pandas

April 8, 2023

1 pandas and Data Wrangling

1.1 Introduction

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language1.

GeeksforGeeks describes **Data wrangling** as, "the process of gathering, collecting, and transforming Raw data into another format for better understanding, decision-making, accessing, and analysis in less time. Data Wrangling is also known as Data Munging."2

This session will provide an overview of common Pandas' API (methods) used to transform your datasets.

We will learn to : - Install Pandas - Define the DataFrame and Series classes - Work with columns and rows - Filter rows - Apply or update data - ~~Merge and~~ concatenate datasets - Read and ~~write~~ to a file - Do more, time permitting

This presentation assumes that Python 3.7.x or greater is already installed on your Raspberry Pi or other SBC (or Windows, Mac, or Linux system).

1.1.1 Notes

- pandas relies on a variety of different libraries for reading and writing data from remote resources depending on the protocol. Support for IPv6 should be documented by the authors of the libraries.
- Security issues with pandas are beyond the scope of this presentation.

1.1.2 Coding Style/Syntax/Nomenclature

Where possible (and I remember),

- function/method arguments/parameters will include the keyword. For example, instead of calling a function using `quadratic(1,2,3)`, I will use `quadratic(a=1, b=2, c=3)`.
- pass a list of values when the parameter accepts either a single item or a list of items. For example, I use `foo(data=['1'])` rather than `foo(data='1')`.
- stick with 'method' and 'parameters' rather than 'function' and 'arguments' respectively, because pandas is object-oriented and uses that terminology in the documentation.

1.2 Install pandas

Prerequisites * python >= 3.7.x * pip * git [optional] * python-virtualenv [recommended]

To install pandas, using pip on the command line

```
pip install pandas
```

Details on installing pandas via Anaconda, Miniconda, Linux Distro, etc. can be found here: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html

To use pandas in your code, simply import the module. The convention is to rename the module to 'pd' during the import.

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

1.3 Define Panda Data Structures

1.3.1 Series

A *Series* is an one-dimension ndarray (N-dimensional array), with labels, that can hold any Python data type (integers, floats, strings, objects). The labels along the axis are called the index.

Conceptually, a visual representation of a series, with labels "a", "b", "c" and values 1.0, 2.0, 3.0, respectively is:

index	data
"a"	1.0
"b"	2.0
"c"	3.0

Labels should be a hashable type, such as an integer, float, or string. The labels don't need to be unique and can be different types. (I would advise against duplicate labels and mixing types.)

Creating Series objects Series of integers, with pandas defining the index.

Notice that the first column of the output is the index and the second column contains the values. The last line is the data type of all the elements.

[2]:
```
pd.Series(data=[1,2,3])
```

[2]:
```
0    1
1    2
2    3
dtype: int64
```

Series of strings, with labels provided.

By default, strings are considered objects but we can override that by specifying the data type.

[3]:
```
pd.Series(data=['a','b','c'], index=['a','b','c'], dtype='string')
```

[3]:
```
a    a
b    b
c    c
```

```
dtype: string
```

Series with different data types and labels provided. The data type is 'object' because of the mixed values.

[4]:
```
pd.Series(data=['a',1,2.0,[1,2,3],{'x': 1.0, 'y': 2.0, 'z': 4.0}],
          index=['string','int','float','list','dict'])
```

[4]:
```
string                                 a
int                                    1
float                                2.0
list                          [1, 2, 3]
dict      {'x': 1.0, 'y': 2.0, 'z': 4.0}
dtype: object
```

Accessing Series object values First value of a series with a default index. (Remember that Python uses zero-based indexes, so 0 refers to the first element.)

[5]:
```
series = pd.Series(data=[1,2,3])
label = 0

series[label]
```

[5]:
```
1
```

Value of the series at the label 'c'.

[6]:
```
series = pd.Series(data=[1,2,3], index=['a','b','c'])
label = 'c'

series[label]
```

[6]:
```
3
```

An error occurs if we try to access a label that doesn't exist in the Series.

[7]:
```
try:
    print(series[1000])
except IndexError:
    print('IndexError')

try:
    print(series['invalid'])
except KeyError:
    print('KeyError')
```

```
IndexError
KeyError
```

Value of a specific list item at the label 'list'.

[8]:
```
series = pd.Series(data=['a',1,2.0,[1,2,3],{'x': 1.0, 'y': 2.0, 'z': 4.0}],
                   index=['string','int','float','list','dict'])
label = 'list'
index = 1

series[label][index]
```

[8]: 2

Dictionary value at the label 'dict' with the key 'y'.

[9]:
```
series = pd.Series(data=['a',1,2.0,[1,2,3],{'x': 1.0, 'y': 2.0, 'z': 4.0}],
                   index=['string','int','float','list','dict'])
label = 'dict'
key   = 'y'

series[label][key]
```

[9]: 2.0

Manipulating Series objects These are various ways of changing, adding, or removing values in a Series.

[10]:
```
series = pd.Series(data=[1,2,3])

# change a value
series[0] = 1000

# append a value
series[len(series)] = '4'

# remove an row
series = series.drop(labels=[1])

# remove row without reassignment; default for inplace is False
series.drop(labels=[3], inplace=True)

# add values at arbitrary indices
series[9] = '10'
series['ten'] = '11'

print(series)
```

```
0      1000
2         3
9        10
ten      11
dtype: object
```

Reset the index to a default integer index. (Notice that the original index is preserved as a new column. Use the parameter, `drop=True`, to exclude it.)

[11]:
```
series.reset_index()
```

[11]:
```
   index     0
0      0  1000
1      2     3
2      9    10
3    ten    11
```

1.3.2 DataFrame

A *DataFrame* is an two-dimension array, with labels for both the rows (index) and columns. Each column can hold any Python data type (integers, floats, strings, objects). Think of a DataFrame as a SQL table or spreadsheet file.

Labels for both axes should be a hashable type, such as an integer, float, or string. The labels don't need to be unique and can be different types. (I would advise against duplicate labels and mixing types.)

Creating DataFrame objects DataFrame, with a column of integers and another of strings, with pandas defining the index.

[12]:
```
df = pd.DataFrame(data = [[1,'a'],
                          [2,'b'],
                          [3,'c']])

df
```

[12]:
```
   0  1
0  1  a
1  2  b
2  3  c
```

DataFrame, with a column of integers and another of strings, with a index and column labels provided.

[13]:
```
df = pd.DataFrame(data = [[1,'a'],
                          [2,'b'],
                          [3,'c']],
                  index = ['row_1','row_2','row_3'],
                  columns = ['column_1','column_2'])
```

```
df
```

[13]:
```
        column_1 column_2
row_1          1        a
row_2          2        b
row_3          3        c
```

DataFrame constructed from a dictionary of series, with a column of integers and another of strings, with pandas defining the index and column labels provided.

[14]:
```
df = pd.DataFrame(data = {'column_1' : [1, 2, 3],
                          'column_2' : ['a','b','c']})

df
```

[14]:
```
   column_1 column_2
0         1        a
1         2        b
2         3        c
```

Accessing DataFrame object values

[15]:
```
# DataFrame for section
df = pd.DataFrame(data = [[1,'a',1.0,'abc'],
                          [2,'b',2.0,'def'],
                          [3,'c',3.0,'ghi']])
```

Select first column of a DataFrame with a default index.

[16]:
```
column_label = 0

df[column_label]
```

[16]:
```
0    1
1    2
2    3
Name: 0, dtype: int64
```

Use Python slices to access row(s):

[17]:
```
df[2:3]
```

[17]:
```
   0  1    2    3
2  3  c  3.0  ghi
```

Select multiple columns, in a specific order, and slices to access row(s).

[18]:
```
column_labels = [2,0]

print(df[column_labels][-1:])
```

```
     2  0
2  3.0  3
```

Manipulating DataFrame objects

[19]:
```
# DataFrame for section
df = pd.DataFrame(data = [[1,'a',1.0,'abc'],
                          [2,'b',2.0,'def'],
                          [3,'c',3.0,'ghi']])
```

Rename one or more columns by passing a dictionary of the mappings, with each key being the existing column position or name and value is new name.

[20]:
```
df.rename(columns={0:'integers'}, inplace=True)

df
```

[20]:

	integers	1	2	3
0	1	a	1.0	abc
1	2	b	2.0	def
2	3	c	3.0	ghi

Reassign the value when `inplace=True` isn't used.

[21]:
```
df = df.rename(columns={'ints':'foo'})
df
```

[21]:

	integers	1	2	3
0	1	a	1.0	abc
1	2	b	2.0	def
2	3	c	3.0	ghi

Drop one or more columns.

[22]:
```
df.drop(columns=[1,2])
```

[22]:

	integers	3
0	1	abc
1	2	def
2	3	ghi

1.4 Data Wrangling Exercise

The best way to understand pandas is by working on a dataset. This exercise will use a dataset, containing a list of repeaters in BC, scraped from https://bcarcc.org/frequency_list into a CSV.

Read the data pandas has many methods for reading data from a variety of sources, such as CSV, JSON, SQL, etc…

The method for reading a CSV has many parameters, such as specifying the seperator character, quote character, line termination type. In this demonstration, we will simply pass the name and use the parameter defaults - specified in the documentation.

My preference is to allow pandas to define the index, rather than using an existing column that may not contain unique values.

[23]:
```
csv_df = pd.read_csv("data/bcarcc-repeater_list.csv")

csv_df
```

[23]:
```
     Band  Rpt Xmit   Rpt Rec        Location    call     Mode   Access  \
0     440  442.4500  447.4500      Salmon Arm  VE7RAM       FM      NaN
1     144  146.1600  146.7600      Salmon Arm  VE7RNH       FM      NaN
2     144  146.6400  146.0400        Sorrento  VE7RXX       FM      NaN
3     440  443.1000  448.1000  Port Coquitlam  VE7UDX  FM/C4FM   94.8HZ
4      52   52.8900   51.1900       Vancouver  VE7HCP       FM      NaN
..    …       …       …              …     …      …      …
482   144  147.2200  147.8200          Vernon  VA7VMR       FM  123.0Hz
483   440  449.7250  444.7250         Nanaimo  VE7DJA       FM  141.3Hz
484   440  440.0250  445.0250    Dawson Creek  VE7DQZ      DMR      CC1
485   144  145.5100  144.9100     Dawsn Creek  VE7DQZ      DMR      CC1
486   144  145.5100  144.9100   Fort St. John  VE7YXJ  Dig,pkt      NaN

         Sponsor                      Region
0         VE7LOG                   BC SOUTH
1    Shuswap ARC                   BC SOUTH
2         VE7EHL                   BC SOUTH
3     EPCOM Club  LOWER MAINLAND &amp; COAST
4         VE7HCP  LOWER MAINLAND &amp; COAST
..           …                         …
482       VE7OHM                   BC SOUTH
483         NARA            VANCOUVER ISLAND
484       VE7SST                   BC NORTH
485       VE7SST                   BC NORTH
486       VE7SST                   BC NORTH

[487 rows x 9 columns]
```

Notice that Jupyter notebook has nicely formatted the output of the DataFrame. The continuation ellipses are a function of pandas and can be overwritten by:

```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Given that the dataset has close to five hundred rows, this example will keep the default display

options.

Show the size of the table.

[24]:
```
csv_df.shape
```

[24]:
```
(487, 9)
```

Show the (default) index.

[25]:
```
csv_df.index
```

[25]:
```
RangeIndex(start=0, stop=487, step=1)
```

Show the column names.

[26]:
```
csv_df.columns
```

[26]:
```
Index(['Band', 'Rpt Xmit', 'Rpt Rec', 'Location', 'call', 'Mode', 'Access',
       'Sponsor', 'Region'],
      dtype='object')
```

Rename the columns so the labels are all lowercase and more descriptive.

[27]:
```
csv_df.columns =␣
 ↪['band','transmit','receive','location','callsign','mode','access','sponsor','region']

csv_df.head()
```

[27]:

	band	transmit	receive	location	callsign	mode	access	sponsor	region
0	440	442.4500	447.4500	Salmon Arm	VE7RAM	FM	NaN	VE7LOG	BC SOUTH
1	144	146.1600	146.7600	Salmon Arm	VE7RNH	FM	NaN	Shuswap ARC	BC SOUTH
2	144	146.6400	146.0400	Sorrento	VE7RXX	FM	NaN	VE7EHL	BC SOUTH
3	440	443.1000	448.1000	Port Coquitlam	VE7UDX	FM/C4FM	94.8HZ	EPCOM Club	LOWER MAINLAND & COAST
4	52	52.8900	51.1900	Vancouver	VE7HCP	FM	NaN	VE7HCP	LOWER MAINLAND & COAST

Sort the DataFrame by one column, without reassignment.

[28]:
```
csv_df.sort_values(by=['band'])
```

[28]:

	band	transmit	receive	location	callsign	mode	access
159	52	52.9700	51.2700	Victoria	VE7RFR	FM	NaN
58	52	52.9100	51.2100	Anvil Island	VE7QRO	FM	NaN

```
4      52     52.8900     51.1900       Vancouver   VE7HCP      FM      NaN
389    52     52.8700     51.1700          Duncan   VE7AQW      FM  141.3Hz
36     52     52.8300     51.1300        Victoria   VE7RSX      FM  100.0Hz
..    …         …          …              …       …     …       …
191  1200  1,291.9400  1,271.9400       Vancouver   VE7RAG      FM      NaN
408  1200  1,290.5000  1,270.5000       Coquitlam    VE7NZ  D-Star      NaN
142  1200  1,289.0000  2,434.0000       Vancouver   VE7VHF     ATV      NaN
450  1200  1,292.2000  1,272.2000  Campbell River   VE7ZZH  D-STAR      NaN
340  1200  1,247.0000  1,259.0000         Kelowna   VA7DIG  D-Star      NaN

                sponsor                      region
159              VE7DAT           VANCOUVER ISLAND
58               VE7LWB  LOWER MAINLAND &amp; COAST
4                VE7HCP  LOWER MAINLAND &amp; COAST
389               CVARS           VANCOUVER ISLAND
36                 WARA           VANCOUVER ISLAND
..                  …                         …
191              BCFMCA  LOWER MAINLAND &amp; COAST
408               VE7NZ  LOWER MAINLAND &amp; COAST
142              VE7HMW  LOWER MAINLAND &amp; COAST
450  Campbell River ARS           VANCOUVER ISLAND
340               OCARC                   BC SOUTH

[487 rows x 9 columns]
```

Sort the DataFrame by multiple columns, without reassignment.

[29]:
```
csv_df.sort_values(by=['band','region'])
```

[29]:
```
     band    transmit     receive        location callsign    mode   access
4      52     52.8900     51.1900       Vancouver   VE7HCP      FM      NaN  \
58     52     52.9100     51.2100    Anvil Island   VE7QRO      FM      NaN
36     52     52.8300     51.1300        Victoria   VE7RSX      FM  100.0Hz
159    52     52.9700     51.2700        Victoria   VE7RFR      FM      NaN
389    52     52.8700     51.1700          Duncan   VE7AQW      FM  141.3Hz
..    …         …          …              …       …     …       …
191  1200  1,291.9400  1,271.9400       Vancouver   VE7RAG      FM      NaN
408  1200  1,290.5000  1,270.5000       Coquitlam    VE7NZ  D-Star      NaN
311  1200  1,291.5000  1,271.5000        Victoria   VE7VIC  D-Star      NaN
312  1200  1,298.7500  1,298.7500        Victoria   VE7VIC     NaN      NaN
450  1200  1,292.2000  1,272.2000  Campbell River   VE7ZZH  D-STAR      NaN

                sponsor                      region
4                VE7HCP  LOWER MAINLAND &amp; COAST
58               VE7LWB  LOWER MAINLAND &amp; COAST
36                 WARA           VANCOUVER ISLAND
159              VE7DAT           VANCOUVER ISLAND
```

```
389               CVARS            VANCOUVER ISLAND
..                  …                           …
191              BCFMCA  LOWER MAINLAND &amp; COAST
408               VE7NZ  LOWER MAINLAND &amp; COAST
311                WARA            VANCOUVER ISLAND
312                WARA            VANCOUVER ISLAND
450  Campbell River ARS            VANCOUVER ISLAND

[487 rows x 9 columns]
```

Replace the 'NaN' values with an empty string.

[30]:
```
csv_df.fillna('', inplace=True)

csv_df
```

[30]:
```
     band  transmit   receive        location callsign     mode   access
0     440  442.4500  447.4500      Salmon Arm   VE7RAM       FM           \
1     144  146.1600  146.7600      Salmon Arm   VE7RNH       FM
2     144  146.6400  146.0400        Sorrento   VE7RXX       FM
3     440  443.1000  448.1000  Port Coquitlam   VE7UDX  FM/C4FM   94.8HZ
4      52   52.8900   51.1900       Vancouver   VE7HCP       FM
..    …       …       …              …      …      …      …
482   144  147.2200  147.8200          Vernon   VA7VMR       FM  123.0Hz
483   440  449.7250  444.7250         Nanaimo   VE7DJA       FM  141.3Hz
484   440  440.0250  445.0250    Dawson Creek   VE7DQZ      DMR      CC1
485   144  145.5100  144.9100     Dawsn Creek   VE7DQZ      DMR      CC1
486   144  145.5100  144.9100   Fort St. John   VE7YXJ  Dig,pkt

         sponsor                      region
0         VE7LOG                   BC SOUTH
1    Shuswap ARC                   BC SOUTH
2         VE7EHL                   BC SOUTH
3     EPCOM Club  LOWER MAINLAND &amp; COAST
4         VE7HCP  LOWER MAINLAND &amp; COAST
..           …                           …
482       VE7OHM                   BC SOUTH
483         NARA            VANCOUVER ISLAND
484       VE7SST                   BC NORTH
485       VE7SST                   BC NORTH
486       VE7SST                   BC NORTH

[487 rows x 9 columns]
```

Search for all repeaters that are located in Victoria.

[31]:
```
csv_df[csv_df['location'] == 'Victoria']
```

[31]:

	band	transmit	receive	location	callsign	mode	access	\
10	440	444.8750	449.8750	Victoria	VE7VIC	FM		
28	440	444.2500	449.2500	Victoria	VE7IA	FM		
30	440	449.8750	146.2400	Victoria	VE7XIC			
36	52	52.8300	51.1300	Victoria	VE7RSX	FM	100.0Hz	
53	440	443.5750	448.5750	Victoria	VE7RAA	FM		
94	144	146.8400	146.2400	Victoria	VE7VIC	FM	100.0Hz	
104	144	147.1200	147.7200	Victoria	VE7RBA	FM	100.0Hz	
159	52	52.9700	51.2700	Victoria	VE7RFR	FM		
170	144	147.2400	147.8400	Victoria	VE7RFR	FM		
171	440	442.7000	447.7000	Victoria	VE7FRF	FM		
194	440	443.9000	448.9000	Victoria	VE7RFR	FM	141.3Hz	
254	440	443.8250	448.8250	Victoria	VE7VEP	FM	100.0Hz	
260	440	444.1500	449.1500	Victoria	VE7US	FM	103.5Hz	
261	144	145.1300	144.5300	Victoria	VE7US	FM	114.8Hz	
284	440	442.0000	447.0000	Victoria	VE7VIC	D-Star		
286	144	145.0800	145.6800	Victoria	VE7VIC	D-Star		
305	220	224.1400	222.5400	Victoria	VE7VIC	FM	100.0Hz	
306	440	443.9500	448.9500	Victoria	VE7RTC	FM	123.0Hz	
311	1200	1,291.5000	1,271.5000	Victoria	VE7VIC	D-Star		
312	1200	1,298.7500	1,298.7500	Victoria	VE7VIC			
371	440	440.8750	445.8750	Victoria	VE7RUV	FM		
417	144	145.5500	144.9500	Victoria	VE7VIC	DMR		

	sponsor	region
10	WARA	VANCOUVER ISLAND
28	VE7IA	VANCOUVER ISLAND
30	WARA	VANCOUVER ISLAND
36	WARA	VANCOUVER ISLAND
53	VE7CCD	VANCOUVER ISLAND
94	WARA	VANCOUVER ISLAND
104	VE7IA	VANCOUVER ISLAND
159	VE7DAT	VANCOUVER ISLAND
170	VE7DAT	VANCOUVER ISLAND
171	VE7DAT	VANCOUVER ISLAND
194	VE7DAT	VANCOUVER ISLAND
254	Victoria EMA	VANCOUVER ISLAND
260	VE7RWS	VANCOUVER ISLAND
261	VE7RWS	VANCOUVER ISLAND
284	WARA	VANCOUVER ISLAND
286	WARA	VANCOUVER ISLAND
305	WARA	VANCOUVER ISLAND
306	WARA	VANCOUVER ISLAND
311	WARA	VANCOUVER ISLAND
312	WARA	VANCOUVER ISLAND
371	Uvic exp. Radio gp.	VANCOUVER ISLAND
417	WARA	VANCOUVER ISLAND

The statement `csv_df[csv_df['location'] == 'Victoria']` instructs pandas to evaluate the 'location' value for each row and create a boolean array. pandas then uses this array to filter the DataFrame.

The following conditions for filtering are permitted: <,>,==,<=,>=,!=

To negate the filter, use the tilde before the condition. For example, to find all locations not in Victoria, use `csv_df[~csv_df['location'] == 'Victoria']`

Since the resulting DataFrame wasn't reassigned to itself or to a new variable, the filtering isn't preserved.

You may also see columns referenced using a dot notation. For example, `csv_df['location']` is the same as csv_df.location. I prefer using square brackets so I can pass use a list to display more than one column.

[32]:
```
csv_df['location'] == 'Victoria'
```

[32]:
```
0      False
1      False
2      False
3      False
4      False
       …
482    False
483    False
484    False
485    False
486    False
Name: location, Length: 487, dtype: bool
```

The same filter as above but using `loc` and dropping two columns, without reassignment.

[33]:
```
csv_df.loc[csv_df['location'] == 'Victoria'].drop(columns=['location','region'])
```

[33]:

	band	transmit	receive	callsign	mode	access	\
10	440	444.8750	449.8750	VE7VIC	FM		\
28	440	444.2500	449.2500	VE7IA	FM		
30	440	449.8750	146.2400	VE7XIC			
36	52	52.8300	51.1300	VE7RSX	FM	100.0Hz	
53	440	443.5750	448.5750	VE7RAA	FM		
94	144	146.8400	146.2400	VE7VIC	FM	100.0Hz	
104	144	147.1200	147.7200	VE7RBA	FM	100.0Hz	
159	52	52.9700	51.2700	VE7RFR	FM		
170	144	147.2400	147.8400	VE7RFR	FM		
171	440	442.7000	447.7000	VE7FRF	FM		
194	440	443.9000	448.9000	VE7RFR	FM	141.3Hz	
254	440	443.8250	448.8250	VE7VEP	FM	100.0Hz	
260	440	444.1500	449.1500	VE7US	FM	103.5Hz	
261	144	145.1300	144.5300	VE7US	FM	114.8Hz	

```
284   440    442.0000    447.0000   VE7VIC  D-Star
286   144    145.0800    145.6800   VE7VIC  D-Star
305   220    224.1400    222.5400   VE7VIC      FM  100.0Hz
306   440    443.9500    448.9500   VE7RTC      FM  123.0Hz
311  1200  1,291.5000  1,271.5000   VE7VIC  D-Star
312  1200  1,298.7500  1,298.7500   VE7VIC
371   440    440.8750    445.8750   VE7RUV      FM
417   144    145.5500    144.9500   VE7VIC     DMR

                sponsor
10                 WARA
28                VE7IA
30                 WARA
36                 WARA
53                VE7CCD
94                 WARA
104               VE7IA
159               VE7DAT
170               VE7DAT
171               VE7DAT
194               VE7DAT
254         Victoria EMA
260               VE7RWS
261               VE7RWS
284                WARA
286                WARA
305                WARA
306                WARA
311                WARA
312                WARA
371  Uvic exp. Radio gp.
417                WARA
```

pandas can create a boolean array for filtering based on multiple conditions. Unlike other languages the 'and' condition is one ampersand (&) and, the 'or' is one pipe (|). Each condition **must** be surrounded by parentheses. If you want to negate the all of the conditions, you must then wrap the conditions with parentheses and use a tilde before the opening parenthesis (or switch up your logic). You can also negate individual conditions by including a tilde before the opening parenthesis of the condition.

[34]:
```
csv_df[(csv_df['location'] == 'Victoria') & (csv_df['sponsor'] != 'WARA')]
```

[34]:
```
     band  transmit   receive  location callsign mode   access
28    440  444.2500  449.2500  Victoria    VE7IA   FM           \
53    440  443.5750  448.5750  Victoria   VE7RAA   FM
104   144  147.1200  147.7200  Victoria   VE7RBA   FM  100.0Hz
159    52   52.9700   51.2700  Victoria   VE7RFR   FM
```

```
170   144  147.2400  147.8400  Victoria   VE7RFR   FM
171   440  442.7000  447.7000  Victoria   VE7FRF   FM
194   440  443.9000  448.9000  Victoria   VE7RFR   FM  141.3Hz
254   440  443.8250  448.8250  Victoria   VE7VEP   FM  100.0Hz
260   440  444.1500  449.1500  Victoria    VE7US   FM  103.5Hz
261   144  145.1300  144.5300  Victoria    VE7US   FM  114.8Hz
371   440  440.8750  445.8750  Victoria   VE7RUV   FM

                 sponsor            region
28                 VE7IA  VANCOUVER ISLAND
53                VE7CCD  VANCOUVER ISLAND
104                VE7IA  VANCOUVER ISLAND
159               VE7DAT  VANCOUVER ISLAND
170               VE7DAT  VANCOUVER ISLAND
171               VE7DAT  VANCOUVER ISLAND
194               VE7DAT  VANCOUVER ISLAND
254         Victoria EMA  VANCOUVER ISLAND
260               VE7RWS  VANCOUVER ISLAND
261               VE7RWS  VANCOUVER ISLAND
371  Uvic exp. Radio gp.  VANCOUVER ISLAND
```

pandas has a method for removing duplicate values.

[35]:
```
csv_df['band'].drop_duplicates()
```

[35]:
```
0       440
1       144
4        52
8       220
140     902
141    1200
Name: band, dtype: int64
```

Imagine we want to know all the 2-metre and 70-centimetre repeaters

[36]:
```
bands = [144,440]

csv_df[csv_df['band'].isin(bands)]
```

[36]:
```
     band  transmit   receive        location callsign     mode   access
0     440  442.4500  447.4500      Salmon Arm   VE7RAM       FM           \
1     144  146.1600  146.7600      Salmon Arm   VE7RNH       FM
2     144  146.6400  146.0400        Sorrento   VE7RXX       FM
3     440  443.1000  448.1000  Port Coquitlam   VE7UDX  FM/C4FM   94.8HZ
5     144  146.8000  146.2000        Hazelton   VE7RHD       FM  100.0Hz
..    …       …       …              …      …      …      …
482   144  147.2200  147.8200          Vernon   VA7VMR       FM  123.0Hz
483   440  449.7250  444.7250         Nanaimo   VE7DJA       FM  141.3Hz
```

```
484   440  440.0250  445.0250    Dawson Creek   VE7DQZ      DMR      CC1
485   144  145.5100  144.9100     Dawsn Creek   VE7DQZ      DMR      CC1
486   144  145.5100  144.9100   Fort St. John   VE7YXJ  Dig,pkt

         sponsor                      region
0         VE7LOG                   BC SOUTH
1    Shuswap ARC                   BC SOUTH
2         VE7EHL                   BC SOUTH
3     EPCOM Club  LOWER MAINLAND &amp; COAST
5          BVARC                   BC NORTH
..           …                         …
482       VE7OHM                   BC SOUTH
483         NARA           VANCOUVER ISLAND
484       VE7SST                   BC NORTH
485       VE7SST                   BC NORTH
486       VE7SST                   BC NORTH

[442 rows x 9 columns]
```

Note quirk about using regex Does whole

[37]:
```
csv_df = csv_df.replace(to_replace='&amp;',value='&',regex=True)

csv_df
```

[37]:
```
     band  transmit   receive        location callsign     mode   access
0     440  442.4500  447.4500      Salmon Arm   VE7RAM       FM           \
1     144  146.1600  146.7600      Salmon Arm   VE7RNH       FM
2     144  146.6400  146.0400        Sorrento   VE7RXX       FM
3     440  443.1000  448.1000  Port Coquitlam   VE7UDX  FM/C4FM   94.8HZ
4      52   52.8900   51.1900       Vancouver   VE7HCP       FM
..    …       …         …              …       …      …      …
482   144  147.2200  147.8200          Vernon   VA7VMR       FM  123.0Hz
483   440  449.7250  444.7250         Nanaimo   VE7DJA       FM  141.3Hz
484   440  440.0250  445.0250    Dawson Creek   VE7DQZ      DMR      CC1
485   144  145.5100  144.9100     Dawsn Creek   VE7DQZ      DMR      CC1
486   144  145.5100  144.9100   Fort St. John   VE7YXJ  Dig,pkt

         sponsor                  region
0         VE7LOG               BC SOUTH
1    Shuswap ARC               BC SOUTH
2         VE7EHL               BC SOUTH
3     EPCOM Club  LOWER MAINLAND & COAST
4         VE7HCP  LOWER MAINLAND & COAST
..           …                     …
482       VE7OHM               BC SOUTH
483         NARA        VANCOUVER ISLAND
```

```
484       VE7SST                BC NORTH
485       VE7SST                BC NORTH
486       VE7SST                BC NORTH

[487 rows x 9 columns]
```

[38]:
```
csv_df['region'].replace(to_replace=r'^BC\s(.*)',value=r'BRITISH COLUMBIA \1',␣
↪regex=True, inplace=True)

csv_df
```

[38]:
```
     band  transmit   receive        location callsign     mode   access
0     440  442.4500  447.4500      Salmon Arm   VE7RAM       FM           \
1     144  146.1600  146.7600      Salmon Arm   VE7RNH       FM
2     144  146.6400  146.0400        Sorrento   VE7RXX       FM
3     440  443.1000  448.1000  Port Coquitlam   VE7UDX  FM/C4FM   94.8HZ
4      52   52.8900   51.1900       Vancouver   VE7HCP       FM
..    …       …         …              …       …      …       …
482   144  147.2200  147.8200          Vernon   VA7VMR       FM  123.0Hz
483   440  449.7250  444.7250         Nanaimo   VE7DJA       FM  141.3Hz
484   440  440.0250  445.0250    Dawson Creek   VE7DQZ      DMR      CC1
485   144  145.5100  144.9100     Dawsn Creek   VE7DQZ      DMR      CC1
486   144  145.5100  144.9100   Fort St. John   VE7YXJ  Dig,pkt

         sponsor                   region
0         VE7LOG  BRITISH COLUMBIA SOUTH
1    Shuswap ARC  BRITISH COLUMBIA SOUTH
2         VE7EHL  BRITISH COLUMBIA SOUTH
3     EPCOM Club  LOWER MAINLAND & COAST
4         VE7HCP  LOWER MAINLAND & COAST
..           …                       …
482       VE7OHM  BRITISH COLUMBIA SOUTH
483         NARA        VANCOUVER ISLAND
484       VE7SST  BRITISH COLUMBIA NORTH
485       VE7SST  BRITISH COLUMBIA NORTH
486       VE7SST  BRITISH COLUMBIA NORTH

[487 rows x 9 columns]
```

Advanced Exercise https://climate.weather.gc.ca/

https://sunrise-sunset.org/

[39]:
```
april_weather_data = {}
april_suntimes_data = {}

for year in range(2018,2023):
```

```
    april_weather_data[year] = pd.read_csv(f'data/
↪en_climate_hourly_BC_1018621_04-{year}_P1H.csv')
    april_suntimes_data[year] = pd.read_csv(f'data/
↪sunrise-sunset_victoria_bc_canada_{year}-04.csv')
```

[40]:
```
april_weather_data[2022][['Temp (°C)','Dew Point Temp (°C)','Rel Hum (%)']].
↪describe()
```

[40]:

	Temp (°C)	Dew Point Temp (°C)	Rel Hum (%)
count	719.000000	719.000000	716.000000
mean	7.358136	3.205702	77.370112
std	3.059450	2.981640	18.538975
min	-0.500000	-8.900000	26.000000
25%	5.200000	1.300000	63.000000
50%	7.500000	3.200000	78.000000
75%	9.500000	5.300000	96.000000
max	16.200000	11.700000	100.000000

[41]:
```
all_april_weather_data = pd.concat([april_weather_data[2018],
                                    april_weather_data[2019],
                                    april_weather_data[2020],
                                    april_weather_data[2021],
                                    april_weather_data[2022]],
                                   axis=1)

all_april_weather_data.shape
```

[41]:
```
(720, 150)
```

[42]:
```
# T(°F) = T(°C) × 9/5 + 32
april_weather_data[2022]['Temp (°F)'] = april_weather_data[2022]['Temp (°C)'] *␣
↪9/5 + 32

april_weather_data[2022][['Temp (°C)','Temp (°F)']].head()
```

[42]:

	Temp (°C)	Temp (°F)
0	5.3	41.54
1	4.0	39.20
2	3.8	38.84
3	3.8	38.84
4	3.0	37.40

1.5 Conclusion

This presentation is just the tip of the iceberg when it comes to the pandas API. I have been working with pandas for almost three years and discovered new methods and techniques for wrangling data.

1.6 References

1. https://pandas.pydata.org