Contents

[Long and wide format: 2](#_Toc141791087)

[Setting index and selecting colums and transpose : 2](#_Toc141791088)

[pivot method(denormalizing): 3](#_Toc141791089)

[Pivot\_table method: 4](#_Toc141791090)

[Drop dataframe: 5](#_Toc141791091)

[Melt method(normalizer): 5](#_Toc141791092)

[Wide\_to\_Long method: 5](#_Toc141791093)

[List,dict to DataFrame: 8](#_Toc141791094)

[String Function in dataframe: 8](#_Toc141791095)

[Multindexing and stacking: 9](#_Toc141791096)

[Reshaping and remodeling ( applying aggregate function with stack and unstack) 10](#_Toc141791097)

[Exploding columns having list as data to different rows with single column data. 11](#_Toc141791098)

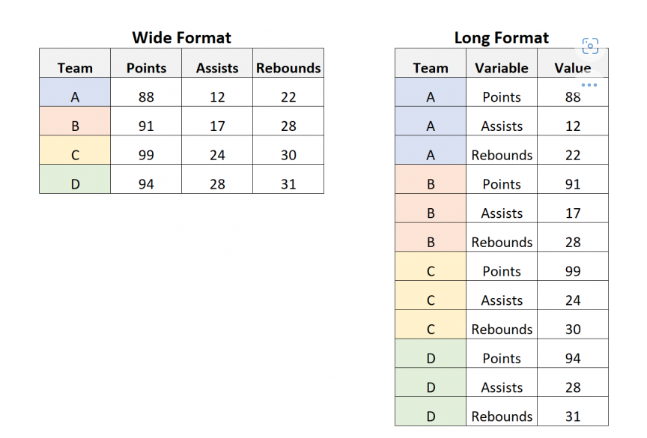
[Explode when there string to be split in the column 13](#_Toc141791099)

[JSON normalizer 14](#_Toc141791100)

[Dealing with nested data inside dataframe column 16](#_Toc141791101)

[Json.loads(),json.dumps() 16](#_Toc141791102)

# Long and wide format:

Long format is like normalized(key,value) format of data and wide format is like denormalized format

# Setting index and selecting colums and transpose :

new\_df = df.set\_index('name')

this set index name in the dataframe

new\_df = df.set\_index('name')[['height','weight']]

this set index name and select column height and weight (there is double bracket, while selecting multiple columns)

new\_df = df.set\_index('name')[['height', 'weight']].transpose()

this transposes the row to columns and vice versa

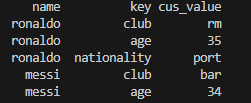
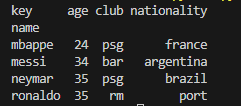
# pivot method(denormalizing):

index column should be unique, cannot aggregate values  
  
columns: kun column ko basis mah vayera column devision create garney? Tya key vanney column mah (club,age,nationality) cha. Ho tei key vanney column ko basis mah further 3 ota column create garney.

Values: aba kun column ko basis mah values halney. 2ota vanda ni badhi value table huna sakcha

denorm\_norm\_df=norm\_df.pivot(index='name',columns='key',values='cus\_value')

this will pivot the data frame as:

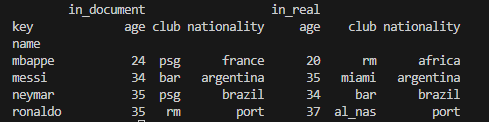
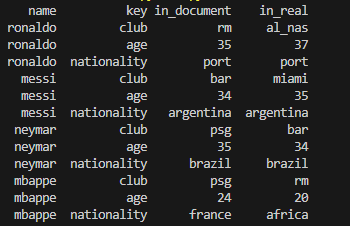


denorm\_norm\_df=norm\_df.pivot(index='name',columns='key',values=['in\_document','in\_real'])

or

denorm\_norm\_df=norm\_df.pivot(index='name',columns='key')

this will pivot the data frame as:



# Pivot\_table method:

When we want to do aggregate function in pivot table, then we use the pivot\_table method.

When there is multiple values in the index and value pair then the pivot method will raise an error and is confused which value to select. Here, we can use the pivot\_table method and use the aggregate function as required.

df.pivot\_table( values=None, index=None, columns=None, aggfunc=’mean’, fill\_value=None, margins=False, dropna=True, margins\_name=’All’)

aggfunc: default is mean

fill\_value: what to fill of found NaN

margins: calculate subtotal and grandtotal for each row and columns and show it

margins\_name: change the name of the margin column from All to specified name

dropna: to drop any row containing all value NaN

# Drop dataframe:

DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors=’raise’)

* Labels: index\_name, if you have specified any index name then the name should be kept index. E.g in above denorm\_norm\_df we should keep ‘ronaldo’ as label to delete that row.   
  If the index in not kept we should keep numbers.   
  e.g 1 to remove 2nd row.
* Axis: 0 for row and 1 for columns. Axis =1 rakhera column names specify garey column nai delete huncha
* Inplace: True if making changes in the original dataframe

# Melt method(normalizer):

df.melt(id\_vars=None, value\_vars=None, var\_name=None, value\_name=’value’, col\_level=None)

this will normalize the table.

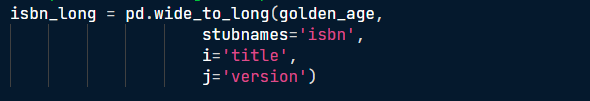
Id\_vars: which column to take as index

Value\_vars: which columns to normalize, if not set does for all columns

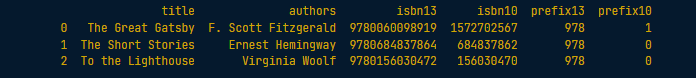
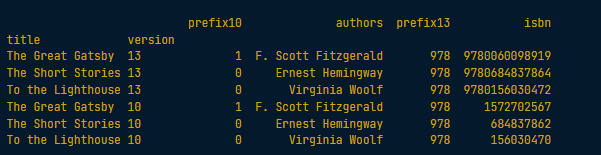
Var\_name and value\_name: new column names for the key and value columns

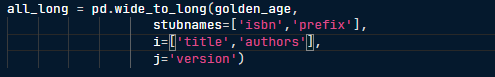
# Wide\_to\_Long method:

When we want to apply melt (normalize or wide-to-long) table but column name bata word chuttayera banauna parney cha vaney

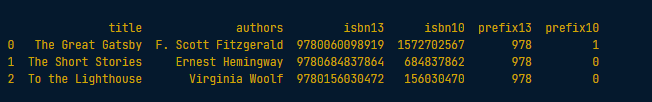
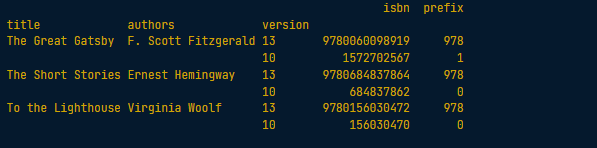


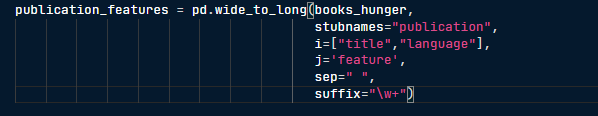
In the following example, isbn13 and isbn10 is separated to long format as follows. The 10 and 13 is extracted and given a column name ‘version’.



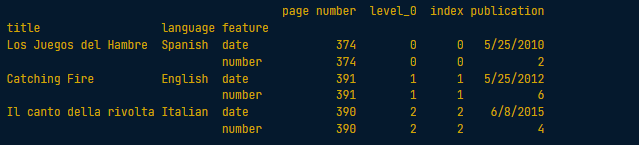
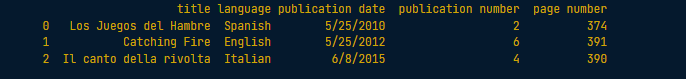


This will output:





This will output:



# List,dict to DataFrame:

list=[['ram',33],['shyam',35]]

df=pd.DataFrame(list,columns=['name','age'])

print(df)

dict={'name':['ram','shyam','hari'],'age':[24,25,26]}

df=pd.DataFrame(dict)

print(df)

Dict mah chai dict of dict mah mildaina.. dict of list mah matrai milcha.

# String Function in dataframe:

* To split the field

books\_dys.index = books\_dys.index.str.split("-")

#or (both works same if the index of the dataframe is main\_title

books\_dys[‘main\_title’] = books\_dys[‘main\_title’].str.split("-")

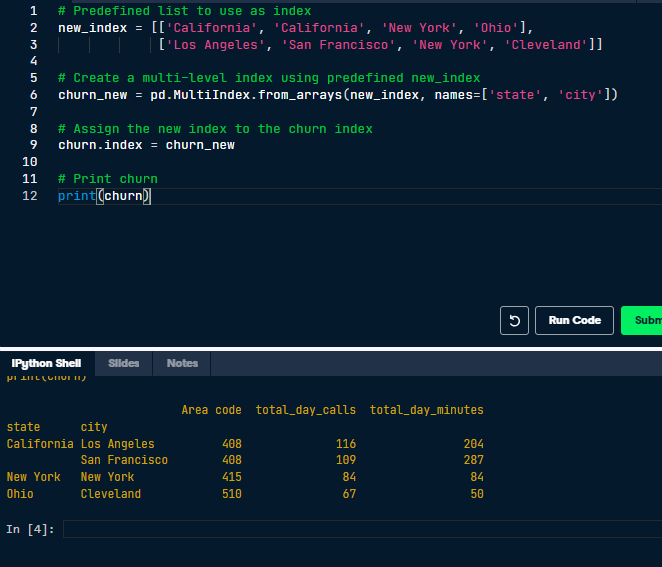
* To get the first value of each splited column data.

books\_dys.index = books\_dys.index.str.split('-').str.get(0)

* To split the column and store it into two different column

hp\_books[['writer', 'illustrator']] = hp\_books['authors'].str.split("/",expand=True)

# Multindexing and stacking:

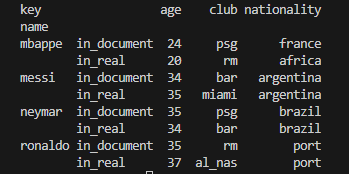
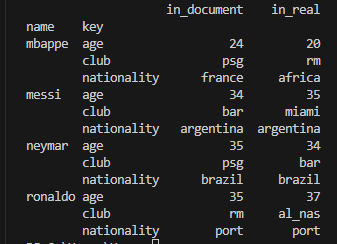
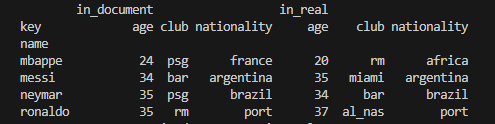


* Stacking is kind of changing the form of data to long form.

print(piv.stack(level=1))

#or( can define using the level also by name of the level)

print(piv.stack(level=”key”))



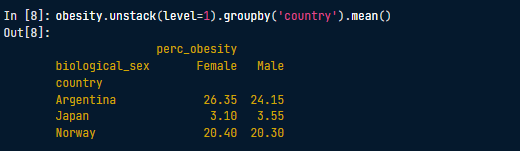
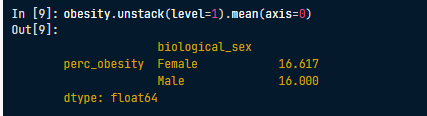
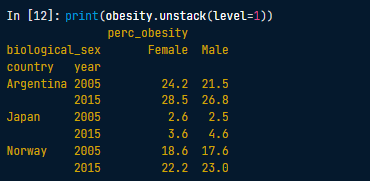
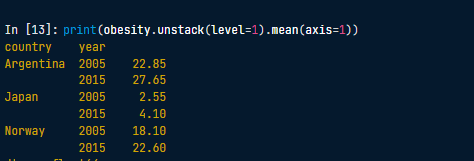
Level 1( key)

Level 0

* **We can do multiple level stacking at one by passing list in the level assignment.  
  level=[0,1] or level=[“first”, “second”]**
* **We can swap the index levels by swaplevel(0,2,axis=[0 for row level indexes and 1 for column level]) method**

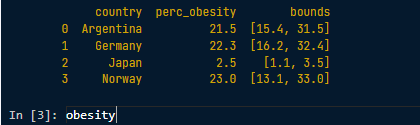
# Reshaping and remodeling ( applying aggregate function with stack and unstack)

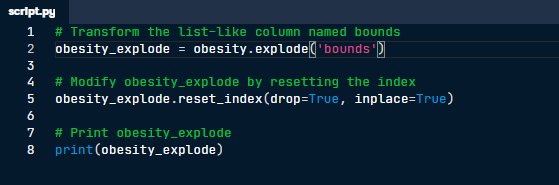
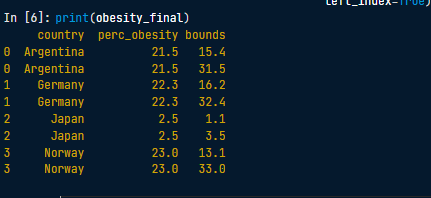
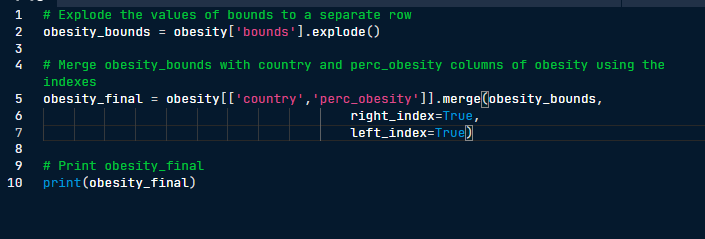
Obesity dataframe:



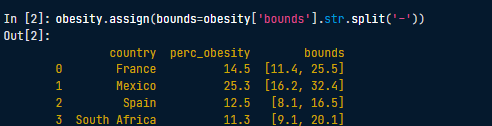
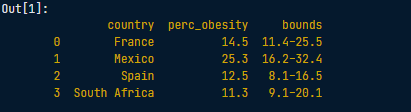
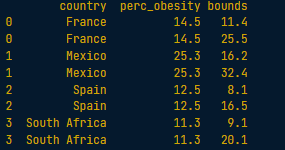
Axis mah chai kunchai axis mah garney ho tyo define garney. E.g. in above data, values in Female and male columns are different and we have to find the mean between them. Hence, we have to set the axis as 1 because we are finding mean between different columnar data. Or we can say we are to find mean between different columnar indexes, hence yesto gareko. This logic applies to swaplevel() method as well.

# Exploding columns having list as data to different rows with single column data.





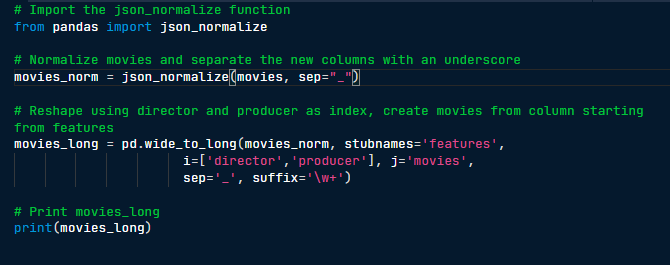
## Explode when there string to be split in the column

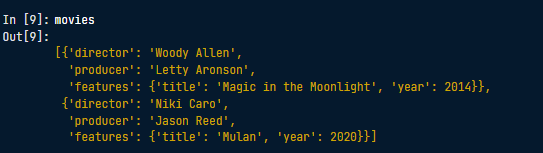


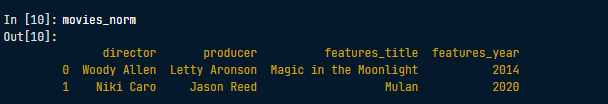
This will split the bounds column data and assign it to the bounds column.

# JSON normalizer

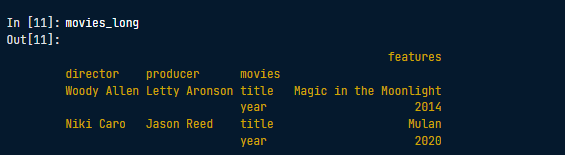
* First example, we normalize the json.





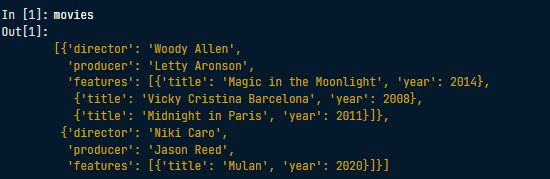


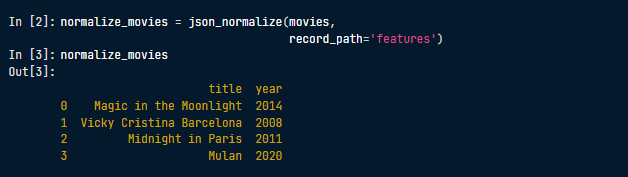
Sep from json\_normalize (). Default is ‘.’(dot)



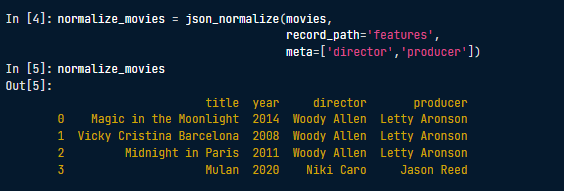
* Second example, in this there is a list of dict in features column. We tell json\_normalize() to look for that column only and make it presentable through **record\_path.** After that we tell it to have other column as well through **meta.**





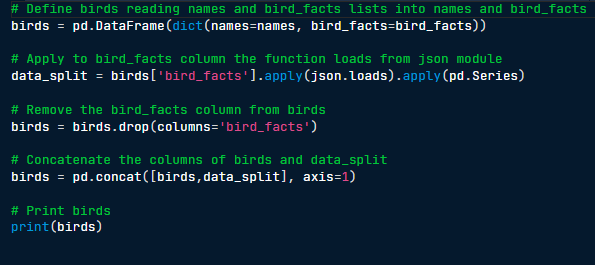
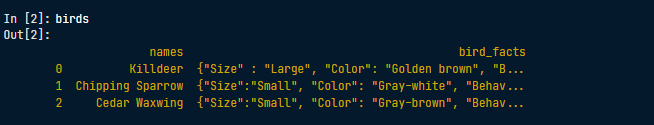


Just doing normalize movies with **record\_path** without entering **meta** columns. This will select the features column and make it like this.

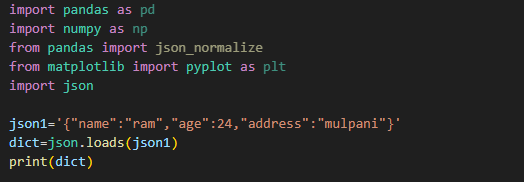


If we specify the **meta** columns, this will include the listed columns in the result.

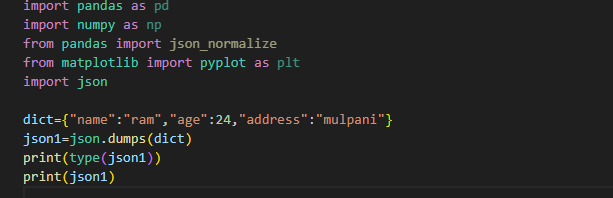
## Dealing with nested data inside dataframe column



# Json.loads(),json.dumps()



Json1 should be double quotes on the inside.  
json.loads() converts json to dict,list,etc. Tyo json1 defination jasto cha testai convert grcha from json to other.



Json.dumps() is the vice-versa of json.loads(). Which converts dict,list to json