

Pandas Data Frames

DSE5002, Peter Gyorda, March 2025

Pandas is a python implementation of the R or SQL style data frame or data table

Indexing is a bit different, and there are some other "wrinkles" to it

There are a lot of member functions (aka methods) in Pandas to do a lot of basic data processing

Pandas data frames have variables along columns, which can be of different types

More resources

<https://pandas.pydata.org/>

```
In [1]: import pandas as pd
```

```
/home/bb9023db-ac0f-4605-8fb7-449aa3135ccb/.local/lib/python3.11/site-packages/pandas/core/arrays/masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
```

```
    from pandas.core import (
```

we will load a data frame describing public wifi access sites in Boston

the file is called wicked_free_wifi_boston.csv

There is a read_csv function in pandas, that will attempt to assign variable types to columns

You will need to insert the full file path into the variable infile below, or make sure that the file is in the current working directory

below is the command from the os library to show the current working directory

we can use os.chdir() to change the current working directory

```
In [2]: import os
```

```
os.getcwd()
```

```
Out[2]: '/home/bb9023db-ac0f-4605-8fb7-449aa3135ccb/DSE 5002/Module 2/Pair Programming'
```

```
In [4]: infile="Wicked_Free_WiFi_Locations.csv"
```

```
wifi=pd.read_csv(infile)
```

```
In [5]: # head function, called as a method belonging to the dataframe (a python object) ca
```

```
# wifi is said to be an instance of a python dataframe
wifi.head(7)
```

Out[5]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
0	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q3AK-SUL7-7FC4
1	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q2ZY-RF99-YN45
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
3	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-DU9C-2UXZ
4	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-FGAN-3AR9
5	-7.913037e+06	5.209642e+06	N_568579452955527921	Parks	Q2EK-4PWN-GALS
6	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SSY2-PBYW



In [6]:

```
#unlike the R data frames, head doesn't show us the data types
# we need to do that manually, looking at the attribute dtypes

wifi.dtypes
```

Out[6]:

X	float64
Y	float64
neighborhood_id	object
neighborhood_name	object
device_serial	object
device_connectedto	object
device_address	object
device_lat	float64
device_long	float64
device_tags	object
etl_updatedtimestamp	object
is_current	int64
org1	object
org2	object
inside_outside	object
landmark	object
ObjectId	int64
dtype:	object

What is the data type "object" in Pandas

An object is a string storage form

```
In [7]: # Generating a Summary
# describes only numeric values

wifi.describe()
```

	X	Y	device_lat	device_long	is_current	Objectid
count	2.830000e+02	2.830000e+02	283.000000	283.000000	297.0	297.000000
mean	-7.912135e+06	5.210210e+06	42.327796	-71.075917	1.0	149.000000
std	3.034883e+03	4.447129e+03	0.029537	0.027263	0.0	85.880731
min	-7.922403e+06	5.198613e+06	42.250739	-71.168161	1.0	1.000000
25%	-7.913761e+06	5.207725e+06	42.311295	-71.090520	1.0	75.000000
50%	-7.912078e+06	5.210305e+06	42.328431	-71.075407	1.0	149.000000
75%	-7.910171e+06	5.214371e+06	42.355431	-71.058271	1.0	223.000000
max	-7.904348e+06	5.218989e+06	42.386080	-71.005970	1.0	297.000000

Subsetting and slicing in pandas

```
In [9]: #accessing a column, there are several options

n_name=wifi.neighborhood_name
w_address=wifi["device_address"]
```

Pandas series

If we extract a column it is in the form of a pandas data series, which still has a lot of pandas style member functions

```
In [10]: type(n_name)
```

```
Out[10]: pandas.core.series.Series
```

```
In [11]: # we can convert this to a list, using a member function

n_name_list=n_name.to_list()
type(n_name_list)
```

Out[11]: list

```
In [13]: # dimensions of a dataframe are obtained using the attribute shape
print(wifi.shape)
print(n_name.shape)
```

(297, 17)
(297,)

In [12]: #indexing several columns

wifi[["X", "Y"]].head()

Out[12]:

	X	Y
0	NaN	NaN
1	NaN	NaN
2	-7.912255e+06	5.206228e+06
3	NaN	NaN
4	NaN	NaN

Question/Action

use head to show the first 5 rows of the neighborhood id and name

In [77]: wifi[["neighborhood_id", "neighborhood_name"]].head()

Out[77]:

	neighborhood_id	neighborhood_name
0	L_601230550253963849	Mobile WiFi Kit 1
1	L_601230550253963849	Mobile WiFi Kit 1
2	L_601230550253964116	Nubian-Bus-Stop
3	L_601230550253964116	Nubian-Bus-Stop
4	L_601230550253964116	Nubian-Bus-Stop

In [15]: #Basic calculations

```
print(wifi.device_lat.max())
print(wifi.device_lat.min())
print(wifi.device_lat.mean())
```

42.38608
42.2507393
42.32779576223686

```
In [16]: # filtering rows using conditional dependence
#Let's find all devices with Latitude above 42.3271405
above_wifi=wifi[wifi.device_lat>=42.3271405]
above_wifi.head()
```

Out[16]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
6	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SSY2-PBYW
7	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SU8N-5VU8
15	-7.912357e+06	5.210581e+06	N_579275502070532581	Roxbury	Q2CK-HDFV-VYBC
18	-7.912846e+06	5.210317e+06	N_579275502070532581	Roxbury	Q2CK-SF4S-8JL2
20	-7.912858e+06	5.210361e+06	N_579275502070532581	Roxbury	Q2CK-MR56-4QY6



In [17]: #slicing by values in a set, notice that pandas has a isin() member function for the wifi[wifi.neighborhood_name.isin(["Parks", "Charlestown"])]

Out[17]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
5	-7.913037e+06	5.209642e+06	N_568579452955527921	Parks	Q2E\4PWN-GAL
28	-7.910979e+06	5.214437e+06	N_568579452955527921	Parks	Q3AK-CVA\CUZ
44	-7.916707e+06	5.202882e+06	N_568579452955527921	Parks	Q2AK-U4T\J9E
188	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-V3L\5V6
202	-7.910805e+06	5.214387e+06	N_568579452955527921	Parks	Q3AK-DGS\GM7
203	-7.911261e+06	5.214274e+06	N_568579452955527921	Parks	Q3AK-EK6\T4F\
205	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK\CWU\RBG\
214	-7.911012e+06	5.214442e+06	N_568579452955527921	Parks	Q3AK-DRLE\LEZ
223	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-SNT\WJW
226	-7.911133e+06	5.214283e+06	N_568579452955527921	Parks	Q3AK-CR6\YPB
227	-7.916761e+06	5.208413e+06	N_568579452955527921	Parks	Q2CK-7ANL\RF7
228	-7.910954e+06	5.213995e+06	N_568579452955527921	Parks	Q3AK-CR9\A99
229	-7.916761e+06	5.208413e+06	N_568579452955527921	Parks	Q2CD-6YGI\H7P\

	X	Y	neighborhood_id	neighborhood_name	device_serial
230	-7.910787e+06	5.214274e+06	N_568579452955527921	Parks	Q3Ak DGWD-NEF
231	-7.910746e+06	5.214357e+06	N_568579452955527921	Parks	Q3Ak CVAW H5UI
240	-7.916761e+06	5.208413e+06	N_568579452955527921	Parks	Q2CK-7CSF NUQ
241	-7.911216e+06	5.214476e+06	N_568579452955527921	Parks	Q3Ak CRWB-RXV
242	-7.910801e+06	5.214373e+06	N_568579452955527921	Parks	Q3Ak CQWK-ZYF

In [18]:

```
# there is also a str.contains function, which searches for a regex string within t
# we will talk about regex in more detail later

# notice the use of .str.contains(),    a string function within pandas

# see https://pandas.pydata.org/docs/user_guide/text.html for a bunch of examples o
wifi[wifi.neighborhood_name.str.contains("town")]
```

Out[18]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
53	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-4FLASJC
54	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-HJRC D68
55	-7.909698e+06	5.215107e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ARNEHFA
56	-7.909702e+06	5.215082e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-PMFQ-JUX
57	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-3S8ECZF
58	-7.909943e+06	5.215065e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AK-C98IJUR
59	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-TBTSD72
61	-7.909943e+06	5.215065e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AKD5DU-9HS
62	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ZUFY73
69	-7.909890e+06	5.215063e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-F6RVVH
70	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ZXWH9ZI
71	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-3A2VJNV
152	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-JBPI DSN
153	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-ZDPZ2LN

	X	Y	neighborhood_id	neighborhood_name	device_serial
154	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-QYJF RXI
155	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AI 2N4M BWU
156	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-TF7L TX4
157	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-95XZ 766
158	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-M6TA MFV
159	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-R6BC D5K
160	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-M9Y5 PAT
161	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-E751 LLQ
162	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-BFVE YGN
163	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-8YN1 BM6
164	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AI CANN-4NE
167	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-2F27 DVJ
168	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-EETC KYU

	X	Y	neighborhood_id	neighborhood_name	device_serial
169	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-A94E FES
170	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE WU87-WVG
171	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-JXU2 QPC
172	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-A793 TEM
188	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-V3LS 5V6
205	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK CWU\ RBG\
223	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-SNTE WJW
256	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-GDTF 3C3\
257	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE MULH-9V9
258	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-JLUI 293
275	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE QMTD-57L
278	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-Q4LC 6HK
289	-7.909896e+06	5.215085e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-Q7YC V97

```
In [19]: # notna returns true or false depending on whether there are Nan values in the Loca
# the list of True/False values produced by notna can be used to slice

wifi[wifi.X.notna()].head()
```

Out[19]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
5	-7.913037e+06	5.209642e+06	N_568579452955527921	Parks	Q2EK-4PWN-GALS
6	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SSY2-PBYW
7	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SU8N-5VU8
8	-7.913466e+06	5.208391e+06	N_579275502070532581	Roxbury	Q2CK-H5VS-5UKS



In [20]: # Row and column specification
use paired conditions on [row, column]
we now have to use the method .loc[] to do this

wifi.loc[wifi.X.notna(),"X"].head()

Out[20]: 2 -7.912255e+06
5 -7.913037e+06
6 -7.913800e+06
7 -7.913800e+06
8 -7.913466e+06
Name: X, dtype: float64

In [21]: # you have to have a boolean return type in the row indexing function or a set of i
we can send a list of column names to get several of them

wifi.loc[wifi.neighborhood_name.str.contains("Charlestown"),['device_lat','device_l

Out[21]:

	device_lat	device_long
188	42.380109	-71.06107
205	42.380109	-71.06107
223	42.380109	-71.06107

In [22]:

```
# Indexing using integer values is done using the iloc[] function
# so remember- used .Loc for boolean and named columns,    .iloc for Integer Location
wifi.iloc[0:8,0:6]
```

Out[22]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
0	Nan	Nan	L_601230550253963849	Mobile WiFi Kit 1	Q3AK-SUL7-7FC4
1	Nan	Nan	L_601230550253963849	Mobile WiFi Kit 1	Q2ZY-RF99-YN45
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
3	Nan	Nan	L_601230550253964116	Nubian-Bus-Stop	Q3AK-DU9C-2UXZ
4	Nan	Nan	L_601230550253964116	Nubian-Bus-Stop	Q3AK-FGAN-3AR9
5	-7.913037e+06	5.209642e+06	N_568579452955527921	Parks	Q2EK-4PWN-GALS
6	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SSY2-PBYW
7	-7.913800e+06	5.210828e+06	N_579275502070532581	Roxbury	Q2CK-SU8N-5VU8

Plotting with Pandas functions

Pandas has basic plotting built in

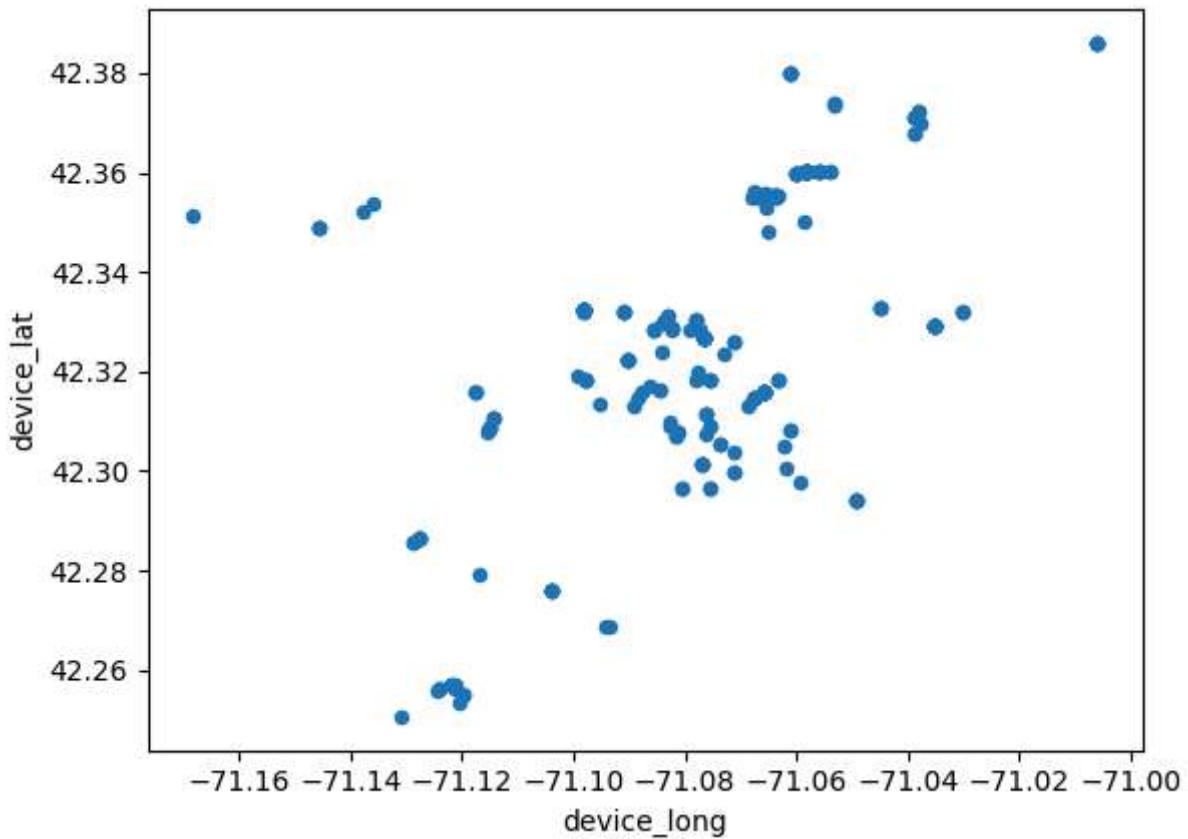
I typically use Matplotlib, but Pandas has the basics

In [23]:

```
#Pandas built in plots
wifi.plot.scatter(x="device_long",y="device_lat")
```

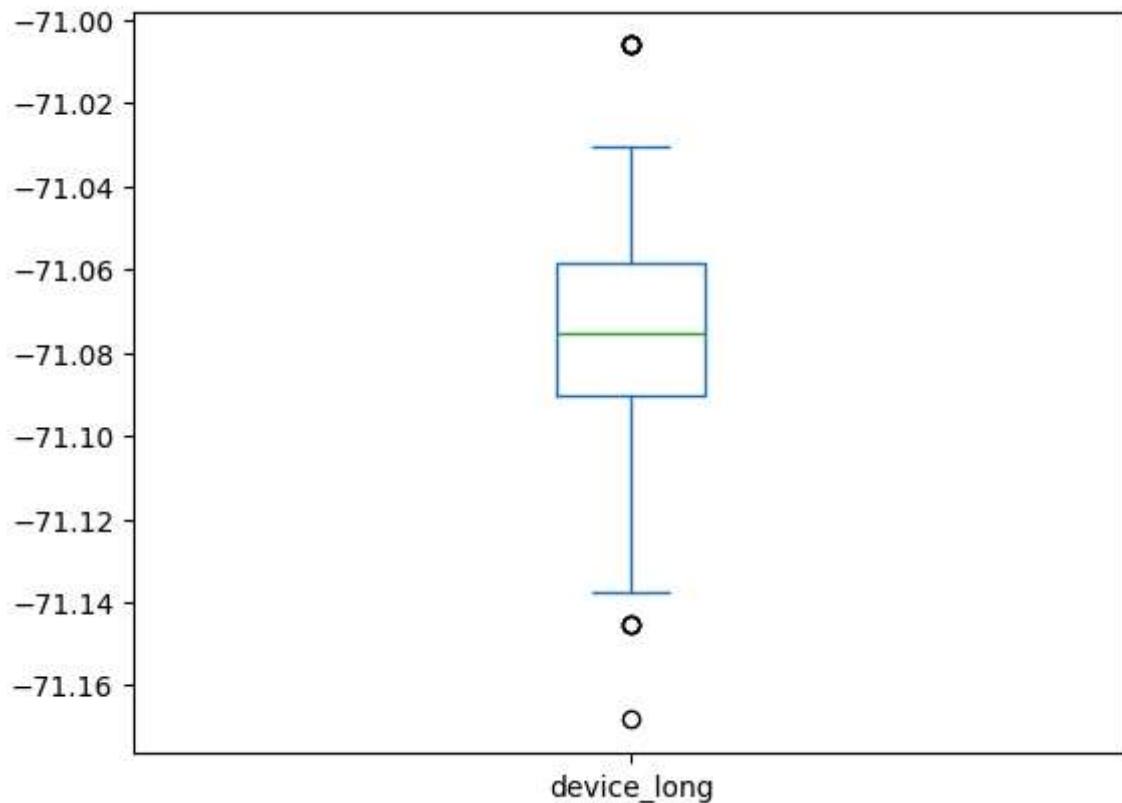
Out[23]:

```
<Axes: xlabel='device_long', ylabel='device_lat'>
```



```
In [24]: #here is a boxplot  
wifi[["device_long"]].plot.box()
```

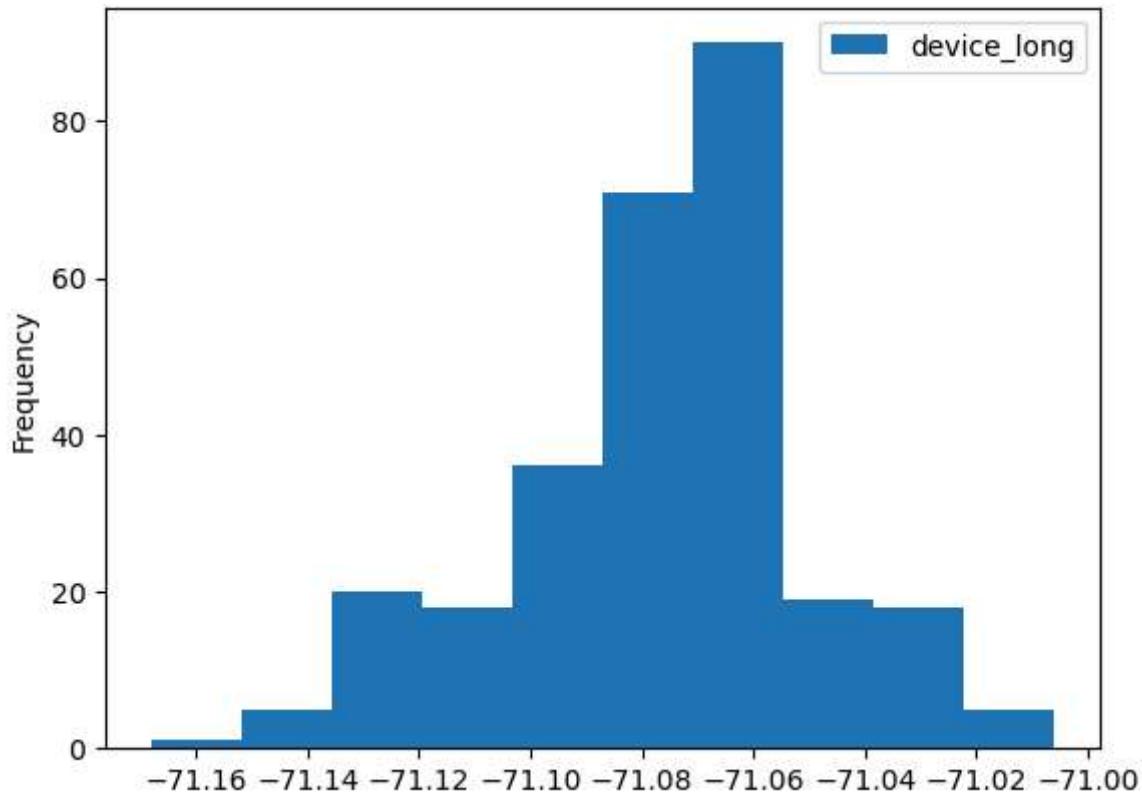
```
Out[24]: <Axes: >
```



```
In [23]: #histogram
```

```
wifi[["device_long"]].plot.hist()
```

```
Out[23]: <Axes: ylabel='Frequency'>
```



```
In [25]: #creating new columns

#just name the column and assign a value

# this is a nonsensical value, but it shows the idea

wifi["x over y"] = wifi.X/wifi.Y

wifi.head()
```

Out[25]:

	X	Y	neighborhood_id	neighborhood_name	device_serial
0	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q3AK-SUL7-7FC4
1	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q2ZY-RF99-YN45
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
3	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-DU9C-2UXZ
4	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-FGAN-3AR9

Aggregation or grouping for tables and statistics

Pandas has a nice groupby function, reminiscent of the dplyr methods

```
In [26]: # we specify the columns we want to work with in the dataframe, then specify which

# at the end, we add a Pandas summary function

# Note, if we had more grouping variables the input to groupby could be a list

wifi[['device_long','device_lat','neighborhood_name']].groupby("neighborhood_name")
```

Out[26]:

neighborhood_name	device_long	device_lat
2 Center Plaza	-71.060201	42.359796
Alston Brighton	-71.146410	42.350730
BCYF Indoor	-71.084504	42.320980
BCYF Tremont	-71.098229	42.332227
BCYF-Curley	-71.035150	42.329115
Bolling	-71.083664	42.330141
Charlestown	-71.061070	42.380109
City Hall Truck	NaN	NaN
Dorchester	-71.067166	42.306111
Downtown Boston - City Hall - Quincy Market	-71.057176	42.360229
Downtown Boston - City Hall Plaza and Pavilion	-71.058271	42.360291
East Boston	-71.038550	42.370514
East Boston Senior Center	-71.005970	42.386080
Hyde Park	-71.122637	42.255603
Jamaica Plain	-71.114836	42.309554
Maintenance	-71.063579	42.347038
Mattahunt BCYF	-71.103910	42.275826
Mattapan-Bus-stop	-71.093735	42.268617
Mobile WiFi Kit 1	NaN	NaN
Mobile WiFi Kit 2	NaN	NaN
Mobile WiFi Kit 3	NaN	NaN
Mobile WiFi Kit 4	NaN	NaN
Mobile WiFi Kit 5	NaN	NaN
NOC-TEST	-71.076985	42.331899
Navy Yard	-71.053276	42.373728
Nubian-Bus-Stop	-71.077000	42.301350
OPAT	-71.082958	42.331086
Parks	-71.080528	42.340156
Roslindale	-71.128088	42.286149

	device_long	device_lat
neighborhood_name		
Roxbury	-71.082860	42.320404
South Boston	-71.037557	42.332484
Strand Theatre - External	-71.065961	42.315948
Strand Theatre - Internal	-71.065961	42.315948
YEE Tremont	-71.098240	42.332229

Multiple grouping variables

```
In [27]: # we can use groupby to get counts per grouping variable as well  
# I tried using is_current as a grouping variable, but they are all 1, indicating c  
wifi[["device_long","device_lat","neighborhood_name","is_current"]].groupby(["neigh
```

Out[27]:

neighborhood_name	is_current	device_long	device_lat
2 Center Plaza	1	6	6
Alston Brighton	1	6	6
BCYF Indoor	1	27	27
BCYF Tremont	1	9	9
BCYF-Curley	1	12	12
Bolling	1	6	6
Charlestown	1	3	3
City Hall Truck	1	0	0
Dorchester	1	24	24
Downtown Boston - City Hall - Quincy Market	1	16	16
Downtown Boston - City Hall Plaza and Pavilion	1	21	21
East Boston	1	9	9
East Boston Senior Center	1	5	5
Hyde Park	1	12	12
Jamaica Plain	1	6	6
Maintenance	1	6	6
Mattahunt BCYF	1	5	5
Mattapan-Bus-stop	1	2	2
Mobile WiFi Kit 1	1	0	0
Mobile WiFi Kit 2	1	0	0
Mobile WiFi Kit 3	1	0	0
Mobile WiFi Kit 4	1	0	0
Mobile WiFi Kit 5	1	0	0
NOC-TEST	1	2	2
Navy Yard	1	3	3
Nubian-Bus-Stop	1	5	5
OPAT	1	2	2
Parks	1	15	15
Roslindale	1	5	5

			device_long	device_lat
	neighborhood_name	is_current		
	Roxbury	1	52	52
	South Boston	1	6	6
	Strand Theatre - External	1	3	3
	Strand Theatre - Internal	1	10	10
	YEE Tremont	1	5	5

Categorical data

We can set data to be of type Categorical, which is akin to a factor

It is also possible to use integer group codes or dummy coding to represent categories or factors, this is done using utility tools in libraries such as scikit-learn or keras that focus on modeling

```
In [28]: wifi['neighborhood_name']=pd.Categorical(wifi.neighborhood_name)

wifi.head()
```

	X	Y	neighborhood_id	neighborhood_name	device_serial
0	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q3AK-SUL7-7FC4
1	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q2ZY-RF99-YN45
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
3	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-DU9C-2UXZ
4	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-FGAN-3AR9

```
# did this work

wifi.dtypes
```

```
Out[29]: X           float64
          Y           float64
neighborhood_id      object
neighborhood_name    category
device_serial        object
device_connectedto   object
device_address       object
device_lat           float64
device_long          float64
device_tags          object
etl_updatedtimestamp object
is_current           int64
org1                 object
org2                 object
inside_outside       object
landmark              object
ObjectId             int64
x over y             float64
dtype: object
```

Question/Action

What other variables should be Categorical variables (there aren't many) ANSWER:
neighborhood_id

Convert this variable to a category

```
In [78]: wifi['neighborhood_id']=pd.Categorical(wifi.neighborhood_id)

wifi.dtypes
```

```
Out[78]: X           float64
          Y           float64
neighborhood_id      category
neighborhood_name    category
device_serial        object
device_connectedto   object
device_address       object
device_lat           float64
device_long          float64
device_tags          object
etl_updatedtimestamp datetime64[ns, UTC]
is_current           int64
org1                 object
org2                 object
inside_outside       object
landmark              object
ObjectId             int64
x over y             float64
dtype: object
```

Dummy Coding

It looks like Pandas can generate dummy codes for us

Pandas does not have a 'factor' variable, so in models like multiple regression, logistic regression or neural networks, we use dummy coding to code categorical variables. You will see more on this later.

What does this look like?

What does a True in this table seem to mean?

This is also called "one-hot" encoding, since there is only one "True" per row of the table

```
In [32]: pd.get_dummies(wifi.neighborhood_name)
```

Out[32]:

	2 Center Plaza	Alston	BCYF Indoor	BCYF Tremont	BCYF-Curley	Bolling	Charlestown	City Hall	Dorchester Truck
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...
292	False	False	False	False	False	False	False	False	False
293	False	False	False	False	False	True	False	False	False
294	False	False	False	False	False	True	False	False	False
295	False	False	False	False	False	True	False	False	False
296	False	False	False	False	False	True	False	False	False

297 rows × 34 columns



Question/Action

Create a dummy coding for the variable that you turned into a Categorical variable in the Question above

```
In [41]: neighborhood_id = 'neighborhood_id' # Replace with your actual column name
dummy_variables = pd.get_dummies(wifi[neighborhood_id], prefix=neighborhood_id)
wifi.head()
```

	X	Y	neighborhood_id	neighborhood_name	device_serial
0	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q3AK-SUL7-7FC4
1	NaN	NaN	L_601230550253963849	Mobile WiFi Kit 1	Q2ZY-RF99-YN45
2	-7.912255e+06	5.206228e+06	L_601230550253964116	Nubian-Bus-Stop	Q3AE-QFTK-E55W
3	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-DU9C-2UXZ
4	NaN	NaN	L_601230550253964116	Nubian-Bus-Stop	Q3AK-FGAN-3AR9

date-time values

It looks like we have one date-time variable in the dataset right now

etl_updatedtimestamp

it looks like a fairly standard format

```
In [42]: wifi.etl_updatedtimestamp.head(5)
```

```
Out[42]: 0    2024/08/20 04:31:34+00
1    2024/08/20 04:31:34+00
2    2024/08/20 04:31:38+00
3    2024/08/20 04:31:38+00
4    2024/08/20 04:31:38+00
Name: etl_updatedtimestamp, dtype: object
```

```
In [43]: wifi.etl_updatedtimestamp=pd.to_datetime(wifi.etl_updatedtimestamp)
wifi.etl_updatedtimestamp.head()
```

```
Out[43]: 0    2024-08-20 04:31:34+00:00
          1    2024-08-20 04:31:34+00:00
          2    2024-08-20 04:31:38+00:00
          3    2024-08-20 04:31:38+00:00
          4    2024-08-20 04:31:38+00:00
Name: etl_updatedtimestamp, dtype: datetime64[ns, UTC]
```

```
In [44]: # We can now get days, months, years

wifi.etl_updatedtimestamp.dt.day.head()
```

```
Out[44]: 0    20
          1    20
          2    20
          3    20
          4    20
Name: etl_updatedtimestamp, dtype: int32
```

```
In [45]: wifi.etl_updatedtimestamp.dt.month.head()
```

```
Out[45]: 0    8
          1    8
          2    8
          3    8
          4    8
Name: etl_updatedtimestamp, dtype: int32
```

```
In [46]: wifi.etl_updatedtimestamp.dt.year.head()
```

```
Out[46]: 0    2024
          1    2024
          2    2024
          3    2024
          4    2024
Name: etl_updatedtimestamp, dtype: int32
```

Converting to Long form

uses the melt function

<https://pandas.pydata.org/docs/reference/api/pandas.melt.html>

Form more ideas on wide to long, look up

Pandas pivot pandas pivot_table pandas unstack pandas wide_to_long

```
In [47]: df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
                           'B': {0: 1, 1: 3, 2: 5},
                           'C': {0: 2, 1: 4, 2: 6}})

df
```

Out[47]:

	A	B	C
0	a	1	2
1	b	3	4
2	c	5	6

In []:

In [49]: *# note we specify a list of i variables and a list of value variables, much like i*

```
pd.melt(df, id_vars=['A'], value_vars=['B', 'C'])
```

Out[49]:

	A	variable	value
0	a	B	1
1	b	B	3
2	c	B	5
3	a	C	2
4	b	C	4
5	c	C	6

In [50]: *# alternative form, two id variables
this is a "composite key" form*

```
pd.melt(df, id_vars=['A', 'C'], value_vars=['B'])
```

Out[50]:

	A	C	variable	value
0	a	2	B	1
1	b	4	B	3
2	c	6	B	5

In [51]: *# create df2 for question/action*

```
df2 = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c', 3: 'd'},
                    'B': {0: 1, 1: 3, 2: 5, 3: 6},
                    'C': {0: 2, 1: 4, 2: 6, 3: 8},
                    'D': {0: "Biscuit", 1: "Chips", 2: "Banana", 3: "hard case"})
```

df2

Out[51]:

	A	B	C	D
0	a	1	2	Biscuit
1	b	3	4	Chips
2	c	5	6	Banana
3	d	6	8	hard case

Question/Action

Melt df2 to wide form, using D and A as the index variables, assign the other two columns as values

```
In [70]: # alternative form, two id variables
# this is a "composite key" form

pd.melt(df2, id_vars=['D', 'A'], value_vars=['B', 'C'])
```

Out[70]:

	D	A	variable	value
0	Biscuit	a	B	1
1	Chips	b	B	3
2	Banana	c	B	5
3	hard case	d	B	6
4	Biscuit	a	C	2
5	Chips	b	C	4
6	Banana	c	C	6
7	hard case	d	C	8

Joins

A join connects two dataframes (or SQL data tables) together based on matching values of keys (identifiers) in the two data frames or tables. You may have seen this in R, and we will see it again in SQL.

Joins are done on two dataframes (or tables) at a time, the first is called the "left" table and the second is called the "right" table and several different forms of joins exist.

Joins are done on Pandas data frames are done using the merge function

You can specify the type of join desired, inner, outer, left, right etc

<https://pandas.pydata.org/docs/reference/api/pandas.merge.html>

```
In [71]: df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo', "bix"], 'value': [1, 2, 3, 5, 11]}  
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'], 'value': [5, 6, 7, 8]})
```

```
In [72]: df1
```

```
Out[72]:   lkey  value
```

0	foo	1
1	bar	2
2	baz	3
3	foo	5
4	bix	11

```
In [73]: df2
```

```
Out[73]:   rkey  value
```

0	foo	5
1	bar	6
2	baz	7
3	foo	8

```
In [74]: # this is an inner join of the left frame df1 and the right frame df2  
# notice that "bix" was dropped  
df1.merge(df2, left_on='lkey', right_on='rkey')
```

```
Out[74]:   lkey  value_x  rkey  value_y
```

0	foo	1	foo	5
1	foo	1	foo	8
2	bar	2	bar	6
3	baz	3	baz	7
4	foo	5	foo	5
5	foo	5	foo	8

```
In [75]: # here is a Left join
```

```
# what happens to "bix"?
```

```
df1.merge(df2, how="left", left_on='lkey', right_on='rkey')
```

Out[75]:

	lkey	value_x	rkey	value_y
0	foo	1	foo	5.0
1	foo	1	foo	8.0
2	bar	2	bar	6.0
3	baz	3	baz	7.0
4	foo	5	foo	5.0
5	foo	5	foo	8.0
6	bix	11	NaN	NaN

In []: ANSWER: Since we are doing a left join, a left merge prioritizes keeping all rows from the left DataFrame. Since "bix" is present in df1, it's remains in the even though there's no corresponding "bix" in df2. The columns from the right dataf populated with NaN values.

In []:

In []: