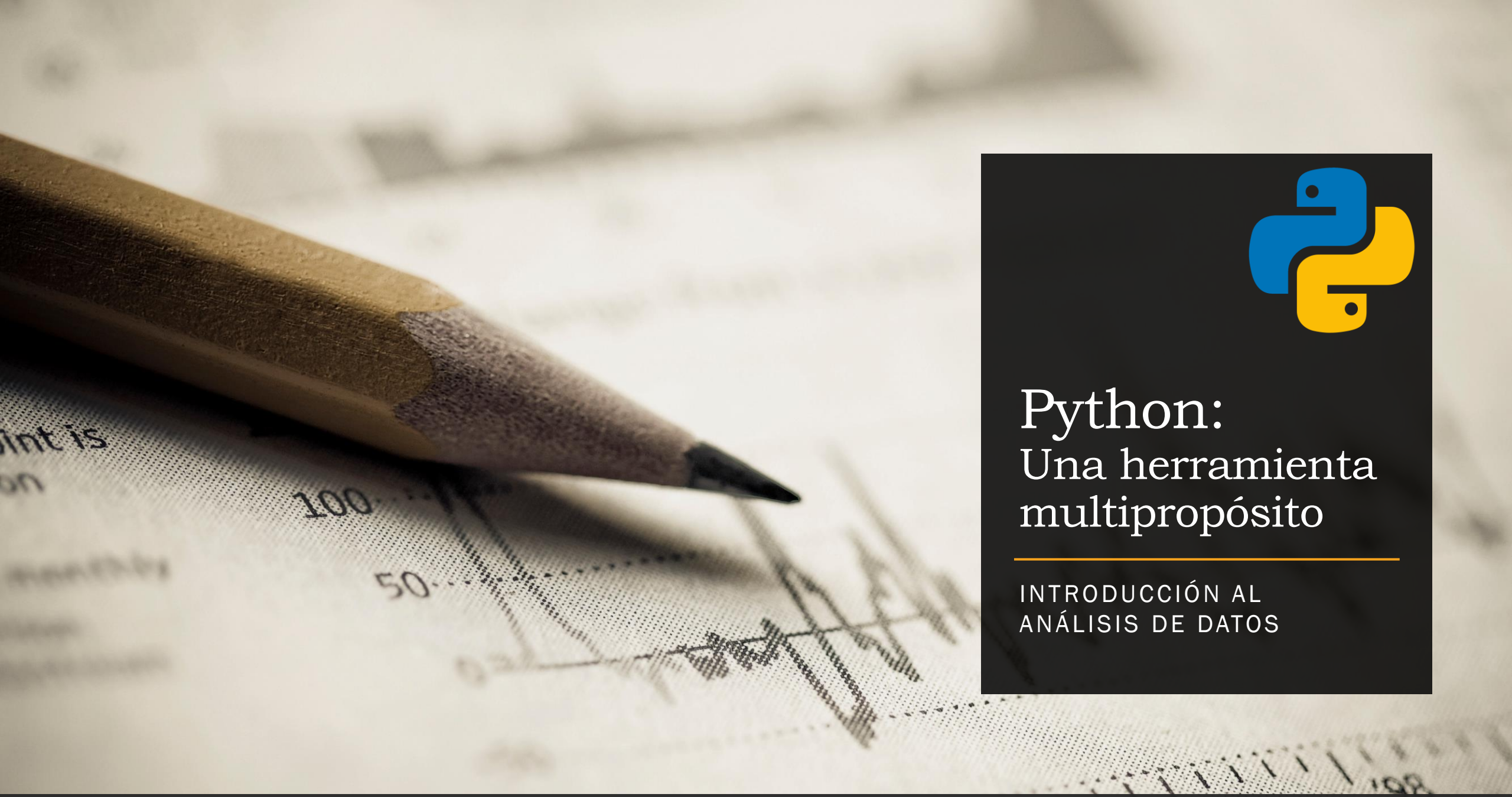


# Python:
## Una herramienta multipropósito

INTRODUCCIÓN AL ANÁLISIS DE DATOS

# Esta clase

- ¿Qué es análisis de datos?
- Formas de almacenar datos
- Exploración de datos: numpy, pandas
- Introducción al análisis con Python
- Visualización de datos con matplotlib

# Datos

Los datos son cualquier clase de información que identifica conceptos, números o relaciones entre situaciones naturales del mundo que nos rodea.

Nombres de personas, estadísticas de rendimiento de un computador, información económica de un país, etc.

# Datos

¿De dónde vienen los datos? Todo lugar!

**Ciencia**: Bases de datos de astronomía, genética, datos ambientales, datos de transporte, etc.

**Ciencias sociales**: Libros escaneados, documentos históricos,  datos de redes sociales, información de herramientas como el GPS.

**Comercio**: Reportes de ventas, transacciones del mercado de valores, tráfico de aerolíneas, etc.

**Entretenimiento**: Imágenes de internet, taquillas de Hollywood, datos de tráfico de Netflix, etc.

**Medicina**: Records de pacientes, resultados de estudios clínicos, etc.

# Ejemplos

**Colisionador de Hadrones en Suiza**

Investigación de la naturaleza de las partículas

Durante sus experimentos captura más

de **40 terabytes de datos por segundo**!

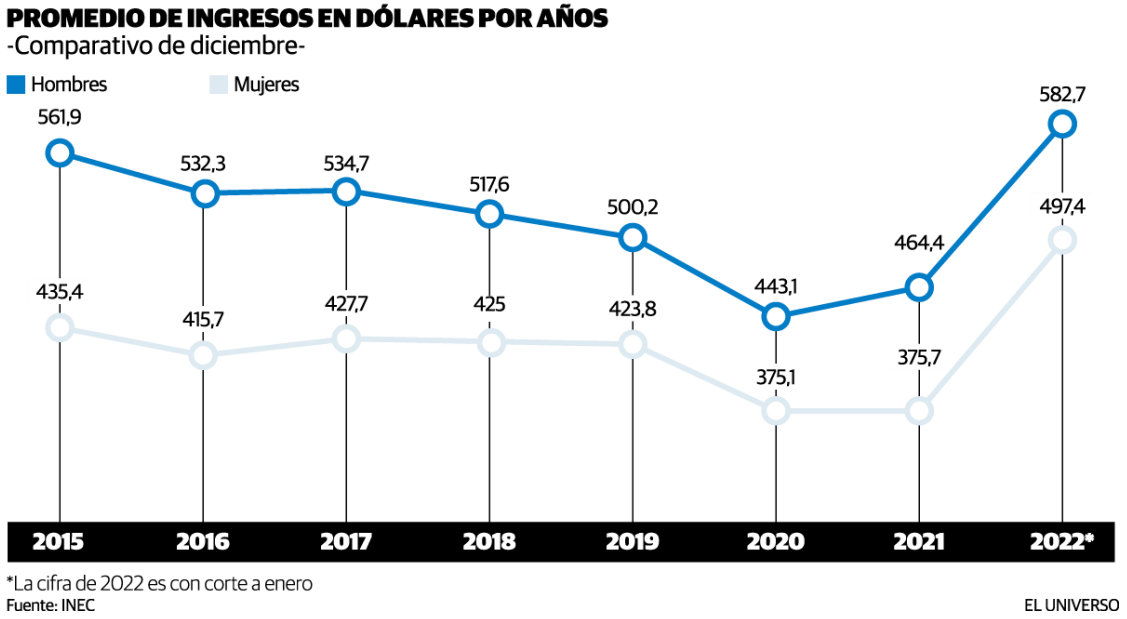
# Ejemplos

## El INEC captura datos socioeconómicos

Durante un censo por ejemplo la entidad

Captura información de millones de ecuatorianos!

Adicionalmente genera reportes periódicos de otros datos micro y macro económicos!



# Ejemplos

## Deportes

Durante un evento deportivo se capturan muchos números que permiten definir el rendimiento de los atletas.


| # | Delegaciones | Total | | | | Masculina | | | | Femenina | | | | Mixta | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | T | | | | T | | | | T | | | | T |
| 1 | Colombia | 171 | 104 | 79 | 354 | 80 | 49 | 37 | 166 | 81 | 47 | 37 | 165 | 10 | 8 | 5 | 23 |
| 2 | Venezuela | 61 | 69 | 78 | 208 | 36 | 41 | 35 | 112 | 22 | 28 | 39 | 89 | 3 | 0 | 4 | 7 |
| 3 | Ecuador | 40 | 51 | 55 | 146 | 17 | 19 | 24 | 60 | 19 | 26 | 27 | 72 | 4 | 6 | 4 | 14 |
| 4 | Chile | 37 | 49 | 71 | 157 | 17 | 22 | 41 | 80 | 19 | 26 | 28 | 73 | 1 | 1 | 2 | 4 |
| 5 | Perú | 33 | 40 | 72 | 145 | 10 | 17 | 33 | 60 | 21 | 20 | 35 | 76 | 2 | 3 | 4 | 9 |
| 6 | República Dominicana | 22 | 24 | 44 | 90 | 12 | 9 | 22 | 43 | 8 | 14 | 18 | 40 | 2 | 1 | 4 | 7 |
| 7 | Guatemala | 7 | 18 | 23 | 48 | 0 | 9 | 7 | 16 | 6 | 8 | 11 | 25 | 1 | 1 | 5 | 7 |
| 8 | Paraguay | 7 | 15 | 15 | 37 | 4 | 8 | 7 | 19 | 3 | 5 | 8 | 16 | 0 | 2 | 0 | 2 |
| 9 | Panamá | 5 | 8 | 20 | 33 | 2 | 3 | 9 | 14 | 3 | 5 | 11 | 19 | 0 | 0 | 0 | 0 |
| 10 | El Salvador | 4 | 5 | 5 | 14 | 3 | 1 | 3 | 7 | 0 | 2 | 2 | 4 | 1 | 2 | 0 | 3 |
| 11 | Bolivia | 2 | 6 | 6 | 14 | 1 | 4 | 4 | 9 | 1 | 2 | 1 | 4 | 0 | 0 | 1 | 1 |
| | Total | 389 | 389 | 468 | 1246 | 182 | 182 | 222 | 586 | 183 | 183 | 217 | 583 | 24 | 24 | 29 | 77 |

# Datos

Entonces, como podemos ver, toda clase de información capturada puede ser considerado un dato.

Lo importante es que con información tenemos el poder de crear nueva información.

Aplicar operaciones estadísticas, generar reportes, visualizaciones, etc.

# Big Data

El término que describe cantidades muy grandes (muy muy grandes) de información.

En algunos campos Big Data es aprovechada con el fin de general modelos estadísticos que describan el mundo de manera muy efectiva.

Almacenar y operar en Big Data requiere un conocimiento extendido y es un campo bastante grande.

En este curso nos enfocamos en aprender como operar con datos básicos, pero siempre es posible aprender más de esto en nuestra propia cuenta.

¿Qué es el big data? | Oracle España

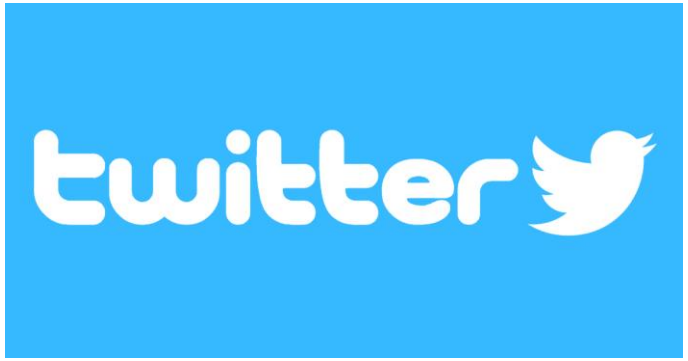
# Datos

A medida que el tiempo pasa, y nuestro mundo se digitaliza más y más, existen cada vez más fuentes de datos disponibles.

**Una gran oportunidad!**

# Datos como producto

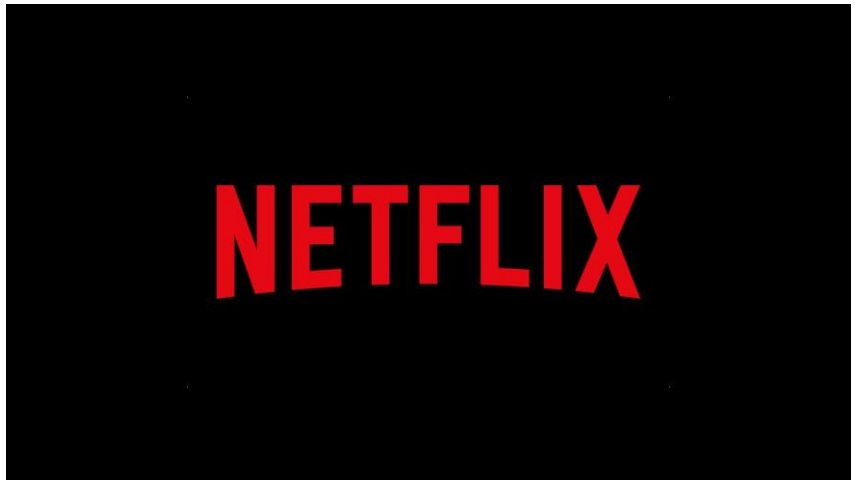
- Personalización de contenido
- Detección de spam
- Generación de temas tendencias




**400 millones de usuarios generando muchísimos datos aprovechables!**

# Datos como producto

- Películas recomendadas al usuario
- Películas y series similares
- Categorización automática



# Datos como producto

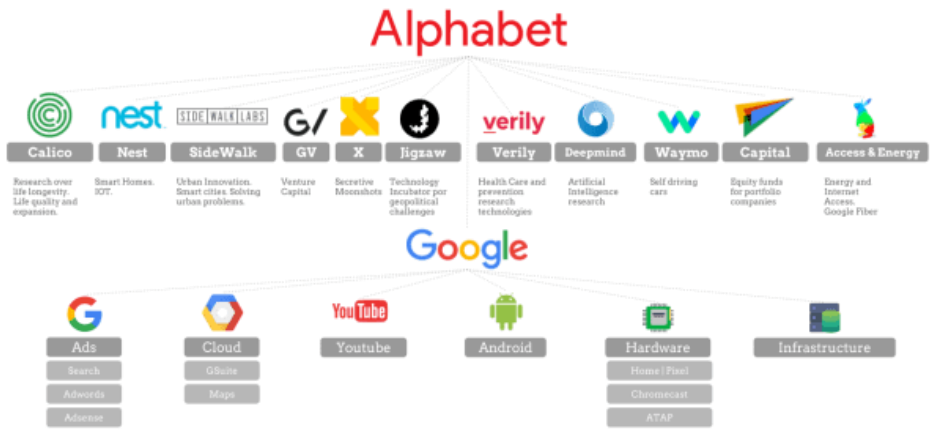
- Gente que podríamos conocer
- Temas recomendados
- Clasificación de contenido
- Determinación de preferencias de usuario (Facebook

por ejemplo puede determinar de manera

automática que tan probable es que estés casado!)


# Datos como producto

- La mayor parte de productos de Alphabet usan datos para proveer diferentes funcionalidades!
- **Google Ads**: Publicidad personalizada
- **Waymo**: Vehículos autónomos! Muchísimos datos son requeridos para crearlos.
- **Google Analytics**: Herramientas para análisis de información de usuario.



# Datos como producto

Es decir, los datos nos permiten producir recursos de primera clase gracias a la tecnología y capacidades computaciones de estos momentos!

# Trabajando con datos

Ya sea como ingeniero trabajando en productos que empleen datos, o un analista de oficina que produzca reportes numéricos en Excel, todos estos roles emplean datos con el fin de producir algo que tenga valor en el contexto correspondiente.

Muchos trabajos: Científico de datos, analista de datos, analista de bases de datos, etc.

# La vida del análisis

Fuentes de Datos



Obtener -> Limpiar -> Integrar -> Analizar -> Visualizar

Reportes, exploración, etc.

# Trabajando con datos

Un analista de datos simplemente trabajará en las diferentes partes de aquel gráfico!

Por ejemplo, antes de las computadoras uno podía manualmente analizar información y crear estimaciones.

Los computadores simplemente nos permiten hacerlo de manera más eficiente.

# Trabajando con datos

De hecho, uno no necesita programar. Existen muchas utilidades que no requieren que el analista tenga conocimientos avanzados de bases de datos o lenguajes!

Por ejemplo Excel. Funciona como una base de datos, y a su vez nos permite realizar operaciones matemáticas.

# ¿Por qué usar Python?

Por su facilidad de uso, una persona puede fácilmente entender el lenguaje y aprovecharlo para realizar operaciones analíticas de manera más eficiente.

Para empezar es gratuito y al ser un lenguaje, permite fácil colaboración entre los diferentes usuarios del mismo.

# Python para análisis de datos

La presencia de muchas librerías permite empoderar al analista mediante utilidades que interactúen con distintos entornos (leer archivos, leer bases de datos, etc.)

Herramientas como Excel por ejemplo son bastante útiles en ciertos contextos, pero a medida que los sets de datos se vuelven más grandes, la utilidad pierde sus beneficios.

# Python para análisis de datos

Podemos crear productos de datos! Es decir, crear productos cuyo fin aproveche los datos con fines específicos.

Por ejemplo, crear un modelo que permita obtener la probabilidad de que alguien le guste cierta película dependiendo de su historia.

Esto es algo imposible con Excel.

# La vida del análisis

Fuentes de Datos

**Obtener** -> Limpiar -> Integrar -> Analizar -> Visualizar

Reportes, exploración, etc.

En Python esto simplemente identifica los diferentes mecanismos disponibles para acceder a datos, ya sea por medio de archivos, bases de datos, o servicios web.

# La vida del análisis

Fuentes de Datos



Obtener -> **Limpiar** -> Integrar -> Analizar -> Visualizar

Reportes, exploración, etc.

Los datos vienen en muchas formas, a veces estructurados, a veces no. Limpiar se refiere al proceso de estandarizarlos en forma que puedan ser utilizados para el análisis correspondiente.

# La vida del análisis

Fuentes de Datos



Obtener -> Limpiar -> **Integrar** -> Analizar -> Visualizar

Reportes, exploración, etc.

Integrar simplemente hace referencia a unificar las fuentes de datos requeridas en el contexto analítico.

# La vida del análisis

Fuentes de Datos



Obtener -> Limpiar -> Integrar -> **Analizar** -> Visualizar

Reportes, exploración, etc.

Analizar hace referencia a la información que queremos obtener de esos datos, esto depende mucho del problema en el que estamos enfocándonos.

# La vida del análisis

Fuentes de Datos



Obtener -> Limpiar -> Integrar -> Analizar -> Visualizar

Reportes, exploración, etc.

Visualizar hace referencia a las distintas formas en las que podemos explorar nuestro análisis en el contexto actual, por ejemplo creando gráficos, o imprimiendo tablas numéricas.

# La vida del análisis

Fuentes de Datos



Obtener -> Limpiar -> Integrar -> Analizar -> Visualizar

Reportes, exploración, etc.

El objetivo final del análisis es siempre contestar una o más preguntas. La forma en que esto se haga depende del problema y la razón por la que lo ejecutamos.

# Análisis

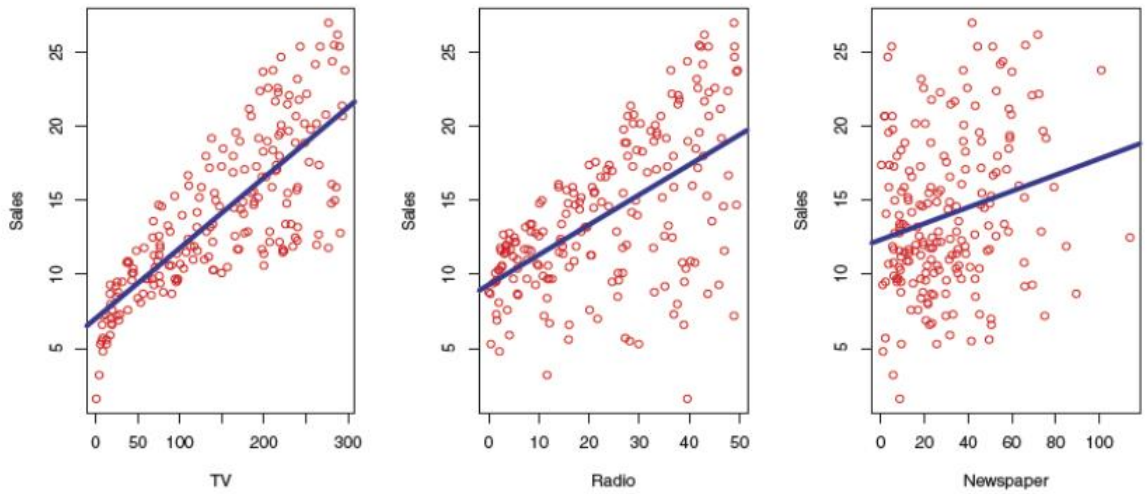
¿Pero a qué nos referimos específicamente cuando decimos análisis?

Es nada más el proceso mediante el cual obtenemos nueva información a través de métodos analíticos.

Esto puede ser algo simple como métodos estadísticos base (promedios, medianas), modelos estadísticos avanzados (regresiones, inferencias), predicciones, o inclusive análisis visual (contestar preguntas viendo los datos obtenidos a mano)!

# Análisis

Ejemplo, ¿Existe una asociación entre publicidad y ventas que permita a una compañía determinar como incrementar los presupuestos para maximizar las ventas?



# Análisis

Al ser una clase introductoria, nos enfocaremos enteramente en las utilidades disponibles en Python para operar en sets de datos de manera básica.

Pero el campo es bastante grande y por lo tanto podemos continuar aprendiendo!

A Beginner's Guide to Data Analysis in Python | by Natassha Selvaraj | Towards Data Science

Data Analytics Definition (investopedia.com)

The Age Of Analytics And The Importance Of Data Quality (forbes.com)

# El rol de un analista

Alguien que obtiene información de grandes cantidades de datos complejos!

Todo es contextual, un analista generalmente tiene que tener más habilidades aparte de saber programar.

Por ejemplo experiencia de dominio, conocimiento de estadísticas, capacidad de entender la mejor forma de visualizar información, capacidad de comunicar!

# Datos

Una vez que hemos decidido que queremos analizar información, es importante entender dos cosas.

1. ¿Dónde se encuentra esta información?
2. ¿Cómo debo ajustarla para cumplir mis objetivos?

# Almacenamiento

Los datos pueden almacenarse de diferentes formas, y es importante entender que dependiendo de la clase de datos, es posible que necesitemos realizar diferentes procesos para extraer dicha información.

# Almacenamiento: Bases de datos

Una base de datos es una colección organizada de información estructurada almacenada electrónicamente, y usualmente gestionada por un sistema de gestión de bases de datos (DBMS).

Comúnmente, los datos en una base de datos se modelan en tables con columnas y filas, denominándose bases de datos relacionales (relaciones entre tablas). Por ejemplo MySQL, Postgres, etc.

Sin embargo, existen bases de datos no relacionales que almacenan la información de manera diferente, por ejemplo como objetos puros (mongo, cassandra, etc.) o en una relación de grafos (neo4j).

# Almacenamiento: Bases de datos

Las bases de datos relacionales generalmente permiten acceder y gestionar la información dentro de las mismas utilizando un lenguaje conocido como SQL.

SQL permite que tanto un usuario como una aplicación accedan a la información en las mismas de manera estructurada!

# Almacenamiento: Bases de datos

Generalmente todos los lenguajes de programación incluyen una o más librerías que permiten a un usuario interactuar con bases de datos.

En este curso introduciremos como conectarse a bases de datos relacionales mediante Python, sin embargo existen de igual manera mecanismos muy bien documentados para interactuar con bases de datos NoSQL (nombre utilizado para todas aquellas que no siguen el modelo relacional).

# sqlite3 en Python

**sqlite3** es una base de datos relacional muy sencilla, no requiere un motor o servidor ya que los datos de la misma viven uno o más archivos! Python incluye una librería de manera predeterminada!

```
import sqlite3
from sqlite3 import Error

def crear_conexion(parametros):
    conexion = None
    try:
        conexion = sqlite3.connect(parametros)
        print("Conexión exitosa")
    except Error as e:
        print(f"Error '{e}'")

    return conexion

conexion =
crear_conexion("ubicacion/del/archivo.sqlite")
conexion.execute("select * from tabla")
resultados = conexion.fetchall()
print(resultados)
conexion.close()
```

[sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.10.5 documentation]

# MySQL en Python

A diferencia de sqlite3, Python no incluye módulos para interactuar con MySQL, pero existen muchos disponibles en el repositorio libre!

```
python -m pip install mysql-connector-python
```

```python
import mysql.connector
from mysql.connector import Error

def crear_conexion(servidor, usuario, clave):
    conexion = None
    try:
        conexion = mysql.connector.connect(
            host=servidor,
            user=usuario,
            passwd=clave
        )
        print("Conexión a MYSQL existosa")
    except Error as e:
        print(f"Error '{e}'")
    return conexion

conexion = crear_conexion("localhost", "miusuario", "")
cursor = conexion.cursor()
cursor.execute("SELECT * FROM tabla")
resultado = cursor.fetchall()
conexion.close()
```

[MySQL :: MySQL Connector/Python Developer Guide]

# sqlalchemy

Un kit de utilidades para interactuar con bases de datos, ofrece la flexibilidad de trabajar con bases de datos relacionales de manera más eficiente, en particular dentro de aplicaciones complejas.

- Ofrece un modelo relacional que permite operar en una base de datos usando objetos nativos a Python.
- Un sistema eficiente de distribución de objetos y operaciones relacionales usando Python en lugar de SQL.
- Generación e introspección de bases de datos.

SQLAlchemy · PyPI

# Archivos

A pesar de que las bases de datos son maneras optimas de mantener información, los archivos son una forma también muy común de almacenar objetos de datos, particularmente en el contexto de análisis!

SQLAlchemy · PyPI

# Archivos CSV

CSV – comma separated values

Un formato tabular en el que podemos representar una table de información mediante una serie de comas que indiquen la separación entre columnas.

Al ser un formato de texto plano, el mismo puede leerse de manera sencilla mediante cualquier utilidad, e inclusive por programas como Excel.

# Archivos CSV

Un archivo csv es bastante simple, por ejemplo:

```
Usuario;Identificador;Nombre;Apellido
booker12;9012;Rachel;Booker
grey07;2070;Laura;Grey
johnson81;4081;Craig;Johnson
jenkins46;9346;Mary;Jenkins
smith79;5079;Jamie;Smith
```

La primera línea puede definir los encabezados de cada columna, así como no definir nada. Es necesario que entendamos la estructura del mismo cuando lo estemos analizando!

# Archivos CSV: Python

Al ser un archivo de texto plano, es bastante fácil leer este archivo como cualquier otro:

```
with open("ubicacion.csv") as archivo:
    for fila in archivo:
        # Dividiamos las columnas usando la , pero tambien puede ser ;
        columnas = fila.split(",")
        print(columnas)
```

# Archivos CSV: Python

Sin embargo, existen muchas particularidades que debemos controlar,  y por eso es mejor usar la librería incluida con Python.

```python
import csv
with open("ubicacion.csv") as archivo:
    # Definir el delimitador que separa las columnas
    filas = csv.reader(archivo, delimiter=';')
    for fila in filas:
        # Automáticamente se convirtieron en columnas
        print(fila[0])

# Podemos fácilmente agregar filas!
with open('ubicacion.csv', 'w') as archivo:
    escritor = csv.writer(archivo, delimiter=';')
    escritor.writerow(['Pollo', 'Papas'])
```

csv — CSV File Reading and Writing — Python 3.10.5 documentation

# Archivos

Saber leer archivos CSV es muy importante ya que una gran cantidad de sets de datos pueden ser definidos de esta manera. De hecho librerías como pandas permiten de manera sencilla leer los mismos sin tener que gestionar el recurso de manera manual!

Sin embargo, cuando hablamos de cantidades muy extensas (muchos GBs), es importante considerar otros formatos más extensibles que no sean necesariamente texto plano, entre ellos Parquet, que no cubriremos en esta clase por ser un concepto más avanzado, pero que no va mal conocer.

Parquet – Databricks

# Servicios web

Tal como aprendimos ayer, es posible comunicarse con servicios web alrededor del mundo.

Muchos sets de datos pueden ser obtenidos mediante servicios web.

# Datos inesperados

A veces los datos no se encuentran de forma estructurada, o debemos recogerlos mediante herramientas que permitan extraerlos de manera eficiente.

Por ejemplo, a veces la información se encuentra en archivos PDF, o en páginas web, por lo tanto debemos procesar el contenido con librerías especializadas a manera de poder extraerla.

# Trabajando con datos

Recordemos que tanto el intérprete interactivo, como herramientas externas (cuadernos Jupyter) nos permiten interactuar con Python de manera visual.

Estos métodos permiten que mediante librerías especializadas, experimentemos de manera más eficiente con la información que estamos analizando.

# Arreglos, matrices y tablas

Python por defecto incluye el concepto de listas, lo que no es más que una colección de valores identificados por una variable.

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

Recordemos que una lista puede contener cualquier objeto dentro de sí, inclusive otras listas!

```
tabla = [[1, 2], [4, 5], [4, 2]]
```

Por lo que de manera natural podemos representar listas en este lenguaje.

# Listas en Python

Sin embargo, como podemos recordar, al ser un lenguaje dinámico, operaciones que trabajen con listas suelen perder eficiencia a medida que el tamaño de las mismas crece.

Leer un archivo CSV en forma de una lista de listas puede ser bastante costoso e ineficiente cuando trabajamos con miles de filas!

**Los sets de datos pueden tener millones de filas y cientos de columnas!**

# Listas en Python: numpy

**numpy** es uno de los paquetes más fundamentales para análisis de datos en Pyhton ya que provee objetos de arreglos multidimensionales que utilizan las mismas interfaces de las listas regulares, pero que han sido optimizados por eficiencia (Python usa librerías compiladas en un lenguaje mucho más bajo nivel como C o C++).

**Los arreglos multidimensionales** en **numpy** pueden ser por lo tanto usados para operaciones matemáticas complejas, álgebra lineal, etc.

NumPy

# numpy

Las operaciones en arreglos numpy son MUCHO más rápidas, nótese en algunos casos más de 100 veces!

```
import numpy as np

# Creando un arreglo de 1000000 de elementos
a1 = np.arange(1_000_000)
tradicional = list(range(1_000_000))

%timeit a1 = a1 * 2
628 us per loop

%timeit tradicional = [x * 2 for x in tradicional]
38 ms per loop
```

NumPy

# numpy

La importancia de numpy es por lo tanto que puede ser aprovechado como la base del análisis en muchos contextos al facilitar operaciones altamente optimizadas.

En el contexto de machine learning, procesamiento de imágenes, o en el contexto de exploración de datos mediante otras librerías como **pandas**.

NumPy

# pandas

El paquete **pandas** es una herramienta esencial para el análisis y manipulación de datos tabulares!

**pandas** toma ventaja de **numpy** para realizar operaciones tabulares de manera muy eficiente y por lo tanto comparte muchas de las interfaces.

Introducción a Pandas, la librería de Python para trabajar con datos (profile.es)

# pandas

Pandas viene por defecto con varias estructuras de datos esenciales, pero nos enfocaremos en el DataFrame.

- **DataFrame:** Una estructura tabular que contiene una colección ordenada de columnas.

```
datos = {"ciudad": ["Quito", "Loja", "Cuenca", "Cuenca",
"Quito", "Quito"],
        "año": [2000, 2001, 2002, 2001, 2002, 2003],
        "costo": [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
```

```
In [50]: frame
Out[50]:
    ciudad  año  costo
0   Quito   2000  1.5
1   Loja    2001  1.7
2   Cuenca  2002  3.6
3   Cuenca  2001  2.4
4   Quito   2002  2.9
5   Quito   2003  3.2
```

El valor de un DataFrame es que nos permite realizar operaciones exploratorias de manera sencilla!

# pandas

El método **head** muestra la cabeza de la tabla, es decir las primeras 5 filas. Es útil para visualizar el contenido de alguna de las filas.

```
In [51]: frame.head()
Out[51]:
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
```

# pandas

El método **tail** muestra lo opuesto, las últimas 5 filas de la tabla.

```
In [52]: frame.tail()
Out[52]:
    state  year  pop
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
5  Nevada  2003  3.2
```

# pandas

Al crear un DataFrame podemos especificar el orden de las columnas explícitamente.

```
import pandas

datos = {
    "ciudad": ["Loja", "Cuenca", "Quito"],
    "poblacion": [10000, 10000000, 100000000],
    "tiene_volcan": ["no", "no", "si"]
}

marco_de_datos = pandas.DataFrame(datos)
print(marco_de_datos)
marco_de_datos = pandas.DataFrame(datos,
columns=["poblacion", "ciudad", "tiene_volcan"])
print(marco_de_datos)
```

```
   ciudad  poblacion tiene_volcan
0    Loja      10000           no
1  Cuenca   10000000           no
2   Quito  100000000           si

   poblacion  ciudad tiene_volcan
0      10000    Loja           no
1   10000000  Cuenca           no
2  100000000   Quito           si
```

# pandas

Filas pueden ser obtenidas mediante la función iloc

```
import pandas

datos = {
    "ciudad": ["Loja", "Cuenca", "Quito"],
    "poblacion": [10000, 10000000, 100000000],
    "tiene_volcan": ["no", "no", "si"]
}

marco_de_datos = pandas.DataFrame(datos)
print(marco_de_datos.iloc[1])
```

```
ciudad            Cuenca
poblacion       10000000
tiene_volcan          no
Name: 1, dtype: object
```

# pandas

Columnas  se pueden cambiar por asignación, por ejemplo:

```
import pandas

datos = {
    "ciudad": ["Loja", "Cuenca", "Quito"],
    "poblacion": [10000, 10000000, 100000000],
    "tiene_volcan": ["no", "no", "si"]
}

marco_de_datos = pandas.DataFrame(data=datos)
marco_de_datos["poblacion"] = marco_de_datos["poblacion"] * 2
print(marco_de_datos)

   ciudad  poblacion tiene_volcan
0    Loja      20000           no
1  Cuenca   20000000           no
2   Quito  200000000           si
```

# pandas

Como notamos, es fácil convertir un diccionario o lista Python en un DataFrame.

Es también posible cargarlo directamente desde un archivo .csv!

```
In [14]: pd.read_csv("ubicacion_sin_encabezado.csv", header=None)
Out[14]:
   0   1   2   3       4
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo

In [15]: pd.read_csv("ubicacion.csv", names=["a", "b", "c", "d", "message"])
Out[15]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

# pandas

Podemos realizar operaciones estadísticas como el promedio (mean), la moda (mode), o la mediana (median)!

```
import pandas

datos = {
    "edades": [20, 30, 35],
    "poblacion": [10000, 10000000, 100000000],
}

marco_de_datos = pandas.DataFrame(data=datos)

print(marco_de_datos.mean())

edades       2.833333e+01
poblacion    3.667000e+07
dtype: float64
```

# pandas

Es posible extrar el arreglo numpy con la función

```
frame.to_numpy()
```

Esto devuelve un arreglo bi-dimensional que incluye los datos de la tabla.

# pandas

Es posible de igual manera combinar distintos DataFrames de manera similar a la que uno lo haría en una base de datos!

```
df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})
df1.merge(df2, how='inner', on='a')
```

pandas.merge — pandas 1.4.3 documentation (pydata.org)

# pandas

Es posible extrar el arreglo numpy con la función

```
frame.to_numpy()
```

Esto devuelve un arreglo bi-dimensional que incluye los datos de la tabla.

# pandas

Mediante los patrones de acceso de pandas, podemos aplicar diferentes operaciones de manera eficiente!

Pero a veces es necesario también graficar diferentes patrones en los datos.

# pandas

Mediante los patrones de acceso de pandas, podemos aplicar diferentes operaciones de manera eficiente!

Pero a veces es necesario también graficar diferentes patrones en los datos.

# Visualización de datos

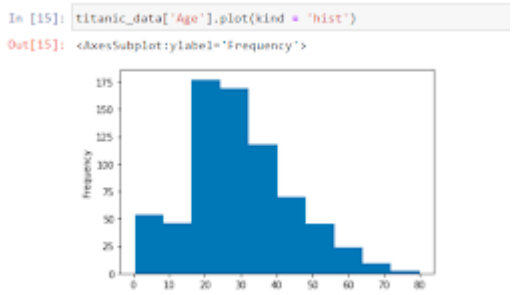
Mediante los patrones de acceso de pandas, podemos aplicar diferentes operaciones de manera eficiente!

Pero a veces es necesario también graficar diferentes patrones en los datos.

# plot

Pandas viene con una utilidad de visualización dentro de su librería. Dependiendo del tipo uno puede especificar el tipo de grafo:

- distribución
- Pastel
- Densidad
- Linea
- Histogramas
- Etc. etc.



# matplotlib

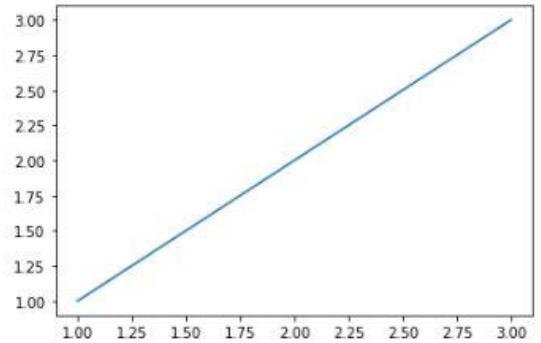
Una librería muy poderosa para visualización de matrices y datos tabulares. Usada en contexto de análisis de datos!

Al ser optimizada, también puede usarse en contextos de aplicaciones para generar visualizaciones automatizadas (por ejemplo, generar un gráfico de manera dinámica en un servidor y emitirlo a una página web).

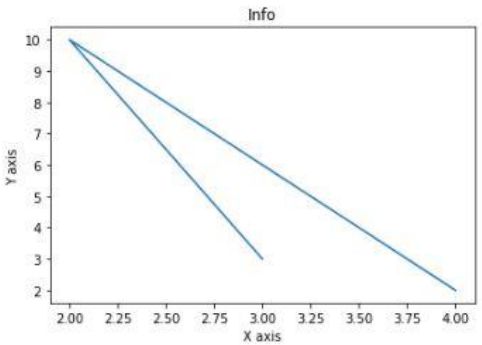
# matplotlib

Podemos de manera sencilla emitir las series de datos.

```
In [3]: from matplotlib import pyplot as plt

        #plotting to our canvas
        plt.plot([1,2,3],[1,2,3])

        #showing what we plotted
        plt.show()
```




```
In [4]: from matplotlib import pyplot as plt

        x = [4,2,3]
        y = [2,10,3]
        plt.plot(x,y)
        plt.title('Info')
        plt.ylabel('Y axis')
        plt.xlabel('X axis')
        plt.show()
```



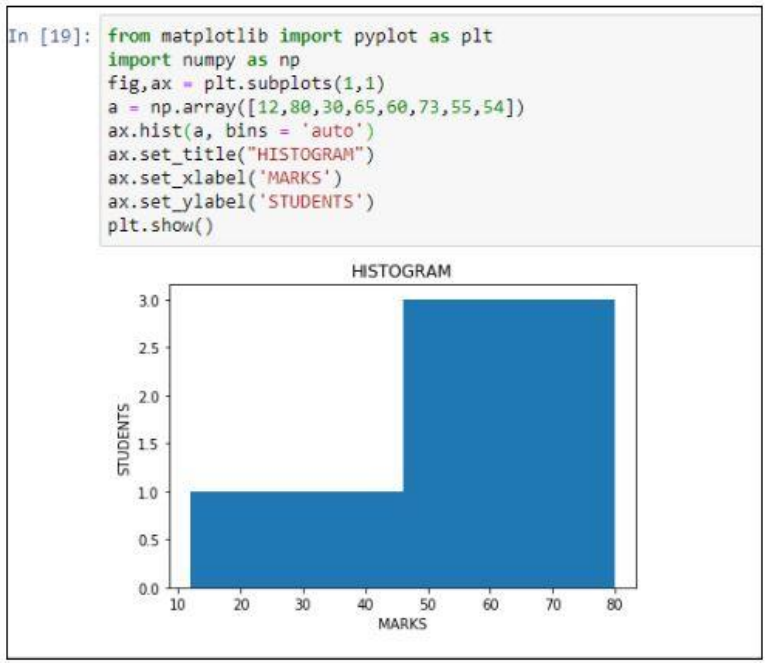
# matplotlib

Naturalmente es posible trabajar con arreglos de **numpy**, por lo que en los contextos de pandas, podemos fácilmente extraer toda la información necesaria usando los métodos ya aprendidos.

# matplotlib

Un ejemplo de un gráfico de barras usando una serie de numpy.

```
In [19]: from matplotlib import pyplot as plt
         import numpy as np
         fig,ax = plt.subplots(1,1)
         a = np.array([12,80,30,65,60,73,55,54])
         ax.hist(a, bins = 'auto')
         ax.set_title("HISTOGRAM")
         ax.set_xlabel('MARKS')
         ax.set_ylabel('STUDENTS')
         plt.show()
```



# matplotlib

Bastante poderosa y con una documentación muy extensa! Pero regularmente es Google que tiene muchas de las respuestas.

Matplotlib documentation — Matplotlib 3.5.2 documentation

# matplotlib

Finalmente, estas se pueden de igual manera guardar en el disco!

```
import matplotlib.pyplot

figura, eje = matplotlib.pyplot.subplots( nrows=1, ncols=1 )  # crear figura & 1 eje
eje.plot([0,1,2], [10,20,3])
figura.savefig('ubicacion/disco.png')   # guardar en archivo
matplotlib.pyplot.close(figura)    # cerrar
```

# Aprendiendo

Los temas cubiertos aquí son muy generales, sin embargo el mundo de análisis en Python meses en cubrirse!

Un buen recurso gratuito

https://wesmckinney.com/book

En inglés, pero podemos traducirlo muy fácilmente mediante Google!

# Conclusión

Preguntas?

# Siguiente clase

- Última clase antes del proyecto
- Temas complementarios de Python
- Automatización de tareas del Sistema
- Unas cuantas librerías útiles
- Dudas sobre el lenguaje y el proyecto