DATA SELECTION

In [8]:
```
import pandas as pn
import numpy as np
```

In [9]:
```
dates= pn.date_range('1/1/2000', periods=8)
df = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])
df

colB=df.B.values
colC=df.C.values
colD=df.D.values
boolean=[]

print(df)
df.loc[(df.B>df.A) & (df.B<df.C),'B']
```

```
                   A         B         C         D
2000-01-01 -0.000783 -0.666538  0.641780  1.889946
2000-01-02  0.203998  1.565900 -1.576848  1.008718
2000-01-03 -0.442027 -1.422871  0.393688 -0.793887
2000-01-04 -0.483422 -0.564830  0.065075  0.508323
2000-01-05  1.954107 -0.550484  0.811359  0.496025
2000-01-06 -1.280861  0.669350  0.179829 -0.402134
2000-01-07 -2.173499  1.444701 -2.074414 -0.698938
2000-01-08  1.365486 -0.567001  0.488785 -0.775794
```

Out[9]:
```
Series([], Freq: D, Name: B, dtype: float64)
```

Llamar a un valor de una dataframe

Podemos seleccionar o modificar datos de distintas formas, es decir, para llamar a un elemento concreto de un data frame o series tenemos distintos elementos. Ahora nos centramos en los dataframes, pero para las series se hace de la misma forma:

a) Llamamos a una columna y después a un índice

Por ejemplo si queremos el sexto elemento de la columna A:

In [10]:
```
df['A'][dates[5]]
```

Out[10]:
```
-1.2808605940399247
```

b) Llamamos al índice y después a la columna con .loc[]

In [11]:
```
df.loc[dates[5],'A']
```

Out[11]:
```
-1.2808605940399247
```

EJEMPLO: Cambiar el orden de 2 columnas

Para ello simplemente llamamos a 2 columnas, y las igualamos a las columnas al revés. Es decir:

In [12]:
```
df[['A','B']]=df[['B','A']]
df
```

Out[12]:

	A	B	C	D
2000-01-01	-0.666538	-0.000783	0.641780	1.889946
2000-01-02	1.565900	0.203998	-1.576848	1.008718
2000-01-03	-1.422871	-0.442027	0.393688	-0.793887
2000-01-04	-0.564830	-0.483422	0.065075	0.508323
2000-01-05	-0.550484	1.954107	0.811359	0.496025
2000-01-06	0.669350	-1.280861	0.179829	-0.402134
2000-01-07	1.444701	-2.173499	-2.074414	-0.698938
2000-01-08	-0.567001	1.365486	0.488785	-0.775794

Llamada por índice, .loc[ ] y .iloc[ ]

Básicamente hacen lo mismo pero .loc[ ] únicamente funciona con el nombre del índice mientras que .iloc[ ] solo funciona con la posición en número:

In [13]:
```
from IPython.display import Image
Image("loc_iloc.png")
```

Out[13]:

Input	.loc	.iloc
Single value	5 or 'a'	5
List of values	['a', 'b', 'c']	[4, 3, 0]
Slice object	'a':'f'	1:7
Boolean array	[True, False, ...]	[True, False, ...]
Callable object	Function	Function

Llamar elementos a partir de booleanos

Básicamente, metes la condicion dentro del corchete y lo que de True, es decir, cuando se cumpla la condición, se selecciona el valor correspondiente. Por ejemplo, si hacemos un dataframe.loc[dataframe.columna>0], lo que hace es comparar todos los elementos de la columna a 0, dando por ejemplo [True,True,False,true]. Esto hará que se impriman la primera, segunda y cuarta fila por parte del .loc

In [14]:
```
df.loc[df.A > 0]
```

Out[14]:

	A	B	C	D
2000-01-02	1.565900	0.203998	-1.576848	1.008718
2000-01-06	0.669350	-1.280861	0.179829	-0.402134
2000-01-07	1.444701	-2.173499	-2.074414	-0.698938

IMPOTANTE: Tienes que usar dataframe.columna, no vale poner directamente el nombre de la columna, porque >, < o = no pueden ir entre un string y un int.

Filtrar números de una tabla, poniendo NaN a los que no cumplan la condicion

El comando es **dataframe[dataframe > 34]**, lo que hace la condicion del corchete es crear un dataframe de booleanos, y después los valores True conservan su valor mientras que los False pasan a perder el valor(NaN).

Por ejemplo, filtar del dataframe los valores mayores de 0:

In [15]:
```
df[df >0]
```

Out[15]:

	A	B	C	D
2000-01-01	NaN	NaN	0.641780	1.889946
2000-01-02	1.565900	0.203998	NaN	1.008718
2000-01-03	NaN	NaN	0.393688	NaN
2000-01-04	NaN	NaN	0.065075	0.508323
2000-01-05	NaN	1.954107	0.811359	0.496025
2000-01-06	0.669350	NaN	0.179829	NaN
2000-01-07	1.444701	NaN	NaN	NaN
2000-01-08	NaN	1.365486	0.488785	NaN

Dar la vuelta a las columnas y a las filas

In [16]:
```
df_ej5 = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])

df_ej5.loc[::-1] #Esto de la vuelta a las columnas

df.loc[::-1] #Esto da la vuelta a las filas

df.loc[::-1].reset_index(drop = True) #Revierte las filas pero no los índices
```

Out[16]:

	A	B	C	D
0	-0.567001	1.365486	0.488785	-0.775794
1	1.444701	-2.173499	-2.074414	-0.698938
2	0.669350	-1.280861	0.179829	-0.402134
3	-0.550484	1.954107	0.811359	0.496025
4	-0.564830	-0.483422	0.065075	0.508323
5	-1.422871	-0.442027	0.393688	-0.793887
6	1.565900	0.203998	-1.576848	1.008718
7	-0.666538	-0.000783	0.641780	1.889946

Mostrar los valores de la columna B que son más grandes que la columna A y más pequeños que la columna C

In [17]:
```
dates= pn.date_range('1/1/2000', periods=8)
df = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])

print(df)
df.loc[(df.B>df.A) & (df.B<df.C),'B'] #Si quieres poner varias condiciones
                                       #tienes que poner cada una entre
                                       #paréntesis y el and es &, el or su-
                                       #pongo que será | pero no sé
```

```
                   A         B         C         D
2000-01-01 -1.251315  1.027612 -0.278824 -0.329489
2000-01-02  0.208602  1.019449  0.916072 -0.114387
2000-01-03 -0.484974  1.807709 -0.181017  0.591778
2000-01-04 -1.249420  0.852016 -0.123124  0.344837
2000-01-05 -1.315916 -0.477611 -0.173063 -0.080366
2000-01-06 -0.776168 -1.187224  0.489135  1.172228
2000-01-07  0.190130 -1.369850 -1.268442  0.727483
2000-01-08 -0.529417  0.210373 -1.742735  1.469252
```

Out[17]:
```
2000-01-05   -0.477611
Freq: D, Name: B, dtype: float64
```

SELECTION BY DF.QUERY(CONDITION) (muy util)

dataframe.query('condicion') es una herramienta muy útil que nos muestra los valores que cumplen la condición que se indique, simpre comparando por columnas. Es decir, si ponemos **df.query('(a>b)')** nos mostrará por filas los valores de la columna a que sean mayores que b, es el equivalente a df.loc[df.A>df.B,:]

DATA MANIPULATION IN PANDAS

RENAMING

dataframe.rename( ) permite renombrar índices o columnas, y ello se hace a través de un diccionario. Es decir, si queremos por ejemplo renombrar las columnas de la tabla df:

In [18]:
```
df.rename(columns={'A':'One','B':'Two','C':'Three','D':'Four'})
```

Out[18]:

	One	Two	Three	Four
2000-01-01	-1.251315	1.027612	-0.278824	-0.329489
2000-01-02	0.208602	1.019449	0.916072	-0.114387
2000-01-03	-0.484974	1.807709	-0.181017	0.591778
2000-01-04	-1.249420	0.852016	-0.123124	0.344837
2000-01-05	-1.315916	-0.477611	-0.173063	-0.080366
2000-01-06	-0.776168	-1.187224	0.489135	1.172228
2000-01-07	0.190130	-1.369850	-1.268442	0.727483
2000-01-08	-0.529417	0.210373	-1.742735	1.469252

Ahora hacemos lo mismo para los índices:

In [19]:
```
df = pn.DataFrame(np.random.randn(8,4), index=[1,2,3,4,5,6,7,8], columns=['A','B','C','D'])

df.rename(index={1:'a',2:'b',3:'c',4:'d',5:'e',6:'f',7:'g',8:'h'})
```

Out[19]:

	A	B	C	D
a	-0.940880	0.904670	-0.880998	-0.294990
b	-1.549842	0.041309	0.317861	-0.750837
c	-0.082279	0.053818	-0.063780	1.463687
d	-0.372310	0.248445	1.352968	1.341535
e	-0.981832	0.085458	2.677317	0.209802
f	-0.025465	1.134713	-0.337710	1.597972
g	0.197687	-0.172664	1.296326	0.318335
h	-1.057785	0.802463	-0.018280	-0.032496

REINDEXING

dataframe.reindex([nuevo_indice_1,nuevo_indice_2...]) Permite cambiar o añadir nuevos índices, ESA ES LA DIFERENCIA RESPECTO DE DF.RENAME(), QUE PUEDES AÑADIR NUEVOS ÍNDICES EN LA TUPLA Y ÉSTOS SE AÑADIRÁN EN ORDEN DE APARICIÓN. También funciona para columnas, pero es necesario especificarlo con **dataframe.reindex(columns=[nueva_columna_1,nueva_columna_2...])**

In [20]:
```
a=df.reindex([1,2,3,4,5,6,7,8,9,10])
a
```

Out[20]:

	A	B	C	D
1	-0.940880	0.904670	-0.880998	-0.294990
2	-1.549842	0.041309	0.317861	-0.750837
3	-0.082279	0.053818	-0.063780	1.463687
4	-0.372310	0.248445	1.352968	1.341535
5	-0.981832	0.085458	2.677317	0.209802
6	-0.025465	1.134713	-0.337710	1.597972
7	0.197687	-0.172664	1.296326	0.318335
8	-1.057785	0.802463	-0.018280	-0.032496
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN

DROPPING ENTRIES

.drop(['índe_de_fila_a_elimar_1, indice_de_fila_a_eliminar_2...]) / .drop([titulo_columna_a_eliminar_1, titulo_columna_a_eliminar_2], axis=1)

Partiendo de la siguiente tabla:

In [21]:
```
df_drop=pn.DataFrame(np.arange(16).reshape((4,4)), index=['Ohio','Colorado','Utah','New York'], columns=['one','two','three','four'])
df_drop
```

Out[21]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Eliminamos las filas 'Ohio' y 'Colorado'

In [22]:
```
df_drop.drop(['Ohio','Colorado'])
```

Out[22]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

Eliminamos las columnas 'two' y 'four':

In [23]:
```
df_drop.drop(['two','four'],axis=1)
```

Out[23]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

ARITHEMTIC AND DATA ALIGNMENT

Si queremos hacer operaciones aritméticas sobre 2 tablas y juntarlas, por ejemplo si quieremos sumar 2 tablas, hay que tener en cuenta que si hacemos df1+df2 ambas tablas deben tener el mismo tamaño, porque los valores que no tengan una pareja tomarán el valor de NaN. Por tanto ara evitar ésto tenemos **los operados aritméticos de pandas:**

In [24]:
```
from IPython.display import Image
Image("operadores_aritmeticos.png")
```

Out[24]:

Method	Description
add	Method for addition (+)
sub	Method for subtraction (-)
div	Method for division (/)
mul	Method for multiplication (*)

In [25]:
```
df1=pn.DataFrame(np.arange(12.).reshape((3,4)),columns=list('abcd'))
df2=pn.DataFrame(np.arange(20.).reshape((4,5)),columns=list('abcde'))

print(df1.add(df2, fill_value=0))
print('')
print(df1.sub(df2, fill_value=0))
print('')
print(df1.mul(df2, fill_value=0))
print('')
print(df1.div(df2, fill_value=0))

#Es importante el fill_value=0, porque si no lo pones es como si
#hicieras por ejemplo df1+df2
```

```
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0  11.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0

      a     b     c     d     e
0   0.0   0.0   0.0   0.0  -4.0
1  -1.0  -1.0  -1.0  -1.0  -9.0
2  -2.0  -2.0  -2.0  -2.0 -14.0
3 -15.0 -16.0 -17.0 -18.0 -19.0

      a     b      c      d    e
0   0.0   1.0    4.0    9.0  0.0
1  20.0  30.0   42.0   56.0  0.0
2  80.0  99.0  120.0  143.0  0.0
3   0.0   0.0    0.0    0.0  0.0

     a         b         c         d    e
0  NaN  1.000000  1.000000  1.000000  0.0
1  0.8  0.833333  0.857143  0.875000  0.0
2  0.8  0.818182  0.833333  0.846154  0.0
3  0.0  0.000000  0.000000  0.000000  0.0
```

FUNCTION APPLICATION AND MAPPING

dataframe.apply(funcion) --> Se ejecuta la funcion entre paréntesis usando como input el dataframe especificado, PERO SE EJECUTA POR COLUMNAS

dataframe.apply(funcion, axis=1) --> Se ejecuta la funcion entre paréntesis usando como input el dataframe especificado, PERO SE EJECUTA POR FILAS

In [26]:
```
f= lambda x: x.max()-x.min() #La funcion

frame=pn.DataFrame(np.random.randn(4,3), columns=['b','d','e'], index=['Utah','Ohio','Texas','Oregon'])

print(frame.apply(f)) #Se busca el mayor valor de la columna y se le resta el
               #menor de la misma, siendo el ouput una serie con cada
               #columna

print(frame.apply(f, axis=1)) #Se busca el mayor valor de la fila y se le resta el
                     #menor de la misma, siendo el ouput una serie con cada
                     #fila
```

```
b    2.576337
d    2.809652
e    1.484490
dtype: float64
Utah      1.740991
Ohio      3.167368
Texas     0.589877
Oregon    1.845711
dtype: float64
```

SORTING AND RANKING

In [27]:
```
df5=pn.DataFrame({'a':[0,1,0,1],'b':[4,7,-3,2]},index=[0,1,2,3])
```

Sorting

In [28]:
```
df5.sort_values(by='a') #Ordenar respecto a la columna 'a'
```

Out[28]:

	a	b
0	0	4
2	0	-3
1	1	7
3	1	2

In [29]:
```
df5.sort_values(by=['a','b']) #Ordenar respecto a las columnas 'a' y 'b'
```

Out[29]:

	a	b
2	0	-3
0	0	4
3	1	2
1	1	7

Ranking

El ranking asigna un valor de 1 al número de datos no nulos en una serie/tabla. El ranking se puede aplicar a una serie o a un dataframe.

Existen distintos métodos de hacer ranking, que se marcan por .rank(method=X)

In [30]:
```
from IPython.display import Image
Image("metodos_ranking.png")
```

Out[30]:

Method	Description
'average'	Default: assign the average rank to each entry in the equal group
'min'	Use the minimum rank for the whole group
'max'	Use the maximum rank for the whole group
'first'	Assign ranks in the order the values appear in the data

In [31]:
```
obj=pn.Series([7,-5,7,4,2,0,4])

obj.rank()
```

Out[31]:
```
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

También podemos hacer un ranking de dataframes, pero éste se puede hacer por filas o por columnas. Por defecto se hará por filas, por lo que deberemos especificar con axis=1 para que se haga por columnas:

In [32]:
```
frame=pn.DataFrame({'b':[4.3,7,-3,2],'a':[0,1,0,1],'c':[-2,5,8,-2.5]})
print(frame)

print(frame.rank(method='first',axis=1)) #Ranking por columnas

print(frame.rank(method='first')) #Ranking por filas
```

```
     b  a    c
0  4.3  0 -2.0
1  7.0  1  5.0
2 -3.0  0  8.0
3  2.0  1 -2.5
     b    a    c
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
     b    a    c
0  3.0  1.0  2.0
1  4.0  3.0  3.0
2  1.0  2.0  4.0
3  2.0  4.0  1.0
```

EJERCICIOS

EJERCICIO 1: Write a Pandas program to replace all the NaN values with Zero's in a column of a dataframe.

In [63]:
```
df = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])
df_nan=df[df>0] #CREAMOS LA TABLA QUE SERVIRA DE ENUNCIADO

df_nan[df_nan.isnull()]=0
df_nan
```

Out[63]:

	A	B	C	D
2000-01-01	0.312950	0.422464	0.223764	0.000000
2000-01-02	0.000000	0.000000	0.000000	1.030326
2000-01-03	0.978438	0.402716	0.000000	0.000000
2000-01-04	0.000000	0.637970	0.535376	0.000000
2000-01-05	0.000000	0.000000	0.000000	0.664958
2000-01-06	0.102181	0.000000	1.074017	0.854179
2000-01-07	0.875921	0.000000	0.887574	1.533723
2000-01-08	0.000000	1.122834	0.247540	0.000000

EJERCICIO 2: Write a Pandas program to convert index in a column of the given dataframe.

In [27]:
```
df_ej2 = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])
print(df_ej2)



df_ej2['nueva_columna']=df_ej2.index
df_ej2
```

```
                   A         B         C         D
2000-01-01 -0.535101 -0.364561 -0.530170 -0.171163
2000-01-02 -0.680146 -0.588430 -0.364841  0.138270
2000-01-03 -0.489039 -0.165032 -1.735786 -1.570797
2000-01-04  1.303573  0.506258  1.221943  0.017820
2000-01-05 -2.071717 -0.272240 -0.364922  0.951685
2000-01-06 -1.372558 -0.763122  1.137094  1.035435
2000-01-07 -0.931777  0.199696 -0.144288  2.290485
2000-01-08 -0.784667  0.038746  1.117412  1.055123
```

Out[27]:

	A	B	C	D	nueva_columna
2000-01-01	-0.535101	-0.364561	-0.530170	-0.171163	2000-01-01
2000-01-02	-0.680146	-0.588430	-0.364841	0.138270	2000-01-02
2000-01-03	-0.489039	-0.165032	-1.735786	-1.570797	2000-01-03
2000-01-04	1.303573	0.506258	1.221943	0.017820	2000-01-04
2000-01-05	-2.071717	-0.272240	-0.364922	0.951685	2000-01-05
2000-01-06	-1.372558	-0.763122	1.137094	1.035435	2000-01-06
2000-01-07	-0.931777	0.199696	-0.144288	2.290485	2000-01-07
2000-01-08	-0.784667	0.038746	1.117412	1.055123	2000-01-08

EJERCICIO 3: Write a Pandas program to count the NaN values in one or more columns in DataFrame

In [28]:
```
df_ej3 = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])
df_ej3=df_ej3[df_ej3>0]
print(df_ej3)

sum=0
df_bool=df_ej3.isnull()
for i in df_bool.columns:
    for j in df_bool.index:
        if df_bool.loc[j,i]==True:
            sum+=1

sum
```

```
                   A         B         C         D
2000-01-01  0.138290  1.947074  1.389102       NaN
2000-01-02  0.226521  0.858733       NaN  0.739855
2000-01-03  0.039906       NaN       NaN  1.275305
2000-01-04       NaN       NaN  1.191685  0.767562
2000-01-05       NaN  0.144297  2.020633       NaN
2000-01-06       NaN  0.467210       NaN       NaN
2000-01-07  0.210746  0.600205       NaN  0.808142
2000-01-08  0.300563  0.996810       NaN  0.788788
```

Out[28]:
```
13
```

EJERCICIO 4: Write a Pandas program to add a prefix or suffix to all columns of a given DataFrame

In [29]:
```
df_ej4 = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])

prefijo='A_'
columnas=df_ej4.columns
pref_col=prefijo+columnas
pref_col=pref_col.values

df_ej4_rename=df_ej4.reindex(columns=pref_col)  #Se haría así pero el reindex no me va para columnas
```

EJERCICIO 5: Write a Pandas program to reverse order (rows, columns) of a given DataFrame

In [30]:
```
df_ej5 = pn.DataFrame(np.random.randn(8,4), index=dates, columns=['A','B','C','D'])
df_ej5.loc[::-1]
```

Out[30]:

	A	B	C	D
2000-01-08	-0.131006	-1.788441	1.819614	1.228068
2000-01-07	2.392694	0.267355	-0.860639	2.105230
2000-01-06	-0.834638	0.822418	0.334117	0.529068
2000-01-05	0.581159	0.368130	-0.425623	-2.971583
2000-01-04	1.321511	0.000311	0.210278	-0.057851
2000-01-03	1.054368	0.086818	0.725416	-1.221532
2000-01-02	-0.356919	1.301396	-0.356771	-0.805650
2000-01-01	-1.703022	-0.537547	-0.221594	-0.154773

EJERCICIO 6: Escribir un programa que pregunte al usuario por las ventas de un rango de años y muestre por pantalla una serie con los datos de las ventas indexada por los años, antes y después de aplicarles un descuento del 10%.

In [69]:
```
df=pn.DataFrame({'Ventas':[13434,20000,3000,3455565,23254,234543]},index=[2000,2001,2002,2003,2004,2005])

inicio=input('Año inicio: ')
fin=input('Año fin: ')

df_2=df.loc[inicio:fin]
df_2['Descontado']=df_2.values-(0.1*df_2.values)
df_2
```

```
Año inicio: 2000
Año fin: 2005
```

```
<ipython-input-69-76549d498028>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_2['Descontado']=df_2.values-(0.1*df_2.values)
```

Out[69]:

	Ventas	Descontado
2000	13434	12090.6
2001	20000	18000.0
2002	3000	2700.0
2003	3455565	3110008.5
2004	23254	20928.6
2005	234543	211088.7

EJERCICIO 7: Los ficheros emisiones-2016.csv, emisiones-2017.csv, emisiones-2018.csv y emisiones-2019.csv, contienen datos sobre las emisiones contaminates en la ciudad de Madrid en los años 2016, 2017, 2018 y 2019 respectivamente. Escribir un programa con los siguientes requisitos:

- Generar un DataFrame con los datos de los cuatro ficheros.
- Filtrar las columnas del DataFrame para quedarse con las columnas ESTACION, MAGNITUD, AÑO, MES y las correspondientes a los días D01, D02, etc.
- Reestructurar el DataFrame para que los valores de los contaminantes de las columnas de los días aparezcan en una única columna.
- Añadir una columna con la fecha a partir de la concatenación del año, el mes y el día (usar el módulo datetime).
- Eliminar las filas con fechas no válidas (utilizar la función isnat del módulo numpy) y ordenar el DataFrame por estaciones, contaminantes y fecha.
- Mostrar por pantalla las estaciones y los contaminantes disponibles en el DataFrame.
- Crear una función que reciba una estación, un contaminante y un rango de fechas y devuelva una serie con las emisiones del contaminante dado en la estación y rango de fechas dado.
- Mostrar un resumen descriptivo (mímino, máximo, media, etc) para cada contaminante.
- Mostrar un resumen descriptivo para cada contaminente por distritos.
- Crear una función que reciba una estación y un contaminante y devuelva un resumen descriptivo de las emisiones del contaminante indicado en la estadión indicada.
- Crear una función que devuelva las emisiones medias mensuales de un contaminante y un año dados para todos las estaciones.
- Crear un función que reciba una estación de medición y devuelva un DataFrame con las medias mensuales de los distintos tipos de contaminantes.

In [ ]:
```
df_2016=pn.read_csv('emisiones-2016.csv', sep=';')
df_2017=pn.read_csv('emisiones-2017.csv', sep=';')
df_2018=pn.read_csv('emisiones-2018.csv', sep=';')
df_2019=pn.read_csv('emisiones-2019.csv', sep=';')

#Generar un único DataFrame

df_total=pn.concat([df_2016,df_2017,df_2018,df_2019])

#Filtrar las columnas del DataFrame para quedarse con las columnas ESTACION, MAGNITUD, AÑO, MES y las correspondientes
#a los días D01, D02, etc.

columnas=['ESTACION','MAGNITUD','ANO','MES']
add_columnas=[]

d='D'
for i in range(1,32,1):
    if i <=9:
        string=d+'0'+str(i)
        add_columnas.append(string)
    else:
        string=d+str(i)
        add_columnas.append(string)

columnas=columnas+add_columnas
df_apartado_2=df_total[columnas]

#Reestructurar el DataFrame para que los valores de los contaminantes de las columnas de los días aparezcan en una única columna.

df_D=df_total[add_columnas]
lista_dia=[]
lista_valor=[]
lista_mes=[]

for i in df_D.columns:
    cont=1
    for j in df_D[i].values:
        lista_mes.append(cont)
        lista_dia.append(i)
        lista_valor.append(j)
        if cont <12:
            cont=cont+1
        else: cont=1
df_ap_3=pn.DataFrame()
df_ap_3['MES']=lista_mes
df_ap_3['DIA']=lista_dia
df_ap_3['VALOR']=lista_valor
df_ap_3_1=df_apartado_2[['ESTACION','MAGNITUD','ANO','MES']]
pn.merge(df_ap_3_1,df_ap_3, on='MES', how='left')
```

ejercicio 8: El fichero cotizacion.csv contiene las cotizaciones de las empresas del IBEX35 con las siguientes columnas: nombre (nombre de la empresa), Final (precio de la acción al cierre de bolsa), Máximo (precio máximo de la acción durante la jornada), Mínimo (precio mínimo de la acción durante la jornada), volumen (Volumen al cierre de bolsa), Efectivo (capitalización al cierre en miles de euros). Construir una función que construya un DataFrame a partir del un fichero con el formato anterior y devuelva otro DataFrame con el mínimo, el máximo y la media de dada columna.

In [ ]:
```
df=pn.read_csv('cotizacion.csv', sep=';')
print(df)

maximo={}
minimo={}
media={}
columnas=df.columns
for i in columnas:
    if i != 'Nombre':
        maximo[i]=max(df[i].values)
        minimo[i]=min(df[i].values)
        media[i]=sum(df[i].values)/len(df[i].values)

indice=['Máximo','Mínimo','Media']
df=pn.DataFrame(columns=maximo.keys(),index=indice)

df_1=pn.DataFrame(maximo,index=['Máximo'])
df_2=pn.DataFrame(minimo,index=['Mínimo'])
df_3=pn.DataFrame(media,index=['Media'])

df_1_2=pn.merge(df_1,df_2,how='outer')
df_1_2_3=pn.merge(df_1_2,df_3,how='outer')
df_fin=df_1_2_3.rename(index={0:'Máximo',1:'Mínimo',2:'Media'})
df_fin
```

EJERCICIO 9: Escribir programa que genere y muestre por pantalla un DataFrame con los datos de la tabla siguiente:

Mes Ventas Gastos Enero 30500 22000 Febrero 35600 23400 Marzo 28300 18100 Abril 33900 20700

In [ ]:
```
import pandas as pd

datos = {'Mes':['Enero', 'Febrero', 'Marzo', 'Abril'], 'Ventas':[30500, 35600, 28300, 33900], 'Gastos':[22000, 23400, 18100, 20700]}
cont = pd.DataFrame(datos)
print(cont)
```

EJERCICIO 10: Escribir una función que reciba un DataFrame con el formato del ejercicio anterior, una lista de meses, y devuelva el balance (ventas - gastos) total en los meses indicados.

In [ ]:
```
def balance(df,meses):
    resultado=[]
    for i in meses:
        fila=df.loc[df.Mes==i]
        resultado.append(fila.Ventas.values-fila.Gastos.values)
    df_output=pd.DataFrame(columns=['balance'], index=meses)
    df_output['balance']=resultado
    return(df_output)

mes=['Enero','Febrero','Marzo','Abril']
balance(cont,mes)
```