Pandas Data Frames

DSE5002, HD Sheets, July 2024

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 id

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 [6]:
```
import pandas as pd
```

we will load a data frame descriping 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 [8]:
```
import os

os.getcwd()
```

Out[8]:
```
'C:\\Users\\Luke\\Documents\\Class5002\\Module_2\\Pair Programming'
```

In [9]:
```
infile="Wicked_Free_WiFi_Locations.csv"

wifi=pd.read_csv(infile)
```

In [10]:
```
# head function,  called as a methd belonging to the dataframe (a python object) ca

# wifi is said to be an instance of a python dataframe

wifi.head(7)
```

Out[10]:

	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 [11]:

```
#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[11]:

```
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 [13]:
```
# Generating a Summary
# describes only numeric values

wifi.describe()
```

Out[13]:

	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 [15]:
```
#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 [17]:
```
type(n_name)
```

Out[17]:
```
pandas.core.series.Series
```

In [18]:
```
# we can convert this to a list, using a member function

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

Out[18]:
```
list
```

In [19]:
```
# dimensions of a dataframe are obtained using the attribute shape
print(wifi.shape)
```

```
print(n_name.shape)
```

```
(297, 17)
(297,)
```

In [20]:
```
#indexing several columns

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

Out[20]:

	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 [22]:
```
wifi.head
```

```
Out[22]: <bound method NDFrame.head of                X             Y       neighborhood_i
d  neighborhood_name  \
0             NaN           NaN  L_601230550253963849  Mobile WiFi Kit 1
1             NaN           NaN  L_601230550253963849  Mobile WiFi Kit 1
2   -7.912255e+06  5.206228e+06  L_601230550253964116    Nubian-Bus-Stop
3             NaN           NaN  L_601230550253964116    Nubian-Bus-Stop
4             NaN           NaN  L_601230550253964116    Nubian-Bus-Stop
..            ...           ...                   ...                ...
292 -7.910789e+06  5.214371e+06  N_601230550253961607        Maintenance
293 -7.912997e+06  5.210562e+06  N_601230550253961809            Bolling
294 -7.912997e+06  5.210562e+06  N_601230550253961809            Bolling
295 -7.912997e+06  5.210562e+06  N_601230550253961809            Bolling
296 -7.912997e+06  5.210562e+06  N_601230550253961809            Bolling

      device_serial    device_connectedto  \
0    Q3AK-SUL7-7FC4                MR76-1
1    Q2ZY-RF99-YN45                MG41-1
2    Q3AE-QFTK-E55W        ROX-Nubian-AP1
3    Q3AK-DU9C-2UXZ        ROX-Nubian_AP6
4    Q3AK-FGAN-3AR9        ROX-Nubian_AP7
..              ...                   ...
292  Q2CK-N8A5-VD4F  PARKS-COMMONWEST-AP3
293  Q2FD-MTWE-FN62  BOL-WELCOMELOBBY-AP1
294  Q2FD-6RML-C4S6      BOL-SCHLOBBY-AP1
295  Q2FD-3YYZ-LW7E      BOL-PUBLOBBY-AP1
296  Q2FD-3Q7F-9C4J         BOL-FOYER-AP1

                          device_address  device_lat  device_long  \
0                                    NaN         NaN          NaN
1                                    NaN         NaN          NaN
2    247 Washington St, Boston, MA 02121   42.301350   -71.077000
3                                    NaN         NaN          NaN
4                                    NaN         NaN          NaN
..                                   ...         ...          ...
292     139 Tremont St, Boston, MA 02111   42.355431   -71.063828
293   2300 Washington St., Roxbury 02119   42.330141   -71.083664
294   2300 Washington St., Roxbury 02119   42.330141   -71.083664
295   2300 Washington St., Roxbury 02119   42.330141   -71.083664
296   2300 Washington St., Roxbury 02119   42.330141   -71.083664

                               device_tags    etl_updatedtimestamp  \
0                                       []  2024/08/20 04:31:34+00
1                                       []  2024/08/20 04:31:34+00
2                      ['recently-added']  2024/08/20 04:31:38+00
3                                       []  2024/08/20 04:31:38+00
4                                       []  2024/08/20 04:31:38+00
..                                     ...                     ...
292                    ['recently-added']  2024/08/20 04:31:46+00
293  ['Bolling', 'CoB-Employee', 'Inside']  2024/08/20 04:31:46+00
294  ['Bolling', 'CoB-Employee', 'Inside']  2024/08/20 04:31:46+00
295                 ['Bolling', 'Inside']  2024/08/20 04:31:46+00
296  ['Bolling', 'CoB-Employee', 'Inside']  2024/08/20 04:31:46+00

     is_current org1 org2 inside_outside             landmark  ObjectId
0             1  NaN  NaN            NaN                  NaN         1
1             1  NaN  NaN            NaN                  NaN         2
```

```
2             1  NaN  NaN         NaN                    NaN       3
3             1  NaN  NaN         NaN                    NaN       4
4             1  NaN  NaN         NaN                    NaN       5
..          ...  ...  ...         ...                    ...     ...
292           1  NaN  NaN         NaN                    NaN     293
293           1  NaN  NaN      Inside  Bolling CoB Employee      294
294           1  NaN  NaN      Inside  Bolling CoB Employee      295
295           1  NaN  NaN      Inside                Bolling     296
296           1  NaN  NaN      Inside  Bolling CoB Employee      297

[297 rows x 17 columns]>
```

In [23]:

```
#Basic calculations

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

```
42.38608
42.2507393
42.32779576223686
```

In [24]:

```
# 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[24]:

	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 [25]:
```
#slicing by values in a set, notice that pandas has a isin() member function for th

wifi[wifi.neighborhood_name.isin(["Parks","Charlestown"])]
```

Out[25]:

	X	Y	neighborhood_id	neighborhood_name	device_seri
5	-7.913037e+06	5.209642e+06	N_568579452955527921	Parks	Q2EK 4PWN-GAL
28	-7.910979e+06	5.214437e+06	N_568579452955527921	Parks	Q3AK-CVAE CUZ
44	-7.916707e+06	5.202882e+06	N_568579452955527921	Parks	Q2AK-U4TT J95
188	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-V3LS 5V6
202	-7.910805e+06	5.214387e+06	N_568579452955527921	Parks	Q3AK-DGSZ GM7
203	-7.911261e+06	5.214274e+06	N_568579452955527921	Parks	Q3AK-EK6U T4FV
205	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK CWUY RBGV
214	-7.911012e+06	5.214442e+06	N_568579452955527921	Parks	Q3AK-DRLE LEZ
223	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-SNTE WJW
226	-7.911133e+06	5.214283e+06	N_568579452955527921	Parks	Q3AK-CR6K YPB
227	-7.916761e+06	5.208413e+06	N_568579452955527921	Parks	Q2CK-7ANU RF7
228	-7.910954e+06	5.213995e+06	N_568579452955527921	Parks	Q3AK-CR92 A99
229	-7.916761e+06	5.208413e+06	N_568579452955527921	Parks	Q2CD-6YGI H7PV

	X	Y	neighborhood_id	neighborhood_name	device_seri
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 [26]:
```
# 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[26]:

	X	Y	neighborhood_id	neighborhood_name	device_seri
53	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-4FL SJ
54	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-HJR D68
55	-7.909698e+06	5.215107e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ARN HFA
56	-7.909702e+06	5.215082e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3A PMFQ-JUX
57	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-3S8 CZ
58	-7.909943e+06	5.215065e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AK-C98 JUR
59	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-TBT D72
61	-7.909943e+06	5.215065e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3A D5DU-9HS
62	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ZUF Y73
69	-7.909890e+06	5.215063e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-F6R VVH
70	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-ZXW H9Z
71	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-3A2 JNV
152	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-JBP DSN
153	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-ZDP 2LN

	X	Y	neighborhood_id	neighborhood_name	device_seria
154	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-QYJ5 RXI
155	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE 2N4M BWU
156	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-TF7U 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 MFW
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-M9YS PAT
161	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-E75T LLQ
162	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-BFV5 YGN
163	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-8YNT BM6
164	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE CANN-4NB
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_seri
169	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-A94 FE
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-JXU QPC
172	-7.910171e+06	5.215103e+06	N_601230550253966673	Downtown Boston - City Hall Plaza and Pavilion	Q3AE-A79 TEM
188	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-V3L 5V
205	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2C CWU RBG
223	-7.910482e+06	5.218089e+06	N_568579452955538062	Charlestown	Q2CK-SNT WJW
256	-7.910171e+06	5.215103e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-GDT 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-JLU 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-Q4L 6HK
289	-7.909896e+06	5.215085e+06	N_601230550253961310	Downtown Boston - City Hall - Quincy Market	Q3AE-Q7Y V97

In [27]:
```
# 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[27]:

	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 [28]:
```
# Row and column specification
# use paired conctions on [row,column]
# we now have to use the method .loc[] to do this

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

Out[28]:
```
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 [29]:
```
# 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[29]:

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

In [30]:

```
# Indexing using integer values is done using the iloc[] function

# so remember- used .loc for boolean and named columns,   .iloc for Integer locatio

wifi.iloc[0:8,0:6]
```

Out[30]:

	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 [32]:

```
#Pandas built in plots

wifi.plot.scatter(x="device_long",y="device_lat")
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[32], line 3
      1 #Pandas built in plots
----> 3 wifi.plot.scatter(x="device_long",y="device_lat")

File ~\anaconda3\envs\Class5002\Lib\site-packages\pandas\plotting\_core.py:1748, in
PlotAccessor.scatter(self, x, y, s, c, **kwargs)
   1660 def scatter(
   1661     self,
   1662     x: Hashable,
   (...)
   1666     **kwargs,
   1667 ) -> PlotAccessor:
   1668     """
   1669     Create a scatter plot with varying marker point size and color.
   1670 
   (...)
   1746         ...                         colormap='viridis')
   1747     """
-> 1748     return self(kind="scatter", x=x, y=y, s=s, c=c, **kwargs)

File ~\anaconda3\envs\Class5002\Lib\site-packages\pandas\plotting\_core.py:947, in P
lotAccessor.__call__(self, *args, **kwargs)
    946 def __call__(self, *args, **kwargs):
--> 947     plot_backend = _get_plot_backend(kwargs.pop("backend", None))
    949     x, y, kind, kwargs = self._get_call_args(
    950         plot_backend.__name__, self._parent, args, kwargs
    951     )
    953     kind = self._kind_aliases.get(kind, kind)

File ~\anaconda3\envs\Class5002\Lib\site-packages\pandas\plotting\_core.py:1944, in
_get_plot_backend(backend)
   1941 if backend_str in _backends:
   1942     return _backends[backend_str]
-> 1944 module = _load_backend(backend_str)
   1945 _backends[backend_str] = module
   1946 return module

File ~\anaconda3\envs\Class5002\Lib\site-packages\pandas\plotting\_core.py:1874, in
_load_backend(backend)
   1872         module = importlib.import_module("pandas.plotting._matplotlib")
   1873     except ImportError:
-> 1874         raise ImportError(
   1875             "matplotlib is required for plotting when the "
   1876             'default backend "matplotlib" is selected.'
   1877         ) from None
   1878     return module
   1880 found_backend = False

ImportError: matplotlib is required for plotting when the default backend "matplotli
b" is selected.
```

In [ ]:

```
#here is a boxplot
```

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

In [ ]:
```
#histogram

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

In [ ]:
```
#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()
```

Aggregation or grouping for tables and statistics

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

In [ ]:
```
# 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")
```

Multiple grouping variables

In [ ]:
```
# 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
```

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

```
wifi.head()
```

In [ ]:
```
# did this work

wifi.dtypes
```

Question/Action

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

Convert this variable to a category

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

wifi.head()
```

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, logisitic 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 [ ]:
```
pd.get_dummies(wifi.neighborhood_name)
```

Question/Action

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

In [59]:
```
pd.get_dummies(wifi.neighborhood_id)
```

Out[59]:

	L_601230550253962684	L_601230550253962688	L_601230550253963849	L_601230550
0	False	False	True	
1	False	False	True	
2	False	False	False	
3	False	False	False	
4	False	False	False	
...	...	...	...	
292	False	False	False	
293	False	False	False	
294	False	False	False	
295	False	False	False	
296	False	False	False	

297 rows × 34 columns

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

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

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

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

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

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

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 [ ]:
```
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
```

In [ ]:
```

```

In [ ]:
```
# 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'])
```

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

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

In [75]:
```
# 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"}})
```

Question/Action

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

In [85]:
```
df2.melt
```

Out[85]:
```
<bound method DataFrame.melt of    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>
```

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

In [ ]:
```
df1
```

In [ ]:
```
df2
```

In [ ]:
```
# 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')
```

In [ ]:
```
# here is a left join

# what happens to "bix"?

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: