Line 2432.

Need to find the moment when the abmt\_rc\_215 df was created and what to do when actually performing regression- i.e. does it matter if there are columns in the df that are not relevant to the particular regression attempt being made? Additionally, do all the columns in a dataframe (df) need to be relevant to the regression being attempted when it may be a multivariable regression?

New Spyder version

SSL certificate verification failed while checking for Spyder updates.

Please contact your network administrator for assistance.

Make the merge, do the regression for CS2, make the map, make the 3D map, make the splits, write the report.

Basemap has imported correctly. (05.05.2024)

how can a merge such as described above be accomplished so as to only have the resulting dataframe consist of the column that represents the common key, and all the columns of the first (or left, as it would appear first in the merge expression) dataframe,

#Mandatory

IPython >=8.13.0, <9.0.0, !=8.17.1 : 8.12.3 (NOK)

Please install them to avoid this message.

Note: Spyder could work without some of these dependencies, however to have a smooth experience using Spyder, we strongly recommend you install all the listed missing dependencies.

Failing to install these dependencies might result in bugs. Please be sure that any found bugs are not the direct result of missing dependencies, prior to reporting a new issue.

If the directory containing the Spyder executable is not already included in your system's PATH environment variable, you may need to provide the full path to the Spyder executable. For example: C:\ProgramData\Anaconda3\Scripts\spyder.exe # On Windows

mklink C:\ProgramData\anaconda3\envs\bridgeEnv\Scripts\spyder.exe C:\ProgramData\anaconda3\Scripts\spyder.exe

mklink C:\ProgramData\anaconda3\envs\bridgeEnv\Scripts\spyder C:\ProgramData\anaconda3\Scripts\spyder

(bridgeEnv) C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData>mklink C:\ProgramData\anaconda3\envs\bridgeEnv\Scripts\spyder C:\ProgramData\anaconda3\Scripts\spyder

symbolic link created for C:\ProgramData\anaconda3\envs\bridgeEnv\Scripts\spyder <<===>> C:\ProgramData\anaconda3\Scripts\spyder

(bridgeEnv) C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData>

C:\ProgramData\anaconda3\envs\bridgeEnv

if given the following formatted list of dependencies:

- scipy.stats

- io

- sklearn.pipeline

- glob

- statsmodels.graphics.tsaplots

- matplotlib.dates

- time

- re

- statsmodels.tsa.arima.model

- matplotlib.ticker

- array

- os

- xml.etree.ElementTree

- numpy.ma

- warnings

- base64

- json

- numpy

- webbrowser

- sklearn.preprocessing

- plotly.graph\_objects

- math

- collections

- sklearn.metrics

- statsmodels.tsa.stattools

- functools

- pandas.plotting

- sklearn.linear\_model

- copy

- statsmodels.api

- matplotlib.pyplot

- statsmodels.tsa.seasonal

- datetime

- seaborn

- statsmodels.tsa.arima\_model

- geopandas

- pmdarima

- pandas

- sklearn.model\_selection

Does the user need to list all the different sklearn “submodules” in the environment.yml file?

A: No. Just including sklearn is sufficient, it will handle the installation of all necessary submodules.

The environment.yml file is used to create the virtual environment directly, even uses the beginning of the file to create the name of the environment. Following a correct creation of the environment.yml file the following should be run from the Anaconda Prompt:

conda env create -f environment.yaml

Let’s re-affirm the reasons we’re in this rabbit hole:

1. Trying to get either (or one of many) Cartopy, Basemap or some other dependency working in order to make a good plot of the map of the state appear and make the coordinates appear as they should without any ‘grid’ space between the clusters of coordinates and without plotting over top of the coordinates either (i.e. obscuring them).
   1. Make the plot of the coordinates change color by percentage of condition state of the element at the location
   2. Figure out why the other coordinates outside the state are appearing, looks like they’re all the way to the other side of the country, could it perhaps be an ‘origin’ coordinate? If so, where is it located and how can it be used (provided that the origin coordinate is necessary) without it making the plot unfocused on the boundaries of the state?
2. Need to make a try at the effort to separate out the different parts of the program into other parts of the ‘template’ like is so often suggested.
3. Need to make a go at multi-variate regression, i.e. using the CS2 data from all of the bridges that have CS1 data that I’ve been using as a first attempt and base the execution of the regression on the hypothesis that I expect CS1 to drop due to the only possible factors that could increase the values of CS1 and cause an upward trend being that i.) bridge elements are added to the existing bridges by way of expansion of the bridge (for instance a widening of the bridge, adding lanes to it), or replacement of elements that have been worn with new ones that would cause the percentage of CS1 per bridge of an element to rise from a prior value

In main.py (or or CAL\_BridgeData.py in our case) does

if \_\_name\_\_ == ”\_\_main\_\_”:

…..

Need to be placed at the end of the file?

Pre opening of any files or IDEs, I think this should be the plan:

Revision: I installed Anaconda again but with Admin privileges instead of ‘just me’ as in previous installments.

04.17.2024 Anaconda reinstall: 1. Open Anaconda prompt with Admin privileges

2. Check dependencies: make sure all required dependencies are installed prior to running any files-

04.14.2024: Remember- the “Install IPython using conda” is the thread being used, and that the ‘conda-meta’ directory is not inside the ‘pkgs’ directory and this was a main clue to the problem, and that ‘conda-meta’ is located inside the ‘anaconda3’ directory. Basically, anaconda is not installed in the right places on the computer- or possibly at all.

conda install -c conda-forge basemap

conda install -c anaconda basemap

#Mandatory

IPython >=8.13.0, <9.0.0, !=8.17.1 : 8.12.3 (NOK)

##### (fix) [**main**](https://anaconda.org/main)**/**[**ipython 8.20.0**](https://anaconda.org/main/ipython)

try first conda install quandl then use conda install basemap

pandas numpy matplotlib.pyplot os matplotlib.dates seaborn time matplotlib.ticker statsmodels.api adfuller seasonal\_decompose ARIMA register\_matplotlib\_converters train\_test\_split LinearRegression make\_pipeline PolynomialFeatures mean\_squared\_error mean\_absolute\_error geopandas plotly.graph\_objects

pandas

numpy

matplotlib.pyplot

os

matplotlib.dates

seaborn

time

matplotlib.ticker

statsmodels.api

adfuller

seasonal\_decompose

ARIMA

register\_matplotlib\_converters

train\_test\_split

LinearRegression

make\_pipeline

PolynomialFeatures

mean\_squared\_error

mean\_absolute\_error

geopandas

plotly.graph\_objects

|  |  |
| --- | --- |
| pandas |  |
| numpy |  |
| matplotlib.pyplot |  |
| os |  |
| matplotlib.dates |  |
| seaborn |  |
| time |  |
| matplotlib.ticker |  |
| statsmodels.api |  |
| adfuller |  |
| seasonal\_decompose |  |
| ARIMA |  |
| register\_matplotlib\_converters |  |
| train\_test\_split |  |
| LinearRegression |  |
| make\_pipeline |  |
| PolynomialFeatures |  |
| mean\_squared\_error |  |
| mean\_absolute\_error |  |
| Geopandas |  |
| plotly.graph\_objects |  |

How should the order of operations go when creating and installing packages in an environment? (Ok, so to answer my question, yes, the environment needs to be activated before calling conda install.

1. conda create –-name myenv
2. conda activate myenv
3. conda install numpy, pandas, matplotlib, etcetera.

Part of Python’s standard library, not necessary to be installed separately using conda: xml.etree.ElementTree collections datetime copy io functools math warnings re json

<https://www.youtube.com/watch?v=MF7asKblfm8>

https://www.youtube.com/watch?v=CqVGbfPVBk4

C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\mingw-w64\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\usr\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Scripts;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\bin;C:\ProgramData\anaconda3\condabin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\WINDOWS\System32\OpenSSH;C:\Program Files\Git\cmd;C:\Program Files\nodejs;C:\Program Files\PuTTY;C:\ProgramData\anaconda3\Scripts;C:\Users\Chris\AppData\Local\Programs\Microsoft VS Code\bin;C:\Program Files\JetBrains\PyCharm Community Edition 2023.3.5\bin

C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\mingw-w64\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\usr\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Library\bin;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\Scripts;C:\Users\Chris\CodingBootcamp\Homework\CAL\_BridgeData\envs\bridgeEnv\bin;C:\ProgramData\anaconda3\condabin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\WINDOWS\System32\OpenSSH;C:\Program Files\Git\cmd;C:\Program Files\nodejs;C:\Program Files\PuTTY;C:\ProgramData\anaconda3\Scripts;C:\Users\Chris\AppData\Local\Programs\Microsoft VS Code\bin;C:\Program Files\JetBrains\PyCharm Community Edition 2023.3.5\bin

conda env list

Left join means all rows from the left dataframe are preserved. And the “left” dataframe is the variable corresponding to the dataframe that comes first in the expression performing the merge.

https://www.youtube.com/watch?v=pZrX0iJCh5s

03-31-2024: line 1702 10:42 am

commit 8c9b8a9b04bd71a9fd923075cdc76948f1bd412b (HEAD -> main, origin/main)

Author: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Date: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Frequent commit to maintain working copy

diff --git a/CAL\_BridgeData.py b/CAL\_BridgeData.py

index cef619b..f9b0c85 100644

--- a/CAL\_BridgeData.py

+++ b/CAL\_BridgeData.py

@@ -4,61 +4,49 @@ Created on Thu Oct 21 16:29:08 2021

@author: Chris

"""

-import hashlib

-

-def calculate\_hash(file\_path):

- with open(file\_path, 'rb') as f:

- return hashlib.sha256(f.read()).hexdigest()

-

-# Example usage:

-hash\_value = calculate\_hash(r'bridgeEnv\Lib\site-packages\numpy-1.26.4-cp39-cp39-win\_amd64.whl')

-

-print(hash\_value)

- # e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

-

-

+

import pandas as pd

# pip install --upgrade pandas

- # pytz, tzdata, six, numpy, python-dateutil, pandas

:

project\_root/

│

├── main.py

├── data\_processing.py

├── linear\_regression.py

├── input\_output.py

├── config.py

├── data/ (folder containing your data files)

│ ├── data\_file1.csv

│ ├── data\_file2.csv

│ └── ...

└── ...

03-30-2024: Goal: to get the multivariate linear regression to work with the abmt\_rc\_215\_no\_outls DataFrame or to get some multivariate regression involving one of the abmt\_215\_... DataFrames to create a sensible regression analysis using the CS1 and CS2 data.

How: go back to the area of the code that creates the dataframes with just the CS1 column and attempts to analyze just the single data column as it relates to the datetime index already successfully created, and then rather than eliminating the other CS2-CS4 columns eliminate only the CS3 and CS4- then use the two columns to look for anything that could show a trend in the data. The reason to attempt this is that I believe that the data is more likely to trend upward when looking at the CS2 column because generally, CS1 is ’pristine’ or nearly new material as it relates to bridge elements and it seems unlikely that material in said state would increase in any way.

Remember, the point of the troubleshooting with the pipreqs and requirements.txt is to get the map plotting function to work. This is the goal, just to plot a map.

The steps I’ve gotten so far are that the virtual environment should be activated when you plan to do any work to the dependencies, and to create a requirements.txt file should be in the root directory of the project. Probably need to look back at notes on the internet about how to create the requirements.txt file. It might be fixed easily by using pipreqs and having the .py file scanned for the dependencies again and just making sure their installed. The current requirements.txt file is shorter than the one from earlier.

also in regard to the very first solution of this chat, if the dataframe being used to create the map has the format of loc\_data = {'index': [0, 1, 2, 3, etc.], 'STRUCNUM': [06 0021, 1CA0070, 1CA0095, 1CA0141, etc.], 'LAT': [40.4524, 34.0608, 32.4336, 34.0659, etc.], 'LONG': [-122.191, -119.061, -117.085, -119.055, etc.]} just as an example, does the for statement using such data (from \*Create California Bridge Map+excel parse)

03.09.2024: Need to get the “modules” to work, i.e. make the different model files to work with a main.py file that may need to not resemble the typical main.py files that usually accompany a data science template, i.e. just make the model.py, model1.py, model2.py file connect and work with the current CAL\_BridgeData.py file. Then do the Test and Train regression analysis on the abmt\_rc\_215

Next, make the “map” of the bridges that are to be in the final data set i.e. all the bridge\_counts\_un (variable, means bridge counts unique) bridges- and use the Folium or Plotly modules to make the map. The coordinates (lat and long) of the bridges are in an Excel file that will have some particulars as far as reading the data into the program. The data from the excel file may not have to be read in in its entirety, there may be a means to read only the necessary columns that hold the STRUCNUM, and the lat and long of the associated STRUCNUM into the program and not require the rest of the data (of which there is much) that may be superfluous. Once the STUCNUM, lat and longs have been read in, join the lat and long columns to the overall dataframe, in all likelihood join or merge the two additional columns on the STRUCNUM column and to the dataframe called df\_all\_yrs\_merged

03.07.2024:

What is a global namespace dictionary? How does it relate from module to module?

03.06.2024, cont’d. The initial function to read the xml files was not reading anything into the dataframes. A re-copying of the function from github fixed the program back to how it had operated before.

Now: need to make the different models possible for the programs originate/and or store & save in different files a directory forward from the source code.

03.06.2024:

KeyError: ‘abmt\_rc\_215’

At present, all 4 dicts attempted to be created using the find\_three\_lrgst\_dfs\_in\_ea\_dict function to create the top3\_lrgst\_dfs\_in\_dict\_dicts dictionary of dictionaries are all just single dataframes that are not in the correct form as they had been before, thusly the function to get the dataframes needed as they had been corralled before is not working correctly. So we’ll start by looking at the find\_three\_... function to begin with.

The variable as created in the program is called new\_dict\_dicts\_CS1 meaning new dictionary of dictionaries (the inner dictionaries being dictionaries of dataframes) because there is a previous dict of dicts that was used originally in the function.

02.29.2024

Remaining:

1. Test and train approach with the regression analysis
   1. Can test and train be used with other types of regression (i.e. multi, polynomial, logarithm of the dependent variable,

project\_root/

│

├── main.py

├── data\_processing.py

├── linear\_regression.py

├── input\_output.py

├── config.py

├── data/

│ ├── raw\_data/

│ │ ├── file1.csv

│ │ ├── file2.csv

│ │ └── ...

│ └── processed\_data/

│ ├── cleaned\_file1.csv

│ ├── cleaned\_file2.csv

│ └── ...

└── ...

project\_root/

│

├── main.py

├── data\_processing.py

├── linear\_regression.py

├── input\_output.py

├── config.py

├── data/ (folder containing your data files)

│ ├── data\_file1.csv

│ ├── data\_file2.csv

│ └── ...

└── ...

Probably don’t need to trick the program

Do I need to “trick” the program by making all the frequencies between every observation the same all along the entirety of the data? i.e. make the frequency something manageable like an hour between each observation if that would be “close” to a frequency the program could understand? i.e. the number of days between January 1st 2016 and December 31st 2023 is 2922. The number of observations in the time being considered is 49534. This spans a time of 70128 hours between those dates (and when the command already part of the numpy package is used, ie. resample()the code automatically makes the number of rows in the df to 70128 on its own- there are not that many actual observations so the .

If there is data that occurs at a frequency that does not easily fit an…

Given a merged dataframe (created from merging other dataframes that would have resulted in a longer overall merged dataframe if merged on one column) merged earlier in the program on two columns-

Ok, Given a merged dataframe resulting from multiple dataframes being merged on two columns and another resultant dataframe created using the same data but merged on only a single column,

Ok ok given two dataframes created from merging the same data, one from dataframes merged on 2 columns and the other from merging on one column- in this case making the resultant merged on one column longer in number of rows, i.e. more observations-

Abutment

An important point about where I am now and how I’m most interested in proceeding is that I’m willing and expecting to look at what I’ve done most recently producing my project and to admit to myself that there are better ways of accomplishing the results of this analysis- there are more advanced tools and ways to program a computer to produce an analysis, more efficient data structures, more correct tools to manipulate them, better, quicker ways of cleaning data to produce succinct results. Regardless of what I’m working on I proceed striving to be productive, efficient and produce a superior finished product. I expect to look at my CAL\_BridgeData repository and see many ways to update it and make it leaner and more efficient, more advanced and just a better product overall. I want to do this whenever I’m working on anything in tech/data science, etc.

Make certain to include in your application: Once you’ve gone thru the “I prefer challenging jobs, I want to learn more about new things, that keeps me interested blah blah. Then point out how you want to embark on a journey that will take you new places perpetually- how the idea of learning new ways to handle challenging tasks is your lane in your career already- point out how important it is to you to stop solving the same problems- or at least to attempt to solve problems that cross your desk once and for all- or at least be afforded the opportunity to solve problems more efficiently- because you’ve seen the same challenges appear seasonally- in your current position causes a near seasonal depression when the same tasks reprise themselves year after year as winter turns to spring, to summer, to fall, etc.

def mk\_time\_col\_df(df, year\_column='year'):

# Initialize variables for the previous year and a list to store DataFrames

prev\_year = None

new\_dfs = [] # create list of new dfs to be the copies of the

for index, row in df.iterrows():

current\_year = row[year\_column]

# Check if the 'year' column value has changed

if current\_year != prev\_year:

# Create a new DataFrame for the current year

year\_df = df[df[year\_column] == current\_year].copy()

# Calculate the number of rows in the current year

num\_rows\_in\_year = len(year\_df)

# Create and populate the 'date\_time' column using Timedelta

start\_time = pd.to\_datetime(f'{current\_year}-01-01 00:00:00')

end\_time = pd.to\_datetime(f'{current\_year+1}-01-01 00:00:00')

# Spread time evenly within the year

year\_df['date\_time'] = pd.date\_range(start=start\_time, end=end\_time, periods=num\_rows\_in\_year)

new\_dfs.append(year\_df)

prev\_year = current\_year

# Concatenate the dataframes in the list

new\_df = pd.concat(new\_dfs, ignore\_index=True)

return new\_df

# Function to apply the equal elapse of time between each row in each DataFrame in a dictionary of DataFrames.

def spread\_time\_in\_dict(dataframes, year\_column='year'):

result = {}

for key, df in dataframes.items():

new\_df = mk\_time\_col\_df(df, year\_column)

result[key] = new\_df

return result

# function call

dict\_to\_shallowcopy = spread\_time\_in\_dict(dict\_to\_shallowcopy, year\_column='year')

#

when performing linear regression in Spyder using Python and Pandas should the x-axis data always be in an array consisting of a single column, meaning can the x-axis data be part of a larger dataframe? Answer: no, the x-axis data can be part of a larger dataframe.

model = LinearRegression()

# The above is considered an instance of a class. This statement creates the [variable](https://realpython.com/python-variables/) model as an instance of LinearRegression

Make the point in your application that you are always looking for ways to avoid the nagging aspects of the typing and coding process in order to get to the more dynamic aspects of the endeavour- I look for ways to avoid minutiae and find the advanced means of accomplishing things, the way I can make my coding more advanced. A good way to help do that is to keep a well maintained set of modular functions that you can come back to somewhat repeatedly and have ready to use when you need them- Boilerplate basically.

>>> x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

>>> y = np.array([5, 20, 14, 32, 22, 38])

>>> x

array([[ 5],

[15],

[25],

[35],

[45],

[55]])

>>> y

array([ 5, 20, 14, 32, 22, 38])

As you can see, x has two dimensions, and x.shape is (6, 1), while y has a single dimension, and y.shape is (6,).

A discrete variable is one whose value is Obtained by counting. Another definition is that it is a variable that has only a finite number of real values (e.g. 1, 3, 5 and 1000).

* **Continuous**: it can have an infinite number of possible values within a selected range. ‘float’ is usually used for continuous data in python. e.g. the footage of the living area
* **Nominal:**nominal variables use a numerical representation to interpret types or attributes of objects. They are categorical values with 2 or more possible values and have no inherent order or ranking sequence. For example, origin country of cars, US can be 1, Japan 2, Germany 3
* **Ordinal:**ordinal variables are actually presenting numerical values. They usually have 2 or more possible values in a limited range and these values have ordered categories. e.g. grade of house. When you plot ordinal values with target, you will often see clear vertical line. Ordinal data might be read as integer or float in your dataset so data visualization is always helpful to detect them
* **Binary:**only have 2 possible values usually 0 and 1. Dummy variables are binary

|\|

State that you’re interested in the apprenticeship for its many facets- that the prospect of cyber security, development and data science training is more than agreeable to you from your past experience where time as an engineer could’ve been devoted to learning other “in house” tasks that would have made your career more sustainable. (i.e. at Thermcraft when furnace work dried up and diffuser orders were high and if you’d been trained you could have stayed on)

State that your viewpoint is in no way narrow.

Point out in your application that you want to stress that you went out of your way to produce a finished product that is as polished as possible and that doing so certainly delayed the date of your application considerably!

http://www.google.com/ig?brand=TSNA&bmod=TSNA

Jay: 919 609 2421

U\/\//-\nthe1p

1. STATE\_CODE\_001,
2. STRUCTURE\_NUMBER\_008,
3. RECORD\_TYPE\_005A,
4. ROUTE\_PREFIX\_005B,
5. SERVICE\_LEVEL\_005C,
6. ROUTE\_NUMBER\_005D,
7. DIRECTION\_005E,
8. HIGHWAY\_DISTRICT\_002,
9. COUNTY\_CODE\_003,
10. PLACE\_CODE\_004,
11. FEATURES\_DESC\_006A,
12. CRITICAL\_FACILITY\_006B,
13. FACILITY\_CARRIED\_007,
14. LOCATION\_009,
15. MIN\_VERT\_CLR\_010,
16. KILOPOINT\_011,
17. BASE\_HWY\_NETWORK\_012,
18. LRS\_INV\_ROUTE\_013A,
19. SUBROUTE\_NO\_013B,
20. LAT\_016,
21. LONG\_017,
22. DETOUR\_KILOS\_019,
23. TOLL\_020,
24. MAINTENANCE\_021,
25. OWNER\_022,
26. FUNCTIONAL\_CLASS\_026,
27. YEAR\_BUILT\_027,
28. TRAFFIC\_LANES\_ON\_028A,
29. TRAFFIC\_LANES\_UND\_028B,
30. ADT\_029,
31. YEAR\_ADT\_030,
32. DESIGN\_LOAD\_031,
33. APPR\_WIDTH\_MT\_032,
34. MEDIAN\_CODE\_033,
35. DEGREES\_SKEW\_034,
36. STRUCTURE\_FLARED\_035,
37. RAILINGS\_036A,
38. TRANSITIONS\_036B,
39. APPR\_RAIL\_036C,
40. APPR\_RAIL\_END\_036D,
41. HISTORY\_037,
42. NAVIGATION\_038,
43. NAV\_VERT\_CLR\_MT\_039,
44. NAV\_HORR\_CLR\_MT\_040,
45. OPEN\_CLOSED\_POSTED\_041,
46. SERVICE\_ON\_042A,
47. SERVICE\_UND\_042B,
48. STRUCTURE\_KIND\_043A,
49. STRUCTURE\_TYPE\_043B,
50. APPR\_KIND\_044A,
51. APPR\_TYPE\_044B,
52. MAIN\_UNIT\_SPANS\_045,
53. APPR\_SPANS\_046,
54. HORR\_CLR\_MT\_047,
55. MAX\_SPAN\_LEN\_MT\_048,
56. STRUCTURE\_LEN\_MT\_049,
57. LEFT\_CURB\_MT\_050A,
58. RIGHT\_CURB\_MT\_050B,
59. ROADWAY\_WIDTH\_MT\_051,
60. DECK\_WIDTH\_MT\_052,
61. VERT\_CLR\_OVER\_MT\_053,
62. VERT\_CLR\_UND\_REF\_054A,
63. VERT\_CLR\_UND\_054B,
64. LAT\_UND\_REF\_055A,
65. LAT\_UND\_MT\_055B,
66. LEFT\_LAT\_UND\_MT\_056,
67. DECK\_COND\_058,
68. SUPERSTRUCTURE\_COND\_059,
69. SUBSTRUCTURE\_COND\_060,
70. CHANNEL\_COND\_061,
71. CULVERT\_COND\_062,
72. OPR\_RATING\_METH\_063,
73. OPERATING\_RATING\_064,
74. INV\_RATING\_METH\_065,
75. INVENTORY\_RATING\_066,
76. STRUCTURAL\_EVAL\_067,
77. DECK\_GEOMETRY\_EVAL\_068,
78. UNDCLRENCE\_EVAL\_069,
79. POSTING\_EVAL\_070,
80. WATERWAY\_EVAL\_071,
81. APPR\_ROAD\_EVAL\_072,
82. WORK\_PROPOSED\_075A,
83. WORK\_DONE\_BY\_075B,
84. IMP\_LEN\_MT\_076,
85. DATE\_OF\_INSPECT\_090,
86. INSPECT\_FREQ\_MONTHS\_091,
87. FRACTURE\_092A,
88. UNDWATER\_LOOK\_SEE\_092B,
89. SPEC\_INSPECT\_092C,
90. FRACTURE\_LAST\_DATE\_093A,
91. UNDWATER\_LAST\_DATE\_093B,
92. SPEC\_LAST\_DATE\_093C,
93. BRIDGE\_IMP\_COST\_094,
94. ROADWAY\_IMP\_COST\_095,
95. TOTAL\_IMP\_COST\_096,
96. YEAR\_OF\_IMP\_097,
97. OTHER\_STATE\_CODE\_098A,
98. OTHER\_STATE\_PCNT\_098B,
99. OTHR\_STATE\_STRUC\_NO\_099,
100. STRAHNET\_HIGHWAY\_100,
101. PARALLEL\_STRUCTURE\_101,
102. TRAFFIC\_DIRECTION\_102,
103. TEMP\_STRUCTURE\_103,
104. HIGHWAY\_SYSTEM\_104,
105. FEDERAL\_LANDS\_105,
106. YEAR\_RECONSTRUCTED\_106,
107. DECK\_STRUCTURE\_TYPE\_107,
108. SURFACE\_TYPE\_108A,
109. MEMBRANE\_TYPE\_108B,
110. DECK\_PROTECTION\_108C,
111. PERCENT\_ADT\_TRUCK\_109,
112. NATIONAL\_NETWORK\_110,
113. PIER\_PROTECTION\_111,
114. BRIDGE\_LEN\_IND\_112,
115. SCOUR\_CRITICAL\_113,
116. FUTURE\_ADT\_114,
117. YEAR\_OF\_FUTURE\_ADT\_115,
118. MIN\_NAV\_CLR\_MT\_116,
119. FED\_AGENCY,
120. SUBMITTED\_BY,
121. BRIDGE\_CONDITION,
122. LOWEST\_RATING,
123. DECK\_AREA

1. 45,
2. 2SC0300,
3. 1,
4. 8,
5. 0,
6. 00000,
7. 0,
8. 00,
9. 013,
10. 04690,
11. 'BALLAST CREEK',
12. ,
13. 'CUBA STREET',
14. '1.1 KM S. OF MALECON DR.',
15. 99.99,
16. 0,
17. 0,
18. ,
19. ,
20. 32200440,
21. 080401910,
22. 7,
23. 3,
24. 73,
25. 73,
26. 09,
27. 1941,
28. 2,
29. 0,
30. 50,
31. 2013,
32. 3,
33. 6.7,
34. 0,
35. 0,
36. 0,
37. 1,
38. 0,
39. 1,
40. 1,
41. 4,
42. 0,
43. 0,
44. 0,
45. A,
46. 1,
47. 5,
48. 1,
49. 04,
50. 0,
51. 00,
52. 12,
53. 0,
54. 67,
55. 9.1,
56. 109.7,
57. 0.2,
58. 0.2,
59. 7.3,
60. 7.8,
61. 99.99,
62. N,
63. 0,
64. N,
65. 0,
66. 0,
67. 6,
68. 7,
69. 7,
70. 6,
71. N,
72. 1,
73. 45.8,
74. 1,
75. 27.1,
76. 6,
77. 6,
78. N,
79. 5,
80. 9,
81. 8,
82. 38,
83. 1,
84. 109.7,
85. 421,
86. 24,
87. N ,
88. Y60,
89. N ,
90. ,
91. 0519,
92. ,
93. 6,
94. 1,
95. 7,
96. 2021,
97. ,
98. ,
99. ,
100. 0,
101. N,
102. 2,
103. ,
104. 0,
105. 0,
106. ,
107. 1,
108. 0,
109. 0,
110. 0,
111. 5,
112. 0,
113. ,
114. Y,
115. 5,
116. 55,
117. 2032,
118. ,
119. Y,
120. 73,
121. F,
122. 6,
123. 855.66

08/23/22: Gameplan

1. Get the individual years to plot their data in scatter DONE
2. Make the line of best fit from “scratch” (i.e. not using KJee)
3. Think on the issue you want to discuss w/ JRB (i.e. the “does it make sense to find a trend based on what all of a specific EN for all bridges in the state do in one year?“)
4. ~~np.linspace(start, stop, num)~~
   1. ~~start(obvious)~~
   2. ~~stop(obvious)~~
   3. ~~num is an integer that determines how many elements the output array will have. The default num is 50.~~
   4. (np.linspace will not work for datetime)

08/22/22:

1. The type of any object or array or list to be used for any time data will be critical.

08/21/2022:

1. Still not getting dates to plot correctly on x-axis
   1. Is it a formatting issue? i.e. does the formatting need to be part of the line of code that makes the scatter plot?
2. Need to get better data, i.e. CAL.
   1. See what that looks like, can it be a better means of performing a regression?
   2. Can Kaplan-Meier be used to estimate the lifespan of bridge elements?
3. Need to make the replacement of missing data a priority- get the bridges that appear in all years but have only a single or less than total number of years considered number of observations for a particular EN produce data for that EN for all years considered. Use the .replace() method to achieve this if possible.

What do I need: I need plots of best fit line graphs to perform linear regression and be able to figure out where the numbers in the CS1-CS4 ratings are going. And I need to have the dates associated with those observations showing on the x-axis when those plots are made. Basically I need to familiarize myself with linear regression and make the exercise I perform in linear regression a proper analysis- i.e. not something that a seasoned analyst would automatically poke holes into because the approach or methods were just misguided and/or amateurish. In other words, make sure the regression type (linear, nonlinear, polynomial, etc.) is correct.

I need to make the data subsets for each EN larger by replacing data using the methods available in python/pandas/spyder. Basically, if a bridge has observations in all 5 (or however many there are) years then observations for all EN observed across all 5 of those years shall be present in all years. The set or subset of EN for any bridge consists of the EN observed across all years of data possessed by the analyst, and all EN shall have entries in all years either by observation or by replacing data using python/pandas/spyder methods.

I need to fix the invalid frequency error! (I think I fixed this.)

Re-set of priorities for 08/14/2022:

1. Get the dates plotting on the graphs- see [https://www.tutorialspoint.com/plotting-dates-on-the-x-axis-with-python-s-matplotlib#](https://www.tutorialspoint.com/plotting-dates-on-the-x-axis-with-python-s-matplotlib)
2. Basically, if a bridge is observed in each and possesses any EN observations in each year, all EN observations will be will be replaced as missing data in every year using Python/Pandas/spyder methods to replace said data. In other words (See a. below)
   1. Go back to the way we were selecting in the beginning, i.e. AND use all the bridges that have any observations in all of the years that are used to have data for each year for any EN that appears in at least one observed year- i.e. if the bridge does not have data for all years for at least one of its elements (or ENs) do not eliminate the bridge entirely from the data analysis but instead replace the missing data possibly using the interpolate() method to come up with those numbers. (# Rationale for the replacement of data for deck\_rc is that the subset of data will consist of all bridges that have observations in all years AND at least one EN observation in one year- thus replacing the the EN observations for years where no data is present but at least one observation is present in at least one year for a bridge- maintaining the number of observations per year as the same across all years and adding to the data to get a more representative sample.) Go back to the subset that used a merge on STRUCNUM and get the entire subset of bridges that have observations in all years. Next, use the methods mentioned earlier to replace missing data.

Does the problem need to be revised to “When can I expect the bridges at a CS1 of 0 (zero) to be at a “normalized” score of 1? Also, do any bridges already at a 0 (zero) of CS1 but with scores registering in the upper categories (Cs2-Cs4) need to be omitted somehow? Is it reasonable to think any of that element will “revert” to the CS1 category unless acted upon from the outside some way? Does the deck\_rc need to be re-run with some elimination of the bridges with CS1=0 and CS2=1 (or any other value?

(Below) I don’t believe that was the problem…

~~The problem may be related to the 3h19.544419135min term in the freq of the deck\_rc\_dates\_20XX = pd.date\_range(… statement meaning the freq may need to be equal to something in minutes and seconds and the decimal cannot show up until the fractional part of the formatting!~~

Bandwidth/Republic

PowerSecure

1. Cancel tablet
2. Cancel T-Mobile
3. Reservations at Ruff Housing
4. Pay phone bill
5. There was one more but I can’t recall….! 😐
6. And mow the silly lawn
7. Updating
   1. conda update anaconda
   2. conda install spyder=5.2.2

Reset of priorities 08/05/2022: https://stackoverflow.com/questions/23294197/plotting-chart-with-epoch-time-x-axis-using-matplotlib

1. ~~Change the freq to something generated by the program not entered by the user (avoid the TypeError: DType <class 'numpy.dtype[datetime64]'> could not be promoted by <class 'numpy.dtype[float64]'>.)~~
2. Convert datetime to numeric for purposes of the regression analysis
3. Convert numeric back to datetime or do something to make the numeric units meaningful when plotted?
   1. Follow up with making the plots appear to have the numeric representation correspond to datetime in some way OR 336 765 8132
   2. Adjust the labels of the x-axis to represent dates, datetime, etc. something…

Convert the datetime to epoch

Make the plots using epoch

Change the labels after plotting?

Make all the deck\_rc\_dates\_20XX for all 5 years

R2 = [Var(mean) – Var(fit)] / Var(mean)

https://www.youtube.com/watch?v=d1ylCNJsU-M

07/29/2022 reset of priorities: so I’m trying to make the dates appear as dates on the plots (<https://stackoverflow.com/questions/22048792/how-do-i-display-dates-when-plotting-in-matplotlib-pyplot>),

AND use all the bridges that have any observations in all of the years that are used to have data for each year for any EN that appears in at least one observed year- i.e. if the bridge does not have data for all years for at least one of its elements (or ENs) do not eliminate the bridge entirely from the data analysis but instead replace the missing data possibly using the interpolate() method to come up with those numbers. (# Rationale for the replacement of data for deck\_rc is that the subset of data will consist of all bridges that have observations in all years AND at least one EN observation in one year- thus replacing the the EN observations for years where no data is present but at least one observation is present in at least one year for a bridge- maintaining the number of observations per year as the same across all years and adding to the data to get a more representative sample.) Go back to the subset that used a merge on STRUCNUM and get the entire subset of bridges that have observations in all years. Next, use the methods mentioned earlier to replace missing data.

1. Make the dates “plottable” on the x-axis (i.e. not the just plain numbers that come from the conversion that I used earlier)
2. Fix the missing data for the deck\_rc using fillna() or replace() and possibly interpolate()-

07/28/2022 reset of priorities: 2017 1st bridge: 2305

2018 1st bridge: 7540

07/27/2022 reset of priorities:

1. Did I mean to make the 5 individual years ascend in terms of CS1 for each year or just ascend for the first year (2017) and have the order of that STRUCNUM list be the same for the following years also? It looks like presently I am making each year ascend individually which would mean the 1 year between observations wouldn’t be true.

07/03/2022 reset of priorities:

1. Should I use a method to “make” missing data for bridges that don’t have entries for the years where a bridge is observed but no observation is made for the EN that was observed in other years? (i.e. “missing data”?)

<https://stackoverflow.com/questions/70949098/how-to-work-around-the-date-range-limit-in-pandas-for-plotting>

My hypothesis with regard to this data is that the condition states will persist for roughly 20 years before approaching unity and therefore moving from one condition state to the next.

The independent variables and the dependent variables do not have to be normally distributed in and of themselves, but you’d like the residuals of the outcomes to be normally distributed- the residual is just like an error- i.e. the distance from the line is the residual- if the residuals are not normally distributed that means that our line is only representative of a portion of our data rather than the whole data set.

We do not want our independent variables to be too highly correlated with each other- make a correlation plot- we want all our independent variables to be highly correlated with CS but fairly lowly correlated with themselves.

3.325740318906606 hrs per period- how many in one year? (2634 I believe…) Then 3.325740318906606 \* 13170 = 43800 hours which is the number of hours in 5 years ignoring leap years.

07/05/2022: figure out if the alias and the offset instance being fungible will make the year end accurately <https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases>

You can combine together day and intraday offsets:

**In [240]:** pd.date\_range(start, periods=10, freq="2h20min")

**Out[240]:**

DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 02:20:00',

'2011-01-01 04:40:00', '2011-01-01 07:00:00',

'2011-01-01 09:20:00', '2011-01-01 11:40:00',

'2011-01-01 14:00:00', '2011-01-01 16:20:00',

'2011-01-01 18:40:00', '2011-01-01 21:00:00'],

dtype='datetime64[ns]', freq='140T')

(3.325740318906606 hr-3 hr)\*60min/hr = 19.544419134 min

(19.544419134 – 19) \* 60sec/min = 32.66514804 sec

32.66514804 - 32

2633 STRUCNUM = 10053, 5267 STRUCNUM = 10053, 7901 STRUCNUM = 10053, 10535 STRUCNUM = 10053, 13169 STRUCNUM = 10053

## Slicing Subsets of Rows in Python

Slicing using the [] operator selects a set of rows and/or columns from a DataFrame. To slice out a set of rows, you use the following syntax: data[start:stop]. When slicing in pandas the start bound is included in the output. The stop bound is one step BEYOND the row you want to select. So if you want to select rows 0, 1 and 2 your code would look like this:

*# Select rows 0, 1, 2 (row 3 is not selected)*

surveys\_df[0:3]

You might think that the code ref\_surveys\_df = surveys\_df creates a fresh distinct copy of the surveys\_df DataFrame object. However, using the = operator in the simple statement y = x does **not** create a copy of our DataFrame. Instead, y = x creates a new variable y that references the **same** object that x refers to. To state this another way, there is only **one** object (the DataFrame), and both x and y refer to it.

In contrast, the copy() method for a DataFrame creates a true copy of the DataFrame.

byte-by-byte.com/training

07/03/2022:

1. Clean ear
2. Laundry
3. Bills
4. Refrigerator?

07/03/2022 reset of priorities: The distance from the line to the data point is called a residual.

1. You’ve got the df\_data (that has all the relevant info in it) made- so take that to it’s conclusion:
   1. Divide out by the TOTALQTY and make percentages
   2. Resample or make the time components part of the individual databases “manually” by looking at the number of observations per year
   3. ?
2. Follow Ken Jee on the data science fundamentals video and get the linear regression by OLS (ordinary least squares) started, looked at, thought about, for a few..
   1. How many of the different dataframes are you going to use to complete an OLS analysis?
   2. How important is the graphing of best fit lines going to be?
   3. How many different CS (condition states) need to be explored?
   4. How will you know if the data is actually “taking you anywhere?” You actually might not be able to tell… is that ok?
   5. Notes from Ken Jee’s video
      1. OLS penalizes the really far off samples- and helps fit a line that will be most representative.
      2. Pandas as pd, numpy as np and matplotlib.pyplot as plt
      3. df\_data.isnull().any()
      4. What is a continuous variable?
      5. Sk learn linear regression and the stats models OLS linear regression
         1. Sk learn will add the constant in for you
         2. Stats models OLS will NOT add the constant for you.
         3. The constant is to figure out the location of the line’s start (y intercept).
      6. Actually getting into the linear regression modeling portion of the program:
         1. Starts out with a simple linear regression with only one variable: Odometer
         2. Imports
            1. from sklearn.model\_selection import train\_test\_split

separate data into train case and a test case.

* + - * 1. from sklearn.linear\_model import LinearRegression
        2. from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

to evealuate the regression model

* + - * 1. import math

to use root mean square error instead of mean square error

* 1. split data into a train and test set: training set is 70% of the data, test set is 30% of the data, and the random state is set to replicate the analysis later.
  2. Fit the linear regression data
     1. Using any of the sk learn algorithms means you create a linear regression object then fit the data to that object and afterwards get a score of the data- r-squared in this case, which is the amount of variance explained by this linear model. (in the KJee case, the 0.26 means 26% of the variance in price is explained by the number that is on the odometer). (ROUGHLY 18:00 min mark in the video.)
  3. Next we want to predict both the train set and the test set
     1. Evaluate the analyses: r-squared is one way, but first look at mean absolute error
        1. mae: takes an average of the distance of each data point from the line and takes the absolute value of that- won’t skew the results for values that are very high, doesn’t penalize as much- perhaps does not penalize as much as it should-
        2. root mean square error: KJee considers this to be a better decision criteria metric
  4. Same analysis of the data but now using the statsmodels.api, considered more useful for the summary information- CONSTANT MUST BE INCLUDED MANUALLY BECAUSE THEY DON’T DO IT FOR YOU!
     1. Omnibus statistic should be zero (0) which would suggest that the model is significant
     2. FOR YOUR MODEL IT MAY BE RELEVANT TO LOOK AT THE NUMBER OF DAYS BETWEEN OBSERVATIONS AND USE THAT AS PART OF THE REPRESENTATION OF THE MODEL ONCE YOU’RE ATTEMPTING TO MAKE A LINEAR EQUATION POST ANALYSIS!
     3. What needs to be the protocol for the different condition states? Which of them need to be analyzed rigorously? C1? C4? ?
     4. Line 176 in the KJee file, could I use pd.get\_dummies as a means of including other condition states in the analysis of a single condition state that I’m interested in looking for correlation with another CS. ????? We can’t interpret a categorical variable in multilinear regression
     5. Be sure to check the relationship between our actual and our projected data. See if the use of a different shape of line would help, make a better fit.
     6. Validate the data!!! Five-fold cross validation!

1. Make the “Report” section of the project
   1. Use the copy you made of the tds article to make an overview of your results.
   2. Be prepared for speedbumps in the final step…

filename | STRUCNUM | EN | TOTALQTY | CS1 | CS2 | CS3 | CS4

Remember, the hypothesis/goal is to determine when the condition state will approach 1- meaning that the element (EN) being evaluated has been completely overtaken by a particular condition state- (how will this effect the plots I will make? Need to watch the scale of the axes on the plots- as the scale may need to be adjusted).

06/26/2022: Regression- from the PlayingNumbers GitHub regression page: Where the data is read in on line 19 is where I should start after I’ve gotten all the dataframes created and made into the correct form with the CS1-CS4 made into percentages of the total (TOTALQTY) and the time factor added to the dataframes as a column. The time component does not need to be a factor until the individual dataframes have been assembled. The point of the STRUCNUM will become prevalent at this point as the use of the STRUCNUM as a means of placing the observations in order will be apparent- using the numerical order of the bridges as a means of making the time between observations of each EN relevant and that the condition state of each element will go from CS1 to higher, more perilous to the functioning of the overall bridge structure CS’s as time passes.

\*\*The removal of duplicates and nulls may not be necessary as I see the data as having been cleaned rather well since I’ve pent so much time getting it into a form that could be plotted and analyzed. \*\*

The snippets on lines 32 thru 37 may not be necessary because the nulls (or zeros) in the data- in the CS’s condition states in this case- are useful for our purposes because they may mean that there has been no change in observed condition in the field since previous observations have been made- thusly- the zeros are probably important.

06/18/2022: for each unique STRUCNUM from strucnum\_in\_all make the EN parked in the EN column next to that STRUCNUM for a single year’s particular df into a list of each EN for that STRUCNUM for that year. DO this for each STRUCNUM from strucnum\_in\_all for each year assessed then make the comparison of the lists of EN for each STRUCNUM per year- this will give you the EN common to each STRUCNUM across all years assessed.

Probably not doing the loops below- probably merge then select a subset of a dataframe and select specific columns to make into separate dfs and then concat those new dfs!

~~Loops:~~

~~Looks like I need a while/else loop statement to pull this off.~~

~~The while portion being while the entry in the STRUCNUM column = a STRUCNUM from the strucnum\_in\_all list/variable; make a list out of the entries in the EN column and save to a variable name eN\_in\_sTrucnum000000000XXXXX where the first 9 zeros represent the leading zeros as in the raw XML data which I have left unchanged in the manipulation and exploration of the data prior to any analysis.~~

~~The idea was to make lists for each STRUCNUM for each year and then compare those lists and make only the EN common across all years being assessed be the final data to be used for the individual year and then make the original df2017 df2018 df2019 df2020 and df2021 dataframes contain only those rows with the EN common across all years. The problem I’m not sure about is how to remove the rows of data from the original dataframes after all the lists have been compared! So I think I’ll go with the method of using merge to add data from different years that will append the succeeding years horizontally across the dataframe to make the set of data that meets the requirements. !!!!! How do I know the ENs will be made to have the correct requirements of STRUCNUM common across all years and STRUCNUM with only EN common across all years as the data to be concatenated be true? Using what method to take from the merged dfs the actual sets of data and then de-construct that data to make a cut down dataframe for each year and then be able to concatenate? GO back to dict\_att.py and use that miniaturized data to see…~~

06/18/2022: Update: The reason I am doing this is to make sure I have numbers of bridges across all years assessed that match (if STRUCNUM not present/assessed in all years it is eliminated) and that those bridges (STRUCNUM) have EN (element numbers) assessed in all those years as well. (If within the set of STRUCNUM present in all years there are EN not present in all years assessed then those EN are eliminated.)

06/13/22: The approach I have decided upon is to merge the dataframes on the STRUCNUM and EN as I had attempted before- but now I need to make sure I’m not omitting anything by not merging them at the same time. So I’m looking for a method to merge the dfs all at once. Update: Use the approach that gets the 8847 number of bridges first then look for an approach to merge individual dataframes on EN????

Try merging the longest two dfs on STRUCNUM and EN as these would have the most “excess” EN’s as part of their makeup- as all the shorter dfs would likely have fewer EN’s because the approach yielding the 8847 will mean the longer dfs will be “EN-heavy.” Follow this up by merging the third longest df to the resultant of the first two longest until all the dfs are merged- then get the “list” of ENs (somehow) from the df of all five of the merged years. (Is that possible????)

06/11/22: I need to iterate over the individual dataframes after they have been made for a single year- and I assume pull out numbers of observations for individual elements. i.e. how many EN = 12 or deck\_rc\_2017 are there in 2017? GO back to the getattr code used to come up with the individual dataframes for each variable as they occurred for the concatenated df and make the individual EN dataframes as they occur for a year!?!?!?!?! ??

06/12/22: Do I need to do any of the things I mentioned yesterday? Can I use the “original” method of the merge or join that did so on both STRUCNUM and EN in order to get the set of bridges for each year that would be a “uniform set” meaning that only bridges from all years would be considered because any bridges not observed in all years being evaluated would be eliminated from the set of bridges and also only EN (bridge elements, Element Number) observed in all years being considered that also are parts of the bridges observed in all years considered will be evaluated in this set of data. Thusly the bridges would be the same (i.e. uniform) for all years considered. This would mean that if a bridge is observed in all years being considered that only the elements (EN) of that bridge present in all years under consideration would be kept as part of the data as well. SO, didn’t I already come up with this group of bridges in the first data cleaning attempt? And would it not make sense to isolate that set of bridges across all years considered to come up with a means of making plots and performing regression analysis that has the same bridge part for the same bridge each year to see how changes in condition occur?

NOW, what is the purpose of the dictionary comprehensions I’m looking at doing that will also count out the number of each type of bridge element present in each year regardless of whether that element was present and observed and its condition recorded for that bridge in all other years being considered? Isn’t it to come up with the number of observations of each individual element per year and then manually (meaning I look at the numbers myself and decide what to do- not leaving it up to the computer) compare the numbers of each element observation across all years being considered and then decide how to proceed with regression? The point being to manually look at the numbers of each EN present in each year and see how closely the numbers match and then to decide which elements are best to be regressed and presented to the hiring managers at MAXX?  **I believe the answer is yes- and that I should use the earlier method of using a merge to get the bridges that appear in all years and whose bridge elements are present in all years and those elements are observed and the condition state of those elements is recorded in all years being considered.**

What is left to do?

1. Use the merge method from the scbDataCleaning.py file to come up with a group of bridges that have STRUCNUM present and observed and recorded in all years being considered and have the and also have the same EN present observed and recorded in all the bridges being considered. To clarify:
   1. IF a bridge (STRUCNUM) is not present in all years being considered that STRUCNUM (bridge) is eliminated from the data.
      1. How to accomplish: use sets? DONE (I believe)
      2. IF a bridge element present in a bridge *not* meeting the criteria outlined in a. above (i.e. a bridge or STRUCNUM that is not eliminated from the data because that STRUCNUM *IS* present in all years under consideration) is not present and its condition state observed and recorded in ***all years*** being considered, that bridge element (EN) is eliminated from the data. (so eliminate the bridges first if they aren’t present in all the years then eliminate the elements from the bridges if the elements are not present in all the years for those bridges)
      3. Nested tuples to handle the STRUCNUM as one column and the EN as the next column in order to make comparisons to have all bridge + bridge elements match across the years????
2. Continue with the examples of TDS (Canada climate data) to make the data being used contain a time component (probably need to add a column once the sets of bridges and elements for each year have been narrowed down) in order to make a regression analysis possible. This will probably mean that the time elapsed between observations will vary depending on the element (EN) being regressed.
3. Continue with the examples of KJee (craigslist.com vehicle prices) to create a regression analysis using OLS (ordinary least squares)

Is the tangent on tuples even necessary?

Tuples? Tuples are used to store multiple items in a single variable. Tuple items are ordered, unchangeable and allow duplicates.

I’m trying to make variables from a dictionary using the keys of the dictionary as a means to look through a dataframe that already exists and then make the value associated with that key equal to the number of times the key appears in a specific column of that dataframe. Use value\_counts to set the values of a dictionary equal to the number of times the key associated with that value appears in a column

06/07/22: Go back to this page <https://stackoverflow.com/questions/48681634/adding-a-string-to-all-keys-in-dictionary-python>

Use this (page shown above) to try to get a set of quantities of each element per year … this white whale I’m chasing.

????? Pay H2O (has it been paid already???) (as of 06/11/22 still hasn’t been checked)

Pay Power (done 06/07/22)

Pay Phone (done 06/05/22)

How to push changes to GitHub:

1. Navigate to the proper directory in GitBash (i.e. one level outside or above the directory where the files being pushed up are located, also known as the files that have been changed in some way since last having changes pushed)
2. Type git add -A
3. Type git commit -m “any type of message to yourself regarding the changes since last you pushed changes”
4. Type git push

06/01/2022: reset of priorities:

First let’s review where we were last time:

1. Got the columns needed to be numeric rather than string changed as such in the early part of the program (around line 142 at that point)
2. Made the CS1 – CS4 entries in the columns into percentages by dividing by the total (TOTALQTY).
3. Need to “resample” to attach time component to each observation
4. Need to perform linear regression.

05/30/2022: reset of priorities:

Clean out car so the oil can be changed.

Cancel the tablet on Verizon.

Cancel or go all in on T-Mobile (Just use it is all I’m saying, not switch everything over to it like your phone and all those things too).

1. Before all of those things:
   1. Make the CS numbers into percentages of the totalqtys.
2. tds\_CAN\_clim\_data.py as exercise for resampling/adding time data to the dfs.
3. KJeeCraigslist.py as exercise for linear regression.

05/22/2022: reset of priorities:

1. Make all the dataframes for each EN (bridge part aka element number)
2. Make the condition states into percentages (i.e. divide out the individual states by the totalqtys in each row) ((May need to search how to make sure you divide the cs in each row by the totalqty in each row so as to avoid trouble)).
3. Sort the dataframes based on the STRUCNUM to quasi create an order in which the observations are made, see this post: <https://stackoverflow.com/questions/37787698/how-to-sort-pandas-dataframe-from-one-column>
4. The above creates a possible problem in that the different years of data have already been concatenated and sorting the data again based on STRUCNUM once multiple years have been joined will cause, as an example data from 2018 to appear prior to 2017- because of the presence of a STRUCNUM that appears earlier in the numerical chronology than it should because of the – I was going to say absence of that STRUCNUM from prior years- but that shouldn’t be a problem because I’m always making the basis for inclusion for all data for each year be that the STRUCNUM for one year must be present in all successive years.

Linear Regression:

1. Fit a line to the data
2. Compute R – squared (R2)
3. Compute a p-value for (R2)

When one is parsing from xml to a pandas dataframe it is important to match the case of the elements in the tree in the code that parses the data into pandas, i.e.

df2021=parse\_XML("2021SC\_ElementData.xml", ["FHWAED", "STATE", "STRUCNUM", "EN", "EPN", "TOTALQTY", "CS1", "CS2", "CS3", "CS4"])

NOT

df2021=parse\_XML("2021SC\_ElementData.xml", ["fhwaed", "state", "strucnum", "en", "epn", "totalqty", "cs1", "cs2", "cs3", "cs4"])

05/17/2022: reset of priorities:

1. Perhaps read in the dfs as df1, df2, .. etc. and then pair them down to the columns you want as was done in the tds Canada exercise?

05/16/2022: reset of priorities:

1. Linear regression.
2. Clean up your actual workspace.
3. Place the same value or string in every cell of a dataframe column pandas.
4. (Resample, essentially) How to divide the number of observations into equal parts of a year pandas.

05/14/2022 reset of priorities:

Search: Attach time-stamp to data points Python.

05/08/2022 reset of priorities:

1. Brute force the part that makes the dataframes
2. Figure out how to resample depending upon number of “observations” in a given time period (a year in this case)
3. Get the regression portion done for at least some of the data
4. Make the “report” using the Medium/TDS article

04/26/2022 reset of priorities:

1. Is the order of the “observations” in the dfs created using getattr going to be chronological? ????
2. Add a column to the 4 different dfs and put the year in there to be pulled out during the getattr part of the program
3. Still need to “brute force” the dfs needed to perform the regression

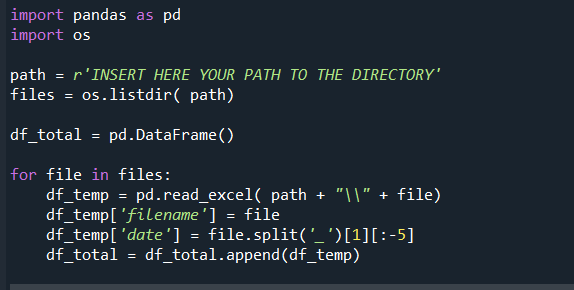
You create a for loop and perform the following manipulations per file:

Read in the file

Create a new column and assign the file name

Create a date column based on the filename

Append the file to a consolidated dataframe



04/25/2022 reset of priorities:

1. Going to “brute force” the dataframes for the individual bridge elements- i.e. deck\_rc = getattr(element\_df, '12')
   1. Typing out the above for each element as it is pulled from the concatenated df will be necessary after the class declaration is made- making the overall process less than automatic- hopefully figure this out later.
2. Make the modifications to the columns to make the Cs’s into rates of Progression of the state from one to the next or for the state to overtake a more significant portion of the overall element amount for a given bridge. ?????
3. Regression Analysis.
4. (Pre-packaged) Webpage report.

elements is the set of numbers representing the EN after the concatenated df has been created. el\_names is the dictionary of all the possible keys and values of the EN numeral and the name of the possible df respectively. The class df\_names may be capable of meeting my needs: I basically need the class to look at one set of items or the keys of the dict perhaps and see them as one set of items found in the concatenated df, take the instances of those keys as they would be found in the df, it would need to be looking at the EN column, and create an object of type df\_names that is then filled with the dataframes for each EN in the form it was pulled from the concatenated df - then I need the class to look at the second set of items in the dict, i.e. the values, and then make that value the name of a variable and set that variable equal to the dataframe corresponding to the numeral that denotes the EN from the element\_df object.

**def** filter\_el\_names(dictObj, callback):

newDict = dict()

# Iterate over all the items in dictionary

**for** (key, value) **in** element\_df.items():

# Check if item satisfies the given condition then add to new dict

**if** callback((key, value)):

newDict[key] = value

**return** newDict

This function accepts,

* A dictionary
* A function that accepts a key/value pair and returns True or False

This function iterate over all the key value pairs in dictionary and call the given callback function() on each pair. Items for which callback() function returns **True** are added to the new dictionary. In the end new dictionary is returned.

Re-set of priorities 04.21.2022:

1. Division (i.e. make percentages of the CS1-4) and add the column(s)
2. “Automate” the process of making the additional columns holding the percentages (for loop? Use the dict somehow in the process to turn all the dfs into dfs with the additional columns)
3. Resample of the data based on spreading all observations across a year’s time
4. Still want to try the whole iterate over the list thing? ??? …(to assign the variables? Look at that guy’s Medium article or GitHub?) (Not looking promising)
5. Transfer over the coding for the larger files in xml? (i.e. to the scbDataCleaning.py file?)
6. Regression analysis.
7. Make the report with the visualizations and package it up.

Re-set of priorities 04.18.2022:

1. Use the getattr to access the df of each EN and make a variable for each.
2. Make the CS1-4 into percentages of the total (TOTALQTY) as given for each element (EN) in the xml.
3. Make the necessary resamples to get the time component of each observation (i.e. is there a way to just make the program divide the number of observations for a year into equal segments and then graph the percentages versus the time component?)…
4. Run the regression analysis for the best fit and see what the data is trying to say.
5. Use the Medium article to make the “report” and hopefully nicely packaged data visualization into a webpage.

https://stackoverflow.com/questions/64932778/filter-using-a-for-loop-and-return-multiple-data-frames-in-python

https://www.geeksforgeeks.org/python-assign-multiple-variables-with-list-values/

Make the “test\_list” the list of numbers corresponding to the EN in the original XML i.e EN=12 means reinforced concrete deck-

Last page: https://stackoverflow.com/questions/46607685/using-a-loop-to-create-multiple-variables

Re-set of priorities for scbDataCleaning.py:

1. What have I finished that I had been doing?
   1. The removal from the concatenated database problem has been fixed- using the list(set.intersection(\*map(set, [a, b, c]))) as a means of finding the matches between datasets.
   2. So the “Final” database has been/can be created and can be drawn from and manipulated to create visualizations and plots to use for regression analysis, and then the next problem is the creation of rates or percentages as a means of making sense of the large but varied values of the different condition states (CS1-CS4) because plotting the actual values of the states would be problematic for making plots that would be proportional. Creating a model that could be used for analysis would likely be difficult if values for each bridge element are graphed with their dimensions still intact.
2. From the concatenated database the CS’s need to be converted to a rate of change (which may create a problem of having only a 3 year period to analyze- so the better way to go may be to divide the CS by the total quantity (TOTALQTY in the original xml files) and plot those for a means of regression and visualization.

Coding Font: Consolas

Re-set of priorities for scbDataCleaning.py:

1. start in dict\_att.py
   1. take the (~~elements~~)strucnum\_in\_all variable and create a “range” or other acceptable type of variable and use as a means of selecting rows from each dataframe prior to any merge or concatenation of the dataframes into a “final” dataframe to be used for regression or any other type of analysis.
   2. ~~See~~ [~~https://www.statology.org/pandas-select-rows-based-on-column-values/~~](https://www.statology.org/pandas-select-rows-based-on-column-values/) ~~for methods to accomplish~~
      1. Above didn’t work, but this did df[np.isin(df['A'].to\_numpy(), list\_of\_values)] from <https://stackoverflow.com/questions/12096252/use-a-list-of-values-to-select-rows-from-a-pandas-dataframe>
2. So having made all the above moves in 1. a. & b. and, one may then concatenate the dataframes. Or should I draw out all the different individual bridge elements and make individual dataframes of each of those before doing any concatenation?
   1. If this is the path we take do I need to make a range for all the different bridge element numbers that are involved so that I can make an automatic loop style process for the creation of the individual dataframes?
      1. Pull rows from each year based on their EN and make those rows into dataframes.
         1. How do I make sure the loop function knows which key is in use (i.e. which column?)
3. Convert the CS values of the concatenated dataframe to a rate? Or convert them to a percentage of the overall total for each condition? (e.g. CS2018 - CS2017) or (CSX2017/ CSTotal2017)

~~Can I make a dataframe that consists of the 4 years of STRUCNUM with the 4 column headings across the top like 17, 18, 19, 20 and then the corresponding STRUCNUM for each year printing down beneath each heading- kinda like an array (may have to make some NaN adjustments for length or perhaps solve that by broadcasting or something) and then have the function pull the common STRUCNUM out of all 4 columns and make a single list of those numbers?~~

Convert ~~list~~(set) to numpy array! Broadcast the arrays as necessary to make them of equal length so they can be compared for equality….

Built in function len() returns the size of the first dimension (i.e. the dimensions of a numpy array are (no. of columns x no. of rows).

A [list](https://www.geeksforgeeks.org/python-list/) in Python is a linear data structure that can hold heterogeneous elements they do not require to be declared and are flexible to shrink and grow. On the other hand, an array is a data structure which can hold homogeneous elements, arrays are implemented in Python using the [NumPy](https://www.geeksforgeeks.org/python-numpy/) library. Arrays require less memory than list.

The similarity between an array and a list is that the elements of both array and a list can be identified by its index value.

https://www.geeksforgeeks.org/convert-python-list-to-numpy-arrays/

MAY NEED TO MAKE ALL THE COLUMN HEADINGS IN THE DFs IN LOWER CASE!!!!

Make the numpy arrays, make it possible to compare the arrays of different length, compare the arrays for matches, return the matches.

<https://stackoverflow.com/questions/64637774/how-to-compare-one-list-to-multiple-lists-in-python-to-see-if-there-are-any-matc>

## Broadcasting

The term broadcasting is [used by the NumPy library](https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html) deals with arrays of different sizes. Keep in mind; lists are just 1-dimensional arrays. The concept of broadcasting among arrays has been leveraged by many of the most powerful machine learning libraries such as [Tensorflow](https://www.tensorflow.org/xla/broadcasting), [Theano](https://theano-pymc.readthedocs.io/en/latest/tutorial/broadcasting.html), [Octave](https://octave.org/doc/v6.1.0/Broadcasting.html), and others. The [term is defined by Scipy as such](https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html):

The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is “broadcast” across the larger array so that they have compatible shapes.

Would Broadcasting (since it adds multiple copies of the last element added to a list as necessary to match the length of another list to which it is being compared) be the means to compare the lists of different lengths?

Laundry

Milk

Walks

Dishes

Lunch for the week?

List comparisons:

1 Get the series as below value\_counts() , then sort the series. Make each series a variable.

2 Compare one series to another. Get the Boolean returns.

3 Look for functionality that will return the differences in the two series, or that will return the entries common to the two series.

1. Remove all the non matching rows from a dataframe
2. Need to get back to that place where the goal is to set the STRUCNUM of each df can be set equal to the STRUCNUM of all the other dfs and returns ‘True.’

The value\_counts() function is used to get a **Series**containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

It is important to note that value\_counts only works on pandas series, not Pandas dataframes. As a result, we only include one bracket df['your\_column'] and not two brackets df[['your\_column']].

**Drawback of using set() method:**

The set() method removes all the duplicate elements in the list. If your list contains duplicate elements, this solution will not work for you.

Update 03/03/2022:

-Need to convert the CS values to a rate of change in CS for each possible element and use as a means to do the regression analysis. The numbers might not plot all that well if not converted to a rate.

-Plot the rates vs the bridge number spread evenly across the year of the applicable data and the bridge numbers need to be a “label” if that is possible as a “feature” of the spyder software…

-May not need to join or merge anything on both STRUCNUM and EN. Probably only on need to join on STRUCNUM as the rate of change in CS should be applicable even if all the same bridge elements are not observed and recorded every year, although that would leave some gaps or zeros/NaN in the plots.

-Still need to get the total of bridges common to all 4 years computed, probably with the method that uses a set and then get all rows for those bridges selected and made into a data frame for each year that can then be concatenated one on top of the other then use those dfs as the inputs for the plotting tables. May be important to be careful as you need to think about possibility that the plots may work best once one for each year is plotted on one pate line plotted to each page and then the rationale that to some of the observations it may be necessary to assume that an EN will not be losing he[]] tome the job may be very general

1 Make the arrays for each STRUCNUM from a single dataframe from one year (one array from 2017, one from 2018, … etc.).

2 Get the number of unique STRUCNUM from each array, and then compare for quantity of unique from each individual df (2017, 2018, 2019, 2020) i.e. make sure the STRUCNUM from the Console in Spyder are in the actual dfs directly after they have been read in from the XML.

3 Think about only merging on the basis of STRUCNUM and just letting all of the ENs from all the years be in the dataframe so that the data for elements can be checked for rate of deterioration regardless of its observation in the same bridge year after year.

In Python "*The expression x and y first evaluates x; if x is false, its value is returned; otherwise, y is evaluated and the resulting value is returned.*"

More background on where I am in this project and what needs to happen next:

Make one “list” of

Join/merge the dfs as in the data cleaning process- and get the STRUCNUM common to each year (and possibly the other method you used to get the number of common STRUCNUM in the first method that came up with a different common set after the first one that I thought was the correct common set (the second method was a larger number of STRUCNUM than the first) - then pull the set of common STRUCNUM and all the accompanying data and concatenate them. Then make plots and perform the OLS (ordinary least squares).

Loops, dicts, keys and values…

Make a dict that uses EN as the key the key-value pairs

Parse or loop over the df making a new df for each value that uses the key as the means of selecting each row that has said key in the EN column

Possible number of different Bridge Elements:

Elements:

Deck and slabs – 13

12 deck\_rc

13 deck\_pc

15 topFlg\_pc

16 topFlg\_rc

28 stDeck\_og

29 stDeck\_cfg

30 stDeck\_corrOrtho

31 deck\_timb

38 slab\_rc

39 slab\_pc

54 slab\_timb

60 deck\_other

65 slab\_other

Superstructure – 36 (38)

102 cwBg\_steel

104 cwBg\_pc

“103 cwBg\_pc” (103 isn’t supposed to exist?)

105 cwBg\_rc

106 cwBg\_other

107 oGb\_steel

109 oGb\_pc

110 oGb\_rc

111 oGb\_timb

112 oGb\_other

113 stringer\_steel

115 stringer\_pc

116 stringer\_rc

117 stringer\_timb

118 stringer\_other

120 truss\_steel

135 truss\_timb

136 truss\_other

141 arch\_steel

142 arch\_other

143 arch\_pc

144 arch\_rc

145 arch\_masonry

146 arch\_timb

147 cbl\_mSt

148 cbl\_secSt

149 cbl\_secOthr

152 flrB\_steel

154 flrB\_pc

155 flrB\_rc

156 flrB\_timb

157 flrB\_other

161 spph

162 sgp

170 rrcf

171 miscSS

180 eqrcII

181 eqrcC1

182 eqrc\_Othr

Substructure – 32

202 col\_st

203 col\_othr

204 col\_pc

205 col\_rc

206 col\_timb

207 twr\_st

208 tres\_timb

210 pw\_rc

211 pw\_othr

212 pw\_timb

213 pw\_mas

215 abmt\_rc

216 abmt\_timb

217 abmt\_mas

218 abmt\_othr

219 abmt\_steel

220 pcf\_rc

225 pile\_st

226 pile\_pc

227 pile\_rc

228 pile\_timb

229 pile\_othr

231 pc\_steel

233 pc\_PrConc

234 pc\_rc

235 pc\_timb

236 pc\_othr

240 culv\_st

241 culv\_rc

242 culv\_timb

243 culv\_othr

244 culv\_mas

245 culv\_pc

250 tunnel

251 pile\_castSh

252 pile\_castDr

254 cSh\_stFH

255 cSh\_stPH

256 slopeScP

Joints – 10

300 joint\_sse

301 joint\_ps

302 joint\_cs

303 joint\_aws

304 joint\_oe

305 joint\_awo

306 joint\_othr

307 joint\_ap

308 joint\_ssp

309 joint\_sf

Bearings – 7

310 brg\_el

311 brg\_mov

312 brg\_ec

313 brg\_fxd

314 brg\_pot

315 brg\_dsk

316 brg\_othr

Approach Slabs – 2`

320 appSl\_pc

321 appSl\_rc

Railings – 5

330 br\_m

331 br\_rc

332 br\_timb

333 br\_othr

334 br\_mas

Culverts – 2

Bridge Management Elements:

Wearing Surfaces – 10

510 dws\_ac

511 dws\_cp

512 dws\_ep

513 dws\_timb

515 spc\_p

516 spc\_galv

517 spc\_ws

520 rsps

521 cpc

522 deck\_memb

# Make the total number of observations of each bridge element spread evenly across the total number of years inventoried. Or should it be spread the observations evenly across the year for which the observations have been made?

Still need to get the “period” on the individual “observations” correct through making of datetime objects accompanying the CS’s in increasing order of STRUCNUM and resampling. The x-axis entry of the scatter needs to be a time object of equal intervals for each observation. The ends of each df needs to be demarcated so as to make the years succinct and to know when each observation takes place (i.e. show the extents of observations for each year). (DONE. Accomplished this before concatenating the dfs together.)

How to handle the “common bridges (STRUCNUMs)” across all 4 years? Are there some elements where that will be necessary? Some where it won’t?

Resampling needs to be sorted out: i.e. how many “periods” per year are there and how will those be handled. (Divide the time for an entire year by the number of observations for a year, then start the first observation of the year at that first interval.)

[**GitHub - Schramm9/South\_CarolinaBridgeData: Data exploration and analysis of Bridge Condition data for South Carolina.**](https://github.com/Schramm9/South_CarolinaBridgeData)

Start button up arrow maximizes window

Start button down arrow twice minimizes window

In Pandas: Ctrl + left or right arrow moves cursor from anywhere between the two quotation marks to the mark in the direction you pressed (keeps the cursor between the two “connected” marks- or from inside a pair of quotes to the next punctuation mark.

I need to concatenate (I think) the dfs.

The dfs will have the same ENs each year- meaning the data from each of the structures common to all 4 years will be in all four years along the x-axis which will be the best method to scatter plot the data. How to make the key for the df first the STRUCNUM and then have the EN the key for an additional df? Is that even what I need to do? What have I already done with the merge commands? Is that equivalent? Can I concatenate something from what I have already pulled from the larger merged df? IS THIS A NaN SITUATION? i.e. make an EN column and repeat the EN in the column alongside the other rows (originally I wrote columns, i.e. have the EN repeated in the “other columns” – I think this was a mistake and I meant to say “then remove later”) in the df and then remove later before plotting?

To use .append(), you call it on one of the datasets you have available and pass the other dataset (or a list of datasets) as an argument to the method:

concatenated = df1.append(df2)

May need to make plots and look for when the CS1, CS2, etc. goes from some value to zero in succeeding years? Then find some way to make the follow on years values switching to the next condition state be “continuous” with the line from the previous year? Or overlay multiple graphs? (Discontinue the lines where the change in year causes a change of the previous condition state to drop all the way to zero because there is no longer any of that element in the previous state on that bridge….

The idea was to make the scatter plots use the number of structures in which the EN is observed be the number of resamplings for the entire year- then order the resamplings by ascending STRUCNUM. So the STRUCNUM must accompany each value used as a datapoint. So, it goes get the STRUCNUM common to each year- THEN

1. Get all entries of a common EN within the common STRUCNUM from each of the 4 years.
2. Create the number of resamplings by dividing the number of EN observed at each STRUCNUM evenly over the year’s time (recall that you have to do this for all four years, and spread out all the entries over a year’s time).
3. Order all entries (an entry is a CS value and a point in time at which that CS is deemed to have been observed during the year) by ascending STRUCNUM.
4. The CS (at its value on the Y-Axis) will represent the value of the CS of a single element (EN) on a bridge (STRUCNUM) at the “moment” of the observation of the condition state (CS) as it has been resampled in pandas. The visualization will represent the moment of observation (time, resampled for the year, resampling based upon the number of bridges with that element (EN) being present and observed in said bridge, and the period between observations being the year in days as arrived at by dividing the year by the number of applicable bridges- again the results using the “moments of observation” ordered by ascending STRUCNUM or by ascending CS) vs. the condition state (CS) observed at that instant.

For each (1st row of STRUCNUM)

Other concerns: making use of any additional from the FHWA website? Adding it into this analysis?

Scenario 1: May need to make the 4 dfs again and have the ENs and the associated CSs be more readily searchable by row rather than column, and the years be kept separate, unless I can think of some slick way of making the scatter plot without having the CS for following years be in columns associated to STRUCNUM increasing as a quasi index- in all likelihood the first set of values that will be plotted as STRUCNUM increases- i.e. I am modeling the STRUCNUM as increasing with time- will require no further sorting than having the df be created as it has been previously with the STRUCNUM increasing from lowest to highest as it already does in the raw data.

BUT I will need to have the STRUCNUMs repeat along the x-axis – and since that is likely impossible, I’ll have to time the STRUCNUM to a datetime rather than just expect them to repeat each year.

Scenario 2: The second scenario likely will require the same initial steps as the first listed above- however in this set the CSs will need to be related to the date of “observation” rather than the STRUCNUM related to the date as in Scenario 1.

Is the answer to just do away with the original merged dfs and join them together by slapping the 2018 set beneath the 2017 and the 2019 beneath the 2018 and so on? In other words concat the dfs instead of merging them? Can I make it look like I didn’t have to merge the dfs to find out what STRUCNUM were common to all 4 sets? If I use concat is it likely I’ll find more common STRUCNUM that were omitted from the original df20\_19\_18\_17? Basically I’ve found that the number of bridges unique to all 4 dfs is 8992. I thought it was around 9117.

New ideas for plots especially, i) scatter plots for elements with unit of each, using a resampling technique- make the order of the bridges increase from smallest to largest STRUCNUM vs CS numbers, i.e. the “chronology” of the “observations” of the CS are to be plotted with the observations for each year being made as if the observer started at the bridge with the lowest STRUCNUM and ending on the highest. I have the feeling this may produce a scatter plot that may be highly irregular. Another possible technique (as I see it) is to order the bridges based upon the CS value per bridge- which would possibly have some interesting outcomes that would be less “irregular” than the first method (based on STRUCNUM, lowest to highest), and would possibly have issues with multiple bridges having the same y-value on the plot (meaning there could be bridges at the same x-value that represent multiple bridges with the same observed CS, and should this be handled by having the multiple bridges right next to each other on the x-axis and putting the y-values (CS values) that are the same right next to each other? i.e. a line between the two would have a slope of zero)- but again, this would probably create a scatter plot that has a more smoothly upward sloping scatter.

In univariate time series, the data file only includes the date, and the data series. In order to utilize a supervised learning algorithm to model, we need to transform the data into input feature and output target. The approach to do it is using shift function with the number of lag observations, and the rolling window statistics

t = current time

obs(t) = observation at the current time

prior times are t-1, t-2, t-n corresponding obs are obs(t-1) obs(t-2) obs(t-n)

trend, seasonality and noise

#cols = df20\_19\_18\_17.select\_dtypes(exclude=['int']).columns

#df20\_19\_18\_17 = pd.to\_numeric([cols], errors='coerce').fillna(0).astype(np.int64)

#cols = df20\_19\_18\_17.select\_dtypes(exclude=['int']).columns

#df20\_19\_18\_17[cols] = df20\_19\_18\_17[cols].apply(pd.to\_numeric, downcast='int', errors='coerce')

#df20\_19\_18\_17['CS1\_17'] = df20\_19\_18\_17['CS1\_17'].astype('|S80')

#df20\_19\_18\_17['CS1\_17'].astype('float')

#df20\_19\_18\_17['CS1\_17'] = df20\_19\_18\_17['CS1\_17'].astype('float')

#df20\_19\_18\_17['CS1\_17'] = df20\_19\_18\_17['CS1\_17'].astype('str')

#df20\_19\_18\_17['CS1\_17'] = df20\_19\_18\_17.CS1\_17.astype(int())

#df20\_19\_18\_17['CS1\_18'] = df20\_19\_18\_17['CS1\_18'].astype('str') # #doesn't work

#df20\_19\_18\_17['CS1\_18'].dtype # still gives an object as dtype

#df20\_19\_18\_17['CS1\_18'].str.decode("utf-8")

#['CS4\_20', 'CS4\_19', 'CS4\_18', 'CS4\_17']

#df20\_19\_18\_17.CS4\_20.sum().hist()

You have the right, upon written request made within a reasonable time, to request from the Company (1) whether an investigative consumer report has been obtained about you, (2) disclosure of the nature and scope of any investigative consumer report and (3) a copy of your report. These reports will be prepared by Sterling Infosystems, 4511 Rockside Road, 4th Floor, Independence, OH 44131; Tel. # 800-899-2272; www.Sterlingcheck.com.

/\/\What am I trying to do?

A: use the resample feature of pandas/python in spyder.

What have I tried in this regard?

A: Looked at many examples on the web, with little in the way of results

What has been the shortcoming(s) of this approach?

A: I’ve missed critical requirements of the resampling, for instance that the index of the df must be in the datetime format.

What have I attempted going forward?

A: Reshuffled the index, and looked for the method used to actually populate the df with numbers once the additional entries had been created- i.e. realized that the total for each year will be divided by 12, and then the graph will “trend” a particular way as the year continues.

Are there any difficulties regarding the execution of this resampling method that are problematic?

A: The examples I’ve found tend to make the dataframe in a different way than I typically would- meaning that accessing the df in order to make plots is different than what is presented.

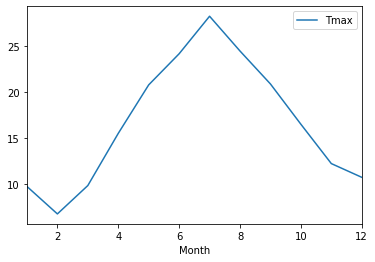
I also think the datetime conversions I need to make are done differently using the examples I’ve found.

Go through the process of creating the df as is done in the examples- and look at the datetime conversion and the index needs to be the datetime type for this to work.

ETFs = ETFs.resample(‘W’).agg([‘adjClose’ : ‘mean’, ‘high’ : ‘max’, ‘low’ : ‘min’, ‘volume’ : ‘sum’])

weather.plot(y=’Tmax’, x=’Month’)  
plt.show()

Python Pandas line chart



When you run this code, you’ll see the result like the graph shown here where the maximum temperature, Tmax, for each month of the year is plotted against 12 months.

The first line of the code above is the one that does the work of creating the plot. It calls the DataFrame method .plot() from the DataFrame weather and passes two parameters to that function, the first is the value for y, the value that will be plotted vertically, and the second is x, the value for the horizontal axis.

The second line of code refers to the the mathplotlib library, plt. plt.show() is a function from that library and does what you would expect, it displays the graph.

The x,y parameters are fundamental to drawing a line chart but plot() can take a number of other parameters, some of which you will come across later.

# **Simple charts**

You are going to discover a few types of chart. We start with the simple ones.

## Line charts

Line charts are suitable for visualizing data that changes continuously over time, so are a good way to show temperature changes as these tend to be gradual.

You can create more than just one type of plot and you can specify which type you want in two ways: you can pass a parameter, or you can modify the function call. The code above does not specify the type of plot because the default in Pandas is a line plot. But if you wanted to be specific you could pass the type of plot in the kind parameter like this:

weather.plot(kind=’line’, y=’Tmax’, x=’Month’)

Alternatively, there is a specific method for a line plot that you can use like this:

weather.plot.line(y=’Tmax’, x=’Month’)

These two alternatives produce exactly the same result.

Later, you will see how to produce bar charts, pie charts, scatter diagrams, histograms and box plots. In each case, you can specify the type of plot using the kind parameter or use the method call for that type of plot.

## Multiple line plots

What if you wanted to plot both the maximum and minimum temperatures in the same figure? It’s a reasonable thing to want to do and it’s easy to do. You create a list of y values like this:

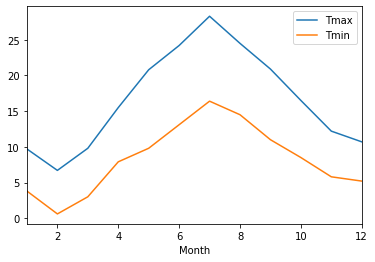
['Tmax','Tmin']

and assign this to the y parameter in the plot function.

Take a look at the following code and the resulting plot.

weather.plot(y=['Tmax','Tmin'], x='Month')  
plt.show()

Python Pandas line chart



You can see that the code is almost identical to the first plot but the y parameter now contains values for both the maximum and minimum temperatures (Tmax and Tmin) in a list. And the result is a graph above.

You could add more values to the list that are assigned to y, if we wished to, but in this data set we don’t have any more suitable values (it wouldn’t make sense to plot, say, temperatures and rainfall on the same graph because they are different measurements with different units i.e. degrees Celsius and millimeters).

So, just for illustrative purposes, we’ll use a little Pandas magic to create a new column and make a Pandas plot of that, too. In the code, below, we create a column Tmed which is the average of Tmax and Tmin (the sum of Tmax and Tmin divided by 2).

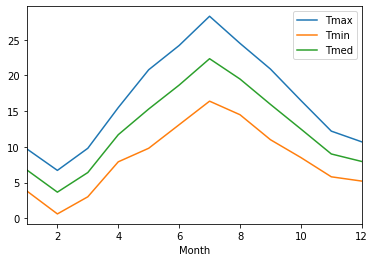
weather['Tmed'] = (weather['Tmax'] + weather['Tmin'])/2

If you want to see what it looks like you can print(weather) and you’ll see the extra column with the average values.

Now we plot as before but with the extra column added to the list of y values.

weather.plot(y=['Tmax','Tmin','Tmed'], x='Month')  
plt.show()

Python Pandas line chart



Unsurprisingly, the new y value draws a line drawn halfway between the maximum and minimum temperatures.

While line charts are great for plotting continuous values, some measurements are more discrete in their nature. Temperatures tend to change gradually without sudden leaps from one value to another but rainfall is a different matter.

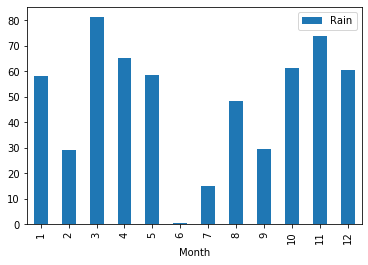
It’s not unusual for it to rain one day and not the next. Rain doesn’t change gradually but rather it can start and stop quite abruptly. A line chart is not really suitable for visualizing that sort of behavior — better to use a bar chart.

## Bar Charts

If you draw a bar chart of the rainfall data for 2018, you can see that the changes over time are more abrupt than the temperature data and, so, a bar chart representation is quite appropriate. Here’s what it looks like:

weather.plot(kind='bar', y='Rain', x='Month')  
plt.show()

Python Pandas bar chart



There is quite a difference in rainfall between May and June, and it is very clear from the bar chart that this is the case. A line chart would make it look like there was a smooth transition between the two months whereas this is unlikely to be the case.

This sort of representation makes to easier to spot phenomena like April showers — April being the month when it typically rains a lot in the UK. Except that, in 2018, April was less wet than March, and not much worse than May.

Ah well, that’s folklore for you.

Looking at the code you can see that it is very similar to the line chart code. The only difference is that the kind parameter is ‘bar’, rather than ‘line’. As we discussed above, the same result would be obtained if you called the specific bar chart method, like this:

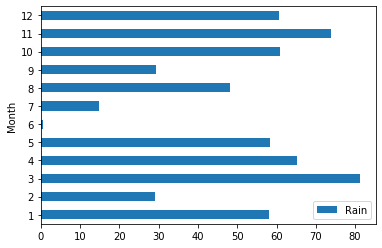
weather.plot.bar(y='Rain', x='Month')

I won’t continue to talk about the two ways of specifying a particular chart, suffice it to say that in all of the charts you can use either construction.

What if we want our bars to be horizontal? In that case we specify barh as the type of chart, as below:

weather.plot(kind=’barh’, y=’Rain’, x=’Month’)  
plt.show()

Python Pandas bar chart

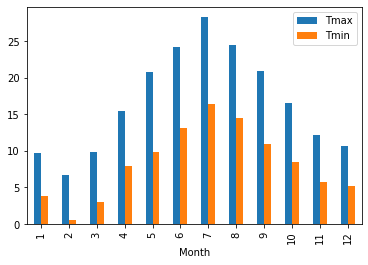


And, as you might expect, you can create multiple bars by adding a list of y values in the same way as the line chart.

Clearly, if you are going to plot two values on the same chart, the type of data needs to be similar. It would not make sense to have rainfall and temperature on the same chart as they are measured in different units. So, to illustrate a multiple bar chart we are going back to temperatures. Here is a bar chart that plots both Tmax and Tmin:

weather.plot(kind='bar', y=['Tmax','Tmin'], x='Month')  
plt.show()

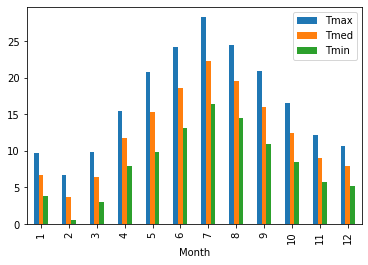
Python Pandas bar chart



And one that plots the three temperatures, as you did with an earlier line chart:

weather.plot(kind='bar', y=['Tmax','Tmed','Tmin'], x='Month')  
plt.show()

Python Pandas bar chart



## Scatter diagram

A scatter diagram plots a series of points that correspond to two variables and allows us to determine if there is a relationship between them.

Is there a relationship between the amount of sunshine in any particular month and the level of rainfall? Probably there is.

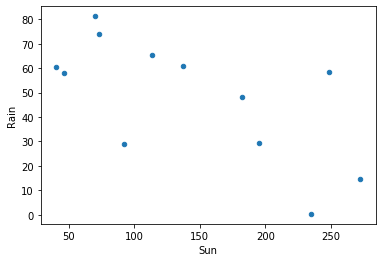
The scatter plot below plots Sun against Rain. There are 12 points, one for each row in the table, and the points plot the value of Rain on the vertical axis against Sun on the horizontal one.

It’s not particularly clear but you can see a vague linear relationship.

A straight line that was the best fit through the twelve points would start somewhere high up on the left and end up low on the right. That tells us that when Rain has a high level, Sun has a low one, and vice versa. Which common sense tells us is probably right — there is, generally speaking, an inverse relationship between the amount of sun and the amount of rain. Because when it’s raining it’s also cloudy and so there is less sunshine.

weather.plot(kind=’scatter’, x=’Sun’, y=’Rain’)  
plt.show()

Python Pandas scatter diagram



In this example, the relationship between the values is fairly obvious and if, from some hypothetical data set, we were to plot rainfall against umbrella sales, we might also see what we expected. But it can be useful to be be able to demonstrate such relationships when they are not so obvious.

## Pie charts

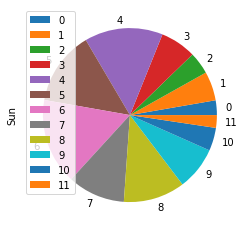
Pie charts are typically used to show what proportion of some value can be associated with a particular group or category. You might, for example, show peoples’ preferences for different types of fast food — 80% like pizza and 20% prefer burgers — or the market share of various types, or makes, of automobile.

Here we are going to see what proportion of the annual sunshine happens in each month.

So, the pie chart only takes one parameter for the data, in the case Sun. This is then plotted against the index of the DataFrame (the index is the value in the first column in the DataFrame). Here’s a first try at creating the pie chart:

weather.plot(kind=’pie’, y=’Sun’)  
plt.show()





This is the sort of thing that we want but, frankly, it could be better.

There are two problems.

One is that the legend, being quite big, obscures part of the chart and the second is that what you are interested in is the proportion of sunshine in each month, whereas what you have here is the proportion of sunshine in some categories labelled 0 to 11.

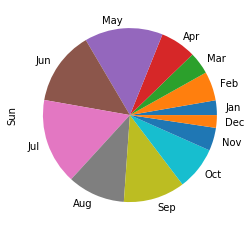
To fix this, you need to do a couple of things.

Firstly, you are going to change the index into something more meaningful by assigning a list of strings to the index of the weather DataFrame. You can see this is the first line of the code, below.

Secondly, you can simply dispose of the legend, as it doesn’t really add any value to the plot. You do this by adding a new parameter to the plot method. The parameter is legend and you simply set it to False.

weather.index=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']  
weather.plot(kind='pie', y = 'Sun', legend=False)  
plt.show()





# **Statistical charts and spotting unusual events**

The second set of charts are to do with statistics.

## Getting more data

You’ve been using a small Dataset up to now and the simple charts that you’ve used have been entirely appropriate.

Now, you are going to download a larger dataset. It’s the same format as before but covers several decades, not just one year.

Again, it is a csv file and you will get it from the same site as before. The following code downloads the data and stores it as a DataFrame, more\_weather. It then print the first four years (48 rows) of data:

more\_weather = pd.read\_csv('https://raw.githubusercontent.com/alanjones2/dataviz/master/londonweather.csv')  
print(more\_weather[0:48])Year Month Tmax Tmin Rain Sun  
0 1957 1 8.7 2.7 39.5 53.0  
1 1957 2 9.0 2.9 69.8 64.9  
2 1957 3 13.9 5.7 25.4 96.7  
3 1957 4 14.2 5.2 5.7 169.6  
4 1957 5 16.2 6.5 21.3 195.0  
5 1957 6 23.6 10.7 22.4 284.5  
6 1957 7 22.5 13.8 87.0 152.3  
7 1957 8 21.1 12.5 86.2 154.4  
8 1957 9 17.6 10.1 51.7 88.5  
9 1957 10 15.5 7.7 47.0 85.9  
10 1957 11 9.4 4.3 59.5 67.5  
11 1957 12 7.6 1.0 42.1 40.8  
12 1958 1 6.8 0.9 64.3 40.1  
13 1958 2 8.9 1.9 58.7 45.7  
14 1958 3 8.1 1.1 26.0 105.2  
15 1958 4 12.3 3.8 29.5 153.2  
16 1958 5 17.3 7.8 59.5 189.2  
17 1958 6 19.4 10.7 104.3 152.2  
18 1958 7 21.7 12.9 51.9 190.5  
19 1958 8 20.8 13.1 75.2 103.1  
20 1958 9 20.0 12.1 83.8 134.8  
21 1958 10 14.9 8.3 50.7 94.2  
22 1958 11 9.7 4.4 50.7 40.8  
23 1958 12 8.0 2.7 85.1 29.6  
24 1959 1 5.7 -1.1 54.8 76.2  
25 1959 2 7.4 1.2 2.4 54.8  
26 1959 3 11.9 4.4 43.8 103.9  
27 1959 4 14.2 6.3 52.9 139.1  
28 1959 5 18.7 8.0 21.9 221.4  
29 1959 6 22.1 11.1 16.2 231.6  
30 1959 7 24.7 13.3 86.5 276.9  
31 1959 8 24.2 13.7 27.6 240.0  
32 1959 9 22.7 10.6 5.1 209.2  
33 1959 10 17.8 8.5 46.9 150.1  
34 1959 11 10.8 3.5 53.5 53.0  
35 1959 12 9.3 3.0 75.7 30.2  
36 1960 1 6.9 1.8 47.9 34.4  
37 1960 2 7.9 1.6 48.0 80.1  
38 1960 3 10.2 4.5 33.9 65.0  
39 1960 4 14.3 4.6 12.4 156.1  
40 1960 5 18.4 9.3 45.6 181.7  
41 1960 6 22.1 12.1 42.8 248.6  
42 1960 7 20.1 12.4 67.2 139.7  
43 1960 8 20.3 11.8 60.8 150.9  
44 1960 9 18.5 10.5 75.3 128.4  
45 1960 10 14.2 8.2 155.5 75.2  
46 1960 11 11.2 4.5 89.5 69.4  
47 1960 12 6.9 2.1 56.5 44.5

To get a feel for this new set of data you can use the Pandas method describe. The following code prints out a description of the Rain column:

print(more\_weather.Rain.describe())count 748.000000  
mean 50.408957  
std 29.721493  
min 0.300000  
25% 27.800000  
50% 46.100000  
75% 68.800000  
max 174.800000  
Name: Rain, dtype: float64

You can see from this that there are 748 rows of data (representing 748 months, that’s over 62 years), the mean monthly rainfall over that time was a little over 50mm, the minimum in any month was 0.3mm and the maximum over 174mm.

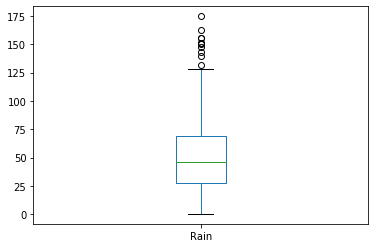
But if you want to communicate that data graphically, the charts that you’ve seen, so far, are not much help. A boxplot, however give you a great summary of the data in one simple graphic.

## Box plots

The code below is a box plot of the Rain data and it contains a great deal of information in a single graphic.

more\_weather.plot.box(y='Rain')  
plt.show()

Python Pandas box plot



This is also called a box and whisker plot because of the lines, or whiskers, coming from the top and bottom of the plot.

Here’s how you interpret the graphic.

The box itself represents the range of the Rain data between the first and third quartiles. Quartiles are simply the boundaries when you split the data into quarters. So the first quartile (Q1) is at the 25% mark — 25% of the data points are below Q1 and 75% is above it. The third quartile (Q3) is at the 75% mark and so has 25% of the data points above Q3 and 75% below.

And as you probably realize, the second quartile (Q2) is the one halfway through the data, 50% the data points are above Q2 and 50% below.

So, the box itself, represents 50% of the data: the top of the box is Q3 and the bottom of the box is Q1. The horizontal line inside the box is Q2, which is also the median: the middle value of all the data points

By convention, the whiskers are set to a value calculated from the inter quartile range (IQR). The IQR is the value of Q3 minus Q1 and the ends of the whiskers are set at 1.5 times the IQR, from the top and the bottom of the boxes.

The idea is that the bulk of the data is represented by the box and whiskers and anything beyond then are considered outliers.

In the graph, each outlier is represented by a circle — there are 8 outliers in the plot above.

The numbers down the left are, of course, the values that we are measuring, in this case, monthly rainfall in millimeters.

## The outliers are when it was really, really wet!

There’s a lot of information in a boxplot and you can see at a glance the shape of the data that you are looking at: where the bulk of the values lie and what outliers there are in the data. You can see that the monthly rainfall varies quite a lot, from less than 1mm to around 125mm. That, by London standards, is a range from really very dry to really rather wet. The outliers that represent extremely wet months but in the 748 months for which we have data, there are only 8 of them.

To summarize, then, in the box plot:

* The centre line is the second quartile (Q2) and is also the median of the data
* The bottom of box is the first quartile (Q1) and is also the median of the bottom half of the data
* The top of the box is the third quartile (Q3), and is also the the median of the top half of the data
* The whiskers extend to 1.5 times inter quartile range (IQR) , which is Q3-Q1, from edge of box
* The circles are outliers, the individual values that lie beyond the end of the whiskers

## Histograms

You could see from the box plot of rainfall that half the rainfall was in the range around 25 to around 75 millimeters, that the bulk was between roughly 0 and 125 and that there were a handful of outliers when Londoners got completely soaked.

But you can look at this sort of distribution in more detail with a histogram.

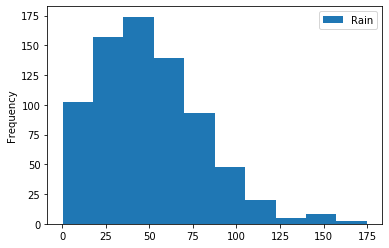
The diagram, below, is a histogram of the monthly rainfall in our data — a histogram is plotted by setting the kind parameter to ‘hist’.

You can see that the data is split up into ranges, or bins, each being represented by a bar. The width of the bar is the range of values and the height of the bar is the number of times that values in that range have occurred.

The default number of bins is 10, so when you run the code below, you will get 10 bars.

more\_weather.plot(kind='hist', y='Rain')  
plt.show()

Python Pandas histogram



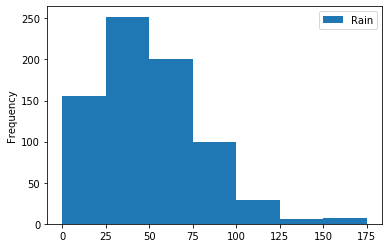
## More bins

You can adjust the number of bins by setting the bins parameter. You can set this to a number such as bins=15 or with a list of values which represent the boundaries of the bins. For example, say you wanted to take a closer look at the outliers that you found in the box plot earlier. Those outliers are in the range of about 125 to 175mm, so you could make sure that your bins match those ranges.

The code below sets the bins parameter to a list of 8 values which, because these represent the boundaries, gives us 7 bins each representing a range of 25mm. The last two bins represent the outliers and you can see in the new histogram that, indeed, they are not particularly significant.

more\_weather.plot(kind='hist', y='Rain', bins=[0,25,50,75,100,125,150,175])  
plt.show()

Python Pandas histogram



Let’s take another view. You could decide that, according to the previous plots, normal rainfall is in the range 25 to 75mm and that everything else is unusual. So, to indentify the frequency of unusual events you could display 3 bins, one representing unusually dry weather, a second for normal weather, and a third that records unusually wet weather.

The code below give you 3 bins, representing 0 to 25mm (unusually dry), 25 to 75mm (normal) and 75 to 175mm (unusually wet).

more\_weather.plot.hist(y=’Rain’, bins=[0,25,75,175])  
plt.show()

Python Pandas histogram



The resulting graph shows us that there are around 450 normal months, and a similar number of unusually wet and unusually dry months — about 150 of each.

# **Pandas Plot utilities**

A few useful things:

## Multiple charts

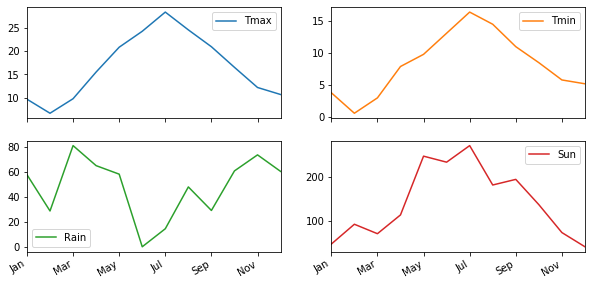
You can also create a set of separate charts for each series of data points.

We set the x and y values as usual but, in addition, we specify a parameter subplots as being True (the default is False) and, if we wish, we can set the layout as you can see below and the size of the individual plots using the layout and figsize parameters. Each of these parameters takes a list as a value. In the case of layout the first value in the list specifies the number of rows and the second one they number of figures in each row. For figsize, the first value is the width of the figure and the second its height.

Try changing the values in the list to see what effect they have.

weather.plot(y=['Tmax', 'Tmin','Rain','Sun'], subplots=True, layout=(2,2), figsize=(10,5))  
plt.show()

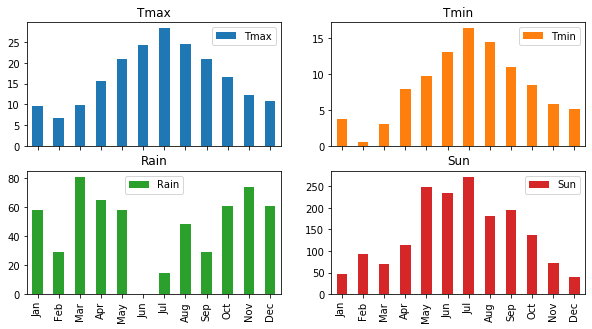
Python Pandas line charts



Here’s a set of bar charts:

weather.plot(kind='bar', y=['Tmax', 'Tmin','Rain','Sun'], subplots=True, layout=(2,2), figsize=(10,5))  
plt.show()

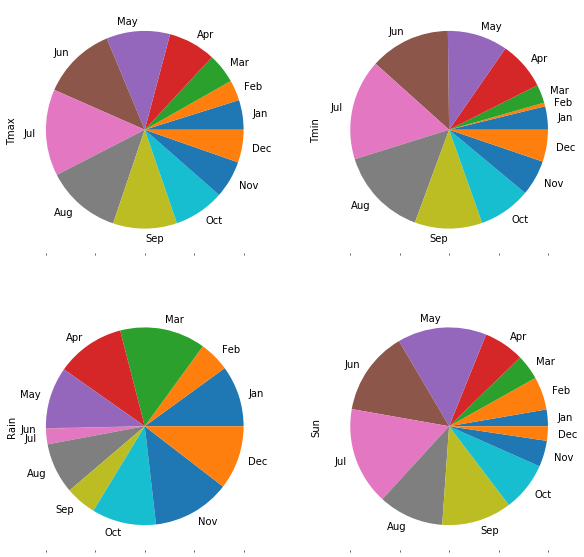
Python Pandas bar charts



And a set of pie charts:

weather.plot(kind='pie', y=['Tmax', 'Tmin','Rain','Sun'], subplots=True, legend=False, layout=(2,2), figsize=(10,10))  
plt.show()



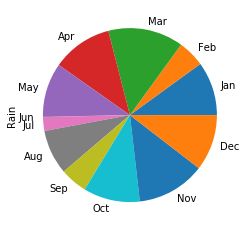


## Saving the Charts

This is all very well but maybe you want to be able to use the charts that you produce. If you want to use them in a presentation or document, then it would be useful to be able to export them as image files that you can include in another file. The simple way of saving the images is like this:

weather.plot(kind='pie', y='Rain', legend=False)  
plt.show()  
plt.savefig('pie.png')





We’ve used functions from mathplotlib before and this is just another one called savefig(). This function You can see that the name of the file is specified as a parameter and the type of image that you save is assumed from the file extension — in this case a png file called “pie.png”. As we haven’t specified a path, the file will be saved in the same directory as the notebook or program.

# **What you have learned and what more you can learn**

You have seen how to create various charts using Pandas and how they can be used. One the that you have probably realized working through these examples, is that you are not really getting an accurate picture of the data using these sorts of graphics.

You could see more detail by simply increasing the size of the plots (using the parameter figsize) but the thing to bear in mind is that these figures used to communicate broad ideas about data, not to provide a detailed analysis.

Use these plots in documents, web pages or in presentations but if more detail is needed, you really need to provide your audience with proper numbers.

Finally, you can download the datasets used in this tutorial, as well as the example code as either a Jupyter Notebook or a plain old Python program:

* [London Weather 2018](https://raw.githubusercontent.com/alanjones2/dataviz/master/london2018.csv)
* [More London Weather](https://raw.githubusercontent.com/alanjones2/dataviz/master/londonweather.csv)

STATE\_CODE\_001,STRUCTURE\_NUMBER\_008,RECORD\_TYPE\_005A,ROUTE\_PREFIX\_005B,SERVICE\_LEVEL\_005C,ROUTE\_NUMBER\_005D,DIRECTION\_005E,HIGHWAY\_DISTRICT\_002,COUNTY\_CODE\_003,PLACE\_CODE\_004,FEATURES\_DESC\_006A,CRITICAL\_FACILITY\_006B,FACILITY\_CARRIED\_007,LOCATION\_009,MIN\_VERT\_CLR\_010,KILOPOINT\_011,BASE\_HWY\_NETWORK\_012,LRS\_INV\_ROUTE\_013A,SUBROUTE\_NO\_013B,LAT\_016,LONG\_017,DETOUR\_KILOS\_019,TOLL\_020,MAINTENANCE\_021,OWNER\_022,FUNCTIONAL\_CLASS\_026,YEAR\_BUILT\_027,TRAFFIC\_LANES\_ON\_028A,TRAFFIC\_LANES\_UND\_028B,ADT\_029,YEAR\_ADT\_030,DESIGN\_LOAD\_031,APPR\_WIDTH\_MT\_032,MEDIAN\_CODE\_033,DEGREES\_SKEW\_034,STRUCTURE\_FLARED\_035,RAILINGS\_036A,TRANSITIONS\_036B,APPR\_RAIL\_036C,APPR\_RAIL\_END\_036D,HISTORY\_037,NAVIGATION\_038,NAV\_VERT\_CLR\_MT\_039,NAV\_HORR\_CLR\_MT\_040,OPEN\_CLOSED\_POSTED\_041,SERVICE\_ON\_042A,SERVICE\_UND\_042B,STRUCTURE\_KIND\_043A,STRUCTURE\_TYPE\_043B,APPR\_KIND\_044A,APPR\_TYPE\_044B,MAIN\_UNIT\_SPANS\_045,APPR\_SPANS\_046,HORR\_CLR\_MT\_047,MAX\_SPAN\_LEN\_MT\_048,STRUCTURE\_LEN\_MT\_049,LEFT\_CURB\_MT\_050A,RIGHT\_CURB\_MT\_050B,ROADWAY\_WIDTH\_MT\_051,DECK\_WIDTH\_MT\_052,VERT\_CLR\_OVER\_MT\_053,VERT\_CLR\_UND\_REF\_054A,VERT\_CLR\_UND\_054B,LAT\_UND\_REF\_055A,LAT\_UND\_MT\_055B,LEFT\_LAT\_UND\_MT\_056,DECK\_COND\_058,SUPERSTRUCTURE\_COND\_059,SUBSTRUCTURE\_COND\_060,CHANNEL\_COND\_061,CULVERT\_COND\_062,OPR\_RATING\_METH\_063,OPERATING\_RATING\_064,INV\_RATING\_METH\_065,INVENTORY\_RATING\_066,STRUCTURAL\_EVAL\_067,DECK\_GEOMETRY\_EVAL\_068,UNDCLRENCE\_EVAL\_069,POSTING\_EVAL\_070,WATERWAY\_EVAL\_071,APPR\_ROAD\_EVAL\_072,WORK\_PROPOSED\_075A,WORK\_DONE\_BY\_075B,IMP\_LEN\_MT\_076,DATE\_OF\_INSPECT\_090,INSPECT\_FREQ\_MONTHS\_091,FRACTURE\_092A,UNDWATER\_LOOK\_SEE\_092B,SPEC\_INSPECT\_092C,FRACTURE\_LAST\_DATE\_093A,UNDWATER\_LAST\_DATE\_093B,SPEC\_LAST\_DATE\_093C,BRIDGE\_IMP\_COST\_094,ROADWAY\_IMP\_COST\_095,TOTAL\_IMP\_COST\_096,YEAR\_OF\_IMP\_097,OTHER\_STATE\_CODE\_098A,OTHER\_STATE\_PCNT\_098B,OTHR\_STATE\_STRUC\_NO\_099,STRAHNET\_HIGHWAY\_100,PARALLEL\_STRUCTURE\_101,TRAFFIC\_DIRECTION\_102,TEMP\_STRUCTURE\_103,HIGHWAY\_SYSTEM\_104,FEDERAL\_LANDS\_105,YEAR\_RECONSTRUCTED\_106,DECK\_STRUCTURE\_TYPE\_107,SURFACE\_TYPE\_108A,MEMBRANE\_TYPE\_108B,DECK\_PROTECTION\_108C,PERCENT\_ADT\_TRUCK\_109,NATIONAL\_NETWORK\_110,PIER\_PROTECTION\_111,BRIDGE\_LEN\_IND\_112,SCOUR\_CRITICAL\_113,FUTURE\_ADT\_114,YEAR\_OF\_FUTURE\_ADT\_115,MIN\_NAV\_CLR\_MT\_116,FED\_AGENCY,DATE\_LAST\_UPDATE,TYPE\_LAST\_UPDATE,DEDUCT\_CODE,REMARKS,PROGRAM\_CODE,PROJ\_NO,PROJ\_SUFFIX,NBI\_TYPE\_OF\_IMP,DTL\_TYPE\_OF\_IMP,SPECIAL\_CODE,STEP\_CODE,STATUS\_WITH\_10YR\_RULE,SUFFICIENCY\_ASTERC,SUFFICIENCY\_RATING,STATUS\_NO\_10YR\_RULE

45, 2SC0300,1,8,0,00000,0,00,013,4690,'Ballast Creek ',,'Cuba Street ','1.1 km S. of Malecon Dr. ',99.99,0,0,,,32200440,080401910,7,3,73,73,09,1941,2,0,50,2011,0,6.7,0,0,0,1,0,1,1,4,0,0,0,A,1,5,1,04,0,00,12,0,67,9.1,109.7,0.2,0.2,7.3,7.8,99.99,N,0,N,0,0,6,6,7,6,N,1,26.1,1,15.7,4,6,N,5,9,8,38,1,109.7,711,24,N ,Y60,N ,,0711,,13,0,13,2011,,,,0,N,2,,0,0,,1,0,0,0,5,0,,Y,6,55,2031,,Y, , , , , , , , , , , ,0, ,75.8,0

Hypothesis:

(originally) To determine when the bridges might approach condition states of CS4.

(now) To determine when the bridge elements with a unit of EACH might approach condition states of CS3 or CS4.

Primmer: The elements with units of square feet or linear feet are such that any model that could be trained by the computer will be meant to determine how many square feet or linear feet of that element might be at said condition state at a point in the future. The results arrived at may conceivably be larger (square feet) or longer (linear feet) than the actual bridge element being considered- i.e. a 400 foot girder can’t have some specific condition states applied to more than 400 feet of it. The model must know the cutoffs of the units applied to each condition state- the model will presumably consider the condition states to grow at a rate determined by regression, and as such may say that a condition state of CS2 as an example will overtake all of a particular element at a point in the future. At present any model would not know the extent of an element that may be in such a condition- meaning the model will just assume growth of the condition upon the 400 foot girder- but the model at present cannot know that the girder is only 400 feet long and in all likelihood will just say the condition will effect more and more of the element over time. I’m saying that a model telling us when in the future a state of CS2 will equal for instance- 800 feet for a 400 foot girder is erroneous. More data about each bridge and each bridge element is needed- and more guidance from the FHWA or the state is needed. Is it acceptable for all 400 feet of a 400 foot long girder to be under a condition state of CS2? CS3? Is it ok for all of a major load carrying element to have a condition of “FAIR?” Would the authorities deem it necessary to classify such an element as CS3 (“POOR”) or worse if its entire length was in such condition? I would like to know the answer to that question. More specifics of the bridge under consideration need to be known, the lengths and areas of the crucial elements or the intended service life of the bridge (and the date it was built) could be explored- but at this time I do not possess that data. SO, at present it’s conceivable to determine the future “life expectancy” of the elements that have a unit of each because those items do not have an indeterminate quantity- if an item is evaluated to a condition state and the amount of said element effected by that condition state cannot change as time passes but is determined by its condition in the field, and is determined and recorded by a professional tasked with making those calls. There’s just the item (element) and how soon its condition may go from one state to another for those with units of EACH.

s = pd.value\_counts(df20\_19\_18\_17.EN)

s1 = pd.Series({'nunique': len(s), 'unique values': s.index.tolist()})

s.append(s1) # methods like this create a list that uses the values from the iterable object as the index of the new object- in the order in which they occur.

CAL TRANS:

3.2.5 - Miscellaneous Superstructure Elements

EN 147 Cables - Main Steel

EN148: Cables - Secondary Steel

EN 149: Cables - Secondary Other

EN 161: Steel Pin and Pin & Hanger Assembly

EN 162: Steel Gusset Plate

Probably the document to pay closest attention to is the download from Purdue that is called Element level bridge inspection.

Manual for Bridge Element Inspection

|  |  |  |  |
| --- | --- | --- | --- |
| Element No. | Element Name | Unit of Measure | Desc |
| 12 | Deck- Reinforced Concr. | Sq. Ft. | Reinforced Concrete Deck ( Driving Surface) |
| 13 | Deck- Prestressed Concr. | Sq. Ft. | Prestressed Concrete Deck ( Driving Surface) |
| 15 | Top-Flange - Prestressed Concr. | Sq. Ft. |  |
| 16 | Top Flange - Reinforced Concrete | Sq. Ft. |  |
| 28 | Steel Deck - Open Grid | Sq. Ft. |  |

Need to determine the elements changing from CS3 and CS4 and create a linear regression to project when the elements not already in CS3 will switch from CS3 to CS4.

Q: Would the idea of regression for each bridge be too specific? A: probably it would be too specific because each bridge probably has a threshold unique to itself that isn’t evident here in the data.

I would say that the most necessary portion of the project is to create a regression analysis and say that the basis of that analysis is that the rate at which most of the bridges in SC are progressing through the relevant condition states (CS1 through CS4) is the same for all because the individual specifics of each bridge, as I stated earlier, are not present in the data.

1. Sum all CS\_X columns in pandas
2. Make a plot that consists of each year’s values on the same plot

Sum all of the contents of a column- where can the sum be stored? (iloc)??

https://stackoverflow.com/questions/43745301/converting-column-from-dataframe-to-float-for-sum-usage-python-pandas

https://medium.com/@robertopreste/from-xml-to-pandas-dataframes-9292980b1c1c

<https://stackoverflow.com/questions/53645882/pandas-merging-101>

### Setup of the dataframes

np.random.seed(0)

left = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': np.random.randn(4)})

right = pd.DataFrame({'key': ['B', 'D', 'E', 'F'], 'value': np.random.randn(4)})

left

key value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right

key value

0 B 1.867558

1 D -0.977278

2 E 0.950088

3 F -0.151357

### Different names for key columns

If the key columns are named differently—for example, left has keyLeft, and right has keyRight instead of key—then you will have to specify left\_on and right\_on as arguments instead of on:

left2 = left.rename({'key':'keyLeft'}, axis=1)

right2 = right.rename({'key':'keyRight'}, axis=1)

left2

keyLeft value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right2

keyRight value

0 B 1.867558

1 D -0.977278

2 E 0.950088

3 F -0.151357

left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')

keyLeft value\_x keyRight value\_y

0 B 0.400157 B 1.867558

1 D 2.240893 D -0.977278

### Avoiding duplicate key column in output

When merging on keyLeft from left and keyRight from right, if you only want either of the keyLeft or keyRight (but not both) in the output, you can start by setting the index as a preliminary step.

left3 = left2.set\_index('keyLeft')

left3.merge(right2, left\_index=True, right\_on='keyRight')

value\_x keyRight value\_y

0 0.400157 B 1.867558

1 2.240893 D -0.977278

Contrast this with the output of the command just before (that is, the output of left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')), you'll notice keyLeft is missing. You can figure out what column to keep based on which frame's index is set as the key. This may matter when, say, performing some OUTER JOIN operation.

https://stackoverflow.com/questions/53645882/pandas-merging-101/65167327#65167327

09-12-2021 MAXX Potential continuing project work for application.

Copying content and questions of the web page as the application is in this form and starting to make some responses and notes to see how my process evolves and any changes to my approach or tendencies as I complete the process of getting an apprenticeship.

1. Standard first and last name, email, city and state, phone
   1. Chris Schramm
   2. [schrammcj@gmail.com](mailto:schrammcj@gmail.com)
   3. 9192727669
   4. Winston-Salem North Carolina
2. Upload resume:

Steps in execution of a data project:

1. Identify a clear objective, the question you want to answer, the product you want to build, etc.
   1. Can a dataset be used to optimize or “pick” the coordinates for the shape of a bridge?
      1. i.e. can a shape of a 2-dimensional bridge be “chosen” by a dataset based on a span and a load magnitude?
2. Hypothesis: Fairly simple, that the bridges will see more elements and parent elements approach a condition state of CS4 as time progresses. The trick will be to make/train a model or analytic that will show how soon the bridges will reach the CS4 critical state- basically be able to predict the service life of bridge components.
   1. Note: Probably a good idea to find info on the meanings behind the numbers in the manual used to instruct field inspectors- and how those numbers will relate to the service life of a component.
   2. <https://www.analyticsvidhya.com/blog/2020/09/10-things-know-before-first-data-science-project/>

**Is there anything you would like to highlight from your previous experience that would benefit you as an Apprentice? \***

*\*Not sure to include this\** A lot of what has set me apart and made my characteristics helpful to past employers as well as my current employer is my willingness to care little about the opinion’s others may hold of me. Line 151 of my scbDataCleaning.py is a good example of how preparing an application such as this and preparing for an interview are very different than going to work. I spent entirely too much time looking for an efficient method to change data types in that portion of the file but eventually had to give that up for the sake of time and to submit this application. To be certain, my goals are to be productive, efficient and to produce a superior finished product- those are my goals. The clumsy means of changing the data types of the dataframe I have worked with for this application notwithstanding- I do endeavor to be efficient and find the quickest means of producing results while I work. But if my current job has taught me anything (a job I am very ready to discontinue) it is that I do what I do without concern for what others may think of my methods, or of me personally for that matter. Succinctly, I don’t do what I do to be loved.

I am an engineer; I possess a Bachelor of Science in Civil Engineering and a Bachelor of Science in Environmental engineering from NC State University. I have roughly 7 years’ experience as a structural engineer working in bridge and building design, 3.5 years’ experience as a mechanical designer, 10 years’ experience in Computer Aided Drafting and I have completed a course in SolidWorks Fundamentals as well as a certificate offered by the software company (Dassault Systems) known as CSWA or certified SolidWorks associate (SolidWorks is a drafting program that is used by engineers and drafters to create three dimensional representations of solid objects and especially to create assemblies of sets of objects so as to see interactions between them and avoid interferences.) I have also sat for and passed the FE (Fundamentals of Engineering) exam (many years ago) although I also sat for and was unsuccessful at the Professional Engineering licensure exam a few years later. Even for the moderately successful engineer those are all very standard things and I’m not trying to pile on the accolades for myself.

I have also completed a coding bootcamp through the University of North Carolina at Chapel Hill and Trilogy Education Services that resulted in a certificate in full stack development. These things will benefit me as an apprentice because I am a naturally curious person and would really like to know how things work down to the *nth* level, I want to be able to understand the way a computer works and how coding languages are created and developed- so to facilitate better working with computers and being able to approach challenges with a better chance of success. My previous experience with computers and especially coding and the computer science classes I took as an undergrad were very challenging and tended to scare me off from attempting to move forward and try to get deeper into using computers for things other than spreadsheets and other more typical software. I tended to see computers as a tool and not as an answer to how to approach the solving of larger problems- and I tended to “hope” things I tried would work- and unfortunately sometimes I still do think that way (2 most frustrating things about coding: My code isn’t working and I don’t know why, my code *IS* working and I don’t know why). My approach now is to remember that the computer does only what it is told to do, and that on it’s own, regardless of how “powerful” a machine is due to it’s processor speed or memory or storage, that it can ultimately be a “pretty dumb beast” as a computer science professor used to say.

More importantly with regard to my prior experiences I have learned that my strength is my work ethic and that as fun and exhilarating as being a “natural” when first starting to use a new tool, or software or CAD package, learning through doing and learning through explaining to others who may need help is very effective and rewarding. I can say with some certainty that without my work ethic I wouldn’t have become an engineer or even graduated with the degrees I received. My work ethic is what I use to approach challenges every day.

What have you been working on since Career Lab? \* Provide any GitHub Repository Links, Code Snippets, Project Descriptions, etc.

Description: given a length or span ℓ and a number of load joints along the bottom chord of a bridge truss,

Instead of above idea

Pseudo code for the project as related to the main problems associated with the data to be solved:

1. Compare no. of rows for each individual structure for year 2020 to year 2019- i.e. not all structures were inspected and given the same number of entries from one year to the next.

Ok, nevermind, the code needed was just a simple merge with the keys set to on=[‘STRUCNUM’, ‘EN’]

I’ve been working full time, unfortunately I do not already work in tech- I don’t even work indoors, but I do have a demanding job, that is one thing that has kept me from applying sooner- but also I am interested in broadening my skillset to include data analysis/engineering. So there has been a learning curve to contend with on my part.

I’m going to perform a data analysis on a set of data about the conditions of roadway bridges. I am not fluent in Python- hadn’t even touched it before this undertaking and I am stumbling and struggling through (I work full time- and unfortunately I don’t presently work in tech, or even indoors so it has been a tough road) using an IDE to perform an analysis but I’m confident I can at least start to see the results of the data and be able to start to draw some conclusions about the data. The project is meant to analyze the condition of the bridge elements over time. Bridge elements are things such as the roadway surface, the superstructure (i.e. beams and girders supporting the roadway, made of steel or concrete mostly), the substructure (columns, piers, abutments, caps also typically steel and concrete), as well as other numerous structural elements that are outlined in the SNBIBE (Specification for the National Bridge Inventory Bridge Elements) document, which is also available on the internet. The bridges I am looking into are in the state of South Carolina.

I have been working to familiarize myself with the processes and practices of data science since finishing career lab. I have been using spyder as an IDE to do this and I am working on a project that relates to my prior career experience as an engineer. The github repository that contains what I’ve been doing is <https://github.com/Schramm9/South_CarolinaBridgeData>.

I have been parsing data into spyder/pandas dataframes from xml files available from the FHWA (Federal Highway Administration).

The individual bridges are identified by a Structure Number (STRUCTNUM in the dataframe) and the element that has been assessed for each year is not always the same for each bridge from one year to the next- so this is a challenge. I am learning to multi index the dataframes in pandas to be able to compare the condition of each bridge element from one year to the next where possible. The bridge elements are assessed for condition each year in one of four (4) states and the with CS-3 and CS-4 being serious and poor conditions respectively- and these would typically require further analysis and possibly repair. I am working to create plots of data with the Condition State of each element that was assessed for an individual bridge and how that condition changes for the years of data available- and thus to attempt to see a trend that may be visible about the individual element and how it may effect the performance and serviceability of the structure.

I also plan to look for trends that will show if there is a tendency for some distinct elements of the structures (bridges) to progress to condition states more readily than other elements of the structure. I am also still learning to make the data more usable in the sense that I can create plots that would point out “condition state of reinforced concrete deck vs. time” rather than something like “condition state of element no. 12 vs. time”

This analysis/application is still in the development stage and is not ready to be used to draw conclusions- but from what was said by facilitators in career lab I believe it is ok for me to send my application now, and I am still working on this data project.

This question doesn’t ask about the passion piece involved in this project for me but I’ll throw this comment in because it was emphasized during career lab that applicants should show his/her passion with regard to what an applicant intends to accomplish and work towards through a career in tech. I’ll say this (quickly I hope), as an engineer I have witnessed coworkers put into ethically compromising situations by their superiors and I have been in compromising positions as an engineer myself- I have “blown the whistle” so to speak when I’ve found myself in these situations- it is not an activity for the faint of heart- but engineers are supposed to protect the welfare of the public above all other objectives. Ask yourself what these names have in common over the last 10-12 years: Massey Energy, Glaxo Smith-Kline, Toyota, Takata Corp., BP, Boeing? All of these companies have had widespread and/or large scale failures in the last several years starting around 2009. I know at least 5 of the 6 companies listed had fatalities associated with the failures. Nearly all of the failures associated with these companies were preventable and looking at the legal discovery for each of these, individuals who were usually engineers raised a red flag prior to their occurrences- and they were overruled. Being able to use data to point out ways to protect the public sounds like my speed. (Let’s recall: Boeing’s catastrophic failures which resulted in 346 fatalities were caused by software!)

The application I am trying to create/problem I’m attempting to solve is one of how to optimize the design of a bridge truss

**Why are you interested in a career in technology? \***

To me a career in technology would mean that I no longer must attempt to talk anyone I would work with into looking for more productive means of accomplishing the goals of a particular enterprise. Usually these goals require a willingness to seek out and learn new technologies. In any activity that I am truly interested in doing and am passionate about I seek to be a difference maker and to work on things that can be significant to the bettering of society. I have a “technical” background to begin with being an engineer- and although it seems forgotten by many, the first objective of any engineer is to protect the welfare of the public-and doing this can mean being a difference maker because you often have to take actions that may be unpopular. I have been disappointed by how little success I’ve had finding ways to use technology in doing the things that are significant and that can make a difference usually because someone prevented me from doing those things, they wouldn’t let me even make an attempt, because learning those things for the rest of the organization would be too cumbersome or purchasing such things is thought too expensive. There can be a lot of foot dragging and unwillingness on the part of my past supervisors to embrace change or to try new means of production. I have been frustrated many times in my career by how I can see a career path and see ahead how the individuals that have been my superiors have become stagnant in their careers- and how that can lead to seeing one’s career as just a grind that one then tends to “wait out” and I see that as tremendously unsatisfying- and I see it as a reason I have perhaps not lasted in those prior jobs as a result. In all my most recent jobs, there has been an appetite on the part of myself and perhaps that of some coworkers for more challenging things to work on, more productive means of completing our work and a willingness to take the time to create the technological means to avoid repetitive tasks by using such technology so that more challenging tasks could be undertaken as a result of those gains in productivity- and I have been frustrated by the superiors charged with making decisions about those opportunities nearly every time- resulting in hardly any noticeable gains in said productivity. This has led to me finding myself stuck doing those repetitive, uninteresting tasks again and again, and has made me very frustrated in my career and overall as well. So, a career in technology would mean I stay engaged in what I am doing, and that I’ll feel like I am on the path to achieving my potential- and that I’ll have an opportunity to be a difference maker. A career in technology will mean that I am no longer the individual asking my boss to let me try a new software or for the opportunity to develop a new piece of software or for a chance to create a better approach to solving the problems and meeting the challenges of a particular enterprise, because I would work with and for individuals who think in those terms already.

**Why are you interested in an Apprenticeship with MAXX Potential? \***

I am interested because I want to work on a team of like minded professionals who want to be challenged in their careers and want to meet those challenges through using tech and are willing to support each other to do so- I know to do this will require plenty of effort- which is something I have in a more than adequate amount as I have developed and maintained what I think of as a great work ethic throughout my career. One former employer even said of me in a performance review that in this regard I possess “qualities that we can’t teach you.” I wish to connect that work ethic and willingness and drive to meet challenges that are difficult and require work with opportunities to do difficult things- and to have as many mentors as possible around helping me to do so. The opportunity as presented during career lab sounds like what I have described: Opportunities to take on Challenging tasks while having the support of individuals and mentors who want to take on and meet challenges of the same type.

**What are your plans if you are not hired with MAXX Potential? \***

As much as I am very hopeful to be hired for an apprenticeship with MAXX Potential- even if that does not happen, I will still pursue a career in technology- I have no future with my present employer, although I doubt I will have the flexibility to point out my interest in data analysis as much because MAXX has is one of the few organizations that allows for the range of entry level experiences in the variety of disciplines that were discussed in Career Lab. But I will still pursue a career in tech even if not hired for an apprenticeship through MAXX Potential. My current “career” for lack of a better word, is not sustainable for me for a variety of reasons that I won’t go into here but, I know I must continue to learn and keep myself aware of and as trained as I can keep on the tools and techniques of full stack development and data science/data analysis in order to be successful in my pursuit.

Are you willing to relocate? \*

 Yes



 It Depends



 No



Would you be willing to relocate: \* Please select all of the following which apply to best describe your willingness to relocate before, during, or after your apprenticeship at MAXX Potential.

 Prior to the Apprenticeship



 During the Apprenticeship



 For a post-Apprenticeship Career Opportunity



Are you willing to relocate outside of your current state? \*

 Yes



 No



Please provide any feedback you would like us to know regarding your preferences for relocation.

<https://www.tutorialspoint.com/the-elementtree-xml-api-in-python>

**To parse XML file**

Let us now read back the 'myfile.xml' created in above example. For this purpose following functions in ElementTree module will be used

**ElementTree()** This function is overloaded to read the hierarchical structure of elements to a tree objects.

tree = et.ElementTree(file='students.xml')

**getroot()** This function returns root element of the tree

root = tree.getroot()

**getchildren()** This function returns the list of sub-elements one level below of an element.

children = root.getchildren()

In following example, elements and sub-elements of the 'myfile.xml' are parsed into a list of dictionary items.

import xml.etree.ElementTree as et

tree = et.ElementTree(file='employees.xml')

root = tree.getroot()

students = []

children = root.getchildren()

for child in children:

employee={}

pairs = child.getchildren()

for pair in pairs:

employee[pair.tag]=pair.text

employees.append(student)

print (employees)

**Output**

[{'name': 'aaa', 'age': '21', 'sal': '5000'}, {'name': 'xyz', 'age': '22', 'sal': '6000'}]

## To modify XML file

We shall use iter() function of Element. It creates a tree iterator for given tag with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order.

Let us build iterator for all 'marks' subelements and increment text of each sal tag by 100.

import xml.etree.ElementTree as et

tree = et.ElementTree(file='students.xml')

root = tree.getroot()

for x in root.iter('sal'):

s = int (x.text)

s = s+100

x.text=str(s)

with open("employees.xml", "wb") as fh:

tree.write(fh)

Our 'employees.xml' will now be modified accordingly.

We can also use set() to update value of a certain key.

x.set(marks, str(mark))

<https://www.fhwa.dot.gov/bridge/nbi/element2020.cfm>

Above is location of the xml bridge data.

https://medium.com/@robertopreste/from-xml-to-pandas-dataframes-9292980b1c1c

[XML](http://www.w3.org/TR/xml/) is a markup language used to represent and distribute data structures which can be often difficult to create using more standard tabular formats.

Basically, the XML format is similar to [HTML](https://en.wikipedia.org/wiki/HTML) (which is another markup language, indeed), in that data are organised in elements, which define the type of information exposed, and each element contains the actual value in the form of content or attributes.

The [XML page](https://en.wikipedia.org/wiki/XML) on Wikipedia offers an extensive overview of all the details and technicalities of this format, but the key concepts are simple. Each piece of information is delimited by a specific tag, like this:

<data>  
<student name="John">  
<email>john@mail.com</email>  
<grade>A</grade>  
<age>16</age>  
</student>  
<student name="Alice">  
<email>alice@mail.com</email>  
<grade>B</grade>  
<age>17</age>  
</student>  
<student name="Bob">  
<email>bob@mail.com</email>  
<grade>C</grade>  
<age>16</age>  
</student>  
<student name="Hannah">  
<email>hannah@mail.com</email>  
<grade>A</grade>  
<age>17</age>  
</student>  
</data>

In this example, each student is represented by a <student> element, which has a name attribute containing the name of a specific student. Each of these elements has then sub-elements defined by the <email>, <grade> and <age> tags; between these tags the actual data content referring to the given student is present. Let’s say this data is saved in an XML file called “students.xml”.

We can think of this structure as a [pandas](https://pandas.pydata.org/) DataFrame in which each student represents an observation, with its name attribute being the main identifier and the sub-elements being other features of the observation. A tabular representation of these data would be like this:

| **name** | **email** | **grade** | **age** |
| --- | --- | --- | --- |
|  | John | john@mail.com | A | 16 |
|  | Alice | alice@mail.com | B | 17 |
|  | Bob | bob@mail.com | C | 16 |
|  | Hannah | hannah@mail.com | A | 17 |

OR

name,email,grade,age

John,john@mail.com,A,16

Alice,alice@mail.com,B,17

Bob,bob@mail.com,C,16

Hannah,hannah@mail.com,A,17

So we want to find a way to convert XML-structured data to a more functional table.

Given the structure of XML files, we can represent them as a tree, and this is the approach used by the [xml.etree.ElementTree](https://docs.python.org/3/library/xml.etree.elementtree.html)Python module. The parsing of our “students.xml” file starts at the root of the tree, namely the <data> element, which contains the entire data structure.

import xml.etree.ElementTree as et   
  
xtree = et.parse("students.xml")  
xroot = xtree.getroot()

Now we can iterate through each node of the tree, which means we will get each student element and grab its name attribute and all of its sub-elements to build our dataframe.

for node in xroot:   
s\_name = node.attrib.get("name")  
s\_mail = node.find("email").text  
s\_grade = node.find("grade").text  
s\_age = node.find("age").text

In order to get the name attribute, we use the attrib.get() function, while the text content of each element can be retrieved using the find() function of nodes.  
Each iteration will return a set of data that can be thought as an observation in a pandas DataFrame; we can build this procedure as follows:

import pandas as pd   
import xml.etree.ElementTree as et   
  
xtree = et.parse("students.xml")  
xroot = xtree.getroot()   
  
df\_cols = ["name", "email", "grade", "age"]  
rows = []  
  
for node in xroot:   
s\_name = node.attrib.get("name")  
s\_mail = node.find("email").text if node is not None else None  
s\_grade = node.find("grade").text if node is not None else None  
s\_age = node.find("age").text if node is not None else None  
  
rows.append({"name": s\_name, "email": s\_mail,   
"grade": s\_grade, "age": s\_age})  
  
out\_df = pd.DataFrame(rows, columns = df\_cols)

The downside to this approach is that you need to know the structure of the XML file in advance, and you have to hard-code column names accordingly.  
We can try to convert this code to a more useful and versatile function, without having to hard-code any values:

import pandas as pd  
import xml.etree.ElementTree as et  
  
def parse\_XML(xml\_file, df\_cols):   
"""Parse the input XML file and store the result in a pandas   
DataFrame with the given columns.   
  
The first element of df\_cols is supposed to be the identifier   
variable, which is an attribute of each node element in the   
XML data; other features will be parsed from the text content   
of each sub-element.   
"""  
  
xtree = et.parse(xml\_file)  
xroot = xtree.getroot()  
rows = []  
  
for node in xroot:   
res = []  
res.append(node.attrib.get(df\_cols[0]))  
for el in df\_cols[1:]:   
if node is not None and node.find(el) is not None:  
res.append(node.find(el).text)  
else:   
res.append(None)  
rows.append({df\_cols[i]: res[i]   
for i, \_ in enumerate(df\_cols)})  
  
out\_df = pd.DataFrame(rows, columns=df\_cols)  
  
return out\_df

An interesting piece of code is the dict comprehension in:

{df\_cols[i]: res[i] for i, \_ in enumerate(df\_cols)}

where a dictionary in the form column name: node value is created and added to the list of rows. These will finally be used to create the complete dataframe.

If we apply our function to the “students.xml” file using parse\_XML("students.xml", ["name", "email", "grade", "age"]), the result is precisely the table we saw above.

This is a more efficient implementation of the XML parsing function, although we still need to know the basic structure of the input XML document we want to parse.

000000000000004

000000000000004

import pandas as pd  
import xml.etree.ElementTree as et  
  
def parse\_XML(xml\_file, df\_cols):   
"""Parse the input XML file and store the result in a pandas   
DataFrame with the given columns.   
  
The first element of df\_cols is supposed to be the identifier   
variable, which is an attribute of each node element in the   
XML data; other features will be parsed from the text content   
of each sub-element.   
"""  
  
xtree = et.parse(xml\_file)  
xroot = xtree.getroot()  
rows = []  
  
for node in xroot:   
res = []  
res.append(node.attrib.get(df\_cols[0]))  
for el in df\_cols[1:]:   
if node is not None and node.find(el) is not None:  
res.append(node.find(el).text)  
else:   
res.append(None)  
rows.append({df\_cols[i]: res[i]   
for i, \_ in enumerate(df\_cols)})  
  
out\_df = pd.DataFrame(rows, columns=df\_cols)  
  
return out\_df

Categorical data: Typically unitless, an individual’s language, or gender etc. 1 for female 0 for male, … numerical values have no mathematical meaning in this context however.

Discrete Units: a separate part of something larger, i.e. a room is a discrete space in a house, a crankshaft is part of an engine, if something is discrete it has it’s own space (an ice cube comes from an ice tray but has it’s own discrete compartment in that tray).

Nominal data: values that represent discrete units and are used to label variables- but these variables would have no quantitative value. Nominal data can be put in any order and the meanings of those values would not change. Can be of the type that contains only two categories, or can be the type that is a list (like a list of languages).

Ordinal Data: data values represented by discrete ordered units- so it is like Nominal Data, except that the ordering of the data matters. Ordinal data is usually employed to measure non-numeric features like happiness or customer satisfaction.

<https://datacarpentry.org/python-ecology-lesson/05-merging-data/>

<https://hackersandslackers.com/compare-rows-pandas-dataframes/>

**def** **dataframe\_difference**(df1: DataFrame, df2: DataFrame, which=**None**):

"""Find rows which are different between two DataFrames."""

comparison\_df = df1.merge(

df2,

indicator=**True**,

how='outer'

)

**if** which **is** **None**:

diff\_df = comparison\_df[comparison\_df['\_merge'] != 'both']

**else**:

diff\_df = comparison\_df[comparison\_df['\_merge'] == which]

diff\_df.to\_csv('data/diff.csv')

**return** diff\_df

git clone or

git init (ssh key from github post creating new repo)

git add -A

git commit -m “(message about the commit)”

git push

https://nationalbridges.com/nbiDesc.html