

# Hillforts Primer

## An Analysis of the Atlas of Hillforts of Britain and Ireland

### Appendix 1

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](#)

### 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
```

## 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 [Appendix 1](#).

## 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](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](http://naturalearthdata.com).

## Reload Data and Python Functions

This study is split over multiple documents. Each file needs to be configured and have the source data imported. As this section does not focus on the assessment of the data it is minimised to facilitate the documents readability.

## Python Modules and Code Setup

The Python imports enable the Hillforts Atlas data to be analysed and mapped within this document. The Python code can be run on demand, (see: [User Settings](#)). This means that as new research becomes available, the source for this document can be updated to a revised copy of the Atlas data and the impact of that research can be reviewed using the same code and graphic output. The Hillforts Atlas is a baseline and this document is a tool that can be used to assess the impact new research is making in this area.

```
In [5]: 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>

Ref. <https://www>

Ref: <https://scik>

Ref: <https://pan>

Ref: <https://num>

Ref: <https://mat>

Ref. <https://tinyurl.com/292333>

Ref: <https://seal>

Ref: <https://doc>

Ref: <https://pvp>

1 / 191

# Plot Figures and Maps functions

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

```
In [6]: 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)
```

```

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 [8]: 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 [9]: 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 [10]: 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 [11]: 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 [12]: 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 [13]: def plot_bar_chart(data, split_pos, x_label, y_label, title, clip=False):
    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:])
    if clip:
        x_data_new = x_data_new[:-1]
        new_data = data.copy()
        data = new_data.drop(['Dating_Date_Unknown'], axis=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 [14]: 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 [15]: def plot_bar_chart_numeric(data, split_pos, x_label, y_label, title, n_bins, \
extra=''):
    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)
    title = f'{title} {extra}'
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()
```

```
In [16]: def plot_continuous(data, x_label, title):
    fig = plt.figure(figsize=(12,8))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel(x_label)
    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 [17]: def plot_data_range(data, feature, o="v"):
    fig = plt.figure(figsize=(12,2))
    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", whis=[2.2, 97.8])
    else:
        sns.boxplot(data=data, orient="h", whis=[2.2, 97.8])
    save_fig(feature + " Range")
    plt.show()

    bp = boxplot_stats(data, whis=[2.2, 97.8])

    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 [18]: def plot_data_range_plus(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 (Outlier Steps)"))
    plt.ticklabel_format(style='plain')
    if o == "v":
        sns.boxplot(data=data, orient="v", whis=[2.2, 97.8])
    else:
        sns.boxplot(data=data, orient="h", whis=[2.2, 97.8])

    # Add annotation Lines
    x = [24, 24, 54, 54]
    y = [-0.05, -0.075, -0.075, -0.05]
    x1 = [54, 54, 84, 84]
    y1 = [-0.1, -0.125, -0.125, -0.1]
    x2 = [84, 84, 114, 114]
    y2 = [-0.05, -0.075, -0.075, -0.05]

    line_1 = plt.plot(x,y)
    line_2 = plt.plot(x1,y1)
    line_3 = plt.plot(x2,y2)

    # Add annotation text
    text_kw_args = dict(ha='center', va='center', fontsize=16, color='k')
    plt.text(39, -0.1, '30 Ha', **text_kw_args)
    plt.text(69, -0.1, '30 Ha', **text_kw_args)
    plt.text(99, -0.1, '30 Ha', **text_kw_args)

    save_fig(feature + " Range")
    plt.show()

    return

```

```

In [19]: def location_XY_plot():
    plt.ticklabel_format(style='plain')
    plt.xlim(-1200000,220000)
    plt.ylim(6400000,8700000)
    add_annotation_l_xy=plt

```

```

In [20]: 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 [21]: 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()
    print(f'{round(((len(plot_data)/4147)*100), 2)}%')

```

```

In [22]: 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()

```

In [23]:

```

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 [24]:

```

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 [25]:

```

def plot_density_over_grey(data, data_type, extra='', inner=False, fringe=False):
    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)
    if fringe:
        add_21Ha_fringe()
    if inner:
        add_21Ha_line()
        plt.legend(loc='lower left')
    plt.colorbar(label='Density')
    title = f'Density - {data_type} {extra}'
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()

```

In [26]:

```

def plot_density_over_grey_three(data_low, data_iqr, data_high, title, \
                                  extra='', inner=False, fringe=False):
    new_data_low = data_low.copy()
    new_data_low = new_data_low.drop(['Density'], axis=1)
    new_data_low = add_density(new_data_low)

    new_data_iqr = data_iqr.copy()
    new_data_iqr = new_data_iqr.drop(['Density'], axis=1)
    new_data_iqr = add_density(new_data_iqr)

    new_data_high = data_high.copy()
    new_data_high = new_data_high.drop(['Density'], axis=1)
    new_data_high = add_density(new_data_high)

    fig, ax = plt.subplots(1, 3)
    fig.set_figheight(7)
    fig.set_figwidth(15)

    bounds = get_bounds()
    folder = "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    # "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/
    # hillforts-topo"
    background = os.path.join(folder, "hillforts-bw-02.png")
    bounds = bounds[0]
    img = plt.imread(background)
    ax[0].imshow(img, extent=bounds)
    ax[1].imshow(img, extent=bounds)
    ax[2].imshow(img, extent=bounds)

    ax[0].scatter(new_data_low['Location_X'], new_data_low['Location_Y'], \
                  c=new_data_low['Density'], cmap=cm.rainbow, s=25)

```

```

ax[1].scatter(new_data_iqr['Location_X'], new_data_iqr['Location_Y'], \
             c=new_data_iqr['Density'], cmap=cm.rainbow, s=25)
ax[2].scatter(new_data_high['Location_X'], new_data_high['Location_Y'], \
             c=new_data_high['Density'], cmap=cm.rainbow, s=25)

ax[0].get_yaxis().set_visible(False)
ax[1].get_yaxis().set_visible(False)
ax[2].get_yaxis().set_visible(False)

ax[0].get_xaxis().set_visible(False)
ax[1].get_xaxis().set_visible(False)
ax[2].get_xaxis().set_visible(False)

ax[0].set_title("1st Quarter (Tiny Hillforts)")
ax[1].set_title("IQR (Small to Medium Hillforts)")
ax[2].set_title("4th Quarter (Large Hillforts)")

fig.suptitle(get_print_title(title), y=1.08)
ax[0].annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
               color='grey', xy=(0, -0.1), xycoords='axes fraction', \
               horizontalalignment = 'left')
ax[2].annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
               size='small', color='grey', xy=(1, -0.1), \
               xycoords='axes fraction', horizontalalignment = 'right')
save_fig(title)
plt.show()

```

```

In [27]: def plot_density_over_grey_four(data_1, data_2, data_3, data_4, title, \
                                         extra=',', inner=False, fringe=False):
    new_data_1 = data_1.copy()
    new_data_1 = new_data_1.drop(['Density'], axis=1)
    new_data_1 = add_density(new_data_1)

    new_data_2 = data_2.copy()
    new_data_2 = new_data_2.drop(['Density'], axis=1)
    new_data_2 = add_density(new_data_2)

    new_data_3 = data_3.copy()
    new_data_3 = new_data_3.drop(['Density'], axis=1)
    new_data_3 = add_density(new_data_3)

    new_data_4 = data_4.copy()
    new_data_4 = new_data_4.drop(['Density'], axis=1)
    new_data_4 = add_density(new_data_4)

    fig, ax = plt.subplots(1, 4)
    fig.set_figheight(7)
    fig.set_figwidth(20)

    bounds = get_bounds()
    folder = "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    # "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/
    # hillforts-topo/"
    background = os.path.join(folder, "hillforts-bw-02.png")
    bounds = bounds[0]
    img = plt.imread(background)
    ax[0].imshow(img, extent=bounds)
    ax[1].imshow(img, extent=bounds)
    ax[2].imshow(img, extent=bounds)
    ax[3].imshow(img, extent=bounds)

    ax[0].scatter(new_data_1['Location_X'], new_data_1['Location_Y'], \
                 c=new_data_1['Density'], cmap=cm.rainbow, s=25)
    ax[1].scatter(new_data_2['Location_X'], new_data_2['Location_Y'], \
                 c=new_data_2['Density'], cmap=cm.rainbow, s=25)
    ax[2].scatter(new_data_3['Location_X'], new_data_3['Location_Y'], \
                 c=new_data_3['Density'], cmap=cm.rainbow, s=25)
    ax[3].scatter(new_data_4['Location_X'], new_data_4['Location_Y'], \
                 c=new_data_4['Density'], cmap=cm.rainbow, s=25)

    ax[0].get_yaxis().set_visible(False)
    ax[1].get_yaxis().set_visible(False)
    ax[2].get_yaxis().set_visible(False)
    ax[3].get_yaxis().set_visible(False)

    ax[0].get_xaxis().set_visible(False)
    ax[1].get_xaxis().set_visible(False)
    ax[2].get_xaxis().set_visible(False)
    ax[3].get_xaxis().set_visible(False)

    ax[0].set_title("NE")
    ax[1].set_title("SE")
    ax[2].set_title("SW")
    ax[3].set_title("NW")

```

```

fig.suptitle(get_print_title(title), y=1.08)
ax[0].annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
               color='grey', xy=(0, -0.1), xycoords='axes fraction', \
               horizontalalignment = 'left')
ax[3].annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
               size='small', color='grey', xy=(1, -0.1), \
               xycoords='axes fraction', horizontalalignment = 'right')
save_fig(title)
plt.show()

```

```

In [28]: def plot_density_over_grey_five(data_1, data_2, data_3, data_4, data_5, title, \
                                      extra='', inner=False, fringe=False):
    new_data_1 = data_1.copy()
    new_data_1 = new_data_1.drop(['Density'], axis=1)
    new_data_1 = add_density(new_data_1)

    new_data_2 = data_2.copy()
    new_data_2 = new_data_2.drop(['Density'], axis=1)
    new_data_2 = add_density(new_data_2)

    new_data_3 = data_3.copy()
    new_data_3 = new_data_3.drop(['Density'], axis=1)
    new_data_3 = add_density(new_data_3)

    new_data_4 = data_4.copy()
    new_data_4 = new_data_4.drop(['Density'], axis=1)
    new_data_4 = add_density(new_data_4)

    new_data_5 = data_5.copy()
    new_data_5 = new_data_5.drop(['Density'], axis=1)
    new_data_5 = add_density(new_data_5)

    fig, ax = plt.subplots(1, 5)
    fig.set_fignheight(7)
    fig.set_fignwidth(24)

    bounds = get_bounds()
    folder = "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    # "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/
    # hillforts-topo"
    background = os.path.join(folder, "hillforts-bw-02.png")
    bounds = bounds[0]
    img = plt.imread(background)
    ax[0].imshow(img, extent=bounds)
    ax[1].imshow(img, extent=bounds)
    ax[2].imshow(img, extent=bounds)
    ax[3].imshow(img, extent=bounds)
    ax[4].imshow(img, extent=bounds)

    ax[0].scatter(new_data_1['Location_X'], new_data_1['Location_Y'], \
                  c=new_data_1['Density'], cmap=cm.rainbow, s=25)
    ax[1].scatter(new_data_2['Location_X'], new_data_2['Location_Y'], \
                  c=new_data_2['Density'], cmap=cm.rainbow, s=25)
    ax[2].scatter(new_data_3['Location_X'], new_data_3['Location_Y'], \
                  c=new_data_3['Density'], cmap=cm.rainbow, s=25)
    ax[3].scatter(new_data_4['Location_X'], new_data_4['Location_Y'], \
                  c=new_data_4['Density'], cmap=cm.rainbow, s=25)
    ax[4].scatter(new_data_5['Location_X'], new_data_5['Location_Y'], \
                  c=new_data_5['Density'], cmap=cm.rainbow, s=25)

    ax[0].get_yaxis().set_visible(False)
    ax[1].get_yaxis().set_visible(False)
    ax[2].get_yaxis().set_visible(False)
    ax[3].get_yaxis().set_visible(False)
    ax[4].get_yaxis().set_visible(False)

    ax[0].get_xaxis().set_visible(False)
    ax[1].get_xaxis().set_visible(False)
    ax[2].get_xaxis().set_visible(False)
    ax[3].get_xaxis().set_visible(False)
    ax[4].get_xaxis().set_visible(False)

    ax[0].set_title("0")
    ax[1].set_title("1")
    ax[2].set_title("2")
    ax[3].set_title("3")
    ax[4].set_title("4")

    fig.suptitle(get_print_title(title), y=1.08)
    ax[0].annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
                   color='grey', xy=(0, -0.1), xycoords='axes fraction', \
                   horizontalalignment = 'left')
    ax[4].annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
                   size='small', color='grey', xy=(1, -0.1), \
                   xycoords='axes fraction', horizontalalignment = 'right')

```

```
save_fig(title)
plt.show()
```

```
In [29]: def plot_density_over_grey_six(data_1, data_2, data_3, data_4, data_5, \
                                   data_6, title, extra='', inner=False, fringe=False):
    new_data_1 = data_1.copy()
    new_data_1 = new_data_1.drop(['Density'], axis=1)
    new_data_1 = add_density(new_data_1)

    new_data_2 = data_2.copy()
    new_data_2 = new_data_2.drop(['Density'], axis=1)
    new_data_2 = add_density(new_data_2)

    new_data_3 = data_3.copy()
    new_data_3 = new_data_3.drop(['Density'], axis=1)
    new_data_3 = add_density(new_data_3)

    new_data_4 = data_4.copy()
    new_data_4 = new_data_4.drop(['Density'], axis=1)
    new_data_4 = add_density(new_data_4)

    new_data_5 = data_5.copy()
    new_data_5 = new_data_5.drop(['Density'], axis=1)
    new_data_5 = add_density(new_data_5)

    new_data_6 = data_6.copy()
    new_data_6 = new_data_6.drop(['Density'], axis=1)
    new_data_6 = add_density(new_data_6)

    fig, ax = plt.subplots(1, 6)
    fig.set_figheight(6)
    fig.set_figwidth(24)

    bounds = get_bounds()
    folder = "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    # "https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-topo/"
    background = os.path.join(folder, "hillforts-bw-02.png")
    bounds = bounds[0]
    img = plt.imread(background)
    ax[0].imshow(img, extent=bounds)
    ax[1].imshow(img, extent=bounds)
    ax[2].imshow(img, extent=bounds)
    ax[3].imshow(img, extent=bounds)
    ax[4].imshow(img, extent=bounds)
    ax[5].imshow(img, extent=bounds)

    ax[0].scatter(new_data_1['Location_X'], new_data_1['Location_Y'], \
                  c=new_data_1['Density'], cmap=cm.rainbow, s=25)
    ax[1].scatter(new_data_2['Location_X'], new_data_2['Location_Y'], \
                  c=new_data_2['Density'], cmap=cm.rainbow, s=25)
    ax[2].scatter(new_data_3['Location_X'], new_data_3['Location_Y'], \
                  c=new_data_3['Density'], cmap=cm.rainbow, s=25)
    ax[3].scatter(new_data_4['Location_X'], new_data_4['Location_Y'], \
                  c=new_data_4['Density'], cmap=cm.rainbow, s=25)
    ax[4].scatter(new_data_5['Location_X'], new_data_5['Location_Y'], \
                  c=new_data_5['Density'], cmap=cm.rainbow, s=25)
    ax[5].scatter(new_data_5['Location_X'], new_data_5['Location_Y'], \
                  c=new_data_5['Density'], cmap=cm.rainbow, s=25)

    ax[0].get_yaxis().set_visible(False)
    ax[1].get_yaxis().set_visible(False)
    ax[2].get_yaxis().set_visible(False)
    ax[3].get_yaxis().set_visible(False)
    ax[4].get_yaxis().set_visible(False)
    ax[5].get_yaxis().set_visible(False)

    ax[0].get_xaxis().set_visible(False)
    ax[1].get_xaxis().set_visible(False)
    ax[2].get_xaxis().set_visible(False)
    ax[3].get_xaxis().set_visible(False)
    ax[4].get_xaxis().set_visible(False)
    ax[5].get_xaxis().set_visible(False)

    ax[0].set_title("Part Univallate")
    ax[1].set_title("Univallate")
    ax[2].set_title("Part Bivallate")
    ax[3].set_title("Bivallate")
    ax[4].set_title("Part Multivallate")
    ax[5].set_title("Multivallate")

    fig.suptitle(get_print_title(title), y=1.08)
    ax[0].annotate("Middleton, M. 2024, Hillforts Primer", size='small', \
                   color='grey', xy=(0, -0.1), xycoords='axes fraction', \
                   horizontalalignment = 'left')
```

```

        ax[5].annotate("Source Data: Lock & Ralston, 2017. hillforts.arch.ox.ac.uk", \
                       size='small', color='grey', xy=(1, -0.1), \
                       xycoords='axes fraction', horizontalalignment = 'right')
    save_fig(title)
    plt.show()

```

In [30]:

```

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 [31]:

```

def plot_over_grey(merged_data, a_type, yes_no, extra="", inner=False, \
                   fringe=False, oxford=False, swindon=False, topo=False):
    # plots selected data over the grey dots. yes_no controls 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)/4147)*100), 2}%')
    return plot_data

```

In [32]:

```

def plot_type_values(data, data_type, title, extra=''):
    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()
    add_grey()
    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)
    title = f'{data_type} {extra}'
    plt.title(get_print_title(title))
    save_fig(title)
    plt.show()

```

In [33]:

```

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 [34]:

```

def add_cluster_split_lines(plt, ax, extra=None):
    x_min = -550000
    if extra == 'ireland':
        x_min = -1200000
    plt.vlines(x=[-500000], ymin=7070000, ymax=9000000, colors='r', ls='-', lw=3)
    plt.hlines(y=[7070000], xmin=x_min, xmax=200000, colors='r', ls='-', lw=3)
    ax.annotate("N/S split", color='k', xy=(50000, 7090000), xycoords='data')
    ax.annotate("E/W split", color='k', xy=(-480000, 8660000), xycoords='data')
    ax.annotate("Irish Sea split", color='k', xy=(-1150000, 7510000), \
                xycoords='data')
    plot_line((-800000, 6400000), (-550000, 7070000))
    plot_line((-550000, 7070000), (-666000, 7440000))
    plot_line((-666000, 7440000), (-900000, 7500000))

```

In [35]:

```

def plot_values(data, feature, title, extra=''):
    fig, ax = plt.subplots(figsize=((14.2 * 0.66)+2.4, 23.0 * 0.66))

```

```

show_background(plt, ax)
location_XY_plot()
plt.scatter(data['Location_X'], data['Location_Y'], c=data[feature], \
            cmap=cm.rainbow, s=25)
plt.colorbar(label=feature)
title = f'{title} {extra}'
plt.title(title)
save_fig(title)
plt.show()

```

```
In [36]: def plot_line(point1, point2):
    x_values = [point1[0], point2[0]]
    y_values = [point1[1], point2[1]]
    plt.plot(x_values, y_values, 'r', ls='0', lw=2, alpha=1)
```

```
In [37]: def density_scatter_lines(location_data, scatter_data, plot_title, inner=False, \
                                fringe=False):
    fig, ax = plt.subplots(figsize=((14.2 * 0.66)+2.0, 23.0 * 0.66))
    show_background(plt, ax)
    location_XY_plot()
    plt.scatter(location_data['Location_X'], location_data['Location_Y'], \
                c=location_data['Density_trans'], cmap=cm.rainbow, s=25)
    plt.colorbar(label='Density Transformed')
    if inner:
        add_21Ha_line()
    if fringe:
        add_21Ha_fringe()
    plt.scatter(scatter_data['Location_X'], scatter_data['Location_Y'], c='Red')
    plt.legend(loc='lower left')
    plt.title(get_print_title(plot_title))
    save_fig(plot_title)
    plt.show()
```

```
In [38]: def add_limits():
    x_values = [-584307, 115292]
    y_values = [7019842, 7019842]
    xx_values = [115292, 115292]
    yy_values = [7019842, 6485285]
    xxx_values = [-584307, 115292]
    yyy_values = [6485285, 6485285]
    x4_values = [-584307, -584307]
    y4_values = [7019842, 6485285]

    plt.plot(x_values, y_values, 'g', ls='-', lw=8, alpha=0.6, \
              label = 'Bounds of test')
    plt.plot(xx_values, yy_values, 'g', ls='-', lw=8, alpha=0.6)
    plt.plot(xxx_values, yyy_values, 'g', ls='-', lw=8, alpha=0.6)
    plt.plot(x4_values, y4_values, 'g', ls='-', lw=8, alpha=0.6)
```

```
In [39]: def add_linear_south():
    x_values = [-115637, -286900]
    y_values = [6678188, 6585812]
    xx_values = [-244249, -363049]
    yy_values = [6555133, 6589612]
    xxx_values = [-392213, -363146]
    yyy_values = [6577365, 6647256]
    x4_values = [-169664, -207084]
    y4_values = [6599254, 6615290]
    x5_values = [-238560, -200891]
    y5_values = [6668083, 6637826]

    plt.plot(x_values, y_values, 'g', ls='-', lw=8, alpha=0.6, \
              label = 'Poss. correlation to linear routes?')
    plt.plot(xx_values, yy_values, 'g', ls='-', lw=8, alpha=0.6)
    plt.plot(xxx_values, yyy_values, 'g', ls='-', lw=8, alpha=0.6)
    plt.plot(x4_values, y4_values, 'g', ls='-', lw=8, alpha=0.6)
    plt.plot(x5_values, y5_values, 'g', ls='-', lw=8, alpha=0.6)
```

```
In [40]: def plot_over_grey_south(merged_data, a_type, yes_no, extra=""):
    # plots selected data over the grey dots. yes_no controls filtering the
    # data for a positive or negative values.
    plot_data = merged_data[merged_data[a_type] == yes_no]
    fig, ax = plt.subplots(1, figsize=((6.73*1.5), (4.62*1.5)))
    show_background(plt, ax, 's')
    plt.ticklabel_format(style='plain')
    plt.xlim(-800000, 220000)
    plt.ylim(6400000, 7100000)
    add_annotation_l_xy=plt
    add_grey('s')
    add_grey('s')
    add_linear_south()
    plt.scatter(plot_data['Location_X'], plot_data['Location_Y'], c='Red')
    plt.legend(loc='lower right')
```

```

plt.title(get_print_title(f'{a_type} {extra}'))
save_fig(f'{a_type}_{extra}')
plt.show()
return plot_data

```

```

In [41]: 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 [42]: 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()

```

```

In [43]: def get_pcent_list(old_list):
    pcnt_list = []
    total = sum(old_list)
    for item in old_list:
        pcnt_list.append(round(item/total,2))
    return pcnt_list

```

```

In [44]: def order_set(set_list, x_data, pcnt=False):
    new_list = []
    set_values = set_list.index.tolist()
    for val in x_data:
        if val in set_values:
            new_list.append(set_list.loc[[val]].values[0])
        else:
            new_list.append(0)
    if pcnt:
        new_list = get_pcent_list(new_list)
    return new_list

```

```

In [45]: def plot_feature_by_region(nw,ne,ni,si,s, feature, title, clip):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    max_val = int(max([nw[feature].max(),ne[feature].max(),\
                      ni[feature].max(),si[feature].max(),s[feature].max()]))+2

    x_data = [x-1 for x in range(max_val+2)]

    set0_name = 'NW'
    set1_name = 'NE'

```

```

set2_name = 'N Ireland'
set3_name = 'S Ireland'
set4_name = 'S'

set0 = nw[feature].value_counts()
set1 = ne[feature].value_counts()
set2 = ni[feature].value_counts()
set3 = si[feature].value_counts()
set4 = s[feature].value_counts()

set0 = order_set(set0,x_data, True)[:clip]
set1 = order_set(set1,x_data, True)[:clip]
set2 = order_set(set2,x_data, True)[:clip]
set3 = order_set(set3,x_data, True)[:clip]
set4 = order_set(set4,x_data, True)[:clip]

X_axis = np.arange(len(x_data[:clip]))

budge = 0.2

plt.bar(X_axis - 0.6 + budge, set0, 0.3, label = set0_name)
plt.bar(X_axis - 0.45 + budge, set1, 0.3, label = set1_name)
plt.bar(X_axis - 0.3 + budge, set2, 0.3, label = set2_name)
plt.bar(X_axis - 0.15 + budge, set3, 0.3, label = set3_name)
plt.bar(X_axis + 0 + budge, set4, 0.3, label = set4_name)

plt.xticks(X_axis, x_data)
plt.xlabel('Number')
plt.ylabel('Percentage of regional total')
plt.title(title)
plt.legend()
add_annotation_plot(ax)
save_fig(title)
plt.show()

```

```

In [46]: def plot_quadrants(ramparts,ditches,ne,se,sw,nw):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    x_data = [x for x in range(11)]

    set0_name = 'Ramparts'
    set00_name = 'Ditches'
    set1_name = 'NE'
    set2_name = 'SE'
    set3_name = 'SW'
    set4_name = 'NW'
    set0 = ramparts['Enclosing_Max_Ramparts'].value_counts()
    set0 = order_set(set0,x_data)[:8]
    set00 = ditches['Enclosing_Ditches_Number'].value_counts()
    set00 = order_set(set00,x_data)[:8]
    set1 = ne['Enclosing_NE_Quadrant'].value_counts()
    set1 = order_set(set1,x_data)[:8]
    set2 = se['Enclosing_SE_Quadrant'].value_counts()
    set2 = order_set(set2,x_data)[:8]
    set3 = sw['Enclosing_SW_Quadrant'].value_counts()
    set3 = order_set(set3,x_data)[:8]
    set4 = nw['Enclosing_NW_Quadrant'].value_counts()
    set4 = order_set(set4,x_data)[:8]

    X_axis = np.arange(len(x_data[:8]))

    budge = 0.2

    plt.bar(X_axis - 0.6 + budge, set0, 0.2, label = set0_name)
    plt.bar(X_axis - 0.46 + budge, set00, 0.2, label = set00_name)
    plt.bar(X_axis - 0.32 + budge, set1, 0.2, label = set1_name)
    plt.bar(X_axis - 0.18 + budge, set2, 0.2, label = set2_name)
    plt.bar(X_axis - 0.04 + budge, set3, 0.2, label = set3_name)
    plt.bar(X_axis + 0.1 + budge, set4, 0.2, label = set4_name)

    plt.xticks(X_axis, x_data)
    plt.xlabel('Number')
    plt.ylabel('Count')
    title = 'Ditches, Ramparts and Quadrant by Number'
    plt.title(title)
    plt.legend()
    add_annotation_plot(ax)
    save_fig(title)
    plt.show()

```

```

In [47]: def plot_regions(nw,ne,ni,si,s, features, xlabel, title, split_pos, \
                    yes_no, pcent=False):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])

```

```

x_data = features
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:])

set0_name = 'NW'
set1_name = 'NE'
set2_name = 'N Ireland'
set3_name = 'S Ireland'
set4_name = 'S'

set0_list = []
set1_list = []
set2_list = []
set3_list = []
set4_list = []

for feature in features:
    set0_list.append((nw[feature].values == yes_no).sum())
    set1_list.append((ne[feature].values == yes_no).sum())
    set2_list.append((ni[feature].values == yes_no).sum())
    set3_list.append((si[feature].values == yes_no).sum())
    set4_list.append((s[feature].values == yes_no).sum())

set0 = set0_list
set1 = set1_list
set2 = set2_list
set3 = set3_list
set4 = set4_list

if pcent:
    set0 = get_pcent_list(set0)
    set1 = get_pcent_list(set1)
    set2 = get_pcent_list(set2)
    set3 = get_pcent_list(set3)
    set4 = get_pcent_list(set4)

X_axis = np.arange(len(x_data))

budge = 0.3

plt.bar(X_axis - 0.6 + budge, set0, 0.13, label = set0_name)
plt.bar(X_axis - 0.45 + budge, set1, 0.13, label = set1_name)
plt.bar(X_axis - 0.3 + budge, set2, 0.13, label = set2_name)
plt.bar(X_axis - 0.15 + budge, set3, 0.13, label = set3_name)
plt.bar(X_axis + 0 + budge, set4, 0.13, label = set4_name)

plt.xticks(X_axis, x_data_new)
plt.xlabel(xlabel)
if pcent:
    plt.ylabel('Percentage of Regional Total')
else:
    plt.ylabel('Count')
plt.title(get_print_title(f'{title}'))
plt.legend()
add_annotation_plot(ax)
save_fig(title)
plt.show()

```

In [48]:

```

def plot_data_range_95(data, feature, title, o="v"):
    fig = plt.figure(figsize=(12,2))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel(feature)
    add_annotation_plot(ax)
    plt.title(title)
    plt.ticklabel_format(style='plain')
    plt.axvline(x=mse_h1, color='r', lw=4)
    if o == "v":
        sns.boxplot(data=data, orient="v", whis=[2.5, 97.5])
    else:
        sns.boxplot(data=data, orient="h", whis=[2.5, 97.5])

    plot_title = get_print_title(title)
    save_fig(plot_title)
    plt.show()

    bp = boxplot_stats(data, whis=[2.5, 97.5])

    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 [49]: def plot_mse_histogram(mse_list, mse_h1):
    fig = plt.figure(figsize=(12,5))
    ax = fig.add_axes([0,0,1,1])
    ax.set_xlabel('MSE')
    ax.set_ylabel('Count')
    plt.ticklabel_format(style='plain')
    plt.hist(mse_list, 100)
    plot_title = \
        get_print_title('Mean Squared Error for hillforts offset from test lines - H1 MSE shown in red.')
    #'Mean Squared Error for hillforts offset from test Lines -
    # H1 MSE shown in red.'
    plt.title(plot_title)
    plt.axvline(x=mse_h1, color='r', lw=4)
    add_annotation_plot(ax)
    save_fig(plot_title)
    plt.show()

```

```

In [50]: def rotate_around_point(xy, radians, rotation_origin=(0, 0)):
    x, y = xy
    offset_x, offset_y = rotation_origin
    adjusted_x = (x - offset_x)
    adjusted_y = (y - offset_y)
    cos_rad = math.cos(radians)
    sin_rad = math.sin(radians)
    qx = offset_x + cos_rad * adjusted_x + sin_rad * adjusted_y
    qy = offset_y + -sin_rad * adjusted_x + cos_rad * adjusted_y

    return qx, qy

```

```

In [51]: def get_random_line(dist):
    test_bounds_x = [-584307, 115292]
    test_bounds_y = [6485285, 7019842]
    rand_point = (random.randrange(test_bounds_x[0],test_bounds_x[1]), \
                  random.randrange(test_bounds_y[0],test_bounds_y[1]))
    new_second_point = (-1000000002, 1000000000)

    count = 0
    while (new_second_point[0] <= test_bounds_x[0]) or \
        (new_second_point[0] >= test_bounds_x[1]) or \
        (new_second_point[1] <= test_bounds_y[0]) or \
        (new_second_point[1] >= test_bounds_y[1]):
        rand_second_point = (rand_point[0]+dist, rand_point[1])
        rand_angle_rad = random.uniform(0.0,3.14)
        new_second_point = rotate_around_point(rand_second_point, \
                                              rand_angle_rad, rand_point)
        count+=1
        if count == 10:
            rand_point = (random.randrange(test_bounds_x[0],test_bounds_x[1]), \
                          random.randrange(test_bounds_y[0],test_bounds_y[1]))
            count = 0
    return [rand_point, new_second_point, math.degrees(rand_angle_rad)]

```

```

In [52]: def line_intersection(line1, line2):
    x = 1000000
    y = 1000000
    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
        pass
    else:
        d = (det(*line1), det(*line2))
        x = det(d, xdiff) / div
        y = det(d, ydiff) / div

    return x, y

```

```

In [53]: def show_perpendicular_lines(points, x_values, y_values, angle, scatter_data, \
                                show_perpendicular):
    length_divider = 4.85
    distance_list = []

    x_bounds = [-584307,115292]
    y_bounds = [6485285,7019842]

```

```

max_len = \
math.sqrt( (x_bounds[0] - x_bounds[1])**2 + (y_bounds[0] - y_bounds[1])**2 )
x0 = points[0][0]
y0 = points[0][1]
x1 = points[1][0]
y1 = points[1][1]
line_dist = math.sqrt( (x0 - x1)**2 + (y0 - y1)**2 )
for index, row in scatter_data.iterrows():
    x2 = row['Location_X']
    y2 = row['Location_Y']
    if x2 >= x_bounds[0] and x2 <= x_bounds[1] and \
    y2 >= y_bounds[0] and y2 <= y_bounds[1]:
        p1 = (x2,y2)
        p2 = (x2+max_len,y2)
        new_p2 = p2
        value = ((x1 - x0)*(y2 - y0)) - ((x2 - x0)*(y1 - y0))
        if value < 0:
            pass
        new_p2 = rotate_around_point(p2, math.radians((angle-90)), p1)
        elif value > 0:
            new_p2 = rotate_around_point(p2, math.radians((angle+90)), p1)
        else:
            # on Line
            new_p2 = p1
    line1 = ((x_values[0], y_values[0]), (x_values[1], y_values[1]))
    line2 = ((x2, y2), new_p2)
    x3, y3 = line_intersection(line1, line2)

    if x3 == 1000000:
        x3 = x2
        y3 = y2

    px_values = [x2, x3]
    py_values = [y2, y3]

    #set east/west
    west = x0
    east = x1
    if east < west:
        west = x1
        east = x0

    #set north/south
    north = y0
    south = y1
    if east < west:
        north = y1
        greater_than_21ha_south = y0

    if x3 >= west and x3 <= east and y3 >= south and y3 <= north:
        dist = \
        abs(math.sqrt( (px_values[1] - px_values[0])**2 + \
        (py_values[1] - py_values[0])**2 ))
        if dist < line_dist/length_divider:
            #print(row['Main_Atlas_Number'])
            distance_list.append(dist)
            if show_perpendicular:
                plt.plot(px_values, py_values, 'magenta', ls='-', \
                lw=3, alpha=0.6)
return distance_list

```

In [54]:

```

def angle_between(p1, p2):
    ang1 = np.arctan2(*p1[::-1])
    ang2 = np.arctan2(*p2[::-1])
    return np.rad2deg((ang1 - ang2) % (2 * np.pi))

```

In [55]:

```

def add_h1_line(scatter_data, show_perpendicular_from_main):
    penycloddiau_m = (312888, 367647)
    bozedown_m = (464350, 178250)
    x_values = [-376969, -119597]
    y_values = [7019842, 6710127]
    angle_rad = \
    math.atan((bozedown_m[1] - penycloddiau_m[1])/(bozedown_m[0] - \
    penycloddiau_m[0]))
    angle = \
    angle_between((x_values[0], y_values[0]), (x_values[1], y_values[1]))+45
    plt.plot(x_values, y_values, 'b', ls='-', lw=8, alpha=0.6, label = 'H1')

    points = [(x_values[0], y_values[0]), (x_values[1], y_values[1]), angle]
    distance_list = []

    if show_perpendicular_from_main:
        distance_list = show_perpendicular_lines(points, x_values, y_values, \

```

```

angle, scatter_data, \
show_perpendicular=True)

return distance_list

```

```
In [56]: def add_random_lines(scatter_data, test_lines, dist, show_perpendicular):
    add_lines = test_lines

    distance_lists = []

    for i in range(add_lines):
        points = get_random_line(dist)
        x_values = [points[0][0], points[1][0]]
        y_values = [points[0][1], points[1][1]]
        angle = points[2]
        plt.plot(x_values, y_values, 'darkmagenta', ls='-', lw=3, alpha=0.6)
        distance_list = []

        distance_list = show_perpendicular_lines(points, x_values, \
                                                    y_values, angle, scatter_data, \
                                                    show_perpendicular)
        distance_lists.append(distance_list)
    return distance_lists
```

```
In [57]: def south_density_scatter_lines(location_data, scatter_data, plot_title, \
                                      inner=False, fringe=False, h1=False, \
                                      limits=False, random_lines=False, test_lines=0, \
                                      show_perpendicular=False, \
                                      show_perpendicular_from_main=False):
    penycloddiau = (-367969, 7019842)
    bozedown = (-119597, 6710127)
    dist = math.sqrt( (bozedown[0] - penycloddiau[0])**2 + (bozedown[1] - \
                                                               penycloddiau[1])**2 )

    h1_distance_list = []
    distance_lists = []
    fig, ax = plt.subplots(figsize=((6.73*1.5)+2.0, (4.62*1.5)))
    show_background=plt, ax, 's')
    plt.ticklabel_format(style='plain')
    plt.xlim(-800000,2200000)
    plt.ylim(6400000,7100000)
    plt.scatter(location_data['Location_X'], location_data['Location_Y'], \
               c=location_data['Density_trans'], cmap=cm.rainbow, s=25)
    plt.colorbar(label='Density Transformed')
    if inner:
        add_21Ha_line()
    if fringe:
        add_21Ha_fringe()
    if h1:
        h1_distance_list = add_h1_line(scatter_data, show_perpendicular_from_main)
    if limits:
        add_limits()
    if random_lines:
        distance_lists = add_random_lines(scatter_data, test_lines, dist, \
                                           show_perpendicular)
    plt.scatter(scatter_data['Location_X'], scatter_data['Location_Y'], \
               c='red', s=60)
    add_annotation_plot=plt)
    plt.legend(loc='lower right')
    plt.title(get_print_title(plot_title))
    save_fig(plot_title)
    plt.show()

    return h1_distance_list, distance_lists
```

## Review Data Functions

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

```
In [58]: 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 [59]: 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 [60]: 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 [61]: 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 [62]: def find_duplicates(data):
print(f'{data.count() - data.duplicated(keep=False).count()} duplicates.')

```

```

In [63]: 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 [64]: def fill_nan_with_minus_one(data, feature):
new_data = data.copy()
new_data[feature] = data[feature].fillna(-1)
return new_data

```

```

In [65]: def fill_nan_with_NA(data, feature):
new_data = data.copy()
new_data[feature] = data[feature].fillna("NA")
return new_data

```

```

In [66]: def test_numeric_value_in_feature(feature, value):
test = feature.isin([-1]).sum()
return test

```

```

In [67]: def test_catagorical_value_in_feature(dataframe, feature, value):
test = dataframe[feature][dataframe[feature] == value].count()
return test

```

```

In [68]: 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 [69]: 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 [70]: 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 [71]: 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 [72]: 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
```

```
In [72]:
```

## Save Image Functions

```
In [73]: # Set-up figure numbering
fig_no = 0
part = 'Appendix01'
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 [74]: # Remove unicode characters from file names
def get_file_name(title):
    file_name = slugify(title)
    return file_name
```

```
In [75]: # Remove underscore from figure titles
def get_print_title(title):
    title = title.replace("_", " ")
    title = title.replace("-", " ")
    title = title.replace(", ", ";")
    return title
```

```
In [76]: # Format figure numbers to have three digits
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 [77]: # Mount Google Drive if figures to be saved
if save_images == True:
    drive.mount('/content/drive')
    os.getcwd()
else:
    pass
```

```
In [78]: 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_01_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 [79]: hillforts_csv = r"https://raw.githubusercontent.com/MikeDairsie/Hillforts-Primer/main/hillforts-atlas-source-data.csv"
#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-79-a029b1de176b>: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[79]:

	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 [80]: 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.

## Load Date Data

```
In [81]: 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 = hillforts_data[date_features]
```

## Download Function

```
In [82]: 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', 'northern-ireland', \
                                'isle-of-man', 'roi-ni', 'eng-wal-sco-iom', \
                                'twenty-one-ha-and-over-south']:
                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)
```

```

d1 = d1.replace('\n',' ', regex=True)
fn = 'hillforts_primer_` + filename
fn = get_file_name(fn)
d1.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 [83]: name_and_number_features = ['Main_Atlas_Number', 'Main_Display_Name']
name_and_number = hillforts_data[name_and_number_features].copy()
name_and_number.head()
```

	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 [84]: location_numeric_data_short_features = \
['Location_X', 'Location_Y', 'Main_X', 'Main_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()
```

	Location_X	Location_Y	Main_X	Main_Y	Density
0	-303295	6798973	350350	233050	1.632859e-12
1	-296646	6843289	354700	260200	1.540172e-12
2	-289837	6808611	358700	238900	1.547729e-12
3	-320850	6862993	340000	272400	1.670548e-12
4	-261765	6810587	376000	240000	1.369981e-12

## Reload Regional Data Packages

See Cluster Data Packages in Part 1: Name, Admin & Location Data

<https://colab.research.google.com/drive/1C7HcuLuGGhG8o4EGciS-XTAhxVs3MhX3?usp=sharing>

```
In [85]: 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]

```

## Reload Enclosing Area 1

```
In [86]: enclosing_data = hillforts_data.copy()
enclosing_numeric_features = ['Enclosing_Area_1']
enclosing_numeric_data = enclosing_data[enclosing_numeric_features].copy()
```

## Enclosing Numeric Data - Resolve Null Values

Test for -1.

```
In [87]: test_num_list_for_minus_one(enclosing_numeric_data, enclosing_numeric_features)

Enclosing_Area_1 0

Replace null with -1.
```

```
In [88]: enclosing_numeric_data = \
update_num_list_for_minus_one(enclosing_numeric_data, enclosing_numeric_features)
enclosing_numeric_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4147 entries, 0 to 4146
Data columns (total 1 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Enclosing_Area_1  4147 non-null   float64 
dtypes: float64(1)
memory usage: 32.5 KB
```

## Appendix 1

### Enclosing Area 1: Distribution of Outliers Over 21 Ha Mapped

After multiple tests to filter the data for forts over various sizes, a possible alignment of hillforts was isolated for forts over 21 Ha.

```
In [89]: enclosing_numeric_data = \
update_num_list_for_minus_one(enclosing_numeric_data, enclosing_numeric_features)
location_enclosing_data = \
pd.merge(location_numeric_data_short, enclosing_numeric_data, left_index=True, \
        right_index=True)
enclosing_area_1_21 = location_enclosing_data.copy()
enclosing_area_1_21 = \
enclosing_area_1_21[enclosing_area_1_21['Enclosing_Area_1']>=21]
enclosing_area_1_21['Enclosing_Area_1'].describe()
```

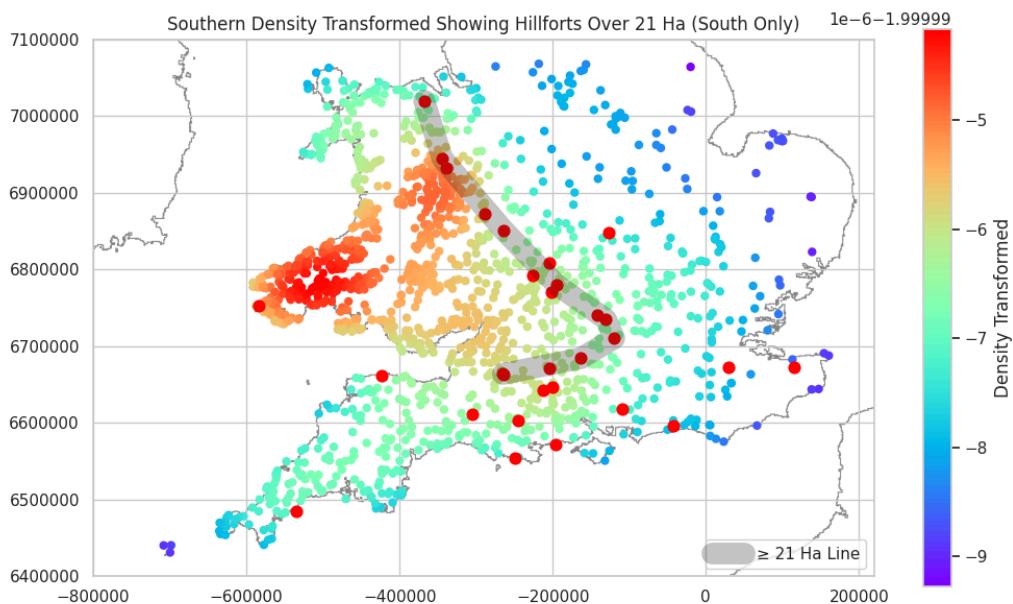
```
Out[89]: count    38.000000
mean      42.501053
std       26.680691
min       21.000000
25%      24.875000
50%      30.000000
75%      51.875000
max      130.000000
Name: Enclosing_Area_1, dtype: float64
```

### Enclosing Area 1: Southern Hillfort Density Transformed overlayed by hillforts over 21 Ha

Map showing possible alignment of hillforts which are 21 hectares or larger.

```
In [90]: cluster_south = south.copy()
enclosing_area_1_21_s = \
enclosing_area_1_21[enclosing_area_1_21['Location_X'] > -600000]
cluster_south = add_density(cluster_south)
cluster_south['Density_trans'] = stats.boxcox(cluster_south['Density'], 0.5)
```

```
In [91]: _, _ = \
south_density_scatter_lines(cluster_south, enclosing_area_1_21_s, \
                             'Southern Density Transformed Showing Hillforts Over 21 Ha (South Only)', \
                             True, False)
# 'Southern Density Transformed Showing Hillforts Over 21 Ha (South Only)'
```



Middleton, M. 2024, Hillforts Primer

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

### Hillforts 21 Ha and over in the South

A full list of hillforts over 21 hectares in the southern data package.

```
In [92]: greater_than_21ha_south = \
pd.merge(name_and_number, enclosing_area_1_21_s, left_index=True, \
         right_index=True)
twenty_one_ha_and_over_south = \
greater_than_21ha_south\ 
[['Main_Atlas_Number', 'Main_Display_Name', 'Enclosing_Area_1', 'Location_X', \
'Location_Y', 'Main_X', 'Main_Y']].\
copy().sort_values(by='Enclosing_Area_1').style.hide_index()
twenty_one_ha_and_over_south = twenty_one_ha_and_over_south.data
twenty_one_ha_and_over_south
```

```
<ipython-input-92-2737903187fa>:8: FutureWarning: this method is deprecated in favour of `Styler.hide(axis="index")` \
copy().sort_values(by='Enclosing_Area_1').style.hide_index()
```

Out[92]:	Main_Atlas_Number	Main_Display_Name	Enclosing_Area_1	Location_X	Location_Y	Main_X	Main_Y
1127	1155	Penycloddiau, Denbighshire (Pen y Cloddiau)		21.0	-367969	7019842	312888 367647
621	643	Dodman Castle, Cornwall (Dodman Point; The Dod...		21.0	-534770	6485285	200100 39850
731	753	Norbury Camp, Northleach, Gloucestershire		22.0	-202278	6770873	412700 215500
1889	1997	Wooltack Point, Pembrokeshire (Deer Park Fort)		22.0	-584307	6752378	175770 209050
3403	3595	Hod Hill, Dorset		22.0	-245476	6602754	385670 110648
735	757	Salmonsbury Camp, Gloucestershire		23.0	-194654	6779602	417400 220900
357	367	Woodbury, Great Witley, Worcestershire (Woodbu...		23.0	-263690	6850593	374939 264523
136	139	Bozedown Camp, Oxfordshire (Binditch)		23.5	-119597	6710127	464350 178250
3552	3749	Cissbury Ring, West Sussex (Cissbury Camp)		24.0	-42657	6596650	513886 108026
1437	1504	Roulston Scar, North Yorkshire (Sutton Bank; C...		24.5	-134884	7213231	451490 481520
388	404	Ogbury Camp, Wiltshire		26.0	-200007	6646748	414320 138294
443	461	Tedbury Camp, Somerset		26.0	-263616	6663457	374400 148800
373	389	Casterley Camp, Wiltshire (Catterley Banks)		27.5	-204306	6671153	411584 153560
734	756	Willersey Camp, Gloucestershire (Willersey Hil...		28.0	-203808	6807893	411700 238300
410	427	Ebsbury Hill, Wiltshire (Grovely Earthworks)		28.0	-212997	6642126	406160 135380
1234	1276	y Breiddin, Powys (Breiddin Hillfort; The Breid...		28.0	-338884	6931868	329570 314408
89	91	Titterstone Clee, Shropshire		29.6	-289034	6872454	359516 277973
3598	3795	Butser Hill, Hampshire		30.0	-109375	6617356	471527 120313
446	464	Wadbury Camp, Somerset (Wadbury Hillfort)		30.0	-265052	6663609	373500 148900
166	173	Abingdon, The Vineyard, Oxfordshire		33.0	-142388	6740992	449950 197250
95	97	Walbury Camp, West Berkshire		33.0	-162994	6684152	437408 161800
3271	3459	Countisbury Castle, Devon (Shoulsbury; Wind Hill)		35.0	-423647	6662041	274024 149399
3625	3823	Homestall Wood, Kent		35.0	115292	6672762	611760 158930
165	172	Dyke Hills, Oxfordshire (Dike Hills)		46.0	-130542	6735058	457350 193650
738	760	Nottingham Hill Camp, Gloucestershire		48.6	-225551	6791821	398300 228400
3577	3774	Oldbury Camp, Kent		51.5	29679	6671658	558165 156303
751	773	Borough Hill 1, Northamptonshire (Borough Hill)		52.0	-126771	6847138	458877 262696
194	201	Mull of Galloway, Dumfries & Galloway		54.0	-542988	7291829	214381 530735
70	71	Llanymynech Hill, Powys		57.0	-344171	6944572	326479 322138
3402	3594	Hengistbury Head, Dorset		80.0	-195572	6571275	417263 90784
430	448	Ham Hill, Somerset (Hamdon Hill Camp)		84.0	-304545	6611780	348409 116570
3390	3582	Bindon Hill, Dorset		114.0	-248650	6554366	383568 80077

In [93]: `download(['twenty_one_ha_and_over_south'], 'twenty_one_ha_and_over_south')`

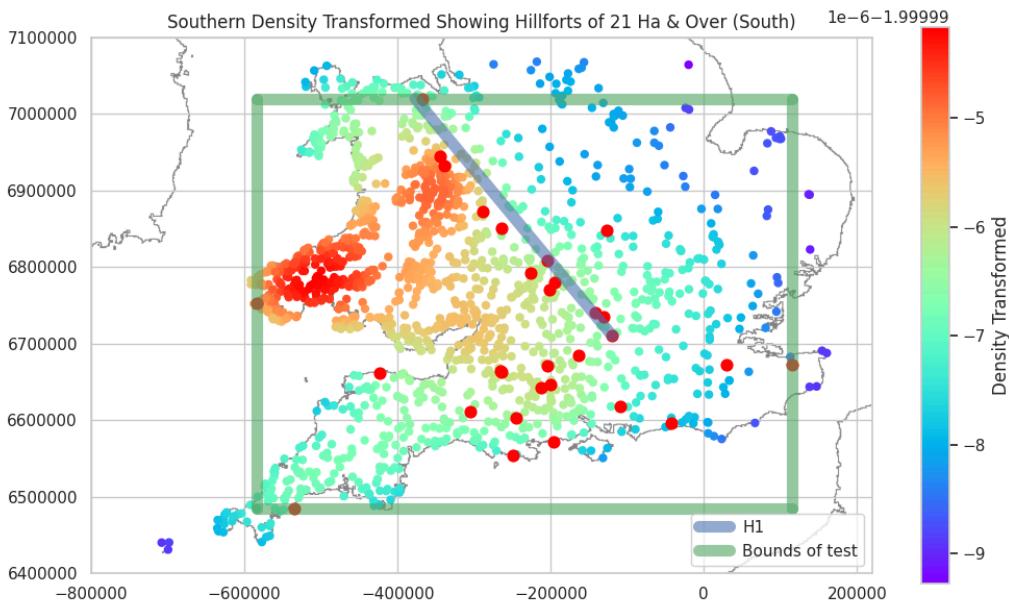
## Hypothesis testing

It is beyond the skills of the author to test the hypothesis of the curved line shown in the image above. This being so, the longer straight section from (1155) Penycloddiau, Denbighshire (Pen y Cloddiau) and (139) Bozedown Camp, Oxfordshire (Binditch) will be tested.

Hypothesis (H1): Hillforts of 21 ha and over are aligned between (1155) Penycloddiau, Denbighshire (Pen y Cloddiau) and (139) Bozedown Camp, Oxfordshire (Binditch).

Null Hypothesis (H0): Hillforts of 21 ha and over, between (1155) Penycloddiau, Denbighshire (Pen y Cloddiau) and (139) Bozedown Camp, Oxfordshire (Binditch), are not aligned.

In [94]: `_ , _ = \
south_density_scatter_lines(cluster_south, enclosing_area_1_21_s, \
 'Southern Density Transformed Showing Hillforts of 21 Ha & Over (South)', \
 False, False, True, True)
#'Southern Density Transformed Showing Hillforts of 21 Ha & Over (South)'`



Middleton, M. 2024, Hillforts Primer

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

The line is 242,511 m (242.5 Km) long.

```
In [95]: penycloddiau = (-367969, 7019842)
bozedown = (-119597, 6710127)
dist = math.sqrt( (bozedown[0] - penycloddiau[0])**2 + (bozedown[1] - \
penycloddiau[1])**2 )
print(f"round(dist)} units")
```

397004 units

```
In [96]: penycloddiau_m = (312888, 367647)
bozedown_m = (464350, 178250)
dist_m = math.sqrt( (bozedown_m[0] - penycloddiau_m[0])**2 + (bozedown_m[1] - \
penycloddiau_m[1])**2 )
print(f"round(dist_m)} m")
```

242512 m

```
In [97]: print(f"Line offset from line {round(dist_m/4.85)} m")
```

Line offset from line 50002 m

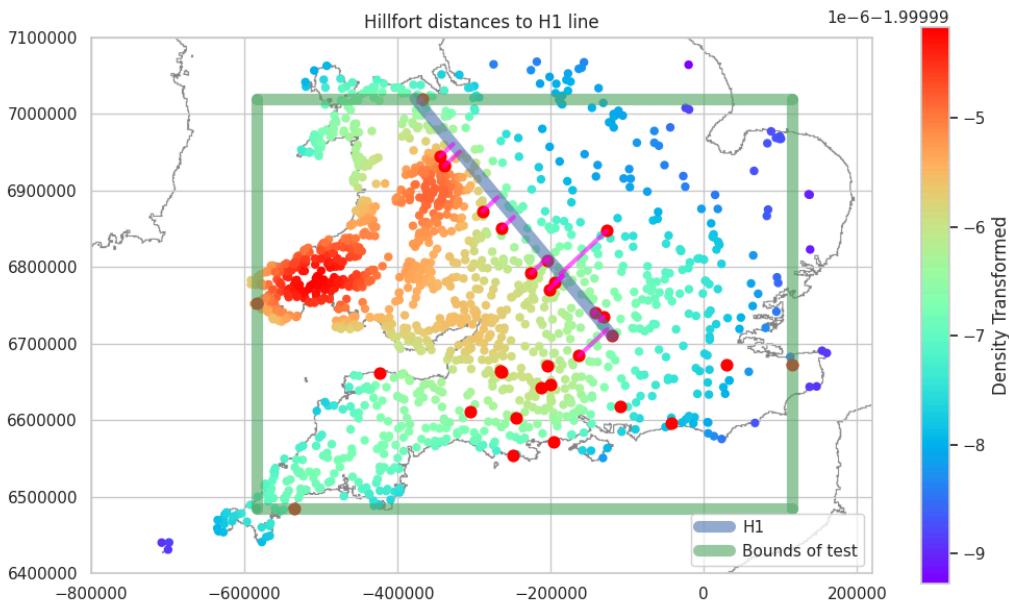
## Random Alignments

To test the hypothesis, 1500 random alignments will be created within the bounds of the forts in the South. The test will measure the perpendicular distance from each hillfort to the line, within a buffer of 50 Km. These distances will be used to calculate the Mean Squared Error (MSE) for all forts falling within the limits of each test alignment. The alignments will all be the same length as the hypothesis line (H1). The perpendicular distance from the line will be classified as the error. The error, for each fort of 21 Ha or above, will be squared and the sum of the squared values, for each line, will be divided by the number of forts that fall within the length of that line. 1500 random alignments will be created but only those with five or more hillforts, within the length of the line will be retained to ensure that the result of the MSE is based on a group of hillforts and not just a couple of individuals. 1500 was chosen to ensure that, at least, 1000 alignments remain after the lines with less than five hillforts have been removed. A single example of a random line is shown below with the perpendicular distances to the line shown.

The resulting histogram for all 1000+ alignments will be a normal curve with the most common MSE toward the centre and the least toward the edges. For the null hypothesis to be rejected, the test alignment (H1 - the blue line) would be expected to be to the far left of the curve with a probability of less than 5%.

Shown below are the hillforts perpendicular to the H1 line within an offset distance of 50 km.

```
In [98]: h1_distance_list, _ = \
south_density_scatter_lines(cluster_south, greater_than_21ha_south, \
'Hillfort distances to H1 line', False, False, \
True, True, True, 0, False, True)
```



Middleton, M. 2024, Hillforts Primer

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

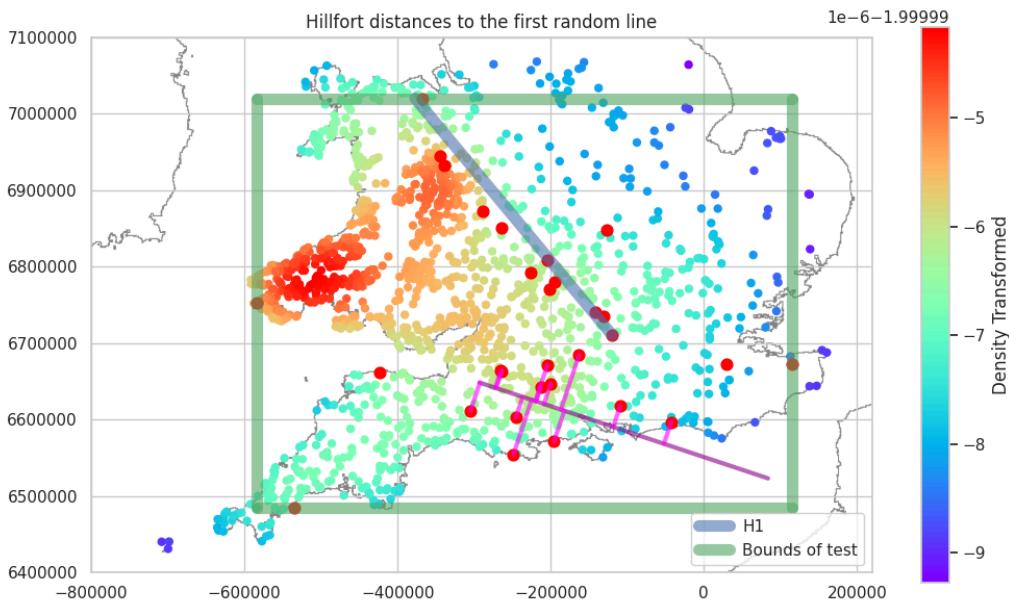
The H1 line has 14 hillforts, withing 50 km, along its length.

In [99]: `len(h1_distance_list)`

Out[99]: 14

A random seed is used to ensure that the random numbers created are the same for each run of this document. Below, hillforts perpendicular to the first random alignment are shown.

In [100...]  
`_ = \  
 south_density_scatter_lines(cluster_south, enclosing_area_1_21_s, \  
 'Hillfort distances to the first random line', \  
 False, False, True, True, True, 1, True)`

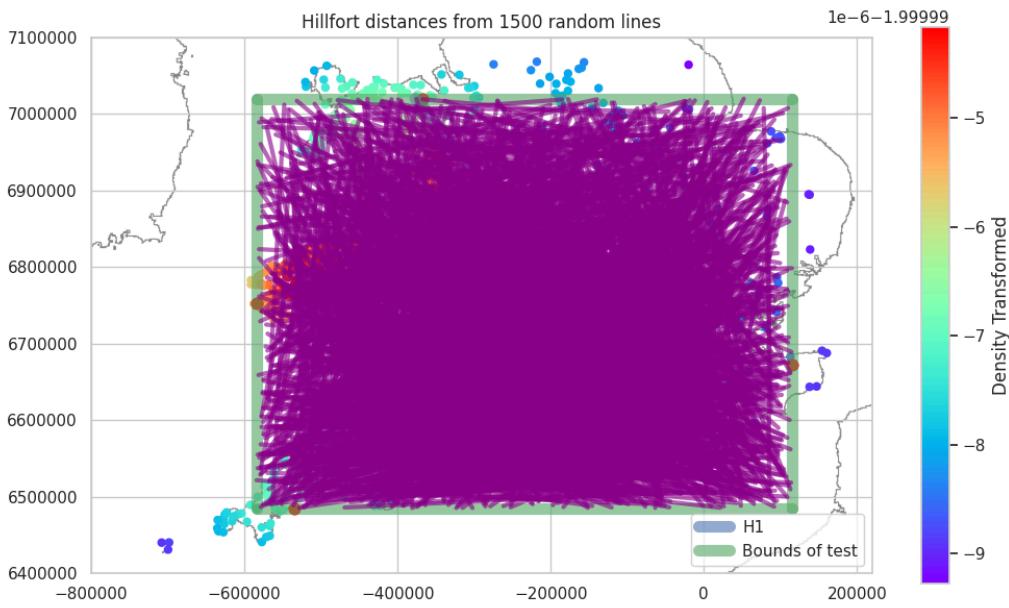


Middleton, M. 2024, Hillforts Primer

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

Next, 1500 random alignments area created within the bounds. The distance to hillforts perpendicular to each line are returned for analysis below.

In [101...]  
`_, distance_lists = \  
 south_density_scatter_lines(cluster_south, enclosing_area_1_21_s, \  
 'Hillfort distances from 1500 random lines', \  
 False, False, True, True, True, 1500)`



Middleton, M. 2024, Hillforts Primer

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

A list of the number of hillforts in each list is created.

```
In [102]: length_list = pd.DataFrame([len(x) for x in distance_lists], columns=["count"])
length_list.head()
```

```
Out[102]: count
0    6
1    9
2   15
3   13
4    9
```

There H1 alignment has 14 hillforts along its length. The 1,000 alignments have alignments with between 0 and 19 hillforts along their lengths. For this analysis we will exclude alignments with less than 5 hillforts along their length to reduce the potential for low hillfort counts skewing the results. We want the mean values to be the mean of groups of hillforts not just isolated individuals. For instance, a single hillfort laying close to the line will give a very low MSE while an alternative alignment with a single hillfort far from the line would give a very high MSE. The influence of individual values is reduced, the larger the sample.

```
In [103]: length_list.value_counts().sort_index()
```

```
Out[103]: count
0      22
1      62
2      75
3     101
4      91
5     100
6     104
7      85
8      92
9     107
10     116
11     114
12     139
13     132
14      69
15      45
16      28
17      10
18       4
19       4
dtype: int64
```

The filtered list contains 1149 alignments where there are at least 5 or more hillforts perpendicular to the line along the length of the line.

```
In [104]: filtered_length_list = [x for x in distance_lists if len(x) >= 5]
len(filtered_length_list)
```

```
Out[104]: 1149
```

First we get the MSE for the H1 line.

```
In [105... def get_mean_sum_of_squares(lst):
    sum_list = 0
    for i in lst:
        sum_list += i**2
    mse = sum_list/len(lst)
    return mse
```

```
In [106... mse_h1 = int(get_mean_sum_of_squares(h1_distance_list))
mse_h1
```

```
Out[106]: 959477730
```

Then we get the MSE for the filtered lines.

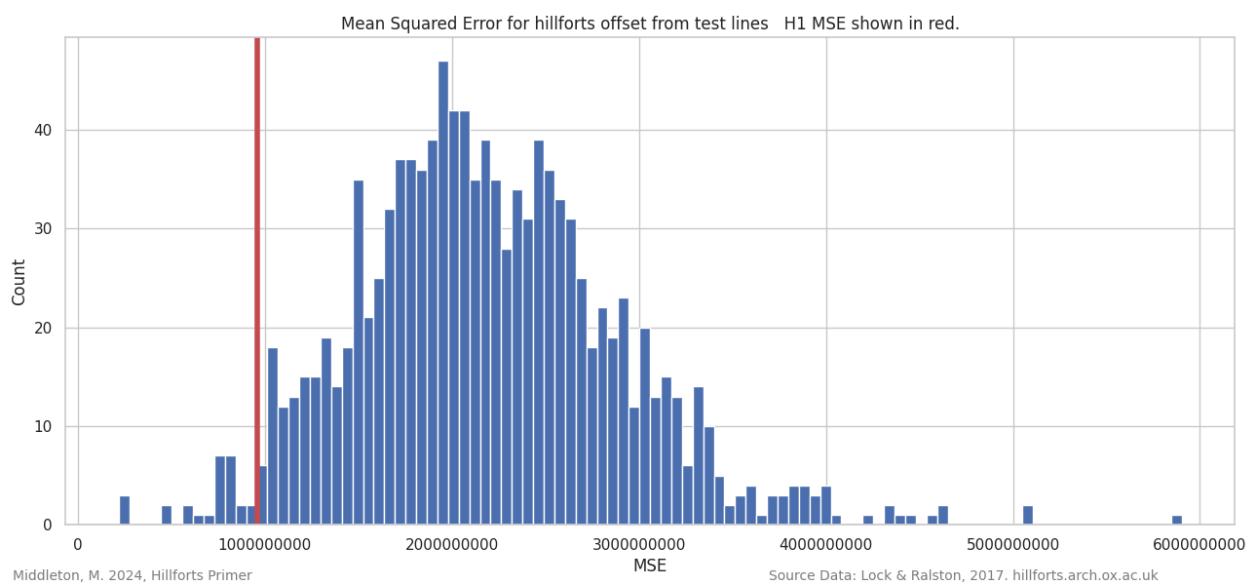
```
In [107... mse_list = [int(get_mean_sum_of_squares(x)) for x in filtered_length_list]
mse_df = pd.DataFrame({'mse':mse_list})
mse_df.head()
```

```
Out[107]:      mse
0   1329023715
1   1832084194
2   3014799598
3   2769225228
4   1299407983
```

## Mean Squared Error (offset)

With these results we can plot a histogram of the MSE data. As anticipated the random alignments have produced (more-or-less) a normal curve. The H1 alignment is toward the far left of the curve.

```
In [108... plot_mse_histogram(mse_list, mse_h1)
```

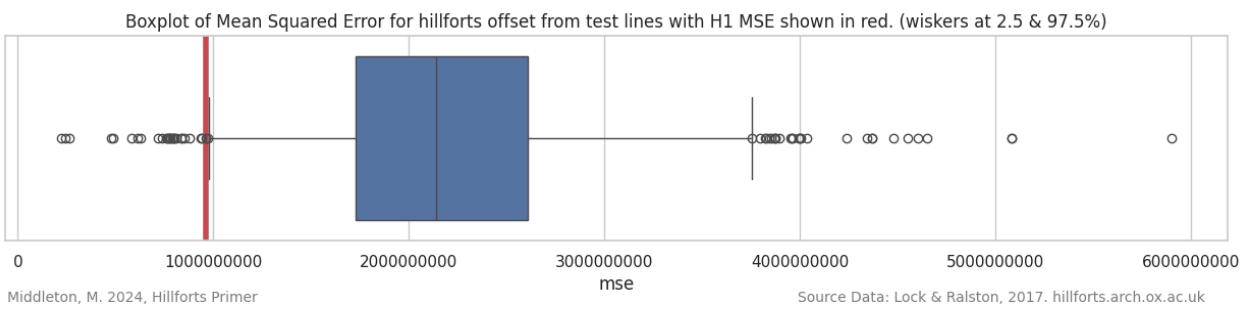


```
In [109... mse_h1
```

```
Out[109]: 959477730
```

A boxplot of the same data shows the interquartile range (IQR) (the blue box) and the first and fourth quartiles bounded by the whiskers. The whiskers, on either side of the IQR are located at 2.5% and 97.5%.

```
In [110... offset_range =
plot_data_range_95(mse_list, "mse", "Boxplot of Mean Squared Error for hillforts offset from test lines with H1 MSE
#"mse", "BoxPlot of Mean Squared Error for hillforts offset from test lines with H1 MSE shown in red. (wiskers at 2.
```



```
In [111]: print(f"H1 MSE = {round(mse_h1/(offset_range[0]/2.5),2)}%")
```

H1 MSE = 2.46%

The H1 line (red) has a probability of 2.46%, just under 2.5%. Below 2.5% or over 97.5% are routinely considered the values above and below which the null hypothesis can be rejected.

Hillforts of 21 Ha and over, falling within 50 km of a straight line between (1155) Penycloddiau, Denbighshire (Pen y Cloddiau) and (139) Bozedown Camp, Oxfordshire (Binditch), are aligned.

## Save Figure List

```
In [112]: if save_images:
    path = os.path.join(IMAGES_PATH, f"fig_list_{part.lower()}.csv")
    fig_list.to_csv(path, index=False)
```