**12/16/2023: CAL\_BridgesData.py,**

**3:30p Line 493 check array content of each year procedure**

**4:17p Line 563 merge different dataframe years procedure**

**4:35p Line 804 slicing of the large 2016 thru 2023 dataframe procedure**

**np.array([5, 15, 25, 35, 45, 55]) This code snippet creates a 1-dimensional numpy array that looks like this: [5, 15, 25, 35, 45, 55]. When the .reshape((-1, 1)) is applied to it the result is a 2-d array with 6 rows and 1 column:**

**[[ 5],**

**[15],**

**[25],**

**[35],**

**[45],**

**[55]]**

**Need to add the prefix or suffix to the keys in the filtered\_dfs dict to distinguish the variables in there from the variables in the other zeros\_ones\_CS1\_dict dictionary to avoid any conflict when the individual keys are separated out from the dictionary in order to make the dataframes associated with each (formerly key, now) variable different from each other. This can be done using a strategy similar to the function add\_key\_desc- be sure to add the description while the dictionary is still in tact!**

Reshaping the data using **.reshape((-1, 1))** ensures that the input data has the correct shape for many scikit-learn algorithms that expect features to be represented as a 2-dimensional array, even if you originally have a 1-dimensional array.

The estimated function has the equation f(x) = b0 + b1x . Your goal is to calculate the optimal values of b0 and b1 that minimize SSR (sum of squared residuals) and determine the estimated regression function.

1. Definition of the research question:

Determine when a bridge element in the given location (one of the 50 US States in this case) is likely to have its condition state (CS1, CS2, CS3 and CS4) change from one state to another. When will an element from the bridge change go from being in the state known as CS1 to CS2 as an example. When will an element go from CS3 to CS4? The ultimate question of the analysis is to gauge the lifespan of individual elements in the bridges of a particular US State.

The variables of interest are:

Dependent variables:

Condition States (CS1-CS4), the total quantities associated with each element (TOTALQTY) which determines the percentage of the element that exists in a particular condition state

Independent variables:

Time. The data is collected yearly, and the data in this case is for the years 2016 through 2022. The data is assumed to be collected continuously (although that is probably unrealistic) meaning that any one element in the highway system is

Location of the bridge. The location data for each bridge is presently no more sophisticated than to give which of the 50 US States in which the bridge is located. The location of each bridge should be considered more rigorously for the purpose of a detailed analysis taking into account latitude and longitude and the proximity of the bridge to corrosive environments like salt air and industrial areas, but also tremendously important to consider is the average daily traffic a bridge experiences, and generally depends on the location of the bridge. (Presently the model is not using any type of latitude and longitude data to adjust the rate at which any bridge element is assumed to progress through the different condition states.)

1. Data Collection:
   1. The data comes from the FHWA (Federal Highway Administration) website and is available in possibly multiple formats- the format used in this analysis being that the data is parsed from a file in the XML format.
   2. No additional steps have been taken to procure additional data for the analysis.
2. Data Preprocessing:
   1. Outliers have been removed from the

|\|

What would cause this plotting function to throw the error KeyError: ‘Time’

from scipy.stats import linregress

# Begin data visualization procedure

def plots\_pre\_post\_outliers(ones\_dict, filtered\_dfs):

for key in ones\_dict.keys():

df1 = ones\_dict[key]

df2 = filtered\_dfs[key]

# Convert datetime columns to datetime values

df1['Time'] = pd.to\_datetime(df1['Time'])

df2['Time'] = pd.to\_datetime(df2['Time'])

# Perform linear regression for dataframe 1

slope1, intercept1, \_, \_, \_ = linregress(mpl\_dates.date2num(df1['Time']), df1['CS1'])

line1 = slope1 \* mpl\_dates.date2num(df1['Time']) + intercept1

# Perform linear regression for dataframe 2

slope2, intercept2, \_, \_, \_ = linregress(mpl\_dates.date2num(df2['Time']), df2['CS1'])

line2 = slope2 \* mpl\_dates.date2num(df2['Time']) + intercept2

# Plot the dataframes and best fit lines

plt.figure()

plt.scatter(df1['Time'], df1['CS1'], label='ones\_dict')

plt.scatter(df2['Time'], df2['CS1'], label='filtered\_dfs')

plt.plot(df1['Time'], line1, color='blue', label='Best Fit Line (ones\_dict)')

plt.plot(df2['Time'], line2, color='red', label='Best Fit Line (filtered\_dfs)')

plt.xlabel('Time')

plt.ylabel('CS1')

plt.title(f'Plot for key: {key}')

plt.legend()

# Format x-axis as dates

date\_format = mpl\_dates.DateFormatter('%Y-%M-%D')

plt.gca().xaxis.set\_major\_formatter(date\_format)

plt.gca().xaxis.set\_major\_locator(mpl\_dates.AutoDateLocator())

# Display the equation of each line on the plot

plt.text(0.1, 0.9, f'Line (element\_dfs): y = {slope1:.2f}x + {intercept1:.2f}', transform=plt.gca().transAxes)

plt.text(0.1, 0.8, f'Line (filtered\_dfs): y = {slope2:.2f}x + {intercept2:.2f}', transform=plt.gca().transAxes)

plt.show()

plots\_pre\_post\_outliers(ones\_dict, filtered\_dfs)

# End data visualization procedure