Lab Assignment 2 - Kate Peard

200005441

https://github.com/KateElla/PY4SA23_Assignment

In [ ]:
```
#Task 1

num = int(input("Your chosen number,"))
 #identifies that input is an integer
if num%3 == 0:
          print("is divisible by 3 :)")
else:
          print("is not divisble by 3 :(")


#As chosen number is an input, it will
#not show up on print form, but shows up
#in code execution:
#Your chosen number, 1728 is divisible by 3 :)
#( > I find this method nicer to look at in
#code, so have chosen to leave it this way)
```

In [2]:
```
#Task 2
fruits = ["apple","orange","pear","kiwi","strawberry"]
#defines list of acceptable fruit

frt = (input()) #identifies input as a string of text
if frt == fruits:
          print("is an acceptable fruit.")
else:
        print("is not an acceptable fruit.")
#As chosen fruit is an input, it will not show up on
#print form, but shows up in code
#(Guava is not an acceptable fruit)
```

```
is not an acceptable fruit.
```

In [3]:
```
# Task 3

R = 6371.0    #mean radius of Earth in km

 #import the required functions for the equations
from math import sin, cos, sqrt, atan2, radians


 #input for each coordinate as able to be decimal
print("      Coordinate 1:")
lat1, long1 = float(input("Latitude:")), float(input("Longitude:"))
print("      Coordinate 2:")
lat2, long2 = float(input("Latitude:")), float(input("Longitude:"))

#distance between longitude 2 and longitude 1
```

```
# > converted to radians
dlong = radians(long2) - radians(long1)
# distance between latitude 2 and latitude 1
dlat = radians(lat2) - radians(lat1)


#haversine formula with the correct variables inputted
a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlong / 2)**2
c = 2 * atan2(sqrt(a), sqrt(1 - a))

distance = c * R

print("The distance between these two points is", distance, "km.")
```

```
      Coordinate 1:

      Coordinate 2:

The distance between these two points is 7934.977814270405 km.
```

Pandas and NumPy

In [4]:
```
import numpy as np
import pandas as pd


#read portland tree data as port_trees
port_trees = pd.read_csv("data/portland_park_trees.csv")

#select required columns of tree data
trees = port_trees[["DBH", "Condition", "TreeHeight", "CrownWidth", "Family", "G
trees.head()
```

Out[4]:

	DBH	Condition	TreeHeight	CrownWidth	Family	Genus	Common_name	Genus_s
0	37.4	Fair	105.0	44.0	Pinaceae	Pseudotsuga	Douglas-fir	Pseudots menz
1	32.5	Fair	94.0	49.0	Pinaceae	Pseudotsuga	Douglas-fir	Pseudots menz
2	9.7	Fair	23.0	28.0	Rosaceae	Crataegus	Lavalle hawthorn	Crataeg lava
3	10.3	Poor	28.0	38.0	Fagaceae	Quercus	northern red oak	Que r
4	33.2	Fair	102.0	43.0	Pinaceae	Pseudotsuga	Douglas-fir	Pseudots menz

In [5]:
```
#Question 1


#a list denoting the required genera
QAgenus = ['Quercus', 'Acer']

 #query all values that appear in the list QAgenus
trees['Genus'].isin(QAgenus).sum()
```

```
#there are 5675 trees with over 50 inch diameter at breast height
```

Out[5]: 5675

Q1: There are 5675 trees with over 50 inch diameter at breast height.

In [6]:
```
#Question 2
treeGenus = trees.query("Genus in @QAgenus")
#query all values of genus Quercus and Acer

QAdbh = treeGenus.query('DBH >50')
#query of all items that have DBH greater than 50 and genus Quercus or Acer

print(len(QAdbh))
```

```
124
```

Q2: 124 trees have a DBH greater than 50in.

In [7]:
```
#Question 3
QAgenus.append('Fraxinus')
#add Fraxinus to required genera list

treeGenus = trees.query("Genus in @QAgenus")
#update treeGenus query to include Fraxnius data

treeMean= treeGenus.groupby("Genus")['DBH'].mean()
#evaluate mean of the 3 genuses, grouped by DBH

print(treeMean)
```

```
Genus
Acer        18.419085
Fraxinus    11.033610
Quercus     23.568238
Name: DBH, dtype: float64
```

Q3: Quercus has the highest mean.

In [8]:
```
#Question 4
acer = trees.query('Genus == "Acer"')
#identify acer as genus of interest


acerSpec = acer.Genus_spec.value_counts()
#organisate the data by number of trees of each species

print(acerSpec)
print(len(acerSpec))
#length for sum of species in genus
```

```
Acer platanoides                 1502
Acer macrophyllum                 490
Acer rubrum                       303
Acer circinatum                   299
Acer pseudoplatanus               211
Acer x freemanii                  207
Acer palmatum                     176
Acer saccharum                    153
Acer campestre                     68
Acer saccharinum                   52
Acer tataricum ssp. ginnala        48
Acer griseum                       44
Acer davidii                        8
Acer grandidentatum 'Schmidt'       6
Acer negundo                        6
Acer buergerianum                   5
Acer japonicum                      4
Acer rufinerve                      2
Acer monspessulanum                 1
Acer heldreichii                    1
Name: Genus_spec, dtype: int64
20
```

Q4: There are 20 species recorded in the Acer genus.

Q5:

In [9]:
```
cities_df = pd.read_csv("data/world_cities.csv")
#read csv file using pandas

cities_df.head(10)
```

Out[9]:

	city	country	pop	lat	lon	capital
0	'Abasan al-Jadidah	Palestine	5629	31.31	34.34	0
1	'Abasan al-Kabirah	Palestine	18999	31.32	34.35	0
2	'Abdul Hakim	Pakistan	47788	30.55	72.11	0
3	'Abdullah-as-Salam	Kuwait	21817	29.36	47.98	0
4	'Abud	Palestine	2456	32.03	35.07	0
5	'Abwein	Palestine	3434	32.03	35.20	0
6	'Adadlay	Somalia	9198	9.77	44.65	0
7	'Adale	Somalia	5492	2.75	46.30	0
8	'Afak	Iraq	22706	32.07	45.26	0
9	'Afif	Saudi Arabia	41731	23.92	42.93	0

In [10]:
```
cities_df["pop_m"] = cities_df["pop"] / 1000000
#create new column by dividing pop by 1 million

print(cities_df.head())
```

```
                 city    country    pop    lat    lon  capital     pop_m
0  'Abasan al-Jadidah  Palestine   5629  31.31  34.34        0  0.005629
1  'Abasan al-Kabirah  Palestine  18999  31.32  34.35        0  0.018999
2        'Abdul Hakim   Pakistan  47788  30.55  72.11        0  0.047788
3  'Abdullah-as-Salam     Kuwait  21817  29.36  47.98        0  0.021817
4               'Abud  Palestine   2456  32.03  35.07        0  0.002456
```

In [11]:
```
cities_df = cities_df.drop('pop', axis=1)
#remove column (axis = 1) called 'pop' from dataframe

print(cities_df.head())
```

```
                 city    country    lat    lon  capital     pop_m
0  'Abasan al-Jadidah  Palestine  31.31  34.34        0  0.005629
1  'Abasan al-Kabirah  Palestine  31.32  34.35        0  0.018999
2        'Abdul Hakim   Pakistan  30.55  72.11        0  0.047788
3  'Abdullah-as-Salam     Kuwait  29.36  47.98        0  0.021817
4               'Abud  Palestine  32.03  35.07        0  0.002456
```

In [16]:
```
Kcities = cities_df[cities_df['city'].str.startswith('K')]
#subset the data to select cities with a string that starts with K

KCity = Kcities.sample()   #select one row randomly

print(KCity)
```

```
           city  country   lat    lon  capital     pop_m
17793  Kesalahti  Finland  61.9  29.83        0  0.002627
```

In [17]:
```
Finland = (cities_df.query("country == 'Finland'")
   .sort_values(by="pop_m")
   .tail(5))
#subset the data by country, then sort by population in millions
#(smallest to largest > tail end of data is the largest population)

print(Finland)
```

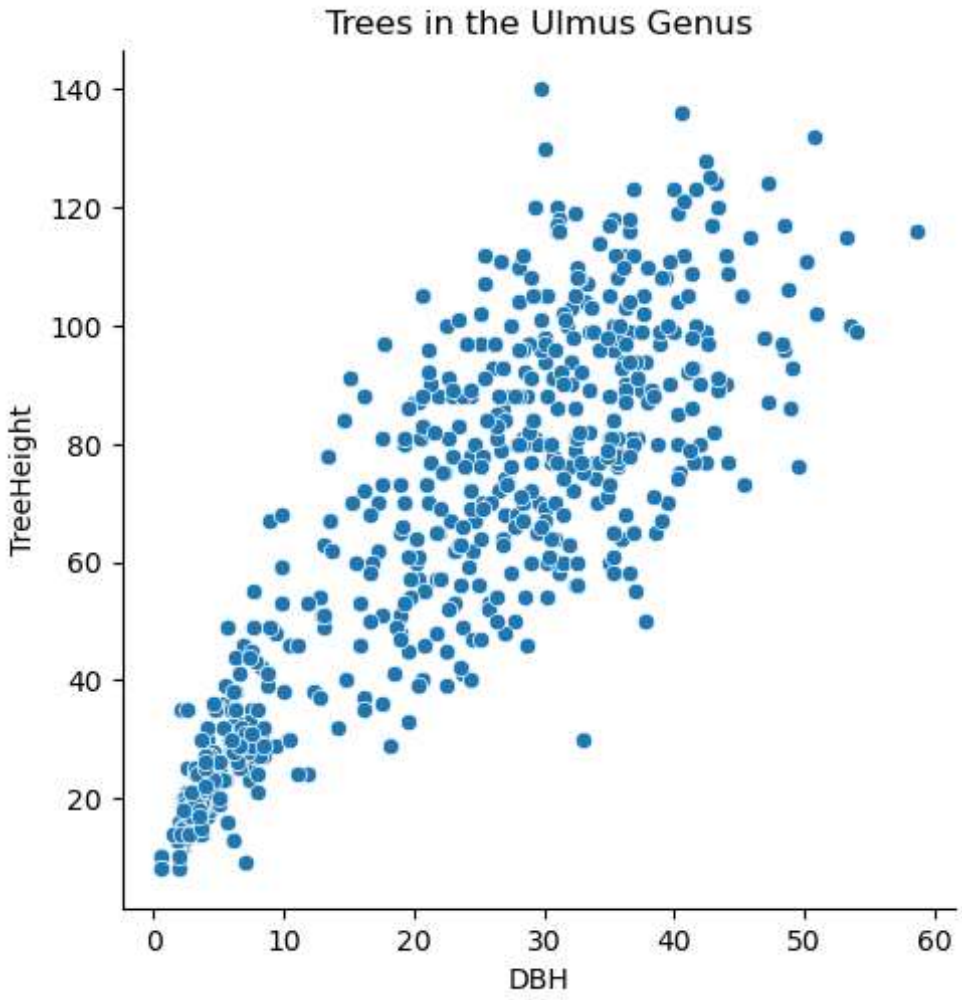
```
           city  country    lat    lon  capital     pop_m
39227     Turku  Finland  60.45  22.25        0  0.176425
40066    Vantaa  Finland  60.29  25.04        0  0.190622
37255   Tampere  Finland  61.52  23.76        0  0.203606
10950     Espoo  Finland  60.21  24.66        0  0.232293
14194  Helsinki  Finland  60.17  24.94        1  0.558341
```

In [18]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

Graph 1

In [19]:
```
Ulmus = trees.query('Genus == "Ulmus"')
#categorise data by Ulmus genus

sns.relplot(x="DBH", y="TreeHeight", data=Ulmus).set(title='Trees in the Ulmus G
```

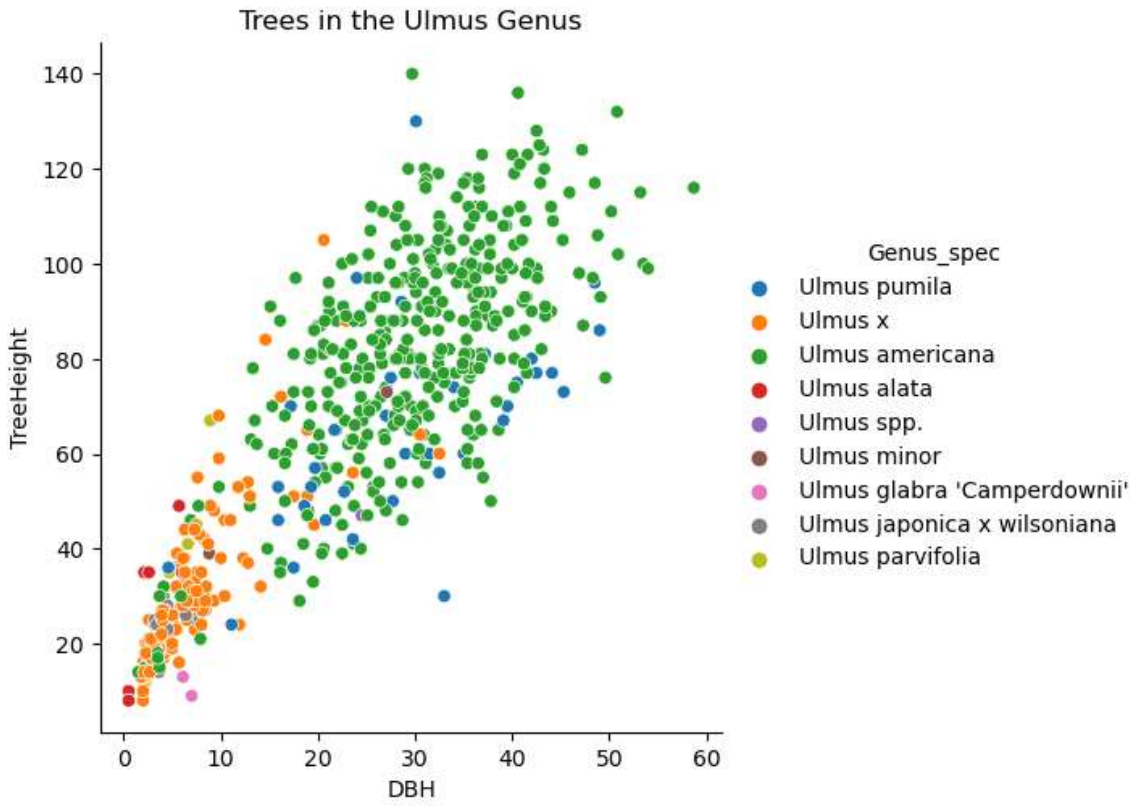
Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x26d9c5f3ee0>
```



Graph 2

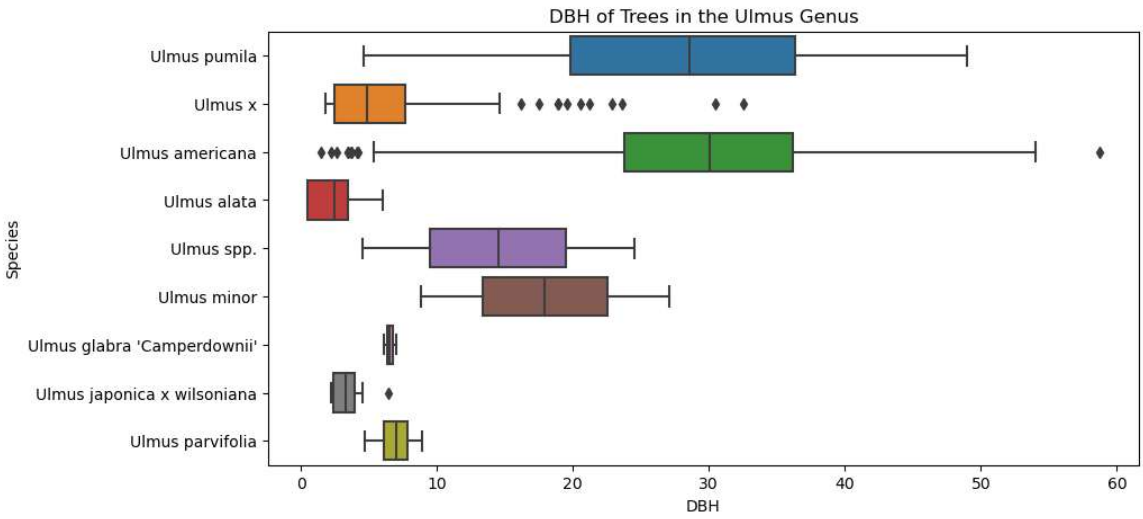
In [20]:
```
sns.relplot(x="DBH", y="TreeHeight", data=Ulmus, hue="Genus_spec",).set(title='T
# plot dbh by treeheight, with genus species determined by hue, using just the u
```

Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x26d9c687e80>
```



Graph 3

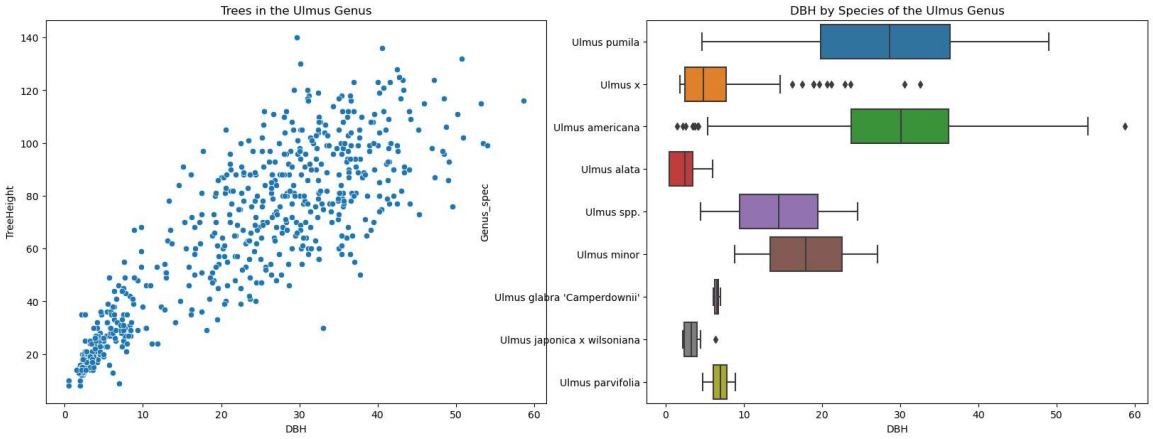
In [21]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(data=Ulmus, x='DBH', y='Genus_spec')
    #create a boxpot of dbh against species
plt.title('DBH of Trees in the Ulmus Genus')
plt.ylabel("Species")
    #change the label of the y axis - x remains unchanged
plt.show()
```



Graph 4

In [22]:
```
fig, axes = plt.subplots(1, 2, figsize=(20, 7))
#subplot with 1 row, 2 columns, with the figure size changed from default

sns.scatterplot(ax=axes[0], x=Ulmus.DBH, y=Ulmus.TreeHeight)
#scatterplot using graph 1's data prompt
```

```
ax=axes[0].set_title("Trees in the Ulmus Genus")
sns.boxplot(ax=axes[1], x=Ulmus.DBH, y=Ulmus.Genus_spec)
#boxplot of the Genus by species

ax=axes[1].set_title("DBH by Species of the Ulmus Genus")
```



GeoPandas

In [23]:
```
import os
os.environ['USE_PYGEOS']='0'
import geopandas as gpd
```

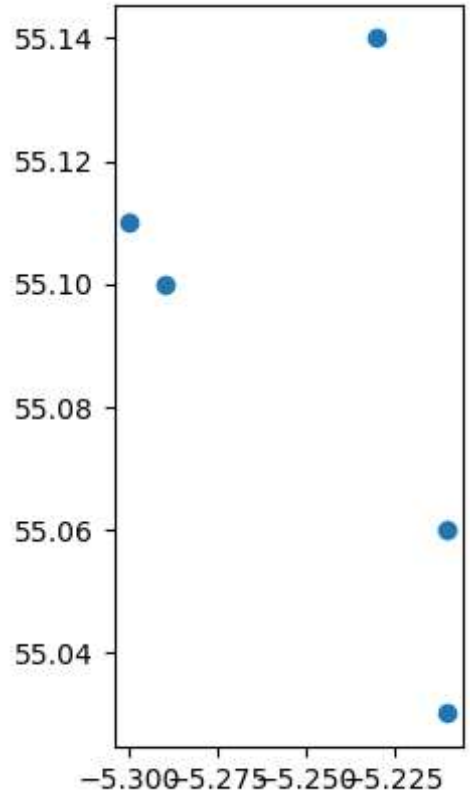
In [24]:
```
#Task 1

lobster = gpd.read_file('species_distribution_nephrops_burrow_densitiesPoint.shp
#read data as gpd - data on the abundance and density of lobster burrows across

lobster.head()
```

Out[24]:

	fu	year	lat	lon	avdens	station	geometry
0	Clyde	2007.0	55.03	-5.21	0.23	CL07001	POINT (-5.21000 55.03000)
1	Clyde	2007.0	55.06	-5.21	0.15	CL07002	POINT (-5.21000 55.06000)
2	Clyde	2007.0	55.10	-5.29	0.64	CL07003	POINT (-5.29000 55.10000)
3	Clyde	2007.0	55.11	-5.30	0.89	CL07004	POINT (-5.30000 55.11000)
4	Clyde	2007.0	55.14	-5.23	1.55	CL07005	POINT (-5.23000 55.14000)

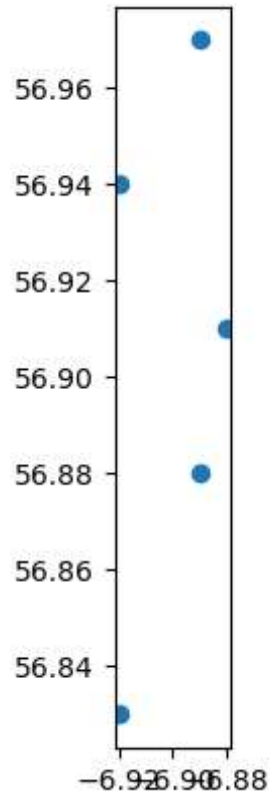
In [46]:
```
#Task 2
lobster.head().plot()
#a plot of the longitude and latitude of the 5 first records
```

Out[46]:
```
<AxesSubplot: >
```



In [47]:
```
lobster.tail().plot()
#a plot of the last 5 records
```

Out[47]: <AxesSubplot: >



In [27]:
```
#Task 3
lobster.explore(column ='avdens', cmap='RdYlBu')
#an explorable map showing the average density of lobster burrows
#[fu = body of water, avdens = avergae density, latitude,longitude and station]
```

```
#Does not print in pdf form - says I need to go to File >  Trust Notebook
#But I don't have a trust notebook option
#So this is a screenshot
```

Out[27]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [28]:
```
#Task 4 - GEOGCS WGS 84
lobster.crs
```

Out[28]:
```
<Geographic 2D CRS: GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84" ...>
Name: WGS 84
Axis Info [ellipsoidal]:
- lon[east]: Longitude (degree)
- lat[north]: Latitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [29]:
```
#Task 5 - 3052 features
len(lobster)
```

Out[29]:
```
3052
```

In [30]:
```
#Task 6

lobsterfladen = lobster['fu'] == 'Fladen'
#subsetting the data to only those that have a their body of water as Fladen
#I know fu is relating to which body of water the data is in but I cannot
#figure out what fu stands for, and I can't find it in the data anywhere

lobstergdf =  lobster[lobsterfladen]
lobstergdf
```

Out[30]:

	fu	year	lat	lon	avdens	station	geometry
956	Fladen	2007.0	57.93	-0.78	0.00	FL07001	POINT (-0.78000 57.93000)
957	Fladen	2007.0	58.09	-0.79	0.14	FL07002	POINT (-0.79000 58.09000)
958	Fladen	2007.0	58.24	-0.31	0.19	FL07003	POINT (-0.31000 58.24000)
959	Fladen	2007.0	57.97	-0.38	0.13	FL07004	POINT (-0.38000 57.97000)
960	Fladen	2007.0	58.00	-0.23	0.19	FL07005	POINT (-0.23000 58.00000)
...	...	...	...	...	...	...	...
1668	Fladen	2016.0	58.58	-0.80	0.24	FL16074	POINT (-0.80000 58.58000)
1669	Fladen	2016.0	58.47	-0.92	0.25	FL16075	POINT (-0.92000 58.47000)
1670	Fladen	2016.0	58.34	-0.96	0.02	FL16076	POINT (-0.96000 58.34000)
1671	Fladen	2016.0	58.26	-0.08	0.36	FL16077	POINT (-0.08000 58.26000)
1672	Fladen	2016.0	58.15	0.02	0.33	FL16078	POINT (0.02000 58.15000)

717 rows × 7 columns

In [31]:
```
#Task 7
print(lobster["avdens"].mean())
#a calculation for the mean density of lobster
#burrows across the whole dataset

lobsterdens = lobster["avdens"]>0.42
#a subset of data with burrow density greater than 0.42
#(a rounded up version of the mean)

lobstermean = lobster[lobsterdens]
#creating a new dataframe with this subset
lobstermean
```

```
0.416497378768021
```

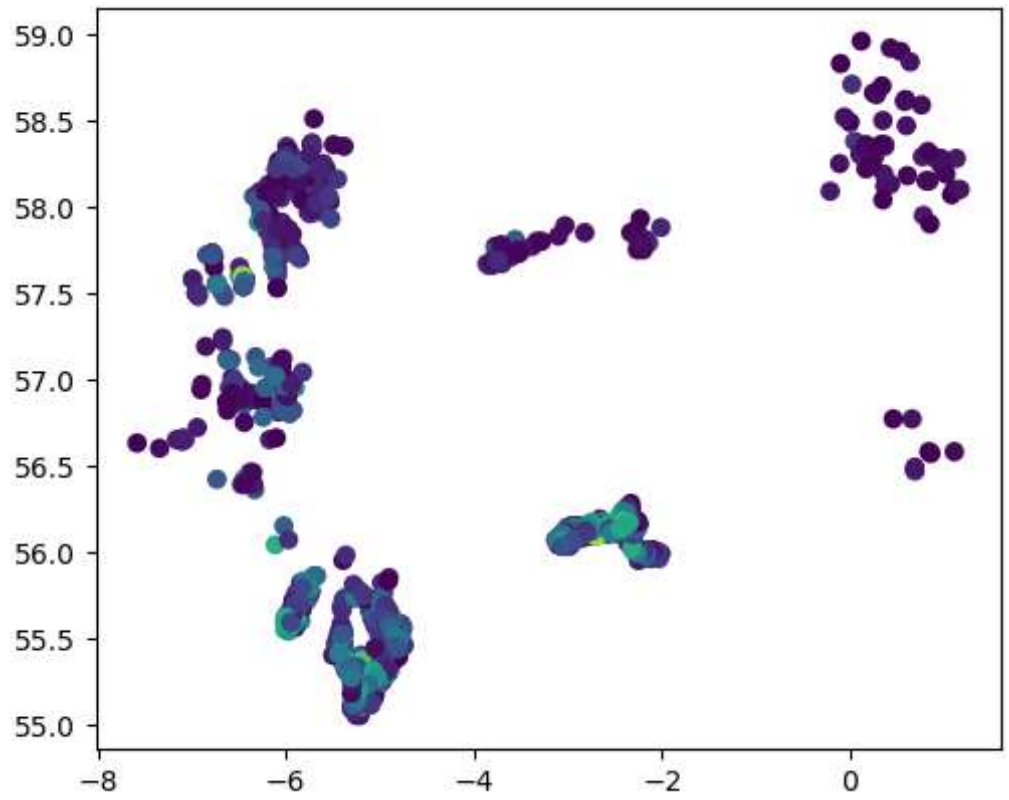
Out[31]:

	fu	year	lat	lon	avdens	station	geometry
2	Clyde	2007.0	55.10	-5.29	0.64	CL07003	POINT (-5.29000 55.10000)
3	Clyde	2007.0	55.11	-5.30	0.89	CL07004	POINT (-5.30000 55.11000)
4	Clyde	2007.0	55.14	-5.23	1.55	CL07005	POINT (-5.23000 55.14000)
5	Clyde	2007.0	55.16	-5.18	0.84	CL07006	POINT (-5.18000 55.16000)
12	Clyde	2007.0	55.60	-4.91	0.46	CL07013	POINT (-4.91000 55.60000)
...	...	...	...	...	...	...	...
3043	South Minch	2016.0	56.92	-6.57	0.47	SM16029	POINT (-6.57000 56.92000)
3044	South Minch	2016.0	56.88	-6.48	0.49	SM16030	POINT (-6.48000 56.88000)
3045	South Minch	2016.0	56.82	-6.62	0.45	SM16031	POINT (-6.62000 56.82000)
3046	South Minch	2016.0	56.87	-6.63	0.45	SM16032	POINT (-6.63000 56.87000)
3051	South Minch	2016.0	56.97	-6.89	0.48	SM16037	POINT (-6.89000 56.97000)

1120 rows × 7 columns

In [32]:
```
lobstermean.plot(column='avdens')
```

Out[32]: <AxesSubplot: >



Rasterio

In [33]:
```
import rasterio as rio
from rasterio import plot
from rasterio.plot import show
```

In [34]:
```
#Task1
elev = rio.open('data/elev.tif')
```

In [41]:
```
#Task 2 - EPSG:32617
print(elev.crs)
```

EPSG:32617

In [42]:
```
#Task 3
print(elev.count)
print(elev.width)
print(elev.height)
print(elev.bounds)
print(elev.crs)


#the raster data set has only 1 band, uses a EPSG:32617 coordinate reference sys
#(with WGS 84 datum) and the four corners of the raster's extent are left=479753
#bottom=4170823.2037591375, right=668843.3994558785, top=4347733.203759138)
```

```
1
6303
5897
BoundingBox(left=479753.39945587853, bottom=4170823.2037591375, right=668843.39
94558785, top=4347733.203759138)
EPSG:32617
```

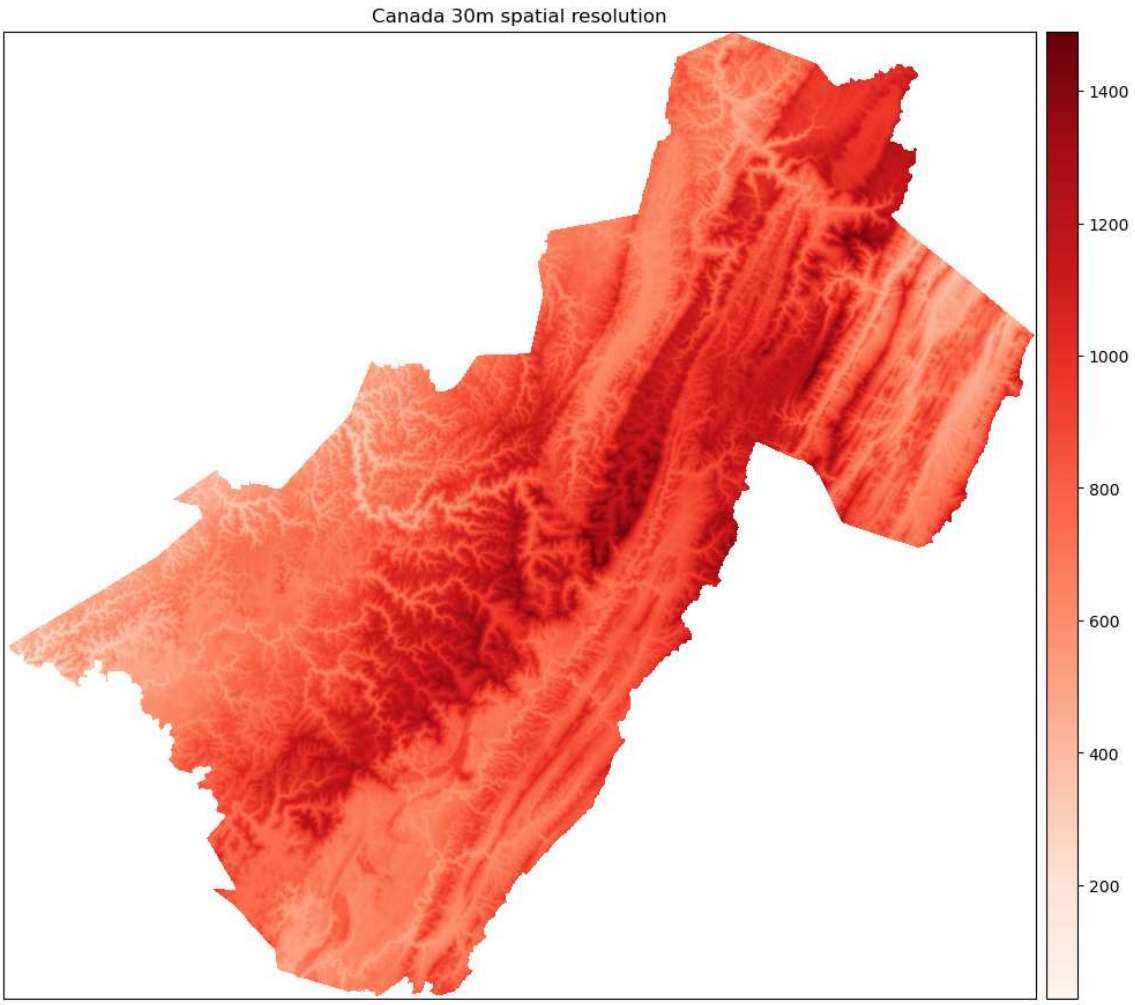
In [43]:
```
#Task 4

import earthpy as et
import earthpy.plot as ep
from matplotlib import pyplot


elev_arr = elev.read(1)
elev_arr



print("the minimum raster value is: ", elev_arr.min())
#minimun eleveation is likely to be skewed by no data values

print("the maximum raster value is: ", elev_arr.max())
```

```
the minimum raster value is:  -32768
the maximum raster value is:  1490
```

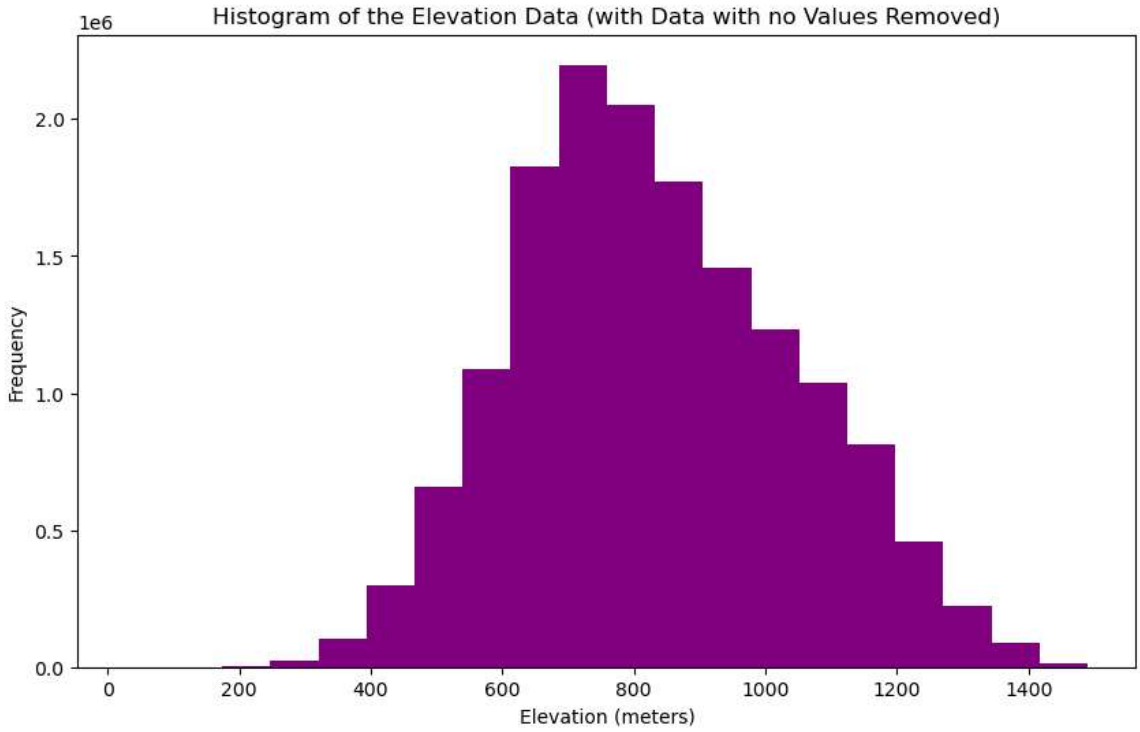
In [44]:
```
elev_arr = elev.read(1, masked=True)
#masked=true will mask all no data values
elev_arr

print("the minimum raster value is: ", elev_arr.min())
#minimum and maximum to check no data values have been removed

print("the maximum raster value is: ", elev_arr.max())
```

```
the minimum raster value is:  28
the maximum raster value is:  1490
```

In [45]:
```
ep.plot_bands(elev_arr,
              title="Canada 30m spatial resolution",
              cmap="Reds")
plt.show()
```



In [40]:
```
#Task 5

ep.hist(elev_arr,
        figsize=(10, 6),
        title="Histogram of the Elevation Data (with Data with no Values Removed
        xlabel='Elevation (meters)',
        ylabel='Frequency')
plt.show()
```


Histogram of the Elevation Data (with Data with no Values Removed)
1e6
2.0
1.5
1.0
0.5
0.0
Frequency
0
200
400
600
800
1000
1200
1400
Elevation (meters)