

David Bermejo Simón.

Jose Luis Luengo Ramos.

Investigación en Tecnologías Data Science y Machine Learning con Python.



ANEXO I

DOCUMENTACIÓN DATA SCIENCE

En este anexo se incluye toda la documentación impresa del repositorio.
El enlace a git de esta sección es:

<https://github.com/AC-SweetShop/Repo-TFG/tree/master/DATA%20SCIENCE>

David Bermejo Simón.

Jose Luis Luengo Ramos.

Investigación en Tecnologías Data Science y Machine Learning con Python.



```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-30T20:57:56+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system   : Linux  
release  : 5.1.5-arch1-2-ARCH  
machine  : x86_64  
processor:  
CPU cores: 4  
interpreter: 64bit
```

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.
- *categóricas*: valores booleanos, días de la semana etc.

A su vez, pueden categorizarse según su estructura:

- **Estructurados <10%:** Son datos que se relacionan entre sí y comparten información como el catálogo de biblioteca, bases de datos sql.
- **Semiestructurados <10%:** no tienen estructura, pero es fácil asignarle una estructura mediante la lógica. xml, json, csv.
- **No estructurados:** emails, fotos, pdf. Hoy en día más del 80% de los datos son no estructurados, por lo que perdemos mucha información al dificultarnos a nosotros mismos el análisis.

Numpy

Numpy es la piedra angular de la computación científica en Python. Nos permite trabajar con arrays 'n' dimensionales, los cuales nos proporcionan ventajas frente a las listas de Python.

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

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

Preparación del Entorno.

Para poder trabajar con Numpy (y más librerías 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' será 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'.

Creación de numpy arrays

In [2]:

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

Aquí explicaremos mediante markdown el significado de las variables y para qué utilizamos la herramienta.

Vectores.

In [3]:

```
#Instanciación de un array de 1 dimensión.
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]:

```
#Instanciación 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([[0.14252823, 0.84562171, 0.23374288],
       [0.64021973, 0.14124967, 0.93168794]])
```

Flujo de lectura y volcado en array.

In [9]:

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

Seleccion por secciones y por indices (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],
       [ 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]])
```

Desde python 3.5, podemos usar el simbolo @ para indicar una multiplicación de matrices (para versiones anteriores se usa la función 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("Tamaño de la lista en memoria: {} bytes".format(sys.getsizeof(lista_2d)))
print("Tamaño del numpy array en memoria: {} bytes".format(sys.getsizeof(array_2d)))
```

Tamaño de la lista en memoria: 80 bytes
Tamaño del numpy array en memoria: 160 bytes

In [24]:

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

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

Tamaño de la lista en memoria: 90112 bytes
Tamaño del numpy array en memoria: 800096 bytes

Pandas.

[Pandas](#) es una librería escrita como extensión de numpy para manipulación y análisis de datos para el lenguaje de programación Python.

Preparación del Entorno

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

- Via Conda:

```
conda install pandas
```

- Via conda forge:

```
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 dato en memoria y formatos de archivos variados
- Permite la alineación de dato y manejo integrado de datos fallantes, 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.

fuente: <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
```

También podemos instanciar un dataframe pasandole listas y especificando en el segundo parámetro 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 convención, 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 fragmento de Dataframe se podra mostrar posteriormente gracias al metodo `read_clipboard`

```
In [31]:
```

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

```
Out[31]:
```

```
http://localhost:8888/login?next=%2Ftree
```

Exploración de DataFrames

Para la demostración de los métodos de exploración 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 utilizará el metodo `shape`

```
In [33]:
```

```
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 metodo `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 DataFrames es el **dtypes**. Este metodo 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 información precisa acerca de los registros del dataframe se podrá 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	tips	1375.000000	votes	0.479000	fraction_votes
max	9.560004e+07		590502.000000		1.000000	

Selección: Indexing y Slicing.

Como las listas en python se puede hacer selección mediante el indexing y el slicing En este apartado veremos además 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 información 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
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` será 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 posición. Si lo que queremos por el contrario es seleccionar el número de fila en lugar del indice se deberá utilizar el método `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]:

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

Out[48]:

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

Se puede además filtrar un dataframe de la misma forma que se filtra en numpy. Además 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 selección de registros de un dataframe es el método **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	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes	shape
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 observarán los métodos más relevantes para procesar DataFrames.

Para ordenar un DataFrame se utilizará el método `sort_values`, el cual ordenará en función al valor de la columna que recibe como parámetro. Además como segundo parámetro se le puede ordenar que los ordene ascendente o descendenteamente.

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
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 función **apply** al que se puede agregar valores a una columna a través de los resultados de una función

In [54]:

```
#mediante esta función 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
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

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-30T20:59:14+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system   : Linux  
release  : 5.1.5-arch1-2-ARCH  
machine  : x86_64  
processor:  
CPU cores: 4  
interpreter: 64bit
```

ESTADÍSTICOS DESCRIPTIVOS

```
In [2]:
```

```
import numpy as np  
import pandas as pd  
import scipy.stats as stats
```

Aplicacion de la teoría

Para comprobar la teoría y aplicar los distintos estadísticos descriptivos se utilizará una lista con distintos valores numéricos. En el ejemplo se tratarán como una lista de pesos de productos.

```
In [3]:
```

```
pesos = [100,150,150,200,250,300,325,400,415,500,600,1000]
```

Estadísticos de tendencia Central

Las medidas de tendencia central, son medidas estadísticas que pretenden resumir en un solo valor a un conjunto de valores. Representan un centro en torno al cual se encuentra ubicado el conjunto de los datos. Las medidas de tendencia central más utilizadas son la media, la mediana y la moda.

Media: para calcular la media del peso, utilizamos la librería numpy y su método mean pasandole como parametro la lista de pesos

```
In [4]:
```

```
#Obtenemos la media de los  
peso_media = np.mean(pesos)  
print(peso_media)
```

```
365.8333333333333
```

Mediana: para calcular la mediana del peso, utilizamos la librería numpy y su método median pasandole como parametro la lista de pesos

```
In [5]:
```

```
peso_mediana = np.median(pesos)  
print(peso_mediana)
```

```
312.5
```

Moda: para calcular la moda del peso, utilizamos la librería scipy (clase stats) y su método mode pasandole como parametro la lista de pesos

```
In [6]:
```

```
peso_moda = stats.mode(pesos)
```

```

peso_moda = stats.mode(pesos)
print(peso_moda)

ModeResult(mode=array([150]), count=array([2]))

```

Estadísticos de dispersión

Las medidas de dispersión miden el grado de dispersión de los valores de una variable. Pretenden evaluar en qué medida los datos difieren entre sí, es decir, si las diferentes puntuaciones de una variable están muy alejadas de la media, por lo que cuanto mayor sea ese valor mayor será la variabilidad y cuanto menor sea, más homogénea será a la media.

Rango: para calcular el rango del peso, utilizamos la librería numpy, utilizamos los métodos maximo y minimo y los restamos

In [7]:

```

peso_rango = np.max(pesos) - np.min(pesos)
print(peso_rango)

```

900

Cuartiles y IQR: Dentro de la librería stats disponemos de una clase mstats y un método mquantiles

In [8]:

```

peso_cuantiles = stats.mstats.mquantiles(pesos)
print(peso_cuantiles)

```

[172.5 312.5 461.75]

La función pandas.qcut convierte una serie en una lista de N cuartiles. Si le pasasemos 4, tenemos los cuartiles. qcut devuelve una serie categórica nueva con los cuantiles como categorías

In [9]:

```

pesos_cuartiles = pd.qcut(pesos,4)
pesos_cuartiles.categories

```

Out [9]:

```

IntervalIndex([(99.999, 187.5], (187.5, 312.5], (312.5, 436.25], (436.25, 1000.0]],
              closed='right',
              dtype='interval[float64]')

```

Desviación Standard: Con la librería numpy y el método std podemos obtener la desviación standard

In [10]:

```

peso_std = np.std(pesos)
print(peso_std)

```

239.6771555423856

Estadísticos de forma

Son aquellos que nos hablan de la forma de la distribución de datos en cuanto a su simetría y su apuntamiento.

Coeficiente de simetría: El coeficiente de simetría (skewness) se define como el tercer momento dividido por la desviación standard al cubo.

$$\frac{1}{N} \cdot \frac{\sum_{n=1}^N (X_i - \bar{X})^3}{\sigma^3}$$

In [11]:

```

pow3 = lambda x: x*x*x
tercer_momento = lambda x: pow3(x-peso_media)
simetria_pesos = sum(map(tercer_momento,pesos)) / (12*pow3(peso_std))
print(simetria_pesos)

```

1.3623858394083475

Coeficiente de curtosis: El coeficiente de curtosis se define como el cuarto momento dividido por la desviación estandar a la cuarta, tiene la siguiente formula:

$$\$ \$ \frac{1}{N} * \frac{\sum_{n=1}^N (X_i - \bar{X})^4}{\sigma^4} - 3$$

Generalmente se le resta 3 al cuarto momento, ya que una distribución normal perfecta tiene un coeficiente de curtosis de 3. En la siguiente ejecución describimos la ecuación a mano

In [12]:

```
pow4 = lambda x: x*x*x*x
cuarto_momento = lambda x: pow4(x - peso_media)
curtosis_peso = sum(map(cuarto_momento, pesos)) / (12 * pow4(peso_std)) - 3
print(curtosis_peso)
```

1.4285722765161841

En este caso gracias a la librería **scipy** y a su metodo **kurtosis** podemos obtenerlo directamente sin necesidad de desarrollar nosotros la ecuación

In [13]:

```
curtosis_peso2 = stats.kurtosis(pesos)
print(curtosis_peso2)
```

1.4285722765161841

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-17T17:14:34+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.0.13-arch1-1-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

VISUALIZACIÓN BÁSICA DE DATOS

Carga de Datos y Preparacion de DataSet

Vamos a usar un dataset clasico para empezar a aprender técnicas de visualización. Se trata del Boston Housing Dataset. Recopilado en 1976 y publicado en [Berkeley](#)

Consiste en mediciones de distintas zonas del área de Boston, teniendo como variables independientes un conjunto de mediciones de la habitabilidad de dichas zonas, y como variable independiente el valor medio de las casas en dicha zona.

En concreto vamos a usar un [dataset actualizado](#) que incluye la geolocalización estimada de las mediciones.

```
In [2]:
```

```
import pandas as pd  
df = pd.read_csv("boston_dataset.csv")  
df.head()
```

```
Out [2]:
```

	TOWN	LON	LAT	MEDV	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	Nahant	-70.955	42.2550	24.0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90
1	Swampscott	-70.950	42.2875	21.6	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90
2	Swampscott	-70.936	42.2830	34.7	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83
3	Marblehead	-70.928	42.2930	33.4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63
4	Marblehead	-70.922	42.2980	36.2	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90

Renombramos las columnas para facilitar el manejo.

```
In [3]:
```

```
df = df.rename(columns={  
    "TOWN": "CIUDAD",  
    "CRIM": "INDICE_CRIMEN",  
    "ZN": "PCT_ZONA_RESIDENCIAL",  
    "INDUS": "PCT_ZONA_INDUSTRIAL",  
    "CHAS": "RIO_CHARLES",  
    "NOX": "OXIDO_NITROSO_PPM",  
    "RM": "N_HABITACIONES_MEDIO",  
    "AGE": "PCT_CASAS_40S",  
    "DIS EMPLEO": "DISTANCIA CENTRO EMPLEO",
```

```

    "RAD": "DIS_AUTOPISTAS",
    "TAX": "CARGA_FISCAL",
    "PTRATIO": "RATIO_PROFESORES",
    "B": "PCT_NEGRA",
    "MEDV": "VALOR_MEDIANO",
    "LSTAT": "PCT_CLASE_BAJA"
  })

```

```
df.head()
```

Out [3] :

	CIUDAD	LON	LAT	VALOR_MEDIANO	INDICE_CRIMEN	PCT_ZONA_RESIDENCIAL	PCT_ZONA_INDUSTRIAL
0	Nahant	-70.955	42.2550	24.0	0.00632	18.0	2.31
1	Swampscott	-70.950	42.2875	21.6	0.02731	0.0	7.07
2	Swampscott	-70.936	42.2830	34.7	0.02729	0.0	7.07
3	Marblehead	-70.928	42.2930	33.4	0.03237	0.0	2.18
4	Marblehead	-70.922	42.2980	36.2	0.06905	0.0	2.18

In [4] :

```
df.dtypes
```

Out [4] :

```

CIUDAD          object
LON            float64
LAT            float64
VALOR_MEDIANO   float64
INDICE_CRIMEN   float64
PCT_ZONA_RESIDENCIAL float64
PCT_ZONA_INDUSTRIAL float64
RIO_CHARLES      int64
OXIDO_NITROSO_PPM float64
N_HABITACIONES_MEDIO float64
PCT_CASAS_40S    float64
DIS             float64
DIS_AUTOPISTAS   int64
CARGA_FISCAL     int64
RATIO_PROFESORES float64
PCT_NEGRA        float64
PCT_CLASE_BAJA    float64
dtype: object

```

Cómo elegir el gráfico

Como podemos ver, la guía se divide en cuatro categorías principales y luego se clasifican los distintos métodos de visualización que mejor representan cada una de esas categorías. Veamos un poco más en detalle cada una de ellas:

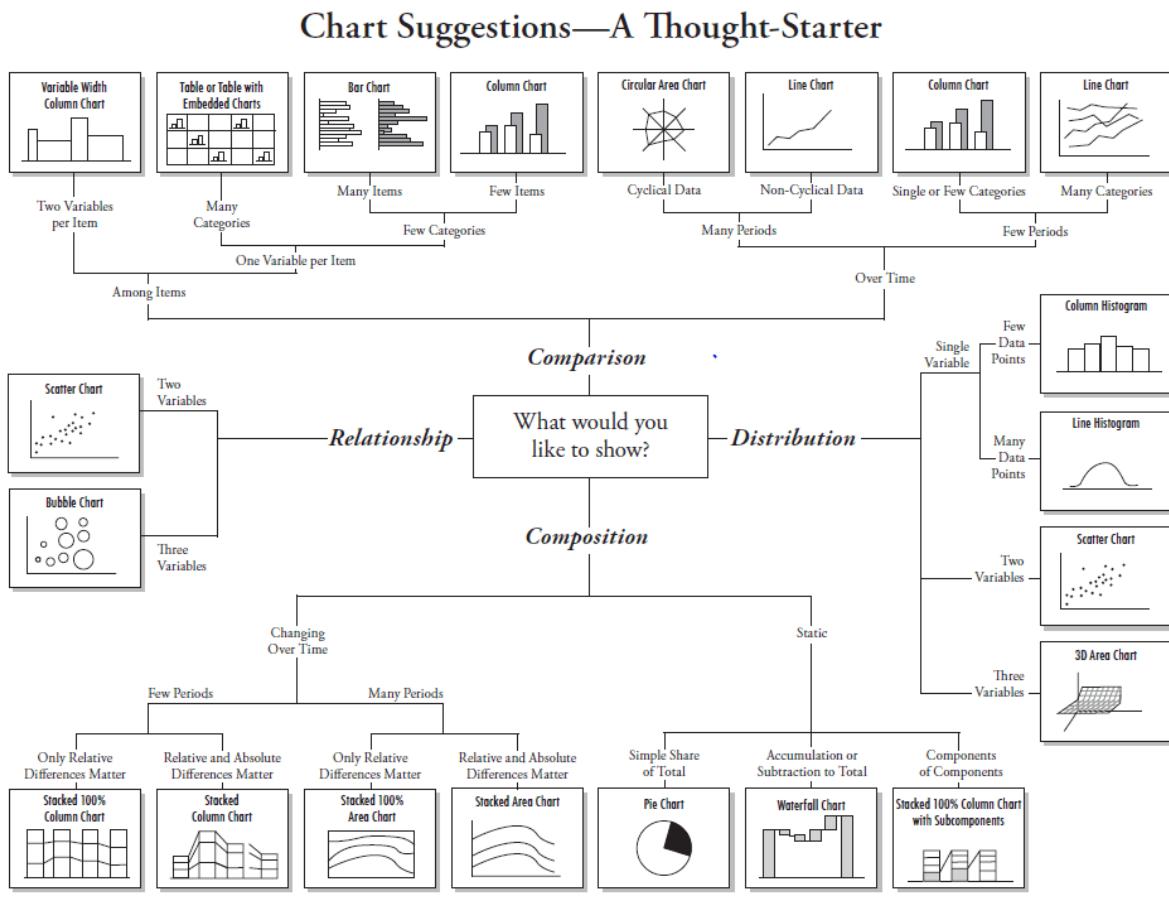
- *Distribuciones*: En esta categoría intentamos comprender como los datos se distribuyen. Se suelen utilizar en el comienzo de la etapa de exploración de datos, cuando queremos comprender las variables. Aquí también nos vamos a encontrar con variables de dos tipos cuantitativas y categóricas. Dependiendo del tipo y cantidad de variables, el método de visualización que vamos a utilizar.
- *Comparaciones*: En esta categoría el objetivo es comparar valores a través de diferentes categorías y con el tiempo (tendencia). Los tipos de gráficos más comunes en esta categoría son los diagramas de barras para cuando estamos comparando elementos o categorías y los diagramas de puntos y líneas cuando comparamos variables cuantitativas.
- *Relaciones*: Aquí el objetivo es comprender la relación entre dos o más variables. La visualización más utilizada en esta categoría es el gráfico de dispersión.
- *Composiciones*: En esta categoría el objetivo es comprender como esta compuesta o distribuida una variable; ya sea a través del tiempo o en forma estática. Las visualizaciones más comunes aquí son los diagramas de barras y los gráficos de tortas.

Fuente: <https://relopezbriga.github.io/blog/2016/09/18/visualizaciones-de-datos-con-python/>

In [17]:

```
from IPython.display import Image  
Image("../RESOURCES/chart-chooser-data-visualization.png")
```

Out [17]:



Aquí hay una herramienta online para ayudar a decidir el tipo de gráfico a usar

Matplotlib

[Matplotlib](#) es una librería para generar gráficos a partir de conjuntos de datos, bien sea datos contenidos en arrays de numpy o en dataframes de Pandas. En los siguientes casos utilizaremos esta librería para mostrar información del dataframe que hemos cargado con anterioridad on Pandas.

In [18]:

```
?matplotlib
```

In [19]:

```
import matplotlib.pyplot as plt
```

Scatter Plot / Gráfico de dispersión

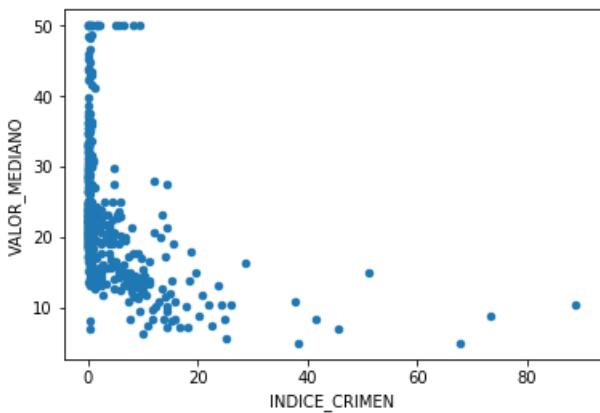
Los gráficos de dispersión son una de las mejores formas de representar la relación entre dos variables. Según [Wikipedia](#) Un diagrama de dispersión se emplea cuando una o varias variables está bajo el control del experimentador. Si existe un parámetro que se incrementa o disminuye de forma sistemática por el experimentador, se le denomina parámetro de control o variable independiente y habitualmente se representa a lo largo del eje horizontal (eje de las abscisas). La variable medida o dependiente usualmente se representa a lo largo del eje vertical (eje de las ordenadas). Si no existe una variable dependiente, cualquier variable se puede representar en cada eje y el diagrama de dispersión mostrará el grado de correlación (no causalidad) entre las dos variables.

In [20]:

```
#Usamos la funcion scatter especificando los ejes `x` e `y`
df.plot.scatter(x="INDICE_CRIMEN", y="VALOR_MEDIANO")
```

Out [20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d2641cdd8>
```



In [23]:

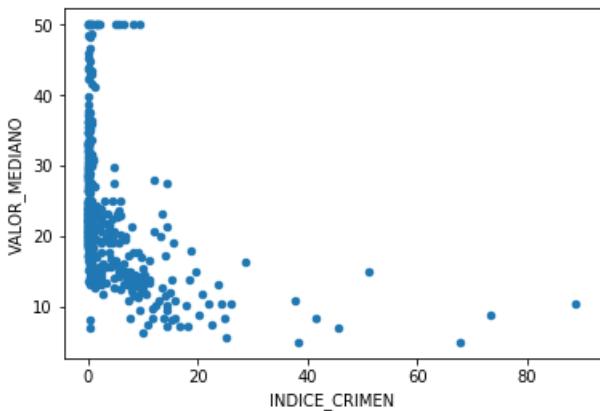
```
#Este comando adapta la grafica al documento
%matplotlib notebook
%matplotlib inline
```

In [24]:

```
df.plot.scatter(x="INDICE_CRIMEN", y="VALOR_MEDIANO")
```

Out [24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d26348208>
```



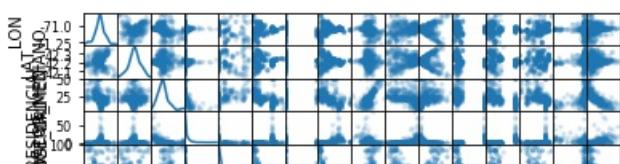
Matriz de dispersión

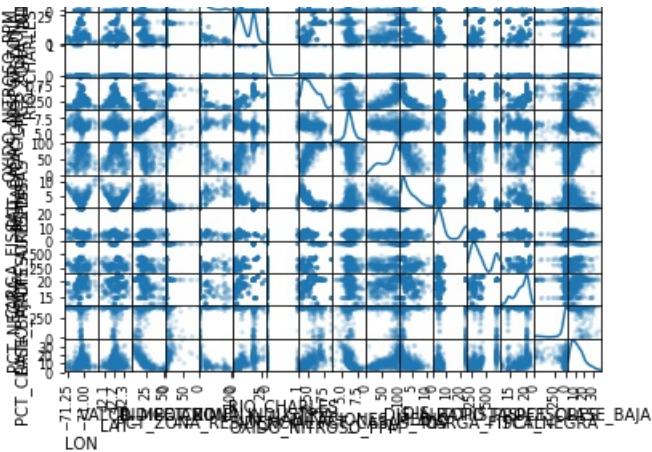
Un gráfico de matriz de dispersión es una herramienta de exploración de datos que permite comparar varios datasets para buscar patrones y relaciones.

El gráfico de matriz de dispersión (SPM) acepta una capa o una tabla como entrada. Seleccione los campos que deseé utilizar en el gráfico. El gráfico tiene dos componentes principales: una matriz de gráficos de dispersión pequeños para cada uno de los campos y una ventana Vista previa mayor que muestra el gráfico de dispersión para un par de campos seleccionados con mayor detalle. También puede habilitar el trazado de histogramas, mostrando la distribución de valores para cada uno de los campos.

In [25]:

```
from pandas.plotting import scatter_matrix
sm = scatter_matrix(df, alpha=0.2, figsize=(6, 6), diagonal='kde')
```





In [26]:

sm

Out [26]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d262aa8d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2640db38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2eeb2f98>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2ebdf0b8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2627f748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2627f780>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d262584a8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d26201b38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d261b3208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d261da898>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d26185f28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d261355f8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2615bc88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2610f358>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d260b79e8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d260680b8>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2608f748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d26039dd8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25fe94a8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2600fb38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25fc4208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25f6d898>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25f96f28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25f475f8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25eec88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25ea0358>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25ec89e8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25e7a0b8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25e22748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25e4ddd8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25dfb4a8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25da3b38>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25dd8208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25d7e898>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25d27f28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25d595f8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25d02c88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25cb3358>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25cdc9e8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25c8e0b8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25c36748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25c5ddd8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25c0e4a8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25bb5b38>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25b69208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25b91898>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25b3bf28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25aeb5f8>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25b13c88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25ac4358>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25a6c9e8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25a9f0b8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d25a48748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d259f2dd8>,
```



```

[<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23f35748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23f5ddd8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23f0d4a8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23eb4b38>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23e6a208>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23e90898>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23e3bf28>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23dec5f8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23e13c88>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23dc4358>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23d6d9e8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23d9e0b8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23d46748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23cefdd8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23ca14a8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23cc8b38>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23c7b208>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23c21898>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23c4bf28>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23bfc5f8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23ba4c88>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23bd5358>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23b7f9e8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23b310b8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23b59748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23b00dd8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23ab34a8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23addb38>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23a8c208>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23a35898>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23a5ef28>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23a0d5f8>],
 [<matplotlib.axes._subplots.AxesSubplot object at 0x7f8d239b6c88>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2396b358>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d239909e8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d239410b8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d238ed748>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23913dd8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d238c44a8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2386eb38>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2389f208>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d23846898>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d237eef28>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2381f5f8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d237c6c88>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d2377a358>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d237239e8>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f8d237550b8>],
 dtype=object)

```

Este es uno de los problemas de matplotlib, que su api es bastante complicada si quieres hacer algo que se salga de lo corriente. Por ejemplo, en este caso necesitamos un montón de código para rotar las etiquetas de los ejes

In [27]:

```

sm = scatter_matrix(df, alpha=0.2, figsize=(14, 14), diagonal='kde')

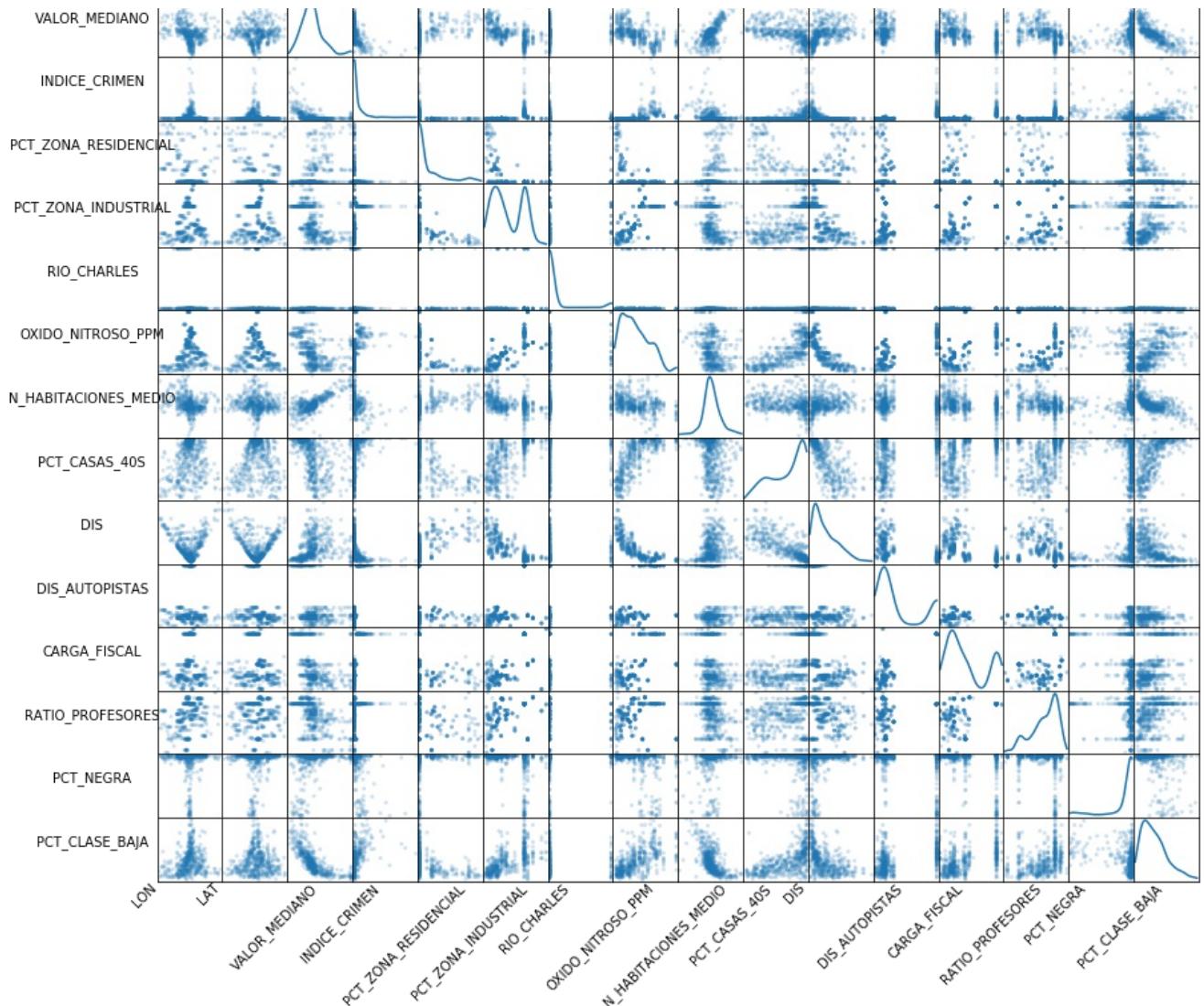
https://stackoverflow.com/questions/32560932/how-to-customize-a-scatter-matrix-to-see-all-titles
#Change label rotation
[s.xaxis.label.set_rotation(45) for s in sm.reshape(-1)]
[s.yaxis.label.set_rotation(0) for s in sm.reshape(-1)]

#May need to offset label when rotating to prevent overlap of figure
[s.get_yaxis().set_label_coords(-1,0.5) for s in sm.reshape(-1)]
[s.get_xaxis().set_label_coords(-0.2,0) for s in sm.reshape(-1)]

#Hide all ticks
[s.set_xticks([]) for s in sm.reshape(-1)]
[s.set_yticks([]) for s in sm.reshape(-1)];

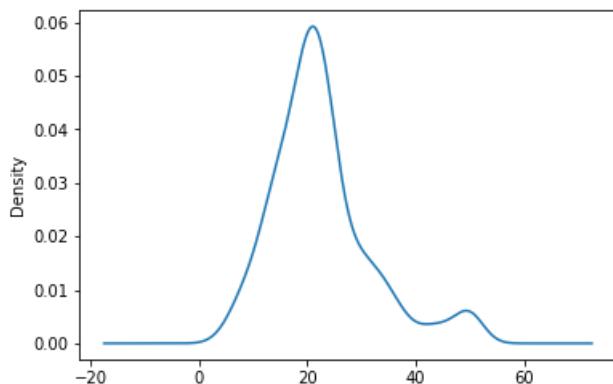
```





In [28]:

```
df.VALOR_MEDIANO.plot.kde();
```



Histograma

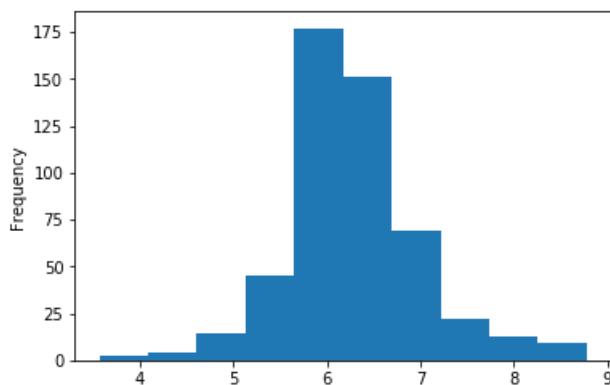
Los histogramas se usan para representar la distribución de una variable, esto es, que rango de valores tiene, cuales son los valores más comunes Segun [Wikipedia](#) un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. Sirven para obtener una "primera vista" general, o panorama, de la distribución de la población, o de la muestra, respecto a una característica, cuantitativa y continua (como la longitud o el peso). De esta manera ofrece una visión de grupo permitiendo observar una preferencia, o tendencia, por parte de la muestra o población por ubicarse hacia una determinada región de valores dentro del espectro de valores posibles (sean infinitos o no) que pueda adquirir la característica. Así pues, podemos evidenciar comportamientos, observar el grado de homogeneidad, acuerdo o concisión entre los valores de todas las partes que componen la población o la muestra, o, en contraposición, poder observar el grado de variabilidad, y por ende, la dispersión de todos los valores que toman las partes, también es posible no evidenciar ninguna tendencia y obtener que cada miembro de la población toma por su lado y adquiere un valor de la característica aleatoriamente sin mostrar ninguna preferencia o tendencia, entre otras cosas.

In [29]:

```
df.N_HABITACIONES_MEDIO.plot.hist()
```

Out [29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d19526780>
```



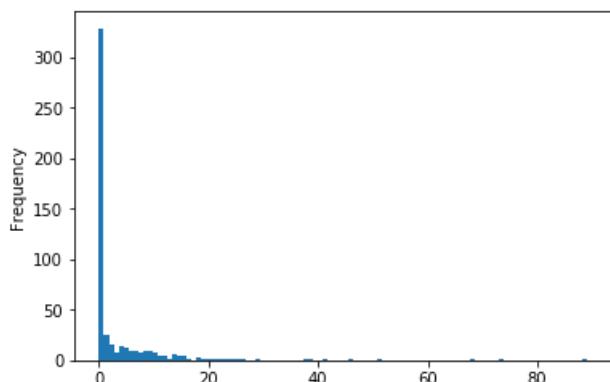
podemos especificar cuantos grupos queremos en el histograma

In [30]:

```
df.INDICE_CRIMEN.plot.hist(bins=100)
```

Out [30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d194be7f0>
```



Tambien podemos filtrar el gráfico poniendo límites a los ejes

In [31]:

```
df.INDICE_CRIMEN.plot.hist(bins=100, xlim=(0,20))
```

Out [31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d199e8be0>
```

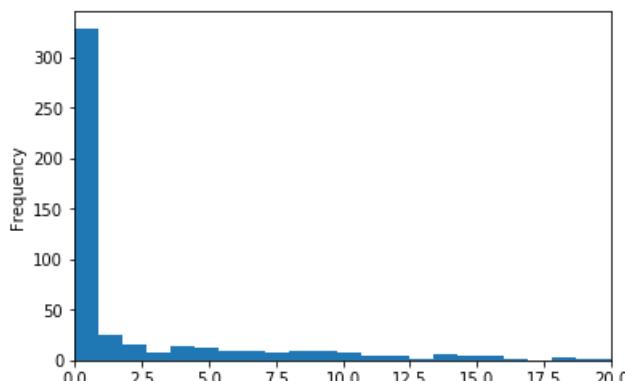


Gráfico de barras/columnas

Los gráficos de barras se utilizan comúnmente para representar y comparar una variable entre distintos grupos

In [32]:

```
valor_por_ciudad = df.groupby("CIUDAD") ["VALOR_MEDIANO"].mean()  
valor_por_ciudad.head()
```

Out [32]:

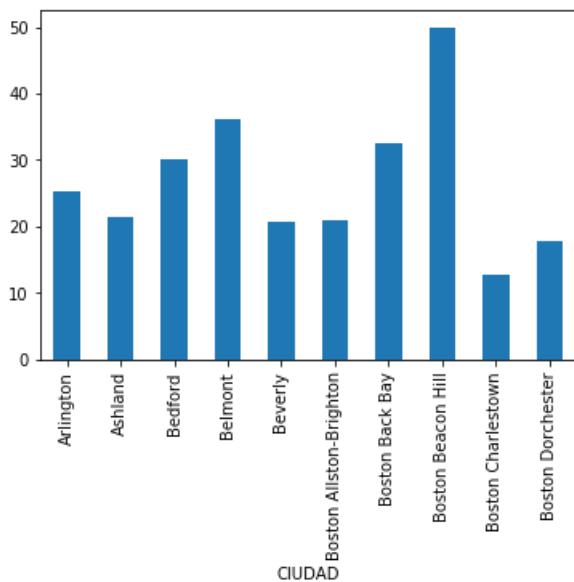
```
CIUDAD  
Arlington    25.2  
Ashland      21.4  
Bedford       30.1  
Belmont       36.2  
Beverly       20.8  
Name: VALOR_MEDIANO, dtype: float64
```

In [33]:

```
valor_por_ciudad.head(10).plot.bar()
```

Out [33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d1980c208>
```



In [34]:

```
valor_por_ciudad.head(10).plot.bart()
```

Out [34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d19860080>
```

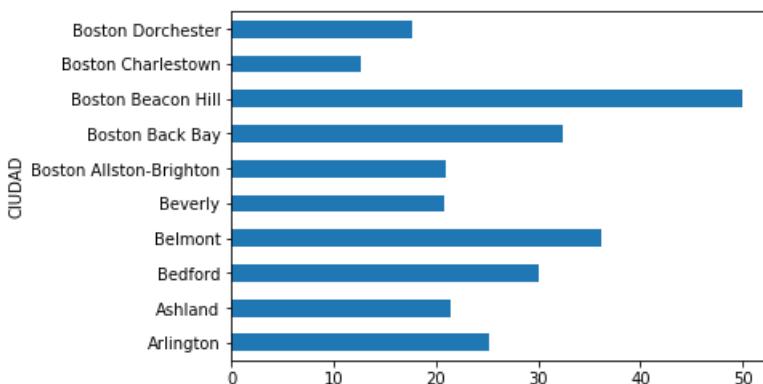


Gráfico de linea

Los gráficos de linea se usan principalmente para representar tendencias, esto es, se usan para variables que varian con el tiempo

In [35]:

```
df.groupby("RATIO_PROFESORES").VALOR_MEDIANO.mean().plot.line();
```

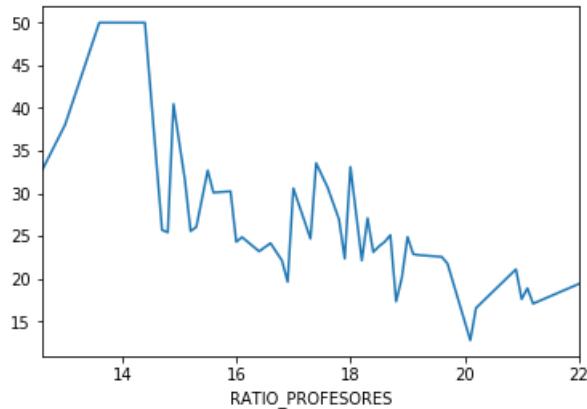


Diagrama de caja (Box Plot)

Los diagramas de caja son útiles a la hora de representar grupos de datos y comparar entre ellos. Otra ventaja de los boxplots es que identifican de forma sencilla si una variable tiene muchos outliers, esto es, elementos que se alejan de los valores frecuentes de dicha variable.

In [36]:

```
df[ "VALOR_CUANTILES" ] = pd.qcut(df.VALOR_MEDIANO, 5)
```

In [37]:

```
df.boxplot(column="INDICE_CRIMEN", by="VALOR_CUANTILES", figsize=(10,10))
```

Out [37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d19738b38>
```

Boxplot grouped by VALOR_CUANTILES

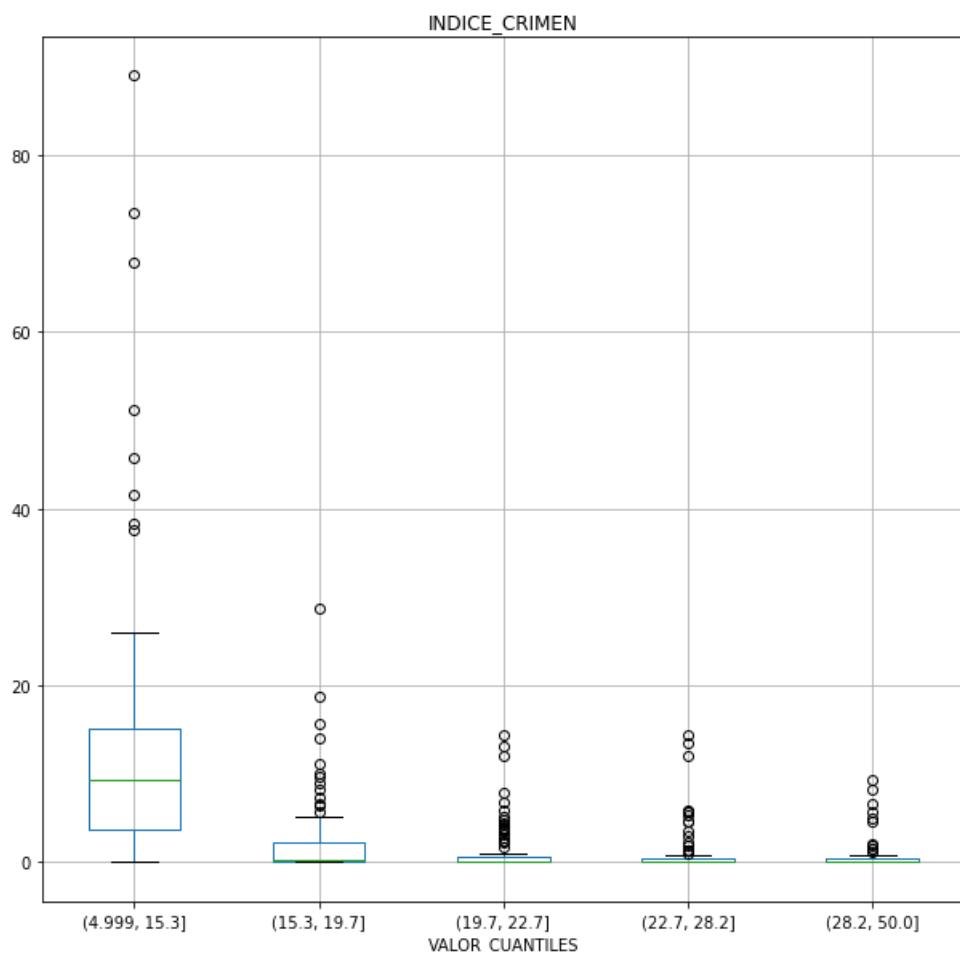


Gráfico circular

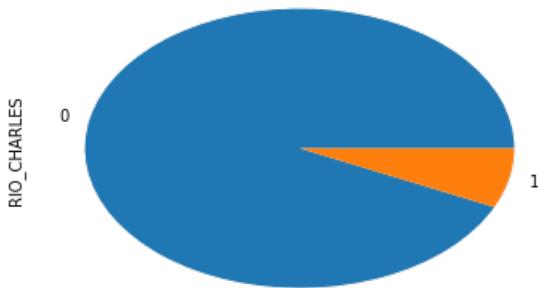
Un gráfico circular es una representación gráfica de una serie de cantidades y consiste en un círculo dividido en varios sectores, cuyo tamaño se corresponde con las proporciones de las cantidades. Básicamente, este tipo de gráfico muestra la relación porcentual entre las partes con relación a su conjunto.

In [38]:

```
df.RIO_CHARLES.value_counts().plot.pie()
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d1912b588>
```



<https://conda-forge.github.io/>

<https://anaconda.org/conda-forge/repo>

<https://seaborn.pydata.org/>

<http://bokeh.pydata.org/en/latest/>

<https://altair-viz.github.io/>

<https://plot.ly/python/getting-started/>

<https://ipywidgets.readthedocs.io/en/latest/examples/Using%20Interact.html> conda install -c conda-forge ipywidgets

https://github.com/bokeh/bokeh/blob/0.12.5/examples/howto/notebook_comms/Jupyter%20Interactors.ipynb

```
In [1]:
```

```
%load_ext watermark
%watermark
%matplotlib inline
```

```
2019-05-30T21:24:43+02:00
```

```
CPython 3.6.5
IPython 6.4.0
```

```
compiler    : GCC 7.2.0
system      : Linux
release     : 5.1.5-arch1-2-ARCH
machine     : x86_64
processor   :
CPU cores   : 4
interpreter: 64bit
```

VISUALIZACIÓN DE DATOS AVANZADA

En este capítulo se trataran los métodos para poder personalizar las gráficas de Matplotlib y como cargar Estilos ya predefinidos. Además se realizaran ejemplos con otras librerías interesantes para la visualización de datos:

- IPyWidgets.
- Cartopy.
- Seaborn.
- BokehJS

Carga de Datos y Preparacion de DataSet

Como en el apartado de visualización básica de datos utilizaremos el Boston Housing Dataset. Recopilado en 1976 y publicado en [Berkeley](#)

```
In [ ]:
```

```
import pandas as pd
df = pd.read_csv("boston_dataset.csv")

#renombramos las variables
df = df.rename(columns={
    "TOWN": "CIUDAD",
    "CRIM": "INDICE_CRIMEN",
    "ZN": "PCT_ZONA_RESIDENCIAL",
    "INDUS": "PCT_ZONA_INDUSTRIAL",
    "CHAS": "RIO_CHARLES",
    "NOX": "OXIDO_NITROSO_PPM",
    "RM": "N_HABITACIONES_MEDIO",
    "AGE": "PCT_CASAS_40S",
    "DIS_EMPLEO": "DISTANCIA_CENTRO_EMPLEO",
    "RAD": "DIS_AUTOPISTAS",
    "TAX": "CARGA_FISCAL",
    "PTRATIO": "RATIO_PROFESORES",
    "B": "PCT_NEGRA",
    "MEDV": "VALOR_MEDIANO",
    "LSTAT": "PCT_CLASE_BAJA"
})

df.head()
```

```
Out [ ]:
```

	CIUDAD	LON	LAT	VALOR_MEDIANO	INDICE_CRIMEN	PCT_ZONA_RESIDENCIAL	PCT_ZONA_INDUSTRIAL
0	Nahant	-70.955	42.2550	24.0	0.00632	18.0	2.31
1	Swampscott	-70.950	42.2875	21.6	0.02731	0.0	7.07

2	SUPERFICIE	LON	VALOR_MEDIANO	INDICE_CRIMEN	POT_ZONA_RESIDENCIAL	POT_ZONA_INDUSTRIAL
2	Swampscott	-70.936	42.2880			
3	Marblehead	-70.928	42.2930	33.4	0.03237	0.0
4	Marblehead	-70.922	42.2980	36.2	0.06905	2.18

Personalización de Gráficos

Mediante los métodos de la librería **PyPlot** de **Matplotlib** veremos como especificar títulos para los gráficos y como personalizar la forma de los punteros en gráficos, su tamaño y su color.

In [3]:

```
%matplotlib notebook
```

In [4]:

```
import matplotlib.pyplot as plt
```

In [5]:

```
#Cambiamos el punto con marker, color y el tamaño tan solo llamando a los parametros
df.plot.scatter(x="N_HABITACIONES_MEDIO", y="VALOR_MEDIANO", marker="*", color="pink", figsize=(8,8))

plt.title("Relacion entre el numero de habitaciones y el valor de las viviendas")

plt.xlabel("Numero medio de habitaciones")

plt.ylabel("Valor mediano de las viviendas ($1000s)")
```

Out [5]:

```
Text(0,0.5,'Valor mediano de las viviendas ($1000s)')
```

Damos un tamaño para las figuras por defecto en las librerías

```
In [6]:
```

```
import matplotlib as mpl  
  
mpl.rcParams['figure.figsize'] = (8,8)
```

```
In [7]:
```

```
df.plot.scatter(x="N_HABITACIONES_MEDIO", y="VALOR_MEDIANO", marker="*", color="pink")  
  
plt.title("Relacion entre el numero de habitaciones y el valor de las viviendas")  
  
plt.xlabel("Numero medio de habitaciones")  
  
plt.ylabel("Valor mediano de las viviendas ($1000s)")
```

```
Out[7]:
```

```
Text(0,0.5,'Valor mediano de las viviendas ($1000s)')
```

Establecer estilos en Matplotlib

Por defecto Matplotlib tiene un estilo definido, un aspecto muy característico y fácilmente reconocible. Pero también permite personalizar los estilos de gráficas de una forma muy sencilla, utilizando hojas de estilos predefinidas y que vienen incluidas con Matplotlib

```
In [8]:
```

```
#mostramos la lista disponible de estilos en pyplot.  
plt.style.available
```

```
Out[8]:
```

```
['fivethirtyeight',  
'bmh',  
'seaborn-talk',  
'seaborn-notebook',  
'seaborn',  
'seaborn-whitegrid',  
'ggplot',  
'tableau-colorblind10',
```

```
'dark_background',
'seaborn-deep',
'_classic_test',
'seaborn-colorblind',
'seaborn-darkgrid',
'seaborn-poster',
'grayscale',
'seaborn-dark',
'seaborn-paper',
'fast',
'seaborn-pastel',
'seaborn-white',
'seaborn-ticks',
'classic',
'seaborn-bright',
'Solarize_Light2',
'seaborn-muted',
'seaborn-dark-palette']
```

Podemos encontrar mas estilos [aqui](#)

In [9]:

```
plt.style.use("fivethirtyeight")
```

In [10]:

```
df.plot.scatter(x="N_HABITACIONES_MEDIO", y="VALOR_MEDIANO")

plt.title("Relacion entre el numero de habitaciones y el valor de las viviendas")

plt.xlabel("Numero medio de habitaciones")

plt.ylabel("Valor mediano de las viviendas ($1000s)")
```

Out[10]:

```
Text(0,0.5,'Valor mediano de las viviendas ($1000s)')
```

IPyWidgets

[IpyWidgets](#) es una librería que nos permite importar widgets FrontEnd para poder interactuar con las gráficos. Podemos invocarlo con interact.

In [11]:

```
from ipywidgets import interact
```

In [12]:

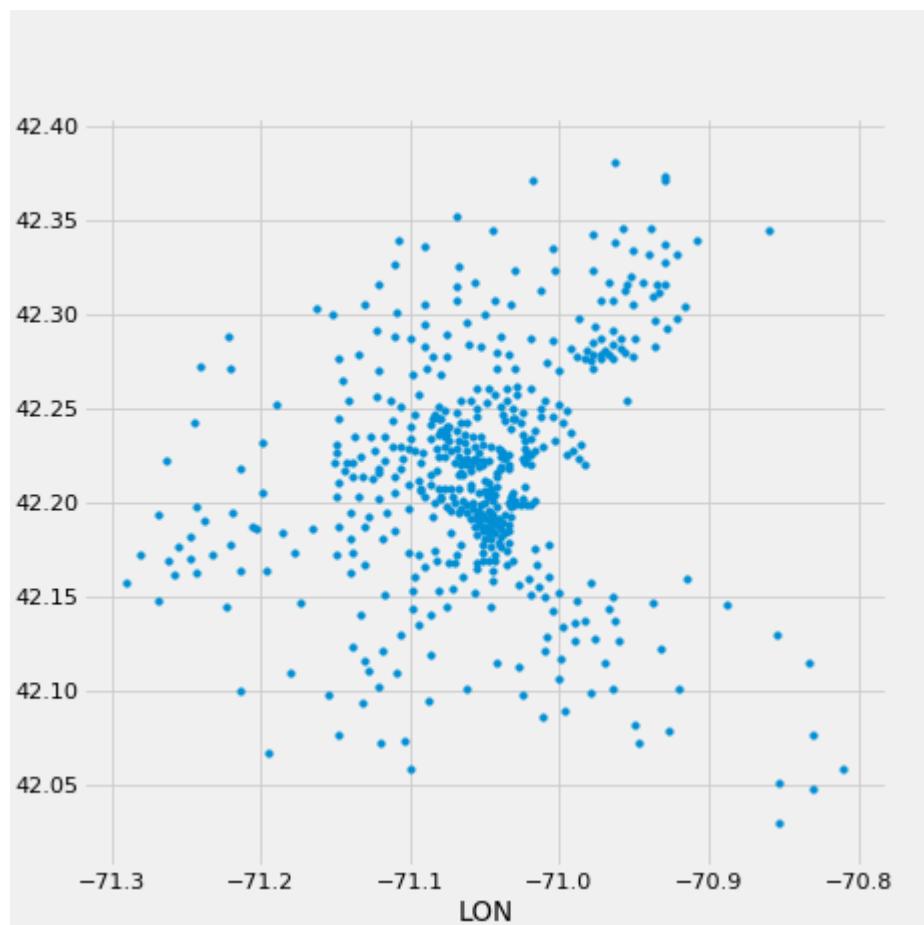
```
#Creamos la función grafico variable para comparar la columna 1 seleccionable desde el ComboBox con el Valor Mediano
@interact(coll=df.columns.tolist())
def grafico_variable(coll):
    df.plot.scatter(x=coll, y="VALOR_MEDIANO")
    plt.title("{} vs VALOR_MEDIANO".format(coll))
```

In [13]:

```
#Indicamos a matplotlib que estamos trabajando con notebook para que se reescale mejor.
%matplotlib notebook
```

In [17]:

```
df.plot.scatter(x="LON", y="LAT")
```



Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8027c7400>
```

Cartopy

[Cartopy](#) es una librería diseñada para procesar datos geoespaciales en orden para "plotear" mapas y poder realizar análisis de datos.

In [18]:

```
import cartopy.crs as ccrs
from cartopy.io import img_tiles
```

```

ModuleNotFoundError                                     Traceback (most recent call last)
<ipython-input-18-fcalb84c8aa4> in <module>()
----> 1 import cartopy.crs as ccrs
      2
      3 from cartopy.io import img_tiles

ModuleNotFoundError: No module named 'cartopy'

In [19]:
df.VALOR_MEDIANO.plot.kde()

Out[19]:
<matplotlib.axes._subplots.AxesSubplot at 0x7fd8029c4208>

In [20]:
primer_quintil = df.VALOR_MEDIANO.quantile(0.2)
primer_quintil

Out[20]:
15.3

In [21]:
cuarto_quintil = df.VALOR_MEDIANO.quantile(0.8)
cuarto_quintil

Out[21]:
28.2

In [22]:
imagery = img_tiles.GoogleTiles()

ax = plt.axes(projection=imagery.crs)

limites_mapa = (-71.38 , -70.77, 42.03 , 42.47)

ax.set_extent(limites_mapa)

ax.add_image(imagery, 10)

df_primer_qt = df[df.VALOR_MEDIANO<primer_quintil]
df_tercer_qt = df[df.VALOR_MEDIANO>cuarto_quintil]

plt.plot(df_primer_qt.LON, df_primer_qt.LAT, transform=ccrs.Geodetic(), marker=".",
          markersize=10, color="red", linewidth=0, alpha=0.5)

plt.plot(df_tercer_qt.LON, df_primer_qt.LAT, transform=ccrs.Geodetic(), marker=".",
          markersize=10, color="green", linewidth=0, alpha=0.5)

plt.show()

NameError                                         Traceback (most recent call last)
<ipython-input-22-e458cdb2207d> in <module>()
----> 1 imagery = img_tiles.GoogleTiles()
      2
      3
      4 ax = plt.axes(projection=imagery.crs)
      5

NameError: name 'img_tiles' is not defined

```

Seaborn

Basada en matplotlib, se usa para hacer más atractivos los gráficos e información estadística en Python. Su objetivo es darle una mayor relevancia a las visualizaciones, dentro de las tareas de exploración e interpretación de los datos.

```
In [ ]:
```

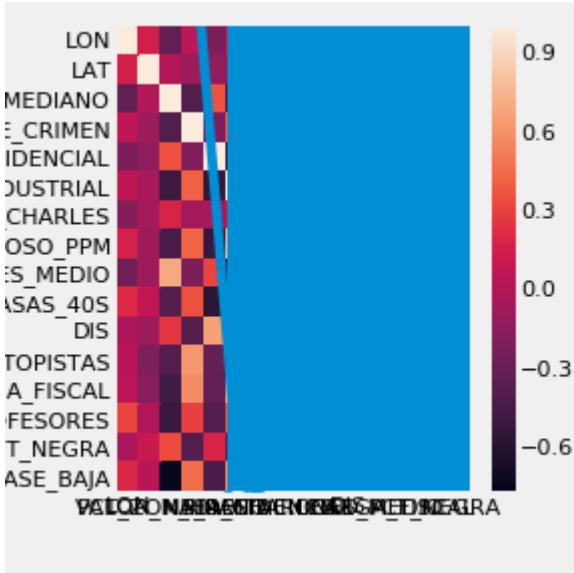
```
#especificamos a matplotlib para incluir los gráficos en el notebook  
%matplotlib inline
```

```
In [23]:
```

```
import seaborn as sns
```

```
In [24]:
```

```
sns.heatmap(x="N_HABITACIONES_MEDIO", y="VALOR_MEDIANO", data=df)
```



```
Out[24]:
```

```
<seaborn.axisgrid.FacetGrid at 0x7fd7f99a5a58>
```

```
In [25]:
```

```
sns.heatmap(df.corr())
```

```
Out[25]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7f92aa668>
```

BokehJS

[BokehJS](#) es una librería que nos permitirá realizar graficas pensadas para mostrar gráficos en un navegador. Bokeh es una librería para visualizaciones interactivas diseñada para funcionar en los navegadores web modernos. Su objetivo es proporcionar una construcción elegante y concisa de gráficos modernos al estilo de D3.js, y para ampliar esta capacidad con la interactividad y buen rendimiento sobre grandes volúmenes de datos. Bokeh puede ayudar a cualquier persona a crear en forma rápida y sencilla gráficos interactivos, dashboards y aplicaciones de datos

```
In [26]:
```

```
import bokeh.plotting as bk  
  
bk.output_notebook()
```

Loading BokehJS ...

```
In [27]:
```

```
df.INDICE_CRIMEN
```

```
Out[27]:
```

```
0      0.00632  
1      0.02731  
2      0.02729  
3      0.03237  
4      0.06905  
5      0.02985  
6      0.08829
```

```
7      0.14455
8      0.21124
9      0.17004
10     0.22489
11     0.11747
12     0.09378
13     0.62976
14     0.63796
15     0.62739
16     1.05393
17     0.78420
18     0.80271
19     0.72580
20     1.25179
21     0.85204
22     1.23247
23     0.98843
24     0.75026
25     0.84054
26     0.67191
27     0.95577
28     0.77299
29     1.00245
...
476    4.87141
477    15.02340
478    10.23300
479    14.33370
480    5.82401
481    5.70818
482    5.73116
483    2.81838
484    2.37857
485    3.67367
486    5.69175
487    4.83567
488    0.15086
489    0.18337
490    0.20746
491    0.10574
492    0.11132
493    0.17331
494    0.27957
495    0.17899
496    0.28960
497    0.26838
498    0.23912
499    0.17783
500    0.22438
501    0.06263
502    0.04527
503    0.06076
504    0.10959
505    0.04741
Name: INDICE_CRIMEN, Length: 506, dtype: float64
```

In [28]:

```
df["CRIMEN_QUINTIL"] = pd.qcut(df.INDICE_CRIMEN, 5)
```

In [29]:

```
df.CRIMEN_QUINTIL.cat.categories
```

Out [29]:

```
IntervalIndex([(0.00532, 0.0642], (0.0642, 0.15], (0.15, 0.55], (0.55, 5.581], (5.581, 88.976]),
              closed='right',
              dtype='interval[float64]')
```

In [30]:

```
from bokeh.palettes import brewer

colors = brewer["Spectral"][len(df.CRIMEN_QUINTIL.unique())]
colors
```

Out [30]:

```
['#2b83ba', '#abdda4', '#ffffbf', '#fdae61', '#d7191c']
```

In [31]:

```
p = bk.figure(  
    plot_width=600,  
    plot_height=600,  
    title="Habitaciones vs Valor vivienda vs crimen"  
)  
  
for i, quintil in enumerate(df.CRIMEN_QUINTIL.cat.categories):  
    df_q = df[df.CRIMEN_QUINTIL==quintil]  
    p.scatter(df_q.N_HABITACIONES_MEDIO, df_q.VALOR_MEDIANO, color=colors[i],  
              legend="{}-{}".format(quintil.left, quintil.right))  
  
bk.show(p);
```

In [32]:

```
df.VALOR_MEDIANO.plot.hist()
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd7f92aa668>
```

In [33]:

```
import numpy as np  
  
hist, edges = np.histogram(df.VALOR_MEDIANO, bins=20)
```

In [34]:

```
hist
```

Out[34]:

```
array([ 9, 12, 18, 36, 41, 41, 84, 71, 72, 12, 23, 18, 17, 14, 6, 1, 5,  
      5, 2, 19])
```

In [35]:

```
edges
```

Out[35]:

```
array([ 5. , 7.25, 9.5 , 11.75, 14. , 16.25, 18.5 , 20.75, 23. ,  
      25.25, 27.5 , 29.75, 32. , 34.25, 36.5 , 38.75, 41. , 43.25,  
      45.5 , 47.75, 50. ])
```

In [36]:

```
p1 = bk.figure(title="Histograma valor viviendas", tools="save,hover", background_fill_color="#E8DD  
CB")  
  
p1.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], fill_color="#026560")  
bk.show(p1)
```

```
In [1]:
```

```
%load_ext watermark  
%watermark  
%matplotlib inline
```

```
2019-05-30T21:28:39+02:00
```

```
CPython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Analisis exploratorio de Datos

Ingesta y procesado inicial

En esta sección vamos a hacer un ejemplo completo de como hacer un Análisis Exploratorio de Datos (EDA en inglés).

Hoy en dia no hay un proceso standard a la hora de realizar EDA, pero un proceso que a mi me gusta está basado en el proceso propuesto por Distric Data Labs ([aquí](#) un blog post, y [aquí](#) una charla sobre el proceso).

El archivo original está en: <https://www.fueleconomy.gov/feg/epadata/vehicles.csv.zip> El archivo que vamos a usar es una versión modificada (con menos columnas)

Descripcion del dataset <http://www.fueleconomy.gov/feg/ws/index.shtml#ft7>

Supongamos que somos la agencia de portecion ambiental americana, la EPA. Uno de sus trabajos es analizar los coches nuevos que se venden en EEUU y estudiar su contaminacion.

```
In [2]:
```

```
import pandas as pd
```

```
In [3]:
```

```
%matplotlib notebook
```

Lectura de datos

```
In [4]:
```

```
vehiculos = pd.read_csv("data/vehiculos-original.csv")
```

```
In [5]:
```

```
vehiculos.shape
```

```
Out[5]:
```

```
(38436, 11)
```

```
In [6]:
```

```
vehiculos.head()
```

```
Out[6]:
```

	make	model	year	displ	cylinders	trany	drive	VClass	fuelType	comb08	co2TailpipeGpm
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706

	make	model	year	displ	cylinders	trany	drive	VClass	fuelType	comb08	co2TailpipeGpm
1	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
2	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
3	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
4	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500

Es conveniente renombrar las columnas y darles un nombre descriptivo

In [7]:

```
vehiculos = vehiculos.rename(columns={
    "cylinders": "cilindros",
    "trany": "transmision",
    "make": "fabricante",
    "model": "modelo",
    "displ": "desplazamiento", #volumen de desplazamiento del motor
    "drive": "traccion",
    "VClass": "clase",
    "fuelType": "combustible",
    "comb08": "consumo", #combined MPG for fuelType1
    "co2TailpipeGpm": "co2", # tailpipe CO2 in grams/mile
})
```

In [8]:

```
vehiculos.head()
```

Out[8]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co2
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
2	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
3	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
4	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500

In [9]:

```
vehiculos.dtypes
```

Out[9]:

fabricante	object
modelo	object
year	int64

```
desplazamiento      float64
cilindros          float64
transmision         object
traccion            object
clase               object
combustible         object
consumo             int64
co2                 float64
dtype: object
```

¿Cual es el objetivo de este análisis?

Un aspecto importante que me gusta siempre tener claro al empezar un EDA es saber el objetivo del mismo. Generalmente hay una lista de preguntas concretas que responder, o al menos las personas a cargo de recolectar el dataset lo crearon con un objetivo en mente.

En este caso, uno de los objetivos por los cuales la EPA crea este dataset es para controlar lo que contamina cada coche, en este caso la variable co2

Descripción de la entidad.

- fabricante
- fabricante-modelo
- fabricante-model-año
- fabricante-año

Exportación a csv:

In [10]:

```
vehiculos.to_csv("data/vehiculos.1.procesado_inicial.csv", index=False)
```

Diagnóstico de calidad de los datos (QA)

matplotlib notebook, es muy util para hacer plots mas visibles, pero es un poco complicada de usar ya que requiere el cerrar cada plot para poder continuar.

Alternativamente, se puede usar %matplotlib inline, que es más sencilla ya que simplemente muestra el gráfico original en el jupyter notebook. Para modificar el tamaño de los plots en este caso basta con cambiar el parámetro general de matplotlib figure.figsize al tamaño de gráfico deseado (en pulgadas)

In [11]:

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12,12) # 10 pulgadas de ancho y 10 de alto para todos los plots
```

En este apartado vamos a revisar el dataset. El objetivo sera tener un dataset lo mas parecido al original pero sin fallos en el mismo que puedan llevarnos a sacar conclusiones erroneas.

El input de este paso es el output del anterior

In [12]:

```
vehiculos = pd.read_csv("data/vehiculos.1.procesado_inicial.csv")
```

In [13]:

```
vehiculos.shape
```

Out[13]:

```
(38436, 11)
```

Duplicados

En este apartado se intentan ver dos cosas:

- Asegurarse de que datos que no debieron estar duplicados no lo están (por ejemplo, números de teléfono)

- Asegurarse de que datos que no deberian estar duplicados no lo estan (por ejemplo, numeros de telefono)
- Revisar si hay columnas que tengan un alto numero de duplicados, ya que pueden no aportar mucha informacion

Recordemos que las entidades posibles eran:

fabricante
fabricante-modelo
fabricante-modelo-año
fabricante-año

De estas entidades, las unicas que no se deberian repetir coches especificos, es decir, fabricante+modelo +año

In [14]:

```
vehiculos['modelo_unico'] = vehiculos.fabricante.str.cat([vehiculos.modelo,
vehiculos.year.apply(str)], sep=' - ')
```

In [15]:

```
vehiculos.modelo_unico.value_counts()
```

Out[15]:

Jeep-Cherokee/Wagoneer-1985	24
Ford-F150 Pickup 2WD-1984	19
GMC-C15 Pickup 2WD-1984	19
Chevrolet-C10 Pickup 2WD-1984	19
GMC-C15 Pickup 2WD-1985	18
GMC-S15 Pickup 2WD-1984	18
Chevrolet-C10 Pickup 2WD-1985	18
Chevrolet-S10 Pickup 2WD-1984	18
Dodge-Ram 50 Pickup 2WD-1984	17
Chevrolet-C10 Pickup 2WD-1986	17
Ford-Ranger Pickup 2WD-1984	17
Mitsubishi-Truck 2WD-1984	17
GMC-C15 Pickup 2WD-1986	17
Volkswagen-Rabbit-1984	16
Ford-Escort-1984	16
Ford-Escort-1985	16
Mercury-Lynx-1984	16
Ford-F150 Pickup 2WD-1985	16
Chevrolet-G10/20 Van 2WD-1984	16
GMC-Vandura G15/25 2WD-1984	16
Volkswagen-Jetta-1984	15
Ford-F250 Pickup 2WD-1985	15
Dodge-D100/D150 Pickup 2WD-1985	15
Ford-Bronco 4WD-1984	15
Ford-E150 Econoline 2WD-1984	15
Ford-F150 Pickup 4WD-1984	15
Volvo-240 DL/GL/Turbo Wagon-1984	14
Chevrolet-C1500 Pickup 2WD-1991	14
Chevrolet-G10/20 Van 2WD-1986	14
Chevrolet-S10 Pickup 2WD-1985	14
..	
Volvo-XC70 AWD-2008	1
Mercedes-Benz-E63 AMG 4matic (wagon)-2014	1
Toyota-Sienna 2WD-2006	1
Chevrolet-Caprice/Impala Wagon-1996	1
Bentley-Continental Flying Spur-2007	1
Porsche-911 Turbo-2017	1
Subaru-Justy 4WD-1988	1
Toyota-RAV4 Hybrid AWD-2016	1
Toyota-Highlander Hybrid 4WD-2008	1
Volvo-XC60 FWD-2010	1
Kia-Sedona-2005	1
Cadillac-Funeral Coach / Hearse-2005	1
Mcevoy Motors-240 DL/240 GL Sedan-1987	1
Lotus-Espirit V8-1997	1
BMW-X3 xDrive28i-2014	1
Mercury-Grand Marquis-1989	1
Lexus-IS F-2009	1
BMW-640i xDrive Gran Coupe-2015	1
Dodge-Avenger AWD-2008	1
Toyota-Sienna AWD-2016	1
Mercedes-Benz-500E-1993	1
Roush Performance-Stage 3 Mustang Coupe-2007	1

```

Mercedes-Benz-E320 Wagon-1995          1
Mercedes-Benz-E320 (Wagon)-2002         1
Lexus-LS 430-2006                      1
Vector-Avtech SC / M12-1996            1
Maserati-Ghibli V6-2017                1
Ford-F150 Dual-fuel 2WD (LPG)-2002     1
GMC-Safari 2WD (passenger)-1991        1
Volkswagen-Touareg-2011                1
Name: modelo_unico, Length: 17448, dtype: int64

```

Nos damos cuenta de que hay muchos repetidos, veamos un ejemplo

In [16]:

```
vehiculos[vehiculos.modelo_unico=='Chevrolet-C1500 Pickup 2WD-1991'].head()
```

Out [16]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	
4957	Chevrolet	C1500 Pickup 2WD	1991	4.3	6.0	Automatic 4-spd	Rear-Wheel Drive	Standard Pickup Trucks	Regular	17	522.764
4958	Chevrolet	C1500 Pickup 2WD	1991	4.3	6.0	Manual 4-spd	Rear-Wheel Drive	Standard Pickup Trucks	Regular	17	522.764
4959	Chevrolet	C1500 Pickup 2WD	1991	4.3	6.0	Manual 5-spd	Rear-Wheel Drive	Standard Pickup Trucks	Regular	17	522.764
4960	Chevrolet	C1500 Pickup 2WD	1991	4.3	6.0	Manual 5-spd	Rear-Wheel Drive	Standard Pickup Trucks	Regular	17	522.764
4961	Chevrolet	C1500 Pickup 2WD	1991	5.0	8.0	Automatic 4-spd	Rear-Wheel Drive	Standard Pickup Trucks	Regular	15	592.466

Como vemos , cada modelo unico tiene diferentes configuraciones de cada coche (3 velocidades o 4 por ejemplo) Por lo tanto, solo vamos a considerar duplicados aquellos records que sean idénticos en todas sus columnas

In [17]:

```
vehiculos[vehiculos.duplicated()].shape
```

Out [17]:

```
(1506, 12)
```

Vemos que hay 1506 records duplicados, podemos eliminarlos para el resto del análisis ya que pueden distorsionar las conclusiones

In [18]:

```
vehiculos = vehiculos.drop_duplicates()
vehiculos.shape
```

Out [18]:

```
(36930, 12)
```

Borramos la columna modelo_unico

In [19]:

```
del vehiculos['modelo_unico']
```

Ahora falta ver si hay variables en las que haya una gran cantidad de registros que tengan el mismo valor (cardinalidad).

In [20]:

```
n_records = len(vehiculos)
```

```

def valores_duplicados_col(df):
    for columna in df:
        n_por_valor = df[columna].value_counts()
        mas_comun = n_por_valor.iloc[0]
        menos_comun = n_por_valor.iloc[-1]
        print("{} | {}-{} | {}".format(
            df[columna].name,
            round(mas_comun / (1.0*n_records),3),
            round(menos_comun / (1.0*n_records),3),
            df[columna].dtype
        )))

```

valores_duplicados_col(vehiculos)

Categoría	Porcentaje	Dato
fabricante	0.1-0.0	object
modelo	0.005-0.0	object
year	0.038-0.007	int64
desplazamiento	0.095-0.0	float64
cilindros	0.38-0.0	float64
transmision	0.287-0.0	object
traccion	0.353-0.005	object
clase	0.145-0.0	object
combustible	0.652-0.0	object
consumo	0.097-0.0	int64
co2	0.084-0.0	float64

Vemos que los campos traccion, transmission, cilindros y combustible pueden tener un problema de valores repetidos, en realidad 30% no es algo muy dramático, sería distinto si el valor más común tuviera un 90%.

In [21]:

```
vehiculos.traccion.value_counts(normalize=True)
```

Out [21]:

Clase	Porcentaje
Front-Wheel Drive	0.360280
Rear-Wheel Drive	0.352863
4-Wheel or All-Wheel Drive	0.177345
All-Wheel Drive	0.062325
4-Wheel Drive	0.030886
2-Wheel Drive	0.011402
Part-time 4-Wheel Drive	0.004899

Name: traccion, dtype: float64

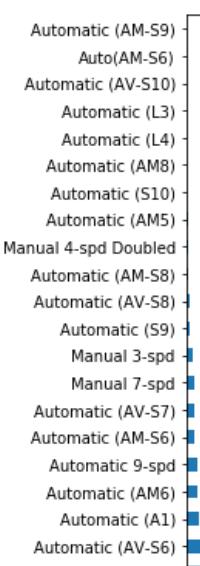
Vemos que esto tiene sentido, ya que la inmensa mayoría de vehículos tienen tracción a dos ruedas, sean delanteras o traseras.

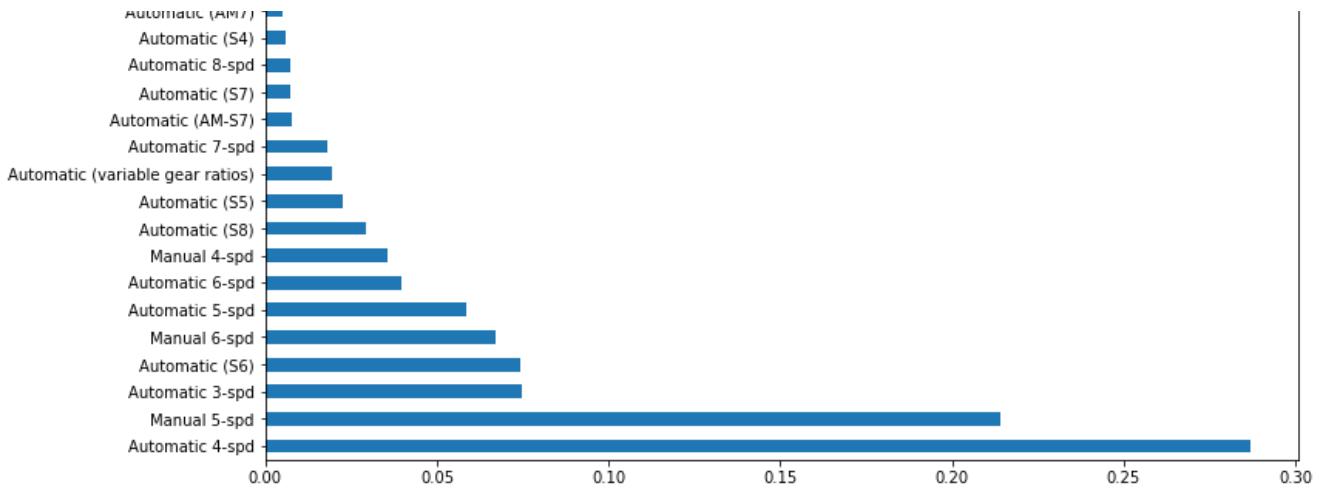
En este caso alguien con conocimiento de dominio podría darse cuenta de que probablemente 2-Wheel Drive sea simplemente una clasificación distinta de Front o Rear.

<https://www.edmunds.com/car-technology/what-wheel-drive.html>

In [22]:

```
vehiculos.transmision.value_counts(normalize=True).plot.barch();
```

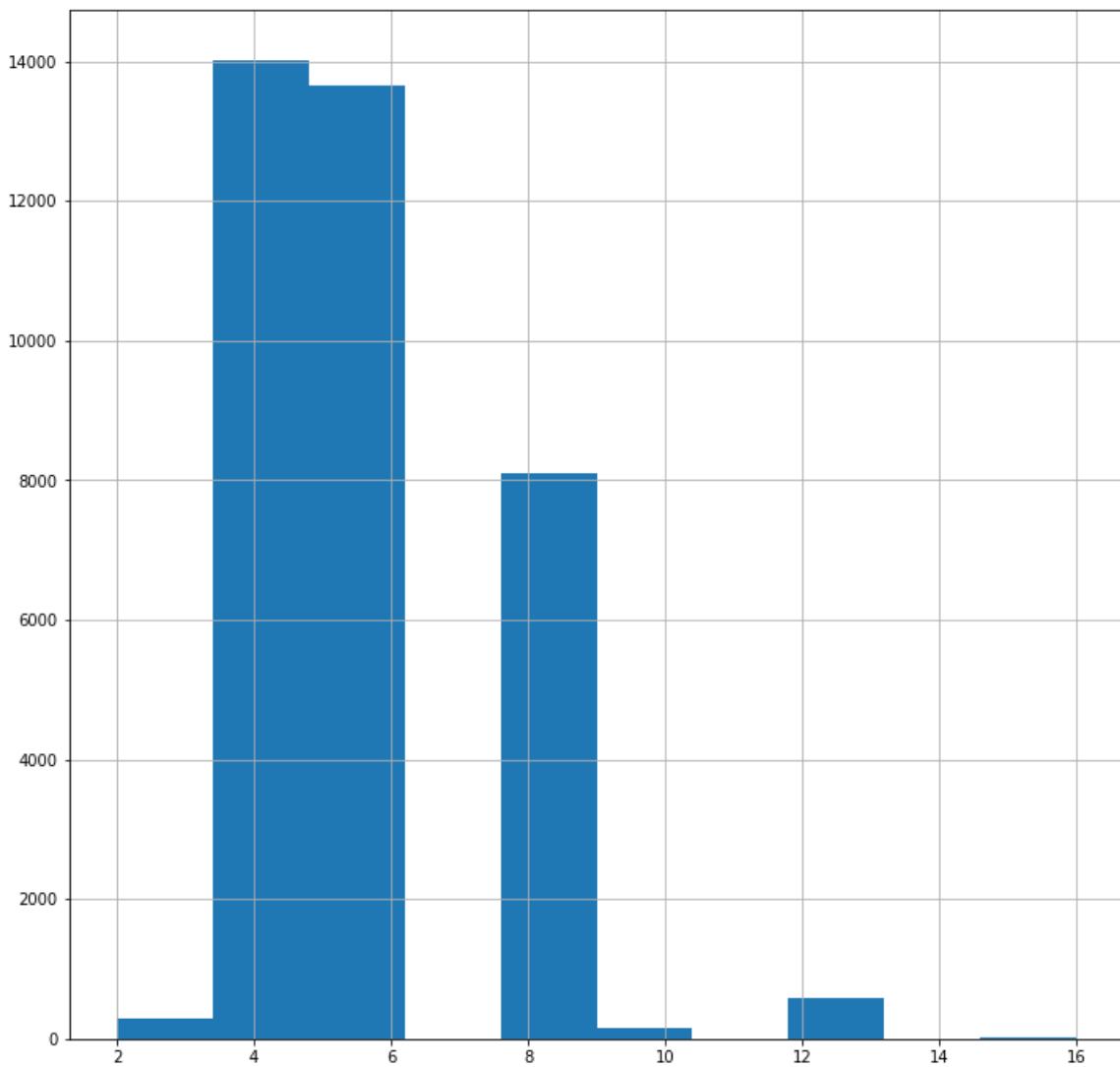




Vemos que aunque transmisión tenga una cardinalidad relativamente alta, en realidad muestra una distribución de tipos, con dos clases mayoritarias

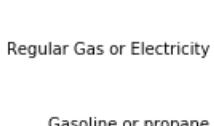
In [23]:

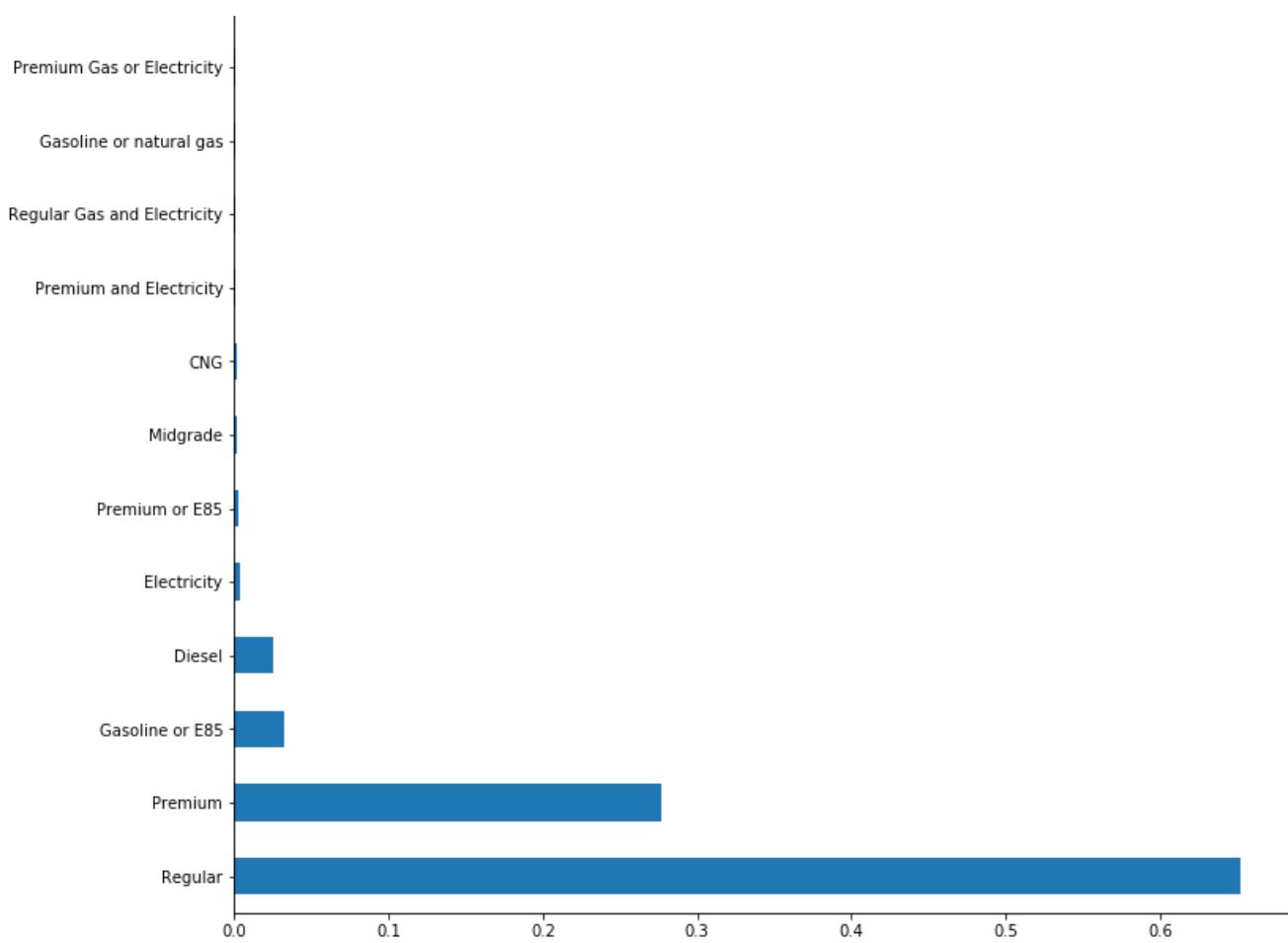
```
vehiculos.cilindros.hist();
```



In [24]:

```
vehiculos.combustible.value_counts(normalize=True).plot.barh();
```





Vemos que la columna `combustible` si puede tener un problema al tener el 65% de los casos gasolina `Regular`

Valores inexistentes

In [25]:

```
n_records = len(vehiculos)
def valores_inexistentes_col(df):
    for columna in df:
        print("{} | {} | {}".format(
            df[columna].name, len(df[df[columna].isnull()]) / (1.0*n_records), df[columna].dtype
        ))
valores_inexistentes_col(vehiculos)

fabricante | 0.0 | object
modelo | 0.0 | object
year | 0.0 | int64
desplazamiento | 0.0037909558624424585 | float64
cilindros | 0.003845112374763065 | float64
transmision | 0.00029786081776333605 | object
traccion | 0.02158137015976171 | object
clase | 0.0 | object
combustible | 0.0 | object
consumo | 0.0 | int64
co2 | 0.0 | float64
```

Vemos que campo `traccion`, `cilindros` y `transmision` tienen valores inexistentes. Sin embargo son cantidades despreciables (maximo es la variable `traccion` con un 3% inexistente)

Valores extremos (Outliers)

Para detectar valores extremos una medida general es considerar outliers aquellos valores con una puntuación Z (z score) mayor de 3, esto es, que se alejan 3 veces o mas desviaciones standard de la media.

El z score se define como:

$$z(x) = \frac{x - \mu}{\sigma}$$

In [26]:

```
from scipy import stats
import numpy as np

def outliers_col(df):
    for columna in df:
        if df[columna].dtype != np.object:
            n_outliers = len(df[np.abs(stats.zscore(df[columna])) > 3])
            print("{} | {} | {}".format(
                df[columna].name,
                n_outliers,
                df[columna].dtype
            ))
    )

outliers_col(vehiculos)

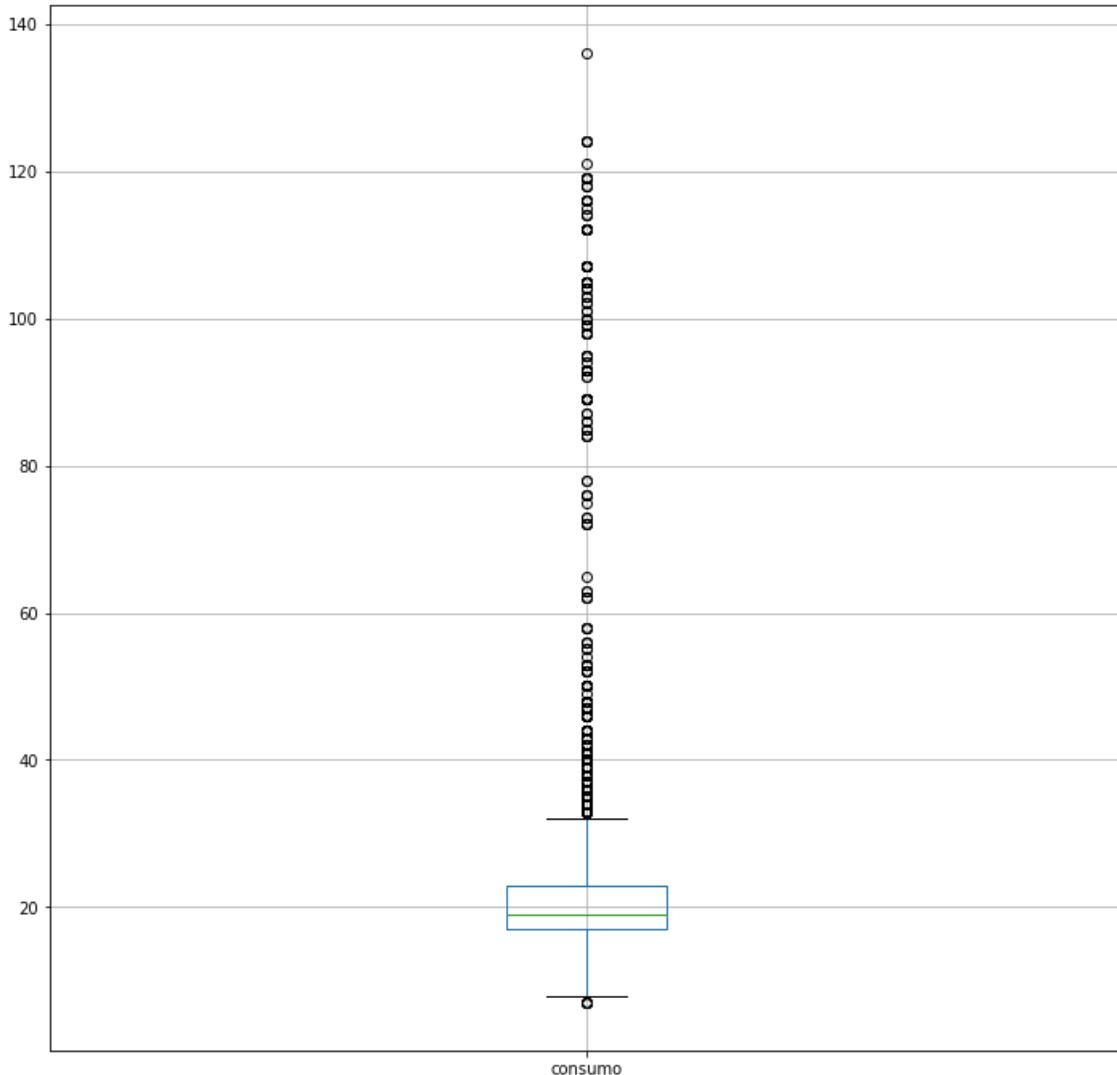
year | 0 | int64
desplazamiento | 0 | float64
cilindros | 0 | float64
consumo | 233 | int64
co2 | 358 | float64

/opt/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: RuntimeWarning: invalid value
encountered in greater
    import sys
```

Vemos que las variables de consumo y co2 tienen outliers. Podemos hacer un boxplot para visualizar esto mejor

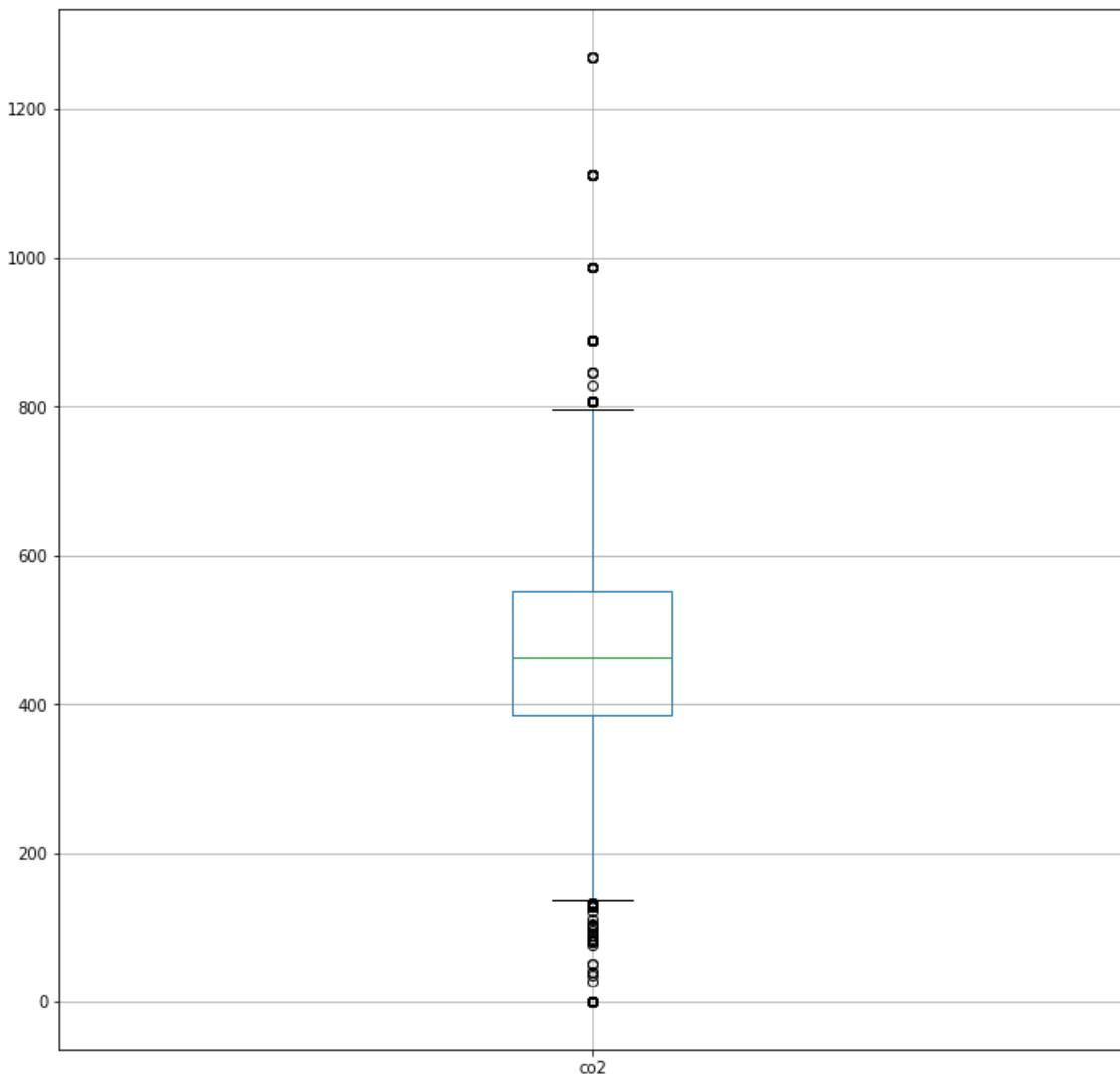
In [27]:

```
vehiculos.boxplot(column='consumo');
```



```
In [28]:
```

```
vehiculos.boxplot(column='co2');
```



Vemos que en cuanto a consumo hay algunos coches que contaminan mas de lo normal y otros que contaminan menos de lo normal, tambien vemos que hay coches que no contaminan nada!

Es posible que haya coches que no usen gasolina en el dataset?

```
In [29]:
```

```
vehiculos[vehiculos.co2==0].combustible.unique()
```

```
Out[29]:
```

```
array(['Electricity'], dtype=object)
```

```
In [30]:
```

```
vehiculos.combustible.unique()
```

```
Out[30]:
```

```
array(['Regular', 'Premium', 'Diesel', 'Premium and Electricity',
       'Premium or E85', 'Electricity', 'Premium Gas or Electricity',
       'Gasoline or E85', 'Gasoline or natural gas', 'CNG',
       'Regular Gas or Electricity', 'Midgrade',
       'Regular Gas and Electricity', 'Gasoline or propane'], dtype=object)
```

Vemos que en este dataset hay vehiculos hibridos y vehiculos electricos puros. Dado que el objetivo es la contaminacion, convendria remover al menos aquellos vehiculos que no contaminan

```
In [31]:
```

```
vehiculos_no_electricos = vehiculos[vehiculos.co2>0]
```

Ya que hemos descubierto esto, volvamos a revisar los datos con el nuevo dataset

In [32]:

```
outliers_col(vehiculos_no_electricos)

year | 0 | int64
desplazamiento | 0 | float64
cilindros | 0 | float64
consumo | 400 | int64
co2 | 221 | float64

/opt/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: RuntimeWarning: invalid value
encountered in greater
    import sys
```

Vemos que siguen habiendo outliers en cuanto a consumo y co2. ¿Será por que los vehículos híbridos consumen menos? Si fuera por esto estos valores extremos son valores perfectamente válidos y por tanto no deberíamos eliminarlos

In [33]:

```
vehiculos_no_electricos[
    np.abs(stats.zscore(vehiculos_no_electricos["consumo"])) > 3
].combustible.value_counts(normalize=True)
```

Out [33]:

```
Regular           0.7175
Diesel            0.1525
Premium          0.0550
Regular Gas and Electricity 0.0475
Premium Gas or Electricity   0.0200
Regular Gas or Electricity   0.0050
Premium and Electricity      0.0025
Name: combustible, dtype: float64
```

Vemos que no es el caso, que hay outliers en un rango de combustibles tanto híbridos como no híbridos. Así que son outliers reales.

In [34]:

```
valores_duplicados_col(vehiculos_no_electricos)

fabricante | 0.099-0.0 | object
modelo | 0.005-0.0 | object
year | 0.038-0.007 | int64
desplazamiento | 0.095-0.0 | float64
cilindros | 0.38-0.0 | float64
transmision | 0.287-0.0 | object
traccion | 0.351-0.005 | object
clase | 0.145-0.0 | object
combustible | 0.652-0.0 | object
consumo | 0.097-0.0 | int64
co2 | 0.084-0.0 | float64
```

Valores inexistentes

In [35]:

```
valores_inexistentes_col(vehiculos)

fabricante | 0.0 | object
modelo | 0.0 | object
year | 0.0 | int64
desplazamiento | 0.0037909558624424585 | float64
cilindros | 0.003845112374763065 | float64
transmision | 0.00029786081776333605 | object
traccion | 0.02158137015976171 | object
clase | 0.0 | object
combustible | 0.0 | object
consumo | 0.0 | int64
co2 | 0.0 | float64
```

Ahora vemos que no hay valores inexistentes extremos en ninguna variable. La variable con mayor numero de valores inexistentes es traccion (2%). Obviamente, los coches electricos no tienen motor de gasolina y por tanto, no tienen cilindros.

Conclusion

- Hay 1506 records duplicados (los hemos removido)
- las variables desplazamiento, cilindros, transmision y traccion tienen valores inexistentes
- La variable combustible tiene una clase dominante (65% de coches tienen combustible Regular)
- hay un outlier en las variables co2 y consumo
- Hay coches hibridos y coches electricos (hemos removido estos ultimos ya que no contaminan).

Exportamos el trabajo

In [36]:

```
vehiculos_no_electricos.to_csv("data/vehiculos.2.limpio_analisis.csv", index=False)
```

Agrupacion de variables

In [37]:

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12,12) # 10 pulgadas de ancho y 10 de alto para todos los plots
```

Ingesta de Datos

In [38]:

```
vehiculos = pd.read_csv("data/vehiculos.2.limpio_analisis.csv")
```

In [39]:

```
vehiculos.head()
```

Out[39]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co:
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
2	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500
3	AM General	Post Office DJ8 2WD	1985	4.2	6.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
4	ASC Incorporated	GNX	1987	3.8	6.0	Automatic 4-spd	Rear-Wheel Drive	Midsize Cars	Premium	16	555.437500

En este paso vamos a agregar aquellas variables que se puedan agrupar en tipos más genéricos. Esto ayuda a analizarlos por grupos en vez de por elementos individuales.

Para ello podemos ver el numero de valores distintos que cada variable categorica tiene

In [40]:

```

def unique_col_values(df):
    for column in df:
        print("{} | {} | {}".format(
            df[column].name, len(df[column].unique()), df[column].dtype
        ))
    print()

unique_col_values(vehiculos)

fabricante | 129 | object
modelo | 3734 | object
year | 35 | int64
desplazamiento | 66 | float64
cilindros | 10 | float64
transmision | 38 | object
traccion | 8 | object
clase | 34 | object
combustible | 13 | object
consumo | 49 | int64
co2 | 596 | float64

```

Agrupacion de variables categoricas

Clase de vehiculo

In [41]:

```
vehiculos.clase.unique()
```

Out[41]:

```
array(['Special Purpose Vehicle 2WD', 'Midsize Cars', 'Subcompact Cars',
       'Compact Cars', 'Sport Utility Vehicle - 4WD',
       'Small Sport Utility Vehicle 2WD',
       'Small Sport Utility Vehicle 4WD', 'Two Seaters',
       'Sport Utility Vehicle - 2WD', 'Special Purpose Vehicles',
       'Special Purpose Vehicle 4WD', 'Small Station Wagons',
       'Minicompact Cars', 'Midsize-Large Station Wagons',
       'Midsize Station Wagons', 'Large Cars',
       'Standard Sport Utility Vehicle 4WD',
       'Standard Sport Utility Vehicle 2WD', 'Minivan - 4WD',
       'Minivan - 2WD', 'Vans', 'Vans, Cargo Type',
       'Vans, Passenger Type', 'Standard Pickup Trucks 2WD',
       'Standard Pickup Trucks', 'Standard Pickup Trucks/2wd',
       'Small Pickup Trucks 2WD', 'Standard Pickup Trucks 4WD',
       'Small Pickup Trucks 4WD', 'Small Pickup Trucks', 'Vans Passenger',
       'Special Purpose Vehicle', 'Special Purpose Vehicles/2wd',
       'Special Purpose Vehicles/4wd'], dtype=object)
```

In [42]:

```
pequeno = ['Compact Cars', 'Subcompact Cars', 'Two Seaters', 'Minicompact Cars']
medio = ['Midsize Cars']
grande = ['Large Cars']

vehiculos.loc[vehiculos['clase'].isin(pequeno),
              'clase_tipo'] = 'Coches pequeños'

vehiculos.loc[vehiculos['clase'].isin(medio),
              'clase_tipo'] = 'Coches Medianos'

vehiculos.loc[vehiculos['clase'].isin(grande),
              'clase_tipo'] = 'Coches Grandes'

vehiculos.loc[vehiculos['clase'].str.contains('Truck'),
              'clase_tipo'] = 'Camionetas'

vehiculos.loc[vehiculos['clase'].str.contains('Special Purpose'),
              'clase_tipo'] = 'Vehículos Especiales'

vehiculos.loc[vehiculos['clase'].str.contains('Sport Utility'),
              'clase_tipo'] = 'Deportivos'

vehiculos.loc[vehiculos['clase'].str.contains('Station'),
              'clase_tipo'] = 'Coche Familiar'

vehiculos.loc[(vehiculos['clase'].str.lower() == 'van')]
```

```
vehiculos['clase'].unique() # solo contiene 'var' //,  
'clase_tipo'] = 'Furgoneta'
```

In [43]:

```
vehiculos.clase_tipo = vehiculos.clase_tipo.astype("category")
```

In [44]:

```
vehiculos.clase_tipo.dtype
```

Out[44]:

```
CategoricalDtype(categories=['Camionetas', 'Coche Familiar', 'Coches Grandes',  
'Coches Medianos', 'Coches pequeños', 'Deportivos',  
'Furgoneta', 'Vehículos Especiales'],  
ordered=False)
```

In [45]:

```
vehiculos.clase_tipo.value_counts()
```

Out[45]:

Coches pequeños	13007
Camionetas	5439
Deportivos	5289
Coches Medianos	4261
Coche Familiar	2533
Vehículos Especiales	2214
Furgoneta	2211
Coches Grandes	1837
Name: clase_tipo, dtype:	int64

Traccion

In [46]:

```
vehiculos.traccion.unique()
```

Out[46]:

```
array(['2-Wheel Drive', 'Rear-Wheel Drive', 'Front-Wheel Drive',  
'4-Wheel or All-Wheel Drive', 'All-Wheel Drive', nan,  
'4-Wheel Drive', 'Part-time 4-Wheel Drive'], dtype=object)
```

In [47]:

```
vehiculos["traccion_tipo"] = "dos"  
vehiculos["traccion_tipo"][vehiculos.traccion.isin([  
    "4-Wheel or All-Wheel Drive", "All-Wheel Drive",  
    "4-Wheel Drive", "Part-time 4-Wheel Drive"  
])] = "cuatro"
```

```
/opt/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-  
docs/stable/indexing.html#indexing-view-versus-copy  
"""
```

Este warning es un warning de que estamos alterando una copia de vehicles (vehicles_year), no vehicles.

para deshabilitar se puede utilizar:

In [48]:

```
pd.set_option('chained_assignment', None)
```

o simplemente usar loc para asignar valores en pandas

In [49]:

```
vehiculos["traccion_tipo"] = "dos"  
  
vehiculos.loc[vehiculos.traccion.isin([  
    "4-Wheel or All-Wheel Drive", "All-Wheel Drive",  
    "4-Wheel Drive", "Part-time 4-Wheel Drive"])] = "cuatro"
```

```
"4-Wheel Drive", "Part-time 4-Wheel Drive"
]), "traccion_tipo"] = "cuatro"
```

Pandas tiene un dtype especial para variables categoricas llamado `category`. El especificar a pandas que un campo es una categoria en vez de un object (un string generalmente) amplia las funcionalidades que podemos usar. Además, las categorias en general ocupan menos espacio en memoria (si su cardinalidad no es muy elevada).

In [50]:

```
vehiculos.traccion_tipo = vehiculos.traccion_tipo.astype("category")
```

Transmision

In [51]:

```
vehiculos.transmision.unique()
```

Out[51]:

```
array(['Automatic 3-spd', 'Automatic 4-spd', 'Manual 5-spd',
       'Automatic (S5)', 'Manual 6-spd', 'Automatic 5-spd',
       'Automatic (AM8)', 'Automatic (AM-S8)', 'Automatic (AV-S7)',
       'Automatic (S6)', 'Automatic (S9)', 'Automatic (AM-S7)',
       'Automatic (S4)', 'Automatic (AM-S9)', 'Automatic (S7)',
       'Automatic (AM7)', 'Automatic (AM6)', 'Automatic 6-spd',
       'Automatic 8-spd', 'Manual 4-spd', 'Automatic (S8)',
       'Manual 7-spd', 'Automatic (AM-S6)', 'Auto(AM-S6)',
       'Automatic (variable gear ratios)', 'Automatic (AV-S8)',
       'Automatic (A1)', 'Automatic (AV-S6)', 'Manual 3-spd',
       'Automatic (S10)', 'Automatic 9-spd', 'Manual 4-spd Doubled', nan,
       'Automatic (L4)', 'Automatic (L3)', 'Automatic (AV-S10)',
       'Automatic 7-spd', 'Automatic (AM5)'], dtype=object)
```

Vemos que las transmisiones se pueden agregar en manual o automatica

In [52]:

```
vehiculos['transmision_tipo'] = "Automatica"
vehiculos['transmision_tipo'][
    (vehiculos['transmision'].notnull()) & (vehiculos['transmision'].str.startswith('M'))]
] = "Manual"
```

In [53]:

```
vehiculos.transmision_tipo = vehiculos.transmision_tipo.astype("category")
```

In [54]:

```
vehiculos.transmision_tipo.value_counts()
```

Out[54]:

```
Automatica    24937
Manual        11854
Name: transmision_tipo, dtype: int64
```

Combustible

In [55]:

```
vehiculos.combustible.value_counts()
```

Out[55]:

Regular	24078
Premium	10206
Gasoline or E85	1215
Diesel	933
Premium or E85	124
Midgrade	77
CNG	60
Premium and Electricity	30
Regular Gas and Electricity	20
Gasoline or natural gas	20
Premium Gas or Electricity	18

```
Gasoline or propane          8  
Regular Gas or Electricity  2  
Name: combustible, dtype: int64
```

In [56]:

```
vehiculos['combustible_tipo'] = 'Otros tipos de combustible'  
vehiculos.loc[vehiculos['combustible']=='Regular',  
             'combustible_tipo'] = 'Normal'  
vehiculos.loc[vehiculos['combustible']=='Premium',  
             'combustible_tipo'] = 'Premium'  
  
vehiculos.loc[vehiculos['combustible'].str.contains('Electricity'),  
              'combustible_tipo'] = 'Hibrido'
```

In [57]:

```
vehiculos.combustible_tipo = vehiculos.combustible_tipo.astype("category")
```

In [58]:

```
vehiculos.combustible_tipo.value_counts()
```

Out[58]:

```
Normal                24078  
Premium               10206  
Otros tipos de combustible 2437  
Hibrido                70  
Name: combustible_tipo, dtype: int64
```

In [59]:

```
vehiculos.head()
```

Out[59]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co:
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
2	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500
3	AM General	Post Office DJ8 2WD	1985	4.2	6.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
4	ASC Incorporated	GNX	1987	3.8	6.0	Automatic 4-spd	Rear-Wheel Drive	Midsize Cars	Premium	16	555.437500

Agrupar variables continuas.

Una manera sencilla de convertir variables continuas en otras categoricas es mediante el uso de quintiles.

In [60]:

```
tipos_tamaño_motor = ['muy pequeño', "pequeño", "mediano", "grande", "muy grande"]  
  
vehiculos['tamano_motor_tipo'] = pd.qcut(vehiculos['desplazamiento'],  
                                         5, tipos_tamaño_motor)
```

```
In [61]:
```

```
tipos_consumo = ['muy bajo', 'bajo', 'moderado', 'alto', 'muy alto']

vehiculos['consumo_tipo'] = pd.qcut(vehiculos['consumo'],
                                     5, tipos_consumo)
```

```
In [62]:
```

```
tipos_co2 = ['muy bajo', 'bajo', 'moderado', 'alto', 'muy alto']

vehiculos['co2_tipo'] = pd.qcut(vehiculos['co2'],
                                 5, tipos_co2)
```

```
In [63]:
```

```
vehiculos.head()
```

```
Out[63]:
```

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co2
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
2	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500
3	AM General	Post Office DJ8 2WD	1985	4.2	6.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
4	ASC Incorporated	GNX	1987	3.8	6.0	Automatic 4-spd	Rear-Wheel Drive	Midsize Cars	Premium	16	555.437500

Al ver las primeras filas spuede ver algo raro. Se observa que tipo_consumo y tipo_co2 estan correlacionadas negativamente. Es decir, para cada coche, aquellos con un consumo bajo tienen un co2 alto y viceversa. Sin ser un experto en coches, la lógica me diria lo contrario, es decir, que aquellos coches que mas gasolina consumen son aquellos que mas contaminan.

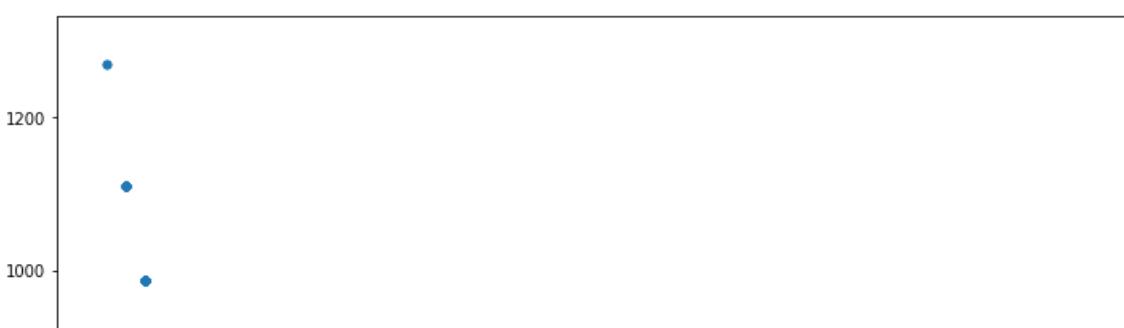
En este momento lo mejor es visualizar la relacion de dichas variables y ver como están relacionadas. Éste es un ejemplo de como el EDA no es un proceso lineal.

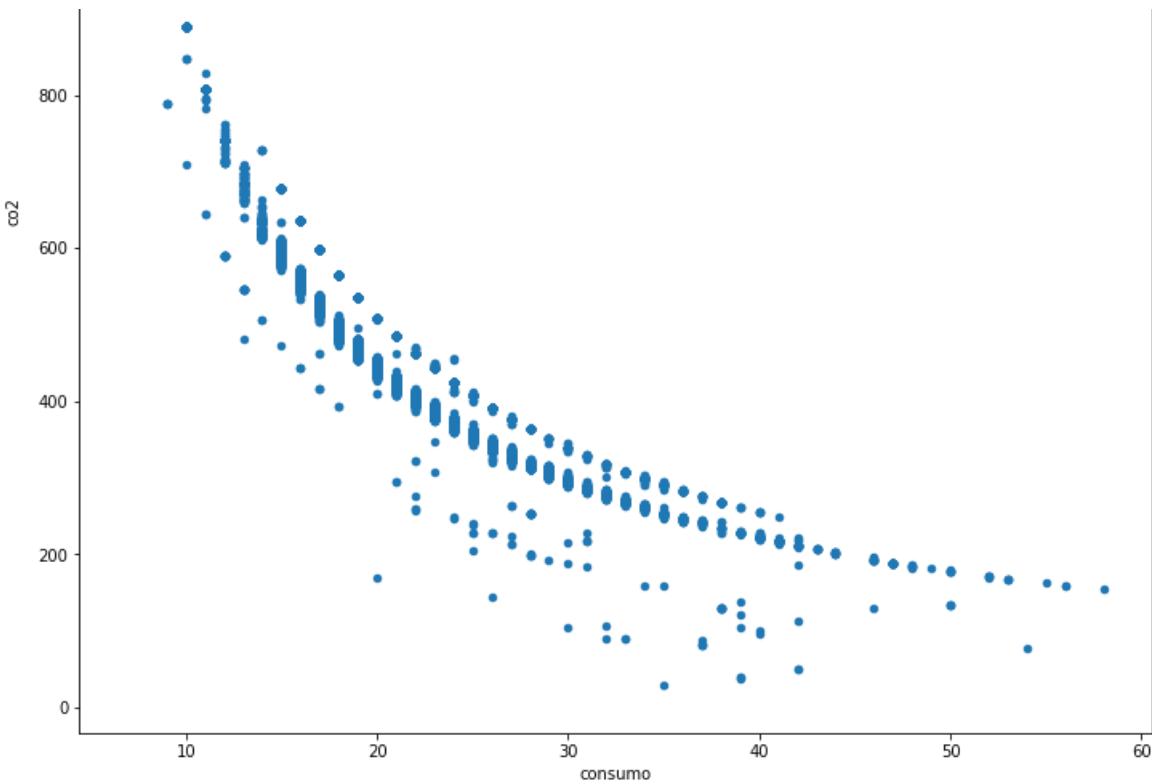
```
In [64]:
```

```
vehiculos.plot.scatter(x="consumo", y="co2")
```

```
Out[64]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1aaaf76e2b0>
```





En la descripción del dataset, se ve que CO2 se mide en gramos por milla, mientras que el consumo se mide en millas por gallon.

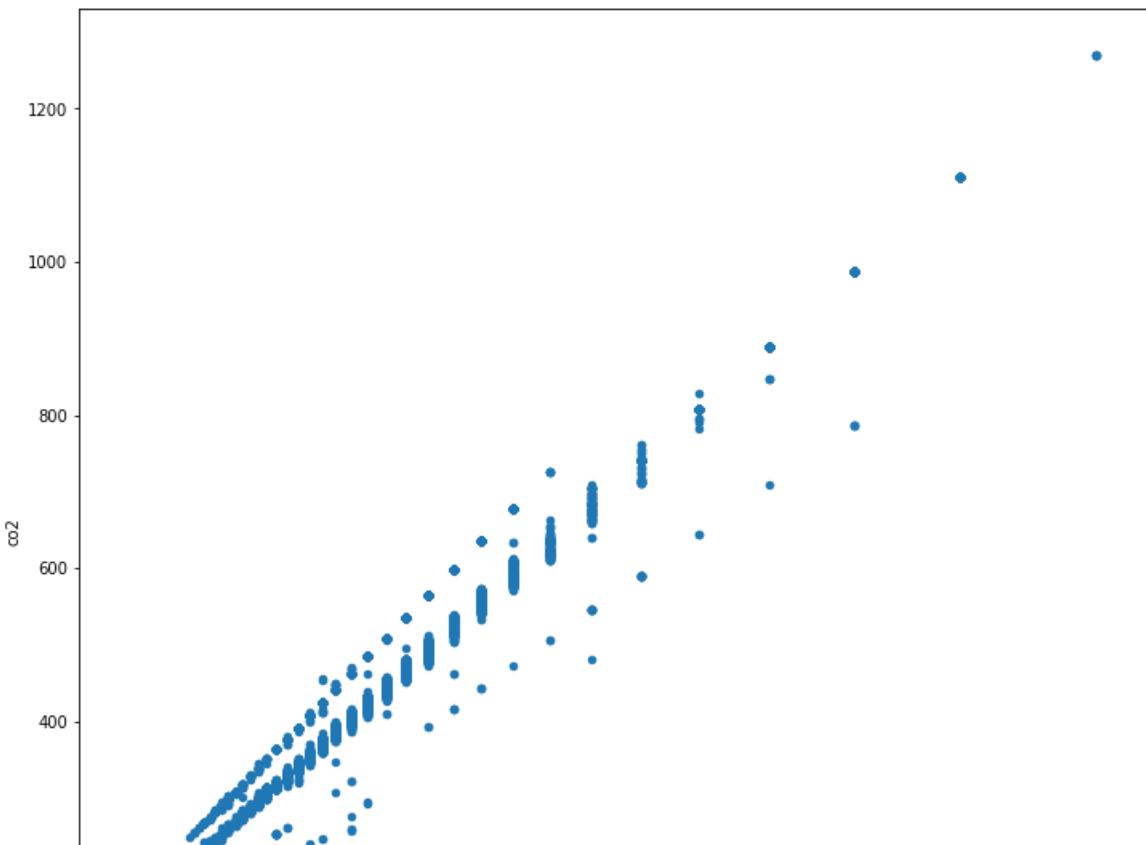
Lo que debemos hacer es convertir el consumo a galones por milla y así ambas variables son relativas a la milla y podemos compararlas.

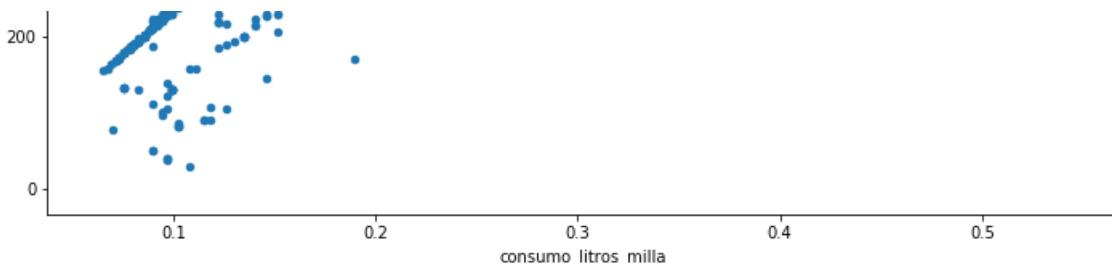
In [65]:

```
litros_por_galon = 3.78541
vehiculos["consumo_litros_milla"] = litros_por_galon / vehiculos.consumo
```

In [66]:

```
vehiculos.plot.scatter(x="consumo_litros_milla", y="co2");
```





Se vuelve a calcular los niveles de consumo con la nueva variable

In [67]:

```
tipos_consumo = ['muy bajo', 'bajo', 'moderado', 'alto', 'muy alto']

vehiculos['consumo_tipo'] = pd.qcut(vehiculos['consumo_litros_milla'],
                                    5, labels=tipos_consumo)

vehiculos.consumo_tipo.head()
```

Out [67]:

```
0      alto
1  muy alto
2      alto
3  muy alto
4      alto
Name: consumo_tipo, dtype: category
Categories (5, object): [muy bajo < bajo < moderado < alto < muy alto]
```

In [68]:

```
vehiculos.head()
```

Out [68]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
2	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500
3	AM General	Post Office DJ8 2WD	1985	4.2	6.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615385
4	ASC Incorporated	GNX	1987	3.8	6.0	Automatic 4-spd	Rear-Wheel Drive	Midsized Cars	Premium	16	555.437500

Una vez que se han agrupado todas las variables que nos interesaban, se guarda el dataframe

In [69]:

```
vehiculos.dtypes
```

Out [69]:

fabricante	object
modelo	object
year	int64

```

year           int64
desplazamiento    float64
cilindros        float64
transmision      object
traccion         object
clase            object
combustible      object
consumo          int64
co2              float64
clase_tipo       category
traccion_tipo    category
transmision_tipo category
combustible_tipo category
tamano_motor_tipo category
consumo_tipo     category
co2_tipo         category
consumo_litros_milla float64
dtype: object

```

Conclusion

La variable `consumo` esta definida en millas por galon y la variable `co2` está definida como gramos por milla. Dado que el `co2` es la variable principal del dataset, hemos creado la variable `consumo_litros_milla` definida como litros por milla para poder comparar con `co2`

Exportar

Generalmente, un formato muy extendido para guardar datos es csv. Esto, que normalmente no es mala idea, no es recomendable entre pasos cuando se está trabajando con python y pandas.

CSV (o Comma Separated Values, es decir, Valores separados por comma), es un formato muy simple, que en general consiste de un elemento por fila, y cada campo separado por una coma.

El principal problema que esto tiene es que al guardar datos en csv se pierden todos los datos que pandas a obtenido sobre el dataframe (por ejemplo, que tipo de variable se guarda en cada columna, o que variables son categóricas).

Por convenio se debe utilizar un formato nativo de python para guardar dataframes entre pasos, y guardar los datos finales a un formato como csv, para que se puedan compartir con otras personas que no usen python.

El formato standard de serialización en Python (y serialización significa básicamente guardar un archivo al disco duro) es `pickle`. Pandas puede leer y escribir a pickle sin problemas, y al leer un archivo pickle es como si nunca hubiésemos cerrado el jupyter notebook, el dataframe no habrá perdido ninguna propiedad.

In [70]:

```
vehiculos.to_pickle("data/vehiculos.3.variables_agrupadas.pkl")
```

Distribución de variables

La magia de matplotlib `%matplotlib notebook`, es muy util para hacer plots mas visibles, pero es un poco complicada de usar ya que requiere el cerrar cada plot para poder continuar.

Alternativamente, se puede usar la el comando `%matplotlib inline`, que es más sencilla ya que simplemente muestra el gráfico original en el jupyter notebook. Para modificar el tamaño de los plots en este caso basta con cambiar el parámetro general de matplotlib `figure.figsize` al tamaño de gráfico deseado (en pulgadas)

In [71]:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

```

In [72]:

```

%matplotlib inline

plt.rcParams['figure.figsize'] = (10,10) # 10 pulgadas de ancho y 10 de alto para todos los plots

```

Ingesta de Datos

```
In [73]:
```

```
vehiculos = pd.read_pickle("data/vehiculos.3.variables_agrupadas.pkl")
```

```
In [74]:
```

```
vehiculos.dtypes
```

```
Out[74]:
```

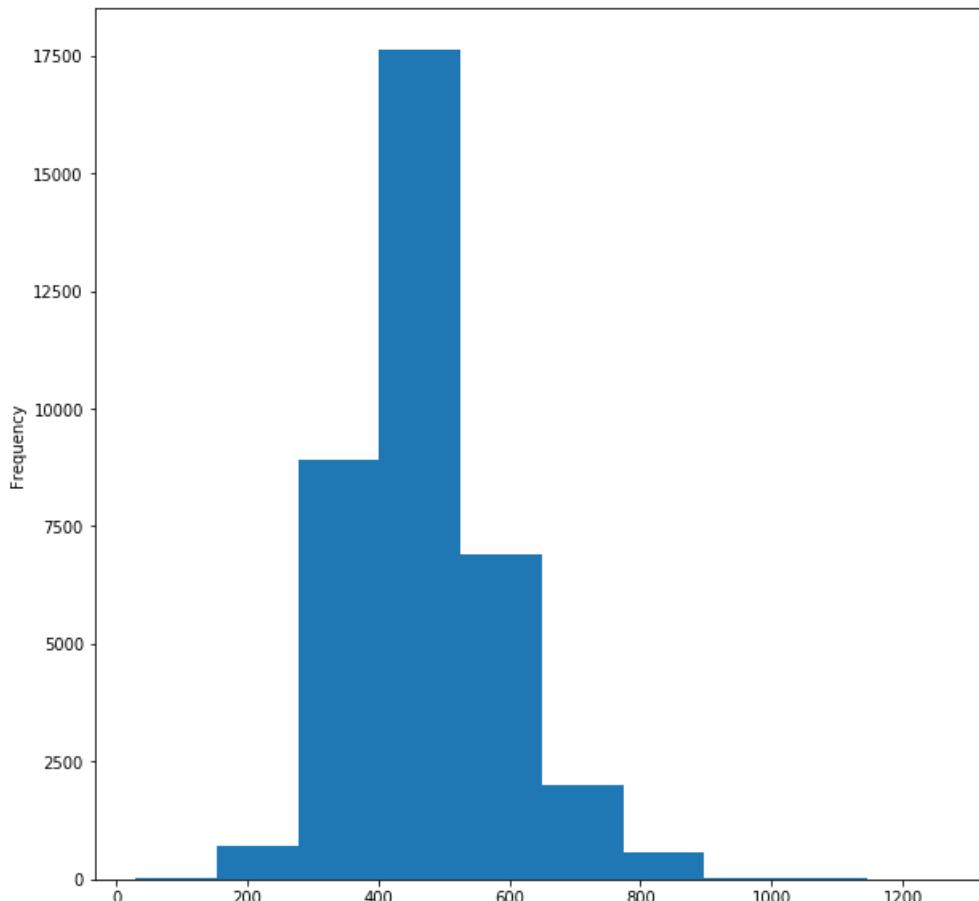
```
fabricante          object  
modelo             object  
year               int64  
desplazamiento    float64  
cilindros          float64  
transmision         object  
traccion           object  
clase              object  
combustible        object  
consumo            int64  
co2                float64  
clase_tipo          category  
traccion_tipo       category  
transmision_tipo   category  
combustible_tipo   category  
tamano_motor_tipo  category  
consumo_tipo        category  
co2_tipo            category  
consumo_litros_milla float64  
dtype: object
```

Distribución de variables numéricas

Usamos histogramas para ver la distribución de una variable

```
In [75]:
```

```
vehiculos['co2'].plot.hist();
```

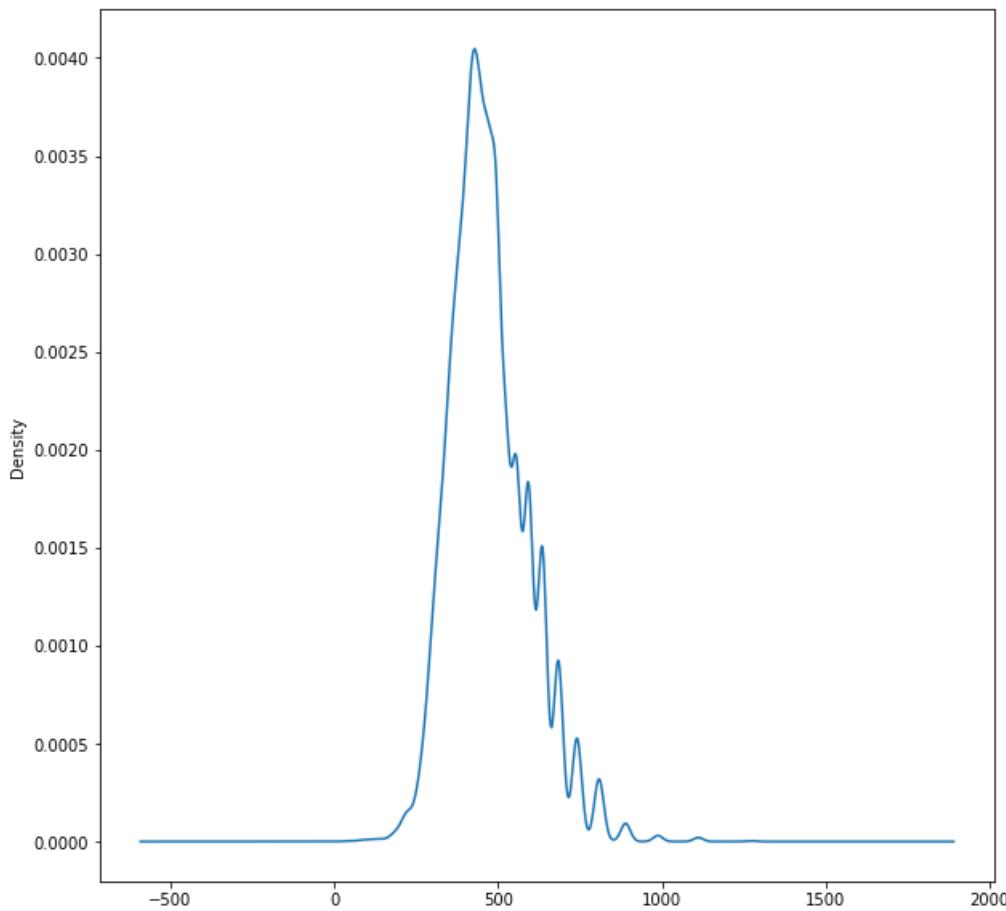


Alternativamente, podemos hacer un gráfico KDE , o Kernel Density Estimate, que produce una función estimada de la distribución

de la variable ([Función de densidad de probabilidad](#)).

In [76]:

```
vehiculos['co2'].plot.kde();
```



In [77]:

```
def distribucion_variable_numerica(df, col):
    df[col].plot.kde()
    plt.xlabel('Distribucion de la variable {}'.format(col))
    plt.show()
```

Ahora podemos usar pywidgets para poder analizar las distribuciones más cómodamente. Tiene sentido hacer estos gráficos solo para variables numéricas. El método pandas.select_dtypes devuelve aquellas columnas de unos tipos específicos

In [78]:

```
columnas_numericas = vehiculos.select_dtypes(['int', 'float']).columns
```

In [79]:

```
from ipywidgets import interact, fixed
```

In [80]:

```
interact(distribucion_variable_numerica, col=columnas_numericas, df=fixed(vehiculos));
```

Vemos que la variable cilindros pese a ser numérica no tiene una distribución equilibrada de valores

In [81]:

```
vehiculos['cilindros'].value_counts(normalize=True)
```

Out[81]:

4.0	0.381184
6.0	0.350767
8.0	0.220344
5.0	0.020115
...	...

```

12.0    0.015630
3.0     0.006252
10.0    0.004132
2.0     0.001359
16.0    0.000217
Name: cilindros, dtype: float64

```

Parece que gran parte de las variables siguen una distribucion normal. Que las variables sigan una distribución normal es importante dado que muchos algoritmos asumen que la distribución de las variables es normal.

La distribucion normal tienen la forma de la linea verde en el siguiente gráfico:



Podemos comprobar esto con un grafico de probabilidad.

La funcion de `scipy.stats.probplot` compara la distribucion de una variable con una distribucion teorica (la normal por defecto), cuanto más se parezca la gráfica a una linea de 45 grados más normal será

In [82]:

```
from scipy import stats
```

In [83]:

```
def normalidad_variable_numerica(col):
    stats.probplot(vehiculos[col], plot=plt)
    plt.xlabel('Diagrama de Probabilidad(normal) de la variable {}'.format(col))
    plt.show()
```

In [84]:

```
interact(normalidad_variable_numerica, col=columnas_numericas);
```

Parece que siguen una distribución normal, no obstante, conviene asegurarse haciendo un test de normalidad. En un test de normalidad, lo que queremos es rechazar la hipótesis nula de que la variable a analizar se ha obtenido de una población que sigue una distribución normal. Para un nivel de confianza de 95%, rechazamos la hipótesis nula si el p-value es inferior a 0.05. Esto es, si se obtiene un valor P (p-value) menor de 0.05, significa que las probabilidades de que la hipótesis nula sean ciertas es tan baja (menos de un 5%) que la rechazamos.

scipy tiene la función `normaltest` que devuelve el p-value

In [85]:

```
for num_col in columnas_numericas:
    _, pval = stats.normaltest(vehiculos[num_col])
    if(pval < 0.05):
        print("Columna {} no sigue una distribución normal".format(num_col))
```

Columna year no sigue una distribución normal
 Columna consumo no sigue una distribución normal
 Columna co2 no sigue una distribución normal
 Columna consumo_litros_milla no sigue una distribución normal

Con lo cual vemos que ninguna de las variables numéricas siguen una distribución normal correcta.

Distribución de variables categóricas

Una manera de ver como se distribuyen las variables categóricas es mediante la función `pandas.value_counts`. Dicha función nos devuelve el numero de records existentes para cada valor de una columna

In [86]:

```
def distribucion_variable_categorica(col):
    vehiculos[col].value_counts(ascending=True, normalize=True).tail(20).plot.barh()
    plt.show()
```

In [87]:

```
columnas_categoricas = vehiculos.select_dtypes(['object', 'category']).columns
```

In [88]:

```
interact(distribucion_variable_categorica, col=columnas_categoricas);
```

Conclusiones

- Ninguna variable numérica sigue una distribución normal
- la variable numérica cilindros tiene una distribución de valores discretos no balanceada (cilindrada de 2 y 4 y 8 suman el 95% de los vehículos). Podría agruparse como variable categórica (2, 4 , 8 y otro)
- El fabricante con la mayor cantidad de modelos es Chevrolet (10% del total)
- 65% de los vehículos usan gasolina normal
- La distribución de tamaños de motor y de consumo y co2 está equilibrada en todo el rango
- 70% de los vehículos usan tracción a las dos ruedas
- dos tercios de los coches tienen transmisión automática
- La clase mayoritaria de vehículos es la de coches pequeños (35% del total)
- Los mayores fabricantes en cuanto a vehículos analizados son los estadounidenses. Esto tiene sentido ya que la EPA es la agencia americana y probablemente es la que tiene más interés en estudiar coches de USA

Comparaciones

Importamos las librerías que utilizaremos y configuraremos el matplotlib inline para una mejor visualización de los gráficos.

In [89]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [90]:

```
%matplotlib inline

plt.rcParams['figure.figsize'] = (10,10) # 10 pulgadas de ancho y 10 de alto para todos los plots
```

Ingesta de datos

In [91]:

```
vehiculos = pd.read_pickle("data/vehiculos.3.variables_agrupadas.pkl")
```

In [92]:

```
vehiculos.dtypes
```

Out[92]:

fabricante	object
modelo	object
year	int64
desplazamiento	float64
cilindros	float64
transmision	object
traccion	object
clase	object
combustible	object
consumo	int64
co2	float64
clase_tipo	category
traccion_tipo	category
transmision_tipo	category
combustible_tipo	category
tamano_motor_tipo	category
consumo_tipo	category
co2_tipo	category
consumo_litros_milla	float64
dtype:	object

Vemos que al leer de un dataframe pickled, pandas no tiene que averiguar de nuevo qué tipo de dato es cada columna, lo carga todo en memoria directamente

In [93]:

```
vehiculos.head()
```

Out[93]:

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clase	combustible	consumo	co:
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	17	522.764706
1	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
2	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	16	555.437500
3	AM General	Post Office DJ8 2WD	1985	4.2	6.0	Automatic 3-spd	Rear-Wheel Drive	Special Purpose Vehicle 2WD	Regular	13	683.615388
4	ASC Incorporated	GNX	1987	3.8	6.0	Automatic 4-spd	Rear-Wheel Drive	Midsize Cars	Premium	16	555.437500

In [94]:

```
from ipywidgets import interact, fixed
```

Recuento por multiples grupos

In [95]:

```
def pivot_recuento(df, rows, columns, calc_field):
    df_pivot = df.pivot_table(values=calc_field,
                               index=rows,
                               columns=columns,
                               aggfunc=np.size
                               ).dropna(axis=0, how='all')
    return df_pivot
```

In [96]:

```
consumo_combustible = pivot_recuento(vehiculos,"combustible_tipo","consumo_tipo", "year")
consumo_combustible
```

Out[96]:

consumo_tipo	muy bajo	bajo	moderado	alto	muy alto
combustible_tipo					
Hibrido	61.0	6.0	1.0	2.0	NaN
Normal	5686.0	4931.0	5736.0	3158.0	4567.0
Otros tipos de combustible	521.0	340.0	508.0	482.0	586.0
Premium	1386.0	2211.0	3836.0	1146.0	1627.0

In [97]:

```
def heatmap_recuento_tipos (df, col1, col2):
    pivot_table = pivot_recuento(df,col1, col2, "year")
    sns.heatmap(pivot_table, annot=True, fmt='g')
```

```
sns.plt.ylabel(col1)
sns.plt.xlabel(col2)
plt.show()
```

In [98]:

```
interact(heatmap_recuento_tipos, col1=vehiculos.columns, col2=vehiculos.columns,
df=fixed(vehiculos));
```

Conclusiones

- Hay mas vehiculos de dos ruedas de bajo consumo que de traccion a las 4 ruedas
- Los fabricantes se concentran mas en vehiculos de motor pequeno/bajo consumo y motor muy grande/alto consumo
- La mayor parte de coches tienen transmision automatica, con los coches pequeños teniendo valores similares de coches automaticos y manuales
- Hay una cantidad de camionetas que son mas contaminantes que cualquier otro tipo de coche

Medias por variables categoricas

In [99]:

```
def medias_por_categoria(col_grupo, col_calculo):
    vehiculos.groupby(col_grupo)[col_calculo].mean().plot.barh()
    plt.ylabel(col_grupo)
    plt.xlabel('Valores medios de {}'.format(col_calculo))
    plt.show()
```

In [100]:

```
columnas_numericas = vehiculos.select_dtypes(['int', 'float']).columns
columnas_categoricas = vehiculos.select_dtypes(['object', 'category']).columns
columnas_tipo = [col for col in vehiculos.columns if col.endswith("_tipo")]
```

In [101]:

```
interact(medias_por_categoria, col_grupo=columnas_categoricas, col_calculo=columnas_numericas);
```

Conclusiones

- Vehiculos con transmision automatica tienen valores de co2 y consumo ligeramente más altos
- Furgonetas y camionetas tienen el consumo más alto (alrededor de 0.25 litros/milla). Por otra parte, los coches familiares y pequeños tienen el menor consumo de gasolina (~0.15 litros/milla)
- Los vehículos híbridos emiten menos de la mitad de CO2 que el resto de vehiculos (que tienen similares emisiones)

Medias por multiples tipos

In [102]:

```
def pivot_media(rows, columns, calc_field):
    df_pivot = vehiculos.pivot_table(values=calc_field,
                                       index=rows,
                                       columns=columns,
                                       aggfunc=np.mean
                                       ).dropna(axis=0, how='all')
    return df_pivot
```

In [103]:

```
pivot_media("combustible_tipo", "clase_tipo", "co2")
```

Out[103]:

clase_tipo	Camionetas	Coche Familiar	Coches Grandes	Coches Medianos	Coches pequeños	Deportivos	Furgoneta	Vehículos Especiales
combustible_tipo								
Hibrido	NaN	NaN	221.875000	127.380952	149.645161	260.222222	106.000000	NaN
Normal	560.216673	399.493604	461.726146	409.673286	384.393158	476.788251	590.807827	543.838453

Otros tipos de combustible	Camionetas	Coche	Coches	Coches	Coches	Deportivos	Furgoneta	Vehiculos Especiales
combustible_tipo	552.958788	39.161024	447.90068	41.16025408	34.99572108	520.672914	602.867693	599.268790
Premium	644.975464	439.247746	520.511823	479.033374	467.943014	515.605886	547.793508	

In [104]:

```
def heatmap_medias_tipos(col1, col2, col3):
    pivot_table = pivot_media(col1, col2, col3)
    sns.heatmap(pivot_table, annot=True, fmt='g')
    sns.plt.ylabel(col1)
    sns.plt.xlabel(col2)
    plt.show()
```

In [105]:

```
interact(heatmap_medias_tipos, col1=vehiculos.columns, col2=vehiculos.columns,
       col3=columnas_numericas);
```

Conclusiones

- Camionetas de gasolina Premium consumen un 38% más que vehiculos pequeños que usan el mismo tipo de gasolina

Comparacion de tendencias (temporales)

En este dataset en particular, tenemos una variable temporal `year`. A la cual echamos un vistazo en un apartado anterior pero que conviene observar de nuevo. Podemos considerarla una variable ordinal (ya que no es una variable continua).

Para la cual tiene sentido ver la evolución. Para ello graficos de linea son lo ideal.

In [106]:

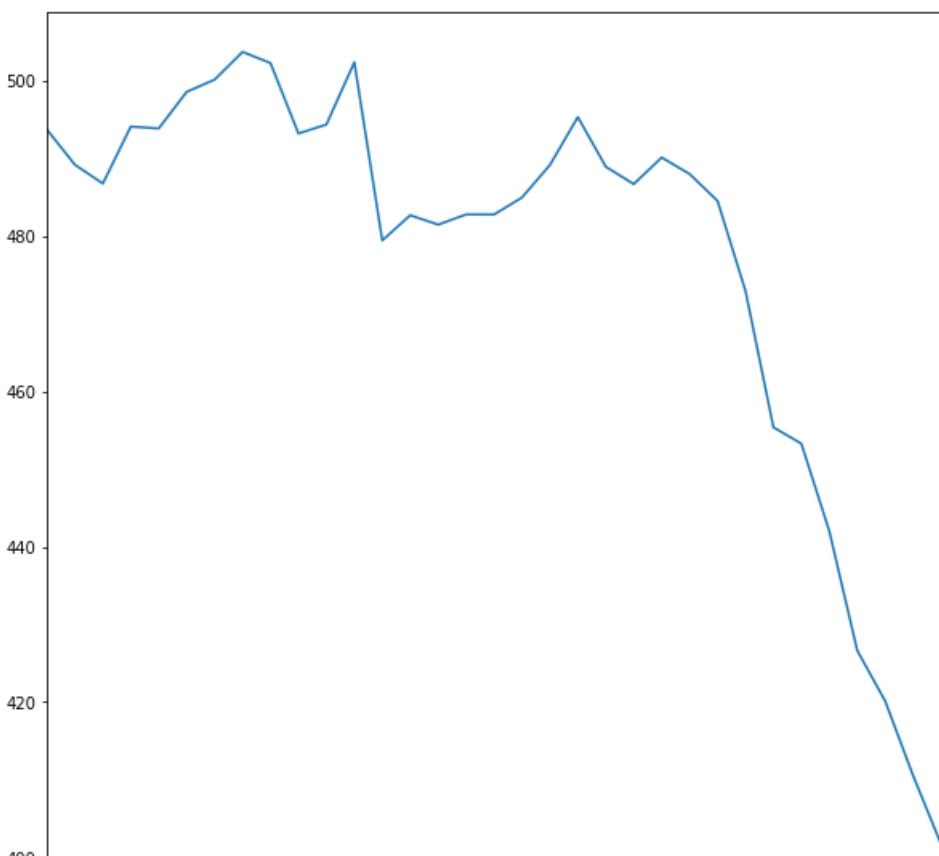
```
vehiculos_pre_2017 = vehiculos.query("year<2017")
```

In [107]:

```
vehiculos_pre_2017.groupby('year')[['co2']].mean().plot()
```

Out [107]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1aae4ab518>
```





Vemos que en general la emisión de co2 se ha reducido bastante con el tiempo

In [108]:

```
def evolución_medias(col_calculo):
    vehiculos_pre_2017.groupby('year')[col_calculo].mean().plot()
    plt.show()
```

In [109]:

```
interact(evolución_medias, col_calculo=columnas_numericas);
```

Vemos que históricamente se ha ido aumentando la cilindrada (y desplazamiento). en los vehículos fabricados, pero a partir de 2010 esta tendencia se invierte. Vemos que ambas variables están linealmente relacionadas.

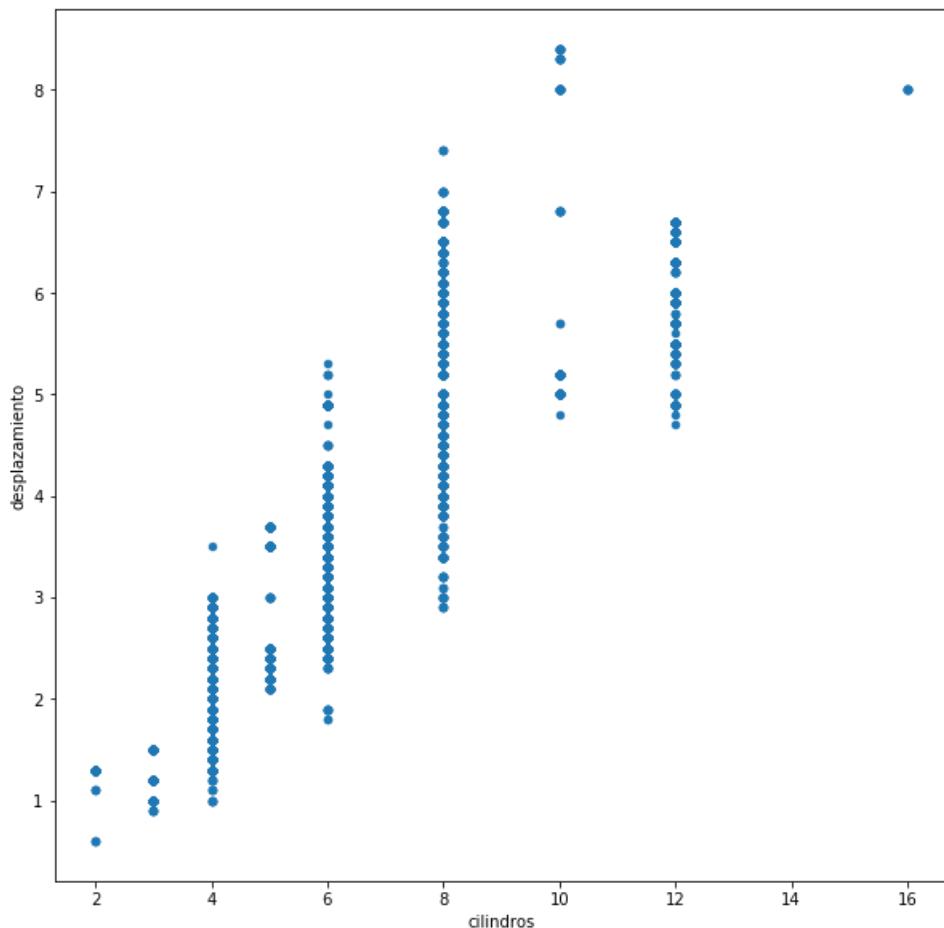
También vemos que el consumo y emisión de CO2 ha ido bajando a lo largo de los años, y dicha tendencia se acentuó a partir de 2006 ([¿Cambio climático?, la emisión de Una Verdad Incomoda?](#))

In [110]:

```
vehiculos.plot.scatter(x="cilindros", y="desplazamiento")
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1aac64f908>
```



In [111]:

```
def evolución_recuento(col_calculo):
    for categoría in vehiculos_pre_2017[col_calculo].unique():
        n_vehiculos_categoría_ano = vehiculos_pre_2017[vehiculos_pre_2017[col_calculo]==categoría].groupby(
            'year').apply(np.size)
        plt.plot(
            n_vehiculos_categoría_ano.index,
            n_vehiculos_categoría_ano,
```

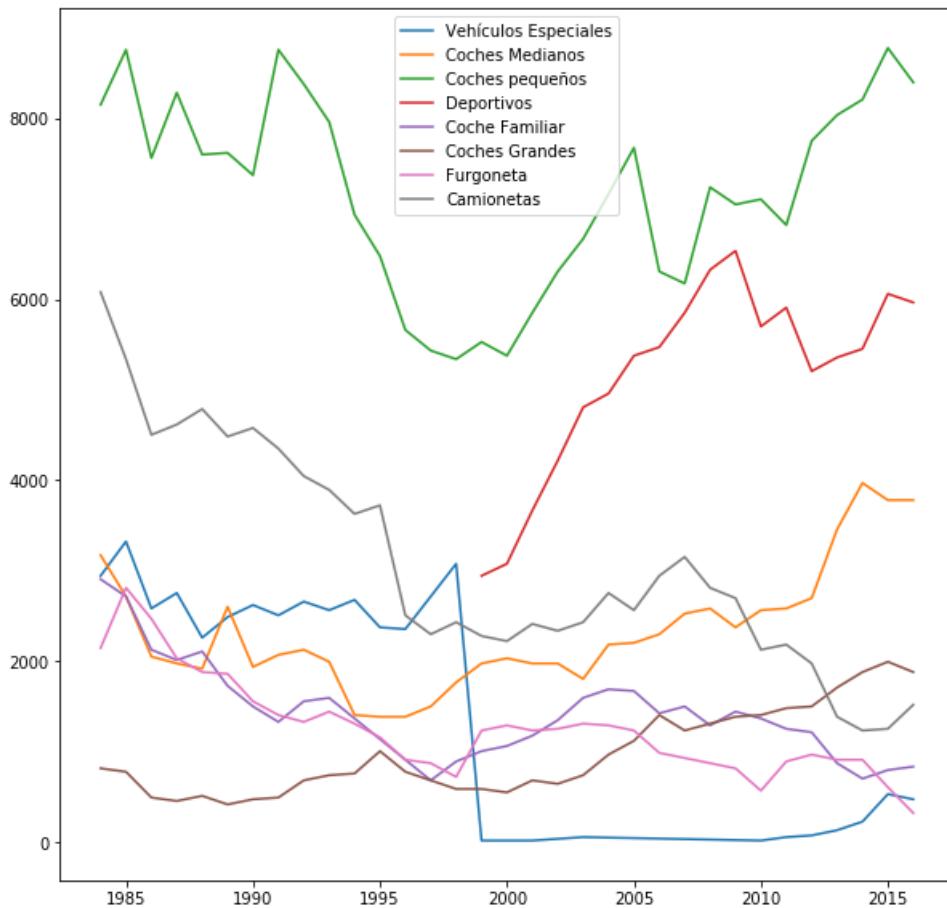
```

        label=categoría
    )
plt.legend()

```

In [112]:

```
evolución_recuento("clase_tipo")
```



Una vez hemos terminado el análisis, el paso final es compilar toda la información obtenida en un documento. Dicho documento tiene dos objetivos principales:

- Informar a aquellas personas interesadas de los descubrimientos encontrados. ¿Esto nos incluye a nosotros mismos en el futuro!
- Facilitar la replicación del análisis por otros Data Scientists.

Conclusiones del análisis

Descripción

El dataset consiste en información relativa a vehículos vendidos en USA desde 1985. Dicha información incluye detalles técnicos (cilindrada, tipo de transmisión) y detalles medioambientales (consumo de gasolina, emisiones de CO₂)

El archivo original está en: <https://www.fueleconomy.gov/feg/epadata/vehicles.csv.zip> El archivo que vamos a usar es una versión modificada (con menos columnas)

Descripción del dataset Original <http://www.fueleconomy.gov/feg/ws/index.shtml#ft7>

Las entidades de las que consta dicho dataset son:

- fabricante
- fabricante-modelo
- fabricante-model-año
- fabricante-año

Las variables que existen en dicho dataset son:

fabricante	categorica
------------	------------

model	categorica
year	ordinal/numérica
desplazamiento	numérica
cilindros	numérica
transmision	categorica
traccion	categorica
clase	categorica
combustible	categorica
consumo	numérica
co2	numérica

QA

- Las variables desplazamiento, cilindros, transmision y traccion tienen valores inexistentes
- hay un outlier en las variables co2 y consumo
- Hay coches hibridos y coches electricos (hemos removido estos ultimos ya que no contaminan).
- La variable consumo esta definida en millas por galon y la variable co2 está definida como gramos por milla. Dado que el co2 es la variable principal del dataset, hemos creado la variable consumo_litros_milla definida como litros por milla para poder comparar con co2

Distribución de variables

- Ninguna variable numérica sigue una distribución normal
- la variable numérica cilindros tiene una distribución de valores discretos no balanceada (cilindrada de 2 y 4 y 8 suman el 95% de los vehiculos). Podria agruparse como variable categórica (2, 4 , 8 y otro)
- El fabricante con la mayor cantidad de modelos es Chevrolet (10% del total)
- 65% de los vehiculos usan gasolina normal
- La distribución de tamaños de motor y de consumo y co2 está equilibrada en todo el rango
- 70% de los vehiculos usan traccion a las dos ruedas
- dos tercios de los coches tienen transmision automática
- La clase mayoritaria de vehiculos es la de coches pequeños (35% del total)
- Existen relaciones lineales entre cilindros/desplazamiento y co2/consumo_litros_milla

Comparaciones

- Hay mas vehiculos de dos ruedas de bajo consumo que de traccion a las 4 ruedas
- Los fabricantes se concentran mas en vehiculos de motor pequeño/bajo consumo y motor muy grande/alto consumo
- La mayor parte de coches tienen transmision automatica, con los coches pequeños teniendo valores similares de coches automaticos y manuales
- Hay una cantidad de camionetas que son mas contaminantes que cualquier otro tipo de coche
- Vehiculos con transmision automatica tienen valores de co2 y consumo ligeramente más altos
- Furgonetas y camionetas tienen el consumo más alto (alrededor de 0.25 litros/milla). Por otra parte, los coches familiares y pequeños tienen el menor consumo de gasolina (~0.15 litros/milla)
- los valores de co2 son similares independientemente del tipo de gasolina empleado
- Camionetas de gasolina Premium consumen un 38% más que vehiculos pequeños que usan el mismo tipo de gasolina
- El consumo y emisión de co2 han ido bajando de forma continuada desde 1985
- Históricamente se ha ido aumentando la cilindrada en los vehiculos fabricados, pero a partir de 2010 esta tendencia se invierte

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-17T16:12:02+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system   : Linux  
release   : 5.0.13-arch1-1-ARCH  
machine   : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Analisis Exploratorio de Datos - Herramientas adicionales

Aquí incluyo unas herramientas que son bastante útiles a la hora de hacer EDA

Ingesta de datos

```
In [2]:
```

```
import pandas as pd  
  
vehiculos = pd.read_csv("../data/vehiculos.1.procesado_inicial.csv")
```

Pandas-profiling

<https://github.com/JosPolflet/pandas-profiling>

```
In [3]:
```

```
!conda install -y pandas-profiling
```

```
Collecting package metadata: done  
Solving environment: done
```

```
## Package Plan ##  
  
environment location: /anaconda3  
  
added / updated specs:  
  - pandas-profiling
```

The following packages will be downloaded:

package	build	
certifi-2019.3.9	py37_0	155 KB
pandas-profiling-1.4.1	py37_0	39 KB
Total:		194 KB

The following NEW packages will be INSTALLED:

```
pandas-profiling    pkgs/main/osx-64::pandas-profiling-1.4.1-py37_0
```

The following packages will be SUPERSEDED by a higher-priority channel:

```
ca-certificates      conda-forge::ca-certificates-2019.3.9~ --> pkgs/main::ca-certificates-  
2019.1.23-0  
certifi                  conda-forge --> pkgs/main  
conda                   conda-forge::conda-4.6.12-py37_2 --> pkgs/main::conda-4.6.12-py37_1  
openssl                 conda-forge::openssl-1.1.1b-h01d97ff_2 --> pkgs/main::openssl-1.1.1b-h1de35cc_
```

```
Downloading and Extracting Packages
certifi-2019.3.9      | 155 KB    | #####|#####|#####|#####|#####|#####|#####| 100%
pandas-profiling-1.4  | 39 KB     | #####|#####|#####|#####|#####|#####|#####| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [4]:

```
import pandas_profiling
pandas_profiling.ProfileReport(vehiculos)
```

Out[4]:

Overview

Dataset info

Number of variables	11
Number of observations	38436
Total Missing (%)	0.3%
Total size in memory	3.2 MiB
Average record size in memory	88.0 B

Variables types

Numeric	4
Categorical	6
Boolean	0
Date	0
Text (Unique)	0
Rejected	1
Unsupported	0

Warnings

- `cilindros` is highly correlated with `desplazamiento` ($\rho = 0.90304$) Rejected
- `fabricante` has a high cardinality: 133 distinct values Warning
- `modelo` has a high cardinality: 3791 distinct values Warning
- `traccion` has 1189 / 3.1% missing values Missing
- Dataset has 1506 duplicate rows Warning

Variables

`cilindros`

Highly correlated

This variable is highly correlated with `desplazamiento` and should be ignored for analysis

Correlation 0.90304

`clase`

Categorical

Distinct count	34
Unique (%)	0.1%
Missing (%)	0.0%

Missing (n)

0



[Toggle details](#)

co2

Numeric

Distinct count 597

Unique (%) 1.6%

Missing (%) 0.0%

Missing (n) 0

Infinite (%) 0.0%

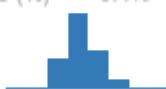
Infinite (n) 0

Mean 472.09

Minimum 0

Maximum 1269.6

Zeros (%) 0.4%



[Toggle details](#)

combustible

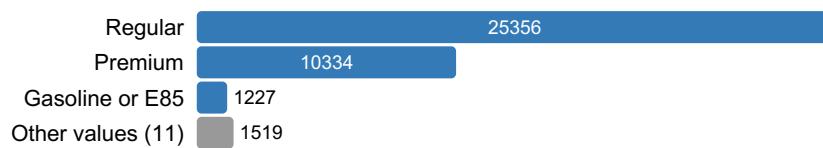
Categorical

Distinct count 14

Unique (%) 0.0%

Missing (%) 0.0%

Missing (n) 0



[Toggle details](#)

consumo

Numeric

Distinct count 84

Unique (%) 0.2%

Missing (%) 0.0%

Missing (n) 0

Infinite (%) 0.0%

Infinite (n) 0

Mean 20.252

Minimum 7

Maximum 136

Zeros (%) 0.0%



[Toggle details](#)

desplazamiento

Numeric

Distinct count	67
Unique (%)	0.2%
Missing (%)	0.4%
Missing (n)	140
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.3143
Minimum	0
Maximum	8.4
Zeros (%)	0.0%



[Toggle details](#)

fabricante

Categorical

Distinct count	133
Unique (%)	0.3%
Missing (%)	0.0%
Missing (n)	0



[Toggle details](#)

modelo

Categorical

Distinct count	3791
Unique (%)	9.9%
Missing (%)	0.0%
Missing (n)	0



[Toggle details](#)

traccion

Categorical

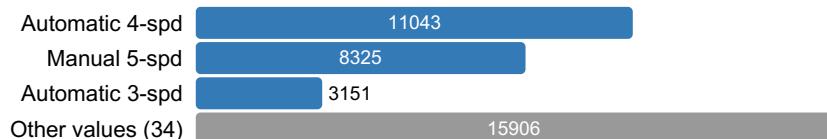
Distinct count	8
Unique (%)	0.0%
Missing (%)	3.1%
Missing (n)	1189



[Toggle details](#)**transmision**

Categorical

Distinct count 38
Unique (%) 0.1%
Missing (%) 0.0%
Missing (n) 11

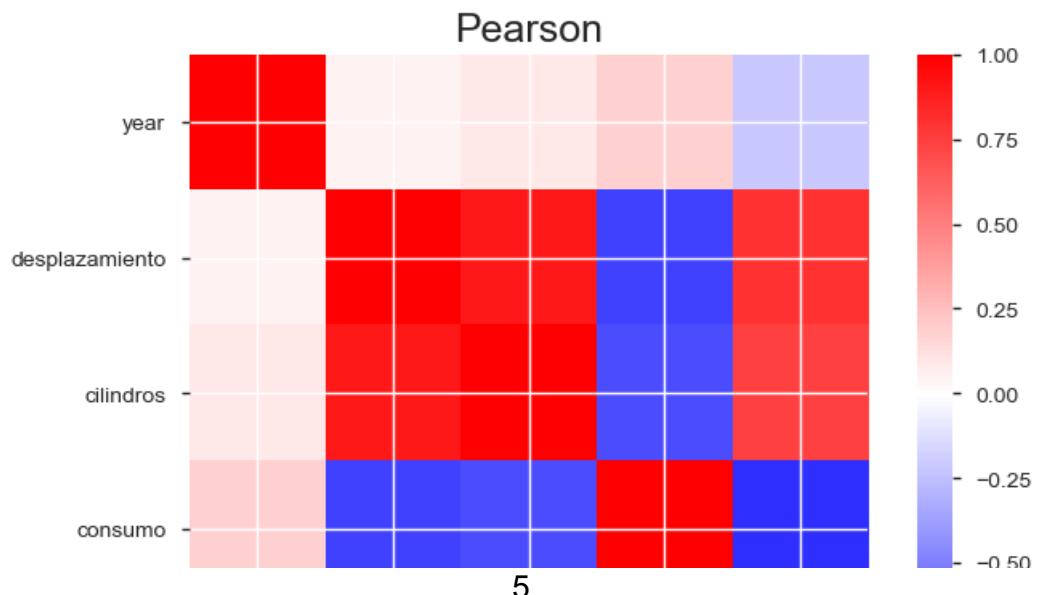
[Toggle details](#)**year**

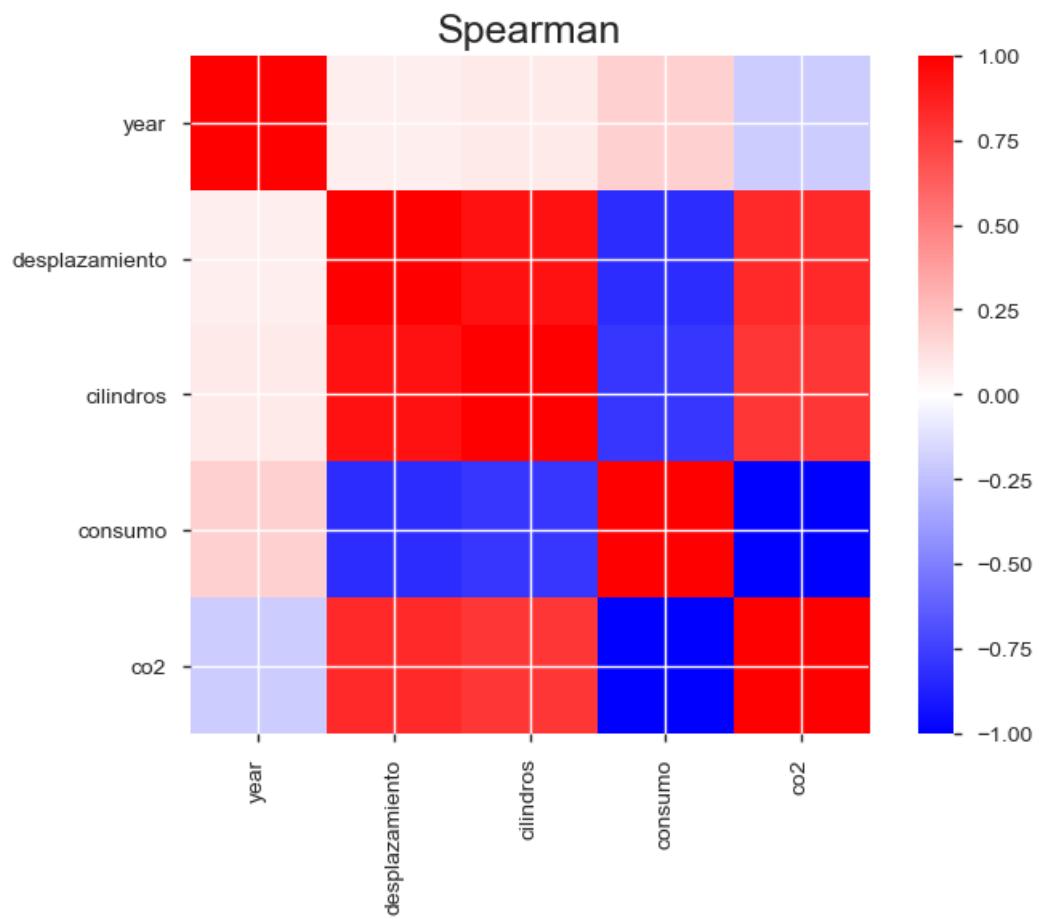
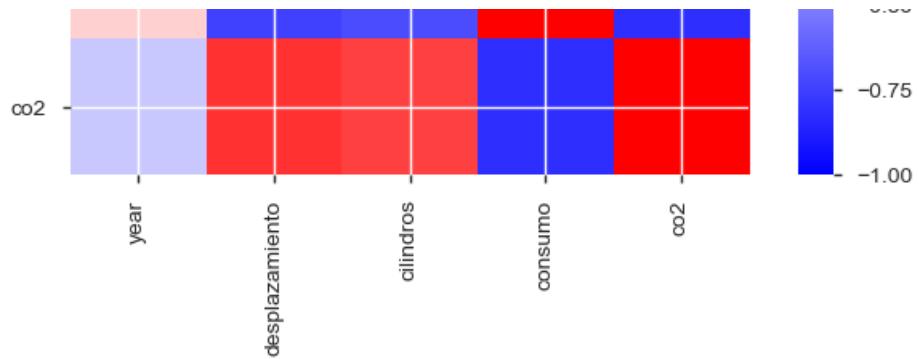
Numeric

Distinct count 35
Unique (%) 0.1%
Missing (%) 0.0%
Missing (n) 0
Infinite (%) 0.0%
Infinite (n) 0
Mean 2000.3
Minimum 1984
Maximum 2018
Zeros (%) 0.0%

[Toggle details](#)

Correlations





Sample

	fabricante	modelo	year	desplazamiento	cilindros	transmision	traccion	clas
0	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Spec
1	AM General	DJ Po Vehicle 2WD	1984	2.5	4.0	Automatic 3-spd	2-Wheel Drive	Spec
2	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Spec
3	AM General	FJ8c Post Office	1984	4.2	6.0	Automatic 3-spd	2-Wheel Drive	Spec
4	AM General	Post Office DJ5 2WD	1985	2.5	4.0	Automatic 3-spd	Rear-Wheel Drive	Spec

[1] ▶

In [5]:

```
%matplotlib inline
```

Missigno

<https://github.com/ResidentMario/missingno>

In [7]:

```
!conda install -c conda-forge missingno
```

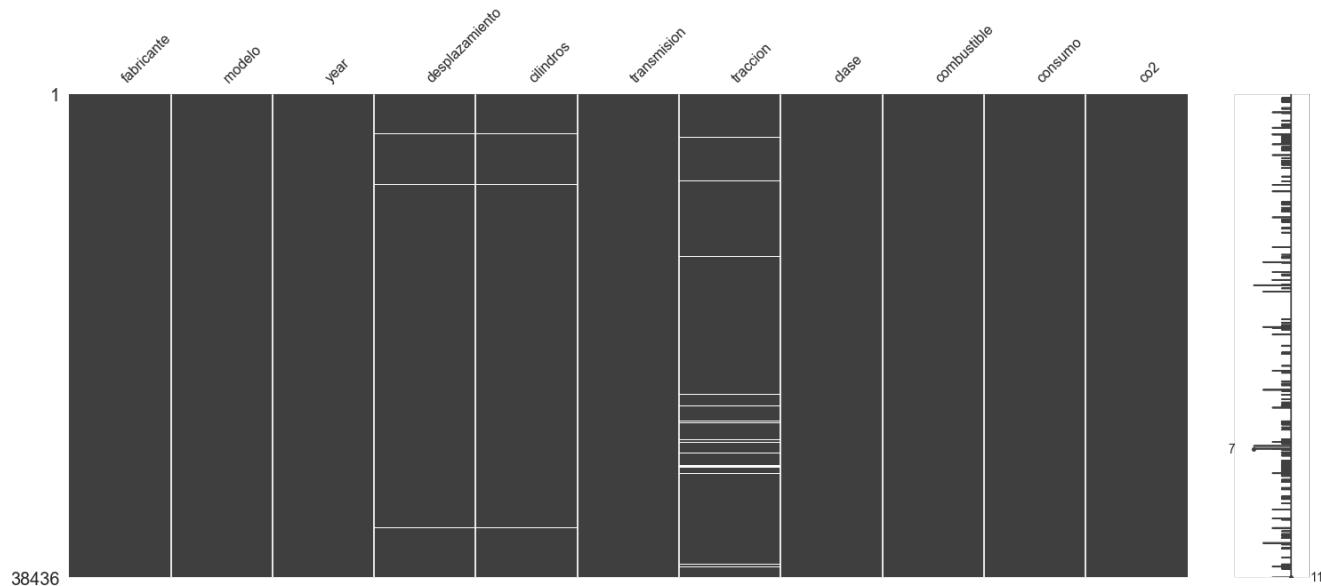
In [8]:

```
import missingno as msno
```

```
msno.matrix(vehiculos)
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1c3e3630>
```



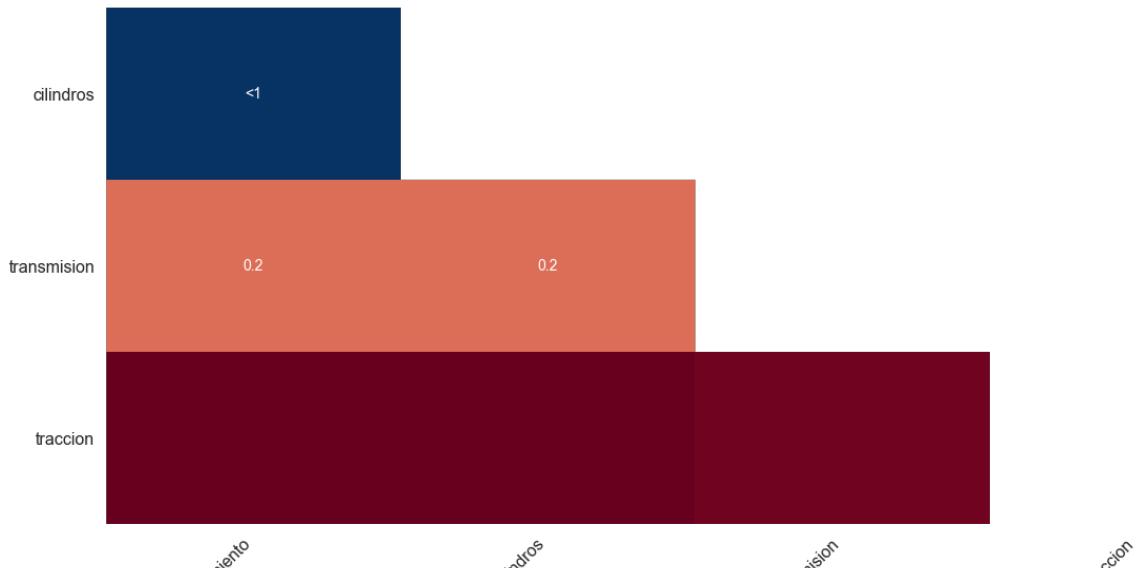
In [9]:

```
msno.heatmap(vehiculos)
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1d69cb38>
```

desplazamiento



desplazam.

dim.

transm.

rat

David Bermejo Simón.

Jose Luis Luengo Ramos.



Investigación en Tecnologías Data Science y Machine Learning con Python.

ANEXO II

DOCUMENTACIÓN MACHINE LEARNING

En este anexo se incluye toda la documentación impresa del repositorio.
El enlace a git de esta sección es:

<https://github.com/AC-SweetShop/Repo-TFG/tree/master/MACHINE%20LEARNING>

In [3]:

```
%load_ext watermark  
%watermark
```

The watermark extension is already loaded. To reload it, use:

```
%reload_ext watermark
```

```
2019-05-30T21:29:36+02:00
```

```
Cython 3.6.5
```

```
IPython 6.4.0
```

```
compiler    : GCC 7.2.0  
system      : Linux  
release     : 5.1.5-arch1-2-ARCH  
machine     : x86_64  
processor   :  
CPU cores   : 4  
interpreter: 64bit
```

Introducción a Machine Learning

Definición.

El Aprendizaje Automático consiste en una disciplina de las ciencias informáticas, relacionada con el desarrollo de la Inteligencia Artificial, y que sirve, como ya se ha dicho, para crear sistemas que pueden aprender por sí solos.

Es una tecnología que permite hacer automáticas una serie de operaciones con el fin de reducir la necesidad de que intervengan los seres humanos. Esto puede suponer una gran ventaja a la hora de controlar una ingente cantidad de información de un modo mucho más efectivo.

Lo que se denomina aprendizaje consiste en la capacidad del sistema para identificar una gran serie de patrones complejos determinados por una gran cantidad de parámetros.

Es decir, la máquina no aprende por sí misma, sino un algoritmo de su programación, que se modifica con la constante entrada de datos en la interfaz, y que puede, de ese modo, predecir escenarios futuros o tomar acciones de manera automática según ciertas condiciones. Como estas acciones se realizan de manera autónoma por el sistema, se dice que el aprendizaje es automático, sin intervención humana.

Tipos de Machine Learning

Podemos definir a estos tipos de machine learning como *Tipos de Algoritmos Machine Learning*. Se puede dividir en 3 grandes familias:

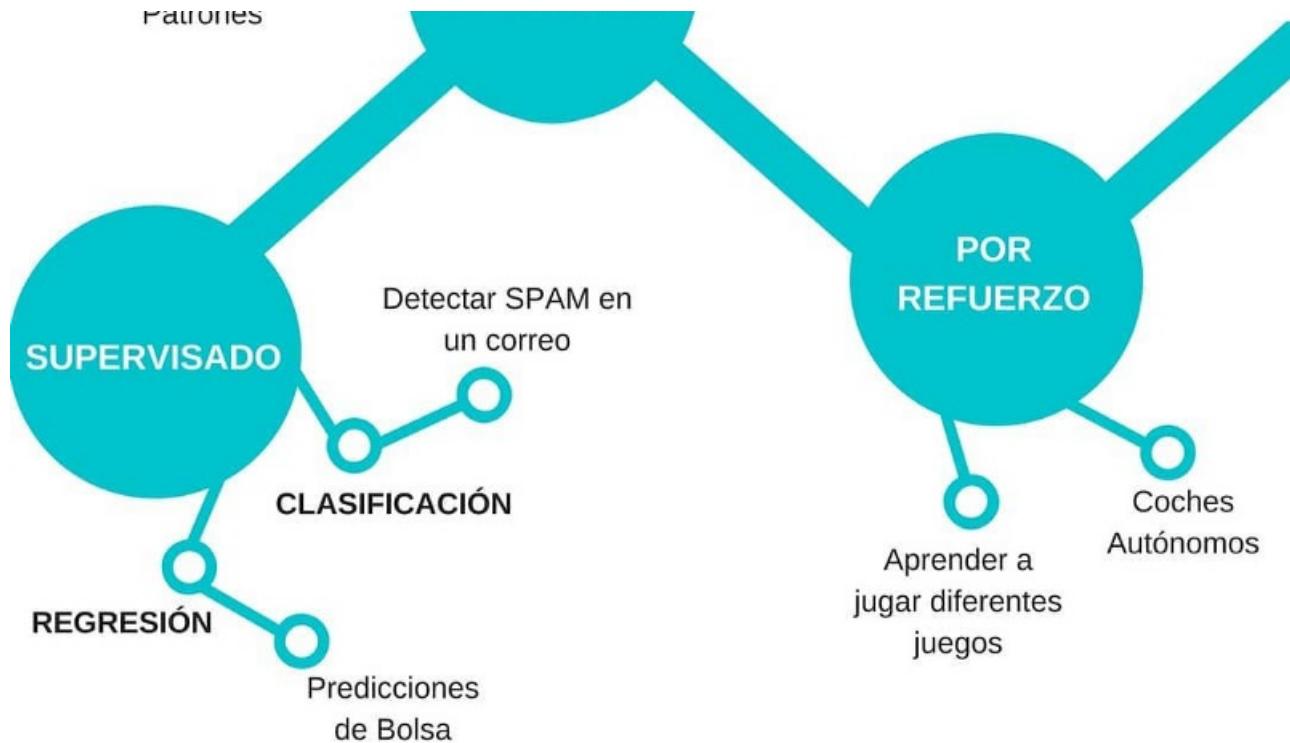
In [4]:

```
from IPython.display import Image  
Image("../RESOURCES/tipos_machine_learning.png")
```

Out [4]:

TIPOS DE MACHINE LEARNING





Aprendizaje Supervisado

Utilizamos la información contenida en el **dataset** de entrenamiento con el objetivo de declarar unas **labels** o etiquetas que permitan clasificar los datos (por ejemplo identificar y distinguir un coche de una moto) Este tipo de problemas, conocidos como **Problemas de clasificación** podría ser el realizado en el conjunto de datos de la planta Iris data de los años '30, que es empleado con frecuencia como ejemplo por diferentes librerías que trabajan con datos o gráficos como pandas o el propio scikit-learn. De cada planta de la especie Iris (setosa, versicolor y virginica) se han tomado medidas de longitud y ancho de sépalo y pétalo.

Otro tipo de problema sería el que permitiese utilizar el número de habitaciones, coordenadas geográficas y tamaño total de una vivienda para estimar su precio en el mercado (**Problema de regresión**).

Después del proceso de entrenamiento, dado un elemento sin **label** el modelo podrá determinar si se trata de un elemento u otro. A este tipo de aprendizaje se le denomina supervisado debido a que es necesario "entrenar" el modelo con muchos ejemplos para poder crear un "molde" o etiqueta para nuevos casos observados.

Aprendizaje No Supervisado.

En este caso y a diferencia del tipo de algoritmo para aprendizaje supervisado no existen "labels" o etiquetas extraídas del ajuste del modelo, si no que más bien abstrae patrones de la información directamente para tomar las decisiones (**Problema de clustering**)

Aprendizaje por Refuerzo.

Aquí el modelo se "entrena" a partir de la experiencia. Aquí la máquina toma una serie de decisiones, de tal forma que cuando toma una mala decisión se le castiga y cuando por el contrario toma una buena decisión se le premia. Es básicamente una prueba de ensayo-error que utiliza una función de "premio" para que se vaya optimizando con el tiempo. En este tipo de algoritmos se está apostando mucho ya que no requiere de grandes cantidades de datos.

En este proyecto se realizarán pruebas con ScikitLearn en el ámbito de los problemas de regresión y de clasificación.

In [1]:

```
%reload_ext watermark  
%watermark
```

2019-05-30T21:29:55+02:00

Cython 3.6.5
IPython 6.4.0

```
compiler : GCC 7.2.0  
system   : Linux  
release  : 5.1.5-arch1-2-ARCH  
machine  : x86_64  
processor:  
CPU cores: 4  
interpreter: 64bit
```

Regresión Lineal

En estadística la regresión lineal o ajuste lineal es un modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente Y, las variables independientes Xi y un término aleatorio ϵ . Este modelo puede ser expresado como:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \dots + b_n X_n + \epsilon$$

Donde:

- y: variable dependiente, explicada o regresando.
- x: variables explicativas, independientes o regresores.
- b: parámetros, miden la influencia que las variables explicativas tienen sobre el regrediendo.

Tipos de Regresión Lineal

Regresión Lineal Simple

Sólo se maneja una variable independiente, por lo que sólo cuenta con dos parámetros. Son de la siguiente forma:

$$Y = b_0 + b_1 X$$

Regresión Lineal Multiple

La regresión lineal permite trabajar con una variable a nivel de intervalo o razón. De la misma manera, es posible analizar la relación entre dos o más variables a través de ecuaciones, lo que se denomina regresión múltiple o regresión lineal múltiple.

Constantemente en la práctica de la investigación estadística, se encuentran variables que de alguna manera están relacionadas entre sí, por lo que es posible que una de las variables puedan relacionarse matemáticamente en función de otra u otras variables.

Maneja varias variables independientes. Cuenta con varios parámetros. Se expresan de la forma:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \dots + b_n X_n$$

Rectas de Regresión.

Las rectas de regresión son las rectas que mejor se ajustan a la nube de puntos (o también llamado diagrama de dispersión) generada por una distribución binomial

Supuestos del modelo de Regresión Lineal.

Para poder crear un modelo de regresión lineal es necesario que se cumpla con los siguientes supuestos:

1. Que la relación entre las variables sea lineal.
2. Que los errores en la medición de las variables explicativas sean independientes entre sí.
3. Que los errores tengan varianza constante. (Homocedasticidad)
4. Que los errores tengan una esperanza matemática igual a cero (los errores de una misma magnitud y distinto signo son equiprobables).
5. Que el error total sea la suma de todos los errores.

Scikit-Learn

Scipy.org nos proporcionara la mayoria de las librerias que se necesita tanto para la parte de machine-learning como la representacion de los datos mediante data science.

Librerias mas destacadas:

- Numpy.
- Scipy.
- Matplotlib.
- IPython.
- Pandas.
- Scikit-Learn.

Enlace a documentacion de Scikit: <https://www.scipy.org/scikits.html>

Utilizaremos ScikitLearn para Implementar Un Ejemplo de Regresión Lineal.

Modelo de Regresion Lineal mediante Scikit-Learn

Para ello utilizaremos el dataset de Boston House Data ya incluido en la libreria sklearn. sklearn nos proporciona además una estructura para el manejo de grandes cantidades de datos.

Un conjunto de datos o dataset corresponde a los contenidos de una única tabla de base de datos o una única matriz de datos de estadística, donde cada columna de la tabla representa una variable en particular, y cada fila representa a un miembro determinado del conjunto de datos que estamos tratando. En un conjunto de datos o dataset tenemos todos los valores que puede tener cada una de las variables, como por ejemplo la altura y el peso de un objeto, que corresponden a cada miembro del conjunto de datos. Cada uno de estos valores se conoce con el nombre de dato. El conjunto de datos puede incluir datos para uno o más miembros en función de su número de filas.

El dataset incluye también las relaciones entre las tablas que contienen los datos.

Si nos movemos en el contexto de Big Data, entendemos por dataset aquellos conjuntos de datos tan grandes que las aplicaciones de procesamiento de datos tradicionales no los pueden procesar debido a la gran cantidad de datos contenidos en la tabla o matriz.

Podríamos definir un dataset como una colección o representación de datos residentes en memoria con un modelo de programación relacional coherente e independientemente sea cual sea el origen de los datos que contiene.

Una de las principales características de los datasets es que ya tienen una estructura, a diferencia de los RDD, conocidos como conjuntos de datos desestructurados y definidos como una colección de elementos tolerante a fallos y son capaces de operar en paralelo.

Fuente: <https://www.deustoformacion.com/blog/programacion-diseno-web/que-son-datasets-dataframes-big-data>

In [2]:

```
from sklearn import datasets
```

In [3]:

```
# Cargamos el dataset de boston house.
dataset = datasets.load_boston()
```

In [4]:

```
dataset.keys()
```

Out[4]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [5]:

```
#Mostramos la descripcion del dataset.
print(dataset['DESCR'])
```

```
.. _boston_dataset:
```

Boston house prices dataset

Data Set Characteristics:

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is

usually the target.

```
:Attribute Information (in order):
- CRIM    per capita crime rate by town
- ZN      proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS   proportion of non-retail business acres per town
- CHAS    Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX     nitric oxides concentration (parts per 10 million)
- RM      average number of rooms per dwelling
- AGE     proportion of owner-occupied units built prior to 1940
- DIS     weighted distances to five Boston employment centres
- RAD     index of accessibility to radial highways
- TAX     full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B       1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT   % lower status of the population
- MEDV    Median value of owner-occupied homes in $1000's
```

```
:Missing Attribute Values: None
```

```
:Creator: Harrison, D. and Rubinfeld, D.L.
```

This is a copy of UCI ML housing dataset.

<https://archive.ics.uci.edu/ml/machine-learning-databases/housing/>

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression problems.

```
.. topic:: References
```

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Quinlan, R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

In [6]:

```
# array con los valores medios de las casas en miles de dolares.
var_objetivo = dataset['target']
var_objetivo
```

Out[6]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8 21.9 20.9 44.50 36.30.1 33.8 43.1 48.8 31]
```

```
36.5, 22.8, 30.7, 50., 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
32., 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46., 50., 32.2, 22.,
20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
20.3, 22.5, 29., 24.8, 22., 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
21., 23.8, 23.1, 20.4, 18.5, 25., 24.6, 23., 22.2, 19.3, 22.6,
19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19., 18.7,
32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25., 19.9, 20.8,
16.8, 21.9, 27.5, 21.9, 23.1, 50., 50., 50., 50., 50., 13.8,
13.8, 15., 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3, 8.8,
7.2, 10.5, 7.4, 10.2, 11.5, 15.1, 23.2, 9.7, 13.8, 12.7, 13.1,
12.5, 8.5, 5., 6.3, 5.6, 7.2, 12.1, 8.3, 8.5, 5., 11.9,
27.9, 17.2, 27.5, 15., 17.2, 17.9, 16.3, 7., 7.2, 7.5, 10.4,
8.8, 8.4, 16.7, 14.2, 20.8, 13.4, 11.7, 8.3, 10.2, 10.9, 11.,
9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4, 9.6, 8.7, 8.4, 12.8,
10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13., 13.4,
15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20., 16.4, 17.7,
19.5, 20.2, 21.4, 19.9, 19., 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
29.8, 13.8, 13.3, 16.7, 12., 14.6, 21.4, 23., 23.7, 25., 21.8,
20.6, 21.2, 19.1, 20.6, 15.2, 7., 8.1, 13.6, 20.1, 21.8, 24.5,
23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22., 11.9])
```

In [7]:

```
# Lista con las columnas
nombre_var_independiente = dataset['feature_names']
nombre_var_independiente
```

Out[7]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [8]:

```
# Lista con los registros del dataset por cada columna.
var_independiente = dataset['data']
var_independiente
```

Out[8]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ... , 1.5300e+01, 3.9690e+02,
       4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ... , 1.7800e+01, 3.9690e+02,
       9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ... , 1.7800e+01, 3.9283e+02,
       4.0300e+00],
       ... ,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ... , 2.1000e+01, 3.9690e+02,
       5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ... , 2.1000e+01, 3.9345e+02,
       6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ... , 2.1000e+01, 3.9690e+02,
       7.8800e+00]])
```

In [9]:

```
from sklearn.linear_model import LinearRegression
```

In [10]:

```
# Documentacion LinearRegression
LinearRegression?
```

In [11]:

```
model = LinearRegression()
```

In [12]:

```
#Ajustamos el modelo, de tal forma que las x sean las variables
#independientes y la y que sea la variable objetivo.
model.fit(X=var_independiente,y=var_objetivo)
```

Out[12]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                 normalize=False)
```

In [13]:

```
#Una vez ajustado, tenemos el termino independiente
model.intercept_
```

Out[13]:

36.4594883850899

In [14]:

```
"""
Mostramos tambien las betas. que miden la influencia
que las variables independientes tienen sobre el regrediendo
(variables dependientes o 'Y').
"""

model.coef_
```

Out[14]:

```
array([-1.08011358e-01,  4.64204584e-02,  2.05586264e-02,  2.68673382e+00,
       -1.77666112e+01,  3.80986521e+00,  6.92224640e-04,  -1.47556685e+00,
       3.06049479e-01, -1.23345939e-02, -9.52747232e-01,  9.31168327e-03,
      -5.24758378e-01])
```

In [15]:

```
#Documentacion de predict
model.predict?
"""

.predict solo necesita el valor x, que es la estimacion.
Es decir, si hemos entrenado con el metodo .fit el modelo
con las variables independientes y la Y como la variable
depdiente u objetivo con con predict lo que hacemos es obtener
la estimacion de Y
"""
```

Out[15]:

```
'\n.predict solo necesita el valor x, que es la estimacion.\nEs decir, si hemos entrenado con el m
etodo .fit el modelo\ncon las variables independientes y la Y como la variable \ndepdiente u obje
tivo con con predict lo que hacemos es obtener\nla estimacion de Y \n'
```

In [16]:

```
#Guardamos las predicciones en una variable
predicciones = model.predict(var_independiente)
predicciones
```

Out[16]:

```
array([30.00384338, 25.02556238, 30.56759672, 28.60703649, 27.94352423,
       25.25628446, 23.00180827, 19.53598843, 11.52363685, 18.92026211,
       18.99949651, 21.58679568, 20.90652153, 19.55290281, 19.28348205,
       19.29748321, 20.52750979, 16.91140135, 16.17801106, 18.40613603,
       12.52385753, 17.67103669, 15.83288129, 13.80628535, 15.67833832,
       13.38668561, 15.46397655, 14.70847428, 19.54737285, 20.8764282 ,
       11.45511759, 18.05923295, 8.81105736, 14.28275814, 13.70675891,
       23.81463526, 22.34193708, 23.10891142, 22.91502612, 31.35762569,
       34.21510225, 28.02056414, 25.20386628, 24.60979273, 22.94149176,
       22.09669817, 20.42320032, 18.03655088, 9.10655377, 17.20607751,
       21.28152535, 23.97222285, 27.6558508 , 24.04901809, 15.3618477 ,
       31.15264947, 24.85686978, 33.10919806, 21.77537987, 21.08493555,
       17.8725804 , 18.51110208, 23.98742856, 22.55408869, 23.37308644,
       30.36148358, 25.53056512, 21.11338564, 17.42153786, 20.78483633,
       25.20148859, 21.7426577 , 24.55744957, 24.04295712, 25.50499716,
       23.9669302 , 22.94545403, 23.35699818, 21.26198266, 22.42817373,
       28.40576968, 26.99486086, 26.03576297, 25.05873482, 24.78456674,
       27.79049195, 22.16853423, 25.89276415, 30.67461827, 30.83110623,
       27.1190194 , 27.41266734, 28.94122762, 29.08105546, 27.03977365,
       28.62459949, 24.72744978, 35.78159518, 35.11454587, 32.25102801,
       24.58022019, 25.59413475, 19.79013684, 20.31167129, 21.43482591,
       18.53994008, 17.18755992, 20.75049026, 22.64829115, 19.7720367 ,
       20.64965864, 26.52586744, 20.77323638, 20.71548315, 25.17208881,
       20.43025591, 23.37724626, 23.69043261, 20.33578364, 20.79180873,
       21.91632071, 22.47107777, 20.55738556, 16.36661977, 20.56099819,
       22.48178446, 14.61706633, 15.17876684, 18.93868592, 14.05573285,
       20.03527399, 19.41013402, 20.06191566, 15.75807673, 13.25645238,
       17.26277735, 15.87841883, 19.36163954, 13.81483897, 16.44881475.
```

```

11.28277700, 15.07011000, 19.00100000, 19.01100000, 20.10011000,
13.57141932, 3.98885508, 14.59495478, 12.1488148, 8.72822362,
12.03585343, 15.82082058, 8.5149902, 9.71844139, 14.80451374,
20.83858153, 18.30101169, 20.12282558, 17.28601894, 22.36600228,
20.10375923, 13.62125891, 33.25982697, 29.03017268, 25.56752769,
32.70827666, 36.77467015, 40.55765844, 41.84728168, 24.78867379,
25.37889238, 37.20347455, 23.08748747, 26.40273955, 26.65382114,
22.5551466, 24.29082812, 22.97657219, 29.07194308, 26.5219434,
30.72209056, 25.61669307, 29.13740979, 31.43571968, 32.92231568,
34.72440464, 27.76552111, 33.88787321, 30.99238036, 22.71820008,
24.7664781, 35.88497226, 33.42476722, 32.41199147, 34.51509949,
30.76109485, 30.28934141, 32.91918714, 32.11260771, 31.55871004,
40.84555721, 36.12770079, 32.6692081, 34.70469116, 30.09345162,
30.64393906, 29.28719501, 37.07148392, 42.03193124, 43.18949844,
22.69034796, 23.68284712, 17.85447214, 23.49428992, 17.00587718,
22.39251096, 17.06042754, 22.73892921, 25.21942554, 11.11916737,
24.51049148, 26.60334775, 28.35518713, 24.91525464, 29.68652768,
33.18419746, 23.77456656, 32.14051958, 29.7458199, 38.37102453,
39.81461867, 37.58605755, 32.3995325, 35.45665242, 31.23411512,
24.48449227, 33.28837292, 38.0481048, 37.16328631, 31.71383523,
25.26705571, 30.10010745, 32.71987156, 28.42717057, 28.42940678,
27.29375938, 23.74262478, 24.12007891, 27.40208414, 16.3285756,
13.39891261, 20.01638775, 19.86184428, 21.2883131, 24.0798915,
24.20633547, 25.04215821, 24.91964007, 29.94563374, 23.97228316,
21.69580887, 37.51109239, 43.30239043, 36.48361421, 34.98988594,
34.81211508, 37.16631331, 40.98928501, 34.44634089, 35.83397547,
28.245743, 31.22673593, 40.8395575, 39.31792393, 25.70817905,
22.30295533, 27.20340972, 28.51169472, 35.47676598, 36.10639164,
33.79668274, 35.61085858, 34.83993382, 30.35192656, 35.30980701,
38.79756966, 34.33123186, 40.33963075, 44.67308339, 31.59689086,
27.3565923, 20.10174154, 27.04206674, 27.2136458, 26.91395839,
33.43563311, 34.40349633, 31.8333982, 25.81783237, 24.42982348,
28.45764337, 27.36266999, 19.53928758, 29.11309844, 31.91054611,
30.77159449, 28.94275871, 28.88191022, 32.79887232, 33.20905456,
30.76831792, 35.56226857, 32.70905124, 28.64244237, 23.58965827,
18.54266897, 26.87889843, 23.28133979, 25.54580246, 25.48120057,
20.53909901, 17.61572573, 18.37581686, 24.29070277, 21.32529039,
24.88682244, 24.86937282, 22.86952447, 19.45123791, 25.11783401,
24.66786913, 23.68076177, 19.34089616, 21.17418105, 24.25249073,
21.59260894, 19.98446605, 23.33888, 22.14060692, 21.55509929,
20.61872907, 20.16097176, 19.28490387, 22.1667232, 21.24965774,
21.42939305, 30.32788796, 22.04734975, 27.70647912, 28.54794117,
16.54501121, 14.78359641, 25.27380082, 27.54205117, 22.14837562,
20.45944095, 20.54605423, 16.88063827, 25.40253506, 14.32486632,
16.59488462, 19.63704691, 22.71806607, 22.20218887, 19.20548057,
22.66616105, 18.93192618, 18.22846804, 20.23150811, 37.4944739,
14.28190734, 15.54286248, 10.83162324, 23.80072902, 32.6440736,
34.60684042, 24.94331333, 25.9998091, 6.126325, 0.77779806,
25.30713064, 17.74061065, 20.23274414, 15.83331301, 16.83512587,
14.36994825, 18.47682833, 13.4276828, 13.06177512, 3.27918116,
8.06022171, 6.12842196, 5.6186481, 6.4519857, 14.20764735,
17.21225183, 17.29887265, 9.89116643, 20.22124193, 17.94181175,
20.30445783, 19.29559075, 16.33632779, 6.55162319, 10.89016778,
11.88145871, 17.81174507, 18.26126587, 12.97948781, 7.37816361,
8.21115861, 8.06626193, 19.98294786, 13.70756369, 19.85268454,
15.22308298, 16.96071981, 1.71851807, 11.80578387, -4.28131071,
9.58376737, 13.36660811, 6.89562363, 6.14779852, 14.60661794,
19.6000267, 18.12427476, 18.52177132, 13.1752861, 14.62617624,
9.92374976, 16.34590647, 14.07519426, 14.25756243, 13.04234787,
18.15955693, 18.69554354, 21.527283, 17.03141861, 15.96090435,
13.36141611, 14.52079384, 8.81976005, 4.86751102, 13.06591313,
12.70609699, 17.29558059, 18.740485, 18.05901029, 11.51474683,
11.97400359, 17.68344618, 18.12695239, 17.5183465, 17.22742507,
16.52271631, 19.41291095, 18.58215236, 22.48944791, 15.28000133,
15.82089335, 12.68725581, 12.8763379, 17.18668531, 18.51247609,
19.04860533, 20.17208927, 19.7740732, 22.42940768, 20.31911854,
17.88616253, 14.37478523, 16.94776851, 16.98405762, 18.58838397,
20.16719441, 22.97718032, 22.45580726, 25.57824627, 16.39147632,
16.1114628, 20.534816, 11.54272738, 19.20496304, 21.86276391,
23.46878866, 27.09887315, 28.56994302, 21.08398783, 19.45516196,
22.22225914, 19.65591961, 21.32536104, 11.85583717, 8.22386687,
3.66399672, 13.75908538, 15.93118545, 20.62662054, 20.61249414,
16.88541964, 14.01320787, 19.10854144, 21.29805174, 18.45498841,
20.46870847, 23.53334055, 22.37571892, 27.6274261, 26.12796681,
22.34421229]
)

```

In [17]:

```
#Registros de la prediccion (igual que la variable dependiente Y)
predicciones.shape
```

Out[17]:

(506,)

In [18]:

```
"""
Si queremos mostrarlo de una forma mas visible segun el
valor real y el valor estimado podemos realizarlo mediante
la siguiente funcion, la cual llama al metodo zip que concatena
elementos de una lista 'n1' con elementos de una lista 'n2'
"""

for y, y_pred in list(zip(var_objetivo,predicciones)):
    print('Valor Real: {:.3f} Valor Estimado: {:.5f}'.format(y,y_pred))
```

Valor Real: 24.000 Valor Estimado: 30.00384
Valor Real: 21.600 Valor Estimado: 25.02556
Valor Real: 34.700 Valor Estimado: 30.56760
Valor Real: 33.400 Valor Estimado: 28.60704
Valor Real: 36.200 Valor Estimado: 27.94352
Valor Real: 28.700 Valor Estimado: 25.25628
Valor Real: 22.900 Valor Estimado: 23.00181
Valor Real: 27.100 Valor Estimado: 19.53599
Valor Real: 16.500 Valor Estimado: 11.52364
Valor Real: 18.900 Valor Estimado: 18.92026
Valor Real: 15.000 Valor Estimado: 18.99950
Valor Real: 18.900 Valor Estimado: 21.58680
Valor Real: 21.700 Valor Estimado: 20.90652
Valor Real: 20.400 Valor Estimado: 19.55290
Valor Real: 18.200 Valor Estimado: 19.28348
Valor Real: 19.900 Valor Estimado: 19.29748
Valor Real: 23.100 Valor Estimado: 20.52751
Valor Real: 17.500 Valor Estimado: 16.91140
Valor Real: 20.200 Valor Estimado: 16.17801
Valor Real: 18.200 Valor Estimado: 18.40614
Valor Real: 13.600 Valor Estimado: 12.52386
Valor Real: 19.600 Valor Estimado: 17.67104
Valor Real: 15.200 Valor Estimado: 15.83288
Valor Real: 14.500 Valor Estimado: 13.80629
Valor Real: 15.600 Valor Estimado: 15.67834
Valor Real: 13.900 Valor Estimado: 13.38669
Valor Real: 16.600 Valor Estimado: 15.46398
Valor Real: 14.800 Valor Estimado: 14.70847
Valor Real: 18.400 Valor Estimado: 19.54737
Valor Real: 21.000 Valor Estimado: 20.87643
Valor Real: 12.700 Valor Estimado: 11.45512
Valor Real: 14.500 Valor Estimado: 18.05923
Valor Real: 13.200 Valor Estimado: 8.81106
Valor Real: 13.100 Valor Estimado: 14.28276
Valor Real: 13.500 Valor Estimado: 13.70676
Valor Real: 18.900 Valor Estimado: 23.81464
Valor Real: 20.000 Valor Estimado: 22.34194
Valor Real: 21.000 Valor Estimado: 23.10891
Valor Real: 24.700 Valor Estimado: 22.91503
Valor Real: 30.800 Valor Estimado: 31.35763
Valor Real: 34.900 Valor Estimado: 34.21510
Valor Real: 26.600 Valor Estimado: 28.02056
Valor Real: 25.300 Valor Estimado: 25.20387
Valor Real: 24.700 Valor Estimado: 24.60979
Valor Real: 21.200 Valor Estimado: 22.94149
Valor Real: 19.300 Valor Estimado: 22.09670
Valor Real: 20.000 Valor Estimado: 20.42320
Valor Real: 16.600 Valor Estimado: 18.03655
Valor Real: 14.400 Valor Estimado: 9.10655
Valor Real: 19.400 Valor Estimado: 17.20608
Valor Real: 19.700 Valor Estimado: 21.28153
Valor Real: 20.500 Valor Estimado: 23.97222
Valor Real: 25.000 Valor Estimado: 27.65585
Valor Real: 23.400 Valor Estimado: 24.04902
Valor Real: 18.900 Valor Estimado: 15.36185
Valor Real: 35.400 Valor Estimado: 31.15265
Valor Real: 24.700 Valor Estimado: 24.85687
Valor Real: 31.600 Valor Estimado: 33.10920
Valor Real: 23.300 Valor Estimado: 21.77538
Valor Real: 19.600 Valor Estimado: 21.08494

Valor Real: 19.000 Valor Estimado: 21.0000
Valor Real: 18.700 Valor Estimado: 17.87258
Valor Real: 16.000 Valor Estimado: 18.51110
Valor Real: 22.200 Valor Estimado: 23.98743
Valor Real: 25.000 Valor Estimado: 22.55409
Valor Real: 33.000 Valor Estimado: 23.37309
Valor Real: 23.500 Valor Estimado: 30.36148
Valor Real: 19.400 Valor Estimado: 25.53057
Valor Real: 22.000 Valor Estimado: 21.11339
Valor Real: 17.400 Valor Estimado: 17.42154
Valor Real: 20.900 Valor Estimado: 20.78484
Valor Real: 24.200 Valor Estimado: 25.20149
Valor Real: 21.700 Valor Estimado: 21.74266
Valor Real: 22.800 Valor Estimado: 24.55745
Valor Real: 23.400 Valor Estimado: 24.04296
Valor Real: 24.100 Valor Estimado: 25.50500
Valor Real: 21.400 Valor Estimado: 23.96693
Valor Real: 20.000 Valor Estimado: 22.94545
Valor Real: 20.800 Valor Estimado: 23.35700
Valor Real: 21.200 Valor Estimado: 21.26198
Valor Real: 20.300 Valor Estimado: 22.42817
Valor Real: 28.000 Valor Estimado: 28.40577
Valor Real: 23.900 Valor Estimado: 26.99486
Valor Real: 24.800 Valor Estimado: 26.03576
Valor Real: 22.900 Valor Estimado: 25.05873
Valor Real: 23.900 Valor Estimado: 24.78457
Valor Real: 26.600 Valor Estimado: 27.79049
Valor Real: 22.500 Valor Estimado: 22.16853
Valor Real: 22.200 Valor Estimado: 25.89276
Valor Real: 23.600 Valor Estimado: 30.67462
Valor Real: 28.700 Valor Estimado: 30.83111
Valor Real: 22.600 Valor Estimado: 27.11902
Valor Real: 22.000 Valor Estimado: 27.41267
Valor Real: 22.900 Valor Estimado: 28.94123
Valor Real: 25.000 Valor Estimado: 29.08106
Valor Real: 20.600 Valor Estimado: 27.03977
Valor Real: 28.400 Valor Estimado: 28.62460
Valor Real: 21.400 Valor Estimado: 24.72745
Valor Real: 38.700 Valor Estimado: 35.78160
Valor Real: 43.800 Valor Estimado: 35.11455
Valor Real: 33.200 Valor Estimado: 32.25103
Valor Real: 27.500 Valor Estimado: 24.58022
Valor Real: 26.500 Valor Estimado: 25.59413
Valor Real: 18.600 Valor Estimado: 19.79014
Valor Real: 19.300 Valor Estimado: 20.31167
Valor Real: 20.100 Valor Estimado: 21.43483
Valor Real: 19.500 Valor Estimado: 18.53994
Valor Real: 19.500 Valor Estimado: 17.18756
Valor Real: 20.400 Valor Estimado: 20.75049
Valor Real: 19.800 Valor Estimado: 22.64829
Valor Real: 19.400 Valor Estimado: 19.77204
Valor Real: 21.700 Valor Estimado: 20.64966
Valor Real: 22.800 Valor Estimado: 26.52587
Valor Real: 18.800 Valor Estimado: 20.77324
Valor Real: 18.700 Valor Estimado: 20.71548
Valor Real: 18.500 Valor Estimado: 25.17209
Valor Real: 18.300 Valor Estimado: 20.43026
Valor Real: 21.200 Valor Estimado: 23.37725
Valor Real: 19.200 Valor Estimado: 23.69043
Valor Real: 20.400 Valor Estimado: 20.33578
Valor Real: 19.300 Valor Estimado: 20.79181
Valor Real: 22.000 Valor Estimado: 21.91632
Valor Real: 20.300 Valor Estimado: 22.47108
Valor Real: 20.500 Valor Estimado: 20.55739
Valor Real: 17.300 Valor Estimado: 16.36662
Valor Real: 18.800 Valor Estimado: 20.56100
Valor Real: 21.400 Valor Estimado: 22.48178
Valor Real: 15.700 Valor Estimado: 14.61707
Valor Real: 16.200 Valor Estimado: 15.17877
Valor Real: 18.000 Valor Estimado: 18.93869
Valor Real: 14.300 Valor Estimado: 14.05573
Valor Real: 19.200 Valor Estimado: 20.03527
Valor Real: 19.600 Valor Estimado: 19.41013
Valor Real: 23.000 Valor Estimado: 20.06192
Valor Real: 18.400 Valor Estimado: 15.75808
Valor Real: 15.600 Valor Estimado: 13.25645
Valor Real: 18.100 Valor Estimado: 17.26278
Valor Real: 17.400 Valor Estimado: 15.87842

valor Real: 17.400 valor Estimado: 19.00000
Valor Real: 17.100 Valor Estimado: 19.36164
Valor Real: 13.300 Valor Estimado: 13.81484
Valor Real: 17.800 Valor Estimado: 16.44881
Valor Real: 14.000 Valor Estimado: 13.57142
Valor Real: 14.400 Valor Estimado: 3.98886
Valor Real: 13.400 Valor Estimado: 14.59495
Valor Real: 15.600 Valor Estimado: 12.14881
Valor Real: 11.800 Valor Estimado: 8.72822
Valor Real: 13.800 Valor Estimado: 12.03585
Valor Real: 15.600 Valor Estimado: 15.82082
Valor Real: 14.600 Valor Estimado: 8.51499
Valor Real: 17.800 Valor Estimado: 9.71844
Valor Real: 15.400 Valor Estimado: 14.80451
Valor Real: 21.500 Valor Estimado: 20.83858
Valor Real: 19.600 Valor Estimado: 18.30101
Valor Real: 15.300 Valor Estimado: 20.12283
Valor Real: 19.400 Valor Estimado: 17.28602
Valor Real: 17.000 Valor Estimado: 22.36600
Valor Real: 15.600 Valor Estimado: 20.10376
Valor Real: 13.100 Valor Estimado: 13.62126
Valor Real: 41.300 Valor Estimado: 33.25983
Valor Real: 24.300 Valor Estimado: 29.03017
Valor Real: 23.300 Valor Estimado: 25.56753
Valor Real: 27.000 Valor Estimado: 32.70828
Valor Real: 50.000 Valor Estimado: 36.77467
Valor Real: 50.000 Valor Estimado: 40.55766
Valor Real: 50.000 Valor Estimado: 41.84728
Valor Real: 22.700 Valor Estimado: 24.78867
Valor Real: 25.000 Valor Estimado: 25.37889
Valor Real: 50.000 Valor Estimado: 37.20347
Valor Real: 23.800 Valor Estimado: 23.08749
Valor Real: 23.800 Valor Estimado: 26.40274
Valor Real: 22.300 Valor Estimado: 26.65382
Valor Real: 17.400 Valor Estimado: 22.55515
Valor Real: 19.100 Valor Estimado: 24.29083
Valor Real: 23.100 Valor Estimado: 22.97657
Valor Real: 23.600 Valor Estimado: 29.07194
Valor Real: 22.600 Valor Estimado: 26.52194
Valor Real: 29.400 Valor Estimado: 30.72209
Valor Real: 23.200 Valor Estimado: 25.61669
Valor Real: 24.600 Valor Estimado: 29.13741
Valor Real: 29.900 Valor Estimado: 31.43572
Valor Real: 37.200 Valor Estimado: 32.92232
Valor Real: 39.800 Valor Estimado: 34.72440
Valor Real: 36.200 Valor Estimado: 27.76552
Valor Real: 37.900 Valor Estimado: 33.88787
Valor Real: 32.500 Valor Estimado: 30.99238
Valor Real: 26.400 Valor Estimado: 22.71820
Valor Real: 29.600 Valor Estimado: 24.76648
Valor Real: 50.000 Valor Estimado: 35.88497
Valor Real: 32.000 Valor Estimado: 33.42477
Valor Real: 29.800 Valor Estimado: 32.41199
Valor Real: 34.900 Valor Estimado: 34.51510
Valor Real: 37.000 Valor Estimado: 30.76109
Valor Real: 30.500 Valor Estimado: 30.28934
Valor Real: 36.400 Valor Estimado: 32.91919
Valor Real: 31.100 Valor Estimado: 32.11261
Valor Real: 29.100 Valor Estimado: 31.55871
Valor Real: 50.000 Valor Estimado: 40.84556
Valor Real: 33.300 Valor Estimado: 36.12770
Valor Real: 30.300 Valor Estimado: 32.66921
Valor Real: 34.600 Valor Estimado: 34.70469
Valor Real: 34.900 Valor Estimado: 30.09345
Valor Real: 32.900 Valor Estimado: 30.64394
Valor Real: 24.100 Valor Estimado: 29.28720
Valor Real: 42.300 Valor Estimado: 37.07148
Valor Real: 48.500 Valor Estimado: 42.03193
Valor Real: 50.000 Valor Estimado: 43.18950
Valor Real: 22.600 Valor Estimado: 22.69035
Valor Real: 24.400 Valor Estimado: 23.68285
Valor Real: 22.500 Valor Estimado: 17.85447
Valor Real: 24.400 Valor Estimado: 23.49429
Valor Real: 20.000 Valor Estimado: 17.00588
Valor Real: 21.700 Valor Estimado: 22.39251
Valor Real: 19.300 Valor Estimado: 17.06043
Valor Real: 22.400 Valor Estimado: 22.73893
Valor Real: 28.100 Valor Estimado: 25.21012

valor Real: 20.100 valor Estimado: 20.21940
Valor Real: 23.700 Valor Estimado: 11.11917
Valor Real: 25.000 Valor Estimado: 24.51049
Valor Real: 23.300 Valor Estimado: 26.60335
Valor Real: 28.700 Valor Estimado: 28.35519
Valor Real: 21.500 Valor Estimado: 24.91525
Valor Real: 23.000 Valor Estimado: 29.68653
Valor Real: 26.700 Valor Estimado: 33.18420
Valor Real: 21.700 Valor Estimado: 23.77457
Valor Real: 27.500 Valor Estimado: 32.14052
Valor Real: 30.100 Valor Estimado: 29.74582
Valor Real: 44.800 Valor Estimado: 38.37102
Valor Real: 50.000 Valor Estimado: 39.81462
Valor Real: 37.600 Valor Estimado: 37.58606
Valor Real: 31.600 Valor Estimado: 32.39953
Valor Real: 46.700 Valor Estimado: 35.45665
Valor Real: 31.500 Valor Estimado: 31.23412
Valor Real: 24.300 Valor Estimado: 24.48449
Valor Real: 31.700 Valor Estimado: 33.28837
Valor Real: 41.700 Valor Estimado: 38.04810
Valor Real: 48.300 Valor Estimado: 37.16329
Valor Real: 29.000 Valor Estimado: 31.71384
Valor Real: 24.000 Valor Estimado: 25.26706
Valor Real: 25.100 Valor Estimado: 30.10011
Valor Real: 31.500 Valor Estimado: 32.71987
Valor Real: 23.700 Valor Estimado: 28.42717
Valor Real: 23.300 Valor Estimado: 28.42941
Valor Real: 22.000 Valor Estimado: 27.29376
Valor Real: 20.100 Valor Estimado: 23.74262
Valor Real: 22.200 Valor Estimado: 24.12008
Valor Real: 23.700 Valor Estimado: 27.40208
Valor Real: 17.600 Valor Estimado: 16.32858
Valor Real: 18.500 Valor Estimado: 13.39891
Valor Real: 24.300 Valor Estimado: 20.01639
Valor Real: 20.500 Valor Estimado: 19.86184
Valor Real: 24.500 Valor Estimado: 21.28831
Valor Real: 26.200 Valor Estimado: 24.07989
Valor Real: 24.400 Valor Estimado: 24.20634
Valor Real: 24.800 Valor Estimado: 25.04216
Valor Real: 29.600 Valor Estimado: 24.91964
Valor Real: 42.800 Valor Estimado: 29.94563
Valor Real: 21.900 Valor Estimado: 23.97228
Valor Real: 20.900 Valor Estimado: 21.69581
Valor Real: 44.000 Valor Estimado: 37.51109
Valor Real: 50.000 Valor Estimado: 43.30239
Valor Real: 36.000 Valor Estimado: 36.48361
Valor Real: 30.100 Valor Estimado: 34.98989
Valor Real: 33.800 Valor Estimado: 34.81212
Valor Real: 43.100 Valor Estimado: 37.16631
Valor Real: 48.800 Valor Estimado: 40.98929
Valor Real: 31.000 Valor Estimado: 34.44634
Valor Real: 36.500 Valor Estimado: 35.83398
Valor Real: 22.800 Valor Estimado: 28.24574
Valor Real: 30.700 Valor Estimado: 31.22674
Valor Real: 50.000 Valor Estimado: 40.83956
Valor Real: 43.500 Valor Estimado: 39.31792
Valor Real: 20.700 Valor Estimado: 25.70818
Valor Real: 21.100 Valor Estimado: 22.30296
Valor Real: 25.200 Valor Estimado: 27.20341
Valor Real: 24.400 Valor Estimado: 28.51169
Valor Real: 35.200 Valor Estimado: 35.47677
Valor Real: 32.400 Valor Estimado: 36.10639
Valor Real: 32.000 Valor Estimado: 33.79668
Valor Real: 33.200 Valor Estimado: 35.61086
Valor Real: 33.100 Valor Estimado: 34.83993
Valor Real: 29.100 Valor Estimado: 30.35193
Valor Real: 35.100 Valor Estimado: 35.30981
Valor Real: 45.400 Valor Estimado: 38.79757
Valor Real: 35.400 Valor Estimado: 34.33123
Valor Real: 46.000 Valor Estimado: 40.33963
Valor Real: 50.000 Valor Estimado: 44.67308
Valor Real: 32.200 Valor Estimado: 31.59689
Valor Real: 22.000 Valor Estimado: 27.35659
Valor Real: 20.100 Valor Estimado: 20.10174
Valor Real: 23.200 Valor Estimado: 27.04207
Valor Real: 22.300 Valor Estimado: 27.21365
Valor Real: 24.800 Valor Estimado: 26.91396
Valor Real: 20.500 Valor Estimado: 22.12562

Valor Real: 20.500 Valor Estimado: 25.45505
Valor Real: 37.300 Valor Estimado: 34.40350
Valor Real: 27.900 Valor Estimado: 31.83340
Valor Real: 23.900 Valor Estimado: 25.81783
Valor Real: 21.700 Valor Estimado: 24.42982
Valor Real: 28.600 Valor Estimado: 28.45764
Valor Real: 27.100 Valor Estimado: 27.36267
Valor Real: 20.300 Valor Estimado: 19.53929
Valor Real: 22.500 Valor Estimado: 29.11310
Valor Real: 29.000 Valor Estimado: 31.91055
Valor Real: 24.800 Valor Estimado: 30.77159
Valor Real: 22.000 Valor Estimado: 28.94276
Valor Real: 26.400 Valor Estimado: 28.88191
Valor Real: 33.100 Valor Estimado: 32.79887
Valor Real: 36.100 Valor Estimado: 33.20905
Valor Real: 28.400 Valor Estimado: 30.76832
Valor Real: 33.400 Valor Estimado: 35.56227
Valor Real: 28.200 Valor Estimado: 32.70905
Valor Real: 22.800 Valor Estimado: 28.64244
Valor Real: 20.300 Valor Estimado: 23.58966
Valor Real: 16.100 Valor Estimado: 18.54267
Valor Real: 22.100 Valor Estimado: 26.87890
Valor Real: 19.400 Valor Estimado: 23.28134
Valor Real: 21.600 Valor Estimado: 25.54580
Valor Real: 23.800 Valor Estimado: 25.48120
Valor Real: 16.200 Valor Estimado: 20.53910
Valor Real: 17.800 Valor Estimado: 17.61573
Valor Real: 19.800 Valor Estimado: 18.37582
Valor Real: 23.100 Valor Estimado: 24.29070
Valor Real: 21.000 Valor Estimado: 21.32529
Valor Real: 23.800 Valor Estimado: 24.88682
Valor Real: 23.100 Valor Estimado: 24.86937
Valor Real: 20.400 Valor Estimado: 22.86952
Valor Real: 18.500 Valor Estimado: 19.45124
Valor Real: 25.000 Valor Estimado: 25.11783
Valor Real: 24.600 Valor Estimado: 24.66787
Valor Real: 23.000 Valor Estimado: 23.68076
Valor Real: 22.200 Valor Estimado: 19.34090
Valor Real: 19.300 Valor Estimado: 21.17418
Valor Real: 22.600 Valor Estimado: 24.25249
Valor Real: 19.800 Valor Estimado: 21.59261
Valor Real: 17.100 Valor Estimado: 19.98447
Valor Real: 19.400 Valor Estimado: 23.33888
Valor Real: 22.200 Valor Estimado: 22.14061
Valor Real: 20.700 Valor Estimado: 21.55510
Valor Real: 21.100 Valor Estimado: 20.61873
Valor Real: 19.500 Valor Estimado: 20.16097
Valor Real: 18.500 Valor Estimado: 19.28490
Valor Real: 20.600 Valor Estimado: 22.16672
Valor Real: 19.000 Valor Estimado: 21.24966
Valor Real: 18.700 Valor Estimado: 21.42939
Valor Real: 32.700 Valor Estimado: 30.32789
Valor Real: 16.500 Valor Estimado: 22.04735
Valor Real: 23.900 Valor Estimado: 27.70648
Valor Real: 31.200 Valor Estimado: 28.54794
Valor Real: 17.500 Valor Estimado: 16.54501
Valor Real: 17.200 Valor Estimado: 14.78360
Valor Real: 23.100 Valor Estimado: 25.27380
Valor Real: 24.500 Valor Estimado: 27.54205
Valor Real: 26.600 Valor Estimado: 22.14838
Valor Real: 22.900 Valor Estimado: 20.45944
Valor Real: 24.100 Valor Estimado: 20.54605
Valor Real: 18.600 Valor Estimado: 16.88064
Valor Real: 30.100 Valor Estimado: 25.40254
Valor Real: 18.200 Valor Estimado: 14.32487
Valor Real: 20.600 Valor Estimado: 16.59488
Valor Real: 17.800 Valor Estimado: 19.63705
Valor Real: 21.700 Valor Estimado: 22.71807
Valor Real: 22.700 Valor Estimado: 22.20219
Valor Real: 22.600 Valor Estimado: 19.20548
Valor Real: 25.000 Valor Estimado: 22.66616
Valor Real: 19.900 Valor Estimado: 18.93193
Valor Real: 20.800 Valor Estimado: 18.22847
Valor Real: 16.800 Valor Estimado: 20.23151
Valor Real: 21.900 Valor Estimado: 37.49447
Valor Real: 27.500 Valor Estimado: 14.28191
Valor Real: 21.900 Valor Estimado: 15.54286

Valor Real: 23.100 Valor Estimado: 10.83162
Valor Real: 50.000 Valor Estimado: 23.80073
Valor Real: 50.000 Valor Estimado: 32.64407
Valor Real: 50.000 Valor Estimado: 34.60684
Valor Real: 50.000 Valor Estimado: 24.94331
Valor Real: 50.000 Valor Estimado: 25.99981
Valor Real: 13.800 Valor Estimado: 6.12632
Valor Real: 13.800 Valor Estimado: 0.77780
Valor Real: 15.000 Valor Estimado: 25.30713
Valor Real: 13.900 Valor Estimado: 17.74061
Valor Real: 13.300 Valor Estimado: 20.23274
Valor Real: 13.100 Valor Estimado: 15.83331
Valor Real: 10.200 Valor Estimado: 16.83513
Valor Real: 10.400 Valor Estimado: 14.36995
Valor Real: 10.900 Valor Estimado: 18.47683
Valor Real: 11.300 Valor Estimado: 13.42768
Valor Real: 12.300 Valor Estimado: 13.06178
Valor Real: 8.800 Valor Estimado: 3.27918
Valor Real: 7.200 Valor Estimado: 8.06022
Valor Real: 10.500 Valor Estimado: 6.12842
Valor Real: 7.400 Valor Estimado: 5.61865
Valor Real: 10.200 Valor Estimado: 6.45199
Valor Real: 11.500 Valor Estimado: 14.20765
Valor Real: 15.100 Valor Estimado: 17.21225
Valor Real: 23.200 Valor Estimado: 17.29887
Valor Real: 9.700 Valor Estimado: 9.89117
Valor Real: 13.800 Valor Estimado: 20.22124
Valor Real: 12.700 Valor Estimado: 17.94181
Valor Real: 13.100 Valor Estimado: 20.30446
Valor Real: 12.500 Valor Estimado: 19.29559
Valor Real: 8.500 Valor Estimado: 16.33633
Valor Real: 5.000 Valor Estimado: 6.55162
Valor Real: 6.300 Valor Estimado: 10.89017
Valor Real: 5.600 Valor Estimado: 11.88146
Valor Real: 7.200 Valor Estimado: 17.81175
Valor Real: 12.100 Valor Estimado: 18.26127
Valor Real: 8.300 Valor Estimado: 12.97949
Valor Real: 8.500 Valor Estimado: 7.37816
Valor Real: 5.000 Valor Estimado: 8.21116
Valor Real: 11.900 Valor Estimado: 8.06626
Valor Real: 27.900 Valor Estimado: 19.98295
Valor Real: 17.200 Valor Estimado: 13.70756
Valor Real: 27.500 Valor Estimado: 19.85268
Valor Real: 15.000 Valor Estimado: 15.22308
Valor Real: 17.200 Valor Estimado: 16.96072
Valor Real: 17.900 Valor Estimado: 1.71852
Valor Real: 16.300 Valor Estimado: 11.80578
Valor Real: 7.000 Valor Estimado: -4.28131
Valor Real: 7.200 Valor Estimado: 9.58377
Valor Real: 7.500 Valor Estimado: 13.36661
Valor Real: 10.400 Valor Estimado: 6.89562
Valor Real: 8.800 Valor Estimado: 6.14780
Valor Real: 8.400 Valor Estimado: 14.60662
Valor Real: 16.700 Valor Estimado: 19.60003
Valor Real: 14.200 Valor Estimado: 18.12427
Valor Real: 20.800 Valor Estimado: 18.52177
Valor Real: 13.400 Valor Estimado: 13.17529
Valor Real: 11.700 Valor Estimado: 14.62618
Valor Real: 8.300 Valor Estimado: 9.92375
Valor Real: 10.200 Valor Estimado: 16.34591
Valor Real: 10.900 Valor Estimado: 14.07519
Valor Real: 11.000 Valor Estimado: 14.25756
Valor Real: 9.500 Valor Estimado: 13.04235
Valor Real: 14.500 Valor Estimado: 18.15956
Valor Real: 14.100 Valor Estimado: 18.69554
Valor Real: 16.100 Valor Estimado: 21.52728
Valor Real: 14.300 Valor Estimado: 17.03142
Valor Real: 11.700 Valor Estimado: 15.96090
Valor Real: 13.400 Valor Estimado: 13.36142
Valor Real: 9.600 Valor Estimado: 14.52079
Valor Real: 8.700 Valor Estimado: 8.81976
Valor Real: 8.400 Valor Estimado: 4.86751
Valor Real: 12.800 Valor Estimado: 13.06591
Valor Real: 10.500 Valor Estimado: 12.70610
Valor Real: 17.100 Valor Estimado: 17.29558
Valor Real: 18.400 Valor Estimado: 18.74049
Valor Real: 15.400 Valor Estimado: 18.05901

```
Valor Real: 10.800 Valor Estimado: 11.51475
Valor Real: 11.800 Valor Estimado: 11.97400
Valor Real: 14.900 Valor Estimado: 17.68345
Valor Real: 12.600 Valor Estimado: 18.12695
Valor Real: 14.100 Valor Estimado: 17.51835
Valor Real: 13.000 Valor Estimado: 17.22743
Valor Real: 13.400 Valor Estimado: 16.52272
Valor Real: 15.200 Valor Estimado: 19.41291
Valor Real: 16.100 Valor Estimado: 18.58215
Valor Real: 17.800 Valor Estimado: 22.48945
Valor Real: 14.900 Valor Estimado: 15.28000
Valor Real: 14.100 Valor Estimado: 15.82089
Valor Real: 12.700 Valor Estimado: 12.68726
Valor Real: 13.500 Valor Estimado: 12.87634
Valor Real: 14.900 Valor Estimado: 17.18669
Valor Real: 20.000 Valor Estimado: 18.51248
Valor Real: 16.400 Valor Estimado: 19.04861
Valor Real: 17.700 Valor Estimado: 20.17209
Valor Real: 19.500 Valor Estimado: 19.77407
Valor Real: 20.200 Valor Estimado: 22.42941
Valor Real: 21.400 Valor Estimado: 20.31912
Valor Real: 19.900 Valor Estimado: 17.88616
Valor Real: 19.000 Valor Estimado: 14.37479
Valor Real: 19.100 Valor Estimado: 16.94777
Valor Real: 19.100 Valor Estimado: 16.98406
Valor Real: 20.100 Valor Estimado: 18.58838
Valor Real: 19.900 Valor Estimado: 20.16719
Valor Real: 19.600 Valor Estimado: 22.97718
Valor Real: 23.200 Valor Estimado: 22.45581
Valor Real: 29.800 Valor Estimado: 25.57825
Valor Real: 13.800 Valor Estimado: 16.39148
Valor Real: 13.300 Valor Estimado: 16.11146
Valor Real: 16.700 Valor Estimado: 20.53482
Valor Real: 12.000 Valor Estimado: 11.54273
Valor Real: 14.600 Valor Estimado: 19.20496
Valor Real: 21.400 Valor Estimado: 21.86276
Valor Real: 23.000 Valor Estimado: 23.46879
Valor Real: 23.700 Valor Estimado: 27.09887
Valor Real: 25.000 Valor Estimado: 28.56994
Valor Real: 21.800 Valor Estimado: 21.08399
Valor Real: 20.600 Valor Estimado: 19.45516
Valor Real: 21.200 Valor Estimado: 22.22226
Valor Real: 19.100 Valor Estimado: 19.65592
Valor Real: 20.600 Valor Estimado: 21.32536
Valor Real: 15.200 Valor Estimado: 11.85584
Valor Real: 7.000 Valor Estimado: 8.22387
Valor Real: 8.100 Valor Estimado: 3.66400
Valor Real: 13.600 Valor Estimado: 13.75909
Valor Real: 20.100 Valor Estimado: 15.93119
Valor Real: 21.800 Valor Estimado: 20.62662
Valor Real: 24.500 Valor Estimado: 20.61249
Valor Real: 23.100 Valor Estimado: 16.88542
Valor Real: 19.700 Valor Estimado: 14.01321
Valor Real: 18.300 Valor Estimado: 19.10854
Valor Real: 21.200 Valor Estimado: 21.29805
Valor Real: 17.500 Valor Estimado: 18.45499
Valor Real: 16.800 Valor Estimado: 20.46871
Valor Real: 22.400 Valor Estimado: 23.53334
Valor Real: 20.600 Valor Estimado: 22.37572
Valor Real: 23.900 Valor Estimado: 27.62743
Valor Real: 22.000 Valor Estimado: 26.12797
Valor Real: 11.900 Valor Estimado: 22.34421
```

SkLearn y Pandas

En la parte anterior hemos trabajado los conjuntos de datos del modelo con Datasets. En este apartado trabajaremos con DataFrames de **Pandas** como estructura principal para el conjunto de datos.

In [19]:

```
import pandas as pd
```

In [20]:

```
df = pd.DataFrame(var_independiente, columns=nombre_var_independiente)
df.head()
```

```
ur.meda()
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [21]:

```
# aqui en el dataset no tenemos el target o valor mediano, asi
# que lo introducimos a partir de la variable objetivo
df['MEDV'] = var_objetivo
df.head()
```

Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [22]:

```
model.fit(X=df[nombre_var_independiente], y=df['MEDV'])
```

Out[22]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

In [23]:

```
# Ahora añadiremos al dataframe una columna para la prediccion
df['MEDV_pred'] = model.predict(df[nombre_var_independiente])
df.head(5)
```

Out[23]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	MEDV_pred
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	30.003843
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	25.025562
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	30.567597
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	28.607036
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	27.943524

In [24]:

```
# vamos a obtener el modelo pero esta vez vamos a normalizarlo
modelo_normalizado = LinearRegression(normalize=True)
# Ajustamos el modelo normalizado
modelo_normalizado.fit(X=df[nombre_var_independiente], y=df['MEDV'])
# anadimos ademas una columna para la prediccion del modelo
# normalizado.
df['MEDV_pred_norm'] = modelo_normalizado.predict(df[nombre_var_independiente])
```

In [25]:

```
# visualizamos los cambios, aunque en un dataset tan pequeño no
# veremos cambios.
```

```
df.head(5)
```

```
Out[25]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	MEDV_pred	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0	30.003843	30.003843
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6	25.025562	25.025562
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	30.567597	30.567597
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	28.607036	28.607036
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2	27.943524	27.943524

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-30T21:30:27+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Evaluacion de modelos de regresion

```
In [2]:
```

```
from sklearn import datasets
```

Cargamos el dataset. Para este ejemplo utilizaremos el dataset de viviendas de Boston (Boston Housing Dataset)

```
In [3]:
```

```
boston = datasets.load_boston()
```

Creamos un modelo de regresión lineal

```
In [4]:
```

```
from sklearn.linear_model import LinearRegression
```

```
In [5]:
```

```
model = LinearRegression()
```

Con el modelo de regresión lineal instanciado ajustamos el modelo teniendo como variable independiente la columna "data" del dataset y como variable dependiente los valores de la columna "target"

```
In [6]:
```

```
model.fit(X=boston["data"], y=boston["target"])  
y_pred = model.predict(boston["data"])
```

```
In [7]:
```

```
y_objetivo = boston["target"]  
y_pred = model.predict(boston["data"])
```

```
In [8]:
```

```
from sklearn import metrics
```

Error Absoluto medio

El Error absoluto medio (Mean Absolute Error o MAE) se define como:

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Es decir, la media de las diferencias entre la variables objeto y las predicciones sin el signo. MAE es una métrica robusta, en cuanto a que no varia mucho si hay valores extremos en los datos. El error se puede interpretar como unidades de la variable objetivo (por ejemplo, si la variable objetivo es en dólares MAE estará tambien en dólares).

```
In [9]:
```

```
metrics.mean_absolute_error?
```

In [10]:

```
metrics.mean_absolute_error(y_objetivo, y_pred)
```

Out[10]:

```
3.2708628109003137
```

Error cuadrático medio

El Error cuadrático medio (Mean Squared Error o MSE)

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Dado que el MSE se define en unidades al cuadrado, lo cual no es intuitivo, generalmente se usa su raíz.

Raíz del error cuadrático medio

La raíz del error cuadrático medio (Root Mean Squared o RMSE) se diferencia del MSE en que el resultado se medir en las mismas unidades que la variable objetivo.

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Sin embargo, tiene un problema y es que da más importancia a los errores grandes. Por ejemplo en el Boston Housing DataSet, si tenemos dos observaciones y sus predicciones:

```
observacion1: MEDV: 10 MEDV_pred: 15 RMSE: (10-15)^2=25  
observacion2: MEDV: 1000 MEDV_pred: 1010 RMSE: (1000-1010)^2=100
```

El usar RMSE como medida de error significa que se le dará más peso de la observacion2 que al de la observacion1, pese a que un error de 5.000 dólares en una zona donde el valor media es 10.000 dólares es un error mucho más grave que un error de 10.000 dólares en una zona donde el valor medio de las casas es de 1 millón de dólares.

RMSE es más sensible que MAE a variaciones en los errores de predicción aquí tenemos un ejemplo donde se ve esto:

En este ejemplo se ve como para los tres casos, el MAE no varía, mientras que en función de la distribución de errores el RMSE puede ser igual al MAE, (todos los errores iguales) o mucho mayor (muchos errores nulos con un error muy grande)

Para aquellos casos donde lo que queremos es esto (evitar errores grandes) y cuando la distribución de la variable objetivo esté bien distribuida se puede usar RMSE en vez de MAE.

```
In [11]:  
import numpy as np
```

Se calcula haciendo la función sqrt

```
In [12]:  
np.sqrt(metrics.mean_squared_error(y_objetivo, y_pred))  
Out[12]:  
4.679191295697281
```

Coeficiente de determinación

El coeficiente de determinación (R2 o R-squared)(Se usa sobre todo en regresión lineal) mide la porción de la varianza de la variable objetivo que se puede explicar por el modelo.

R2 tiene un valor máximo de 1 (cuando el modelo explica toda la varianza), aunque puede tener valores negativos.

Hay **varias formas de definir R2**, pero una de las más sencillas es simplemente la correlación (definida como la **Correlación de Pearson**) entre la variable objetivo y las predicciones, elevada al cuadrado.

Un problema importante que tiene R2 es que no nos indica si el modelo explica la varianza debido a que está sobreajustado

Un problema importante que tiene R^2 es que no nos indica si el modelo explica la variación observada que es la **sobreajustado** consideración la complejidad del modelo.

$$\$ \$ 1 - \frac{(1 - R^2)(n-1)}{(n-k-1)} \$ \$$$

donde n es el número de observaciones y k es el número de coeficientes del modelo (sin contar el término independiente)

In [13]:

```
model_r2 = metrics.r2_score(y_objetivo,y_pred)
model_r2
```

Out[13]:

```
0.7406426641094095
```

In [14]:

```
import numpy
```

In [15]:

```
np.corrcoef(y_objetivo, y_pred)**2
```

Out[15]:

```
array([[1.          , 0.74064266],
       [0.74064266, 1.          ]])
```

In [16]:

```
len(model.coef_)
```

Out[16]:

```
13
```

In [17]:

```
model_r2_ajustado = 1 - (1-model_r2)*(len(boston["target"])-1)/(len(boston["target"])-boston["data"].shape[1]-1)
model_r2_ajustado
```

Out[17]:

```
0.733789726372463
```

```
In [1]:
```

```
%load_ext watermark  
%watermark  
%matplotlib inline
```

```
2019-05-30T21:31:08+02:00
```

```
CPython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

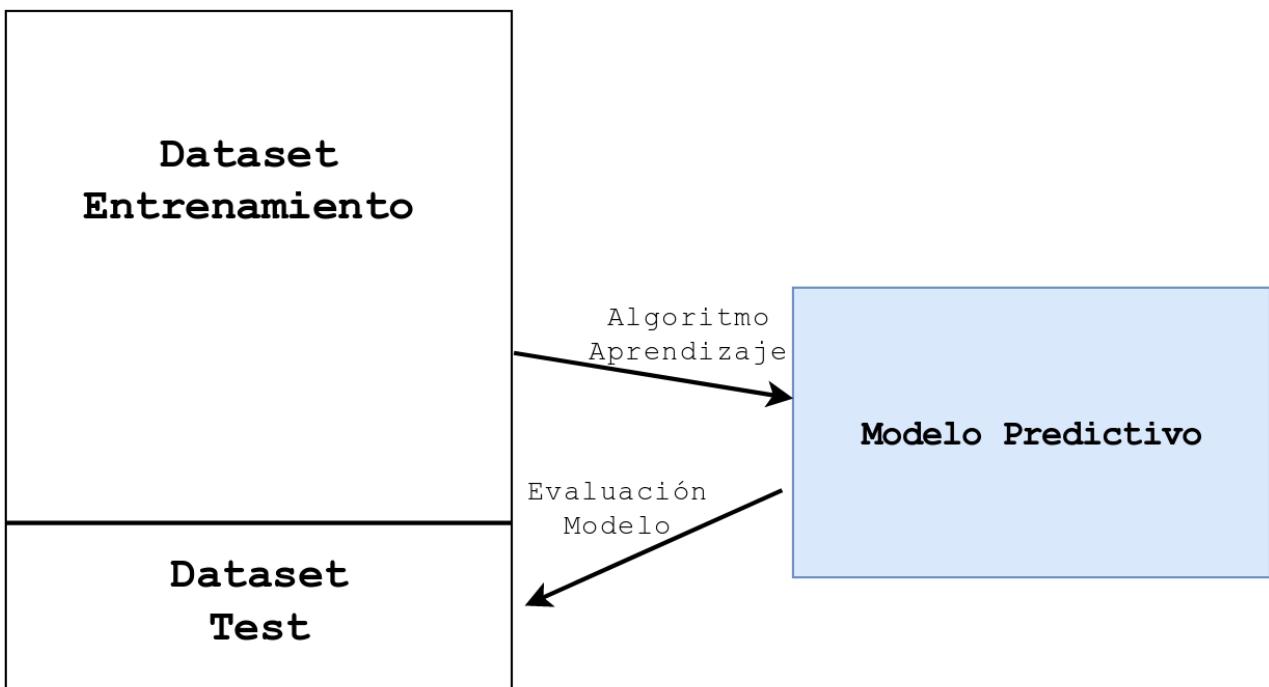
Separacion de datos entrenamiento, test y validacion cruzada

Separamos nuestro dataset en 2 partes el dataset de entrenamiento y el dataset de test, basicamente quitamos un trozo del dataset original, entrenamos a la máquina con el resto del dataset, hacemos el modelo predictivo y evaluamos el modelo con el trozo que quitamos. Evitamos así el sobreajuste, no queremos que memorice el modelo, sino que generalice

```
In [2]:
```

```
from IPython.display import Image  
Image("../RESOURCES/train_test_split.jpg")
```

```
Out[2]:
```



```
In [3]:
```

```
import numpy as np  
  
from sklearn.linear_model import LinearRegression  
from sklearn import metrics  
from sklearn import datasets
```

```
In [4]:
```

```
boston = datasets.load_boston()
```

```
In [5]:
# Error cuadrático medio
def root_mean_square_error(objetivo, estimaciones):
    return np.sqrt(metrics.mean_squared_error(objetivo, estimaciones))
# Ajuste al r2
def adjusted_r2(objetivo, estimaciones, n, k):
    r2 = metrics.r2_score(objetivo, estimaciones)
    return 1 - (1-r2)*(n-1) / (n - k - 1)
# Evaluación del modelo
def evaluar_modelo(objetivo, estimaciones, n, k):
    return {
        "rmse": root_mean_square_error(objetivo, estimaciones),
        "mae": metrics.mean_absolute_error(objetivo, estimaciones),
        "adjusted_r2": adjusted_r2(objetivo, estimaciones, n, k)
    }
```

In [6]:

```
modelo_ols = LinearRegression()

modelo_ols.fit(X=boston["data"], y=boston["target"])

modelo_ols_preds = modelo_ols.predict(boston["data"])
```

In [7]:

```
RESULTADOS = {}
```

In [8]:

```
# tamaño de la muestra
N = boston["data"].shape[0]

RESULTADOS["ols"] = evaluar_modelo(boston["target"],
                                    modelo_ols_preds,
                                    N,
                                    len(modelo_ols.coef_))

RESULTADOS
```

Out [8]:

```
{'ols': {'rmse': 4.679191295697281,
         'mae': 3.2708628109003137,
         'adjusted_r2': 0.733789726372463}}
```

Esto no tiene sentido porque estamos haciendo un sobreajuste por defecto y en el entrenamiento le damos todo los datos y es probable que no genere análisis bien con datos nuevos que no tenemos ahora mismo (autosampler)

Vamos a separar los datos de entrenamiento de los de test

In [9]:

```
from sklearn.model_selection import train_test_split
```

In [10]:

```
train_test_split?
```

In [11]:

```
boston["data"].shape
```

Out [11]:

```
(506, 13)
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(
    boston["data"],
    boston["target"],
    test_size=0.33,
    random_state=13
)
```

sacamos un tercio para el test

casamiento en circulo para el test

In [13]:

```
print(X_train.shape, y_train.shape)
(339, 13) (339,)
```

In [14]:

```
print(X_test.shape, y_test.shape)
(167, 13) (167,)
```

Entrenamos al modelo con los datos de entrenamiento

In [15]:

```
modelo_ols = LinearRegression()
modelo_ols.fit(X=X_train, y=y_train)
modelo_ols_train_preds = modelo_ols.predict(X_train)
```

In [16]:

```
RESULTADOS["ols_train"] = evaluar_modelo(
    y_train,
    modelo_ols_train_preds,
    X_train.shape[0],
    len(modelo_ols.coef_)
)
```

Ahora vamos a predecir usando el modelo entrenado con los datos de test

Y evaluamos el modelo con los datos de test

In [17]:

```
modelo_ols_test_preds = modelo_ols.predict(X_test)

RESULTADOS["ols_test"] = evaluar_modelo(
    y_test,
    modelo_ols_test_preds,
    X_test.shape[0],
    len(modelo_ols.coef_)
)
```

In [18]:

```
import pandas as pd
```

In [19]:

```
pd.DataFrame(RESULTADOS)
```

Out[19]:

	ols	ols_train	ols_test
adjusted_r2	0.733790	0.731491	0.688716
mae	3.270863	3.300868	3.558434
rmse	4.679191	4.721732	4.784178

Vemos que al separar los datos de entrenamiento y los que test se obtiene un resultado peor al evaluar los datos de test.

Podríamos parar aquí y decir "El error RMSW de mi modelo es 4.787026", podríamos pensar que esta todo bien ya que no hemos entrenados el modelo en los que hemos usado para evaluarlo.

Pero estaríamos en un grave error. ¿Por qué?

Recordad que hemos un `random_state=13` para la función `train_test_split` que garantiza que la separación de entrenamiento y test sea siempre la misma. Podemos usar cualquier número para este argumento.

Qué pasa si usamos por ejemplo `random_state=42`?

In [20]:

```
RESULTADOS = {}

X_train, X_test, y_train, y_test = train_test_split(
    boston["data"],
    boston["target"],
    test_size=0.33,
    random_state=42
)

modelo_ols = LinearRegression()
modelo_ols.fit(X=X_train, y=y_train)
modelo_ols_train_preds = modelo_ols.predict(X_train)

RESULTADOS["ols_train"] = evaluar_modelo(
    y_train,
    modelo_ols_train_preds,
    X_train.shape[0],
    len(modelo_ols.coef_)
)

modelo_ols_test_preds = modelo_ols.predict(X_test)

RESULTADOS["ols_test"] = evaluar_modelo(
    y_test,
    modelo_ols_test_preds,
    X_test.shape[0],
    len(modelo_ols.coef_)
)

pd.DataFrame(RESULTADOS)
```

Out[20]:

	ols_train	ols_test
adjusted_r2	0.728804	0.702889
mae	3.376419	3.148256
rmse	4.794269	4.552365

El error en los datos de test es menor que en los entrenamiento ¿Por qué? Sencillamente, por que ha dado la casualidad de que hemos separado los datos de una forma que los datos de test son muy fáciles de estimar.

In [21]:

```
model = LinearRegression()
results = []
def test_seed(seed):
    X_train, X_test, y_train, y_test = train_test_split(
        boston["data"], boston["target"],
        test_size=0.33, random_state=seed
    )
    test_preds = model.fit(X_train, y_train).predict(X_test)
    seed_rmse = root_mean_square_error(y_test, test_preds)
    results.append([seed_rmse, seed])
```

In [22]:

```
for i in range(1000):
    test_seed(i)
```

In [23]:

```
results_sorted = sorted(results, key=lambda x: x[0], reverse=False)
```

In [24]:

```
results_sorted[0]
```

Out[24]:

```
[3.631314217076931, 635]
```

```
In [25]:
```

```
results_sorted[-1]
```

```
Out[25]:
```

```
[6.788054714003132, 645]
```

conclusion.... Estos significa que no podemos hacer solo una particion de los datos, eso es jugartela demasiado.

Validación Cruzada

Vemos que entre la semilla con menor error de test y la semilla con mayor error hay una diferencia del doble

Una forma de evitar el cometer este error es mediante la **Validación cruzada**

```
In [26]:
```

```
from sklearn.model_selection import cross_val_score
```

```
In [27]:
```

```
cross_val_score?
```

```
In [28]:
```

```
modelo_ols = LinearRegression()
X = boston["data"]
y = boston["target"]

resultados_validacion_cruzada = cross_val_score(
    estimator=modelo_ols,
    X=X,
    Y=Y,
    scoring = "neg_mean_squared_error",
    cv=10
)
```

```
In [29]:
```

```
resultados_validacion_cruzada
```

```
Out[29]:
```

```
array([-9.28694671, -14.15128316, -14.07360615, -35.20692433,
       -31.88511666, -19.83587796, -9.94726918, -168.37537954,
       -33.32974507, -10.96041068])
```

```
In [30]:
```

```
def rmse_cross_validation(estimator, X, y):
    y_pred = estimator.predict(X)
    return np.sqrt(metrics.mean_absolute_error(y, y_pred))
```

```
In [37]:
```

```
resultados_cv = []
for i in range(10,200):
    cv_rmse = cross_val_score(
        estimator=modelo_ols,
        X=X,
        Y=Y,
        scoring=rmse_cross_validation,
        cv=i
    ).mean()
    resultados_cv.append(cv_rmse)
```

```
In [38]:
```

```
import matplotlib.pyplot as plt
```

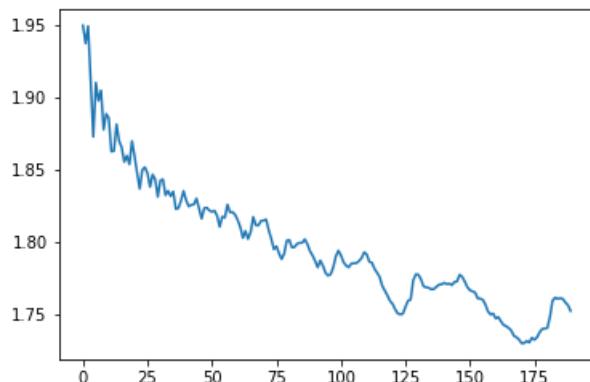
```
In [39]:
```

```
plt.plot(resultados_cv)
```

```
In [38]: plt.plot(residuals_cv)
```

Out[39]:

```
[<matplotlib.lines.Line2D at 0x7f38824a8f28>]
```



In [40]:

```
from sklearn.model_selection import cross_val_score
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_validation}
estimator = modelo_ols
scores = cross_val_score(estimator,
                         boston["data"],
                         boston["target"],
                         scoring=scoring,
                         cv=10,
                         return_train_score=True)
```

In [41]:

```
cross_val_score?
```

In [42]:

```
pd.DataFrame(scores).mean()
```

Out[42]:

```
fit_time      0.001144
score_time    0.000811
test_mae     -4.004947
train_mae    -3.247644
test_rmse     1.949203
train_rmse    1.800855
dtype: float64
```

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-30T21:31:53+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Regularización.

La regularización es un método usado ampliamente para tratar el sobreajuste (overfitting) de los modelos. Se realiza principalmente con las siguientes técnicas:

1. Reducir el tamaño del modelo: a través de la reducción del número de parámetros de aprendizaje del modelo, y con ello su capacidad de aprendizaje. El objetivo es llegar a un punto de equilibrio entre una capacidad de aprendizaje excesiva y una capacidad de aprendizaje insuficiente. Desafortunadamente, no hay fórmulas mágicas para determinar este equilibrio, debe ser probado y evaluado estableciendo un número diferente de parámetros y observando su rendimiento.
2. Añadir la regularización de pesos: En general, cuanto más sencillo es un modelo, mejor es. Mientras pueda “aprender bien”, un modelo simple es menos susceptible de caer en el sobreajuste. Una forma normal de conseguir esto es restringir la complejidad de la red forzando a sus pesos a tomar únicamente valores pequeños. Esto se realiza añadiendo a la función de pérdida de la red “loss function” un coste asociado al uso de grandes pesos. El coste se puede obtener de dos formas:
 - Regularización L1: El coste es proporcional al valor absoluto de los coeficientes de peso (norma del peso L1).
 - Regularización L2: El coste es proporcional al cuadrado de los coeficientes de peso (norma de peso L2).

Para decidir cuál de las dos aplicar a nuestro modelo, se recomienda tener en cuenta la siguiente información en relación con la naturaleza de nuestro problema:

```
In [2]:
```

```
import numpy as np  
import pandas as pd  
from sklearn import metrics  
from sklearn.model_selection import cross_validate
```

```
In [3]:
```

```
#importamos un dataset de vehiculos  
vehiculos = pd.read_csv("../..//RESOURCES/vehiculos_procesado.csv")  
datos_entrenamiento = vehiculos[["desplazamiento","cilindros","consumo"]]  
  
objetivo = vehiculos["co2"]  
datos_entrenamiento.head()
```

```
Out [3]:
```

	desplazamiento	cilindros	consumo
0	2.5	4.0	17
1	4.2	6.0	13
2	2.5	4.0	16
3	4.2	6.0	13
4	3.8	6.0	16

```
In [4]:
```

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
```

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
```

In [5]:

```
modelo_ols = LinearRegression()
modelo_ols.fit(datos_entrenamiento, objetivo)
modelo_ols.coef_
```

Out[5]:

```
array([ 11.76787991,    1.23791071, -19.80355606])
```

In [6]:

```
"""Mediremos la complejidad del modelo con la norma L1 y la norma L2"""
def norma_l1(coeficientes):
    return np.abs(coeficientes).sum()

def norma_l2(coeficientes):
    return np.sqrt(np.power(coeficientes, 2).sum())

"""Podemos utilizar también los métodos de numpy
que proporciona para estas normas"""
def norma_l1(coeficientes):
    return np.linalg.norm(coeficientes, ord=1)

def norma_l2(coeficientes):
    return np.linalg.norm(coeficientes, ord=2)

print(norma_l1(modelo_ols.coef_))
print(norma_l2(modelo_ols.coef_))
```

```
32.80934668020822
23.069379124497573
```

Veremos como regularizar la complejidad de los modelos en cuanto a número de parámetros

In [7]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [8]:

```
transformador_polinomial = PolynomialFeatures(5)
```

In [9]:

```
transformador_polinomial.fit(datos_entrenamiento)
```

Out[9]:

```
PolynomialFeatures(degree=5, include_bias=True, interaction_only=False)
```

In [10]:

```
#igual que utilizábamos el método predict utilizamos el método transform
variables_polinomiales=transformador_polinomial.transform(
    datos_entrenamiento)
```

In [11]:

```
variables_polinomiales.shape
```

Out[11]:

```
(35539, 56)
```

In [12]:

```
datos_entrenamiento.loc[0]
```

Out[12]:

```
desplazamiento      2.5
cilindros          4.0
consumo            17.0
Name: 0, dtype: float64
```

```
In [13]:
```

```
variables_polinomiales[0]
```

```
Out[13]:
```

```
array([1.000000e+00, 2.500000e+00, 4.000000e+00, 1.700000e+01,
       6.250000e+00, 1.000000e+01, 4.250000e+01, 1.600000e+01,
       6.800000e+01, 2.890000e+02, 1.562500e+01, 2.500000e+01,
       1.062500e+02, 4.000000e+01, 1.700000e+02, 7.225000e+02,
       6.400000e+01, 2.720000e+02, 1.156000e+03, 4.913000e+03,
       3.906250e+01, 6.250000e+01, 2.656250e+02, 1.000000e+02,
       4.250000e+02, 1.806250e+03, 1.600000e+02, 6.800000e+02,
       2.890000e+03, 1.228250e+04, 2.560000e+02, 1.088000e+03,
       4.624000e+03, 1.965200e+04, 8.352100e+04, 9.765625e+01,
       1.562500e+02, 6.640625e+02, 2.500000e+02, 1.062500e+03,
       4.515625e+03, 4.000000e+02, 1.700000e+03, 7.225000e+03,
       3.070625e+04, 6.400000e+02, 2.720000e+03, 1.156000e+04,
       4.913000e+04, 2.088025e+05, 1.024000e+03, 4.352000e+03,
       1.849600e+04, 7.860800e+04, 3.340840e+05, 1.419857e+06])
```

```
In [14]:
```

```
#Estos últimos pasos se suelen agrupar por convenio
```

```
variables_polinomiales = PolynomialFeatures(5).fit_transform(datos_entrenamiento)
```

```
In [15]:
```

```
variables_polinomiales.shape
```

```
Out[15]:
```

```
(35539, 56)
```

Modelo OLS con variables polinomiales

Comenzamos realizando una regresion lineal con todas esas variables sin ningun tipo de regularizacion

```
In [16]:
```

```
RESULTADOS = {}

modelo_ols = LinearRegression()
modelo_ols.fit(variables_polinomiales, objetivo)
print(modelo_ols.coef_)
RESULTADOS["ols"] = {
    "norma_11": norma_11(modelo_ols.coef_),
    "norma_12": norma_12(modelo_ols.coef_)
}

[-1.33581691e-04 -2.16202007e+03 -4.74743025e+02 -9.33379425e+02
 2.06179368e+03 -1.98652141e+03 4.07183956e+02 6.98188922e+02
 -8.60206830e+00 5.68565581e+01 -5.04043554e+02 5.97572364e+02
 -2.39915727e+02 -1.93155159e+02 2.06936511e+02 -2.46366997e+01
 -7.53120039e+00 -6.16587972e+01 2.77167162e+00 -1.72213964e+00
 3.22404684e+01 -2.32860787e+01 3.67724866e+01 -1.28564531e+01
 -4.24157145e+01 9.08423002e+00 1.17166991e+01 1.52755499e+01
 -7.40121867e+00 6.04111957e-01 -9.27527797e-01 -8.19881989e-01
 1.96304324e+00 -8.55528671e-02 2.44388052e-02 -2.44193545e+00
 6.17851513e+00 -1.41371913e+00 -7.50158521e+00 1.59153882e+00
 -6.92569799e-01 5.18941968e+00 -3.95368960e-01 8.24178212e-01
 -1.15475754e-01 -1.85224537e+00 -2.10933021e-02 -3.68950001e-01
 9.99115993e-02 -5.66794738e-03 2.41619836e-01 -2.05795973e-02
 5.80537284e-02 -2.77634650e-02 1.11091113e-03 -1.26148048e-04]
```

Modelo Regularizacion L1 (de Lasso) con variables polinomiales

Como podemos ver en la documentacion, la clase lasso admite un parametro alpha (coeficiente de regularizacion) que indica la complejidad del modelo, donde 0 seria una regresion lineal normal y 1 regulariza al maximo. Tambien tenemos que tener en cuenta el *max_iter* que seria el maximo de iteraciones que necesita para considerar que ya ha aprendido suficiente. ademas tiene un parametro *tol* que indica la tolerancia.

```
In [17]:
```

```
Lasso?
```

In [18]:

```
modelo_11 = Lasso(alpha=1.0,tol=0.01,max_iter=5000)
modelo_11.fit(variables_polinomiales,objetivo)
print(modelo_11.coef_)

RESULTADOS["regularizacion_11"]={
    "norma_11":norma_11(modelo_11.coef_),
    "norma_12":norma_12(modelo_11.coef_)
}

[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -3.30237023e+01
 5.12409278e+00  1.44708364e+00 -1.47969971e+00  6.52491862e-01
-5.04276126e-01  5.26431228e-02 -1.48062992e-01 -0.00000000e+00
-1.32137016e-01  1.02289253e-01 -9.17626300e-02 -1.30192905e-02
 6.15573125e-02 -3.01413839e-02  2.31053922e-02  5.17315962e-03
 1.91792946e-02 -1.51220456e-03 -1.88077866e-03 -1.09782669e-03
-6.50140180e-03 -2.91026927e-04  1.73571147e-03 -3.33999083e-03
-1.64104930e-03  1.59784747e-03  1.31929178e-03  1.69629743e-04
-1.64562586e-03  2.03369130e-04  1.99415033e-05 -2.81072727e-03
 7.11467485e-04  2.46531750e-03 -2.20515928e-03  2.38235311e-04
 6.38712931e-04 -1.09015174e-03 -3.89575283e-04 -8.09552788e-05
 2.70933679e-04 -1.56154368e-04 -1.88340282e-04 -2.27097112e-04
 3.81560605e-05  2.29005451e-06 -5.56615246e-05  1.44114807e-04
 4.13806087e-06  5.51970343e-05  7.62940670e-07 -1.14293565e-06]
```

observamos que la regularizacion ha convertido muchos de los parametros originales a 0.

Modelo Regularizacion L2 (Ridge) con variables polinomiales

Aunque la regularizacion l2 va mas rapido ponemos el mismo numero de iteraciones maximas y la misma tolerancia

In [19]:

```
modelo_12 = Ridge(alpha=1.0,tol=0.01,max_iter=5000)
modelo_12.fit(variables_polinomiales,objetivo)

print(modelo_12.coef_)

RESULTADOS["regularizacion_12"]={
    "norma_11":norma_11(modelo_12.coef_),
    "norma_12":norma_12(modelo_12.coef_)
}

[ 0.00000000e+00  1.51700145e+01  1.42697225e+00 -9.87021958e+00
 4.84917097e+01  3.40056811e+01  4.08617962e+01  1.87320285e+01
-3.38485904e+01 -4.32030250e+00 -2.38651378e+01  2.49480682e+01
-1.68754953e+01 -1.28017569e+01 -6.76114933e+00 -2.53957421e+00
-4.79464960e+00  6.20674303e+00  2.06740524e+00  2.74904588e-01
 1.539696032e+01 -3.67044578e+01  3.58929869e+00  3.45188723e+01
-4.38964155e+00  9.62648142e-01 -1.53176933e+01  4.03068321e+00
-3.81950229e-02  4.61124419e-02  3.15167406e+00 -1.12387047e+00
-2.48505114e-01 -3.05623638e-02 -7.73541022e-03 -2.33673851e+00
 6.80494202e+00 -8.69748260e-01 -7.95078796e+00  2.20022716e+00
-1.37697480e-01  4.60214517e+00 -2.17510167e+00  2.25462479e-01
-1.86520095e-02 -1.28785706e+00  9.40202486e-01 -1.93271866e-01
 1.34228047e-02 -4.68455498e-04  1.24956055e-01 -1.58111335e-01
 5.90665314e-02 -2.26320625e-03  2.49827232e-04  8.13219567e-05]
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125: LinAlgWarning:
scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number 4.797447e-19
  overwrite_a=True).T
```

Modelo Regularizacion ElasticNet con variables polinomiales

Tiene un parametro l1_ratio que es "elastico" donde 1 sera regularizacion l1 y si pones 0 sera regularizacion l2. En nuestro caso pondremos el mismo peso entre regularizacion de lasso y de ridge

In [20]:

```
modelo_elasticnet = ElasticNet(l1_ratio=0.5,tol=0.01,max_iter=5000)
modelo_elasticnet.fit(variables_polinomiales,objetivo)

print(modelo_elasticnet.coef_)
```

```

print(modelo_elasticnet.coef_)

RESULTADOS["regularizacion_elasticnet"] = {
    "norma_l1": norma_l1(modelo_elasticnet.coef_),
    "norma_l2": norma_l2(modelo_elasticnet.coef_)
}

[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.33310440e+00
 4.89645007e+00  6.09439149e+00 -1.31816563e+00  2.37723778e+00
-3.10050434e+00 -6.41934303e-01 -7.84374220e-01 -9.59081564e-02
-7.18954629e-02  7.19478059e-03 -4.57252233e-02 -6.37787774e-02
 1.22426563e-02  3.46550752e-02  4.26955780e-02  2.66673958e-03
 1.39812144e-02 -8.09891082e-03  8.76409316e-03 -1.69690074e-02
-2.05177215e-03  1.54020194e-03 -6.29758134e-03 -2.92855315e-03
-6.86058449e-04  1.42133650e-03 -9.92879254e-04  1.28704557e-04
 5.03220762e-04  6.61024458e-04  5.80200778e-05  3.41020839e-03
 2.27563082e-03  2.71125911e-03 -1.90350591e-03 -7.82250851e-05
 6.56665428e-04 -7.34759017e-04 -9.12167373e-04 -2.43120629e-04
 1.60164558e-04  3.89147755e-04 -5.45731423e-04 -3.49391006e-04
 3.45240438e-05  3.46406797e-05  2.59823852e-04 -6.35175029e-05
-9.14762151e-05  5.08095098e-05  1.14384625e-05  7.36969809e-07]

```

Visualización de los Resultados

In [21]:

```
pd.set_option("display.float_format", lambda x: str(round(x, 6)))
```

In [22]:

```

resultados_df = pd.DataFrame(RESULTADOS).T
l1_ols = resultados_df.loc["ols", "norma_l1"]
l2_ols = resultados_df.loc["ols", "norma_l2"]

resultados_df["pct_reduccion_l1"] = 1 - resultados_df.norma_l1 / l1_ols
resultados_df["pct_reduccion_l2"] = 1 - resultados_df.norma_l2 / l2_ols

resultados_df

```

Out [22]:

	norma_l1	norma_l2	pct_reduccion_l1	pct_reduccion_l2
ols	10853.747955	3922.194711	0.0	0.0
regularizacion_l1	42.945147	33.494056	0.996043	0.99146
regularizacion_l2	457.523205	109.55068	0.957847	0.972069
regularizacion_elasticnet	21.002924	8.997096	0.998065	0.997706

In [23]:

```
#exportamos el dataset a un csv
resultados_df.to_csv("resultados_regularizacion.csv")
```

```
In [1]:
```

```
%load_ext watermark  
%watermark  
%matplotlib inline
```

```
2019-05-30T21:33:44+02:00
```

```
CPython 3.6.5  
IPython 6.4.0
```

```
compiler    : GCC 7.2.0  
system      : Linux  
release     : 5.1.5-arch1-2-ARCH  
machine     : x86_64  
processor   :  
CPU cores   : 4  
interpreter: 64bit
```

Regresion logistica

Teoría

Variables Categóricas.

Como hemos estudiado anteriormente, el aprendizaje en machine Learning se divide en aprendizaje supervisado y aprendizaje no supervisado. A su vez, el aprendizaje supervisado contempla dos tipos de variables: Las variables continuas y las variables categóricas.

Los problemas de clasificación son muy extensos, desde clasificación de imágenes a la clasificación de caracteres escritos.

¿Que es la Regresión Logística?

En estadística, la regresión logística es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica (una variable que puede adoptar un número limitado de categorías) en función de las variables independientes o predictoras. Es útil para modelar la probabilidad de un evento ocurriendo como función de otros factores. El análisis de regresión logística se enmarca en el conjunto de Modelos Lineales Generalizados (GLM por sus siglas en inglés) que usa como función de enlace la función logit. Las probabilidades que describen el posible resultado de un único ensayo se modelan, como una función de variables explicativas, utilizando una función logística.

La regresión logística es usada extensamente en las ciencias médicas y sociales. Otros nombres para regresión logística usados en varias áreas de aplicación incluyen modelo logístico, modelo logit, y clasificador de máxima entropía.

La regresión logística se puede utilizar para resolver problemas donde queremos clasificar, ya que la principal diferencia entre la regresión lineal y la logística es que en estas últimas las variables objetivo que trata son probabilidades de una clase (*Clase A o Clase B* o valores booleanos).

Esto traduce a su función lineal a una función donde el resultado siempre estará en cero.

Implementacion

Ingesta de datos

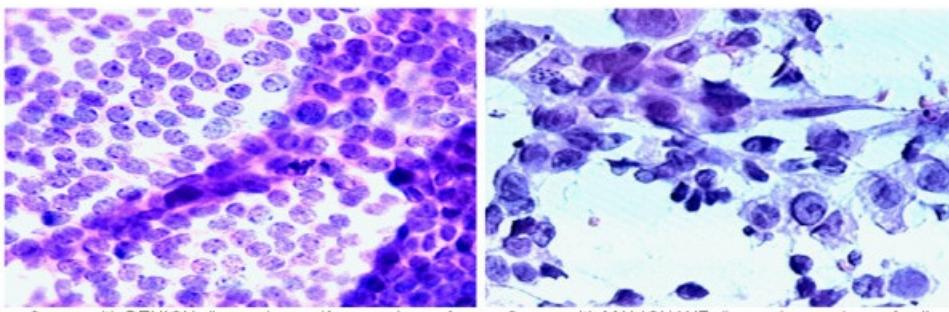
Para este ejemplo vamos a utilizar el (**Wisconsin Breast Cancer Dataset**). Es un dataset de imágenes de células obtenidas de análisis de personas que sufren un posible cáncer de mama.

Las imágenes tienen el siguiente aspecto:

```
In [2]:
```

```
from IPython.display import Image  
  
Image("../RESOURCES/breast_cancer.jpeg")
```

Out [2]:



Smear with BENIGN diagnosis – uniform nucleus of cells, symmetrical, homogeneous, with areas within normal size

Smear with MALIGNANT diagnosis – nucleus of cells without uniformity, asymmetrical, not homogeneous (multiple sizes) and with areas above normal size

In [3]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import datasets
```

In [4]:

```
cancer_datos = datasets.load_breast_cancer()
cancer_datos.keys()
```

Out [4]:

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [5]:

```
print(cancer_datos["DESCR"])
```

```
.. _breast_cancer_dataset:
```

Breast cancer wisconsin (diagnostic) dataset

Data Set Characteristics:

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness ($\text{perimeter}^2 / \text{area} - 1.0$)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:

- WDBC-Malignant
- WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0

area (mean):	145.0	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

===== ===== =====

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
<https://goo.gl/U2Uwz2>

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

.. topic:: References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)

In [6]:

cancer_datos["target"][:20]

Out[6]:

array([0, 1])

In [7]:

cancer_datos.target_names

Out[7]:

array(['malignant', 'benign'], dtype='<U9')

In [8]:

cancer_df = pd.DataFrame(cancer_datos["data"],
 columns=cancer_datos["feature_names"])

cancer_df["objetivo"] = cancer_datos.target

El dataset contiene los valores medios de ciertos parámetros del núcleo de las células mostradas en las imágenes, así como dichos valores para la célula con características más preocupantes

In [9]:

cancer_df.shape

Out[9]:

(569, 31)

In [10]:

cancer_df.head()

Out[10]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	per
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152

5 rows × 31 columns

In [11]:

cancer_df.objetivo.value_counts(True)

Out[11]:

```
1      0.627417
0      0.372583
Name: objetivo, dtype: float64
```

Implementación

Predicción de los datos mediante regresión lineal

Vamos a intentar predecir resultados en función al dataset anteriormente cargado mediante regresión lineal, donde demostraremos que los resultados no se encuentran entre los posibles valores 0 y 1

In [12]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [13]:

```
train_df, test_df = train_test_split(cancer_df, test_size=0.4)

variables_entrenamiento = cancer_datos["feature_names"]
variables_objetivo = "objetivo"
```

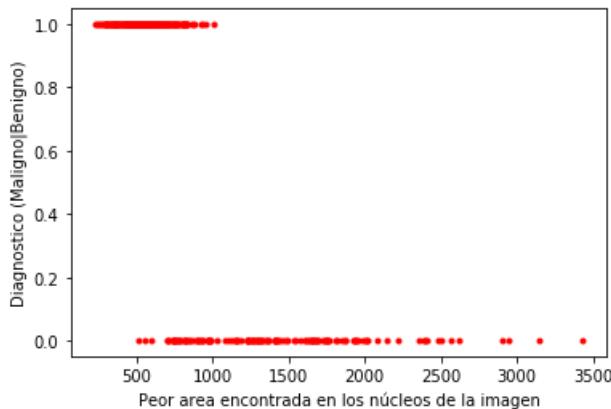
In [14]:

```
columna_entrenamiento = "worst area"

plt.plot(train_df[columna_entrenamiento], train_df.objetivo, '.r')
plt.xlabel("Peor area encontrada en los núcleos de la imagen")
plt.ylabel("Diagnóstico (Maligno|Benigno)")
```

Out[14]:

Text(0,0.5,'Diagnóstico (Maligno|Benigno)')



In [15]:

```
modelo_ols = LinearRegression()

modelo_ols.fit(train_df[[columna_entrenamiento]],
               train_df[variables_objetivo])

predicciones = modelo_ols.predict(test_df[[columna_entrenamiento]])

predicciones[:10]
```

Out[15]:

array([0.11885511, 0.15354393, 0.52291572, 0.12784851, 0.80826346,
 0.95222221 , 0.66558959, 0.88175239, 0.85033973, 0.57931719])

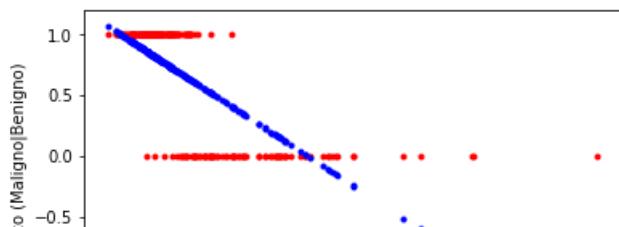
Como hemos observado en las predicciones de Regresión lineal, los resultados predecidos se encuentran más allá de los máximos y mínimos (0 y 1). Si obtenemos un gráfico de esta predicción, veremos más claramente como la linea sale de los margenes de los valores de las clases

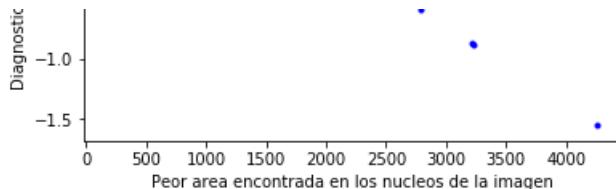
In [16]:

```
plt.plot(test_df[columna_entrenamiento], test_df.objetivo, '.r')
plt.plot(test_df[columna_entrenamiento], predicciones, '.b')
plt.xlabel("Peor area encontrada en los nucleos de la imagen")
plt.ylabel("Diagnóstico (Maligno|Benigno)")
```

Out[16]:

Text(0,0.5,'Diagnóstico (Maligno|Benigno)')





Aplicación de la función teórica

In [17]:

```
from ipywidgets import interact

def funcion_logistica(x, L=1, k=1, x0=0):
    return L / (1 + np.exp(-k*(x-x0)))

@interact(L=range(1,10), k=range(-5, 5), x0=range(0,10))
def plot_funcion_logit(L, k, x0):
    x = np.linspace(-5*k, 5*k, 500)
    y = funcion_logistica(x, k=k, L=L, x0=x0)
    plt.figure(1)
    plt.plot(x, y)
#    plt.show()
```

In [18]:

```
predicciones_probabilidades = list(map(funcion_logistica, predicciones))
```

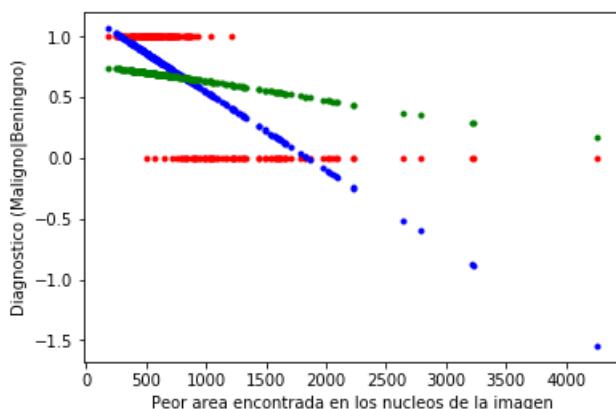
In [19]:

```
plt.plot(test_df[columna_entrenamiento], test_df.objetivo, '.r')
plt.plot(test_df[columna_entrenamiento], predicciones, '.b')
plt.plot(test_df[columna_entrenamiento], predicciones_probabilidades, '.g')

plt.xlabel("Peor area encontrada en los nucleos de la imagen")
plt.ylabel("Diagnóstico (Maligno|Beningno)")
```

Out[19]:

```
Text(0, 0.5, 'Diagnóstico (Maligno|Beningno)')
```



In [20]:

```
from functools import partial

funcion_logit_k5 = partial(funcion_logistica, k=5)

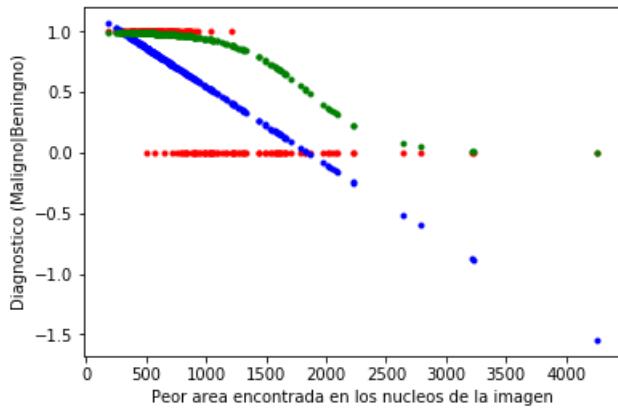
predicciones_probabilidades = list(map(funcion_logit_k5, predicciones))

plt.plot(test_df[columna_entrenamiento], test_df.objetivo, '.r')
plt.plot(test_df[columna_entrenamiento], predicciones, '.b')
plt.plot(test_df[columna_entrenamiento], predicciones_probabilidades, '.g')

plt.xlabel("Peor area encontrada en los nucleos de la imagen")
plt.ylabel("Diagnóstico (Maligno|Beningno)")
```

Out[20]:

```
Text(0,0.5,'Diagnóstico (Maligno|Beningno)')
```



Regresión Logística con Sklearn

```
In [21]:
```

```
from sklearn.linear_model import LogisticRegression
```

```
In [22]:
```

```
#Documentación  
LogisticRegression?
```

Cargamos las variables del dataframe y las ajustamos en datos de entrenamiento y datos de test

```
In [23]:
```

```
X = cancer_df[variables_entrenamiento]  
y = cancer_df[variables_objetivo]  
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Ajustamos el modelo y predecimos en función a la variable test de X. *clf* es un clasificador que por convención se llaman así al utilizarlos con ScikitLearn. Si observamos el resultado de las predicciones, vemos que ha predecido según la etiqueta objetivo (columna *objetivo* en el dataframe).

```
In [24]:
```

```
clf = LogisticRegression()  
clf.fit(X_train, y_train)  
predicciones = clf.predict(X_test)  
  
predicciones[:10]  
  
/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.  
FutureWarning)
```

```
Out[24]:
```

```
array([0, 1, 0, 1, 1, 0, 0, 1, 1, 1])
```

El método *predict_proba* nos da más información de la columna objetivo. El método nos permite obtener la probabilidad de la predicción, ofreciéndonos así un coeficiente de la posibilidad de que un cáncer sea maligno o benigno. Esto es muy útil en una aplicación para el mundo real, ya que con el método *predict* predecimos si el cáncer es maligno o benigno y con el método *predict_proba* podemos ver realmente las probabilidades de que sea maligno o benigno de cara a decírselo a un paciente (sin arriesgarnos a darle una respuesta absoluta y tener la posibilidad de fallar)

```
In [25]:
```

```
predicciones_probabilidades = clf.predict_proba(X_test)  
predicciones_probabilidades[:10]
```

```
Out[25]:
```

```
array([[1.00000000e+00, 7.53628070e-16],  
[9.95463787e-03, 9.90045362e-01],  
[1.00000000e+00, 4.90386956e-11],  
[4.35955773e-02, 9.56404423e-01],
```

```
[6.28861369e-02, 9.37113863e-01],
[9.93008258e-01, 6.99174183e-03],
[9.99945277e-01, 5.47228506e-05],
[6.16577457e-03, 9.93834225e-01],
[1.33981058e-03, 9.98660189e-01],
[5.82927101e-03, 9.94170729e-01]])
```

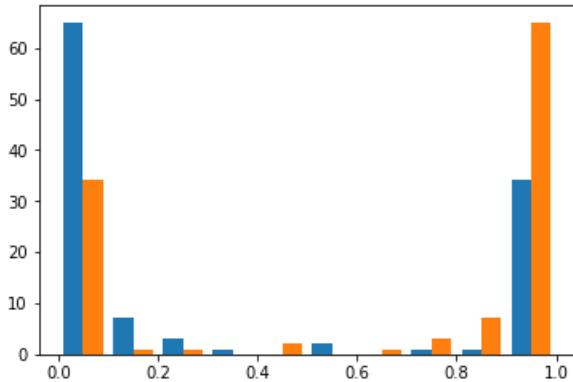
Obtenemos un histograma con las probabilidades de las predicciones realizadas

In [26]:

```
plt.hist(predicciones_probabilidades)
```

Out [26]:

```
([array([65., 7., 3., 1., 0., 2., 0., 1., 1., 34.]),
 array([34., 1., 1., 0., 2., 0., 1., 3., 7., 65.])],
 array([8.84843152e-17, 1.00000000e-01, 2.00000000e-01, 3.00000000e-01,
        4.00000000e-01, 5.00000000e-01, 6.00000000e-01, 7.00000000e-01,
        8.00000000e-01, 9.00000000e-01, 1.00000000e+00]),
 <a list of 2 Lists of Patches objects>)
```



Para exportar y visualizar mejor los resultados, exportamos las predicciones y la probabilidad a un dataframe, organizando las columnas del mismo de la siguiente manera:

- **objetivo**: valor real del dataframe original.
- **prediccion**: valor predecido por el algoritmo.
- 0: posibilidad de que sea cáncer maligno.
- 1: probabilidad de que sea cáncer benigno.

In [27]:

```
probs_df = pd.DataFrame(predicciones_probabilidades)
```

In [28]:

```
X = X_test.reset_index().copy()
X["objetivo"] = y_test.tolist()
X["prediccion"] = predicciones
X = pd.concat([X, probs_df], axis=1)
X[['objetivo','prediccion',0,1]].head(10)
```

Out [28]:

	objetivo	prediccion	0	1
0	0	0	1.000000	7.536281e-16
1	1	1	0.009955	9.900454e-01
2	0	0	1.000000	4.903870e-11
3	1	1	0.043596	9.564044e-01
4	1	1	0.062886	9.371139e-01
5	0	0	0.993008	6.991742e-03
6	0	0	0.999945	5.472285e-05
7	1	1	0.006166	9.938342e-01
8	1	1	0.001210	9.999999e-01

0	objetivo	prediccion	0.001540	9.900000e-01
9	1	1	0.005829	9.941707e-01

In [29]:

```
probs_df.to_csv("prediccion_con_RegresionLogistica.csv")
```

```
In [1]:
```

```
%load_ext watermark  
%watermark
```

```
2019-05-30T21:34:12+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Evaluación de Modelos de Clasificación

En este apartado evaluaremos la calidad del modelo, utilizando técnicas como la clasificación binaria. Como en el apartado de regresión logística, emplearemos el dataset `breast_cancer`

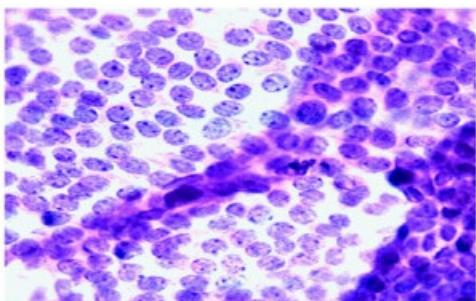
```
In [2]:
```

```
%matplotlib inline  
import matplotlib.pyplot as plt  
  
import pandas as pd  
import numpy as np  
from sklearn import datasets
```

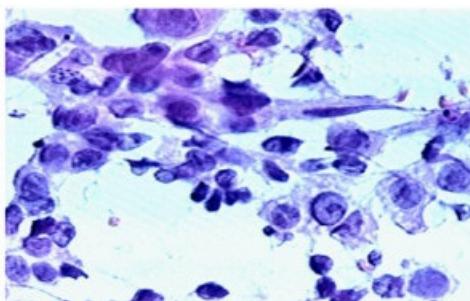
```
In [3]:
```

```
from IPython.display import Image  
  
Image("../RESOURCES/breast_cancer.jpeg")
```

```
Out[3]:
```



Smear with BENIGN diagnosis – uniform nucleus of cells, symmetrical, homogeneous, with areas within normal size



Smear with MALIGNANT diagnosis – nucleus of cells without uniformity, asymmetrical, not homogeneous (multiple sizes) and with areas above normal size

Ingesta de Datos

```
In [4]:
```

```
cancer_datos = datasets.load_breast_cancer()  
  
cancer_df = pd.DataFrame(cancer_datos["data"],  
                         columns=cancer_datos["feature_names"]  
                         )  
  
cancer_df["objetivo"] = cancer_datos.target  
cancer_df["objetivo"] = cancer_df["objetivo"].replace({0:1, 1:0})
```

```
In [5]:
```

```
cancer_df["objetivo"].value_counts(True)
```

```
Out[5]:
```

```
0    0.627417
1    0.372583
Name: objetivo, dtype: float64
```

```
In [6]:
```

```
cancer_df.head()
```

```
Out[6]:
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	per
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152

5 rows × 31 columns



Creción de Modelo de Regresión Logística

```
In [7]:
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
In [8]:
```

```
X = cancer_df[cancer_datos.feature_names]
y = cancer_df["objetivo"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
In [9]:
```

```
modelo = LogisticRegression()

modelo.fit(X_train, y_train)

predicciones = modelo.predict(X_test)
clases_reales = y_test
#predicimos las probabilidades
predicciones_probabilidades = modelo.predict_proba(X_test)

/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
```

Implementaremos una función que devuelva una lista para cada observación entre la clase real y la clase predicho

```
In [10]:
```

```
def tupla_clase_prediccion(y_real, y_pred):
    return list(zip(y_real, y_pred))

tupla_clase_prediccion(clases_reales, predicciones)[:10]
```

```
Out[10]:
```

```
[(0, 0),
 (1, 1),
 (1, 1),
 (0, 0),
 (0, 0),
 (1, 1),
```

```
(1, 1),
(1, 1),
(0, 0),
(0, 0)]
```

Conceptos de Clasificación binaria

En clasificación binaria, tenemos el concepto de casos negativos (clase 0, en el caso del dataset de cancer de mama serian los casos donde el cancer es benigno) y casos positivos (clase 1, en el caso del dataset de cancer de mama serian los casos donde el cancer es maligno). Esto nos lleva a cuatro tipos de posibilidades u observaciones posibles:

- Verdaderos Positivos(True positives), serian las imágenes con un cancer maligno que se detectan como cancer maligno.
- Falsos Positivos (False positives), serian los cánceres benignos que se detectan como un cancer maligno.
- Verdaderos Negativos(True Negatives), serian los canceres benignos que se clasifican como cánceres benignos.
- Falsos Negativos(False Negatives), serian los canceres malignos que se clasifican como cánceres benignos.

Implementaremos una serie de funciones encargadas de calcular los valores correspondientes para los cuatro tipo de observaciones realizadas.

In [11]:

```
def VP(clases_reales, predicciones):
    par_clase_prediccion = tupla_clase_prediccion(clases_reales, predicciones)
    return len([obs for obs in par_clase_prediccion if obs[0]==1 and obs[1]==1])

def VN(clases_reales, predicciones):
    par_clase_prediccion = tupla_clase_prediccion(clases_reales, predicciones)
    return len([obs for obs in par_clase_prediccion if obs[0]==0 and obs[1]==0])

def FP(clases_reales, predicciones):
    par_clase_prediccion = tupla_clase_prediccion(clases_reales, predicciones)
    return len([obs for obs in par_clase_prediccion if obs[0]==0 and obs[1]==1])

def FN(clases_reales, predicciones):
    par_clase_prediccion = tupla_clase_prediccion(clases_reales, predicciones)
    return len([obs for obs in par_clase_prediccion if obs[0]==1 and obs[1]==0])

print("""
Verdaderos Positivos: {}
Verdaderos Negativos: {}
Falsos Positivos: {}
Falsos Negativos: {}
""".format(
    VP(clases_reales, predicciones),
    VN(clases_reales, predicciones),
    FP(clases_reales, predicciones),
    FN(clases_reales, predicciones)
))
)

Verdaderos Positivos: 59
Verdaderos Negativos: 106
Falsos Positivos: 2
Falsos Negativos: 4
```

Ratios de clasificación

Exactitud (Accuracy)

La exactitud es una medida general de como se comporta el modelo, mide simplemente el porcentaje de casos que se han clasificado correctamente.

$$\text{Exactitud} = \frac{\text{Número de observaciones correctamente clasificadas}}{\text{Número de observaciones totales}} = \frac{VP + VN}{VP + VN + FP + FN}$$

In [12]:

```
def exactitud(clases_reales, predicciones):
    vp = VP(clases_reales, predicciones)
    vn = VN(clases_reales, predicciones)
    return (vp+vn) / len(clases_reales)
```

```
exactitud(clases_reales, predicciones)
```

Out[12]:

```
0.9649122807017544
```

In [13]:

```
from sklearn import metrics  
metrics.accuracy_score(clases_reales, predicciones)
```

Out[13]:

```
0.9649122807017544
```

Precisión (Precision)

La precisión indica la habilidad del modelo para clasificar como positivos los casos que son positivos.

$$\text{Precisión} = \frac{\text{Número de observaciones positivas correctamente clasificadas}}{\text{Número de observaciones clasificadas como positivas}} = \frac{VP}{VP+FP}$$

In [14]:

```
def precision(clases_reales, predicciones):  
    vp = VP(clases_reales, predicciones)  
    fp = FP(clases_reales, predicciones)  
    return vp / (vp+fp)
```

```
precision(clases_reales, predicciones)
```

Out[14]:

```
0.9672131147540983
```

In [15]:

```
metrics.average_precision_score(clases_reales, predicciones)
```

Out[15]:

```
0.9291945711272717
```

Exhaustividad o sensibilidad(Recall o True Positive Rate)

La sensibilidad nos da una medida de la habilidad del modelo para encontrar todos los casos positivos. La sensibilidad se mide en función de una clase.

$$\text{Sensibilidad} = \frac{\text{Número de observaciones positivas clasificadas como positivas}}{\text{Número de observaciones positivas totales}} = \frac{VP}{VP+FN}$$

In [16]:

```
def sensibilidad(clases_reales, predicciones):  
    vp = VP(clases_reales, predicciones)  
    fn = FN(clases_reales, predicciones)  
    return vp / (vp+fn)
```

```
sensibilidad(clases_reales, predicciones)
```

Out[16]:

```
0.9365079365079365
```

In [17]:

```
metrics.recall_score(clases_reales, predicciones)
```

Out[17]:

```
0.9365079365079365
```

Matriz de confusión

La matriz de confusión es una forma muy sencilla de comparar como ha clasificado cada observación el modelo.

In [18]:

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(clases_reales, predicciones)
```

```
Out[18]:
```

```
array([[106,    2],  
       [ 4,  59]])
```

Puntuación F1 (F1 score)

La puntuación F1 es una media ponderada entre la sensibilidad (que intenta obtener cuantos mas verdaderos positivos independientemente de los falsos positivos) y la precisión (que intenta obtener solo verdaderos positivos que sean casos claros para limitar los falsos positivos).

La puntuación F1 se define como la media harmónica de la precisión y la sensibilidad:

$$\text{F1} = \frac{2 * \text{precision} * \text{sensibilidad}}{\text{precision} + \text{sensibilidad}}$$

```
In [19]:
```

```
def puntuacion_f1(clases_reales, predicciones):  
    precision_preds = precision(clases_reales, predicciones)  
    sensibilidad_preds = sensibilidad(clases_reales, predicciones)  
    return 2 * (precision_preds * sensibilidad_preds) / (precision_preds + sensibilidad_preds)
```

```
puntuacion_f1(clases_reales, predicciones)
```

```
Out[19]:
```

```
0.9516129032258064
```

```
In [20]:
```

```
metrics.f1_score(clases_reales, predicciones)
```

```
Out[20]:
```

```
0.9516129032258064
```

Ratio de Falsos Positivos (Ratio de Falsa Alarma o FPR)

El ratio de falsos positivos nos da una medida de las probabilidades de nuestro modelo de asignar una clase positiva a un caso negativo.

Se define como:

$$\text{FPR} = \frac{\text{Número de observaciones negativas clasificadas como positivas}}{\text{Número de observaciones negativas}} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

```
In [21]:
```

```
def fpr(clases_reales, predicciones):  
    return (FP(clases_reales, predicciones) /  
           (FP(clases_reales, predicciones) + VN(clases_reales, predicciones)))  
fpr(clases_reales, predicciones)
```

```
Out[21]:
```

```
0.018518518518517
```

```
In [60]:
```

```
%reload_ext watermark  
%watermark
```

```
2019-05-30T21:34:58+02:00
```

```
Cython 3.6.5  
IPython 6.4.0
```

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Procesado de variables

```
In [1]:
```

```
%matplotlib inline  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
In [2]:
```

```
datos = pd.read_csv("../..../RESOURCES/datos_procesamiento.csv")  
datos.head()
```

```
Out[2]:
```

	col_inexistente1	col2	col3	col_outliers	col_outliers2	col_categorica	col_ordinal	col_texto
0	59.0	52.0	2.232832	-50	0.771666	ratón	muy bien	Tenía en su casa una ama que pasaba de los cua...
1	31.0	74.0	0.906147	-5	1.068558	elefante	regular	El resto della concluían sayo de velarte, calz...
2	81.0	28.0	0.626750	-32	0.846396	ratón	muy mal	El resto della concluían sayo de velarte, calz...
3	34.0	16.0	0.816738	-84	0.637381	gato	mal	Una olla de algo más vaca que carnero, salpicó...
4	32.0	28.0	0.571131	65	4.540614	gato	bien	Tenía en su casa una ama que pasaba de los cua...

```
In [3]:
```

```
datos.dtypes
```

```
Out[3]:
```

```
col_inexistente1      float64  
col2                  float64  
col3                  float64  
col_outliers          int64  
col_outliers2         float64  
col_categorica        object  
col_ordinal           object  
col_texto             object  
dtype: object
```

```
In [4]:
```

```
datos.shape
```

```
Out[4]:
```

```
(1000, 8)
```

Variables numéricas

Imputacion de datos

```
In [5]:
```

```
from sklearn import preprocessing
```

```
In [6]:
```

```
#Seleccionamos todas aquellas columnas que sean int o float, quitando asi las de string
var_numericas_df = datos.select_dtypes([int,float])
var_numericas_df.columns
```

```
Out[6]:
```

```
Index(['col_inexistente1', 'col2', 'col3', 'col_outliers', 'col_outliers2'], dtype='object')
```

```
In [7]:
```

```
#vamos a ver cuantos casos son NaN
var_numericas_df[var_numericas_df.isnull().any(axis=1)].shape
```

```
Out[7]:
```

```
(96, 5)
```

```
In [8]:
```

```
var_numericas_df[var_numericas_df.isnull().any(axis=1)].head()
```

```
Out[8]:
```

	col_inexistente1	col2	col3	col_outliers	col_outliers2
9	NaN	53.0	2.270999	62	1.067230
10	NaN	99.0	1.394209	98	4.145716
16	NaN	50.0	0.437365	59	20.549474
17	NaN	73.0	0.324893	98	0.761684
23	NaN	85.0	3.664671	-48	3.154153

```
In [9]:
```

```
#mediante el imputador preprocesamos el dataset mediante una estrategia, en este caso la media.
imputador = preprocessing.Imputer(strategy="mean")
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/utils/deprecation.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22.
Import impute.SimpleImputer from sklearn instead.
warnings.warn(msg, category=DeprecationWarning)
```

```
In [10]:
```

```
var_numericas_imputadas = imputador.fit_transform(var_numericas_df)
```

```
In [11]:
```

```
var_numericas_imputadas
```

```
Out[11]:
```

```
array([[ 59.          ,  52.          ,   2.23283208, -50.          ,
         0.77166646],
       [ 31.          ,  74.          ,   0.90614714,  -5.          ,
        1.06855838],
       [ 81.          ,  28.          ,   0.62675042, -32.          ,
        0.84639576],
       ...,
       [ 19.          ,  53.          ,   0.73723413,  73.          ,
        1.34525201],
       [ 88.          ,  94.          ,   0.76008706,  68.          ,
        1.00000000]])
```

```
1.3692463 ] ,  
[ 94. , 56. , 1.2299403 , 61. ,  
0.94395714]])
```

In [12]:

```
#convertimos el array de numpy del imputador a un dataframe  
var_numericas_imputadas_df = pd.DataFrame(  
    var_numericas_imputadas,index=var_numericas_df.index,columns=var_numericas_df.columns  
)  
var_numericas_imputadas_df.head(10)
```

Out [12]:

	col_inexistente1	col2	col3	col_outliers	col_outliers2
0	59.000000	52.0	2.232832	-50.0	0.771666
1	31.000000	74.0	0.906147	-5.0	1.068558
2	81.000000	28.0	0.626750	-32.0	0.846396
3	34.000000	16.0	0.816738	-84.0	0.637381
4	32.000000	28.0	0.571131	65.0	4.540614
5	81.000000	4.0	1.618844	51.0	0.812940
6	57.000000	31.0	0.167880	78.0	1.235137
7	34.000000	20.0	20.229813	93.0	1.283176
8	37.000000	96.0	2.407978	54.0	1.298613
9	48.382743	53.0	2.270999	62.0	1.067230

In [13]:

```
#Comprobamos que ahora ya no hay variables nulas  
var_numericas_imputadas_df[var_numericas_imputadas_df.isnull().any(axis=1)].shape
```

Out [13]:

```
(0, 5)
```

Estandarizacion

El proceso de estandarizacion es un proceso requerido en el que el objetivo es obtener una variable con media 0 y desviacion estandar 1

In [14]:

```
var_numericas_df.columns
```

Out [14]:

```
Index(['col_inexistente1', 'col2', 'col3', 'col_outliers', 'col_outliers2'], dtype='object')
```

In [15]:

```
#ibtenemos la media de cada una de las columnasabs  
var_numericas_df.mean()
```

Out [15]:

```
col_inexistente1      48.382743  
col2                  49.660000  
col3                  1.466095  
col_outliers         4.253000  
col_outliers2        131.193340  
dtype: float64
```

In [16]:

```
#obtenemos la desviacion estandar de cada una de las columnas  
var_numericas_df.std()
```

Out [16]:

```
col_inexistente1      27.987174  
col2                  28.272668
```

```
col3           1.732358
col_outliers   78.145901
col_outliers2  3401.164776
dtype: float64
```

Ahora procedemos a estandarizar el dataframe, para ello utilizaremos una herramienta de sklearn: **StandardScaler**

In [17]:

```
escalador = preprocessing.StandardScaler()
var_numericas_imputadas_escalado_standard = escalador.fit_transform(var_numericas_imputadas)
```

In [18]:

```
escalador.mean_
```

Out[18]:

```
array([ 48.38274336,  49.66      ,  1.46609489,  4.253      ,
       131.19333968])
```

In [19]:

```
#Observamos la media despues de la estandarizacion, en la que sus campos son muy proximos a 0
var_numericas_imputadas_escalado_standard.mean(axis=0)
```

Out[19]:

```
array([-5.86197757e-17,  1.26121336e-16, -3.81916720e-17,  3.55271368e-18,
       -3.55271368e-18])
```

In [20]:

```
#Observamos la desviacion estandard y comprobamos que es 1
var_numericas_imputadas_escalado_standard.std(axis=0)
```

Out[20]:

```
array([1., 1., 1., 1., 1.])
```

In [21]:

```
#Observamos la primera observacion
var_numericas_imputadas_escalado_standard[0]
```

Out[21]:

```
array([ 0.39921733,  0.08280686,  0.44281884, -0.69460006, -0.03836537])
```

Para aquellos datasets con valores muy extremos, esta herramienta no sería la mas apropiada y sería más optimo usar estimadores mas robustos (menos sensibles a los Outliers). Para ello podemos emplear **RobustScaler** que funciona substrayendo la mediana y escalando mediante el rango intercuartil (IQR). (Este escalador robusto funciona igual que el anterior, salvo por la diferencia de que en vez de la media utiliza la mediana y en vez de la desviacion estandard utiliza el rango intercuartil)

In [22]:

```
escalador_robusto = preprocessing.RobustScaler()
var_numericas_imputadas_escalado_robusto = escalador_robusto.fit_transform(var_numericas_imputadas)
```

In [23]:

```
var_numericas_imputadas_escalado_robusto.mean(axis=0)
```

Out[23]:

```
array([-3.81916720e-17, -2.85106383e-02,  4.01958704e-01,  3.04018692e-02,
       7.03130782e+01])
```

In [24]:

```
var_numericas_imputadas_escalado_robusto.std(axis=0)
```

Out[24]:

```
array([6.33218559e-01,  6.01245275e-01,  1.38651621e+00,  7.29970260e-01,
       1.83690817e+03])
```

Escalado a un rango específico.

Hay casos en los que en vez de estandarizar el modelo nos interesa mas ajustar los datos a un rango específico (generalmente -1,1 o 0,1). Para ello utilizamos la herramienta **MinMaxScaler** que hace escalado minmax. Tambien podemos utilizar el **MaxAbsScaler** que simplemente divide cada valor de una variable por su valor maximo (y por tanto convierte el valor maximo a 1)

In [25]:

```
#comprobamos el valor minimo antes del escalado  
var_numericas_imputadas.min()
```

Out[25]:

-100.0

In [26]:

```
#comprobamos el valor maximo antes del escalado  
var_numericas_imputadas.max()
```

Out[26]:

107357.85777352

In [27]:

```
#escalamos con la libreria preprocessing y la herramienta MinMaxScaler  
escalador_minmax = preprocessing.MinMaxScaler()  
var_numericas_imputadas_escalado_minmax = escalador_minmax.fit_transform(var_numericas_imputadas)
```

In [28]:

```
#comprobamos el minimo despues de escalarlo  
var_numericas_imputadas_escalado_minmax.min()
```

Out[28]:

0.0

In [29]:

```
#comprobamos el maximo despues de escalarlo  
var_numericas_imputadas_escalado_minmax.max()
```

Out[29]:

1.0

In [30]:

```
#Creamos ahora un escalador con la herramienta MaxAbsScaler  
escalador_maxabs = preprocessing.MaxAbsScaler()  
var_numericas_imputadas_escalado_maxabs = escalador_maxabs.fit_transform(var_numericas_imputadas)
```

In [31]:

```
#comprobamos los maximos tras aplicar el escalador MaxAbs  
var_numericas_imputadas_escalado_maxabs.max()
```

Out[31]:

1.0

In [32]:

```
#comprobamos los minimos tras aplicar el escalador MaxAbs y comprobamos que no se preocupa por el  
minimo y solo  
#divide por el maximo  
var_numericas_imputadas_escalado_maxabs.min()
```

Out[32]:

-0.1122334455667789

Hay casos en los que es necesario tener observaciones con norma unitaria (norma L2 o eucladiana). Para estos casos se utilizará el método **normalizer**

In [33]:

```
normalizador = preprocessing.Normalizer()
```

```
var_numericas_imputadas_normal = normalizador.fit_transform(var_numericas_imputadas)
```

In [34]:

```
var_numericas_imputadas_normal[:1]
```

Out[34]:

```
array([[ 0.63288908,  0.55780055,  0.02395144, -0.53634668,  0.00827761]])
```

In [35]:

```
np.linalg.norm(var_numericas_imputadas[1,:])
```

Out[35]:

```
80.39877436664493
```

Otras transformaciones

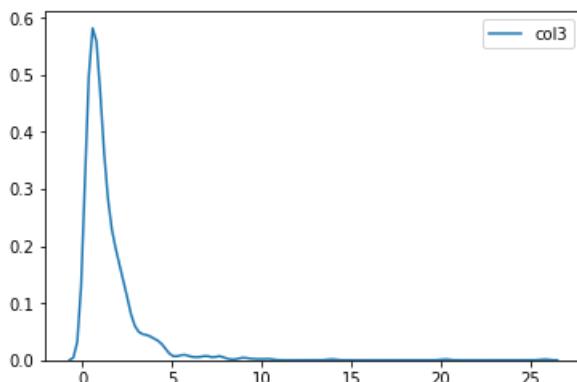
Para aquellos casos en los que queremos aplicar una función arbitraria a una variable podemos usar **FunctionTransformer**. La variable *col3* no tiene una distribución normal, sino que tiene una asimetría muy marcada.

In [36]:

```
sns.kdeplot(datos.col3)
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1943176128>
```



Una práctica muy frecuente es aplicar el logaritmo a dichas variables para convertirlas a variables con una distribución más normal.

In [37]:

```
from sklearn.preprocessing import FunctionTransformer
#la función np.log1p es para sumar siempre uno para evitar el logaritmo de 0 (que es infinito)
transformer = FunctionTransformer(np.log1p)
```

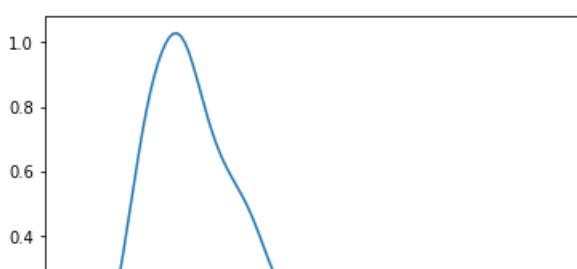
In [38]:

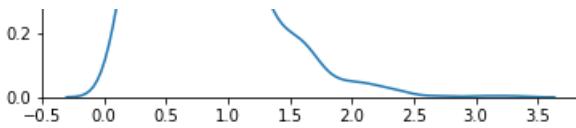
```
col3_transformada= transformer.transform(datos[["col3"]])
col3_transformada = col3_transformada.reshape(col3_transformada.shape[0],)
sns.kdeplot(col3_transformada)
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_function_transformer.py:98:
FutureWarning: The default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f193ffd3e48>
```





Vemos que tras aplicar el transformador de FunctionTransformer aplicando como parámetro la función logaritmo de numpy el registro de col3 está mucho más normalizado

Variables Categóricas

Los modelos están diseñados para trabajar con variables numéricas. Esto implica que para poder entrenar los modelos con variables categóricas tenemos que convertirlas a números. Este proceso se le conoce como **codificación**(encoding). Básicamente el proceso consiste en utilizar los registros de texto en valores ordinarios.

In [39]:

```
datos = pd.read_csv("../..../RESOURCES/datos_procesamiento.csv")
datos.head()
```

Out [39]:

	col_inexistente1	col2	col3	col_outliers	col_outliers2	col_categorica	col_ordinal	col_texto
0	59.0	52.0	2.232832	-50	0.771666	ratón	muy bien	Tenía en su casa una ama que pasaba de los cu...
1	31.0	74.0	0.906147	-5	1.068558	elefante	regular	El resto della concluían sayo de velarte, calz...
2	81.0	28.0	0.626750	-32	0.846396	ratón	muy mal	El resto della concluían sayo de velarte, calz...
3	34.0	16.0	0.816738	-84	0.637381	gato	mal	Una olla de algo más vaca que carnero, salpicó...
4	32.0	28.0	0.571131	65	4.540614	gato	bien	Tenía en su casa una ama que pasaba de los cu...

In [40]:

```
var_categoricas = datos[['col_categorica', 'col_ordinal']]
```

In [41]:

```
var_categoricas.head()
```

Out [41]:

	col_categorica	col_ordinal
0	ratón	muy bien
1	elefante	regular
2	ratón	muy mal
3	gato	mal
4	gato	bien

ahora debemos 'separar' estos valores por valores numéricos. A este proceso en Scikit Learn se conoce como **LabelEncoder**. En nuestro caso, como no queremos que los animales tengan un valor 'real' numérico (que no pueda darse una situación de que un ratón más un gato es igual a un elefante por ejemplo) lo que haremos será emplear una técnica conocida como **OneHotEncoder**.

LabelEncoder

Aplicamos un transformador LabelEncoder()

In [42]:

```
label_codificador_categorico=preprocessing.LabelEncoder()
label_codificador_categorico.fit_transform(datos.col_categorica)[1:101]
```

Out[42]:

```
array([3, 0, 3, 1, 1, 2, 2, 2, 0, 0])
```

OneHotEncoder

Aplicamos un transformador OneHotEncoder()

In [43]:

```
ohcodificador = preprocessing.OneHotEncoder()
```

In [44]:

```
ohcodificador.fit(datos.col categorica)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-44-f63cf08d548e> in <module>()
      1 ohcodificador.fit(datos.col_categorica)
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py in fit(self, X, y)
    427         return self
    428     else:
--> 429         self._fit(X, handle_unknown=self.handle_unknown)
    430     return self
    431
```

/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py in _fit(self, X, handle_unknown)

```
 59
 60     def _fit(self, X, handle_unknown='error'):
--> 61         X = self._check_X(X)
 62
 63         n_samples, n_features = X.shape
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py in _check_X(self, X)
    45
    46         """
--> 47         X_temp = check_array(X, dtype=None)
    48         if not hasattr(X, 'dtype') and np.issubdtype(X_temp.dtype, np.str_):
    49             X = check_array(X, dtype=np.object)
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
```

```
550             "Reshape your data either using array.reshape(-1, 1) if "
551             "your data has a single feature or array.reshape(1, -1) "
--> 552             "if it contains a single sample.".format(array))
```

553 # in the future np.flexible dtypes will be handled like object dtypes
554

```
ValueError: Expected 2D array, got 1D array instead:  
array=['ratón' 'elefante' 'ratón' 'gato' 'gato' 'perro' 'perro' 'perro'  
'elefante' 'elefante' 'perro' 'elefante' 'gato' 'ratón' 'gato' 'ratón'  
'elefante' 'ratón' 'ratón' 'elefante' 'ratón' 'ratón' 'ratón' 'gato'  
'gato' 'ratón' 'gato' 'elefante' 'gato' 'elefante' 'elefante' 'elefante'  
'elefante' 'ratón' 'elefante' 'gato' 'elefante' 'gato' 'gato' 'gato'  
'gato' 'ratón' 'elefante' 'gato' 'gato' 'perro' 'elefante' 'gato'  
'elefante' 'perro' 'ratón' 'perro' 'elefante' 'ratón' 'ratón' 'gato'  
'gato' 'elefante' 'ratón' 'perro' 'gato' 'gato' 'gato' 'ratón'  
'elefante' 'elefante' 'perro' 'gato' 'ratón' 'perro' 'elefante' 'gato'  
'gato' 'ratón' 'ratón' 'elefante' 'perro' 'ratón' 'ratón' 'elefante'  
'gato' 'perro' 'ratón' 'gato' 'elefante' 'elefante' 'perro' 'perro'  
'perro' 'ratón' 'gato' 'ratón' 'gato' 'gato' 'elefante' 'gato' 'gato'  
'perro' 'perro' 'ratón' 'perro' 'gato' 'elefante' 'ratón' 'perro' 'gato'  
'gato' 'ratón' 'gato' 'perro' 'perro' 'perro' 'gato' 'perro' 'perro'  
'elefante' 'perro' 'ratón' 'gato' 'gato' 'gato' 'ratón' 'ratón'  
'ratón' 'perro' 'elefante' 'elefante' 'perro' 'ratón' 'gato' 'elefante'  
'gato' 'ratón' 'ratón' 'elefante' 'gato' 'perro' 'ratón' 'gato'  
'elefante' 'elefante' 'elefante' 'elefante' 'perro' 'elefante' 'elefante'  
'gato' 'ratón' 'perro' 'perro' 'gato' 'perro' 'perro' 'elefante' 'gato'  
'perro' 'perro' 'perro' 'perro' 'gato' 'ratón' 'gato' 'gato' 'gato'  
'perro' 'ratón' 'ratón' 'elefante' 'ratón' 'ratón' 'elefante' 'elefante'  
'ratón' 'elefante' 'elefante' 'elefante' 'ratón' 'ratón' 'gato' 'perro'
```



```
'perro' 'elefante' 'elefante' 'elefante' 'gato' 'elefante' 'perro' 'gato'
'elefante' 'elefante' 'ratón' 'gato' 'gato' 'ratón' 'perro' 'elefante'
'perro' 'perro' 'perro' 'elefante' 'perro' 'perro' 'elefante' 'perro'
'elefante' 'gato' 'elefante' 'gato' 'gato' 'perro' 'gato' 'ratón'
'elefante' 'perro' 'perro' 'perro' 'elefante' 'perro' 'ratón' 'gato'
'elefante' 'gato' 'ratón' 'gato' 'gato' 'ratón' 'elefante' 'gato'
'elefante' 'elefante' 'gato' 'perro' 'gato' 'perro' 'ratón'
'elefante' 'elefante' 'elefante' 'perro' 'ratón' 'perro'
'ratón' 'gato' 'elefante' 'ratón' 'ratón' 'perro' 'elefante' 'ratón'
'ratón' 'perro' 'gato' 'elefante' 'ratón' 'ratón' 'gato' 'gato'
'elefante' 'perro' 'elefante' 'perro' 'gato' 'perro' 'perro' 'perro'
'perro' 'ratón' 'ratón' 'gato' 'perro' 'gato' 'elefante' 'perro' 'ratón'
'perro' 'ratón' 'perro' 'gato' 'elefante' 'ratón' 'gato' 'perro' 'ratón'
'perro' 'elefante' 'gato' 'ratón' 'perro' 'elefante' 'ratón' 'ratón'
'ratón' 'perro' 'gato' 'gato' 'gato' 'perro' 'ratón' 'perro' 'ratón'
'elefante' 'perro' 'perro' 'ratón' 'elefante' 'ratón' 'elefante' 'ratón'
'gato' 'gato' 'gato' 'perro' 'elefante' 'gato' 'perro' 'gato' 'ratón'
'ratón' 'gato' 'elefante' 'gato' 'elefante' 'elefante' 'gato' 'ratón'
'ratón' 'ratón' 'gato' 'elefante' 'perro' 'perro' 'elefante' 'ratón'
'perro' 'elefante' 'ratón' 'gato' 'perro' 'ratón' 'gato' 'ratón'
'ratón' 'gato' 'perro' 'perro' 'perro' 'elefante' 'gato' 'ratón' 'ratón'
'elefante' 'ratón' 'elefante' 'elefante'].
```

Reshape your data either using `array.reshape(-1, 1)` if your data has a single feature or `array.reshape(1, -1)` if it contains a single sample.

Vemos que el transformador OneHotEncoder falla cuando recibe cadenas en vez de numeros. Por ello debemos convertir las variables categóricas a numéricas usando LabelEncoder.

In [45]:

```
categorias_codificadas = label_codificador_categorico.transform(datos.col_categorica)
```

In [46]:

```
categorias_codificadas
```

Out[46]:

```
array([3, 0, 3, 1, 1, 2, 2, 2, 0, 0, 2, 0, 1, 3, 1, 3, 0, 3, 3, 0, 3, 3,
       3, 1, 1, 3, 1, 0, 1, 0, 0, 0, 0, 3, 0, 1, 0, 1, 1, 1, 1, 3, 0, 1,
       1, 2, 0, 1, 0, 2, 3, 2, 0, 3, 3, 1, 1, 0, 3, 2, 1, 1, 1, 1, 3, 0,
       0, 2, 1, 3, 2, 0, 1, 1, 3, 3, 0, 2, 3, 3, 0, 1, 2, 3, 1, 0, 0, 2,
       2, 2, 3, 1, 3, 1, 1, 0, 1, 1, 2, 2, 3, 2, 1, 0, 3, 2, 1, 1, 3, 1,
       2, 2, 2, 1, 2, 2, 0, 2, 3, 1, 1, 1, 3, 3, 3, 2, 0, 0, 2, 3, 1,
       0, 1, 3, 3, 0, 1, 2, 3, 1, 0, 0, 0, 2, 0, 0, 1, 3, 2, 2, 1, 2,
       2, 0, 1, 2, 2, 2, 1, 3, 1, 1, 1, 2, 3, 3, 0, 3, 3, 0, 0, 3, 0,
       0, 0, 3, 3, 1, 2, 0, 0, 1, 1, 2, 0, 0, 0, 3, 2, 2, 1, 3, 1, 2, 3,
       0, 3, 3, 1, 3, 1, 3, 0, 1, 1, 2, 0, 0, 0, 3, 2, 2, 1, 3, 1, 2, 3,
       0, 2, 0, 0, 2, 1, 1, 0, 3, 2, 2, 3, 3, 2, 2, 0, 0, 2, 1, 2, 3, 2,
       3, 1, 3, 2, 0, 1, 1, 3, 3, 2, 1, 3, 3, 2, 0, 1, 3, 2, 0, 3, 0, 2,
       3, 2, 2, 2, 3, 3, 3, 2, 0, 2, 0, 1, 0, 0, 0, 3, 0, 3, 0, 2, 3,
       1, 0, 2, 2, 3, 2, 3, 1, 1, 2, 1, 3, 2, 0, 1, 2, 1, 0, 2, 3, 1, 2,
       0, 0, 1, 3, 1, 0, 2, 2, 3, 1, 1, 2, 1, 2, 3, 3, 2, 3, 1, 1, 3,
       3, 1, 2, 0, 0, 1, 3, 1, 1, 3, 0, 3, 3, 0, 0, 1, 0, 3, 0, 2, 0,
       0, 0, 3, 3, 1, 2, 0, 0, 1, 1, 2, 0, 0, 0, 3, 2, 2, 1, 3, 1, 2, 3,
       0, 3, 3, 1, 2, 0, 1, 1, 3, 3, 1, 2, 1, 3, 2, 0, 1, 3, 2, 0, 3, 0,
       1, 0, 2, 2, 3, 2, 3, 1, 1, 2, 1, 3, 2, 0, 1, 2, 1, 0, 2, 3, 1, 2,
       0, 0, 1, 3, 1, 0, 2, 2, 3, 1, 1, 2, 1, 2, 3, 3, 2, 3, 1, 1, 3,
       3, 1, 2, 0, 0, 1, 3, 1, 1, 3, 0, 3, 3, 0, 0, 1, 0, 3, 0, 2, 0,
       0, 2, 1, 1, 0, 2, 2, 2, 1, 2, 1, 3, 2, 0, 0, 3, 3, 0, 0, 3, 2, 1,
       2, 2, 2, 2, 1, 3, 1, 0, 2, 1, 2, 3, 2, 0, 0, 0, 2, 2, 3, 2, 2,
       3, 3, 3, 1, 2, 1, 1, 3, 3, 1, 2, 1, 3, 2, 2, 2, 3, 3, 0, 1, 1,
       0, 1, 1, 3, 3, 3, 2, 2, 0, 0, 0, 0, 0, 2, 3, 3, 3, 1, 3, 1, 1, 0,
       0, 0, 2, 3, 2, 2, 2, 1, 3, 0, 2, 0, 0, 0, 0, 2, 0, 2, 3, 0, 0,
       2, 1, 3, 3, 0, 0, 1, 3, 3, 0, 3, 3, 2, 2, 2, 0, 0, 0, 1, 3, 0,
       3, 1, 2, 1, 0, 1, 3, 1, 1, 0, 1, 1, 2, 2, 1, 2, 1, 3, 1, 3, 2,
       2, 3, 0, 2, 2, 1, 2, 1, 2, 0, 3, 0, 3, 2, 1, 2, 1, 2, 2, 0, 2,
       1, 3, 3, 2, 3, 1, 1, 3, 2, 3, 2, 0, 1, 3, 3, 0, 0, 2, 1, 2, 2, 1,
       3, 2, 3, 1, 3, 0, 0, 2, 0, 2, 1, 3, 1, 2, 0, 0, 0, 3, 0, 3, 0, 0,
       3, 2, 2, 0, 3, 1, 1, 1, 2, 0, 0, 0, 1, 0, 2, 1, 0, 0, 0, 3, 1, 1,
       3, 2, 0, 2, 2, 2, 0, 2, 0, 2, 0, 1, 0, 1, 2, 1, 3, 0, 2, 2, 2,
```

```
0, 2, 3, 1, 0, 1, 3, 1, 1, 3, 0, 1, 0, 0, 1, 2, 1, 0, 2, 3, 0, 0,
0, 0, 2, 3, 2, 3, 1, 0, 3, 3, 2, 0, 3, 3, 2, 1, 0, 3, 3, 1, 1, 0,
2, 0, 2, 1, 2, 2, 2, 3, 3, 1, 2, 1, 0, 2, 3, 2, 3, 2, 1, 0, 3,
1, 2, 3, 2, 0, 1, 3, 2, 0, 3, 3, 2, 1, 1, 2, 3, 2, 3, 0, 2,
2, 3, 0, 3, 0, 3, 1, 1, 1, 2, 0, 1, 2, 1, 3, 3, 1, 0, 1, 0, 0, 1,
3, 3, 3, 1, 0, 2, 2, 0, 3, 2, 0, 3, 1, 2, 3, 1, 1, 3, 3, 1, 2, 2,
2, 0, 1, 3, 3, 0, 3, 0, 0, 0])
```

In [47]:

```
#obtenemos el total de registros
categorias_codificadas.shape
```

Out[47]:

```
(1000,)
```

Aquí utilizamos el metodo resape en la lista obtenida de categorías_codificadas y para convertirla en una lista de listas, ya que el codificador de **OneHotEncoder** en su método fit_transform solo admite esto como parámetro.

In [48]:

```
categorias_oh_codificadas = ohcodificador.fit_transform(categorias_codificadas.reshape(1000,1))
categorias_oh_codificadas
```

Out[48]:

```
<1000x4 sparse matrix of type '<class 'numpy.float64'>'  
with 1000 stored elements in Compressed Sparse Row format>
```

Lo que nos devuelve el OneHotEncoder es más conocido como "matriz escasa" (sparse matrix) y es una manera de representar matrices con muchos ceros para consumir muy poca memoria. esta variable se puede convertir a una estructura de array de la siguiente forma:

In [49]:

```
categorias_oh_codificadas.toarray()
```

Out[49]:

```
array([[0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [0., 0., 0., 1.],
       ...,
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.]])
```

Eficiencia

Ahora realizaremos una comparación en memoria de una matriz escasa y su correspondiente np.array usando la función **sys.getsizeof** que nos devuelve el uso de memoria de un objeto de python en bytes.

In [50]:

```
import sys
#Obtenemos el espacio de memoria en bytes de la matriz sparse
sys.getsizeof(categorias_oh_codificadas)
```

Out[50]:

```
56
```

In [51]:

```
#obtenemos el espacio de memoria en bytes del array de la matriz
sys.getsizeof(categorias_oh_codificadas.toarray())
```

Out[51]:

```
32112
```

Podemos comprobar que es mucho más eficiente una matriz escasa antes que el array. Igualmente si queremos crear un transformador OneHotEncoder y almacenarlo en un np.array podemos hacerlo asignandole al parámetro sparse que sea igual a *False*

In [52]:

```
oh_codificador = preprocessing.OneHotEncoder(sparse=False)
categorias_oh_codificadas = oh_codificador.fit_transform(categorias_codificadas.reshape(1000,1))
categorias_oh_codificadas

/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py:371: FutureWarning:
The handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, t
hen you can now use the OneHotEncoder directly.
    warnings.warn(msg, FutureWarning)
```

Out[52]:

```
array([[0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [0., 0., 0., 1.],
       ...,
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.]])
```

In [53]:

```
oh_codificador.feature_indices_

/opt/anaconda3/lib/python3.6/site-packages/sklearn/utils/deprecation.py:77: DeprecationWarning: Fu
nction feature_indices_ is deprecated; The ``feature_indices_`` attribute was deprecated in
version 0.20 and will be removed 0.22.
    warnings.warn(msg, category=DeprecationWarning)
```

Out[53]:

```
array([0, 4])
```

Pandas tiene la función auxiliar **get_dummies** que hace esto automáticamente de forma más fácil

In [54]:

```
pd.get_dummies(datos.col_categorica).head()
```

Out[54]:

	elefante	gato	perro	ratón
0	0	0	0	1
1	1	0	0	0
2	0	0	0	1
3	0	1	0	0
4	0	1	0	0

Texto

Veremos como vectorizar texto con el paquete **feature_extraction** de **sklearn**. Aplicaremos el ejemplo sobre la columna **col_texto** que incluye frases del Quijote.

In [55]:

```
from sklearn import feature_extraction
datos.col_texto.values[:5]
```

Out[55]:

```
array(['Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los ve
inte, y un mozo de campo y plaza, que así ensillaba el rocin como tomaba la podadera.',
       'El resto della concluian sayo de velarte, calzas de velludo para las fiestas con sus
       pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.',
       'El resto della concluian sayo de velarte, calzas de velludo para las fiestas con sus
       pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.',
       'Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sá
bados, lentejas los viernes, algún palomino de añadidura los domingos, consumian las tres partes d
e su hacienda.'],
```

```
'Tenía en su casa una ama que pasaba de los cuarenta, y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza, que así ensillaba el rocín como tomaba la podadera.',  
dtype=object)
```

Para convertir texto en variables numéricas, se puede proceder de igual forma que con las variables categóricas, simplemente separando las palabras antes. Para ello se utilizarán **vectorizadores** de sklearn que convierten el texto en vectores.

CountVectorizer devuelve un vector con el valor 0 en todas las palabras que no existen en una frase y con el número de ocurrencias de las palabras que si existen.

Para que los conceptos queden claros, vamos a realizar un ejemplo que ejemplifique este concepto de forma más sencilla

In [56]:

```
ejemplo_frases=['los coches son rojos',  
                 'los aviones son rojos',  
                 'los coches y los aviones son rojos',  
                 'los camiones son rojos'  
                ]  
vectorizador_count=feature_extraction.text.CountVectorizer()  
x= vectorizador_count.fit_transform(ejemplo_frases)  
x
```

Out[56]:

```
<4x6 sparse matrix of type '<class 'numpy.int64'>'  
with 17 stored elements in Compressed Sparse Row format>
```

In [57]:

```
pd.DataFrame(x.toarray(),columns=vectorizador_count.get_feature_names())
```

Out[57]:

	aviones	camiones	coches	los	rojos	son
0	0	0	1	1	1	1
1	1	0	0	1	1	1
2	1	0	1	2	1	1
3	0	1	0	1	1	1

Esto realmente provoca un problema, y es que da mayor peso a aquellas palabras que aparecen muchas veces pero que no aporta valor semántico (por ejemplo, son o los).

Para ello podemos utilizar el vectorizador TfidfVectorizer() que realmente se encarga de medir la frecuencia del texto ya que es una medida que asigna valores a cada palabra en función de la frecuencia de aparición en todos los documentos.

In [58]:

```
vectorizador_tfidf = feature_extraction.text.TfidfVectorizer()
```

In [59]:

```
x = vectorizador_tfidf.fit_transform(ejemplo_frases)  
pd.DataFrame(x.toarray(),columns=vectorizador_tfidf.get_feature_names())
```

Out[59]:

	aviones	camiones	coches	los	rojos	son
0	0.000000	0.000000	0.657341	0.435087	0.435087	0.435087
1	0.657341	0.000000	0.000000	0.435087	0.435087	0.435087
2	0.464810	0.000000	0.464810	0.615306	0.307653	0.307653
3	0.000000	0.74187	0.000000	0.387139	0.387139	0.387139

Lo que ha hecho ha sido asignar 'pesos' de aparición en el texto. En el ejemplo vemos 'camiones' que la considera de más peso aunque solo aparece una vez.

In [8]:

```
%reload_ext watermark  
%watermark
```

2019-05-30T21:35:03+02:00

Cython 3.6.5
IPython 6.4.0

```
compiler : GCC 7.2.0  
system : Linux  
release : 5.1.5-arch1-2-ARCH  
machine : x86_64  
processor :  
CPU cores : 4  
interpreter: 64bit
```

Procesado de DataSet

Con lo visto en el apartado anterior de procesado de variables vamos a transformar un dataset en otro que pueda ser utilizado para entrenar modelos de predicción.

Ingesta de datos

Cargamos las librerías que vamos a utilizar y el dataset original

In [1]:

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn import preprocessing  
from sklearn import feature_extraction
```

In [2]:

```
datos = pd.read_csv("../..//RESOURCES/datos_procesamiento.csv")  
datos.head()
```

Out [2]:

	col_inexistente1	col2	col3	col_outliers	col_outliers2	col_categorica	col_ordinal	col_texto
0	59.0	52.0	2.232832	-50	0.771666	ratón	muy bien	Tenía en su casa una ama que pasaba de los cua...
1	31.0	74.0	0.906147	-5	1.068558	elefante	regular	El resto della concluían sayo de velarte, calz...
2	81.0	28.0	0.626750	-32	0.846396	ratón	muy mal	El resto della concluían sayo de velarte, calz...
3	34.0	16.0	0.816738	-84	0.637381	gato	mal	Una olla de algo más vaca que carnero, salpicó...
4	32.0	28.0	0.571131	65	4.540614	gato	bien	Tenía en su casa una ama que pasaba de los cua...

Transformación del DataSet

Separación de variables

In [3]:

```
col_numericas = ['col_inexistente1', 'col2', 'col3', 'col_outliers', 'col_outliers2']  
col_categorica = ['col_categorica']  
col_texto = ['col_texto']
```

Variables numéricas

In [4]:

```
imputador = preprocessing.Imputer(strategy="mean")
escalador = preprocessing.StandardScaler()
var_numericas_imputadas_escalado_standard = escalador.fit_transform(
    imputador.fit_transform(datos[col_numericas]))
df_numericos_procesados = pd.DataFrame(var_numericas_imputadas_escalado_standard,
                                         columns=col_numericas)

/opt/anaconda3/lib/python3.6/site-packages/sklearn/utils/deprecation.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22.
Import impute.SimpleImputer from sklearn instead.
warnings.warn(msg, category=DeprecationWarning)
```

Variables Categóricas

In [5]:

```
label_codificador_categorico = preprocessing.LabelEncoder()
categorias_codificadas = label_codificador_categorico.fit_transform(datos[col_categorica])
oh_codificador = preprocessing.OneHotEncoder(sparse=False)
categorias_oh_codificadas = oh_codificador.fit_transform(categorias_codificadas.reshape(1000,1))

df_categorico_procesado = pd.DataFrame(categorias_oh_codificadas,
                                         columns=label_codificador_categorico.classes_)

/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/label.py:235:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
/opt/anaconda3/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py:371: FutureWarning:
The handling of integer data will change in version 0.22. Currently, the categories are determined
based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then
you can now use the OneHotEncoder directly.
warnings.warn(msg, FutureWarning)
```

Variables de Texto

In [6]:

```
vectorizador_tfidf = feature_extraction.text.TfidfVectorizer()
texto_vectorizado = vectorizador_tfidf.fit_transform(datos.col_texto)
df_texto_procesado = pd.DataFrame(texto_vectorizado.toarray(),
                                   columns=vectorizador_tfidf.get_feature_names())
```

Exportación y muestra final

In [7]:

```
datos_procesados = pd.concat([
    df_numericos_procesados,
    df_categorico_procesado,
    df_texto_procesado
], axis=1)

label_codificador_ordinal = preprocessing.LabelEncoder()
datos_procesados['col_ordinal'] = label_codificador_ordinal.fit_transform(datos.col_ordinal)
datos_procesados.head()
```

Out[7]:

	col_inexistente1	col2	col3	col_outliers	col_outliers2	elefante	gato	perro	ratón	acordarme	...	vaca	...
0	0.399217	0.082807	0.442819	-0.694600	-0.038365	0.0	0.0	0.0	1.0	0.0	...	0.000000	0.2
1	-0.653605	0.861333	-0.323390	-0.118466	-0.038278	1.0	0.0	0.0	0.0	0.0	...	0.000000	0.0

2	col2	col3	col4_outliers	col3_outliers2	elefante	gato	perro	ratón	acordarme	...	0.000000	0.0
	-0.766494	-0.484752										
3	-0.540803	-1.191145	0.375028	-1.129901	-0.038405	0.0	1.0	0.0	0.0	...	0.194272	0.0
4	-0.616004	-0.766494	-0.516874	0.777743	-0.037257	0.0	1.0	0.0	0.0	...	0.000000	0.2

5 rows × 144 columns

In [17]:

```
datos_procesados.to_csv("../..../RESOURCES/dataset_procesado.csv")
```