```
cars = pd.read_csv('path/x.csv',index_col=0)
# import csv nhưng chỉ rõ cột đầu là index

pd[['country']]: 2 dấu ngoặc sẽ trả về kiểu pandas
pd['country']: 1 dấu ngoặc sẽ trả về kiểu series

np.logical_and(bmi>10,bmi<15)

for key,value in dict.items():
        print (key,value)

for index,value in enumerate(list):
        print (index,value)

for val in np.nditer(array_2D_numpy):
        print val
# Liệt kê từng phần tử của mảng

for lab,row in pd.iterrows():
        pd.loc(lab,"new_column") = len(row['existing_colum'])

pd['new_column'] = pd['existing_colum'].apply(len)
cars['COUNTRY'] = cars['country'].apply(str.upper)
```

2)

sales

```
            eggs  salt  spam
state month
CA    1       47  12.0    17
      2      110  50.0    31
NY    1      221  89.0    72
      2       77  87.0    20
TX    1      132   NaN    52
      2      205  60.0    55
# Look up data for NY in month 1: NY_month1
NY_month1 = sales.loc[(['NY'],1),:]
print (NY_month1)
# Look up data for CA and TX in month 2: CA_TX_month2
CA_TX_month2 = sales.loc[(['CA','TX'],2),:]
print(CA_TX_month2)
```

```
# Look up data for all states in month 2: all_month2
all_month2 = sales.loc[(slice(None),2),:]
print (all_month2)
```

```
             eggs  salt  spam
state month
NY    1       221  89.0    72
```

```
           eggs  salt  spam
state month
CA    2       110  50.0    31
TX    2       205  60.0    55
```

```
           eggs  salt  spam
state month
CA    2       110  50.0    31
NY    2        77  87.0    20
TX    2       205  60.0    55
```

Chapter3:

1)

```
users[0:2]
  weekday    city  visitors  signups
0     Sun  Austin       139        7

# Pivot the users DataFrame: visitors_pivot
visitors_pivot =
users.pivot(index="weekday",columns="city",values="visitors")

# Print the pivoted DataFrame
print(visitors_pivot)
   city     Austin  Dallas
   weekday
   Mon         326     456
   Sun         139     237
```

2)

```
# Unstack users by 'weekday': byweekday
byweekday = users.unstack(level='weekday')

# Print the byweekday DataFrame
print(byweekday)
          visitors      signups
   weekday      Mon  Sun     Mon Sun
   city
   Austin       326  139       3   7
   Dallas       456  237       5  12
# Stack byweekday by 'weekday' and print it
print(byweekday.stack(level='weekday'))
                     visitors  signups
   city      weekday
   Austin Mon           326        3
             Sun           139        7
   Dallas Mon           456        5
             Sun           237       12


# Stack 'city' back into the index of bycity: newusers
newusers = bycity.stack(level="city")

# Swap the levels of the index of newusers: newusers
newusers = newusers.swaplevel(0,1)
```

```
# Print newusers and verify that the index is not sorted
print(newusers)

# Sort the index of newusers: newusers
newusers = newusers.sort_index()

# Print newusers and verify that the index is now sorted
print(newusers)

# Verify that the new DataFrame is equal to the original
print(newusers.equals(users))

# Pivot users with signups indexed by weekday and city: signups_pivot
signups_pivot = users.pivot(index='weekday',columns='city',values='signups')

# Print signups_pivot
print(signups_pivot)
city         Austin    Dallas
weekday
Mon          3         5
Sun          7         12

# Pivot users pivoted by both signups and visitors: pivot
pivot = users.pivot(index='weekday',columns='city')

# Print the pivoted DataFrame
print(pivot)
visitors        signups
   city      Austin Dallas  Austin Dallas
   weekday
   Mon          326    456       3      5
   Sun          139    237       7     12
```

3)

```
# Reset the index: visitors_by_city_weekday
visitors_by_city_weekday
```

```
city     Austin  Dallas
weekday
Mon         326     456
Sun         139     237
```

```
visitors_by_city_weekday = visitors_by_city_weekday.reset_index()

# Print visitors_by_city_weekday
print(visitors_by_city_weekday)
```

```
  city weekday  Austin  Dallas
  0       Mon     326     456
  1       Sun     139     237
```

```
# Melt visitors_by_city_weekday: visitors
# Melt: giữ lại 1 cột, biến các cột khác từ xoay ngang thành xoay dọc. Khi đó sẽ cần thêm 1 cái name để lưu giá trị của các cột kia.
visitors = pd.melt(visitors_by_city_weekday, id_vars="weekday",
value_name="visitors")

# Print visitors
print(visitors)
```

```
    weekday    city  visitors
  0     Mon  Austin       326
  1     Sun  Austin       139
  2     Mon  Dallas       456
  3     Sun  Dallas       237
```

4)

```
# Melt users: skinny
users
```

```
Out[16]:
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

```
skinny =
pd.melt(users,id_vars=["weekday","city"],value_vars=["visitors","signups"])

# Print skinny
print(skinny)
```

```
    weekday    city  variable  value
  0     Sun  Austin  visitors    139
  1     Sun  Dallas  visitors    237
  2     Mon  Austin  visitors    326
  3     Mon  Dallas  visitors    456
```

5)

```
# Set the new index: users_idx
users_idx = users.set_index(['city','weekday'])
# Print the users_idx DataFrame
print(users_idx)
```

```
                  visitors  signups
city   weekday
Austin Sun         139       7
Dallas Sun         237       12
Austin Mon         326       3
Dallas Mon         456       5
```

```
# Obtain the key-value pairs: kv_pairs
kv_pairs = pd.melt(users_idx,col_level=0)

# Print the key-value pairs
print(kv_pairs)
```

```
variable  value
   0  visitors    139
   1  visitors    237
   2  visitors    326
   3  visitors    456
```

6)

```
# Create the DataFrame with the appropriate pivot table: by_city_day
users
```

```
  weekday    city  visitors  signups
0     Sun  Austin       139        7
1     Sun  Dallas       237       12
2     Mon  Austin       326        3
3     Mon  Dallas       456        5
```

```
by_city_day = users.pivot_table(index='weekday',columns='city')

# Print by_city_day
print(by_city_day)
```

```
          visitors        signups
   city      Austin Dallas  Austin Dallas
   weekday
   Mon          326    456       3      5
   Sun          139    237       7     12
```

7)

```
# Use a pivot table to display the count of each column: count_by_weekday1
count_by_weekday1 = users.pivot_table(index='weekday',aggfunc='count')

# Print count_by_weekday
print(count_by_weekday1)
```

```
         city  signups  visitors
weekday
Mon         2        2         2
Sun         2        2         2
```

```
# Replace 'aggfunc='count'' with 'aggfunc=len': count_by_weekday2
count_by_weekday2 = users.pivot_table(index='weekday',aggfunc=len)

# Verify that the same result is obtained
print('==========================================')
print(count_by_weekday1.equals(count_by_weekday2))
```

```
==========================================
True
```

8)

```
# Create the DataFrame with the appropriate pivot table: signups_and_visitors
signups_and_visitors = users.pivot_table(index='weekday',aggfunc=sum)

# Print signups_and_visitors
print(signups_and_visitors)
```

```
         signups  visitors
weekday
Mon            8       782
Sun           19       376
```

```
# Add in the margins: signups_and_visitors_total
signups_and_visitors_total
=users.pivot_table(index='weekday',aggfunc=sum,margins=True)

# Print signups_and_visitors_total
print(signups_and_visitors_total)
```

```
         signups  visitors
weekday
Mon          8.0     782.0
Sun         19.0     376.0
All         27.0    1158.0
```

Chaper 4:

1)

```
print (titanic.columns.values)
['pclass' 'survived' 'name' 'sex' 'age' 'sibsp' 'parch' 'ticket' 'fare'
 'cabin' 'embarked' 'boat' 'body' 'home.dest']

# Group titanic by 'pclass'
by_class = titanic.groupby("pclass")

# Aggregate 'survived' column of by_class by count
count_by_class = by_class['survived'].count()

# Print count_by_class
print(count_by_class)
   pclass
   1    323
   2    277
   3    709
   Name: survived, dtype: int64

# Group titanic by 'embarked' and 'pclass'
by_mult = titanic.groupby(["embarked","pclass"])

# Aggregate 'survived' column of by_mult by count
count_mult = by_mult['survived'].count()

# Print count_mult
print(count_mult)
embarked  pclass
   C        1        141
            2         28
            3        101
   Q        1          3
            2          7
            3        113
   S        1        177
            2        242
            3        495
   Name: survived, dtype: int64
```

2)

```
# Read life_fname into a DataFrame: life
life = pd.read_csv(life_fname, index_col='Country')
life
```

```
              1964    1965    1966   1967    1968    1969    1970    1971  \
Country
Afghanistan  33.639  34.152  34.662  35.17  35.674  36.172  36.663  37.143
```

```
# Read regions_fname into a DataFrame: regions
regions = pd.read_csv(regions_fname, index_col='Country')
regions
```

```
                          region
Country
Afghanistan             South Asia
Albania      Europe & Central Asia
# Group life by regions['region']: life_by_region
life_by_region = life.groupby(regions['region'])

# Print the mean over the '2010' column of life_by_region
print(life_by_region["2010"].mean())
```

```
region
America                     74.037350
East Asia & Pacific         73.405750
Europe & Central Asia       75.656387
Middle East & North Africa  72.805333
South Asia                  68.189750
Sub-Saharan Africa          57.575080
Name: 2010, dtype: float64
```

3)

```
# Group titanic by 'pclass': by_class
by_class = titanic.groupby("pclass")

# Select 'age' and 'fare'
by_class_sub = by_class[['age','fare']]

# Aggregate by_class_sub by 'max' and 'median': aggregated
aggregated = by_class_sub.agg(['max','median'])
aggregated
```

	age		fare	
	max	median	max	median
pclass				
1	80.0	39.0	512.3292	60.0000
2	70.0	29.0	73.5000	15.0458
3	74.0	24.0	69.5500	8.0500

```
# Print the maximum age in each class
print(aggregated.loc[:, ('age','max')])
```

```
pclass
1    80.0
2    70.0
3    74.0
Name: (age, max), dtype: float64
```

```
# Print the median fare in each class
print(aggregated.loc[:,('fare','median')])
```

```
pclass
1    60.0000
2    15.0458
3     8.0500
Name: (fare, median), dtype: float64
```

4)

```
# Read the CSV file into a DataFrame and sort the index: gapminder
gapminder =
pd.read_csv('gapminder.csv',index_col=['Year','region','Country']).sort_index()

# Group gapminder by 'Year' and 'region': by_year_region
print(gapminder[0:1])
                                    fertility   life        population  \
  Year region  Country
  1964 America Antigua and Barbuda      4.25      63.775     58653.0
                                    child_mortality    gdp
  Year region  Country
  1964 America Antigua and Barbuda      72.78              5008.0
by_year_region = gapminder.groupby(level=['Year','region'])

# Define the function to compute spread: spread
def spread(series):
    return series.max() - series.min()

# Create the dictionary: aggregator
aggregator = {'population':'sum', 'child_mortality':'mean', 'gdp':spread}

# Aggregate by_year_region using the dictionary: aggregated
aggregated = by_year_region.agg(aggregator)

# Print the last 6 entries of aggregated
print(aggregated.tail(6))
                                    child_mortality      gdp          population
  Year region
  2013 America                        17.745833      49634.0       9.629087e+08
       East Asia & Pacific            22.285714      134744.0      2.244209e+09
       Europe & Central Asia          9.831875       86418.0       8.968788e+08
       Middle East & North Africa     20.221500      128676.0      4.030504e+08
       South Asia                     46.287500      11469.0       1.701241e+09
       Sub-Saharan Africa             76.944490      32035.0       9.205996e+08
```

5)

```
# Read file: sales
sales = pd.read_csv('sales.csv',index_col='Date',parse_dates=True)
print (sales[0:2])
                     Company    Product      Units
Date
2015-02-02 08:30:00    Hooli        Software    3
2015-02-02 21:00:00  Mediacore      Hardware    9

# Create a groupby object: by_day
```

```
# .strftime('%a') to transform the index datetime values to abbreviated days of the week.

by_day = sales.groupby(sales.index.strftime('%a'))

# Create sum: units_sum
units_sum = by_day.sum()

# Print units_sum
print(units_sum)
```
```
     Units
Mon     48
Sat      7
Thu     59
Tue     13
Wed     48
```

6)

```
# Import zscore
from scipy.stats import zscore

# Group gapminder_2010: standardized
standardized =
gapminder_2010.groupby('region')['life','fertility'].transform(zscore)

# Construct a Boolean Series to identify outliers: outliers
print (standardized[0:1])
```
```
                life  fertility
Country
Afghanistan -1.743601   2.504732
```
```
outliers = (standardized['life'] < -3) | (standardized['fertility']>3)

# Filter gapminder_2010 by the outliers: gm_outliers
gm_outliers = gapminder_2010.loc[outliers]
```

```
# Print gm_outliers
print(gm_outliers)
```

```
fertility    life  population  child_mortality     gdp  \
Country
Guatemala        3.974  71.100  14388929.0           34.5  6849.0
Haiti            3.350  45.000   9993247.0          208.8  1518.0
Tajikistan       3.780  66.830   6878637.0           52.6  2110.0
Timor-Leste      6.237  65.952   1124355.0           63.8  1777.0

                      region
Country
Guatemala               America
Haiti                   America
Tajikistan   Europe & Central Asia
Timor-Leste    East Asia & Pacific
```

7)

Nhóm những ông cùng giới tính (sex) và pclass giống nhau vào cùng 1 group. Trong group đó, nếu ông nào chưa có tuổi thì điền median của group đó vào.

```
# Create a groupby object: by_sex_class
by_sex_class = titanic.groupby(['sex','pclass'])
titanic.tail(10)
```

```
     pclass  survived                                name     sex   age  \
1299       3         0                   Yasbeck, Mr. Antoni    male  27.0
1300       3         1  Yasbeck, Mrs. Antoni (Selini Alexander)  female  15.0
1301       3         0                   Youseff, Mr. Gerious    male  45.5
1302       3         0                    Yousif, Mr. Wazli    male   NaN
1303       3         0                   Yousseff, Mr. Gerious    male   NaN
```

```
# Write a function that imputes median
def impute_median(series):
    return series.fillna(series.median())

# Impute age and assign to titanic['age']
titanic.age = by_sex_class['age'].transform(impute_median)

# Print the output of titanic.tail(10)
print(titanic.tail(10))
```

```
     pclass  survived                                name     sex   age  \
1299       3         0                   Yasbeck, Mr. Antoni    male  27.0
1300       3         1  Yasbeck, Mrs. Antoni (Selini Alexander)  female  15.0
1301       3         0                   Youseff, Mr. Gerious    male  45.5
1302       3         0                    Yousif, Mr. Wazli    male  25
1303       3         0                   Yousseff, Mr. Gerious    male  25
```

8)

gapminder_2010[0:1]

Country	fertility	life	population	child_mortality	gdp	region
Afghanistan	5.659	59.612	31411743.0	105.0	1637.0	South Asia

Mục tiêu: Cho biết spread gdp của từng region, và zcore gdp của từng quốc gia trong region đó (aggregate spread of per capita GDP in each region and the individual country's z-score of the regional per capita GDP.)

```
def disparity(gr):
    # Compute the spread of gr['gdp']: s
    s = gr['gdp'].max() - gr['gdp'].min()
    # Compute the z-score of gr['gdp'] as (gr['gdp']-
gr['gdp'].mean())/gr['gdp'].std(): z
    z = (gr['gdp'] - gr['gdp'].mean())/gr['gdp'].std()
    # Return a DataFrame with the inputs {'z(gdp)':z, 'regional
spread(gdp)':s}
    return pd.DataFrame({'z(gdp)':z , 'regional spread(gdp)':s})
```

```
# Group gapminder_2010 by 'region': regional
regional = gapminder_2010.groupby('region')

# Apply the disparity function on regional: reg_disp
reg_disp = regional.apply(disparity)

# Print the disparity of 'United States', 'United Kingdom', and 'China'
print(reg_disp.loc[['United States','United Kingdom','China']])
```

Country	regional spread(gdp)	z(gdp)
United States	47855.0	3.013374
United Kingdom	89037.0	0.572873
China	96993.0	-0.432756

9)

```
def c_deck_survival(gr):

    c_passengers = gr['cabin'].str.startswith('C').fillna(False)

    return gr.loc[c_passengers, 'survived'].mean()
```

Take the Titanic data set and analyze survival rates from the 'C' deck, which contained the most passengers

```
# Create a groupby object using titanic over the 'sex' column: by_sex
by_sex = titanic.groupby('sex')

# Call by_sex.apply with the function c_deck_survival and print the result
c_surv_by_sex = by_sex.apply(c_deck_survival)
```

```
# Print the survival rates
print(c_surv_by_sex)
```

```
sex
female    0.913043
male      0.312500
dtype: float64
```

10)

Take the February sales data and remove entries from companies that purchased less than 35 Units in the whole month.

```
# Read the CSV file into a DataFrame: sales
sales = pd.read_csv('sales.csv', index_col='Date', parse_dates=True)
print (sales[0:1])
```

```
                     Company   Product  Units
Date
2015-02-02 08:30:00     Hooli  Software      3
2015-02-02 21:00:00  Mediacore  Hardware      9
```

```
# Group sales by 'Company': by_company
by_company = sales.groupby('Company')

# Compute the sum of the 'Units' of by_company: by_com_sum
by_com_sum = by_company['Units'].sum()
print(by_com_sum)
```

```
Company
Acme Coporation    34
Hooli              30
Initech            30
Mediacore          45
Streeplex          36
Name: Units, dtype: int64
```

```
# Filter 'Units' where the sum is > 35: by_com_filt
by_com_filt = by_company.filter(lambda g:g['Units'].sum()>35)
print(by_com_filt)
```

```
                     Company   Product  Units
Date
2015-02-02 21:00:00  Mediacore  Hardware      9
2015-02-04 15:30:00  Streeplex  Software     13
2015-02-09 09:00:00  Streeplex   Service     19
```

11)

The key here is that the Series is indexed the same way as the DataFrame, can also mix and match column grouping with Series grouping.

Mục tiêu: Investigate survival rates of passengers on the Titanic by 'age' and 'pclass'. In particular, the goal is to find out what fraction of children under 10 survived in each 'pclass'.

```
# Create the Boolean Series: under10
under10 = (titanic['age']<10).map({True:'under 10',False:'over 10'})

# Group by under10 and compute the survival rate
survived_mean_1 = titanic.groupby(under10)['survived'].mean()
print(survived_mean_1)
```

```
age
over 10     0.366748
under 10    0.609756
Name: survived, dtype: float64
```

```
# Group by under10 and pclass and compute the survival rate
survived_mean_2 = titanic.groupby([under10,'pclass'])['survived'].mean()
print(survived_mean_2)
```

```
age       pclass
over 10   1        0.617555
          2        0.380392
          3        0.238897
under 10  1        0.750000
          2        1.000000
          3        0.446429
Name: survived, dtype: float64
```

Chapter 5

1)

```
In [1]: medals[0:1]
Out[1]:
     City  Edition     Sport Discipline        Athlete  NOC Gender  \
0  Athens     1896  Aquatics   Swimming  HAJOS, Alfred  HUN    Men

           Event Event_gender Medal
0  100m freestyle           M  Gold
```

Suppose you have loaded the data into a DataFrame medals. You now want to find the total number of medals awarded to the USA per edition. To do this, filter the 'USA' rows and use the groupby() function to put the 'Edition'column on the index:

```
USA_edition_grouped = medals.loc[medals.NOC == 'USA'].groupby('Edition')
USA_edition_grouped['Medal'].count()
```

```
Edition
1896     20
1900     55
1904    394
1908     63
```

2)

Obj: Use the pandas Series method .value_counts() to determine the top 15 countries ranked by total number of medals.

```
# Select the 'NOC' column of medals: country_names
country_names = medals['NOC']

# Count the number of medals won by each country: medal_counts
medal_counts = country_names.value_counts()

# Print top 15 countries ranked by medals
print(medal_counts.head(15))
```

```
USA    4335
   URS    2049
   GBR    1594
   FRA    1314
   ITA    1228
   GER    1211
   AUS    1075
   HUN    1053
   SWE    1021
Name: NOC, dtype: int64
```

3)

```
# Construct the pivot table: counted
# Index theo quốc gia, chỉ quan tâm vào cột Medal, đếm xem mỗi loại trong Medal có bao nhiêu cái.
counted = medals.pivot_table(index='NOC',columns='Medal',values='Athlete',aggfunc='count')


# Create the new column: counted['totals']
# Tạo cột total có gía trị bằng tổng các cột còn lại
counted['totals'] = counted.sum(axis='columns')

# Sort counted by the 'totals' column
counted = counted.sort_values('totals', ascending=False)

# Print the top 15 rows of counted
print(counted.head(15))
```

```
Medal  Bronze    Gold  Silver  totals
   NOC
   USA    1052.0  2088.0  1195.0  4335.0
   URS     584.0   838.0   627.0  2049.0
   GBR     505.0   498.0   591.0  1594.0
```

4)

```
# Select columns: ev_gen
ev_gen = medals[['Event_gender','Gender']]
# Drop duplicate pairs: ev_gen_uniques
ev_gen_uniques = ev_gen.drop_duplicates()

# Print ev_gen_uniques
print(ev_gen_uniques)
```

```
     Event_gender Gender
0               M    Men
348             X    Men
416             W  Women
639             X  Women
23675           W    Men
```

5)

```
# Group medals by the two columns: medals_by_gender
medals_by_gender = medals.groupby(['Event_gender','Gender'])

# Create a DataFrame with a group count: medal_count_by_gender
medal_count_by_gender = medals_by_gender.count()

# Print medal_count_by_gender
print(medal_count_by_gender)
```

```
                      Medal
Event_gender Gender
M            Men      20067
W            Men          1
             Women     7277
X            Men       1653
             Women      218
```

=> Có 1 ông là Men mà lại có event_gender là Woman

6)

```
# Create the Boolean Series: sus
sus = (medals.Event_gender=='W')&(medals.Gender=='Men')

# Create a DataFrame with the suspicious row: suspect
suspect = medals[sus]

# Print suspect
print(suspect)
```

```
          City  Edition      Sport Discipline           Athlete  NOC Gender  \
23675  Sydney     2000  Athletics  Athletics  CHEPCHUMBA, Joyce  KEN    Men

           Event Event_gender   Medal
23675  marathon            W  Bronze
```

7)

Compute the number of distinct sports in which each country won medals

```
# Group medals by 'NOC': country_grouped
country_grouped = medals.groupby('NOC')

# Compute the number of distinct sports in which each country won medals:
Nsports
Nsports = country_grouped['Sport'].nunique()

# Sort the values of Nsports in descending order
```

```
Nsports = Nsports.sort_values(ascending=False)

# Print the top 15 rows of Nsports
print(Nsports.head(15))
NOC
  USA   34
  GBR   31
  FRA   28
  GER   26
  CHN   24
```

8)

Aggregate the number of distinct sports in which the USA and the USSR won medals during the Cold War years.

```
# Extract all rows for which the 'Edition' is between 1952 & 1988:
during_cold_war
during_cold_war = (medals.Edition >=1952)&(medals.Edition<=1988)

# Extract rows for which 'NOC' is either 'USA' or 'URS': is_usa_urs
is_usa_urs = medals.NOC.isin(['USA','URS'])

# Use during_cold_war and is_usa_urs to create the DataFrame:
cold_war_medals
cold_war_medals = medals.loc[during_cold_war & is_usa_urs]

# Group cold_war_medals by 'NOC'
country_grouped = cold_war_medals.groupby('NOC')

# Create Nsports
Nsports = country_grouped['Sport'].nunique().sort_values(ascending=False)

# Print Nsports
print(Nsports)
  NOC
  URS   21
  USA   20
  Name: Sport, dtype: int64
```

9)

?: which country, the USA or the USSR, won the most medals consistently over the Cold War period.

```
# Create the pivot table: medals_won_by_country
```

```
medals_won_by_country =
medals.pivot_table(index='Edition',columns='NOC',values='Athlete',aggfunc='c
ount')

# Slice medals_won_by_country: cold_war_usa_usr_medals
cold_war_usa_usr_medals = medals_won_by_country.loc[1952:1988,
['USA','URS']]
NOC       USA    URS
Edition
1952     130.0  117.0
1956     118.0  169.0
1960     112.0  169.0
1964     150.0  174.0
1968     149.0  188.0
1972     155.0  211.0
1976     155.0  285.0
1980       NaN  442.0
1984     333.0    NaN
1988     193.0  294.0
URS    8
USA    2
dtype: int64
# Create most_medals
most_medals = cold_war_usa_usr_medals.idxmax(axis='columns')
print (most_medals)
   Edition
   1952    USA
   1956    URS
   1960    URS
   1964    URS
   1968    URS
   1972    URS
   1976    URS
   1980    URS
   1984    USA
   1988    URS
   dtype: object

# Print most_medals.value_counts()
print(most_medals.value_counts())

URS    8
USA    2
dtype: int64
```

10)

```
# Create the DataFrame: usa
usa = medals[medals.NOC=="USA"]
```

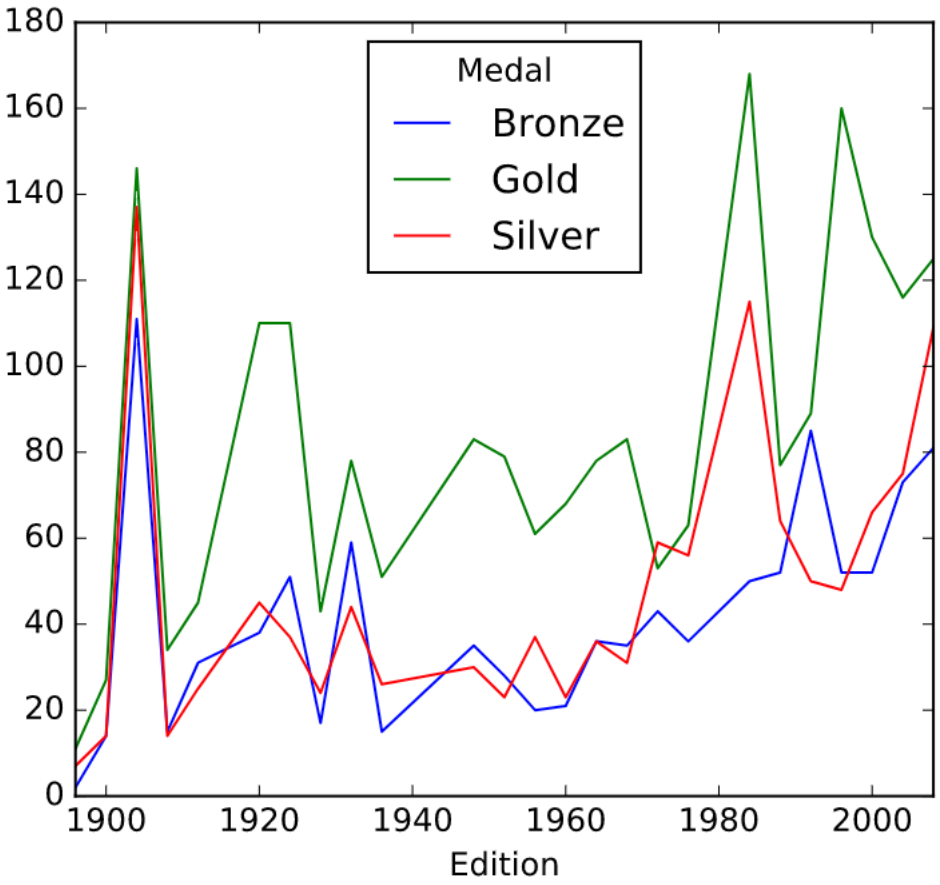
```
# Group usa by ['Edition', 'Medal'] and aggregate over 'Athlete'
usa_medals_by_year = usa.groupby(['Edition','Medal'])['Athlete'].count()
print (usa_medals_by_year[0:3])
```

```
Edition  Medal
1896     Bronze      2
         Gold       11
         Silver      7
   Name: Athlete, dtype: int64
```

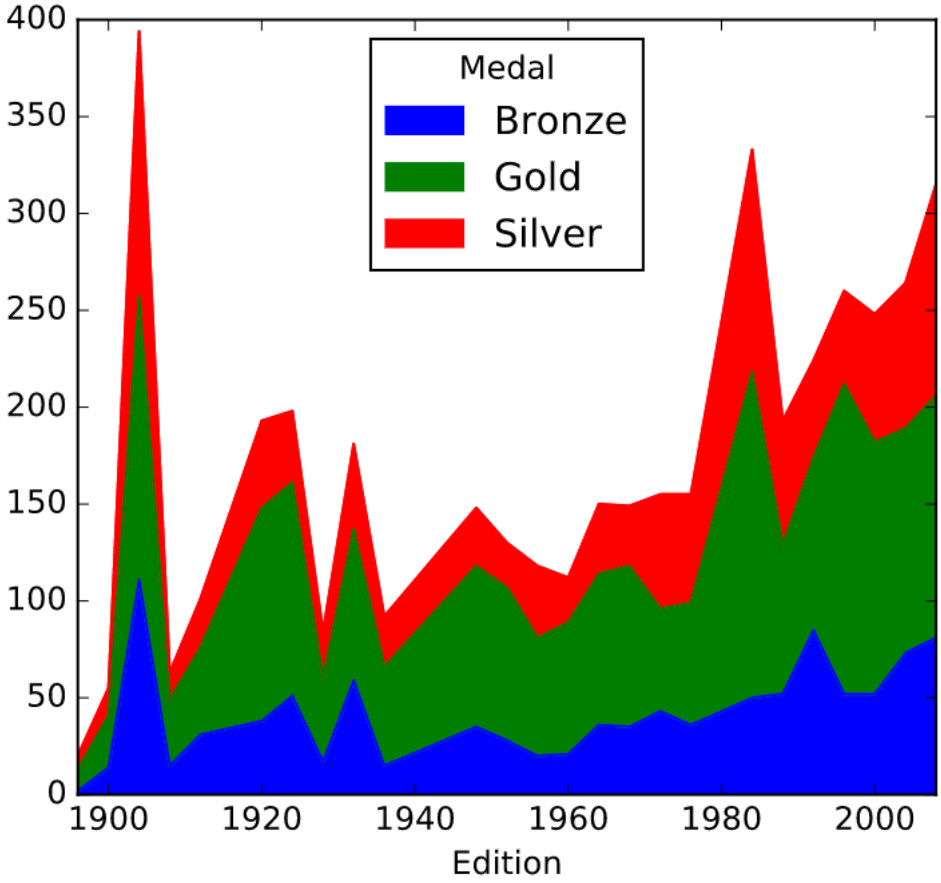
```
# Reshape usa_medals_by_year by unstacking
usa_medals_by_year = usa_medals_by_year.unstack(level='Medal')
print (usa_medals_by_year[0:3])
```

```
Medal    Bronze  Gold  Silver
Edition
1896          2    11       7
1900         14    27      14
1904        111   146     137
```

```
# Plot the DataFrame usa_medals_by_year
usa_medals_by_year.plot()
plt.show()
```



```
usa_medals_by_year.plot.area()
plt.show()
```




```
# Redefine 'Medal' as an ordered categorical
medals.Medal =
pd.Categorical(values=medals.Medal,categories=['Bronze','Silver','Gold'],ordere
d=True)

medals.info()
Event_gender    29216 non-null object
Medal           29216 non-null category
dtypes: category(1), int64(1), object(8)
memory usage: 2.0+ MB
```