From Excel to Python: Using Pandas for Data Analysis

Trista McKenzie

Dept. of Earth Sciences Tech Seminar

October 29, 2019

Why use Python Pandas over Microsoft Excel?

- Faster
- Less prone to human/software error
- Free
- More complicated analyses are easy to do

Quick Python Tutorial

Installation & Running Python:

- Anaconda Navigator (https://www.anaconda.com/)
- Google Colaboratory (Colab) (https://colab.research.google.com)

Python's Library (HUGE!)

- composed of *packages*
- Pandas is an example of a Python package

Importing packages into Python

Packages need to be imported - let's open Google Colab and do this now

In [9]:
```
import pandas as pd
```

What is Pandas?

Pandas = Python Data Analysis Library

Data Structures:

- Series (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html): like a single column or row in Excel
- Data Frames (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html): like the entire spreadsheet

In [48]:

```
# creating a Series

s1 = (1, 2, 3)
s2 = (3, 4, 5)

series = pd.Series(s1)

print(series)
```

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

In [54]:

```
# creating a DataFrame

df = pd.DataFrame([s1,s2], columns = ['a', 'b', 'c'])

print(df)
```

```
   a  b  c
0  1  2  3
1  3  4  5
```

Pandas Data Types

Data Type	Description
object	strings (letters)
int64	integers
float64	floats (numbers with decimal points)
bool	Boolean (true/false)
datetime64	Date and Time

In [12]:
```
# data types

df.dtypes
```

Out[12]:
```
0    int64
1    int64
2    int64
dtype: object
```

In [13]:

```
# creating another DataFrame with strings and integers

s3 = ('a', 'b', 'c')
s4 = (7, 8, 9)

df2 = pd.DataFrame([s3,s4])

print(df2)
```

```
   0  1  2
0  a  b  c
1  7  8  9
```

In [14]:
```
df2.dtypes
```

Out[14]:
```
0    object
1    object
2    object
dtype: object
```

Loading and Saving External Files

Loading an Excel file (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html)

```
df = pd.read_excel('filepath/filename', sheet_name ='sheet name')
```

Loading a CSV file (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

```
df = pd.read_csv('filepath/filename', skiprows = #)
```

Saving an Excel file (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_excel.html)

```
df.to_excel('filepath/filename')
```

Saving a CSV file (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html#pandas.DataFrame.to_csv)

```
df.to_csv('filepath/filename')
```

Viewing and Inspecting Data

First n rows (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.head.html#pandas.DataFrame.head)

```
df.head(n)
```

Last n rows (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.tail.html#pandas.DataFrame.tail)

```
df.tail(n)
```

Number of rows and columns (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shape.html#pandas.DataFrame.shape)

```
df.shape
```

Index, datatype, memory info (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html)

```
df.info()
```

Loading data into Google Colab

In [ ]:
```
# Mounting Google Drive in your VM (instructions from Google)

from google.colab import drive
drive.mount('/gdrive')
```

In [ ]:
```
# Import PyDrive and associated libraries.
# This only needs to be done once per notebook.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Download a file based on its file ID.
#
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
file_id = 'REPLACE_WITH_YOUR_FILE_ID'
downloaded = drive.CreateFile({'id': file_id})
print('Downloaded content "{}"'.format(downloaded.GetContentString()))
```

Example: Mauna Loa Monthly Average CO_2 dataset

Download file here (https://github.com/tamck/Pandas_tutorial/blob/master/co2_mm_mlo.csv)

Source: NOAA Earth System Research Laboratory Global Monitoring Division (https://www.esrl.noaa.gov/gmd/ccgg/trends/data.html)(downloaded 10/17/19)

This dataset contains the monthly mean CO_2 mole fraction in parts per million (ppm) from March 1958 through September 2019 measured from the top of Mauna Loa.

Tasks:

- load csv
- view first 5 rows
- view last 5 rows

In [42]:
```
CO2 = pd.read_csv("co2_mm_mlo.csv")
CO2 = pd.DataFrame(CO2, columns = ['Year', 'Month', 'Date', 'CO2_avg'])
CO2.head(5)
```

Out[42]:

	Year	Month	Date	CO2_avg
0	1958	3	1958.208	315.71
1	1958	4	1958.292	317.45
2	1958	5	1958.375	317.50
3	1958	6	1958.458	-99.99
4	1958	7	1958.542	315.86

In [43]:
```
CO2.tail(5)
```

Out[43]:

	Year	Month	Date	CO2_avg
734	2019	5	2019.375	414.66
735	2019	6	2019.458	413.92
736	2019	7	2019.542	411.77
737	2019	8	2019.625	409.95
738	2019	9	2019.708	408.54

Indexing and Data Selection

Indexing: selecting specific rows/columns

`DataFrame['label']` = index columns by label

`DataFrame.loc['label']` = index by label

`DataFrame.iloc[#]` = index by position or integer

In [44]:
```
# selecting a column based on label

CO2.loc[:,'Date']
```

Out[44]:
```
0      1958.208
1      1958.292
2      1958.375
3      1958.458
4      1958.542
5      1958.625
6      1958.708
7      1958.792
8      1958.875
9      1958.958
10     1959.042
11     1959.125
12     1959.208
13     1959.292
14     1959.375
15     1959.458
16     1959.542
17     1959.625
18     1959.708
19     1959.792
20     1959.875
21     1959.958
22     1960.042
23     1960.125
24     1960.208
25     1960.292
26     1960.375
27     1960.458
28     1960.542
29     1960.625
          ...
709    2017.292
710    2017.375
```

```
710    2017.375
711    2017.458
712    2017.542
713    2017.625
714    2017.708
715    2017.792
716    2017.875
717    2017.958
718    2018.042
719    2018.125
720    2018.208
721    2018.292
722    2018.375
723    2018.458
724    2018.542
725    2018.625
726    2018.708
727    2018.792
728    2018.875
729    2018.958
730    2019.042
731    2019.125
732    2019.208
733    2019.292
734    2019.375
735    2019.458
736    2019.542
737    2019.625
738    2019.708
Name: Date, Length: 739, dtype: float64
```

In [45]:
```
# selecting a column based on position

CO2.iloc[:,2]
```

Out[45]:
```
0       1958.208
1       1958.292
2       1958.375
3       1958.458
4       1958.542
5       1958.625
6       1958.708
7       1958.792
8       1958.875
9       1958.958
10      1959.042
11      1959.125
12      1959.208
13      1959.292
14      1959.375
15      1959.458
16      1959.542
17      1959.625
18      1959.708
19      1959.792
20      1959.875
21      1959.958
22      1960.042
23      1960.125
24      1960.208
25      1960.292
26      1960.375
27      1960.458
28      1960.542
29      1960.625
          ...
709     2017.292
710     2017.375
```

```
710    2017.375
711    2017.458
712    2017.542
713    2017.625
714    2017.708
715    2017.792
716    2017.875
717    2017.958
718    2018.042
719    2018.125
720    2018.208
721    2018.292
722    2018.375
723    2018.458
724    2018.542
725    2018.625
726    2018.708
727    2018.792
728    2018.875
729    2018.958
730    2019.042
731    2019.125
732    2019.208
733    2019.292
734    2019.375
735    2019.458
736    2019.542
737    2019.625
738    2019.708
Name: Date, Length: 739, dtype: float64
```

In [58]:
```
# selecting the same column

CO2['Date']
```

Out[58]:
```
0      1958.208
1      1958.292
2      1958.375
3      1958.458
4      1958.542
5      1958.625
6      1958.708
7      1958.792
8      1958.875
9      1958.958
10     1959.042
11     1959.125
12     1959.208
13     1959.292
14     1959.375
15     1959.458
16     1959.542
17     1959.625
18     1959.708
19     1959.792
20     1959.875
21     1959.958
22     1960.042
23     1960.125
24     1960.208
25     1960.292
26     1960.375
27     1960.458
28     1960.542
29     1960.625
          ...
709    2017.292
710    2017.375
```

```
710    2017.375
711    2017.458
712    2017.542
713    2017.625
714    2017.708
715    2017.792
716    2017.875
717    2017.958
718    2018.042
719    2018.125
720    2018.208
721    2018.292
722    2018.375
723    2018.458
724    2018.542
725    2018.625
726    2018.708
727    2018.792
728    2018.875
729    2018.958
730    2019.042
731    2019.125
732    2019.208
733    2019.292
734    2019.375
735    2019.458
736    2019.542
737    2019.625
738    2019.708
Name: Date, Length: 739, dtype: float64
```

In [65]:
```
# selecting multiple columns using the generic column indexer

CO2[['Date','CO2_avg']]
```

Out[65]:

	Date	CO2_avg
0	1958.208	315.71
1	1958.292	317.45
2	1958.375	317.50
3	1958.458	-99.99
4	1958.542	315.86
5	1958.625	314.93
6	1958.708	313.20
7	1958.792	-99.99
8	1958.875	313.33
9	1958.958	314.67
10	1959.042	315.62
11	1959.125	316.38
12	1959.208	316.71
13	1959.292	317.72
14	1959.375	318.29
15	1959.458	318.15
16	1959.542	316.54
17	1959.625	314.80
18	1959.708	313.84
19	1959.792	313.26
20	1959.875	314.80
21	1959.958	315.58
22	1960.042	316.43
23	1960.125	316.97
24	1960.208	317.58
25	1960.292	319.02
26	1960.375	320.03
27	1960.458	319.59
28	1960.542	318.18

	Date	CO2_avg
29	1960.625	315.91
...	...	...
709	2017.292	409.04
710	2017.375	409.69
711	2017.458	408.88
712	2017.542	407.12
713	2017.625	405.13
714	2017.708	403.37
715	2017.792	403.63
716	2017.875	405.12
717	2017.958	406.81
718	2018.042	407.96
719	2018.125	408.32
720	2018.208	409.41
721	2018.292	410.24
722	2018.375	411.23
723	2018.458	410.79
724	2018.542	408.71
725	2018.625	406.99
726	2018.708	405.51
727	2018.792	406.00
728	2018.875	408.02
729	2018.958	409.07
730	2019.042	410.83
731	2019.125	411.75
732	2019.208	411.97
733	2019.292	413.32
734	2019.375	414.66
735	2019.458	413.92
736	2019.542	411.77
737	2019.625	409.95
738	2019.708	408.54

739 rows × 2 columns

In [55]:
```
# selecting a row

CO2.iloc[2,:]
```

Out[55]:
```
Year       1958.000
Month         5.000
Date       1958.375
CO2_avg     317.500
Name: 2, dtype: float64
```

In [57]:
```
# selecting multiple columns

CO2.iloc[:,1:4]
```

Out[57]:

	Month	Date	CO2_avg
0	3	1958.208	315.71
1	4	1958.292	317.45
2	5	1958.375	317.50
3	6	1958.458	-99.99
4	7	1958.542	315.86
5	8	1958.625	314.93
6	9	1958.708	313.20
7	10	1958.792	-99.99
8	11	1958.875	313.33
9	12	1958.958	314.67
10	1	1959.042	315.62
11	2	1959.125	316.38
12	3	1959.208	316.71
13	4	1959.292	317.72
14	5	1959.375	318.29
15	6	1959.458	318.15
16	7	1959.542	316.54
17	8	1959.625	314.80
18	9	1959.708	313.84
19	10	1959.792	313.26
20	11	1959.875	314.80
21	12	1959.958	315.58
22	1	1960.042	316.43
23	2	1960.125	316.97
24	3	1960.208	317.58
25	4	1960.292	319.02
26	5	1960.375	320.03
27	6	1960.458	319.59
28	7	1960.542	318.18

	Month	Date	CO2_avg
29	8	1960.625	315.91
...	...	...	...
709	4	2017.292	409.04
710	5	2017.375	409.69
711	6	2017.458	408.88
712	7	2017.542	407.12
713	8	2017.625	405.13
714	9	2017.708	403.37
715	10	2017.792	403.63
716	11	2017.875	405.12
717	12	2017.958	406.81
718	1	2018.042	407.96
719	2	2018.125	408.32
720	3	2018.208	409.41
721	4	2018.292	410.24
722	5	2018.375	411.23
723	6	2018.458	410.79
724	7	2018.542	408.71
725	8	2018.625	406.99
726	9	2018.708	405.51
727	10	2018.792	406.00
728	11	2018.875	408.02
729	12	2018.958	409.07
730	1	2019.042	410.83
731	2	2019.125	411.75
732	3	2019.208	411.97
733	4	2019.292	413.32
734	5	2019.375	414.66
735	6	2019.458	413.92
736	7	2019.542	411.77
737	8	2019.625	409.95
738	9	2019.708	408.54

739 rows × 3 columns

In [ ]:

```
# selecting a specific cell

# select first element of first column
CO2.iloc[0,0]

# select specific date - later
```

Indexing by Date and Time

- useful for temporal, time series data

```
pd.to_datetime()
```

In [68]:

```
# create a new column that combines year and month and convert to datetime format
CO2['Date'] = pd.to_datetime(CO2[['Year', 'Month']].assign(Day = 1))

# indexing by time stamp, inplace = True means changes will occur in orignal dataframe
CO2.set_index(CO2['Date'], inplace = True)

CO2.head(5)
```

Out[68]:

	Year	Month	Date	CO2_avg
Date				
1958-03-01	1958	3	1958-03-01	315.71
1958-04-01	1958	4	1958-04-01	317.45
1958-05-01	1958	5	1958-05-01	317.50
1958-06-01	1958	6	1958-06-01	-99.99
1958-07-01	1958	7	1958-07-01	315.86

Dropping columns and rows

Dropping entire columns or rows based on index

```
df.drop('Label', axis = 1)
```

Axis = 1 for columns Axis = 0 for rows

Dropping entire rows (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html#pandas.DataFrame.dropna) because of missing values

```
df.dropna()
```

Dropping entire columns because of missing values

```
df.dropna(axis = 1)
```

Dropping a row based on value `df[df.column != value]`

In [71]:
```
# deleting the 'Date' column because data are now indexed by date
CO2 = CO2.drop('Date', axis = 1)

CO2.head(5)
```

Out[71]:

Date	Year	Month	CO2_avg
1958-03-01	1958	3	315.71
1958-04-01	1958	4	317.45
1958-05-01	1958	5	317.50
1958-06-01	1958	6	-99.99
1958-07-01	1958	7	315.86

In [78]:

```
# deleting null values

CO2 = CO2[CO2.CO2_avg != -99.99]
```

Filtering, Sorting, and Groupby

sort column values into ascending order

```
df.sort_values(column)
```

sort column values into descending order

```
df.sort_values(column, ascending = False)
```

filtering (only column values with year greater than a certain value)

```
df[df[column] > value]
```

groupby

```
df.groupby(column)
```

In [79]:
```
# sort CO2 concentrations into ascending order

CO2_a = CO2.sort_values('CO2_avg')

print(CO2_a)
```

```
            Year  Month  CO2_avg
Date
1958-09-01  1958      9   313.20
1959-10-01  1959     10   313.26
1958-11-01  1958     11   313.33
1960-10-01  1960     10   313.83
1959-09-01  1959      9   313.84
1960-09-01  1960      9   314.16
1958-12-01  1958     12   314.67
1959-11-01  1959     11   314.80
1961-09-01  1961      9   314.80
1959-08-01  1959      8   314.80
1958-08-01  1958      8   314.93
1960-11-01  1960     11   315.00
1961-10-01  1961     10   315.38
1962-10-01  1962     10   315.42
1959-12-01  1959     12   315.58
1959-01-01  1959      1   315.62
1958-03-01  1958      3   315.71
1958-07-01  1958      7   315.86
1960-08-01  1960      8   315.91
1963-10-01  1963     10   315.99
1961-11-01  1961     11   316.10
1960-12-01  1960     12   316.19
1963-09-01  1963      9   316.21
1962-09-01  1962      9   316.26
1959-02-01  1959      2   316.38
1960-01-01  1960      1   316.43
1959-07-01  1959      7   316.54
1962-11-01  1962     11   316.69
1964-09-01  1964      9   316.70
```

1959-03-01	1959	3	316.71
...	...	...	...
2017-01-01	2017	1	406.17
2017-02-01	2017	2	406.46
2017-12-01	2017	12	406.81
2016-06-01	2016	6	406.83
2018-08-01	2018	8	406.99
2017-07-01	2017	7	407.12
2017-03-01	2017	3	407.22
2016-04-01	2016	4	407.45
2016-05-01	2016	5	407.72
2018-01-01	2018	1	407.96
2018-11-01	2018	11	408.02
2018-02-01	2018	2	408.32
2019-09-01	2019	9	408.54
2018-07-01	2018	7	408.71
2017-06-01	2017	6	408.88
2017-04-01	2017	4	409.04
2018-12-01	2018	12	409.07
2018-03-01	2018	3	409.41
2017-05-01	2017	5	409.69
2019-08-01	2019	8	409.95
2018-04-01	2018	4	410.24
2018-06-01	2018	6	410.79
2019-01-01	2019	1	410.83
2018-05-01	2018	5	411.23
2019-02-01	2019	2	411.75
2019-07-01	2019	7	411.77
2019-03-01	2019	3	411.97
2019-04-01	2019	4	413.32
2019-06-01	2019	6	413.92
2019-05-01	2019	5	414.66

[732 rows x 3 columns]

In [82]:
```
# # sort CO2 concentrations into descending order

CO2_d = CO2.sort_values('CO2_avg', ascending = False)

print(CO2_d)
```

```
            Year  Month  CO2_avg
Date
2019-05-01  2019      5   414.66
2019-06-01  2019      6   413.92
2019-04-01  2019      4   413.32
2019-03-01  2019      3   411.97
2019-07-01  2019      7   411.77
2019-02-01  2019      2   411.75
2018-05-01  2018      5   411.23
2019-01-01  2019      1   410.83
2018-06-01  2018      6   410.79
2018-04-01  2018      4   410.24
2019-08-01  2019      8   409.95
2017-05-01  2017      5   409.69
2018-03-01  2018      3   409.41
2018-12-01  2018     12   409.07
2017-04-01  2017      4   409.04
2017-06-01  2017      6   408.88
2018-07-01  2018      7   408.71
2019-09-01  2019      9   408.54
2018-02-01  2018      2   408.32
2018-11-01  2018     11   408.02
2018-01-01  2018      1   407.96
2016-05-01  2016      5   407.72
2016-04-01  2016      4   407.45
2017-03-01  2017      3   407.22
2017-07-01  2017      7   407.12
2018-08-01  2018      8   406.99
2016-06-01  2016      6   406.83
2017-12-01  2017     12   406.81
2017-02-01  2017      2   406.46
```

2017-01-01	2017	1	406.17
...	...	...	...
1959-03-01	1959	3	316.71
1964-09-01	1964	9	316.70
1962-11-01	1962	11	316.69
1959-07-01	1959	7	316.54
1960-01-01	1960	1	316.43
1959-02-01	1959	2	316.38
1962-09-01	1962	9	316.26
1963-09-01	1963	9	316.21
1960-12-01	1960	12	316.19
1961-11-01	1961	11	316.10
1963-10-01	1963	10	315.99
1960-08-01	1960	8	315.91
1958-07-01	1958	7	315.86
1958-03-01	1958	3	315.71
1959-01-01	1959	1	315.62
1959-12-01	1959	12	315.58
1962-10-01	1962	10	315.42
1961-10-01	1961	10	315.38
1960-11-01	1960	11	315.00
1958-08-01	1958	8	314.93
1959-08-01	1959	8	314.80
1959-11-01	1959	11	314.80
1961-09-01	1961	9	314.80
1958-12-01	1958	12	314.67
1960-09-01	1960	9	314.16
1959-09-01	1959	9	313.84
1960-10-01	1960	10	313.83
1958-11-01	1958	11	313.33
1959-10-01	1959	10	313.26
1958-09-01	1958	9	313.20

[732 rows x 3 columns]

In [86]:
```
# filter CO2_avg column for concentrations greater than 400 ppm
CO2[CO2['CO2_avg'] > 400]
```

Out[86]:

Date	Year	Month	CO2_avg
2014-04-01	2014	4	401.38
2014-05-01	2014	5	401.78
2014-06-01	2014	6	401.25
2015-02-01	2015	2	400.28
2015-03-01	2015	3	401.54
2015-04-01	2015	4	403.28
2015-05-01	2015	5	403.96
2015-06-01	2015	6	402.80
2015-07-01	2015	7	401.31
2015-11-01	2015	11	400.16
2015-12-01	2015	12	401.85
2016-01-01	2016	1	402.56
2016-02-01	2016	2	404.12
2016-03-01	2016	3	404.87
2016-04-01	2016	4	407.45
2016-05-01	2016	5	407.72
2016-06-01	2016	6	406.83
2016-07-01	2016	7	404.41
2016-08-01	2016	8	402.27
2016-09-01	2016	9	401.05
2016-10-01	2016	10	401.59
2016-11-01	2016	11	403.55
2016-12-01	2016	12	404.45
2017-01-01	2017	1	406.17
2017-02-01	2017	2	406.46
2017-03-01	2017	3	407.22
2017-04-01	2017	4	409.04
2017-05-01	2017	5	409.69
2017-06-01	2017	6	408.88

Date	Year	Month	CO2_avg
2017-07-01	2017	7	407.12
2017-08-01	2017	8	405.13
2017-09-01	2017	9	403.37
2017-10-01	2017	10	403.63
2017-11-01	2017	11	405.12
2017-12-01	2017	12	406.81
2018-01-01	2018	1	407.96
2018-02-01	2018	2	408.32
2018-03-01	2018	3	409.41
2018-04-01	2018	4	410.24
2018-05-01	2018	5	411.23
2018-06-01	2018	6	410.79
2018-07-01	2018	7	408.71
2018-08-01	2018	8	406.99
2018-09-01	2018	9	405.51
2018-10-01	2018	10	406.00
2018-11-01	2018	11	408.02
2018-12-01	2018	12	409.07
2019-01-01	2019	1	410.83
2019-02-01	2019	2	411.75
2019-03-01	2019	3	411.97
2019-04-01	2019	4	413.32
2019-05-01	2019	5	414.66
2019-06-01	2019	6	413.92
2019-07-01	2019	7	411.77
2019-08-01	2019	8	409.95
2019-09-01	2019	9	408.54

Basic Summary Statistics

Summary stats

`df.describe()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe)

Correlation between columns

`df.corr()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html#pandas.DataFrame.corr)

Other useful functions (all covered under summary stats)

`df.mean()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.html#pandas.DataFrame.mean)

`df.count()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.count.html#pandas.DataFrame.count)

`df.max()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.max.html#pandas.DataFrame.max)

`df.min()` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.min.html#pandas.DataFrame.min)

In [89]:
```
# summary stats for the whole dataframe

CO2.describe()
```

Out[89]:

	Year	Month	CO2_avg
count	732.000000	732.000000	732.000000
mean	1988.666667	6.501366	354.859085
std	17.727635	3.442909	28.123411
min	1958.000000	1.000000	313.200000
25%	1973.000000	4.000000	329.112500
50%	1989.000000	7.000000	352.375000
75%	2004.000000	9.000000	377.565000
max	2019.000000	12.000000	414.660000

In [90]:
```
# summary stats for just the 'CO2_avg' column

CO2['CO2_avg'].describe()
```

Out[90]:
```
count    732.000000
mean     354.859085
std       28.123411
min      313.200000
25%      329.112500
50%      352.375000
75%      377.565000
max      414.660000
Name: CO2_avg, dtype: float64
```

In [91]:
```
# correlation between columns for the dataframe

CO2.corr()
```

Out[91]:

	Year	Month	CO2_avg
Year	1.000000	-0.018371	0.989209
Month	-0.018371	1.000000	-0.050321
CO2_avg	0.989209	-0.050321	1.000000

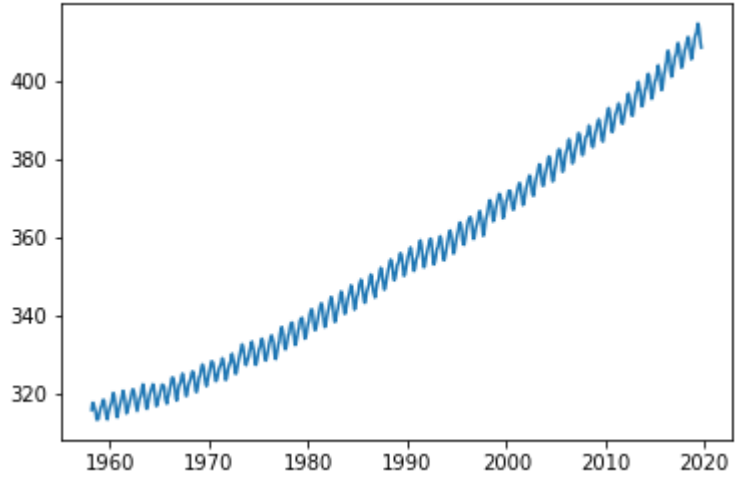
Brief Intro to Plotting in Python

Matplotlib (https://matplotlib.org/) Package = similar plotting to MATLAB

- similar functionality to MATLAB plotting, i.e. subplots, data visualization, etc.

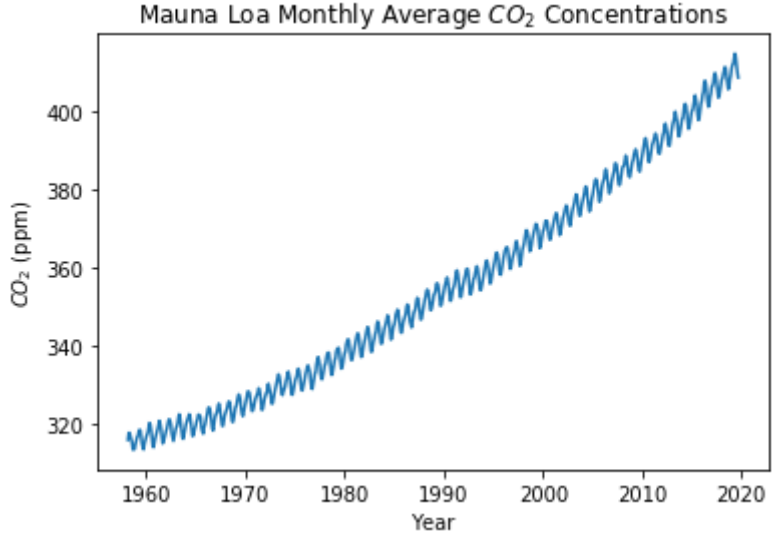
In [134]:
```
# creating a simple plot

import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(CO2['CO2_avg'])
```

Out[134]:
```
[<matplotlib.lines.Line2D at 0xb385160>]
```



In [135]:
```
plt.plot(CO2['CO2_avg'])
plt.title('Mauna Loa Monthly Average $CO_2$ Concentrations')
plt.xlabel('Year')
plt.ylabel('$CO_2$ (ppm)')
```

Out[135]:
```
Text(0, 0.5, '$CO_2$ (ppm)')
```



Pivot Tables

- allow for quick summarization of data

Creating a pivot table (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html) in Pandas

```
pd.pivot_table(data, values = '', index = '', columns = [col names])
```

Example from here (https://towardsdatascience.com/intro-to-pandas-for-excel-super-users-dac1b38f12b0) using Chicago Public Schools 2016-2017 Progress Report data

Download datasets here (https://github.com/tamck/Pandas_tutorial/blob/master/Chicago_Public_Schools_-_School_Progress_Reports_SY1516.csv) and here (https://github.com/tamck/Pandas_tutorial/blob/master/Chicago_Public_Schools_-_School_Progress_Reports_SY1617.csv)

In [118]:
```
# Load dataset
sy1617 = pd.read_csv('Chicago_Public_Schools_-_School_Progress_Reports_SY1617.csv', inde
x_col='School_ID')

sy1617.head()
```

Out[118]:

School_ID	Short_Name	Long_Name	School_Type	Primary_Category	Address	City	State	Zip	Phone	
610104	ONAHAN	William J Onahan Elementary School	Neighborhood	ES	6634 W RAVEN ST	Chicago	Illinois	60631	7735341180	7.735
610390	RICKOVER MILITARY HS	Hyman G Rickover Naval Academy High School	Military academy	HS	5900 N GLENWOOD AVE	Chicago	Illinois	60660	7735342890	7.735
610070	MCPHERSON	James B McPherson Elementary School	Neighborhood	ES	4728 N WOLCOTT AVE	Chicago	Illinois	60640	7735342625	7.735
400117	NOBLE - HANSBERRY HS	Noble - Hansberry College Prep	Charter	HS	8748 S ABERDEEN ST	Chicago	Illinois	60620	7737293400	7.733
610251	CATHER	Willa Cather Elementary School	Neighborhood	ES	2908 W WASHINGTON BLVD	Chicago	Illinois	60612	7735346780	7.735

5 rows × 165 columns

In [126]:
```
# Pivot Tables in Pandas

# create pivot table that groups school type (rows) and primary category (columns) and c
alculates average student response rate
pd.pivot_table(sy1617, values='School_Survey_Student_Response_Rate_Pct', index='School_T
ype', columns=['Primary_Category'], aggfunc=np.mean)

#CO2_pivot = pd.pivot_table(CO2, columns =  ['Month','CO2_avg'])
```

Out[126]:

Primary_Category	ES	HS	MS
School_Type			
Career academy	NaN	65.375000	NaN
Charter	66.033333	69.556061	99.400000
Citywide-Option	0.000000	65.235000	NaN
Classical	98.320000	NaN	NaN
Contract	45.700000	84.450000	NaN
Magnet	88.722222	84.342857	NaN
Military academy	NaN	87.883333	NaN
Neighborhood	84.537681	80.424444	93.814286
Regional gifted center	93.380000	NaN	NaN
Selective enrollment	NaN	82.190909	NaN
Small	87.466667	77.571429	NaN
Special Education	0.000000	58.125000	NaN

VLOOKUP

In Excel, VLOOKUP looks for a specific value in a range of cells and then returns a value that is in the same row as where the value that is being searched is located - it's very useful for merging large datasets.

In the following example, we will search by school ID and compare student attainment ratings from both the 2015-2016 and 2016-2017 school years (example from from here (https://towardsdatascience.com/intro-to-pandas-for-excel-super-users-dac1b38f12b0).

In [125]:

```
# Emulating VLOOKUP in Pandas

# read in the csv for prior year's data
sy1516 = pd.read_csv('Chicago_Public_Schools_-_School_Progress_Reports_SY1516.csv', inde
x_col='School_ID')


sy1617_short = sy1617[['Student_Attainment_Rating', 'Long_Name']]
sy1516_short = sy1516[['Student_Attainment_Rating']]

# merge function allows the two datasets to be merged by their indicies, School_ID
pd.merge(sy1516_short, sy1617_short, how='right', left_index=True, right_index=True, suf
fixes=('_1516','_1617'))
sy1617_short.join(sy1516_short, how='left', lsuffix='_1617', rsuffix='_1516')
```

Out[125]:

School_ID	Student_Attainment_Rating_1617	Long_Name	Student_Attainment_Rating_1516
610104	ABOVE AVERAGE	William J Onahan Elementary School	FAR ABOVE AVERAGE
610390	BELOW AVERAGE	Hyman G Rickover Naval Academy High School	AVERAGE
610070	AVERAGE	James B McPherson Elementary School	AVERAGE
400117	AVERAGE	Noble - Hansberry College Prep	AVERAGE
610251	AVERAGE	Willa Cather Elementary School	AVERAGE
610268	BELOW AVERAGE	Arthur R Ashe Elementary School	BELOW AVERAGE
610089	AVERAGE	John B Murphy Elementary School	ABOVE AVERAGE
610158	AVERAGE	Harriet E Sayre Elementary Language Academy	ABOVE AVERAGE
609866	FAR ABOVE AVERAGE	John C Coonley Elementary School	FAR ABOVE AVERAGE
610218	AVERAGE	Joseph Warren Elementary School	AVERAGE
610090	ABOVE AVERAGE	Phillip Murray Elementary Language Academy	ABOVE AVERAGE
609986	BELOW AVERAGE	Charles R Henderson Elementary School	FAR BELOW AVERAGE
400046	AVERAGE	L.E.A.R.N. - Romano Butler Campus	ABOVE AVERAGE
610128	AVERAGE	Marcus Moziah Garvey Elementary School	AVERAGE
610116	AVERAGE	Parkside Elementary Community Academy	BELOW AVERAGE
609971	BELOW AVERAGE	John Harvard Elementary School of Excellence	BELOW AVERAGE
610291	AVERAGE	Richard Henry Lee Elementary School	AVERAGE
609967	BELOW AVERAGE	William F Finkl Elementary School	BELOW AVERAGE

School_ID	Student_Attainment_Rating_1617	School_Name	Student_Attainment_Rating_1516
609737	AVERAGE	Friedrich W von Steuben Metropolitan Science HS	AVERAGE
610041	ABOVE AVERAGE	Ravenswood Elementary School	FAR ABOVE AVERAGE
609859	ABOVE AVERAGE	DeWitt Clinton Elementary School	FAR ABOVE AVERAGE
400175	FAR BELOW AVERAGE	Excel - Woodlawn HS	FAR BELOW AVERAGE
610221	AVERAGE	Daniel Webster Elementary School	BELOW AVERAGE
400141	BELOW AVERAGE	YCCS-Truman Middle College HS	NO DATA AVAILABLE
609870	AVERAGE	Daniel J Corkery Elementary School	AVERAGE
609732	BELOW AVERAGE	Charles P Steinmetz College Preparatory HS	BELOW AVERAGE
609798	ABOVE AVERAGE	Hiram H Belding Elementary School	ABOVE AVERAGE
610581	FAR BELOW AVERAGE	Magic Johnson- Brainerd HS	NO DATA AVAILABLE
610185	AVERAGE	Adlai E Stevenson Elementary School	ABOVE AVERAGE
609780	BELOW AVERAGE	Marine Leadership Academy at Ames	AVERAGE
...	...	...	...
400149	BELOW AVERAGE	Acero Charter Schools - Victoria Soto	NO DATA AVAILABLE

400114	ABOVE AVERAGE	Acero Charter Schools - Esmeralda Santiago	ABOVE AVERAGE
400084	AVERAGE	Acero Charter Schools - Rufino Tamayo	AVERAGE
609949	AVERAGE	William P Gray Elementary School	ABOVE AVERAGE
400085	BELOW AVERAGE	Acero Charter Schools - Major Hector P. Garcia MD	BELOW AVERAGE
400101	ABOVE AVERAGE	Acero Charter Schools - Sandra Cisneros	ABOVE AVERAGE
400121	AVERAGE	Acero Charter Schools - Sor Juana Inés de la Cruz	AVERAGE
400080	ABOVE AVERAGE	Acero Charter Schools - PFC Omar E. Torres	ABOVE AVERAGE
400089	AVERAGE	Acero Charter Schools - Officer Donald J. Marquez	ABOVE AVERAGE
610163	NO DATA AVAILABLE	Frederick Stock Elementary School	NO DATA AVAILABLE
400129	FAR BELOW AVERAGE	YCCS-Progressive Leadership Academy	NO DATA AVAILABLE
400022	AVERAGE	Chicago High School for the Arts (ChiArts)	AVERAGE
400164	NO DATA AVAILABLE	Instituto - Justice Lozano Mastery	NO DATA AVAILABLE
610565	FAR BELOW AVERAGE	Excel - Englewood HS	FAR BELOW AVERAGE
400163	AVERAGE	KIPP Chicago Charter School - KIPP Bloom	AVERAGE
400146	AVERAGE	KIPP Chicago Charter School - KIPP Create	AVERAGE
400075	AVERAGE	University of Chicago - Donoghue	ABOVE AVERAGE
610572	NO DATA AVAILABLE	Camelot Safe Elementary	NO DATA AVAILABLE
400147	NO DATA AVAILABLE	Chicago Excel Academy	FAR BELOW AVERAGE
400137	FAR BELOW AVERAGE	Little Black Pearl Art and Design Academy	BELOW AVERAGE
610573	NO DATA AVAILABLE	Camelot Safe HS	NO DATA AVAILABLE

School_ID	Student_Attainment_Rating_1617	Long_Name	Student_Attainment_Rating_1516
400180	NO DATA AVAILABLE	KIPP One Academy	NaN
400176	NO DATA AVAILABLE	Noble - The Noble Academy	NO DATA AVAILABLE
610121	AVERAGE	Washington Irving Elementary School	AVERAGE
400021	BELOW AVERAGE	Catalyst Elementary Charter School - Circle Rock	BELOW AVERAGE
400048	BELOW AVERAGE	L.E.A.R.N. - Excel Campus	BELOW AVERAGE
609821	AVERAGE	Burnham Elementary Inclusive Academy	AVERAGE
609723	BELOW AVERAGE	John Marshall Metropolitan High School	BELOW AVERAGE
610305	BELOW AVERAGE	George Leland Elementary School	FAR BELOW AVERAGE
609874	ABOVE AVERAGE	Everett McKinley Dirksen Elementary School	FAR ABOVE AVERAGE

661 rows × 3 columns

Time Stamps

A **time stamp** (a specific point in time) element in `Pandas` can be created using the following syntax:

```
pd.Timestamp()
```

The time stamp 'January 2, 2018' can be specified with `Pandas` in one of three ways:

```
1. pd.Timestamp(datetime(2018, 1, 2))

2. pd.Timestamp('2018-01-02')

3. pd.Timestamp(2018, 1, 2)
```

In [121]:

```
# time stamps in pandas
from datetime import datetime

# create a time stamp element in pandas
timestamp1 = pd.Timestamp(datetime(2018, 1, 2))
print(timestamp1)

timestamp2 = pd.Timestamp('2018-01-02')
print(timestamp2)

timestamp3 = pd.Timestamp(2018, 1, 2)
print(timestamp3)
```

```
2018-01-02 00:00:00
2018-01-02 00:00:00
2018-01-02 00:00:00
```

Can also specify time

The time stamp 'January 2, 2018 4:04:00 pm' can be specified with `Pandas` as follows:

1. `pd.Timestamp(datetime(2018, 1, 2, 16, 4, 0))`
2. `pd.Timestamp('2018-01-02 16:04:00')`
3. `pd.Timestamp(2018, 1, 2, 16, 4, 0)`

In [122]:

```
# time stamp elements in pandas (cont)
timestamp4 = pd.Timestamp(datetime(2018, 1, 2, 16, 4, 0))
print(timestamp4)

timestamp5 = pd.Timestamp('2018-01-02 16:04:00')
print(timestamp5)

timestamp6 = pd.Timestamp(2018, 1, 2, 16, 4, 0)
print(timestamp6)
```

```
2018-01-02 16:04:00
2018-01-02 16:04:00
2018-01-02 16:04:00
```

Time Intervals

A **time interval** element in `Pandas` can be created using the following syntax:

```
pd.Period()
```

The time interval 'January 2018' can be specified with pandas in one of two ways:

1. `pd.Period('2018-01')`
2. `pd.Period('2018-01', freq = 'M')```

Where 'freq' refers to the frequency of the interval.

In [128]:

```
# period elements in pandas

period1 = pd.Period('2018-01')
print(period1)

period2 = pd.Period('2018-01', freq = 'M')
print(period2)
```

```
2018-01
2018-01
```

Pandas has the following aliases for the specified periods:

Period	Frequency
hour	H
day	D
week	W
month	M
year	Y

Time Deltas

A **time delta** element can be created by finding the difference between two date, time, or `datetime` elements.

In [127]:
```
# time delta between two datetime objects using arithmetic

t1 = pd.Timestamp(datetime(2018, 1, 2, 16, 4, 0))
t2 = pd.Timestamp(datetime(2017, 2, 5, 14, 2, 5))

timedelta = t1 - t2

print(timedelta)
```

```
331 days 02:01:55
```

Other Useful Functions

Join/Combine

add the rows in df1 to the end of df2 (must have identical columns) `df1.append(df2)`

add the columns in df1 to the end of df2 (must have identical rows) `df.concat([df1, df2], axis = 1)`

Renaming columns

```
df.rename(columns = {'old_name': 'new_name'})
```

```
df.set_index('column_one')
```

arithmetic `df['total'] = df['Jan'] + df['Feb'] + df['Mar']`

Converting string data to datetime format

```
datetime_conv = pd.to_datetime(pd.Series(['Jan 01, 2018', '2018-12-31']))
```

Additional Resources

Pandas Documentation (https://pandas.pydata.org/pandas-docs/stable/index.html)

Additional DataFrame functions in Pandas (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#attributes-and-underlying-data)

Pivot Tables Tutorial (https://www.dataquest.io/blog/pandas-pivot-table/)

Intro to Pandas for Excel Super Users (https://towardsdatascience.com/intro-to-pandas-for-excel-super-users-dac1b38f12b0)

Creating slides using Jupyter Notebooks (https://medium.com/learning-machine-learning/present-your-data-science-projects-with-jupyter-slides-75f20735eb0f)