how could the dataframes in the dictionary be merged again along columns (axis = 1) but by merging on two of the columns common to each dictionary, and for multiple dataframes in the dictionary? Assuming the merged\_df would result in the longest (in rows) possible merged\_df by comparing all of the dictionaries by number of rows prior to making each merge?

How can the dataframes in a dictionary of dataframes – all dataframes in the dictionary being of differing lengths in rows- be merged along columns (i.e. axis = 1) on the basis of multiple common columns (on = [“col\_1”, “col\_2”, … “col\_n”] in a manner that will check the dataframes inside the dictionary prior to each merge for length in rows and only start by merging the two longest dataframes to each other followed by merging the next longest dataframe into the first merge, and continue in this fashion until all the dataframes have been merged into a resultant merged\_df?

An existing dictionary is a dictionary of dataframes, and one column from each dataframe must be copied and assigned to each of its respective keys in the new dictionary- after the keys of the new dictionary have been made by modifying the original keys of the original dictionary. The keys of the original dictionary take the form df + a four digit year like 2016 as an example. how could this be accomplished?

Using an existing dictionary of dataframes, create a new dictionary from the original by modifying the keys of the original dictionary and then copy one specific column of the original dictionary to the new dictionary assigning that column to its respective new key. The new keys shall take the form of b\_16 for example- as the original keys are of the form df2016 as an example. How could this be accomplished?

A pandas dataframe has a column ‘year’ with a 4-digit year in each row. The total number of rows in the dataframe covers a set of observations that occur over a years’ time. The period of time between observations is to be the same over the course of a single year, so the number of rows is to have a years’ time divided evenly across all the observations/rows. The coding for such a function must have the ability to accommodate datetime data types of any scale required to allow for the period between observations to be equal regardless of how many rows are present in the dataframe. The first observation is to occur at midnight of January 1st of the year designated in the ‘year’ column and the final observation is to occur just before midnight on December 31st of the same year. The function will then create a column with the heading ‘time of obs’ meaning time of observation, that will show the date and time at which each observation occurred.

1970-01-01 00:00:00.000002016

A function is required to create a new column in each dataframe in a dictionary of dataframes. There are multiple dictionaries of dataframes in a list that all require the same treatment. The new column in each dataframe is to have the heading ‘time’ and will contain an entry in a datetime format that can be plotted on a scatter plot and may appear in a year and month format depending on scale. The number of rows in each year may be the same for all the years observed or may be different from year to year due to removal of rows that would contain unneeded data like outliers. A column exists already in each dataframe in the list of dictionaries with the heading ‘year’ and denotes only the year in which the observation was made. The years are already grouped together with the earliest years occurring in the beginning of the dataframe, i.e. the earliest years start at index=0. The function must create an entire calendar year of datetime entries in the ‘time’ column of each dataframe for each year with the first entry of a year occurring at midnight on January 1st and the last entry occurring on December 31st of the same year just before the year changes. The ‘time’ column entries should also be plottable for a scatter plot and appear on the x-axis in the year and month format depending on the scale of the plot. And the function should also add the extra day and correctly create the intervals between each row for any leap years in the data.

import pandas as pd

from dateutil.relativedelta import relativedelta

def add\_time\_column(data\_dict\_list):

for data\_dict in data\_dict\_list:

for key, dataframe in data\_dict.items():

if 'year' in dataframe.columns:

# Calculate the time interval within each year

years = dataframe['year'].unique()

intervals = {year: 1 / dataframe['year'].value\_counts()[year] for year in years}

dataframe['time'] = pd.to\_datetime(dataframe['year'], format='%Y') + dataframe.groupby('year').cumcount().map(intervals) \* relativedelta(years=1)

return data\_dict\_list

# Example usage

data\_dict\_list = [

{'df1': pd.DataFrame({'year': [2020, 2020, 2021, 2022]}),

'df2': pd.DataFrame({'year': [2022, 2022, 2023, 2023, 2023]})},

{'df3': pd.DataFrame({'year': [2021, 2021, 2022]}),

'df4': pd.DataFrame({'year': [2020, 2021, 2021, 2023]})}

]

updated\_data\_dict\_list = add\_time\_column(data\_dict\_list)

for data\_dict in updated\_data\_dict\_list:

for key, dataframe in data\_dict.items():

print(f"Dataframe: {key}")

print(dataframe)

print("\n")

Remember that the reason the dictionaries such as element\_dfs\_CS1\_1s, element\_dfs\_CS1\_0s, dict\_w\_outls\_rmvd are not “even” throughout the years over which they’re observed is because these dataframes have had outliers and 1s and 0s removed to allow for better processing of data and to avoid problems of the trends being influenced by outliers that are always the same number in the cases of 0s and 1s and in general in the other cases. In the case of element\_dfs\_CS1, I assume the entries and the observations are all the same for each year because none of the outliers or 1s or 0s has been addressed as of yet. But in the other cases it makes sense that there are extra entries in each year at the ends of the year because of some quirk that doesn’t take into account in the computation of the number of entries that each year needs to be looked at based on the number of observations that have the year beside it- AND that only a time-stamp in that year can appear on a row with that year next to it.

is there a method by which the examples returned by ChatGPT can be run directly by an instance of Jupyter Notebooks in a new tab right next to the ChatGPT tab in the browser?

How can a groupby be performed in Python pandas inside a list of dictionaries of dataframes- where the groupby is for a single dataframe inside the list, and the requirement is to find the number of occurrences of each unique entry in a single column in that one dataframe?

The dictionaries to be used to make the plots before carrying out the regression are element\_dfs\_CS1, element\_dfs\_CS1\_0s, element\_dfs\_CS1\_1s, and dict\_w\_outls\_rmvd.

element\_dfs\_CS1

element\_dfs\_CS1\_1s

element\_dfs\_CS1\_0s

dict\_w\_outls\_rmvd

# dicts\_to\_exam = [element\_dfs\_CS1, element\_dfs\_CS1\_0s, element\_dfs\_CS1\_1s, dict\_w\_outls\_rmvd]

* I don’t think what I have below works correctly. Going to try a more direct route.

A function to scatter plot the sets of data points contained in 4 separate sets of observations is required. The data represent observations of the condition of components being used in the field over a period of years. The data will be drawn from 4 dictionaries of dataframes. Each dataframe has multiple sets of data points to be plotted, and each dictionary has the same number and types of components being observed. One scatter plot should be created for each component, with the data points from each of the 4 dictionaries being plotted on one plot. Each set of data shall have a line of best fit shown in different colors on each plot, and the equation of the line shall be shown on the plot also.

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

def plot\_and\_bestfit(data\_dicts):

colors = ['b', 'g', 'r', 'c'] # Different colors for each data dictionary

for component, data\_dict in data\_dicts.items():

plt.figure(figsize=(10, 6))

plt.title(f'Scatter Plot for {component}')

plt.xlabel('Years')

plt.ylabel('Observations')

for i, (label, data) in enumerate(data\_dict.items()):

x = data['Year']

y = data['Observation']

# Scatter plot

plt.scatter(x, y, label=label, color=colors[i])

# Line of best fit

model = LinearRegression()

model.fit(x.values.reshape(-1, 1), y)

y\_pred = model.predict(x.values.reshape(-1, 1))

plt.plot(x, y\_pred, color=colors[i], linestyle='dashed')

# Equation of the line

equation = f'y = {model.coef\_[0]:.2f}x + {model.intercept\_:.2f}'

plt.text(0.1, 0.9 - i \* 0.1, equation, transform=plt.gca().transAxes)

plt.legend()

plt.show()

# Sample data dictionaries (replace with your actual data)

data\_dict1 = {

'Set A': pd.DataFrame({'Year': [2010, 2011, 2012, 2013],

'Observation': [12, 15, 18, 21]}),

'Set B': pd.DataFrame({'Year': [2010, 2011, 2012, 2013],

'Observation': [10, 13, 17, 20]})

}

data\_dict2 = {

'Set A': pd.DataFrame({'Year': [2010, 2011, 2012, 2013],

'Observation': [9, 12, 15, 18]}),

'Set B': pd.DataFrame({'Year': [2010, 2011, 2012, 2013],

'Observation': [8, 11, 14, 17]})

}

data\_dicts = {

'Component 1': data\_dict1,

'Component 2': data\_dict2

}

scatter\_plot\_with\_fit(data\_dicts)

A function to scatter plot the sets of data points contained in 4 separate sets of observations is required. The data represent observations of the condition of components being used in the field over a period of years. The data will be drawn from 4 dictionaries of dataframes. Each dataframe has multiple sets of data points to be plotted, and each dictionary has the same number and types of components being observed. One scatter plot should be created for each component, with the data points from each of the 4 dictionaries being plotted on one plot. Each set of data shall have a line of best fit shown in different colors on each plot, and the equation of the line shall be shown on the plot also.

# Data visualization prior to regression analysis.

A function to plot all the data from 4 separate dictionaries of dataframes is required. The data in each dataframe in each dictionary will come from 2 columns, one of which will be a time component. The

Need to get the plotting and data visualization prior to any regression analysis working- and don’t forget the making the plots of each of the 4 different dicts on one plot per bridge element and using that to have something on which to base your further regression computations. Get that done and I guess you can launch into regression analysis for real. (GO Schramm!)

# End year extract procedure

A function to extract out part of a filename from each entry in a column called ‘filename’ is required. The function is to be applied to a list of dictionaries, and each dictionary is a dictionary of dataframes. The filename entry in each row of each dataframe has the year during which the observation was made in the filename itself, the year is the only numeric portion of the filename and is four digits long. Once the four digit year has been extracted that year should be placed in the same row as the observation in a new column called ‘year’ and the 4 digit year should be of the correct data type to create datetime entries from them. The result should be that each dataframe in each dictionary in the list has this newly created ‘year’ column and that does not result in an attempt at setting a value on a copy of a slice from a dataframe

A function to create a new column that takes the number of entries in the dataframe and divides a year’s time evenly across all the entries is required. The function is to be applied to a list of dictionaries, and each dictionary in the list is a dictionary of dataframes. Each dataframe in each dictionary needs a new column added with the heading ‘time’ and the entry in each row must be in a time data type that will plot in Pandas and will appear along the x-axis in a month and year format.

2nd attempt: (Forgot to tell the computer that the number of years corresponds to the number of unique 4 digit entries in the ‘year’ column of the dataframe and that each year begins with the first entry of that unique 4 digit year and ends with the last entry of said 4 digit year

A function to create a new column that takes the number of entries in a year and divides the number of entries in that year evenly across the year is required. The column already present in the dataframes with the heading ‘year’ will have a 4 digit year in it, and one year will elapse between the first entry with a unique 4 digit year in it and the last entry with the same 4 digit year. The function is to be applied to a list of dictionaries, and each dictionary in the list is a dictionary of dataframes. Each dataframe in each dictionary needs a new column added with the heading ‘time’ and the entry in each row must be of a time data type that will plot in Pandas and will appear along the x-axis in a month and year format.

The year column in the existing dataframes has a 4 digit integer in each entry, and the year in which the data associated with each row is observed is represented by that 4 digit integer. The first entry in the ‘year’ column in each dataframe in each dictionary is the beginning of the year and would correspond to January 1 at midnight. Each successive observation with the same 4 digit year in that row will make up the entire year of observations and it is this group of rows all having the same 4 digit year over which one year’s time must be divided.

All the observations for a year are made, … how can the "date" of each row be computed so that the

# element\_dfs\_CS1 = {} # new\_data

# person = 'Mike'

# references

# name: person --> 44

# memory

# address: 44 --> value --> Mike

# Reference means the name of the variable, and the

# The reference means the variable, and the reference has a name (person in this case) AND the reference creates a memory address (or at least the value for the memory address) The memory address actually holds the value of the variable corresponding to the name of the reference.

# So the name given to a variable is like the shingle for an office, and the reference with the corresponding name of the variable is like your address book: There's the name of what lives at the address (could be a person like in this case- so one would say "we know a person lives at this address-"), followed by the number of the address (i.e. where the memory corresponding to the value of the variable is located).

# The memory for the name of the variable holds the value of the variable, the value at that memory location represents the specifics of the variable at the (i.e. a name of a person in this case, someone named Mike) and the address created in reference represents Mike's address, so specifically Mike has an address the number of which is created in reference when the variable is created.

# The object in memory is the actual location of the address that holds the value of a variable, and person is a reference.

# Python has names that refer to objects. Objects exist separately from names and names exist separately from the objects to which they refer.

def getyr\_fr\_filename(dict\_list):

new\_dictionaries = []

for dictionary in dict\_list:

new\_dictionary = {}

for key, dataframe in dictionary.items():

new\_dataframe = dataframe.copy()

# get year from filename

new\_dataframe['year'] = new\_dataframe['filename'].str.extract(r'(\d{4})')

# make four digit year of type datetime

new\_dataframe['year'] = pd.to\_datetime(new\_dataframe['year'], format='%Y')

new\_dictionary[key] = new\_dataframe

new\_dictionaries.append(new\_dictionary)

return new\_dictionaries

# List of dictionaries you want to examine

dicts\_to\_exam = [element\_dfs\_CS1, element\_dfs\_CS1\_0s, element\_dfs\_CS1\_1s, dict\_w\_outls\_rmvd]

# Apply the function to extract year from filename

dicts\_w\_yr\_added = getyr\_fr\_filename(dicts\_to\_exam)

import pandas as pd

import re

def extract\_digits\_from\_filename(dict\_list):

new\_dict\_list = []

for dictionary in dict\_list:

new\_dict = {}

for key, dataframe in dictionary.items():

new\_dataframe = dataframe.copy() # Create a new DataFrame

# Apply a function to create the 'year' column

new\_dataframe['year'] = new\_dataframe.apply(lambda row: process\_row(row), axis=1)

new\_dict[key] = new\_dataframe

new\_dict\_list.append(new\_dict)

return new\_dict\_list

def process\_row(row):

# Extract digits from filename using regex

digit\_str = re.search(r'\d{4}', row['filename'])

# Convert digits to int data type

if digit\_str:

return int(digit\_str.group())

else:

return None

# List of dictionaries you want to examine

dicts\_to\_exam = [element\_dfs\_CS1, element\_dfs\_CS1\_0s, element\_dfs\_CS1\_1s, dict\_w\_outls\_rmvd]

# Apply the function to extract digits and create 'year' column

dicts\_with\_digits\_added = extract\_digits\_from\_filename(dicts\_to\_exam)