

Laboratorio Nacional de Modelaje y Sensores Remotos

Taller Python

DR. VICTOR M. RODRIGUEZ MORENO

Responsable del Laboratorio Nacional de Modelaje y Sensores Remotos

IIS JORGE E. MAURICIO RUVALCABA. PSP. INIFAP

MC Arturo Corrales Suastegui INIFAP

Breve historia

- Creado por Guido Van Rossum en 1991.
- Es sucesor del lenguaje ABC
- Su nombre esta inspirado en el programa de televisión de la BBC “Monty Python Flying Circus”

¿Qué es?

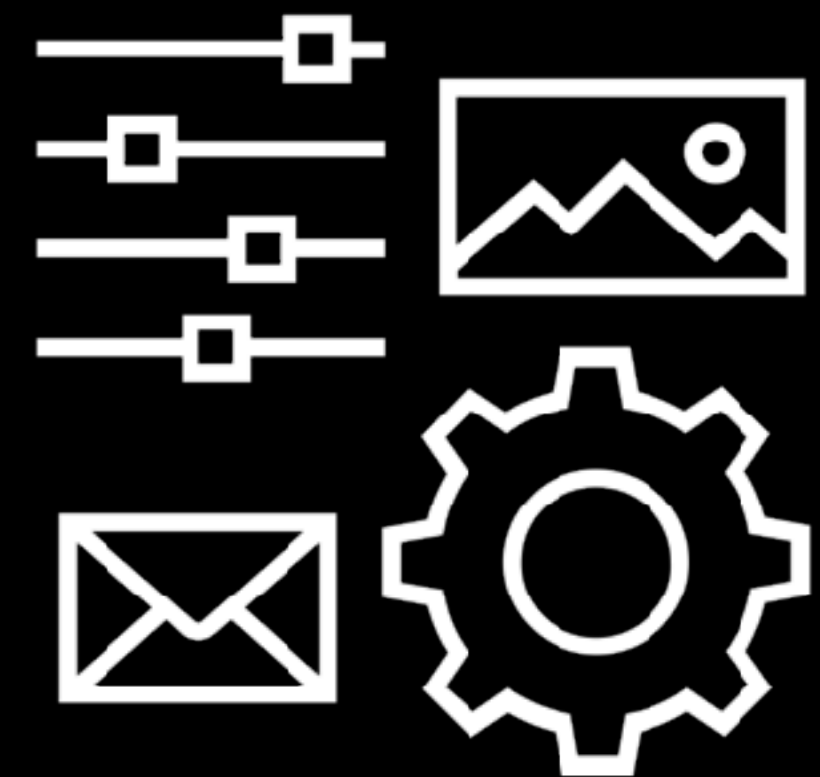
- Interpretado
- Multiparadigma
- Tipado dinámico
- Sintaxis de código legible
- Open source
- Multiplataforma

¿Porqué aprenderlo?

- Curva de aprendizaje baja; esto lo vamos a ver desde el primer ejercicio de este taller. Una sola línea de código nos da el resultado directamente
- Sintaxis legible (simple)
- Utilidad en muchos entornos de trabajo. Servidor, páginas web, aplicaciones desktop, etc; agricultura, algoritmos de estimación de datos perdidos, reconstrucción de series de datos, etc, etc

¿Big Data?

Es un término para conjuntos de datos que son tan grandes o complejos que el software de aplicación de procesamiento de datos tradicional es inadecuado para tratar con ellos.



¿Aplicación?



Detección de fraudes en
tiempo real



Análisis de
sentimientos del
consumidor



Gestión de inteligencia del tráfico

¿Características?

- Volumen: demasiados datos para manejar fácilmente
- Velocidad: la velocidad de entrada y salida de datos dificulta su análisis
- Variedad: el rango y el tipo de fuentes de datos son demasiado grandes para asimilar

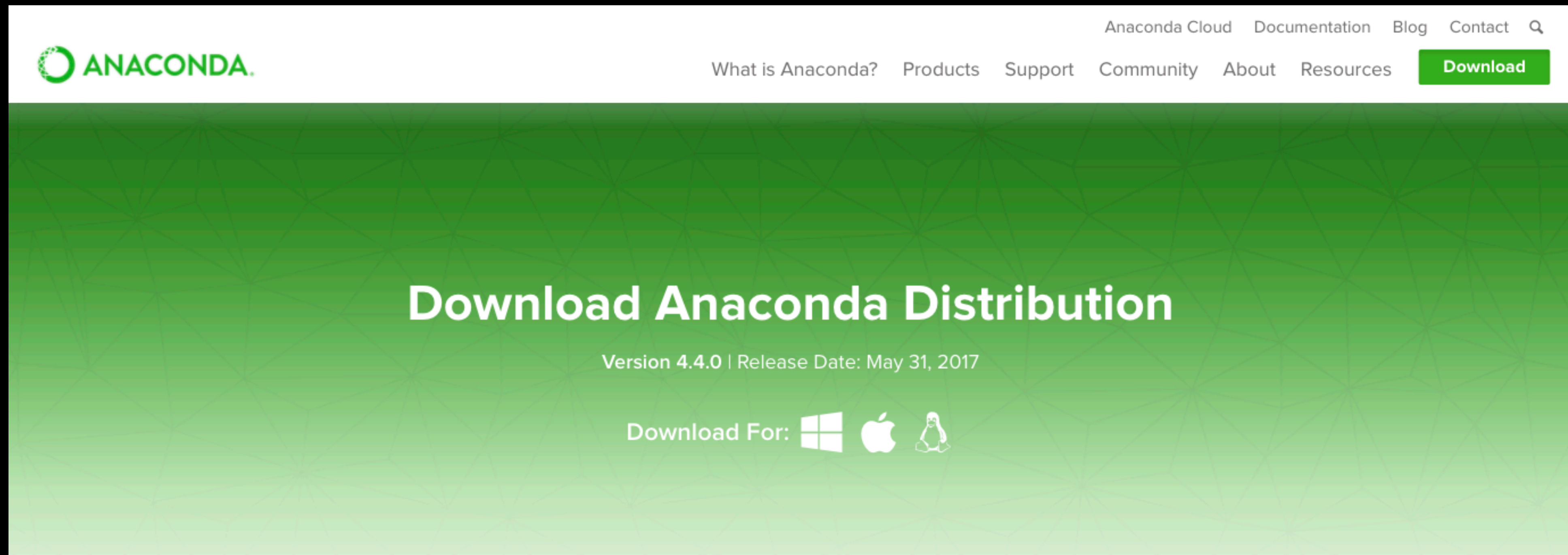
Inicio del taller

Exámen

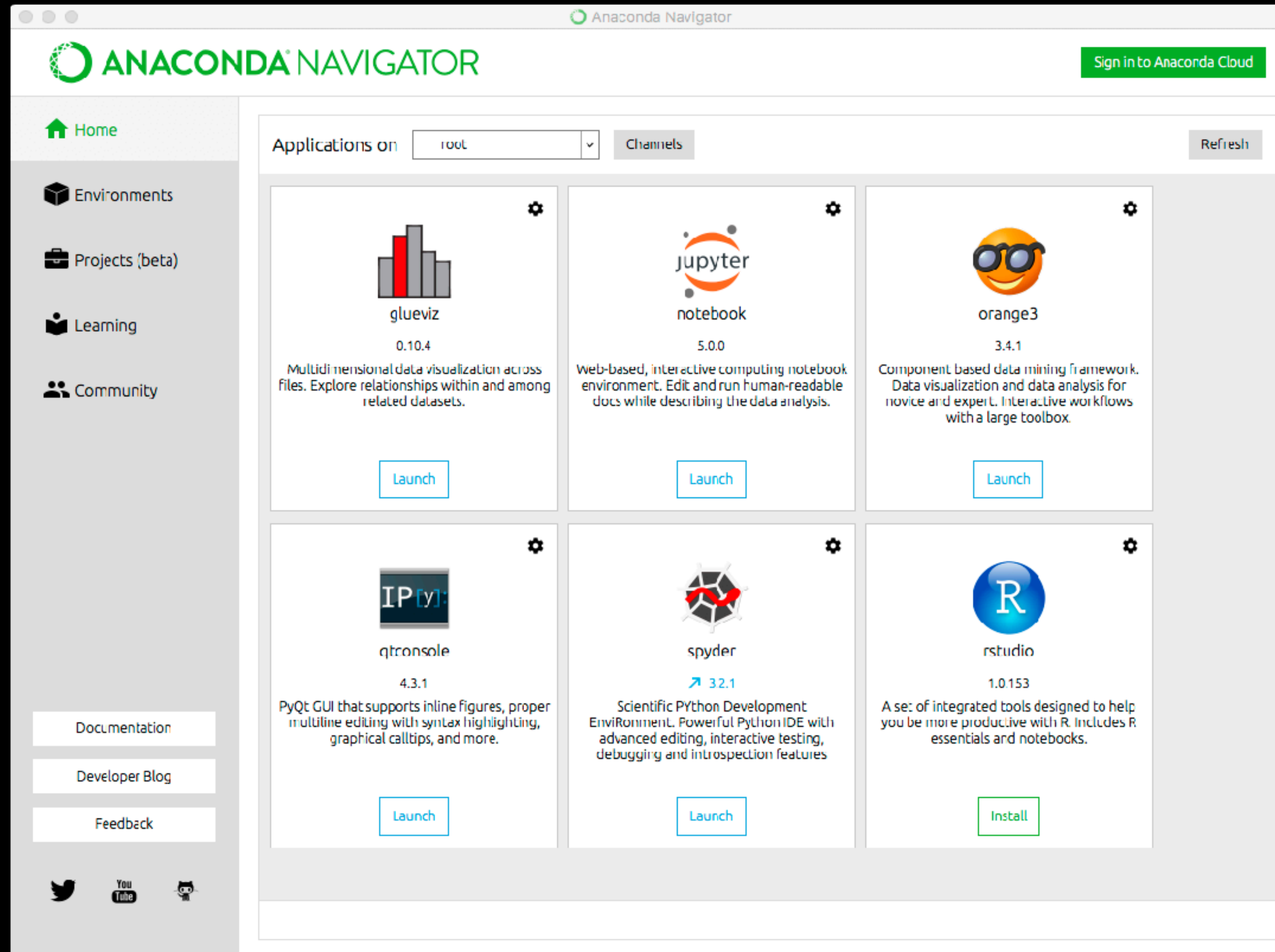
INIFAP_Course/ejercicios/Pandas/Ejercicio_Estaciones_Aguascalientes.md

Instalación Anaconda

- www.anaconda.com/downloads
- Windows, macOS, Linux
- Windows-macOS: asistente, clic, clic, clic, Finalizar
- Linux: descargar el archivo .sh, ejecutarlo.

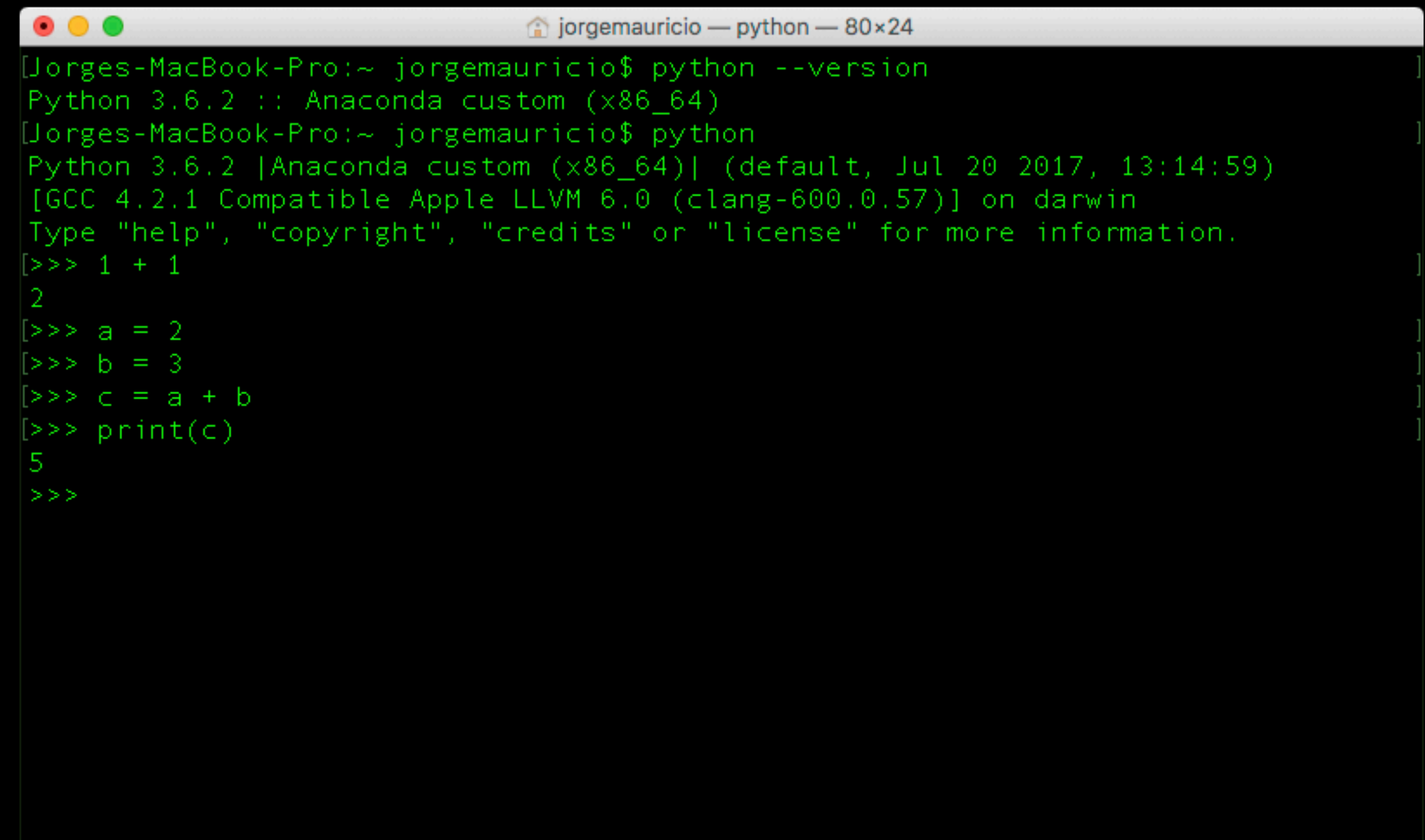


Anaconda Navigator



Modo Interactivo

- Para emplear el modo interactivo, se debe de ingresar el comando **python** en la terminal de comandos. De ahí en adelante las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa.

A screenshot of a terminal window titled 'jorgemaurocio — python — 80x24'. The terminal shows the execution of the 'python --version' command, which returns 'Python 3.6.2 :: Anaconda custom (x86_64)'. Then, the 'python' command is executed, which starts the Python interactive shell. The shell displays the version and environment information, followed by a prompt '>>>'. The user enters '1 + 1', and the shell returns '2'. Then, the user enters 'a = 2', 'b = 3', 'c = a + b', and 'print(c)', and the shell returns '5'.

Modo IDE

- Existen varios IDE (Interface Development Enviroment), los más populares son:
 - Spyder
 - Jupyter



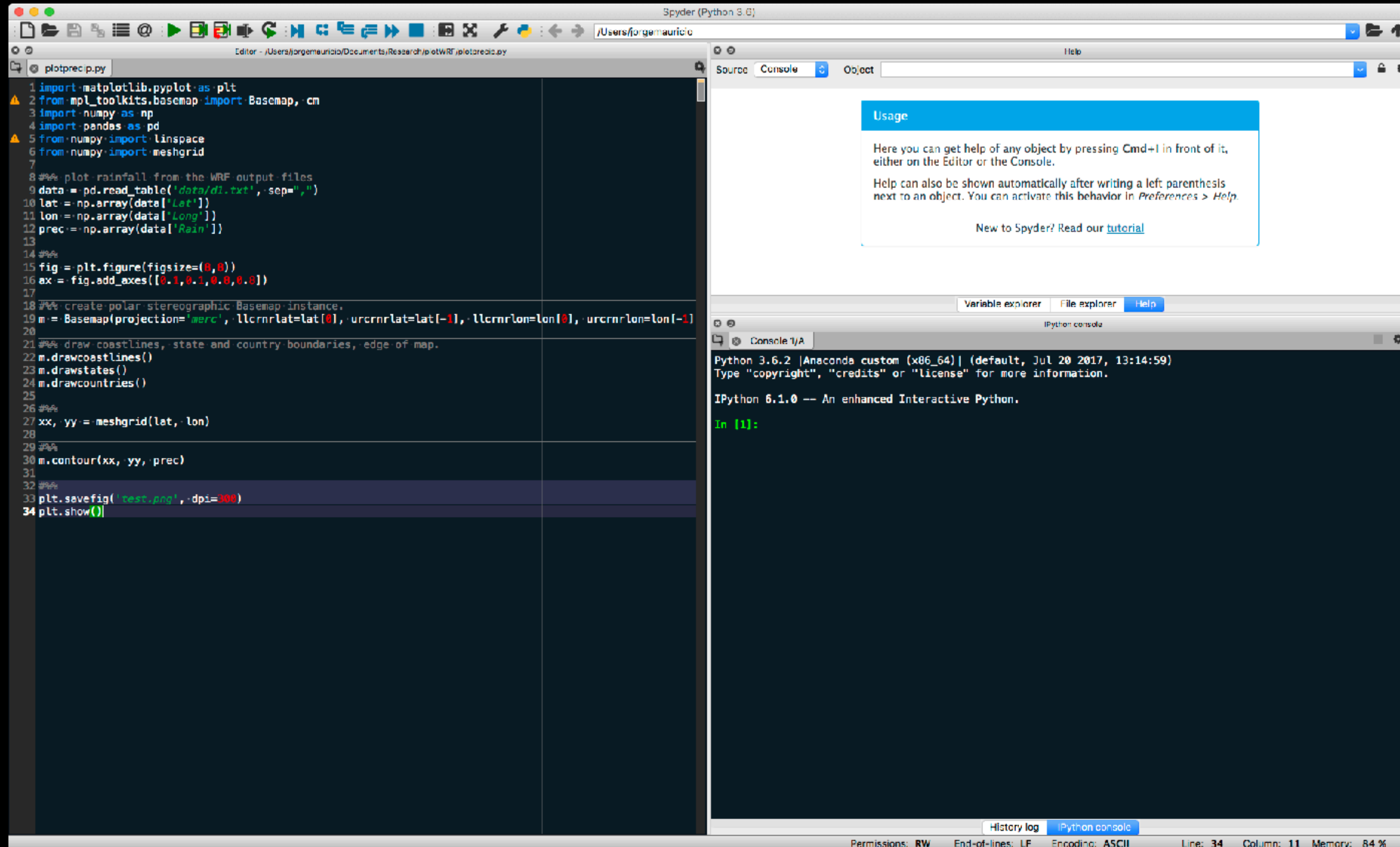
spyder



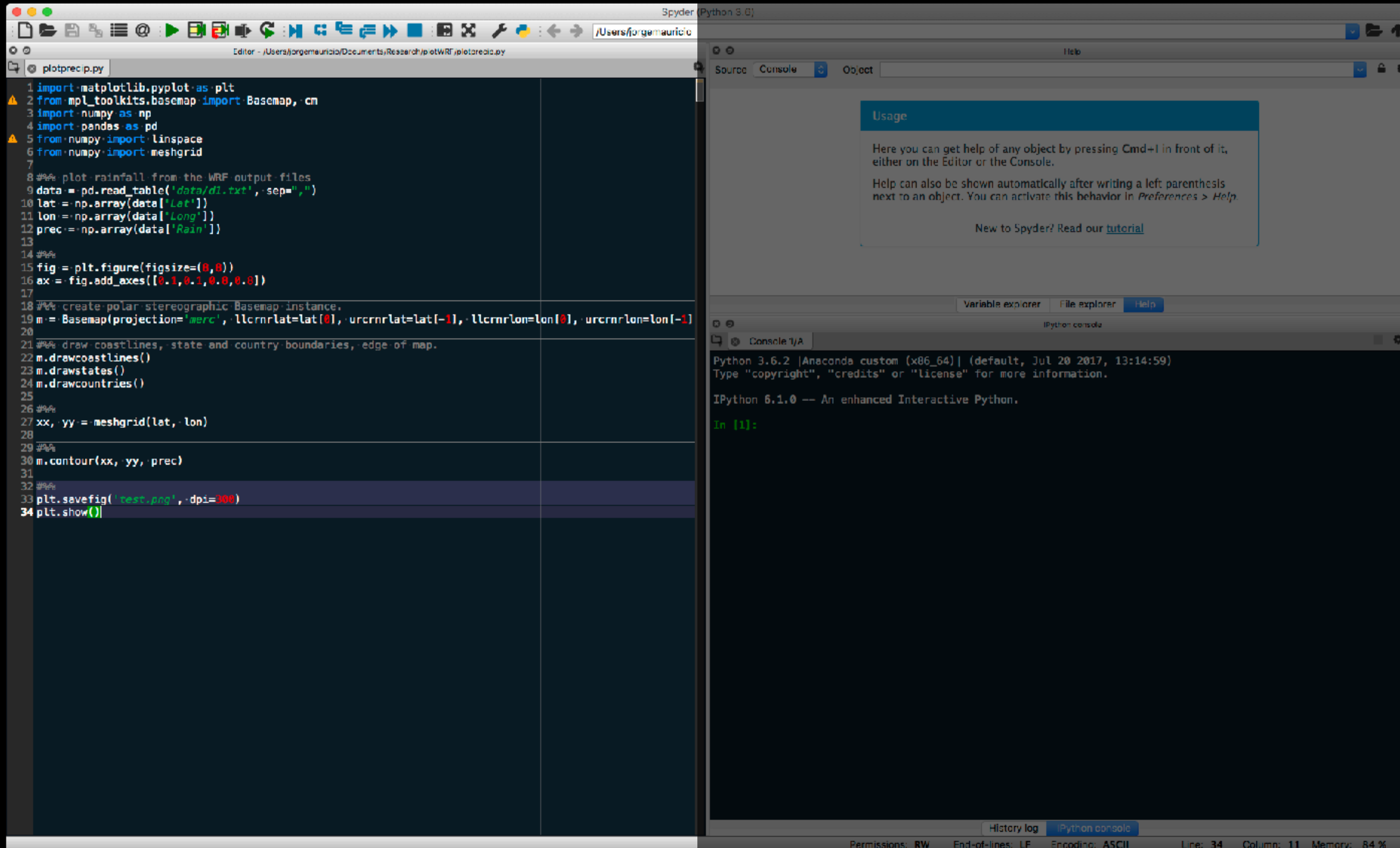
Spyder

- Entorno de desarrollo interactivo.
- Entorno de computación numérica que permite el uso de:
 - NumPy (álgebra lineal)
 - SciPy (procesamiento de señales e imágenes)
 - Matplotlib (trazado interactivo 2D / 3D)

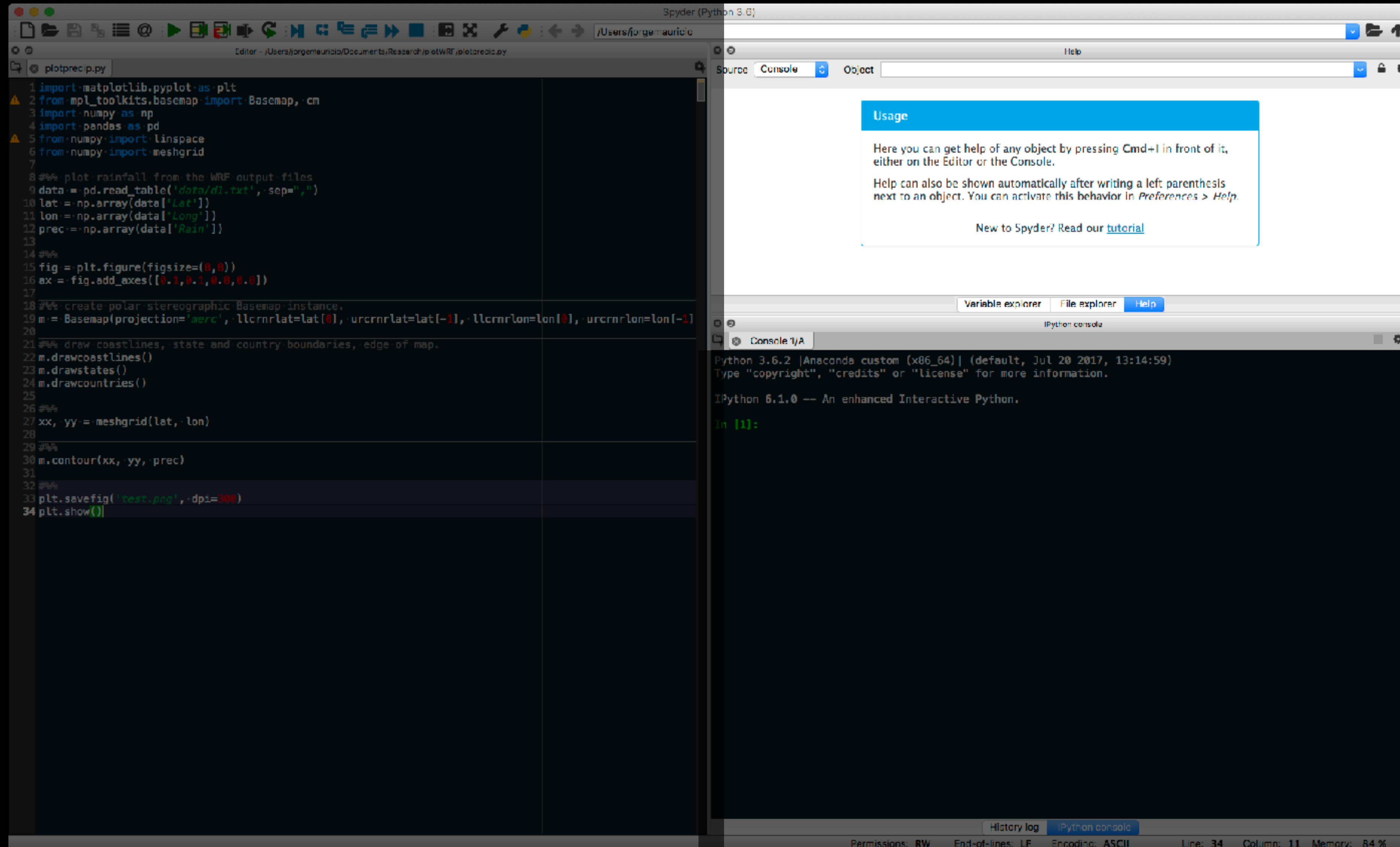
IDE



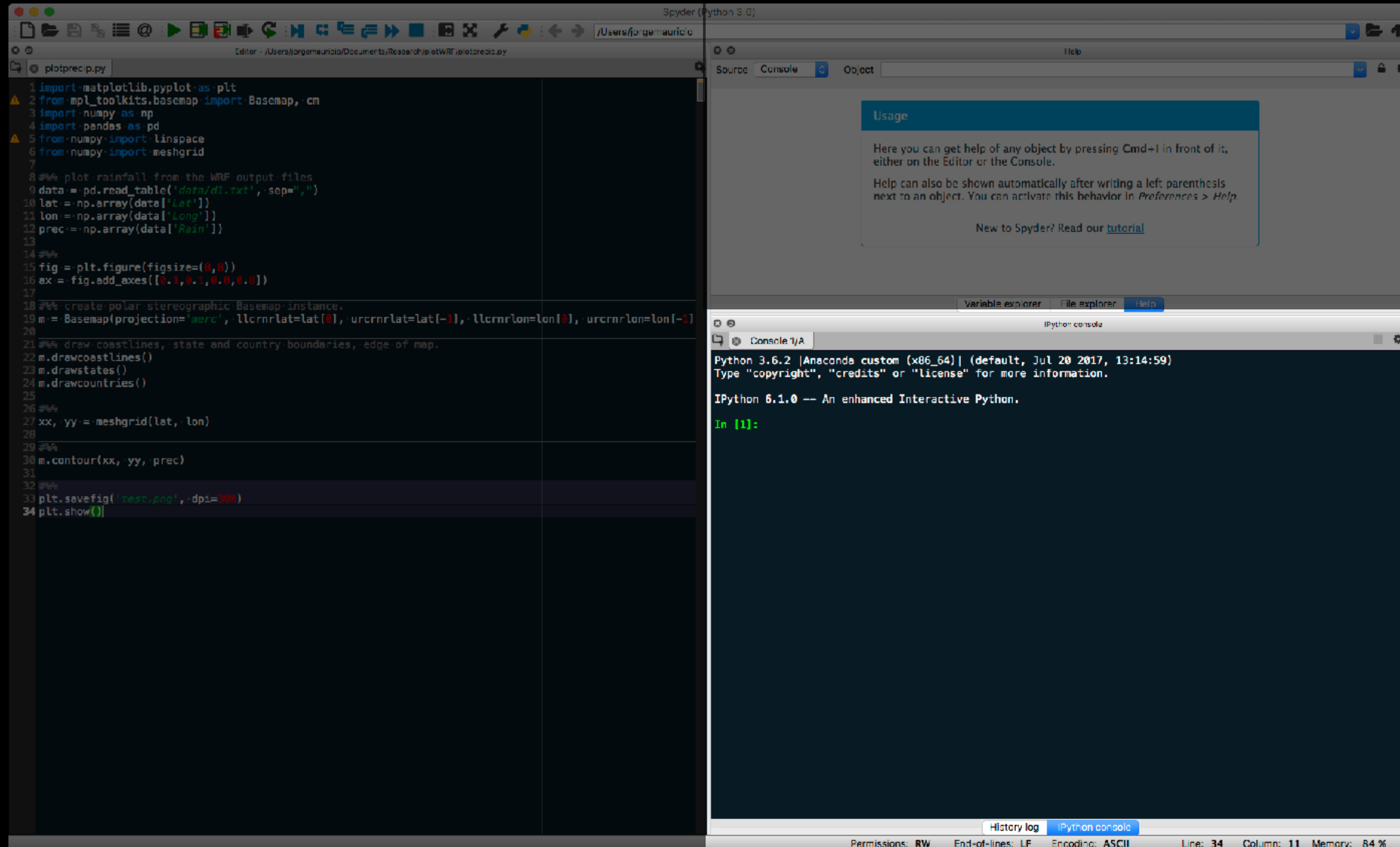
Editor



Visualizador de archivos





Consola














Jupyter Notebook

- Aplicación web
- Permite código en vivo, visualizaciones y texto
- Soporta más de 40 lenguajes, widgets interactivos y big data
- Los archivos se pueden compartir

IDE

 jupyter 1_Arreglos Numpy (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

          Code 

```
In [3]: # librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

```
In [4]: my_list = [1,2,3]
my_list
```

```
Out[4]: [1, 2, 3]
```

```
In [5]: np.array(my_list)
```



```
Out[5]: array([1, 2, 3])
```

```
In [6]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```












```
Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Barra de herramientas

 jupyter 1_Arreglos Numpy (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

          Code 

```
In [3]: # librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

```
In [4]: my_list = [1,2,3]
my_list
```

```
Out[4]: [1, 2, 3]
```

```
In [5]: np.array(my_list)
```

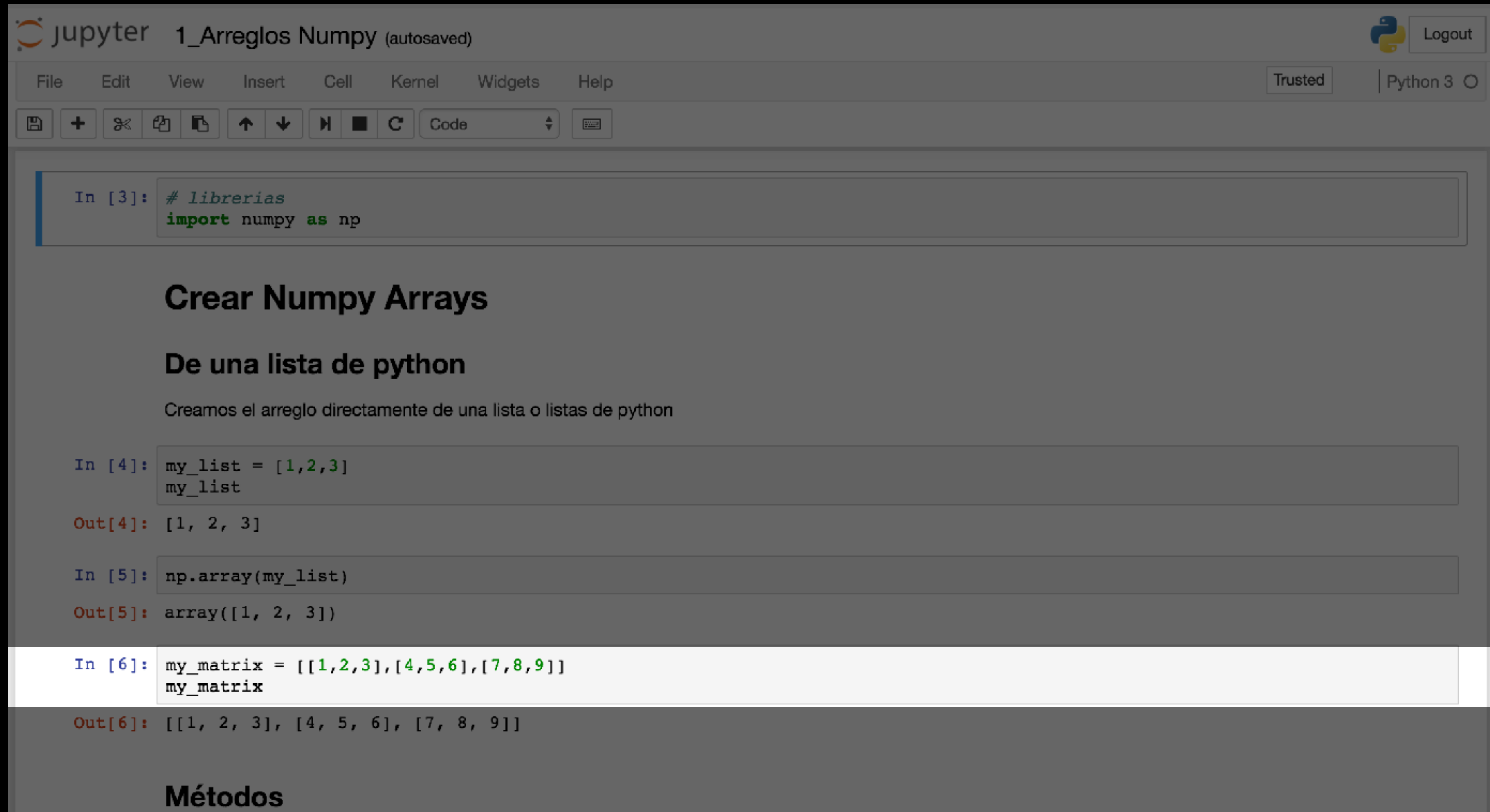
```
Out[5]: array([1, 2, 3])
```

```
In [6]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

```
Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Línea de código



The image shows a Jupyter Notebook interface with the title "1_Arreglos Numpy (autosaved)". The top bar includes a menu (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a "Trusted" status indicator, and a "Python 3" kernel selector. Below the menu is a toolbar with icons for saving, adding new files, undo, redo, and other editing functions. The main area contains three code cells. The first cell, labeled "In [3]:", imports the NumPy library as 'np'. The second cell, labeled "In [4]:", creates a Python list 'my_list' with values [1, 2, 3] and prints it. The output is "[1, 2, 3]". The third cell, labeled "In [5]:", creates a NumPy array from 'my_list' using 'np.array(my_list)' and prints it. The output is "array([1, 2, 3])". The fourth cell, labeled "In [6]:", creates a 3x3 matrix 'my_matrix' with values [[1, 2, 3], [4, 5, 6], [7, 8, 9]] and prints it. The output is "[[1, 2, 3], [4, 5, 6], [7, 8, 9]]".

```
In [3]: # librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

```
In [4]: my_list = [1,2,3]
my_list
```

```
Out[4]: [1, 2, 3]
```

```
In [5]: np.array(my_list)
```

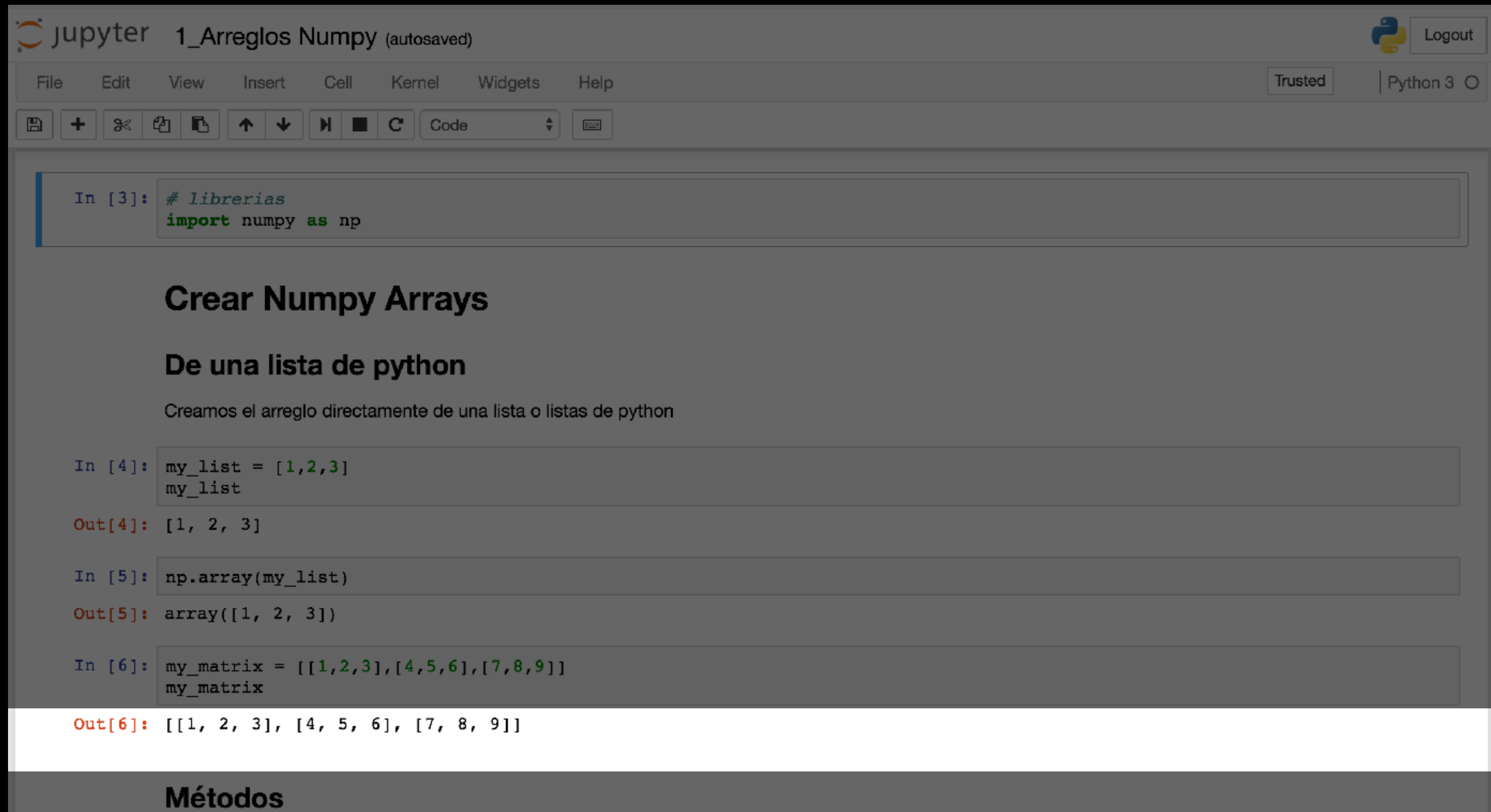
```
Out[5]: array([1, 2, 3])
```

```
In [6]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

```
Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Línea de resultado



The image shows a Jupyter Notebook interface with the title "1_Arreglos Numpy (autosaved)". The top bar includes a menu (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a "Trusted" status indicator, and a "Python 3" kernel selector. Below the menu is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook contains three code cells. The first cell imports the NumPy library. The second cell creates a Python list and converts it to a NumPy array. The third cell creates a 2D NumPy array from a list of lists. The output of the third cell is highlighted in a white box at the bottom of the notebook area.

```
In [3]: # librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

```
In [4]: my_list = [1,2,3]
my_list
```

```
Out[4]: [1, 2, 3]
```

```
In [5]: np.array(my_list)
```

```
Out[5]: array([1, 2, 3])
```

```
In [6]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

```
Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Markdown

[illegible]

Sintaxis Markdown

Título

Subtítulo

Título normal

**** Negrita ****

* Lista

* ` Código `

* *_ Itálica _*

[link](http://github.com/jorgemauricio)

Título

Subtítulo

Título normal

Negrita

- Lista
- Código
- *Itálica*

[link](#)

Sintaxis

Indentación

El contenido de los bloques de código es delimitado mediante espacios o tabuladores, conocidos como **indentación**.

C (Indentación opcional)

```
int numeroMayor (int x, int y)
{
    if (x == y){
        printf("Los dos valores son iguales");
    }else if (x > y){
        printf("El primer valor es mayor");
    }else (x > y){
        printf("El primer valor es mayor");
    }
}
```

C (Indentación opcional)

```
int numeroMayor (int x, int y)
{
    if (x == y){
        printf("Los dos valores son iguales");
    }else if (x > y){
        printf("El primer valor es mayor");
    }else (x > y){
        printf("El primer valor es mayor");
    }
}
```


Python (Indentación obligatoria)

```
def numeroMayor(x,y):  
    if x == y:  
        print("Los dos valores son iguales")  
    elif x > y:  
        print("El primer valor es mayor")  
    else:  
        print("El segundo valor es mayor")
```

Python (Indentación obligatoria)

```
def numeroMayor(x,y):  
<--> if x == y:  
<--><--> print("Los dos valores son iguales")  
<--> elif x > y:  
<--><--> print("El primer valor es mayor")  
<--> else:  
<--><--> print("El segundo valor es mayor")
```

Comentarios

Existen dos formas. La primera y más apropiada para comentarios largos es utilizando la notación

''' comentario ''', tres apóstrofes de apertura y tres de cierre.

La segunda notación utiliza el símbolo **#**, y se extienden hasta el final de la línea.

```
'''  
Comentario multilínea  
'''
```

```
print("Hola Mundo") # Comentario en línea
```

Imprimir variables

```
a = 2
print(a)
# imprime 2
print("Hola mundo")
# imprime Hola mundo
a = 2
b = 3
print("Multiplicar {} x {} = {}".format(a,b,a*b))
# imprime Multiplicar 2 x 3 = 6
```

Variables

Las variables se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro módulo, función o proceso, incluso de un tipo diferente al que tenía previamente.

Se utiliza el símbolo “=” para asignar valores

```
x = 1
```

```
x = "Texto"
```

```
# Esto es posible por que los tipos son asignados dinámicamente
```

Operadores lógicos

!	not
	or
&	and

Tipos de datos

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	Cadena'
unicode	Cadena	Versión Unicode de str	u'Cadena'
list	Secuencia	Mutable, puede contener objetos de diversos tipos	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	(4.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, no contiene duplicados	set([4.0, 'Cadena', True])
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	frozenset([4.0, 'Cadena', True])
dict	Mapping	Grupo de pares clave:valor	{'key1': 1.0, 'key2': False}
int	Número entero	Precisión fija, convertido en long en caso de overflow.	42
long	Número entero	Precisión arbitraria	42L ó 456966786151987643L
float	Número decimal	Coma flotante de doble precisión	3.1415927
complex	Número complejo	Parte real y parte imaginaria j.	(4.5 + 3j)
bool	Booleano	Valor booleano verdadero o falso	True o False

Lista

Para declarar una lista se usan los corchetes “[]”.

Para acceder a los elementos de una lista se utiliza un índice entero (empezando por "0", no por "1"). Se pueden utilizar índices negativos para acceder elementos a partir del final.

Las listas se caracterizan por ser mutables, es decir, se puede cambiar su contenido en tiempo de ejecución.

```
>>>> lista = ["abc", 42, 3.1416]

>>>> lista[0] # Acceder a un elemento por su índice

'abc'

>>>> lista[-1] # Acceder a un elemento usando un índice negativo

3.1416

>>>> lista.append(True) # Añadir un elemento al final de la lista

>>>> lista

['abc', 42, 3.1416, True]

>>>> del lista[3] # Borra el elemento número 3 de la lista

>>>> lista[0] = "xyz" # Re-asignar el valor del primer elemento de la lista

>>>> lista[0:2] # Mostrar los elementos de la lista del índice 0 al 2

["xyz", 42]

>>>> lista_anidada = [lista, [True, 42L]]

>>>> lista_anidada

[['xyz', 42, 3.1416], [True, 42L]]

>>>> lista_anidada[1][0] # Acceder a un elemento de una lista dentro de otra lista
```

Diccionarios

Para declarar un diccionario se usan las llaves “{ }”. Contienen elementos separados por comas, donde cada elemento está formado por un par **clave : valor** (el símbolo : separa la clave de su valor correspondiente).

Los diccionarios son mutables, es decir, se puede cambiar el contenido de un valor en tiempo de ejecución.

En cambio, las claves de un diccionario deben ser inmutables. Esto quiere decir, por ejemplo, que no podremos usar ni listas ni diccionarios como claves.

El valor asociado a una clave puede ser de cualquier tipo de dato, incluso un diccionario.

```
>>>> diccionario = {"cadena": "abc", "numero": 42, "lista": [True, 42L]}
# Diccionario que tiene diferentes valores por cada clave
>>>> diccionario["cadena"] # Accede a su valor usando clave
'abc'
>>>> diccionario["lista"][0] # Acceder a un elemento de una lista dentro de un valor
True
>>>> diccionario["cadena"] = "xyz" # Re-asignar el valor de una clave
>>>> diccionario["cadena"]
"xyz"
>>>> diccionario["decimal"] = 3.1416 # Insertar un nuevo elemento clave:valor
>>>> diccionario["decimal"]
3.1416
```

Clases

Las clases se definen con la palabra clave `class`, seguida del nombre de la clase y, si hereda de otra clase, el nombre de esta.

En una clase un "método" equivale a una "función", y un "atributo" equivale a una "variable".

"**`__init__`**" es un método especial que se ejecuta al instanciar la clase, se usa generalmente para inicializar atributos y ejecutar métodos necesarios. Al igual que todos los métodos en Python, debe tener al menos un parámetro, generalmente se utiliza **`self`**. El resto de parámetros serán los que se indiquen al instanciar la clase.

Los atributos que se desee que sean accesibles desde fuera de la clase se deben declarar usando `self.` delante del nombre.

En python no existe el concepto de encapsulación, por lo que el programador debe ser responsable de asignar los valores a los atributos

```
>>>> class Persona:
...     def __init__(self, nombre, edad):
...         self.nombre = nombre # atributo
...         self.edad = edad # atributo
...     def mostrar_edad(self): # Es necesario que, al menos, tenga un parámetro, generalmente "self"
...         print(self.edad) # mostrar un atributo
...     def modificar_edad(self, edad): #modificando Edad
...         if edad < 0 or edad > 150: # Comprobar que la edad no sea menor de 0, ni mayor a 150
...             return False
...         else: # Si esta en el rango 0-150, entonces se modifica la variable
...             self.edad = edad # modificar la edad
```

```
>>>> p = Persona("Jorge", 20) # Instanciar la clase, como se puede ver, no se especifica el valor de "self"
```

```
>>>> p.nombre # la variable "nombre" del objeto sí es accesible desde fuera
```

```
'Jorge'
```

```
>>>> p.nombre = 'Ernesto' # Y por tanto, se puede cambiar su contenido
```

```
'Ernesto'
```

```
>>>> p.mostrar_edad() # Se llama a un método de la clase
```

```
20
```

```
>>>> p.modificar_edad(21)
```

```
# Es posible cambiar la edad usando el método específico que hemos hecho para hacerlo de forma controlada
```

```
>>>> p.mostrar_edad()
```

```
21
```


Condicionales

Una sentencia condicional **if** ejecuta su bloque de código interno sólo si se cumple cierta condición. Se define usando la palabra clave `if` seguida de la condición, y el bloque de código.

Condiciones adicionales, si las hay, se introducen usando **elif** seguida de la condición y su bloque de código. Todas las condiciones se evalúan secuencialmente hasta encontrar la primera que sea verdadera, y su bloque de código asociado es el único que se ejecuta.

Opcionalmente, puede haber un bloque final (la palabra clave es **else** seguida de un bloque de código) que se ejecuta sólo cuando todas las condiciones fueron falsas.

```
>>>> verdadero = True
>>>> if verdadero: # No es necesario poner "verdadero == True"
...     print("Verdadero")
... else:
...     print("Falso")
Verdadero
```

```
>>>> lenguaje = "Python"

>>>> if lenguaje == "C": # Lenguaje no es "C", por lo que este bloque se obviará y evaluará la siguiente condición
...     print("Lenguaje de programación: C")
...     elif lenguaje == "Python": # se pueden añadir tantos bloques "elif" como se quiera
...         print("Lenguaje de programación: Python")
...     else: # en caso de que ninguna de las anteriores condiciones fuera cierta, se ejecutaría este bloque
...         print("Lenguaje de programación: Indefinido")
...
...
```

Lenguaje de programación: Python

```
>>> if verdadero and lenguaje == "Python": # Uso de "and" para comprobar que ambas condiciones son verdaderas
...     print("Verdadero y Lenguaje de programación: Python")
...
Verdadero y Lenguaje de programación: Python
```

Ciclos

For

Es similar a **foreach** en otros lenguajes.

Recorre un objeto iterable, como una lista, una tupla o un generador, y por cada elemento del iterable ejecuta el bloque de código interno.

Se define con la palabra clave **for** seguida de un nombre de variable, seguido de **in**, seguido del iterable, y finalmente el bloque de código interno.

En cada iteración, el elemento siguiente del iterable se asigna al nombre de variable especificado

```
>>>> lista = ["a", "b", "c"]
>>>> for i in lista: # iteramos sobre una lista
...     print(i)
...
a
b
c
```

```
>>>> cadena = "abcdef"
>>>> for i in cadena: # iteramos sobre una cadena
...     print(i)
...
a
b
c
d
e
f
```

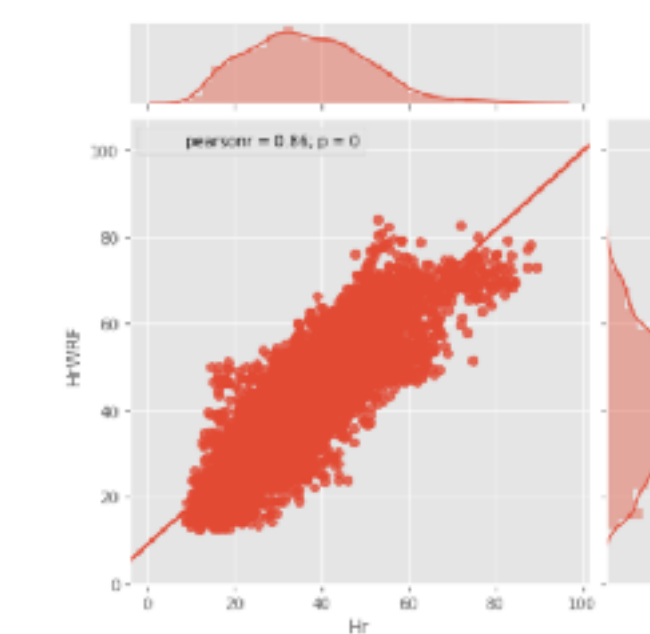
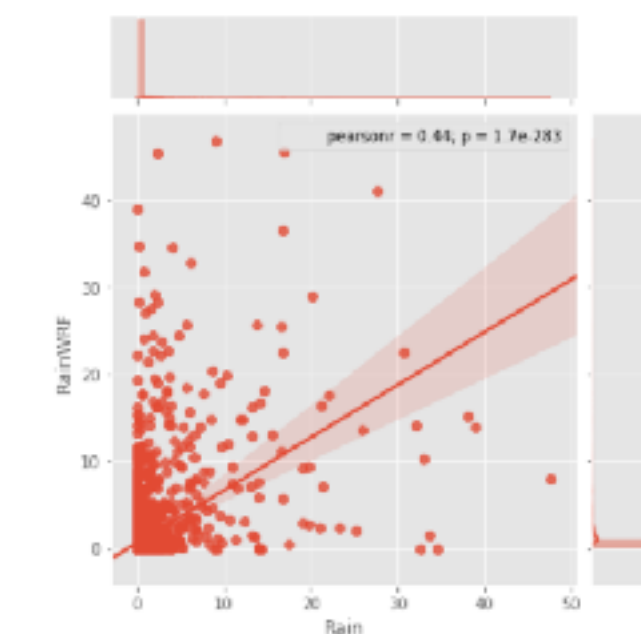
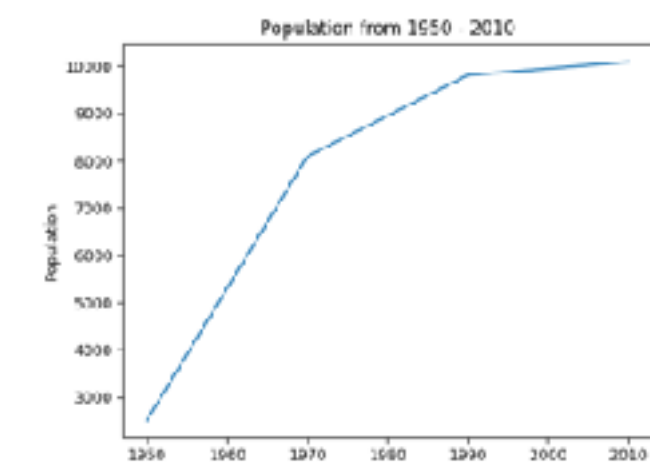
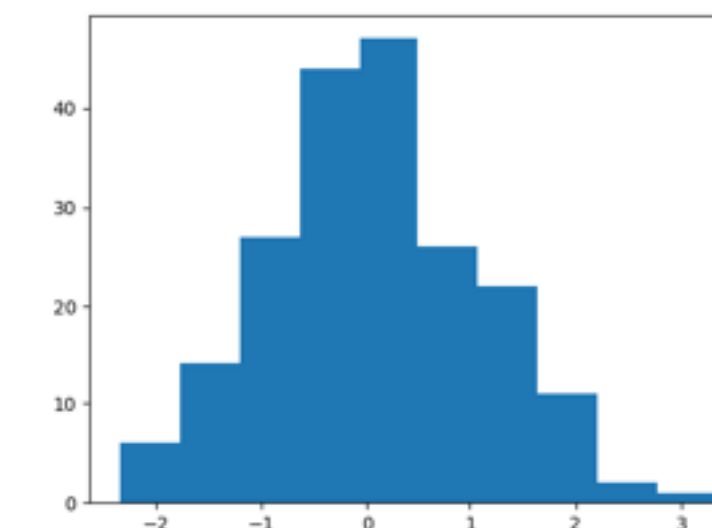
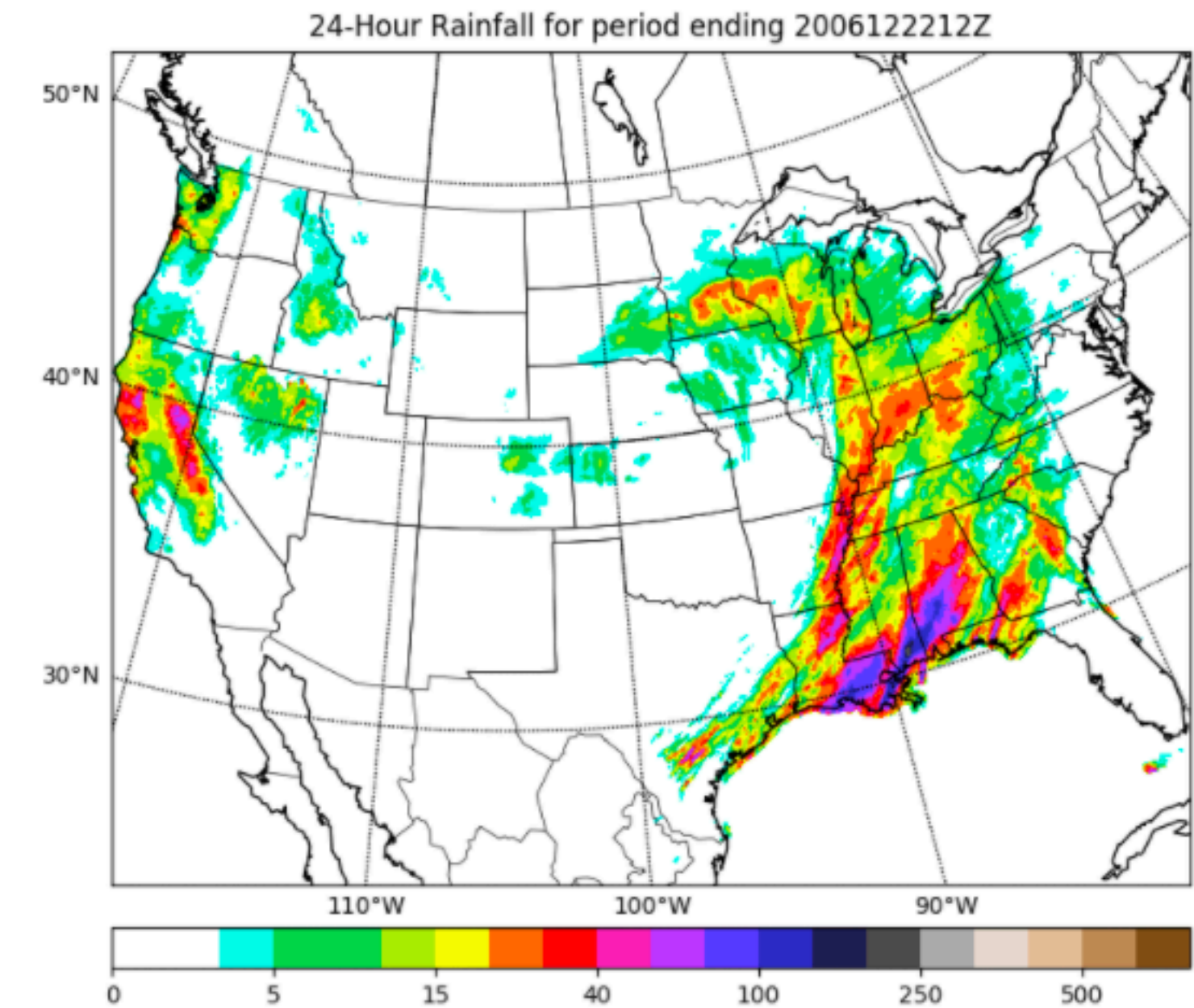

While

Evalúa una condición y, si es verdadera, ejecuta el bloque de código interno. Continúa evaluando y ejecutando mientras la condición sea verdadera. Se define con la palabra clave **while** seguida de la **condición**, y a continuación el bloque de código interno

```
>>>> numero = 0
>>>> while numero < 5:
...     print(numero)
...     numero += 1 # un buen programador modificará las variables
...                 # de control al finalizar el ciclo while
...
0
1
2
3
4
```

Módulos

- Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones y clases para realizar determinadas tareas.
- Un ejemplo es el módulo Basemap, que nos permite crear mapas.
- Otro ejemplo es el módulo os, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo **import** seguida del nombre del módulo que queremos usar.

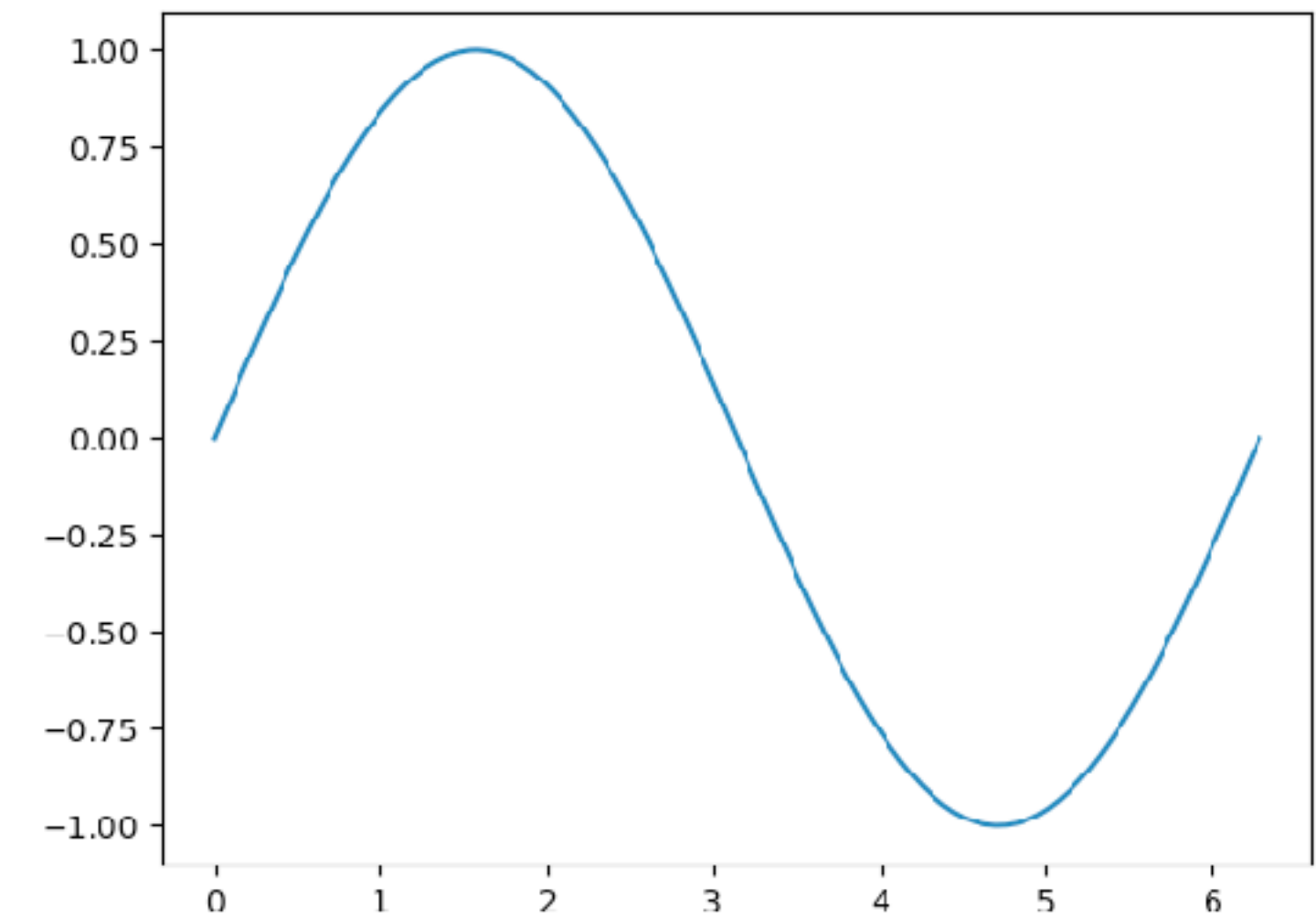


```
>>>> import os # módulo que provee funciones del sistema operativo
>>>> os.name # devuelve el nombre del sistema operativo
'posix'
>>>> os.mkdir("tmp/ejemplo") # crea un directorio en la ruta especificada
>>>> import time # módulo para trabajar con fechas y horas
.... time.strftime("%Y-%m-%d %H:%M:%S")
# dándole un cierto formato, devuelve la fecha y/u hora actual
'2017-09-15 10:52:01'
```

Numpy

Es un módulo de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

```
import numpy as np
from matplotlib import pyplot
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
pyplot.plot(x,y)
pyplot.show()
```



Arreglos

```
import numpy as np
a = np.array([1,2,3]) # arreglo de rank 1
print(type(a)) # imprime "<class 'numpy.ndarray'>"
print(a.shape) # imprime "(3,)"
print(a[0], a[1], a[2]) # imprime "1 2 3"
a[0] = 5 # cambiar un elemento del arreglo
print(a) # imprime "[5,2,3]"

b = np.array([[1,2,3],[4,5,6]]) # arreglo de rank 2
print(b.shape) # imprime ("2,3")
print(b[0,0], b[0,1], b[1,0]) # imprime "1, 2, 4"
```

Funciones

```
import numpy as np
a = np.zeros((2,2)) # crea un arreglo de 0's
print(a)
# imprime "[[ 0.  0.]
           [ 0.  0.]]"
```

Funciones

```
import numpy as np  
b = np.ones((1,2)) # crea un arreglo de 1's  
print(b) # imprime "[[1. 1.]]"
```

Funciones

```
import numpy as np
c = np.full((2,2),7) # crea un arreglo de una constante
print(c) # imprime "[[7. 7.
                  [7. 7.]]"
```

Funciones

```
import numpy as np
d = np.eye(2) # crea una matrix de 2x2
print(d) # imprime "[[1. 0]
              [0. 1.]]"
```

Funciones

```
import numpy as np
e = np.random.random((2,2)) # crea un arreglo de números random
print(e) # imprime "[[0.91209392 0.98736452
                [0.87594037 0.892384719]]"
```

Índices

```
import numpy as np

# Crea un arreglo de rank 2 con la siguiente forma (3,4)

a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])

# [[1  2  3  4]
#
#    [5  6  7  8]
#
#    [9 10 11 12]]

# crear un subarreglo de los 2 primeros renglones y la primer columna

b = a[:2, 1:3]

# [[2  3
#
#    [6  7]]

# una división del arreglo es una vista de los mismos datos, por lo tanto cualquier

# modificación, modificara el arreglo original

print(a[0,1]) #imprime 2

b[0,0] = 77 # b[0,0] es el mismo dato a a[0,1]

print(a[0,1]) # imprime 77
```


Funciones básicas matemáticas

Suma

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
print(x + y)
```

```
print(np.add(x,y))
```

```
# [[6.0  8.0]  
   [10.0 12.0]]
```

Resta

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
print(x - y)
```

```
print(np.subtract(x,y))
```

```
# [[-4.0 -4.0]
   [-4.0 -4.0]]
```

Multiplicación

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
print(x * y)
```

```
print(np.multiply(x,y))
```

```
# [[ 5.0 12.0]
   [21.0 32.0]]
```

División

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
print(x / y)
```

```
print(np.divide(x,y))
```

```
# [[ 0.2          0.33333333]
   [ 0.42857143  0.5         ]]
```

Raíz cuadrada

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
print(np.sqrt(x))
```

```
# [[ 1.          1.41421356]
   [ 1.73205081  2.          ]]
```

Pandas

- Librería de Python
- Limpiar, analizar, visualizar y otro tipo de análisis de información.
- Manipula muy bien diferentes tipos de información
- Esta desarrollado por encima de numpy lo cual permite una gran integración con otras librerías

Series

- Objeto similar a un arreglo de una sola dimensión
- Capaz de soportar cualquier tipo de dato
- Tiene índice

DataFrame

- Estructura de datos de dos dimensiones
- Capaz de soportar la mayoría de tipos de datos
- Índices y columnas

Crear un DataFrame

```
import pandas as pd
```

```
data = {'Country': ['Belgium', 'India', 'Brazil'],  
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],  
        'Population': [11190846, 1303171035, 207847528]}
```

```
df = pd.DataFrame(data, columns=['Country', 'Capital', 'Population'])
```

```
df = pd.read_csv('file.csv', header=None, nrows=5)
```

```
df = pd.read_excel('file.xlsx')
```

Información del DataFrame

`df.shape` # (filas, columnas)

`df.index` # describe los índices

`df.columns` # describe las columnas

`df.info()` # información del dataframe

`df.count()` # número de valores no NaN

Resumen del DataFrame

`df.sum()` # suma de los valores

`df.cumsum()` # acumulado de la sumatoria de los valores

`df.min()` / `df.max()` # mínimo / máximo de los valores

`df.idxmin()` / `df.idxmax()` mínimo / máximo de los índices

`df.describe()` # resumen estadístico

`df.mean()` # media de los valores

`df.median()` # mediana de los valores

Eliminar registros del DataFrame

```
s.drop(['a', 'c']) # eliminar datos de filas (axis=0)  
df.drop('Country', axis=1) # eliminar datos de una columna (axis=1)
```

Ordenar el DataFrame

```
df.sort_index() # ordenar por indice
```

```
df.sort_values(by='Country') # ordenar por valores de la columna Country
```

```
df.rank() # asignar un ranking
```

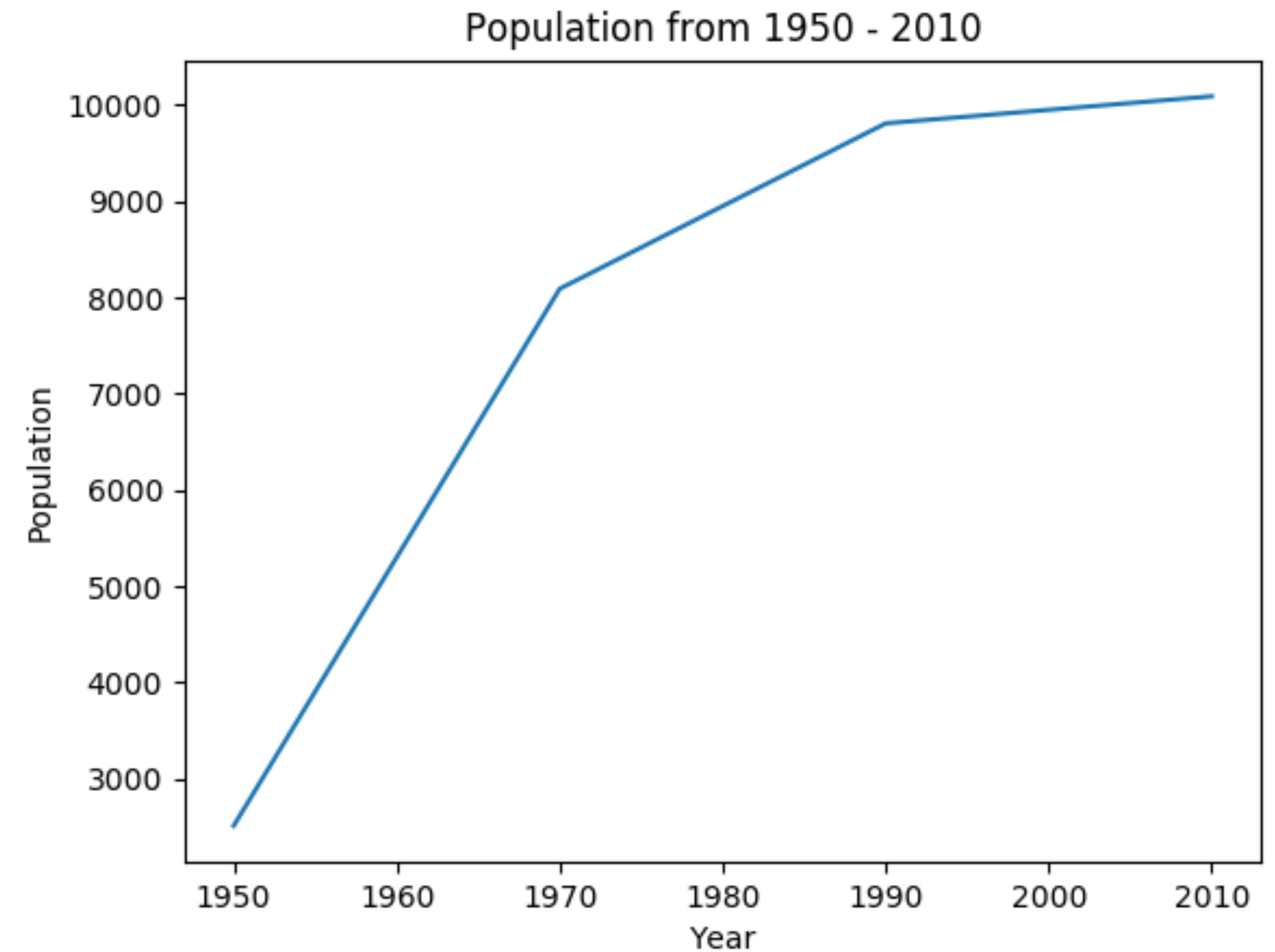
Matplotlib

Matplotlib es un módulo de trazado 2D de Python que produce gráficas de calidad de publicación en una variedad de formatos impresos y entornos interactivos a través de plataformas.

Matplotlib se puede utilizar en shell, scripts, IPython, Jupyter, servidores de aplicaciones web.

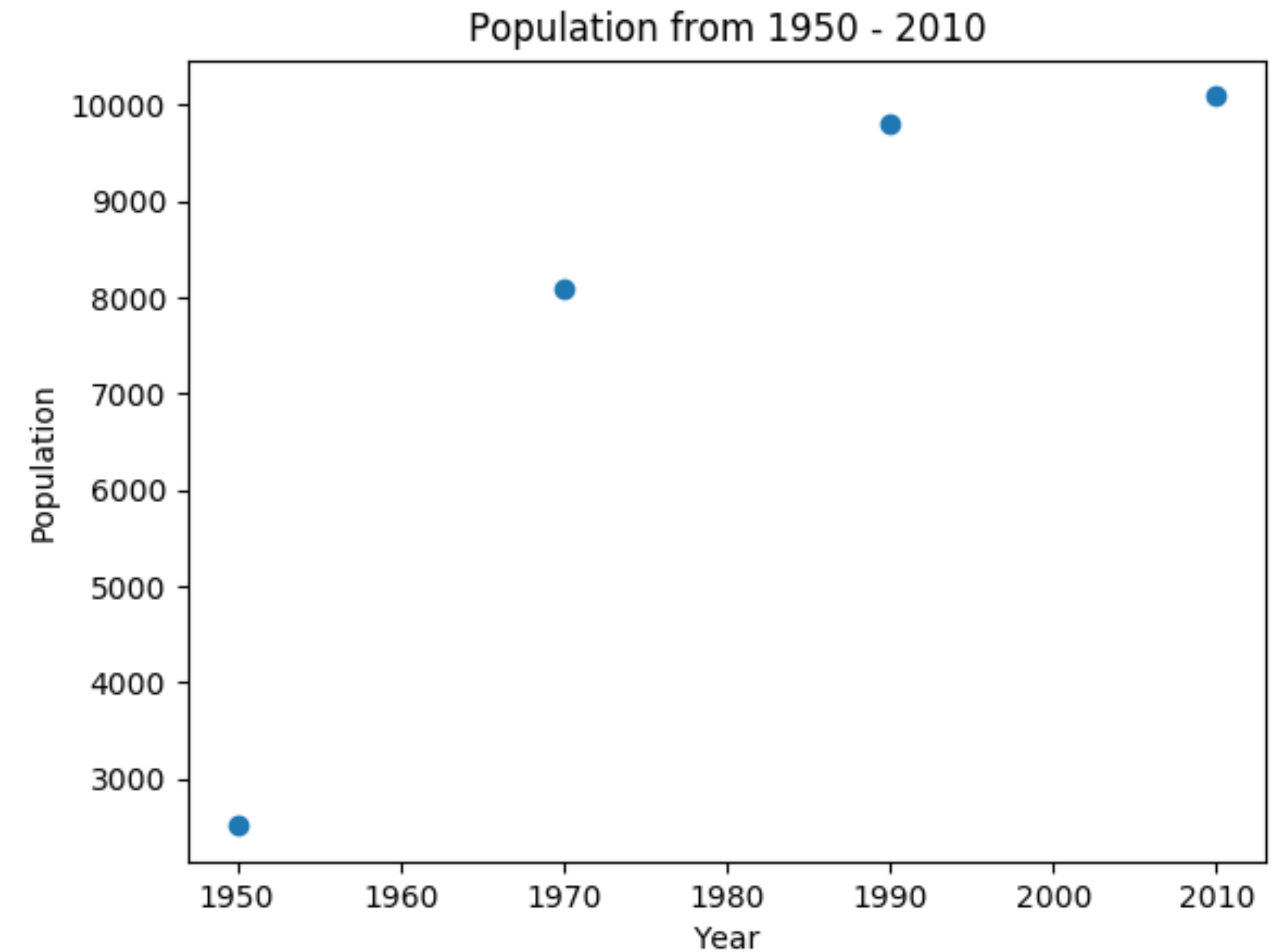
Gráfica de línea

```
import matplotlib.pyplot as plt
year = [1950, 1970, 1990, 2010]
pop = [2510, 8090, 9809, 10090]
plt.plot(year, pop)
plt.xlabel('Year')
plt.ylabel('Population')
plt.title('Population from 1950 - 2010')
plt.show()
```



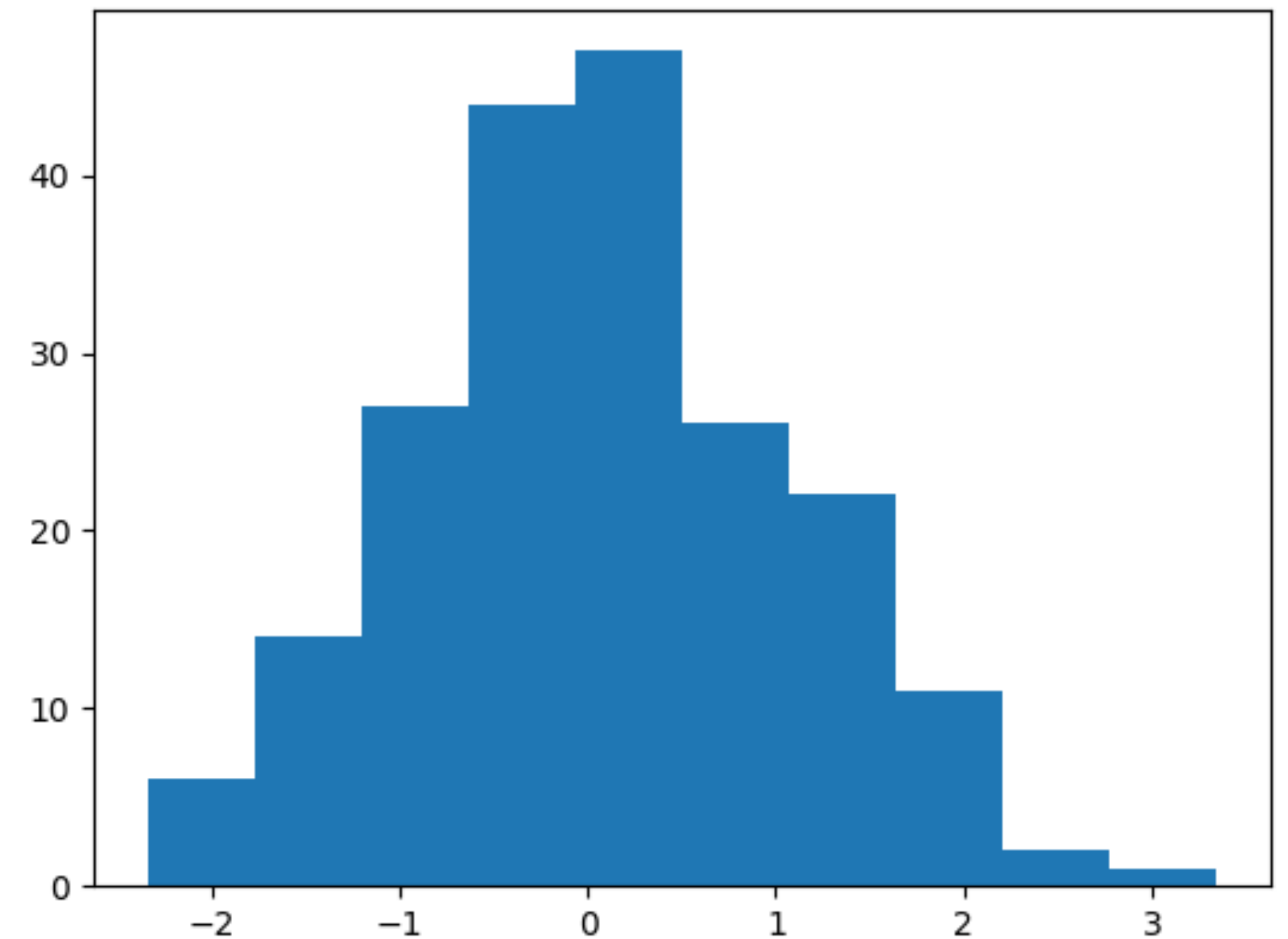
Gráfica de línea

```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2510, 8090, 9809, 10090]  
plt.scatter(year, pop)  
plt.xlabel('Year')  
plt.ylabel('Population')  
plt.title('Population from 1950 - 2010')  
plt.show()
```



Histograms

```
import numpy as np
import matplotlib.pyplot as plt
values = np.random.randn(200)
plt.hist(values)
plt.show()
```



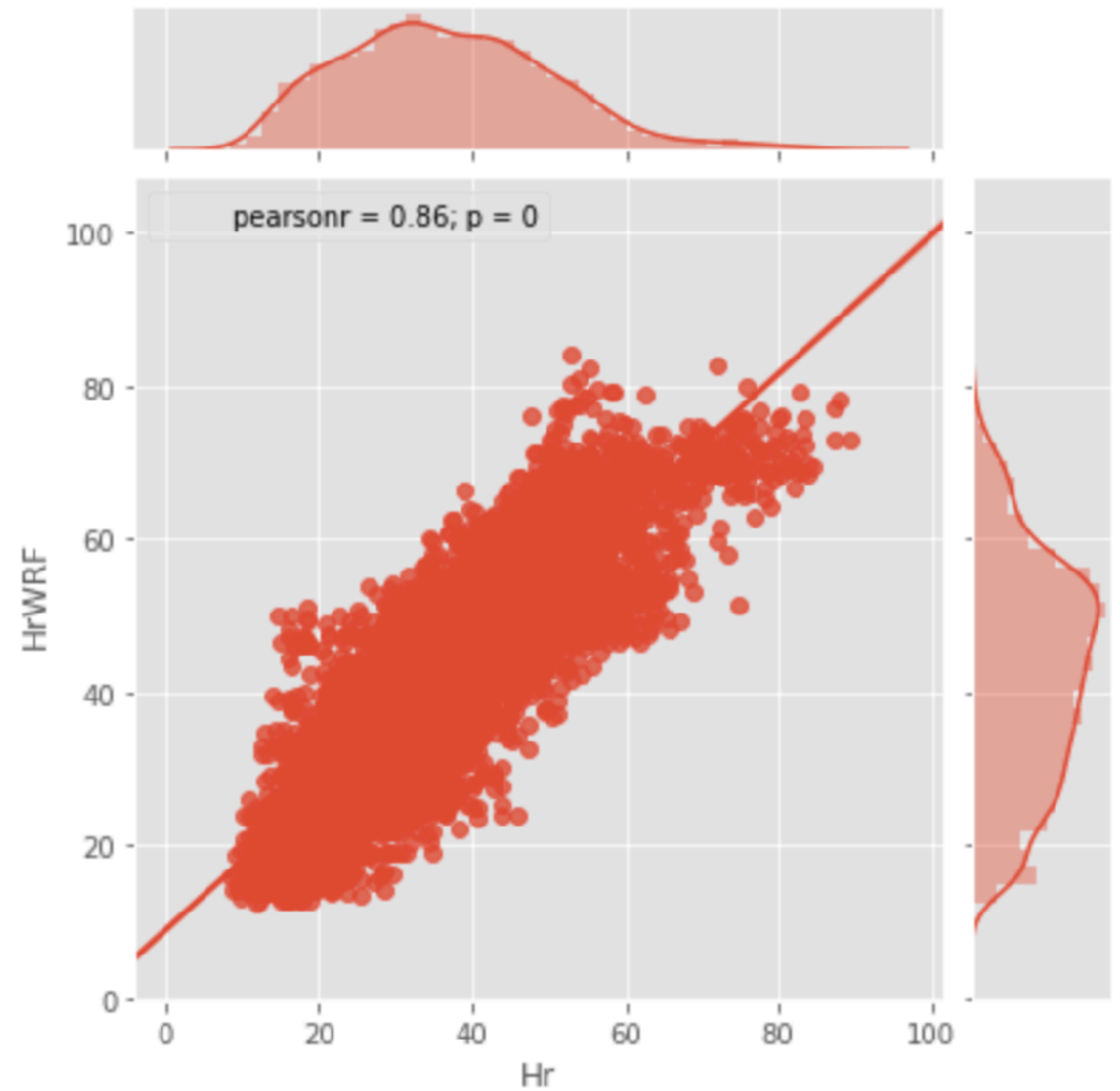
Seaborn

Es una biblioteca para hacer gráficos estadísticos atractivos e informativos en Python.

Está construido sobre matplotlib, está estrechamente integrado con PyData, incluye soporte para estructuras de datos numpy y pandas, así como de rutinas estadísticas de scipy y modelos de estadísticas.

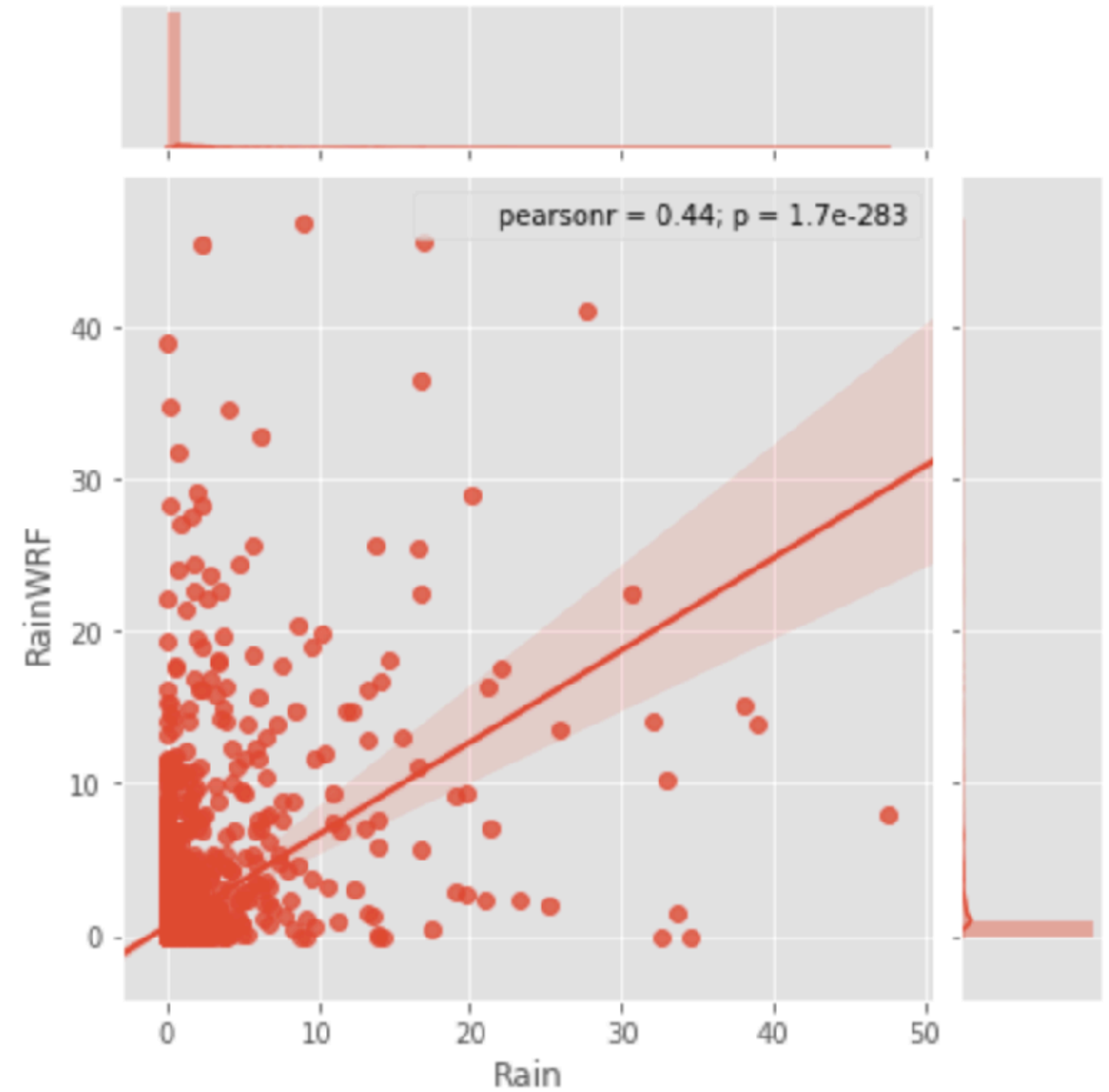
Regresión lineal con p y pearsonr

```
import seaborn as sns
sns.jointplot("Hr", "HrWRF", data=data, kind="reg")
```



Regresión lineal con p y pearsonr

```
import seaborn as sns  
sns.jointplot("Rain", "RainWRF", data=data, kind="reg")
```



ZEN OF PYTHON

Keep in mind Python's philosophy as you code

```
>>> import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess

...

Contacto

Dr. Victor M. Rodríguez M.
rodriguez.victor@inifap.gob.mx

IIS Jorge Ernesto Mauricio Ruvalcaba
jorge.ernesto.mauricio@gmail.com

MC Arturo Corrales Suastegui
corrales.arturo@inifap.gob.mx

Muchas gracias!