https://github.com/lulimontero/curso-udemy-data-science-python

Section 1-2-3

Anaconda

Anaconda es una distribución de Python, es decir que no solo incluye Python sino también muchas bibliotecas y herramientas, además de su propio sistema de entornos virtuales. Básicamente es una instalación “todo en uno” que es muy popular en data science y machine learning.

Jupyter

Jupyter es un entorno de desarrollo en el que podemos escribir el código normal de Python pero también nos permite mostrar imágenes y escribir notas. Es el IDE más popular en data science para explorar y analizar datos.

Instalación

www.anaconda.com

Terminal

cd Downloads ➜ te mete en esa carpeta
cd .. ➜ te lleva atrás de esa carpeta
ls ➜ ver lista de los archivos de tu directorio (carpetas)
clear ➜ borra todo

Atajos Notebook (Anaconda)

- Shift + Enter: Ejecuta lo que esté en la celda
- Alt + Enter: La primera vez ejecuta lo que esté en la celda, la segunda vez que lo volvés a correr agrega una celda vacía abajo

Entornos Virtuales

Los entornos virtuales te permiten configurar instalaciones virtuales de Python y librerías en la computadora. Podés tener múltiples versiones de Python o librerías y fácilmente activar o desactivar esos entornos.

- Por ejemplo, desarrollás un programa con SciKit-Learn 0.17 y sale el 0.18. Vos querés explorar el 0.18 pero no querés que se rompa tu código viejo ➜ podés crear un entorno virtual que corra esa versión específica de la librería
- Otro ejemplo, querés tener múltiples versiones de Python en tu computadora, querés un ambiente con Python 2.7 y otro con Python 3.5

Existe una librería virtualenv para distribuciones normales de Python, que permite crear entornos virtuales y es un administrador de entorno virtual. Anaconda tiene incorporado un manager de ambientes que hace que todo el proceso de creación de un entorno virtual sea

muy facil y también se puede consultar esta documentación:
https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html
https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Section 4: Python Crash Course

Tipos de datos

- Entero: 1
- De punto flotante: 1.0
- String

Variables

El nombre no puede empezar con un número ni con un símbolo especial

Comentarios

Los comentarios se ponen con #

Impresión de resultados

- print(x)

- num = 12
  name = 'Sam'

- print('My number is: {one}, and my name is: {two}'.format(one=num,two=name))
  My number is: 12, and my name is: Sam

- print('My number is: {}, and my name is: {}'.format(num,name))
- My number is: 12, and my name is: Sam

Listas

Una lista es una secuencia de elementos dentro de corchetes separados por comas que pueden tomar básicamente cualquier tipo de datos ej [1, 2, 3] o [‘a’,’ b’, ’c’]

- my_list = [‘a’,’ b’, ’c’]
- Para agregar un elemento ➔ my_list.append(‘d’)
- my_list[1,2] ➔ devuelve [‘b’, ‘c’]
- Para cambiar un elemento ➔ my_list[0]=’NEW’
- my_list ➔ devuelve [‘NEW’, ‘b’, ‘c’, ‘d’]
- Se puede colocar una lista dentro de otra, ej: list = [1,2,[3,4]]
- lst = [1,2,[3,4],[5,[100,200,['hello']],23,11],1,7]
  lst[3][1][2] #Recordar que empieza en el 0
  ['hello']

Diccionarios

Los diccionarios a diferencia de las listas no conservan orden sino que solo son asignaciones de pares de valores claves (si consultas un diccionario d[0] da error, xq no guarda posiciones).

- d = { ‘key1’ : ‘value’ }
- d = { ‘key1’ : ‘value’ , ‘key2’ : 123 }
- d[0] ➜ no devuelve nada (error)
- En el diccionario le podés pasar de valor una lista, x ejemplo d1 = { ‘key1’ : [1,2,3] }
- d1[‘key1’][0] ➜ devuelve 1
- También podés asignar my_list = d1[‘key1’]
- Luego hacer my_list[1] ➜ devuelve 2
- d = {'k1' : {'innerkey' : [1,2,3]} }
- d['k1']['innerkey'][1] ➜ devuelve 2
- d = {'k1':[1,2,3,{'tricky':['oh','man','inception',{'target':[1,2,3,'hello']}]}]}
  d['k1'][3]['tricky'][3]['target'][3]
  'hello'

Tuplas

Una tupla es muy similar a una lista pero usa paréntesis en lugar de corchetes. Además otra diferencia es que las tuplas son inmutables es decir que no se pueden cambiar, no admiten la asignación de elementos. Por ejemplo va a devolver error si intentas hacer t[0] = ‘NEW’. Vas a usar una tupla cuando te quieras asegurar de que un usuario no pueda cambiar los elementos dentro de la secuencia.

- t = (1, 2, 3)
- t[0] ➜ devuelve 1
- t[0] = ‘NEW’ ➜ devuelve error xq la tupla es inmutable, no se puede cambiar

Set / Colección

Un set es un conjunto de elementos **únicos**

- {1,2,3} ➜ devuelve {1,2,3}
- {1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2} ➜ devuelve {1,2,3}
- set([1,1,1,2,2,2,2,3,4,4,4]) ➜ devuelve {1, 2, 3, 4}
- s={1,2,3} ➜ quiero agregar el 5 ➜ s.add(5) ➜ si corro s devuelve {1,2,3,5}

Operadores de comparación

- 1>2 ➜ devuelve False
- 1 >= 1 ➜ devuelve True
- 1 <= 4 ➜ devuelve True
- 1 == 1 ➜ devuelve True #Con un solo signo = es asignación de variables
- 1 != 3 ➜ devuelve True
- and
- or

If statement

```
if 1 < 2:
    print('Yep!')

if True:
    print('perform code')
    #Si es verdadero se ejecuta esto

if 1 == 2:
    print('first')
elif 3 == 3:
    print('middle')
else:
    print('Last')
```

Secuencias con listas

```
seq = [1,2,3,4,5]

for item in seq:
    print(item)
1
2
3
4
5

for num in seq:
    print('Yep')
Yep
Yep
Yep
Yep
Yep

for jelly in seq:
    print(jelly+jelly)
2
4
6
8
10
```

While loops

```
i = 1
while i < 5:
    print('i is: {}'.format(i))
    i = i+1
```

```
i is: 1
i is: 2
i is: 3
i is: 4
```

Rango / Range

Es un generador de valores numéricos; range (0,5) ➜ ponés el número con el que querés empezar y terminar (no incluye el de terminar, ej en este caso va a incluir del 0 al 4 que son 5 elementos). Si no ponés inicio toma desde el 0.

- range(5) ➜ devuelve range (0,5)
- for i in range(**5**):
  print(i)
  0
  1
  2
  3
  4
- list (range(0,5))
  [0, 1, 2, 3, 4]

List comprehension

Es una especie de loop pero al revés, arrancás diciendo lo que querés obtener para tal condición

```
x = [1, 2, 3, 4]
```

EN LUGAR DE TENER QUE HACER ESTO:

```
out = []
for item in x:
    out.append(item**2)
print(out)
```

HACES DIRECTAMENTE ESTO:

```
[item**2 for item in x]
[1, 4, 9, 16]
```

También se lo podés asignar a una variable ej: out = [item**2 for item in x]

Functions

Definición de la función:

```
def my_func(param1='default'):
    """
    Docstring goes here.
    """
    print(param1)
```

También se puede escribir:

```
def my_func(param1):print(param1)
```

Uso de la función:

```
my_func(param1='new param')
```

new param

```
my_func('new param')
```

new param

Otro ejemplo:

```
def square(x):
    return x**2

out = square(2)

print(out)
```

4

Otro ejemplo:

```
def my_func2(name):
    print ('Hello '+name)

my_func2('José')
```

Hello José

Otro ejemplo:

```
def domainGet(email):
    return email.split('@')[1]

domainGet('user@domain.com')
```

'domain.com'

Otro ejemplo:

```
def findDog(st):
    return 'dog' in st.lower().split()
```

findDog('Is there a dog here?')
True

Otro ejemplo:

```
def countDog(frase):
    i=0
    s=frase.lower()
    ss=s.split()
    for word in ss:
        if word =='dog':
            i=i+1
    return i

countDog('This dog runs faster than the other dog dude!')
```

2

Otro ejemplo:

```
def caught_speeding(speed, is_birthday):
    if is_birthday:
        speeding = speed - 5
    else:
        speeding = speed

    if speeding > 80:
        return 'Big Ticket'
    elif speeding > 60:
        return 'Small Ticket'
    else:
        return 'No Ticket'

caught_speeding(81,True)
```

'Small Ticket'

Lambda expressions

Es otra forma de escribir las funciones pero más abreviada.

```
def times2(var):
    return var*2
```

También se puede escribir:

```
def times2(var):return var*2
```

También:

```
lambda var:var*2
```

Ejemplo:
t=lambda var:var*2
t(2)
4

Otro ejemplo:
seq = ['soup','dog','salad','cat','great']
list(filter(lambda word: word[0]=='s',seq))
['soup', 'salad']

Map

Le tenés que pasar como input una función y una secuencia. Lo que hace es aplicarle a esa función los parámetros que le pasas en la secuencia.

```
def times2(var):
    return var*2

times2(5)
10

seq = [1,2,3,4,5]

map(times2, seq)
<map at 0x74d83c1b2c80>
```

Si querés ejecutar el mapa y obtener el resultado como una lista:
list(map(times2,seq)**)**
[2, 4, 6, 8, 10]

Si querés hacerlo con lambda:
list(map(lambda var: var*2,seq))
[2, 4, 6, 8, 10]

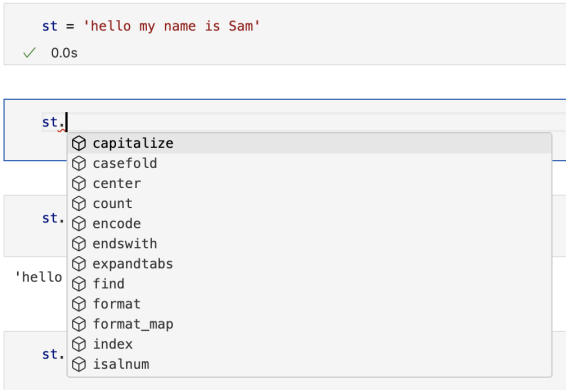
Filter

filter(lambda item: item%2 == 0,seq)
<filter at 0x105316ac8>

list(filter(lambda item: item%2 == 0,seq))
[2, 4]

Methods

Son funciones predefinidas que vienen en Python:



Métodos para cadenas

st = 'hello my name is Sam'
st.lower()
'hello my name is sam'
st.upper()
'HELLO MY NAME IS SAM'
st.split()
['hello', 'my', 'name', 'is', 'Sam']

tweet = 'Go Sports! #Sports #Friends'
tweet.split('**#**')
['Go Sports! ', 'Sports ', 'Friends']
tweet.split('#')[1]
'Sports'

Métodos para diccionarios

d = {'k1': 1, 'k2': 2} #Definimos un diccionario
d
{'k1': 1, 'k2': 2}
d.keys()
dict_keys(['k1', 'k2'])
d.items()
dict_items([('k1', 1), ('k2', 2)])
d.values()
dict_values([1, 2])

Métodos para listas

lst = [1,2,3]
lst.pop() #Este método eliminará y devolverá el último elemento de la lista
3

```
lst
        [1, 2]
lst = [1,2,3,4,5]
item=lst.pop()
item
        4
lst
        [1, 2, 3, 4]
first = lst.pop(0)
first
        1
lst
        [2,3,4]
lst.append('new')
lst
        [2,3,4,'new']
'x' in [1,2,3] #Se fija si 'x' está en la lista
        False
'x' in ['x','y','z']
        True
```

Métodos para tuplas

```
x = [(1,2),(3,4),(5,6)]
x
        [(1, 2), (3, 4), (5, 6)]
x[0]
        (1,2)
x[0][0]
        1
for item in x:
    print(item)
        (1, 2)
        (3, 4)
        (5, 6)
for (a,b) in x:
    print(a)
        1
        3
        5
```

Section 5: NumPy

Introducción a NumPy

NumPy (o Numpy) es una librería de álgebra lineal para Python. NumPy (or Numpy) is a Linear Algebra Library for Python, the reason it is so important for Data Science with Python is that almost all of the libraries in the PyData Ecosystem rely on NumPy as one of their main building blocks.

Usar Numpy (antes hay que instalarlo)

import numpy as np

Numpy Arrays

NumPy arrays are the main way we will use Numpy throughout the course. Numpy arrays essentially come in two flavors: **vectors and matrices**. **Vectors are strictly 1-d arrays** and **matrices are 2-d** (but you should note a matrix can still have only one row or one column).

Creating NumPy Arrays from a Python List

We can create an array by directly converting a list or list of lists:

```
my_list = [1,2,3]
np.array(my_list)
        array([1, 2, 3])

my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
np.array(my_matrix)
        array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

Built-In Methods

There are lots of built-in ways to generate Arrays

arange: Return evenly spaced values within a given interval.

```
np.arange(0,10)
        array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.arange(0,11,2)
        array([ 0,  2,  4,  6,  8, 10])
np.arange(10,51,2)
        array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
            44, 46, 48, 50])
```

zeros and ones: Generate arrays of zeros or ones

```
np.zeros(3)
```

array([0., 0., 0.])

np.zeros((5,5)) #El 1er numero es la cantidad de filas

array([[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0.]])

np.ones(3)

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

np.ones((3,3))

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

linspace: Return evenly spaced numbers over a specified interval.

np.linspace(0,10,3) #Te devuelve 3 numeros entre el 0 y el 10 igualmente espaciados

array([0.,   5., 10.])

np.linspace(0,10,50)

```
array([  0.        ,   0.20408163,   0.40816327,   0.6122449 ,
  0.81632653,   1.02040816,   1.2244898 ,   1.42857143,
  1.63265306,   1.83673469,   2.04081633,   2.24489796,
  2.44897959,   2.65306122,   2.85714286,   3.06122449,
  3.26530612,   3.46938776,   3.67346939,   3.87755102,
  4.08163265,   4.28571429,   4.48979592,   4.69387755,
  4.89795918,   5.10204082,   5.30612245,   5.51020408,
  5.71428571,   5.91836735,   6.12244898,   6.32653061,
  6.53061224,   6.73469388,   6.93877551,   7.14285714,
  7.34693878,   7.55102041,   7.75510204,   7.95918367,
  8.16326531,   8.36734694,   8.57142857,   8.7755102 ,
  8.97959184,   9.18367347,   9.3877551 ,   9.59183673,
  9.79591837,  10.        ])
```

eye: Creates an identity matrix

np.eye(4)

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

Random

Numpy also has lots of ways to create random number arrays:

rand: Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

np.random.rand(2)

array([0.11570539, 0.35279769])

np.random.rand(5,5)

array([[0.66660768, 0.87589888, 0.12421056, 0.65074126, 0.60260888],
[0.70027668, 0.85572434, 0.8464595 , 0.2735416 , 0.10955384],
[0.0670566 , 0.83267738, 0.9082729 , 0.58249129, 0.12305748],
[0.27948423, 0.66422017, 0.95639833, 0.34238788, 0.9578872],
[0.72155386, 0.3035422 , 0.85249683, 0.30414307, 0.79718816]])

randn: Return a sample (or samples) from the "standard normal" distribution. Unlike rand which is uniform.

np.random.randn(2)

array([-0.27954018, 0.90078368])

np.random.randn(5,5)

array([[0.70154515, 0.22441999, 1.33563186, 0.82872577, -0.28247509],
[0.64489788, 0.61815094, -0.81693168, -0.30102424, -0.29030574],
[0.8695976 , 0.413755 , 2.20047208, 0.17955692, -0.82159344],
[0.59264235, 1.29869894, -1.18870241, 0.11590888, -0.09181687],
[-0.96924265, -1.62888685, -2.05787102, -0.29705576, 0.68915542]])

randint: Return random integers from `low` (inclusive) to `high` (exclusive).

np.random.randint(1,100)

44 #Obtenes un nro random entre 1 y 99 incluisve

np.random.randint(1,100,10)

array([13, 64, 27, 63, 46, 68, 92, 10, 58, 24]) #Obtenes 10 nros random entre 1 y 99 incluisve

librería:

from numpy.random import randint

randint(2,10)

3

Attributes and Methods

arr = np.arange(25)

arr

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])

ranarr = np.random.randint(0,50,10)

ranarr

array([10, 12, 41, 17, 49, 2, 46, 3, 19, 39])

reshape: Returns an array containing the same data with a new shape.

arr.reshape(5,5)

array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14],

[15, 16, 17, 18, 19],
[20, 21, 22, 23, 24]])

max,min,argmax,argmin: These are useful methods for finding max or min values. Or to find their index locations using argmin or argmax

ranarr.max()

49

ranarr.argmax()

4 #En qué posición está el max, empezando en 0

ranarr.min()

2

ranarr.argmin()

5 #En qué posición está el min, empezando en 0

Shape: Shape is an attribute that arrays have (not a method):

arr

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])

arr.shape

(25,)

arr.**reshape**(5,5)

array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19],
[20, 21, 22, 23, 24]])

arr.reshape(1,25)

array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24]])
 # Notice the two sets of brackets

arr.reshape(1,25).**shape**

(1, 25)

arr.reshape(25,1)

array([[0],
[1],
[2],
[3],
[4],
[5],

[6],
[7],
[8],
[9],
[10],
[11],
[12],
[13],
[14],
[15],
[16],
[17],
[18],
[19],
[20],
[21],
[22],
[23],
[24]])

arr.reshape(25,1).shape

(25,1)

dtype: You can also grab the data type of the object in the array:

arr.dtype

dtype('int64')

Numpy Indexing and Selection

```
import numpy as np
#Creating sample array
arr = np.arange(0,11)
arr
```

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

Bracket Indexing and Selection

The simplest way to pick one or some elements of an array looks very similar to python lists:

```
#Get a value at an index, te devuelve el valor q esta en esa posicion
arr[8]
```

8

```
#Get values in a range, te devuelve el valor q esta en esa posicion
arr[1:5]
```

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

#Get values in a range, te devuelve el valor q esta en esa posicion
arr[0:5]
array([0, 1, 2, 3, 4])
arr[:5]
array([0, 1, 2, 3, 4])

Broadcasting

Numpy arrays differ from a normal Python list because of their ability to broadcast

#Lo que hace es a los valores en la posicion 0,1,2,3 y 4 cambiarlos a que valgan 100
arr[0:5]=100
arr
*array([**100, 100, 100, 100, 100**, 5, 6, 7, 8, 9, 10])*

# Reset array, we'll see why I had to reset in a moment
arr = np.arange(0,11)
arr
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
#Important notes on Slices
slice_of_arr = arr[0:6]
slice_of_arr
array([0, 1, 2, 3, 4, 5])
#Change Slice
slice_of_arr[:]=99
slice_of_arr
array([99, 99, 99, 99, 99, 99])
#Now note the changes also occur in our original array!
arr
array([99, 99, 99, 99, 99, 99, 6, 7, 8, 9, 10])

#To get a copy, need to be explicit
arr_copy = arr.**copy**()
arr_copy
array([99, 99, 99, 99, 99, 99, 6, 7, 8, 9, 10])

Indexing a 2D array (matrices)

The general format is **arr_2d[row][col]** or **arr_2d[row,col]**. I recommend usually using the comma notation for clarity.

#Definimos la matriz

```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr_2d
        array([[ 5, 10, 15],
                [20, 25, 30],
                [35, 40, 45]])
#Indexing row, te devuelve los valores de esa fila (arranca en el 0)
arr_2d[1]
        array([20, 25, 30])
arr_2d[:2] #Agarra todo menos la fila 2 (que es la ultima xq arranca en 0)
        array([[ 5, 10, 15],
                [20, 25, 30]])
arr_2d[1:] #Agarra todo desde la fila 1 (arranca en 0 recordar)
        array([[20, 25, 30],
                [35, 40, 45]])
arr_2d[2,:]
        array([35, 40, 45])

# Format is arr_2d[row][col] or arr_2d[row,col]
# Getting individual element value, obtenes el valor en la posición 0 de la fila 1(en realidad la
2 xq empieza en 0)
arr_2d[1][0]
        20
arr_2d[1,0] #Otra manera de escribirlo
        20

# 2D array slicing
#Shape (2,2) from top right corner
arr_2d[:2,1:]
        array([[10, 15],
                [25, 30]])

#Shape bottom row
arr_2d[2]
        array([35, 40, 45])

#Set up matrix
arr2d = np.zeros((10,10))
arr2d
        array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
```

```
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
arr_length = arr2d.shape[1]
arr_length
            10
range(arr_length)
            range(0,10)
for i in range(arr_length):
    arr2d[i] = i
arr2d
            array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])

arr2d[[2,4,6,8]]
            array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])

arr2d[[6,4,2,7]]
            array([[6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])

mat = np.arange(1,26).reshape(5,5)
mat
            array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
mat[:3,1:2]
            array([[ 2],
                   [ 7],
                   [12]])
```

Conditional Selection

```
arr = np.arange(1,11)
arr
```

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

arr>4

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

```
bool_arr = arr>4
bool_arr
```

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

arr[bool_arr]

array([5, 6, 7, 8, 9, 10]) #Solo obtenes resultados dnd era true

arr[arr>2]

array([3, 4, 5, 6, 7, 8, 9, 10])

```
x = 2
arr[arr>x]
```

array([3, 4, 5, 6, 7, 8, 9, 10])

Numpy Operations

Array with Array and Array With Scalars

```
arr = np.arange(0,10)
arr
```

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

```
arr+arr
```

array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])

```
arr*arr
```

array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])

```
arr-arr
```

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

```
arr-100
```

array([-100, -99, -98, -97, -96, -95, -94, -93, -92, -91])

```
# Warning on division by zero, but not an error!
# Just replaced with nan
arr/arr
```

array([nan, 1., 1., 1., 1., 1., 1., 1., 1., 1.])

```
# Also warning, but not an error instead infinity
1/arr
```

array([inf, 1.   , 0.5   , 0.33333333, 0.25   , 0.2   , 0.16666667, 0.14285714, 0.125   , 0.11111111])

```
arr**3
```

array([0, 1, 8, 27, 64, 125, 216, 343, 512, 729])

Universal Array Functions

Documentación

#Taking Square Roots
np.**sqrt**(arr)
array([0. , 1. , 1.41421356, 1.73205081, 2. ,
2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.])
#Calcualting exponential (e^)
np.**exp**(arr)
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00,
2.00855369e+01, 5.45981500e+01, 1.48413159e+02,
4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
8.10308393e+03])
np.**max**(arr) #same as arr.max()
9
np.**sin**(arr)
array([0. , 0.84147098, 0.90929743, 0.14112001, -0.7568025 ,
-0.95892427, -0.2794155 , 0.6569866 , 0.98935825, 0.41211849])
np.**log**(arr)
array([-inf, 0. , 0.69314718, 1.09861229, 1.38629436,
1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
mat.**sum**() #Devuelve la suma de los elementos de la matriz
325
mat.**std**() #Devuelve el desvio estandar de los elementos de la matriz
7.211102550927978
mat.s**um(axis=0)** #Devuelve la suma de cada columna de la matriz
array([55, 60, 65, 70, 75])

Section 6: Python for Data Analysis - Pandas

Introducción a Pandas

Pandas es una librería pública construida con las bases de Numpy que permite un rápido análisis, limpieza y preparación de los datos, sobresale en rendimiento y productividad. Tiene funciones de visualización integradas y puede trabajar con datos de diferentes fuentes.

Series

A Series is very similar to a NumPy array (in fact it is built on top of the NumPy array object). What differentiates the NumPy array from a Series, is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. It also doesn't need to hold numeric data, it can hold any arbitrary Python Object.

```
import numpy as np
import pandas as pd
labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}
```

Crear una serie

You can convert a list,numpy array, or dictionary to a Series:

```
#LISTAS
my_list = [10,20,30]
pd.Series(data=my_list)
```

0 10
1 20
2 30
dtype: int64

```
labels = ['a','b','c']
pd.Series(data=my_list,index=labels)
```

a *10*
b *20*
c *30*
dtype: int64

```
pd.Series(my_list,labels)
```

a 10
b 20
c 30
dtype: int64

```
#ARRAYS
arr = np.array([10,20,30])
```

```
pd.Series(arr)
    0   10
    1   20
    2   30
    dtype: int64
pd.Series(arr,labels)
    a   10
    b   20
    c   30
    dtype: int64

#DICTIONARY
d = {'a':10,'b':20,'c':30}
pd.Series(d)
    a   10
    b   20
    c   30
    dtype: int64
```

Data en una serie

A pandas Series can hold a variety of object types:

```
pd.Series(data=labels)
    0   a
    1   b
    2   c
    dtype: object
# Even functions (although unlikely that you will use this)
pd.Series([sum,print,len])
    0      <built-in function sum>
    1   <built-in function print>
    2      <built-in function len>
    dtype: object
```

Index

The key to using a Series is understanding its index. Pandas makes use of these index names or numbers by allowing for fast look ups of information (works like a hash table or dictionary).

```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
ser1
    USA       1
    Germany   2
    USSR      3
    Japan     4
```

dtype: int64

```
ser2 = pd.Series([1,2,5,4],index = ['USA', 'Germany','Italy', 'Japan'])
ser2
```

USA 1
Germany 2
Italy 5
Japan 4
dtype: int64

```
ser1['USA']
```

1

```
#Operations are then also done based off of index:
ser1 + ser2
```

Germany 4.0
Italy NaN
Japan 8.0
USA 2.0
USSR NaN
dtype: float64

Data Frames

We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic!

```
import pandas as pd
import numpy as np
from numpy.random import randn
np.random.seed(101)
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Indexing and Selection

```
df[W]
```

A 2.706850
B 0.651118
C -2.018168
D 0.188695

E 0.190794

Name: W, dtype: float64

Pass a list of column names

df[['W','Z']]

	W	*Z*
A	*2.706850*	*0.503826*
B	*0.651118*	*0.605965*
C	*-2.018168*	*-0.589001*
D	*0.188695*	*0.955057*
E	*0.190794*	*0.683509*

SQL Syntax (NOT RECOMMENDED!)

df.W

A 2.706850

B 0.651118

C -2.018168

D 0.188695

E 0.190794

Name: W, dtype: float64

DataFrame Columns are just Series

type(df['W'])

pandas.core.series.Series

#Crear una columna nueva

df['new'] = df['W'] + df['Y']

	W	*X*	*Y*	*Z*	***new***
A	*2.706850*	*0.628133*	*0.907969*	*0.503826*	*3.614819*
B	*0.651118*	*-0.319318*	*-0.848077*	*0.605965*	*-0.196959*
C	*-2.018168*	*0.740122*	*0.528813*	*-0.589001*	*-1.489355*
D	*0.188695*	*-0.758872*	*-0.933237*	*0.955057*	*-0.744542*
E	*0.190794*	*1.978757*	*2.605967*	*0.683509*	*2.796762*

df.drop('new',axis=1)

	W	*X*	*Y*	*Z*
A	*2.706850*	*0.628133*	*0.907969*	*0.503826*
B	*0.651118*	*-0.319318*	*-0.848077*	*0.605965*
C	*-2.018168*	*0.740122*	*0.528813*	*-0.589001*
D	*0.188695*	*-0.758872*	*-0.933237*	*0.955057*
E	*0.190794*	*1.978757*	*2.605967*	*0.683509*

Not inplace unless specified!

df

	W	*X*	*Y*	*Z*	*new*
A	*2.706850*	*0.628133*	*0.907969*	*0.503826*	*3.614819*
B	*0.651118*	*-0.319318*	*-0.848077*	*0.605965*	*-0.196959*
C	*-2.018168*	*0.740122*	*0.528813*	*-0.589001*	*-1.489355*
D	*0.188695*	*-0.758872*	*-0.933237*	*0.955057*	*-0.744542*
E	*0.190794*	*1.978757*	*2.605967*	*0.683509*	*2.796762*

df.drop('new',axis=1,inplace=True) #Con esto los borras definitivamente de df

df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

df.drop('E',axis=0, inplace=True) #Elimina la fila del indice E de df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

df.loc['A'] #Elegis una fila, pero lo devuelve como columna

W 2.706850
X 0.628133
Y 0.907969
Z 0.503826
Name: A, dtype: float64

df.iloc[2] #Elegis la fila 2, que en realidad es la 3 xq arranca a contar en 0, pero lo devuelve como columna

W -2.018168
X 0.740122
Y 0.528813
Z -0.589001
Name: C, dtype: float64

df.loc['B','Y'] #Devuelve la intersección del campo Y y el índice/fila B

-0.8551960407780934

df.loc[['A','B'],['W','Y']]

	W	Y
A	-1.467514	-0.162535
B	0.392489	-0.855196

Conditional Selection

df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

df>0

	W	X	Y	Z
A	True	True	True	True

B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

df[df>0] #Devuelve los valores de dforiginal dnd se cumple que son >0

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

df['W']>0

A False
B True
C True
D True
E False
Name: W, dtype: bool

df[df['W']>0] #Devuelve todas las columnas y todas filas menos la C que es dnd no se cumple la condición

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

df[df['W']>0]['Y'] #Devuelve la columna Y y todas las filas menos la C que es dnd no se cumple la condición de los valores de W >0

A 0.907969
B -0.848077
D -0.933237
E 2.605967
Name: Y, dtype: float64

df[df['W']>0][['Y','X']] **#Devuelve las columnas Y y X, y todas las filas menos la C que es dnd no se cumple la condición de los valores de W >0**

	Y	*X*
A	*0.907969*	*0.628133*
B	*-0.848077*	*-0.319318*
D	*-0.933237*	*-0.758872*
E	*2.605967*	*1.978757*

#For two conditions you can use | and & with parenthesis:

df[(df['W']>0) & (df['Y'] > 1)]

	W	*X*	*Y*	*Z*
E	*0.190794*	*1.978757*	*2.605967*	*0.683509*

More Index Details

df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Reset to default 0,1...n index

df.**reset_index()**

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

newind = 'CA NY WY OR CO'.split()

newind

['CA', 'NY', 'WY', 'OR', 'CO']

df['States'] = newind

df

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

df.set_index('States')

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

df

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

df.**set_index('States',inplace=True) #Inplace para q se guarde en el df original**

df

	W	*X*	*Y*	*Z*
States				
CA	*2.706850*	*0.628133*	*0.907969*	*0.503826*
NY	*0.651118*	*-0.319318*	*-0.848077*	*0.605965*
WY	*-2.018168*	*0.740122*	*0.528813*	*-0.589001*
OR	*0.188695*	*-0.758872*	*-0.933237*	*0.955057*
CO	*0.190794*	*1.978757*	*2.605967*	*0.683509*

Multi-Index and Index Hierarchy

Multi-Index and Index Hierarchy

```
# Index Levels
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index
```

[('G1', 1), ('G1', 2), ('G1', 3), ('G2', 1), ('G2', 2), ('G2', 3)]

```
hier_index = pd.MultiIndex.from_tuples(hier_index)
hier_index
```

MultiIndex([('G1', 1),
('G1', 2),
('G1', 3),
('G2', 1),
('G2', 2),
('G2', 3)],
)

df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B']) #Crea una matriz de 6 filas y 2 columnas, le pone de indíce el indíce hier_index que definimos, y el nombre de las columnas es A y B

df

		A	B
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

Now let's show how to index this! For index hierarchy we use df.loc[], if this was on the columns axis, you would just use normal bracket notation df[]. Calling one level of the index returns the sub-dataframe:

df.loc['G1']

	A	B
1	*0.153661*	*0.167638*
2	*-0.765930*	*0.962299*
3	*0.902826*	*-0.537909*

df.loc['G1'].loc[1] #Me devuelve la fila 1 en formato de columna

A 0.153661
B 0.167638
Name: 1, dtype: float64

df.index.names

FrozenList([None, None])

df.index.names = ['Group','Num']

df

		A	B
Group	**Num**		
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

df.xs('G1')

	A	B
1	*0.153661*	*0.167638*
2	*-0.765930*	*0.962299*
3	*0.902826*	*-0.537909*

df.xs(['G1',1]) #Me devuelve la fila 1 de G1 pero en formato de columna

A 0.153661
B 0.167638
Name: (G1, 1), dtype: float64

df.xs(1,level='Num') #Me devuelve los q tienen 1 en el indice Num

	A	B
Group		
G1	0.153661	0.167638
G2	-1.549671	0.435253

Missing Data

```
import numpy as np
import pandas as pd
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

df

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

df.**dropna**() *#Elimina las **filas** que tengan algún dato nulo en algún campo*

	A	B	C
0	1.0	5.0	1

df.**dropna**(axis=1) #Elimina las **columnas** q tienen algún nulo en alguna fila

	C
0	1
1	2
2	3

df.dropna(thresh=2)

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

*df.**fillna(value='FILL VALUE')***

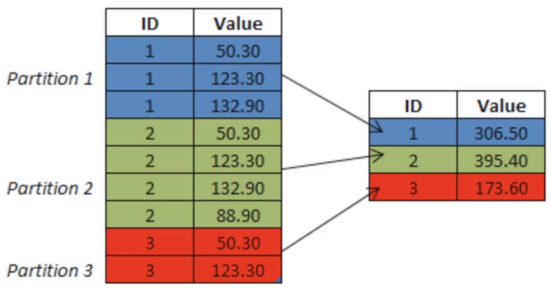
	A	B	C
0	1	5	1
1	2	FILL VALUE	2
2	FILL VALUE	FILL VALUE	3

df['A'].fillna(value=df['A'].mean()) #Reemplaza los nulos de la columna A con el promedio de los valores de la columna A

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

Group By

GroupBy te permite agrupar filas basandose en una columna y realizar alguna función agregada sobre ellas.




```
import pandas as pd
# Create dataframe
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
```

'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
'Sales':[200,120,340,124,243,350]}

data

{'Company': ['GOOG', 'GOOG', 'MSFT', 'MSFT', 'FB', 'FB'],
'Person': ['Sam', 'Charlie', 'Amy', 'Vanessa', 'Carl', 'Sarah'],
'Sales': [200, 120, 340, 124, 243, 350]}

df = pd.DataFrame(data)

df

	Company	*Person*	*Sales*
0	*GOOG*	*Sam*	*200*
1	*GOOG*	*Charlie*	*120*
2	*MSFT*	*Amy*	*340*
3	*MSFT*	*Vanessa*	*124*
4	*FB*	*Carl*	*243*
5	*FB*	*Sarah*	*350*

#** Now you can use the .groupby() method to group rows together based off of a column name. For instance let's group based off of Company. This will create a DataFrameGroupBy object:**

df.**groupby('Company')**

<pandas.core.groupby.DataFrameGroupBy object at 0x113014128>

by_comp = df.groupby("Company")

by_comp.mean()

Company	*Sales*
FB	*296.5*
GOOG	*160.0*
MSFT	*232.0*

df.**groupby('Company').mean()**

Company	*Sales*
FB	*296.5*
GOOG	*160.0*
MSFT	*232.0*

by_comp.**sum().loc['FB']**

Person CarlSarah
Sales 593
Name: FB, dtype: object

df.groupby('Company').sum().loc['FB']

Person CarlSarah
Sales 593
Name: FB, dtype: object

by_comp.std()

Company	*Sales*
FB	*75.660426*

GOOG	*56.568542*	
MSFT	*152.735065*	

by_comp.min() #En campos string devuelve por orden alfabético

	Person	*Sales*
Company		
FB	*Carl*	*243*
GOOG	*Charlie*	*120*
MSFT	*Amy*	*124*

by_comp.max()

	Person	*Sales*
Company		
FB	*Sarah*	*350*
GOOG	*Sam*	*200*
MSFT	*Vanessa*	*340*

by_comp.count()

	Person	*Sales*
Company		
FB	*2*	*2*
GOOG	*2*	*2*
MSFT	*2*	*2*

by_comp.describe()

		Sales
Company		
FB	count	2.000000
	mean	296.500000
	std	75.660426
	min	243.000000
	25%	269.750000
	50%	296.500000
	75%	323.250000
	max	350.000000
GOOG	count	2.000000
	mean	160.000000
	std	56.568542
	min	120.000000
	25%	140.000000
	50%	160.000000
	75%	180.000000
	max	200.000000
MSFT	count	2.000000
	mean	232.000000
	std	152.735065
	min	124.000000
	25%	178.000000
	50%	232.000000
	75%	286.000000
	max	340.000000

by_comp.describe().transpose()

Company	FB								GOOG					MSFT						
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
Sales	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0	2.0	160.0	...	180.0	200.0	2.0	232.0	152.735065	124.0	178.0	232.0	286.0	340.0

by_comp.describe().transpose()['GOOG']

	count	mean	std	min	25%	50%	75%	max
Sales	2.0	160.0	56.568542	120.0	140.0	160.0	180.0	200.0

Merging, Joining, and Concatenating

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
df1
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
df2
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])
df3
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Concatenation

Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use **pd.concat** and pass in a list of DataFrames to concatenate together:

```
pd.concat([df1,df2,df3])
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

pd.concat([df1,df2,df3],axis=1) #Con axis=1 se concatenan las columnas

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Merging

The **merge** function allows you to merge DataFrames together using a similar logic as merging SQL Tables together.

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})
```

left

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

```
right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

right

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

pd.merge(left,right,how='**inner**',on='key')

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                       'A': ['A0', 'A1', 'A2', 'A3'],
                       'B': ['B0', 'B1', 'B2', 'B3']})
right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                      'key2': ['K0', 'K0', 'K0', 'K0'],
                        'C': ['C0', 'C1', 'C2', 'C3'],
                        'D': ['D0', 'D1', 'D2', 'D3']})
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

pd.merge(left, right, on=['key1', 'key2'])

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2

pd.merge(left, right, how='**outer**', on=['key1', 'key2'])

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN
5	NaN	NaN	K2	K0	C3	D3

pd.merge(left, right, how='**right**', on=['key1', 'key2'])

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2
3	NaN	NaN	K2	K0	C3	D3

pd.merge(left, right, how='left', on=['key1', 'key2'])

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN

Joining

Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                     index=['K0', 'K1', 'K2'])
right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

right

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

left.join(right)

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

left.join(right, how=**'outer'**)

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Operations

Info on Unique Values

```
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','xyz']})
df
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

```
df['col2'].unique() #Devuelve los valores distintos que tiene un campo
```

array([444, 555, 666])

```
df['col2'].nunique() #Devuelve la cant de valores distintos que tiene un campo
```

3

```
df['col2'].value_counts() #Devuelve los valores distintos que tiene un campo y la cantidad que hay de cada uno
```

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

Selecting Data

```
newdf = df[(df['col1']>2) & (df['col2']==444)]
newdf
```

	col1	col2	col3
3	4	444	xyz

Applying Functions

```
def times2(x):
    return x*2

df['col1'].apply(times2)
```

```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

```
df['col3'].apply(len)
```

```
0    3
1    3
2    3
3    3
```

Name: col3, dtype: int64

df['col1'].sum() #Suma los valores de la columna

10

del df['col1'] #Elimina una columna

	col2	*col3*
0	*444*	*abc*
1	*555*	*def*
2	*666*	*ghi*
3	*444*	*xyz*

df.columns #Devuelve el nombre del índice y de las columnas

Index(['col2', 'col3'], dtype='object')

df.index() #Hasta que valor llega el índice y de a cuánto pasa de un valor al siguiente

RangeIndex(start=0, stop=4, step=1)

Ordenar DataFrame

df

	col2	*col3*
0	*444*	*abc*
1	*555*	*def*
2	*666*	*ghi*
3	*444*	*xyz*

df.sort_values(by='col2') #inplace=False by default #Ordena por la col2 de menor a mayor

	col2	*col3*
0	*444*	*abc*
3	*444*	*xyz*
1	*555*	*def*
2	*666*	*ghi*

Check Null Values

df.isnull() #Devuelve True or False en c/valor según si es nulo o no

	col2	*col3*
0	*False*	*False*
1	*False*	*False*
2	*False*	*False*
3	*False*	*False*

df.dropna() # Drop rows with NaN Values

	col2	*col3*
0	*444*	*abc*
1	*555*	*def*
2	*666*	*ghi*

3	*444*	*xyz*	

```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df
```

	col1	*col2*	*col3*
0	*1.0*	*NaN*	*abc*
1	*2.0*	*555.0*	*def*
2	*3.0*	*666.0*	*ghi*
3	*NaN*	*444.0*	*xyz*

```
df.fillna('FILL')
```

	col1	*col2*	*col3*
0	*1*	*FILL*	*abc*
1	*2*	*555*	*def*
2	*3*	*666*	*ghi*
3	*FILL*	*444*	*xyz*

```
data = {'A':['foo','foo','foo','bar','bar','bar'],
        'B':['one','one','two','two','one','one'],
        'C':['x','y','x','y','x','y'],
        'D':[1,3,2,5,4,1]}
df = pd.DataFrame(data)
df
```

	A	*B*	*C*	*D*
0	*foo*	*one*	*x*	*1*
1	*foo*	*one*	*y*	*3*
2	*foo*	*two*	*x*	*2*
3	*bar*	*two*	*y*	*5*
4	*bar*	*one*	*x*	*4*
5	*bar*	*one*	*y*	*1*

```
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
```

	C	x	y
A	**B**		
bar	one	4.0	1.0
	two	NaN	5.0
foo	one	1.0	3.0
	two	2.0	NaN

Data Input and Output

CSV

df = **pd.read_csv('example')** #INPUT
df

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

df.**to_csv('example',index=False)** #OUTPUT

EXCEL

pd.**read_excel('Excel_Sample.xlsx',sheetname='Sheet1')** #INPUT #Solo lee data, no formulas ni imagenes

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

df**.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')** #OUTPUT

HTML

You may need to install htmllib5,lxml, and BeautifulSoup4. In your terminal/command prompt run:

conda install lxml
conda install html5lib
conda install BeautifulSoup4

Then restart Jupyter Notebook. (or use pip install if you aren't using the Anaconda Distribution)

Pandas can read table tabs off of html. Pandas read_html function will read tables off of a webpage and return a list of DataFrame objects:

df = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')

df[0]

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date	Loss Share Type	Agreement Terminated	Termination Date
0	First CornerStone Bank	King of Prussia	PA	35312	First-Citizens Bank & Trust Company	May 6, 2016	July 12, 2016	none	NaN	NaN
1	Trust Company Bank	Memphis	TN	9956	The Bank of Fayette County	April 29, 2016	August 4, 2016	none	NaN	NaN
2	North Milwaukee State Bank	Milwaukee	WI	20364	First-Citizens Bank & Trust Company	March 11, 2016	June 16, 2016	none	NaN	NaN
3	Hometown National Bank	Longview	WA	35156	Twin City Bank	October 2, 2015	April 13, 2016	none	NaN	NaN
4	The Bank of Georgia	Peachtree City	GA	35259	Fidelity Bank	October 2, 2015	April 13, 2016	none	NaN	NaN
5	Premier Bank	Denver	CO	34112	United Fidelity Bank, fsb	July 10, 2015	July 12, 2016	none	NaN	NaN
6	Edgebrook Bank	Chicago	IL	57772	Republic Bank of Chicago	May 8, 2015	July 12, 2016	none	NaN	NaN
7	Doral BankEn Espanol	San Juan	PR	32102	Banco Popular de Puerto Rico	February 27, 2015	May 13, 2015	none	NaN	NaN
8	Capitol City Bank & Trust Company	Atlanta	GA	33938	First-Citizens Bank & Trust Company	February 13, 2015	April 21, 2015	none	NaN	NaN
9	Highland Community Bank	Chicago	IL	20290	United Fidelity Bank, fsb	January 23, 2015	April 21, 2015	none	NaN	NaN
10	First National Bank of Crestview	Crestview	FL	17557	First NBC Bank	January 16, 2015	January 15, 2016	none	NaN	NaN
11	Northern Star Bank	Mankato	MN	34983	BankVista	December 19, 2014	January 6, 2016	none	NaN	NaN
12	Frontier Bank, FSB D/B/A El Paseo Bank	Palm Desert	CA	34738	Bank of Southern California, N.A.	November 7, 2014	January 6, 2016	none	NaN	NaN
13	The National Republic Bank of Chicago	Chicago	IL	916	State Bank of Texas	October 24, 2014	January 6, 2016	none	NaN	NaN
14	NBRS Financial	Rising Sun	MD	4862	Howard Bank	October 17, 2014	March 26, 2015	none	NaN	NaN
15	GreenChoice Bank, fsb	Chicago	IL	28462	Providence Bank, LLC	July 25, 2014	July 28, 2015	none	NaN	NaN
16	Eastside Commercial Bank	Conyers	GA	58125	Community & Southern Bank	July 18, 2014	July 11, 2016	none	NaN	NaN
17	The Freedom State Bank	Freedom	OK	12483	Alva State Bank & Trust Company	June 27, 2014	March 25, 2016	none	NaN	NaN

SQL

The pandas.io.sql module provides a collection of query wrappers to both facilitate data retrieval and to reduce dependency on DB-specific API. Database abstraction is provided by SQLAlchemy if installed. In addition you will need a driver library for your database. Examples of such drivers are psycopg2 for PostgreSQL or pymysql for MySQL. For SQLite this is included in Python's standard library by default. You can find an overview of supported drivers for each SQL dialect in the SQLAlchemy docs.

If SQLAlchemy is not installed, a fallback is only provided for sqlite (and for mysql for backwards compatibility, but this is deprecated and will be removed in a future version). This mode requires a Python database adapter which respect the Python DB-API.

The key functions are:

- read_sql_table(table_name, con[, schema, ...])
  Read SQL database table into a DataFrame.
- read_sql_query(sql, con[, index_col, ...])
  Read SQL query into a DataFrame.
- read_sql(sql, con[, index_col, ...])
  Read SQL query or database table into a DataFrame.
- DataFrame.to_sql(name, con[, flavor, ...])
  Write records stored in a DataFrame to a SQL database.

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:')
df.to_sql('data', engine)
sql_df = pd.read_sql('data',con=engine)
sql_df
```

	index	*a*	*b*	*c*	*d*
0	*0*	*0*	*1*	*2*	*3*
1	*1*	*4*	*5*	*6*	*7*
2	*2*	*8*	*9*	*10*	*11*
3	*3*	*12*	*13*	*14*	*15*

Section 8: Python for Data Visualization - Matplotlib

Introducción a Matplotlib

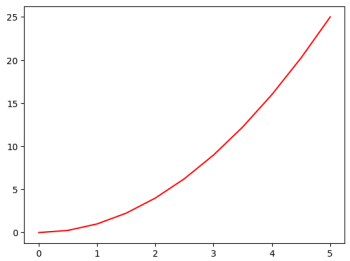
Matplotlib is the "grandfather" library of data visualization with Python. It was created by John Hunter, who created it to try to replicate MatLab's plotting capabilities. It is an excellent 2 and 3D graphics library for generating scientific figures. Some of the major Pros of Matplotlib are:

- Generally easy to get started for simple plots
- Support for custom labels and texts, very customizable in general
- Great control of every element in a figure
- High-quality output in many formats

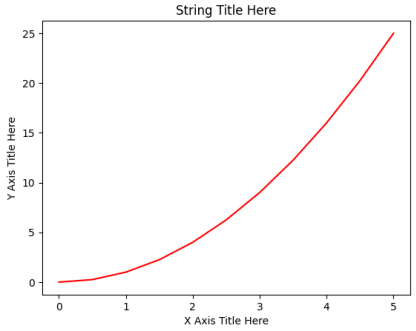
Official Matplotlib web page: http://matplotlib.org/

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 5, 11)
y = x ** 2
x
        array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ])
y
        array([  0.  ,   0.25,   1.  ,   2.25,   4.  ,   6.25,   9.  ,  12.25,
      16.  ,  20.25,  25.  ])
plt.plot(x, y, 'r') # 'r' is the color red
```

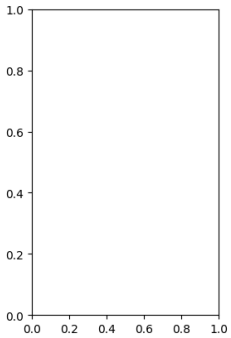



```
plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('X Axis Title Here')
plt.ylabel('Y Axis Title Here')
plt.title('String Title Here')
plt.show()
```

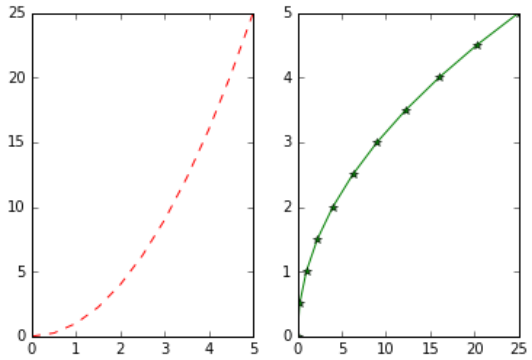


Creating Multiplots on Same Canvas

plt.**subplot(1,2,1) #ancho,alto y nro de gráfico**



 # plt.subplot(nrows, ncols, plot_number)
plt.**subplot(1,2,1)**
plt.plot(x, y, **'r--'**) # More on color options later
plt**.subplot(1,2,2)**
plt.plot(y, x, **'g*-'**);



Matplotlib Object Oriented Method

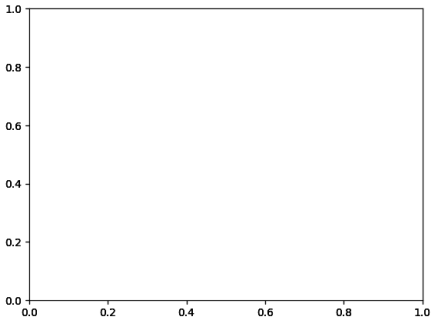
Ej1: Object Oriented Method

The main idea in using the more formal Object Oriented method is to **create figure objects and then just call methods or attributes off of that object.** This approach is nicer when dealing with a canvas that has multiple plots on it.

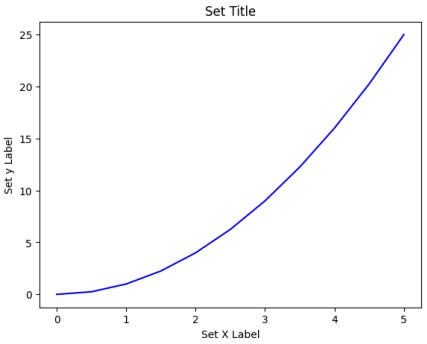
fig=**plt.figure()** # Create Figure (empty canvas)
fig

<Figure size 640x480 with 0 Axes>

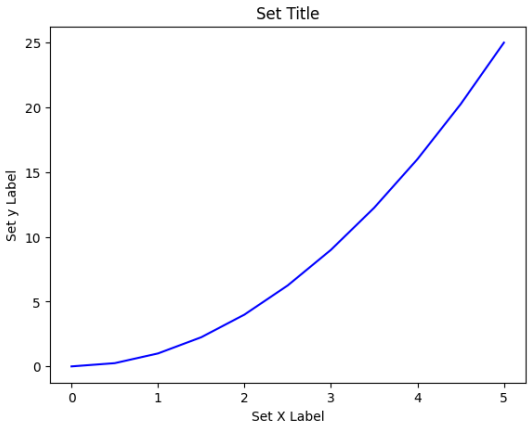
axes = **fig.add_axes**([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1) # Add set of axes to figure
fig



Plot on that set of axes
axes.plot(x, y, 'b')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')
fig

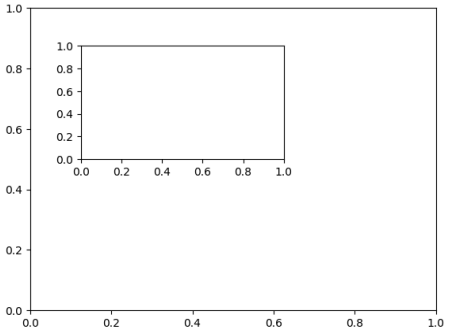


#CÓDIGO COMPLETO
# Create Figure (empty canvas)
fig = plt.figure()
# Add set of axes to figure
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)
# Plot on that set of axes
axes.plot(x, y, 'b')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')

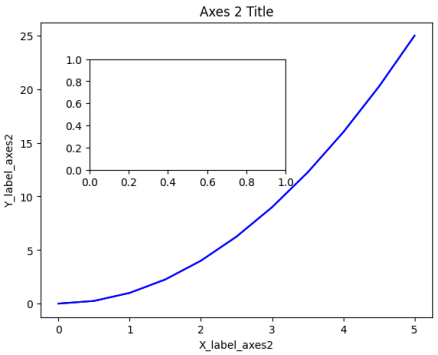


Ej2: Object Oriented Method

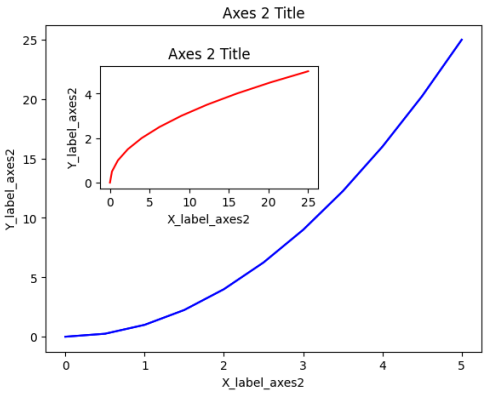
fig = plt.figure() #Create Figure (empty canvas)
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) **# main axes**
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) **# inset axes**



 # Larger Figure Axes 1
axes1.plot(x, y, 'b')
axes1.set_xlabel('X_label_axes2')
axes1.set_ylabel('Y_label_axes2')
axes1.set_title('Axes 2 Title')
fig

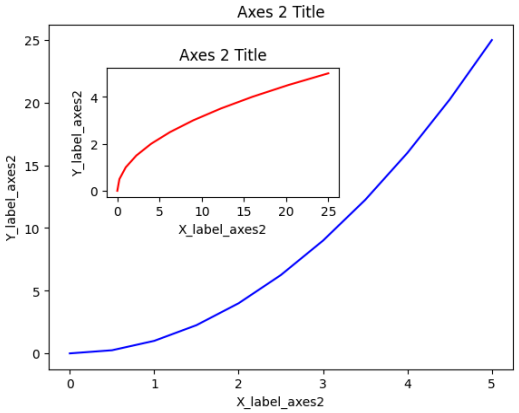


Insert Figure Axes 2
axes2.plot(y, x, 'r')
axes2.set_xlabel('X_label_axes2')
axes2.set_ylabel('Y_label_axes2')
axes2.set_title('Axes 2 Title');
fig



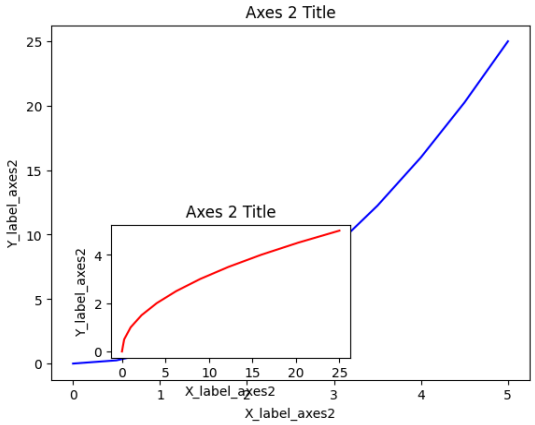
#CÓDIGO COMPLETO

```
fig = plt.figure() # Creates canvas
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes
# Larger Figure Axes 1
axes1.plot(x, y, 'b')
axes1.set_xlabel('X_label_axes2')
axes1.set_ylabel('Y_label_axes2')
axes1.set_title('Axes 2 Title')
# Insert Figure Axes 2
axes2.plot(y, x, 'r')
axes2.set_xlabel('X_label_axes2')
axes2.set_ylabel('Y_label_axes2')
axes2.set_title('Axes 2 Title');
axes2.set_xlim(0,25)
axes2.set_ylim(0,6)
```



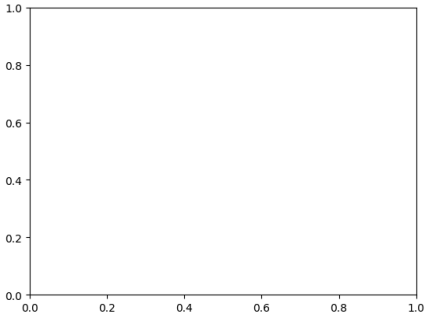
#QUE PASA SI CAMBIO ESTE NÚMERO

```
fig = plt.figure()
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.15, 0.4, 0.3]) # inset axes
...
```

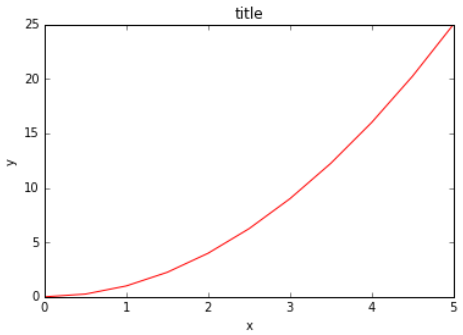


Subplots

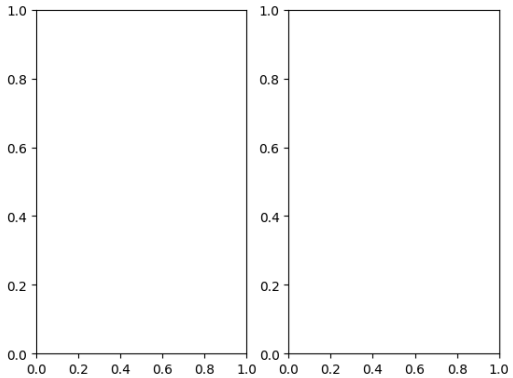
Use similar to plt.figure() except use tuple unpacking to grab fig and axes
fig, axes = plt.subplots()



Use similar to plt.figure() except use tuple unpacking to grab fig and axes
fig, axes = plt.subplots()
# Now use the axes object to add stuff to plot
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');



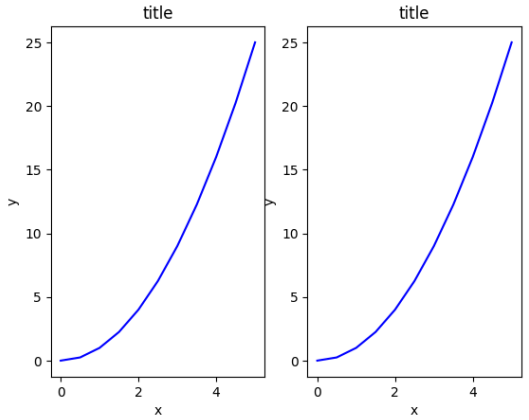
Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)



fig, axes = plt.subplots(nrows=1, ncols=2) #Creamos 2 rectangulos
#We can iterate through this array:

for ax in axes: #Haces lo mismo en los 2 gráficos

```
    ax.plot(x, y, 'b')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')
# Display the figure object
fig
```




```
fig, axes = plt.subplots(nrows=1, ncols=2)
for ax in axes: #Haces lo mismo en los 2 gráficos
    ax.plot(x, y, 'g')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')
fig
```

plt.tight_layout() #Ajusta automáticamente las posiciones de los ejes para que no haya contenido superpuesto

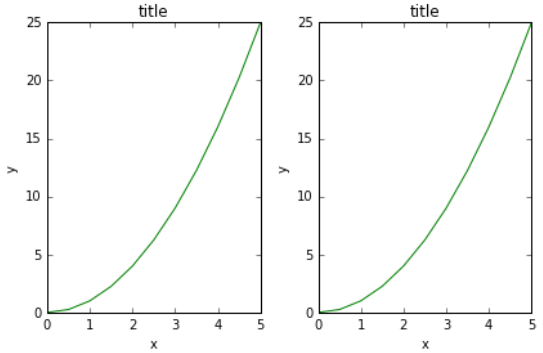
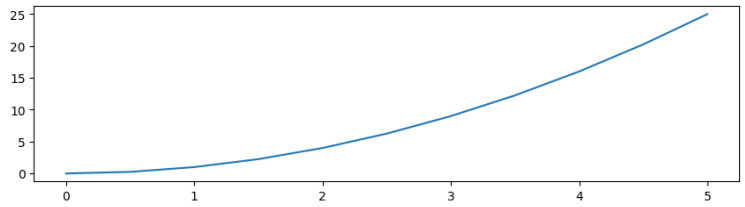


Figure size, aspect ratio and DPI

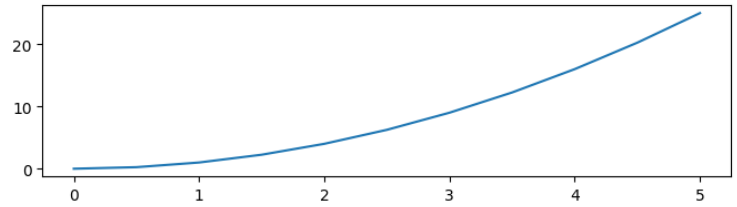
Matplotlib allows the aspect ratio, DPI and figure size to be specified when the Figure object is created. You can use the `figsize` and `dpi` keyword arguments.

- `figsize` is a tuple of the width and height of the figure in inches
- `dpi` is the dots-per-inch (pixel per inch).

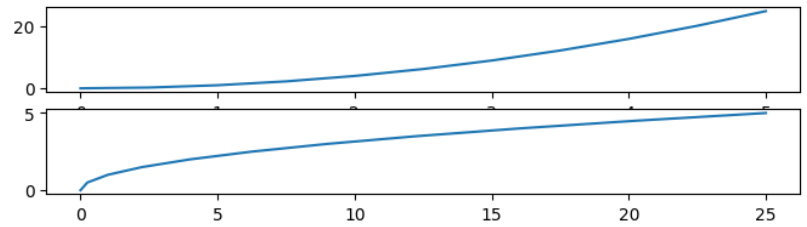
fig=plt.figure(figsize=(8,2))
ax=fig.add_axes([0,0,1,1])
ax.plot(x,y)



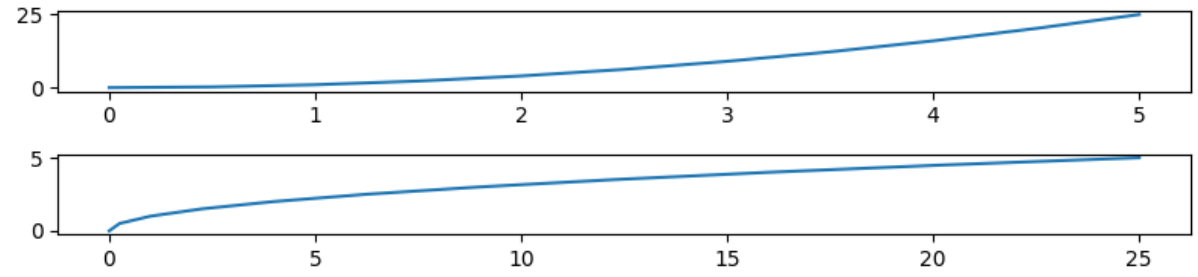
#OTRA FORMA DE HACER CASI LO MISMO
fig,axes=plt.**subplots**(figsize=(8,2))
axes.plot(x,y)



fig,axes=plt.subplots(**nrows=2,ncols=1**,figsize=(8,2))
axes**[0]**.plot(x,y)
axes**[1].**plot(y,x)

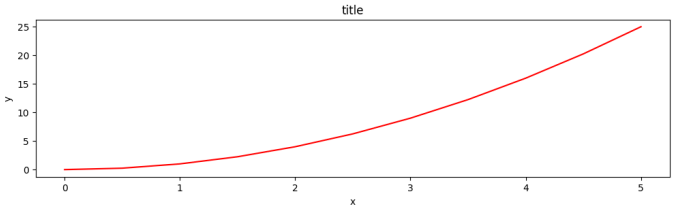


fig,axes=plt.subplots(nrows=2,ncols=1,figsize=(8,2))
axes[0].plot(x,y)
axes[1].plot(y,x)
plt.tight_layout()

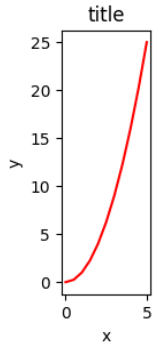


fig, axes = plt.subplots(**figsize**=(12,3))
axes.plot(x, y, 'r')
axes.set_xlabel('x')

axes.set_ylabel('y')
axes.set_title('title');



fig, axes = plt.subplots(**figsize**=(1,3))
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');



Saving Figures

Matplotlib can generate high-quality output in a number formats, including PNG, JPG, EPS, SVG, PGF and PDF. To save a figure to a file we can use the `savefig` method in the `Figure` class:
fig.savefig("filename.png")
fig.savefig("filename.**jpg**")
fig.savefig("filename.png", **dpi**=200)

Legends, labels and titles

Figure Titles

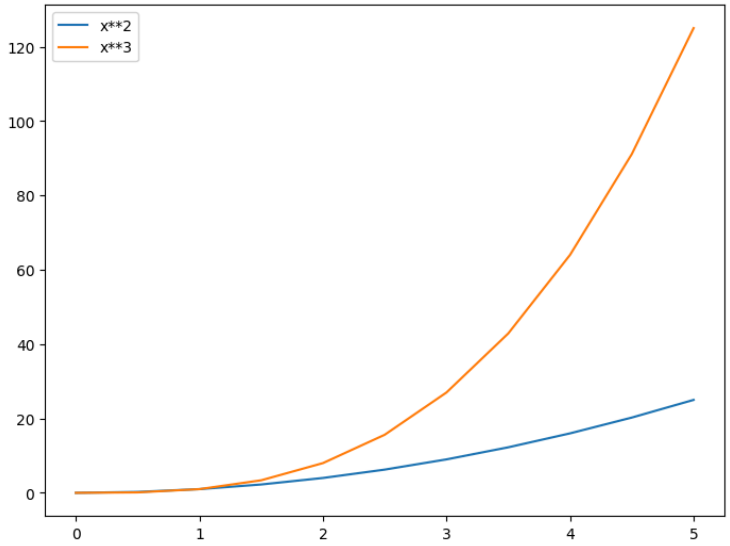
ax.set_title("title");

Axis Labels

ax.set_xlabel("x")
ax.set_ylabel("y");

Legends

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend() #Es el cuadro de referencia, en este caso de colores
```



The **legend** function takes an optional keyword argument **loc** that can be used to specify where in the figure the legend is to be drawn. The allowed values of **loc** are numerical codes for the various places the legend can be drawn. See the [documentation page](http://matplotlib.org/users/legend_guide.html#legend-location) for details. Some of the most common **loc** values are:

```
ax.legend(loc=1) # upper right corner
ax.legend(loc=2) # upper left corner
ax.legend(loc=3) # lower left corner
ax.legend(loc=4) # lower right corner
# .. many more options are available
# Most common to choose
ax.legend(loc=0) # let matplotlib decide the optimal location
```

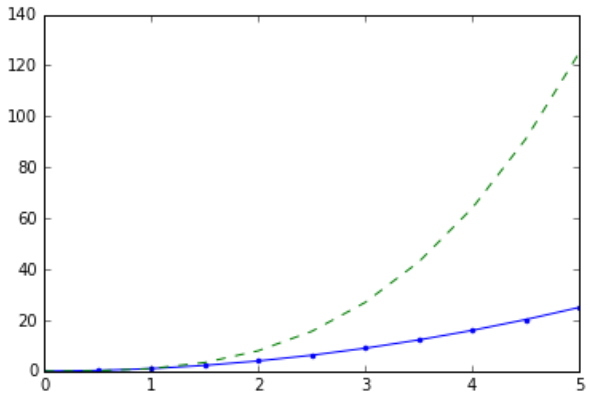
Colors, linewidths, linetypes

Colors with MatLab like syntax

With matplotlib, we can define the colors of lines and other graphical elements in a number of ways. First of all, we can use the MATLAB-like syntax where `'b'` means blue, `'g'` means green, etc. The MATLAB API for selecting line styles are also supported: where, for example, 'b.-' means a blue line with dots:

```
# MATLAB style line color and style
fig, ax = plt.subplots()
```

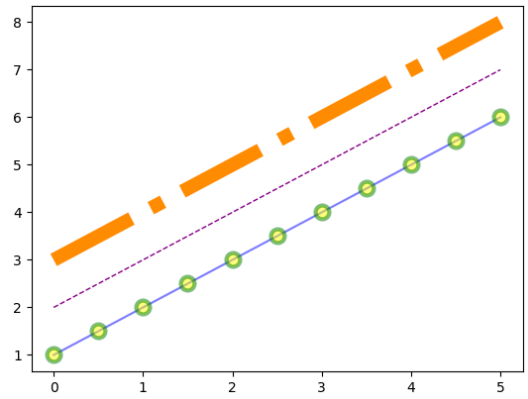
ax.plot(x, x**2, **'b.-') # blue line with dots**
ax.plot(x, x**3, **'g--') # green dashed line**



Colors with the color= parameter

We can also define colors by their names or RGB hex codes and optionally provide an alpha value using the `color` and `alpha` keyword arguments. Alpha indicates opacity.

fig, ax = plt.subplots()
ax.plot(x, x+1, **color="blue", alpha=0.5, marker='o', markersize=10, markerfacecolor='yellow', markeredgewidth=3, markeredgecolor='green'**) **#alpha 0.5 is half-transparant**
ax.plot(x, x+2, **color="#8B008B", linewidth=1, linestyle='--'**) **#RGB hex code**
ax.plot(x, x+3, **color="#FF8C00", lw=10, ls='-.'**) # linewidth como lw y linestyle como ls



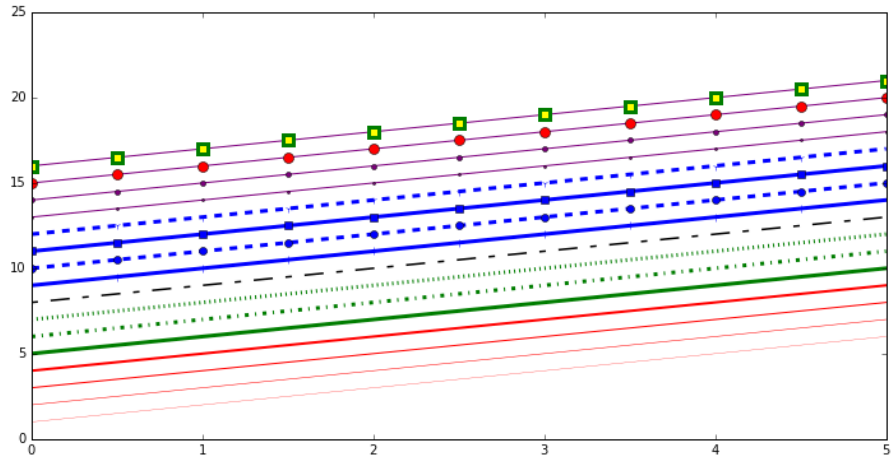
Line and marker styles

To change the line width, we can use the `linewidth` or `lw` keyword argument. The line style can be selected using the `linestyle` or `ls` keyword arguments:

fig, ax = plt.subplots(figsize=(12,6))
ax.plot(x, x+1, **color**="red", **linewidth**=0.25)
ax.plot(x, x+2, color="red", linewidth=0.50)
ax.plot(x, x+3, color="red", linewidth=1.00)
ax.plot(x, x+4, color="red", linewidth=2.00)

possible **linestype options '-', '–', '-.', ':', 'steps'**

```
ax.plot(x, x+5, color="green", lw=3, linestyle='-')
ax.plot(x, x+6, color="green", lw=3, ls='-.')
ax.plot(x, x+7, color="green", lw=3, ls=':')

# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4', ...
ax.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+')
ax.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
ax.plot(x, x+11, color="blue", lw=3, ls='-', marker='s')
ax.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green");
```

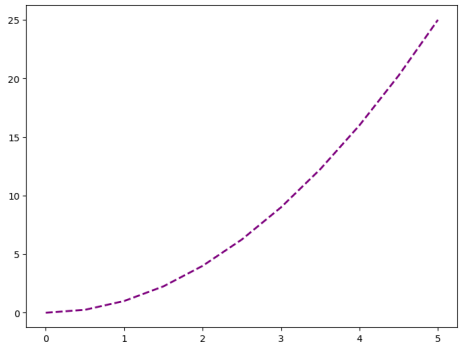


Control over axis appearance

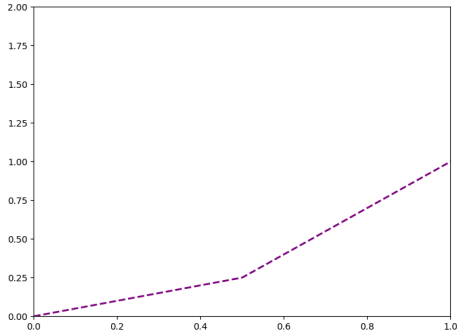
Plot Range

We can configure the ranges of the axes using the `set_ylim` and `set_xlim` methods in the axis object, or `axis('tight')` for automatically getting "tightly fitted" axes ranges:

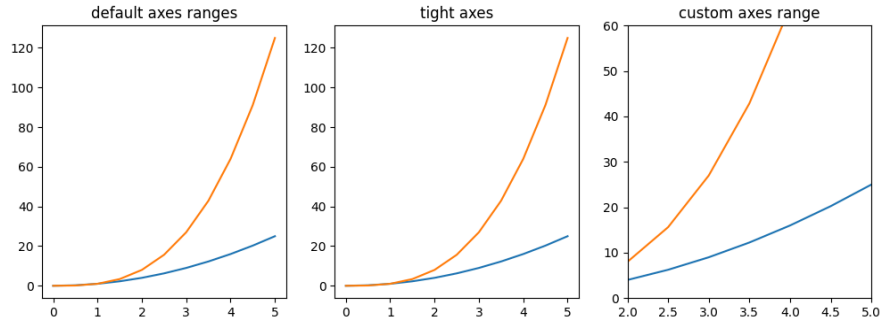
```
fig=plt.figure()
ax =fig.add_axes([0,0,1,1])
ax.plot(x,y,color='purple',lw=2,ls='--')
```



```
fig=plt.figure()
ax =fig.add_axes([0,0,1,1])
ax.plot(x,y,color='purple',lw=2,ls='--')
ax.set_xlim([0,1])
ax.set_ylim([0,2])
```



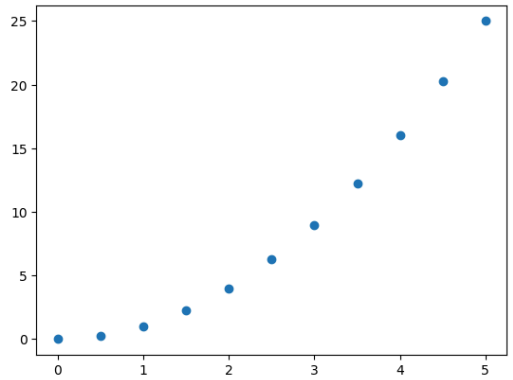
```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("custom axes range");
```



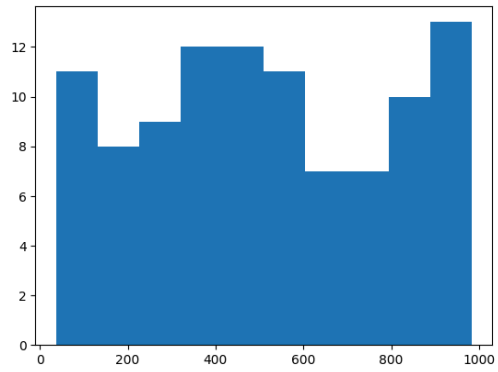
Special Plot Types

There are many specialized plots we can create, such as barplots, histograms, scatter plots, and much more. Most of these type of plots we will actually create using seaborn, a statistical plotting library for Python. But here are a few examples of these type of plots:

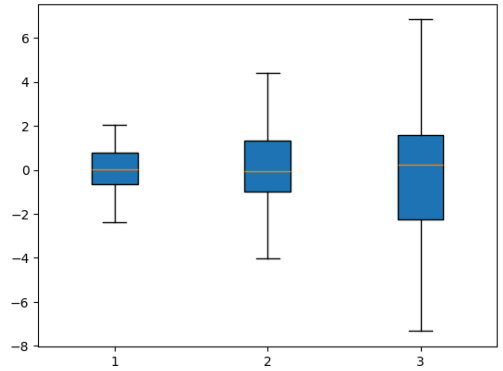
plt.scatter(x,y)



from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)



data = [np.random.normal(0, std, 100) for std in range(1, 4)]
# rectangular box plot
plt.boxplot(data,vert=True,patch_artist=True);



Further Reading

- http://www.matplotlib.org - The project web page for matplotlib.

- https://github.com/matplotlib/matplotlib - The source code for matplotlib.
- http://matplotlib.org/gallery.html - A large gallery showcaseing various types of plots matplotlib can create. Highly recommended!
- http://www.loria.fr/~rougier/teaching/matplotlib - A good matplotlib tutorial.
- http://scipy-lectures.github.io/matplotlib/matplotlib.html - Another good matplotlib reference.

Section 9: Python for Data Visualization - Seaborn

Introducción a Seaborn

Seaborn es una biblioteca de gráficos estadísticos y está construida sobre la trama de Matplotlib.

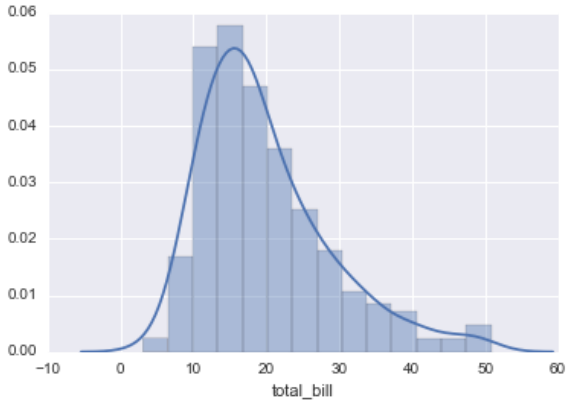
Dado que Seaborn es de código abierto está alojado en Github, así que podés entrar a Google y poner “github seaborn python” y te debería llevar a la página de Github donde usted puede leer todo el código, que es de código abierto en cuanto a la biblioteca de estadísticas para esta visualización de datos.
https://github.com/mwaskom/seaborn?tab=readme-ov-file

Desplazandonos hacia abajo en la página de Github verás un enlace a la documentación oficial. https://seaborn.pydata.org/

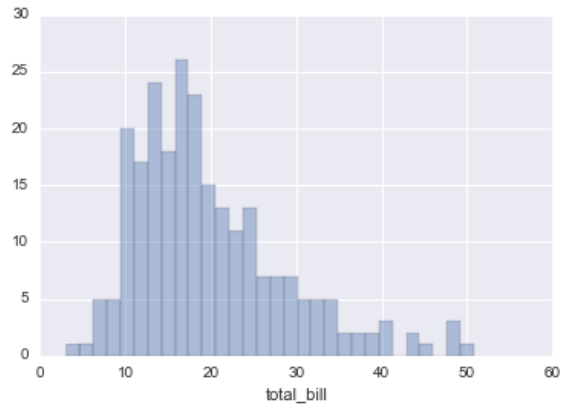
El otro enlace realmente importante es la API y la referencia de la API es básicamente una referencia a los diversos tipos de gráficos aquí.

Distribution Plots

```
import seaborn as sns
%matplotlib inline
tips = sns.load_dataset('tips')

## Distplot: The distplot shows the distribution of a univariate set of observations.
sns.distplot(tips['total_bill'])
# Safe to ignore warnings
```




```
sns.distplot(tips['total_bill'],kde=False,bins=30) #bins es la cantidad de barras
```

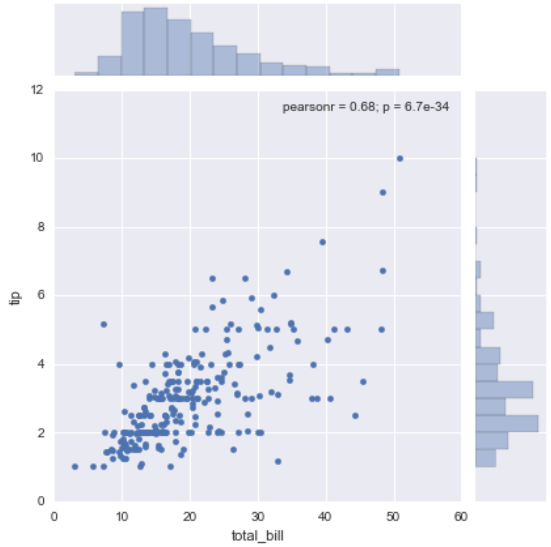


Jointplot:

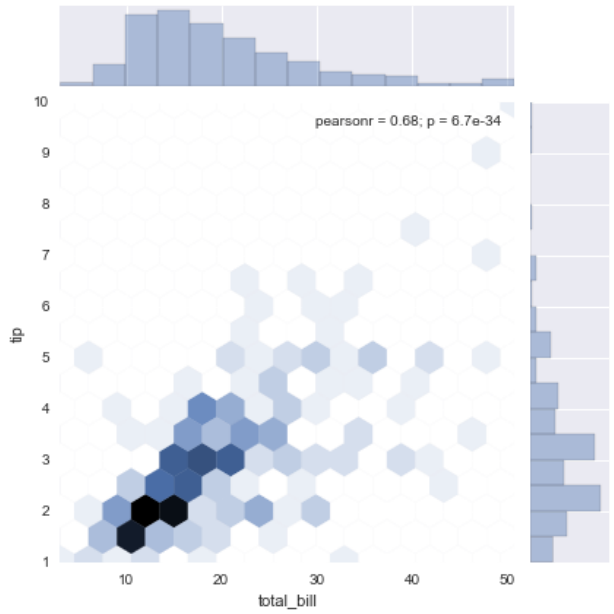
jointplot() allows you to basically match up two distplots for bivariate data. With your choice of what **kind** parameter to compare with:

* “scatter”
* “reg”
* “resid”
* “kde”
* “hex”

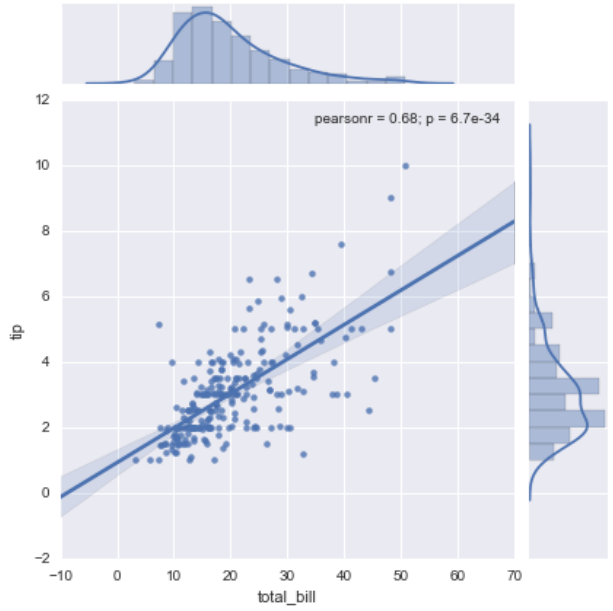
sns.jointplot(x='total_bill',y='tip',data=tips,**kind='scatter'**)



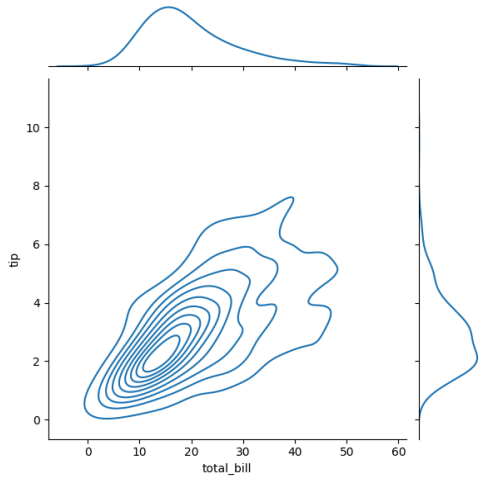
sns.jointplot(x='total_bill',y='tip',data=tips,**kind='hex'**)



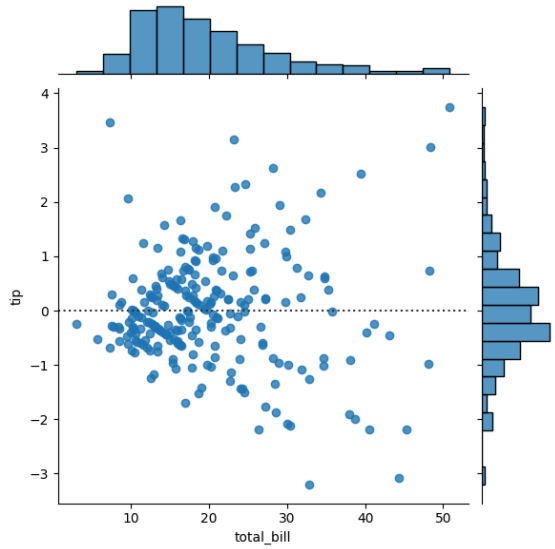
sns.jointplot(x='total_bill',y='tip',data=tips,**kind='reg'**)



sns.jointplot(x='total_bill',y='tip',data=tips,**kind=kde**)

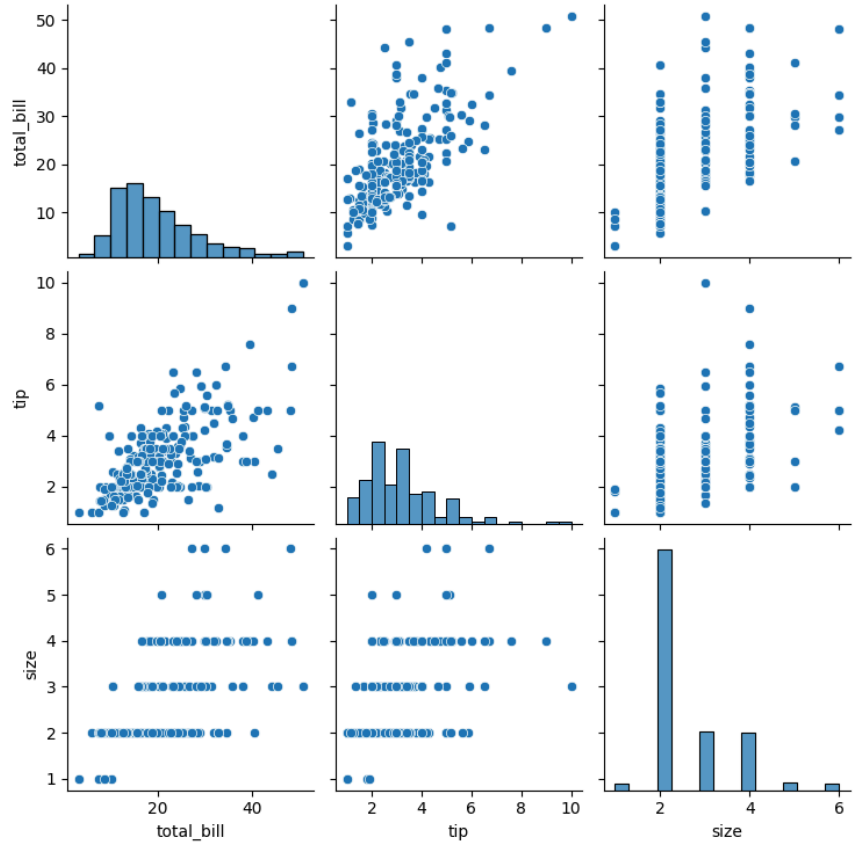


sns.jointplot(x='total_bill',y='tip',data=tips,**kind='resid'**)

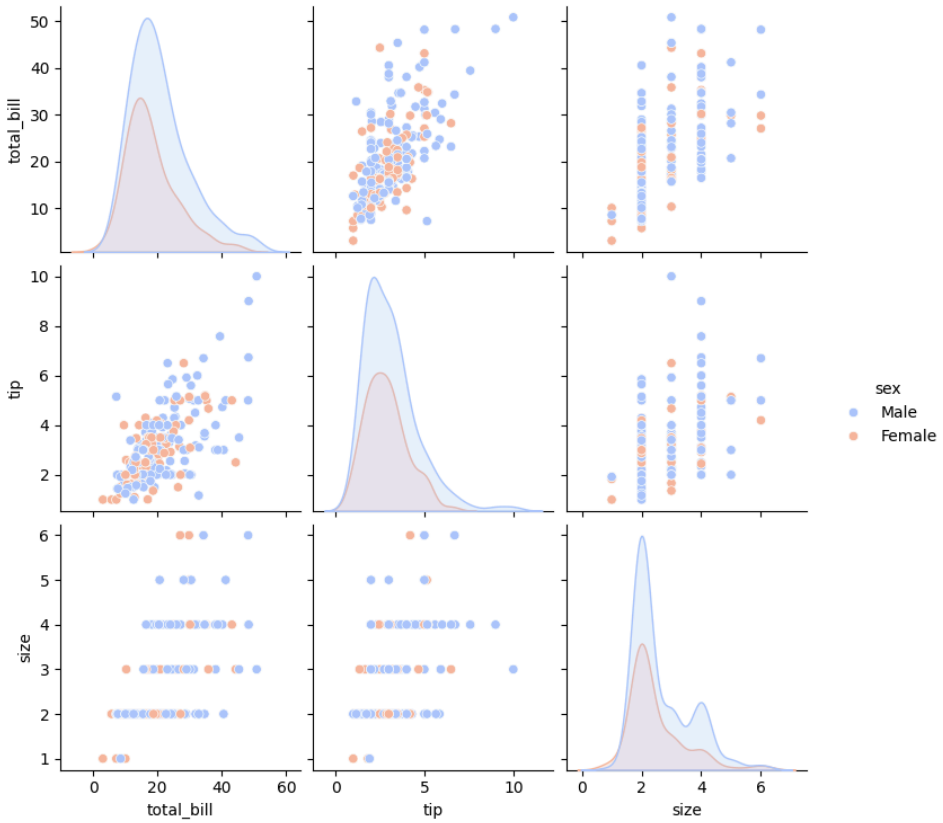


Pairplot: Pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).

sns.**pairplot**(tips) #muestra gráficos relacionando las variables numéricas entre sí

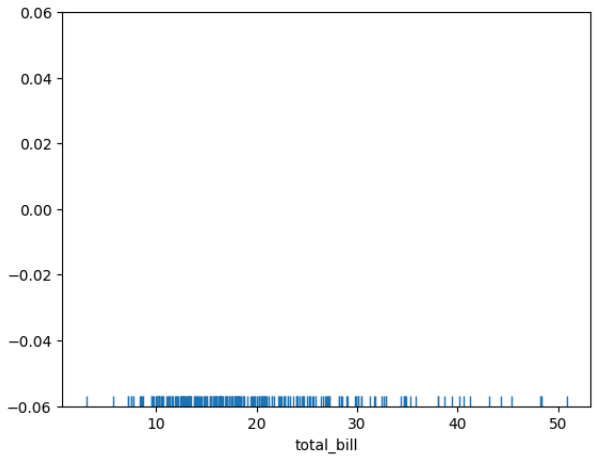


sns.pairplot(tips,**hue='sex',palette='coolwarm'**) #diferenciación por variable categórica



Rugplot: Rugplots just draw a dash mark for every point on a univariate distribution. They are the building block of a KDE plot:

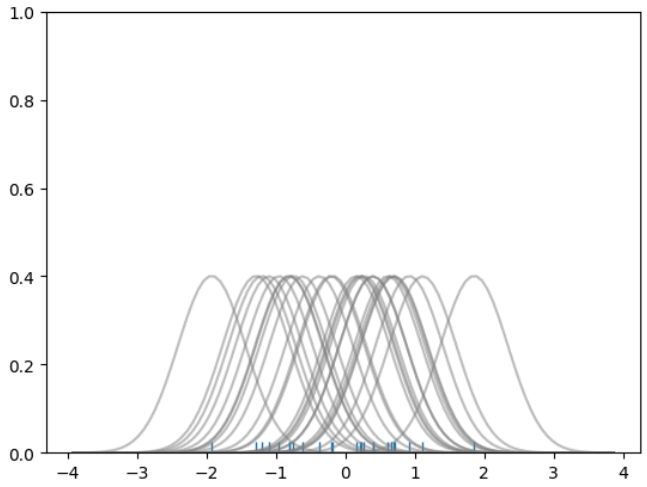
sns.**rugplot**(tips['total_bill']) #marca una linea para cada punto en esa variante uniforme o única



KDEplot: Kernel Density Estimation plots (https://en.wikipedia.org/wiki/Kernel_density_estimation#Practical_estimation_of_the_bandwidth) replace every single observation with a Gaussian (Normal) distribution centered around that value.

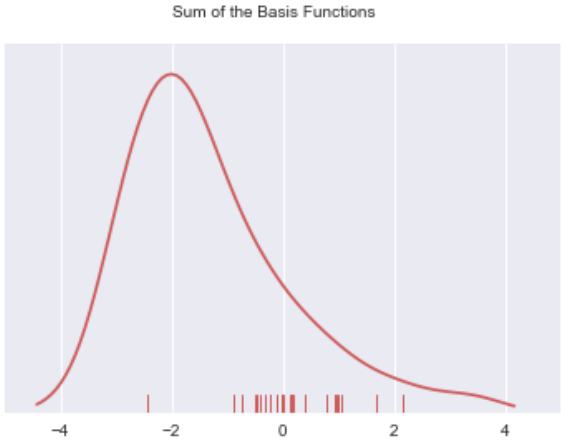
It's just for the diagram below

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
dataset = np.random.randn(25)
# Create another rugplot
sns.rugplot(dataset);
# Set up the x-axis for the plot
x_min = dataset.min() - 2
x_max = dataset.max() + 2
# 100 equally spaced points from x_min to x_max
x_axis = np.linspace(x_min,x_max,100)
# Set up the bandwidth, for info on this:
url =
'http://en.wikipedia.org/wiki/Kernel_density_estimation#Practical_estimation_of_the_bandwidth'
bandwidth = ((4*dataset.std()**5)/(3*len(dataset)))**.2
# Create an empty kernel list
kernel_list = []
# Plot each basis function
for data_point in dataset:
    # Create a kernel for each point and append to list
    kernel = stats.norm(data_point,bandwidth).pdf(x_axis)
    kernel_list.append(kernel)
     #Scale for plotting
    kernel = kernel / kernel.max()
    kernel = kernel * .4
    plt.plot(x_axis,kernel,color = 'grey',alpha=0.5)
plt.ylim(0,1)
```

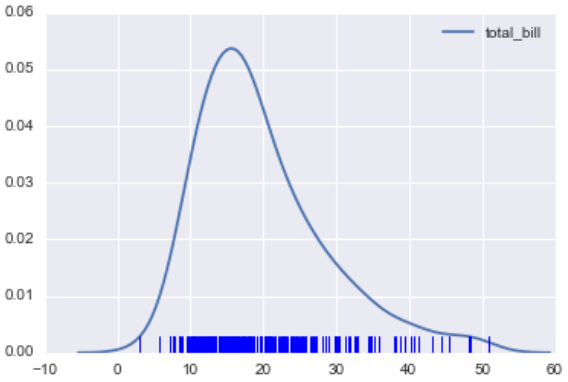



```
# To get the kde plot we can sum these basis functions.
# Plot the sum of the basis function
sum_of_kde = np.sum(kernel_list,axis=0)
```

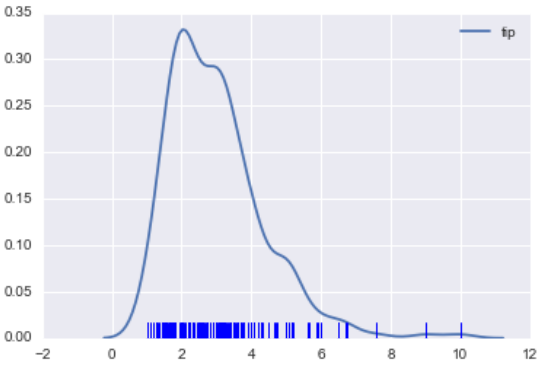
```
# Plot figure
fig = plt.plot(x_axis,sum_of_kde,color='indianred')
# Add the initial rugplot
sns.rugplot(dataset,c = 'indianred')
# Get rid of y-tick marks
plt.yticks([])
# Set title
plt.suptitle("Sum of the Basis Functions")
```




```
sns.kdeplot(tips['total_bill'])
sns.rugplot(tips['total_bill'])
```




```
sns.kdeplot(tips['tip'])
sns.rugplot(tips['tip'])
```

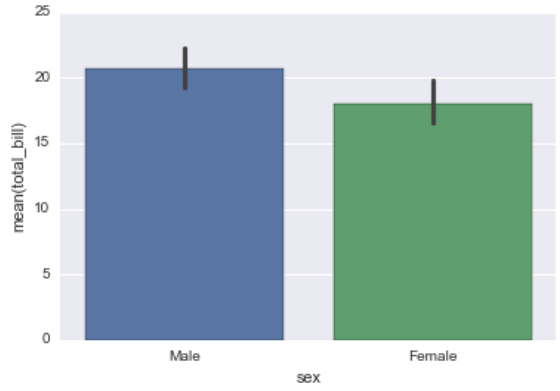


Categorical Plots

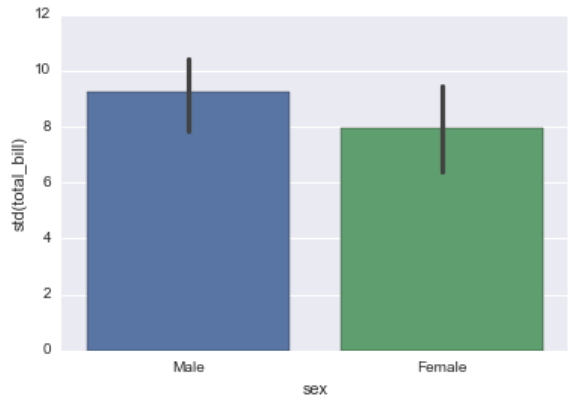
```
import seaborn as sns
%matplotlib inline
tips = sns.load_dataset('tips')
```

Barplot: Barplot is a general plot that allows you to aggregate the categorical data based off some function, by default the mean

```
sns.barplot(x='sex',y='total_bill',data=tips)
```

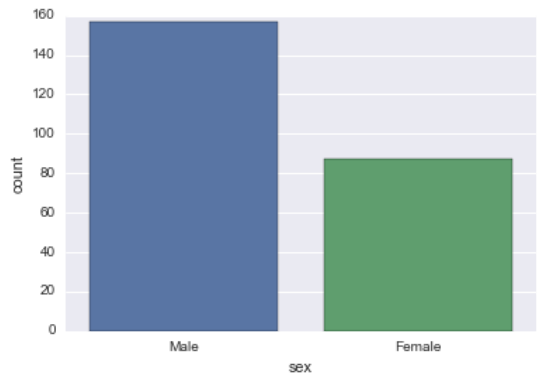



```
sns.barplot(x='sex',y='total_bill',data=tips,estimator=np.std)
```



Countplot: Is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass the x value:

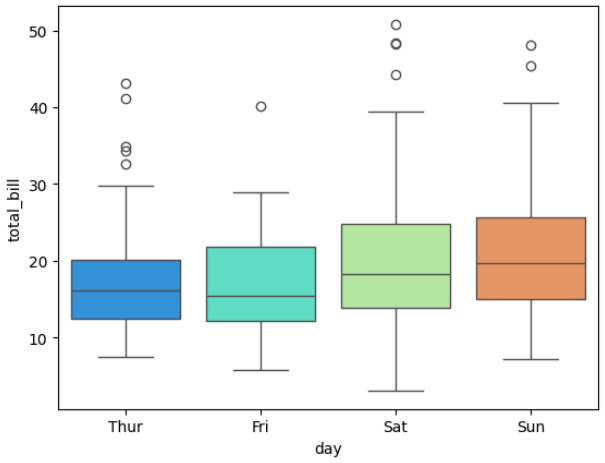
```
sns.countplot(x='sex',data=tips)
```



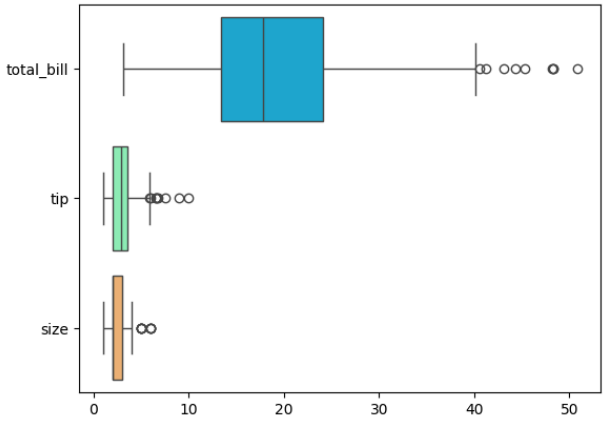
Boxplot

A boxplot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

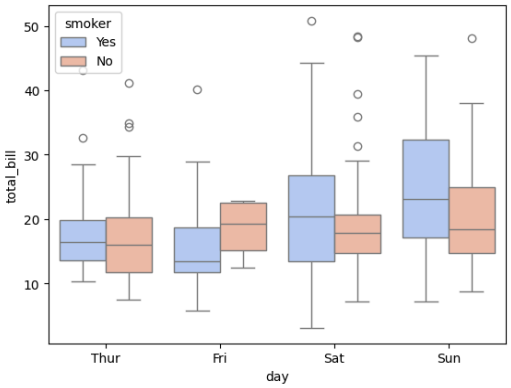
sns.**boxplot**(x="day", y="total_bill", data=tips,palette='rainbow')



sns.boxplot(data=tips,palette='rainbow',**orient='h'**)



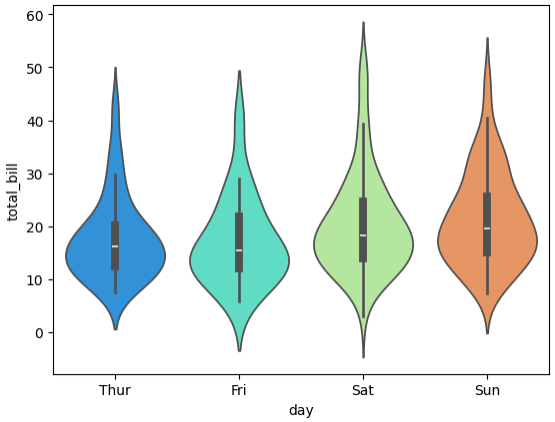
sns.boxplot(x="day", y="total_bill", **hue="smoker"**,data=tips, palette="coolwarm")



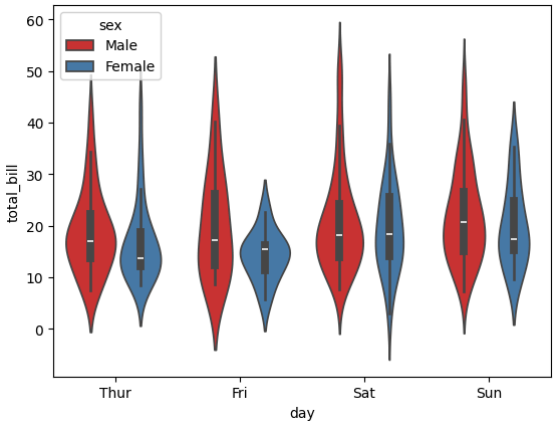
Violinplot:

A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a boxplot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

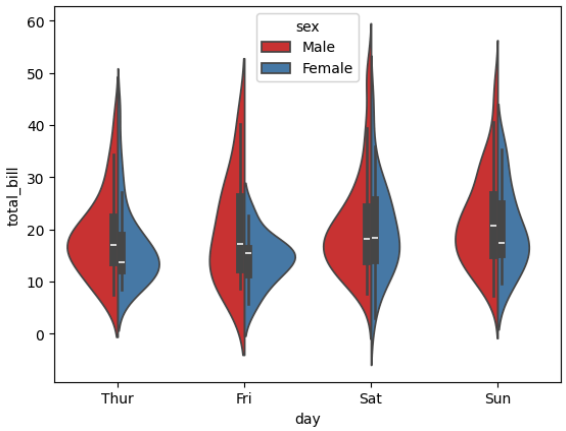
sns.**violinplot**(x="day", y="total_bill", data=tips,palette='rainbow')



sns.violinplot(x="day", y="total_bill", data=tips,**hue='sex'**,palette='Set1')



sns.violinplot(x="day", y="total_bill", data=tips,hue='sex',**split=True**,palette='Set1')

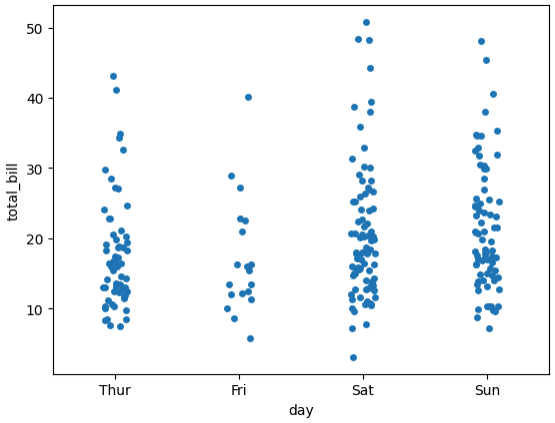


Stripplot and Swarmplot

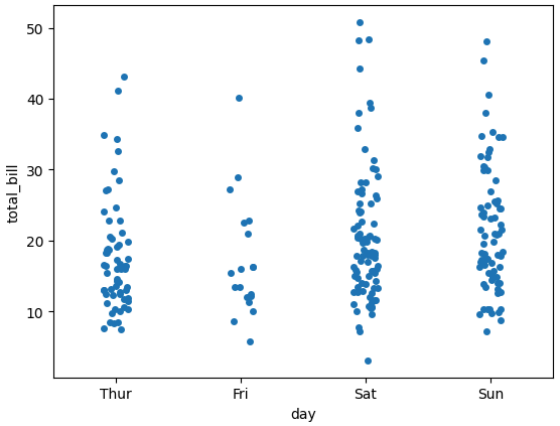
The stripplot will draw a scatterplot where one variable is **categorical**. A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

The swarmplot is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values, although it does not scale as well to large numbers of observations (both in terms of the ability to show all the points and in terms of the computation needed to arrange them).

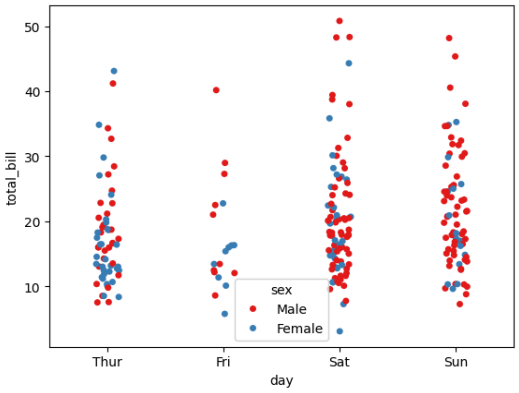
sns.**stripplot**(x="day", y="total_bill", data=tips)



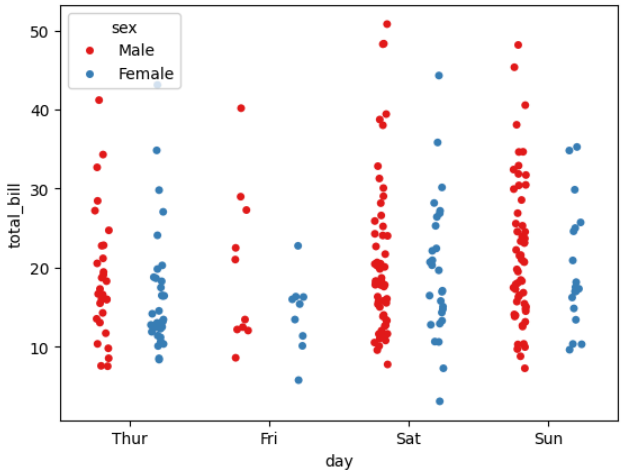
sns.stripplot(x="day", y="total_bill", data=tips,**jitter=True**)



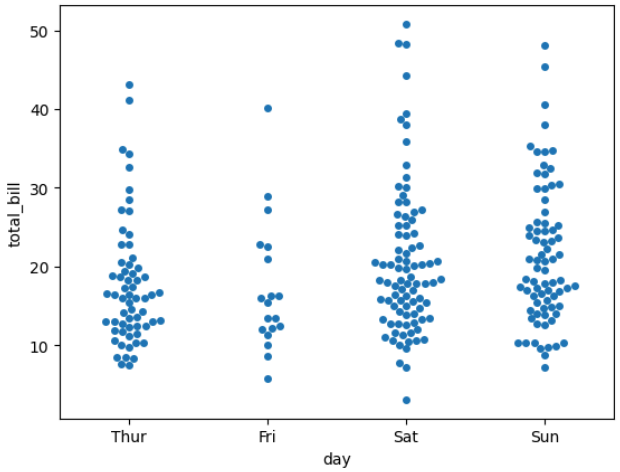
sns.stripplot(x="day", y="total_bill", data=tips,jitter=True,**hue='sex',palette='Set1'**)



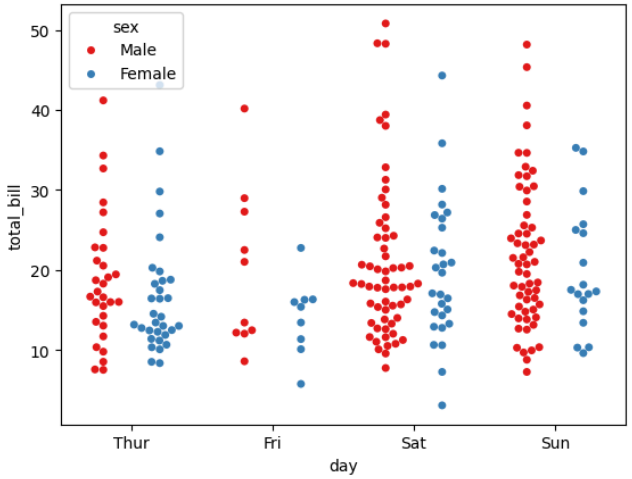
sns.stripplot(x="day", y="total_bill", data=tips,jitter=True,hue='sex',palette='Set1',**dodge=True**)



sns.**swarmplot**(x="day", y="total_bill", data=tips)

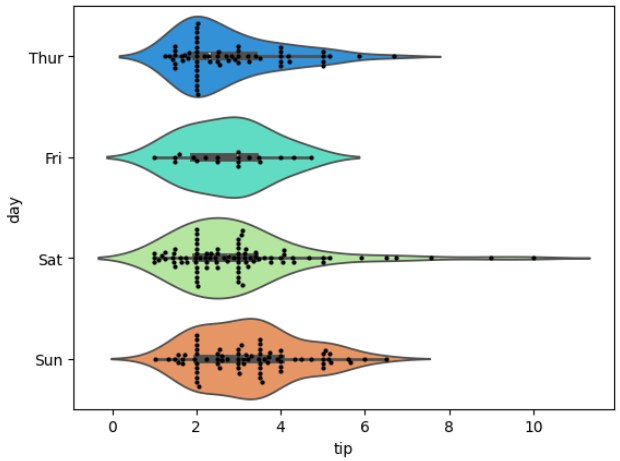


sns.swarmplot(x="day", y="total_bill",hue='sex',data=tips, palette="Set1", **dodge=True**)



Combinación entre Violinplot y Swarmplot

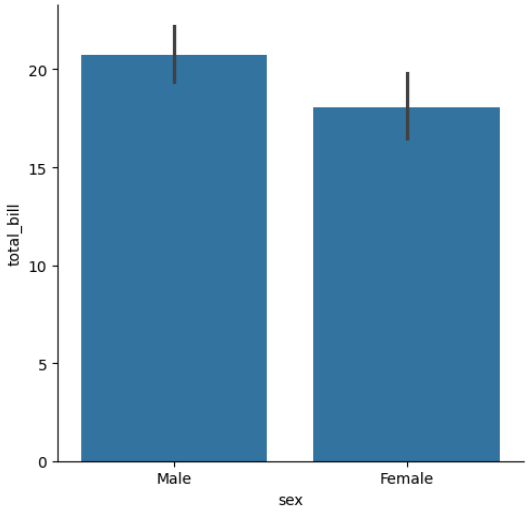
sns.violinplot(x="tip", y="day", data=tips,palette='rainbow')
sns.swarmplot(x="tip", y="day", data=tips,color='black',size=3)



Catplot

Catplot is the most general form of a categorical plot. It can take in a **kind** parameter to adjust the plot type:

sns.catplot(x='sex',y='total_bill',data=tips,kind='bar')



Matrix Plots

Matrix plots allow you to plot data as color-encoded matrices and can also be used to indicate clusters within the data (later in the machine learning section we will learn how to formally cluster data).

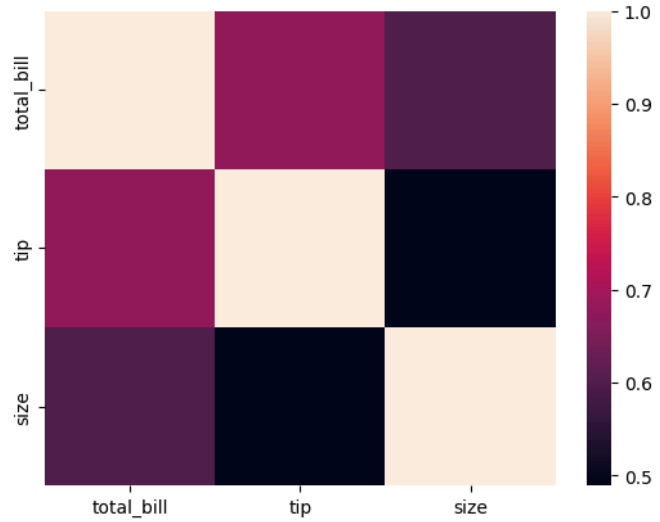
Heatplot

In order for a heatmap to work properly, your data should already be in a matrix form, the sns.heatmap function basically just colors it in for you. For example:

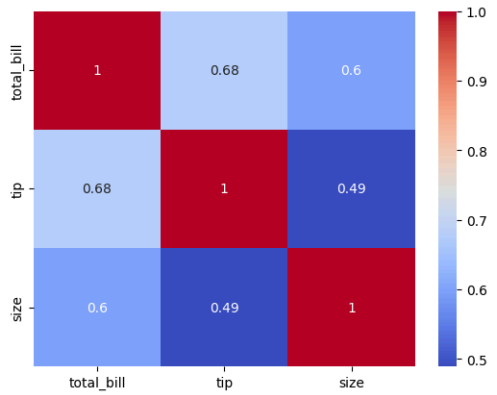
import seaborn as sns
%matplotlib inline
flights = sns.load_dataset('flights')
tips = sns.load_dataset('tips')
# Matrix form for correlation data
tips.corr() #para que un mapa de calor funcione correctamente sus datos deben estar en forma de matriz
Seleccionar solo las columnas numéricas
numerical_tips = tips.select_dtypes(include='number')
Mostrar la matriz de correlación de las columnas numéricas de 'tips'
correlation_matrix = numerical_tips.corr()
correlation_matrix

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

sns.heatmap(correlation_matrix)



sns.heatmap(correlation_matrix,c**map='coolwarm',annot=True**)
plt.title('titanic.corr()')



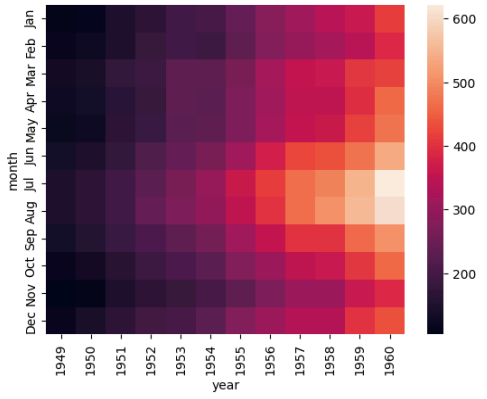
#FLIGHTS

flights.pivot_table(values='passengers',index='month',columns='year')

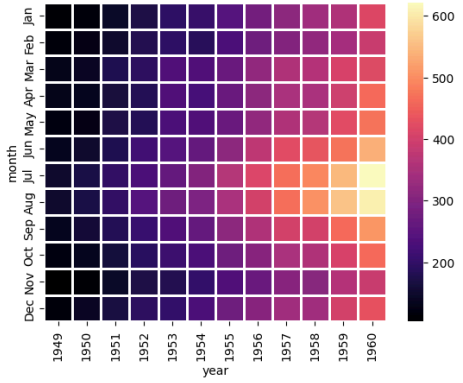
year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
June	135	149	178	218	243	264	315	374	422	435	472	535
July	148	170	199	230	264	302	364	413	465	491	548	622
August	148	170	199	242	272	293	347	405	467	505	559	606
September	136	158	184	209	237	259	312	355	404	404	463	508
October	119	133	162	191	211	229	274	306	347	359	407	461
November	104	114	146	172	180	203	237	271	305	310	362	390
December	118	140	166	194	201	229	278	306	336	337	405	432

pvflights = flights.pivot_table(values='passengers',index='month',columns='year')

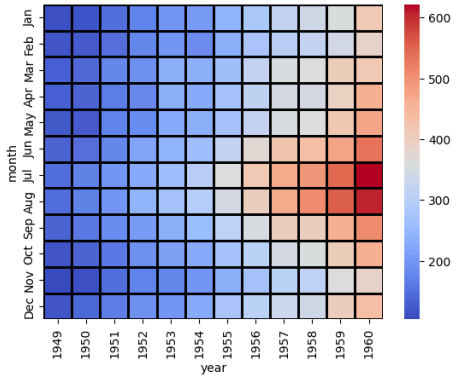
sns.heatmap(pvflights)



sns.heatmap(pvflights,**cmap='magma',linecolor='white',linewidths=1)**

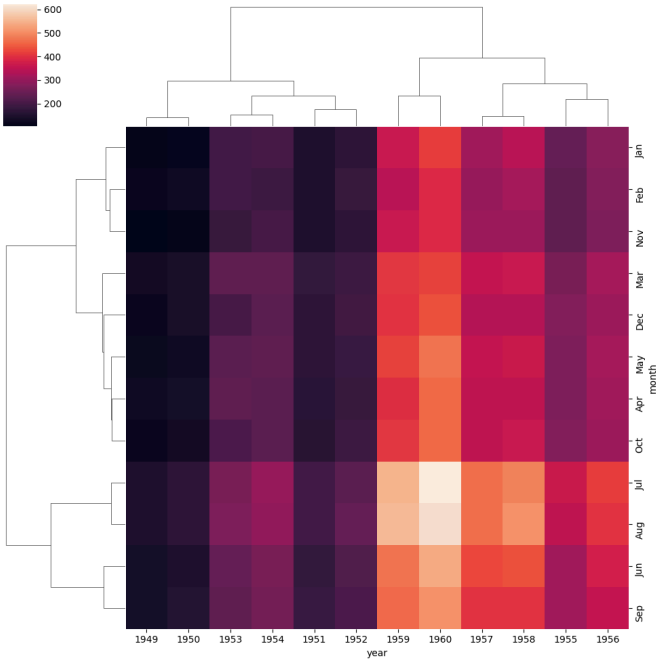


sns.heatmap(pvflights,**cmap='coolwarm',linecolor='black',linewidths=1**)



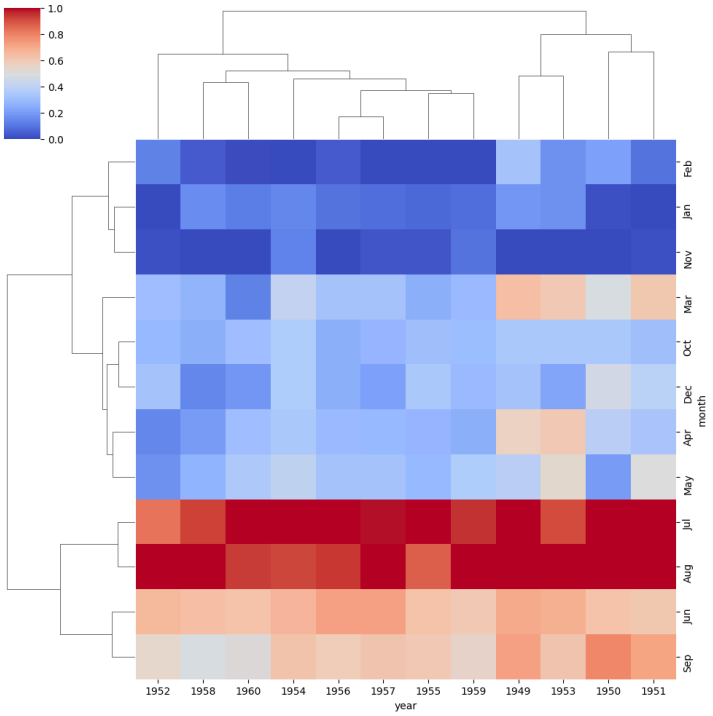
Clustermap

The clustermap uses hierarchal clustering to produce a clustered version of the heatmap.
#Básicamente intenta agrupar columnas y filas por similitud
sns.**clustermap(pvflights)**



Notice now how the years and months are no longer in order, instead they are grouped by similarity in value (passenger count). That means we can begin to infer things, such as August and July being similar (makes sense, since they are both summer travel months)

# More options to get the information a little clearer like normalization
sns.clustermap(pvflights,**cmap='coolwarm',standard_scale=1**)

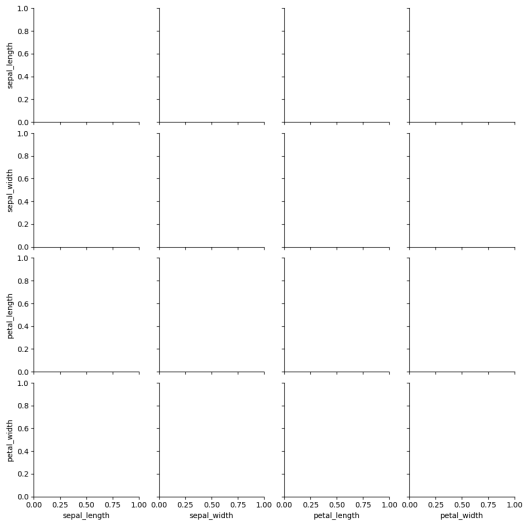


Grids

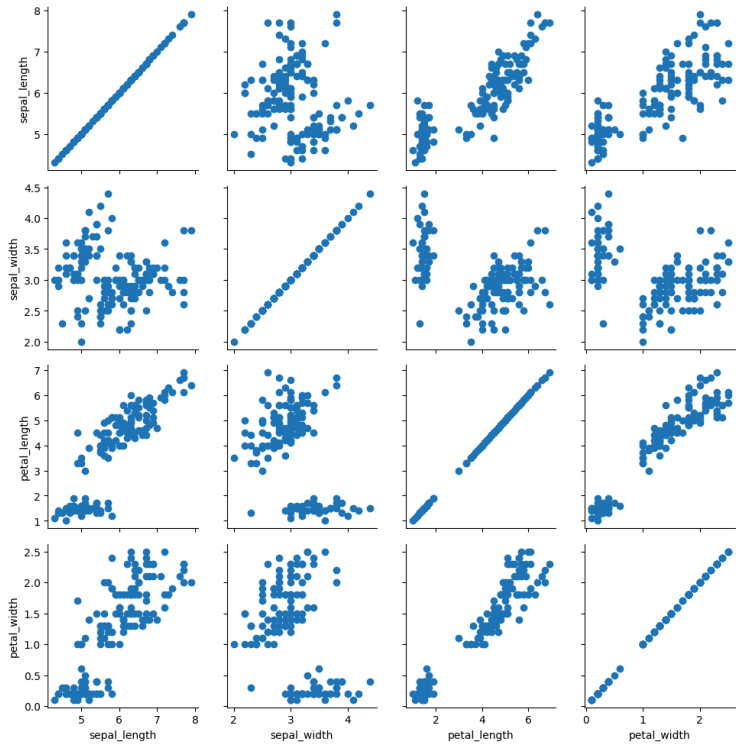
Grids are general types of plots that allow you to map plot types to rows and columns of a grid, this helps you create similar plots separated by features.

PairGrid

Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.
iris = sns.load_dataset('iris')
sns.PairGrid(iris) # Just the Grid



# Then you map to the grid
g = sns.PairGrid(iris)
g.map(plt.scatter)



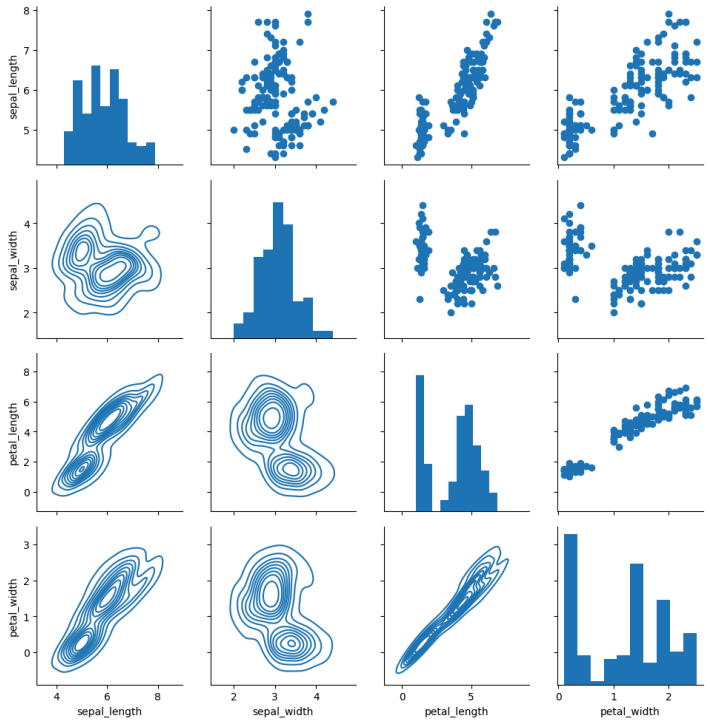
Map to upper,lower, and diagonal

g = sns.PairGrid(iris)

g.map_diag(plt.hist) #que tipos de graficos van en la diagonal, puede ser distplot

g.map_upper(plt.scatter) #que tipos de graficos van en parte superior de la diagonal

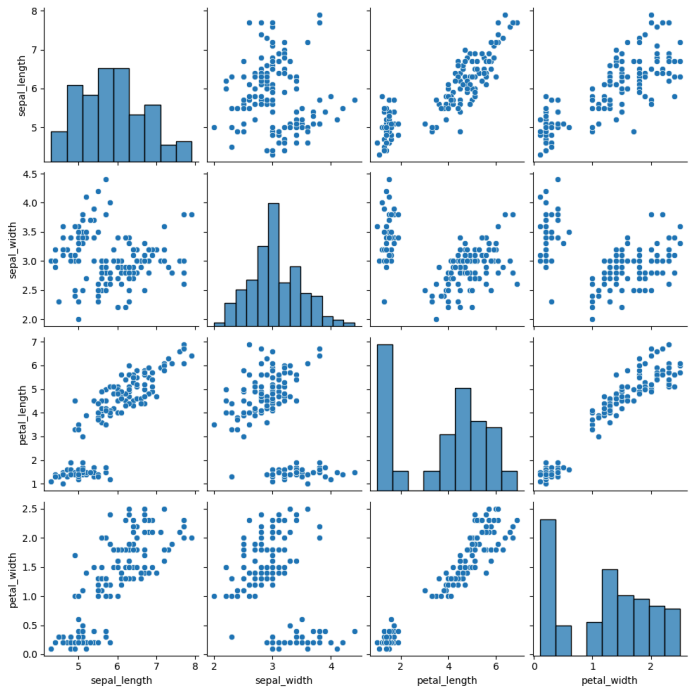
g.map_lower(sns.kdeplot) #que tipos de graficos van en parte inferior de la diagonal



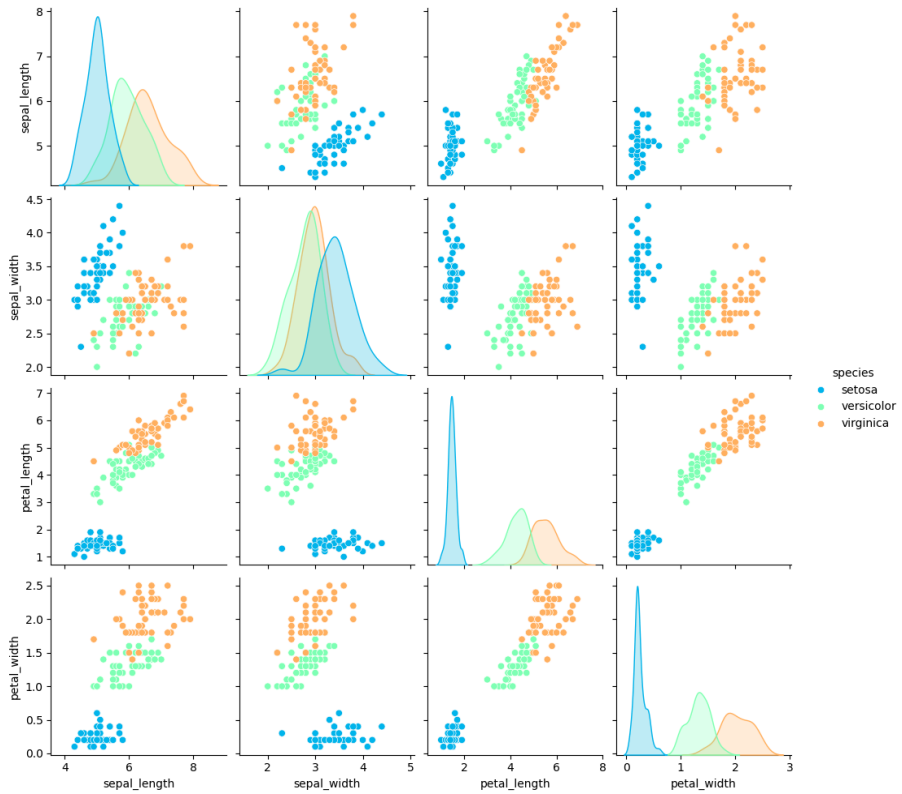
Pairplot

Pairplot is a simpler version of PairGrid (you'll use quite often)

sns.pairplot(iris)



sns.pairplot(iris,**hue='species',palette='rainbow'**)

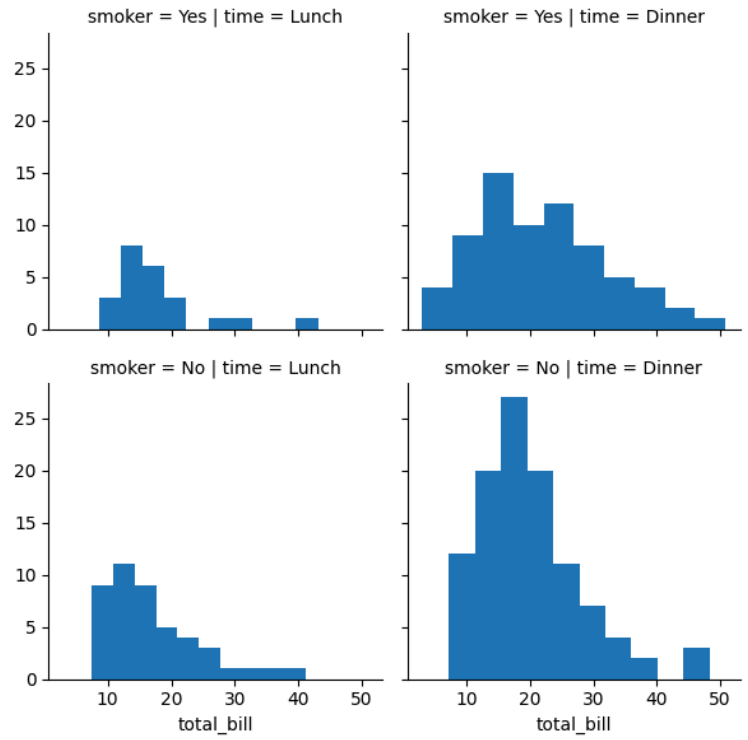


FacetGrid

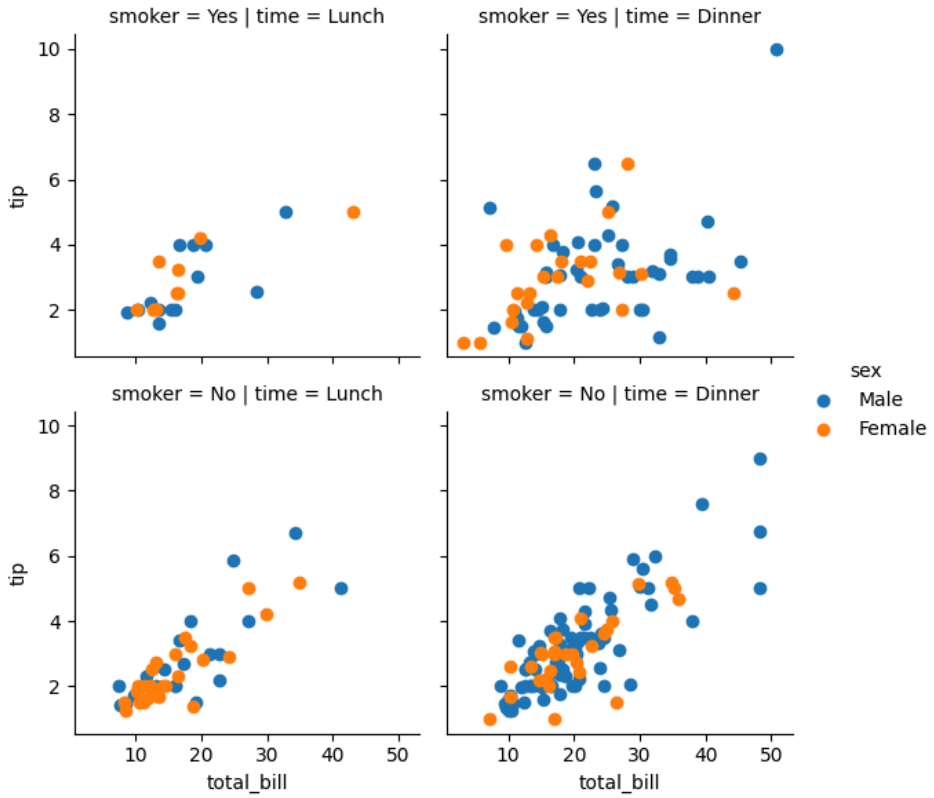
tips = sns.load_dataset('tips')

g = sns.FacetGrid(tips, col="time", row="smoker") # Just the Grid

g = g.map(plt.hist, "total_bill")

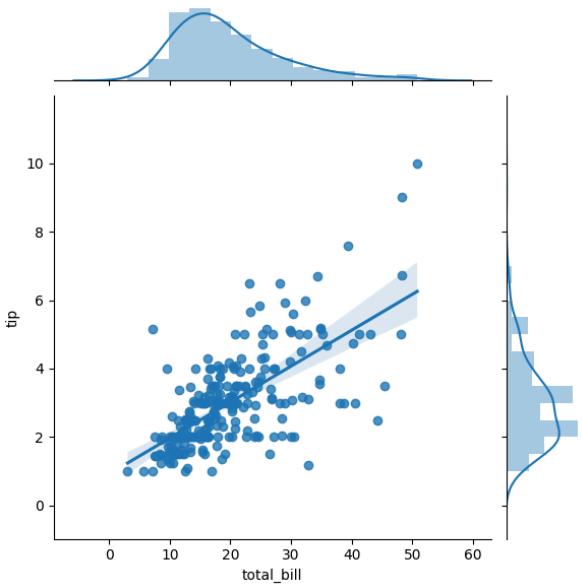


g = sns.FacetGrid(tips, col="time", row="smoker",hue='sex')
 # Notice hwo the arguments come after plt.scatter call
g = g.map(plt.scatter, "total_bill", "tip").add_legend() #necesitas 2 variables p un scatter



JointGrid

JointGrid is the general version for jointplot() type grids
g = sns.JointGrid(x="total_bill", y="tip", data=tips) #Esto solo arma el marco, el cuadrado exterior
g = g.plot(sns.regplot, sns.distplot)

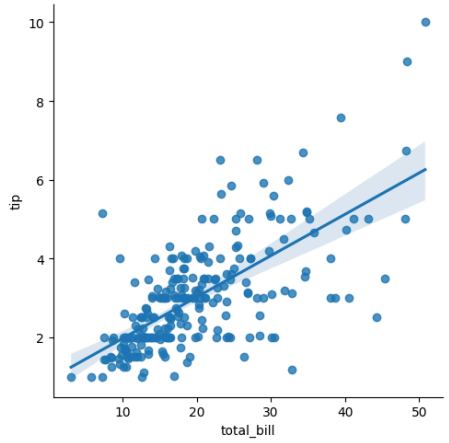


Regression Plots

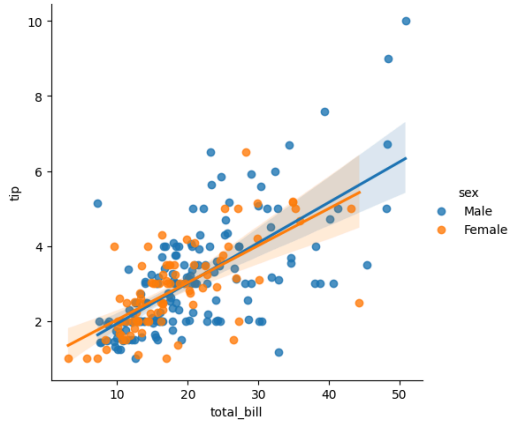
Seaborn has many built-in capabilities for regression plots, now we will only cover the **lmplot()** function. **Lmplot** allows you to display linear models, but it also conveniently allows you to split up those plots based off of features, as well as coloring the hue based off of features.

Lmplot

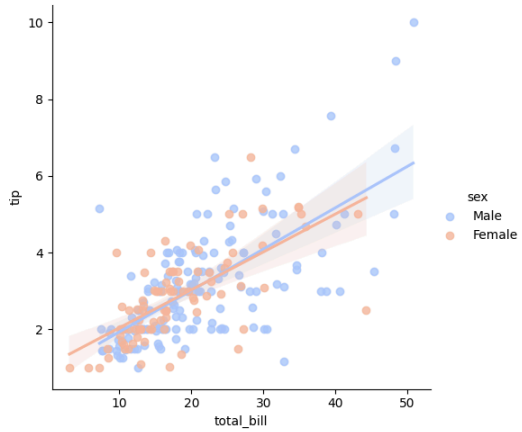
import seaborn as sns
tips = sns.load_dataset('tips')
sns.lmplot(x='total_bill',y='tip',data=tips)



sns.lmplot(x='total_bill',y='tip',data=tips,**hue='sex'**)

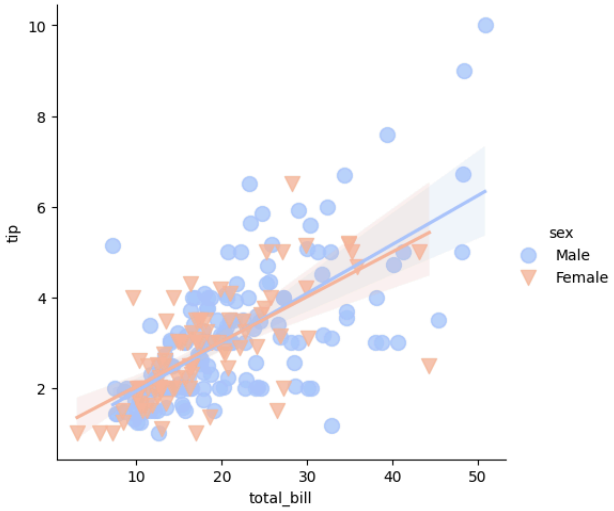


sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex'**,palette='coolwarm'**)

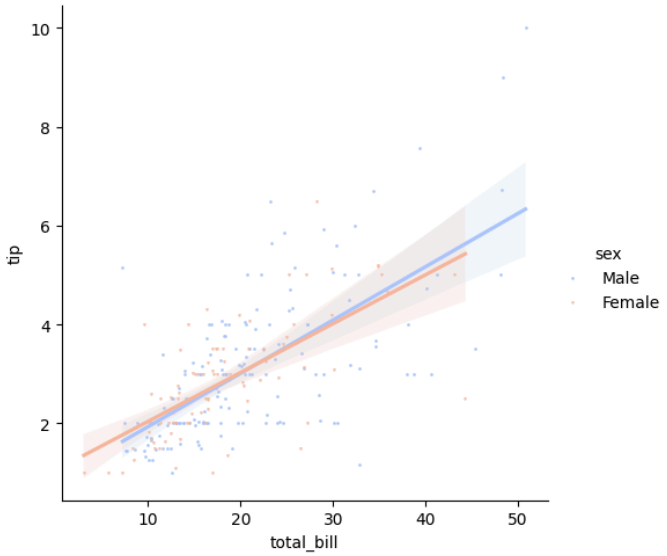


Working with Markers: Lmplot kwargs get passed through to **regplot** which is a more general form of lmplot(). regplot has a **scatter_kws** parameter that gets passed to plt.scatter. So you want to set the s parameter in that dictionary, which corresponds (a bit confusingly) to the squared markersize. In other words you end up passing a dictionary with the base matplotlib arguments, in this case, s for size of a scatter plot. In general, you probably won't remember this off the top of your head, but instead reference the documentation http://matplotlib.org/api/markers_api.html

sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm',**markers=['o','v'],scatter_kws={'s':100}**)
#scatter_kws le pone "zoom" por así decirlo

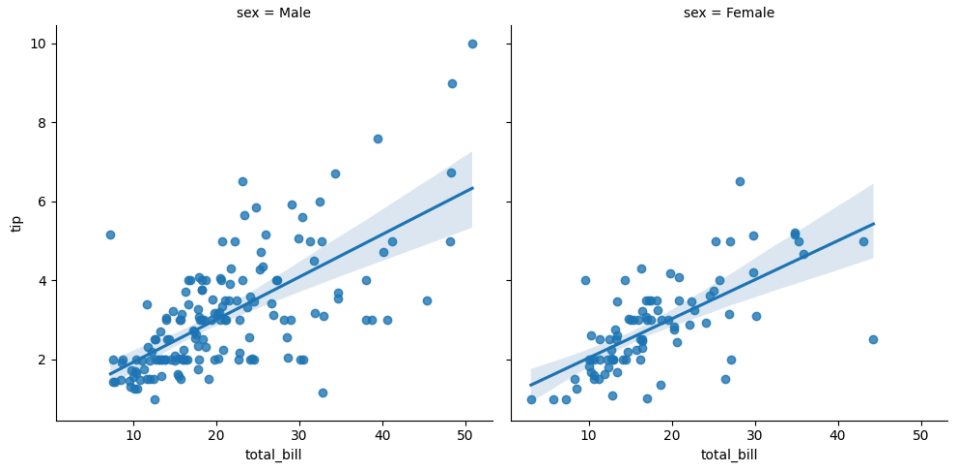


sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm',**markers=['o','v'],scatter_kws={'s':1}**)

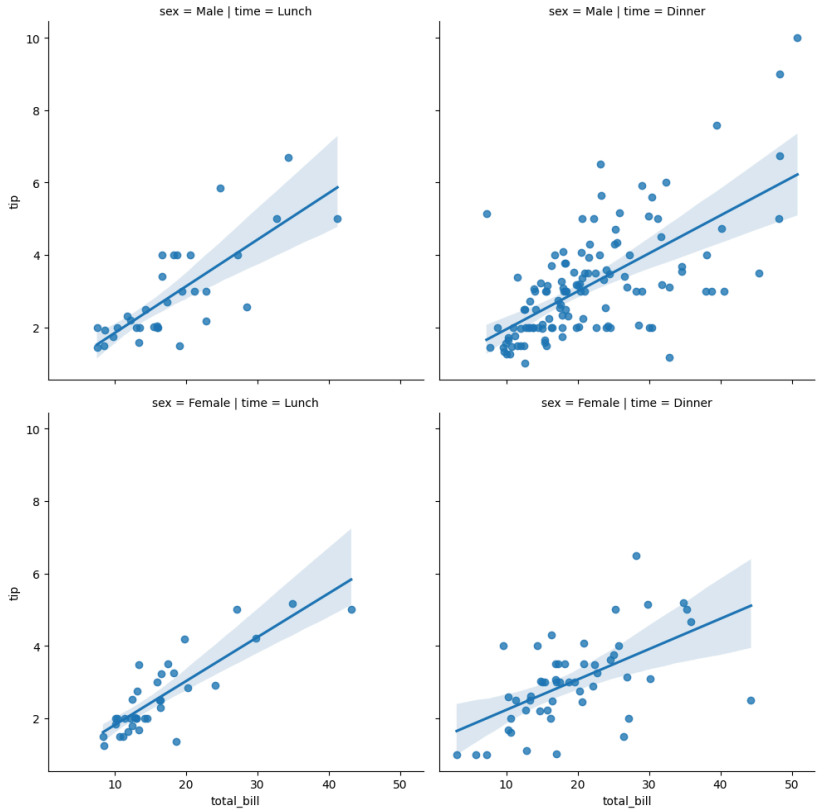


Using a Grid: We can add more variable separation through columns and rows with the use of a grid. Just indicate this with the col or row arguments:

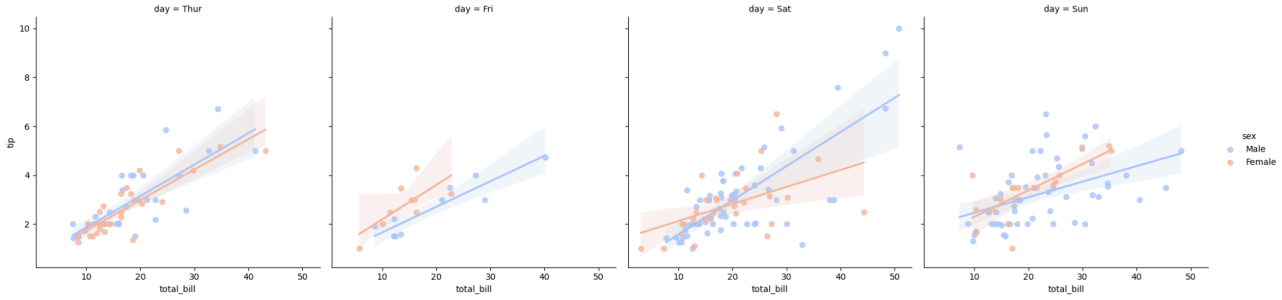
sns.lmplot(x='total_bill',y='tip',data=tips,**col='sex'**) #columnas según sexto



sns.lmplot(x="total_bill", y="tip", r**ow="sex", col="time"**,data=tips) #filas sexo y columnas time

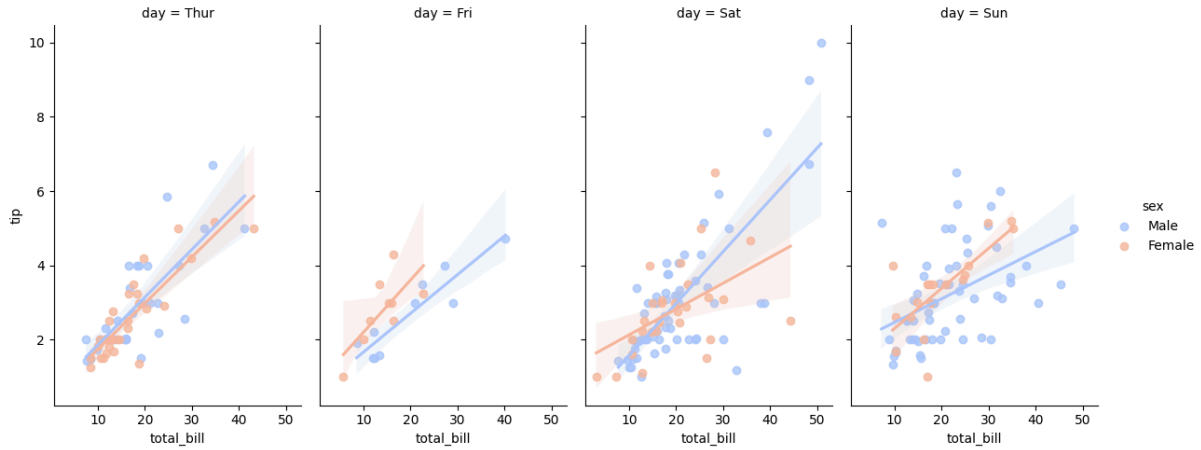


sns.lmplot(x='total_bill',y='tip',data=tips,**col='day',hue='sex',palette='coolwarm'**) #no filas



Aspect and Size: Seaborn figures can have their size and aspect ratio adjusted with the **size** and **aspect** parameters

sns.lmplot(x='total_bill',y='tip',data=tips,col='day',hue='sex',palette='coolwarm', **aspect=0.6**)

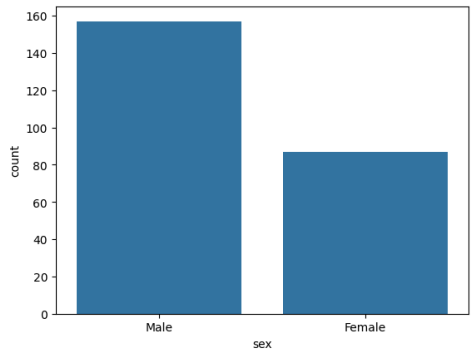


Style and Color

We've shown a few times how to control figure aesthetics in seaborn, but let's now go over it formally.

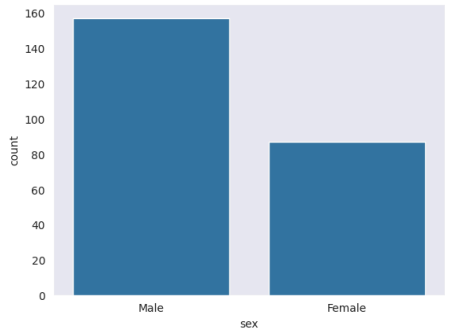
Styles

```
import seaborn as sns
import matplotlib.pyplot as plt
tips = sns.load_dataset('tips')
sns.countplot(x='sex',data=tips)
```

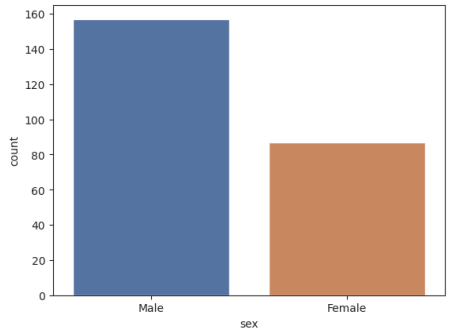


sns.set_style('dark') #styles = **["white", "dark", "whitegrid", "darkgrid", "ticks"]**

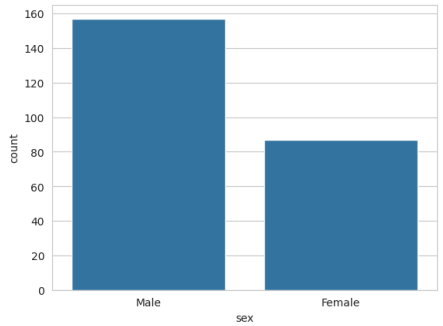
```
sns.countplot(x='sex',data=tips)
```



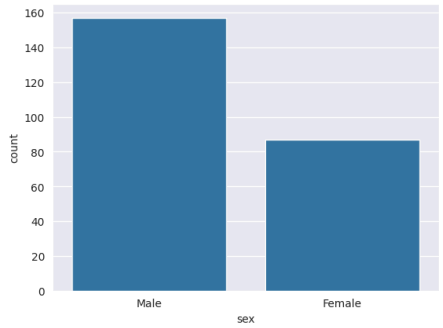
sns.set_style('ticks') #pone las rayitas al lado de los nros del eje
sns.countplot(x='sex',data=tips,**palette='deep'**)



sns.set_style('whitegrid')
sns.countplot(x='sex',data=tips)

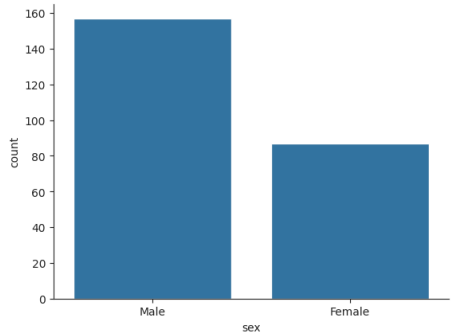


sns.set_style('darkgrid')
sns.countplot(x='sex',data=tips)

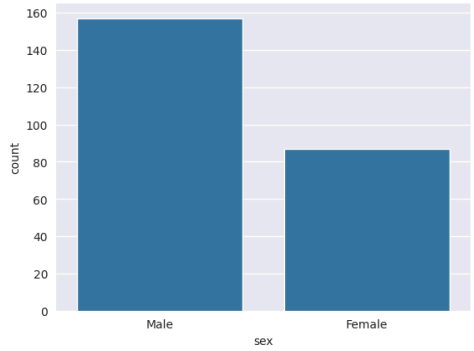


Spine Removal

sns.countplot(x='sex',data=tips)
sns.despine() #elimina la cuadricula superior y derecha



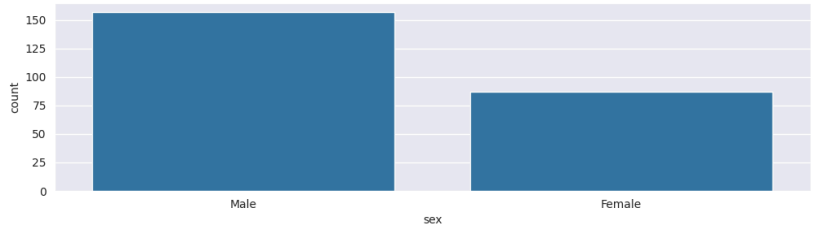
sns.countplot(x='sex',data=tips)
sns.despine(left=True, bottom=True) #elimina la cuadricula superior y derecha + la izquierda e inferior



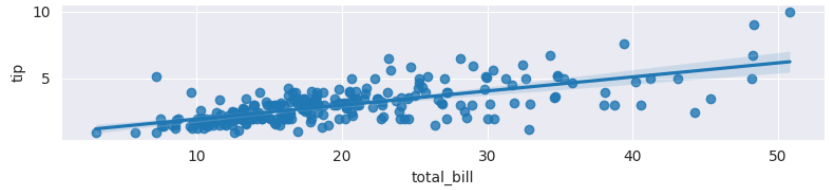
Size and Aspect

You can use matplotlib's **plt.figure(figsize=(width,height)** to change the size of most seaborn plots. You can control the size and aspect ratio of most seaborn grid plots by passing in parameters: size, and aspect.

# Non Grid Plot
plt.figure(**figsize=(12,3)**)
sns.countplot(x='sex',data=tips)

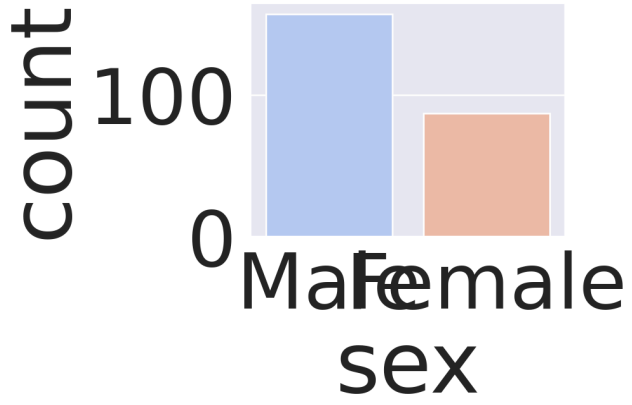


# Grid Type Plot
sns.lmplot(x='total_bill',y='tip'**,height=2,aspect=4**,data=tips)



Scale and Context

The set_context() allows you to override default parameters
sns.set_context('poster',font_scale=4)
sns.countplot(x='sex',data=tips,palette='coolwarm')



Check out the documentation page for more info on these topics:
https://stanford.edu/~mwaskom/software/seaborn/tutorial/aesthetics.html

Section 10: Pandas Built-in Data Visualization

Style Sheets

Matplotlib has style sheets (http://matplotlib.org/gallery.html#style_sheets) you can use to make your plots look a little nicer. These style sheets include plot_bmh, plot_fivethirtyeight, plot_ggplot and more. They basically create a set of style rules that your plots follow. I recommend using them, they make all your plots have the same look and feel more professional. You can even create your own if you want your company's plots to all have the same look (it is a bit tedious to create on though).

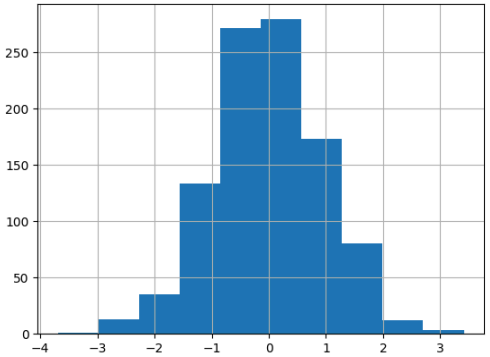
Plot Types

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df1 = pd.read_csv('df1',index_col=0)
df2 = pd.read_csv('df2')
```

##Before plt.style.use() your plots look like this:

df1['A'].hist()

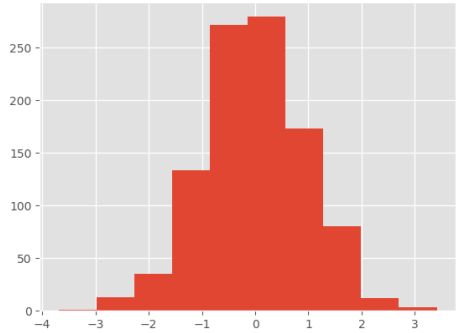
##otra forma es df1['A'].plot(kind='hist')



plt.style.use('ggplot')

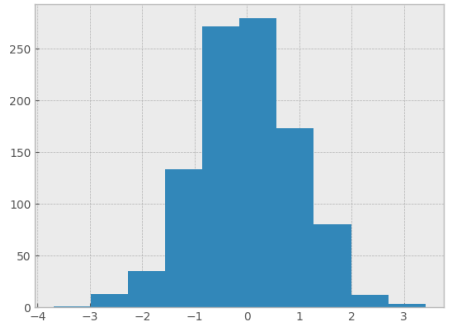
##Now your plots look like this:

df1['A'].hist()



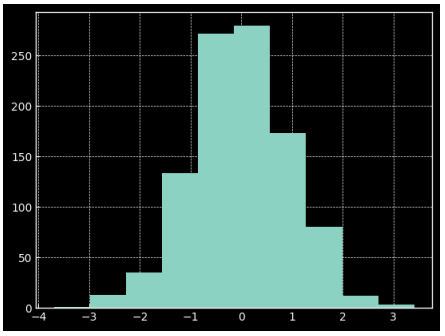
plt.style.use('bmh')

df1['A'].hist()



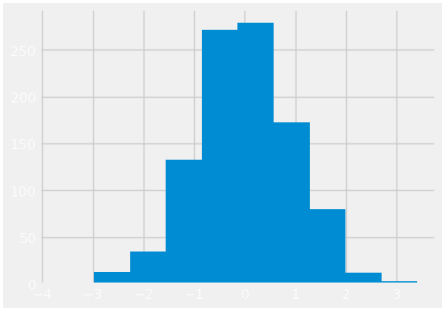
plt.style.use('dark_background')

df1['A'].hist()



plt.style.use('fivethirtyeight')

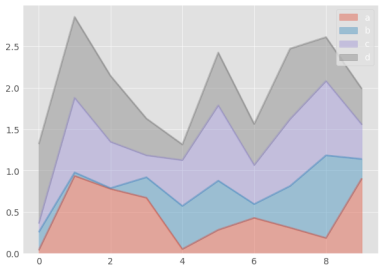
df1['A'].hist()



plt.style.use('ggplot')

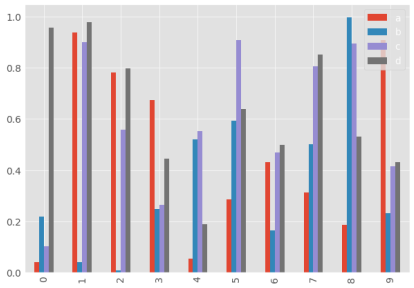
Plot Types: Area

- df.plot.area
    - df2.plot.area(alpha=0.4)

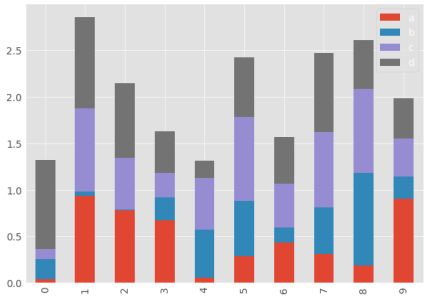


Plot Types: Barplot

- df.plot.barplot
    - df2.plot.bar()

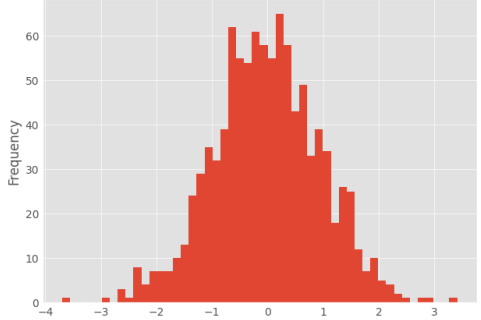


    - df2.plot.bar(stacked=True)



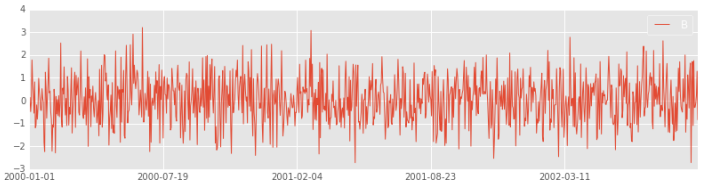
Plot Types: Histogram

- df.plot.hist
    - df1['A'].plot.hist(alpha=0.5,bins=25) #alpha=0.5 le da transparencia



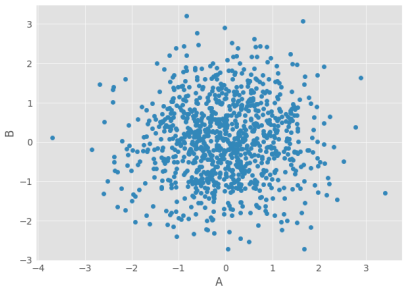
Plot Types: Line Plot

- df.plot.line
    - df1.plot.line(x=df1.index,y='B',figsize=(12,3),lw=1)

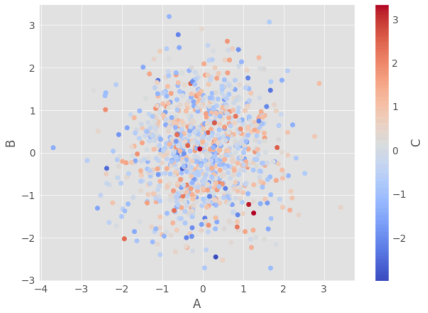


Plot Types: Scatter Plot

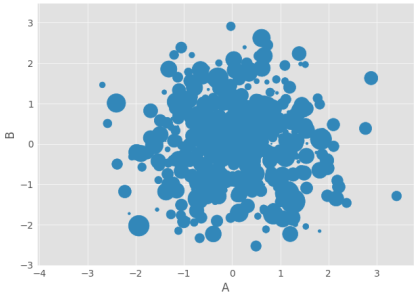
- df.plot.scatter
    - df1.plot.scatter(x='A',y='B') #You can use c to color based off another column value. Use cmap to indicate colormap to use. For all the colormaps, check out: http://matplotlib.org/users/colormaps.html



    - df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm')

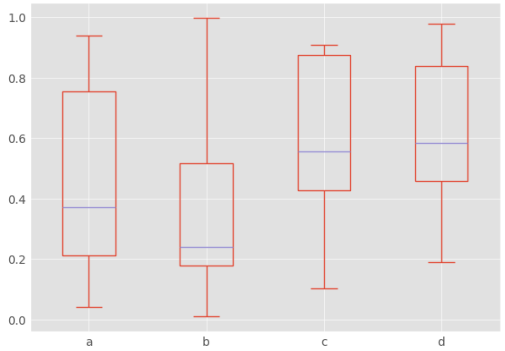


    - df1.plot.scatter(x='A',y='B',s=df1['C']*200) #Or use s to indicate size based off another column. s parameter needs to be an array, not just the name of a column:



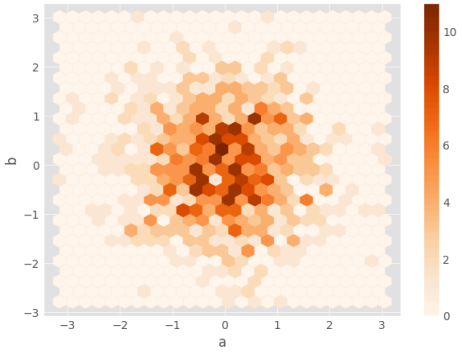
Plot Types: Box Plot

- df.plot.box
    - df2.plot.box() #Can also pass a by= argument for groupby



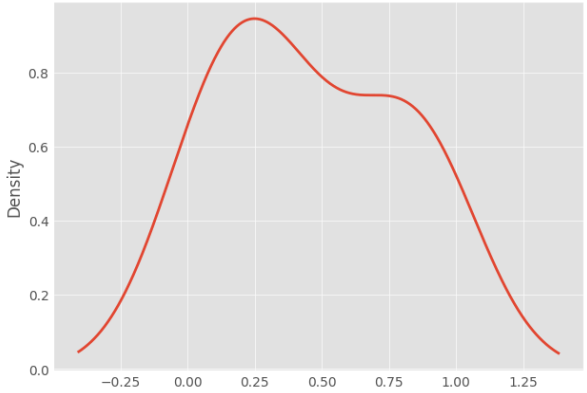
Plot Types: Hexagonal Bin Plot

- df.plot.hexbin
    - #Useful for Bivariate Data, alternative to scatterplot:
      df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])
      df.plot.hexbin(x='a',y='b',gridsize=25,cmap='Oranges')

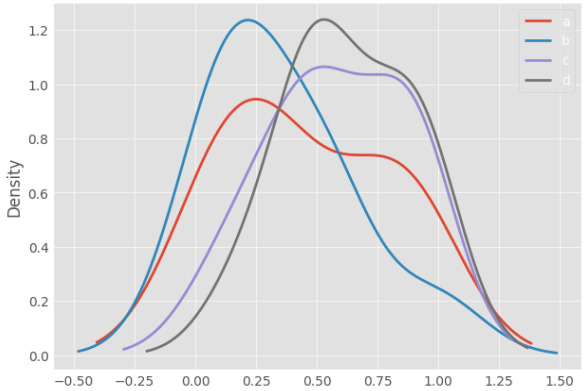


Plot Types: Kernel Density Estimation plot (KDE)

- df.plot.kde
    - df2['a'].plot.kde()



    - df2.plot.density()



- df.plot.density
- df.plot.pie
- df.plot.barh

You can also just call df.plot(kind='hist') or replace that kind argument with any of the key terms shown in the list above (e.g. 'box','barh', etc..)

Section 11: Plotly and Cufflinks

Introduction and Installation

Plotly is a library that allows you to create **interactive plots / gráficos interactivos** that you can use in **dashboards** or **websites** (you can save them as html files or static images). Cufflinks connect plotply with pandas. https://plotly.com/ https://plotly.com/python/ https://github.com/santosjorge/cufflinks

In order for this all to work, you'll need to install plotly and cufflinks to call plots directly off of a pandas dataframe. These libraries are not currently available through conda but are available through pip. Install the libraries at your command line/terminal using:

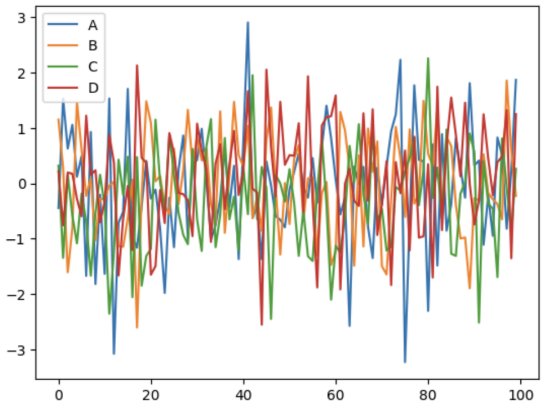
- pip install plotly
- pip install cufflinks

```
import pandas as pd
import numpy as np
%matplotlib inline

from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
print(__version__) # requires version >= 1.9.0
```

5.22.0

```
import cufflinks as cf

# For Notebooks
init_notebook_mode(connected=True)
```

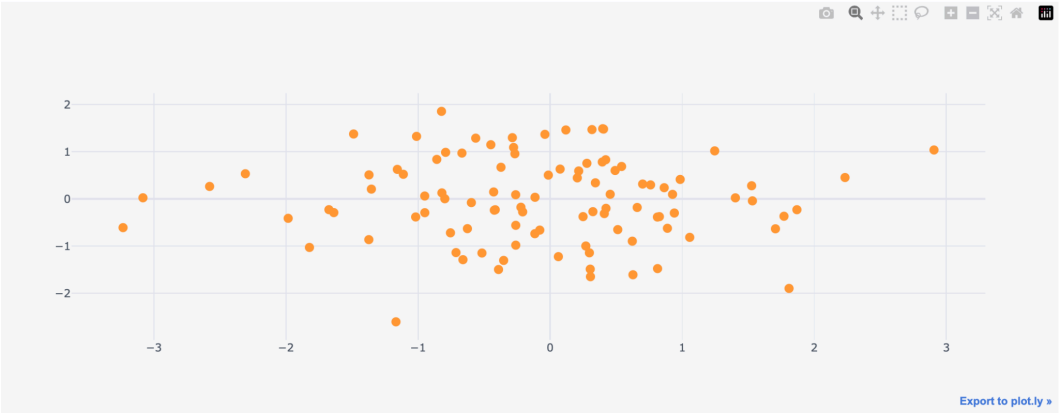
```
# For offline use
cf.go_offline()

df = pd.DataFrame(np.random.randn(100,4),columns='A B C D'.split())
df2 = pd.DataFrame({'Category':['A','B','C'],'Values':[32,43,50]})

df.plot()
```



df.iplot()

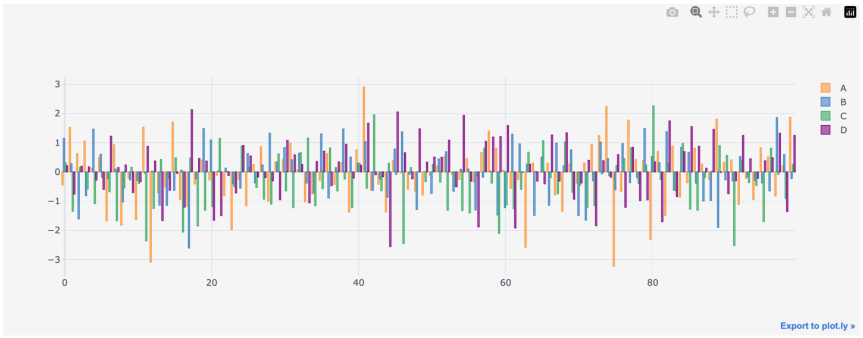
Scatter

```
df.iplot(kind='scatter',x='A',y='B',mode='markers',size=10)
```

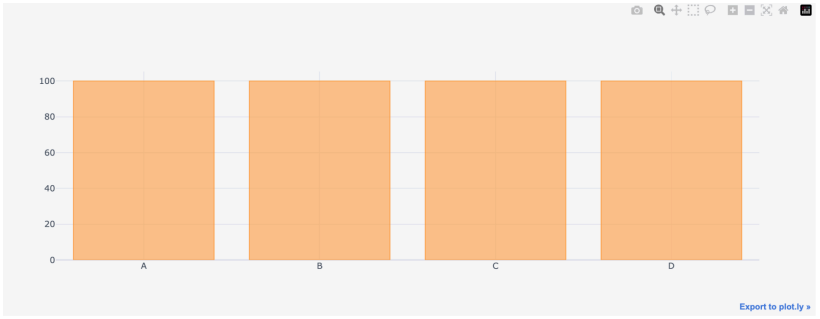


Barplots

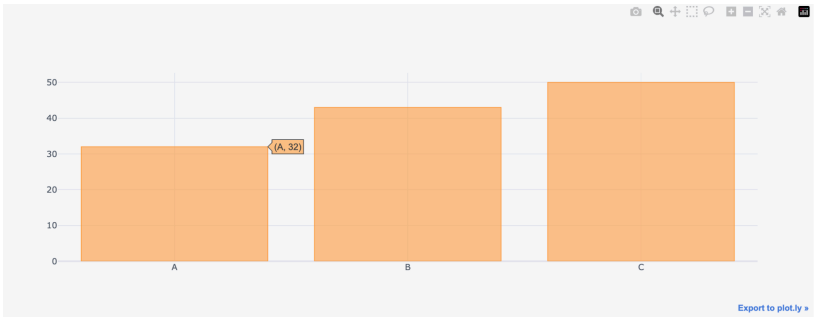
df.iplot(kind='bar')



df.count().iplot(kind='bar')

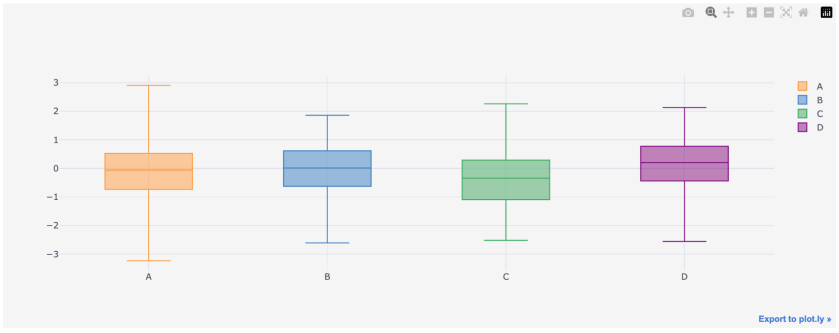


df2.iplot(kind='bar',x='Category',y='Values')



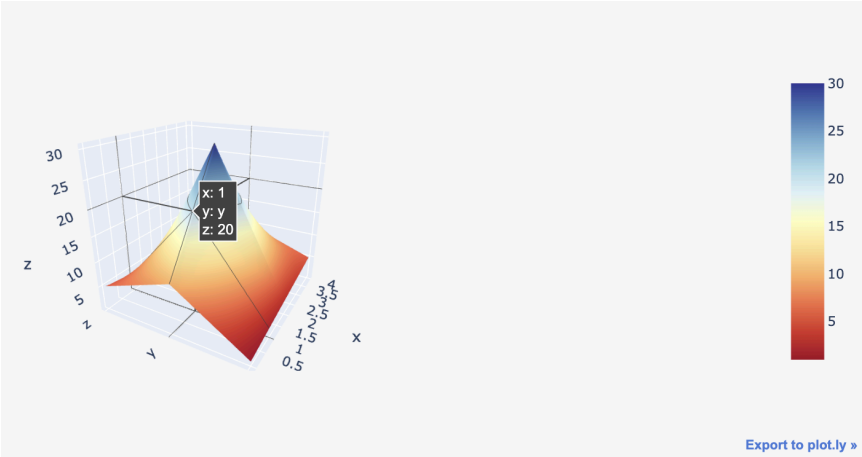
Boxplots

df.iplot(kind='box')



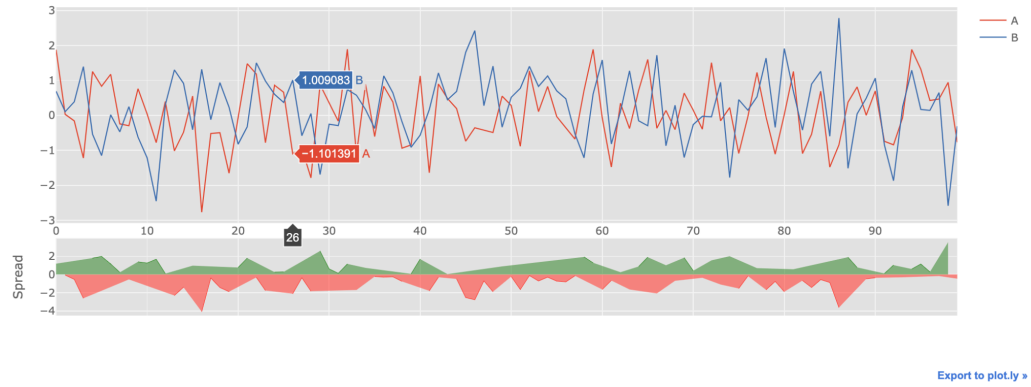
3d Surface

df3 = pd.DataFrame({'x':[1,2,3,4,5],'y':[10,20,30,20,10],'z':[5,4,3,2,1]})
df3.iplot(kind='surface',colorscale='rdylbu')



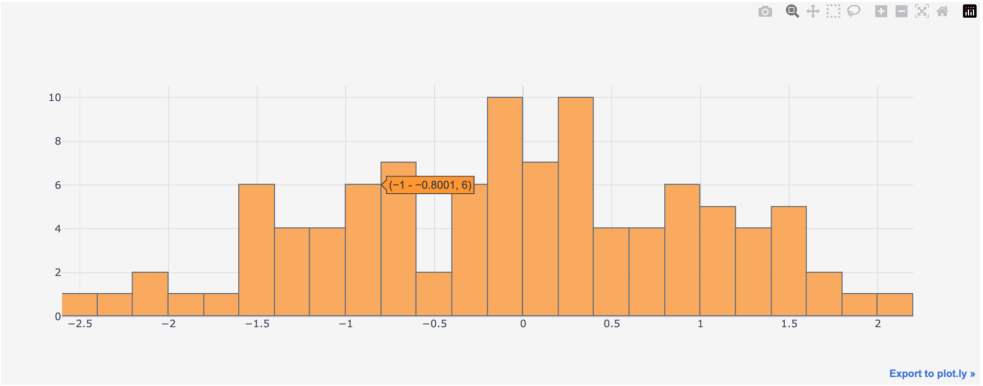
Spread

df[['A','B']].iplot(kind='spread')

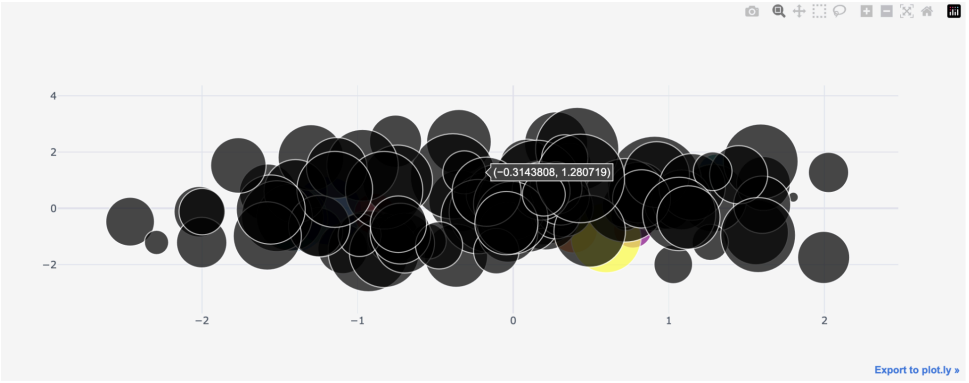


Histogram

df['A'].iplot(kind='hist',bins=25)

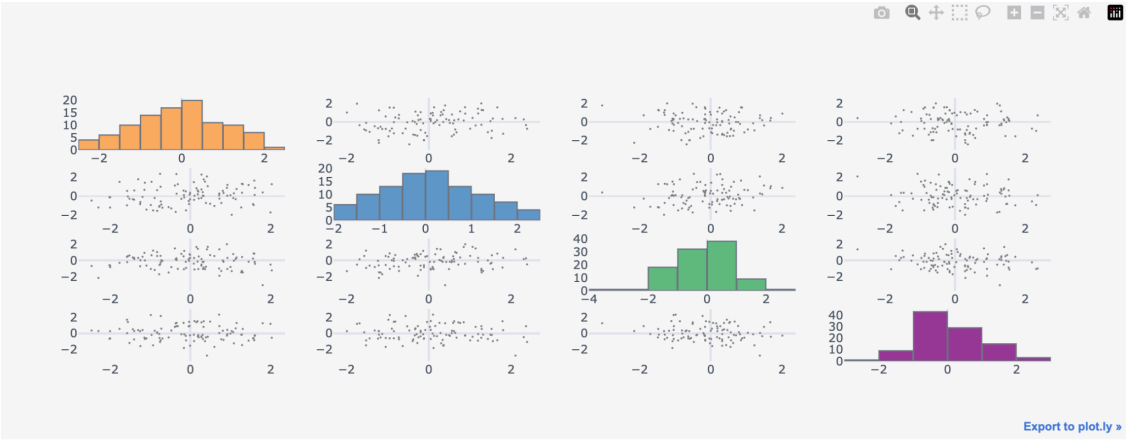


df.iplot(kind='bubble',x='A',y='B',size='C')



Scatter Matrix

df.scatter_matrix()



Section 12: Geographical Plotting

Introduction

Geographical plotting is usually challenging to due the various formats the data can come in. We will focus on using plotly for plotting. Matplotlib also has a basemap extension.

Choropleth Maps

Offline Plotly Usage

pip install chart-studio

import pandas as pd
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

init_notebook_mode(connected=True) #para ver las figuras en el notebook

#More info on other options for Offline Plotly usage can be found here https://plot.ly/python/offline/ .

Choropleth US Maps

Plotly's mapping can be a bit hard to get used to at first, remember to reference the cheat sheet in the data visualization folder, or find it online https://images.plot.ly/plotly-documentation/images/python_cheat_sheet.pdf.

Now we need to begin to build our data dictionary. Easiest way to do this is to use the **dict()** function of the general form:

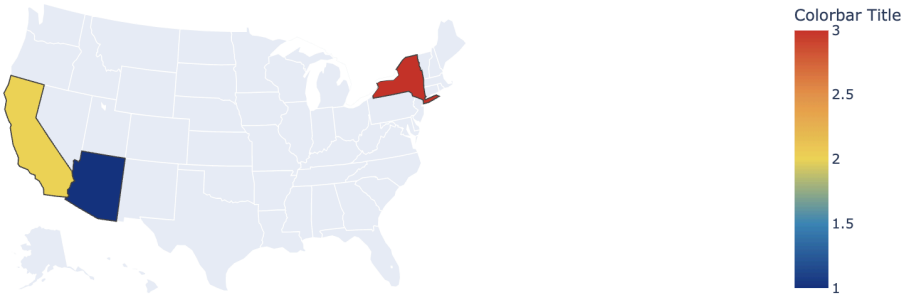
- type = 'choropleth',
- locations = list of states
- locationmode = 'USA-states'
- colorscale= 'pairs' | 'Greys' | 'Greens' | 'Bluered' | 'Hot' | 'Picnic' | 'Portland' | 'Jet' | 'RdBu' | 'Blackbody' | 'Earth' | 'Electric' | 'YIOrRd' | 'YIGnBu' or create a [custom colorscale](https://plot.ly/python/heatmap-and-contour-colorscales/)
- text= list or array of text to display per point
- z= array of values on z axis (color of state)
- colorbar = {'title':'Colorbar Title'})

Here is a simple example:

```
data = dict(type = 'choropleth',
        locations = ['AZ','CA','NY'],
        locationmode = 'USA-states',
        colorscale= 'Portland', #Jet, Greens son otras opciones y tmb  'tealrose', 'tempo', 'temps', 'thermal', 'tropic', 'turbid', 'turbo', 'twilight', 'viridis', 'ylgn', 'ylgnbu', 'ylorbr','ylorrd'
        text= ['text1','text2','text3'],
        z=[1.0,2.0,3.0],
        colorbar = {'title':'Colorbar Title'})

layout = dict(geo = {'scope':'usa'})
```

Then we use: go.Figure(data = [data],layout = layout) to set up the object that finally gets passed into iplot()

```
choromap = go.Figure(data = [data],layout = layout)

iplot(choromap)
```

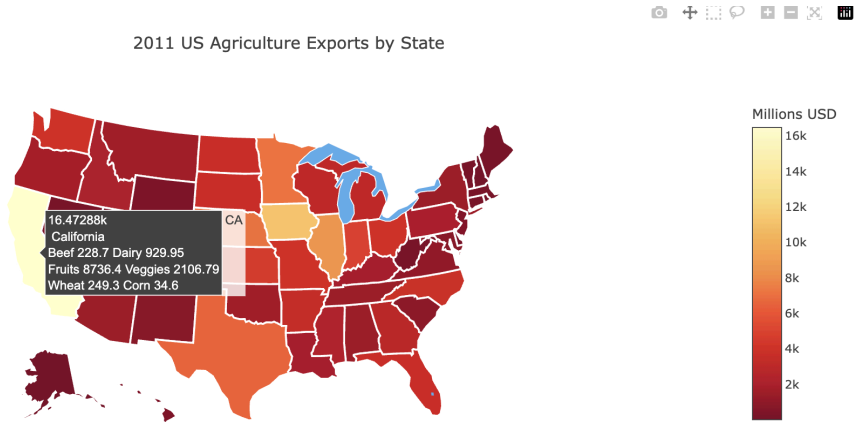


Real Data Choropleth US Maps

```
df = pd.read_csv('2011_US_AGRI_Exports')
df.head()
```

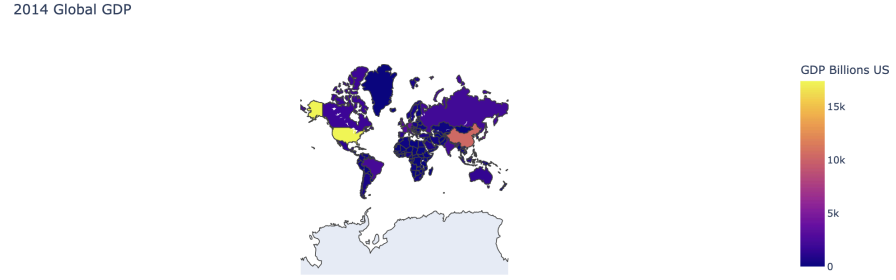
	code	state	category	total exports	beef	pork	poultry	dairy	fruits fresh	fruits proc	total fruits	veggies fresh	veggies proc	total veggies	corn	wheat	cotton	text
0	AL	Alabama	state	1390.63	34.4	10.6	481.0	4.06	8.0	17.1	25.11	5.5	8.9	14.33	34.9	70.0	317.61	Alabama Beef 34.4 Dairy 4.06 Fruits 25.1...
1	AK	Alaska	state	13.31	0.2	0.1	0.0	0.19	0.0	0.0	0.00	0.6	1.0	1.56	0.0	0.0	0.00	Alaska Beef 0.2 Dairy 0.19 Fruits 0.0 Ve...
2	AZ	Arizona	state	1463.17	71.3	17.9	0.0	105.48	19.3	41.0	60.27	147.5	239.4	386.91	7.3	48.7	423.95	Arizona Beef 71.3 Dairy 105.48 Fruits 60...
3	AR	Arkansas	state	3586.02	53.2	29.4	562.9	3.53	2.2	4.7	6.88	4.4	7.1	11.45	69.5	114.5	665.44	Arkansas Beef 53.2 Dairy 3.53 Fruits 6.8...
4	CA	California	state	16472.88	228.7	11.1	225.4	929.95	2791.8	5944.6	8736.40	803.2	1303.5	2106.79	34.6	249.3	1064.95	California Beef 228.7 Dairy 929.95 Frui...

```
data = dict(type='choropleth',
            colorscale = 'YlOrRd',
            locations = df['code'],
            z = df['total exports'],
            locationmode = 'USA-states',
            text = df['text'],
            marker = dict(line = dict(color = 'rgb(255,255,255)',width = 2)),
            colorbar = {'title':"Millions USD"}
            )

layout = dict(title = '2011 US Agriculture Exports by State',
              geo = dict(scope='usa',
                         showlakes = True,
                         lakecolor = 'rgb(85,173,240)')
             )

choromap = go.Figure(data = [data],layout = layout)
iplot(choromap)
```



World Choropleth Map

```
df = pd.read_csv('2014_World_GDP')
df.head()

data = dict(
        type = 'choropleth',
        locations = df['CODE'],
        z = df['GDP (BILLIONS)'],
        text = df['COUNTRY'],
        colorbar = {'title' : 'GDP Billions US'},
      )

layout = dict(
    title = '2014 Global GDP',
    geo = dict(
        showframe = False,
        projection = {'type':'mercator'}
         # mollweide', 'sinusoidal', 'stereographic', 'times',
         #'transverse mercator', 'van der grinten', 'van der grinten2',
         # 'van der grinten3', 'van der grinten4', 'wagner4', 'wagner6', 'wiechel', 'winkel tripel',
    )
)

choromap = go.Figure(data = [data],layout = layout)
iplot(choromap)
```



Section 13: Data Capstone Project

Conocer el tipo de dato de 1 columna

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

type(df['timeStamp'].iloc[0])
```

Pasar un campo de tipo string a tipo fecha

```
df['timeStamp'] = pd.to_datetime(df['timeStamp'],format='%Y-%m-%d %H:%M:%S')
#df['timeStamp'] = pd.to_datetime(df['timeStamp'])
```

Obtener hora, mes y día de la semana

```
df['Hour'] = df['timeStamp'].apply(lambda time: time.hour)

df['Month'] = df['timeStamp'].apply(lambda time: time.month)

df['Day of Week'] = df['timeStamp'].apply(lambda time: time.dayofweek)
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
df['Day of Week'] = df['Day of Week'].map(dmap)

df.groupby('Date').count()['twp'].plot()
```

Propiedad para que se vean bien los datos en el eje x

```
plt.tight_layout()
```

Hacer una matriz cuando no tenés un campo con los nros que queres mostrar en la matriz sino que queres contar

```
dayHour = df.groupby(by=['Day of Week','Hour']).count()['Reason'].unstack()
dayHour.head()
```

Pandas datareader

```
from pandas_datareader import data, wb
import pandas as pd
import numpy as np
import datetime
%matplotlib inline
```

```
start = datetime.datetime(2006, 1, 1)
end = datetime.datetime(2016, 1, 1)

# Bank of America
BAC = data.DataReader("BAC", 'google', start, end)
# CitiGroup
C = data.DataReader("C", 'google', start, end)
# Goldman Sachs
GS = data.DataReader("GS", 'google', start, end)
# JPMorgan Chase
JPM = data.DataReader("JPM", 'google', start, end)
# Morgan Stanley
MS = data.DataReader("MS", 'google', start, end)
# Wells Fargo
WFC = data.DataReader("WFC", 'google', start, end)

# Could also do this for a Panel Object
df = data.DataReader(['BAC', 'C', 'GS', 'JPM', 'MS', 'WFC'],'google', start, end)
```

Section 14: Introduction to Machine Learning

Supervised Learning Overview

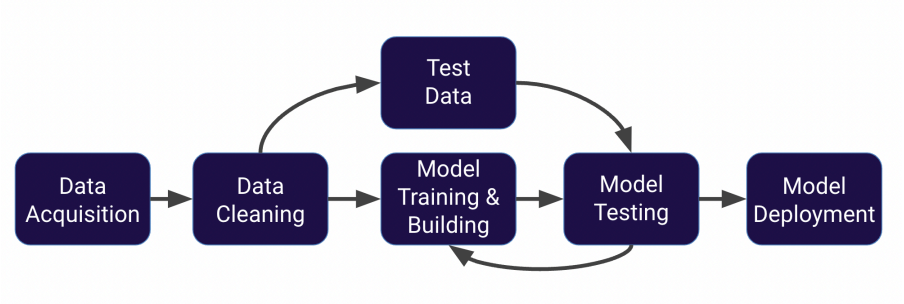
04-Machine-Learning-Overview

Supervised learning algorithms are trained using labeled examples, such as an input where the desired output is known. For example, a segment of text could have a category label, such as: Spam vs. Legitimate Email; Positive vs. Negative Movie Review

The network receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors.
It then modifies the model accordingly.

Supervised learning is commonly used in applications where historical data predicts likely future events.

Machine Learning Process:



What we just showed is a simplified approach to supervised learning, it contains an issue! Is it fair to use our single split of the data to evaluate our models performance? After all, we were given the chance to update the model parameters again and again.

To fix this issue, data is often split into 3 sets

1. **Training Data:** Used to train model parameters
2. **Validation Data:** Used to determine what model hyperparameters to adjust
3. **Test Data:** Used to get some final performance metric

This means after we see the results on the final test set we don't get to go back and adjust any model parameters! This final measure is what we label the true performance of the model to be.

Evaluating Performance - Classification Error Metrics

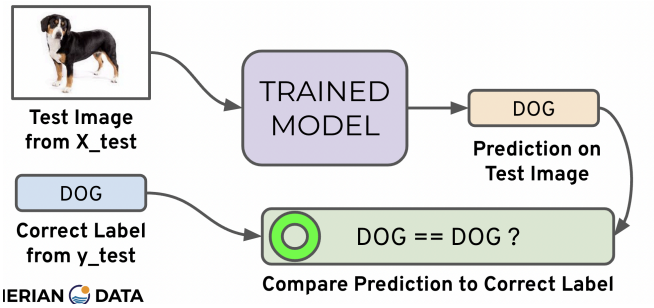
We will use **performance metrics** to evaluate how our model did. The key classification metrics we need to understand are:

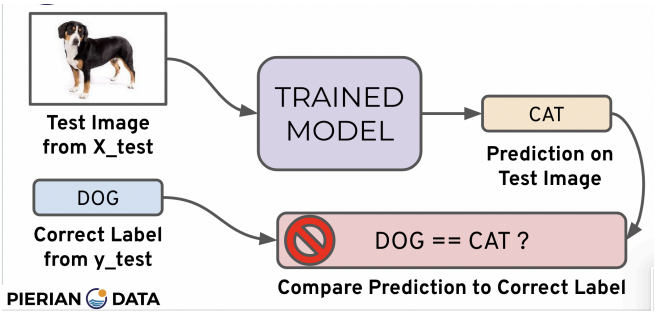
- **Accuracy**
- **Recall**
- **Precision**
- **F1-Score**

Typically in any classification task your model can only achieve two results: Either your model was **correct** in its prediction or your model was **incorrect** in its prediction.

Fortunately incorrect vs correct expands to situations where you have multiple classes.
For the purposes of explaining the metrics, let's imagine a binary classification situation, where we only have two available classes.

In our example, we will attempt to predict if an image is a dog or a cat. Since this is supervised learning, we will first fit/train a model on training data, then test the model on testing data. Once we have the model's predictions from the X_test data, we compare it to the true y values (the correct labels). We repeat this process for all the images in our X test data. At the end we will have a count of correct matches and a count of incorrect matches.





The key realization we need to make, is that **in the real world, not all incorrect or correct matches hold equal value! Also in the real world, a single metric won't tell the complete story!**

To understand all of this, let's bring back the 4 metrics we mentioned and see how they are calculated. We could organize our predicted values compared to the real values in a confusion matrix.

Accuracy

Accuracy in classification problems **is the number of correct predictions made by the model divided by the total number of predictions.** For example, if the X_test set was 100 images and our model correctly predicted 80 images, then we have 80/100 = 0.8 or 80% accuracy.

- Accuracy is **useful when target classes are well balanced.** In our example, we would have roughly the same amount of cat images as we have dog images.
- Accuracy is not a good choice with unbalanced classes! Imagine we had 99 images of dogs and 1 image of a cat. If our model was simply a line that always predicted dog we would get 99% accuracy! In this situation we'll want to understand recall and precision

Recall

Ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives.

Precision

Ability of a classification model to identify only the relevant data points. Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.

Recall and Precision

Often you have a **trade-off between Recall and Precision.** While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant.

F1-Score

In cases where we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F1 score. **The F1 score is the harmonic mean of precision and recall taking both metrics into account** in the following equation:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

We use the harmonic mean instead of a simple average because it punishes extreme values. A classifier with a precision of 1.0 and a recall of 0.0 has a simple average of 0.5 but an F1 score of 0.

Confusion Matrix

We can also view all correctly classified versus incorrectly classified images in the form of a confusion matrix.

		predicted condition	
	total population	prediction positive	prediction negative
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**

		predicted condition		
	total population	prediction positive	prediction negative	Prevalence $= \frac{\Sigma \text{ condition positive}}{\Sigma \text{ total population}}$
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)	True Positive Rate (TPR), Sensitivity, Recall, Probability of Detection $= \frac{\Sigma \text{ TP}}{\Sigma \text{ condition positive}}$
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**	False Positive Rate (FPR), Fall-out, Probability of False Alarm $= \frac{\Sigma \text{ FP}}{\Sigma \text{ condition negative}}$
	Accuracy $= \frac{\Sigma \text{ TP} + \Sigma \text{ TN}}{\Sigma \text{ total population}}$	Positive Predictive Value (PPV), Precision $= \frac{\Sigma \text{ TP}}{\Sigma \text{ prediction positive}}$	False Omission Rate (FOR) $= \frac{\Sigma \text{ FN}}{\Sigma \text{ prediction negative}}$	Positive Likelihood Ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$
		False Discovery Rate (FDR) $= \frac{\Sigma \text{ FP}}{\Sigma \text{ prediction positive}}$	Negative Predictive Value (NPV) $= \frac{\Sigma \text{ TN}}{\Sigma \text{ prediction negative}}$	Negative Likelihood Ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$

The main point to remember with the confusion matrix and the various calculated metrics is that they are all fundamentally ways of comparing the predicted values versus the true values. **What constitutes "good" metrics, will really depend on the specific situation!**

What is a good enough accuracy? This all depends on the context of the situation! Did you create a model to predict presence of a disease? Is the disease presence well balanced in the general population? (Probably not!) Often models are used as quick diagnostic tests to have before having a more invasive test (e.g. getting urine test before getting a biopsy) We also need to consider what is at stake!

Often we have a precision/recall trade off, We need to decide if the model will should focus on fixing False Positives vs. False Negatives. In disease diagnosis, it is probably better to go in the direction of False positives, so we make sure we correctly classify as many cases of disease as possible! All of this is to say, machine learning is not performed in a "vacuum", but instead a collaborative process where we should consult with experts in the domain.

Evaluating Performance - Regression Error Metrics

Regression is a task when a model attempts to predict continuous values (unlike categorical values, which is classification). For example, attempting to predict the price of a house given its features is a regression task. Attempting to predict the country a house is in given its features would be a classification task.

Let's discuss some of the most common evaluation metrics for regression:

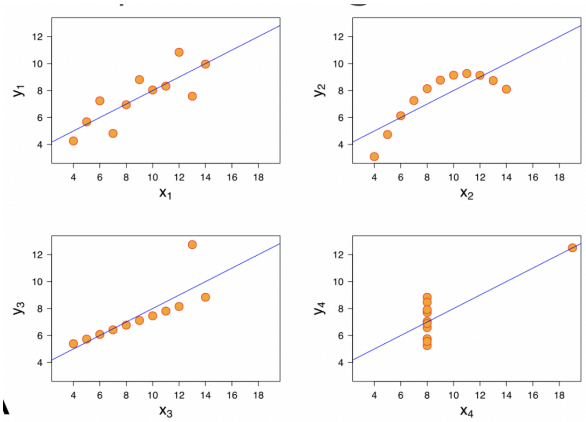
- **Mean Absolute Error**
- **Mean Squared Error**
- **Root Mean Square Error**

Mean Absolute Error (MAE)

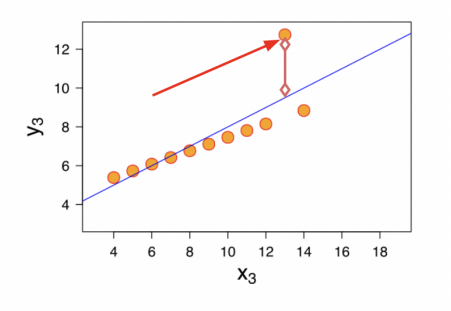
This is the mean of the absolute value of errors.

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

MAE won't punish large errors however.



We want our error metrics to account for these! (Un valor muy atípico)



Mean Squared Error (MSE)

This is the mean of the squared errors. Larger errors are noted more than with MAE, making MSE more popular.

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

Root Mean Square Error (RMSE)

This is the root of the mean of the squared errors. Most popular (has same units as y)

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

Most common question from students: "Is this value of RMSE good?" Context is everything! A RMSE of $10 is fantastic for predicting the price of a house, but horrible for predicting the price of a candy bar!

Compare your error metric to the average value of the label in your data set to try to get an intuition of its overall performance. Domain knowledge also plays an important role here!

Context of importance is also necessary to consider. We may create a model to predict how much medication to give, in which case small fluctuations in RMSE may actually be very significant.

Machine Learning with Python

We will be using the **Scikit Learn package**. It´s the most popular machine learning package for Python and has a lot of algorithms built-in. You will need to install it using: conda install scikit-learn or pip install scikit-learn.

Every algorithm is exposed in scikit-learn via an "Estimator". First you will import the model, the general form is: **from sklearn.family import Model, for example from sklearn.linear_model import Linear Regression.**

Estimator parameters: All the parameters of an estimator can be set when it is instantiated, and have suitable default values. You can use Shift+Tab to check the possible parameters. Por ejemplo, puede instanciar un modelo de regresión lineal especificando un parámetro que se normalizará igual a verdadero y luego imprimir el modelo que acaba de instanciar:
model = LinearRegression(normalize=True)
print(model)

Puede seguir adelante y verificar los parámetros que fueron predeterminados para el modelo:
LinearRegression(copy_X=True, fit_intercept=True, normalize=True)

Once you have your model created with your parameters, it is time to fit your model on some data! But remember we should split this data into a training set and a test set. For example:
import numpy as np
from sklearn.cross_validation import train_test_split
X, y = np.arrange(10).reshape((5,2)),range(5)
X

array ([[0, 1],
[2, 3],
[4, 5],
[6, 7],
[8, 9]])

list(y)

[0, 1, 2, 3, 4]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train

array ([[4, 5],
[0, 1],
[6, 7]])

y_train

[2, 0, 3]

X_test

array ([[2, 3],
[8, 9]])

y_test

[1, 4]

Now that we have split the data, we can train/fit our model on the training data. This is done through the model.fit() method:
model.fit(X_train, y_train) #For supervised learning applications this accepts 2 arguments the data X and the labels y; for unsupervised learning applications this accepts only a single argument, the data X

Now the model has been fit and trained on the training data the model is ready to predict labels or values on the test set. We get predicted values using the predict method:

predictions = **model.predict(X_test)** #Supervised estimators
model.predict(): Predict labels in clustering algorithms #Unsupervised estimators

model.predict_proba(): For classification problems some estimators also provide this method which returns the probability that a new observation has each categorical label. In this case the label with the highest probability is returned by model.predict() #Supervised estimators

model.score(): For classification or regression problems most estimators implement a score method. Scores are between 0 and 1, with a larger score indicating a better fit #Supervised estimators

model.transform(): Given an unsupervised model, transform new data into the new basis. This also accepts one argument X_new, and returns the new representation of the data based on the unsupervised model #Unsupervised estimators

model.fit_transform(): Some estimators implement this method, which more efficiently performs a fit and transform on the same input data. #Unsupervised estimators

We can then evaluate our model by comparing our predictions to the correct values. The **evaluation method depends on what sort of machine learning algorithm we are using** (e.g. Regression, Classification, Clustering, etc.)

Section 15: Linear Regression

Linear Regression Theory

Chapters 2,3 of Introduction to Statistical Learning by Gareth Hames, et al.

When we calculate our regression line we try to draw a line that's as close to every dot as possible. For **classic linear regression or "Least Squares Method"** you only measure the closeness in the up and down direction. Our goal with linear regression is to **minimize the vertical distance between all the data points and our line**, so in determining the best line we are attempting to minimize the distance between all the points and their distance to our line. **There are lots of different ways to minimize this** (sum of squared errors, sum of absolute errores, etc) but all there methods have a general goal of minimizing this distance.

Least Squares Method: The line is fitted by minimizing the sum of squares of the residuals (the residuals for an observation is the difference between the y-value and the fitted line)

Linear Regression With Python

```
## Check the data and EDA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
#Check out the data
USAhousing = pd.read_csv('USA_Housing.csv')
USAhousing.head()
USAhousing.info()
USAhousing.describe()
USAhousing.columns

#EDA
sns.pairplot(USAhousing)
sns.distplot(USAhousing['Price'])
# Seleccionar solo las columnas numéricas
numerical_housing = USAhousing.select_dtypes(include='number')
# Mostrar la matriz de correlación de las columnas numéricas de 'tips'
correlation_matrix = numerical_housing.corr()
correlation_matrix
sns.heatmap(correlation_matrix,annot=True)
#sns.heatmap(USAhousing.corr())
```

Training a Linear Regression Model

We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can't use.

```
USAhousing.columns
```

Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'], dtype='object')

```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price'] #lo que queremos predecir
```

Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101) #El 0.4 es el porcentaje de tu data que queres alocar al test, en este caso el 40%
```

Creating and Training the Model

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Model Evaluation

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

print(lm.intercept_) # print the intercept

-2640159.7968526953

coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])

coeff_df

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Area Income** is associated with an **increase of \$21.52**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area House Age** is associated with an **increase of \$164883.28**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Rooms** is associated with an **increase of \$122368.67**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Bedrooms** is associated with an **increase of \$2233.80**.
- Holding all other features fixed, a 1 unit increase in **Area Population** is associated with an **increase of \$15.15**.

Predictions from our Model

Let's grab predictions off our test set and see how well it did!

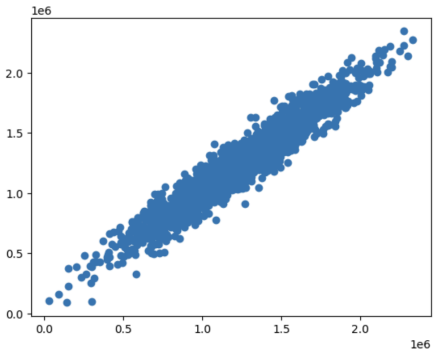
predictions = lm.predict(X_test)

predictions #Estos son los precios que predecimos de la casa

array([1260960.70567627, 827588.75560329, 1742421.24254344, ..., 372191.40626917, 1365217.15140898, 1914519.5417888])

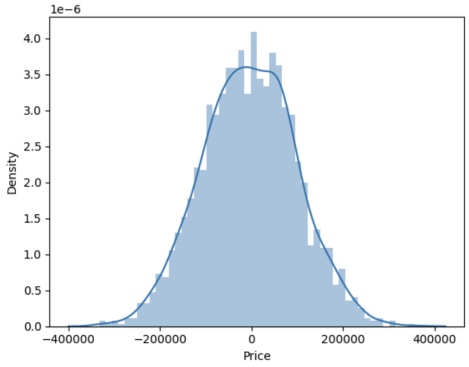
#y_test contiene los precios reales

plt.scatter(y_test,predictions)



#Residual Histogram

sns.distplot((y_test-predictions),bins=50);



Regression Evaluation Metrics

Here are 3 common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

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

Mean Squared Error (MSE) is the mean of the squared errors:

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

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

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

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

MAE: 82288.22251914942
MSE: 10460958907.208977
RMSE: 102278.82922290897

Section 16: Bias-Variance Trade-Off

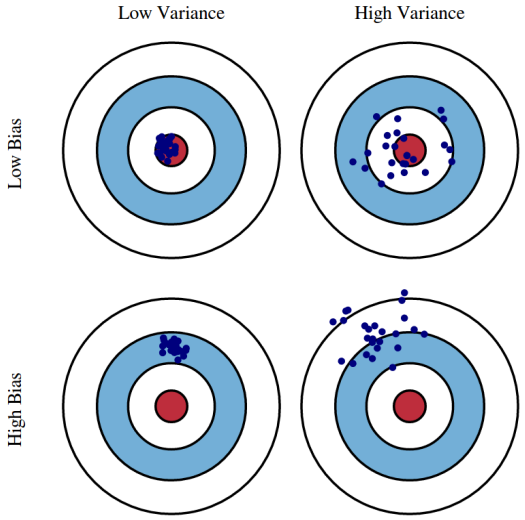
Bias-Variance Trade-Off

Chapter 2 of Introduction to Statistical Learning by Gareth Hames, et al.

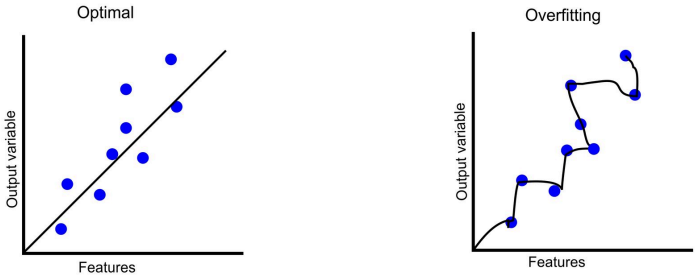
The bias-variance trade-off is the point where we are adding just noise by adding model complexity (flexibility). The training error goes down as it has to, but the test error is starting to go up. The model after the bias trade-off begins to overfit.

Imagine that the center of the target is a model that perfectly predicts the correct values, as we move away from the bulls-eye our predictions get worse. Imagine we can repeat our entire model building process to get a number of separate hits on the target, each hit represents an individual realization of our model, given the chance variability in the training data we gather. Sometimes we will get a good distribution of training data so we predict very well and we are close to the bulls-eye while sometimes our training data might be full of outliers or non-standard values resulting in poorer predictions. There different realizations result in a scatter of hits on the target.

- **Bias / Desvío** = “estar cerca del objetivo”
- **Variance / Varianza** = “estar cerca entre sí las predicciones”



Un error común de principantes es tener un modelo simple con un cierto error y hacer que el modelo sea cada vez más complejo/flexible hasta que alcance a todos los puntos del training set. Sin embargo, al hacer esto el modelo no va a poder predecir nuevos puntos de prueba (new test points), por eso hacemos el test de entrenamiento dividido.



Hacer esto puede generar que el modelo haga **overfit** con la data de entrenamiento y luego se generen grandes errores con datos nuevos, como con los datos de prueba.

Veamos un ejemplo de como podemos ver overfitting (sobreajuste) desde otro punto de vista usando datos de test (prueba). Vamos a usar una curva negra con algunos puntos de ruido/noise para representar la forma real de la data:

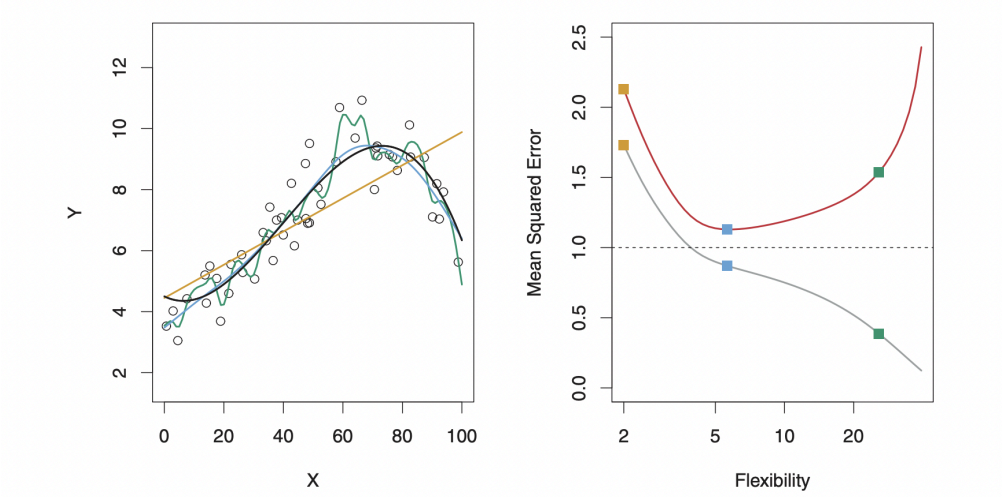


FIGURE 2.9. Left: *Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing spline fits (blue and green curves).* Right: *Training MSE (grey curve), test MSE (red curve), and minimum possible test MSE over all methods (dashed line). Squares represent the training and test MSEs for the three fits shown in the left-hand panel.*

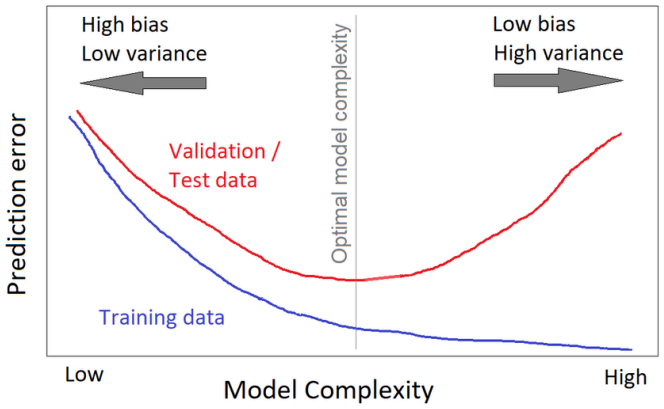
En la primer imagen diferentes ajustes: lineal (amarillo), cuadrático (azul) y spline (verde), cada uno de ellos más complejo, mientras que la curva negra es la realidad. Para evaluar los modelos y comparar las complejidades entre sí hay que trazar la flexibility (segunda imagen), por ejemplo el nivel polinomial de un ajuste de regresión frente a la métrica de error (como el MSE), además hemos trazado los datos de entrenamiento en comparación con los datos de prueba.

- El modelo simple lineal (amarillo) tiene un alto error tanto en los datos de prueba como en los de entrenamiento
- A medida que empezamos a complicarnos más con el modelo cuadrático reducimos el error de los datos de entrenamiento y el error de los datos de prueba
- Luego, a medida que se sigue complejizando el modelo vemos que el error en los datos de entrenamiento dismiinuye pero comienza a aumentar el error en los datos de prueba/test.

Lo que se debe buscar es encontrar el equilibrio entre el sesgo y la varianza del modelo hasta el punto en que la data de prueba y la data de entrenamiento hayan alcanzado algún tipo de mínimo y agrupación.

Este es el clásico gráfico para mostrar en el eje X la complejidad del modelo y en el eje Y algún tipo de error de predicción, a medida que se avanza hacia la izquierda se tiene un

mayor sesgo pero con una varianza menor (**underfitting**) y hacia el otro lado con un modelo de mayor complejidad sucede al revés (**overfitting**). Lo que se debe buscar es elegir un punto en que estemos cómodos con el trade-off.



Section 17: Logistic Regression

Logistic Regression Theory

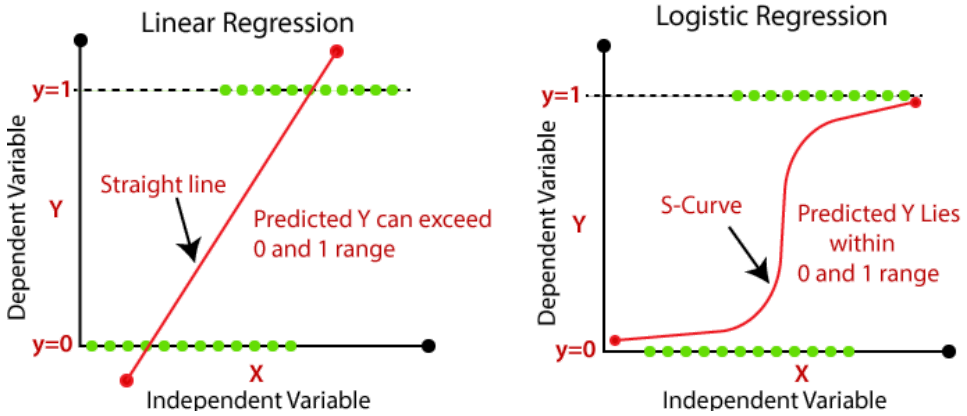
Sections 4-4.3 of Introduction to Statistical Learning by Gareth James, et al.

We want to learn about **Logistic Regression** as a method for **Classification**. Some examples of classification problems (binary classification, 2 classes):

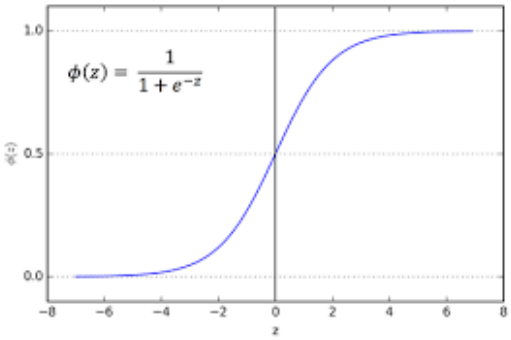
- Spam vs “ham” emails
- Loan default (yes/no)
- Disease diagnosis

So far we’ve only seen regression problems where we try to predict a continuous value. Although the name may be confusing at first, **logistic regression** allows us to solve classification problems where we are trying to predict **discrete categories**. The convention for **binary classification is to have 2 clases: 0 and 1**.

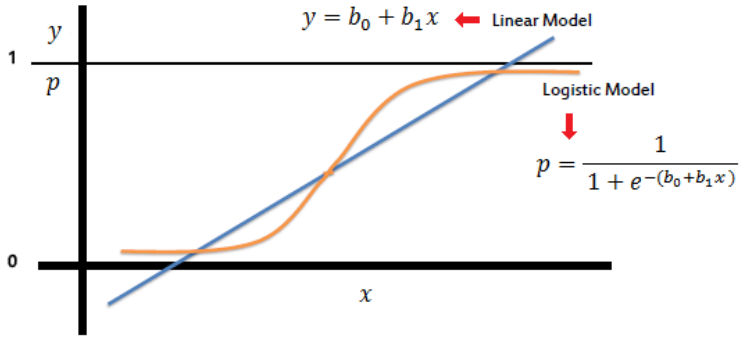
We can’t use a normal linear regression model on binary groups, it won’t lead to good fit. Instead, we can **transform our linear regression to a logistic regression curve**, que en este caso (a diferencia de linear) solo puede ir **entre 0 y 1.**



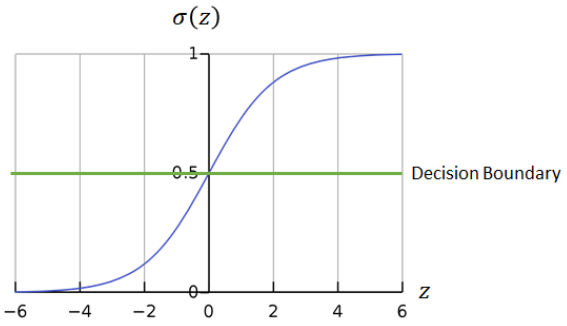
The Sigmoid (aka Logistic) Function takes in any value and outputs it to be **between 0 and 1.**



This means **we can take our Linear Regression Solution and place it into the Sigmoid Function.**



We can set a cutoff point at 0.5, anything below it results in class 0 and anything above is class 1. So we are going to transform that 0.5 probability as a cutoff point.



After you train a logistic regression model on some training data you will evaluate your model's performance on some test data. You can use a confusion matrix to evaluate classification models, for example n=165:

	Predicted NO	Predicted YES	
Actual NO	50	10	60
Actual YES	5	100	105
	55	110	

- True Negative = 50
- True Positive = 100
- False Negative (error tipo 2) = 5
- False Positive (error tipo 1)= 10

- Accuracy = (TP+TN) / Total = (100+50) / 165 = 0.909
- Error Rate = (FP+FN) / Total = (10+5) / 165 = 0.091

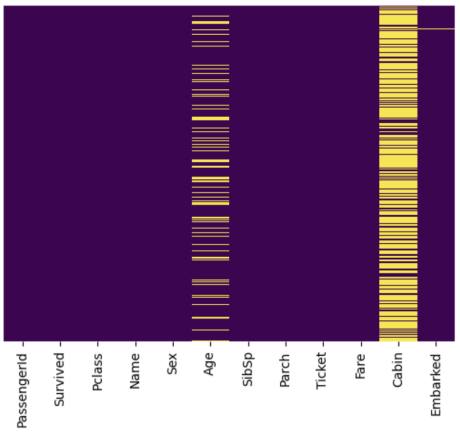
Logistic Regression with Python

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

train = pd.read_csv('titanic_train.csv')
train.head()
```

Missing Data

We can use seaborn to create a simple heatmap to see where we are missing data!

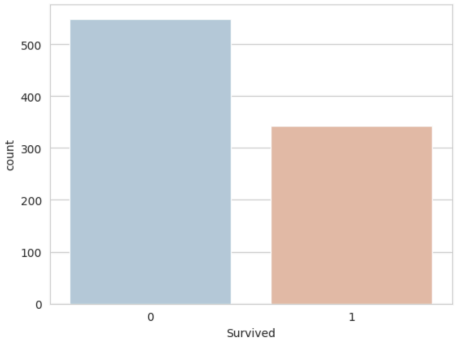
train.isnull()
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')



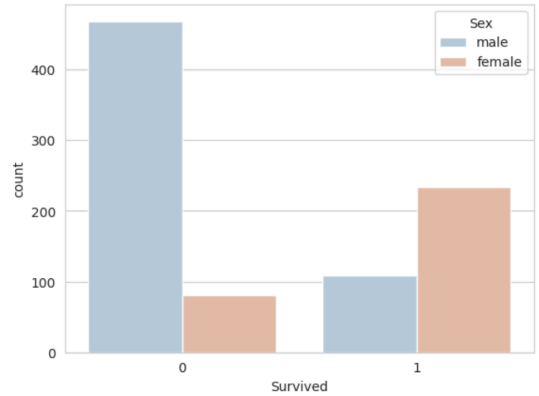
Roughly 20% of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

Data Visualizations

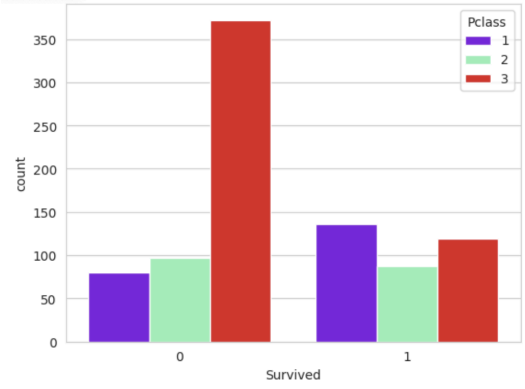
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r') #hay mas q no sobrevivieron



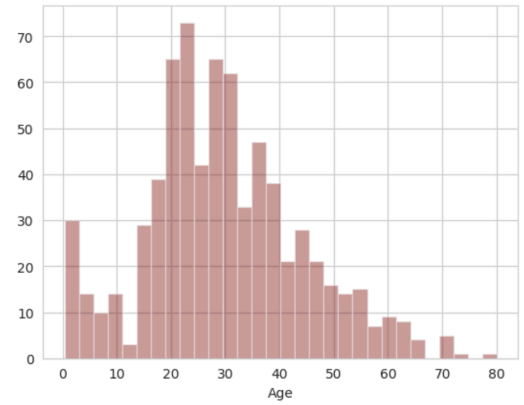
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r') #hay mas q no sobrevivieron hombres, hay mas que sobrevivieron mujeres



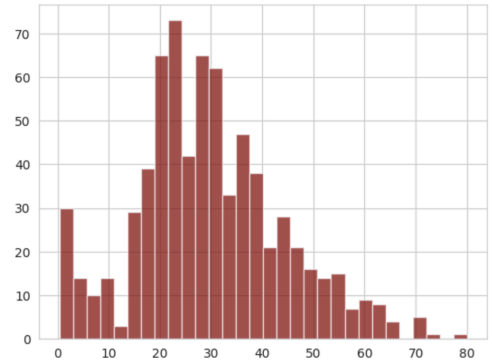
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow') #los que no sobrevivieron eran ppalmente de la clase 3, la mas barata



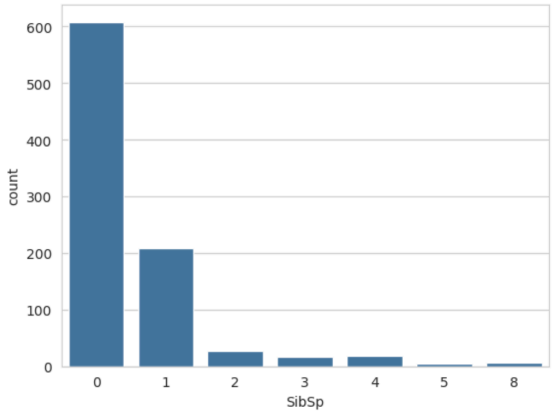
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)



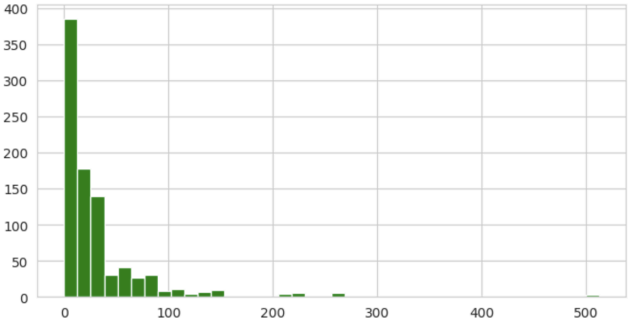
train['Age'].hist(bins=30,color='darkred',alpha=0.7) #otra forma de hacer el mismo grafico



sns.countplot(x='SibSp',data=train) #SibSp es el numero de hermanos o conyugues a bordo

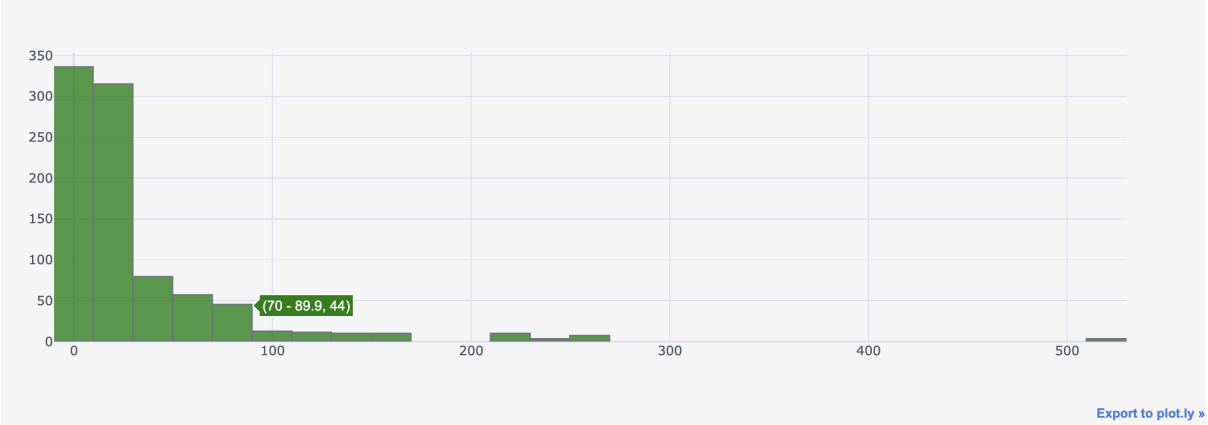


train['Fare'].hist(color='green',bins=40,figsize=(8,4)) #cuanto pagaron



Plots with Cufflinks

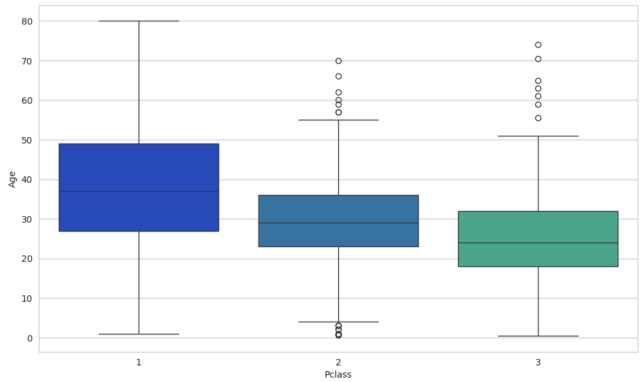
```
import cufflinks as cf
cf.go_offline()

train['Fare'].iplot(kind='hist',bins=30,color='green')
```



Data Cleaning

We want to **fill in missing age data instead of just dropping the missing age data rows**. One way to do this is by filling in the mean age of all the passengers (imputation).
However we can be smarter about this and check the average age by passenger class. For example:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```



We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

```
def impute_age(cols):
    Age = cols[0] #le pasamos de argumento a la funcion estas 2 columnas
    Pclass = cols[1] #le pasamos de argumento a la funcion estas 2 columnas
    if pd.isnull(Age):
        if Pclass == 1:
            return 37
        elif Pclass == 2:
```

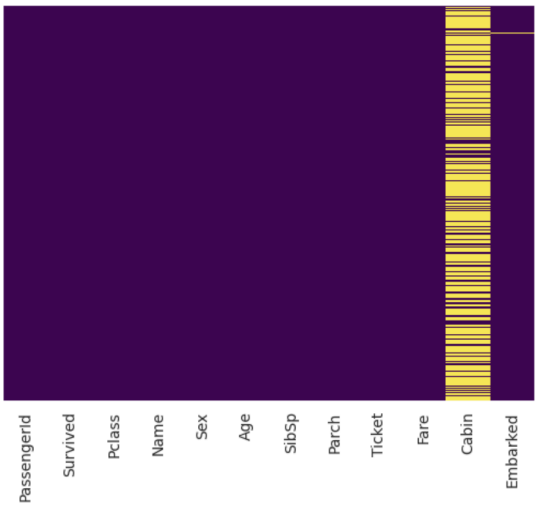
```
        return 29
    else:
        return 24
else:
    return Age
```

Now apply that function!

train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)

Now let's check that heat map again!

sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')

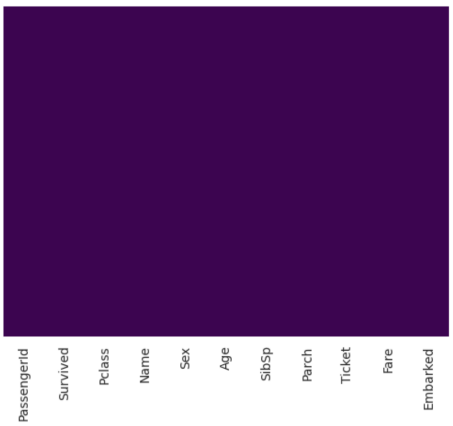


Great! Let's go ahead and drop the Cabin column and the row in Embarked that is NaN.

train.drop('Cabin',axis=1,inplace=True) #Eliminamos la columna de Cabin

train.dropna(inplace=True) #Eliminamos si quedó algún null por ahí

sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')



Converting Categorical Features

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs.

train.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Name         889 non-null    object
 4   Sex          889 non-null    object
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64
 7   Parch        889 non-null    int64
 8   Ticket       889 non-null    object
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

pd.get_dummies(train['Sex'])

	female	male
0	False	True
1	True	False
2	True	False
3	True	False
4	False	True
...	...	...
886	False	True
887	True	False
888	True	False
889	False	True
890	False	True

889 rows × 2 columns

sex = pd.get_dummies(train['Sex'],drop_first=True) #drop_first se queda solo con male en este caso, porque si no es hombre es mujer entonces con uno alcanza
embark = pd.get_dummies(train['Embarked'],drop_first=True) #en este caso se queda sin C que es la primer columna, y se queda con Q y S

train = pd.concat([train,sex,embark],axis=1) #al dataframe de train le agrega lo que definimos como sex, que es la columna de male, y lo que definimos como embark

train.head()

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	male	Q	S
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	True	False	True
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	False	False	False
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	False	False	True
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	False	False	True
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	True	False	True

train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True) #eliminamos las columnas que no necesitamos como Name y Ticket, y las que llevamos a otras como sex que la llevamos a male y embarked que la llevamos a Q y S

train.head()

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	True	False	True
1	2	1	1	38.0	1	0	71.2833	False	False	False
2	3	1	3	26.0	0	0	7.9250	False	False	True
3	4	1	1	35.0	1	0	53.1000	False	False	True
4	5	0	3	35.0	0	0	8.0500	True	False	True

Logistic Regression Model: Train Test Split

Let's start by splitting our data into a training set and test set (there is another test.csv file that you can play around with in case you want to use all this data for training).

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Logistic Regression Model: Training and Predicting

```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)

predictions = logmodel.predict(X_test)
```

Logistic Regression Model: Evaluation

```
from sklearn.metrics import classification_report
```

print(classification_report(y_test,predictions))

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       163
           1       0.81      0.62      0.71       104

    accuracy                           0.80       267
   macro avg       0.80      0.77      0.78       267
weighted avg       0.80      0.80      0.79       267
```

Not so bad! You might want to explore other feature engineering and the other titanic_text.csv file, some suggestions for feature engineering:

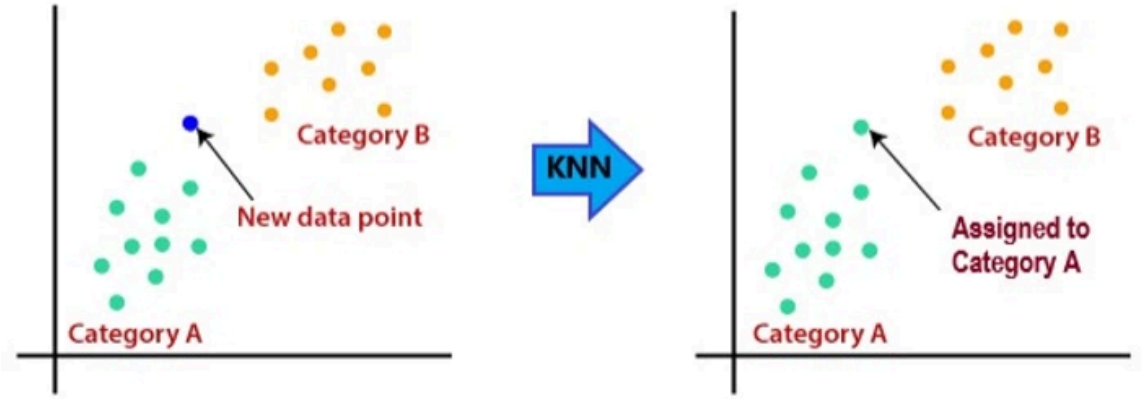
* Try grabbing the Title (Dr.,Mr.,Mrs,etc..) from the name as a feature
* Maybe the Cabin letter could be a feature
* Is there any info you can get from the ticket?

Section 18: K Nearest Neighbors

KNN Theory

Chapter 4 of Introduction to Statistical Learning by Gareth James, et al.

K Nearest Neighbors is a classification algorithm that operates on a very simple principle. Imagine we had some imaginary data on dogs and horses, with heights and weights



Training Algorithm

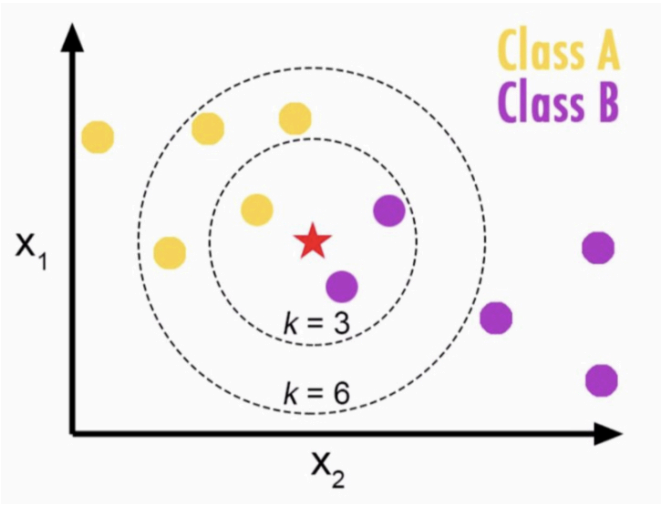
1. Store all the data

Prediction Algorithm

1. Calculate the distance from x to all points in your data
2. Sort the points in your data by increasing distance from x
3. Predict the majority label of the “k” closest points

Choosing a “k” will affect what class a new points is assigned to:

- Si elijo k= 3 miro a los 3 vecinos mas cercanos al punto q quiero predecir (la estrella)
- Si elijo k=6 miro a los 6 vecinos mas cercanos



Pros

- Very simple
- Training is trivial
- Works with any number of classes
- Easy to add more data
- Few parameters: k and distance metric

Cons

- High prediction cost (worse for large datasets)
- Not good with high dimensional data
- Categorical features don't work well

KNN with Python

You've been given a classified data set from a company! They've hidden the feature column names but have given you the data and the target classes. We'll try to use KNN to create a model that directly predicts a class for a new data point based off of the features.

Get Data

df = pd.read_csv("Classified Data",index_col=0) #Set index_col=0 to use the first column as the index.

Standarize Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
                                                    test_size=0.30)
```

Using KNN

Remember that we are trying to come up with a model to predict whether someone will TARGET CLASS or not. We'll start with k=1.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
```

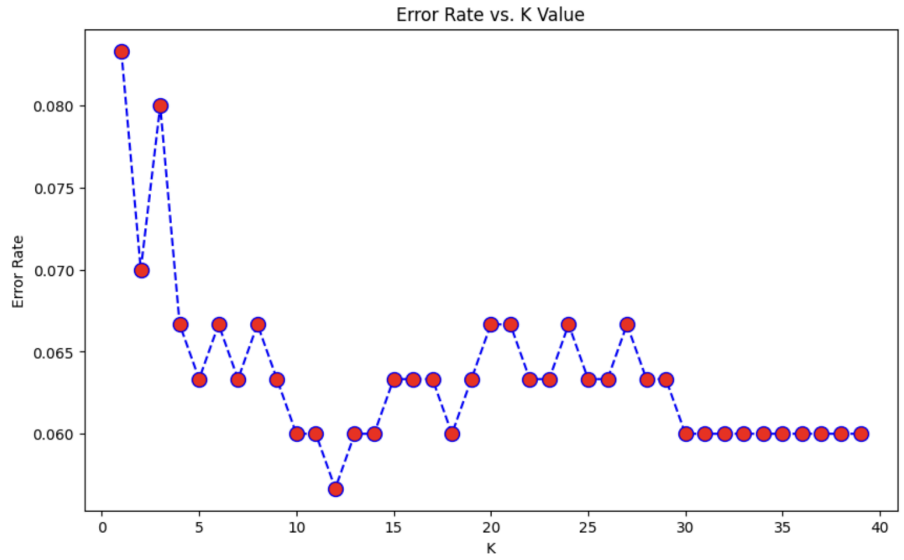
Predictions and Evaluations

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       155
           1       0.91      0.92      0.91       145

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

Choosing a K value

```
error_rate = []

# Will take some time
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```



Here we can see that that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[141  14]
 [ 11 134]]


              precision    recall  f1-score   support

           0       0.93      0.91      0.92       155
           1       0.91      0.92      0.91       145

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

```
# NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
print('WITH K=23')
```

```
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=23


[[143  12]
 [  7 138]]


              precision    recall  f1-score   support

           0       0.95      0.92      0.94       155
           1       0.92      0.95      0.94       145

    accuracy                           0.94       300
   macro avg       0.94      0.94      0.94       300
weighted avg       0.94      0.94      0.94       300
```

Section 19: Decision Trees and Random Forests

Introduction to Tree Methods

Chapter 8 of Introduction to Statistical Learning by Gareth James, et al.
https://medium.com/@imparth/decision-tree-4d0bc22620ab
https://towardsdatascience.com/enchanted-random-forest-b08d418cb411#.hh7n1co54

Decision trees are powerful and popular tools for classification and prediction. **Decision trees** are attractive due to the fact that, in contrast to other machine learning techniques such as neural networks, they **represent rules**.

What is a decision tree? **Decision tree is a classifier in the form of a tree structure.** In this tree we have:

- Nodes: Split for the value of a certain attribute
- Edges: Outcome of a split to next node
- Root: The node that performs the first split
- Leaves: Terminal nodes that predict the outcome

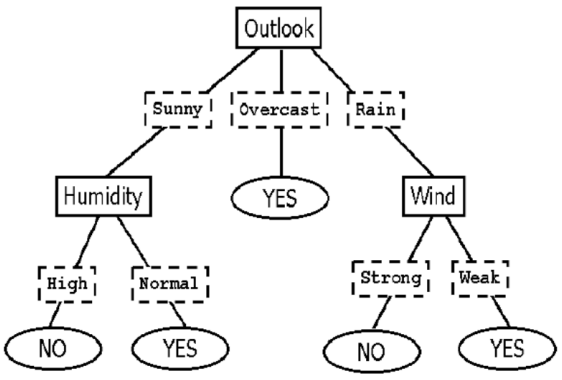


Figure 1: Decision tree: for conditions to play tennis.

La entropía y la ganancia de información son los métodos matemáticos para elegir la mejor división

Entropy

In order to define information gain precisely, we need to define a measure commonly used in information theory, called **entropy**, that **characterizes the(im)purity of an arbitrary collection of examples.** Given a set S, containing only positive and negative examples of some target concept (for a 2 class problem), the entropy of set S relative to this simple, **binary classification** is defined as:

$$Entropy(S) = -p_p log_2 p_p - p_n log_2 p_n$$

where pp is the proportion of positive examples in S and pn is the proportion of negative examples in S. In all calculations involving entropy we define 0log0 to be 0. To illustrate, suppose S is a collection of 25 examples, including 15 positive and 10 negative examples [15+, 10-]. Then the entropy of S relative to thisclassification is

$$Entropy(S) = -(\frac{15}{25})log_2(\frac{15}{25}) - (\frac{10}{25})log_2(\frac{10}{25}) = 0.970$$

Notice that the **entropy is 0 if all members of S belong to the same class**. For example, if all members are positive (pp= 1), then pn is 0, and Entropy(S) =-1*log2(1) - 0*log20=-1*0-0*log20 = 0. **Note the entropy is 1 (at its maxi-mum!) when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and nega-tive examples, the entropy is between 0 and 1.**

Thus far we have discussed entropy in the special case where the target classification is binary. **If the target attribute takes on c different values, then the entropy of S relative to this c-wise classification is defined as**

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

where pi is the proportion of S belonging to class i. Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits. Note also that if the target attribute can take on c possible values, the maximum possible entropy is log2c.

Information Gain

Given entropy as a measure of the impurity in a collection of training examples, we can now define a **measure of the effectiveness of an attribute in classifying the training data**. The measure we will use, called **information gain**, **is simply the expected reduction in entropy caused by partitioning the examples according to this attribute**. More precisely, the information gain Gain(S, A) of an attribute A relative to a collection of examples S is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in V(A)} Entropy \frac{\mid S_v \mid}{\mid S \mid}(S_v)$$

where Values(A) is the set of all possible values for attribute A, and Sv is the subset of S for which attribute A has value v (i.e., Sv={sS|A(s)=v}). Note the first term in the equation for Gain is

just the entropy of the original collection S and the second term is the expected value of the entropy after S is partitioned using attribute A. The expected entropy described by this second term is simply the sum of the entropies of each subset Sv, weighted by the fraction of examples |Sv|/|S| that belong to Sv. **Gain(S, A) is therefore the expected reduction in entropy caused by knowing the value of attribute A**. Putanother way, **Gain(S, A) is the information provided about the target attribute value, given the value of some other attribute A**. The value of Gain(S, A) is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute A.

El proceso de seleccionar un nuevo atributo y dividir los ejemplos de entrenamiento ahora se repite para cada nodo descendiente no terminal, esta vez usando solo los ejemplos de entrenamiento asociados con ese nodo. Los atributos que se han incorporado más arriba en el árbol se excluyen, de modo que cualquier atributo determinado puede aparecer como máximo una vez en cualquier ruta del árbol. Este proceso continúa para cada nuevo nodo hoja hasta que se cumpla cualquiera de dos condiciones:

1. Cada atributo ya ha sido incluido a lo largo de este camino a través del árbol, o
2. Todos los ejemplos de entrenamiento asociados con este nodo hoja tienen el mismo valor de atributo objetivo (es decir, su entropía es cero).

Bagging / Random force

La principal debilidad de los árboles de decisión es que no tienden a tener la mejor precisión predictiva. Esto se debe en parte a la alta varianza, diferentes divisiones en los datos de entrenamiento pueden conducir a árboles muy diferentes

Bagging es un procedimiento de propósito general para reducir la varianza del método de machine learning que se discute en el Capítulo 8. Podemos construir la idea del Bagging usando random force. Random force es solo una ligera variación de estas bolsas de árboles que tiene un rendimiento aún mejor. Lo que hacemos con random forest es crear un conjunto de árboles de decisión utilizando muestras de arranque de las muestras de arranque del conjunto de entrenamiento. Sin embargo, estamos construyendo cada árbol cada vez que se considera una división. Se elige una muestra aleatoria de características M como un candidato de división del conjunto completo de características P. Se elige una nueva muestra aleatoria de características para cada arbol en cada decisión individual para clasificación. Por lo general, se elige una muestra aleatoria de características M para que sea la raíz cuadrada de P, donde P es el conjunto completo de características.

En resumen **para mejorar el rendimiento podemos utilizar muchos árboles con una muestra aleatoria de características elegidas como división:**

- **Se elige una nueva muestra aleatoria de características para cada árbol en cada división.**
- **Para la clasificación, normalmente se elige m como la raíz cuadrada de p.**

Cual es el punto de usar random force?

- Bueno, **supongamos que hay una característica muy fuerte en el conjunto de datos, una característica que es realmente poderosa para predecir** cierta clase. **Cuando se**

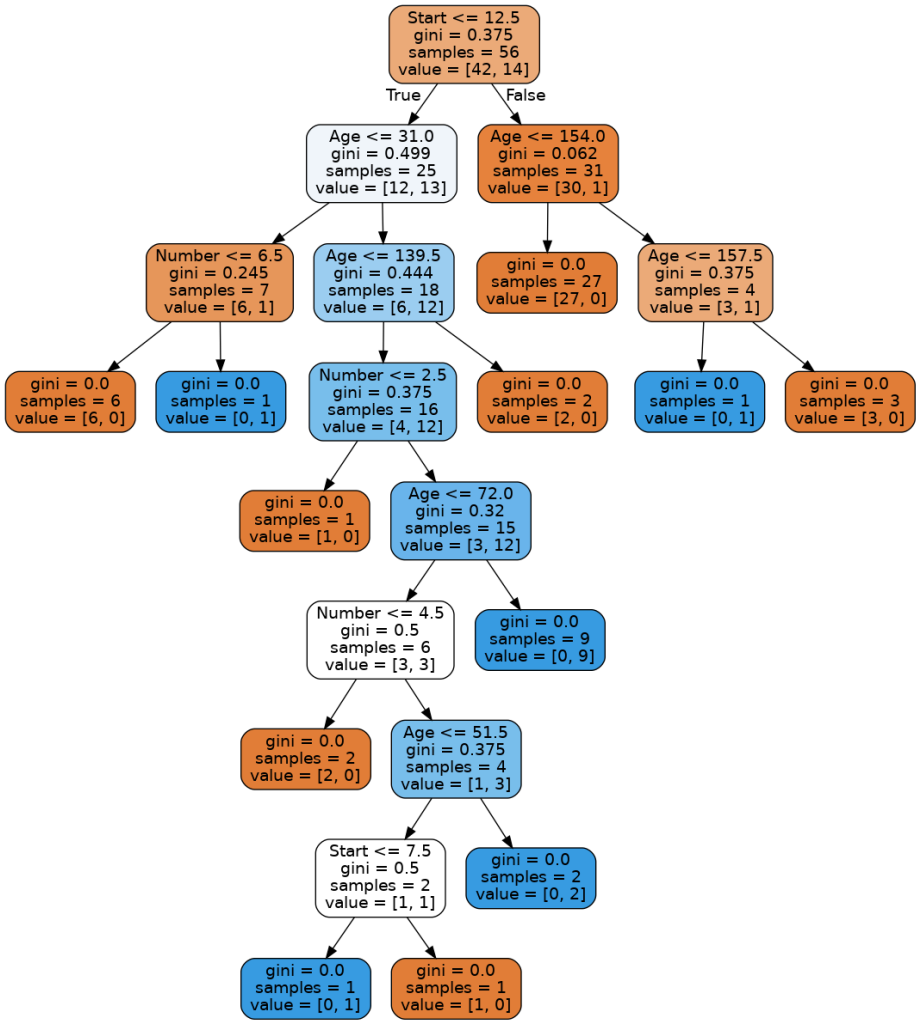
usan “bagged” trees (árboles en bolsas) la mayoría de los árboles usarán esa función como la división superior (the top split) que resultará en un conjunto de árboles muy similares que están altamente correlacionados, que es algo que desea evitar.

- **Promediar cantidades altamente correlacionadas no reduce significativamente la varianza.**
- **Al omitir aleatoriamente las características candidatas de cada división, Random Forests "descorrelaciona" los árboles, de modo que el proceso de promedio puede reducir la varianza del modelo resultante.**

Decision Trees and Random Forest with Python

```
## Import Libraries and Get Data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('kyphosis.csv')
df.head()

## EDA
sns.pairplot(df,hue='Kyphosis',palette='Set1')

## Train Test Split
from sklearn.model_selection import train_test_split
X = df.drop('Kyphosis',axis=1)
y = df['Kyphosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

## Decision Trees
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)

## Prediction and Evaluation
predictions = dtree.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))

## Tree Visualization
from IPython.display import Image
```

```
#from sklearn.externals.six import StringIO
import six
from sklearn.tree import export_graphviz
import pydot
features = list(df.columns[1:])
features

#sudo apt-get update
#sudo apt-get install graphviz
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,feature_names=features,filled=True,rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```



Random Forests

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

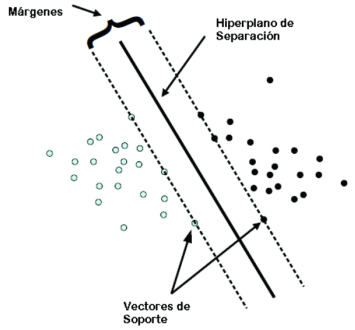
```
rfc_pred = rfc.predict(X_test)

print(confusion_matrix(y_test,rfc_pred))

print(classification_report(y_test,rfc_pred))
```

Section 20: Support Vector Machines

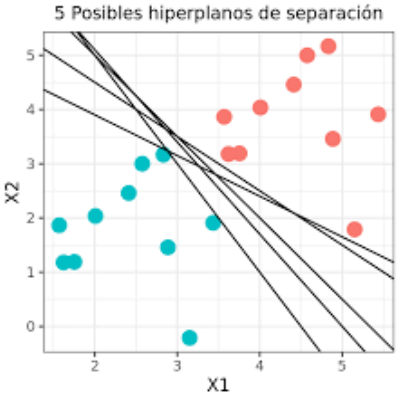
SVM Theory

Chapter 9 of Introduction to Statistical Learning by Gareth James, et al.
https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

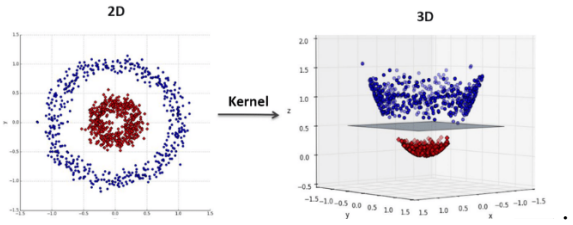
Support Vector Machines (SVM) son modelos de aprendizaje supervisados con algoritmos de aprendizaje asociados que analizan datos y reconocen patrones, y se utilizan para clasificación y análisis de regresión. Dado un conjunto de ejemplos de entrenamiento, cada uno marcado por pertenecer a una o dos categorías, un algoritmo de entrenamiento SVM construye un modelo que asigna nuevos ejemplos a una categoría u otra, convirtiéndolo en un clasificador lineal binario no probabilístico. Un modelo SVM es una presentación de los ejemplos como puntos en el espacio, mapeados de modo que los ejemplos de las categorías separadas estén divididos por una brecha clara que sea lo más amplia posible. Luego se asignan nuevos ejemplos a ese mismo espacio y se predice que pertenecen a una categoría según el lado de la brecha en el que se encuentran. Podemos dibujar un **hiperplano** de separación que separa las clases:



Sin embargo, **puede haber más de 1 posible hiperplano** que separe las clases. **¿Cómo elegimos entones la línea que separa mejor** las clases? Vamos a elegir un hiperplano que **maximice el margen entre las clases**



Podemos expandir esta idea a **datos no linealmente separables** mediante el uso del **“kernel trick”** . Si mirás el gráfico de la izquierda (en 2 dimensiones) tenés un eje x y un eje y, vas a ver que esa data no es separable linealmente porque hay círculo rojo y un círculo exterior de círculos azules y **no hay forma de que podamos dibujar una línea recta para separar estas clases**. Sin embargo a través del kernel trick terminamos haciendo esto en una dimensión superior, en este caso observamos una tercera dimensión z en la imagen de la derecha donde se pueden separar las clases en la tercera dimensión a través de otro hiperplano.



Support Vector Machines with Python

Import Libraries and Get Data

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

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
cancer.keys()
```

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

```
print(cancer['DESCR'])
```

```
Breast Cancer Wisconsin (Diagnostic) Database

Notes
-----
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 (perimeter^2 / 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.
...
   - 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)
     163-171.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

cancer['feature_names']

array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
'mean smoothness', 'mean compactness', 'mean concavity',
'mean concave points', 'mean symmetry', 'mean fractal dimension',
'radius error', 'texture error', 'perimeter error', 'area error',
'smoothness error', 'compactness error', 'concavity error',
'concave points error', 'symmetry error', 'fractal dimension error',
'worst radius', 'worst texture', 'worst perimeter', 'worst area',
'worst smoothness', 'worst compactness', 'worst concavity',
'worst concave points', 'worst symmetry', 'worst fractal dimension'],
dtype='<U23')

cancer['target']

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

cancer['target_names']

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

Set Up Data Frame

df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])

df_feat.info()

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	wo compactne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.66
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.18
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.42
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.86
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.20

5 rows × 30 columns

df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])

df_target.head()

	Cancer
0	0
1	0
2	0
3	0
4	0

EDA

We'll skip the Data Viz part for this lecture since there are so many features that are hard to interpret if you don't have domain knowledge of cancer or tumor cells. In your project you will have more to visualize for the data.

Train Test Split

```
from sklearn.model_selection import train_test_split
X=df_feat
y=cancer['target']
#X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target), test_size=0.30, random_state=101)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

Train the Support Vector Clasifier

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train,y_train)
```

Predictions adn Evaluations

```
predictions = model.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
```

```
[[ 56  10]
 [  3 102]]
```

```
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.95	0.85	0.90	66
1	0.91	0.97	0.94	105
accuracy			0.92	171
macro avg	0.93	0.91	0.92	171
weighted avg	0.93	0.92	0.92	171

Gridsearch

Finding the right parameters (like what C or gamma values to use) is a tricky task! But luckily, we can be a little lazy and just try a bunch of combinations and see what works best! This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV! The CV stands for cross-validation which is the

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}

One of the great things about GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC, and creates a new estimator, that behaves exactly the same - in this case, like a classifier. You should add refit=True and choose verbose to whatever number you want, higher the number, the more verbose (verbose just means the text output describing the process).

grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)

What fit does is a bit more involved then usual. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

 # May take awhile!
grid.fit(X_train,y_train)

You can inspect the best parameters found by GridSearchCV in the best_params_ attribute, and the best estimator in the best_estimator_ attribute:

grid.best_params_

```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

grid.best_estimator_

Then you can re-run predictions on this grid object just like you would with a normal model.

grid_predictions = grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))

```
[[ 59   7]
 [  4 101]]
```

print(classification_report(y_test,grid_predictions))

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91        66
           1       0.94      0.96      0.95       105

    accuracy                           0.94       171
   macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```

Section 21: K Means Clustering

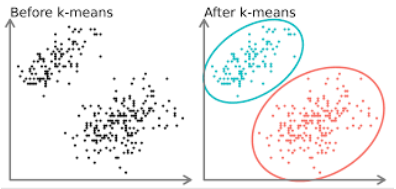
K Means Algorithm Theory

Chapter 10 of Introduction to Statistical Learning by Gareth James, et al.

K Means Clustering is an **unsupervised learning algorithm** that will attempt to group similar clusters together in your data. So what does a typical clustering problem look like?

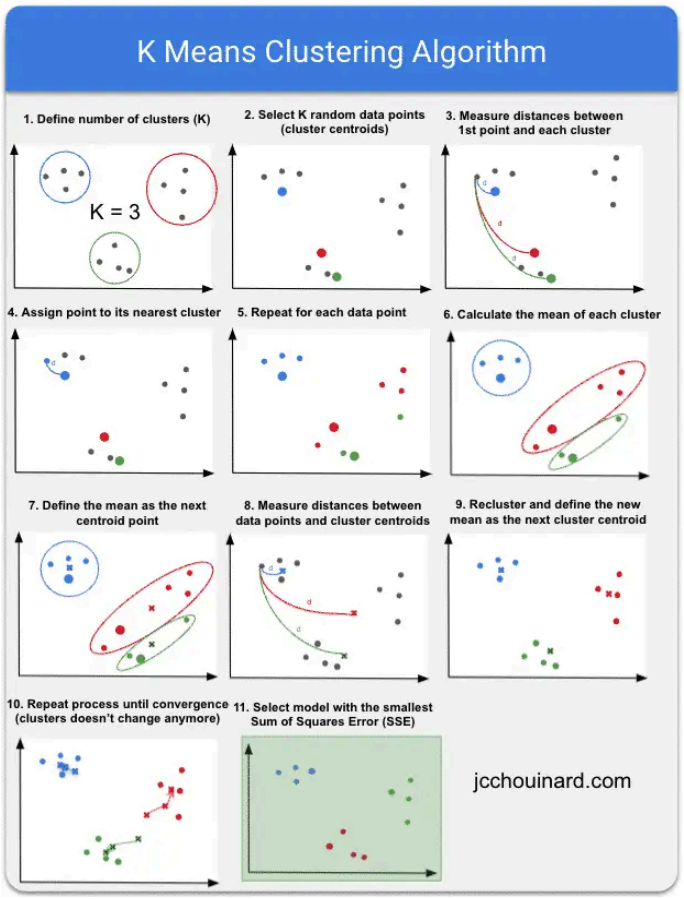
- Cluster similar documents
- Cluster customers based on features
- Market segmentation
- Identify similar physical groups

The overall goal is to divide data into distinct groups such that observations within each group are similar



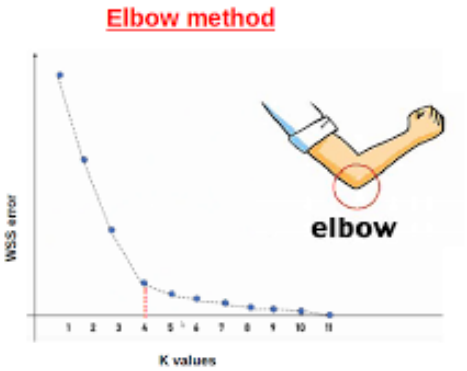
The K Means Algorithm

1. Choose a number of clusters “k”
2. Randomly assign each point to a cluster
3. Until clusters stop changing, repeat the following:
    a. For each cluster, compute the cluster centroid by taking the mean vector of points in the cluster
    b. Assign each data point to the cluster for which the centroid is the closest



There is no easy answer for choosing a best K value, one way is the **elbow method**. First of all, compute the sum of squared error (SSE) for some values of K (2,4,6,8,etc). The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid. If you plot K against SSE you'll see that the error decreases as K gets larger, because as the nmber of clusters increases they should be smaller ➜ distortion is also smaller.

The idea of the **elbow method** is to **choose the K at which the SSE decreases abruptly**. This produces an "elbow effect" in the graph as you can see:



K Means with Python

Import Libraries and Get Data

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Create some data and visualize it

```
from sklearn.datasets import make_blobs
data = make_blobs(n_samples=200, n_features=2,
                  centers=4, cluster_std=1.8,random_state=101)
```

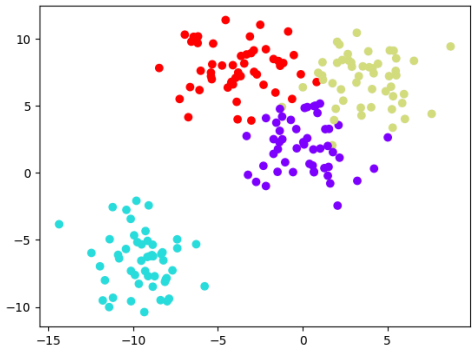
data

```
(array([[-6.42884095e+00,  1.01411174e+01],
        [ 5.86867888e+00,  5.20110356e+00],
        [-3.76109375e-01,  3.26427943e+00],
        [ 2.16679181e+00,  9.56300522e+00],
        [ 5.09508570e+00,  7.20752718e+00],
        [-1.08788882e+01, -6.11318040e+00],
        [ 2.03405554e+00,  9.76664755e+00],
        [-1.71798771e+00,  1.41401140e+00],
        [ 1.16911341e+00,  8.24556988e+00],
        [-1.35185444e+00,  3.13245345e+00],
        [-6.18548214e+00,  9.67406555e+00],
        [-1.19856602e+00,  2.50408937e+00],
        [ 2.90296863e+00,  7.91251003e+00],
        [ 2.39250023e+00,  5.38173971e+00],
        [-5.27545147e+00,  9.63836659e+00],
        [-5.66814687e-01,  5.60262755e-02],
        [ 5.97336628e+00,  5.87172022e+00],
        [-2.31355268e+00,  5.23980092e-01],
        [-1.01344756e+01, -3.43130837e+00],
        [-4.54082629e+00,  1.13920174e+01],
        [-1.04155833e+01, -5.67545836e+00],
        [ 6.64796693e-01,  9.42304718e-02],
        [ 2.11460477e+00,  3.55938488e+00],
        [-1.11790221e+01, -9.30976605e+00],
        [-6.63698251e+00,  6.39426436e+00],
...
        0, 3, 3, 2, 1, 2, 3, 3, 2, 3, 0, 3, 0, 3, 0, 2, 3, 0, 1, 3, 3, 3,
        0, 1, 1, 3, 2, 3, 2, 0, 1, 2, 1, 3, 3, 2, 0, 1, 3, 3, 3, 3, 0, 2,
        0, 3, 2, 2, 2, 0, 2, 0, 0, 3, 1, 3, 0, 2, 3, 0, 2, 0, 3, 3, 0, 3,
        2, 2, 1, 2, 3, 1, 1, 3, 1, 1, 1, 1, 1, 0, 1, 2, 2, 3, 1, 0, 2, 2,
        1, 0]))
```

data[0].shape

(200, 2)

plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')



Creating the Clusters

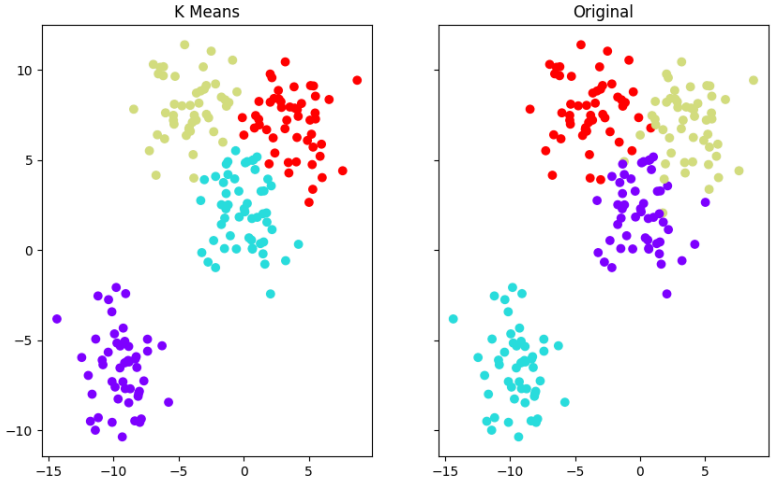
from sklearn.cluster import KMeans
kmeans = KMeans**(n_clusters=4)**
kmeans.fit(data[0])
kmeans.cluster_centers_

```
array([[-9.46941837, -6.56081545],
       [-0.0123077 ,  2.13407664],
       [-4.13591321,  7.95389851],
       [ 3.71749226,  7.01388735]])
```

kmeans.labels_

```
array([2, 3, 1, 3, 3, 0, 3, 1, 3, 1, 2, 1, 3, 3, 2, 1, 3, 1, 0, 2, 0, 1,
       1, 0, 2, 0, 0, 1, 3, 3, 2, 0, 3, 1, 1, 2, 0, 0, 0, 1, 0, 2, 2, 2,
       1, 3, 2, 1, 0, 1, 1, 2, 3, 1, 0, 2, 1, 1, 2, 3, 0, 3, 0, 2, 3, 1,
       0, 3, 3, 0, 3, 1, 0, 1, 0, 3, 3, 1, 2, 1, 1, 0, 3, 0, 1, 1, 1, 2,
       1, 0, 0, 0, 0, 1, 1, 0, 3, 2, 0, 3, 1, 0, 1, 1, 3, 1, 0, 3, 0, 0,
       3, 2, 2, 3, 0, 3, 2, 2, 3, 2, 1, 2, 1, 2, 1, 3, 2, 1, 0, 2, 2, 2,
       1, 0, 0, 2, 3, 2, 3, 1, 0, 3, 0, 2, 2, 3, 1, 0, 2, 2, 2, 2, 1, 3,
       1, 2, 3, 3, 3, 1, 3, 1, 1, 2, 0, 2, 1, 3, 2, 1, 3, 1, 2, 3, 1, 2,
       3, 3, 0, 3, 2, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 3, 3, 2, 0, 1, 3, 3,
       0, 1], dtype=int32)
```

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')



CUANDO HAY QUE COMPARAR CONTRA LA VIDA REAL:

```
def converter(cluster):
   if cluster=='Yes':
        return 1
   else:
        return 0


df['Cluster'] = df['Private'].apply(converter)

from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(df['Cluster'],kmeans.labels_))
print(classification_report(df['Cluster'],kmeans.labels_))
```

```
[[138  74]
 [531  34]]
              precision    recall  f1-score   support

           0       0.21      0.65      0.31       212
           1       0.31      0.06      0.10       565

    accuracy                           0.22       777
   macro avg       0.26      0.36      0.21       777
weighted avg       0.29      0.22      0.16       777
```

Section 22: Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

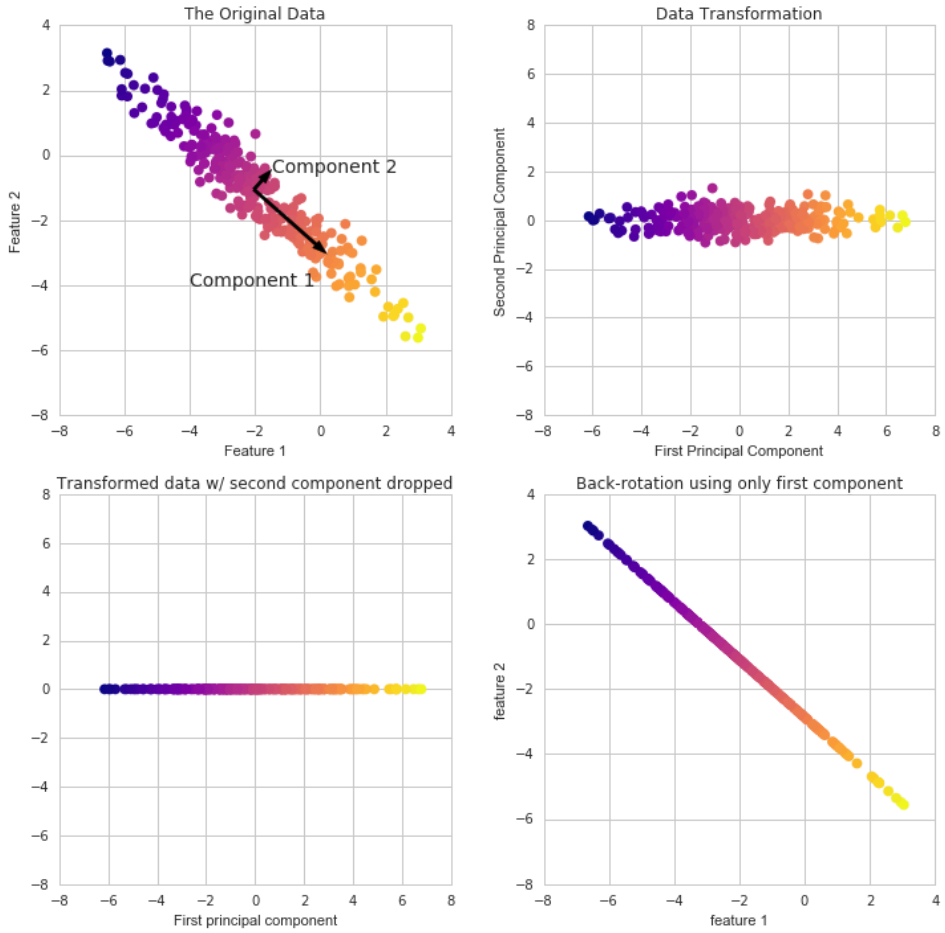
Chapter 10.2 of Introduction to Statistical Learning by Gareth James, et al.

PCA es una técnica artística no supervisada que se utiliza para **examinar las interrelaciones entre un conjunto de variables** con el fin de **identificar la estructura subyacente de esas variables**. También se conoce a veces como un **factor analysis** general donde la **regresión** determina una **línea de mejor ajuste para un conjunto de datos**.

El análisis factorial determina varias líneas ortogonales de mejor ajuste para el conjunto de datos; ortogonal significa ángulos rectos. En realidad, esas líneas son perpendiculares entre sí en el espacio n-dimensional, siendo que el espacio n-dimensional es el espacio muestral variable donde **hay tantas dimensiones como variables** (en un conjunto de datos con 4 variables, el espacio muestral es 4-dimensional).
[[INCOMPLETO PORQUE NO SE ENTENDÍA MUCHO EN EL CURSO]]

PCA With Python

Remember that PCA is just a transformation of your data and attempts to find out what features explain the most variance in your data. For example:



Import Libraries and Get Data

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

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
cancer.keys()
```

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

```
print(cancer['DESCR'])

df = pd.DataFrame(cancer['data'],columns=cancer['feature_names']) #(['DESCR', 'data',
'feature_names', 'target_names', 'target'])
df.head()
```

PCA Visualization

As we've noticed before it is difficult to visualize high dimensional data, we can use PCA to find the first two principal components, and visualize the data in this new, two-dimensional

space, with a single scatter-plot. Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled_data = scaler.transform(df)
```

PCA with Scikit Learn uses a very similar process to other preprocessing functions that come with SciKit Learn. We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform(). We can also specify how many components we want to keep when creating the PCA object.

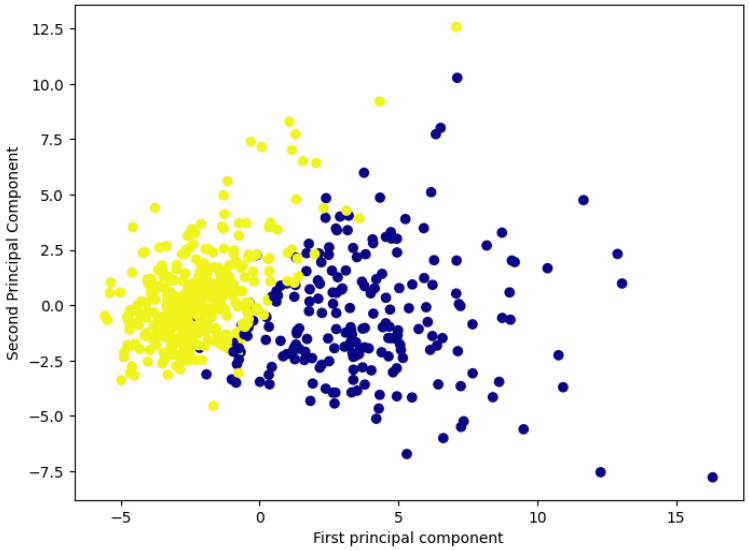
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(scaled_data)
```

Now **we can transform this data to its first 2 principal components.**

```
x_pca = pca.transform(scaled_data)
scaled_data.shape
x_pca.shape
        (569, 2)
```

Great! **We've reduced 30 dimensions to just 2!** Let's plot these two dimensions out!

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```



Clearly by using these two components we can easily separate these two classes.

Interpreting the components

Unfortunately, with this great power of dimensionality reduction, comes the cost of being able to easily understand what these components represent. **The components correspond to combinations of the original features**, the components themselves are stored as an attribute of the fitted PCA object:

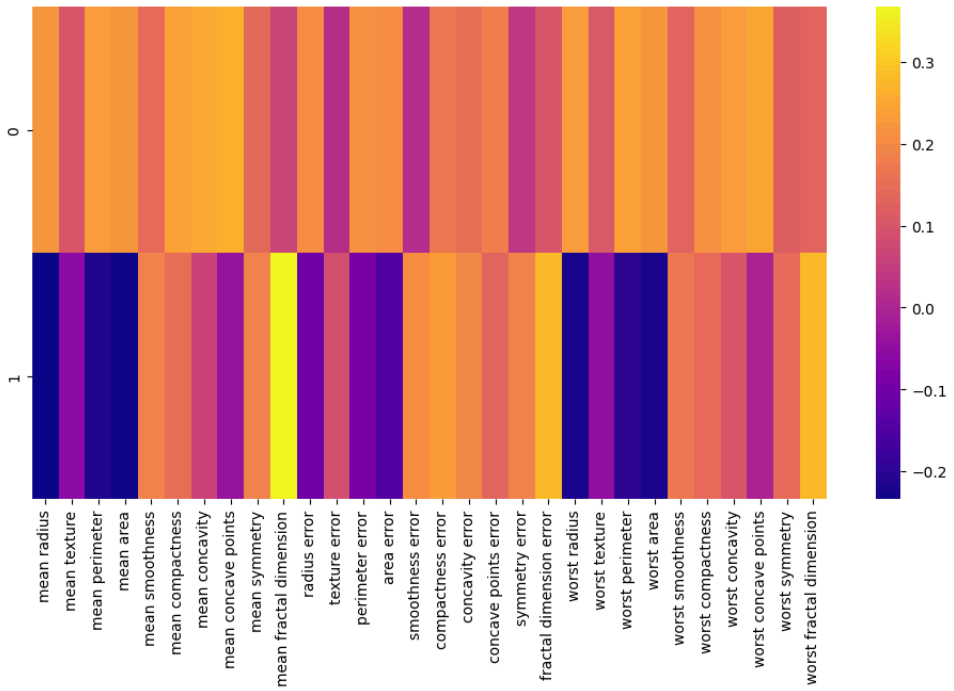
pca.components_

```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

In this numpy matrix array, **each row represents a principal component, and each column relates back to the original features**. We can visualize this relationship with a heatmap:

df_comp = pd.DataFrame(pca.components_,columns=cancer['feature_names'])

plt.figure(figsize=(12,6))
sns.heatmap(df_comp,cmap='plasma',)



This heatmap and the color bar basically represent the correlation between the various feature and the principal component itself. Conclusion: Hopefully this information is useful to you when dealing with high dimensional data!

Section 23: Recommender Systems

Recommender Systems

Libro recomendado: Recommender Systems by Jannach and Zanker

Los **2 tipos más comunes de sistemas de recomendación** son:

- **Collaborative Filtering CF**: El filtrado colaborativo produce **recomendaciones basadas en el conocimiento de la actitud de los usuarios hacia los artículos**. Es decir, utiliza la sabiduría de la multitud para recomendar elementos que se puedan considerar como Amazon, y en base a las experiencias de compra de otras personas, Amazon sugerirá los artículos que cree que podrá disfrutar.
- **Content-Based:** Los sistemas de recomendación basados en el contenido se centran en los atributos de los artículos y le brindan recomendaciones **basadas en la similitud entre los elementos**.

La **diferencia** básica aquí es el filtrado basado en las **preferencias del usuario o** el filtrado en función de la **similitud entre los elementos.**

En general **Collaborative Filtering (CF)** es en general el más usado porque usualmente ofrece **mejores resultados** y es relativamente **fácil de entender** (desde una perspectiva general de implementación). El algoritmo tiene la habilidad de realizar **aprendizaje de características por sí mismo**, lo que significa que **puede comenzar a aprender por sí mismo qué características utilizar al recomendar ciertos artículos**. CF se puede dividir en 2 subcategorías:

- **Memory-Based Collaborative Filtering** (filtrado colaborativo basado en la memoria)
- **Model-Based Collaborative Filtering** (filtrado colaborativo basado en modelos)

En el notebook (github) del tutorial avanzado implementamos Model-Based CF usando la descomposición de valores singulares (singular value decomposition SVD) y Memory-Based CF calculando la similitud del coseno.

Para nuestra implementación en Python vamos a crear un sistema de recomendación basado en el contenido para un conjunto de datos de películas. Estos datos de películas suele ser el primer conjunto de datos del alumno al comenzar a aprender sobre los sistemas de recomendación, es posible que haya oído hablar de ellos antes. Es bastante grande en comparación con algunos de los otros conjuntos de datos con los que hemos trabajado. Pero en general los sistemas de recomendación en la vida real tratan con conjuntos de datos mucho más grandes.

Si crees que tienes el fondo de álgebra lineal necesario, hay muchas explicaciones de texto en el notebook avanzado, por lo que no habrá una guía de video en ese cuaderno porque no es relevante.

Recommender Systems with Python

Import Libraries and Get Data

```
import numpy as np
import pandas as pd
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=column_names)
df.head()
movie_titles = pd.read_csv("Movie_Id_Titles")
movie_titles.head()
df = pd.merge(df,movie_titles,on='item_id')
df.head()
```

Visualization Imports

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
```

```
df.groupby('title')['rating'].mean().sort_values(ascending=False).head()
```

```
title
They Made Me a Criminal (1939)                5.0
Marlene Dietrich: Shadow and Light (1996)     5.0
Saint of Fort Washington, The (1993)          5.0
Someone Else's America (1995)                 5.0
Star Kid (1997)                               5.0
Name: rating, dtype: float64
```

```
df.groupby('title')['rating'].count().sort_values(ascending=False).head()
```

```
title
Star Wars (1977)             584
Contact (1997)               509
Fargo (1996)                 508
Return of the Jedi (1983)    507
Liar Liar (1997)             485
Name: rating, dtype: int64
```

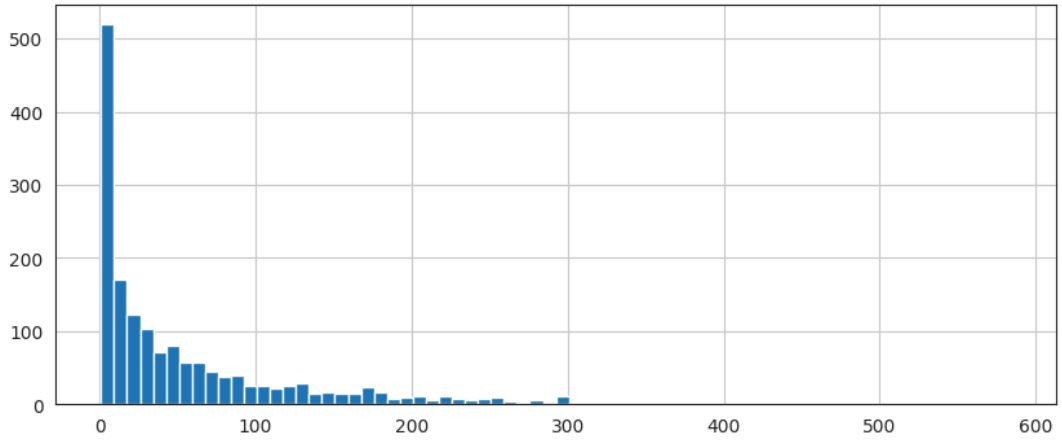
```
ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings.head()
```

	rating
title	
'Til There Was You (1997)	2.333333
1-900 (1994)	2.600000
101 Dalmatians (1996)	2.908257
12 Angry Men (1957)	4.344000
187 (1997)	3.024390

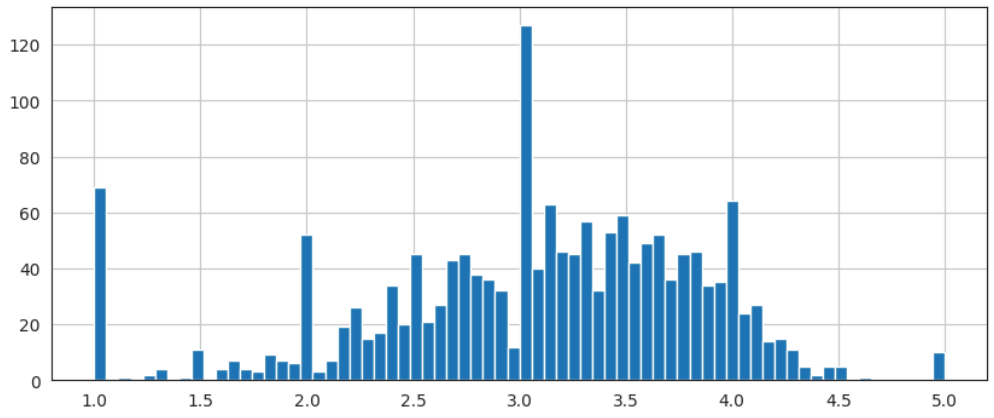
```
ratings['num of ratings'] = pd.DataFrame(df.groupby('title')['rating'].count())
ratings.head()
```

title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

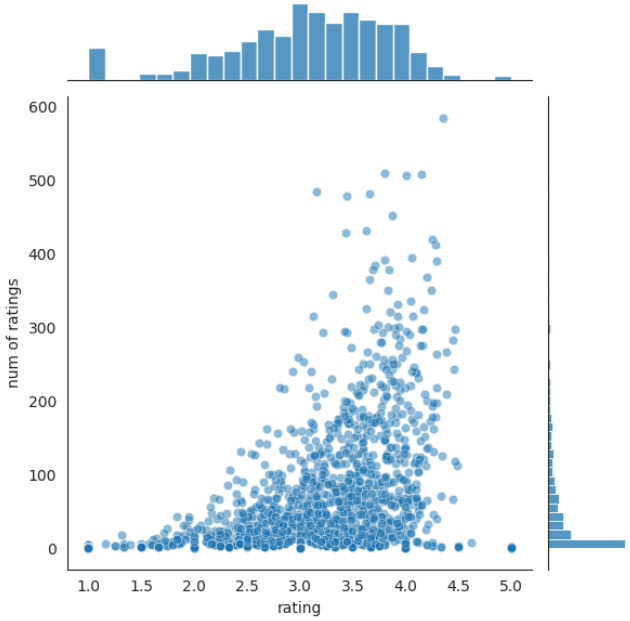
```
plt.figure(figsize=(10,4))
ratings['num of ratings'].hist(bins=70)
```




```
plt.figure(figsize=(10,4))
ratings['rating'].hist(bins=70)
```




```
sns.jointplot(x='rating',y='num of ratings',data=ratings,alpha=0.5)
```



Recommending Similar Movies

Now let's create a matrix that has the user ids on one access and the movie title on another axis. Each cell will then consist of the rating the user gave to that movie. Note there will be a lot of NaN values, because most people have not seen most of the movies.

moviemat = df.pivot_table(index='user_id',columns='title',values='rating')
moviemat.head()

title	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)	Young Guns (1988)	Young Guns II (1990)	Young Poisoner's Handbook, The (1995)
user_id																		
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN	...	NaN	NaN	NaN	5.0	3.0	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1664 columns

#Most Rated Movie
ratings.sort_values('num of ratings',ascending=False).head(10)

title	rating	num of ratings
Star Wars (1977)	4.359589	584
Contact (1997)	3.803536	509
Fargo (1996)	4.155512	508
Return of the Jedi (1983)	4.007890	507
Liar Liar (1997)	3.156701	485
English Patient, The (1996)	3.656965	481
Scream (1996)	3.441423	478
Toy Story (1995)	3.878319	452
Air Force One (1997)	3.631090	431
Independence Day (ID4) (1996)	3.438228	429

Let's choose two movies: starwars, a sci-fi movie. And Liar Liar, a comedy.
ratings.head()

title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

Now let's grab the user ratings for those 2 movies:
starwars_user_ratings = moviemat['Star Wars (1977)']
liarliar_user_ratings = moviemat['Liar Liar (1997)']
starwars_user_ratings.head()

```
user_id
0    5.0
1    5.0
2    5.0
3    NaN
4    5.0
Name: Star Wars (1977), dtype: float64
```

We can then use corrwith() method to get correlations between 2 pandas series:
similar_to_starwars = moviemat.corrwith(starwars_user_ratings)
similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)

Let's clean this by r**emoving NaN values and using a DataFrame instead of a series**:
corr_starwars = pd.DataFrame(similar_to_starwars,columns=['Correlation'])
corr_starwars.dropna(inplace=True)
corr_starwars.head()

title	Correlation
'Til There Was You (1997)	0.872872
1-900 (1994)	-0.645497
101 Dalmatians (1996)	0.211132
12 Angry Men (1957)	0.184289
187 (1997)	0.027398

Now if we sort the dataframe by correlation, we should get the most similar movies, however note that we get some results that don't really make sense. This is because there are a lot of movies only watched once by users who also watched star wars (it was the most popular movie).

corr_starwars.sort_values('Correlation',ascending=False).head(10)

title	Correlation
Hollow Reed (1996)	1.0
Commandments (1997)	1.0
Cosi (1996)	1.0
No Escape (1994)	1.0
Stripes (1981)	1.0
Star Wars (1977)	1.0
Man of the Year (1995)	1.0
Beans of Egypt, Maine, The (1994)	1.0
Old Lady Who Walked in the Sea, The (Vieille qui marchait dans la mer, La) (1991)	1.0
Outlaw, The (1943)	1.0

Let's fix this by **filtering out movies that have less than 100 reviews** (this value was chosen based off the histogram from earlier).

corr_starwars = corr_starwars.join(ratings['num of ratings'])
corr_starwars.head()

title	Correlation	num of ratings
'Til There Was You (1997)	0.872872	9
1-900 (1994)	-0.645497	5
101 Dalmatians (1996)	0.211132	109
12 Angry Men (1957)	0.184289	125
187 (1997)	0.027398	41

Now sort the values and notice how the titles make a lot more sense:
corr_starwars[corr_starwars['num of ratings'] > 100].sort_values('Correlation', ascending = False).head()

title	Correlation	num of ratings
Star Wars (1977)	1.000000	584
Empire Strikes Back, The (1980)	0.748353	368
Return of the Jedi (1983)	0.672556	507
Raiders of the Lost Ark (1981)	0.536117	420
Austin Powers: International Man of Mystery (1997)	0.377433	130

Now the same for the comedy Liar Liar:
corr_liarliar = pd.DataFrame(similar_to_liarliar,columns=['Correlation'])
corr_liarliar.dropna(inplace=True)
corr_liarliar = corr_liarliar.join(ratings['num of ratings'])
corr_liarliar[corr_liarliar['num of ratings']>100].sort_values('Correlation',ascending=False).head()

title	Correlation	num of ratings
Liar Liar (1997)	1.000000	485
Batman Forever (1995)	0.516968	114
Mask, The (1994)	0.484650	129
Down Periscope (1996)	0.472681	101
Con Air (1997)	0.469828	137

Advanced - Recommender Systems with Python

In this tutorial, we will implement Model-Based CF by using singular value decomposition (SVD) and Memory-Based CF by computing cosine similarity.

We will use famous MovieLens dataset, which is one of the most common datasets used when implementing and testing recommender engines. It contains 100k movie ratings from 943 users and a selection of 1682 movies. You can download the dataset here: http://files.grouplens.org/datasets/movielens/ml-100k.zip

Import Libraries and Get Data

```
import numpy as np
import pandas as pd
```

We can then read in the u.data file, which contains the full dataset. You can read a brief description of the dataset here: http://files.grouplens.org/datasets/movielens/ml-100k-README.txt. Note how we specify the separator argument for a Tab separated file.

```
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=column_names)
```

Note how we only have the item_id, not the movie name. We can use the Movie_ID_Titles csv file to grab the movie names and merge it with this dataframe:

```
movie_titles = pd.read_csv("Movie_Id_Titles")
```

Then merge the dataframes:
df = pd.merge(df,movie_titles,on='item_id')
df.head()

Now let's take a quick look at the number of unique users and movies.

n_users = df.user_id.nunique()
n_items = df.item_id.nunique()
print('Num. of Users: '+ str(n_users))
print('Num of Movies: '+str(n_items))

Num. of Users: 944
Num of Movies: 1682

Train Test Split

Recommendation Systems by their very nature are **very difficult to evaluate**, but we will still show you how to evaluate them in this tutorial. In order to do this, we'll split our data into two sets. However, we won't do our classic X_train,X_test,y_train,y_test split. Instead we can actually **just segement the data into 2 sets of data**:

#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.25)

Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering approaches can be divided into 2 main sections:

- **User-item filtering:** will take a particular **user**, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. **“users who are similar to you also liked …”**
- **Item-item filtering:** will take an **item**, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations. **“users who liked this item also liked …”**

En ambos casos, se crea una user-item matrix que se construye a partir de todo el conjunto de datos. Como hemos dividido los datos en pruebas y entrenamiento, necesitaremos crear 2 matrices [943 x 1682` (todos los usuarios por todas las películas). La matriz de entrenamiento contiene el 75% de las calificaciones y la matriz de pruebas contiene el 25%.

Una vez que haya creado la matriz user-item, calcule la similitud y cree una matriz de similitud.

- Los valores de similitud entre elementos en **Item-Item Collaborative Filtering** se miden observando a todos los usuarios que han calificado ambos elementos.
- Para el **User-Item Collaborative Filtering**, los valores de similitud entre usuarios se miden observando todos los elementos calificados por ambos usuarios.

Una **métrica** de distancia comúnmente utilizada en los sistemas de recomendación es la **similitud de coseno**, donde **las calificaciones se ven como vectores en un espacio n-dimensional y la similitud se calcula en función del ángulo entre estos vectores**.

La **similitud del coseno** para los **usuarios a y m** se puede calcular usando la siguiente fórmula, donde se toma el producto escalar del vector del usuario uk y el vector del usuario ua y se divide por la multiplicación de las longitudes euclidianas de los vectores.

$$s_u^{cos}(u_k, u_a) = \frac{u_k \cdot u_a}{||u_k||\,||u_a||} = \frac{\sum x_{k,m} x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

Para calcular la **similitud** entre los **elementos m y b** se utiliza la fórmula:

$$s_u^{cos}(i_m, i_b) = \frac{i_m \cdot i_b}{||i_m||\,||i_b||} = \frac{\sum x_{a,m} x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

El primer paso será crear la **matriz user-item** . Dado que tiene **datos de prueba y de entrenamiento**, necesita crear **2 matrices**.

#Create two user-item matrices, one for training and another for testing
train_data_matrix = np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]

Puede utilizar la **función pairwise_distances** (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html) de sklearn **para calcular la similitud del coseno**. Tenga en cuenta que el resultado variará de 0 a 1 ya que todas las calificaciones son positivas.

from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')

Predictions

El siguiente paso es hacer predicciones. Ya ha creado **matrices de similitud**: `**user_similarity**` y `**item_similarity**` y, por lo tanto, puede hacer una **predicción aplicando la siguiente fórmula para CF basada en usuarios:**

$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum_{u_a} sim_u(u_k, u_a)(x_{a,m} - \bar{x_{u_a}})}{\sum_{u_a} |sim_u(u_k, u_a)|}$$

Puede observar la similitud entre los usuarios k y a como pesos que se multiplican por las calificaciones de un usuario similar a (corregido por la calificación promedio de ese usuario). Deberá normalizarlo para que las calificaciones se mantengan entre 1 y 5 y, como paso final, sumar las calificaciones promedio del usuario que está tratando de predecir.

La idea aquí es que algunos usuarios tiendan siempre a otorgar calificaciones altas o bajas a todas las películas. La diferencia relativa en las valoraciones que dan estos usuarios es más importante que los valores absolutos. Para dar un ejemplo: supongamos que el usuario *k* otorga 4 estrellas a sus películas favoritas y 3 estrellas a todas las demás buenas películas. Supongamos ahora que otro usuario *t* califica las películas que le gustan con 5 estrellas y las películas sobre las que se quedó dormido con 3 estrellas. Estos dos usuarios podrían tener gustos muy similares pero tratar el sistema de calificación de manera diferente.

Al realizar una predicción para CF basada en elementos, no es necesario corregir la calificación promedio de los usuarios, ya que el propio usuario de consulta se utiliza para hacer predicciones.

$$\hat{x}_{k,m} = \frac{\sum_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum_{i_b} |sim_i(i_m, i_b)|}$$

```
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        #You use np.newaxis so that mean_user_rating has same format as ratings
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) /
np.array([np.abs(similarity).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred

item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

Evaluation

There are many evaluation metrics but one of the most popular metric used to evaluate accuracy of predicted ratings is Root Mean Squared Error (RMSE).

$$RMSE = \sqrt{\frac{1}{N}\sum (x_i - \hat{x}_i)^{2"} title = "RMSE = \sqrt{\frac{1}{N}\sum (x_i - \hat{x}_i)^2}$$

Puede utilizar la función mean_square_error (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) (MSE) de `sklearn`, donde el RMSE es solo la raíz cuadrada de MSE. Para leer más sobre las diferentes **métricas de evaluación**, puede consultar este artículo (http://research.microsoft.com/pubs/115396/EvaluaciónMetrics.TR.pdf).

Dado que solo desea considerar las calificaciones pronosticadas que se encuentran en el **conjunto de datos de prueba**, filtra todos los demás elementos en la matriz de predicción con `prediction[ground_truth.nonzero()]`.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))

print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

```
User-based CF RMSE: 3.13466977660475
Item-based CF RMSE: 3.4602070996774246
```

Los algoritmos memory-based son fáciles de implementar y producen una calidad de predicción razonable. El inconveniente del memory-based CF es que no se adapta a escenarios del mundo real y no aborda el conocido problema de arranque en frío, es decir, cuando un nuevo usuario o un nuevo elemento ingresa al sistema.

Los métodos model-based CF son escalables y pueden manejar niveles de escasez más altos que los modelos basados en memoria, pero también sufren cuando nuevos usuarios o elementos que no tienen ninguna calificación ingresan al sistema.

Me gustaría agradecer a Ethan Rosenthal por su publicación (http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/) sobre el Memory-Based Collaborative Filtering.

Model-based Collaborative Filtering

Model-based Collaborative Filtering (CF) se basa en la **factorización matricial (MF)**, que ha recibido una mayor exposición, principalmente como **método de aprendizaje no supervisado para la descomposición de variables latentes y la reducción de dimensionalidad**. La factorización matricial se usa ampliamente para sistemas de recomendación donde puede **lidiar mejor con la escalabilidad y la escasez que la Memory-based CF**. El objetivo de MF es **aprender las preferencias latentes de los usuarios y los atributos latentes de los elementos a partir de calificaciones conocidas (aprender características que describen las características de las calificaciones)** para luego **predecir las calificaciones desconocidas a través del producto escalar de las características latentes de los usuarios y los elementos**. Cuando se tiene una matriz muy escasa, con muchas dimensiones, al realizar la factorización matricial se puede **reestructurar la matriz de elementos de usuario en una estructura de bajo rango y se puede representar la matriz mediante la multiplicación de dos matrices de bajo rango, donde las filas contienen el vector latente**. Esta matriz se ajusta para aproximarse lo más posible a su matriz original, multiplicando las matrices de bajo rango, lo que completa las entradas que faltan en la matriz original.

Calculemos el nivel de escasez (sparsity level) del conjunto de datos MovieLens:

sparsity=round(1.0-len(df)/float(n_users*n_items),3)
print('The sparsity level of MovieLens100K is ' + str(sparsity*100) + '%')

```
The sparsity level of MovieLens100K is 93.7%
```

Para dar un ejemplo de las preferencias latentes aprendidas de los usuarios y elementos: digamos que para el conjunto de datos MovieLens tiene la siguiente información: (*identificación de usuario, edad, ubicación, género, identificación de película, director, actor, idioma, año, calificación*). **Al aplicar la factorización matricial, el modelo aprende que las**

características importantes del usuario son *grupo de edad* (menores de 10 años, 10-18, 18-30, 30-90), *ubicación* y *género*, y para las **características de películas** aprende que ***década***, *director* y *actor* son los más importantes. Ahora bien, si observa la información que ha almacenado, **no existe una característica llamada década, pero el modelo puede aprender por sí solo**. El aspecto importante es que el modelo CF solo utiliza datos (user_id, movie_id, rating) para conocer las características latentes. Si hay pocos datos disponibles, el modelo de CF basado en modelos predecirá mal, ya que será más difícil aprender las características latentes.

##Hybrid Recommender Systems

Los **modelos que utilizan calificaciones y características** de contenido se denominan **Hybrid Recommender Systems** (sistemas de recomendación híbridos) donde **se combinan el filtrado colaborativo y los modelos basados en contenido**. Los sistemas de recomendación híbridos suelen mostrar una mayor precisión que el filtrado colaborativo o los modelos basados en contenido por sí solos: son capaces de abordar mejor el problema del arranque en frío, ya que si no tienes ninguna calificación para un usuario o un elemento, puedes usar los metadatos. del usuario o elemento para hacer una predicción.

##SVD

Un **método de factorización matricial** muy conocido es la **descomposición en valores singulares (SVD)**. El filtrado colaborativo se puede formular **aproximando una matriz "X" mediante descomposición de valores singulares**. El equipo ganador del concurso del Premio Netflix utilizó modelos de factorización matricial SVD para producir recomendaciones de productos. Para obtener más información, recomiendo leer los artículos:

- Recomendaciones de Netflix: Más allá de las 5 estrellas (http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html)
- Premio Netflix y SVD (http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf).

La ecuación general se puede expresar de la siguiente manera:

$$X = USV^T" title = "X = USV^T$$

Dada la matriz mxn `X`:

- `U`* es una matriz ortogonal `(m x r)`
- `S`* es una matriz diagonal `(r x r)` con números reales no negativos en la diagonal
- V^T* es una matriz ortogonal `(r x n)`

Los elementos de la diagonal de `S` se conocen como **valores singulares de `X`**.

La **matriz `X` se puede factorizar en `U`, `S` y `V`.** La **matriz `U`** representa los **vectores de características correspondientes a los usuarios en el espacio de características ocultas** y la **matriz `V`** representa los **vectores de características correspondientes a los elementos en el espacio de características ocultas.**

Ahora puedes hacer una **predicción tomando el producto escalar de `U`, `S` y `V^T`.**

```
import scipy.sparse as sp
from scipy.sparse.linalg import svds

#get SVD components from train matrix. Choose k.
u, s, vt = svds(train_data_matrix, k = 20)
s_diag_matrix=np.diag(s)
X_pred = np.dot(np.dot(u, s_diag_matrix), vt)
print('User-based CF MSE: ' + str(rmse(X_pred, test_data_matrix)))
```

User-based CF MSE: 2.735387160142127

Abordar descuidadamente sólo las relativamente pocas entradas conocidas es muy propenso a un sobreajuste. SVD puede ser muy lento y costoso desde el punto de vista computacional. Un trabajo más reciente minimiza el error al cuadrado aplicando un descenso de gradiente estocástico o de mínimos cuadrados alterno y utiliza términos de regularización para evitar el sobreajuste. En los próximos tutoriales se cubrirá la alternancia de métodos de descenso de gradiente estocástico y de mínimos cuadrados para CF.

Resumen

- Hemos cubierto cómo implementar métodos simples de Filtrado colaborativo, tanto CF basados en memoria como CF basados en modelos.
- Los modelos basados en memoria se basan en la similitud entre elementos o usuarios, donde utilizamos la similitud del coseno.
- CF basado en modelo se basa en la factorización matricial donde utilizamos SVD para factorizar la matriz.
- La creación de sistemas de recomendación que funcionen bien en escenarios de arranque en frío (donde hay pocos datos disponibles sobre nuevos usuarios y elementos) sigue siendo un desafío. El método de filtrado colaborativo estándar funciona mal en tales configuraciones.

Datasets para hacer recommendation system analusis

If you want to tackle your own recommendation system analysis, check out these data sets. Note: The files are quite large in most cases, not all the links may stay up to host the data, but the majority of them still work. Or just Google for your own data set!

Movies

- MovieLens - Movie Recommendation Data Sets http://www.grouplens.org/node/73
- Yahoo! - Movie, Music, and Images Ratings Data Sets http://webscope.sandbox.yahoo.com/catalog.php?datatype=r
- Jester - Movie Ratings Data Sets (Collaborative Filtering Dataset) http://www.ieor.berkeley.edu/~goldberg/jester-data/
- Cornell University - Movie-review data for use in sentiment-analysis experiments http://www.cs.cornell.edu/people/pabo/movie-review-data/

Music

- Last.fm - Music Recommendation Data Sets http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html
- Yahoo! - Movie, Music, and Images Ratings Data Sets http://webscope.sandbox.yahoo.com/catalog.php?datatype=r
- Audioscrobbler - Music Recommendation Data Sets http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html
- Amazon - Audio CD recommendations http://131.193.40.52/data/

Books

- Institut für Informatik, Universität Freiburg - Book Ratings Data Sets http://www.informatik.uni-freiburg.de/~cziegler/BX/

Food

- Chicago Entree - Food Ratings Data Sets http://archive.ics.uci.edu/ml/datasets/Entree+Chicago+Recommendation+Data

Healthcare

- Nursing Home - Provider Ratings Data Set http://data.medicare.gov/dataset/Nursing-Home-Compare-Provider-Ratings/mufm-vy8d
- Hospital Ratings - Survey of Patients Hospital Experiences http://data.medicare.gov/dataset/Survey-of-Patients-Hospital-Experiences-HCAHPS-/rj76-22dk

Dating

- www.libimseti.cz - Dating website recommendation (collaborative filtering) http://www.occamslab.com/petricek/data/

Scholarly Paper

- National University of Singapore - Scholarly Paper Recommendation http://www.comp.nus.edu.sg/~sugiyama/SchPaperRecData.html

Section 24: Natural Language Processing

Natural Language Processing Theory

https://en.wikipedia.org/wiki/Natural_language_processing

El procesamiento de lenguaje natural (NLP) tiene muchos casos de uso cuando se trata de texto o datos de texto no estructurados.

Imagine que trabaja para Google News y desea agrupar artículos periodísticos por tema, o tal vez trabaje para una firma legal y necesite examinar miles de páginas de documentos legales

para encontrar los relevantes. Aquí es donde el procesamiento del lenguaje natural puede ayudar.

Vamos a querer:

- Compilar los documentos de alguna manera
- Obtener funciones de ellos, caracterizarlos
- Comparar sus características.

Veamos un ejemplo simple, donde tenemos 2 documentos:

- “Blue House”
- “Red House”

Caracterización basada en recuento de palabras: Creamos un conteo de palabras vectorizado, creamos un número de vectores de todas las palabras posibles y arrojamos todos los documentos. Transformamos casa azul en un conteo de palabras vectorizado.

- **“Blue House” ➔ (red, blue, house) ➔ (0,1,1)**
- **“Red House” ➔ (red, blue, house) ➔ (1,0,1)**

Un documento representado como un vector de recuento de palabras recibe el nombre de bolsa de palabras (**bag of words**)

- “Blue House” ➔ (red, blue, house) ➔ (0,1,1)
- “Red House” ➔ (red, blue, house) ➔ (1,0,1)

Una vez que se tengan estas bolsas de vectores de palabras, **se puede usar la similitud de coseno (cosine similarity) en los vectores para determinar la similitud de los documentos**.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Esto es realmente útil porque básicamente **tratamos cada documento como un vector de características, lo que significa que podemos realizar operaciones matemáticas** como la similitud de Cocytus tomando sus productos escalares y luego dividirlo por la multiplicación de sus magnitudes, o podemos usar otras métricas de similitud para calcular qué tan similares son 2 documentos de texto entre sí.

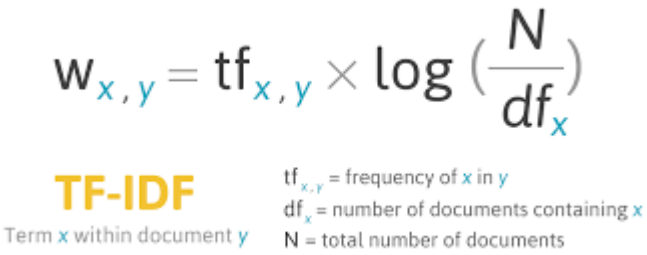
Podemos mejorar en bag of words al ajustar los recuentos de palabras en función de su frecuencia en el **corpus** (el corpus es el grupo de todos los documentos). Podemos usar **TF - IDF** (Term Frequency - Inverse Document Frequency).

➔ **Term Frequency (TF):** La frecuencia de término es la importancia de dicho término dentro del documento.

- ◆ TF(d,t): Número de ocurrencias del término t en el documento d.

- ➔ **Inverse Document Frequency (IDF):** La frecuencia inversa del documento es la importancia del término en el corpus (en el grupo de todos los documentos)
  - ◆ IDF(t) = log(D/t) donde D es el número total de documentos y t es el número de documentos con el término.

Matemáticamente **TD-IDF** se expresa:



La razón por la que hacemos esto es para que podamos obtener tanto un conteo de palabras como también algún tipo de notación sobre cuán importante una palabra es relevante para el documento y para todo el corpus de todos los documentos.

Sin embargo, tendremos que descargar una biblioteca adicional, seguir adelante y acceder a su terminal o línea de comando y usar la instalación y LPK o pipit. instalar una vieja T. K. Dependiendo de la distribución de Python, estás usando la NL T. K. la biblioteca o el paquete nos permitirán trabajar con lenguaje natural o datos de texto muy fácilmente.

NLP with Python

pip install nltk

We'll be using a UCI dataset (https://archive.ics.uci.edu/dataset/228/sms+spam+collection). The file we are using contains a collection of more than 5 thousand SMS phone messages.

```
## Import Libraries and Get Data
# Let's go ahead and use rstrip() plus a list comprehension to get a list of all the lines of text messages:
messages = [line.rstrip() for line in open('smsspamcollection/SMSSpamCollection')]
print(len(messages))
```

5574

```
# A collection of texts is also sometimes called "corpus". Let's print the first ten messages and number them using **enumerate**:
for message_no, message in enumerate(messages[:10]):
    print(message_no, message)
    print('\n')
```

```
0 ham   Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...

1 ham   Ok lar... Joking wif u oni...

2 spam  Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

3 ham   U dun say so early hor... U c already then say...

4 ham   Nah I don't think he goes to usf, he lives around here though

5 spam  FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv

6 ham   Even my brother is not like to speak with me. They treat me like aids patent.

7 ham   As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Caller

8 spam  WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only
...

9 spam  Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Due to the spacing we can tell that this is a **“tab separated values” (TSV) file** (http://en.wikipedia.org/wiki/Tab-separated_values) where **the first column is a label saying whether the given message is a normal message (commonly known as "ham") or "spam". The second column is the message itself.** (Note our **numbers aren't part of the file, they are just from the enumerate call)**. Using these labeled ham and spam examples, we'll train a machine learning model to learn to discriminate between ham/spam automatically. Then, with a trained model, we'll be able to classify arbitrary unlabeled messages as ham or spam. Instead of parsing TSV manually using Python, we can just take advantage of pandas!

import pandas as pd

We'll use read_csv and make note of the sep argument, we can also specify the desired column names by passing in a list of names.

messages = pd.read_csv('smsspamcollection/SMSSpamCollection', sep='\t', names=["label", "message"])

messages.head()

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

EDA

messages.describe()

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

messages.groupby('label').describe()

		message
label		
ham	count	4825
	unique	4516
	top	Sorry, I'll call later
	freq	30
spam	count	747
	unique	653
	top	Please call our customer service representativ...
	freq	4

As we continue our analysis we want to start thinking about the features we are going to be using. This goes along with the general idea of feature engineering (https://en.wikipedia.org/wiki/Feature_engineering). The better your domain knowledge on the data, the better your ability to engineer more features from it. Feature engineering is a very large part of spam detection in general. I encourage you to read up on the topic! Let's make a new column to detect how long the text messages are:

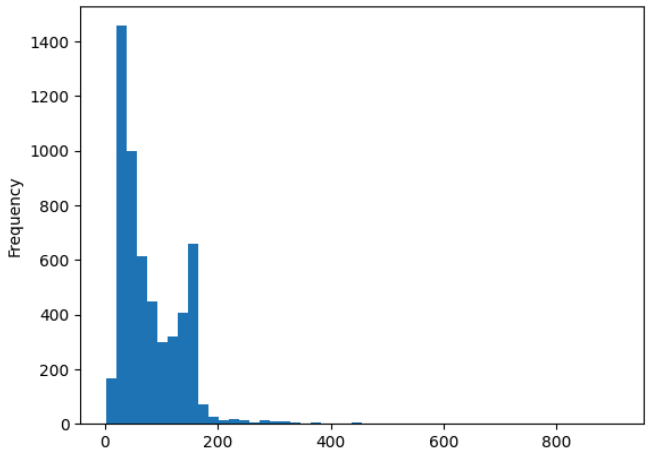
messages['length'] = messages['message'].apply(len)
messages.head()

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Data Visualization

import matplotlib.pyplot as plt
import seaborn as sns

messages['length'].plot(bins=50, kind='hist')



Let's try to explain why the x-axis goes all the way to 1000ish, this must mean that there is some really long message!

messages.length.describe()

```
count    5572.000000
mean       80.489950
std        59.942907
min         2.000000
25%        36.000000
50%        62.000000
75%       122.000000
max       910.000000
Name: length, dtype: float64
```

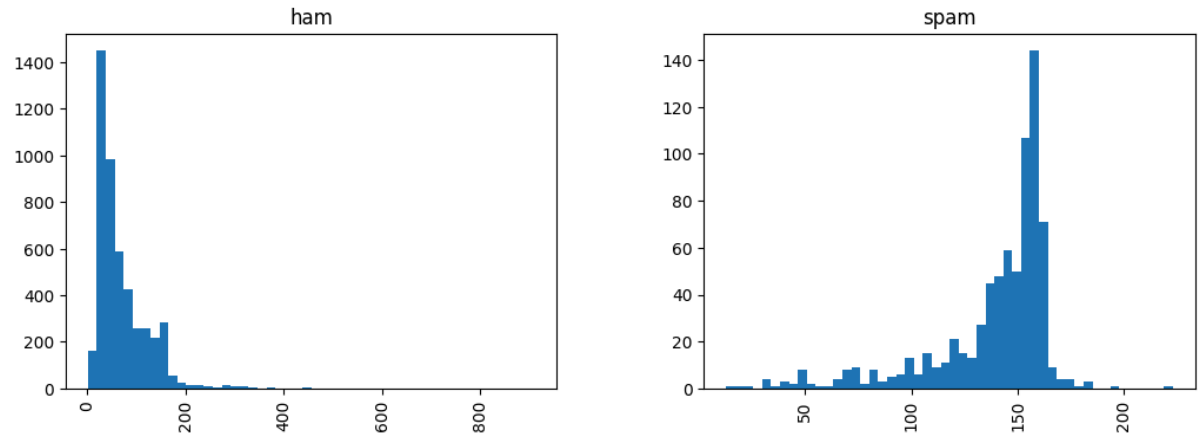
Find this message

messages[messages['length'] == 910]['message'].iloc[0]

"For me the love should start with attraction.i should feel that I need her every time around me.she should be the first thing which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then when my every breath has her name.my life should happen around her.my life will be named to her.I would cry for her.will give all my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the craziest things for her.love will be when I don't have to proove anyone that my girl is the most beautiful lady on the whole planet.I will always be singing praises for her.love will be when I start up making chicken curry and end up makiing sambar.life will be the most beautiful then.will get every morning and thank god for the day because she is with me.I would like to say a lot..will tell later.."

Let's focus back on the idea of trying to see if message length is a distinguishing feature between ham and spam:

messages.hist(column='length', by='label', bins=50,figsize=(12,4))



Text Pre-Processing

Our main issue with our data is that it is all in text format (strings). The classification algorithms that we've learned about so far **will need some sort of numerical feature vector in order to perform the classification task**. There are actually **many methods to convert a corpus to a**

vector format. The **simplest is the bag-of-words** (http://en.wikipedia.org/wiki/Bag-of-words_model) approach, where **each unique word in a text will be represented by one number.**

In this section we'll convert the raw messages (sequence of characters) into vectors (sequences of numbers).

First removing punctuation. We can just take advantage of Python's built-in string library to **get a quick list of all the possible punctuation.** Example::

```
import string
mess = 'Sample message! Notice: it has punctuation.'
# Crea una lista que excluye cualquier carácter presente en string.punctuation. Recorre c/carácter de “mess” y lo incluye en la lista solo si no es un signo de puntuación.
nopunc = [char for char in mess if char not in string.punctuation]
# Join the characters again to form the string.
nopunc = ''.join(nopunc)


#from nltk.corpus import stopwords
import nltk
#nltk.download('stopwords')
# El siguiente código muestra las primeras 10 stopwords (palabras vacías) en inglés. Las stopwords son palabras comunes que generalmente se excluyen en el procesamiento de lenguaje natural (NLP), ya que no aportan mucho significado (por ejemplo, "the", "is", "in").
stopwords.words('english')[0:10] # Show some stop words
    ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]

# Separar en palabras
#df['text length'] = df['text'].apply(len)
nopunc.split()
    ['Sample', 'message', 'Notice', 'it', 'has', 'punctuation']


# Elimina las stopwords (palabras vacías) de una cadena que ya ha sido limpiada de signos de puntuación (en este caso, la variable nopunc)
clean_mess = [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
clean_mess
    ['Sample', 'message', 'Notice', 'punctuation']

# Now let's put both of these together in a function to apply it to our DataFrame later on:
def text_process(mess):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
```

```
    3. Returns a list of the cleaned text
    """
    # Check characters to see if they are in punctuation
    nopunc = [char for char in mess if char not in string.punctuation]
    # Join the characters again to form the string.
    nopunc = ''.join(nopunc)
    # Now just remove any stopwords
    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

# Original dataframe again:
messages.head()

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Now let's "tokenize" these messages. Tokenization is just the term used to describe the process of converting the normal text strings in to a list of tokens (words that we actually want). Let's see an example output on on column:
Note: We may get some warnings or errors for symbols we didn't account for or that weren't in Unicode (like a British pound symbol)

Check to make sure its working
messages['message'].head(5).apply(text_process)

```
0    [Go, jurong, point, crazy, Available, bugis, n...
1                        [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3        [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: message, dtype: object
```

# Show original dataframe
messages.head()

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Continuing Normalization

There are a lot of ways to continue normalizing this text. Such as **Stemming** (https://en.wikipedia.org/wiki/Stemming) or distinguishing by **part of speech** (http://www.nltk.org/book/ch05.html).

NLTK has lots of built-in tools and great documentation on a lot of these methods. Sometimes they don't work well for text-messages due to the way a lot of people tend to use abbreviations or shorthand, For example: 'Nah dawg, IDK! Wut time u headin to da club?' versus 'No dog, I don't know! What time are you heading to the club?'

Some text normalization methods will have trouble with this type of shorthand and so I'll leave you to explore those more advanced methods through the NLTK book online (http://www.nltk.org/book/).

For now we will just focus on using what we have to convert our list of words to an actual vector that SciKit-Learn can use.

Vectorization

Currently, we have the messages as lists of tokens (also known as **lemmas** (http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html)) and **now we need to convert** each of those messages **into a vector** the SciKit Learn's algorithm models can work with. So we'll convert each message, represented as a list of tokens (lemmas) above, into a vector that machine learning models can understand.

We'll do that in **3 steps using the bag-of-words model:**

1. Count how many times does a word occur in each message (**term frequency**)
2. Weigh the counts, so that frequent tokens get lower weight (**inverse document frequency**)
3. Normalize the vectors to unit length, to abstract from the original text length (**L2 norm**)

Each vector will have as many dimensions as there are unique words in the SMS corpus. We will first use SciKit Learn's **CountVectorizer**. This model will convert a collection of text documents to a matrix of token counts. We can imagine this as a 2-Dimensional matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents, in this case a column per text message. For example:

	Message 1	**Message 2**	**...**	**Message N**
Word 1 Count	0	1	...	0
Word 2 Count	0	0	...	0
...	1	2	...	0
Word N Count	0	1	...	1

Since there are so many messages, we can expect a lot of zero counts for the presence of that word in that document. Because of this, SciKit Learn will output a Sparse Matrix (https://en.wikipedia.org/wiki/Sparse_matrix).

There are a lot of arguments and parameters that can be passed to the CountVectorizer. In this case we will just specify the analyzer to be our own previously defined function:

```
from sklearn.feature_extraction.text import CountVectorizer
bow_transformer = CountVectorizer(analyzer=text_process).fit(messages['message'])
# Print total number of vocab words
print(len(bow_transformer.vocabulary_))
```

11425

Let's take one text message and get its bag-of-words counts as a vector, putting to use our new `bow_transformer`:

```
message4 = messages['message'][3]
print(message4)
```

U dun say so early hor... U c already then say...

```
# Now lets see its vector representation
bow4 = bow_transformer.transform([message4])
print(bow4)
print(bow4.shape)
```

```
  (0, 4068)     2
  (0, 4629)     1
  (0, 5261)     1
  (0, 6204)     1
  (0, 6222)     1
  (0, 7186)     1
  (0, 9554)     2
(1, 11425)
```

This means that there are seven unique words in message number 4 (after removing common stop words). Two of them appear twice, the rest only once. Let's go ahead and check and confirm which ones appear twice:

```
print(bow_transformer.get_feature_names_out()[4073])
print(bow_transformer.get_feature_names_out()[9570])
```

UIN

schedule

Now we can use .transform on our Bag-of-Words (bow) transformed object and transform the entire DataFrame of messages. Let's go ahead and check out how the bag-of-words counts for the entire SMS corpus is a large, sparse matrix:

```
messages_bow = bow_transformer.transform(messages['message'])

print('Shape of Sparse Matrix: ', messages_bow.shape)
print('Amount of Non-Zero occurences: ', messages_bow.nnz)
```

Shape of Sparse Matrix: (5572, 11425)

Amount of Non-Zero occurences: 50548

```
sparsity = (100.0 * messages_bow.nnz / (messages_bow.shape[0] * messages_bow.shape[1]))
print('sparsity: {}'.format(round(sparsity)))
```

sparsity:0

TF-IDF

After the counting, the **term weighting and normalization** can be done **with TF-IDF** (http://en.wikipedia.org/wiki/Tf%E2%80%93idf), **using scikit-learn's `TfidfTransformer`.**

What is TF-IDF? **TF-IDF** stands for **term frequency-inverse document frequency**, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate **how important a word is to a document in a collection or corpus**. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Typically, the **tf-idf weight is composed by 2 terms**: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency,** which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
    - TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
- **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
    - IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Example: Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

Let's go ahead and see how we can do this in SciKit Learn:

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf4 = tfidf_transformer.transform(bow4)
print(tfidf4)
```

(0, 4068) 0.4083258993338407
(0, 4629) 0.2661980190608719
(0, 5261) 0.2972995740586873
(0, 6204) 0.2995379972369742
(0, 6222) 0.31872168929491496
(0, 7186) 0.4389365653379858
(0, 9554) 0.5385626262927565

We'll go ahead and check what is the IDF of the word `"u"` and of word `"university"`?

```
print(tfidf_transformer.idf_[bow_transformer.vocabulary_['u']])
print(tfidf_transformer.idf_[bow_transformer.vocabulary_['university']])
```

3.2800524267409408
8.527076498901426

To transform the entire bag-of-words corpus into TF-IDF corpus at once:

```
messages_tfidf = tfidf_transformer.transform(messages_bow)
print(messages_tfidf.shape)
```

(5572, 11425)

There are many ways the data can be preprocessed and vectorized. These steps involve feature engineering and building a "pipeline". I encourage you to check out SciKit Learn's documentation on dealing with text data as well as the expansive collection of available papers and books on the general topic of NLP.

Training a Model

With messages represented as vectors, we can finally **train our spam/ham classifier.** Now we can actually use almost any sort of classification algorithms. For a variety of reasons (http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn-note07-2up.pdf), the **Naive Bayes classifier algorithm** is a good choice (scikit-learn) (http://en.wikipedia.org/wiki/Naive_Bayes_classifier)

```
from sklearn.naive_bayes import MultinomialNB
spam_detect_model = MultinomialNB().fit(messages_tfidf, messages['label'])

#Let's try classifying our single random message and checking how we do:
print('predicted:', spam_detect_model.predict(tfidf4)[0])
print('expected:', messages.label[3])
```

predicted: ham
expected:ham

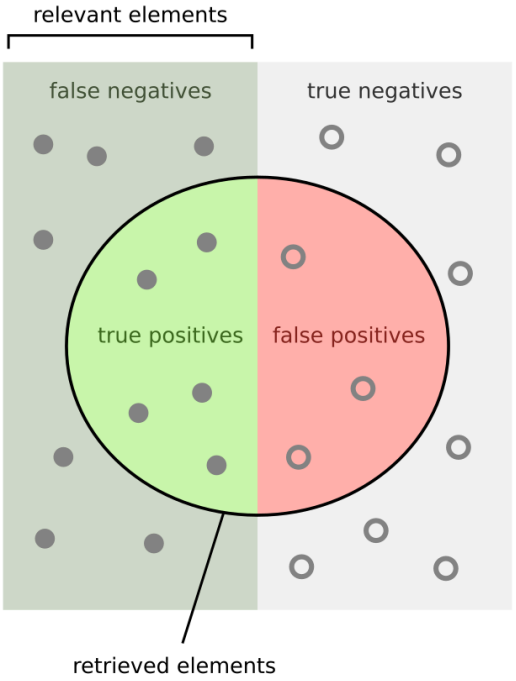
Fantastic! We've developed a model that can attempt to predict spam vs ham classification!

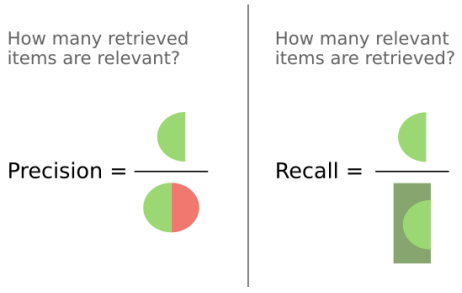
Model Evaluation

```
all_predictions = spam_detect_model.predict(messages_tfidf)
print(all_predictions)
```

```
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

We can use SciKit Learn's built-in classification report, which returns precision, recall, (https://en.wikipedia.org/wiki/Precision_and_recall) f1-score (https://en.wikipedia.org/wiki/F1_score), and a column for support (meaning how many cases supported that classification). Check out the links for more detailed info on each of these metrics and the figure below:






```
from sklearn.metrics import classification_report
print (classification_report(messages['label'], all_predictions))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      4825
        spam       1.00      0.85      0.92       747

    accuracy                           0.98      5572
   macro avg       0.99      0.92      0.95      5572
weighted avg       0.98      0.98      0.98      5572
```

There are quite a few possible metrics for evaluating model performance. Which one is the most important depends on the task and the business effects of decisions based off of the model. For example, the cost of mis-predicting "spam" as "ham" is probably much lower than mis-predicting "ham" as "spam".

In the above "evaluation", we evaluated accuracy on the same data we used for training. Y**ou should never actually evaluate on the same dataset you train on!** Such evaluation tells us nothing about the true predictive power of our model. If we simply remembered each example during training, the accuracy on training data would trivially be 100%, even though we wouldn't be able to classify any new messages. A proper way is to split the data into a training/test set, where the model only ever sees the training data during its model fitting and parameter tuning. The test data is never used in any way. This is then our final evaluation on test data is representative of true predictive performance.

Train Test Split

```
from sklearn.model_selection import train_test_split
msg_train, msg_test, label_train, label_test = \
train_test_split(messages['message'], messages['label'], test_size=0.2)

print(len(msg_train), len(msg_test), len(msg_train) + len(msg_test))
```

4457 1115 5572

The test size is 20% of the entire dataset (1115 messages out of total 5572), and the training is the rest (4457 out of 5572). Note the default split would have been 30/70.

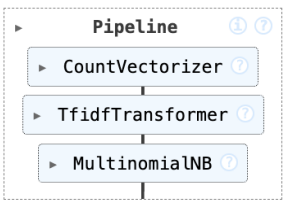
Creating a Data Pipeline

Let's run our model again and then predict off the test set. We'll use **SciKit Learn's pipeline** (http://scikit-learn.org/stable/modules/pipeline.html) capabilities to store a pipeline of workflow that will allow us to set up all the transformations that we'll do to the data for future use.

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_process)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])
```

Now we can directly pass message text data and the pipeline will do our pre-processing for us! We can treat it as a model/estimator API:

```
pipeline.fit(msg_train,label_train)
```



predictions = pipeline.predict(msg_test)
print(classification_report(predictions,label_test))

```
                precision    recall  f1-score   support

           ham       1.00      0.95      0.98       993
          spam       0.72      1.00      0.84       122

      accuracy                           0.96      1115
     macro avg       0.86      0.98      0.91      1115
  weighted avg       0.97      0.96      0.96      1115
```

Now we have a classification report for our model on a true testing set! There is a lot more to Natural Language Processing than what we've covered here, and its vast expanse of topic could fill up several college courses! I encourage you to check out the resources below for more information on NLP!

More Resources

Check out the links below for more info on Natural Language Processing:

- NLTK Book Online: (http://www.nltk.org/book/)
- Kaggle Walkthrough: (https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words)
- SciKit Learn's Tutorial: (http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html)

Natural Language Processing Project

In this NLP project you will be attempting to classify Yelp Reviews into 1 star or 5 star categories based off the text content in the reviews. This will be a simpler procedure than the lecture, since we will utilize the pipeline methods for more complex tasks. We will use the Yelp Review Data Set from Kaggle (https://www.kaggle.com/c/yelp-recsys-2013).

Each observation in this dataset is a review of a particular business by a particular user. The "stars" column is the number of stars (1 through 5) assigned by the reviewer to the business. (Higher stars is better.) In other words, it is the rating of the business by the person who wrote the review.

The "cool" column is the number of "cool" votes this review received from other Yelp users. All reviews start with 0 "cool" votes, and there is no limit to how many "cool" votes a review can receive. In other words, it is a rating of the review itself, not a rating of the business.

The "useful" and "funny" columns are similar to the "cool" column.

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

```
import seaborn as sns

df = pd.read_csv('yelp.csv')

df.head()
df.info()
df.describe()
```

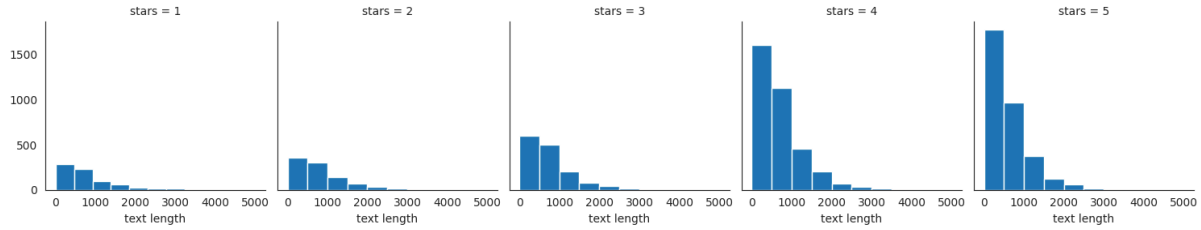
Create a new column called "text length" which is the number of words in the text column.

```
df['text length'] = df['text'].apply(len)
df.head()
```

	business_id	date	review_id	stars	text	type	user_id	cool	useful	funny	text length
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakf...	review	rLtl8ZkDX5vH5nAx9C3q5Q	2	5	0	889
1	ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some people give bad review...	review	0a2KyEL0d3Yb1V6aivbIuQ	0	0	0	1345
2	6oRAC4uyJCsJl1X0WZpVSA	2012-06-14	IESLBzqUCLdSzSqm0eCSxQ	4	love the gyro plate. Rice is so good and I als...	review	0hT2KtfLiobPvh6cDC8JQg	0	1	0	76
3	_1QQZuf4zZOyFCvXc0o6Vg	2010-05-27	G-WvGaISbqqaMHlNnByodA	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...	review	uZetl9T0NcROGOyFfughhg	1	2	0	419
4	6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	General Manager Scott Petello is a good egg!!!...	review	vYmM4KTsC8ZfQBg-j5MWkw	0	0	0	469

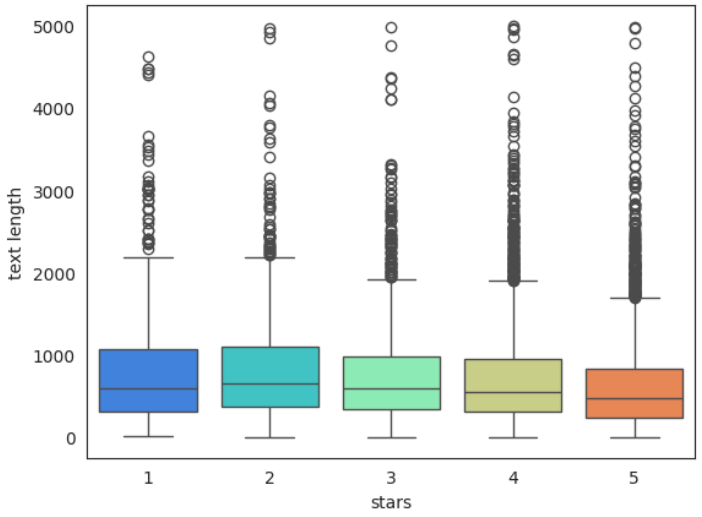
Use FacetGrid from the seaborn library to create a grid of 5 histograms of text length based off of the star ratings. Reference the seaborn documentation for hints on this

```
sns.set_style('white')
g = sns.FacetGrid(df, col="stars") # Just the Grid
g = g.map(plt.hist, "text length")
```



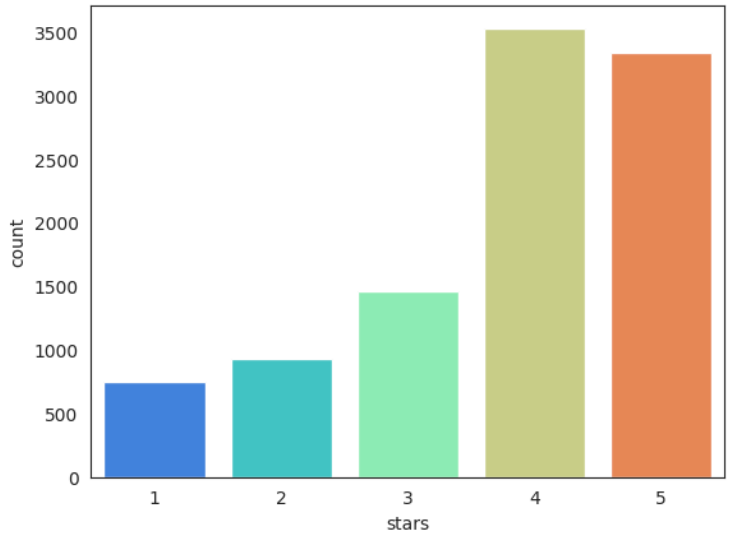
Create a boxplot of text length for each star category.

```
sns.boxplot(x="stars", y="text length", data=df,palette='rainbow')
```



Create a countplot of the number of occurrences for each type of star rating.

```
sns.countplot(x='stars',data=df, palette='rainbow')
```



** Use groupby to get the mean values of the numerical columns, you should be able to create this dataframe with the operation:**

```
# Seleccionar solo las columnas numéricas
numerical_df = df.select_dtypes(include='number')
pd.DataFrame(numerical_df.groupby('stars').mean())
```

	cool	useful	funny	text length
stars				
1	0.576769	1.604806	1.056075	826.515354
2	0.719525	1.563107	0.875944	842.256742
3	0.788501	1.306639	0.694730	758.498289
4	0.954623	1.395916	0.670448	712.923142
5	0.944261	1.381780	0.608631	624.999101

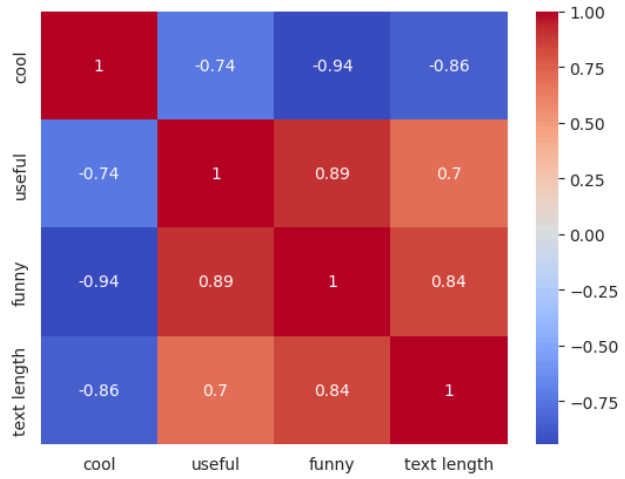
Use the corr() method on that groupby dataframe to produce this dataframe:

```
stars_media=pd.DataFrame(numerical_df.groupby('stars').mean())
correlation_matrix= stars_media.corr()
correlation_matrix
```

	cool	useful	funny	text length
cool	1.000000	-0.743329	-0.944939	-0.857664
useful	-0.743329	1.000000	0.894506	0.699881
funny	-0.944939	0.894506	1.000000	0.843461
text length	-0.857664	0.699881	0.843461	1.000000

Then use seaborn to create a heatmap based off that .corr() dataframe:

```
sns.heatmap(correlation_matrix,cmap='coolwarm',annot=True)
```



NLP Classification Task

Let's move on to the actual task. To make things a little easier, go ahead and only grab reviews that were either 1 star or 5 stars.

Create a dataframe called yelp_class that contains the columns of yelp dataframe but for only the 1 or 5 star reviews.

```
yelp_class=df[(df['stars']==1) | (df['stars'] == 5)]
```

** Create two objects X and y. X will be the 'text' column of yelp_class and y will be the 'stars' column of yelp_class. (Your features and target/labels)**

```
X=yelp_class['text']
y=yelp_class['stars']
```

Import CountVectorizer and create a CountVectorizer object.

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

** Use the fit_transform method on the CountVectorizer object and pass in X (the 'text' column). Save this result by overwriting X.**

```
X = cv.fit_transform(X)
```

Train Test Split: Let's split our data into training and testing data.** Use train_test_split to split up the data into X_train, X_test, y_train, y_test. Use test_size=0.3 and random_state=101 **

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

Training a Model: ** Import MultinomialNB and create an instance of the estimator and call is nb **

```
from sklearn.naive_bayes import MultinomialNB
nb= MultinomialNB()
```

Now fit nb using the training data.

```
nb.fit(X_train,y_train)
```

Predictions and Evaluations: **Use the predict method off of nb to predict labels from X_test.**

```
predictions = nb.predict(X_test)
```

** Create a confusion matrix and classification report using these predictions and y_test **

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[159  69]
 [ 22 976]]
              precision    recall  f1-score   support

           1       0.88      0.70      0.78       228
           5       0.93      0.98      0.96       998

    accuracy                           0.93      1226
   macro avg       0.91      0.84      0.87      1226
weighted avg       0.92      0.93      0.92      1226
```

Great! Let's see what happens if we try to include TF-IDF to this process using a pipeline.

Using Text Processing: ** Import TfidfTransformer from sklearn. **

```
from sklearn.feature_extraction.text import TfidfTransformer
```

** Import Pipeline from sklearn. **

```
from sklearn.pipeline import Pipeline
```

** Now create a pipeline with the following steps:CountVectorizer(), TfidfTransformer(),MultinomialNB()**

```
pipeline = Pipeline([
    ('bow', CountVectorizer()),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])
```

Using the Pipeline: Remember this pipeline has all your pre-process steps in it already, meaning we'll need to re-split the original data (Remember that we overwrote X as the CountVectorized version. What we need is just the text**
Train Test Split: **Redo the train test split on the yelp_class object.**

```
X = yelp_class['text']
y = yelp_class['stars']
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_state=101)
```

Now fit the pipeline to the training data. Remember you can't use the same training data as last time because that data has already been vectorized. We need to pass in just the text and labels

```
pipeline.fit(X_train,y_train)
```

Predictions and Evaluation: ** Now use the pipeline to predict from the X_test and create a classification report and confusion matrix. You should notice strange results.**

```
predictions = pipeline.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[  0 228]
 [  0 998]]
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       228
           5       0.81      1.00      0.90       998

    accuracy                           0.81      1226
   macro avg       0.41      0.50      0.45      1226
weighted avg       0.66      0.81      0.73      1226
```

Looks like Tf-Idf actually made things worse! That is it for this project. But there is still a lot more you can play with: Try going back and playing around with the pipeline steps and seeing if creating a custom analyzer like we did in the lecture helps (note: it probably won't). Or recreate the pipeline with just the CountVectorizer() and NaiveBayes. Does changing the ML model at the end to another classifier help at all?

Section 25: Neural Nets and Deep Learning

Introduction to Artificial Neural Networks (ANN)

05-ANN-Artificial-Neural-Networks

Perceptron Model

Para comenzar a comprender el aprendizaje profundo, desarrollaremos las abstracciones de nuestro modelo:

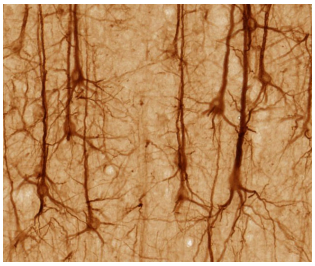
- Neurona Biológica Única (Single Biological Neuron)
- Perceptron
- Modelo de Perceptron Multicapa (Multi-layer Perceptron Model)

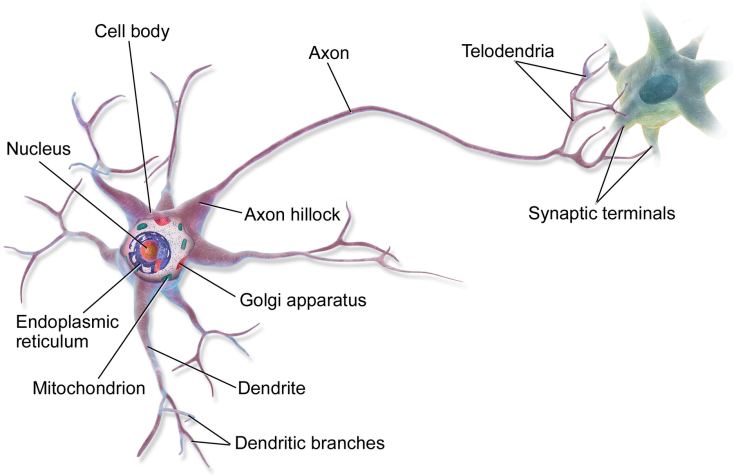
- Red Neuronal de Aprendizaje Profundo (Deep Learning Neural Network)

A medida que aprendamos sobre modelos más complejos, también introduciremos conceptos como:

- Funciones de activación (Activation Functions)
- Descenso de gradiente (Gradient Descent)
- Propagación hacia atrás (BackPropagation)

Si la idea detrás del aprendizaje profundo (deep learning) es hacer que las computadoras imiten artificialmente la inteligencia biológica natural, probablemente deberíamos desarrollar una comprensión general de cómo funcionan las neuronas biológicas. Aquí vemos algunas neuronas en estado real en la corteza cerebral:





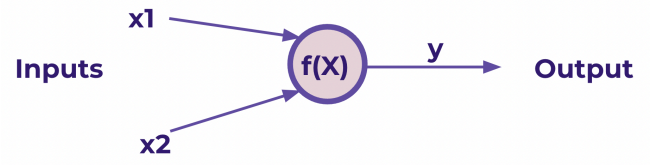
Básicamente vamos a intentar comprender cómo funcionan estas neuronas biológicas y ver si podemos desarrollar una abstracción simplificada de ellas. Podemos pensar a las dendritas como entradas que van al núcleo principal y luego al axon como algún tipo de salida. Básicamente pensaremos que las neuronas biológicamente aceptan algún tipo de señal de entrada que puede provenir de una variedad de fuentes desde las dendritas, luego algo sucede en el núcleo y se emite como una salida única en el axon. Eso puede conducir a otro núcleo y luego a otra neurona, etc.

¿Cómo tomamos este modelo de neurona biológica muy simplificado y lo convertimos en un modelo matemático? Aquí es donde surge la idea de Perceptron. Un Perceptron es una forma de red neuronal introducida en 1958 por Frank Rosenblatt. Sorprendentemente, ya entonces vio un enorme potencial: "... el perceptrón eventualmente podrá aprender, tomar decisiones y traducir idiomas".

Sin embargo, en 1969 Marvin Minsky y Seymour Papert publicaron su libro Perceptrons. Sugirieron que existían graves limitaciones en lo que podían hacer los perceptrones debido a la potencia computacional necesaria, lo cual marcó el comienzo del invierno de la IA, con poca financiación, para la IA y las redes neuronales, en la década de 1970. Afortunadamente ahora conocemos el asombroso poder de las redes neuronales, todas las cuales surgen del modelo de Perceptron simple.

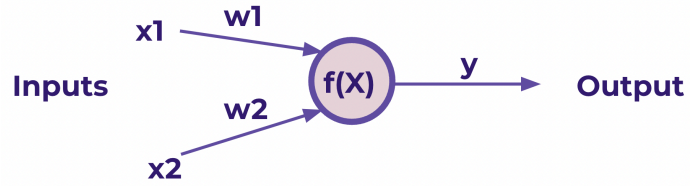
Vamos a reemplazar las dendritas, el núcleo y el axon con algunas matemáticas. En su lugar vamos a definir un conjunto de entradas que entran al pereceptron y obtenemos una salida única, por ejemplo **f(x)=x1+x2**

- If f(X) is just a sum, then y=x1+x2



En la realidad lo que buscamos es **poder ajustar algún parámetro para que el perceptron pueda aprender.** En el caso del ejemplo podemos agregar un peso ajustable para multiplicar por las X y ahora **f(x)=x1w1 + x2w2**

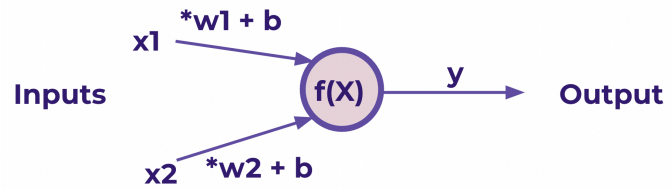
- Now **y = x1w1 + x2w2**



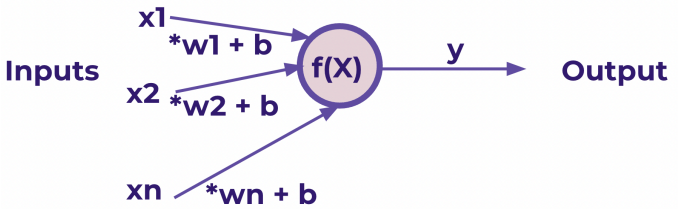
Aquí tomaremos cada entrada de X y le aplicaremos su propio peso. Luego podremos **ajustar ese peso según sea necesario para obtener el valor correcto/esperado de Y**.

Qué sucede si X=0? Los pesos no van a cambiar nada! Para solucionar ese problema podemos **agregar un término de sesgo b (bias term) a las entradas**, es decir agregamos a la neurona su propio sesgo particular para que las entradas terminen no solo multiplicándose por un peso sino que se les agregue un sesgo, ahora **f(x)=(x1w1+b) + (x2w2+b).** Los **pesos pueden ser positivos o negativos y los sesgos también (w y b).** La entrada multiplicada por su peso tiene que superar el valor de sesgo.

- y = (x1w1 + b) + (x2w2 + b)

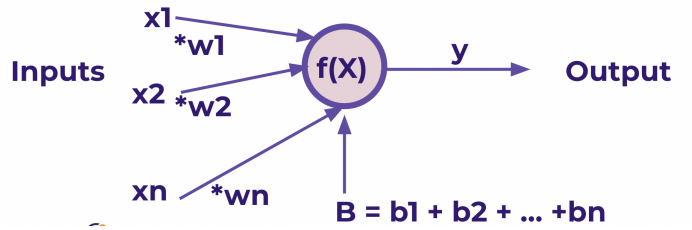


Podemos expandir esto a una generalización:



Y tomar los sesgos como una suma de ellos:

- Theoretically for any number of biases, there exists a bias that is the sum.



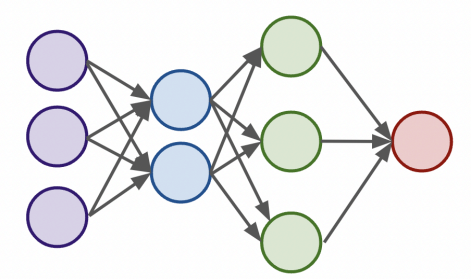
En definitiva, vimos cómo modelar una neurona biológica como un Perceptron simple. Matemáticamente nuestra generalización fue:

$$\hat{y} = \sum_{i=1}^{n} x_i w_i + b_i$$

Neural Networks

Un Perceptron simple no va a ser suficiente para aprender sistemas complicados, afortunadamente podemos expandir la idea de un Perceptron simple para crear un **modelo multicapa de Perceptron**, comúnmente conocido como **red neuronal artificial básica**.

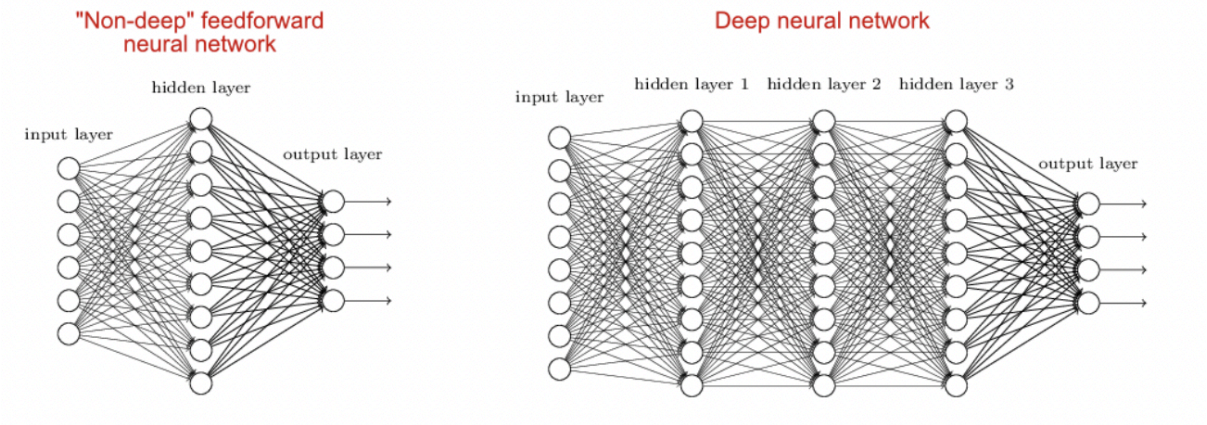
Para construir una red de Perceptrons podemos conectar capas de Perceptrons usando un modelo multicapa de Perceptron donde **la salida de un Perceptron se comunica directamente con la entrada de otro**; si que cada neurona está conectada a cada neurona de la siguiente capa se conoce como capa totalmente conectada (**fully connected layer**), toda la información va desde la capa de entrada hasta el final de la capa de salida. Básicamente esto **le permite a la red, como un todo, aprender sobre las interacciones y las relaciones entre las características**.



- **Input Layer** (capa de entrada): Es la primer capa (en la imagen la violeta), recibe directamente los datos sin procesar.
- **Output Layer** (capa de salida): Es la capa de salida (la roja) que puede estar compuesta por 1 neurona o más, asociado a lo que se intenta predecir.
- **Hidden Layers** (capas ocultas): Son las capas intermedias entre la/s de entrada y salida (en la imagen las azules y verdes). Las capas ocultas son difíciles de interpretar debido a su alta interconectividad y a su distancia de los valores de entrada o salida conocidos.

Una red neuronal se convierte en una **red neuronal profunda (deep neural network)** cuando contiene **2 o más capas ocultas (hidden layers).**

- **Ancho/width de una red:** Cuántas neuronas tiene
- **Profundidad/depth de una red:** Cuántas capas hay en total



Lo increíble del framework de la red neuronal es que se puede utilizar para aproximar cualquier función. Zhou Lu y más tarde Boris Hanin demostraron matemáticamente que las **redes neuronales pueden aproximarse a cualquier función continua convexa**. Para obtener más detalles sobre esto se puede consultar la página de Wikipedia sobre "Teorema de aproximación universal".

Activation Functions

Recordemos que en el modelo de Perceptron las entradas x tienen un peso w y un término de sesgo b, lo que significa que tenemos f(x) = x*w + b

- w implica cuánto peso o fuerza darle a la entrada entrante.
- podemos pensar en b como un valor de compensación, lo que hace que x*w tenga que alcanzar un cierto umbral antes de tener efecto.

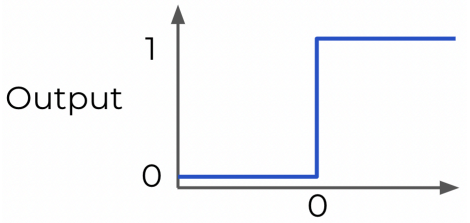
Por ejemplo si b= -10 ➞f(x) = x*w -10 ➞ los efectos de x*w no van a superar el sesgo hasta que su producto supere 10. Después de eso, el efecto se basa únicamente en el valor de w, de ahí el término “sesgo”. Se puede pensar al sesgo como un umbral que la neurona ha colocado para que la entrada por el peso comience a tener algún tipo de efecto mayoritario.

A continuación **queremos establecer límites para el valor de salida** de x*w + **b**. Podemos afirmar **z = x*w + b** y luego **pasar z a través de alguna función de activación f(z) para limitar su valor.** Se han realizado muchas investigaciones sobre las funciones de activación y su eficacia. Exploremos algunas funciones de activación comunes.

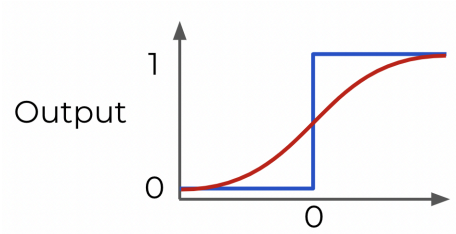
Si tuviéramos un problema de clasificación binaria, querríamos una salida de 0 o 1.

- **Step Function:** Las redes más simples se basan en una función escalonada básica que siempre genera 0 o 1 independientemente de los valores de X. Esto podría resultar útil para la clasificación (clase 0 o 1), sin embargo, esta es una función muy

“fuerte” ya que los pequeños cambios no se reflejan. Sólo hay un corte inmediato que se divide entre 0 y 1.

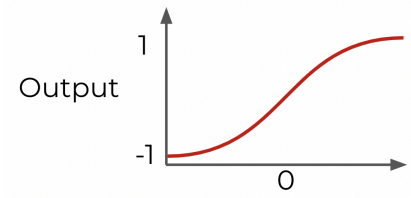


- **Sigmoid Function:** A diferencia de la anterior es más dinámica pero sigue teniendo el mismo límite inferior y superior (0 y 1). Esto significa que todavía funciona para la clasificación y será más sensible a pequeños cambios.



$$f(z) = \frac{1}{1 + e^{(-z)}}$$

- **Hyperbolic Tangent (tanh(z)):** Outputs between -1 and 1 instead of 0 to 1

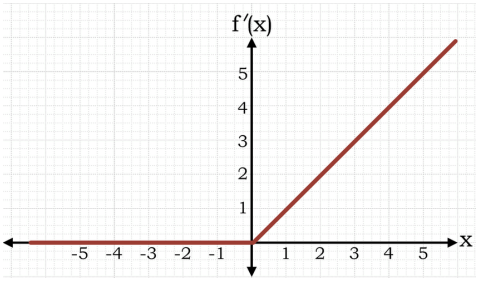


$$\cosh x = \frac{\mathbf{e}^{x} + \mathbf{e}^{-x}}{2}$$

$$\sinh x = \frac{\mathbf{e}^{x} - \mathbf{e}^{-x}}{2}$$

$$\tanh x = \frac{\sinh x}{\cosh x}$$

- **Rectified Linear Unit (ReLU)**: Es en realidad una función bastante simple **max(0,z)**. Se ha descubierto que tiene muy buen rendimiento, especialmente cuando se trata del problema del gradiente evanescente (**vanishing gradient**). A menudo utilizaremos ReLu de forma predeterminada debido a su buen rendimiento general.



Cambiar la función de activación utilizada puede resultar beneficioso según la tarea. Para obtener una lista completa de posibles funciones de activación, consulte: en.wikipedia.org/wiki/Activation_function

Multi-Class Activation Functions

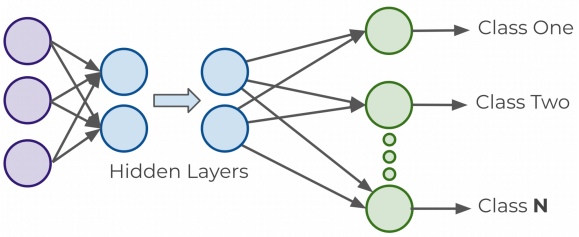
Todas esas funciones de activación tienen sentido para una única salida, ya sea una etiqueta continua o un intento de predecir una clasificación binaria (ya sea un 0 o un 1).
Pero ¿qué debemos hacer si tenemos una situación multiclase?

Hay **2 tipos principales de situaciones multiclase/multi-class**:

- **Non-Exclusive Classes** (clases no exclusivas): Cuando un dato puede tener múltiples clases/categorías asignadas. Por ejemplo, las fotos pueden tener varias etiquetas (playa, familia, vacaciones, etc.)
- **Mutual Exclusive Classes** (clases mutuamente excluyentes): Sólo una clase por dato. Por ejemplo, las fotos se pueden clasificar como en escala de grises (blanco y negro) o a todo color pero una foto no puede ser ambas cosas al mismo tiempo.

Cómo organizamos los datos que tienen múltiples clases? La forma más sencilla de organizar varias clases es simplemente tener **1 nodo de salida por clase**. Necesitaremos organizar **categorías para esta capa de salida**.

- Organizing for Multiple Classes



En la capa de salida no podemos simplemente tener categorías como “rojo”, “azul” y “verde” ya que la red neuronal tomará valores de x numéricos y luego puede aplicarles peso y sesgo. Lo que podemos hacer es utilizar **encoding one-hot**.

- Para clases mutuamente excluyentes:
  - Mutually Exclusive Classes

Data Point 1	RED
Data Point 2	GREEN
Data Point 3	BLUE
...	...
Data Point N	RED

→

	RED	GREEN	BLUE
Data Point 1	1	0	0
Data Point 2	0	1	0
Data Point 3	0	0	1
...	...	...	...
Data Point N	1	0	0

- Para clases no exclusivas:
  - Non-Exclusive Classes

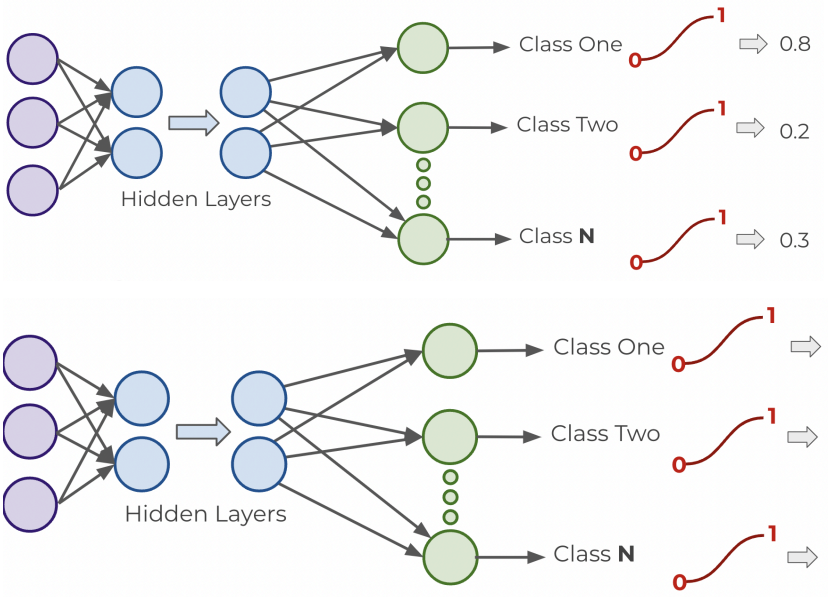
Data Point 1	A,B
Data Point 2	A
Data Point 3	C,B
...	...
Data Point N	B

→

	A	B	C
Data Point 1	1	1	0
Data Point 2	1	0	0
Data Point 3	0	1	1
...	...	...	...
Data Point N	0	1	0

Ahora que tenemos los datos correctamente organizados, solo nos falta elegir la función de activación de clasificación correcta que debería tener la última capa de salida.

- Clases no exclusivas
    - Sigmoid Function: Cada neurona generará un valor entre 0 y 1, indicando la probabilidad de que se le asigne esa clase. Podemos tener una neurona para cada clase. De nuevo, esto no es exclusivo por lo tanto, los puntos de datos pueden tener múltiples clases asignadas a ellos. Tenga en cuenta que esto permite que cada neurona genere salidas independientes de las otras clases, lo que permite que un único punto de datos introducido en la función tenga múltiples clases asignadas.



- Clases mutuamente excluyentes
    - Softmax function: La función Softmax calcula la distribución de probabilidades del evento entre K eventos diferentes, es decir que esta función calculará las probabilidades de cada clase objetivo sobre todas las clases objetivo posibles. El rango será de 0 a 1 y la suma de todas las probabilidades será igual a uno. El modelo devuelve las probabilidades de cada clase y **la clase objetivo elegida tendrá la mayor probabilidad.** Lo principal a tener en cuenta es que si usa softmax para problemas de clases múltiples obtendrá este tipo de resultado:
      [Rojo, Verde, Azul]
      [0,1 , 0,6 , 0,3]

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K$$

Cost Functions

Ahora entendemos que las redes neuronales toman entradas, las multiplican por pesos y les añaden sesgos. Luego, este resultado pasa a través de una función de activación que al final de todas las capas conduce a alguna salida. Esta **salida ŷ** es la **estimación del modelo de lo que predice** que será la etiqueta.

Entonces, después de que la red crea su predicción, ¿cómo la evaluamos? ¿cómo podemos actualizar los pesos y sesgos de la red? Necesitamos tomar los resultados estimados de la red y luego compararlos con los valores reales de la etiqueta. Tenga en cuenta que esto es utilizando el conjunto de datos de entrenamiento durante el ajuste/entrenamiento del modelo.

Entonces, para **comparar la salida de nuestras redes neuronales con el valor verdadero**, usaremos lo que se conoce como f**unción de costo / función perdida / función de error**, que basicamente mide qué tan lejos está del valor real en función de su predicción. Una advertencia importante es que **debe ser un promedio** para que podamos generar un valor único. Luego podemos realizar un **seguimiento de de nuestras pérdidas/costos durante el entrenamiento** para monitorear el rendimiento de la red. Por lo tanto, con suerte, durante cada época de entrenamiento, su **pérdida o costo se reducirá hasta que converja en algún valor de costo mínim**o.

Usaremos las siguientes variables:

- **y** para representar el **valor verdadero**
- **a** para representar la **predicción de la neurona**

En términos de ponderaciones y sesgos:

- **w*x + b = z**
- Pasar z a la función de activación **σ(z) = a**

Entonces a representa el resultado final de una neurona, que tiene en cuenta la función de activación, y eso también tiene en cuenta Z, que a su vez tiene en cuenta los pesos y los sesgos.

Una **función de costos muy común es la función de costos cuadrática**:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

- Simplemente calculamos la diferencia entre los valores reales y(x) frente a nuestros valores predichos a(x) y las elevamos al cuadrado. Elevar esto al cuadrado hace 2 cosas útiles para nosotros:
    - **Mantiene todo positivo:** Al elevarlo al cuadrado se vuelve positivo, lo que es bueno para la **medición absoluta de error**. Si no estuviéramos cuadrando esto y tuviéramos cosas negativas y positivas cuando lo promediamos podrían rondar alrededor de 0, lo que en realidad no es una indicación real del valor absoluto o unidades absolutas de qué tan lejos está.
    - **Castiga los errores grandes:** A veces vas a tener datos en los que vas a estar realmente equivocado y si cuadras ese error va a crecer exponencialmente en términos de tu costo. Tal vez tengas un error de $10 en cualquier unidad que intentes medir, pero tu costo lo reportará en unidades al cuadrado por lo que dirá que estás fuera de 100 en lugar de solo 10. Así, vas a castigar realmente a tu red por estar realmente desconectada de ciertos puntos, lo cual es bueno

porque no quieres que tu red no pueda predecir bien incluso en unos pocos puntos donde te da un gran error. **Preferirías sufrir un poco en todos los demás puntos y no estar totalmente equivocado en esos pocos tipos de casos extremos**. Eso realmente ayuda a castigar los errores grandes.

- Tenga en cuenta que una notación de L solo significa que esa es la función de activación. Salida de la capa L donde L es su última capa, lo que significa que la capa anterior a la de la red es L menos 1, la capa anterior es L menos 2 y así sucesivamente. Más adelante veremos por qué es más conveniente marcar L como su última capa y luego trabajar hacia atrás desde allí en lugar de comenzar desde el principio.
- Nota: La notación que se muestra aquí corresponde a entradas y salidas vectoriales, ya que trataremos con un lote de puntos de entrenamiento y predicciones.
- La idea principal es que tenemos C como función de costo y estamos haciendo algún tipo de promedio, n es el número de puntos allí.

Podemos pensar en la función de costos como una función de 4 cosas principales:

1. Será una función de W, que son los pesos de nuestra red neuronal.
2. B, que son todos los sesgos de nuestra red neuronal.
3. S^r, que es la entrada de una única muestra de entrenamiento,
4. E^r que es la salida deseada de esa muestra de entrenamiento.

Y eso tiene mucho sentido porque el costo depende de cuáles son los pesos y sesgos actuales. También depende de lo que se haya pasado como ejemplo de entrenamiento real, y luego también depende de con qué lo estás comparando, que es E^r.

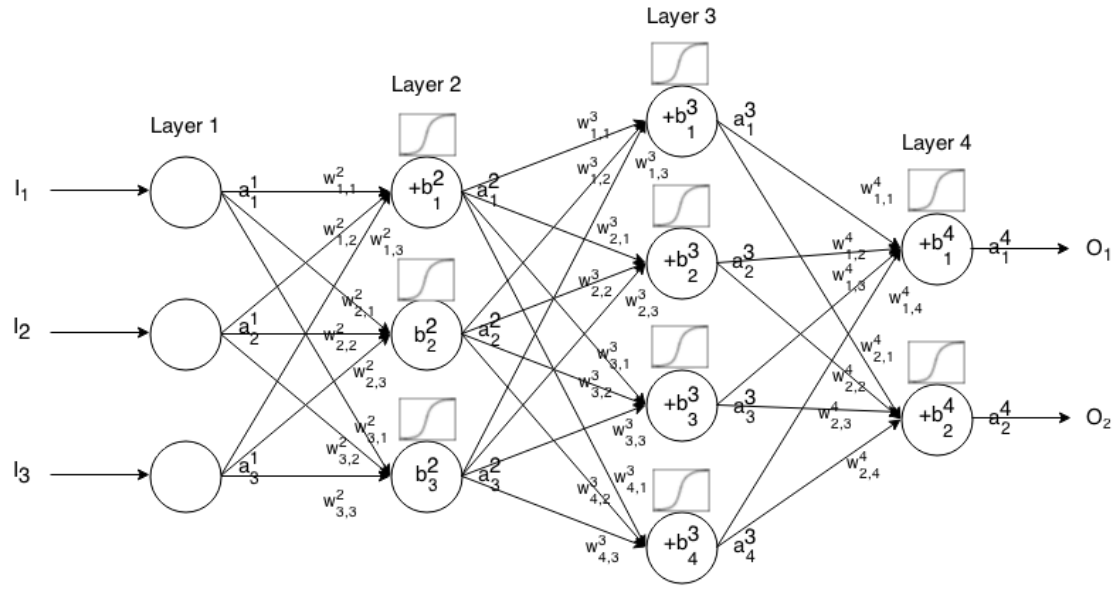
$$C(W, B, S^r, E^r)$$

Quizás se pregunten: ¿no acaban de decir que la función de costos es una función de W y B? ¿Dónde están W y B en la función cuadrática que vimos antes?

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Están codificados dentro de a(x) porque recuerde que a(x) contiene información sobre pesos y sesgos, ya que a(x) es z pasado a través de a la función de activación (σ(z) = a) donde z contiene información sobre W y B (w*x + b = z). Esto significa que si tenemos una red enorme, podemos esperar que la función de costo real sea realmente bastante compleja con un vector o tensor de pesos enorme y otro tensor de sesgos enorme.

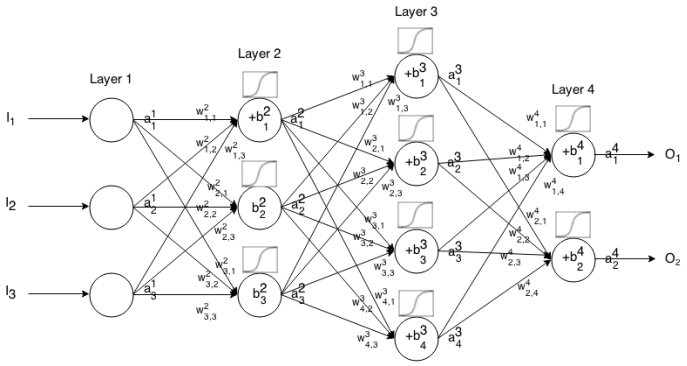
Por ejemplo, si tomáramos una red pequeña y comenzáramos a etiquetar cada peso, cada sesgo y cada salida como una especie de salida de la función de activación, que es a, pueden ver todos los parámetros etiquetados aquí. Podemos ver que puede volverse realmente complicado muy rápido, y eso que esta es una red realmente pequeña de solo 4 capas, con 2 capas ocultas que ni siquiera son tan grandes. Pero como puedes ver aquí, si empezamos a anotar todo, puede volverse bastante complejo. Ya tienes matrices de pesos y sesgos bastante grandes.



Entonces, ¿cómo calculamos esto? ¿**cómo calculamos esa función de costos y luego descubrimos cómo minimizarla**?

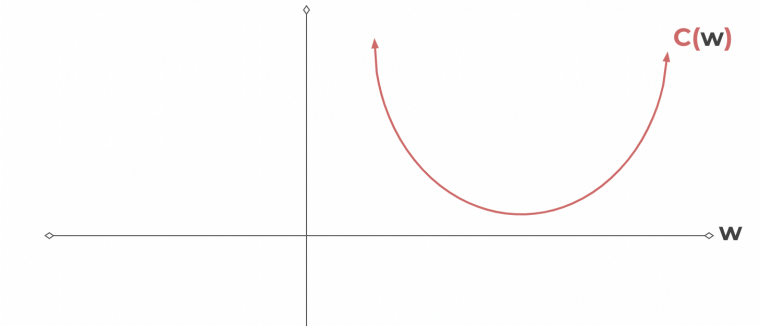
En un caso real esto significaría que **tenemos alguna función de costo C que depende de muchos pesos**. Esa función de costo dependerá del peso del primer input, luego del peso del segundo, del peso del tercero, hasta llegar a w de n: **C(w1,w2,w3,....wn).**

Lo que debemos hacer es **determinar qué pesos particulares nos llevan al costo más bajo** porque queremos volver aquí y descubrir todos estos pesos, ¿cómo los cambio para minimizar mi función de costos al final?

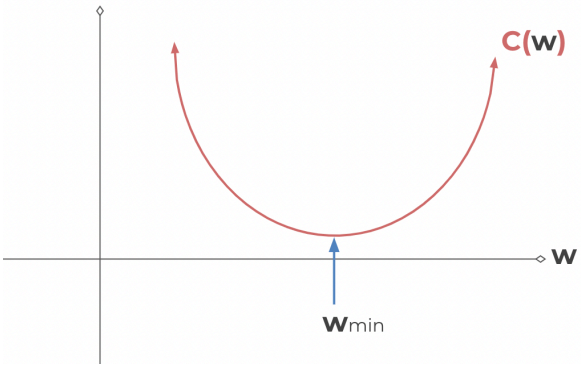


Para simplificar imaginemos que estamos tratando con una red muy simple que solo tiene un peso, esencialmente solo un año después. Entonces, lo que queremos hacer es minimizar nuestra pérdida o costo (error general), lo que nuevamente significa que debemos determinar qué valor de w usamos para dar como resultado el **mínimo de C(w)**.

Esta será nuestra función de costos simple, una red simple que solo contiene un peso:



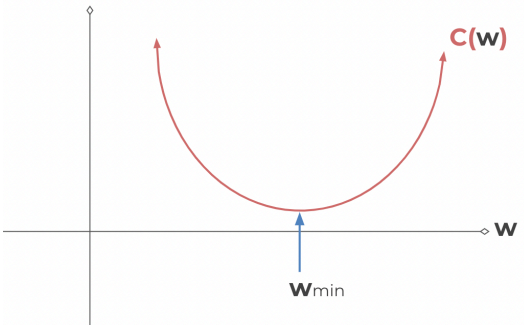
Lo que queremos hacer es averiguar qué valor de w minimiza esta función de costos. Y si bien este es un ejemplo muy simple, probablemente puedas decir que para minimizar la función de costos el mínimo probablemente caiga en algún lugar donde está la flecha:



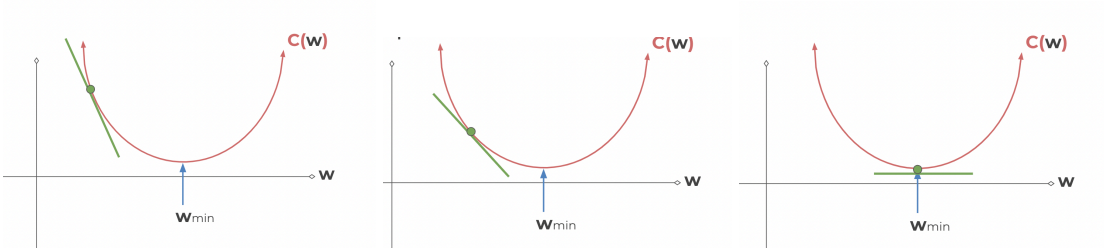
Ese es el peso que minimiza la función de costos, lo que significa que probablemente sea el peso que queremos en la neurona actual o esa entrada a la neurona porque eso reduce el costo al mínimo. Ahora, lo que podríamos hacer es simplemente tomar la derivada de esta función de costos y resolverla para obtener 0. Pero recuerden, **nuestra función de costos** reales será súper compleja y no será unidimensional sino que **tendrá tantas dimensiones como w**, y eso ni siquiera es algo que se pueda trazar por lo que **no podremos tomar esa derivada**.

Entonces, lo que necesitamos hacer es un proceso estocástico y podemos usar el **descenso de gradiente** para resolver este tipo de problema.

Volvamos a la versión simplificada de nuestra red donde solo tenemos un peso y veamos cómo funcionaría el descenso de gradiente. Luego podemos ampliarlo fácilmente a ejemplos más complejos.

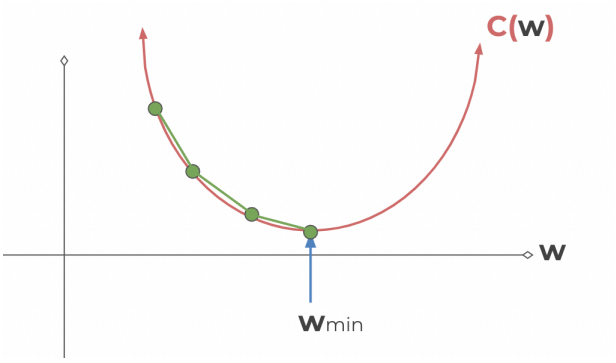


Lo que haremos, nuevamente, es buscar el valor de w que minimice la función de costos. Entonces, **comenzamos en un punto de la función de costo, calculamos la pendiente y luego nos movemos en dirección descendente de la pendiente**. A continuación **seguimos repitiendo este proceso hasta que finalmente converge a 0, lo que indica un mínimo.**

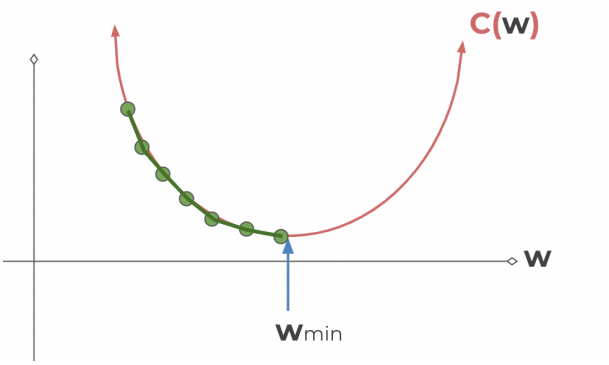


Podríamos haber cambiado el tamaño de nuestro paso para encontrar el siguiente punto:

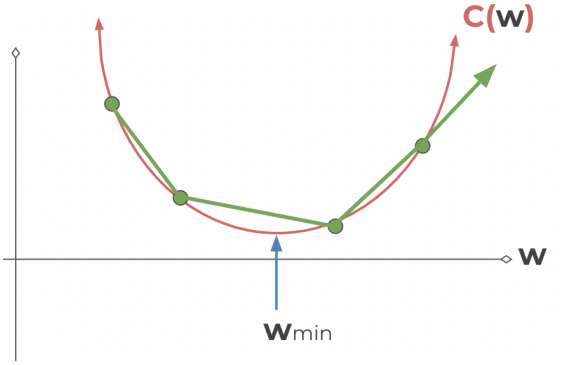
- Aquí tomamos pasos iguales



- Si se toman pasos más pequeños se va a tardar más tiempo en encontrar el mínimo.



- Por otro lado, si se dan pasos más grandes será más rápido pero se corre el riesgo de sobrepasar el mínimo con lo cual no termine convergiendo.

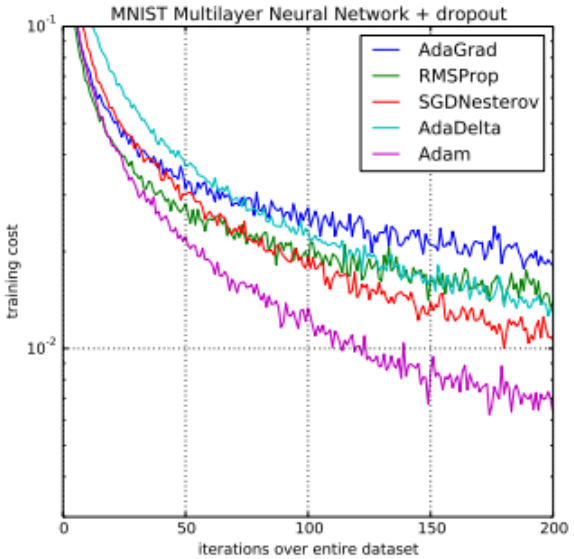


El tamaño del paso se conoce como **tasa de aprendizaje**. Si alguna vez ves en las redes neuronales que están editando la tasa de aprendizaje significa que están editando **qué tan rápido intentarán encontrar ese valor de peso mínimo**. Y funciona igual con los desvíos. Entonces, lo que estamos haciendo es buscar encontrar los pesos mínimos, en realidad los valores de los pesos y sesgos que minimizan la función de costos.

En los ejemplos anteriores la tasa de aprendizaje fue constante, es decir que el tamaño de cada paso fue igual. Ahora **podemos** ser un poco más inteligentes y **adaptar el tamaño de nuestros pasos a medida que avanzamos.** Dado que se comienza en una posición aleatoria dentro del espacio n-dimensional de posibles pesos y sesgos, se puede **comenzar con pasos más grandes e ir reduciendo cada vez más el tamaño del paso a medida que el gradiente o la pendiente se acerca a 0**. Esto se conoce como **descenso de gradiente adaptativo.** Dependiendo del gradiente que obtengas, adaptarás el tamaño de tu paso.

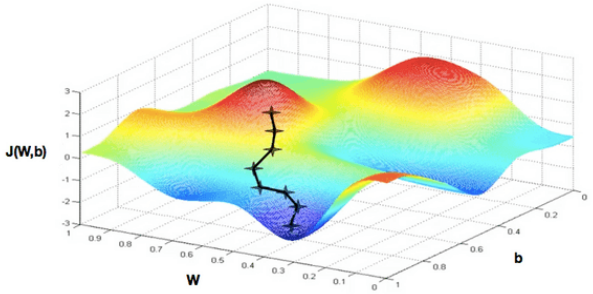
En 2015, Kingma y Ba publicaron su artículo llamado "Adam: Un método para la optimización estocástica." Adam es una forma mucho más eficaz de buscar estos mínimos; a veces indicamos a Adam como nuestro optimizador, nos referimos es a esta forma optimizada de realizar el descenso de gradiente donde tenemos este tamaño de paso adaptativo. Comenzamos en grande y luego, dependiendo de dónde estemos, tal vez nos vayamos haciendo cada vez más pequeños. Entonces obtienes lo mejor de ambos mundos, puedes utilizar los tamaños de paso más grandes y acelerar un poco la búsqueda de ese mínimo, pero luego, a medida que te acercas más y más a él y no quieres excederte, puedes optar por pasos más pequeños.

Puedes comparar Adam con otros algoritmos de descenso de gradiente, aquí se muestra el costo de capacitación versus las iteraciones en todo el conjunto de datos:



Se puede ver que Adam aquí está superando a estos otros algoritmos adaptativos de descenso de gradiente por lo que usaremos Adam; es bastante común usarlo para redes neuronales.

Lo que estamos haciendo es optimizar la forma en que encontramos este mínimo. Sin embargo, estábamos mostrando esa ilustración del descenso de gradiente en una sola w, en una especie de w unidimensional. Lo que realmente estamos haciendo es calcular el descenso de gradiente en un espacio de n dimensiones para todos nuestros pesos. Aquí puede ver cómo calcular el descenso de gradiente en un plano bidimensional, incluidos los pesos y el sesgo.



De manera realista, ni siquiera se podrá ilustrar el espacio de n-dimensiones porque tendrá n-dimensiones de decenas o cientos de pesos y sesgos, por lo que lo simplificamos a una especie de peso único.

Ahora puedes entender lo que hace el descenso de gradiente y llevarlo a lo que realmente estamos haciendo, o lo que la computadora hará por nosotros, que es hacer el mismo tipo de cálculos pero en un espacio de n-dimensiones que no podemos ilustrar de manera realista.

Entonces, cuando se trata de estos **vectores de n-dimensiones**, también conocidos como **tensores**, la notación cambia de derivada a **gradiente**. Eso significa que calculamos el gradiente de la función de costos con respecto a todos estos pesos: ∇**C(w1,w2,...wn)**

Para los **problemas de clasificación**, en lugar de usar la función de costo cuadrático lo que terminamos usando a menudo es la **función de pérdida de entropía cruzada (cross entropy loss function)**. Básicamente lo que hace es suponer que su modelo predice una distribución de probabilidad p(y=i) para cada clase i=1,2,...,C. La forma en que esta fórmula funciona es:

- Para una **clasificación binaria** que tiene solo 2 clases:

$$-(y\log(p)+(1-y)\log(1-p))$$

- Para un **número M de clases >2**:

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Entonces, cuando realizamos una clasificación, especialmente una clasificación de clases múltiples que es mayor que solo una clasificación binaria, recurriremos a la entropía cruzada como nuestra función de costos.

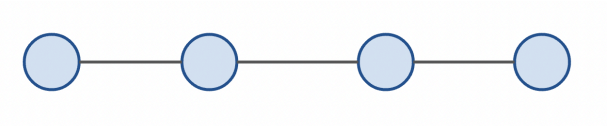
BackPropagation

Vamos a comenzar construyendo la idea detrás de la **retropropagación (backpropagation)** y luego nos sumergiremos en el cálculo y la notación.

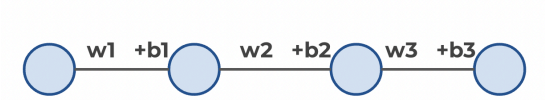
La retropropagación es probablemente la parte más difícil de todo el proceso teórico de aprendizaje profundo debido al cálculo y a la notación involucrada, especialmente cuando se empieza a tratar con una matriz de pesos y otra matriz de sesgos. Sin embargo, si se entiende la idea básica de que simplemente **se retrocede a través de una red para actualizar los pesos y sesgos** es suficiente para continuar con el resto del curso. Si no entiendes la parte de cálculo de esta conferencia, no te preocupes demasiado porque no vamos a necesitar calcular los gradientes nosotros mismos sino que el código hará eso.

Queremos saber **cómo cambia la función de costos con respecto a los pesos** en la red, así **podemos actualizar los pesos** para **minimizar la función de costos**.

Comencemos con una red muy simple para comprender la retropropagación donde cada capa sólo tiene una neurona, luego podremos expandir esta idea a las redes con múltiples neuronas por capa.



Como ya sabemos, cada entrada recibe un peso y un sesgo.



Entonces hay un peso entrante y luego cada nodo, o sea cada neurona, tiene su propio sesgo. Así obtenemos la fórmula: peso 1 + sesgo 1, peso 2 + sesgo 2, peso 3 + sesgo 3, y así sucesivamente. Esto significa que tenemos algún tipo de función de costos que depende de esos pesos y sesgos: **C(w1,b1,w2,b2,w3,b3)** y ya hemos visto cómo este proceso se propaga.

Comencemos por el final para aprender sobre la retropropagación. Recordemos que en nuestra notación **la última capa se llama L**, así:

Concentrémonos en las 2 últimas capas de nuestra red ya que la retropropagación comienza al final de la red, en la capa L. Hasta ahora hemos estado definiendo **z=wx+b**, siendo que **x en realidad solo es válido en la primera capa** porque representa las entradas de características sin procesar. A medida que se avanza de neurona en neurona en las capas, **x técnicamente se convierte en la salida de la neurona anterior, que se define como a** ya que después de aplicar una función de activación a z lo etiquetamos como a, **a = σ(z)**.

A medida que se avanza más y más hacia estas capas, z en realidad sería z=wXa+b, porque x técnicamente solo es válido en la primera capa como entrada de característica sin procesar.

Una vez que pasas eso a otra neurona, técnicamente ya no estás tratando con las características crudas sino que estás lidiando con la salida de la capa neuronal anterior, que se expresa mejor como σ(z) o cualquier función de activación que elijas.

¿Qué significa eso realmente? ¿Cuándo lo tomamos en cuenta para esa última capa? Bueno, eso significa que z en la última capa va a ser igual a los pesos en la última capa por a de L-1, más los sesgos en esa última capa.

$$z^L = w^L a^{L-1} + b^L$$

¿Qué es a^{L-1}? Es simplemente la salida de la neurona de esa capa anterior.

Además, la salida en la última capa es igual a sigmoide, o función de activación, de z de L:

$$a^L = \sigma(z^L)$$

Observe cómo z^L **se define por los pesos y sesgos en esa capa L, y luego se define por la salida de la neurona anterior.**

En definitiva, eso significa que la función de costos va a ser igual a a^L-y, donde **y es la salida real**, todo eso al cuadrado.

$$C_0 = (a^L - y)^2$$

Lo que en realidad queremos entender es **qué tan sensible es la función de costos a los cambios en w**. Aquí entran en juego las **derivadas parciales**, ya que queremos descubrir la **relación entre esa función de costos final y los pesos, en este caso en la capa L**. Entonces, vamos a tomar la derivada parcial de esa función de costos con respecto a los pesos en la capa L:

$$\frac{\partial C_0}{\partial w^L}$$

La regla de la cadena permite tomar la derivada de una función dentro de otra función. Si tomamos las fórmulas que acabamos de ver y les aplicamos la regla de la cadena para resolver su derivada parcial, terminamos calculando:

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L}$$

Recordemos que **la función de costos no es sólo función de los pesos, sino también de los sesgos**. Queremos ser capaces de entender la **relación de la función de costos no sólo cambiando con los pesos, sino que también con el sesgo a lo largo de la red**. Para esto, podemos **calcular la misma derivada parcial de la función de costos con respecto a esos términos de sesgo** de la misma manera (simplemente cambiamos el peso por el sesgo):

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial z^L}{\partial b^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L}$$

La idea principal es que **podemos usar el gradiente para volver a través de la red y ajustar nuestros pesos y sesgos para minimizar la salida del vector de error en la última capa de salida**. Recordemos que el gradiente es la derivada cuando se trata de n-dimensiones.

Usando notación de cálculo podemos **expandir esta idea a las redes con múltiples neuronas por capa.**

Recordemos que el **producto Hadamard** permite realizar multiplicaciones elemento por elemento cuando las matrices son del mismo tamaño, su notación se parece al símbolo del hidrógeno:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Las **matrices de pesos y sesgos tendrán ambas el mismo tamaño**.

Entonces, dada esta notación y la retropropagación, solo necesitamos algunos pasos principales para entrenar redes neuronales. Nuevamente, no es necesario que comprenda completamente estos intrincados detalles sobre el cálculo o la notación para continuar con este curso.

1. Usando la entrada **x** (las características originales) configuramos la función de activación a para la capa de entrada.
    a. **z = w x +b**
    b. **a = σ(z)**

    El resultado, **a**, luego alimenta a la siguiente capa y así a continuación.
2. La siguiente capa toma a, lo que significa que z va a ser z=wXa+b de la capa anterior. Entonces pasas a la siguiente capa, y luego tomas la salida de esa a, la pegas en z, y así sucesivamente. En definitiva para cada capa hay que calcular:
    a. $z^L = w^L a^{L-1} + b^L$
    b. $a^L = \sigma(z^L)$
3. En el tercer paso vamos a calcular nuestro vector de error:

    $$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

    a. El primer término ∇_a**C** lo que está haciendo es **expresar la tasa de cambio de la función de costos con respecto a las activaciones de salida**. En el caso de la función de costos cuadrática, es lo mismo que decir la activación de la última capa de salida menos y (y es el verdadero valor). Entonces, ∇_a**C=(**a^L**−y)**
    b. En definitiva lo que queremos hacer es calcular este vector de error y propagarlo hacia atrás, calculando el error hacia atrás a través de cada capa,

de manera en que podamos ajustar los pesos y sesgos por ese error. Entonces, reemplazando ese primer término obtenemos:

$$\delta^L=(a^L-y)\odot\sigma'(z^L)$$

4. Queremos propagar el error hacia atrás: Obtenemos una fórmula de vector de error generalizado y para cada capa empezando por la última L, luego pasando a L-1, L-2, etc., hasta el final, calculamos la fórmula del vector de error generalizado, el término de error generalizado. Usamos la L minúscula (l porque en teoría la L mayúscula es solo para la última capa.

$$\delta^l=(w^{l+1})^T\delta^{l+1}\odot\sigma'(z^l)$$

   a. Ese término de error (l) va a ser igual a la matriz de pesos l+1 transpuesta (por eso la T) multuplicada por el término de error también en la siguiente capa. Luego tomamos el producto Hadamard, con la z de l pasada a la función de activación. De nuevo, todo lo que estamos haciendo es propagar el error con el error generalizado para cualquier capa, L minúscula.
   b. Cuando aplicamos $(w^{l+1})^T$ podemos pensar intuitivamente en esto como mover el error hacia atrás a través de la red, dándonos algún tipo de medida del error en la salida de esa l-ésima capa.
   c. Cuando tomamos el producto de Hadamard de eso multiplicado por el z, en esa capa l, que pasa por la función de activación. Lo que hace esto es mover el error hacia atrás a través de la función de activación en la capa l, dándonos el error en l (δ^l) en la entrada ponderada a la capa l. Nuevamente es un término generalizado, por eso usamos la l en minúscula.
   d. Ahora podemos entender que el gradiente de la función de costos viene dada por estas 2 fórmulas que calculamos para cada capa L-1, L-2, etc:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \frac{\partial C}{\partial b^l_j} = \delta^l_j$$

   e. Entonces, para cada capa L-1, L-2, y así sucesivamente, todo lo que realmente estamos haciendo es computar esa derivada parcial de la función de costos con respecto a los pesos y los sesgos allí; j y k son sólo una notación para las propias neuronas. Esto luego nos permite ajustar los pesos y sesgos para ayudar a minimizar esa función de costos.

Enlaces externos:

- http://neuralnetworksanddeeplearning.com/chap2.html
- https://www.youtube.com/watch?v=llg3gGewQ5U

TensorFlow vs Keras

TensorFlow es una biblioteca de aprendizaje profundo de código abierto desarrollada por Google; TF (TensorFlow) 2.0 se lanzó oficialmente a finales de 2019. TensorFlow tiene un gran

ecosistema de componentes relacionados, que incluyen bibliotecas como Tensorboard, API de implementación y producción, y soporte para varios lenguajes de programación.

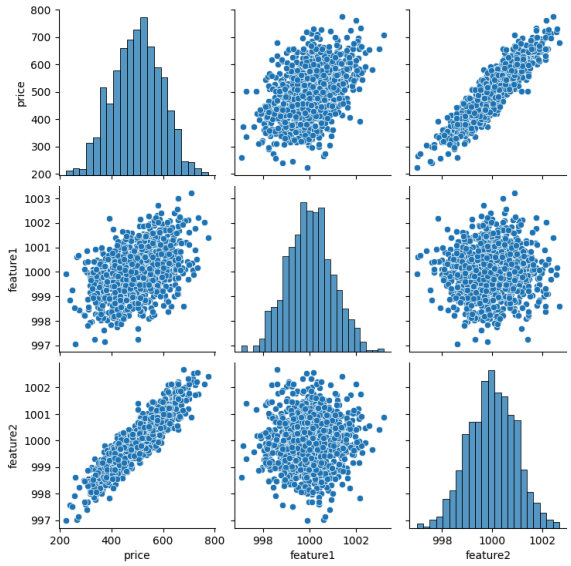
Keras es una biblioteca de Python de alto nivel que puede utilizar una variedad de bibliotecas de aprendizaje profundo, como TensorFlow, CNTK o Theano.

TensorFlow 1.x tenía un complejo sistema de clases de Python para construir modelos y, debido a la enorme popularidad de Keras, cuando se lanzó TF 2.0, TF adoptó Keras como la API oficial para TF. Si bien Keras sigue siendo una biblioteca separada de Tensorflow, ahora también se puede importar oficialmente a través de TF, por lo que no es necesario instalarlo adicionalmente. La API de Keras es fácil de usar y crea modelos simplemente agregando capas una encima de otra mediante llamadas simples.

TF Syntax Basics - Part One - Preparing the Data

Vamos a trabajar en github en la carpeta **TensorFlow_FILES → ANNs → 00-Keras-Syntax-Basics**

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

df = pd.read_csv('../DATA/fake_reg.csv')
df.head() #Es un problema de regresión donde tenemos 2 features y queremos predecir el precio

##Explore the data
sns.pairplot(df)  #la feature2 parece tener una correlación alta con el precio
```



##Test/Train Split

```
from sklearn.model_selection import train_test_split

# Convert Pandas to Numpy for Keras
# Features que vamos a usar
X = df[['feature1','feature2']].values
# Label q vamos a predecir
y = df['price'].values
# Split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)

X_train.shape
        (700,2)
X_test.shape
        (300,2)
y_train.shape
        (700,)
y_test.shape
        (300,)
```

##Normalizing / Scaling the data
We scale the feature data.
Why we don't need to scale the label?
https://stats.stackexchange.com/questions/111467/is-it-necessary-to-scale-the-target-value-in-addition-to-scaling-features-for-re

```
from sklearn.preprocessing import MinMaxScaler
help(MinMaxScaler)
scaler = MinMaxScaler()
# Notice to prevent data leakage from the test set, we only fit our scaler to the training set
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

TF Syntax Basics - Part Two - Creating and Training the Model

TensorFlow 2.0 Syntax
Import Options: There are several ways you can import Keras from Tensorflow (this is hugely a personal style choice, please use any import methods you prefer). We will use the method shown in the official TF documentation.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
help(Sequential)
```

##Creating a Model
There are two ways to create models through the TF 2 Keras API, either pass in a list of layers all at once, or add them one by one. Let's show both methods (its up to you to choose which method you prefer).

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

#Model - as a list of layers
model = Sequential([
   Dense(units=2),
   Dense(units=2),
   Dense(units=2)
])

#Model - adding in layers one by one
model = Sequential() #creamos el modelo vacío y le vamos agregando las capas
model.add(Dense(2))
model.add(Dense(2))
model.add(Dense(2))

#Let's go ahead and build a simple model and then compile it by defining our solver
model = Sequential()
model.add(Dense(4,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(4,activation='relu'))
# Final output node for prediction
model.add(Dense(1))  #1 porque queremos predecir el precio solamente
model.compile(optimizer='rmsprop',loss='mse')
```

###**Choosing an optimizer and loss**
Keep in mind what kind of problem you are trying to solve:

- For a multi-class classification problem: model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
- For a binary classification problem: model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
- For a mean squared error regression problem: model.compile(optimizer='rmsprop', loss='mse')

##Training
Below are some common definitions that are necessary to know and understand to correctly utilize Keras:

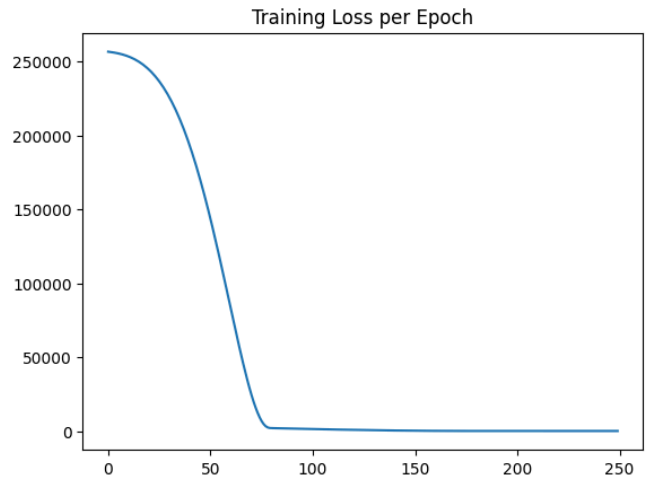
- Sample: one element of a dataset.
    - Example: one image is a sample in a convolutional network
    - Example: one audio file is a sample for a speech recognition model

- Batch: Un conjunto de N muestras. Las muestras en un batch se procesan de manera independiente, en paralelo. Si se está entrenando, un batch resulta en solo una actualización del modelo. Un batch generalmente aproxima mejor la distribución de los datos de entrada en comparación con una sola entrada. Cuanto más grande es el batch, mejor es la aproximación; sin embargo, también es cierto que el batch tomará más tiempo en procesarse y aún así resultará en una sola actualización. Para inferencia (evaluar/predicción), se recomienda elegir un tamaño de batch tan grande como sea posible sin que se agote la memoria (ya que los batches más grandes generalmente resultan en una evaluación/predicción más rápida).
- Epoch (época): Un límite arbitrario, generalmente definido como "un recorrido completo por todo el conjunto de datos", que se utiliza para separar el entrenamiento en fases distintas, lo cual es útil para el registro y la evaluación periódica.
- Cuando se usa validation_data o validation_split con el método fit de los modelos de Keras, la evaluación se ejecutará al final de cada epoch.
- Dentro de Keras, existe la posibilidad de añadir callbacks diseñados específicamente para ejecutarse al final de un epoch. Ejemplos de estos son cambios en la tasa de aprendizaje y el guardado de puntos de control del modelo (checkpointing).

```
model.fit(X_train,y_train,epochs=250)
```

Evaluation
Let's evaluate our performance on our training set and our test set. We can compare these two performances to check for overfitting.

```
model.history.history
loss = model.history.history['loss']
sns.lineplot(x=range(len(loss)),y=loss)
plt.title("Training Loss per Epoch");
```



TF Syntax Basics - Part Three - Model Evaluation

Compare final evaluation (MSE) on training set and test set. These should hopefully be fairly close to each other.

```
model.metrics_names
        ['loss']

training_score = model.evaluate(X_train,y_train,verbose=0)
test_score = model.evaluate(X_test,y_test,verbose=0)
training_score
        250.80068969726562
test_score
        241.88800048828125

## Further Evaluation
test_predictions = model.predict(X_test)
test_predictions
    array([[421.34464],
    [606.23584],
    [581.3122 ],
    [558.92206],
    [381.22943],
    [567.55914],
    [507.18253],
    [469.82248],
    [539.5856 ],
    [460.58853],
    [596.45966],
    [550.32794],
    [432.48642],
    [422.66003],
    [637.36487],
    [450.53128],
    [514.8222 ],
    [630.7812 ],
    [636.61615],
    [560.02594],
    [357.36728],
    [454.38992],
    [399.2757 ],
    [399.1938 ],
    [557.4787 ],
        ...
    [522.90326],
    [502.56296],
    [595.3043 ],
    [434.68304],
    [423.2611 ]], dtype=float32)
```

```
pred_df = pd.DataFrame(y_test,columns=['Test Y'])
pred_df
```

	Test Y
0	402.296319
1	624.156198
2	582.455066
3	578.588606
4	371.224104
...	...
295	525.704657
296	502.909473
297	612.727910
298	417.569725
299	410.538250

300 rows × 1 columns

```
test_predictions = pd.Series(test_predictions.reshape(300,))
test_predictions
```

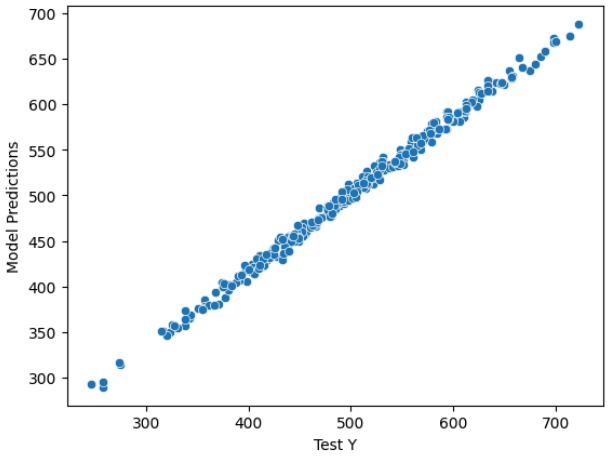
```
0      421.344635
1      606.235840
2      581.312195
3      558.922058
4      381.229431
          ...
295    522.903259
296    502.562958
297    595.304321
298    434.683044
299    423.261108
Length: 300, dtype: float32
```

```
pred_df = pd.concat([pred_df,test_predictions],axis=1)
pred_df.columns = ['Test Y','Model Predictions']
pred_df
```

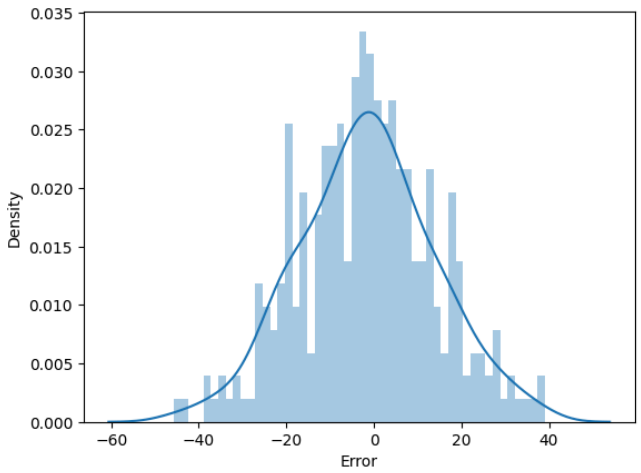
	Test Y	Model Predictions
0	402.296319	421.344635
1	624.156198	606.235840
2	582.455066	581.312195
3	578.588606	558.922058
4	371.224104	381.229431
...	...	...
295	525.704657	522.903259
296	502.909473	502.562958
297	612.727910	595.304321
298	417.569725	434.683044
299	410.538250	423.261108

300 rows × 2 columns

#Let's compare to the real test labels!
sns.scatterplot(x='Test Y',y='Model Predictions',data=pred_df)



pred_df['Error'] = pred_df['Test Y'] - pred_df['Model Predictions']
sns.distplot(pred_df['Error'],bins=50)



from sklearn.metrics import mean_absolute_error,mean_squared_error
mean_absolute_error(pred_df['Test Y'],pred_df['Model Predictions'])
12.243479899634158
mean_squared_error(pred_df['Test Y'],pred_df['Model Predictions'])
241.8879972369151
Essentially the same thing, difference just due to precision
test_score
241.88800048828125
#RMSE
test_score**0.5
15.55274896885696

Predicting on brand new data
What if we just saw a brand new gemstone from the ground? What should we price it at? This is the **exact** same procedure as predicting on a new test data!

[[Feature1, Feature2]]

```
new_gem = [[998,1000]]

# Don't forget to scale!
scaler.transform(new_gem)
```
array([[0.14117652, 0.53968792]])

```
new_gem = scaler.transform(new_gem)
model.predict(new_gem)
```
array([[426.7073]], dtype=float32)

```
## Saving and Loading a Model
from tensorflow.keras.models import load_model

model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'

later_model = load_model('my_model.h5')

later_model.predict(new_gem)
```
array([[420.68692]], dtype=float32)

TF Regression Code Along - Exploratory Data Analysis

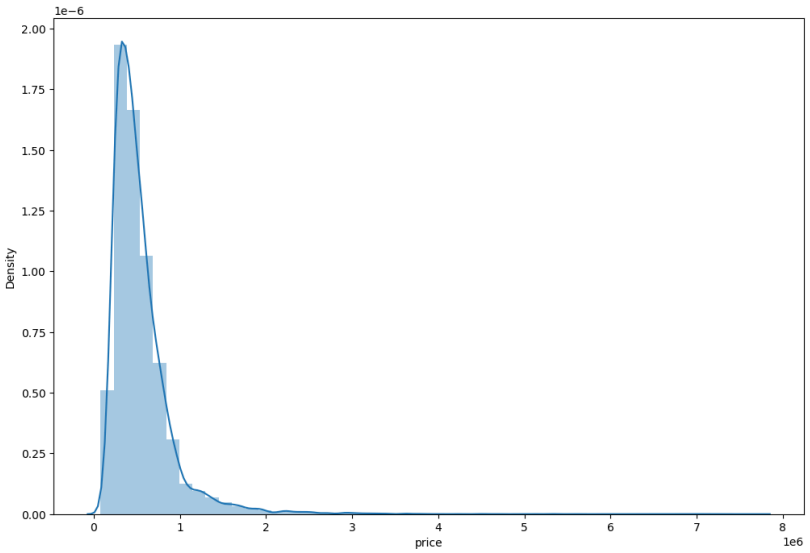
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('../DATA/kc_house_data.csv')

#Exploratory Data Analysis
df.isnull().sum()
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

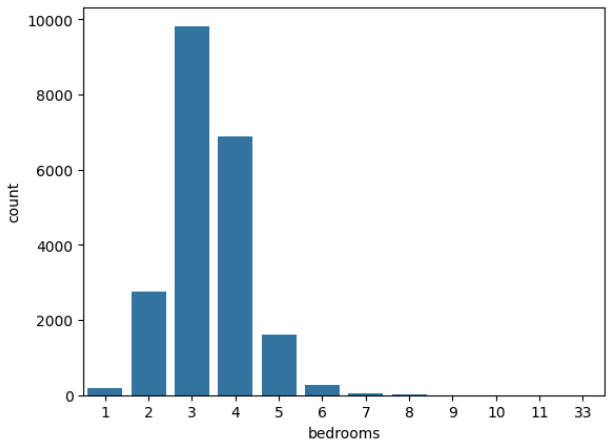
df.describe().transpose()

	count	mean	std	min	25%	50%	75%	max
id	21597.0	4.580474e+09	2.876736e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21597.0	5.402966e+05	3.673681e+05	7.800000e+04	3.220000e+05	4.500000e+05	6.450000e+05	7.700000e+06
bedrooms	21597.0	3.373200e+00	9.262989e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
bathrooms	21597.0	2.115826e+00	7.689843e-01	5.000000e-01	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
sqft_living	21597.0	2.080322e+03	9.181061e+02	3.700000e+02	1.430000e+03	1.910000e+03	2.550000e+03	1.354000e+04
sqft_lot	21597.0	1.509941e+04	4.141264e+04	5.200000e+02	5.040000e+03	7.618000e+03	1.068500e+04	1.651359e+06
floors	21597.0	1.494096e+00	5.396828e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
waterfront	21597.0	7.547345e-03	8.654900e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
view	21597.0	2.342918e-01	7.663898e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	21597.0	3.409825e+00	6.505456e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
grade	21597.0	7.657915e+00	1.173200e+00	3.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
sqft_above	21597.0	1.788597e+03	8.277598e+02	3.700000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
sqft_basement	21597.0	2.917250e+02	4.426678e+02	0.000000e+00	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_built	21597.0	1.971000e+03	2.937523e+01	1.900000e+03	1.951000e+03	1.975000e+03	1.997000e+03	2.015000e+03
yr_renovated	21597.0	8.446479e+01	4.018214e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	21597.0	9.807795e+04	5.351307e+01	9.800100e+04	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	21597.0	4.756009e+01	1.385518e-01	4.715590e+01	4.747110e+01	4.757180e+01	4.767800e+01	4.777760e+01
long	21597.0	-1.222140e+02	1.407235e-01	-1.225190e+02	-1.223280e+02	-1.222310e+02	-1.221250e+02	-1.213150e+02
sqft_living15	21597.0	1.986620e+03	6.852305e+02	3.990000e+02	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
sqft_lot15	21597.0	1.275828e+04	2.727444e+04	6.510000e+02	5.100000e+03	7.620000e+03	1.008300e+04	8.712000e+05

plt.figure(figsize=(12,8))
sns.distplot(df['price'])



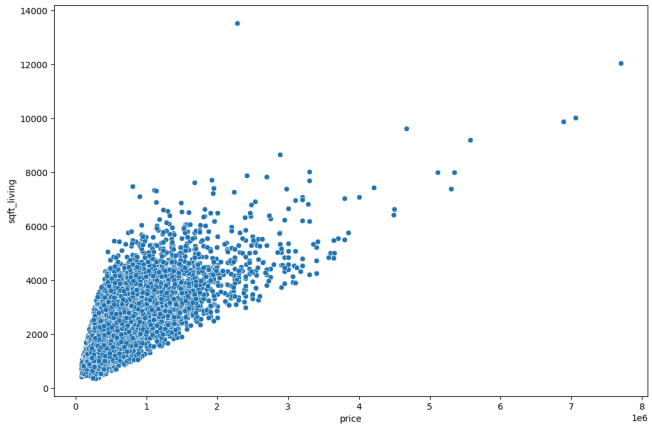
sns.countplot(data=df, x='bedrooms', order=sorted(df['bedrooms'].unique()))



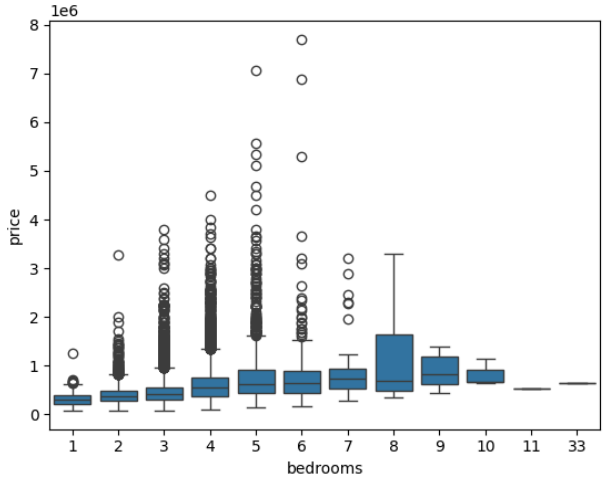
```
numerical_df = df.select_dtypes(include='number')
numerical_df.corr()['price'].sort_values()
```

```
zipcode         -0.053402
id              -0.016772
long             0.022036
condition        0.036056
yr_built         0.053953
sqft_lot15       0.082845
sqft_lot         0.089876
yr_renovated     0.126424
floors           0.256804
waterfront       0.266398
lat              0.306692
bedrooms         0.308787
sqft_basement    0.323799
view             0.397370
bathrooms        0.525906
sqft_living15    0.585241
sqft_above       0.605368
grade            0.667951
sqft_living      0.701917
price            1.000000
Name: price, dtype: float64
```

```
plt.figure(figsize=(12,8))
sns.scatterplot(x='price',y='sqft_living',data=df)
```

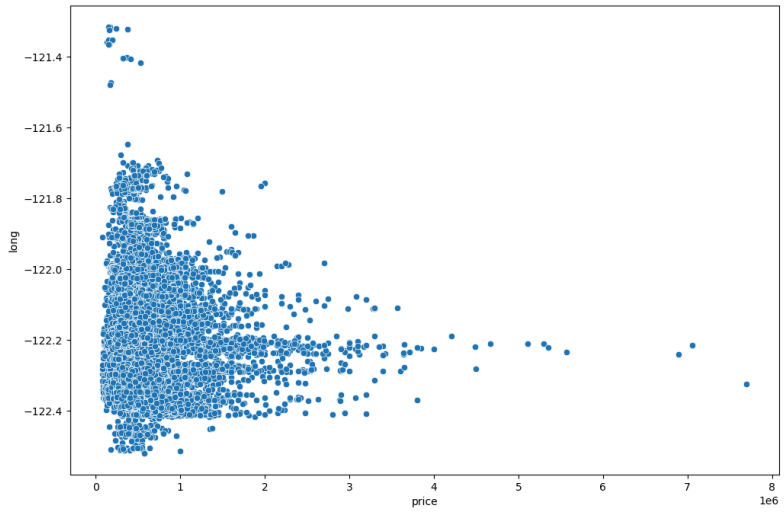



```
sns.boxplot(x='bedrooms',y='price',data=df)
```

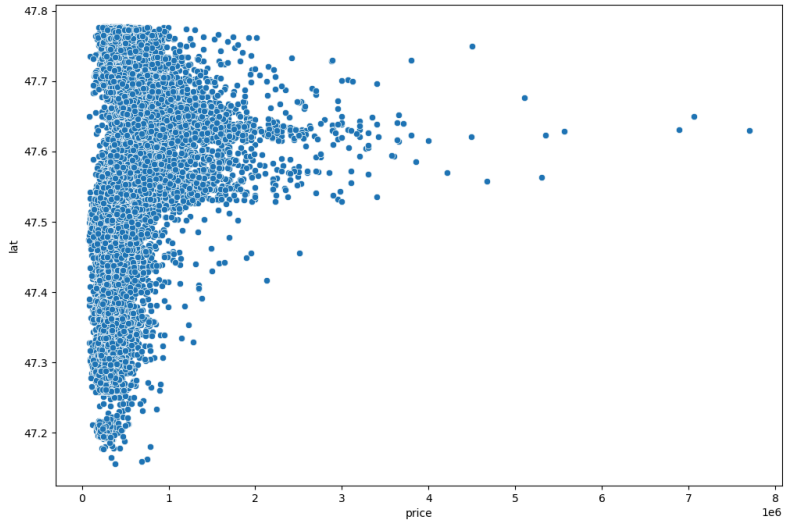


##Geographical Properties

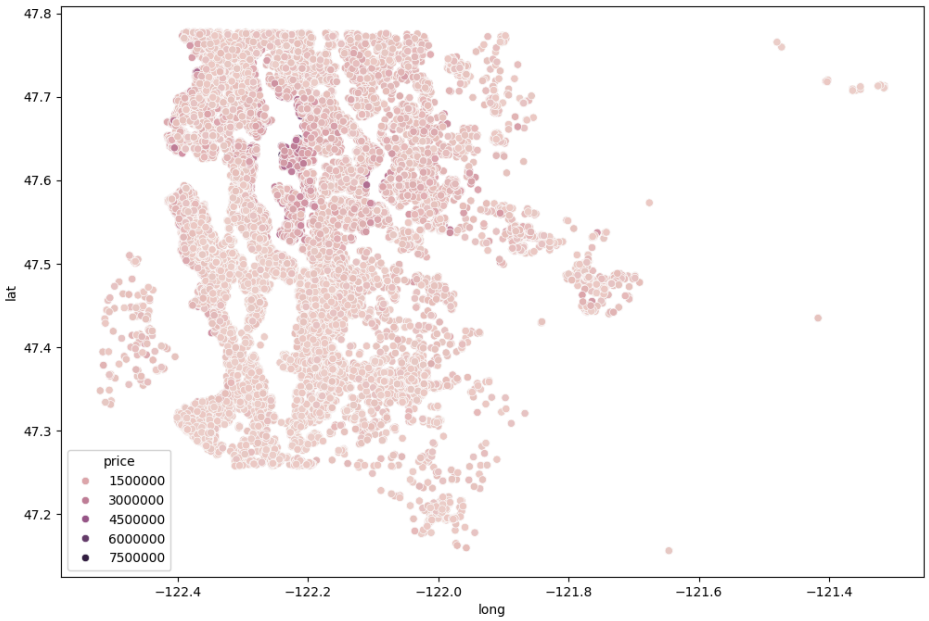
```
plt.figure(figsize=(12,8))
sns.scatterplot(x='price',y='long',data=df)
```




```
plt.figure(figsize=(12,8))
sns.scatterplot(x='price',y='lat',data=df)
```




```
plt.figure(figsize=(12,8))
sns.scatterplot(x='long',y='lat',data=df,hue='price') #Mapa de King Country
```



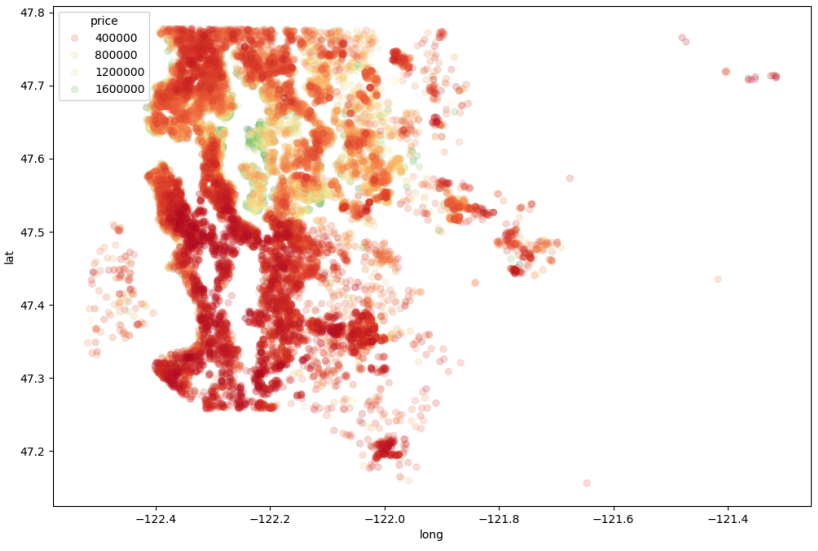
df.sort_values('price',ascending=False).head(20)

len(df)*(0.01) #Quiero saber cuanto es el 1% de la cantidad de datos que tenemos

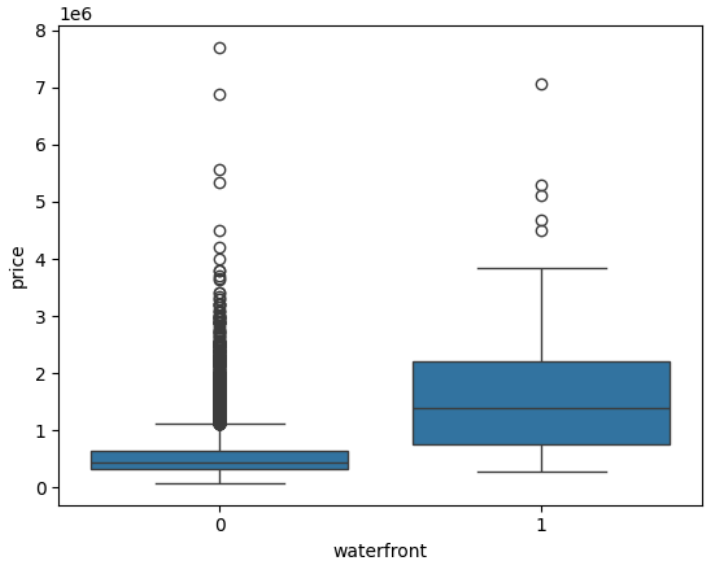
215.97

non_top_1_perc = df.sort_values('price',ascending=False).iloc[216:] #tomo todo menos el 1% mas caro, o sea lo q está después de la posición 216 en el listado ordenado x precio descendentemente

```
plt.figure(figsize=(12,8))
sns.scatterplot(x='long',y='lat',
                data=non_top_1_perc,hue='price',
                palette='RdYlGn',edgecolor=None,alpha=0.2)
```



##Other Features
sns.boxplot(x='waterfront',y='price',data=df)

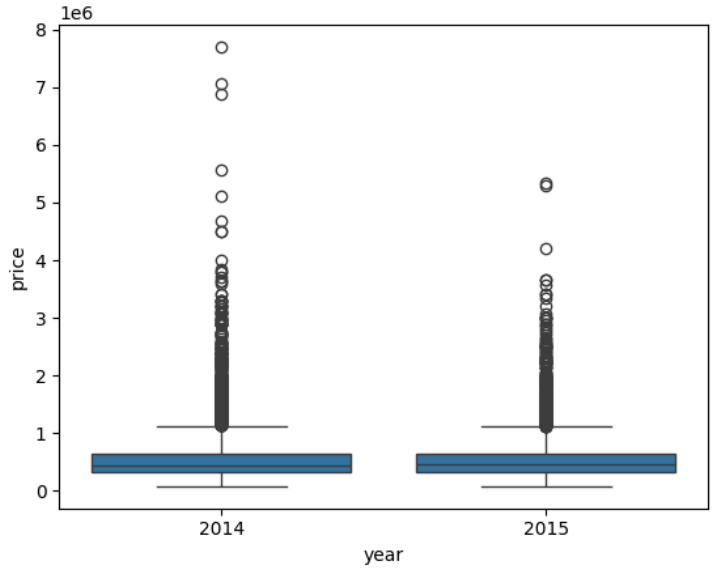


##Working with Feature Data
df.head()
df.info()
df = df.drop('id',axis=1) #Eliminamos la columna de id porque no aporta info

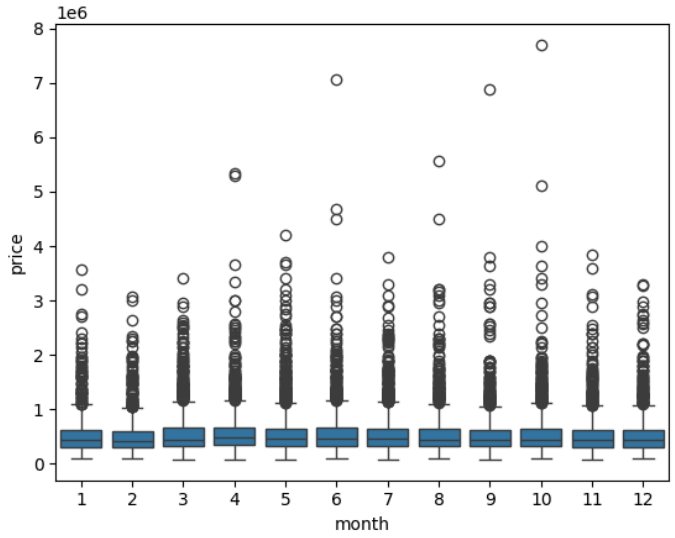
##Feature Engineering from Date
df['date'] #vemos que es dtype: object

df['date'] = pd.to_datetime(df['date']) #ahora es dtype: datetime64

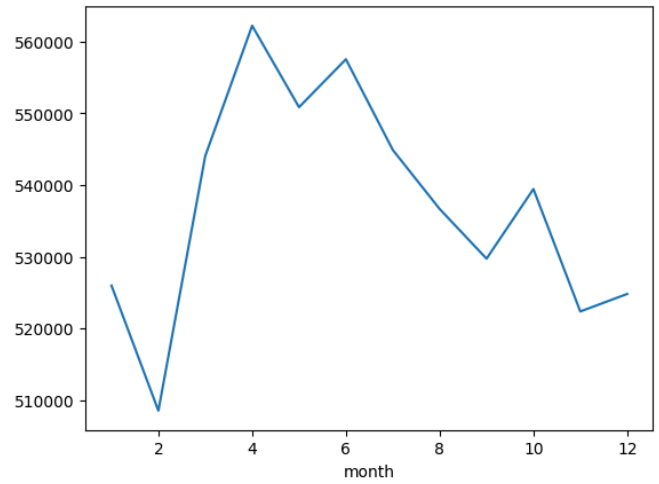
df['month'] = df['date'].apply(lambda date:date.month)
df['year'] = df['date'].apply(lambda date:date.year)
sns.boxplot(x='year',y='price',data=df)



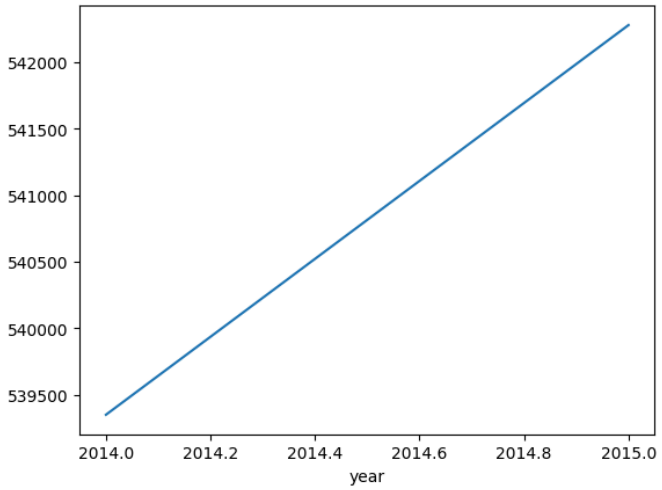
sns.boxplot(x='month',y='price',data=df)



df.groupby('month').mean()['price'].plot()



df.groupby('year').mean()['price'].plot()



```
df = df.drop('date',axis=1)
df.columns
        Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
               'waterfront', 'view', 'condition', 'grade', 'sqft_above',
               'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
               'sqft_living15', 'sqft_lot15', 'month', 'year'],
              dtype='object')

# https://i.pinimg.com/originals/4a/ab/31/4aab31ce95d5b8474fd2cc063f334178.jpg
# May be worth considering to remove this or feature engineer categories from it
df['zipcode'].value_counts() #como hay 70 codigos postales distintos es demasiado para pivotearlos como columnas y ponerles 0 o 1, asi que lo aliminamos
df = df.drop('zipcode',axis=1)

# could make sense due to scaling, higher should correlate to more value
df['yr_renovated'].value_counts() #vemos que la mayoría son 0
```

```
yr_renovated
0       20683
2014       91
2013       37
2003       36
2005       35
          ...
1951        1
1959        1
1948        1
1954        1
1944        1
Name: count, Length: 70, dtype: int64
```

```
df['sqft_basement'].value_counts()
```

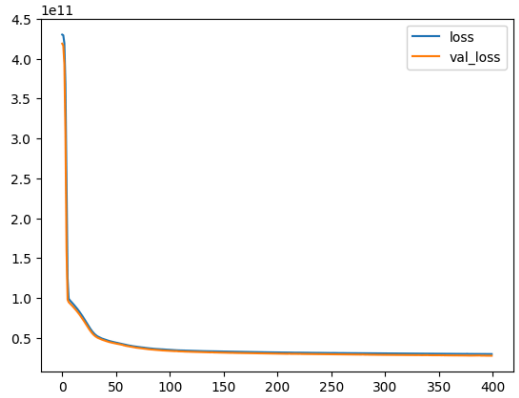
```
sqft_basement
0      13110
600      221
700      218
500      214
800      206
        ...
518        1
374        1
784        1
906        1
248        1
Name: count, Length: 306, dtype: int64
```

TF Regression Code Along - Data Preprocessing and Creating a Model

```
## Scaling and Train Test Split
X = df.drop('price',axis=1)
y = df['price']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=101)
```

Scaling
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train= scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train.shape
```
(15117, 19)
```
X_test.shape
```
(6480, 19)

Creating a Model
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam

X_train.shape
```
(15117, 19)
```
model = Sequential()
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam',loss='mse')
```

Training the Model
```
model.fit(x=X_train,y=y_train.values,
        validation_data=(X_test,y_test.values),
        batch_size=128,epochs=400)

losses = pd.DataFrame(model.history.history)
losses.plot()
```


TF Regression Code Along - Model Evaluation and Predictions

https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

from sklearn.metrics import mean_squared_error,mean_absolute_error,explained_variance_score

Predicting on Brand New Data

X_test

```
array([[0.1       , 0.08      , 0.04239917, ..., 0.00887725, 0.63636364,
        0.        ],
       [0.3       , 0.36      , 0.17269907, ..., 0.00993734, 0.81818182,
        0.        ],
       [0.2       , 0.24      , 0.12512927, ..., 0.00547073, 0.90909091,
        0.        ],
       ...,
       [0.1       , 0.08      , 0.05584281, ..., 0.00506255, 1.        ,
        0.        ],
       [0.3       , 0.2       , 0.22233713, ..., 0.00774485, 0.09090909,
        1.        ],
       [0.3       , 0.32      , 0.27611169, ..., 0.0196531 , 0.45454545,
        0.        ]])
```

predictions = model.predict(X_test)

mean_absolute_error(y_test,predictions)

102961.70023690684

np.sqrt(mean_squared_error(y_test,predictions))

166599.53189479667

explained_variance_score(y_test,predictions)

0.7908291632578597

df['price'].mean()

540296.5735055795

df['price'].median()

450000.0

102961.70023690684 / 540296.5735055795 #hacemos el error absoluto medio dividido el precio promedio y nos da un idea del error, aprox 20%

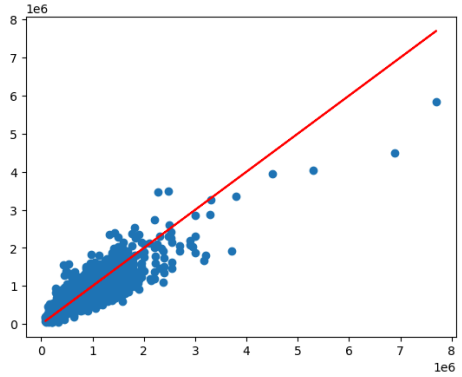
0.1905651549275346

Our predictions

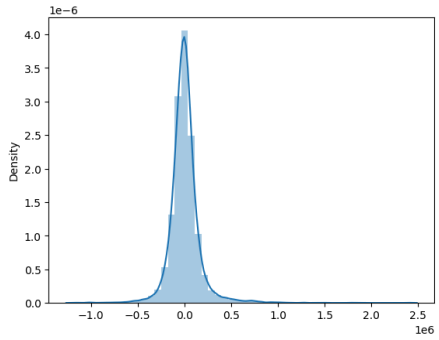
plt.scatter(y_test,predictions)

Perfect predictions

plt.plot(y_test,y_test,'r')



```
errors = y_test.values.reshape(6480, 1) - predictions
sns.distplot(errors)
```




```
### Predicting on a brand new house
single_house = df.drop('price',axis=1).iloc[0]
single_house = scaler.transform(single_house.values.reshape(-1, 19))
single_house
```

```
array([[0.2       , 0.08      , 0.08376422, 0.00310751, 0.        ,
        0.        , 0.        , 0.5       , 0.4       , 0.10785619,
        0.        , 0.47826087, 0.        , 0.57149751, 0.21760797,
        0.16193426, 0.00582059, 0.81818182, 0.        ]])
```

```
model.predict(single_house)
```

```
array([[285569.03]], dtype=float32)
```

```
df.iloc[0]
```

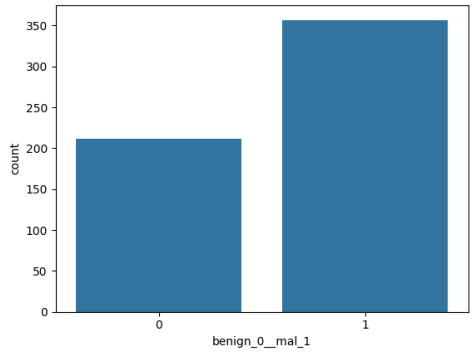
```
price            221900.0000
bedrooms              3.0000
bathrooms             1.0000
sqft_living        1180.0000
sqft_lot           5650.0000
floors                1.0000
waterfront            0.0000
view                  0.0000
condition             3.0000
grade                 7.0000
sqft_above         1180.0000
sqft_basement         0.0000
yr_built           1955.0000
yr_renovated          0.0000
lat                  47.5112
long               -122.2570
sqft_living15      1340.0000
sqft_lot15         5650.0000
month                10.0000
year               2014.0000
Name: 0, dtype: float64
```

TF Classification Code Along - EDA and Preprocessing

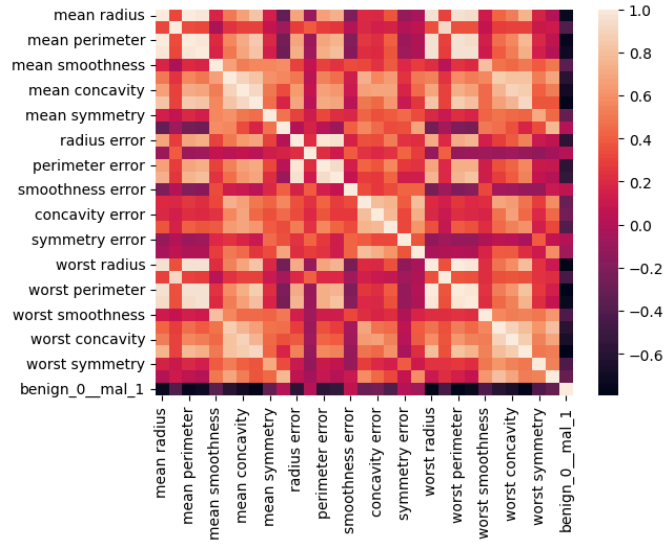
Vamos a mostrar cómo realizar una tarea de clasificación con TensorFlow. También nos centraremos en cómo identificar y abordar el sobreajuste mediante devoluciones de llamada de parada temprana y capas de abandono (Early Stopping Callbacks and Dropout Layers)

- Early Stopping (parada temprana): Keras puede detener automáticamente el entrenamiento en función de una condición de pérdida en los datos de validación que pasamos durante la llamada a model.fit().
- Dropout Layers (capas de abandono): Se puede agregar abandono a las capas para "apagar" las neuronas durante el entrenamiento y evitar un sobreajuste. Cada capa de abandono "eliminará" un porcentaje definido por el usuario de unidades de neuronas en la capa anterior en cada lote. Eso significa que ciertas neuronas no tienen sus pesos o sesgos afectados durante un lote, en lugar de eso, simplemente están apagados .

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

df = pd.read_csv('../DATA/cancer_classification.csv')
df.info()
df.describe().transpose()

#EDA
sns.countplot(x='benign_0__mal_1',data=df)
```




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

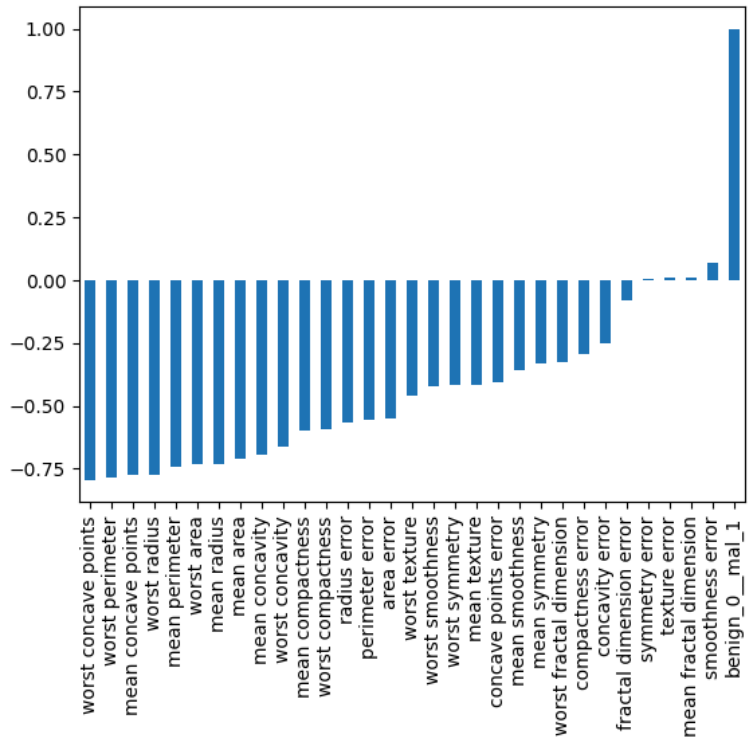


df.corr()['benign_0__mal_1'].sort_values()

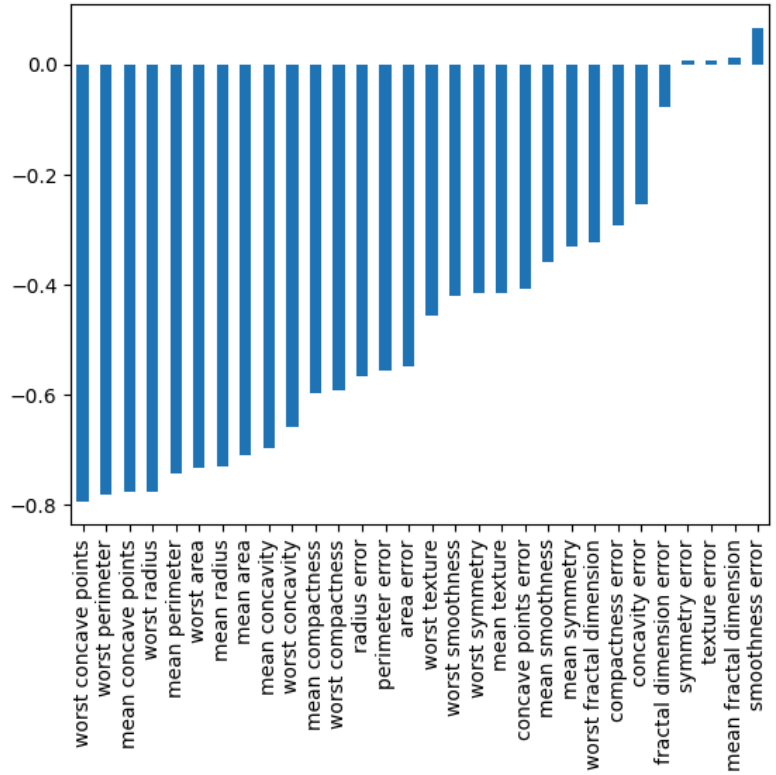
```
worst concave points      -0.793566
worst perimeter           -0.782914
mean concave points       -0.776614
worst radius              -0.776454
mean perimeter            -0.742636
worst area                -0.733825
mean radius               -0.730029
mean area                 -0.708984
mean concavity            -0.696360
worst concavity           -0.659610
mean compactness          -0.596534
worst compactness         -0.590998
radius error              -0.567134
perimeter error           -0.556141
area error                -0.548236
worst texture             -0.456903
worst smoothness          -0.421465
worst symmetry            -0.416294
mean texture              -0.415185
concave points error      -0.408042
mean smoothness           -0.358560
mean symmetry             -0.330499
worst fractal dimension   -0.323872
compactness error         -0.292999
concavity error           -0.253730
...
texture error              0.008303
mean fractal dimension     0.012838
smoothness error           0.067016
benign_0__mal_1            1.000000
Name: benign_0__mal_1, dtype: float64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

df.corr()['benign_0__mal_1'].sort_values().plot(kind='bar')



df.corr()['benign_0__mal_1'][:-1].sort_values().plot(kind='bar')



Train Test Split
X = df.drop('benign_0__mal_1',axis=1).values
y = df['benign_0__mal_1'].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=101)

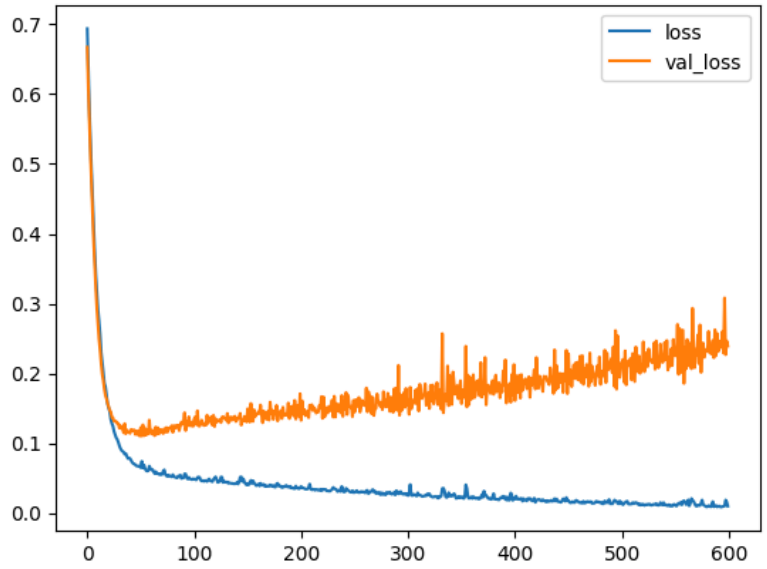
Scaling Data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

TF Classification - Dealing with Overfitting and Evaluation

Creating the Model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
X_train.shape
(426,30)
model = Sequential()

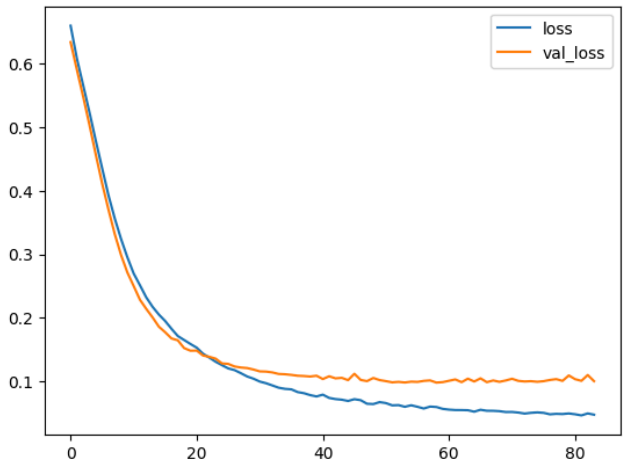
#https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
model.add(Dense(units=30,activation='relu'))
model.add(Dense(units=15,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
# For a binary classification problem
model.compile(loss='binary_crossentropy', optimizer='adam')

Training the Model
Example One: Choosing too many epochs and overfitting!
#https://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network
#https://datascience.stackexchange.com/questions/18414/are-there-any-rules-for-choosing-the-size-of-a-mini-batch
model.fit(x=X_train, y=y_train, epochs=600, validation_data=(X_test, y_test), verbose=1)
# model.history.history
model_loss = pd.DataFrame(model.history.history)
# model_loss
model_loss.plot()



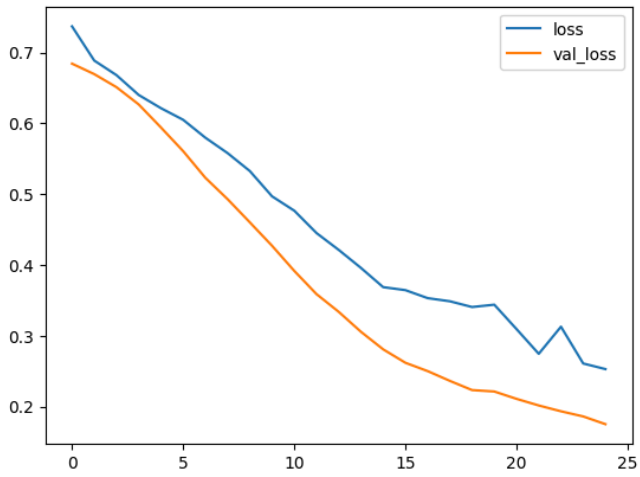
Example Two: Early Stopping
We obviously trained too much! Let's use early stopping to track the val_loss and stop training once it begins increasing too much!
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dense(units=15,activation='relu'))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
from tensorflow.keras.callbacks import EarlyStopping
help(EarlyStopping)

```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)
model.fit(x=X_train,  y=y_train,  epochs=600, validation_data=(X_test, y_test), verbose=1,
callbacks=[early_stop])
model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
```



Example Three: Adding in DropOut Layers

```
from tensorflow.keras.layers import Dropout
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=15,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')

model.fit(x=X_train, y=y_train, epochs=600, validation_data=(X_test, y_test), verbose=1,
callbacks=[early_stop])

model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
```



Model Evaluation
predictions = model.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))

```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94        55
           1       0.97      0.95      0.96        88

    accuracy                           0.95       143
   macro avg       0.95      0.95      0.95       143
weighted avg       0.95      0.95      0.95       143
```

print(confusion_matrix(y_test,predictions))
[[52 3]
 [4 84]]

Keras API Project Exercise

We will be using a subset of the LendingClub DataSet obtained from Kaggle: https://www.kaggle.com/wordsforthewise/lending-club

LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California. It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub is the world's largest peer-to-peer lending platform.

Our Goal: Given historical data on loans given out with information on whether or not the borrower defaulted (charge-off), can we build a model that can predict wether or nor a borrower will pay back their loan? This way in the future when we get a new potential customer we can assess whether or not they are likely to pay back the loan. Keep in mind classification metrics when evaluating the performance of your model! The **"loan_status"** column contains our label.

	LoanStatNew	Description
0	loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
1	term	The number of payments on the loan. Values are in months and can be either 36 or 60.
2	int_rate	Interest Rate on the loan
3	installment	The monthly payment owed by the borrower if the loan originates.
4	grade	LC assigned loan grade
5	sub_grade	LC assigned loan subgrade
6	emp_title	The job title supplied by the Borrower when applying for the loan.*
7	emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
8	home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER
9	annual_inc	The self-reported annual income provided by the borrower during registration.
10	verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
11	issue_d	The month which the loan was funded
12	loan_status	Current status of the loan
13	purpose	A category provided by the borrower for the loan request.
14	title	The loan title provided by the borrower
15	zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.
16	addr_state	The state provided by the borrower in the loan application
17	dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
18	earliest_cr_line	The month the borrower's earliest reported credit line was opened
19	open_acc	The number of open credit lines in the borrower's credit file.

20	pub_rec	Number of derogatory public records
21	revol_bal	Total credit revolving balance
22	revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
23	total_acc	The total number of credit lines currently in the borrower's credit file
24	initial_list_status	The initial listing status of the loan. Possible values are – W, F
25	application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
26	mort_acc	Number of mortgage accounts.
27	pub_rec_bankruptcies	Number of public record bankruptcies

```
import pandas as pd
data_info = pd.read_csv('../DATA/lending_club_info.csv',index_col='LoanStatNew')
print(data_info.loc['revol_util']['Description'])
```

Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

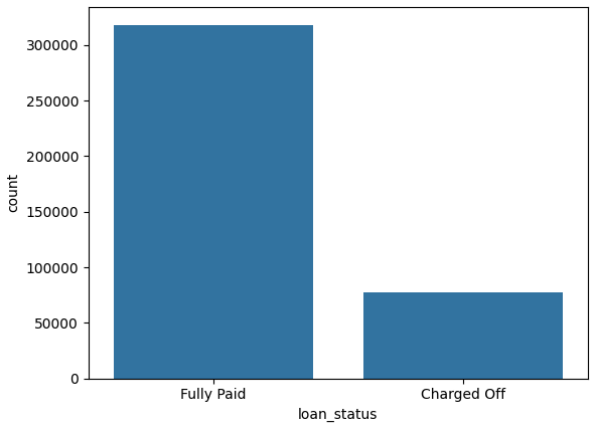
```
def feat_info(col_name):
   print(data_info.loc[col_name]['Description'])
feat_info('mort_acc')
```

Number of mortgage accounts.

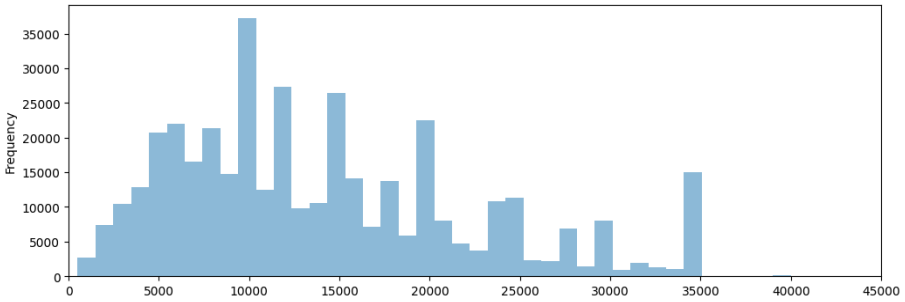
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('../DATA/lending_club_loan_two.csv')
df.info()
```

 # Section 1: Exploratory Data Analysis: Get an understanding for which variables are important, view summary statistics, and visualize the data

```
sns.countplot(x='loan_status',data=df)
```



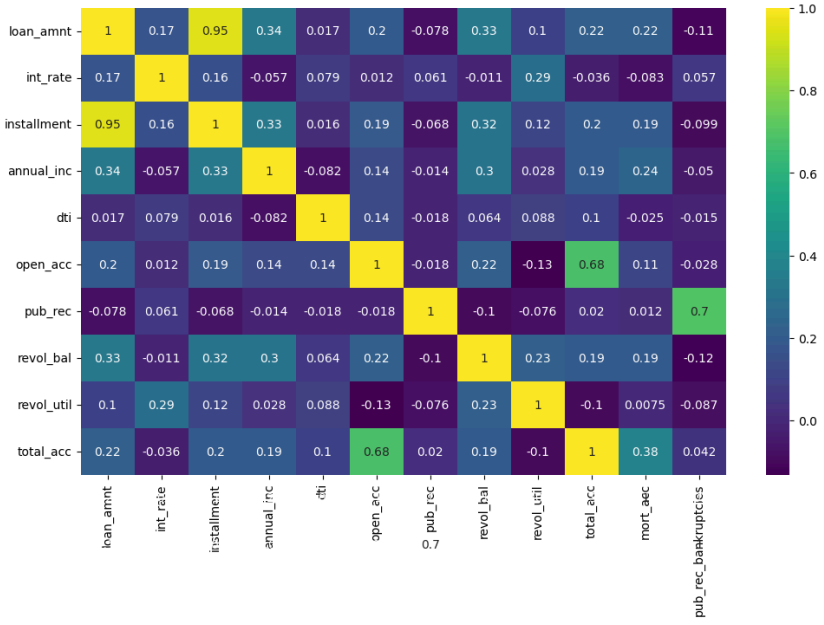

```
plt.figure(figsize=(12,4))
df['loan_amnt'].plot.hist(alpha=0.5,bins=40) #alpha le da la transparencia
plt.xlim(0,45000) #esto es para que llegue hasta 45000
```




```
# Seleccionar solo las columnas numéricas
numerical_df = df.select_dtypes(include='number')
# Mostrar la matriz de correlación de las columnas numéricas de 'tips'
correlation_matrix = numerical_df.corr()
correlation_matrix
```

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc	pub_rec_bankruptcies
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556	-0.077779	0.328320	0.099911	0.223886	0.222315	-0.106539
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649	0.060986	-0.011280	0.293659	-0.036404	-0.082583	0.057450
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973	-0.067892	0.316455	0.123915	0.202430	0.193694	-0.098628
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150	-0.013720	0.299773	0.027871	0.193023	0.236320	-0.050162
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181	-0.017639	0.063571	0.088375	0.102128	-0.025439	-0.014558
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000	-0.018392	0.221192	-0.131420	0.680728	0.109205	-0.027732
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392	1.000000	-0.101664	-0.075910	0.019723	0.011552	0.699408
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192	-0.101664	1.000000	0.226346	0.191616	0.194925	-0.124532
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420	-0.075910	0.226346	1.000000	-0.104273	0.007514	-0.086751
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728	0.019723	0.191616	-0.104273	1.000000	0.381072	0.042035
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205	0.011552	0.194925	0.007514	0.381072	1.000000	0.027239
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732	0.699408	-0.124532	-0.086751	0.042035	0.027239	1.000000

```
plt.figure(figsize=(12,7))
sns.heatmap(correlation_matrix,cmap='viridis',annot=True)
plt.ylim(10, 0)
```



You should have noticed almost perfect correlation with the "**installment**" feature. Explore this feature further. Print out their descriptions and perform a scatterplot between them. Does this relationship make sense to you? Do you think there is duplicate information here?

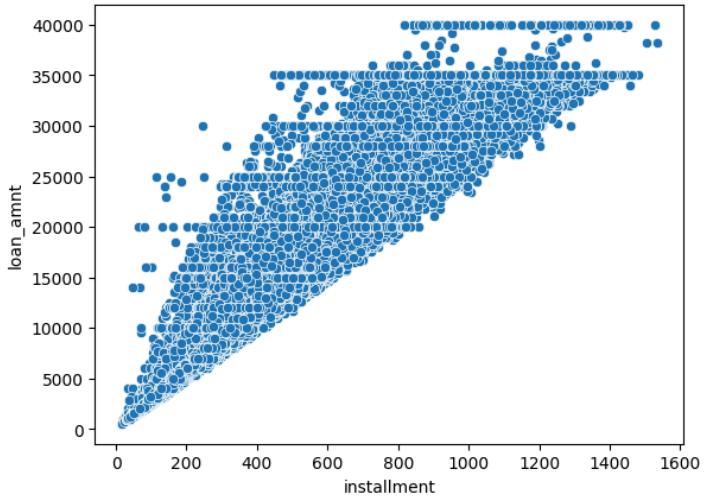
feat_info('installment')

The monthly payment owed by the borrower if the loan originates.

feat_info('loan_amnt')

The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

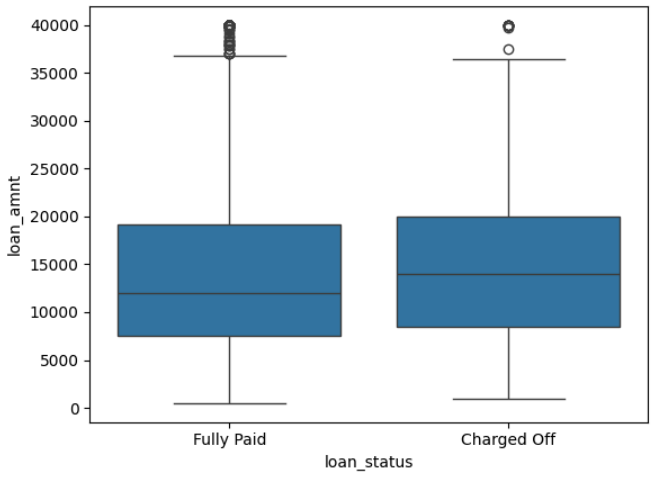
sns.scatterplot(x='installment',y='loan_amnt',data=df)



Existe una relación entre préstamos altos y no poder pagarlos? Por lo que vemos si el préstamo es más bajo tenemos un ligero aumento

#La mediana de los préstamos cancelados ("Charged Off") es superior (aprox $15,000) en comparación con los préstamos pagados completamente ("Fully Paid", alrededor de $12,500). Esto sugiere que, en promedio, los préstamos de mayor monto tienen una mayor probabilidad de no ser pagados y ser cancelados

sns.boxplot(x="loan_status", y="loan_amnt", data=df)



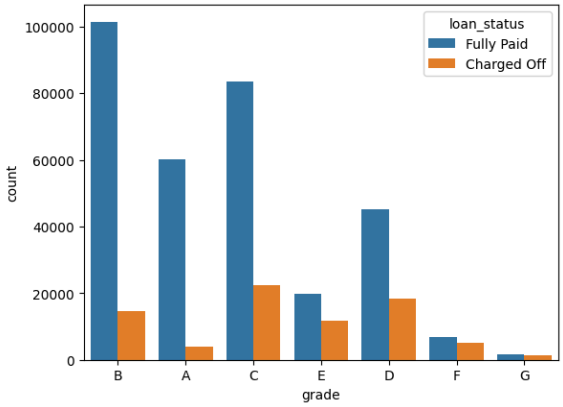
Calculate the summary statistics for the loan amount, grouped by the loan_status.

df.groupby('loan_status')['loan_amnt'].describe()

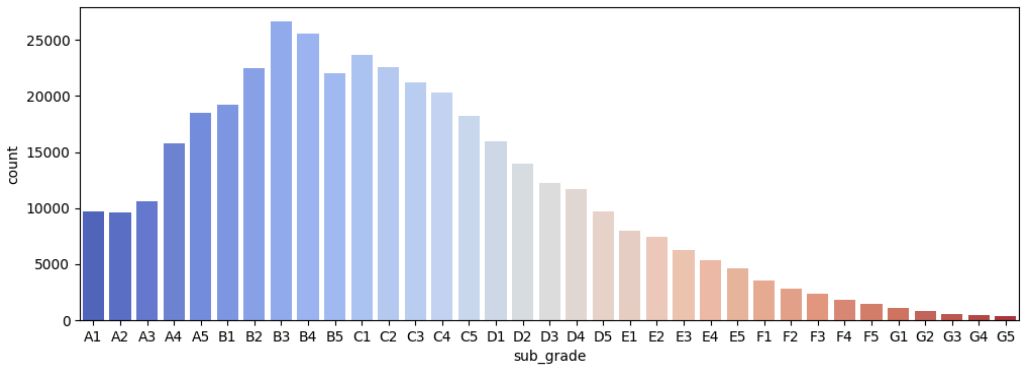
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

Let's explore the Grade and SubGrade columns that LendingClub attributes to the loans. What are the unique possible grades and subgrades?
sorted(df['grade'].unique())
sorted(df['sub_grade'].unique())

Create a countplot per grade. Set the hue to the loan_status label.
#Queremos ver si hay una diferencia grande entre si paga o no y su claificacion
sns.countplot(x='grade',hue='loan_status',data=df)

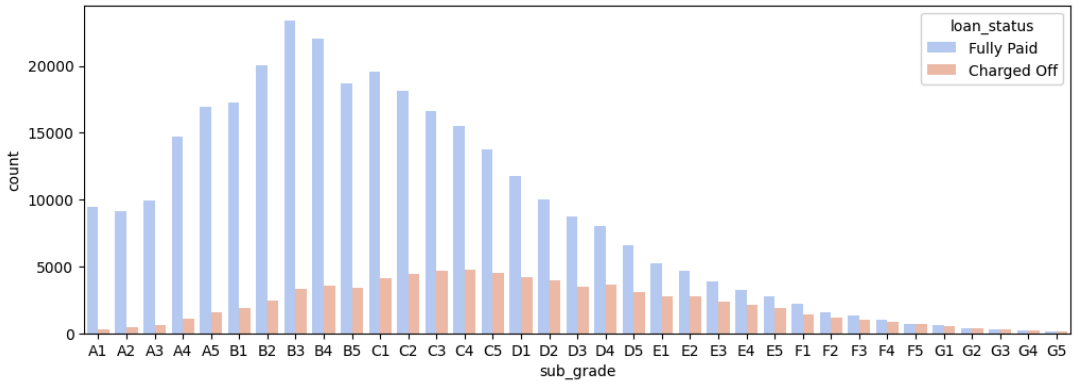


Display a count plot per subgrade. You may need to resize for this plot and reorder (https://seaborn.pydata.org/generated/seaborn.countplot.html#seaborn.countplot) the x axis. Feel free to edit the color palette. Explore both all loans made per subgrade as well being separated based on the loan_status. After creating this plot, go ahead and create a similar plot, but set hue="loan_status"
plt.figure(figsize=(12,4))
sns.countplot(x='sub_grade',
data=df,
order=sorted(df['sub_grade'].unique()),
palette='coolwarm')



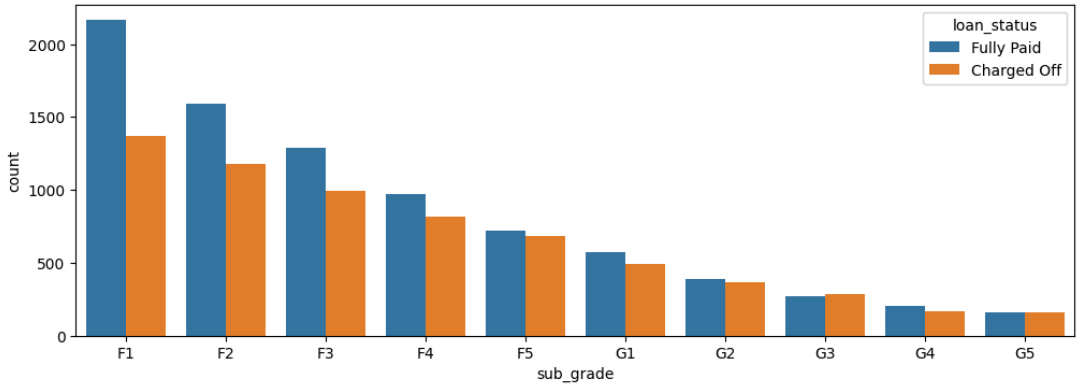
plt.figure(figsize=(12,4))
sns.countplot(x='sub_grade',

```
        data=df,
        order=sorted(df['sub_grade'].unique()),
        palette='coolwarm',
        hue='loan_status'
        )
```



It looks like F and G subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.

```
plt.figure(figsize=(12,4))
f_and_g = df[(df['grade']=='G') | (df['grade']=='F')]
sns.countplot(x='sub_grade',
        data=f_and_g,
        order=sorted(f_and_g['sub_grade'].unique()),
        hue='loan_status'
        )
```



Create a new column called 'loan_repaid' which will contain a 1 if the loan status was "Fully Paid" and a 0 if it was "Charged Off".

```
df['loan_status'].unique()
```

array(['Fully Paid', 'Charged Off'], dtype=object)

```
#Función para asignar 1 o 0 a un campo en función del valor de otro campo
def binary_result(x):
    if x =='Fully Paid':
        i=1
    else: i=0
```

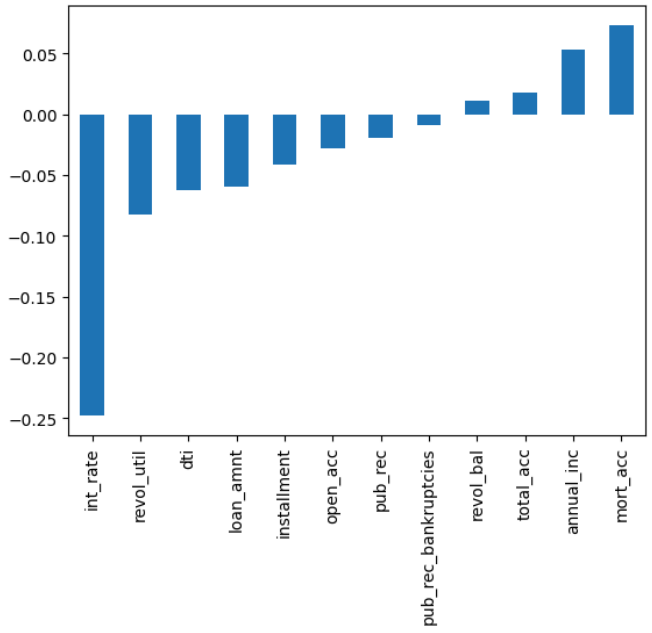
```
    return i

df['loan_repaid']=df['loan_status'].apply(binary_result)

# Otra forma de hacerlo
# df['loan_repaid'] = df['loan_status'].map({'Fully Paid':1,'Charged Off':0})

df[['loan_repaid','loan_status']]
```

	loan_repaid	loan_status
0	1	Fully Paid
1	1	Fully Paid
2	1	Fully Paid
3	1	Fully Paid
4	0	Charged Off
...	...	...
396025	1	Fully Paid
396026	1	Fully Paid
396027	1	Fully Paid
396028	1	Fully Paid
396029	1	Fully Paid

396030 rows × 2 columns

Create a bar plot showing the correlation of the numeric features to the new loan_repaid column. Helpful Link: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.bar.html

```
numerical_df = df.select_dtypes(include='number')
correlation_matrix = numerical_df.corr()
correlation_matrix['loan_repaid'][:-1].sort_values().plot(kind='bar')
```



Section 2: Data PreProcessing: Remove or fill any missing data. Remove unnecessary or repetitive features. Convert categorical string features to dummy variables.
df.head()

Missing Data: Let's explore this missing data columns. We use a variety of factors to decide whether or not they would be useful, to see if we should keep, discard, or fill in the missing data.

What is the length of the dataframe?
len(df)

Create a Series that displays the total count of missing values per column.
df.isnull().sum()

```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1756
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
loan_repaid                 0
dtype: int64
```

Convert this Series to be in term of percentage of the total DataFrame
100* df.isnull().sum()/len(df)

Let's examine emp_title and emp_length to see whether it will be okay to drop them. Print out their feature information using the feat_info() function from the top of this notebook.
feat_info('emp_title')

*The job title supplied by the Borrower when applying for the loan.**

feat_info('emp_length')

Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.

How many unique employment job titles are there?
df['emp_title'].nunique()

173105

df['emp_title'].value_counts()

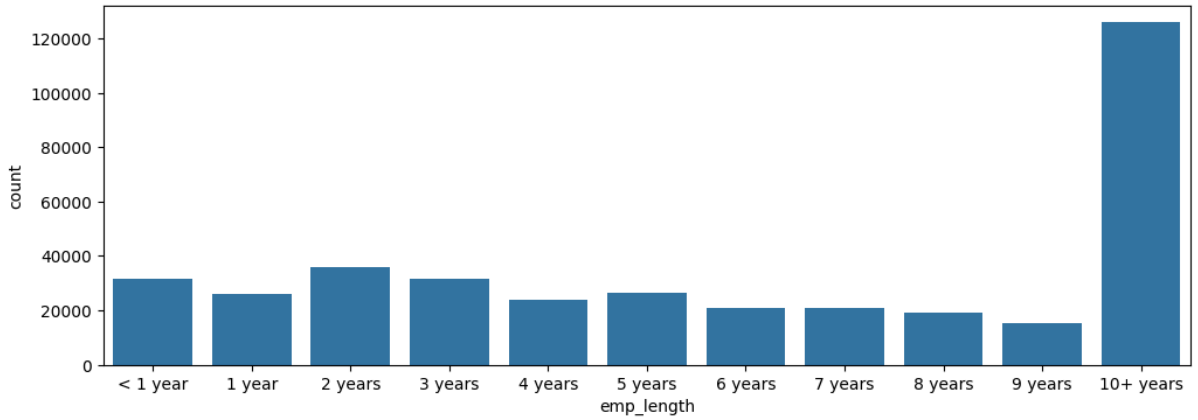
```
emp_title
Teacher                    4389
Manager                    4250
Registered Nurse           1856
RN                         1846
Supervisor                 1830
                           ...
Postman                       1
McCarthy & Holthus, LLC       1
jp flooring                   1
Histology Technologist        1
Gracon Services, Inc          1
Name: count, Length: 173105, dtype: int64
```

Realistically there are too many unique job titles to try to convert this to a dummy variable feature. Let's remove that emp_title column.
df=df.drop('emp_title',axis=1)

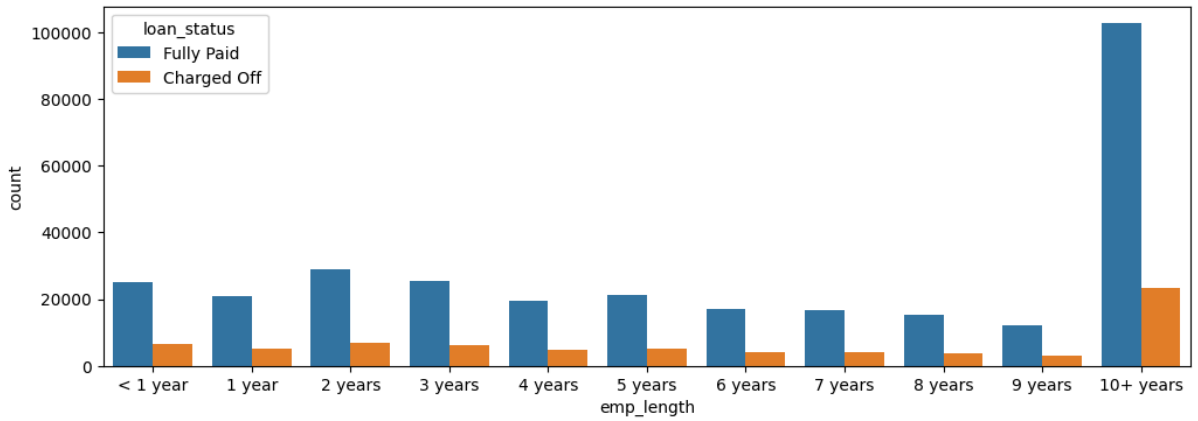
Create a count plot of the emp_length feature column. Challenge: Sort the order of the values.
sorted(df['emp_length'].**dropna**().unique())

['1 year',
'10+ years',
'2 years',
'3 years',
'4 years',
'5 years',
'6 years',
'7 years',
'8 years',
'9 years',
'< 1 year']

```
emp_length_order = [ '< 1 year',
                     '1 year',
                    '2 years',
                    '3 years',
                    '4 years',
                    '5 years',
                    '6 years',
                    '7 years',
                    '8 years',
                    '9 years',
                    '10+ years']

plt.figure(figsize=(12,4))
sns.countplot(data=df, x='emp_length', order=emp_length_order)
```



plt.figure(figsize=(12,4))
sns.countplot(data=df, x='emp_length', order=emp_length_order, hue='loan_status')



This still doesn't really inform us if there is a strong relationship between employment length and being charged off, what we want is the **% of charge offs per category**. Essentially informing us **what % of people per employment category didn't pay back their loan**. There are a multitude of ways to create this Series. Once you've created it, see if visualize it with a bar plot
https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.plot.html
This may be tricky, refer to solutions if you get stuck on creating this Series.
df_chargedoff=df[df['loan_status']=='Charged Off']
emp_co = df_chargedoff.groupby("emp_length").count()['loan_status']
emp_co

```
emp_length
1 year        5154
10+ years    23215
2 years       6924
3 years       6182
4 years       4608
5 years       5092
6 years       3943
7 years       4055
8 years       3829
9 years       3070
< 1 year      6563
Name: loan_status, dtype: int64
```

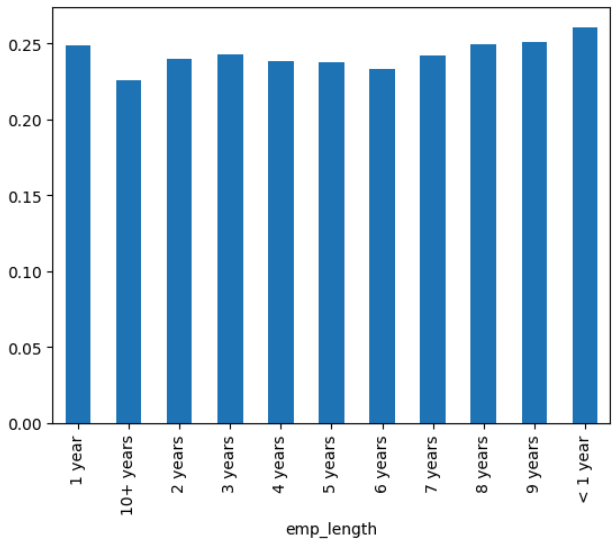
emp_fp = df[df['loan_status']=="Fully Paid"].groupby("emp_length").count()['loan_status']
emp_fp

```
emp_length
1 year        20728
10+ years    102826
2 years       28903
3 years       25483
4 years       19344
5 years       21403
6 years       16898
7 years       16764
8 years       15339
9 years       12244
< 1 year      25162
Name: loan_status, dtype: int64
```

emp_len = emp_co/emp_fp
emp_len

```
emp_length
1 year       0.248649
10+ years    0.225770
2 years      0.239560
3 years      0.242593
4 years      0.238213
5 years      0.237911
6 years      0.233341
7 years      0.241887
8 years      0.249625
9 years      0.250735
< 1 year     0.260830
Name: loan_status, dtype: float64
```

emp_len.plot(kind='bar')



Charge off rates are extremely similar across all employment lengths. Go ahead and drop the emp_length column.
df=df.drop('emp_length',axis=1)

Revisit the DataFrame to see what feature columns still have missing data.
df.isnull().sum()

```
loan_amnt                0
term                     0
int_rate                 0
installment              0
grade                    0
sub_grade                0
home_ownership           0
annual_inc               0
verification_status      0
issue_d                  0
loan_status              0
purpose                  0
title                 1756
dti                      0
earliest_cr_line         0
open_acc                 0
pub_rec                  0
revol_bal                0
revol_util             276
total_acc                0
initial_list_status      0
application_type         0
mort_acc             37795
pub_rec_bankruptcies   535
address                  0
loan_repaid              0
dtype: int64
```

Review the title column vs the purpose column. Is this repeated information?
df['title'].value_counts()
df['purpose'].value_counts()

df['title'].head(10)
df['purpose'].head(10)

The title column is simply a string subcategory/description of the purpose column. Go ahead and drop the title column.
df=df.drop('title',axis=1)

This is one of the hardest parts of the project! Refer to the solutions video if you need guidance, feel free to fill or drop the missing values of the mort_acc however you see fit! Here we're going with a very specific approach.
Find out what the mort_acc feature represents
feat_info('mort_acc')

Number of mortgage accounts.

Create a value_counts of the mort_acc column.
df['mort_acc'].value_counts()

There are many ways we could deal with this missing data. We could attempt to build a simple model to fill it in, such as a linear model, we could just fill it in based on the mean of the other columns, or you could even bin the columns into categories and then set NaN as its own category. There is no 100% correct approach! Let's review the other columsn to see which most highly correlates to mort_acc
print("Correlation with the mort_acc column")
# Seleccionar solo las columnas numéricas
numerical_df = df.select_dtypes(include='number')
# Mostrar la matriz de correlación de las columnas numéricas de 'tips'
correlation_matrix = numerical_df.corr()
correlation_matrix['mort_acc'].sort_values()

```
Correlation with the mort_acc column

int_rate               -0.082583
dti                    -0.025439
revol_util              0.007514
pub_rec                 0.011552
pub_rec_bankruptcies    0.027239
loan_repaid             0.073111
open_acc                0.109205
installment             0.193694
revol_bal               0.194925
loan_amnt               0.222315
annual_inc              0.236320
total_acc               0.381072
mort_acc                1.000000
Name: mort_acc, dtype: float64
```

Looks like the total_acc feature correlates with the mort_acc , this makes sense! Let's try this fillna() approach. We will group the dataframe by the total_acc and calculate the mean value for the mort_acc per total_acc entry. To get the result below:
print("Mean of mort_acc column per total_acc")
#df.groupby('total_acc').mean()['mort_acc']
df.dropna(subset=['mort_acc']).groupby('total_acc')['mort_acc'].mean()

```
Mean of mort_acc column per total_acc

total_acc
2.0      0.000000
3.0      0.052023
4.0      0.066743
5.0      0.103289
6.0      0.151293
           ...
124.0    1.000000
129.0    1.000000
135.0    3.000000
150.0    2.000000
151.0    0.000000
Name: mort_acc, Length: 118, dtype: float64
```

Let's fill in the missing mort_acc values based on their total_acc value. If the mort_acc is missing, then we will fill in that missing value with the mean value corresponding to its total_acc value from the Series we created above. This involves using an .apply() method with two columns. Check out the link below for more info, or review the solutions video/notebook.
Helpful Link
(https://stackoverflow.com/questions/13331698/how-to-apply-a-function-to-two-columns-of-pandas-dataframe)

```
# Calculamos las medias de mort_acc agrupadas por total_acc una sola vez
mort_acc_mean_per_total_acc =
df.dropna(subset=['mort_acc']).groupby('total_acc')['mort_acc'].mean()

def impute_mort_acc(cols):
    Mort_Acc = cols[0] #le pasamos de argumento a la funcion estas 2 columnas
    Total_Acc = cols[1] #le pasamos de argumento a la funcion estas 2 columnas
    if pd.isnull(Mort_Acc):
        return mort_acc_mean_per_total_acc.get(Total_Acc, np.nan)
    else:
        return Mort_Acc
#Now apply that function!
df['mort_acc'] = df[['mort_acc','total_acc']].apply(impute_mort_acc,axis=1)

df.isnull().sum()
```

```
loan_amnt                0
term                     0
int_rate                 0
installment              0
grade                    0
sub_grade                0
home_ownership           0
annual_inc               0
verification_status      0
issue_d                  0
loan_status              0
purpose                  0
dti                      0
earliest_cr_line         0
open_acc                 0
pub_rec                  0
revol_bal                0
revol_util             276
total_acc                0
initial_list_status      0
application_type         0
mort_acc                 0
pub_rec_bankruptcies   535
address                  0
loan_repaid              0
dtype: int64
```

revol_util and the pub_rec_bankruptcies have missing data points, but they account for less than 0.5% of the total data. Go ahead and remove the rows that are missing those values in those columns with dropna().
df = df.dropna()
df.isnull().sum()

```
loan_amnt              0
term                   0
int_rate               0
installment            0
grade                  0
sub_grade              0
home_ownership         0
annual_inc             0
verification_status    0
issue_d                0
loan_status            0
purpose                0
dti                    0
earliest_cr_line       0
open_acc               0
pub_rec                0
revol_bal              0
revol_util             0
total_acc              0
initial_list_status    0
application_type       0
mort_acc               0
pub_rec_bankruptcies   0
address                0
loan_repaid            0
dtype: int64
```

Categorical Variables and Dummy Variables
We're done working with the missing data! Now we just need to deal with the string values due to the categorical columns.
List all the columns that are currently non-numeric. Helpful Link:
https://stackoverflow.com/questions/22470690/get-list-of-pandas-dataframe-columns-based-on-data-type

Another very useful method call:
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.select_dtypes.html

df.select_dtypes(['object']).columns

```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'issue_d', 'loan_status', 'purpose', 'earliest_cr_line',
       'initial_list_status', 'application_type', 'address'],
      dtype='object')
```

term feature
Convert the term feature into either a 36 or 60 integer numeric data type using .apply() or .map().**
df['term'].value_counts()
Or just use .map()
df['term'] = df['term'].apply(lambda term: int(term[:3]))

```
term
36    301247
60     93972
Name: count, dtype: int64
```

grade feature
We already know grade is part of sub_grade, so just drop the grade feature.
df=df.drop('grade',axis=1)

Convert the subgrade into dummy variables. Then concatenate these new columns to the original dataframe. Remember to drop the original subgrade column and to add drop_first=True to your get_dummies call.**
subgrade_dummies = pd.get_dummies(df['sub_grade'],drop_first=True) #drop_first elimina la 1er columna para q no sea redundante
df = pd.concat([df,subgrade_dummies],axis=1) #al dataframe de train le agrega lo que definimos como subgrade_dummies
df=df.drop('sub_grade',axis=1) #elimino la columna de sub_grade original
df.columns

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'home_ownership',
       'annual_inc', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'application_type', 'mort_acc', 'pub_rec_bankruptcies', 'address',
       'loan_repaid', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5',
       'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5', 'E1', 'E2',
       'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4',
       'G5'],
      dtype='object')
```

verification_status, application_type,initial_list_status,purpose
Convert these columns: ['verification_status', 'application_type','initial_list_status','purpose'] into dummy variables and concatenate them with the original dataframe. Remember to set drop_first=True and to drop the original columns.
dummies = pd.get_dummies(df[['verification_status', 'application_type','initial_list_status','purpose']],drop_first=True)
df = df.drop(['verification_status', 'application_type','initial_list_status','purpose'],axis=1)
df = pd.concat([df,dummies],axis=1)

home_ownership
Review the value_counts for the home_ownership column.
df['home_ownership'].value_counts()

Convert these to dummy variables, but replace (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html) NONE and ANY with OTHER, so that we end up with just 4 categories, MORTGAGE, RENT, OWN, OTHER. Then concatenate them with the original dataframe. Remember to set drop_first=True and to drop the original columns.
df['home_ownership']=df['home_ownership'].replace(['NONE', 'ANY'],'OTHER')

homeown_dummies = pd.get_dummies(df['home_ownership'],drop_first=True) #drop_first elimina la 1er columna para q no sea redundante
df = pd.concat([df,homeown_dummies],axis=1)
df=df.drop('home_ownership',axis=1) #elimino la columna de sub_grade original

address
Let's feature engineer a zip code column from the address in the data set. Create a column called 'zip_code' that extracts the zip code from the address column.
df['address'].head()

```
0       0174 Michelle Gateway\nMendozaberg, OK 22690
1    1076 Carney Fort Apt. 347\nLoganmouth, SD 05113
2    87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113
3             823 Reid Ford\nDelacruzside, MA 00813
4              679 Luna Roads\nGreggshire, VA 11650
Name: address, dtype: object
```

df['zip_code'] = df['address'].apply(lambda address:address[-5:]) #extrae los últimos cinco caracteres de cada valor en esa columna

Now make this zip_code column into dummy variables using pandas. Concatenate the result and drop the original zip_code column along with dropping the address column.
zp_dummies = pd.get_dummies(df['zip_code'],drop_first=True) #drop_first elimina la 1er columna para q no sea redundante

```
df = pd.concat([df,zp_dummies],axis=1)
df=df.drop('zip_code',axis=1)
df=df.drop('address',axis=1)
```

issue_d

This would be data leakage, we wouldn't know beforehand whether or not a loan would be issued when using our model, so in theory we wouldn't have an issue_date, drop this feature.

```
df=df.drop('issue_d',axis=1)
```

earliest_cr_line

This appears to be a historical time stamp feature. Extract the year from this feature using a .apply function, then convert it to a numeric feature. Set this new data to a feature column called 'earliest_cr_year'.Then drop the earliest_cr_line feature.

```
df['earliest_cr_line'].head()
```

```
0    Jun-1990
1    Jul-2004
2    Aug-2007
3    Sep-2006
4    Mar-1999
Name: earliest_cr_line, dtype: object
```

```
#Esta forma está mal xq lo guarda como string: df['earliest_cr_year']  =
df['earliest_cr_line'].apply(lambda time:time[-4:]) #extrae los últimos 4 caracteres de cada valor en esa columna
df['earliest_cr_year'] = df['earliest_cr_line'].apply(lambda date:int(date[-4:]))
df['earliest_cr_year']

df = df.drop('earliest_cr_line',axis=1)
df.select_dtypes(['object']).columns
        Index(['loan_status'], dtype='object')
```

Train Test Split

Import train_test_split from sklearn.

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

Drop the load_status column we created earlier, since its a duplicate of the loan_repaid column. We'll use the loan_repaid column since its already in 0s and 1s.

```
df = df.drop('loan_status',axis=1)
```

Set X and y variables to the .values of the features and label.

```
X = df.drop('loan_repaid',axis=1).values
y = df['loan_repaid'].values
```

OPTIONAL
Grabbing a Sample for Training Time
OPTIONAL: Use .sample() to grab a sample of the 490k+ entries to save time on training. Highly recommended for lower RAM computers or if you are not using GPU.
# df = df.sample(frac=0.1,random_state=101)
print(len(df))
395219

Perform a train/test split with test_size=0.2 and a random_state of 101.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)

Normalizing the Data
Use a MinMaxScaler to normalize the feature data X_train and X_test. Recall we don't want data leakge from the test set so we only fit on the X_train data.
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Notice to prevent data leakage from the test set, we only fit our scaler to the training set
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

Creating the Model
Run the cell below to import the necessary Keras functions.
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout

Build a sequential model to will be trained on the data. You have unlimited options here, but here is what the solution uses: a model that goes 78 --> 39 --> 19--> 1 output neuron.
OPTIONAL: Explore adding Dropout layers (https://keras.io/layers/core/)

1. (https://en.wikipedia.org/wiki/Dropout_(neural_networks))
2. (https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab)

#model = Sequential()
Choose whatever number of layers/neurons you want.
#https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
Remember to compile()

model = Sequential()
#https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
input layer

```
model.add(Dense(78,  activation='relu'))
model.add(Dropout(0.2))
# hidden layer
model.add(Dense(39, activation='relu'))
model.add(Dropout(0.2))
# hidden layer
model.add(Dense(19, activation='relu'))
model.add(Dropout(0.2))
# output layer
model.add(Dense(units=1,activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam')
```

Fit the model to the training data for at least 25 epochs. Also add in the validation data for later plotting. Optional: add in a batch_size of 256.

```
model.fit(x=X_train,
        y=y_train,
        epochs=25,
        batch_size=256,
        validation_data=(X_test, y_test),
        )
```

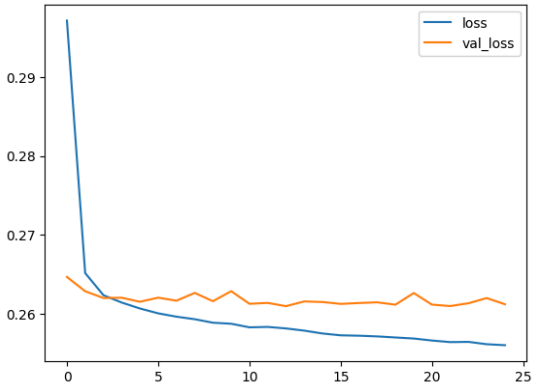
OPTIONAL: Save your model.

```
from tensorflow.keras.models import load_model
model.save('full_data_project_model.h5')
```

Section 3: Evaluating Model Performance.

Plot out the validation loss versus the training loss.

```
model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
#Otra forma
# losses = pd.DataFrame(model.history.history)
# losses[['loss','val_loss']].plot()
```



Create predictions from the X_test set and display a classification report and confusion matrix for the X_test set.

```
predictions = model.predict(X_test)
pred_classes = (predictions >0.5).astype(int)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,pred_classes))
```

	precision	recall	f1-score	support
0	0.99	0.43	0.60	15658
1	0.88	1.00	0.93	63386
accuracy			0.89	79044
macro avg	0.94	0.72	0.77	79044
weighted avg	0.90	0.89	0.87	79044

```
print(confusion_matrix(y_test,pred_classes))
[[ 6789  8869]
 [   48 63338]]
```

Given the customer below, would you offer this person a loan?

```
import random
random.seed(101)
random_ind = random.randint(0,len(df))
new_customer = df.drop('loan_repaid',axis=1).iloc[random_ind]
new_customer
```

loan_amnt 25000.0
term 60
int_rate 18.24
installment 638.11
annual_inc 61665.0
...
48052 False
70466 False
86630 False
93700 False
earliest_cr_year 1996
Name: 305323, Length: 78, dtype: object

```
#model.predict(new_customer.values.reshape(1,78))
new_customer = (scaler.transform(np.array(new_customer).reshape(1,78).astype("float32")>
0.5).astype("int32"))
predictions = (model.predict(new_customer))
```

array([[1.]], dtype=float32)

Now check, did this person actually end up paying back their loan?

```
df.iloc[random_ind]['loan_repaid']
```

1

Tensorboard

Tensorboard es una herramienta de visualización de Google diseñada para funcionar junto con TensorFlow para visualizar varios aspectos de su modelo.
Aquí simplemente entenderemos cómo ver el panel de Tensorboard en nuestro navegador y analizar un modelo existente.
NOTA: ¡Esta lección requiere que comprenda las rutas de los archivos y la ubicación de su computadora portátil o archivo .py!
Tenga en cuenta que Tensorboard es una biblioteca independiente de TensorFlow.
Los usuarios de Google Collab pueden seguir la guía oficial de Google y el notebook prefabricado: https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks

Let's explore the built in data visualization capabilities that come with Tensorboard. Full official tutorial available here: https://www.tensorflow.org/tensorboard/get_started

Data

```
import pandas as pd
import numpy as np
df = pd.read_csv('../DATA/cancer_classification.csv')
```

Train Test Split

```
X = df.drop('benign_0__mal_1',axis=1).values
y = df['benign_0__mal_1'].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=101)
```

Scaling Data

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Creating the Model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout

from tensorflow.keras.callbacks import EarlyStopping,TensorBoard

early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)

pwd
```

'/workspaces/curso-udemy-data-science-python/TensorFlow_FILES/ANNs'

Creating the Tensorboard Callback
TensorBoard is a visualization tool provided with TensorFlow. This callback logs events for TensorBoard, including:

- Metrics summary plots
- Training graph visualization
- Activation histograms
- Sampled profiling

If you have installed TensorFlow with pip, you should be able to launch TensorBoard from the command line: ```**sh tensorboard --logdir=path_to_your_logs**```

You can find more information about TensorBoard here: https://www.tensorflow.org/tensorboard/.

Arguments:

- log_dir: the path of the directory where to save the log files to be parsed by TensorBoard.
- histogram_freq: frequency (in epochs) at which to compute activation and weight histograms for the layers of the model. If set to 0, histograms won't be computed. Validation data (or split) must be specified for histogram visualizations.
- write_graph: whether to visualize the graph in TensorBoard. The log file can become quite large when write_graph is set to True.
- write_images: whether to write model weights to visualize as image in TensorBoard.
- update_freq: `'batch'` or `'epoch'` or integer. When using `'batch'`, writes the losses and metrics to TensorBoard after each batch. The same applies for `'epoch'`. If using an integer, let's say `1000`, the callback will write the metrics and losses to TensorBoard every 1000 samples. Note that writing too frequently to TensorBoard can slow down your training.
- profile_batch: Profile the batch to sample compute characteristics. By default, it will profile the second batch. Set profile_batch=0 to disable profiling. Must run in TensorFlow eager mode.
- embeddings_freq: frequency (in epochs) at which embedding layers will be visualized. If set to 0, embeddings won't be visualized.

```
log_directory = 'logs\fit'
# WINDOWS: Use "logs\\fit"
# MACOS/LINUX: Use "logs\fit"

# OPTIONAL: ADD A TIMESTAMP FOR UNIQUE FOLDER
# timestamp = datetime.now().strftime("%Y-%m-%d--%H%M")
# log_directory = log_directory + '\\' + timestamp

board = TensorBoard(log_dir=log_directory,histogram_freq=1,
```

```
    write_graph=True,
    write_images=True,
    update_freq='epoch',
    profile_batch=2,
    embeddings_freq=1)

#Now create the model layers:
model = Sequential()
model.add(Dense(units=30,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=15,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')



## Train the Model
model.fit(x=X_train,
          y=y_train,
          epochs=600,
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop,board]
          )

## Running Tensorboard: Running through the Command Line
**Watch video to see how to run Tensorboard through a command line call.**
Tensorboard will run locally in your browser at [http://localhost:6006/] (http://localhost:6006/)
print(log_directory)
          logsit

pwd
          '/workspaces/curso-udemy-data-science-python/TensorFlow_FILES/ANNs'

### Use cd at your command line to change directory to the file path reported back by pwd
or your current .py file location.
### Then run this code at your command line or terminal
tensorboard --logdir logsit

https://cuddly-orbit-wgpv994974qf57wr-6006.app.github.dev/
```

Section 26: Big Data and Spark with Python

Big Data Overview

- Explanation of Hadoop, MapReduce, Spark, and PySpark
- Local vs. Distributed Systems
- Overview of Hadoop Ecosystem
- Datailed overview of Spark
- Set-up on Amazon Web Services
- Resources on other Spark Options
- Jupyter Notebook hands-on code with PySpark and RDDs

Hablemos de Big Data en general. Hasta ahora hemos trabajado con datos que pueden caber en una computadora local y que generalmente están en la escala de entre 0 y 8 GB. Estos son datos que pueden caber directamente en la RAM, pero ¿qué podemos hacer si tenemos un conjunto de datos más grande?

1. Podría intentar usar algún tipo de base de datos SQL para mover el almacenamiento a un disco duro en lugar de la RAM, eso funcionará en una computadora local.
2. Sin embargo, si necesitas expandirte más allá de lo que tu máquina local puede manejar, puedes utilizar lo que se conoce como un sistema distribuido (distributed system), que distribuye los datos entre múltiples máquinas o computadoras.

Solo para reiterar, una máquina local es básicamente lo que hemos estado usando: tu propia computadora o una sola máquina, donde estás restringido a la memoria RAM o al disco duro de esa máquina. En un **sistema distribuido** se puede hacer que una máquina controle la distribución de varias máquinas. Por ejemplo, si tienes una máquina local, puedes usar todos los núcleos en esa máquina. Sin embargo, si aprovechas el poder de un sistema distribuido, puedes tener un nodo maestro más pequeño que controle un sistema distribuido de computadoras, lo que te permite contar con mayor poder de procesamiento y más memoria de la que podrías tener en una única máquina local.

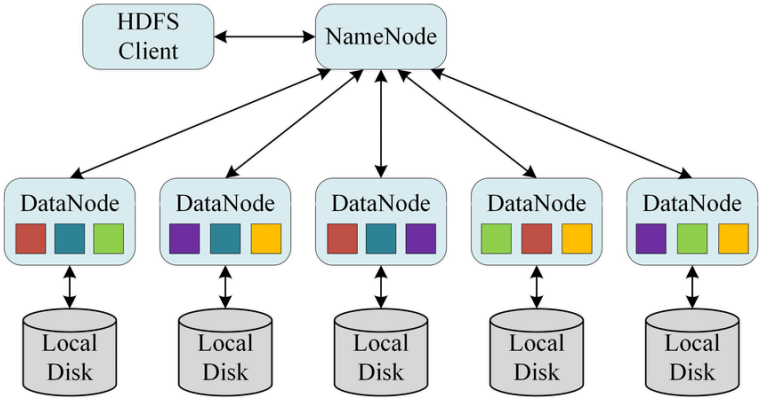
- Como mencioné anteriormente, un proceso local usará los recursos computacionales de una sola máquina, mientras que un proceso distribuido tendrá acceso a los recursos computacionales de varias máquinas conectadas a través de una red.
- Después de cierto punto es más fácil escalar hacia muchas máquinas de menor rendimiento de CPU en lugar de tratar de escalar a una sola máquina con alto rendimiento. Las máquinas distribuidas también tienen la ventaja de ser escalables con facilidad: simplemente se pueden agregar más máquinas, a diferencia de una máquina local única, donde es mucho más complicado actualizar la CPU y eventualmente se llega a un límite.
- Las máquinas distribuidas también pueden incluir algo conocido como tolerancia a fallos, que básicamente significa que si una máquina falla, toda la red puede continuar funcionando. En cambio, si una máquina local falla, todo el trabajo se detiene.

Ahora, avancemos y discutamos el formato típico de una **arquitectura distribuida que utiliza Hadoop** como una visión general básica. Hadoop es esencialmente una forma de distribuir

archivos muy grandes a través de múltiples máquinas. Utiliza lo que se conoce como el sistema de archivos distribuido de Hadoop (HDFS).

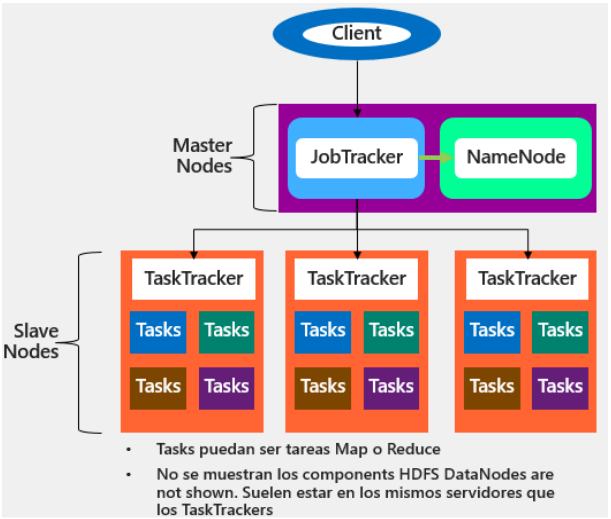
- **HDFS** permite al usuario trabajar con conjuntos de datos grandes y también duplica bloques de datos para asegurar la tolerancia a fallos.
- Luego puede utilizar lo que se conoce como **MapReduce**, que permite realizar cálculos sobre esos datos.

Vamos a ver cómo se vería un sistema de almacenamiento distribuido o **HDFS**. Tendrías un nodo principal (NameNode) con CPU y memoria RAM asignadas, y luego varias otras máquinas distribuidas como Data Nodes.



- HDFS utilizará bloques de datos, normalmente de un tamaño de 128 megabytes por defecto.
- Cada uno de estos bloques se replica 3 veces.
- Los bloques se distribuyen de forma que admitan la tolerancia a fallos, lo que significa que si uno de estos nodos de datos menores falla por alguna razón, tu información estará replicada en otros nodos, evitando así la pérdida de datos.
- Bloques más pequeños permiten una mayor paralelización durante el procesamiento.
- Las copias múltiples de un bloque evitan la pérdida de datos ante la falla de un nodo.

Ahora hablemos de **MapReduce**. MapReduce es una forma de dividir una tarea de cálculo en un conjunto de archivos distribuidos, como en HDFS.



- Consiste en un rastreador de trabajos (job tracker) y múltiples rastreadores de tareas (task trackers).
- El job tracker envía el código para ejecutarse en los task trackers, los cuales asignan CPU y memoria para las tareas y supervisan los procesos en los nodos de trabajo.

Lo que hemos cubierto se puede dividir en dos partes: usar HDFS para distribuir conjuntos de datos grandes y luego usar MapReduce para distribuir una tarea de cálculo/computacional en un conjunto de datos distribuido.

Spark Overview

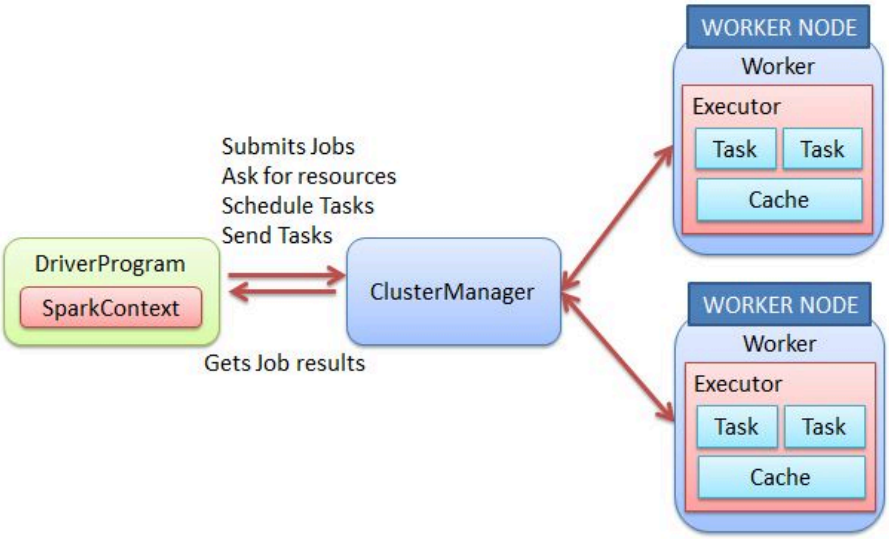
Spark es una de las últimas tecnologías utilizadas para manejar Big Data de manera rápida y sencilla. Es un proyecto de código abierto de Apache, lanzado en febrero de 2013, y ha ganado popularidad debido a su facilidad de uso y velocidad. Fue creado alrededor de 2009 en el AMPLab de UC Berkeley, así que es una tecnología relativamente nueva, pero ha sido adoptada por muchos usuarios.

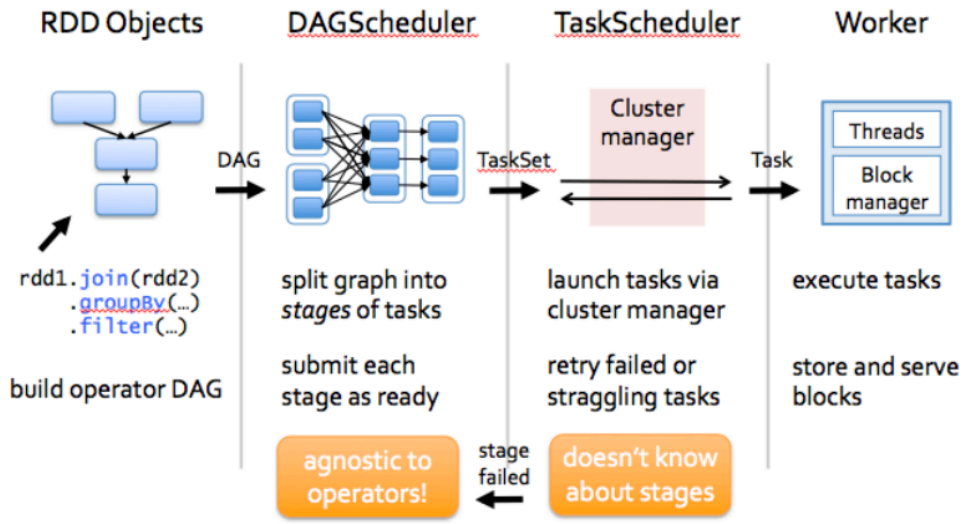
Puedes ver a **Spark** como una **alternativa flexible a MapReduce**. Spark puede ejecutarse sobre la infraestructura de Hadoop Distributed File System (HDFS) para ofrecer funcionalidades adicionales, pero también puede usar datos almacenados en varios formatos, como Cassandra, Amazon S3, HDFS, y otros.

Spark es una alternativa a MapReduce, no un reemplazo de Hadoop. No está destinado a reemplazar Hadoop, sino a ofrecer una solución unificada para manejar distintos casos y requisitos de Big Data. A diferencia de MapReduce, **Spark no necesita que los archivos estén específicamente almacenados en HDFS;** puede trabajar directamente en la memoria, lo que le permite ser hasta **100 veces más rápido** que MapReduce, que escribe datos en el disco después de cada operación.

En el núcleo de Spark está la idea de los RDDs. Los **RDDs** tienen 4 características principales: son colecciones distribuidas de datos, son tolerantes a fallos, soportan operaciones en paralelo y pueden trabajar con diversas fuentes de datos.

Como se mencionó en la clase anterior sobre sistemas distribuidos, Spark opera de la misma manera: tienes un programa principal que maneja el Spark Context, el cual comunica con un Cluster Manager que, a su vez, se comunica con los nodos de trabajo que ejecutan las tareas.





Tienes el planificador DAG (Directed Acyclic Graph, o grafo acíclico dirigido), y luego tienes un planificador de tareas, y finalmente están los nodos de trabajo propiamente dichos. Spark admite el uso compartido de datos en memoria entre estos DAG, lo que permite que diferentes trabajos operen sobre los mismos datos.

Spark permite a los programadores desarrollar flujos de datos complejos usando grafos dirigidos acíclicos (DAG) y también admite el uso compartido de datos en memoria entre estos grafos.

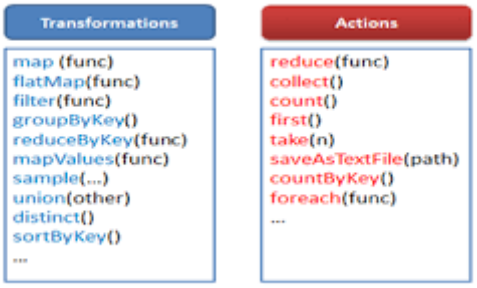
Para nuestro trabajo con Spark en Python, nos centraremos principalmente en los objetos RDD. Los **RDDs** son inmutables, evaluables en paralelo y almacenables en caché. Hay 2 tipos de operaciones RDD: transformaciones y acciones. Algunas de las **acciones** básicas incluyen collect, count, first y take.

- Collect: Return all the elements of the RDD as an array at the driver program
- Count: Return the number of elements in the RDD
- First: Return the first element in the RDD

- Take: Return an array with the first n elements of the RDD

En cuanto a las **transformaciones**, encontraremos operaciones como filter, map y flatMap.

- RDD.filter(): Applies a function to each element and returns elements that evaluate to true
- RDD.map(): Transforms each element and preserves # of elements, very similar idea to pandas .apply() ➜ ex. grabbing first letter of a list of names
- RDD.flatMap(): Transforms each element into 0-N elements and changes # of elements ➜ ex. transforming a corpus of text into a list of words



Generalmente, los RDD almacenarán sus valores en **tuplas**: (key, value). Esto generalmente permite una mejor partición de los datos y da lugar a funcionalidades basadas en una reducción.

Tenemos 2 **métodos** adicionales: reduce y reduceByKey

- Reduce(): Realizará una acción que agrupará elementos del RDD utilizando una función que devuelve un único elemento.
- ReduceByKey(): Es una acción que agrupará elementos de par RDD utilizando una función que devuelve un par RDD.

Estas ideas son similares a una operación groupBy.

Un último apunte: SPARK ha evolucionado continuamente y las nuevas versiones se lanzan con bastante frecuencia. El ecosistema de Spark ahora incluye herramientas como Spark SQL, Spark DataFrames, MLlib, GraphX y Spark Streaming. Así que el ecosistema de Spark se está expandiendo muy rápidamente, y hay muchas novedades interesantes en él.

Ahora que hemos aprendido lo suficiente para comenzar, lo que vamos a hacer es mostrarles cómo configurar una cuenta en Amazon Web Services para ejecutar Spark en la web. También contamos con un artículo en texto que ofrece otras opciones en caso de que no quieran usar Amazon Web Services y prefieran utilizar Spark en su computadora local.

La razón por la que les vamos a enseñar a configurar todo en Amazon Web Services es porque la idea de Spark es que ya no pueden usar una máquina local; necesitan un conjunto distribuido de máquinas o algún tipo de servicio en la nube, y Amazon Web Services es el servicio en la nube más popular.

Si solo están usando Spark en una computadora local, entonces podrían simplemente usar algo como SQL o incluso Pandas, como ya lo hemos hecho a lo largo del curso.

Bien, a continuación, les ofreceremos un artículo en texto con otras opciones en caso de que no quieran usar Amazon Web Services. Después de eso, retomaremos las lecciones en video para mostrarles cómo configurar una cuenta y hacer que Spark funcione. Es un proceso bastante complicado, así que asegúrense de seguir paso a paso para configurarlo correctamente.

Local Spark Set-Up

Esta lección es solo para personas que no quieren usar Amazon Web Services. Sáltala si planeas usar la versión gratuita de AWS y sigue las instrucciones en la próxima lección. Recomiendo encarecidamente que aprendas a configurar Spark en AWS, ya que es una habilidad extremadamente valiosa, especialmente para potenciales empleadores.

Opciones para Configuración Local de Spark: Es muy recomendable que configures Spark en Amazon Web Services, ya que ese es el tipo de operación que realizarías en un entorno real (no tiene mucho sentido instalar Spark en una computadora local, ya que el objetivo de Spark es manejar datos demasiado grandes para una sola computadora).

Sin embargo, si aún deseas instalarlo localmente, la mejor manera es instalar Ubuntu en tu sistema:

- Instrucciones para Windows: http://www.ubuntu.com/download/desktop/create-a-usb-stick-on-windows
- Instrucciones para Mac OS: http://www.ubuntu.com/download/desktop/create-a-usb-stick-on-mac-osx

Una vez que tengas Ubuntu instalado en tu computadora local, puedes seguir las instrucciones en la lección titulada "Configuración de PySpark", que está a 3 lecciones de esta. Allí se explica cómo instalar Spark y Hadoop en una computadora con Ubuntu. Nuevamente, no recomiendo que lo instales localmente; es una mejor práctica, y mejora tu currículum, entender cómo instalar Spark en una instancia EC2 en Amazon Web Services.

AWS Account Set-Up

https://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/AboutAWSAccounts.html

1. Go to: https://aws.amazon.com/es/free ➜ Create Free Account
2. Create an EC2 Instance

Nota rápida sobre la seguridad en AWS: Cuando configuramos la instancia EC2 y los grupos de seguridad para estas instancias spot, mantenemos todos los puertos abiertos para simplificar el proceso. Sin embargo, es importante señalar que estos ajustes deberían ser mucho más estrictos si se utilizaran en producción. Así que tenlo en cuenta si deseas aplicar los conceptos discutidos en la próxima lección para un entorno de producción formal.

EC2 Instance Set-Up

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Ahora que hemos configurado nuestra cuenta de Amazon Web Services, podemos proceder a crear una instancia EC2. **Amazon Elastic Compute Cloud, o EC2,** es un servicio web que ofrece capacidad de cómputo escalable en la nube. En términos simples, pueden pensar en esto como una **computadora virtual** a la que pueden acceder a través de Internet.

1. Usaremos el sitio web de la consola de AWS para crear una instancia EC2.
2. Usaremos SSH, o Secure Shell, para conectarnos a la instancia EC2 por Internet. Sin embargo, deben tener en cuenta que el uso de SSH es un poco diferente para Windows en comparación con Mac y Linux.
3. Una vez conectados, configuraremos Spark y Jupyter en la instancia EC2.

Hablemos un poco de SSH (Secure Shell Connection). Si están en una computadora con Windows, deberán ver toda esta clase. Les mostraremos cómo configurar la instancia EC2 y luego cómo conectarse a través de SSH en Windows. Si están en Mac o Linux, verán la primera mitad de esta clase, donde configuramos EC2. Luego, les avisaré cuándo pueden dejar de ver y saltar a la siguiente clase o la configuración de SSH para Mac o Linux. Mac y Linux tienen un proceso mucho más sencillo, ya que SSH ya está integrado en el sistema operativo, así que solo se necesitan dos comandos. En Windows, es un poco más complicado, por lo que mostraremos todo en esta clase y les avisaré cuándo pueden dejar de ver si están en Mac o Linux.

Nuestro objetivo general es simplemente conectarnos de forma remota a la línea de comandos de nuestra máquina virtual en Amazon. Así que, independientemente del sistema operativo que estén usando, vayan a AWS Console en aws.amazon.com e inicien sesión.

Go to: https://aws.amazon.com/es/free ➜ sign in to the console ➜ EC2 ➜ launch instance ➜ Ubuntu (asegurarse que diga free tier elegible) ➜ t2.micro, que es la más pequeña, gratuita, con un CPU y 1 gb de memoria ➜ Next: Configure instance details, en "Number of instances" dejar 1 (si se tuviera un conjunto de datos enorme podrían utilizar varias instancias, pero ahora solo necesitamos 1) ➜ Next: Add Storage, dejar los valores predeterminados (8 GB) ➜ Next: Tag instance, en Key completar con algo como “myspark” y en Value “mymachine” ➜ Next: Configure Security Group, en Type cambiar a "All traffic" y dejar los otros valores predeterminados ➜ Review and Launch ➜ Launch. Después en el cartel que aparece de “Select an existing key pair or create a new key pair” seleccionar "Create a new key pair", en Key pair name escribir “newspark” y hacer click en “Download Key Pair” ➜ descargar el archivo .pem (este archivo será necesario y no se puede volver a descargar) ➜ Launch Instances; se puede hacer click en el enlace del codigo de la instancia y ver el estado "running" de la instancia en la consola.

Desde la consola, en Actions ➜ Instance State ➜ Terminate: Hacer click acá cuando termines con la instancia para asegurarte de que no te cobren extra.

Ahora, para Mac y Linux, hemos terminado la configuración. Solo deben seguir los pasos para lanzar la instancia y descargar el archivo .pem. Vayan a la siguiente clase para ver la conexión SSH en Mac o Linux.

Para usuarios de Windows, sigan viendo: ahora mostraremos cómo conectarse a esta instancia desde la línea de comandos. Vayan a Google y busquen "SSH Windows EC2". Seleccionen el enlace que explica cómo conectarse usando PuTTY y manténganlo abierto para referencia.

Los usuarios de Windows necesitan instalar PuTTY para conectarse a EC2. Descarguen putty.exe y puttygen.exe, que les permitirán generar las claves necesarias. Luego, van a necesitar el ID de la instancia, eso sale de la consola de AWS junto con el public DNS.

Luego se necesita convertir la private key usando PuTTYgen, para eso start puttygen.exe ➜ en “Type of key to generate” seleccionar SSH-2 RSA ➜ click “Load” y cargar el archivo .pem ➜ “Save private key”

Starting a PuTTY Session, para eso start putty.exe ➜ PuTTY Configuration

- Category: Session ➜ In the Host Name box, to connect using your instance's public DNS name, enter instance-user-name@instance-public-dns-name , o sea ubuntu@dirección_DNS en este caso ➜ Para terminar ensure that the Port value is 22. Get the default username for the AMI that you used to launch your instance:
    - Amazon Linux – ec2-user
    - CentOS – centos or ec2-user
    - Debian – admin
    - Fedora – fedora or ec2-user
    - RHEL – ec2-user or root
    - SUSE – ec2-user or root
    - **Ubuntu – ubuntu**
    - Oracle – ec2-user
    - Bitnami – bitnami
    - Rocky Linux – rocky
    - Other – Check with the AMI provider
- Category: Connection ➜ select SSH ➜ select Auth ➜ click en “Browse” ➜ seleccionar el archivo .ppk (el que transformamos desde el .pem) ➜ click en “Open”

Estarán conectados a su servidor virtual, donde podrán ejecutar comandos como "python" para abrir el intérprete de Python. Aquí tienen acceso directo a la máquina virtual en Amazon desde la línea de comandos.

SSH with Mac and Linux

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connect-linux-inst-ssh.html

Para usuarios de Linux y Mac que solo vieron la primera mitad de la clase anterior, ya hemos creado nuestra instancia EC2 usando la consola de AWS. También deberías haber descargado ese archivo de clave .pem.
Ahora lo que vamos a hacer es conectarnos a nuestra instancia a través de la terminal usando SSH (protocolo de conexión segura).

Asegúrate de poder localizar tu archivo .epm. Te recomendaría moverlo desde donde lo guardaste a tu escritorio para que puedas seguir exactamente lo que estoy haciendo y también asegúrate de tener la dirección DNS de tu instancia EC2; que también puede obtenerse de la consola de AWS.

Vamos a empezar abriendo nuestra terminal:

1. cd Desktop
2. chmod 400 jupyterone.pem
3. ssh -i jupyterone.pem ubuntu@direcciónDNS

Te aparecerá una advertencia que dice "autenticidad". Responde "yes" y presiona Enter de nuevo; deberías estar conectado. Verás algo parecido a "Ubuntu en una dirección IP". Lo que estás viendo ahora es la interfaz de línea de comandos de tu instancia EC2, y estás conectado virtualmente a esa computadora desde la línea de comandos.

Para verificar que todo esté funcionando, puedes ejecutar el comando “python” aquí, y debería funcionar. Probablemente aún no verás Anaconda porque aún necesitamos instalarlo (ya lo tengo instalado desde antes de grabar este video). Pero deberías al menos ver la versión de Python 2.7 en Ubuntu; puedes probar con un print("Hola") y debería ejecutarse. Puedes salir de esto cuando quieras con quit().

PySpark SetUp

https://medium.com/@josemarcialportilla/getting-spark-python-and-jupyter-notebook-running-on-amazon-ec2-dec599e1c297

Vamos a configurar Spark, Hadoop y Jupyter Notebook en nuestra instancia virtual EC2 que fue creada en clases anteriores.

1. Ingresa o haz una conexión SSH a esa instancia desde la terminal de comandos en Ubuntu. Si usas Windows, utiliza Putty, y en Mac o Linux, simplemente usa el comando SSH en la terminal.
2. Instalar Anaconda en esta máquina virtual. Para hacerlo, introduce: wget http://repo.continuum.io/archive/Anaconda3-4.1.1-Linux-x86_64.sh
3. Escribir en la línea de comandos: bash Anaconda 3-4.1.1-Linux-x86_64.sh
4. Cuando termine la descarga, verás un aviso para revisar el acuerdo de licencia de Anaconda. Presiona Enter varias veces para ver todo el acuerdo hasta que te solicite una respuesta, donde deberás escribir "yes" para aceptar y confirmar la instalación en la ubicación predeterminada /home/ubuntu/anaconda3.

5. Para verificar que todo esté bien, usa el comando “which python” y asegúrate de que estás utilizando la versión de Anaconda de Python. Si ves otra ubicación, ejecuta source ~/.bashrc, luego verifica nuevamente.
6. Ahora vamos a configurar Jupyter Notebook. Generaremos un archivo de configuración para Jupyter corriendo: jupyter notebook --generate-config
7. Crearemos una carpeta para los certificados corriendo: mkdir certs
8. Luego navegaremos a esa carpeta: cd certs
9. Generaremos los certificados necesarios con el comando: sudo openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem.
10. Editar el archivo de configuración de Jupyter que creamos anteriormente para que apunte al certificado generado y permita conexiones en todas las direcciones IP:: cd ~/.jupyter/
11. Correr: vi jupyter_notebook_config.py
12. Apretar i en el teclado para activar el modo edición. Then at the top of the file type:
    c=get_config()
    c.NotebookApp.certfile= u’/home/ubuntu/certs/mycert.pem’
    c.NotebookApp.ip = ‘*’
    c.NotebookApp.open_browser = False
    c.NotebookApp.port = 8888
    ((tecla escape))
    :wq! ((abajo de todo))
13. De nuevo en la terminal de comandos, para lanzar Jupyter Notebook, usa el comando: jupyter notebook
14. Volver a la consola de AWS y copiar de nuevo el DNS ; abrir una pestaña en google y poner https://direccDNS**:8888**
15. Si aparece un aviso de seguridad, haz clic en "Avanzado" y luego en "Continuar".
16. Luego, después de un rato, debería ver aparecer el sistema de su notebook Jupiter. Este es el notebook de jupyter al que se está conectando a través de Internet por Amazon, pero todo lo que haces aquí está sucediendo realmente en tu máquina virtual alojada en otro lugar geográficamente.
17. Tenemos el portátil Jupiter en funcionamiento. Debería poder hacer clic en New ➔ Python (route) para obtener nueva vida en una ruta y luego hacer todo lo que hemos hecho antes en las computadoras portátiles Jupiter.

Con Jupyter Notebook en ejecución, puedes crear notebooks nuevos y ejecutar comandos de Python. A continuación, vamos a instalar Spark.

1. En la terminal de Ubuntu, actualiza los paquetes con: sudo apt-get update.
2. Luego instala Java usando: sudo apt-get install default-jre
3. Verifica con: java -version.

Para instalar Scala

1. Usa: sudo apt-get install scala
2. Una vez instalado, verifica con: scala -version.

Finalmente, necesitamos la librería py4j para conectar Python y Java

1. Primero asegurémonos de que pip esté vinculado a la instalación de python Anaconda ejecutando: export PATH=$PATH:$HOME/anaconda3/bin
2. Luego instala pip con: conda install pip
3. Asegúrate de que which pip apunte a la instalación de Anaconda: which pip
4. Instala py4j ejecutando: pip install py4j

Muy bien, ya casi terminamos, solo necesitamos instalar Spark y Hadoop. Luego debemos indicarle a Python dónde encontrar Spark y reiniciaremos el notebook de Jupyter para confirmar que todo funcione correctamente.

1. Para instalar Spark y Hadoop, utiliza el comando wget y envíalo al servidor correcto: wget http://apache.mirrors.tds.net/spark/spark-2.0.0/spark-2.0.0-bin-hadoop2.7.tgz
2. El siguiente paso es simplemente descomprimirlo usando: sudo tar -zxvf spark-2.0.0-bin-hadoop2.7.tgz

Esto instalará Spark y Hadoop.

Finalmente, solo necesitamos decirle a Python dónde encontrar Spark

1. Indicar a Ubuntu dónde está el directorio de Spark: export SPARK_HOME='/home/ubuntu/spark-2.0-bin-hadoop2.7'
2. Luego, para definir el PATH de Spark, copia y pega la siguiente línea en tu consola de Ubuntu: export PATH=$SPARK_HOME:$PATH
3. export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
4. Ahora estamos listos para lanzar el notebook de Jupyter, que debería funcionar igual que la última vez: jupyter notebook
5. Voy a pasarme al navegador para comprobarlo, como antes: Necesitamos la dirección DNS pública. Cópiala y luego escribe https:// y pégala, seguido de :8888 y presiona Enter. Deberías ver el notebook abrirse de nuevo. Si esta es la segunda vez que haces esto, probablemente no recibirás una advertencia de seguridad.

Ahora, asegúrate de que todo funcione.

1. Abre un nuevo notebook en Python. Espera a que cargue, podría tomar un momento dependiendo de tu conexión a Internet.
2. Vamos a confirmar que funcionó. Escribe from pyspark import SparkContext. Si presionas Tab y aparecen opciones, significa que todo está funcionando correctamente.
3. Luego, escribe sc = SparkContext() y ejecútalo. Puede tardar un poco, pero mientras se ejecute, todo está bien.

Perfecto. Si todo esto funcionó, ya has terminado y estás listo para avanzar a la siguiente clase donde aprenderás a usar PySpark.

Solo una nota rápida: cuando termines con todas las clases de Spark y quieras apagar todo, ve a "Actions", selecciona "Instance State" y elige "Terminate". Recuerda que al terminar, ya no podrás acceder a nada guardado en esa instancia de EC2. Si quieres repetir esto en el futuro, tendrás que seguir todo el proceso de instalación de nuevo. Recibes 750 horas al mes

gratis durante 12 meses si eres nuevo en AWS, lo cual es bastante tiempo. Pero ten en cuenta que si superas los límites de la capa gratuita, comenzarán a cobrarte.

Lambda Expressions Review

Recuerda que una expresión lambda nos permite crear funciones anónimas, lo que básicamente significa que podemos crear rápidamente funciones ad hoc sin necesidad de definir una función adecuadamente usando def.

Para repasar cómo crear una función normal, diríamos algo como:

```
def square(num):
    result = num ** 2
    return result
```

Luego, si llamo a square con square(4), devuelve el cuadrado de ese número, que es 16.

Entonces, lo que no necesito hacer aquí es redefinir result. Puedo simplemente decir:

```
def square(num):
    return num ** 2
```

Estpfuncionará exactamente igual. Si llamo a square(4), sigo obteniendo 16.

Ahora, podemos escribir esto en una sola línea. Python lo permite, aunque es una mala práctica de estilo, pero funcionará. Ejecuta la celda y verás que devuelve el mismo resultado.

```
def square(num): return num ** 2
```

Ahora, lo que podemos hacer es tomar esta sintaxis y convertirla en una expresión lambda. Para convertir esto en una expresión lambda, haríamos algo así:

```
lambda num: num ** 2
```

Con lambda, omitimos def, no le asignamos un nombre a la función, pero conservamos el nombre de la variable y luego omitimos la palabra return.

Entonces, si queremos obtener el mismo resultado, podemos asignarla, aunque generalmente no asignamos una expresión lambda porque eso va en contra del propósito de las funciones lambda. Pero si decimos:

```
sq = lambda num: num ** 2
```

sq(5), eso devuelve el cuadrado, o sea 25.

Vamos a hacer un par de expresiones lambda más usando cadenas y números para tener una idea de cómo usarlas. Por ejemplo, si queremos verificar si un número es par, podríamos decir:

```
even = lambda num: num % 2 == 0
even(3)
    False
even(4)
    True
```

Para reiterar, usualmente no asignamos expresiones lambda; las usamos dentro de métodos. Veremos más sobre esto cuando tratemos con Spark y lambda. Pero te mostraré algunos ejemplos más.

Si quisiéramos tomar el primer carácter de una cadena, podríamos decir:
first = lambda s: s[0]
first('hello')
    h

Si quisiéramos revertir una cadena, podríamos hacer:
rev = lambda x: x[::-1]
rev('abcde')
    edcba

Un último ejemplo: al igual que con una función normal, podemos aceptar más de un parámetro en una expresión lambda. Por ejemplo:
def adder(x, y):
    return x + y

Luego,
adder(2,3)
    5

Podemos convertir esto en una expresión lambda como:
adderlam = lambda x, y: x + y
adderlam(2,3)
    5

Las expresiones lambda realmente brillan cuando se usan junto con algún tipo de método como map, filter o reduce, y estos métodos son transformaciones que tenemos en los RDDs de Spark.

Introduction to Spark and Python

Vamos a aprender cómo usar Python y Spark con la biblioteca PySpark desde mi notebook de Jupyter.

from pyspark import SparkContext
sc = SparkContext() #Debemos crear el Spark Context. Básicamente, el Spark Context representa la conexión con el clúster de Spark y se puede usar para crear RDDs y variables en broadcast en ese clúster. Ten en cuenta que generalmente solo se puede tener un Spark Context a la vez.

Ahora que tenemos el Spark Context, vamos a empezar con un ejemplo básico que consiste en leer un archivo de texto. Voy a usar comandos mágicos de Jupyter para escribir rápidamente un archivo de texto aquí:
%%writefile example.txt.
first line #contenido del txt
second line
third line
fourth line

Ahora podemos crear un RDD que contenga este archivo de texto mediante el método textFile de Spark Context, que lee archivos de texto desde HDFS, un sistema de archivos local o cualquier sistema de archivos compatible con Hadoop y lo devuelve como un RDD de strings:
text_rdd = sc.textFile("example.txt")

Podemos realizar acciones o transformaciones en un objeto RDD. Las acciones devuelven valores, y las transformaciones devuelven nuevos RDDs.

Empecemos con una acción básica:
text_rdd.count() #Spark contará el nro de elementos en el RDD (en este caso, c/ línea)
4
text_rdd.first() #nos da el primer elemento del RDD, nos devuelve la primera línea.
'first line'

Pasemos a transformaciones. Por ejemplo, podemos usar filter para buscar líneas que contengan la palabra "second". Para ello, escribimos:
filtered_rdd = text_rdd.filter(lambda line: "second" in line)

Esto no ejecuta inmediatamente la operación, sino que crea una "receta" de instrucciones que se aplicarán cuando realicemos una acción. Para ejecutar realmente las instrucciones, usamos una acción como collect:
filtered_rdd.collect()
"second line"

filtered_rdd.count()
1

Así que recuerda: puedes crear un Spark Context, usarlo para crear un RDD basado en un archivo o conjunto de datos, y luego realizar acciones o transformaciones. Las transformaciones crean una receta que no se ejecuta hasta que se llama a una acción.

RDD Transformations and Actions

Recuerden que un RDD es un conjunto de datos distribuido resiliente (resilient distributed dataset) y tenemos transformaciones y acciones. Una transformación es una operación de Spark que produce un RDD, mientras que una acción es una operación que produce un objeto local. Entonces, pueden pensar en una transformación como un conjunto de instrucciones, básicamente una receta a seguir, que producirá un RDD. Sin embargo, hasta que no llamen a una acción, no obtendrán un objeto local como resultado con el que puedan ver o trabajar. Finalmente, un Spark job es una secuencia de transformaciones sobre datos, seguida de una acción final. Los jobs de Spark son básicamente los RDD a los que se les aplica una transformación y luego una acción.

Important Terms

Let's quickly go over some important terms:

Term	Definition
RDD	Resilient Distributed Dataset
Transformation	Spark operation that produces an RDD
Action	Spark operation that produces a local object
Spark Job	Sequence of transformations on data with a final action

Hay algunas formas comunes de crear un RDD. Una es sc.textFile, donde sc es el contexto de Spark que vimos antes. Si desean crear un RDD de líneas desde un archivo, pueden usar sc.textFile. O si están trabajando con algún tipo de arreglo o lista, también pueden usar sc.parallelize.

Creating an RDD

There are two common ways to create an RDD:

Method	Result
`sc.parallelize(array)`	Create RDD of elements of array (or list)
`sc.textFile(path/to/file)`	Create RDD of lines from file

Luego tenemos las transformaciones. Algunas de los más comunes son **filter**, que descarta elementos que no devuelven True según la expresión lambda o función que pasen. También tienen **map**, con ejemplos como multiplicar un elemento del RDD por 2, o dividir cada cadena en palabras. Luego está **flatMap**, donde, a diferencia de map, devuelve un único arreglo de todos los elementos. También tienen **sample**, que crea una muestra con o sin reemplazo, especificando el porcentaje de muestra del RDD total. **Union** permite añadir un RDD a otro ya existente, **distinct** elimina duplicados, y **sortBy** ordena los elementos en orden ascendente o descendente según los parámetros.

RDD Transformations

We can use transformations to create a set of instructions we want to preform on the RDD (before we call an action and actually execute them).

Transformation Example	Result
`filter(lambda x: x % 2 == 0)`	Discard non-even elements
`map(lambda x: x * 2)`	Multiply each RDD element by `2`
`map(lambda x: x.split())`	Split each string into words
`flatMap(lambda x: x.split())`	Split each string into words and flatten sequence
`sample(withReplacement=True,0.25)`	Create sample of 25% of elements with replacement
`union(rdd)`	Append `rdd` to existing RDD
`distinct()`	Remove duplicates in RDD
`sortBy(lambda x: x, ascending=False)`	Sort elements in descending order

Una vez que tienen las transformaciones, el siguiente paso es ejecutar estas instrucciones mediante una acción. Algunas acciones comunes son **collect**, que convierte el RDD completo en una lista en memoria, **take**, que toma los primeros elementos del RDD, y **top**, que toma los elementos principales del RDD (útil después de ordenar). También tienen **takeSample**, **mean** y **stddev**, que calculan el promedio y la desviación estándar.

RDD Actions

Once you have your 'recipe' of transformations ready, what you will do next is execute them by calling an action. Here are some common actions:

Action	Result
`collect()`	Convert RDD to in-memory list
`take(3)`	First 3 elements of RDD
`top(3)`	Top 3 elements of RDD
`takeSample(withReplacement=True,3)`	Create sample of 3 elements with replacement
`sum()`	Find element sum (assumes numeric elements)
`mean()`	Find element mean (assumes numeric elements)
`stdev()`	Find element deviation (assumes numeric elements)

Vamos a crear un archivo de texto con el que trabajar:
%%writefile example2.txt
first
second line
the third line
then a fourth line

#Now let's perform some transformations and actions on this text file:
from pyspark import SparkContext
sc = SparkContext()

Show RDD
sc.textFile('example2.txt') #crear un RDD que contenga este archivo de texto

# Save a reference to this RDD
text_rdd = sc.textFile('example2.txt')

Map vs flatMap

Map a function (or lambda expression) to each line. Then collect the results.
text_rdd.**map**(lambda line: line.split()).collect()

```
[['first'],
 ['second', 'line'],
 ['the', 'third', 'line'],
 ['then', 'a', 'fourth', 'line']]
```

text_rdd.**collect**()

```
['first ', 'second line', 'the third line', 'then a fourth line']
```

Collect everything as a single flatMap
text_rdd.**flatMap**(lambda line: line.split()).collect()

```
['first',
 'second',
 'line',
 'the',
 'third',
 'line',
 'then',
 'a',
 'fourth',
 'line']
```

RDDs and Key Value Pairs

Now that we've worked with RDDs and how to aggregate values with them, we can begin to look into working with Key Value Pairs. In order to do this, let's create some fake data as a new text file. This data represents some services sold to customers for some SAAS business:

%%writefile services.txt

#EventId	Timestamp	Customer	State	ServiceID	Amount
201	10/13/2017	100	NY	131	100.00
204	10/18/2017	700	TX	129	450.00
202	10/15/2017	203	CA	121	200.00
206	10/19/2017	202	CA	131	500.00
203	10/17/2017	101	NY	173	750.00
205	10/19/2017	202	TX	121	200.00

services = sc.textFile('services.txt')

services.take(2)

```
['#EventId    Timestamp    Customer   State    ServiceID    Amount',
 '201       10/13/2017      100       NY       131          100.00']
```

services.map(lambda x: x.split())

services.map(lambda x: x.split()).take(3) #lo que hacemos con map acá es separar cada valor por ,

```
[['#EventId', 'Timestamp', 'Customer', 'State', 'ServiceID', 'Amount'],
 ['201', '10/13/2017', '100', 'NY', '131', '100.00'],
 ['204', '10/18/2017', '700', 'TX', '129', '450.00']]
```

# Let's remove that first hash-tag! de #Event_Id a Event_Id directamente
services.map(lambda x: x[1:] if x[0]=='#' else x).collect()

```
['EventId    Timestamp    Customer   State    ServiceID    Amount',
 '201       10/13/2017      100       NY       131          100.00',
 '204       10/18/2017      700       TX       129          450.00',
 '202       10/15/2017      203       CA       121          200.00',
 '206       10/19/2017      202       CA       131          500.00',
 '203       10/17/2017      101       NY       173          750.00',
 '205       10/19/2017      202       TX       121          200.00']
```

Using Key Value Pairs for Operations
Let us now begin to use methods that combine lambda expressions that use a ByKey argument. These ByKey methods will assume that your data is in a Key,Value form. For example let's find out the total sales per state:

# From Previous, hacemos todo junto separar por comas los valores y también eliminar el # de #Event_Id
cleanServ = services.map(lambda x: x[1:] if x[0]=='#' else x).map(lambda x: x.split())
cleanServ.collect()

```
[['EventId', 'Timestamp', 'Customer', 'State', 'ServiceID', 'Amount'],
 ['201', '10/13/2017', '100', 'NY', '131', '100.00'],
 ['204', '10/18/2017', '700', 'TX', '129', '450.00'],
 ['202', '10/15/2017', '203', 'CA', '121', '200.00'],
 ['206', '10/19/2017', '202', 'CA', '131', '500.00'],
 ['203', '10/17/2017', '101', 'NY', '173', '750.00'],
 ['205', '10/19/2017', '202', 'TX', '121', '200.00']]
```

# Let's start by practicing grabbing fields
# Queremos conocer las ventas por estado
cleanServ.map(lambda lst: (lst[3],lst[-1])).collect()

```
[('State', 'Amount'),
 ('NY', '100.00'),
 ('TX', '450.00'),
 ('CA', '200.00'),
 ('CA', '500.00'),
 ('NY', '750.00'),
 ('TX', '200.00')]
```

Continue with reduceByKey
reduceByKey asume que va a usar la 1ra columna como clave, en este caso State
cleanServ.map(lambda lst: (lst[3],lst[-1]))\
        .reduceByKey(lambda amt1,amt2 : amt1+amt2)\

```
        .collect()
```

```
[('State', 'Amount'),
 ('NY', '100.00750.00'),
 ('TX', '450.00200.00'),
 ('CA', '200.00500.00')]
```

Uh oh! Looks like we forgot that the amounts are still strings! Let's fix that:

```
cleanServ.map(lambda lst: (lst[3],lst[-1]))\
        .reduceByKey(lambda amt1,amt2 : float(amt1)+float(amt2))\
        .collect()
```

```
[('State', 'Amount'), ('NY', 850.0), ('TX', 650.0), ('CA', 700.0)]
```

We can continue our analysis by sorting this output:

```
# Grab state and amounts
# Add them
# Get rid of ('State','Amount')
# Sort them by the amount value
cleanServ.map(lambda lst: (lst[3],lst[-1]))\
.reduceByKey(lambda amt1,amt2 : float(amt1)+float(amt2))\
.filter(lambda x: not x[0]=='State')\
.sortBy(lambda stateAmount: stateAmount[1], ascending=False)\
.collect()
```

```
[('NY', 850.0), ('CA', 700.0), ('TX', 650.0)]
```

** Remember to try to use unpacking for readability. For example: **

```
x = ['ID','State','Amount']

def func1(lst):
    return lst[-1]

def func2(id_st_amt):
    # Unpack Values
    (Id,st,amt) = id_st_amt
    return amt

func1(x)
        'Amount'

func2(x)
        'Amount'
```