```
link:
https://www.youtube.com/watch?v=WcDaZ67TVRo
data_source:
https://github.com/ifrankandrade/python-course-for-excel-users
```

part A: Dimensions

```
1.core syntax
2.excel with pythons
3.SQL+DB with python(advance)
4.Real case(YT+github) + question banks(buy)
```

Part B: modules

```
1.OS
2.Pandas
  series
  dataframe
  csv
  jason
3.numpy
4.mathlib
```

chapter 1: OS modules

In [5]:
```
"""
To get the location of the current working directory(CWD) os.getcwd() is use
"""
#import os
import pandas as pd
os.getcwd()
```

Out[5]:
```
'C:\\Users\\fengs'
```

In [10]:
```
os.listdir()
#show files(elements) under cwd:'C:\\Users\\fengs'
```

In [13]:
```
"""
 for creating a new folder:
os.mkdir()
os.makedirs()
"""
# Directory
directory = "GeeksforGeeks"

# Parent Directory path
parent_dir = "C:/Users/fengs/Desktop/pd_real"
#copy from pc: C:\Users\fengs\Desktop\pd_real, need to change to /

# Path
path = os.path.join(parent_dir, directory)
print(path)
#os.mkdir(path)
print("Directory '% s' created" % directory)
```

```
C:/Users/fengs/Desktop/pd_real\GeeksforGeeks
Directory 'GeeksforGeeks' created
```

In [18]:
```
#way 1:change \(pc) to /(pw)
os.mkdir('C:/Users/fengs/Desktop/pd_real/gd_2')

#way 2:add r: raw path
os.mkdir(r'C:\Users\fengs\Desktop\pd_real\gd_3')
```

chatper 2 Pandas

```
1.pandas.series() is a one-dimensional labeled array capable of holding
data of any type,like a    list
  df is 2-d array
2.pandas.Series( data, index, dtype, name, copy)
3.類似於DF['xxxx'] 單獨一個col
```

```
excel            pandas
worksheet        Dataframe
column           series
row heading      index
row              row
Empty cell       NaN
```

In [1]:
```
import pandas as pd
import numpy  as np
```

2.1 create DF with list

In [26]:
```
list = ['g', 'e', 'e', 'k', 's']
ser = pd.Series(list)
ser
```

Out[26]:
```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

In [21]:
```
data=np.array([[1,4],[2,5],[8,9]])
print(data)
#[1,4] is 1 row record,[2,5] is 2nd row
```

```
[[1 4]
 [2 5]
 [8 9]]
```

In [25]:
```
#pandas.Series( data, index, dtype, name, copy)
#pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None
data2=pd.DataFrame(data)
data2
```

Out[25]:

	0	1
0	1	4
1	2	5
2	8	9

In [27]:
```
data3=pd.DataFrame(data,index=['row1','row2','row3'],columns=['col1','col2']
data3
#index is rows names

```

Out[27]:

	col1	col2
row1	1	4
row2	2	5
row3	8	9

2.2 create dataframe with dict

In [1]:
```
#list for this example
#without numpy
data=[[1,4],[2,5],[8,9]] #this is n-d list but not array
data
```

Out[1]:
```
[[1, 4], [2, 5], [8, 9]]
```

In [4]:
```
data3=pd.DataFrame(data,index=['row1','row2','row3'],columns=['col1','col2']
data3
#so no need to change to numpy first
```

Out[4]:

	col1	col2
row1	1	4
row2	2	5
row3	8	9

2.2.2step to create DF from dict

In [7]:
```
#a 1-d list
state=['California','Texas','Florida','New York']
population=[1000,2000,4000,5000]
```

In [9]:
```
"""
store list into a dictionary
need to set keys
d = {
    <key>: <value>,
    <key>: <value>,
    ....
    }
"""
dict={"States":state,"Poplulations":population}
print(dict)
```

```
{'States': ['California', 'Texas', 'Florida', 'New York'], 'Poplulations': [100
0, 2000, 4000, 5000]}
```

In [13]:
```
df_country=pd.DataFrame(dict)
df_country
#'States','Population' become col-headind,
#list
```

Out[13]:

	States	Poplulations
0	California	1000
1	Texas	2000
2	Florida	4000
3	New York	5000

chapter 3: create DF from CSV

3.1 show DataFrame

In [2]:
```
#Read csv file =data set
df_exam=pd.read_csv(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel
df_exam.head() #head() show first 5 rows
```

Out[2]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

In [28]:
```
"""basic attributes,methods
"""
#1.show first 5 rows
df_exam.head(5)

#2.show last 5 rows
df_exam.tail(5)

#3.get access to shape attribute
df_exam.shape #(num of row, num of cols)

#4.display n rows/columns
#pandas.set_option(pat, value)
pd.set_option("display.max_columns", 3)
pd.set_option("display.max_rows", 5)
print(df_exam) #dpn't know why it display incorrectly
```

```
     gender  ... writing score
0    female  ...            74
1    female  ...            88
..      ...  ...           ...
998  female  ...            77
999  female  ...            86

[1000 rows x 8 columns]
```

In [30]:
```
df_show=pd.set_option("display.max_rows", 5)
print(df_show) #no need to assign to a variable
```

```
None
```

3.2 functions +attribute + methods

```
1.attribute:
   ~a value associate with an object.
   ~ref by name using dotted expression
   ~df.columns
   ~no related any modules/package

2.function
  ~a group of related statements that performs a specific task
   ~通用到whole py script.
  ~eg: python built-in functions:
  max();min();len()

3.methods
  ~a function which define inside a class body
  ~要import xxxxx
  ~eg：df.head()

```

In [31]:
```
import pandas as pd
import numpy  as np
```

3.2.2. attributes

```
index
columns
axes
dtypes
size
shape
ndim
empty
T
values
```

In [32]:
```
#shape attribute
df_exam.shape #(rows,cols)
```

Out[32]:
```
(1000, 8)
```

In [33]:
```
#index attributes
df_exam.index
```

Out[33]:
```
RangeIndex(start=0, stop=1000, step=1)
```

In [34]:
```
#show cols names
df_exam.columns
```

Out[34]:
```
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
       'test preparation course', 'math score', 'reading score',
       'writing score'],
      dtype='object')
```

In [36]:
```
#DDL(data type) of each columns
df_exam.dtypes
```

Out[36]:
```
gender            object
race/ethnicity    object
                   ...
reading score      int64
writing score      int64
Length: 8, dtype: object
```

3.2.3 methods

In [5]:
```
df_exam=pd.read_csv(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel
```

In [38]:
```
#1.shoew first 4 rows
df_exam.head(4)

#2.show df information
df_exam.info() #better than df_exam.dtypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64 
 6   reading score                1000 non-null   int64 
 7   writing score                1000 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

In [39]:
```
#3. show basic statistic info
df_exam.describe()
```

Out[39]:

	math score	reading score	writing score
count	1000.000	1000.000	1000.000
mean	66.089	69.169	68.054
...	...	...	...
75%	77.000	79.000	79.000
max	100.000	100.000	100.000

8 rows × 3 columns

3.2.3 functions

In [3]:
```
#set tools display rows
pd.set_option('display.max_rows', 10)  #
pd.set_option('display.max_columns', 10)  #
```

In [46]:
```
#1:numbers of row
len(df_exam) #use df_exam.shape is better
```

Out[46]:
```
1000
```

In [48]:
```
#2. max//min of a columns

df_exam['math score'].max()
#   df_exam.columns #show cols name

```

Out[48]: 100

In [66]:
```
"""
3. a row of min/max value of a cols
loc(row,col) + condition

"""
df_exam.loc[df_exam['math score'] == df_exam['math score'].max()]
```

Out[66]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
149	male	group E	associate's degree	free/reduced	completed	100	100	93
451	female	group E	some college	standard	none	100	92	97
458	female	group E	bachelor's degree	standard	none	100	100	100
623	male	group A	some college	standard	completed	100	96	86
625	male	group D	some college	standard	completed	100	97	99
916	male	group E	bachelor's degree	standard	completed	100	100	100
962	female	group E	associate's degree	standard	none	100	100	100

In [67]:
```
"""
4. round function:round(x,d) d:decimal point
"""
round(df_exam,2)
```

Out[67]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

1000 rows × 8 columns

chapter 4 Dataframe Query

In [9]:
```
#1.select df columns
df_exam['gender'].head(4)
```

Out[9]:
```
0    female
1    female
2    female
3      male
Name: gender, dtype: object
```

In [10]:
```
#2.check column type
type(df_exam['gender'])
```

Out[10]:
```
pandas.core.series.Series
```

In [16]:
```
"""
3.select col with dot (has pitfalls陷阱)
"""
#df_exam.gender.head(4) # vs #1
#df_exam.math score  #if name has space,it will error
# vs
df_exam['math score']
```

Out[16]:
```
0      72
1      69
2      90
3      47
4      76
       ..
995    88
996    62
997    59
998    68
999    77
Name: math score, Length: 1000, dtype: int64
```

4.2 Query Many cols

In [20]:
```
"""1.use [['x','y','z']],
    pair[]=DataFrame
    single[]=seriese
"""
df_exam.columns
df_exam[['gender', 'race/ethnicity', 'parental level of education']].head(2)
```

Out[20]:

	gender	race/ethnicity	parental level of education
0	female	group B	bachelor's degree
1	female	group C	some college

Chapter 5: DML DataFrame(Numpy)

```
1.def
NumPy stands for numeric python which is a python package for the
computation and processing of the multidimensional and single dimensional
array elements.
2.real:
pandas handle n-d list, dictionary
numpy relates to n-d array
```

In [22]:
```
df_exam=pd.read_csv(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel
```

In [23]:
```
df_exam.columns
```

Out[23]:
```
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
       'test preparation course', 'math score', 'reading score',
       'writing score'],
      dtype='object')
```

In [26]:
```
#1.add col
df_exam['new score']=df_exam['math score']+10
df_exam.head(3)
```

Out[26]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	82
1	female	group C	some college	standard	completed	69	90	88	79
2	female	group B	master's degree	standard	none	90	95	93	100

In [28]:
```
"""2.add new cols with numpy
#if DF more than 1000 rows use numpy
"""
import numpy as np

np.arange(0,1000)
len(np.arange(0,1000))
```

Out[28]:
```
1000
```

In [32]:
```
df_exam['new score']=np.arange(0,1000)
df_exam.head(3)
```

Out[32]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	0
1	female	group C	some college	standard	completed	69	90	88	1
2	female	group B	master's degree	standard	none	90	95	93	2

In [33]:
```
"""#3.numpy random generate a int [1:100]
   randin(start,end) ,randin=random integer
   random.randint(low, high=None, size=None, dtype=int)
"""
np.random.randint(1,100) #size define 1,gen one int
```

Out[33]:
```
75
```

In [35]:
```
#for each loop, simple
np_ra=np.random.randint(1,100,size=5)
for i in np_ra:
    print(i)
```

```
45
32
88
54
10
```

In [36]:
```
#standard for loop
np_ra=np.random.randint(1,100,size=5)
for i in range(0,len(np_ra),1):
    print(np_ra[i])
```

```
61
90
99
24
17
```

In [38]:
```
#4.add col with numpy,random gen num
df_exam['new score']=np.random.randint(1,100,size=1000) #size=1000 as row=1k
df_exam.head()
```

Out[38]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	34
1	female	group C	some college	standard	completed	69	90	88	69
2	female	group B	master's degree	standard	none	90	95	93	7
3	male	group A	associate's degree	free/reduced	none	47	57	44	37
4	male	group C	some college	standard	none	76	78	75	49

In [39]:
```
#find min/max row of new score column
df_exam.loc[df_exam['new score'] == df_exam['new score'].max()]
```

Out[39]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
24	male	group D	bachelor's degree	free/reduced	completed	74	71	80	99
85	female	group C	some college	standard	none	73	80	82	99
298	male	group C	high school	free/reduced	completed	40	46	50	99
553	male	group D	some college	free/reduced	none	77	62	64	99
612	male	group C	bachelor's degree	standard	completed	94	90	91	99
634	male	group D	some high school	standard	none	84	84	80	99
666	female	group C	some college	free/reduced	completed	63	73	71	99
768	female	group D	some high school	standard	none	68	71	75	99
873	male	group E	associate's degree	free/reduced	none	90	90	82	99
953	male	group C	high school	standard	completed	58	52	54	99

In [40]:
```
df_exam.loc[df_exam['new score'] == df_exam['new score'].min()]
```

Out[40]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
21	female	group B	some college	free/reduced	completed	65	75	70	1
124	male	group E	some college	standard	none	83	80	73	1
156	female	group E	high school	free/reduced	completed	66	74	78	1
236	male	group D	bachelor's degree	free/reduced	none	63	66	67	1
239	male	group C	associate's degree	standard	none	84	80	80	1
...	...	...	...	...	...	...	...	...	...
407	female	group B	associate's degree	standard	none	82	80	77	1
558	female	group B	associate's degree	free/reduced	none	53	70	70	1
580	female	group D	some high school	standard	none	81	97	96	1
899	female	group D	some high school	standard	completed	65	78	82	1
969	female	group B	bachelor's degree	standard	none	75	84	80	1

12 rows × 9 columns

In [5]:
```
"""4.create a floating number between 1 and 100+ round up to 2 decimals
numpy.random.uniform(low,high,size)
"""
df_exam['new score']=np.random.uniform(1,100,size=1000)#if not size=1000, wi
df_exam.round(2) #no round(df_exam,2) as round is gen function of python, no
#要def function
```

Out[5]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	63.24
1	female	group C	some college	standard	completed	69	90	88	64.77
2	female	group B	master's degree	standard	none	90	95	93	90.54
3	male	group A	associate's degree	free/reduced	none	47	57	44	97.71
4	male	group C	some college	standard	none	76	78	75	16.11
...	...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95	61.27
996	male	group C	high school	free/reduced	none	62	55	55	1.34
997	female	group C	high school	free/reduced	completed	59	71	65	56.04
998	female	group D	some college	standard	completed	68	78	77	62.28
999	female	group D	some college	free/reduced	none	77	86	86	6.62

1000 rows × 9 columns

chapter6 operation with Df

In [46]:
```
import pandas as pd
import numpy  as np
df_exam=pd.read_csv(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel
```

In [49]:
```
#set tools display rows
pd.set_option('display.max_rows', 10)  #
pd.set_option('display.max_columns', 10)  #
df_exam
```

In [51]:
```
"""1.cal sum of a columns
"""
df_exam['math score'].sum()
```

Out[51]: 66089

In [54]:
```
"""2.count ,mean, std,
"""
df_exam['math score'].count()
df_exam['math score'].mean()
round(df_exam['math score'].std(),2)
```

Out[54]: 15.16

In [55]:
```
"""3.describle
"""
round(df_exam.describe(),2)
```

Out[55]:

	math score	reading score	writing score
count	1000.00	1000.00	1000.00
mean	66.09	69.17	68.05
std	15.16	14.60	15.20
min	0.00	17.00	10.00
25%	57.00	59.00	57.75
50%	66.00	70.00	69.00
75%	77.00	79.00	79.00
max	100.00	100.00	100.00

In [60]:
```
"""4.quantile()
numpy.quantile(arr, q, axis = None)
"""
print(np.quantile(df_exam['reading score'],0.5))
print(np.quantile(df_exam['reading score'],0.75))
```

```
70.0
79.0
```

6.2 rows process

In [62]:
```
#1.total by sum other
df_exam['total']=df_exam['math score']+df_exam['reading score']+df_exam['rea
df_exam.head()
de_exam.round(2) #precise to 2 decimals
```

Out[62]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	total
0	female	group B	bachelor's degree	standard	none	72	72	74	216
1	female	group C	some college	standard	completed	69	90	88	249
2	female	group B	master's degree	standard	none	90	95	93	280
3	male	group A	associate's degree	free/reduced	none	47	57	44	161
4	male	group C	some college	standard	none	76	78	75	232

chapter 7 Dateframe fine-tune(優化)

In [10]:
```
df_exam.head()
```

Out[10]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	63.238354
1	female	group C	some college	standard	completed	69	90	88	64.767176
2	female	group B	master's degree	standard	none	90	95	93	90.543414
3	male	group A	associate's degree	free/reduced	none	47	57	44	97.705107
4	male	group C	some college	standard	none	76	78	75	16.112690

In [7]:
```
"""7.1 value_count, count elements of a column
Series.value_counts(normalize=False, sort=True, ascending=False, bins=None,
"""
#1.simple count, can only count row NaN VALUE
df_exam['gender'].count()
```

Out[7]:
```
1000
```

In [8]:
```
#value_count can count how many female/ male
#term:count xxx elements by category
df_exam['gender'].value_counts()
```

Out[8]:
```
female    518
male      482
Name: gender, dtype: int64
```

In [9]:
```
"""7.2 return the relative frequency
% of each elements
"""
df_exam['gender'].value_counts(normalize=True)

```

Out[9]:
```
female    0.518
male      0.482
Name: gender, dtype: float64
```

In [12]:
```
#real usage
df_exam['parental level of education'].value_counts(normalize=True).round(4)
```

Out[12]:
```
some college          0.226
associate's degree    0.222
high school           0.196
some high school      0.179
bachelor's degree     0.118
master's degree       0.059
Name: parental level of education, dtype: float64
```

In [14]:
```
"""7.3 sort data
DataFrame.sort(columns=None, axis=0, ascending=True
               , inplace=False, kind='quicksort', na_position='last', **kwar
"""
df_exam.sort_values(by=['math score'],ascending=False).head(4) #by= is optio
```

Out[14]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
962	female	group E	associate's degree	standard	none	100	100	100	81.323253
625	male	group D	some college	standard	completed	100	97	99	41.641384
458	female	group E	bachelor's degree	standard	none	100	100	100	63.719324
623	male	group A	some college	standard	completed	100	96	86	66.962459

In [15]:
```
"""ex2:sort by many cols
"""
df_exam.sort_values(by=['math score','reading score']).head(4)
```

Out[15]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
59	female	group C	some high school	free/reduced	none	0	17	10	96.561534
980	female	group B	high school	free/reduced	none	8	24	23	15.489817
17	female	group B	some high school	free/reduced	none	18	32	28	93.174847
787	female	group B	some college	standard	none	19	38	32	41.011983

In [16]:
```
"""ex3:arg: inplace=False , not update the dataframe
    true update dataframe
"""
df_exam.head(4) #default sorting
```

Out[16]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
0	female	group B	bachelor's degree	standard	none	72	72	74	63.238354
1	female	group C	some college	standard	completed	69	90	88	64.767176
2	female	group B	master's degree	standard	none	90	95	93	90.543414
3	male	group A	associate's degree	free/reduced	none	47	57	44	97.705107

In [17]:
```
df_exam.sort_values(by=['math score','reading score'],inplace=True)
df_exam.head(4) # sorting has update
```

Out[17]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
59	female	group C	some high school	free/reduced	none	0	17	10	96.561534
980	female	group B	high school	free/reduced	none	8	24	23	15.489817
17	female	group B	some high school	free/reduced	none	18	32	28	93.174847
787	female	group B	some college	standard	none	19	38	32	41.011983

In [19]:
```
#sort  by str
#group A--group b---group C
df_exam.sort_values(by=['race/ethnicity'],ascending=True,key=lambda col:col.
```

Out[19]:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	new score
464	male	group A	bachelor's degree	standard	completed	75	58	62	41.912005
378	female	group A	some high school	standard	none	71	83	77	25.613728
769	male	group A	some college	free/reduced	none	58	60	57	87.154273
589	female	group A	some high school	standard	none	48	66	65	46.637284
88	female	group A	some college	standard	none	58	70	67	78.105995
...	...	...	...	...	...	...	...	...	..
884	female	group E	associate's degree	standard	none	51	51	54	73.671729
700	female	group E	bachelor's degree	standard	completed	79	81	82	23.184746
533	female	group E	associate's degree	standard	completed	79	88	94	60.079173
50	male	group E	some college	standard	none	53	55	48	38.522933
962	female	group E	associate's degree	standard	none	100	100	100	81.323253

1000 rows × 9 columns

chapter 8:Pivot table

```
1.pivot()
~reshape data based on columns values
~not support data aggregation

2.pivot_table() #same as excel in MS
create a spreadsheet-style pivot table
(support data aggregation)
```

In [20]:
```
"""7.1 pivot()
"""
df = pd.DataFrame({'fff': ['one', 'one', 'one', 'two', 'two',
                           'two'],
                   'bbb': ['P', 'Q', 'R', 'P', 'Q', 'R'],
                   'baa': [2, 3, 4, 5, 6, 7],
                   'zzz': ['h', 'i', 'j', 'k', 'l', 'm']})
df
```

Out[20]:

	fff	bbb	baa	zzz
0	one	P	2	h
1	one	Q	3	i
2	one	R	4	j
3	two	P	5	k
4	two	Q	6	l
5	two	R	7	m

In [26]:
```
#show in pivot tale type
df.pivot(index='fff', columns='bbb', values='baa')
```

Out[26]:

bbb	P	Q	R
fff			
one	2	3	4
two	5	6	7

7.2 Pivot_table()

In [ ]:
```
import pandas as pd
import numpy  as np
```

In [65]:
```
df_sale=pd.read_excel(r'C:\Users\fengs\Desktop\pd_real\python-course-for-exc
```

In [68]:
```
### use r to break long file path
df_gdp=pd.read_excel(r'C:/Users/fengs/Desktop/pd_real/python-course-for-exce
           r'/3.Pivot Table/gdp.xlsx')
```

In [69]:
```
df_gdp.head(4)
```

Out[69]:

	unid	wbid	country	year	SES	gdppc	yrseduc	popshare
0	4	AFG	Afghanistan	1970	3.474212	709.00000	NaN	0.003097
1	4	AFG	Afghanistan	1920	26.968016	731.75677	NaN	0.003245
2	4	AFG	Afghanistan	1990	1.269530	604.00000	NaN	0.002347
3	4	AFG	Afghanistan	1960	15.763076	739.00000	NaN	0.003039

In [71]:
```
"""
1:
DataFrame.pivot(index=None, columns=None, values=None)
"""
df_gdp.pivot(index='year',columns='country',values='gdppc').head(5)
```

Out[71]:

country	Afghanistan	Albania	Algeria	Angola	Argentina	...	Venezuela	Vietnam	Ye
year									
1880	585.46509	522.0000	819.18604	533.66669	1731.5	...	653.0	556.62793	811.7
1890	635.93024	598.0000	923.37207	567.33331	2152.0	...	737.0	608.25580	881.5
1900	686.39532	685.0000	1027.55810	601.00000	2756.0	...	821.0	659.88373	951.3
1910	736.86047	780.0000	1131.74410	628.69232	3822.0	...	886.0	711.51166	1021.0
1920	731.75677	861.3125	1201.21620	682.62964	3473.0	...	1173.0	713.94592	1017.2

5 rows × 149 columns

```
use salse data to make a excel-type pivot table
```

In [72]:
```
df_sales=pd.read_excel(r'C:/Users/fengs/Desktop/pd_real/python-course-for-ex
          r'/3.Pivot Table/supermarket_sales.xlsx')
df_sales.head(3)
```

Out[72]:

	Invoice ID	Branch	City	Customer type	Gender	...	Payment	cogs	gross margin percentage	gross income
0	750-67-8428	A	Yangon	Member	Female	...	Ewallet	522.83	4.761905	26.1415
1	226-31-3081	C	Naypyitaw	Normal	Female	...	Cash	76.40	4.761905	3.8200
2	631-41-3108	A	Yangon	Normal	Male	...	Credit card	324.31	4.761905	16.2155

3 rows × 17 columns

In [73]:
```
"""
pandas.pivot_table(data, values=None, index=None, columns=None, aggfunc='mea
                   margins=False, dropna=True, margins_name='All', observed=
"""
#Q:compare a male/female performance
df_sales.pivot_table(index='Gender',aggfunc='sum',)

"""
explaination: 1:move gender to first col;
              2:sum by gender factors:male+female
"""
```

Out[73]:

Gender	Quantity	Rating	Tax 5%	Total	Unit price	cogs	gross income	gross margin percentage
Female	2869	3489.2	7994.425	167882.925	27687.24	159888.50	7994.425	2385.714286
Male	2641	3483.5	7384.944	155083.824	27984.89	147698.88	7384.944	2376.190476

7.3 complex examples

In [74]:
```
"""make an AG+SELECT specific cols
"""
df_sales.pivot_table(index='Gender',values=['Quantity','Total']
                     ,aggfunc='sum'
                     )
```

Out[74]:

Gender	Quantity	Total
Female	2869	167882.925
Male	2641	155083.824

In [76]:
```
df_sales.pivot_table(index='Gender'
                     ,columns='Product line'
                     ,values='Quantity'
                     ,aggfunc='sum'
                     )
#value of 'Product line' becomes the rows values
#Q：compare the sale performance on diff products
```

Out[76]:

Product line	Electronic accessories	Fashion accessories	Food and beverages	Health and beauty	Home and lifestyle	Sports and travel
Gender						
Female	488	530	514	343	498	496
Male	483	372	438	511	413	424

In [ ]:
```

```

chapter 8: data visualization

In [79]:
```
import pandas as pd
import numpy  as np
df_pop=pd.read_csv(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel-
            r'\4.Data Visualization\population_total.csv')
df_pop.head(4)
```

Out[79]:

	country	year	population
0	China	2020.0	1.439324e+09
1	China	2019.0	1.433784e+09
2	China	2018.0	1.427648e+09
3	China	2017.0	1.421022e+09

Chapter 8.1DML of DF(cleansing)

In [ ]:
```
"""8.1find out NaN rows

Using isna() to select all rows with NaN under an entire DataFrame:
df[df.isna().any(axis=1)]

Using isna() to select all rows with NaN under a single DataFrame column:
df[df['column name'].isna()]

how about use loc??
ref:
df_exam.loc[df_exam['new score'] == df_exam['new score'].min()]
"""
```

In [81]:
```
#all rows contains any Nan values
df_pop[df_pop.isna().any(axis=1)]
```

Out[81]:

	country	year	population
1174	Micronesia	NaN	NaN
1175	Micronesia	NaN	NaN
1243	Micronesia	NaN	NaN
1244	Micronesia	NaN	NaN
1245	Micronesia	NaN	NaN
1310	Micronesia	NaN	NaN
1311	Micronesia	NaN	NaN

In [82]:
```
#rows that year is nan
df_pop[df_pop['year'].isna()]
```

Out[82]:

	country	year	population
1174	Micronesia	NaN	NaN
1175	Micronesia	NaN	NaN
1243	Micronesia	NaN	NaN
1244	Micronesia	NaN	NaN
1245	Micronesia	NaN	NaN
1310	Micronesia	NaN	NaN
1311	Micronesia	NaN	NaN

In [84]:
```
"""1.drop NAN rows
"""
df_pop.shape  # it is attribute,not function or method
```

Out[84]: (4185, 3)

In [85]:
```
df_pop.dropna(inplace=True)
df_pop.shape
```

Out[85]: (4178, 3)

In [89]:
```
"""2:pivot table show all values
"""
df_pop.pivot_table(index='year',columns='country',values='population',aggfun
```

Out[89]:

country	Afghanistan	Albania	Algeria	American Samoa	Andorra	...	Wallis & Futuna	Western Sahara	Yemen
year									
1955.0	8270991.0	1419994.0	9774283.0	19754.0	9232.0	...	7669.0	21147.0	4965574.0
1960.0	8996973.0	1636090.0	11057863.0	20123.0	13411.0	...	8157.0	32761.0	5315355.0
1965.0	9956320.0	1896171.0	12550885.0	23672.0	18549.0	...	8724.0	50970.0	5727751.0
1970.0	11173642.0	2150707.0	14464985.0	27363.0	24276.0	...	8853.0	76874.0	6193384.0
1975.0	12689160.0	2411732.0	16607707.0	30052.0	30705.0	...	9320.0	74954.0	6784695.0
...	...	...	...	...	...	...	...	...	..
2016.0	35383032.0	2886438.0	40551392.0	55741.0	77297.0	...	12107.0	538749.0	27168208.0
2017.0	36296113.0	2884169.0	41389189.0	55620.0	77001.0	...	11900.0	552615.0	27834819.0
2018.0	37171921.0	2882740.0	42228408.0	55465.0	77006.0	...	11661.0	567402.0	28498683.0
2019.0	38041754.0	2880917.0	43053054.0	55312.0	77142.0	...	11432.0	582463.0	29161922.0
2020.0	38928346.0	2877797.0	43851044.0	NaN	NaN	...	NaN	597339.0	29825964.0

18 rows × 234 columns

In [92]:
```
"""3.select partial col
"""
df_pivot=df_pop.pivot_table(index='year',columns='country',values='populatio


```

In [97]:
```
df_pivot[['China','India','Brazil','United States']].tail(n=5)
```

Out[97]:

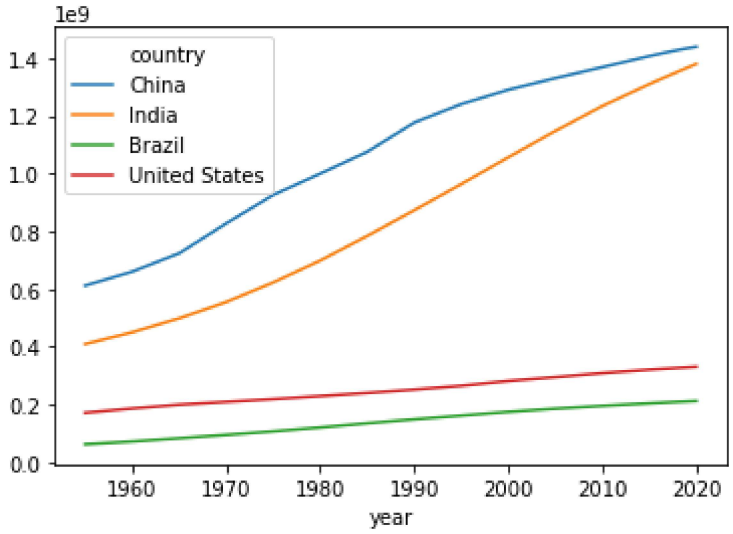
country	China	India	Brazil	United States
year				
2016.0	1.414049e+09	1.324517e+09	206163053.0	323015995.0
2017.0	1.421022e+09	1.338677e+09	207833823.0	325084756.0
2018.0	1.427648e+09	1.352642e+09	209469323.0	327096265.0
2019.0	1.433784e+09	1.366418e+09	211049527.0	329064917.0
2020.0	1.439324e+09	1.380004e+09	212559417.0	331002651.0

chapter 9: graphics

```
1 1.usually we use mathlib to draw graphics,but pd can also do some simple
  jobs
```

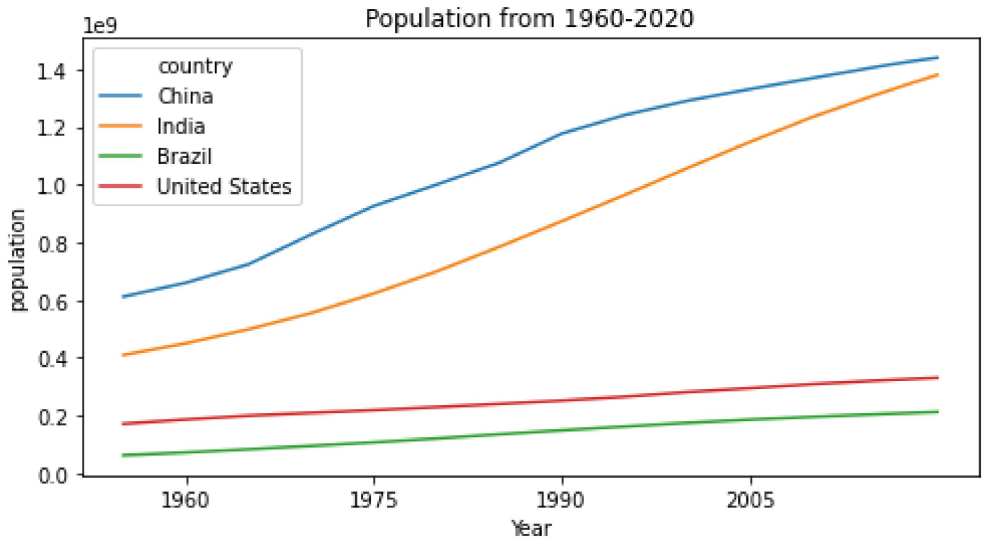
In [98]:
```
1 df_graph=df_pivot[['China','India','Brazil','United States']]
2 df_graph.plot(kind='line')
```

Out[98]: <AxesSubplot:xlabel='year'>



In [125]:
```
1 df_graph.plot(kind='line',xlabel='Year',ylabel='population'
2               ,title='Population from 1960-2020'
3               ,figsize=(8,4) #figsize=(x_size,y_size)
4               ,xticks=range(1960,2010,15) #x-axis interval
5               )
```

Out[125]: <AxesSubplot:title={'center':'Population from 1960-2020'}, xlabel='Year', ylabel='population'>



9.2 single barplot

In [107]:
```
"""select one year
1.as it has change to pivto =type,not a normal DF,
"""
df_graph.columns
```

Out[107]:
```
Index(['China', 'India', 'Brazil', 'United States'], dtype='object', name='coun
try')
```

In [108]:
```
df_graph.index
```

Out[108]:
```
Float64Index([1955.0, 1960.0, 1965.0, 1970.0, 1975.0, 1980.0, 1985.0, 1990.0,
              1995.0, 2000.0, 2005.0, 2010.0, 2015.0, 2016.0, 2017.0, 2018.0,
              2019.0, 2020.0],
             dtype='float64', name='year')
```

In [111]:
```
df_graph.loc[:-1] # not a normal DF
```

Out[111]:

country	China	India	Brazil	United States
year				

In [113]:
```
df_graph[df_graph.index.isin([2020])] #filter by index
```

Out[113]:

country	China	India	Brazil	United States
year				
2020.0	1.439324e+09	1.380004e+09	212559417.0	331002651.0

In [115]:
```
df_2020=df_graph[df_graph.index.isin([2020])]
df_2020.T #table mode
```

Out[115]:

year	2020.0
country	
China	1.439324e+09
India	1.380004e+09
Brazil	2.125594e+08
United States	3.310027e+08

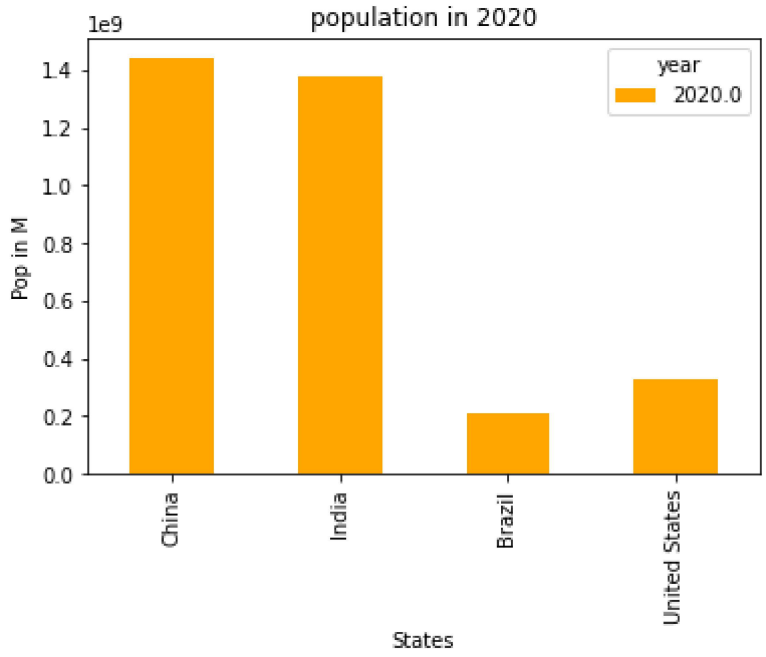
In [116]:
```
df_2020=df_graph[df_graph.index.isin([2020])].T
df_2020
```

Out[116]:

year	2020.0
country	
China	1.439324e+09
India	1.380004e+09
Brazil	2.125594e+08
United States	3.310027e+08

In [119]:
```
df_2020.plot(kind='bar',color='orange',xlabel='States',ylabel='Pop in M',
             title='population in 2020')
```

Out[119]:
```
<AxesSubplot:title={'center':'population in 2020'}, xlabel='States', ylabel='Po
p in M'>
```

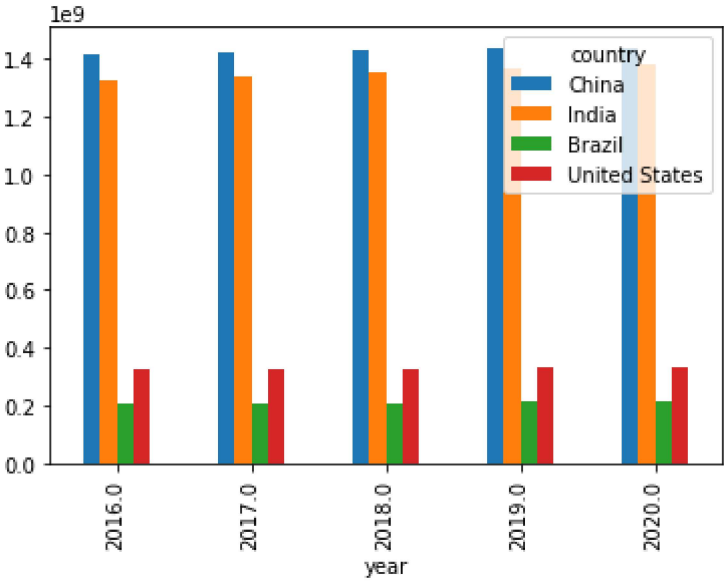


9.3 barplot grouped by 'N' variables

In [122]:
```
"""select few years not only 1 year
"""
#df_2020=df_graph[df_graph.index.isin([2020])]
#df_graph
df_year=df_graph[df_graph.index.isin([2016,2017,2018,2019,2020])]
```

In [126]:
```
#grouped barplot
df_year.plot(kind='bar')
```

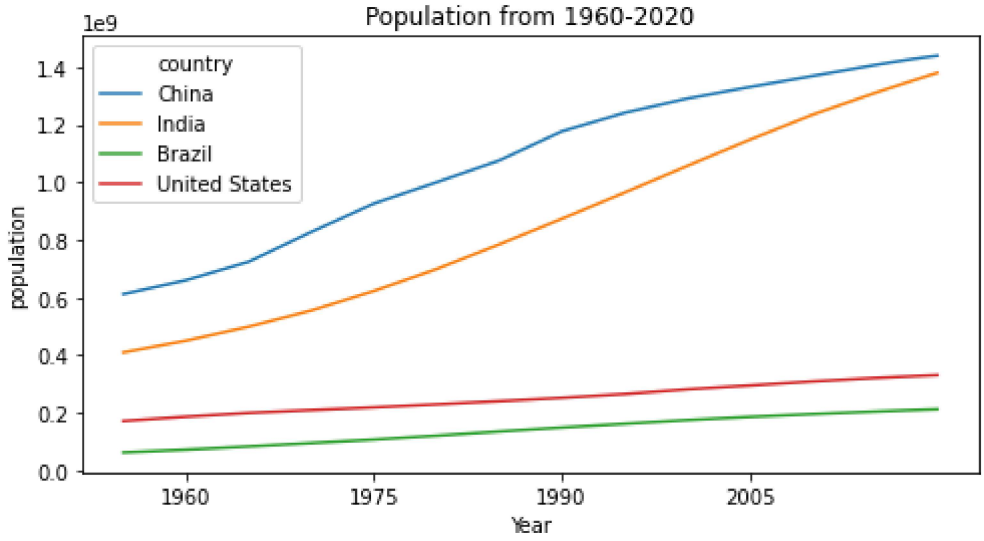
Out[126]: <AxesSubplot:xlabel='year'>



9.4 save graphics

In [127]:
```
df_graph.plot(kind='line',xlabel='Year',ylabel='population'
              ,title='Population from 1960-2020'
              ,figsize=(8,4) #figsize=(x_size,y_size)
              ,xticks=range(1960,2010,15) #x-axis interval
              )
```

Out[127]:
```
<AxesSubplot:title={'center':'Population from 1960-2020'}, xlabel='Year', ylabel='population'>
```



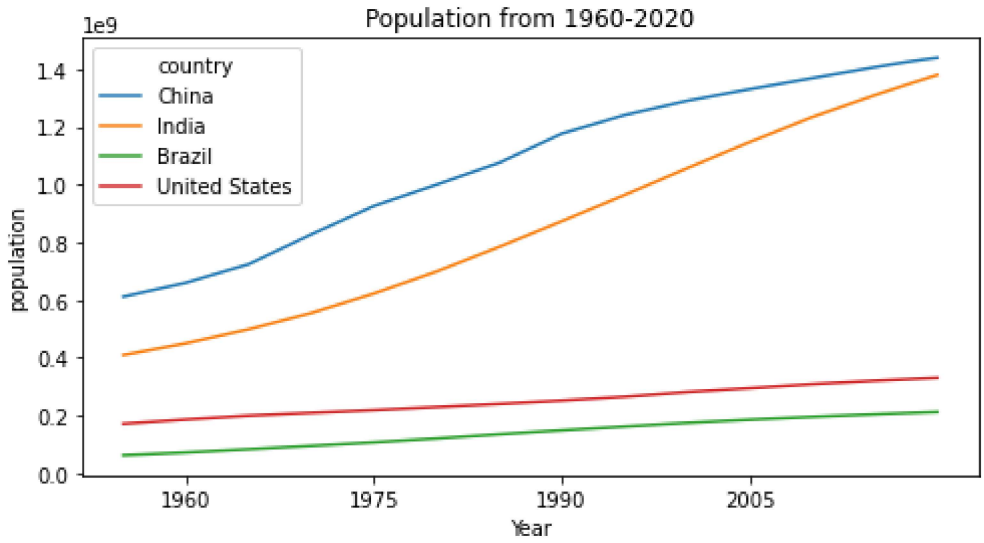
In [132]:
```
"""
import X: this imports everything as X.var1,X.var2,etc
from X import * : this imports everthing as var1,var2 etc ,i.e it floods the
"""
```

Out[132]:
```
'\nimport X: this imports everything as X.var1,X.var2,etc\nfrom X import * : this imports everthing as var1,var2 etc ,i.e it floods the local namespace\n'
```

In [131]:
```
import matplotlib.pyplot as plt
```

```
Syntax: savefig(fname, dpi=None, facecolor='w', edgecolor='w',
orientation='portrait', papertype=None, format=None, transparent=False,
bbox_inches=None, pad_inches=0.1, frameon=None, metadata=None)
```

In [139]:
```
#save as image:
#need to in a same cells
df_graph.plot(kind='line',xlabel='Year',ylabel='population'
              ,title='Population from 1960-2020'
              ,figsize=(8,4) #figsize=(x_size,y_size)
              ,xticks=range(1960,2010,15) #x-axis interval
              )
plt.savefig(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel-users-m
plt.show(
```



In [142]:
```
"""plt.show(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel-users-m
    failed
"""
img=plt.show(r'C:/Users/fengs/Desktop/pd_real/python-course-for-excel-users-
img.show()
```

```
  File "C:\Users\fengs\AppData\Local\Temp/ipykernel_11460/348674691.py", line 3
    """
       ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in positi
on 13-14: truncated \UXXXXXXXX escape
```

In [144]:
```
from PIL import Image
img = Image.open(r'C:/Users/fengs/Desktop/pd_real/python-course-for-excel-u
img.show()
```

9.5 save DF +Pivot table

In [147]:
```
df_pivot.to_excel(r'C:\Users\fengs\Desktop\pd_real\python-course-for-excel-u
```

In [ ]:
```

```