**GroupBy Mechanics:**

split-apply-combine terimi gruplama için kullanılıyordu. Hadley Wickham bu ifadeyi kullanmıştı.

pandas nesnesinde (dataframe veya series olsun) olan veri bir veya birkaç anahtardan dağıtılır (split into).

Dataframe satır (axis=0) veya sütunlardan(axis=1) gruplanabilir. Sonrasında belli fonksiyonlar uygulanır (apply) ve tekrar bağlanır (combine)

**First Example:**

import pandas as pd

import numpy as np

dataframe=pd.DataFrame({"key1": ["a","a","b","b","c"],

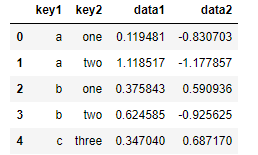
"key2":["one","two","one","two","three"],

"data1":np.random.randn(5),

"data2": np.random.randn(5)})

dataframe

output:



first\_group=dataframe["data1"].groupby(dataframe["key1"])

first\_group #Bu bir groupby objesi.

output:

<pandas.core.groupby.groupby.SeriesGroupBy object at 0x0000018DE86EE630>

#GroupBy mean ve sum metodları

first\_group.mean()

output:

key1

a 0.618999

b 0.500214

c 0.347040

Name: data1, dtype: float64

first\_group.sum()

output:

key1

a 1.237998

b 1.000428

c 0.347040

Name: data1, dtype: float64

ortalama= dataframe["data1"].groupby([dataframe["key1"],dataframe["key2"]]).mean()

ortalama

output:

key1 key2

a one 0.119481

two 1.118517

b one 0.375843

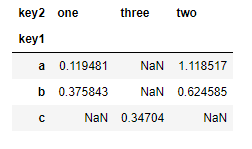
two 0.624585

c three 0.347040

Name: data1, dtype: float64

ortalama.unstack()

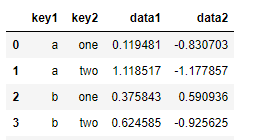
output:



Second Example:

dataframe

output:



states=np.array(["Ohio","California","California","Ohio","Ohio"])

years=np.array([2005,2005,2006,2005,2006])

dataframe["data1"].groupby([states, years]).mean()

output:

California 2005 1.118517

2006 0.375843

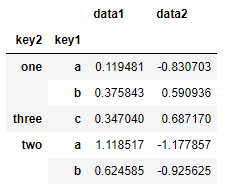
Ohio 2005 0.372033

2006 0.347040

Name: data1, dtype: float64

dataframe.groupby("key2").sum()

output:



dataframe.groupby(["key1","key2"]).size()

output:

key1 key2

a one 1

two 1

b one 1

two 1

c three 1

for name,group in dataframe.groupby("key1"):

print(name)

print(group)

a

key1 key2 data1 data2

0 a one 0.119481 -0.830703

1 a two 1.118517 -1.177857

b

key1 key2 data1 data2

2 b one 0.375843 0.590936

3 b two 0.624585 -0.925625

c

key1 key2 data1 data2

4 c three 0.34704 0.68717

dtype: int64

for name,group in dataframe.groupby("key1"):

print(name)

print(group)

#key1-k1; key2-k2

for (k1,k2), group in dataframe.groupby(["key1","key2"]):

print (k1,k2)

print(group)

a one

key1 key2 data1 data2

0 a one 0.119481 -0.830703

a two

key1 key2 data1 data2

1 a two 1.118517 -1.177857

b one

key1 key2 data1 data2

2 b one 0.375843 0.590936

b two

key1 key2 data1 data2

3 b two 0.624585 -0.925625

c three

key1 key2 data1 data2

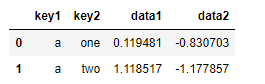
4 c three 0.34704 0.68717

#A recipe you may find useful is computing a dict of the data pieces #as a one-liner:

pieces= dict(list(dataframe.groupby("key1")))

**pieces["a"]**

**output:**



**dataframe.dtypes**

output:

key1 object

key2 object

data1 float64

data2 float64

dtype: object

#Group the columns by dtypes

groups= dataframe.groupby(dataframe.dtypes, axis=1)

dict(list(groups))

output:

{dtype('float64'): data1 data2

0 0.119481 -0.830703

1 1.118517 -1.177857

2 0.375843 0.590936

3 0.624585 -0.925625

4 0.347040 0.687170, dtype('O'): key1 key2

0 a one

1 a two

2 b one

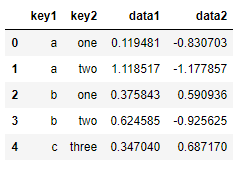
3 b two

4 c three}

Selecting a Column or Subset of Columns:

dataframe

output:



dataframe.groupby("key1")["data1"]

output:

<pandas.core.groupby.groupby.SeriesGroupBy object at 0x0000018DE8BCCB70>

dataframe.groupby("key1")[["data2"]]

output:

<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x0000018DE8BCC898>

dataframe["data1"].groupby(dataframe["key1"])

output:

<pandas.core.groupby.groupby.SeriesGroupBy object at 0x0000018DE8A47A90>

dataframe["data2"].groupby(dataframe["key1"])

output:

<pandas.core.groupby.groupby.SeriesGroupBy object at 0x0000018DE8A47550>

dataframe.groupby(["key1","key2"])["data1"].mean()

output:

key1 key2

a one 0.119481

two 1.118517

b one 0.375843

two 0.624585

c three 0.347040

Name: data1, dtype: float64

#The object returned by this indexing operation is a grouped #DataFrame if a list or array is passed and a grouped Series

#is just a single column name that is passed as a scalar:

a\_groups= dataframe.groupby(["key1","key2"])["data1"]

a\_groups.mean()

output:

key1 key2

a one 0.119481

two 1.118517

b one 0.375843

two 0.624585

c three 0.347040

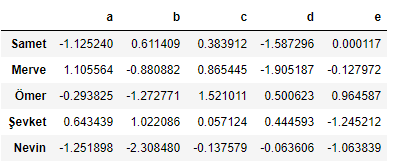
Name: data1, dtype: float64

**Grouping with dict and series:**

human=pd.DataFrame(np.random.randn(5,5), columns=["a","b","c","d","e"], index=["Samet","Merve","Ömer","Şevket","Nevin"])

human

output:



import warnings

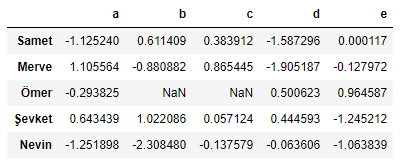
warnings.filterwarnings('ignore')

#human.iloc does not give result

human.ix[2:3,["b","c"]]= np.nan

human

output:

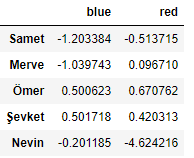


mapping = {"a":"red","b":"red","c":"blue","d":"blue","e":"red","f":"purple"}

by\_color= human.groupby(mapping,axis=1)

by\_color.sum()

output:



#We could do the same groupby with series

map\_series=pd.Series(mapping)

map\_series

output:

a red

b red

c blue

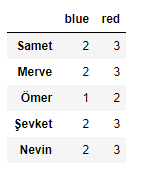
d blue

e red

f purple

human.groupby(map\_series,axis=1).count()

output:



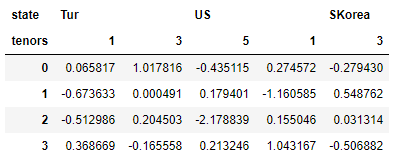
**Grouping by Index Levels:**

columns=pd.MultiIndex.from\_arrays([["Tur","Tur","US","US","SKorea"], [1,3,5,1,3]], names=["state","tenors"])

new\_dataframe=pd.DataFrame(np.random.randn(4,5), columns=columns)

new\_dataframe

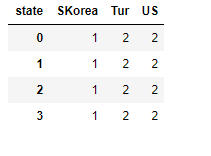
output:



#Kolonlara göre gruplamak için axis=1 yazılır...

new\_dataframe.groupby(level="state",axis=1).count()

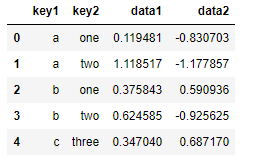
output:



**Data Aggregation:**

dataframe

out put:



group1=dataframe.groupby("key1")

group1["data1"].quantile(0.9)

#quantile is a series method.

output:

key1

a 1.018613

b 0.599711

c 0.347040

Name: data1, dtype: float64

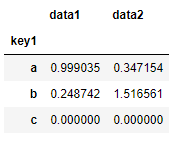
To use your own aggregation functions, pass any function that aggregates an array to the aggregate or agg method

def max\_vs\_min(arr):

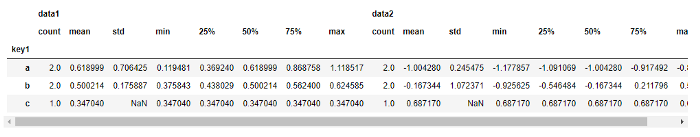
return arr.max()-arr.min()

group1.agg(max\_vs\_min

output:



group1.describe()

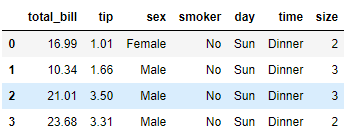


**A new Dataset:**

tips= pd.read\_csv("tips.csv")

tips.head(4)

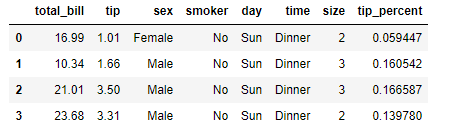
output:



tips["tip\_percent"]=tips["tip"]/tips["total\_bill"]

tips.head(4)

output:



group=tips.groupby(["sex","smoker"])

group\_percent=group["tip\_percent"]

group\_percent.agg("mean")

output:

sex smoker

Female No 0.156921

Yes 0.182150

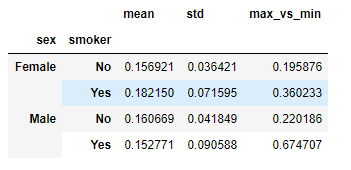
Male No 0.160669

Yes 0.152771

Name: tip\_percent, dtype: float64

group\_percent.agg(["mean","std",max\_vs\_min])

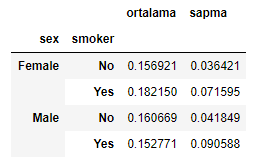
output:



#If you want to give another column names:

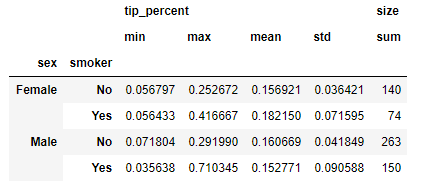
group\_percent.agg([("ortalama", "mean"),("sapma","std")])

**output:**



group.agg({"tip\_percent":["min","max","mean","std"], "size":"sum"})

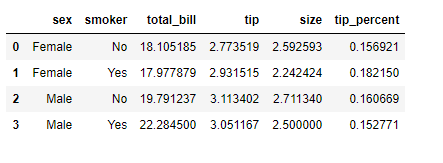
**output:**



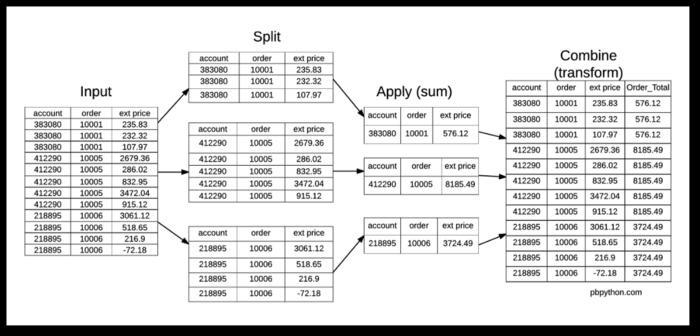
**Returning Aggregated Data in “unindexed” Form**

tips.groupby(["sex","smoker"], as\_index=False).mean()

output:

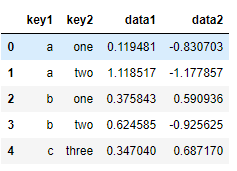


**Group-wise Operations and Transformations**



dataframe

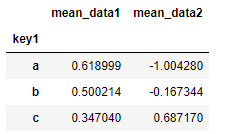
output:



k1\_mean=dataframe.groupby("key1").mean().add\_prefix("mean\_")

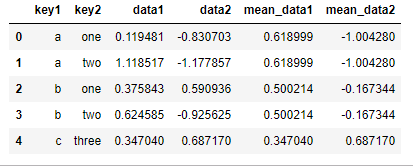
k1\_mean

output:



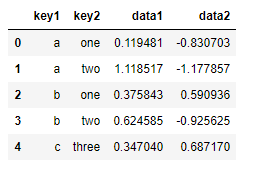
pd.merge(dataframe,k1\_mean, left\_on="key1", right\_index=True)

output:



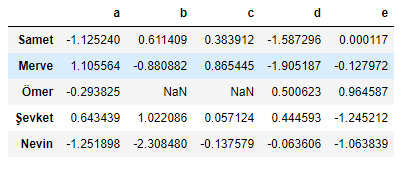
dataframe

output:



human

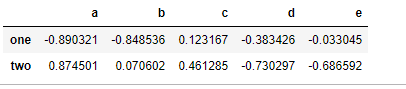
output:



anahtar=["one","two","one","two","one"]

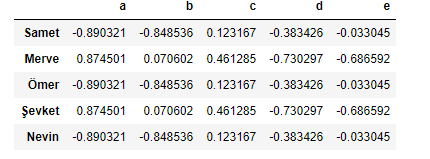
human.groupby(anahtar).mean()

output:



human.groupby(anahtar).transform("mean")

output:



#Create a new function returning the difference between the value #and the mean value.

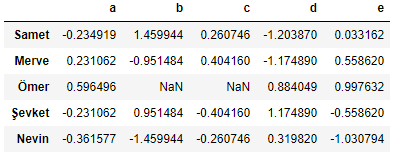
def demean(arr):

return arr-arr.mean()

demeaned= human.groupby(anahtar).transform(demean)

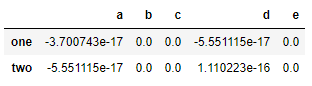
demeaned

**output:**



demeaned.groupby(anahtar).mean()

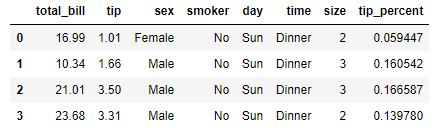
output:



**Apply: General split-apply-combine**

tips.head(4)

output:



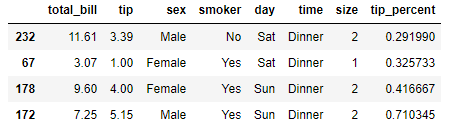
#Create a new function to select the rows with largest value in a #column.

def top\_column(dataframe,n=5, column= "tip\_percent"):

return dataframe.sort\_index(by=column)[-n:]

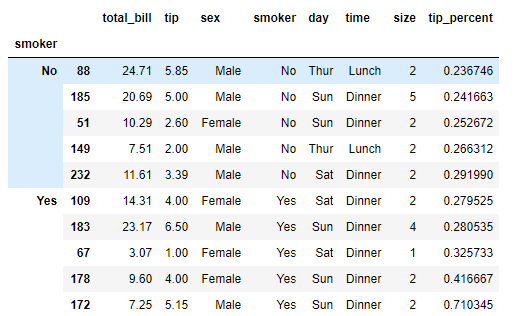
top\_column(tips, n=4,column="tip\_percent")

output:



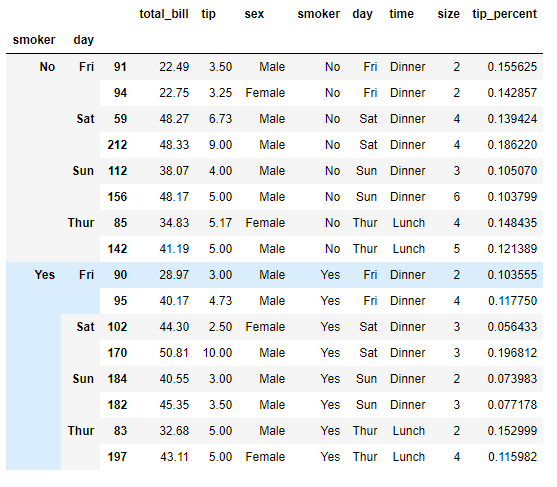
tips.groupby("smoker").apply(top\_column)

output:



tips.groupby(["smoker","day"]).apply(top\_column, n=2, column="total\_bill")

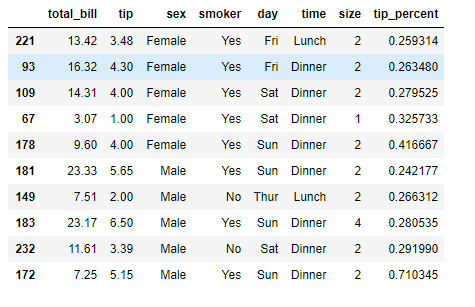
output:



**Suppressing the group keys:**

tips.groupby("sex",group\_keys=False).apply(top\_column)

output:



Example-1:

#Create a series containing random numbers

liste=pd.Series(np.random.rand(6))

liste

output:

0 0.710648

1 0.816111

2 0.494001

3 0.752940

4 0.166028

5 0.352260

dtype: float64

#Replace some rows with null

liste[::2]=np.nan

liste

output:

0 NaN

1 0.022216

2 NaN

3 0.979672

4 NaN

5 0.871373

dtype: float64

#Fill the null values by mean values

liste.fillna(liste.mean())

output:

0 0.710648

1 0.816111

2 0.494001

3 0.752940

4 0.166028

5 0.352260

dtype: float64

#state listesi

#doğu batı listesi (5\*5)

#veri serisini oluştur. rassal 10 sayı ve index de state listesi #olsun

#listeden belli değerleri null yap

#veriseti serisini doğu batı dizisine göre grupla ve ortalamyla bir #seri oluştur

#nul değerlerini ortalam değerlerle dolduracak bir obje oluştur ve #bu objeyi null değerleri yerine koy

state["İzmir","Edirne","Çanakkale","Muğla","Denizli","Kars","Van","Hakkari","Bitlis","Iğdır"]

regions=["West"]\*5+["East"]\*5

dataset=pd.Series(np.random.randn(10),index=state)

dataset[["İzmir","Edirne","Kars","Van"]]=np.nan

dataset

output:

İzmir NaN

Edirne NaN

Çanakkale 1.610974

Muğla 0.116179

Denizli 0.849781

Kars NaN

Van NaN

Hakkari -0.942297

Bitlis -0.967985

Iğdır 0.435215

dtype: float64

dataset.groupby(regions).mean()

output:

East 0.479653

West 0.613102

dtype: float64

#fill the null values by mean values

fill\_mean=lambda g: g.fillna(g.mean())

dataset.groupby(regions).apply(fill\_mean)

output:

İzmir 0.613102

Edirne 0.613102

Çanakkale 0.788366

Muğla 0.807255

Denizli 0.243684

Kars 0.479653

Van 0.479653

Hakkari 1.288655

Bitlis 0.035800

Iğdır 0.114505

dtype: float64

**Pivot Tables in Python:**

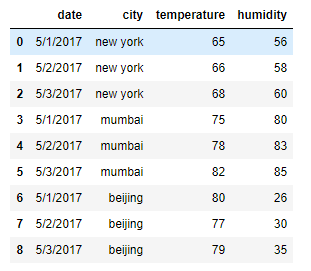
Pivot table is used to summarizeand aggregate data inside dataframe

import pandas as pd

weather\_df= pd.read\_csv("weather.csv")

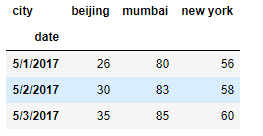
weather\_df

output:



weather\_df.pivot(index="date", columns="city", values="humidity")

output:

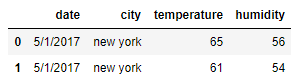


weather2= pd.read\_csv("weather2.csv")

weather3=pd.read\_csv("weather3.csv")

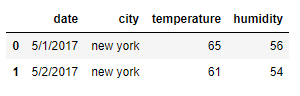
weather2.head(2)

output:



weather3.head(2)

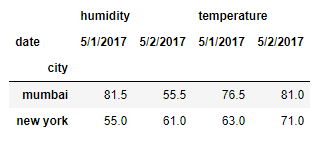
output:



#Return the mean value for the temperature and humidity. default #aggfunc is mean

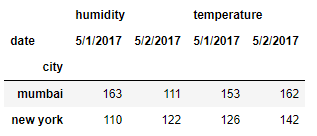
weather2.pivot\_table(index="city",columns="date")

output:



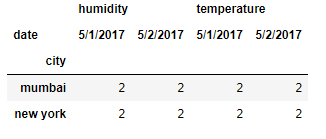
weather2.pivot\_table(index="city",columns="date", aggfunc="sum")

output:



weather2.pivot\_table(index="city",columns="date", aggfunc="count")

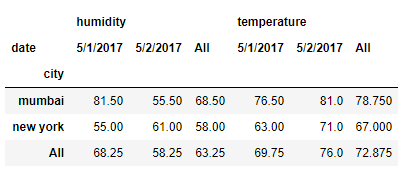
output:



#margins methods will add an All column

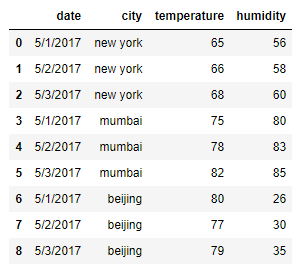
weather2.pivot\_table(index="city", columns="date", margins=True)

out put:



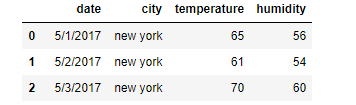
weather\_df

output:



weather3

output:



**The date in the date column is seen as a string so we should convert it to datetime**

weather3["date"]=pd.to\_datetime(weather3["date"])

type(weather3["date"][0]) #Timestamp

pandas.\_libs.tslibs.timestamps.Timestamp

#Monthly average temperature and humidity values

weather3.pivot\_table(index=pd.Grouper(freq="M", key="date"),columns="city")

output:

