

Radon Distribution in Northern Ireland - A Geological Perspective The How-to Guide

Supplement to the GitHub repository: <https://github.com/zsmilliepy/RadonNI>

Link to the radon_bedrock.shapefile: https://ulster-my.sharepoint.com/:f:/r/personal/smillie-z_ulster_ac_uk/Documents/egm722/radon_bedrock_shapefile?csf=1&web=1&e=Ev0OIH

I. Introduction

This document provides instructions about the python codes as part of the GitHub repository: <https://github.com/zsmilliepy/RadonNI>. The document includes background about the importance of the data processed, instructions on what software and accounts required to access, and what the script do.

II. Backgrounds

Radon is a naturally occurring radioactive gas with no smell, colour or taste (WHO, 2021). This radioactive gas is produced from the natural radioactive decay of uranium-238, uranium-235 and thorium-232 (Green et al., 2009) and can be found in all rocks and soils, with various degrees (Otton, 1992). Radon escapes from the ground into the air, where it decays and produces other radioactive particles.

Where generated and spread into the air outdoors, radon quickly dilutes to low concentrations and is generally not considered a threat to health. However, the gas may be found in high concentrations in indoor environments, such as homes and workplaces, with higher concentrations in areas with minimal ventilation (Appleton and Adlam, 2015). Hence, radon could be a significant cause of lung cancer and is estimated to cause between 3% to 14% of all lung cancers in some countries (WHO, 2021).

The core data used in this report are the radon potential classes in Northern Ireland, combined with data about geological and topographical features in the area. These data are sourced from the following institutes: Geological Survey of Northern Ireland (GSNI, 2021), the Northern Ireland Environment Agency (NIEA, 2015) and the Ordnance Survey of Northern Ireland (OSNI, 2020).

III. Aims

The aim of this study is threefold: 1) understand the geographical distribution of the radon potential levels in Northern Ireland; 2) connect the radon distribution to bedrock geology; 3) investigate the relationship between the radon distribution and topography in the area.

IV. Raw data structure

This project benefits from 4 primary datasets; radon potential classes, Northern Ireland counties map, bedrock geology maps and digital terrain model (DTM) raster image (Table I). Detailed instruction on how to access these data are given in the programs and setup section.

Table I: Dataset to be used in the current programme.

No	Dataset	Format	Source
1	Radon Potential Classes	csv	PHE-GSNI-BGS
2	Counties Shapefile	shapefile	OSNI
3	Bedrock Geology Shapefile	shapefiles	BGS
4	Digital Terrain Model	GeoTIFF	OSNI

These raw data are used in the programme to explore and investigate radon, counties and geological and topographical features. They are also used to generate multiple datasets and plot various relationships of the geographical and geological features connected to the radon potential levels.

Radon Potential classes are in the form of spreadsheet datafile (comma-separated values, CSV), adopted after the PHE-GSNI-BGS digital Radon Potential Dataset for Northern (Green et al., 2009). Radon data are classed based on the probability that individual property in Northern Ireland is at or above the Action Level for radon and classified into six probability bands 1 to 6. Table I shows the various radon potential classes and the estimated percentages of dwellings at risk of exposure to radon levels exceeding the Radon Action Level (Appleton and Adlam, 2015).

Table I: Radon Potential Classes in Northern Ireland and their corresponding percentage bands (Appleton and Adlam, 2015)

Radon Potential Class	Estimated percentage of dwellings exceeding the Radon Action Level (Nominal percentage band)	Estimated percentage of dwellings exceeding the Radon Action Level (Actual percentage band)
1	0 - 1	0 to 0.99999
2	1 - 3	1 to 2.99999
3	3 - 5	3 to 4.99999
4	5 - 10	5 to 9.99999
5	10 - 30	10 to 29.99999
6	30-100	30 to 100

V. Programme setup

1) Programs and Setup

To read and run the codes in this programme, you need to install git, GitHub Desktop, and Anaconda in this particular order.

i. Installing Git and GitHub

Git is a version control software that facilitates tracking changes in files, sharing, and collaborating on code (GitHub Guides, 2021). To access the code for the current project, you need to create a GitHub account: <https://github.com/>, and download Git (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>) and GitHub desktop (<https://desktop.github.com/>).

You can then fork, clone and download (see yellow arrows in Fig. 1, respectively) the repository for the current project (<https://github.com/zsmilliepy/RadonNI>). Save the repository to a local folder on your own computer and note the folder path.

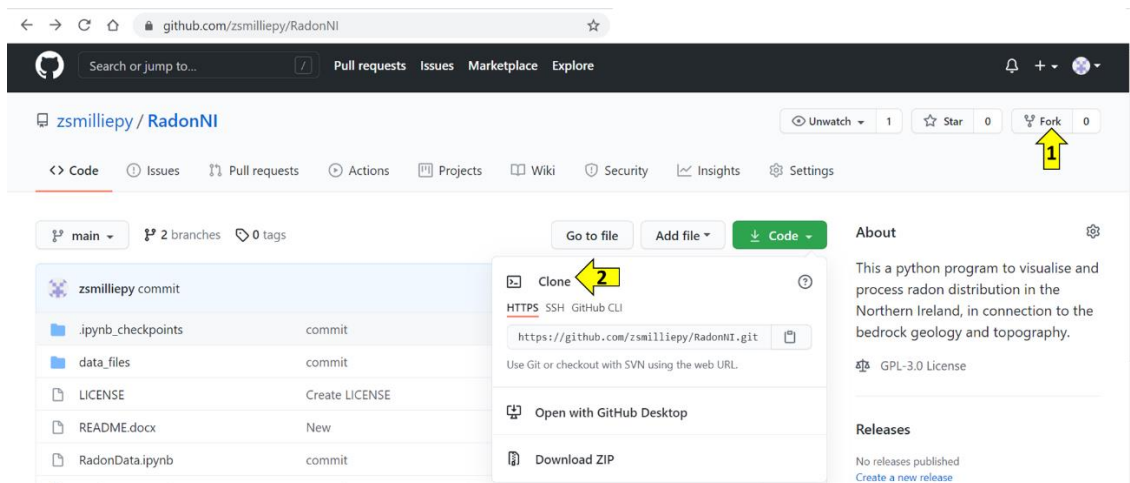


Fig. 1: A screenshot of GitHub showing essential functions: (1) fork and (2) clone the repository required for the current project.

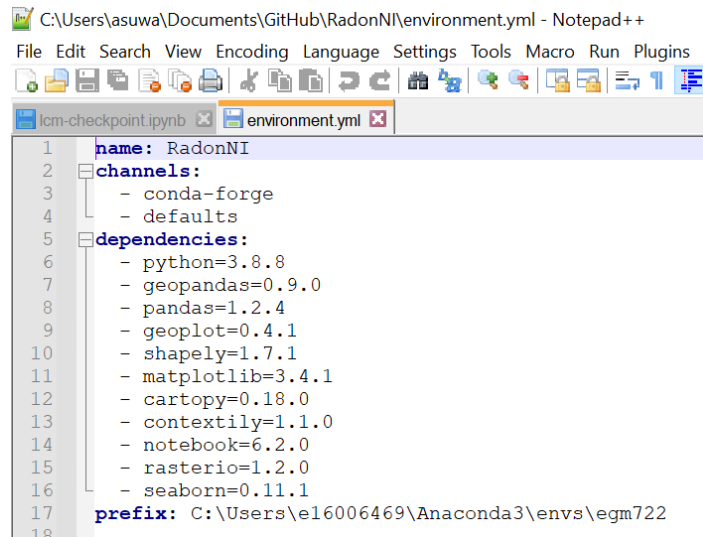
ii. Installing Anaconda

Anaconda is a toolkit that enables you to keep track of and manage open-source packages and libraries. For more info, see <https://docs.anaconda.com/anaconda/install/>.

To run and manage the code provided, you need to install the open-source Individual Edition (<https://www.anaconda.com/products/individual>). **Please check the system requirements for your computer before downloading the Anaconda installer and carefully follow the installation wizard instructions.**

iii. Environment files

Creating different programming environments enables the user to keep track of and manage the python packages required for the code they are working on. An “environment.yml” file contains a list of the packages and is an essential tool to find and install various packages. The cloned repository you downloaded contains one environment file called “environment.yml”. To get an idea about this kind of file structure, open this file in a text editor (NotePad, or Notepad++, **NOT MS Word!**). You can see a list of packages that would support the present programme (Fig. 2).



```

1 name: RadonNI
2 channels:
3   - conda-forge
4   - defaults
5 dependencies:
6   - python=3.8.8
7   - geopandas=0.9.0
8   - pandas=1.2.4
9   - geoplots=0.4.1
10  - shapely=1.7.1
11  - matplotlib=3.4.1
12  - cartopy=0.18.0
13  - contextily=1.1.0
14  - notebook=6.2.0
15  - rasterio=1.2.0
16  - seaborn=0.11.1
17 prefix: C:\Users\el6006469\Anaconda3\envs\egm722
18

```

Fig. 2: An example of an “environment.yml” file

Once the program is installed, open the Anaconda Navigator from the Start menu. Please follow the instructions below to import the environment file (also illustrated in Fig. 3).

1. Click on the Environments tab on the left-hand side of the screen. You should see new parts of the windows appear (A and B in Fig. 3)
2. At the bottom of the environments list, click the “Import” button.
3. An “Import” window will open. Click the folder icon to navigate to the repository folder you have saved earlier, select the “environment.yml” file, and click open. You would notice that the repository file name automatically appears next to “Name”.
4. The final step is to click on “Import.”

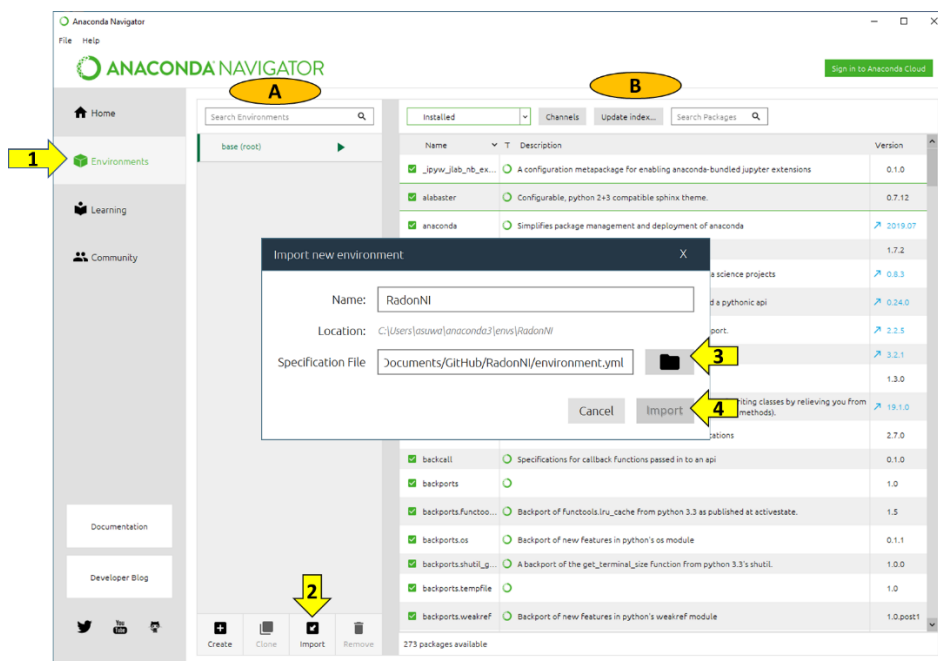


Fig. 3: Steps to import an “environment.yml” file in Anaconda.

iv. Accessing CMD.exe and Jupyter Notebook:

You may need to install CMD.exe and Jupyter Notebook (marked by red squares in Fig. 4). CMD.exe Prompt, a command-line interpreter, enables you to directly launch a Windows Command Prompt with your RadonNI environment loaded. At the same time, Jupyter Notebook (Jupyter, 2021a) is a web application that allows you to create and share documents containing live code, visualisations and narrative text (as [Markdown cells](#)).

First, you need to make sure you are using the correct environment file. The automatic setup of Anaconda prompt, if you run it from the Start Menu, to load the default (base) environment (although, you can manipulate that using file/preferences). Hence, you will need to switch environments to the RadonNI by clicking on the dropdown list next to “Applications on” (see the yellow arrow 2 in Fig. 4).

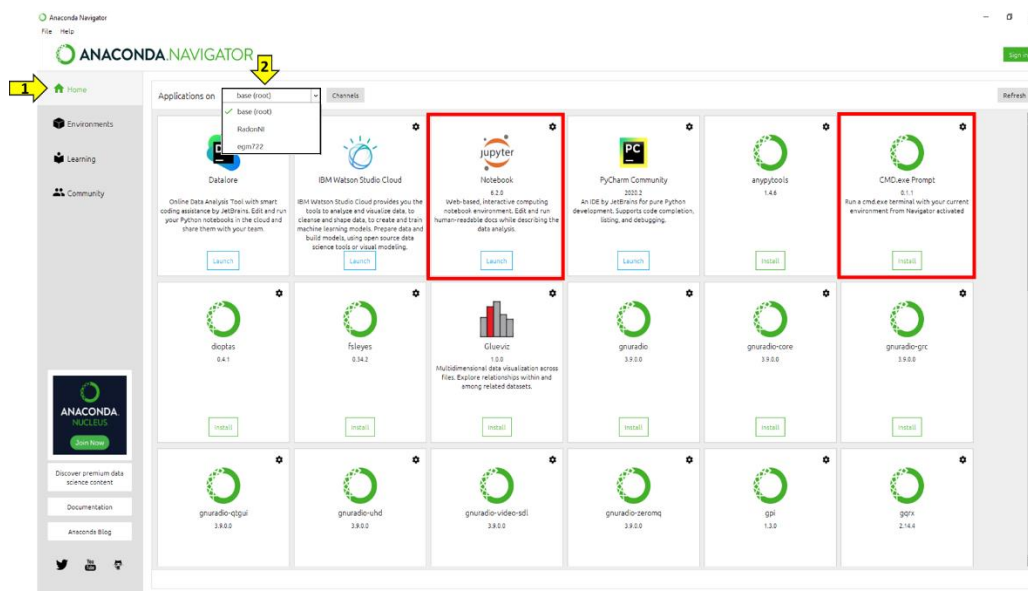


Fig. 4: Anaconda Navigator.

v. Install packages:

To install packages, you can use Anaconda Navigator (full details are here: <https://docs.anaconda.com/anaconda/navigator/tutorials/manage-packages/>) or CMD.exe. To use the latter, launch the command prompt from the Anaconda Navigator. Access the repository folder on your computer (You may find guidance here: <https://www.digitalcitizen.life/command-prompt-how-use-basic-commands/>) and type the code below (here we are giving an example of installing “rasterio”, a package that reads and writes geospatial raster datasets, you may replace the word rasterio with the name of the library or package you desire. This may take few minutes. When prompted, type y and press enter.

```
conda install -c conda-forge rasterio
```

Now, you are ready to work on the RadonNI repository. You may choose to open the script using an IDE installed in your computer or run it using a Jupyter notebook. A detailed description of the scripts and what they do, are provided in the next section with references to the sections as they appear in the Jupyter notebook: (<https://github.com/zsmilliepy/RadonNI/blob/main/RadonData.ipynb>).

VI. Methods (Programme Scripts)

1) Import the required libraries

A list of the packages and libraries required for the script is provided (as a [markdown](#) section) at the top of the notebook. The user needs to ensure all these libraries and packages are installed (see the previous section). Additional notes are also provided when these packages are used within the script.

The script section to follow contains the code to import these libraries and packages. Note that the use of the **%matplotlib notebook** results in producing interactive plots embedded within the notebook. This code should be set before importing **pyplot** (Medium One, 2018).

```
%matplotlib notebook

import pandas as pd
import geopandas as gpd
import geoplot as gplt
from shapely.geometry import Point, LineString, Polygon
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
from matplotlib_scalebar.scalebar import ScaleBar
import cartopy as cp
from cartopy import config
from cartopy.feature import ShapelyFeature
import cartopy.crs as ccrs
import contextily as ctx
import seaborn as sns
import rasterio as rio
import pydoc
```

2) Defining figure formats

The following code helps the user to adjust the format of the figures and maps produced. The code includes setting the font size as 8 (the user may modify) and adjusting the position and format of map features such as legend and scalebar. Note the use of the comments (green italic colour) starting with “#”. These are short comments to help the reader understanding the codes.

Two types of supports are used here to explain the codes. These are comments and docstrings. Comments are descriptive texts, usually short text, used to explain what the line of code or expression does (GeeksforGeeks, 2021). In the meantime, a docstring is a Python documentation string used in the class, module, function, or method definition (Python, 2021). Docstrings are used to understand the functionality of the more significant part of the code, i.e., the general purpose of any class, module, or function (Datacamp, 2020). Doctsting can be short “One-Line Docstring”, as in the code below.

```

plt.rcParams.update({'font.size': 8}) # adjust the font size for the plots to be size 8

plt.ion() # make the plotting interactive

# generate matplotlib handles to create a legend of the features we put in the maps.
def generate_handles(labels, colors, edge='k', alpha=1):

    """ controllers to define handles to create a legend of the features we put in our map """

    lc = len(colors) # get the length of the color list
    handles = []
    for i in range(len(labels)):
        handles.append(mpatches.Rectangle((0, 0), 1, 1, facecolor=colors[i % lc], edgecolor=edge, alpha=alpha))
    return handles

# create a scale bar of length 20 km in the upper right corner of the map
# adapted this question: https://stackoverflow.com/q/32333870, answered by: https://stackoverflow.com/a/35705477

def scale_bar(ax, location=(0.92, 0.95)):
    llx0, llx1, lly0, lly1 = ax.get_extent(ccrs.PlateCarree())
    sblx = (llx1 + llx0) / 2
    sbly = lly0 + (lly1 - lly0) * location[1]

    tmc = ccrs.TransverseMercator(sblx, sbly)
    x0, x1, y0, y1 = ax.get_extent(tmc)
    sbx = x0 + (x1 - x0) * location[0]
    sby = y0 + (y1 - y0) * location[1]

    plt.plot([sbx, sbx - 20000], [sby, sby], color='k', linewidth=9, transform=tmc)
    plt.plot([sbx, sbx - 10000], [sby, sby], color='k', linewidth=6, transform=tmc)
    plt.plot([sbx - 10000, sbx - 20000], [sby, sby], color='w', linewidth=6, transform=tmc)

    plt.text(sbx, sby-4500, '20 km', transform=tmc, fontsize=10)
    plt.text(sbx-12500, sby-4500, '10 km', transform=tmc, fontsize=10)
    plt.text(sbx-24500, sby-4500, '0 km', transform=tmc, fontsize=10)

```

3) Reading the datafiles

Here we ask the program to read the data of the radon, counties and bedrock geology. Note that all the dataset used and generated in this programme set inside a folder called “data_file” in the repository.

```

# Read radon data of the Northern Ireland
radon_table = pd.read_csv('data_files\RadonNI.csv')
""" read csv file """

counties_orig = gpd.read_file('data_files/Counties.shp') # load the Counties shapefile
""" read shapefile """

bedrocks = gpd.read_file('data_files/NIbedrocks.shp') # load the bedrock geology layer of Northern Ireland

```

For each set of data, a group of codes is used to understand and visualise the dataset, including functions such as `info()`, `describe()`, `.head`. This is a good practice to carry out such codes before carrying out any geospatial analyses or visualisation. This ensures the data were read and loaded correctly into the program and help in selecting the appropriate attributes for subsequent analyses.

4) RADON data

In this section, we will run few codes to understand the nature and structure of the radon data. Also, we will attempt converting these data into geospatial data (shapefile). The first set of codes below should help to visualise the data structure and radon distribution. For example, executing the last line of the code should display a histogram plot (with a specific bin interval) of the radon classes.

```

radon_table.shape # display the data structure
radon_table.info() # More information about the data.
radon_table.head(10) #show the top 10 rows with the column headings
radon_table.describe() # summary statistics of all the data.
radon_table["class"].describe() # Summary statistics of the radon data only. Radon classes ranges from 1 to 6.
radon_table["class"].describe().round(0) # descriptive statistics of the radon classes, round numbers to nearest integer.
radon_table.hist(column='class', bins=[0.5,1.5,2.5,3.5,4.5,5.5,6.5]); # Histogram plot of the radon classes.

```

5) Geospatial Analysis of the RADON data

The codes in this section are used to transform the radon spreadsheet data into geospatial points. The first task is to set the coordinates to the Irish National Grid TM65 and plot the data to visualise/check their distribution.

```

# set the coordinate type (to epsg:29902; i.e. TM65 / Irish Grid) and plot the radon data with respect to x, y
pts = [Point(row['x'], row['y']) for id, row in radon_table[['x', 'y']].iterrows()]
""" Assign the xy coordinates from the radon data """
pts = gpd.GeoSeries(pts, crs='+init=epsg:29902')
pts.plot(); # plot the radon data with respect to their coordinates

```

It is always a good practice to check the coordinates, using cartopy - crs, before running any further codes.

```
pts.crs # check the coordinate systems, i.e. check if the previous code worked!
```

Alternative way to assign the coordinates to the data through running the single line code below (GeoPandas, 2021a), then check the coordinates again.

```

# another way to assign the coordinate system (perhaps more simple)
radon_gdf = gpd.GeoDataFrame(radon_table, geometry=gpd.points_from_xy(radon_table.x, radon_table.y), crs='+init=epsg:29902')
radon_gdf.crs # check the coordinate system of the new dataset (radon_gdf)

```

The new georeferenced data can be saved as an ESRI shapefile using “to_file” and can be loaded back in the programme using “read_file” (GeoPandas, 2021b).

```

radon_gdf.to_file('data_files/RadonNI.shp') # create a new shapefile of the radon data
radon = gpd.read_file('data_files/RadonNI.shp') # load the radon shapefile

```

The code below helps plotting the shapefile using a categorised symbology based on the value of each radon class.

```

# Display the radon distribution, symbology is based on the "class" attribute/column.
fig, ax = plt.subplots(figsize=(5, 5))
""" Design the plot """
radon.plot(column='class',
            categorical=True,
            legend=True,
            ax=ax,
            markersize=3)
""" identify the column to plot and add legend """
leg = ax.get_legend() # Adjust legend location
leg.set_bbox_to_anchor((1,1))
ax.set_axis_on() # Add map frames/axes
plt.show() # show the plot of the data

```

6) Radon potential level in various counties

A further step is to understand the geographical distribution of the radon classes using the codes below. The function groupby() is used to select data by attributes.

The codes benefits from the use of spatial join function (gpd.sjoin()) (Tenkanen, 2018). Note the use of “inner” as tool for intersection, i.e. join radon data that intersect with the counties polygons (GeoPandas, 2021c).

```

counties_orig.shape           # Table of attributes of the "counties" shapefile is made up of 6 rows (6 counties) and five columns
counties_orig.columns         # read the column headings
divisions = counties_orig['CountyName'].tolist()           # Get a list of the counties; 6 counties
print(divisions)
counties_orig.head()         # display the data.
# sort the counties according to their area (in km2), starting with the largest.
counties_orig.groupby("CountyName").mean().sort_values(by = "Area_SqKM", ascending = False)

f, ax = plt.subplots(1, figsize=(5, 5), edgecolor="blue", facecolor="yellow") # to show a basemap background
counties_orig.plot(color='k', alpha=0.5, ax=ax)
ctx.add_basemap(ax)
plt.show('CountyName')
f.suptitle('Northern Ireland Administrative divisions')      # to add a title to the map
plt.show()

counties_orig.crs             # check the coordinate system of the counties shapefile, it is EPSG:4326
counties = counties_orig.to_crs(epsg=29902)                 # change the CRS to epsg=29902, to match the radon data coordinates
counties.crs                 # check the coordinates again
# Plot the counties again. Note the difference in the coordinates numbers compared to code 23.
# Note here that we used "edgecolor" and "facecolor" to manipulate the appearance of the polygons
counties.plot(figsize=(5,5), edgecolor="purple", facecolor="None"); # plot the counties in the Northern Ireland
# Plot the counties, symbology follows the county name (attribute "CountyName")
counties.plot('CountyName', legend=True, figsize=(8, 8));
print(radon.crs == counties.crs) # check if the CRS of the radon and counties data are the same

radon_counties = gpd.sjoin(counties, radon, how='inner', lsuffix='left', rsuffix='right') # join the two datasets
radon_counties # display the new dataset, point datasets combining the radon and the county where each point is located
radon_counties.shape # structure of the new dataset; 13855 rows and 10 columns
radon_counties.info() #type of data within the new dataset
# Distribution of various classes within the counties.
# Each county contain the 6 radon classess, except County LONDONDERRY where class 6 is absent.
# Note that we are using Seaborn package
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(8, 5) # set the size of the graph
sns.violinplot(x = "CountyName", y = "class", data = radon_counties) # use seaborn to create a violin plot
fig.savefig('data_files/Radon_classes_Counties.png') #save the figure as an image
# assign the CRS to the new dataset (joined radon - counties dataset)
radon_counties_gdf = gpd.GeoDataFrame(radon_counties, geometry=gpd.points_from_xy(radon_counties.x, radon_counties.y),
crs='init=epsg:29902')
radon_counties_gdf.crs # check the CRS
# create a new shapefile combining the radon and the counties data
radon_counties_gdf.to_file('data_files/radon_counties.shp')
radon_counties = gpd.read_file('data_files/radon_counties.shp') # load the radon_counties shapefile
# show a brief description of the new shapefile. County Tyrone is the largest in data count
radon_counties.CountyName.describe()
radon_counties # display the attributes of the new shapefile
radon_counties.groupby(['class']).count() # display the data counts of each class
# Pie plot of each class count in the Northern Ireland. Class 1 is the top while class 6 is the lowest
radon_counties.groupby(['class']).count().plot(kind='pie', y='CountyName', figsize=(5, 5));
# summarize the radon distribution within each County.
print(radon_counties.groupby(['CountyName', 'class'])['class'].count())
radon_counties.groupby("CountyName")["class"].mean() # Average radon class in each county
# Average radon class in each county, sorted by the highest.
# County Tyrone shows the highest radon potential level, with an average of 3.68
radon_counties.groupby("CountyName").mean().sort_values(by = "class", ascending = False)

```

Optional task is to subset the dataset to focus on one county using the code “==”, for example selecting County Tyrone, as it shows the highest radon potential bands.

```

# subset the data, i.e. create a new dataset representing the radon distribution in the County Tyrone
county_tyrone = radon_counties[radon_counties.CountyName == "TYRONE"]
len(county_tyrone) # returns the number of radon data (data count) in county Tyrone
county_tyrone      # display the Tyrone dataset
# show the maximum and minimum class values in County Tyrone
county_tyrone.loc[[county_tyrone["class"].idxmax(), county_tyrone["class"].idxmin()]]
# plot the radon distribution in County Tyrone.
# Higher radon classes in the northern parts of the county compared to the southern parts.
county_tyrone.plot(column='class', cmap=None, legend=True, figsize=(5, 5));
# calculate the class percentages. Note that total data count is = 3184
tyrone_class_percent = (county_tyrone.groupby(['class'])['class'].count() * 100 / 3184)
tyrone_class_percent
# plot the classes. Note that Class 4 is the highest in the county
ax = tyrone_class_percent.plot.bar(x='class', y='%', rot=0)
county_tyrone.crs      #check CRS
county_tyrone.to_file('data_files/radon_tyrone.shp') # save Tyron data to a new shapefile

```

7) Bedrock Geology

Similar to processing the radon data, codes are used to understand the bedrock geology data, including checking the coordinate system using “.crs”. The last two codes are used to view the areal distribution of each bedrock type in the area.

```

bedrocks.shape      # datasets are made up of 2263 rows (records) and 5 columns (attributes)
bedrocks            # Read the bedrocks dataset
bedrocks['UnitName'].describe() # view the number of bedrock types
# Plot the bedrock dataset with symbology and legend
bedrocks.plot(column='UnitName', edgecolor="black", cmap=None, legend=True, figsize=(8, 8))
bedrocks.crs        # check coordinates
bedrocks['UnitName'].value_counts() # show the number of polygons representing each bedrock type
# Total area of each lithology in km2. Note that basalt covers the largest area
print(bedrocks.groupby(by=["UnitName"])["area"].sum() / 1000000)

```

Using spatial join codes, the radon and bedrocks can be combined into one ESRI shapefile. The code `fig.savefig` is used here to export the figure/map into a user friendly image (.png) file (Matplotlib, 2021).

```

# join the radon and bedrock datasets
radon_bedrocks = gpd.sjoin(bedrocks, radon, how='inner', lsuffix='left', rsuffix='right')
radon_bedrocks.to_file('data_files/radon_bedrock.shp')
radon_bedrocks      # read the new dataset
# distribution of various classes within each bedrock type.
# code adapted from: https://stackoverflow.com/questions/31594549/how-do-i-change-the-figure-size-for-a-seaborn-plot
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(8, 4) # set the size of the graph
sns.violinplot(x = "UnitName", y = "class", data = radon_bedrocks)
fig.savefig('data_files/radon_bedrocks.png') #save the figure as an image

```

8) Digital Terrain Model Data

Differences in bedrock types lead to the formation of different landforms and terrains. For example, hard rock could be resistant to erosion and form high mountains, while weaker rock can be eroded easily and form lowlands. This aspect is worth investigating to detect the relationship between the radon potential distribution and the land topography. The first few codes are employed to understand the topographical data, i.e. digital terrain model (DTM).

```

dtm = rio.open('data_files/ni_dtm.tif') # Open the raster and store metadata
print('{} opened in {} mode'.format(dtm.name, dtm.mode)) #View raster information
print('image has {} band(s)'.format(dtm.count))
print('image size (width, height): {} x {}'.format(dtm.width, dtm.height))
print('band 1 datatype is {}'.format(dtm.dtypes[0])) # note that the band name (Band 1) differs from the list index [0]
print(dtm.bounds)
print(dtm.crs)

```

Now, the DTM data can be sampled for every radon data point using `dtm.sample(coords)`.

```

pts = radon          # Read points from shapefile
pts = pts[['x', 'y', 'class', 'geometry']]
pts.index = range(len(pts))
coords = [(x,y) for x, y in zip(pts.x, pts.y)]
# Sample the raster at every radon point location and store values in DataFrame
pts['Value'] = [x[0] for x in dtm.sample(coords)]
zip(pts.geometry.x, pts.geometry.y) # pairing the xy with the geometry
pts['Value'] # show the raster values (elevation in m)
pts.crs # show the coordinates of the dataset
pts.to_file('data_files/radon_dtm.shp') # create a new shapefile of the radon and elevation data
radon_dtm = gpd.read_file('data_files/radon_dtm.shp') # load the shapefile
radon_dtm.plot("Value", legend=True, figsize=(5, 5), markersize=5); # Show the elevation distribution
radon_dtm # show the new dataset
radon_dtm.hist(column='Value', bins=[0,1000,2000,3000,4000,5000,6000,7000]); # histogram of elevation data

```

Final step is to display the relationship between the radon classes and elevation using `groupby()`.

```

# Show the average ground elevation corresponding to each radon class
DTM_mean=radon_dtm.groupby("class").mean().sort_values(by = "Value", ascending = False)
DTM_mean

```

VII. Results

The codes provided show that the radon potential classes are dominated by low values (class 1, Fig. 5 A) followed by class 4 (5 – 10 % of dwellings exceed the Radon Action Level). In general, low classes are dominant in the eastern and northern parts of Northern Ireland, while higher classes can be found in the southern and western regions (Fig. 5 B).

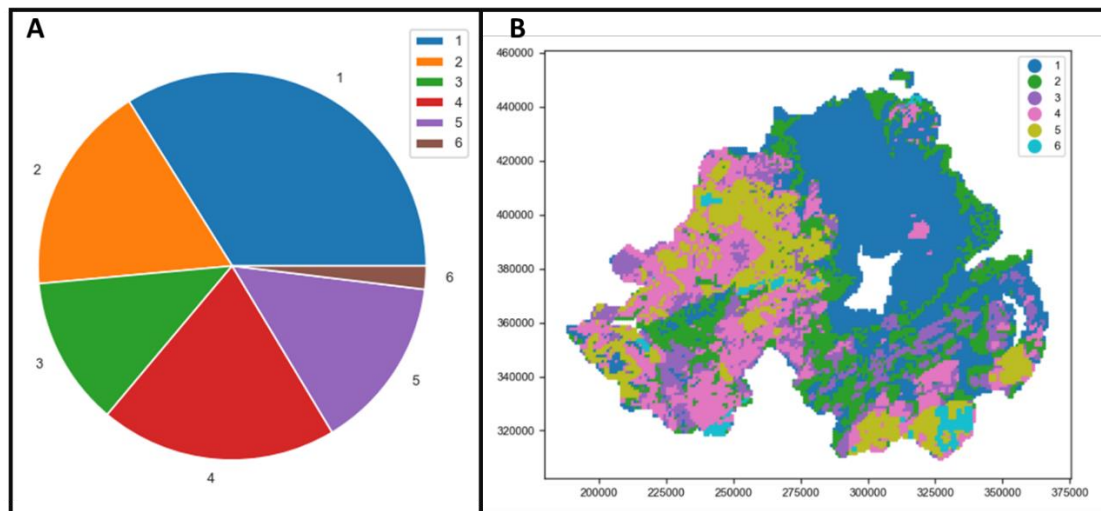


Fig. 5: A) Spatial distribution of radon potential class. B) Pie chart of the radon data.

County Tyrone shows the highest average radon potential class values (3 – 4), while County Antrim shows the lowest mean (1 – 2, Table 3). Radon classes estimated over the granitic, gabbroic and acid volcanic bedrock show higher mean radon potential classes compared to the sedimentary or basaltic bedrocks (Fig. 6). In the meantime, high radon potential classes are more associated with high elevations in the area (Table 4).

Table 3: Average radon potential class of Northern Ireland counties.

COUNTY_ID	Area_SqKM	OBJECTID	index_righ	class	x	y
CountyName						
TYRONE	6.0	3265.796622	1.0	7004.829774	3.677136	253130.025126 374022.613065
FERMANAGH	4.0	1850.832538	4.0	11370.660636	3.533443	223566.885965 344410.635965
DOWN	3.0	2491.238606	6.0	10890.040481	2.786774	333823.046092 347807.815631
LONDONDERRY	5.0	2118.316853	5.0	2655.370722	2.514259	273890.209125 409728.136882
ARMAGH	2.0	1327.228438	3.0	12102.815217	2.456522	294173.913043 339005.434783
ANTRIM	1.0	3097.847750	2.0	3355.223986	1.256081	316042.567568 405103.378378

Fig. 6: Radon potential classes sorted according to the average elevation of their associated grounds.

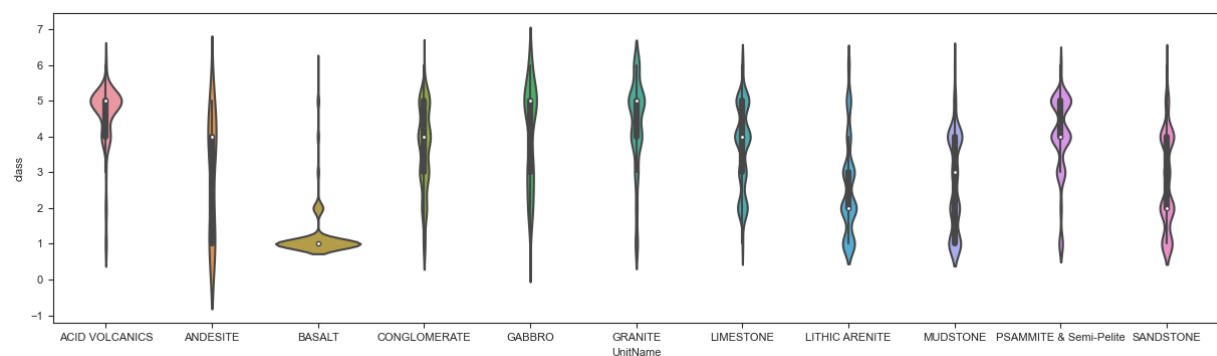


Fig. 6: Violin plot of radon class distribution among various bedrock in Northern Ireland.

Table 4: Average elevation (Value) corresponding to each radon class,

	x	y	Value
class			
5	265557.294174	368956.427540	1603.001926
6	292583.333333	344150.000000	1418.330000
4	258623.614958	364056.440443	1213.928324
3	278087.171053	358404.605263	1085.328947
1	305618.378812	392035.112360	1006.031701
2	292103.324518	368635.247450	950.794862

VIII. Troubleshooting

It is advisable to run the codes in the notebook one by one. Due to the large data size, some graphs may take few minutes to appear correctly. The user may amend the plot dimensions to adapt to the PC graphic settings. General guidance on resolving common problems can be found on the Jupyter troubleshooting page (Jupyter, 2021b).

If the user decide to for the resposirotly and make use of pushing and pulling changes (commits) between the repository folder on the computer and the repository on GitHub, they may benefit from the instructions here on the [DataCamp page](#) (Datacamp, 2019)

IX. References

- Appleton, J.D., Adlam, K.A.M., 2015. User Guide for the PHE-GSNI-BGS Joint Radon Potential Dataset for Northern Ireland. (No. OR/13/048), British Geological Survey Open Report. British Geological Survey.
- Datacamp, 2020. (Tutorial) Docstrings in Python [WWW Document]. DataCamp Community. URL <https://www.datacamp.com/community/tutorials/docstrings-python> (accessed 5.10.21).
- Datacamp, 2019. GIT Push and Pull [WWW Document]. DataCamp Community. URL <https://www.datacamp.com/community/tutorials/git-push-pull> (accessed 5.28.21).
- GeeksforGeeks, 2021. Jupyter notebook Tips and Tricks [WWW Document]. URL <https://www.geeksforgeeks.org/jupyter-notebook-tips-and-tricks/>
- GeoPandas, 2021a. Managing Projections — GeoPandas 0.9.0 documentation [WWW Document]. URL https://geopandas.org/docs/user_guide/projections.html (accessed 5.11.21).
- GeoPandas, 2021b. Reading and Writing Files — GeoPandas 0.9.0 documentation [WWW Document]. URL https://geopandas.org/docs/user_guide/io.html (accessed 5.11.21).
- GeoPandas, 2021c. Merge, join, concatenate and compare — pandas 1.2.4 documentation [WWW Document]. URL https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html (accessed 5.11.21).
- GitHub Guides, 2021. Git Handbook · GitHub Guides [WWW Document]. URL <https://guides.github.com/introduction/git-handbook/> (accessed 5.14.21).
- Green, B.M.R., Larmour, R., Miles, J.C.H., Rees, D.M., Ledgerwood, F.K., 2009. Radon in Dwellings in Northern Ireland: 2009 Review and Atlas 34.
- GSNI, 2021. Geological Survey Northern Ireland (GSNI) [WWW Document]. Br. Geol. Surv. URL <https://www.bgs.ac.uk/geology-projects/gsni/>
- Jupyter, 2021a. Project Jupyter [WWW Document]. URL <https://www.jupyter.org> (accessed 5.26.21).
- Jupyter, 2021b. What to do when things go wrong — Jupyter Notebook 6.4.0 documentation [WWW Document]. URL <https://jupyter-notebook.readthedocs.io/en/stable/troubleshooting.html> (accessed 5.15.21).
- Matplotlib, 2021. matplotlib.pyplot.savefig — Matplotlib 3.4.2 documentation [WWW Document]. URL https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html (accessed 5.26.21).
- Medium One, 2018. Using matplotlib in jupyter notebooks — comparing methods and some tips [Python] [WWW Document]. Medium. URL <https://medium.com/@Med1um1/using-matplotlib-in-jupyter-notebooks-comparing-methods-and-some-tips-python-c38e85b40ba1> (accessed 5.13.21).
- NIEA, 2015. Northern Ireland Environment Agency | Department of Agriculture, Environment and Rural Affairs [WWW Document]. DAERA. URL <https://www.daera-ni.gov.uk/northern-ireland-environment-agency> (accessed 5.3.21).

- OSNI, 2020. OSNI Open Data - 50m DTM [WWW Document]. URL <https://data.gov.uk/dataset/72dbdf54-1447-4bd6-b6ce-787734914e45/osni-open-data-50m-dtm> (accessed 5.15.21).
- Otton, J.K., 1992. The geology of radon. U.S. Dept. of the Interior, U.S. Geological Survey ; For sale by the U.S. G.P.O., Supt. of Docs., Reston, Va.?]; Washington, DC.
- Python, 2021. PEP 257 -- Docstring Conventions [WWW Document]. Python.org. URL <https://www.python.org/dev/peps/pep-0257/> (accessed 5.26.21).
- Tenkanen, H., 2018. Spatial join — Intro to Python GIS documentation [WWW Document]. URL <https://automating-gis-processes.github.io/CSC18/lessons/L4/spatial-join.html> (accessed 5.26.21).
- WHO, 2021. Radon and health [WWW Document]. URL <https://www.who.int/news-room/fact-sheets/detail/radon-and-health> (accessed 5.1.21).