

DS-001 Documentacion Numpy y Pandas

Table of Contents

INTRODUCCIÓN A DATA SCIENCE¶	1
<u>Teoria y Definicion¶</u>	1
<u>El Científico de Datos y su futuro laboral.¶</u>	1
<u>Tipos de datos. Variables y Estructuras¶</u>	1
<u>Numpy¶</u>	2
<u>Preparacion del Entorno.¶</u>	2
<u>Creacion de numpy arrays¶</u>	2
<u>Pandas.¶</u>	6
<u>Preparación del Entorno¶</u>	6
<u>Creación de un DataFrame¶</u>	7
<u>Lectura de Ficheros CSV¶</u>	8
<u>Lectura desde Clipboard¶</u>	8
<u>Exploración de DataFrames¶</u>	8
<u>Selección: Indexing y Slicing.¶</u>	10
<u>Procesado de Dataframes.¶</u>	13
<u>Exportar/Importar DataFrame a excel¶</u>	16

INTRODUCCIÓN A DATA SCIENCE¶

En este apartado expondremos que es el Data Science (ciencia de datos), que tipos de variables utiliza, estructuras y herramientas principales (Numpy y Pandas).

Teoria y Definicion¶

La ciencia de datos es un campo interdisciplinario que involucra varios métodos científicos para el análisis de datos. Según la [wikipedia](#) Data Science se define como:

"Un concepto para unificar estadísticas, análisis de datos, aprendizaje automático y sus métodos relacionados para comprender y analizar los fenómenos reales, empleando técnicas y teorías extraídas de muchos campos dentro del contexto de las matemáticas, la estadística, la ciencia de la información y la informática."

Para ver la actual demanda de los diez puestos mejor pagados que requieran un conocimiento de análisis de datos, visitar el siguiente enlace:

[10 High-Paying Jobs That Require a Knowledge of Data Analytics](#)

El Científico de Datos y su futuro laboral.¶

Un científico de datos debe seguir una serie de pasos en cualquiera de sus proyectos:

- Extraer datos, independientemente de la fuente y de su volumen.
- Limpiar los datos, para eliminar lo que pueda sesgar los resultados.
- Procesar los datos usando métodos estadísticos como inferencia estadística, modelos de regresión, pruebas de hipótesis, etc.
- Diseñar experimentos adicionales en caso de ser necesario.
- Crear visualizaciones gráficas de los datos relevantes de la investigación.²³

Por norma general las fases distinguidas son:

1. Definición de objetivos: Define los problemas a solucionar, se soluciona normalmente con el cliente y los técnicos, estudiando el objetivo a alcanzar. Para los data scientist, un buen objetivo tiene que seguir la regla S.M.A.R.T (specific, measurable, achievable, relevant, time-bound)

2. Obtención de datos: Los datos se obtienen de cualquier forma que podamos imaginar como desarrolladores, desde bases de datos, hasta archivos csv, excel etc... La obtención de datos es una de las fases más importantes en el desarrollo del proyecto, ya que cuantas mas completos y extensos sean los registros, más preciso será el análisis

Tipos de datos. Variables y Estructuras¶

Los datos pueden dividirse en los siguientes tipos de variables:

- *continuas*: edad, altura, colores RGB etc.
- *ordinales*: rating, niveles educativos etc.
- *categoricas*: valores booleanos, días de la semana etc.

A su vez, pueden categorizarse según su estructura:

- *Estructurados <10%*: Son datos que se relacionan entre sÃ– y comparten informaciÃ³n como el catalogo de biblioteca, bases de datos sql.
- *Semiestructurados <10%*: no tienen estructura, pero es facil asignarle una estructura mediante la lÃ³gica. xml, json, csv.
- *No estructurados*: emails, fotos, pdf. Hoy en dÃ–a mÃ¡s del 80% de los datos son no estructurados, por lo que perdemos mucha informaciÃ³n al dificultarnos a nosotros mismos el anÃ¡lisis.

Numpy¶

Numpy es la piedra angular de la computaciÃ³n cientÃ–fica en Python. Nos permite trabajar con array 'n' dimensionales, los cuales nos proporcionan ventajas frente a las listas de Python.

Numpy a bajo nivel esta compilado en C, y al trabajar con arrays (la disposiciÃ³n en las celdas de memoria frente a las listas) es una herramienta muy potente para trabajar en Data Science con Python.

Enlace a la pagina oficial: <https://www.numpy.org/>

Preparacion del Entorno.¶

Para poder trabajar con Numpy (y mÃ¡s librerÃ–as detalladas en los siguientes documentos) necesitamos activar un entorno desde la terminal.

En Linux:

```
source activate data
```

En Windows:

```
activate data
```

En ambos casos 'data' sera el nombre del entorno. Ahora procederemos a instalar en el entorno **Numpy** mediante **conda**

```
conda install numpy
```

Nos preguntará si queremos instalarlo, marcamos 'y' y pulsamos 'enter'.

Creacion de numpy arrays¶

In [2]:

```
import sys
import numpy as np
```

Aqui explicaremos mediante markdown el significado de las variables y para que utilizamos la herramientas.

Vectores.¶

In [3]:

```
#Instanciacion de un array de 1 dimension.
array_1d = np.array([4,5, 3])
type(array_1d)
```

Out[3]:

```
numpy.ndarray
```

In [4]:

```
#np.ones genera un vector de longitud 3 inicializado con todos
#los valores a 1
print("np.ones\n",np.ones(3))

np.ones
[1.  1.  1.]
```

Matrices.¶

In [5]:

```
#Instanciacion de una matriz
matriz = np.array([
    [ 1,2, 1 ],
    [5, 43, 5]
])

matriz
```

Out[5]:

```
array([[ 1,  2,  1],
       [ 5, 43,  5]])
```

In [6]:

```
#np.eye genera una matriz identidad de 3x3
print("np.eye\n",np.eye(3))

np.eye
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
```

In [7]:

```
#np.zeros genera una matriz con todos sus valores a 0
print("np.zeros\n",np.zeros((3,2)))

np.zeros
[[0.  0.]
 [0.  0.]
 [0.  0.]]
```

In [8]:

```
#np.random produce un array con valores aleatorios entre el intervalo [0,1]
np.random.random((2,3))
```

Out[8]:

```
array([[0.93703451, 0.37320958, 0.64261727],
       [0.95977855, 0.83869418, 0.9823097 ]])
```

Flujo de lectura y volcado en array.¶

In [9]:

```
#se puede acceder a un documento de texto y volcarlo
#a un numpy array
```

DS-001 Documentacion Numpy y Pandas

```
np_text = np.genfromtxt("np_text.txt", delimiter=",")
np_text
```

Out[9]:

```
array([[ 1.,  2.,  3.],
       [43.,  2.,  3.],
       [34.,  1.,  1.],
       [ 0.,  1.,  1.]])
```

Selección por secciones y por índices (slicing e indexing)

In [10]:

```
#instanciamos matriz de ejemplo
matriz_34 = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
matriz_34
```

Out[10]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [11]:

```
#obtenemos su primera fila como si de una lista se tratase
matriz_34[0]
```

Out[11]:

```
array([1, 2, 3, 4])
```

In [12]:

```
#seleccionamos ahora hasta la fila 2 (fila 1 y fila 2)
matriz_34[:2]
```

Out[12]:

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [13]:

```
#podemos tambien seleccionar el segundo elemento de cada fila
#es decir, la segunda columna
matriz_34[:,1]
```

Out[13]:

```
array([ 2,  6, 10])
```

In [14]:

```
#al crear secciones solo obtenemos un puntero o referencia
#al mismo array, no instanciamos nuevos objetos.
seccion = matriz_34[:2,:]
print(matriz_34[0,1])
seccion[0, 0] = 0
matriz_34
```

2

Out[14]:

```
array([[ 0,  2,  3,  4],
       [ 5,  6,  7,  8],
```

Creación de numpy arrays

```
[ 9, 10, 11, 12]])
```

Filtrado¶

In [15]:

```
matriz_32 = np.array([[1, 4], [2, 4], [5, 0]])
matriz_32
```

Out[15]:

```
array([[1, 4],
       [2, 4],
       [5, 0]])
```

In [16]:

```
#comprobamos aquellos elementos que sean mayores o iguales que 2
indice_filtrado = (matriz_32 >= 2)
indice_filtrado
```

Out[16]:

```
array([[False,  True],
       [ True,  True],
       [ True, False]])
```

In [17]:

```
matriz_32[indice_filtrado]
```

Out[17]:

```
array([4, 2, 4, 5])
```

Aritmetica con numpy arrays¶

In [18]:

```
array1 = np.array([[2,3],[0,1]])
array2 = np.array([[23,6],[0,42]])

print(array1)
print(array2)

[[2 3]
 [0 1]]
[[23  6]
 [ 0 42]]
```

In [19]:

```
array1 + array2
```

Out[19]:

```
array([[25,  9],
       [ 0, 43]])
```

In [20]:

```
array1 * array2
```

Out[20]:

```
array([[46, 18],
       [ 0, 42]])
```

Creacion de numpy arrays¶

Desde python 3.5, podemos usar el simbolo @ para indicar una multiplicaci3n de matrices (para versiones anteriores se usa la funcion dot

In [21]:

```
array1 @ array2
```

Out[21]:

```
array([[ 46, 138],
       [  0,  42]])
```

este producto equivale a

In [22]:

```
array1.dot(array2)
```

Out[22]:

```
array([[ 46, 138],
       [  0,  42]])
```

Ventajas de np.array vs lists.¶

In [23]:

```
lista_2d = [[1222,2222,2223], [5,23,40004]]

array_2d = np.array([[1222,2222,2223], [5,23,40004]])

print("Tama±o de la lista en memoria: {} bytes".format(sys.getsizeof(lista_2d)))
print("Tama±o del numpy array en memoria: {} bytes".format(sys.getsizeof(array_2d)))

Tama±o de la lista en memoria: 80 bytes
Tama±o del numpy array en memoria: 160 bytes
```

In [24]:

```
big_list = list(range(10000))
big_array = np.array(range(100000))

print("Tama±o de la lista en memoria: {} bytes".format(sys.getsizeof(big_list)))
print("Tama±o del numpy array en memoria: {} bytes".format(sys.getsizeof(big_array)))

Tama±o de la lista en memoria: 90112 bytes
Tama±o del numpy array en memoria: 800096 bytes
```

Pandas.¶

Pandas es una librerÃa escrita como extensi3n de numpy para manipulaci3n y anÃlisis de datos para el lenguaje de programaci3n Python.

Preparaci3n del Entorno¶

Para su instalaci3n en el entorno pueden utilizarse los siguientes comandos:

- Via Conda:
conda install pandas
- Via conda forge:

DS-001 Documentacion Numpy y Pandas

```
conda install -c conda-forge pandas
```

- Vida PyPI:

```
python3 -m pip install --upgrade pandas
```

Las características de la biblioteca son:

- **El tipo de datos son DataFrame** para manipulación de datos con indexación integrada. Tiene herramientas para leer y escribir datos entre estructuras de datos en memoria y formatos de archivos variados
- Permite la alineación de datos y manejo integrado de datos faltantes, la reestructuración y segmentación de conjuntos de datos, la segmentación vertical basada en etiquetas, indexación elegante, y segmentación horizontal de grandes conjuntos de datos, la inserción y eliminación de columnas en estructuras de datos.
- Puedes realizar cadenas de operaciones, dividir, aplicar y combinar sobre conjuntos de datos, la mezcla y unión de datos.
- Permite realizar indexación jerárquica de ejes para trabajar con datos de altas dimensiones en estructuras de datos de menor dimensión, la funcionalidad de series de tiempo: generación de rangos de fechas y conversión de frecuencias, desplazamiento de ventanas estadísticas y de regresiones lineales, desplazamiento de fechas y retrasos.

fuentes: <https://www.master-data-scientist.com/pandas-herramienta-data-science/>

Creación de un DataFrame¶

In [25]:

```
import pandas as pd
```

In [26]:

```
pd.__version__
```

Out[26]:

```
'0.24.2'
```

In [27]:

```
#instanciacion de un DataFrame.
rick_and_morty = pd.DataFrame(
    {
        "nombre":["Rick", "Morty"],
        "apellidos":["Sanchez", "Smith"],
        "edad": [60, 14]
    })
rick_and_morty
```

Out[27]:

	nombre	apellidos	edad
0	Rick	Sanchez	60
1	Morty	Smith	14

In [28]:

```
#Observamos el tipo de objeto de la variable rick_and_morty
type(rick_and_morty)
```

Out[28]:

```
pandas.core.frame.DataFrame
```

Preparación del Entorno¶

Tambi n podemos instanciar un dataframe pasandole listas y especificando en el segundo par metro el nombre de las columnas

In [29]:

```
rick_and_morty = pd.DataFrame(
    [
        ["Rick", "Sanchez", 60],
        ["Morty", "Smith", 14]
    ], columns = ["nombre", "apellidos", "edad"]
)

rick_and_morty
```

Out[29]:

	nombre	apellidos	edad
0	Rick	Sanchez	60
1	Morty	Smith	14

Lectura de Ficheros CSV¶

Lo mas habitual al trabajar con dataframes de pandas es cargar los datos del mismo de un archivo **csv**. Por convenci n, cuando trabajamos con un dataframe "generico" se le suele nombrar **df**

In [30]:

```
#Flujo input para leer csv
df = pd.read_csv("primary_results.csv")
df

#Flujo output para escribir csv
df.to_csv("test.csv")
```

Lectura desde Clipboard¶

Si seleccionamos y copiamos un fragemnto de Dataframe se podra mostrar posteriormente gracias al metodo **read_clipboard**

In [31]:

```
df = pd.read_clipboard()
df
```

Out[31]:

export	to	pdf	python	jupyter
--------	----	-----	--------	---------

Exploraci n de DataFrames¶

Para la demostraci n de los m todos de exploraci n de Dataframes, utilizaremos el dataframe de los resultado de las votaciones primarias en Estados Unidos.

In [32]:

```
df = pd.read_csv("primary_results.csv")
```

Para obtener el numero total de registros y el total de columnas se utilizar  el metodo **shape**

In [33]:

DS-001 Documentacion Numpy y Pandas

```
df.shape
```

```
Out[33]:
(24611, 8)
```

Si queremos visualizar los cinco primeros registros para hacernos una idea de la composición del DataFrame sin tener que cargarlo entero en el notebook, se utilizará el método **head**

```
In [34]:
df.head()
```

```
Out[34]:
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078

Si por el contrario queremos ver los cinco últimos registros, se ejecutará el método **tail**

```
In [35]:
df.tail()
```

```
Out[35]:
```

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
24606	Wyoming	WY	Teton-Sublette	95600028.0	Republican	Ted Cruz	0	0.0
24607	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Donald Trump	0	0.0
24608	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	John Kasich	0	0.0
24609	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Marco Rubio	0	0.0
24610	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Ted Cruz	53	1.0

Otro método muy valioso para el entendimiento del DataFrame es el **dtypes**. Este método nos da información sobre el tipo de datos que almacena cada columna.

```
In [36]:
df.dtypes
```

```
Out[36]:
state                object
state_abbreviation    object
county              object
fips                float64
party              object
candidate           object
votes              int64
fraction_votes      float64
dtype: object
```

Aunque si lo que se quiere conseguir es obtener informaci3n precisa acerca de los registros del dataframe se podr3 utilizar el metodo **describe**

In [37]:

```
df.describe()
```

Out[37]:

	fips	votes	fraction_votes
count	2.451100e+04	24611.000000	24611.000000
mean	2.667152e+07	2306.252773	0.304524
std	4.200978e+07	9861.183572	0.231401
min	1.001000e+03	0.000000	0.000000
25%	2.109100e+04	68.000000	0.094000
50%	4.208100e+04	358.000000	0.273000
75%	9.090012e+07	1375.000000	0.479000
max	9.560004e+07	590502.000000	1.000000

Selección: Indexing y Slicing.¶

Como las listas en python se puede hacer selecci3n mediante el indexing y el slicing En este apartado veremos adem3s como seleccionar por columna o incluso por campo.

Todo dataframe contiene un **index** que aunque no es correspondiente a una columna, podemos hacer referencia a el.

In [38]:

```
df.head()
```

Out[38]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078

In [39]:

```
#obtenemos informaci3n del index
print(df.index)

#seleccionamos el registro con index 0
df.loc[0]

RangeIndex(start=0, stop=24611, step=1)
```

Out[39]:

```
state                Alabama
state_abbreviation    AL
county              Autauga
fips                 1001
party                Democrat
candidate            Bernie Sanders
```

Selección: Indexing y Slicing.¶

DS-001 Documentacion Numpy y Pandas

```
votes          544
fraction_votes 0.182
Name: 0, dtype: object
```

El index es un puntero que hace referencia al orden en el dataframe. Este puntero e puede cambiar a cualquier otra columna:

In [40]:

```
df2 = df.set_index("county")
```

In [41]:

```
df2.head()
```

Out[41]:

	state	state_abbreviation	fips	party	candidate	votes	fraction_votes
county							
Autauga	Alabama	AL	1001.0	Democrat	Bernie Sanders	544	0.182
Autauga	Alabama	AL	1001.0	Democrat	Hillary Clinton	2387	0.800
Baldwin	Alabama	AL	1003.0	Democrat	Bernie Sanders	2694	0.329
Baldwin	Alabama	AL	1003.0	Democrat	Hillary Clinton	5290	0.647
Barbour	Alabama	AL	1005.0	Democrat	Bernie Sanders	222	0.078

Como podemos comprobar ahora la columna *county* serÃ referenciado como index.

In [42]:

```
df2.index
```

Out[42]:

```
Index(['Autauga', 'Autauga', 'Baldwin', 'Baldwin', 'Barbour', 'Barbour',
      'Bibb', 'Bibb', 'Blount', 'Blount',
      ...,
      'Sweetwater-Carbon', 'Sweetwater-Carbon', 'Teton-Sublette',
      'Teton-Sublette', 'Teton-Sublette', 'Teton-Sublette', 'Uinta-Lincoln',
      'Uinta-Lincoln', 'Uinta-Lincoln', 'Uinta-Lincoln'],
      dtype='object', name='county', length=24611)
```

Ahora que se ha cambiado el index, se puede seleccionar por condado:

In [43]:

```
df2.loc["Los Angeles"]
```

Out[43]:

	state	state_abbreviation	fips	party	candidate	votes	fraction_votes
county							
Los Angeles	California	CA	6037.0	Democrat	Bernie Sanders	434656	0.420
Los Angeles	California	CA	6037.0	Democrat	Hillary Clinton	590502	0.570
Los Angeles	California	CA	6037.0	Republican	Donald Trump	179130	0.698
Los Angeles	California	CA	6037.0	Republican	John Kasich	33559	0.131
Los Angeles	California	CA	6037.0	Republican	Ted Cruz	30775	0.120

Con esto demostramos que **loc** selecciona por indice no por posiciÃ³n. Si lo que queremos por el contrario es seleccionar el nÃºmero de fila en lugar del indice se deberÃ utilizar el mÃ©todo **iloc**

In [44]:

```
df2.iloc[0]
```

Out[44]:

```

state                Alabama
state_abbreviation    AL
fips                 1001
party                Democrat
candidate            Bernie Sanders
votes                544
fraction_votes        0.182
Name: Autauga, dtype: object

```

Los dataframes soportan parametros de busqueda entre corchetes como los diccionarios de Python:

In [45]:

```
df["state"][:10]
```

Out[45]:

```

0    Alabama
1    Alabama
2    Alabama
3    Alabama
4    Alabama
5    Alabama
6    Alabama
7    Alabama
8    Alabama
9    Alabama
Name: state, dtype: object

```

Saber esto es muy util, ya que nos permite acceder al contenido de las columnas. En este ejemplo se introducirá una nueva columna y se asignará como valor para esa columna el numero 1.

In [46]:

```

df["shape"] = 1
df.head()

```

Out[46]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182	1
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800	1
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329	1
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647	1
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078	1

Si seleccionamos una columna, obtenemos una Serie, si seleccionamos dos o más, obtenemos un dataframe.

In [47]:

```
type(df['county'])
```

Out[47]:

```
pandas.core.series.Series
```

In [48]:

DS-001 Documentacion Numpy y Pandas

```
type(df[["county", "candidate"]])
```

Out[48]:

```
pandas.core.frame.DataFrame
```

Se puede adem  s filtrar un dataframe de la misma forma que se filtra en numpy. Adem  s estas condiciones se pueden concatenar utilizando el operador **&**

In [49]:

```
df[df.votes>=590502]
```

Out[49]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
1386	California	CA	Los Angeles	6037.0	Democrat	Hillary Clinton	590502	0.57	1

In [50]:

```
df[(df.county=="Manhattan") & (df.party=="Democrat")]
```

Out[50]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
15011	New York	NY	Manhattan	36061.0	Democrat	Bernie Sanders	90227	0.337	1
15012	New York	NY	Manhattan	36061.0	Democrat	Hillary Clinton	177496	0.663	1

Otro metodo muy utilizado para la selecci  n de registros de un dataframe es el m  todo **query** el cual nos permite hacer referencias al contenido de otras variables mediante el operador **@**.

In [51]:

```
county = "Manhattan"
df.query("county==@county")
```

Out[51]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
15011	New York	NY	Manhattan	36061.0	Democrat	Bernie Sanders	90227	0.337	1
15012	New York	NY	Manhattan	36061.0	Democrat	Hillary Clinton	177496	0.663	1
15162	New York	NY	Manhattan	36061.0	Republican	Donald Trump	10393	0.418	1
15163	New York	NY	Manhattan	36061.0	Republican	John Kasich	11251	0.452	1
15164	New York	NY	Manhattan	36061.0	Republican	Ted Cruz	3243	0.130	1

Procesado de Dataframes.¶

En este apartado se observar  n los m  todos m  s relevantes para procesar DataFrames.

Para ordenar un DataFrame se utilizar   el m  todo **sort_values**, el cual ordenar   en funci  n al valor de la columna que recibe como par  metro. Adem  s como segundo par  metro se le puede ordenar que los

ordene ascendente o descendientemente.

In [52]:

```
df_sorted = df.sort_values(by="votes", ascending=False)
df_sorted.head()
```

Out[52]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
1386	California	CA	Los Angeles	6037.0	Democrat	Hillary Clinton	590502	0.570	1
1385	California	CA	Los Angeles	6037.0	Democrat	Bernie Sanders	434656	0.420	1
4451	Illinois	IL	Chicago	91700103.0	Democrat	Hillary Clinton	366954	0.536	1
4450	Illinois	IL	Chicago	91700103.0	Democrat	Bernie Sanders	311225	0.454	1
4463	Illinois	IL	Cook Suburbs	91700104.0	Democrat	Hillary Clinton	249217	0.536	1

Un método que nos permite agrupar columnas es el método **groupby**. Utilizaremos este método para agrupar las columnas referentes al estado y la referente al partido político del actual dataframe. Posteriormente realizaremos una selección de la suma de sus votos.

Con esta operación obtendremos una lista con los resultados de voto por partido en los distintos estados de Estados Unidos.

In [53]:

```
df.groupby(["state", "party"])["votes"].sum()
```

Out[53]:

```
state      party      votes
Alabama    Democrat    386327
           Republican    837632
Alaska     Democrat      539
           Republican    21930
Arizona    Democrat    399097
           Republican    435103
Arkansas   Democrat    209448
           Republican    396523
California Democrat    3442623
           Republican    1495574
Colorado   Democrat    121184
Connecticut Democrat    322485
           Republican    208817
Delaware   Democrat      92609
           Republican      67807
Florida    Democrat    1664003
           Republican    2276926
Georgia    Democrat    757340
           Republican    1275601
Hawaii     Democrat      33658
           Republican      13228
Idaho      Democrat      23705
           Republican      215284
Illinois    Democrat    1987834
           Republican    1384703
```


DS-001 Documentacion Numpy y Pandas

```

Indiana      Democrat      638638
              Republican    1080653
Iowa         Democrat      139980
              Republican    186724
Kansas      Democrat      39043
              ...
Oklahoma     Democrat      313392
              Republican    452731
Oregon       Democrat      572485
              Republican    361490
Pennsylvania Democrat      1638644
              Republican    1537696
Rhode Island Democrat      119213
              Republican    60381
South Carolina Democrat    367491
              Republican    737917
South Dakota Democrat      53004
              Republican    66877
Tennessee   Democrat      365637
              Republican    834939
Texas        Democrat      1410641
              Republican    2737248
Utah         Democrat      76999
              Republican    177204
Vermont      Democrat      134198
              Republican    58762
Virginia     Democrat      778865
              Republican    1012807
Washington   Democrat      26299
              Republican    510851
West Virginia Democrat    210214
              Republican    188138
Wisconsin     Democrat      1000703
              Republican    1072699
Wyoming      Democrat      280
              Republican    903
Name: votes, Length: 95, dtype: int64

```

Mediante la funciÃ³n **apply** al que se puede agregar valores a una columna a travÃ©s de los resultados de una funciÃ³n

In [54]:

```

#mediante esta funciÃ³n obtenemos la primera letra de cada estado
df.state_abbreviation.apply(lambda s:s[0])

#si esto lo agregamos a una columna podemos volcarlo al DataFrame
df["letra_estado"] = df.state_abbreviation.apply(lambda s: s[0])
df.head()

```

Out[54]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape	letra_estado
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182	1	A
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800	1	A
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329	1	A

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape	letra_estado
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647	1	A
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078	1	A

Exportar/Importar DataFrame a excel¶

Además de exportar e importar ficheros csv también podemos exportar e importar de excel, pero para ello será necesario instalar el paquete *xlwt*

```
!conda install -y xlwt
```

In [55]:

```
rick_and_morty.to_excel("rick_y_morty.xls", sheet_name="personajes")
```

In [56]:

```
rick_morty2 = pd.read_excel("rick_y_morty.xls", sheet_name="personajes")
```

In [57]:

```
rick_morty2.head()
```

Out[57]:

	Unnamed: 0	nombre	apellidos	edad
0	0	Rick	Sanchez	60
1	1	Morty	Smith	14