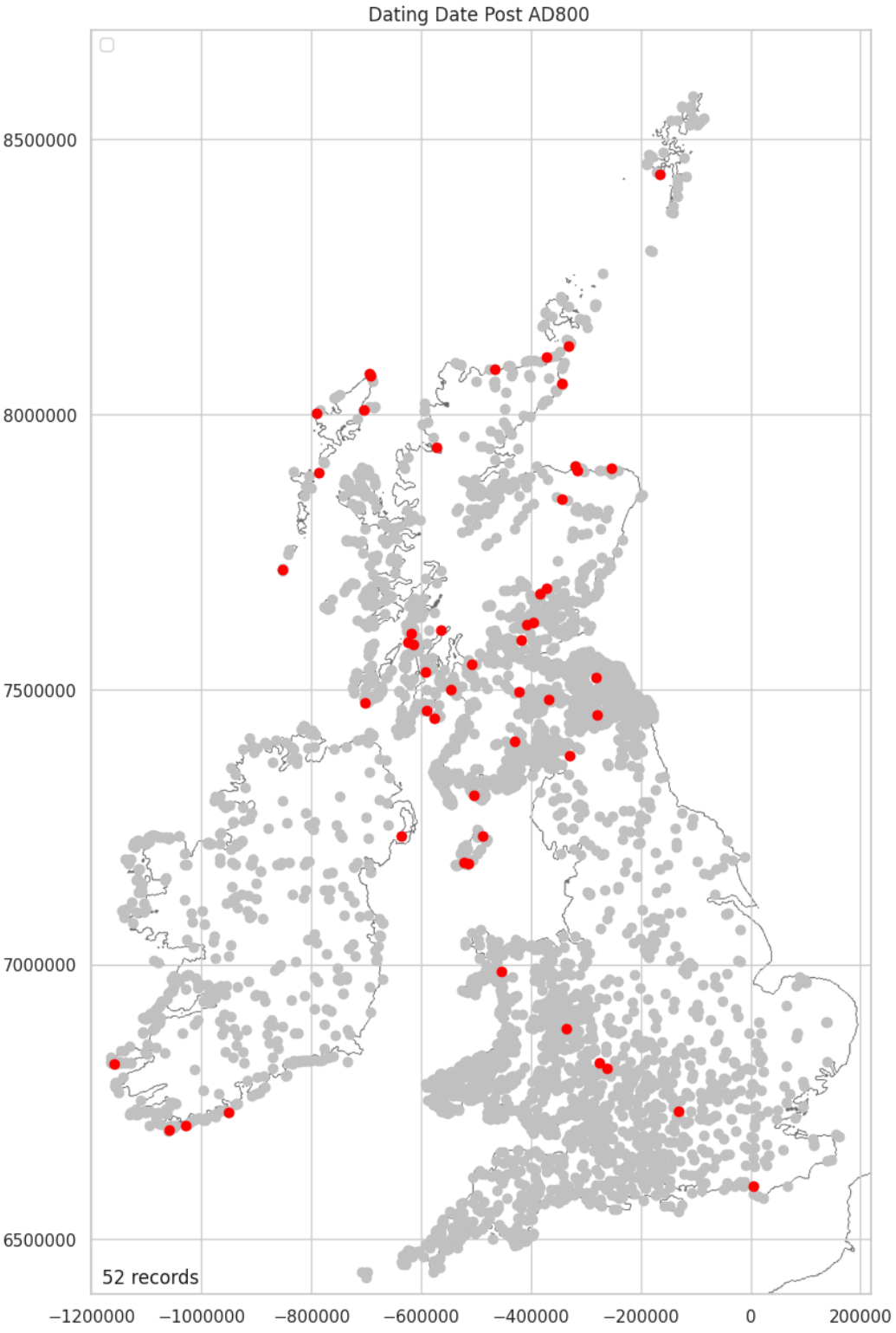
2.07%

Post AD800 Mapped

A small number of hillforts have dates post AD 800. Again, most are outside the bias seen in the dating focus mentioned in 400BC - AD50 Mapped Plus Oxford and Swindon Orbits. This may be an observation that is meaningful for the south of England.

In [ ]:
```
dt_800 = plot_over_grey(location_date_data, 'Dating_Date_Post_AD800', 'Yes')
```

```
Saving figure hillforts_primer_part03-021.png
```



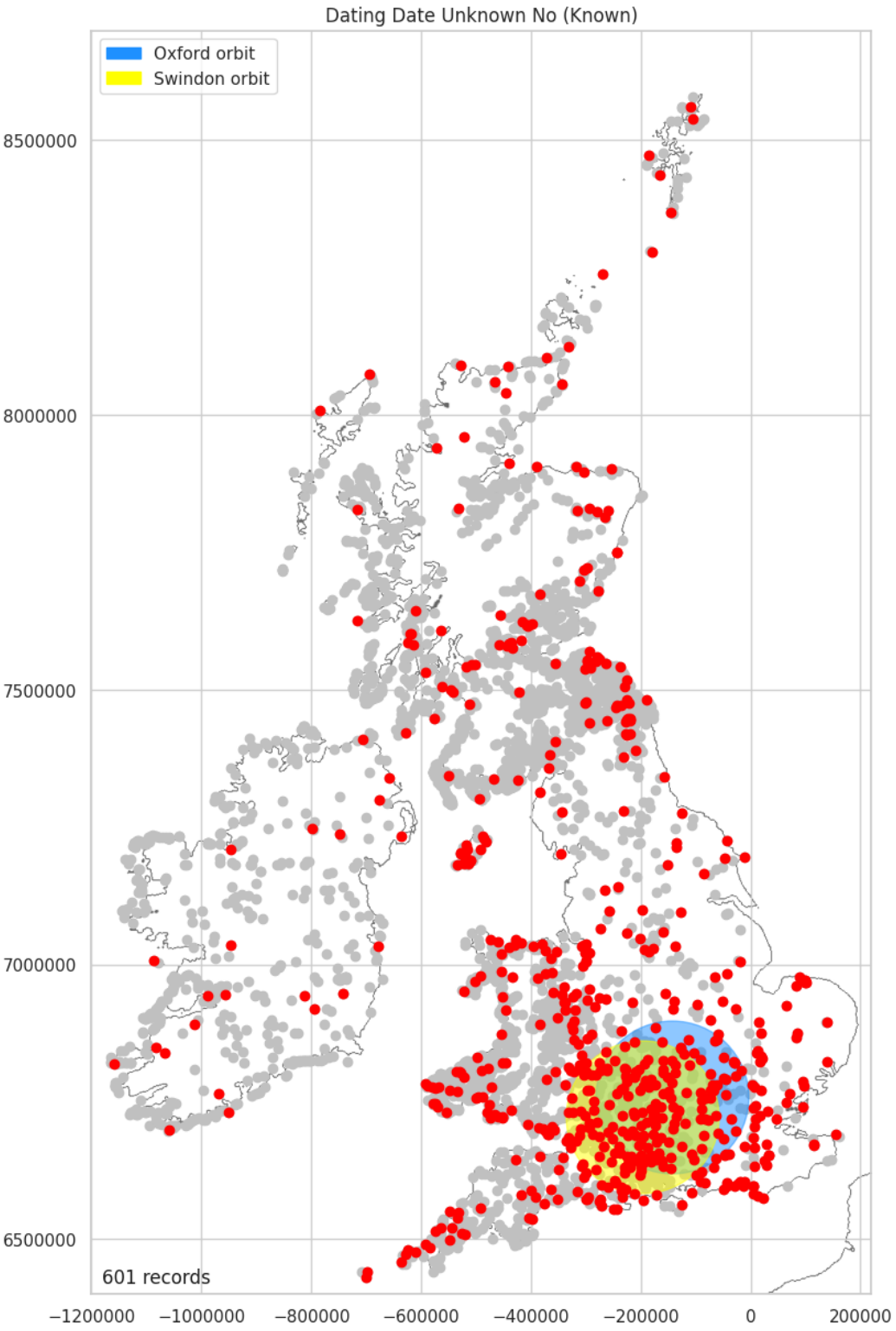
1.25%

Date Known Mapped

Only 14.5% of hillofrts have dating information and the majority of these are in the south of England.

In [ ]:
```
dt_known = plot_over_grey(location_date_data, 'Dating_Date_Unknown', 'No', 'No (Kn
```

```
Saving figure hillforts_primer_part03-022.png
```



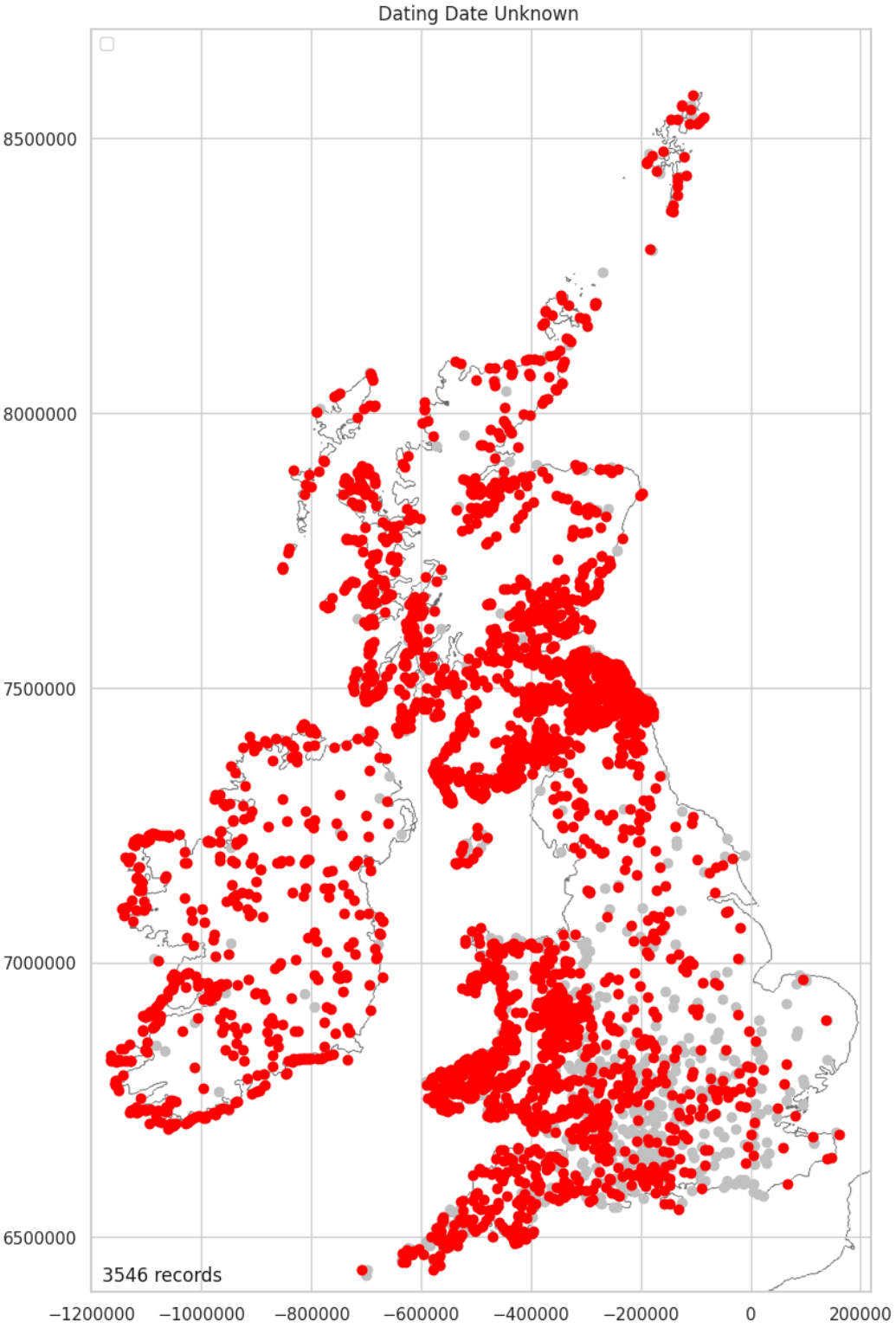

```
14.49%
```

Date Unknown Mapped

Most (85.5%) of hillforts have no dating evidence.

In [ ]:
```
dt_unknown = plot_over_grey(location_date_data, 'Dating_Date_Unknown', 'Yes')
```

```
Saving figure hillforts_primer_part03-023.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

```
85.51%
```

Dating Encodeable Data - Resolve Null Values

There are two features containing null values

In [ ]:
```
dating_encodeable_data[['Dating_Date_Reliability','Related_Dating_Evidence']].info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 2 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Dating_Date_Reliability  4134 non-null   object
 1   Related_Dating_Evidence  805 non-null    object
dtypes: object(2)
memory usage: 64.9+ KB
```

Test for 'NA'.

In [ ]:
```
test_cat_list_for_NA(dating_encodeable_data, ['Dating_Date_Reliability','Related_Da
```

```
Dating_Date_Reliability 0
Related_Dating_Evidence 0
```

Update null values to 'NA'.

In [ ]:
```
dating_encodeable_data = update_cat_list_for_NA(dating_encodeable_data, ['Dating_Da
```

In [ ]:
```
dating_encodeable_data_review = test_numeric(dating_encodeable_data[['Dating_Date_
dating_encodeable_data_review
```

Out[ ]:

	Feature	Entries	Numeric	Non-Numeric	Null
0	Dating_Date_Reliability	4147	0	4147	0
1	Related_Dating_Evidence	4147	0	4147	0

Date Reliability

Data reliability contains five values.

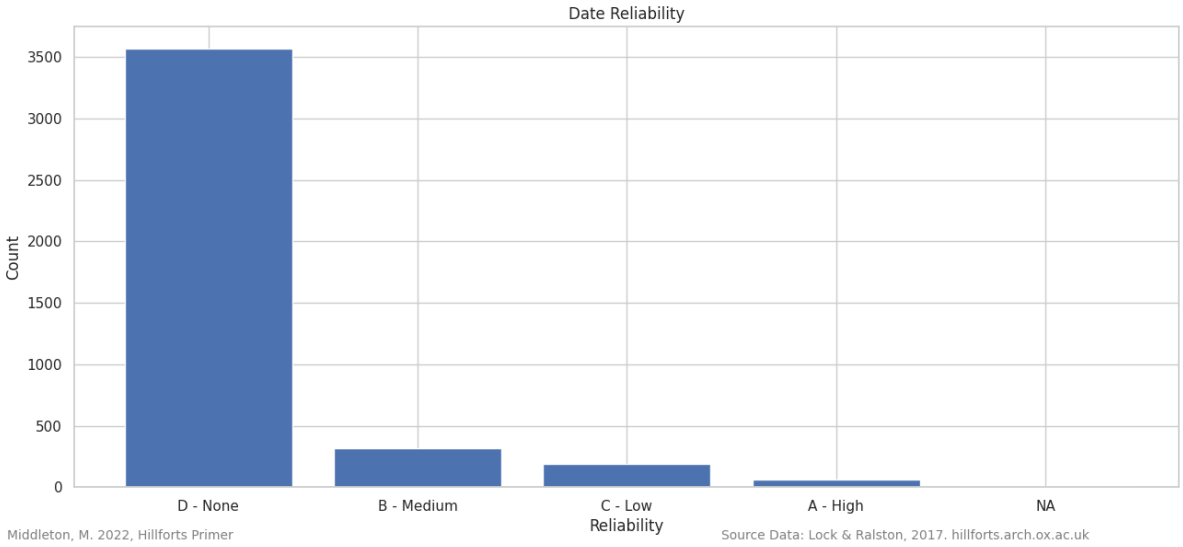
In [ ]:
```
pd.unique(dating_encodeable_data['Dating_Date_Reliability'])
```

Out[ ]:
```
array(['C - Low', 'D - None', 'B - Medium', 'A - High', 'NA'],
      dtype=object)
```

The majority of hillforts have no dataing reliability recorded as most hillforts do not have dating evidence. Of those that do have dating evidence, only 62 have dates that are classified as highly reliable.

In [ ]:
```
plot_bar_chart_value_counts(dating_encodeable_data['Dating_Date_Reliability'], 'Re
```

```
Saving figure hillforts_primer_part03-024.png
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



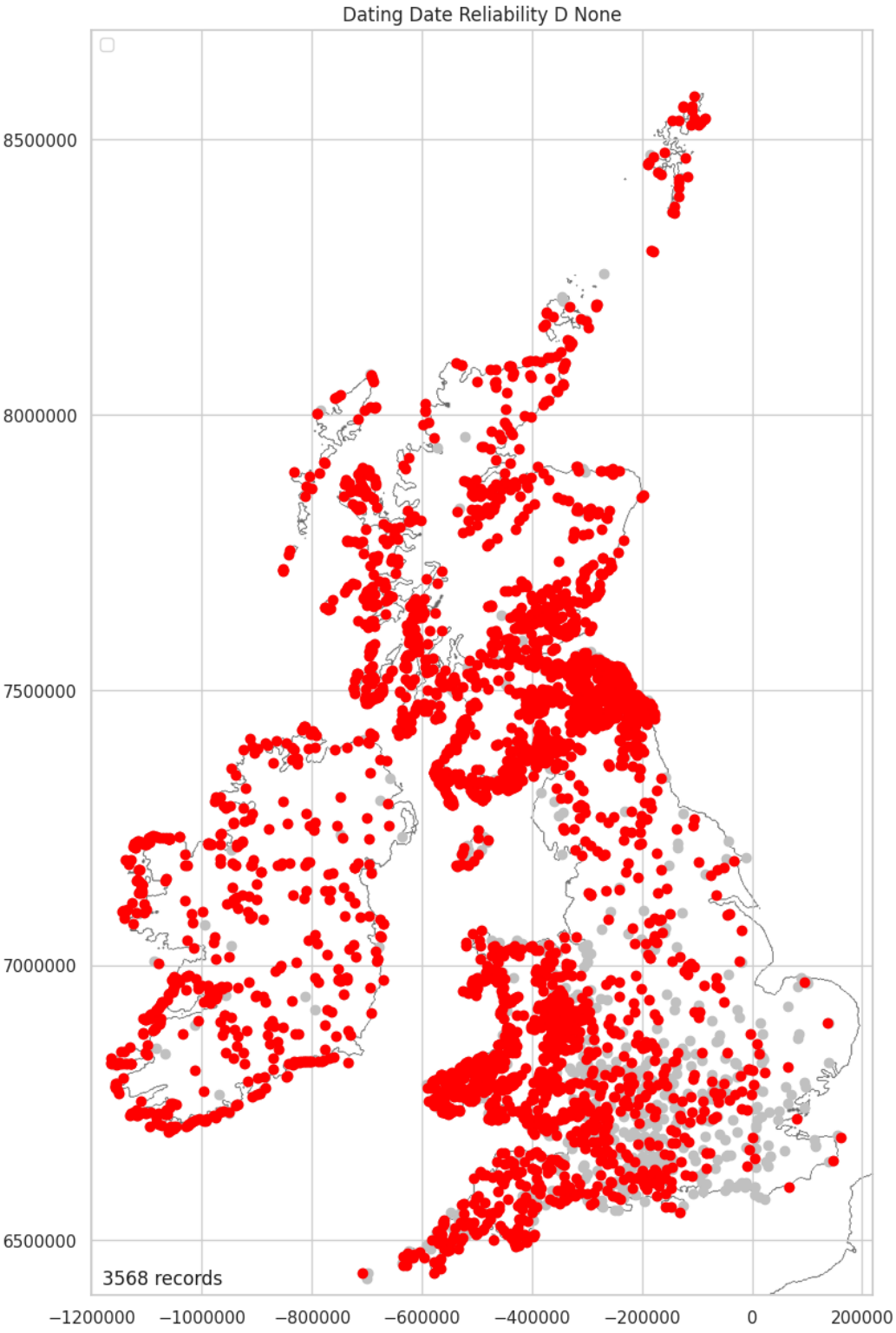
Date Reliability Mapped (D - None)

Most hillforts have no reliable dating evidence.

In [ ]:
```
location_dating_encodeable_data = pd.merge(location_numeric_data_short, dating_enc
```

In [ ]:
```
dating_reliability_none = location_dating_encodeable_data[location_dating_encodeabl
plot_over_grey_numeric(dating_reliability_none, 'Dating_Date_Reliability', 'Dating_
```

```
Saving figure hillforts_primer_part03-025.png
```



Middleton, M. 2022, Hillforts Primer

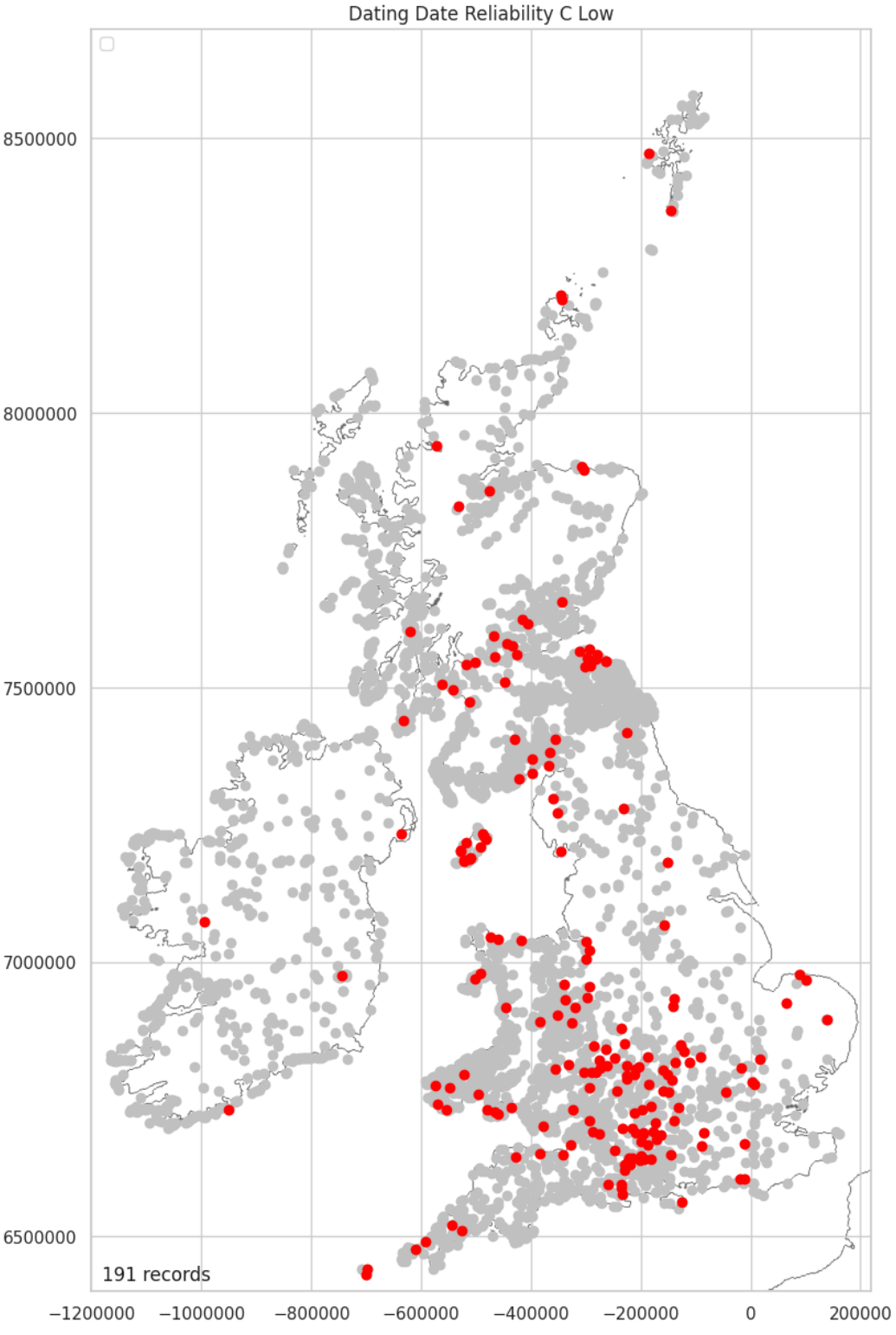
Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Date Reliability Mapped (C - Low)

Of the hillforts that have dataing evidence, only a relativly few have dating that is classified as low reliability. It should be noted that the distibutions based on dating reliability suffer from the same recording bias discussed in in 400BC - AD50 Mapped Plus Oxford and Swindon Orbits.

In [ ]:
```
dating_reliability_low = location_dating_encodeable_data[location_dating_encodeable
plot_over_grey_numeric(dating_reliability_low, 'Dating_Date_Reliability', 'Dating_
```

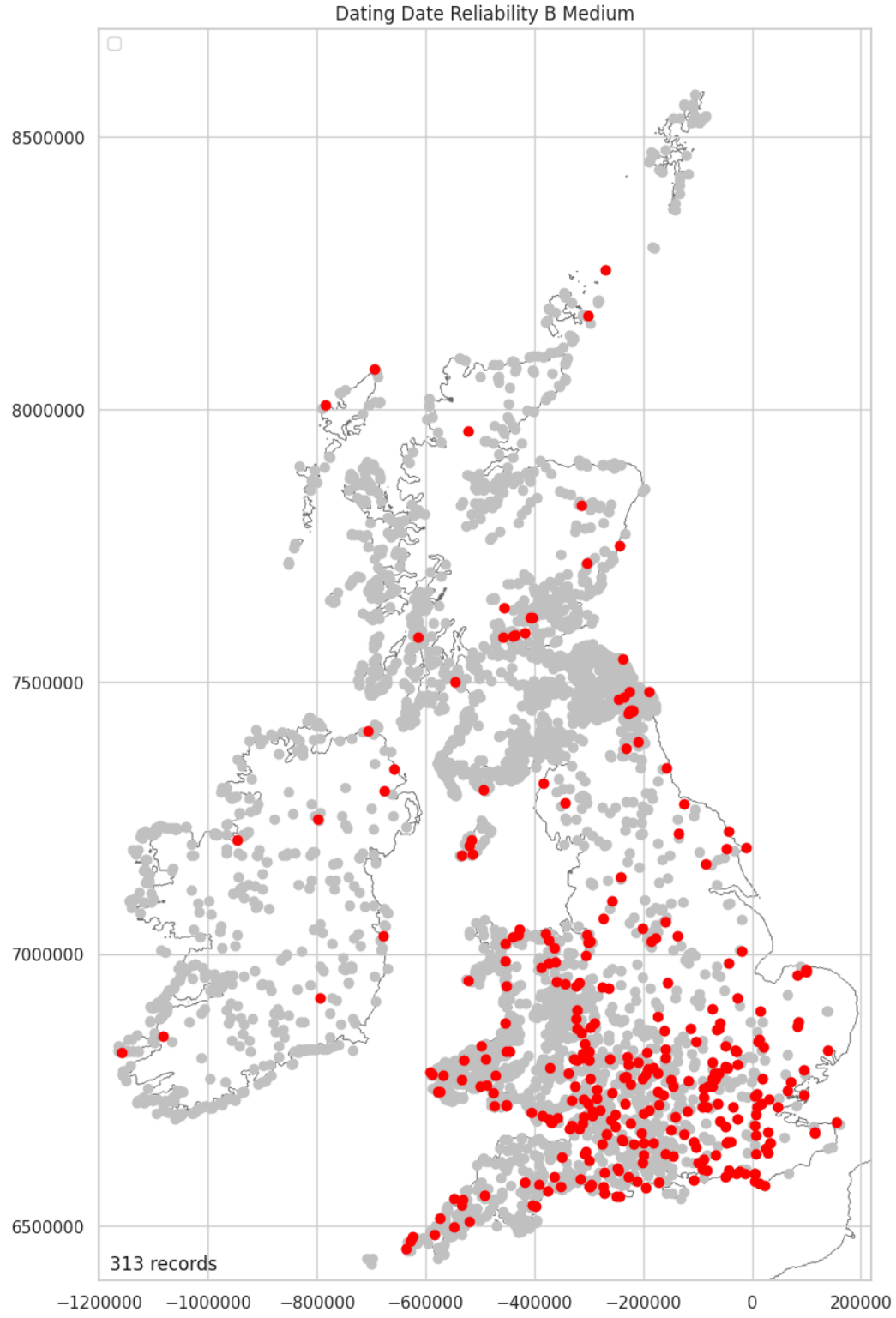
```
Saving figure hillforts_primer_part03-026.png
```



Date Reliability Mapped (B - Medium)

In [ ]:
```
dating_reliability_med = location_dating_encodeable_data[location_dating_encodeable
plot_over_grey_numeric(dating_reliability_med, 'Dating_Date_Reliability', 'Dating_
```

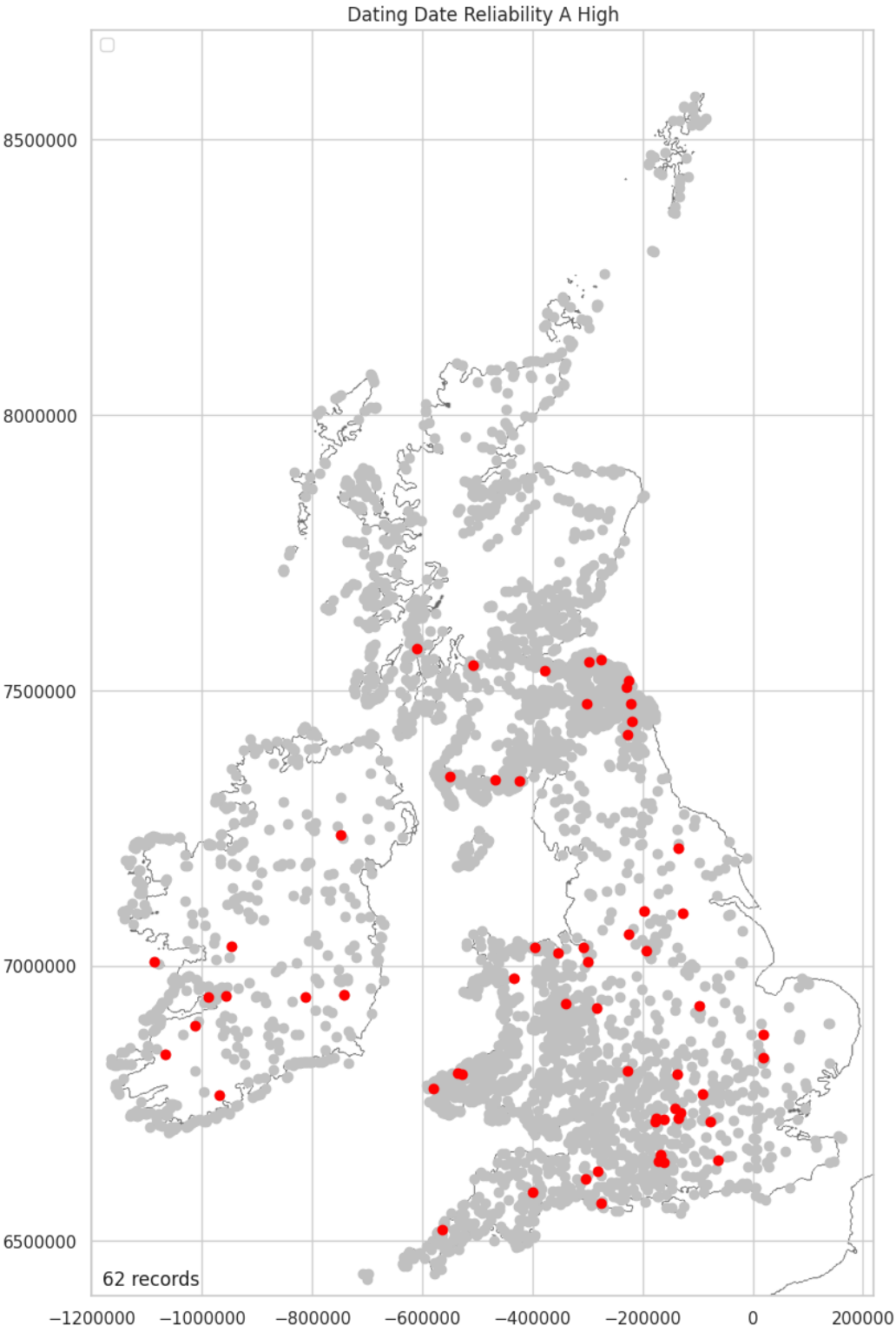
```
Saving figure hillforts_primer_part03-027.png
```



Date Reliability Mapped (A - High)

In [ ]:
```
dating_reliability_high = location_dating_encodeable_data[location_dating_encodeabl
plot_over_grey_numeric(dating_reliability_high, 'Dating_Date_Reliability', 'Dating_
```

```
Saving figure hillforts_primer_part03-028.png
```

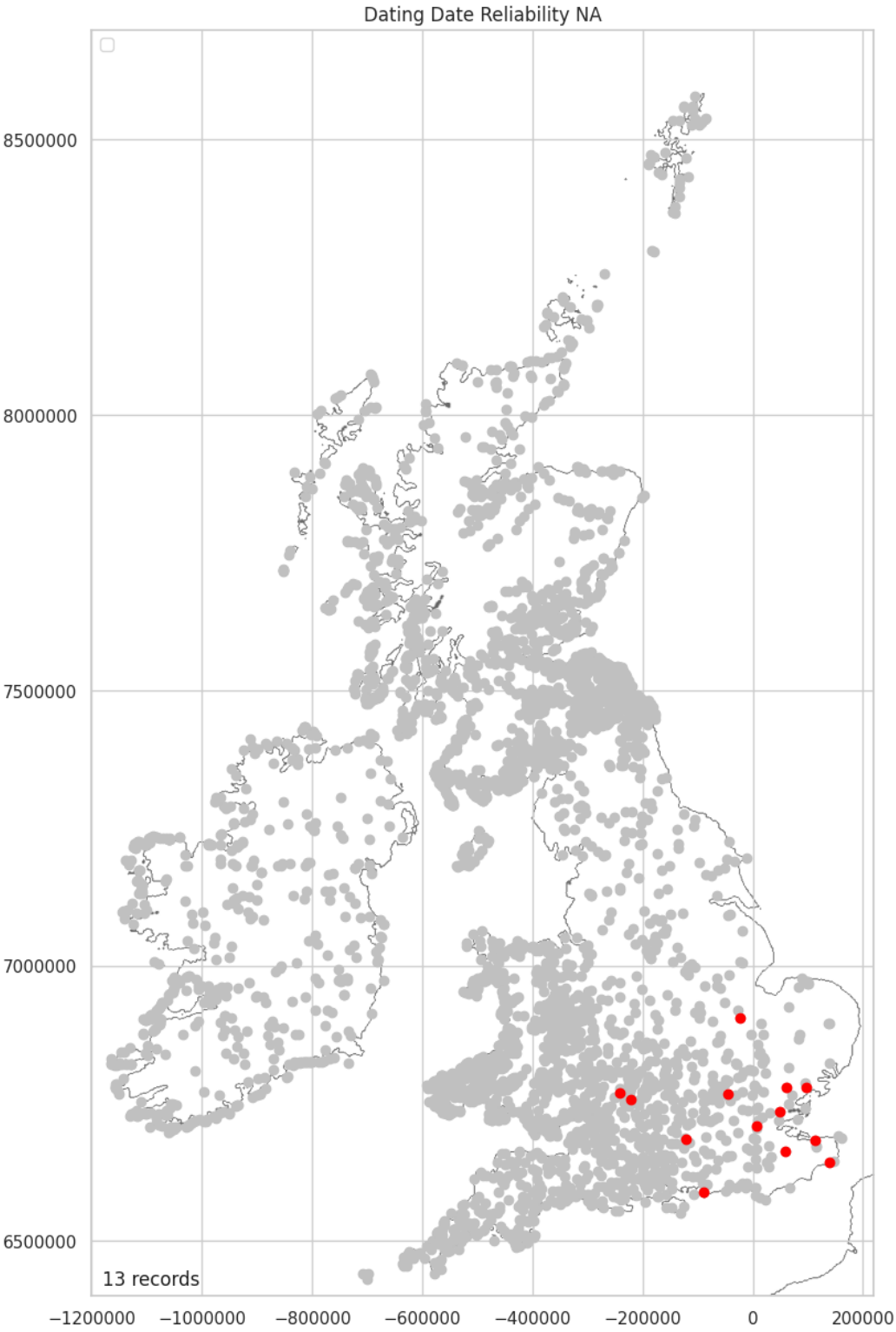


Date Reliability Mapped (NA)

Thirteen records have no information recorded for date reliability.

In [ ]:
```
dating_reliability_na = location_dating_encodeable_data[location_dating_encodeable_
plot_over_grey_numeric(dating_reliability_na, 'Dating_Date_Reliability', 'Dating_Da
```

```
Saving figure hillforts_primer_part03-029.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

Simplify Date Reliablity

When this data is encoded the current values will be added as feature/ column headings. The current format of the data, with spaces and hypen, could lead to problems so it is simplified.

In [ ]:
```
dating_encodeable_data['Dating_Date_Reliability'] = np.where(dating_encodeable_data
pd.unique(dating_encodeable_data['Dating_Date_Reliability'])
```

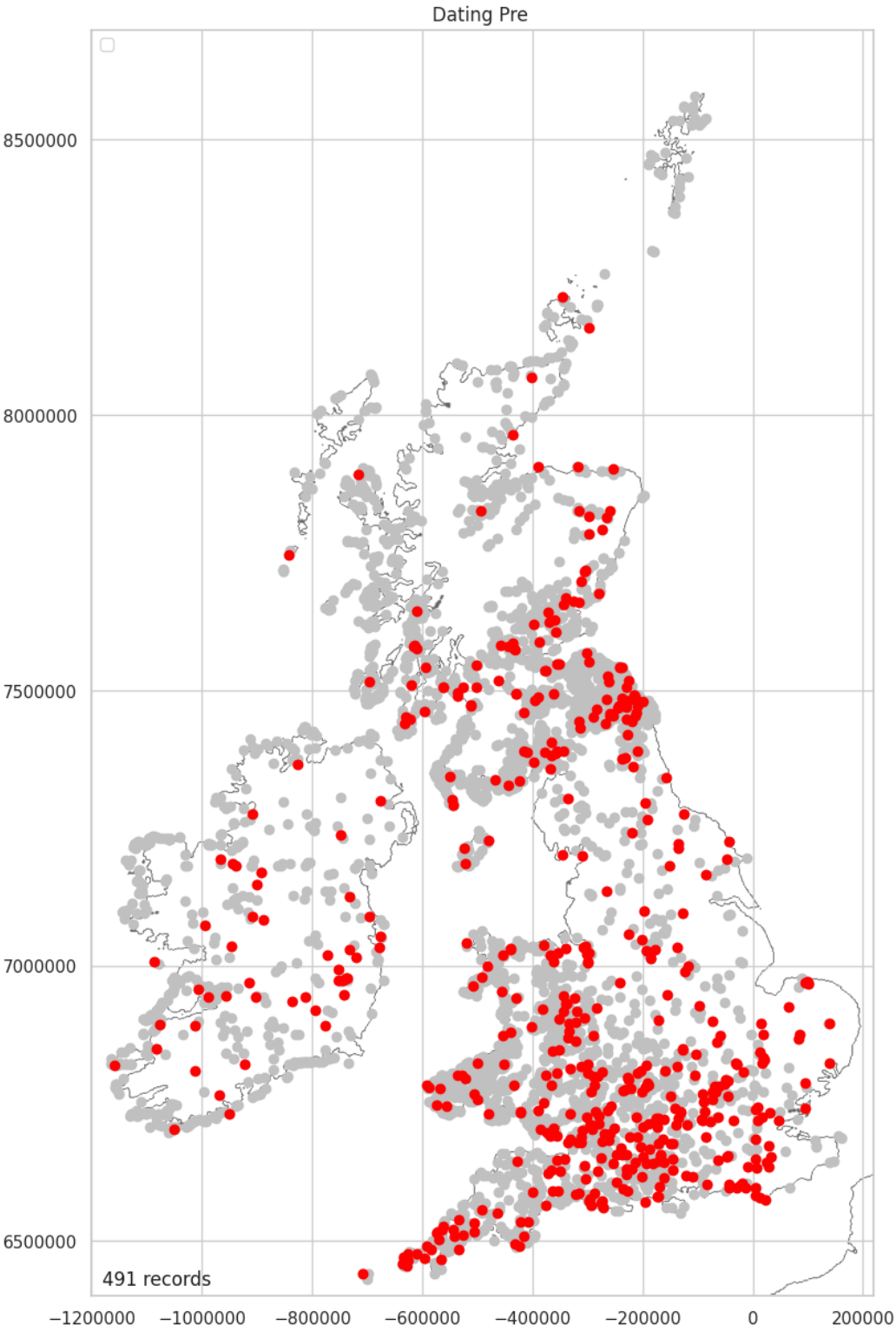
Out[ ]:
```
array(['C', 'D', 'B', 'A', 'NA'], dtype=object)
```

Dating Pre

This feature records if there is activity on site prior to the construction of the hillfort.

In [ ]:
```
dating_pre_stats = plot_over_grey(location_dating_encodeable_data, 'Dating_Pre', '
```

```
Saving figure hillforts_primer_part03-030.png
```



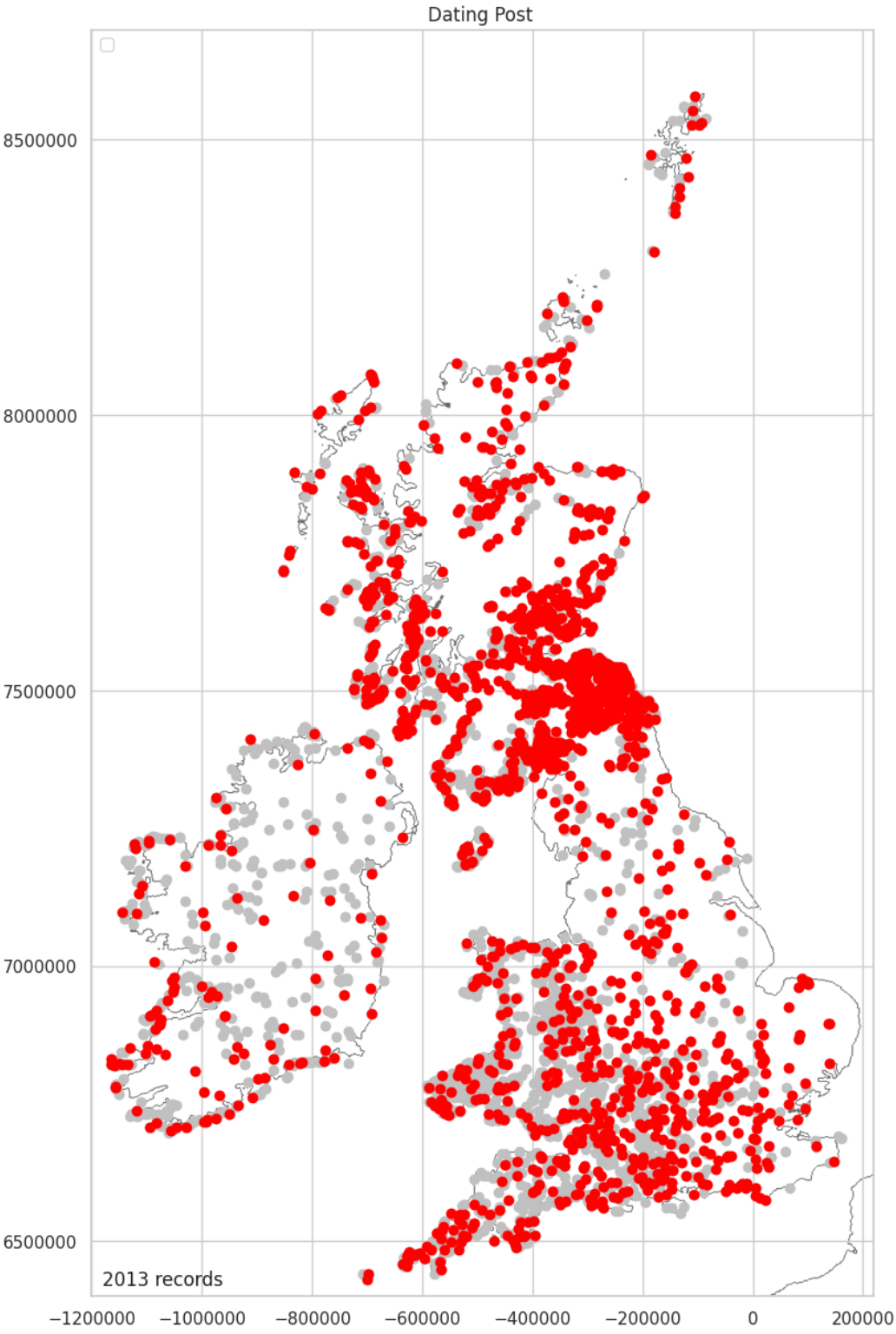
11.84%

Dating Post

This feature records if there is activity on site post the abandon of the hillfort.

In [ ]:
```
dating_post_stats = plot_over_grey(location_dating_encodeable_data, 'Dating_Post',
```

```
Saving figure hillforts_primer_part03-031.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

48.54%

Dating by Region

Only the south has sufficient data to produce a meningful chart. The southern data shows peak between 400 BC and AD 50 with high concentrations of datable material on these sites running from 800 BC to AD 400. The dataing in this reagion is peodomenently derived from artifactul analysis. See: Map Related Dating Artefactual.

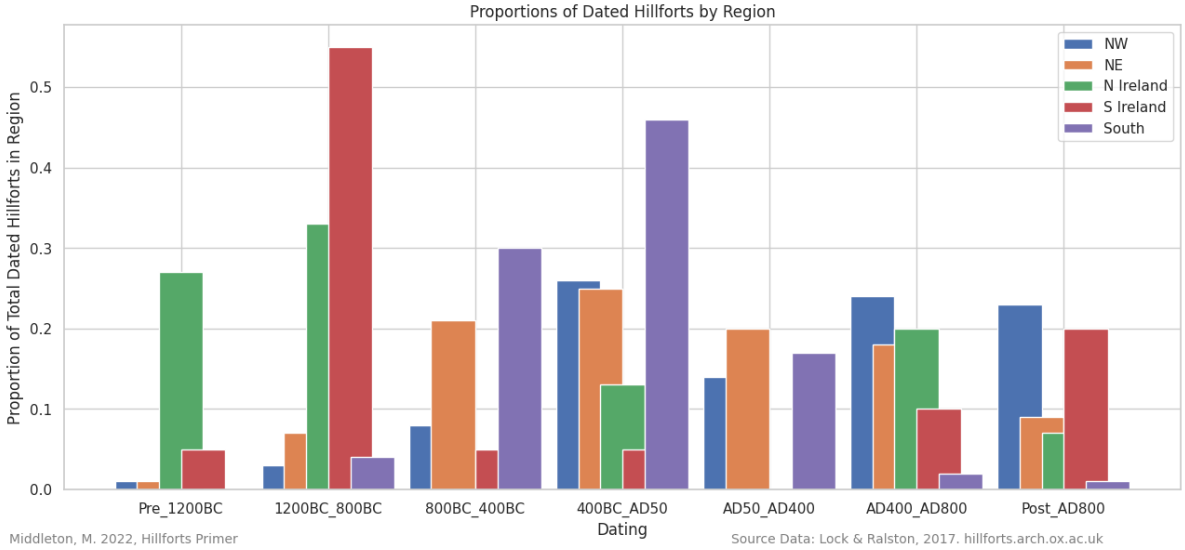
The Northeast and Northwest show a similar distribution of dates but differ in that they show signs of continured use beyond AD 800. Caution is needed in that this distribution of dates is based on a relativly small dataset.

The Irish data is quite differnt. There is a large peak from 1200 BC and 800 BC with the North of Ireland having a similarly large peak for dates pre 1200BC. There are almost no dates in the North of Ireland between 800 BC and 400 BC and only a small peak in South Ireland. There are very few dates in North or South Ireland between AD 50and AD 400 and then increased activity, in terms of dating, from AD 400 on to post AD 800. There are very few dates for hillforts in Ierland. It is highly likely the distridution of dates just discussed could change readically as more dates become available. Extream caution is needed when interpreting the distirbution and spread of Irish dates.

In [ ]:
```
location_enclosing_data_nw_dates = pd.merge(north_west, date_data, left_index=True,
location_enclosing_data_ne_dates = pd.merge(north_east, date_data, left_index=True,
location_enclosing_data_ireland_n_dates = pd.merge(north_ireland, date_data, left_
location_enclosing_data_ireland_s_dates = pd.merge(south_ireland, date_data, left_
location_enclosing_data_south_dates = pd.merge(south, date_data, left_index=True,
```

In [ ]:
```
plot_dates_by_region(location_enclosing_data_nw_dates,location_enclosing_data_ne_da
```

```
Saving figure hillforts_primer_part03-032.png
<ipython-input-53-fcc09a6f289d>:13: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



Related Dating Evidence

In [ ]:
```
list(pd.unique(dating_encodeable_data['Related_Dating_Evidence']))
```

Out[ ]:
```
['Artefactual',
 'NA',
 'Morphology/Earthwork/Typology',
 'Artefactual; C14; Morphology/Earthwork/Typology',
 'Artefactual; C14',
 'C14',
 'Artefactual; Morphology/Earthwork/Typology',
 'Morphology/Earthwork/Typology; Other',
 'C14; Morphology/Earthwork/Typology',
 'Other',
 'Artefactual; Other',
 'Artefactual; C14; Other',
 'C14; Morphology/Earthwork/Typology; Other',
 'C14; Other',
 'Artefactual; C14; Artefactual; C14',
 'Artefactual; Morphology/Earthwork/Typology; Other']
```

A gazateer of five terms (classes) has been used - see below. These terms unclude 'NA' which was added by this study to replace null values. One hillfort may be associated with multiple dating classes. In the distributions that follow the same locations may occour in multiple dataing plots.

In [ ]:
```
related_dating_terms = ['Artefactual', 'NA',  'Morphology/Earthwork/Typology', 'C1
```

Add new Related Dating Evidence Features

To enable the Related Dating Evidence to be interogated more simply, five new, boolean, 'Yes/No' columns will be added to the encodeable data. Initially, they will be set to 'No' and then updated to 'Yes' if the term is found in the current 'Related_Dating_Evidence' feature. Note that the forward slash '/' will be removed from the column heading as this can cause problems.

In [ ]:
```
dating_encodeable_data_plus = dating_encodeable_data.copy()
additiaonal_related_dating_features = ['Related_Dating_Artefactual', 'Related_Datir
for feature in additiaonal_related_dating_features:
    dating_encodeable_data_plus[feature] = 'No'
dating_encodeable_data_plus[additiaonal_related_dating_features].head()
```

Out[ ]:

	Related_Dating_Artefactual	Related_Dating_NA	Related_Dating_Morph_Earth_Typo	Related_Datir
0	No	No	No	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	No	

Add Related Dating Artefactual

Populate the 'Artifactual' column.

In [ ]:
```
dating_encodeable_data_plus['Related_Dating_Artefactual'].loc[dating_encodeable_da
dating_encodeable_data_plus[['Related_Dating_Evidence','Related_Dating_Artefactual
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_Artefactual
11	Artefactual; C14; Morphology/Earthwork/Typology	Yes
12	Artefactual	Yes
14	Artefactual	Yes
16	Artefactual; C14	Yes
18	Artefactual	Yes

Add Related Dating Morphology/Earthwork/Typology

Populate the 'Morphology/Earthwork/Typology' column.

In [ ]:
```
dating_encodeable_data_plus['Related_Dating_Morph_Earth_Typo'].loc[dating_encodeabl
dating_encodeable_data_plus[['Related_Dating_Evidence','Related_Dating_Morph_Earth_
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_Morph_Earth_Typo
4	Morphology/Earthwork/Typology	Yes
11	Artefactual; C14; Morphology/Earthwork/Typology	Yes
39	Artefactual; Morphology/Earthwork/Typology	Yes
47	Morphology/Earthwork/Typology; Other	Yes
60	C14; Morphology/Earthwork/Typology	Yes

Add Related Dating C14

Populate the 'C14' column.

In [ ]:
```
dating_encodeable_data_plus['Related_Dating_C14'].loc[dating_encodeable_data_plus[
dating_encodeable_data_plus[['Related_Dating_Evidence','Related_Dating_C14']].loc[
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_C14
11	Artefactual; C14; Morphology/Earthwork/Typology	Yes
16	Artefactual; C14	Yes
21	Artefactual; C14	Yes
23	C14	Yes
60	C14; Morphology/Earthwork/Typology	Yes

Add Related Dating Other

Populate the 'Other' column.

In [ ]:
```
dating_encodeable_data_plus['Related_Dating_Other'].loc[dating_encodeable_data_plus
dating_encodeable_data_plus[['Related_Dating_Evidence','Related_Dating_Other']].loc
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_Other
47	Morphology/Earthwork/Typology; Other	Yes
65	Other	Yes
70	Other	Yes
75	Other	Yes
154	Artefactual; Other	Yes

Add Related Dating NA

Populate the 'NA' column.

In [ ]:
```
dating_encodeable_data_plus['Related_Dating_NA'].loc[dating_encodeable_data_plus['
dating_encodeable_data_plus[['Related_Dating_Evidence','Related_Dating_NA']].loc[da
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_NA
1	NA	Yes
2	NA	Yes
5	NA	Yes
8	NA	Yes
9	NA	Yes

Review New Related Dating Evidence Features

Review a sampe of the new columns to confirm the features are as expected.

In [ ]:
```
dating_encodeable_data_plus[['Related_Dating_Evidence']+additiaonal_related_dating_
```

Out[ ]:

	Related_Dating_Evidence	Related_Dating_Artefactual	Related_Dating_NA	Related_Dating_
6	Artefactual	Yes	No	
7	Artefactual	Yes	No	
8	NA	No	Yes	
9	NA	No	Yes	
10	Artefactual	Yes	No	
11	Artefactual; C14; Morphology/Earthwork/Typology	Yes	No	

Drop Related Dating Evidence

The inforamtion in 'Related_Dating_Evidence' has now been migrated to the new features so the original feature can now be deleted.

In [ ]:
```
dating_encodeable_data_plus = dating_encodeable_data_plus.drop(['Related_Dating_Ev
dating_encodeable_data_plus.head()
```

Out[ ]:

	Dating_Date_Pre_1200BC	Dating_Date_1200BC_800BC	Dating_Date_800BC_400BC	Dating_Date_4(
0	No	No	Yes	
1	No	No	No	
2	No	No	No	
3	No	No	No	
4	No	No	Yes	

Map New Related Dating Evidence Features

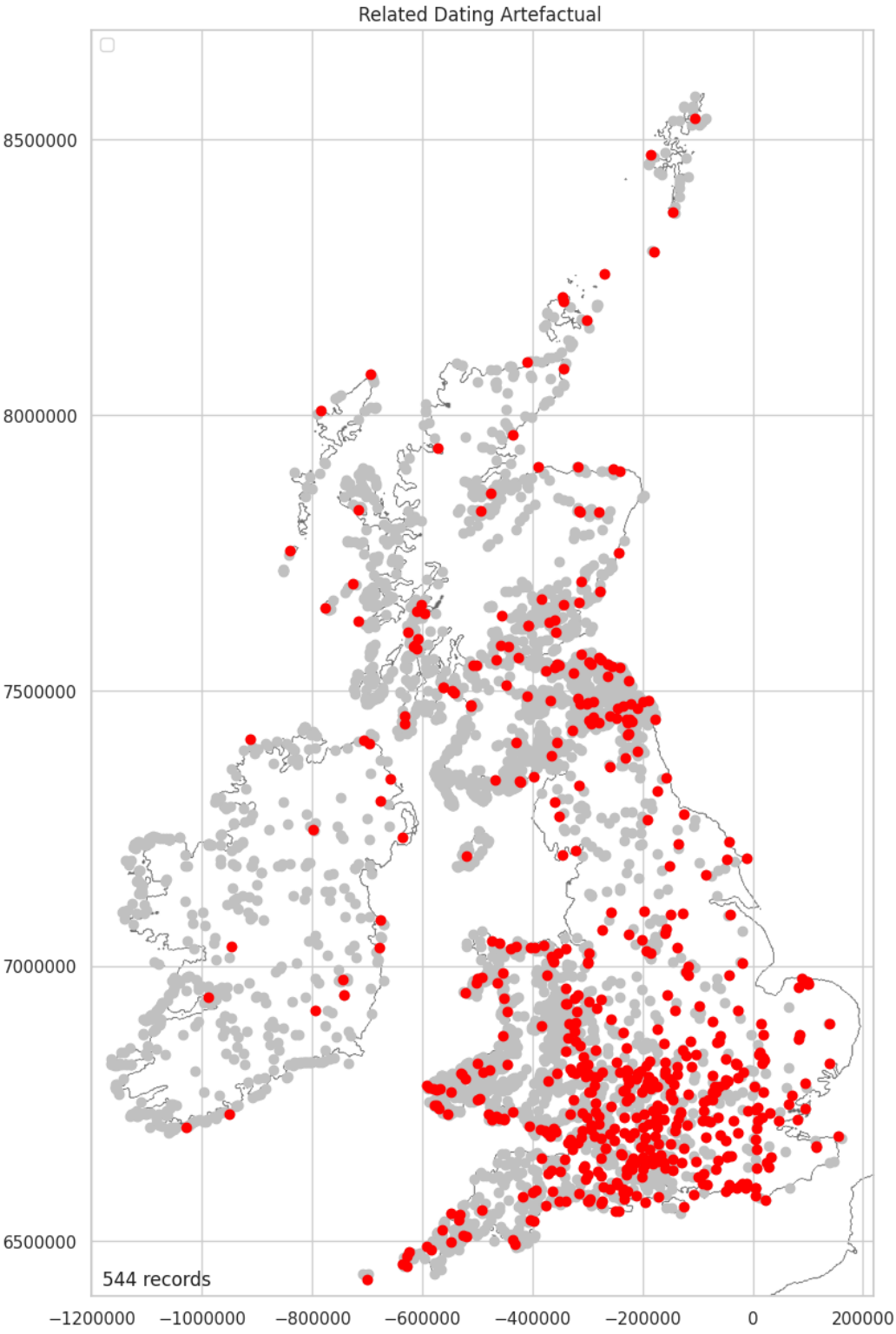
Map Related Dating Artefactual

Although most dated hillforts are dated using artifactual evidence, only 13.12% of hillofrts have been dated in this way.

In [ ]:
```
location_dating_encodeable_plus_data = pd.merge(location_numeric_data_short, datin
```

In [ ]:
```
related_dating_artefactual_stats = plot_over_grey(location_dating_encodeable_plus_
```

```
Saving figure hillforts_primer_part03-033.png
```



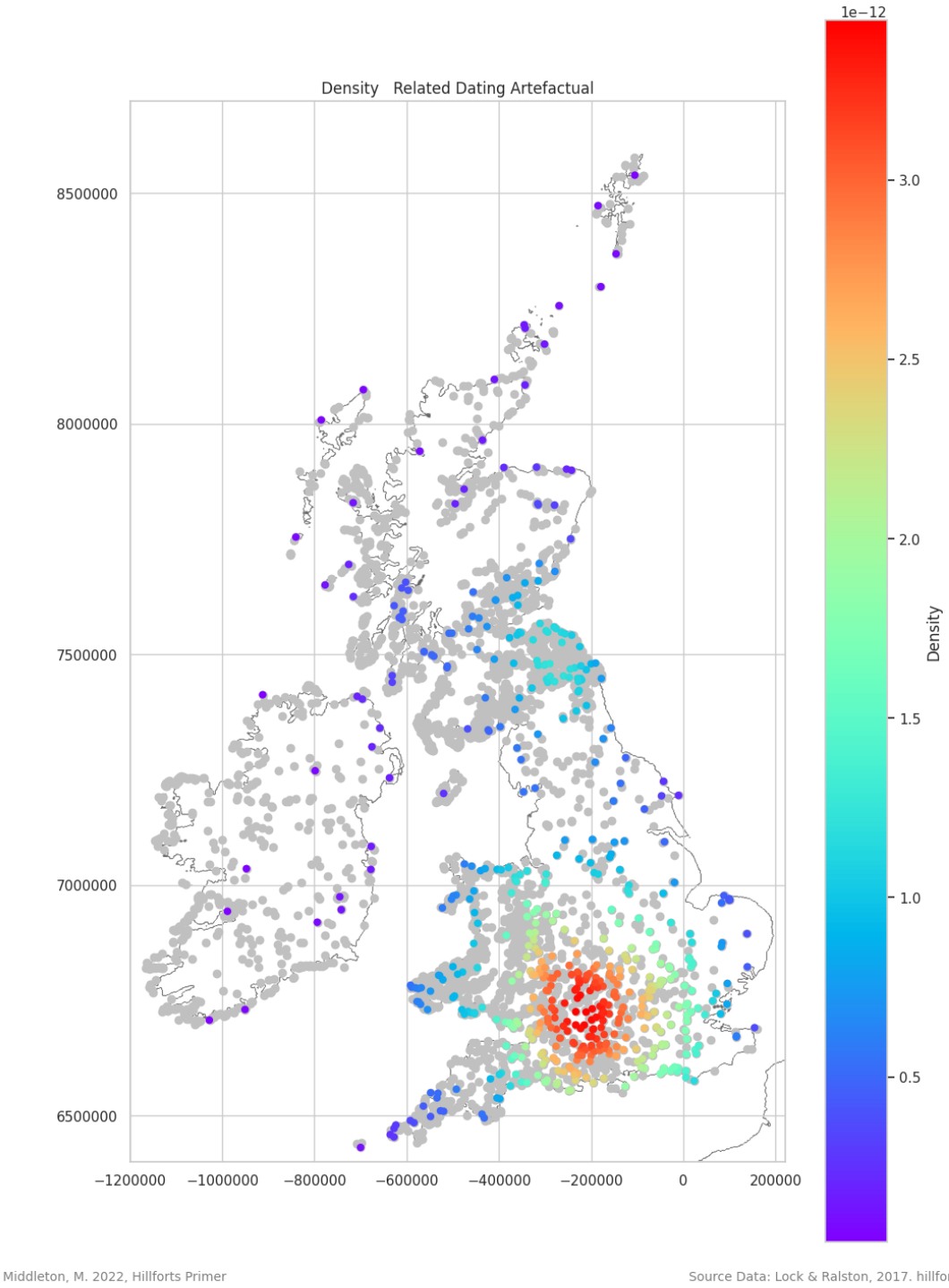

```
13.12%
```

Map Related Dating Artefactual Density

There is a strong concentration of artifactual dates in south-central England.

In [ ]:
```
plot_density_over_grey(related_dating_artefactual_stats, 'Related_Dating_Artefactua
```

```
Saving figure hillforts_primer_part03-034.png
```

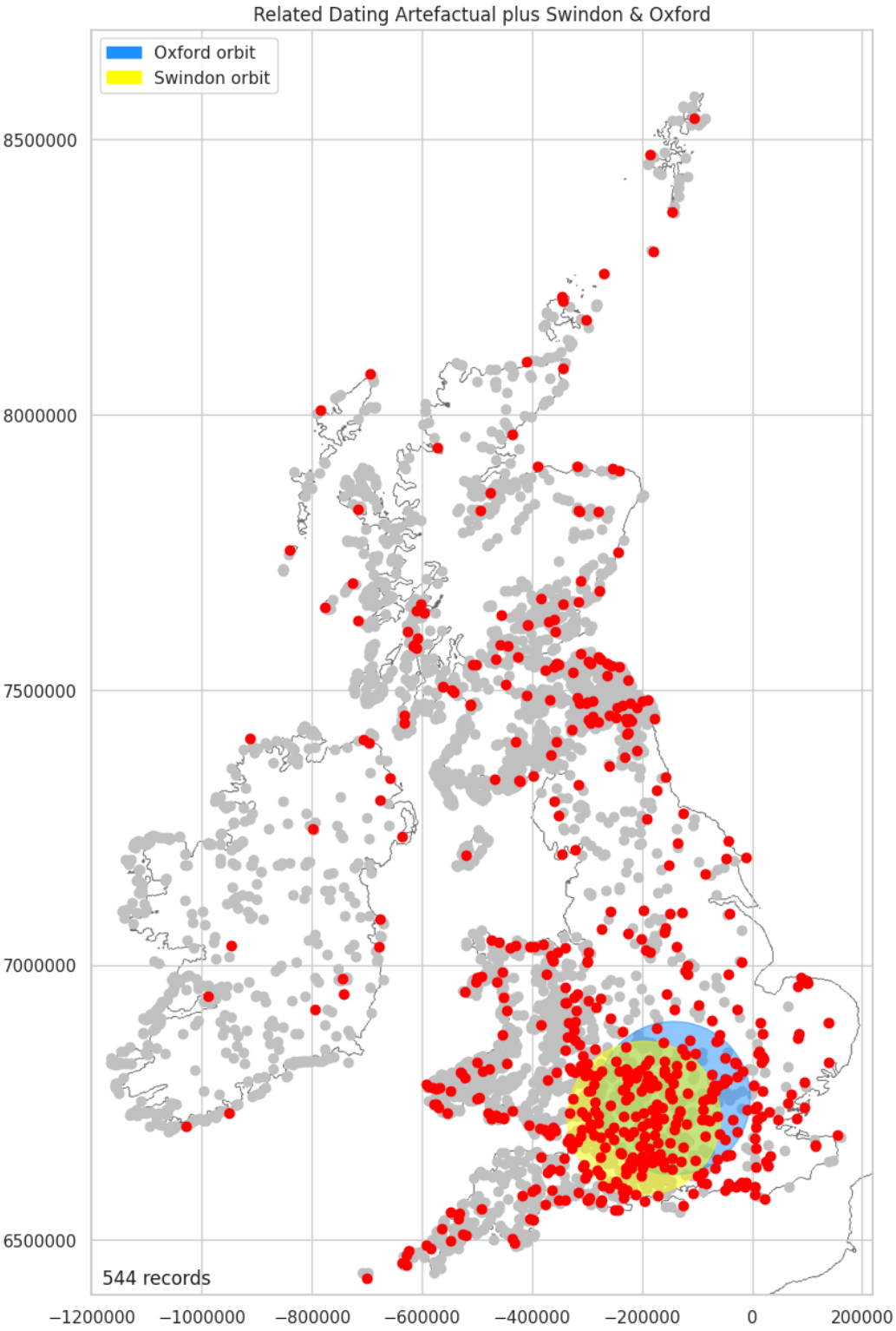


Map Related Dating Artefactual Showing Swindon and Oxford Orbits

When this cluster is plotted against the orbits of Oxford University and the head office of Historic England in Swindon there is a strong correlation between the two suggesting there is a sygnificant recording bias highlighted by this distribution.

In [ ]:
```
date_artif = plot_over_grey(location_dating_encodeable_plus_data, 'Related_Dating_A
```

```
Saving figure hillforts_primer_part03-035.png
```



Middleton, M. 2022, Hillforts Primer

Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk

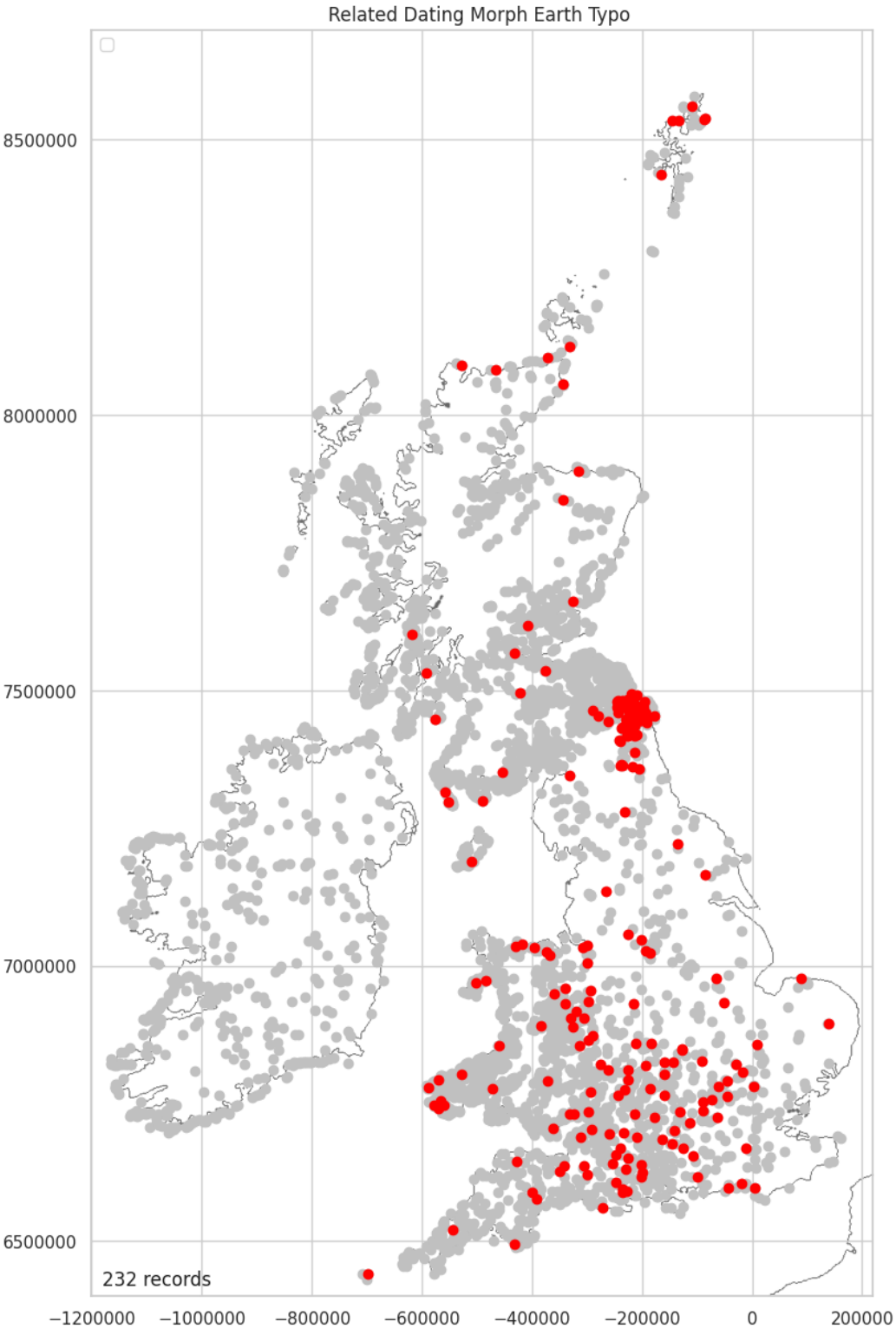
13.12%

Map Related Dating Morphology/Earthwork/Typology

Dating by means of morphology, earthwork and typology has a similar bias toward south central England and toward the northern border of Northumberland. Only 5.59% of hillforts have been dated in this way. Noteably, none are in Irealand.

In [ ]:
```
met_stats = plot_over_grey(location_dating_encodeable_plus_data, 'Related_Dating_M
```

```
Saving figure hillforts_primer_part03-036.png
```

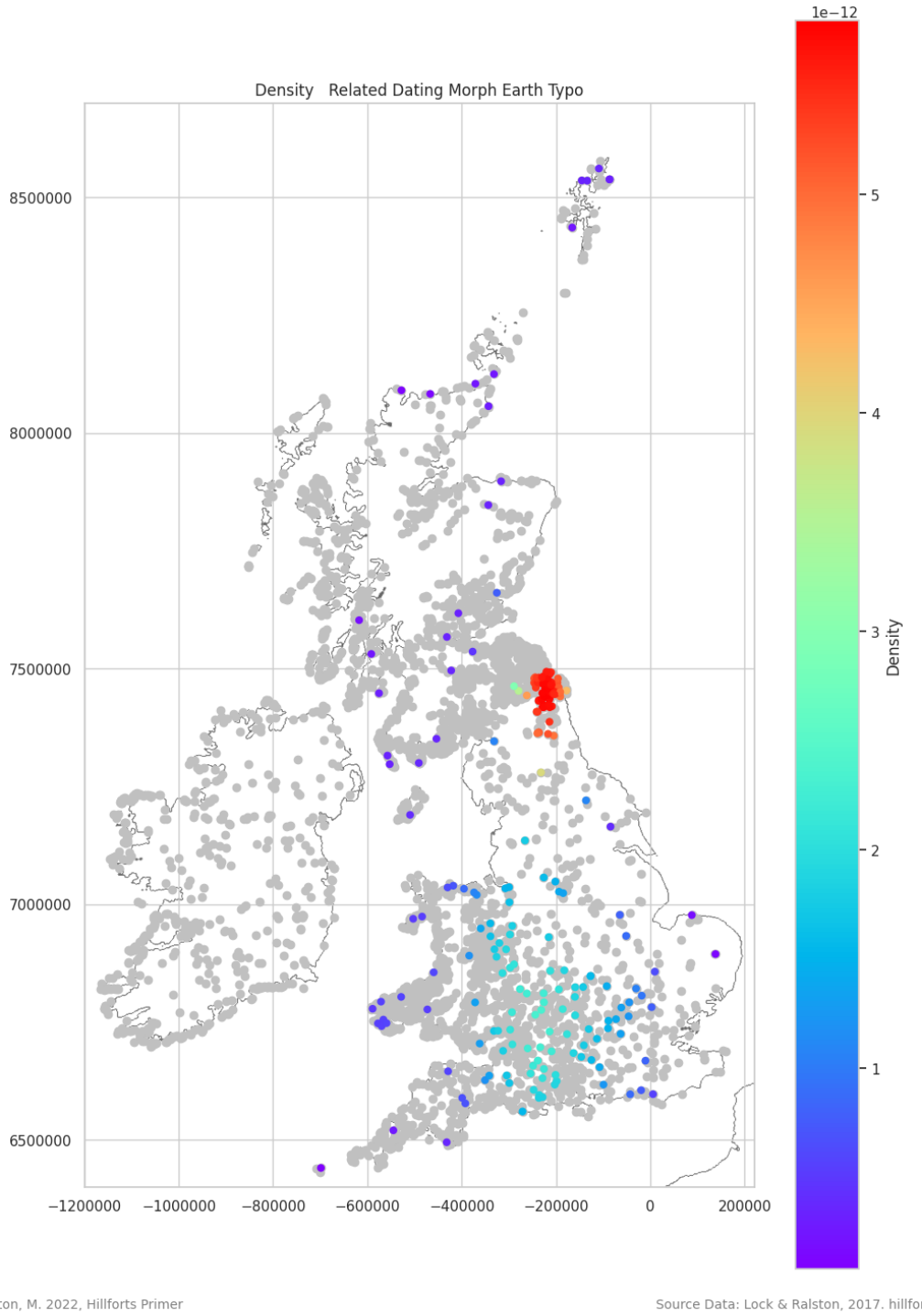



```
5.59%
```

Map Related Dating Morphology/Earthwork/Typology Density

In [ ]:
```
plot_density_over_grey(met_stats, 'Related_Dating_Morph_Earth_Typo')
```

```
Saving figure hillforts_primer_part03-037.png
```

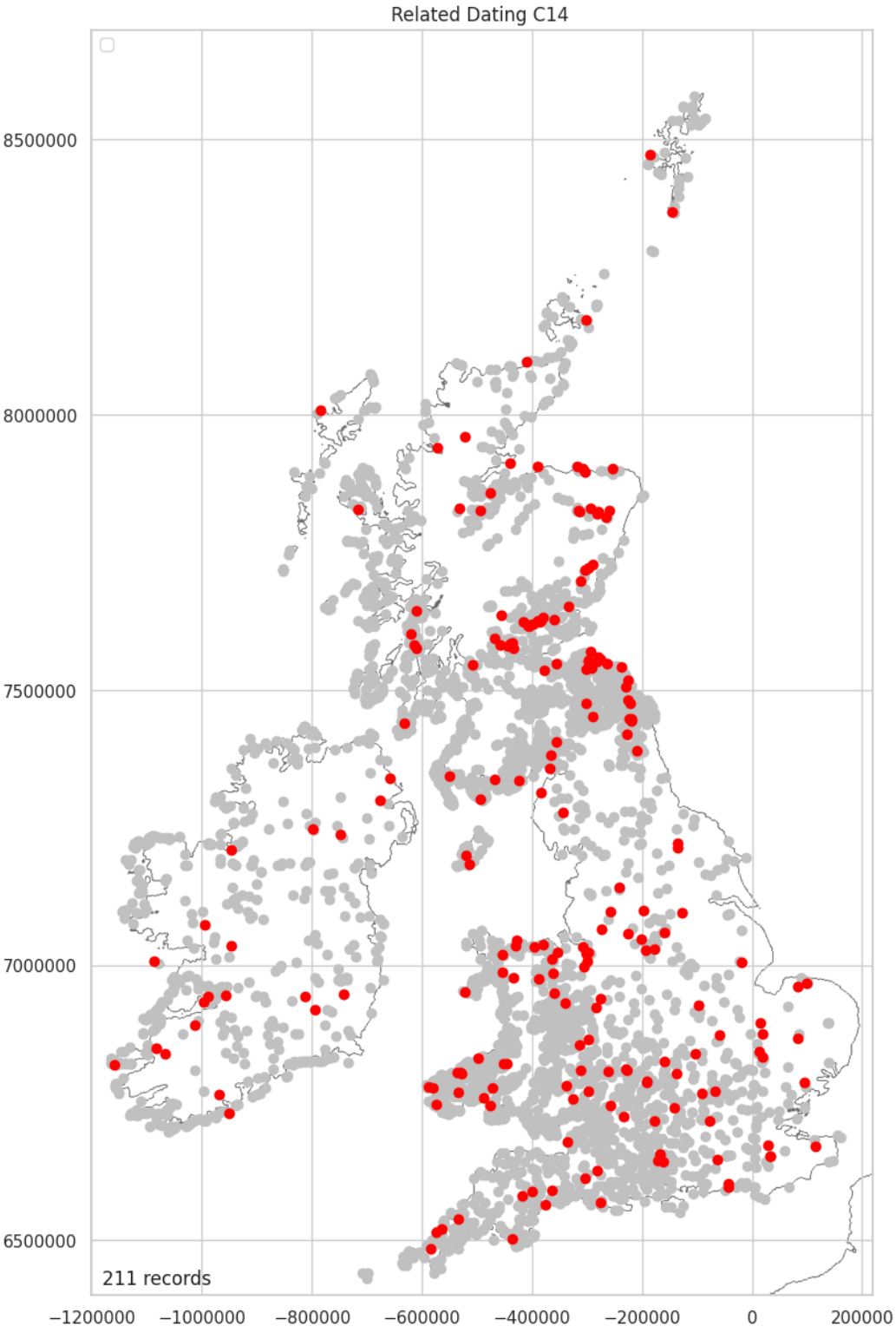


Map Related Dating C14

Carbon 14 dating (C14) is the most scientifically rigerours of the dating techniches recored in the atlas. Only 5.09% of hillforts have a C14 date but there does look to be a more even distribution of dates accross the area of the atlas.

In [ ]:
```
c14_stats = plot_over_grey(location_dating_encodeable_plus_data, 'Related_Dating_C1
```

```
Saving figure hillforts_primer_part03-038.png
```



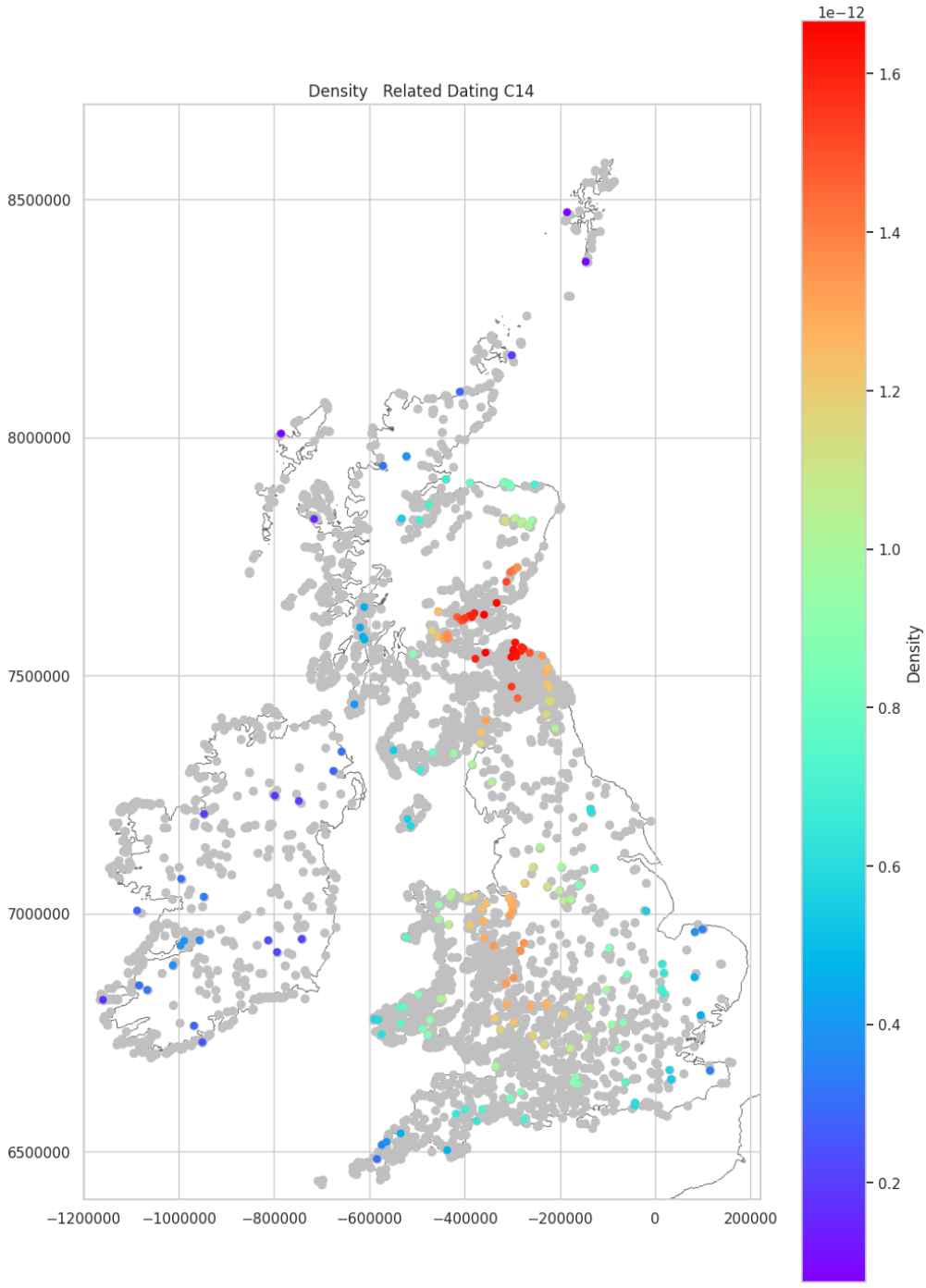
5.09%

Map Related Dating C14 Density

There is a cluster of dates in the eastern Scottish lowlands, particularly around Traprain Law in Eat Lothian and along the line of the Gask Ridge. There is a another, thin concentration, along the northern Welsh/English border, from the Shropshire Hills to the edge of Snowdonia.

In [ ]:
```
plot_density_over_grey(c14_stats, 'Related_Dating_C14')
```

```
Saving figure hillforts_primer_part03-039.png
```

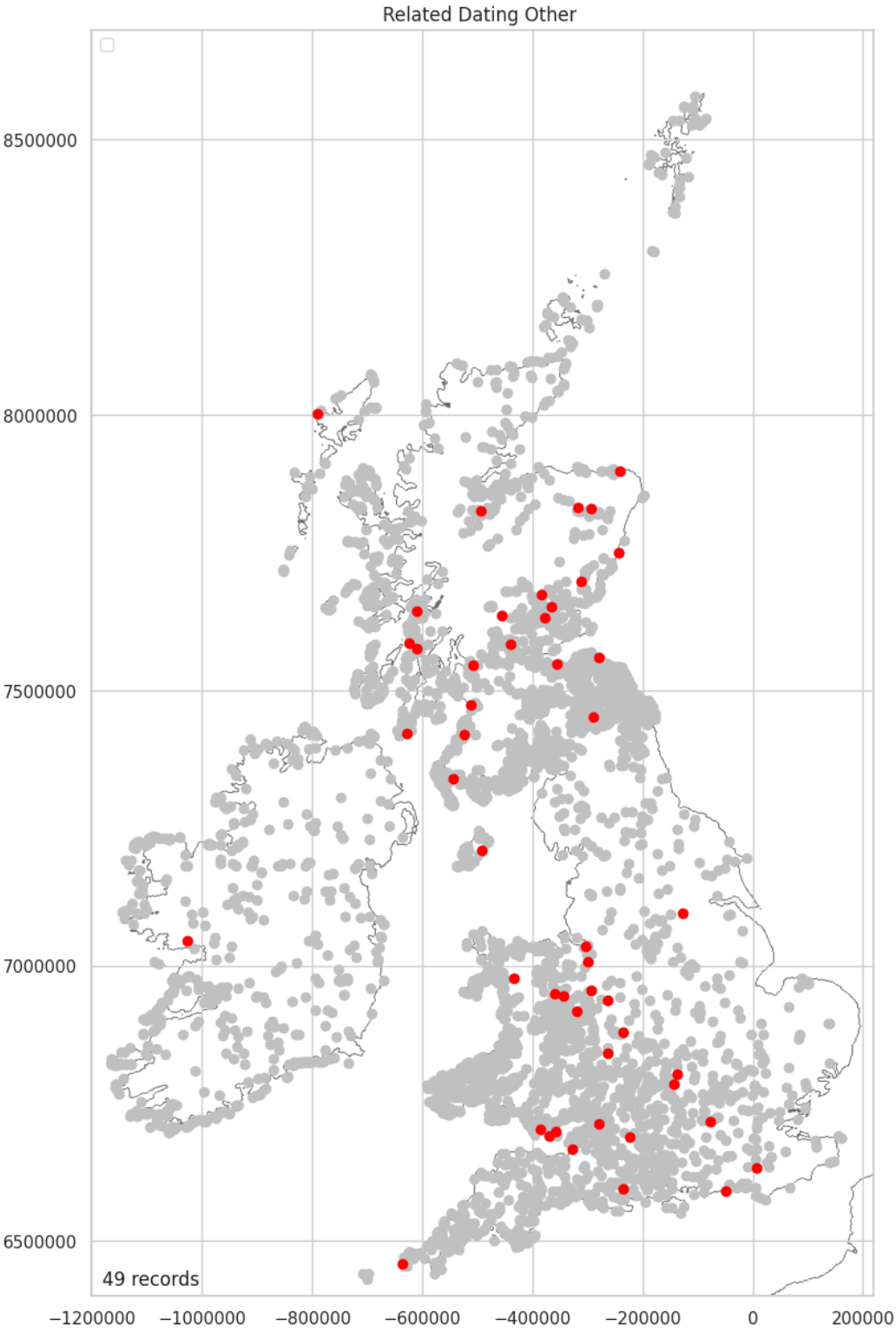


Map Related Dating Other

Fourty nine hillofts (1.18%) of hillforts are identified as having 'other' dating evidence. No further information is available via the online data.

In [ ]:
```
dating_other_stats = plot_over_grey(location_dating_encodeable_plus_data, 'Related_
```

```
Saving figure hillforts_primer_part03-040.png
```



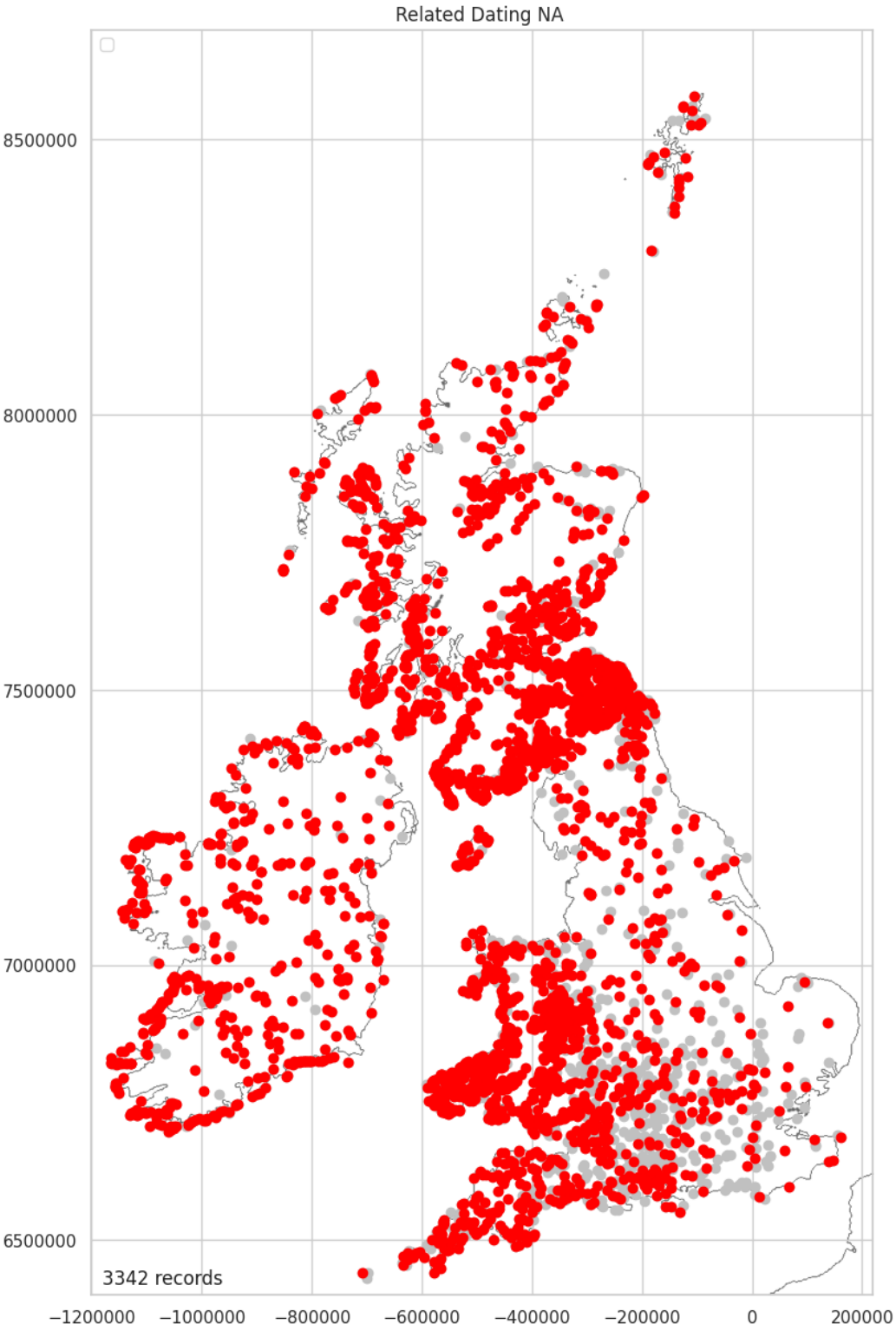
1.18%

Map Related Dating NA

Most (80.59%) of hillforts have no dating evidence.

In [ ]:
```
dating_na_stats = plot_over_grey(location_dating_encodeable_plus_data, 'Related_Da
```

```
Saving figure hillforts_primer_part03-041.png
```



80.59%

Dating Data Package

In [ ]:
```
dating_data_list = [dating_numeric_data, dating_text_data, dating_encodeable_data_
```

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 [ ]:
```
download(dating_data_list, 'Dating_package')
```

Save Figure List

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

Part 4: Investigations & Interior

Colab Notebook: Live code
HTML: Read only
HTML: Read only topographic