**Useful commands in Python**

1. Printing

print('Wage w: %5.2f.' % w\_ss)

print(f’Wage w: {w}’)

print(f'Representing: {wtd\_tot[year] \* 1e-5:,.2f} Lakh taxpayers')

1. Format round decimal places and print

citax\_collection\_billions1 = citax\_collection1/10\*\*9

citax\_collection\_str1 = '{0:.2f}'.format(citax\_collection\_billions1)

total\_revenue\_text1 = "TAX "+str(citax\_collection\_str1)+" bill Zlotys"

print(total\_revenue\_text1)

my\_num = 235478.875425

s = "{:,.2f}".format(my\_num)

print(s)

a\_number = 1 / 3

percentage = "{:.2%}".format(a\_number)

**print**(percentage)

1. Numpy one dimensional matrix

nvec = np.array([1.0, 1.0, 0.2])

two dimensional matrix

bmat = np.zeros((10, 20))

1. Create and Append items to array of float

# Create a NumPy array of floats

sample\_means\_arr = np.array([], dtype=np.float64)

sample\_mean = np.array([df\_sample['Value'].mean()])

# Append a single float value

sample\_means\_arr = np.append(sample\_means\_arr, sample\_mean)

1. Numpy All rows except last

bmat[:-1, :]

1. Numpy last element

A[-1]

Second last element A[-2]

1. Numpy, rowwise summing across columns - hotizontally

np.sum(bmat, axis=1)

bmat.sum(axis=1)

summing across rows, axis = 0

1. Check if two numbers (or two arrays) are close together

np.isclose(a,b)

numpy.isclose(a, b, rtol=1e-05, atol=1e-08, equal\_nan=False)

rtol is relative tolerance and a is the absolute tolerance

checks if absolute(*a* - *b*) <= (*atol* + *rtol* \* absolute(*b*))

1. Reshape

etr\_vec = etr\_vec.reshape(etr\_vec.shape[0], 1)

1. Wide to long

Check <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.wide_to_long.html>

df

famid birth ht1 ht2

0 1 1 2.8 3.4

1 1 2 2.9 3.8

2 1 3 2.2 2.9

3 2 1 2.0 3.2

4 2 2 1.8 2.8

5 2 3 1.9 2.4

6 3 1 2.2 3.3

7 3 2 2.3 3.4

8 3 3 2.1 2.9

l = pd.wide\_to\_long(df, stubnames='ht', i=['famid', 'birth'], j='age')

l

ht

famid birth age

1 1 1 2.8

2 3.4

2 1 2.9

2 3.8

3 1 2.2

2 2.9

2 1 1 2.0

2 3.2

2 1 1.8

2 2.8

3 1 1.9

2 2.4

3 1 1 2.2

2 3.3

2 1 2.3

2 3.4

3 1 2.1

2 2.9

Note = Stubnames (prefixes) could also be a list – if fieldnames are

A1970 A1971 B1970 B1971 X id

pd.wide\_to\_long(df, ["A","B"], i="id", j="year")

df

A1970 A1980 B1970 B1980 X id

0 a d 2.5 3.2 -1.085631 0

1 b e 1.2 1.3 0.997345 1

2 c f 0.7 0.1 0.282978 2

**>>>** pd.wide\_to\_long(df, ["A", "B"], i="id", j="year")

**...**

X A B

id year

0 1970 -1.085631 a 2.5

1 1970 0.997345 b 1.2

2 1970 0.282978 c 0.7

0 1980 -1.085631 d 3.2

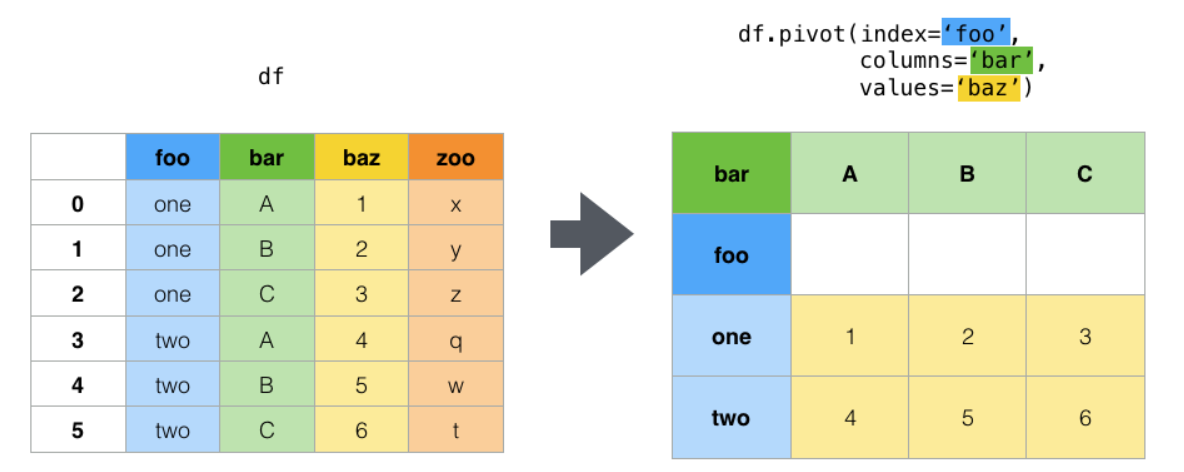
1 1980 0.997345 e 1.3

2 1980 0.282978 f 0.1

\*\*\* After this do

df = df.reset\_index()

1. Reshape a dataset - pivot



1. Reshape a dataset – stack. This reshapes from a wide to long format. In case there is one unique column it is easier. First make that column an index and then run the command.

This file has Country Code and date for various years

Country\_Code 2005 2006 2007, etc. which we want to change to

Country\_Code year

Country\_Code1 2005

Country\_Code1 2006

Country\_Code1 2007

Country\_Code2 2005

Country\_Code2 2006

Country\_Code2 2007

1. First make country code an index

cit\_rates\_df = cit\_rates\_df.set\_index('Country\_Code')

1. Run the stack command

cit\_rates\_df = cit\_rates\_df.stack()

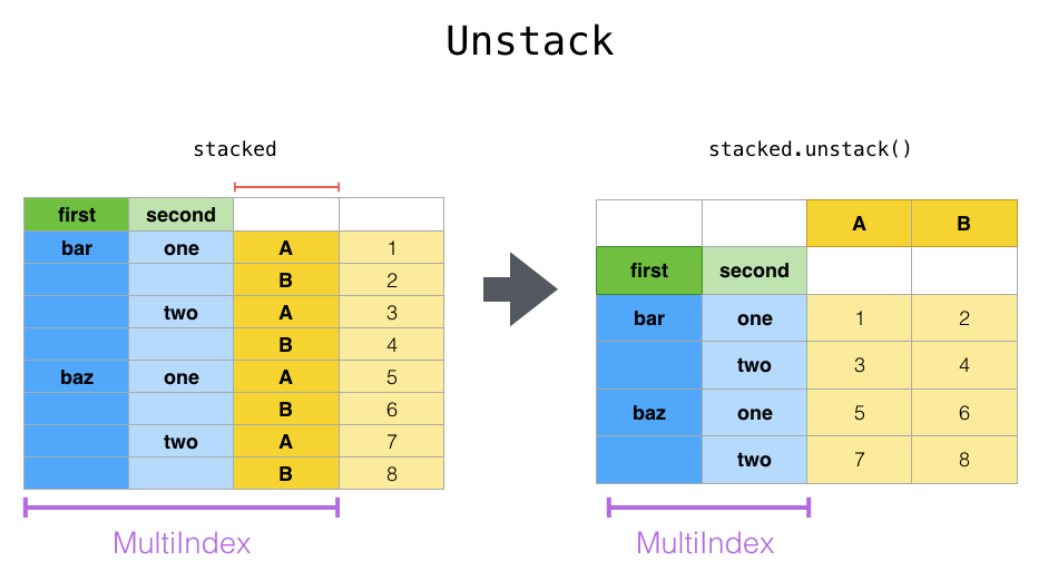
1. Reset the index to make Country\_code a column and then rename the fields.

cit\_rates\_df = cit\_rates\_df.reset\_index()

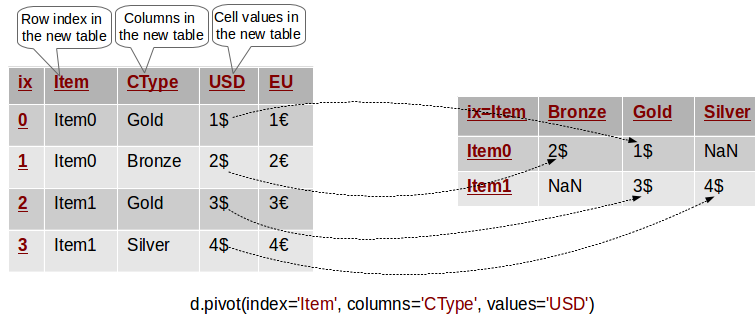
1. Reshape – Unstack

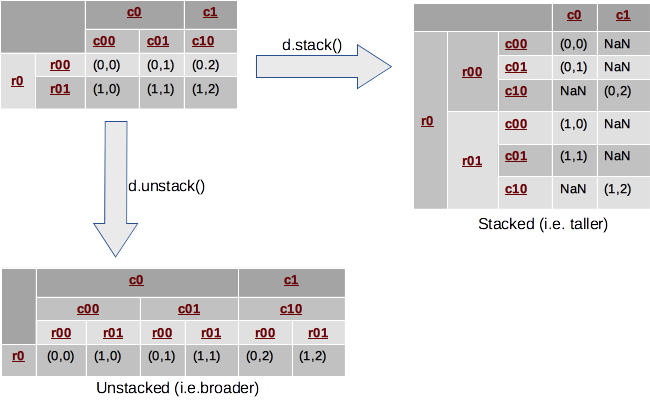
df.unstack()

df.pivot(index=’item’, columns=’CType’, values=’USD’)



df.pivot(index=’item’, columns=’CType’, values=’USD’)





1. Matrix multiplication

A = np.array([[1, 2, 3]])

B = np.array([[4],[5],[6]])

B.dot(A) =

array([[ 4, 8, 12],

[ 5, 10, 15],

[ 6, 12, 18]])

1. Inserting a character in a string

S is a string, this command inserts a ‘,’ after the third character of S

S = S[:3] + ',' + S[3:]

1. str.find(sub,start,end)

Finds the lowest index of the substring if it is found in given string. If not found, then it returns -1. Start – start position in string to start search, end is end position to check.

1. Replace a substring with another

string.replace(old, new, count)

count is the number of times the replacement should be done for multiple occurrences. If this is not included replaces all occurrences.

1. Capitalize words in a string

txt = "hello world."  
x = txt.capitalize()

* “Hello world”

Capitalize every word

txt = “hello world.”

import string

string.capwords(txt)

* “Hello World.”

1. Creating a new directory

import os

#create temp directory to store any intermediate outputs

if not os.path.exists('temp'):

os.makedirs('temp')

to save into the temp directory

df\_rates\_updated.to\_csv("temp\\df\_rates\_updated.csv", index=False)

1. List

Extract values above a certain number from a list

[i for i in list\_a if i>100]

1. Create a dataframe and fill it.

df = pd.DataFrame(columns = ['Name', 'Articles', 'Improved'])

# append rows to an empty DataFrame

df = df.append({'Name' : 'Ankit', 'Articles' : 97, 'Improved' : 2200},

                ignore\_index = True)

df = df.append({'Name' : 'Aishwary', 'Articles' : 30, 'Improved' : 50},

                ignore\_index = True)

df = df.append({'Name' : 'yash', 'Articles' : 17, 'Improved' : 220},

               ignore\_index = True)

Create an empty dataframe with nrows

df= pd.DataFrame(index=range(nRows))

df = pd.DataFrame(index=range(nRows),columns=range(nCols))

method #2

# create an Empty DataFrame object

df = pd.DataFrame()

print(df)

# append columns to an empty DataFrame

df['Name'] = ['Ankit', 'Ankita', 'Yashvardhan']

df['Articles'] = [97, 600, 200]

df['Improved'] = [2200, 75, 100]

method #3

# create an Empty DataFrame object With

# column names and indices

df = pd.DataFrame(columns = ['Name', 'Articles', 'Improved'],

                   index = ['a', 'b', 'c'])

print("Empty DataFrame With NaN values : \n\n", df)

# adding rows to an empty

# dataframe at existing index

df.loc['a'] = ['Ankita', 50, 100]

df.loc['b'] = ['Ankit', 60, 120]

df.loc['c'] = ['Harsh', 30, 60]

method #4 (fastest)

#create a dictionary list and append

row\_list = []

data\_dict = {"State":state, "Year": np.int64(year+1), "Population": new\_pop}

row\_list.append(data\_dict)

df = pd.DataFrame(row\_list)

1. Append rows based on the value in a column.

In this example the rows in the sample dataframe is expanded by the weight field to represent one row for each element of the population.

df\_expanded = df.loc[df.index.repeat(df[weight])].reset\_index(drop=True)

1. Fill a column with a single value

df['A']= 'foo'

1. Create a dataframe from a csv

Makes the first column the index

df = pd.read\_csv('sample.csv', index\_col=0)

1. Make index as row number

df = df.reset\_index(drop=True)

1. Find row with the maximum value of a column ‘A’

df.loc[df['A'].idxmax()]

1. Convert dataframe to dictionary with rows as elements

var\_dict = df\_var\_file.to\_dict(orient='index')

You can also use orient = ‘records’ but the row numbers are not captured

1. Creating a dataframe from Excel

ind\_desc\_df = pd.read\_excel(file\_name, 'Sheet1', index\_col=False)

df = pd.read\_excel(filename,sheet\_name="Results", index\_col=None, header=0, skiprows=[0,1,2])

header=0 : this will take the column names from the first row of the excel file

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html>

skip rows

1. Save a dataframe into csv file

df\_rates\_updated.to\_csv("df\_rates\_updated.csv", index=False)

1. Display options for rows and columns of a dataframe and format

pd.set\_option('display.max\_rows', 50)

pd.set\_option('display.max\_columns', 30)

pd.set\_option('display.expand\_frame\_repr', False)

pd.set\_option('max\_colwidth', -1)

pd.set\_option('max\_colwidth', 800)

pd.set\_option('display.precision', 2)

pd.set\_option('display.float\_format', '{:.2f}'.format)

with commas for thousands

pd.set\_option('display.float\_format', '{:,.2f}'.format)

RESETTING

pd.reset\_option(‘all’)

1. Summarize the dataframe

df.groupby(['Parameter\_Code','Parameter', 'Units']).size().reset\_index().rename(columns={0:'count'})

df.describe()

df.info()

df[‘field\_name’].value\_counts()

df[‘field\_name’].value\_counts(normalize = True)

1. Find centiles of a dataframe column

df['A'].quantile([.1, .2, .3, .4, .5, .6, .7, .8, .9])

1. Adding a new column to a dataframe

# Define a dictionary containing Students data

data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],

        'Height': [5.1, 6.2, 5.1, 5.2],

        'Qualification': ['Msc', 'MA', 'Msc', 'Msc']}

# Convert the dictionary into DataFrame

df = pd.DataFrame(data)

# Declare a list that is to be converted into a column

address = ['Delhi', 'Bangalore', 'Chennai', 'Patna']

# Using 'Address' as the column name

# and equating it to the list

df['Address'] = address

1. Adding a new column to a dataframe and filling it with the same value:

D = {'newcol1': 'a',

'newcol2': 2,

'newcol3': 1}

df = pd.DataFrame({'A':[1,2],

'B':[4,5],

'C':[7,8]})

print (df)

A B C

0 1 4 7

1 2 5 8

print (pd.concat([df, pd.DataFrame(D, index=df.index)], axis=1))

A B C newcol1 newcol2 newcol3

0 1 4 7 a 2 1

1 2 5 8 a 2 1

Or use

A.assign(\*\*d)

A = pd.DataFrame(np.random.rand(10, 5), columns=list('abcde'))

d = {

'newcol1': 'a',

'newcol2': 2,

'newcol3': 1

}

A.assign(\*\*d)

a b c d e newcol1 newcol2 newcol3

0 0.709249 0.275538 0.135320 0.939448 0.549480 a 2 1

1 0.396744 0.513155 0.063207 0.198566 0.487991 a 2 1

2 0.230201 0.787672 0.520359 0.165768 0.616619 a 2 1

3 0.300799 0.554233 0.838353 0.637597 0.031772 a 2 1

4 0.003613 0.387557 0.913648 0.997261 0.862380 a 2 1

5 0.504135 0.847019 0.645900 0.312022 0.715668 a 2 1

6 0.857009 0.313477 0.030833 0.952409 0.875613 a 2 1

7 0.488076 0.732990 0.648718 0.389069 0.301857 a 2 1

8 0.187888 0.177057 0.813054 0.700724 0.653442 a 2 1

1. 0.003675 0.082438 0.706903 0.386046 0.973804 a 2 1
2. Data type of columns of a dataframe

df.dtypes

or df.info()

1. df.describe()

desc = df["Name"].describe()

1. Display Unique Values in a dataframe

df\_cons\_summ['SIZE\_GROUP'].unique()

number of unique rows:

len(df['Country\_Code'].unique())

1. Drop rows

df[df.name != 'Tina']

where ‘name’ is the column

1. Access the 5th row of the dataframe

df.iloc[4,:]

iloc accesses the values of the datafarame like a matrix

df.iloc[row number, column number]

so 3rd row and third column

df.iloc[2,2]

1. Drop last row of dataframe

df = df.iloc[:-1 , :]

df.plot(kind='bar',y=['citax\_2020','citax\_ref\_2020'])

1. df.loc uses retrieval by index value and columns by labels

retrieve index location 2 and column ‘name’

df.loc[2,’name’]

1. Replace last row element with last but one row

df.loc[df.index[-1], 'bin'] = df.loc[df.index[-2], 'bin']

1. Number duplicates sequentially in Pandas DataFrame

df['dup\_number'] = df.groupby(['doc\_reg\_no']).cumcount()+1

1. Split into two columns using a delimiter in each row

df[['V','allele']] = df['V'].str.split('-',expand=True)

1. Changing Order of Columns

df = df[['mean', '0', '1', '2', '3']]

where the new order is in the brackets

1. Dropping columns

df.drop('column\_name', axis=1, inplace=True)

Drop multiple columns

df.drop(['C', 'D'], axis = 1)

To drop a column and not give error if the column name is not there

df2 = df2.drop(['\_merge'], axis=1, errors='ignore')

To drop by column number instead of by column label, try this to delete, e.g. the 1st, 2nd and 4th columns:

df = df.drop(df.columns[[0, 1, 3]], axis=1)

1. Drop Nan dataframe

Drop Nans in columns ‘name’ and ‘toy’

df.dropna(subset=['name', 'toy'])

df.dropna()

drop all rows with any Nans

Drop the rows where all elements are missing.

df.dropna(how='all')

1. Take average of columns correctly addressing nan entries

sum\_a = np.zeros(len(df))

sum\_b = np.zeros(len(df))

for i in range(num\_years):

sum\_a = sum\_a + np.where(df[[a+'\_m\_'+str(year-i),b+'\_m\_'+str(year-i)]].notna().all(1),df[a+'\_m\_'+str(year-i)],0)

sum\_b = sum\_b + np.where(df[[a+'\_m\_'+str(year-i),b+'\_m\_'+str(year- i)]].notna().all(1),df[b+'\_m\_'+str(year-i)],0)

df[a+'\_'+str(num\_years)+'\_year\_avg\_'+str(year)] = sum\_a

df[b+'\_'+str(num\_years)+'\_year\_avg\_'+str(year)] = sum\_b

df[c+'\_'+str(num\_years)+'\_year\_avg\_'+str(year)] = (sum\_a/sum\_b)

df\_cit\_below\_15['cit\_etr\_rate'] = np.where(df\_cit\_below\_15[['Mean cit\_rate','Mean ETR3']].notna().all(1),df\_cit\_below\_15['Mean cit\_rate'],df\_cit\_below\_15['Mean ETR3'])

1. Drop columns starting with or containing a string

unwanted\_cols = df.columns[df.columns.str.contains('toggle')]

if multiple strings:

f\_recs[f\_recs['Behavior'].str.contains("nt|nv", na=False)]

for strartswith replace contains with startswith

df.drop(unwanted\_cols, axis=1, inplace=True)

1. Columns starting with multiple options

filter\_col = [col for col in df if col.startswith(('ETR', 'Income\_Tax', 'EBT'))]

#filter\_col = [col for col in df if col.startswith('Country')]

filter\_col = ['Company\_name\_spl', 'filename', 'Country ISO code', 'BvD sectors', 'NACE', 'Turnover\_m\_2019']+filter\_col

ETR\_df = df[filter\_col]

1. Drop missing rows (NaN) and plot

df = df.dropna(subset=['ShiftedPrice'])

df.plot(x='date', y='ShiftedPrice')

1. Subract rows of dataframe using shift

df1['diff\_log\_X'] = df1['log\_X'] - df1.shift(-150)['log\_X']

1. Finding duplicate rows

duplicateRowsDF = df[df.duplicated(['Name'])]

for multiple field match –

df[df.duplicated(['Country', 'Tax\_Type'])]

1. Drop duplicate rows

data.drop\_duplicates(subset ="First Name",

                     keep = False, inplace = True)

To retain the first of the duplicate rows use keep = ‘first’

For multiple column duplicate,

df = df.drop\_duplicates(subset=['Company name', 'Turnover'], keep='first')

check if entire row is duplicate and drop:

data.drop\_duplicates(keep=False, inplace=True)

1. Useful Print Command to print both text and formatted numbers

Year and total\_consumption\_all1 are variables

print(f'Total Consumption in - {year}: {total\_consumption\_all1:,.0f}')

1. Concatenate and Appending Dictionaries and creating json files

Dataframe to dictionary

# dict1 = df1.to\_dict()

# dict2 = df2.to\_dict()

# Concatenate two dictionaries

# dict1.update(dict2)

# dictf = {"read": dict1, "calc": dict2}

#converting the dictionary to dataframe

# df\_gst\_for\_json = pd.DataFrame(dict\_gst\_rec)

# df = pd.DataFrame(dictf)

# saving dataframe to json file

# df\_gst\_for\_json.to\_json("gst\_rec.json")

# df.to\_json("gst\_records.json")

# Merging the two dictionary along with adding "read" and "calc"

dict\_gst\_rec = {"read": dict\_gst\_read, "calc": dict\_gst\_calc}

# Pretty Print dictionary into json file

with open("gstrecords\_variables.json", "w") as f:

json.dump(dict\_gst\_rec, f, indent=4, sort\_keys=True)

1. Read json from file

import json

with open('taxcalc/current\_law\_policy\_poland.json') as f:

current\_law\_policy = json.load(f)

1. Sort by a column

df.sort\_values(by=['col1'])

1. df.sort\_values(by=['col1', 'col2']), combination of ascending and descending

df.sort\_values(['a', 'b'], ascending=[True, False])

1. Selecting rows based on multiple criteria

revenue\_df[(revenue\_df['Region\_Code']=="LAC") & (revenue\_df['year']==2016)][['Country', 'GDP\_PC']]

1. Selecting columns that start with certain strings

df[df.columns[df.columns.str.startswith(('h','p'))]]

1. Changing name of a column programmatically

df.columns = df.columns.str.replace('CONS\_','GST\_')

1. Cumulative sum

df['Cumulative Sales'] = df['Sales'].cumsum()

1. Adding up columns programmatically

In this example, we add up all the columns with name starting with ‘CONS\_’ and allocates it to ‘CONS\_OTHER’

df['CONS\_OTHER'] = df[df.columns[df.columns.str.startswith('CONS\_')]].sum(axis=1)

1. Creating blank rows in a dataframe

df = pd.concat([df, pd.DataFrame([[np.nan] \* df.shape[1]], columns=df.columns)], ignore\_index=True)

1. Histograms

import seaborn as sns

df\_survey\_mpc = df\_cons\_summ\_all[df\_cons\_summ\_all["Srl\_no"] == 49]

df\_survey\_mpc.index = pd.RangeIndex(len(df\_survey\_mpc.index))

df\_survey\_mpc = df\_survey\_mpc.drop(df\_survey\_mpc.columns[0], axis = 1)

sns.distplot(tuple(df\_survey\_mpc['Value']), hist = False, kde = True, kde\_kws = {'linewidth': 3}, label = "consumption")

1. Histogram

df.hist()

Kernal density

df.plot.kde()

Example

df1 = df[(df['mortgage']==0)][['ln\_total\_income1']]

df1.reset\_index()

df1 = df2.drop('index', axis=1)

df2 = df[(df['mortgage']==1)][['ln\_total\_income1']]

df2.reset\_index()

df2 = df2.drop('index', axis=1)

df1 = df1.rename(columns={'ln\_total\_income1':'No'})

df2 = df2.rename(columns={'ln\_total\_income1':'Yes'})

(unequal columns so we need to reset\_index before concating)

df3 = pd.concat([df1, df2], axis=1)

df3.plot.kde()

import matplotlib.pyplot as plt

#create histogram for points variable

plt.hist(df['points'], edgecolor='black')

1. Extract certain Columns (and otherwise) from a dataframe

df1 = pd.DataFrame(np.array([[1, 0, 3], [4, 5, 6]]))

df1.columns=['A', 'B', 'C']

change\_cols = ['A', 'C']

df1 = df1[change\_cols]

This will give a ‘SettingWithCopyWarning’, so use deepcopy instead

df1 = df[cols].copy(deep=True)

for all columns except ['A', 'C']

df1[df1.columns.difference(change\_cols)]

Again, this will give a ‘SettingWithCopyWarning’, so use deepcopy instead

df2 = df[df1.columns.difference(change\_cols)].copy(deep=True)

1. Join one dataframe to another (concat)

df = pd.concat([df2, df1], axis=1)

axis=1 cocats columns side by side

axis=0 concats lengthwise (appends after the last row)

1. Convert a dataframe row to a list,

Here row ‘i’ of the dataframe df is converted into a list and assigned to ‘row’

row = df.loc[i].values.tolist()

1. Creating lists, appending and inserting elements

row = []

row.append('A')

['A']

say row=['A', 'B', 'C']

row.insert(1,’A1’)

['A', 'A1', 'B', 'C']

1. Remove an item from a list

row=['A', 'B', 'C']

row.pop(1)

'B'

row

['A', 'C']

Also

row.remove('B ')

this gives error if the element is not in the list.

Then use this

df2\_columns = [x for x in df2\_columns if x!="\_merge"]

1. Inserting a list into a dataframe at a particular row position (replacing the row)

This command inserts a list ‘row’ at row ‘i’ of the dataframe df

df.loc[i] = row

1. Merging dataframes

df\_rates = pd.merge(df\_rates, df\_product, how="inner", on="product\_id")

This also works for merging on multiple columns

tax\_rates\_df=pd.merge(cit\_rates\_df, pit\_rates\_df, how="outer", on=['Country\_Code', 'year'], indicator=True)

how = “outer” : Merge keeping even the unmatched entries use

indicator=True : Includes a \_merge indicator

1. Merge on index

df\_cit\_etr\_below\_15 = df\_cit\_rates\_15.merge(df\_table\_etr\_country\_below\_15, how='outer', left\_index=True, right\_index=True)

1. Appending or concatenating dataframes

frames = [df1, df2, df3]

concat/attach files one below the other

df = pd.concat(frames)

you may also use axis=0 for rows axis=1 for columns

Concat side by side

result = pd.concat([df1, df4], axis=1, sort=**False**)

1. Rename dataframe columns

pit\_df = pit\_df.rename(columns={Old\_Name:New\_Name, 'pitax':'pitax\_1'})

1. Rename index

df1.rename(index={1: 'a'})

1. Rename first row index name

df1 = df3.rename(index={df3.index[0]: 'OVERALL SUM’})

1. Rename dataframe columns simple

df.columns=[‘A’, ‘B’]

1. Rename dataframe columns complex – replace the name of multiple columns

df.columns=df.columns.str.replace('neg\_govt\_expenditure','govt\_expenditure')

1. Transpose a dataframe

df = df.T

1. Extract last row of dataframe

df.iloc[[-1]]

1. Dictionary to dataframe and then transpose

inc\_block\_df = pd.DataFrame.from\_dict(inc\_block, orient='columns').T

1. Change type of the Dataframe column three ways

df['DataFrame Column'] = df['DataFrame Column'].apply(str)

df['DataFrame Column'] = df['DataFrame Column'].astype(str)

df = df.applymap(str)

1. Replace one column with a manipulation of another column

inc\_block\_df['year']=inc\_block\_df['year'].str[:4]

strip or trim the leading and following spaces

fdi\_df['Country'] = fdi\_df['Country'].str.strip()

1. Find Replace rows with condition – Complex

fdi\_df['Region\_Code\_1'] = np.where(fdi\_df['Country\_Code']=="BRA", "BRA", fdi\_df['Region\_Code'])

1. Find Replace rows with condition – Multiple conditions

– use ‘&’ not ‘and’

df['A'] = np.where((df['A']=='Harry') & (df['B']=='George') & (df['C']>'2019'), 'Matt', df['A'])

1. Replace inf with nan

df['profit\_margin'].replace(np.inf, np.nan)

df.replace(np.inf, np.nan)

1. Keep rows that are not inf

df1 = df[~df['profit\_margin'].isin([np.inf])]

1. Find Lags and lag difference

Single group

Shift data down by 1 period

df[‘Tax\_Revenue\_-1’] = df[‘Tax\_Revenue’].shift(1)

Lag\_difference

df[‘Tax\_Revenue\_Lag\_diff’] = df[‘Tax\_Revenue’] - df[‘Tax\_Revenue’].shift(1)

Multiple groups

df['lagged\_values'] = df.groupby(['group'])['values'].shift(1)

1. Find Replace rows with condition – Simple

fdi\_df['Region\_Code\_1'] = fdi\_df['Region\_Code\_1'].replace(['EAP'],'BRA')

1. Find replace text substring in a column

df['range'] = df['range'].str.replace(',','-')

1. Find Replace a string with Nan or another string – Simple

df = df.replace('.', np.nan)

1. Replace NaNs in a dataframe with 0

df[‘col’] = df[‘col].fillna(0)

1. Filter or Select rows of a dataframe

dumpdf\_1[['CIT\_ID\_NO', 'sector', 'citax']][dumpdf\_1['citax']>0]

1. Select a random number of rows from a dataframe

df1 = df.sample(n=2000)

1. Random Sample from a sample with weights

df.sample(n=10, weights=df['weights'], replace=True)

1. Random Coin Flips (n=1 means one coin)

n, p = 1, .33 # n = coins flipped, p = prob of success

flips = np.random.binomial(n, p, 100) # repeated 100 times

#Sample an array multiple times and plot the mean

sample\_size=10

for i in range(10000):

sample\_result = np.random.choice(flips, size=sample\_size)

sample\_mean = np.array(sample\_result.mean())

# Append a single float value

sample\_means\_arr = np.append(sample\_means\_arr, sample\_mean)

plt.hist(sample\_means\_arr, label='Sampling Distribution');

plt.title('n = '+str(sample\_size))

#plt.legend()

plt.show()

1. Select Columns in a dataframe by prefix starting with

Option -1

df.loc[:,df.columns.str.startswith("foo")]

Option-2 Complex possible

filter\_col = [col for col in df if col.startswith('foo')]

df[filter\_col]

For startswith multiple strings use tuple (( ‘foo’, ‘bar’))

filter\_col = [col for col in df if col.startswith(('ETR', 'Income\_Tax'))]

filter\_col = ['Company name', 'Company\_name\_spl\_x', 'filename', 'Country ISO code\_x', 'BvD sectors', 'NACE\_x', 'Turnover\_m\_2019', 'Turnover1\_th\_2008']+filter\_col

ETR\_df = df[filter\_col]

1. Convert the index into a field

df[‘index1’] = df.index

also

df.reset\_index(level=0, inplace=True)

1. Split columns of a dataframe using a delimiter

df[['District', 'LAC', 'Local Body', 'Ward']] = df['Place'].str.split(',', expand=True)

1. Using Multi index

df\_gst\_for\_json\_read = df\_cons\_summ.drop(df\_cons\_summ.index)

df\_gst\_for\_json\_read.columns = pd.MultiIndex.from\_arrays([cols\_second\_level, df\_gst\_for\_json\_read.columns])

idx = pd.IndexSlice

df\_gst\_for\_json\_read.loc[0, idx[:, df\_gst\_for\_json\_read.columns.levels[1][df\_gst\_for\_json\_read.columns.levels[1].str.startswith('CONS\_')]]]="float"

df\_gst\_for\_json\_read.loc[1, :] = [e[0] for e in df\_gst\_for\_json\_read.columns.values]

Column values are given by df\_gst\_for\_json\_read.columns.values

This is an array of tuples with the column index of the two levels of indexes

1. Remember in lists

row=['A', 'B', 'C']

row[0] = ‘A’ , a string

while row[0:1] = [‘A’] a list

1. IMPORTANT

Copying a list is not straightforward

list\_tdict\_orig = list\_tdict[:]

(the command list\_tdict\_orig = list\_tdict is NOT CORRECT)

1. One line for loop manipulation of list

[i\*\*2 for i in range(3)]

tax\_expenditure\_desc\_english = [string.capwords(current\_law\_policy[i]['description']) for i in tax\_expenditure\_var\_list]

1. Working with Dictionaries

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

thisdict.items

* dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

thisdict.keys()

* dict\_keys(['brand', 'model', 'year'])

thisdict.values()

* dict\_values(['Ford', 'Mustang', 1964])

thisdict["model"] = ‘Mustang’

Change Value

* thisdict["year"] = 2018

for x in thisdict.values():  
  print(x)

'Ford'

'Mustang'

1964]

Looping through keys and values

for x, y in thisdict.items():  
  print(x, y)

To determine if a specified key is present in a dictionary use the

if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

length of a dictionary

print(len(thisdict))

Adding an item to the dictionary is done by using a new index key

thisdict["color"] = "red"  
print(thisdict)

Removing items

thisdict.pop("model")  
print(thisdict)

Removes the last inserted item

thisdict.popitem()  
print(thisdict)

Emptying a dictionary

thisdict.clear()

Constructing a dictionary

thisdict = dict(brand="Ford", model="Mustang", year=1964)

Makes a dictionary from a list

dict(list\_tdict)

Make a dictionary into a list

list(tdict)

1. Filling in a dictionary (example)

wtd\_tax\_clp={}

wtd\_tax\_clp[year] = calc1.weighted\_total('pitax')

Access the elements using wtd\_tax\_clp[year]

1. Transposing a dictionary

from collections import defaultdict

ref=defaultdict(dict)

for j in range(ref\_num):

i=0

for pkey, sdict in a.items():

print(f'pkey: {pkey}')

print(f'sdict: {sdict}')

print(sdict[j])

ref[j][year+i] = sdict[j] (in this example we transpose the dictionary)

i=i+1

1. Accessing a dictionary

d = dict((k, v)

for k, v in reform[i].items() if k >= BASE\_YEAR)

In this example we only select items that are greater than the BASE\_YEAR

1. Rename keys in a dictionary

mydict = {k[1:]:v for k,v in mydict.items()}

{"two" if k == 2 else k:v for k,v in d.items()}

if newkey!=oldkey:

dictionary[newkey] = dictionary[oldkey]

del dictionary[oldkey]

1. Dictionary to csv

with open('test.csv', 'w') as f:

    for key in my\_dict.keys():

        f.write("%s,%s\n"%(key,my\_dict[key]))

import csv

import numpy as np

with open('file.csv', 'w', newline='') as outfile:

w = csv.writer(outfile)

w.writerow(output\_dict.keys()) # first row of the csv are the keys

transposed\_values = (np.array(list(output\_dict.values())).T).tolist() # transpose the data

for i in range(len(transposed\_values)):

w.writerow(transposed\_values[i]) # write them row by row

1. Summing groups in pandas

Aggregate or collapse

You can select the columns of a groupby:

In [11]: df.groupby(['Country', 'Item\_Code'])[["Y1961", "Y1962", "Y1963"]].sum()

Out[11]:

Y1961 Y1962 Y1963

Country Item\_Code

Afghanistan 15 10 20 30

25 10 20 30

Angola 15 30 40 50

25 30 40 50

Also,

df.groupby(['Country', 'Item\_Code']).agg({'Y1961': np.sum, 'Y1962': [np.sum, np.mean]}) # Added example for two output columns from a single input column

The category grouped by would be pushed into the index. You would need to do a df.reset\_index() to make it a column.

1. Get Attributes of a Class or any Object

print(dir(a))

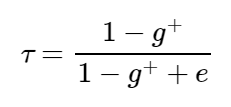
print(vars(a))

**HMTL**

1. Printing special characters from Word imported into html

To center use \[

\[ τ = \frac{1 - g^+}{1 - g^+ +e}\]

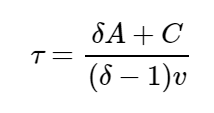


This is aligned with the text – we use \(

\( τ = \frac{1 - g^+}{1 - g^+ +e}\)

\(a^2, e^x\)

\[τ = \frac{δA+C}{(δ-1) v}\] in text gives (centered)



in the webpage.

1. **File Writing and Reading**

# Writing to a File

f= open("guru99.txt","w+")

f.write("This is line %d\r\n" % (i+1))

#Appending to a file

f=open("guru99.txt", "a+")

f.close()

# Reading a File

With open ("microsimulation\_instructions.txt"), ‘r’) as f:

contents = f.read()

print(contents)

#line by line

with open('test.txt','r') as f:

lines = f.readlines()

for line in lines:

print(line.rstrip('\n'))

filepath = 'Iliad.txt’

with open(filepath) as fp:

line = fp.readline()

cnt = 1

while line:

print("Line {}: {}".format(cnt, line.strip()))

line = fp.readline()

cnt += 1

1. Reading a Zip Folder

import zipfile

z = zipfile.ZipFile('member\_income\_20230630\_MS\_csv.zip')

filename = z.namelist()[0]

df = pd.read\_csv(z.open(filename))

1. Regression in Python using statsmodel

import statsmodels.api as sm

import statsmodels.formula.api as smf

<https://faculty.washington.edu/otoomet/machinelearning-py/linear-regression.html>

**basic method**

Y\_vars = df['log\_PIT\_Current\_LCU'] ## Y\_vars usually means our output/dependent variable

X\_vars = df[['log\_GDP\_Current\_LCU', 'PIT Rate', 'PIT\_Zero\_Threshhold', 'PIT\_Maximum\_Rate\_Threshold']]

X = sm.add\_constant(X\_vars) ## let's add an intercept (beta\_0) to our model

Y = Y\_vars

model = sm.OLS(Y, X).fit()

model.summary()

using the varible names of the dataframe

model **=** smf.ols("medv ~ rm + lstat + chas", data**=**df).fit()

**using dummies**

without explicitly mentioning the dummies if variable is not a number

m = smf.ols("survived ~ C(sex)", data=titanic).fit()

it variable is a number, then can use C() to make it into a dummy

m = smf.ols("survived ~ C(sex)", data=titanic).fit()

**interactions**

m **=** smf.ols("survived ~ C(pclass)\*sex", data**=**titanic).fit()

**exponent**

model **=** smf.ols("medv ~ rm + lstat + np.square(lstat)", data**=**df).fit()

1. **Regression plus scatter plot linefit using a dataframe**

**Also Use the feature in the plot\_chart library**

**Simple**

import statsmodels.api as sm

from matplotlib import pyplot as plt

file\_name = "C:/Users/wb305167/OneDrive - WBG/Sri Lanka/Sri Lanka Tax Rates Thresholds and Buoyancy.xlsx"

df1 = pd.read\_excel(file\_name, 'Data', index\_col=False)

df = df1[(df1.year>=2003)&(df1.year<2022)]

df['log\_GDP\_Current\_LCU'] = np.log(df.GDP\_Current\_LCU)

df['log\_PIT\_Current\_LCU'] = np.log(df.PIT\_current\_LCU)

df.dropna(subset=['log\_PIT\_Current\_LCU','log\_GDP\_Current\_LCU', 'PIT Rate', 'PIT\_Zero\_Threshhold', 'PIT\_Maximum\_Rate\_Threshold'])

X\_vars = df[['log\_GDP\_Current\_LCU', 'PIT Rate', 'PIT\_Zero\_Threshhold', 'PIT\_Maximum\_Rate\_Threshold']] ## X\_vars usually means our input variables (or independent variables)

Y\_vars = df[['log\_PIT\_Current\_LCU']] ## Y\_vars usually means our output/dependent variable

X1 = sm.add\_constant(X\_vars) ## let's add an intercept (beta\_0) to our model

Y1 = Y\_vars

model = sm.OLS(Y1, X1).fit()

print(model.summary())

model.params # gives a dataframe of the variable names and coefficient values

model.pvalues # gives a dataframe of the pvalues

model.tvalues # gives the t values

# see the section on Properties in this link <https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.RegressionResults.html>

**When You have a Panel and want predicted values into the dataset**

import statsmodels.api as sm

from matplotlib import pyplot as plt

df = df\_revenue\_data1.dropna(subset=["Country\_Code", "year","ln\_GDP\_PC\_Constant\_USD", "Tax\_Revenue"])

X\_vars = df[["Country\_Code", "year", "ln\_GDP\_PC\_Constant\_USD"]] ## X usually means our input variables (or independent variables)

Y\_vars = df[["year","Tax\_Revenue"]] ## Y usually means our output/dependent variable

X0 = X\_vars["ln\_GDP\_PC\_Constant\_USD"]

df\_revenue\_data1.dropna(subset=["ln\_GDP\_PC\_Constant\_USD", "Tax\_Revenue"])

X1 = sm.add\_constant(X0) ## let's add an intercept (beta\_0) to our model

Y1 = Y\_vars["Tax\_Revenue"]

model = sm.OLS(Y1, X1).fit()

Y\_hat1 = model.predict(X1) # make the predictions by the model

Y\_hat1.name='Predicted\_Tax\_Revenue' # Name the predicted Series

df = pd.concat([df, Y\_hat1], axis=1) # attach to original database

# Plotting only a particular year

X = df[df["year"]==2019]["ln\_GDP\_PC\_Constant\_USD"]

Y = df[df["year"]==2019]['Tax\_Revenue']

Y\_hat = df[df["year"]==2019]['Predicted\_Tax\_Revenue']

X\_labels = df[df["year"]==2019] ["Country\_Code"]

plt.figure(figsize=(12, 6))

plt.plot(X, Y, 'o') # scatter plot showing actual data

plt.plot(X, Y\_hat, 'r', linewidth=2) # regression line

offset\_x = -0.05

offset\_y = 0.5

# For the Marker Labels

for i in range(0, len(X)):

plt.text(X.iloc[i] +offset\_x, Y.iloc[i] +offset\_y , X\_labels.iloc[i], fontsize=12)

plt.xlabel('Log GDP Per Capita')

plt.ylabel('Tax Revenue')

plt.title('Tax Revenue vs GDP Per Capita')

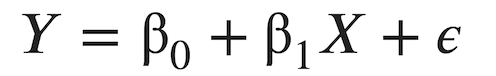
plt.show()

#save figure

filename='PIT Rate versus PIT Revenue Sri Lanka.png'

plt.savefig('C:/Users/wb305167/OneDrive - WBG/Sri Lanka/' + filename, dpi=200, bbox\_inches="tight")

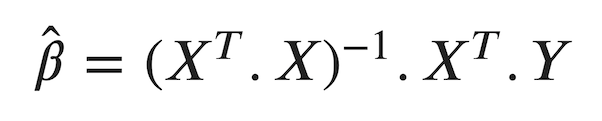
1. Regression – matrix method



Find beta to minimize the root mean square of the error

ε = Y - β[1 X]

solution



Y = df['GR6085'].values

X = df[['GDP60', 'REV']].values

X\_mat = np.hstack((np.ones((len(X),1)), X))

beta\_hat = np.linalg.inv(X\_mat.T.dot(X\_mat)).dot(X\_mat.T).dot(Y)

yhat = X\_m.dot(beta\_hat)

plt.scatter(X, Y)

plt.plot(X, yhat, color='red')

1. Probit

from sklearn.linear\_model import LinearRegression

import statsmodels.api as smf

probit\_model=smf.Probit(y,x)

result=probit\_model.fit()

print(result.summary2())

params = pd.DataFrame(result.params,columns={'coef'},)

print(params)

result1 = x

result1['y\_pred'] = result1['ln\_total\_income1'] \* params['coef'][0]

print(result1)

ax = result1.plot(x='ln\_total\_income1', y='y\_pred')

ax.set\_ylim([0.0, 1.05])

**Charts**

1. Plot bar with error bars

df2.plot(kind='bar', y=['Mean ROI\_2018', 'Mean ROI\_2019', 'Mean ROI\_2020'], yerr=df2[['Std ROI\_2018', 'Std ROI\_2019', 'Std ROI\_2020']].T.values)

1. Multiple plots on same chart

ax = newdf.plot()

newdf2.plot(ax=ax)

1. Multiple charts plots

[fig](https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure), [axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray) = [plt.subplots](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html#matplotlib.pyplot.subplots)(2, 2)

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[0, 0].plot([x](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), [y](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray))

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[0, 0].set\_title('Axis [0, 0]')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[0, 1].plot([x](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), [y](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), 'tab:orange')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[0, 1].set\_title('Axis [0, 1]')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[1, 0].plot([x](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), -[y](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), 'tab:green')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[1, 0].set\_title('Axis [1, 0]')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[1, 1].plot([x](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), -[y](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray), 'tab:red')

[axs](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html#numpy.ndarray)[1, 1].set\_title('Axis [1, 1]')

for [ax](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.html#matplotlib.axes.Axes) in [axs.flat](https://numpy.org/doc/stable/reference/generated/numpy.flatiter.html#numpy.flatiter):

[ax.set](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set.html#matplotlib.axes.Axes.set)(xlabel='x-label', ylabel='y-label')

# Hide x labels and tick labels for top plots and y ticks for right plots.

for [ax](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.html#matplotlib.axes.Axes) in [axs.flat](https://numpy.org/doc/stable/reference/generated/numpy.flatiter.html#numpy.flatiter):

[ax.label\_outer](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.label_outer.html#matplotlib.axes.Axes.label_outer)()

1. Generate a series of equally spaced numbers

# 100 numbers starting from 0 and ending in 100,000

x = np.linspace(0, 100000, 101)

1. Plot with legends

import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 20, 1000)

y1 = np.sin(x)

y2 = np.cos(x)

plt.plot(x, y1, "-b", label="sine")

plt.plot(x, y2, "-r", label="cosine")

plt.legend(loc="upper left")

plt.ylim(-1.5, 2.0)

plt.show()

1. Plot bar with labels

ax = df2[['Mean ROI\_2019']].plot(kind='bar')

for i, label in enumerate(list(df2.index)):

count = df2.loc[label]['Count ROI\_2019']

if np.isnan(count):

count\_str=""

else:

count\_str=str(count.astype(int))

ax.annotate(count\_str, (i-0.2, df2.loc[label]['Mean ROI\_2019'] + 0.01))

for doing it for multiple series put the code into the loop where p is each series

for p in ax.patches:

ax.annotate(df2['Count ROI\_2018'], (p.get\_x() \* 1.005, p.get\_height() \* 1.005))

1. Stacked Bar chart

df2[['Mean ROI\_2019', 'Mean ROI\_2023',]].plot(kind='bar', stacked=True)

1. Regular Bar - horizontal

Country = tmp['Country\_Name'].head(10)

Real\_GDP = tmp['GDP\_real'].head(10)

# Figure Size

fig, ax = plt.subplots(figsize =(16, 9))

# Horizontal Bar Plot

ax.barh(Country, Real\_GDP)

# Show top values

ax.invert\_yaxis()

plt.xlabel("Real GDP in tens of million US$")

#plt.ylabel("Log GDP Per Capita 2019")

plt.title("Top Countries in Real GDP in 2015 US$")

plt.grid(False)

filename="Top Countries in Real GDP in 2015 US$.png"

plt.savefig('C:/Users/ssj34/Documents/OneDrive/Duke/Economic Development/' + filename)

plt.show()

1. Annotate a chart

|  |  |
| --- | --- |
| ax.annotate('local max', xy=(3, 1), xycoords='data',  xytext=(0.8, 0.95), textcoords='axes fraction',  arrowprops=dict(facecolor='black', shrink=0.05), horizontalalignment='right', verticalalignment='top',  ) | A blue line with a black arrow  Description automatically generated |

xy is the point to be annotated – the arrow if needed will end here

xytext is where the text will be printed. xy need not be provided.

1. Matplotlib multiple plots

t = linspace(0,2\*math.pi,400)

a = sin(t)

b = cos(t)

c = a + b

plt.plot(t, a, t, b, t, c)

plt.show()

or

plt.figure()

...

plt.plot(t, a)

plt.plot(t, b)

plt.plot(t, c)

plt.show()

1. Plot polynomial on chart

z = np.polyfit(df1[df1['log\_X']>13.0]['log\_X'], df1[df1['log\_X']>13.0]['log\_P\_x\_gt\_X'], 1)

poly\_fn = np.poly1d(z)

#z gives the coefficients of the polynomial

#poly\_fn takes the coefficients and creates a function

plt.plot(log\_X, log\_P\_x\_gt\_X, 'o', log\_X, poly\_fn(log\_X),"r--")

plt.show()

1. Scatterplot for differently labelled regions with legend and markers and also adjust markers so as not to overlap.

from adjustText import adjust\_text # for non-overlapping texts pip install adjustText

tmp = df[(df.year==1960)|(df.year==1970)|(df.year==2019)] # extract the years you need

tmp = df[(df.year==1970)|(df.year==2019)]

tmp = tmp.pivot(index=['Country\_Code','Region\_Code'], columns='year', values='ln\_GDP\_per\_capita\_const\_usd') # place the rows defined by the different years as separate columns using the columns of 'ln\_GDP\_per\_capita\_const\_usd' as values

tmp = tmp.reset\_index()

tmp = tmp[~tmp.isin([np.nan, np.inf, -np.inf]).any(1)] #remove any NaNs

colors = ['c', 'b', 'y', 'm', 'r', 'k', 'g', '0.5', 'black']

markers = ['^', 'x', 's', '\*', 'D', 'v', '8', 'X', '$IND$']

plt\_={} # we create a dictionary of plots to map the legends

i=0

fig = plt.figure(figsize=(6.4, 4.8), dpi=my\_dpi)

# loop through the regions and plot them

for region\_category in ['SSA', 'LAC', 'SA', 'ECA', 'EAP', 'MENA', 'WER', 'NAM', 'IND']:

tmp1=tmp[tmp.Region\_Code == region\_category][[1970, 2019, 'Country\_Code']]

plt\_[i] = plt.scatter(tmp1[1970], tmp1[2019], marker=markers[i], color=colors[i], cmap="Accent", alpha=0.6, edgecolors="white", linewidth=1)

#plt.xscale('log') # log scale is helpful when we do not need to do a fitted line later in that case # you do not need to transform the variable into logs

#plt.yscale('log')

i=i+1

# here we use a mapping of log scale into antilog values to show on chart

# other useful tick commands

ticks = [400, 4000, 40000]

#locs, labels = plt.xticks() # Get the current locations and labels.

plt.xticks(np.log(ticks), ticks,

rotation=0) # Set text labels and properties.

plt.yticks(np.log(ticks), ticks,

rotation=0) # Set text labels and properties.

plt. xticks([]) # Disable xticks.

#plt.xticks(ticks, ticks) # simple way to just show xticks – no mapping change

#plt.yticks(ticks, ticks)

# way to generate the text labelling in arbitrary locations

texts = [plt.text(x,y,s, fontsize=4) for x, y, s in zip(tmp[1970], tmp[2019], tmp['Country\_Code'])]

# adjust text needs the list of plots

adjust\_text(texts, arrowprops=dict(arrowstyle='->', color='red'))

# Fit linear regression via least squares with numpy.polyfit

# It returns a slope (b) and intercept (a)

# deg=1 means linear fit (i.e., polynomial of degree 1)

b, a = np.polyfit(tmp[1970], tmp[2019], deg=1)

# Plot regression line

#plt.plot(xseq, a + b \* xseq, color="k", lw=2.5);

# single line command for fitted linear regression

plt.plot(np.unique(tmp[1970]), np.poly1d(np.polyfit(tmp[1970], tmp[2019], 1))(np.unique(tmp[1970])))

model = regress(tmp, [2019, 2017]) # this is created in stats\_python.py

model.params()

# Create sequence of 100 numbers from 0 to 100

xseq = np.linspace(np.log(300), 50000, num=100)

plt.show()

# can also put text using lambda function and annotate command

tmp[[1970, 2019, 'Country\_Code']].apply(lambda row: plt.annotate(row['Country\_Code'], # this is the text

(row[1970], row[2019]), # this is the point to label

textcoords="offset points", # how to position the text offset given in xytext below

xytext=(4,0), # distance from text to points (x,y)

ha='center', # horizontal alignment can be left, right or center

fontsize = 4),axis=1) # size of marker)

# this below is used for multiple legends

plt.legend((plt\_[0], plt\_[1], plt\_[2], plt\_[3], plt\_[4], plt\_[5], plt\_[6], plt\_[7]),

('SSA', 'LAC', 'SA', 'ECA', 'EAP', 'MENA', 'WER', 'NAM'),

scatterpoints=1,

loc='upper left',

ncol=3,

fontsize=8)

plt.xlabel("Log GDP Per Capita 1970")

plt.ylabel("Log GDP Per Capita 2019")

plt.title("GDP per capita 1970 versus 2019")

#plt.axis([0, 60, 0, 20])

#plt.ylim(0, max\_fdi)

#plt.xlim(5, 12)

plt.grid(True)

plt.show()

**Web scraping and beautifulsoup**

1. Read a webpage html and extract contents into a table

import requests

import bs4

import pandas as pd

res1 = requests.get('https://results.eci.gov.in/PcResultGenJune2024/index.htm')

res1.raise\_for\_status()

noStarchSoup = bs4.BeautifulSoup(res1.text, "lxml")

# reading options off a drop down list

items = noStarchSoup.select('option[value]')

values = [item.get('value') for item in items]

textValues = [item.text for item in items]

state\_url\_suffixes = values[1:]

state\_names = textValues[1:]

df = pd.DataFrame(columns = ['S.N.', 'State', 'Constituency', 'Candidate Party', 'EVM Votes', 'Postal Votes', 'Total Votes', '% of Votes'])

i=0

for state\_url\_suffix in state\_url\_suffixes:

state\_url = 'https://results.eci.gov.in/PcResultGenJune2024/partywiseresult-'+state\_url\_suffix+'.htm'

print('State: ', state\_names[i])

print('state\_url: ', state\_url)

# to extract Constituency URLs

res2 = requests.get(state\_url)

res2.raise\_for\_status()

soup\_state = bs4.BeautifulSoup(res2.text, "lxml")

items = soup\_state.select('option[value]')

values = [item.get('value') for item in items]

textValues = [item.text for item in items]

constituency\_url\_suffixes = values[1:]

constituency\_names = textValues[1:]

j=0

for constituency\_url\_suffix in constituency\_url\_suffixes:

constituency\_url = "https://results.eci.gov.in/PcResultGenJune2024/Constituencywise"+constituency\_url\_suffix+".htm"

#'https://results.eci.gov.in/PcResultGenJune2024/candidateswise-'+constituency\_url\_suffix+'.htm'

print('Constituency: ', constituency\_names[j])

print('constituency\_url: ', constituency\_url)

res3 = requests.get(state\_url)

res3.raise\_for\_status()

soup\_constituency = bs4.BeautifulSoup(res3.text, "lxml")

tables = pd.read\_html(constituency\_url)

df1 = tables[0]

df1['State'] = state\_names[i]

df1['Constituency'] = constituency\_names[j]

df = pd.concat([df, df1])

j=j+1

df.to\_csv('Data up to State '+state\_names[i]+'.csv')

i=i+1

df.to\_csv('Election Candidate Wise Data Lok Sabha 2024.csv')

res1 = requests.get('https://results.eci.gov.in/PcResultGenJune2024/index.htm')

res1.raise\_for\_status()

1. Extract divs from html

noStarchSoup = bs4.BeautifulSoup(res1.text, "lxml")

mydivs = noStarchSoup.find\_all("div", {"class": "flter-btns"})