

INTRODUCCIÓN



Inteligencia Artificial

CEIA - FIUBA

Dr. Ing. Facundo Adrián
Lucianna

INTRODUCCIÓN



8 clases teórico-práctica



Clases con diapositivas y desarrollo en notebooks



Estructura de las clases

10 minutos de repaso de clase anterior

3 bloques de 50 minutos de clase teórico-prácticas

2 recreos de 10 minutos

INTRODUCCIÓN

Aula virtual:

- <https://campusposgrado.fi.uba.ar/course/view.php?id=253>

Repositorio de la materia:

- https://github.com/FIUBA-Posgrado-Inteligencia-Artificial/intro_ia

Consultas

- Foro de consulta en el aula virtual

Correo

- Facundo Adrián Lucianna: facundolucianna@gmail.com

Discord para chatear y tomar mate ☕

- Servidor de Discord: <https://discord.gg/rGbKPE8P>

EVALUACIÓN

3 Trabajos Prácticos con entrega a 14 días posteriores de cuando fue presentado. Los trabajos deben ser grupales (máximo 6 personas). Los TPs se enuncian en clase 2, 5 y 6.

Los trabajos prácticos se aprueban con 4. Tiene una instancia de corrección 7 días después de la última clase y se puede corregir solamente los trabajos con menos de 4.

Los tres TPs tienen que estar aprobados. Un TP no entregado en fecha se considera desaprobado.

La nota final es la **mediana** de las notas de los TPs.

La entrega es por el aula virtual, ya sea el envío del contenido o el link a repositorio (de GitHub o GitLab) con el trabajo.

Criterio de evaluación:

https://github.com/FIUBA-Posgrado-Inteligencia-Artificial/intro_ia/blob/main/CriteriosAprobacion.md o en aula virtual

```
if city != "all":
    df_operation_table = df_operation_table[df_operation_table["city_id"] == city]
if trunk != "all":
    df_operation_table = df_operation_table[df_operation_table["trunk_id"] == trunk]

if df_operation_table.empty:
    logg.warning("No data to be read", status_code=404, reason="No data to be read",
                country=country, city=city, trunk=trunk)
    return 404, "No data to be read", None

# Force the datatype to avoid problems
df_operation_table['app_store_id'] = df_operation_table['app_store_id'].astype(str, errors='ignore')
df_operation_table['brand_id'] = df_operation_table['brand_id'].astype(np.int64, errors='ignore')
df_operation_table['branch_id'] = df_operation_table['branch_id'].astype(np.int64, errors='ignore')
df_operation_table['internal_store_id'] = df_operation_table['internal_store_id'].astype(str,
                                         errors='ignore')

df_operation_table['lat'] = df_operation_table['lat'].astype(np.float64, errors='ignore')
df_operation_table['lng'] = df_operation_table['lng'].astype(np.float64, errors='ignore')

# Add the main currency
df_operation_table['main_currency'] = max.main_currency(country)

# Rename columns
df_operation_table.rename(columns={"internal_store_id": "store_id"}, inplace=True)

# Make compatibility between changes in backoffice
if 'company_id' not in df_operation_table.columns:
    df_operation_table['operator_id'] = df_operation_table['operator_id'].astype(np.int64, errors='ignore')
    df.operation_table['company_id'] = df.operation_table['operator_id']
    df.operation_table['company_name'] = df.operation_table['operator_name']

else:
    df.operation_table['company_id'] = df.operation_table['company_id'].astype(np.int64, errors='ignore')

# Drop all this columns
drop_columns_list = ['operator_id', 'operator_name']
df.operation_table = df.operation_table.drop(drop_columns_list, errors='ignore', axis=1)

except Exception as e:
    logg.error("Problem loading the operation table", status_code=500, reason=e,
               path=unquote(path.snapshot_operation.as_uri()))
return 500, ("Problem loading the operation table: " + str(e) + " - path: " +
            unquote(path.snapshot_operation.as_uri())), None
```



HERRAMIENTAS

- **Lenguaje de Programación**
 - Python >= 3.10
 - Poetry / Pip / Conda para instalar librerías
 - **Librerías**
 - Numpy, Pandas, SciPy
 - Matplotlib, Seaborn
 - Scikit-learn
 - **Consola Interactiva de Python**
 - Ipython
 - Jupyter Notebook
 - **Herramientas**
 - Github para repositorios
 - **IDE recomendados**
 - Visual Studio Code
 - PyCharm Community Edition
 - Google Colab

PROGRAMA

Clase a clase

- 1 Inteligencia Artificial. Historia de la IA. Beneficios y Riesgo de la IA. Python. Librerías Asociadas
- 2 Agente de resolución de problemas. Programa de los agentes. Resolución de problemas mediante búsqueda. Algoritmo de búsqueda no informada. Algoritmo de búsqueda informada
- 3 Problemas de optimización. Algoritmos de búsqueda Local. Gradiente descendiente o Ascendente. Simulated annealing. Búsqueda Local Beam. Algoritmos Genéticos. Búsqueda en espacios continuos
- 4 Aprendizaje Automático. Datos. Formas de aprendizaje. Aprendizaje supervisado. Sesgo y varianza. Estrategias para disminuir el riesgo empírico. Métricas de clasificación

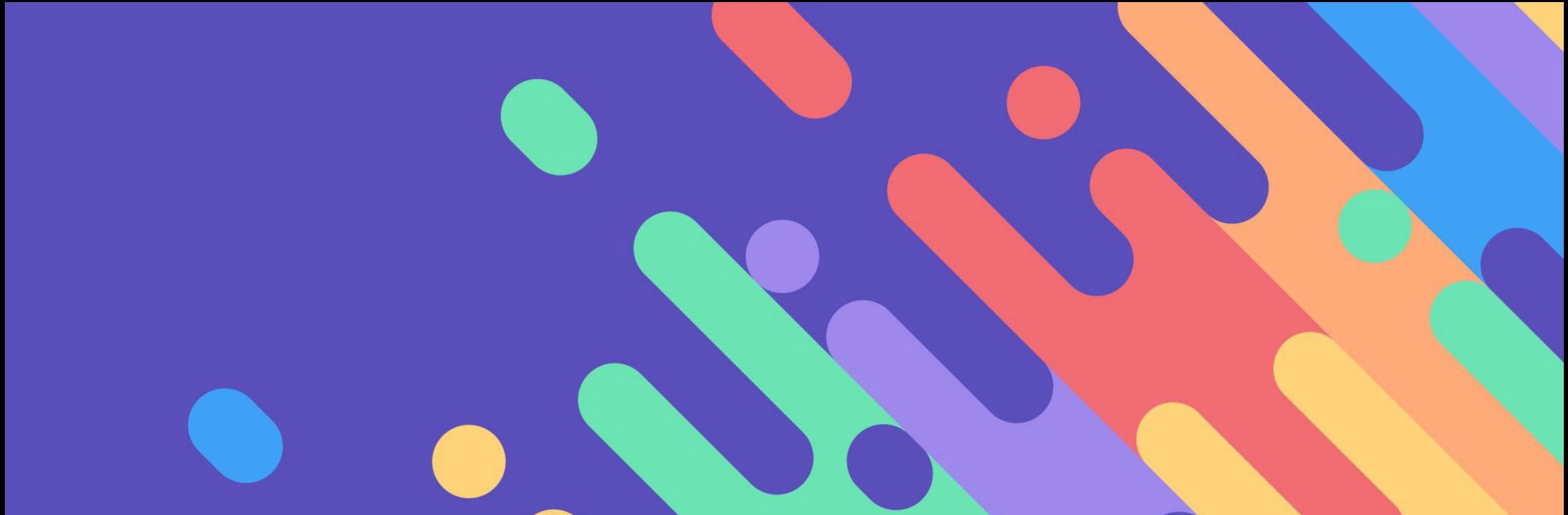
PROGRAMA

Clase a clase

- 5 Conceptos de Regresión. Regresión lineal simple y múltiple. Métodos de evaluación de regresiones.
Tratamiento de variables. Construcción de modelos. Regresión Ridge y Lasso
- 6 Conceptos de Clasificación. Regresión logística simple y múltiple. Regresión logística multi-clase.
Curva ROC. Teorema de Bayes. Clasificador bayesiano ingenuo
- 7 Aprendizaje por Refuerzo. Proceso de decisión de Márkov. Ecuación de Bellman. Algoritmos basados en política o en valor. Soluciones iterativas. Can it Play DOOM? Aplicación de Q-learning

BIBLIOGRAFÍA

- Artificial Intelligence: A Modern Approach - Stuart Russell, Peter Norvig (Ed. Pearson)
- Artificial Intelligence Basics: A Non-Technical Introduction - Tom Taulli (Ed. Apress)
- Artificial Intelligence For Dummies - John Paul Mueller, Luca Massaron (Ed. For Dummies)
- An Introduction to Statistical Learning - Gareth James (Ed. Springer)
- The Elements of Statistical Learning - Trevor Hastie (Ed. Springer)
- Life 3.0: Being Human in the Age of Artificial Intelligence - Max Tegmark (Ed. Knopf)



INTELIGENCIA ARTIFICIAL

INTELIGENCIA ARTIFICIAL

- La primera pregunta que nos hacemos es que es la **Inteligencia Artificial (IA)**
- Como siempre en estos campos de vanguardia, no hay una sola definición.
- Según Stuart Russell y Peter Norvig:
 - A veces se define en función de:
 - La fidelidad del desempeño humano (u otro animal)
 - Hacer “lo correcto” (racionalidad)
 - También se considera una propiedad:
 - De los procesos de pensamiento y razonamiento internos.
 - Del comportamiento, es decir una característica externa.

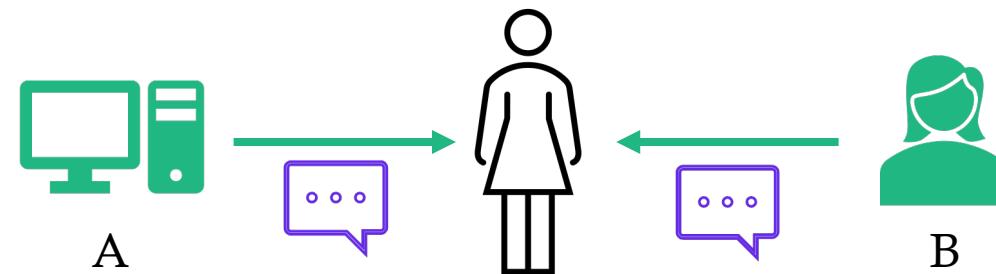
INTELIGENCIA ARTIFICIAL

Actuando humanamente

El test de Turing

En el test, un interrogador humano interactúa con dos participantes, A y B, intercambiando mensajes escritos.

Si el interrogador no es capaz de determinar qué participante, A o B, es una computadora y cuál es un ser humano, se considera que la computadora ha superado la prueba.



INT

Actuar

El test

En el te
mensaje

Si el int
ser hum

¿Carlos es una computadora o una persona?

Int: ¿Cuántas patas tiene un camello?

Carlos: Más o menos entre 2 y 4. ¿Tres, quizás? :-))) Por cierto, sigo sin saber tu especialidad...
¿o igual se me ha pasado?

Int: ¿Cuántas patas tiene un milpiés?

Carlos: Solo dos, pero los mutantes de Chernóbil pueden tener hasta cinco. Sé que se supone que tienes que engañarme.

Int: No, necesito saber que no eres un robot conversacional. Por favor, responde a la pregunta directamente: ¿Cuántas patas tiene una hormiga?

Carlos: Más o menos entre 2 y 4. ¿Tres, quizás? :-))) Vaya, qué conversación más productiva ;-)

Int: ¿Entiendes por qué hago estas preguntas tan básicas? ¿Eres consciente de que simplemente pretendo desenmascararte como robot tan rápido como sea posible, como en la película Blade Runner?

Carlos: ...espera

INTELIGENCIA ARTIFICIAL

Actuando humanamente

El test de Turing

Programar un software para pasar rigurosamente el test implica un gran trabajo. Este software debe contar con las siguientes capacidades:

- **Procesamiento natural del lenguaje** para comunicarse exitosamente en un lenguaje humano.
- **Representación de conocimiento** para almacenar lo que conoce o escucha.
- **Razonamiento automático** para responder a las preguntas y obtener nuevas conclusiones.
- **Aprendizaje automático** para adaptarse a las nuevas circunstancias y para detectar y extraer patrones.

INTELIGENCIA ARTIFICIAL

Pensando racionalmente

El filósofo Aristóteles fue el primero en intentar codificar “pensar correctamente”.

Sus **silogismos** proveyeron un patrón para estructuras argumentales que siempre llevan a conclusiones correctas dados unas premisas correctas.

“Sócrates es un hombre y todos los hombres son mortales entonces Sócrates es mortal”

Estas leyes de pensamiento derivador en el campo de la **lógica**.

En el siglo XVIII se desarrolló una notación precisa para los enunciados sobre los objetos del mundo y las relaciones entre ellos.

Para 1965, programadores pudieron resolver informáticamente cualquier problema de lógica resoluble usando la notación lógica.

INTELIGENCIA ARTIFICIAL

Pensando racionalmente

La lógica espera que el conocimiento del mundo sea cierto... algo que sabemos que no es así, el mundo es regido por la incertidumbre.

La teoría de probabilidad llena este vacío, permitiendo un razonamiento riguroso con información incierta.

Esto nos permite construir un modelo de pensamiento racional (desde la percepción hasta la comprensión de cómo funciona el mundo y hacer predicciones)

Pero esto no genera comportamientos inteligentes, ya que con solo pensar no solo nos alcanza, necesitamos actuar,

”¿Pienso, luego existo” ya no vale más?

INTELIGENCIA ARTIFICIAL

Pensando racionalmente

Un agente es algo que actúa. Un agente se espera que opere autónomamente, perciba el ambiente, persista sobre un tiempo prolongado, se adapte y cree y cumpla objetivos.

Un agente racional es aquel que llega al mejor escenario, o si hay incertidumbre, al mejor escenario esperado.

IA se ha enfocado en el estudio y construcción de agentes que hacen lo correcto. Que cuenta por hacer lo correcto es el objetivo que le proveemos al agente. Esto se llama el **modelo estándar**.

Este modelo no solo ha sido predominante en IA, sino además en *teoría del control*, en *estadística*, y *economía*.

INTELIGENCIA ARTIFICIAL

Máquinas beneficiosas

El **modelo estándar** asume que se va a un objetivo específico a resolver... algo que, en la vida real, es mucho más difícil especificar el objetivo completamente y correctamente.

INTELIGENCIA ARTIFICIAL

Máquinas beneficiosas



[D11u](#) - Trabajo propio
[CC BY-SA 4.0](#)

INTELIGENCIA ARTIFICIAL

HAL 9000

Máquinas beneficiosas

El balance de lograr un acuerdo entre nuestras preferencias y el objetivo que tiene la maquina se llama problema de alineaciones de valores.

Los valores u objetivos que damos a una máquina deben estar alineados con los del ser humano.

Si estamos en un laboratorio, hay una forma fácil de solucionar esto, resetear el sistema.

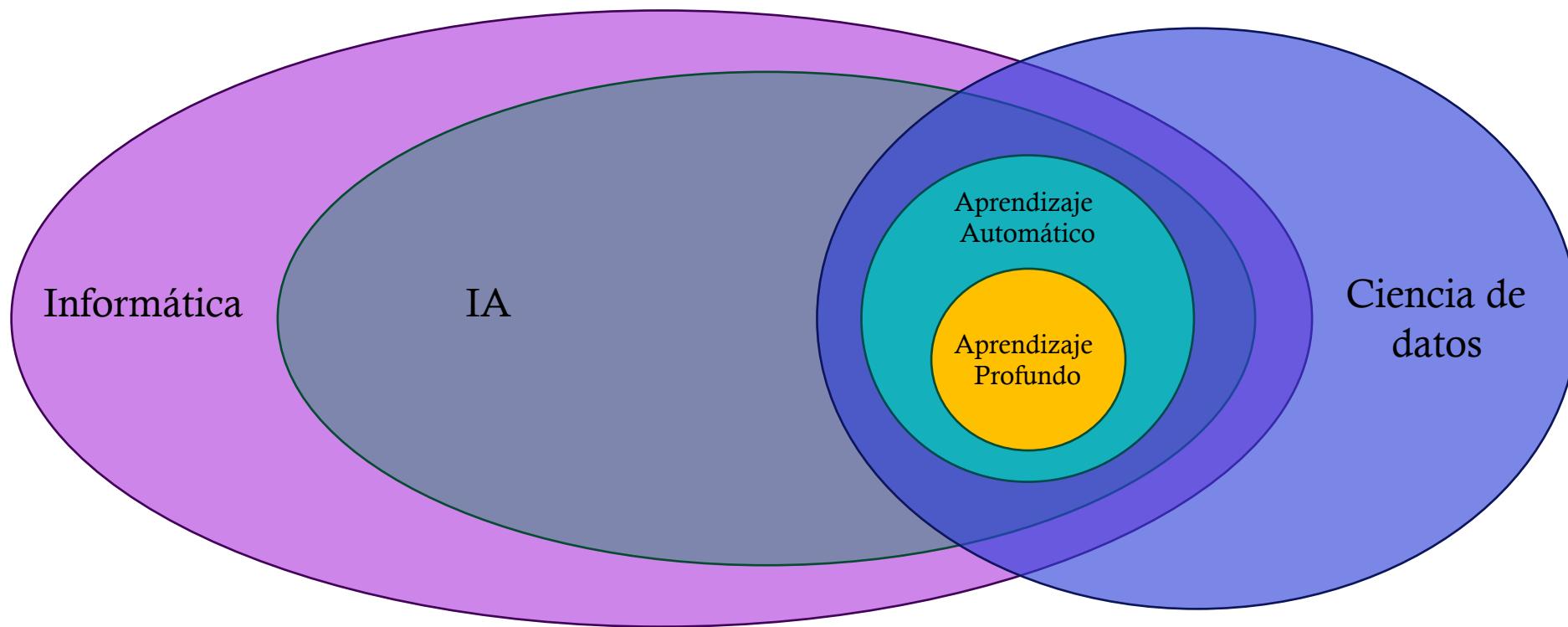
En la vida real, esto no es posible. Por lo que el modelo estándar **no** es apropiado. Se necesita una nueva formulación.

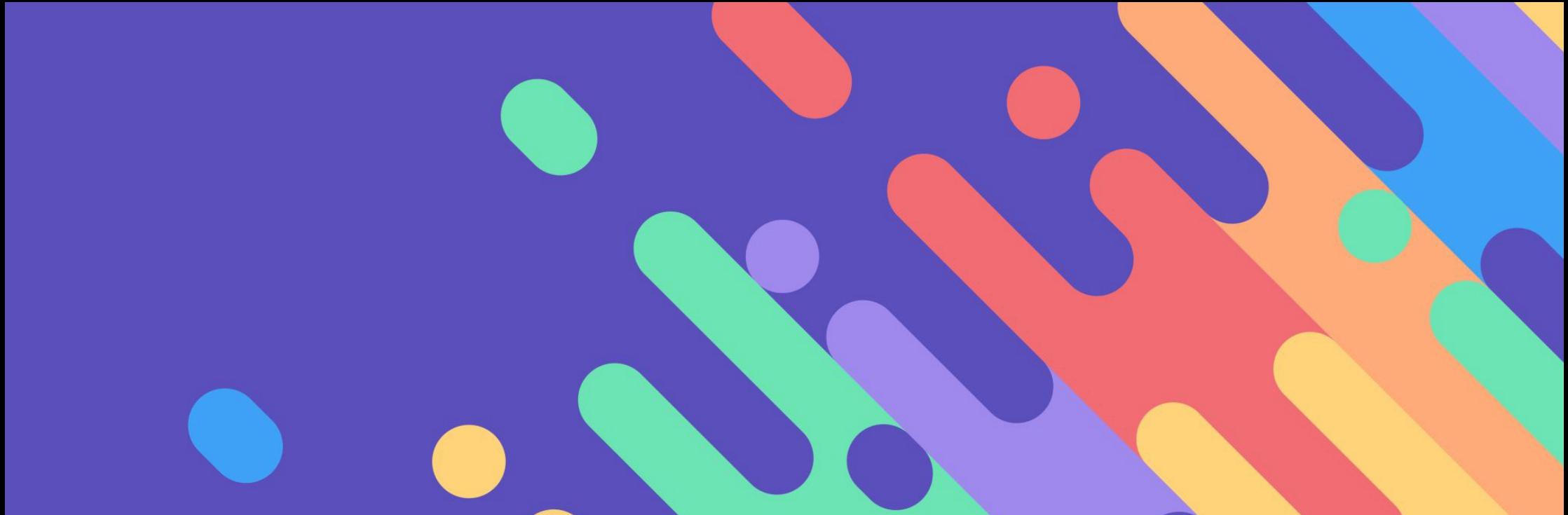
Cuando una máquina sabe que no conoce el objetivo completo, tiene un incentivo para actuar con cautela, pedir permiso, aprender más sobre nuestras preferencias a través de la observación y ceder al control humano.



INTELIGENCIA ARTIFICIAL

Campos conectados





HISTORIA DE LA IA

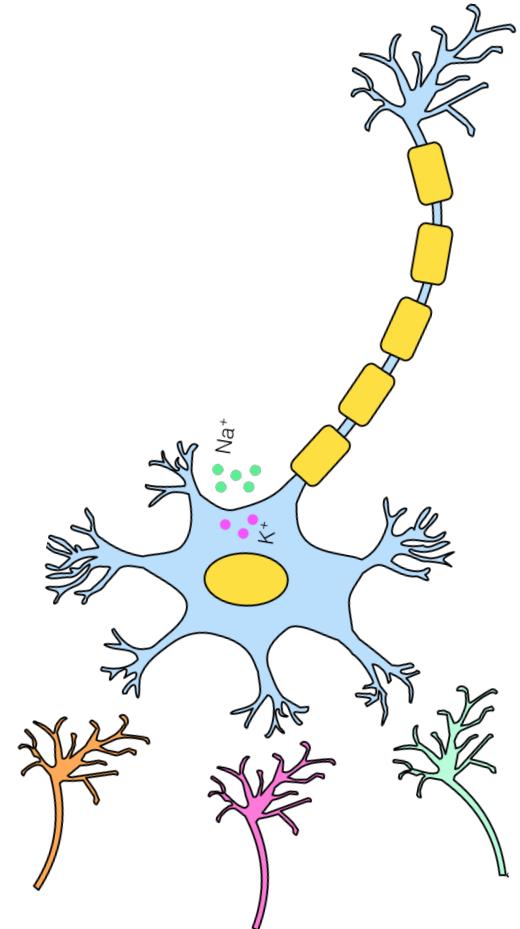
HISTORIA DE LA IA

El principio de la inteligencia artificial (1943-1956)

El primer trabajo reconocido de IA es el desarrollado por Warren **McCulloch** y **Walter Pitts** en 1943.

Modelo de neurona artificial en la cual cada una se puede *prender* o *apagar*. Se *prenden* cuando responde a la estimulación de varias neuronas vecinas.

Donald Hebb (1949) creó una regla que permitía modificar la intensidad de conexión entre neuronas (aprendizaje). Este modelo sigue siendo válido hasta el día de hoy.





HISTORIA DE LA IA

El principio de la inteligencia artificial (1943-1956)

Alan Turing dio sus primeras lecciones en IA en 1947.

Introdujo el test de Turing, aprendizaje automático, algoritmos genéticos y aprendizaje por refuerzo.

Sugirió que sería más fácil crear IA a nivel humano desarrollando algoritmos de aprendizaje y luego enseñando a la máquina en lugar de programar su inteligencia a mano.

HISTORIA DE LA IA

Entusiasmo inicial, grandes expectativas (1952-1969)

En el establishment intelectual de 1950 regia “una máquina nunca podrá hacer X”.

Los investigadores de IA respondieron naturalmente demostrando una X tras otra (juegos, rompecabezas, matemáticas y pruebas de IQ).



HISTORIA DE LA IA

Entusiasmo inicial, grandes expectativas (1952-1969)

Arthur Samuel, usando aprendizaje por refuerzo, creó un programa que podía jugar a las damas (1956). Es decir, el software aprendió por sí solo.

HISTORIA DE LA IA

Entusiasmo inicial, grandes expectativas (1952-1969)

John McCarthy (1958) creo:

1. **Lisp**: El lenguaje de programación de IA durante 30 años.
2. **Advice Taker**, un programa hipotético que encarnaría el conocimiento general del mundo y podría utilizarlo para derivar planes de acción. El programa también fue diseñado para aceptar nuevos axiomas en el curso normal de operación, permitiéndole así alcanzar competencia en nuevas áreas sin ser reprogramado.

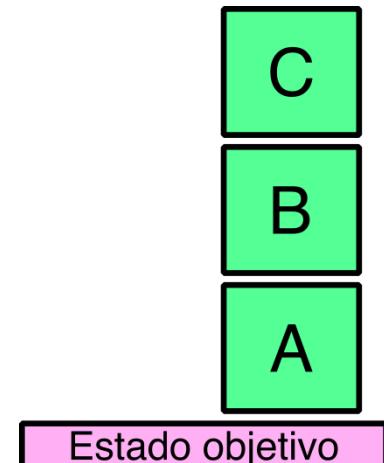
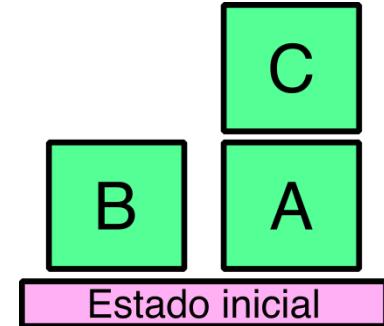
HISTORIA DE LA IA

Entusiasmo inicial, grandes expectativas (1952-1969)

Marvin Lee Minsky en MIT superviso a estudiantes que desarrollaron softwares en dominios limitados, llamados micro mundos (1963-1969).

El micro mundo más famoso es **mundo de bloques**:

El objetivo es construir una o más pilas verticales de bloques. Sólo se puede mover un bloque a la vez: puede colocarse sobre la mesa o encima de otro bloque. Debido a esto, cualquier bloque que esté, en un momento dado, debajo de otro bloque no se puede mover. Además, algunos tipos de bloques no pueden tener otros bloques apilados encima.



HISTORIA DE LA IA

Primer invierno (1966-1973)

Herbert Simon (1957) predijo que en 10 años con IA se iba a lograr batir al campeón mundial de ajedrez, y resolver teoremas matemáticos complejos... cosas que demoraron 40 años.

El problema de esta sobre-expectativas son por dos motivos:

- Algoritmos de ese momento se basaban en **introspección informada** en como los humanos realizan una tarea.
- No se había desarrollado las teorías computacionales de complejidad algorítmica, por lo que no se sabía cómo escalaban los algoritmos.





HISTORIA DE LA IA

Primer invierno (1966-1973)

El libro de Minsky y Papert, llamado **Perceptrones** (1969) se probó que los perceptrones (modelo de neurona), podían representar muy pocas funciones, principalmente la función lógica XOR.

Esto mató todo desarrollo de Deep Learning y neurociencia computacional durante 15 años.

Spoiler: En Aprendizaje de Maquina I veremos que esta afirmación, aunque no incorrecta, no era válida.

HISTORIA DE LA IA

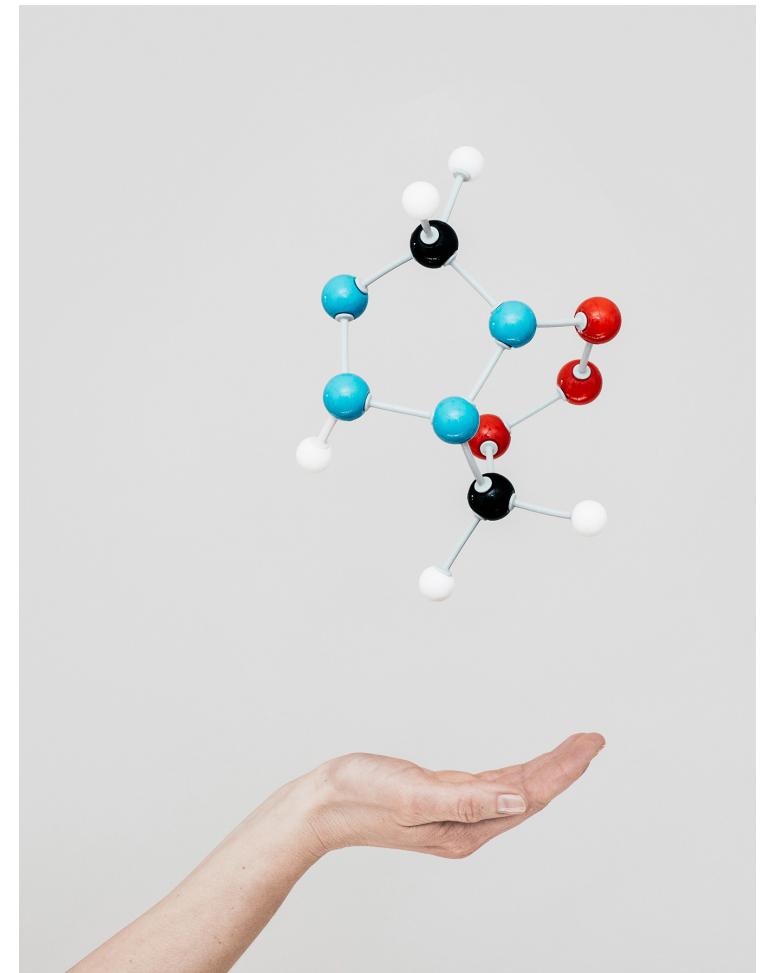
Sistemas expertos (1969-1986)

Ante la falla los algoritmos desarrollados previamente, principalmente por su imposibilidad de escalar,

Llegaron los sistemas expertos, la primera aplicación exitosa de IA.

El primer sistema experto fue el programa DENDRAL (1969) el cual infería estructuras moleculares, pero para poder resolver le introdujeron reglas de expertos químicos para evitar búsquedas innecesarias.

Luego otro invierno de IA, pero esta vez comercial.



HISTORIA DE LA IA

El retorno de las redes neuronales (1986-presente)

En los 80, se re-descubrió el **algoritmo de aprendizaje back-propagation**, tarea fundamental para que las redes se entrenaran.

Geoff Hinton, uno de los principales impulsores de la vuelta de redes neuronales, describió a los símbolos como el “éter de la IA”. Siendo la primera vez que se puso en discusión la idea de que la inteligencia se trataba de símbolos representativos que los primeros modelos de IA tenían en cuenta.

HISTORIA DE LA IA

Aprendizaje Automático (1987-presente)

Durante los finales de los ‘80, IA tomó un enfoque más científico que previamente, se alejó de conceptos filosóficos y lógica booleana, a usar probabilidad, y experimentos validables.

Esto llevo a que IA vuelva a tomar elementos de otras áreas de ciencias que se había alejado (partiendo de un mismo origen)... tales como estadística, teoría de control, teoría de la información, optimizaciones, entre otros.

Esto llevo a la prevalencia de aprendizaje automático y al desarrollo de los modelos que hoy conocemos. Tales como **cadenas de Markov**, **redes bayesianas**, **máquinas de vectores de soporte**, etc.

Estos nuevos modelos fueron más importantes que los de redes neuronales, dado que se llegaban a mejores resultados, con mucho menos procesamiento.

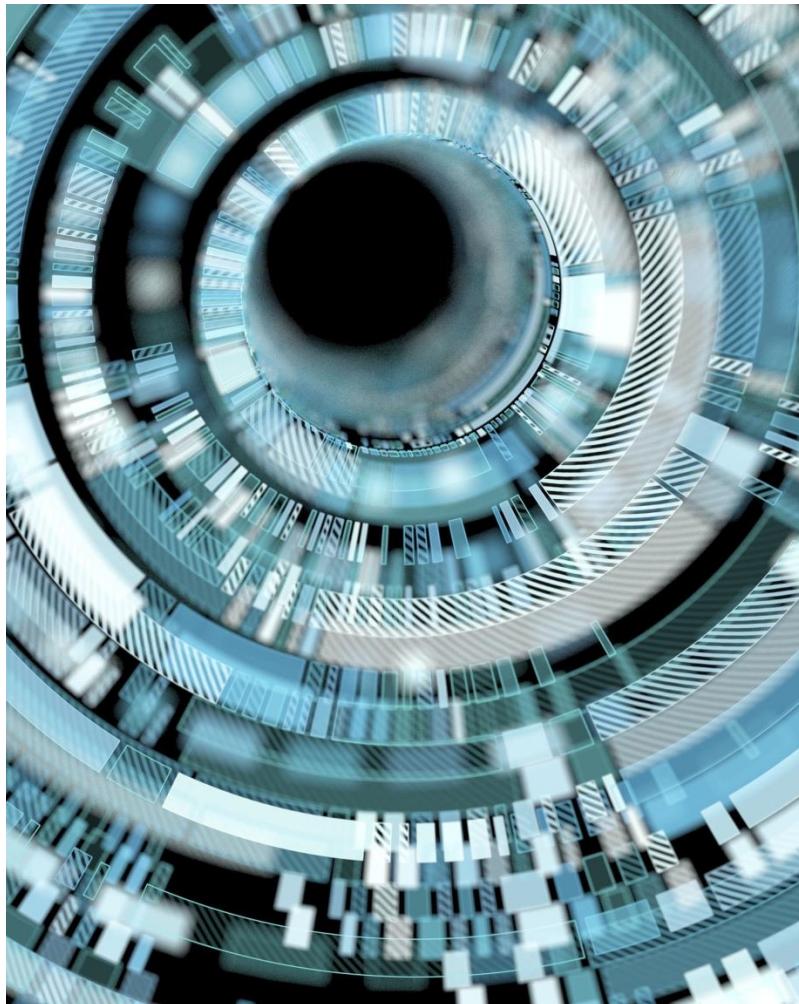
HISTORIA DE LA IA

Big data (2001-presente)

Con la llegada de la World Wide Web y mejoras en las computadoras (gracias a la ley de Moore), empezamos a tener datasets enormes, un fenómeno llamado big data.

Esto llevó a la necesidad de desarrollar algoritmos que tomen ventaja de este nuevo volumen de datos.

La disponibilidad de **Big Data**, ayudó a aprendizaje automático y a IA, recuperar atractivo comercial. Con Big data se logró en 2011 que el sistema IBM Watson llegar a un nivel de campeón humano de Jeopardy!



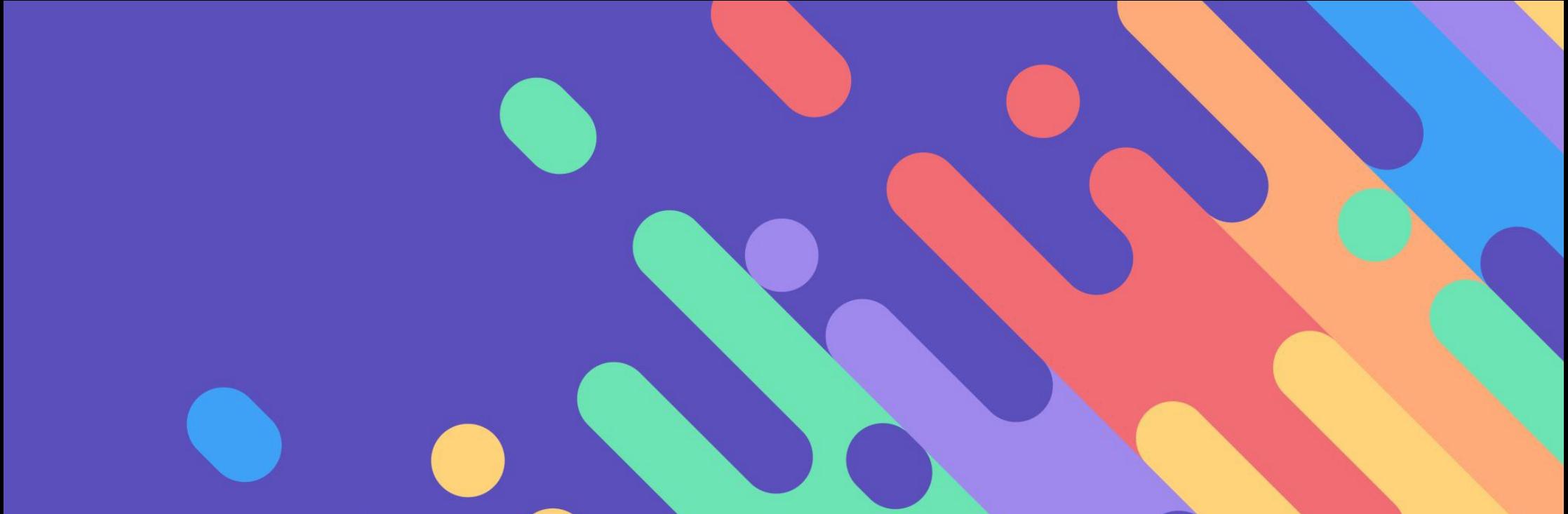
HISTORIA DE LA IA

Deep Learning (2001-presente)

Finalmente, con la llegada de Big Data, las redes neuronales, ahora con suficiente procesamiento como para lograr grandes redes y profundas, se logró explotar su potencial.

Se lograron enormes avances en casi cualquier área que se propusieran implementar estos algoritmos... finalmente con sistemas que son muy parecidos a estructuras nerviosas.

Deep Learning solo se pudo desarrollar en los últimos tiempos, cuando se lograron CPU que pueden realizar 10^{10} operaciones por segundo, o gracias al desarrollo de hardware específico (GPU, TPU o FPGA) para el procesamiento paralelo de tensores (10^{17} operaciones/segundo). Y además gracias a Big Data con Petabytes de datos para entrenar.



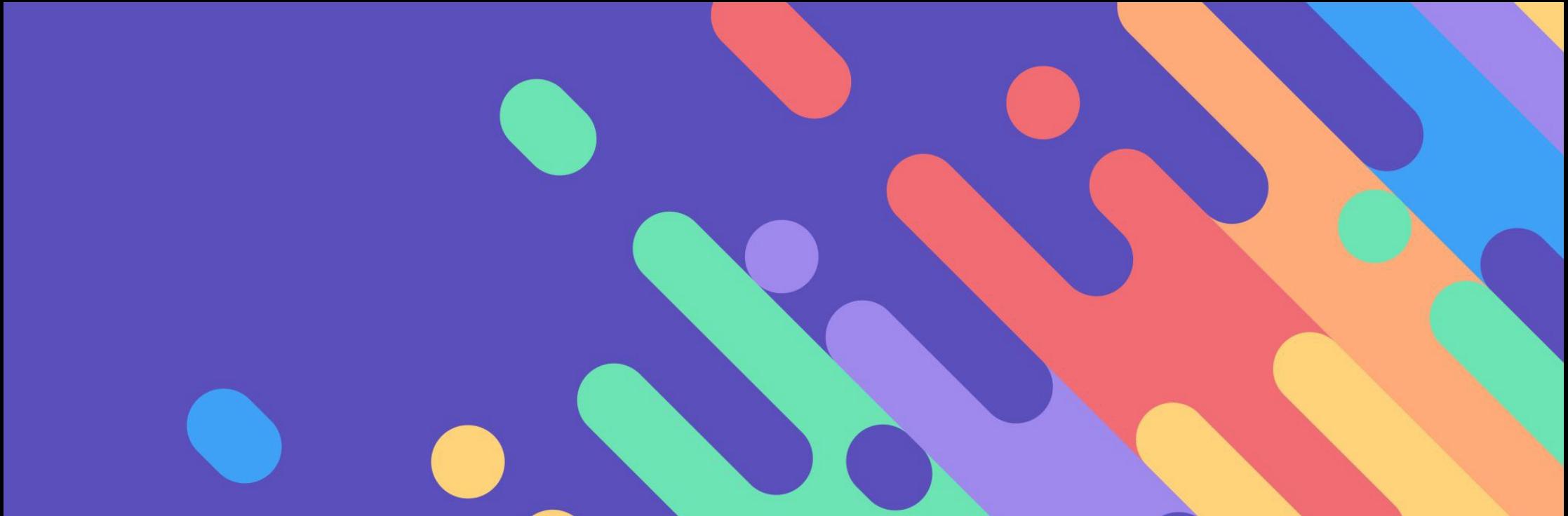
BENEFICIOS Y RIESGOS DE LA IA

BENEFICIOS DE LA IA

- La entera civilización es el producto de inteligencia humana. Las maquinas inteligentes nos pueden elevar este techo.
- Robots e IA pueden eliminar a la humanidad de tareas nimias.
- Puede acelerar investigaciones científicas.
- Y más, ¿ cuales se les ocurren?

RIESGOS DE LA IA

- Armas letales autónomas.
- Vigilancia y persuasión (a lo 1984).
- Toma de decisiones sesgadas.
- Impacto en empleos.
- Implementación en aplicaciones críticas en seguridad.
- Ciberseguridad.

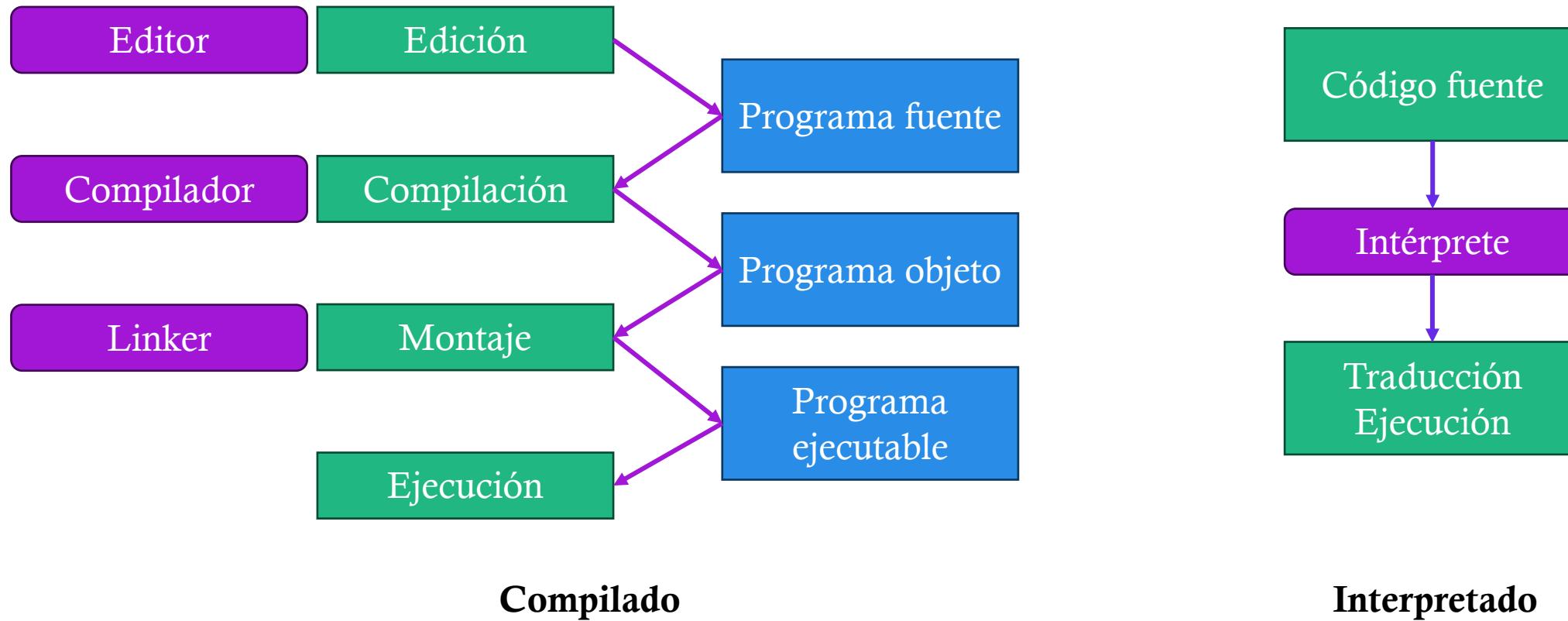


PYTHON

PYTHON

- Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.
- Python es un lenguaje de programación **multiparadigma**. Permite varios estilos: **programación orientada a objetos**, **programación imperativa** y **programación funcional**.
- OBS: En el fondo, Python es un lenguaje orientado a objetos, todo, absolutamente todo es un objeto.
- Python usa tipado dinámico y conteo de referencias para la gestión de memoria.
- Python reemplazó en gran medida a LISP en IA, principalmente por ser multiparadigma.

PYTHON



PYTHON

Python es uno de los lenguajes más usados en Ciencia de datos. ¿Por qué?

- Porque tiene una sintaxis simple y es fácil de adaptar para quienes no vienen de ambientes de ingeniería o ciencia de la informática.

PYTHON

Python es famoso por ser lento comparado con lenguajes como C++, por qué se usa en Machine Learning o IA?

- La respuesta es que no se usa librerías hechas Python. Ninguna de las bibliotecas que se utilizan está realmente escrita en Python.
- Casi siempre están escritos en Fortran o C++ y simplemente interactúan con Python a través de algún wrapper.
- La velocidad de Python es irrelevante si solo se interactúa con las librerías escritas en un C++ altamente optimizado.

Fuente: <https://qr.ae/pKrGdr>

PYTHON

Modo interactivo

- Python posee un **modo interactivo**: Se escriben las instrucciones en una especie de intérprete de comandos. Las expresiones pueden ser introducidas una a una.

```
Python 3.10.4 (main, ) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> var = "Hola, mundo"
>>> print(var)
Hola, mundo
>>>
```

PYTHON

Modo interactivo

- **iPython** (Parte de SciPy): Extiende la capacidad del modo interactivo y provee un kernel para Jupyter

```
:~$ ipython3
Python 3.10.4 (main, ) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: var = "Hola, mundo"

In [2]: print(var)
Hola, mundo

In [3]: 
```

PYTHON

Jupyter Notebook

- Es un entorno computacional interactivo basado en la web para crear documentos de notebook.
- Jupyter Notebook es similar a la interfaz de notebook de otros programas como Maple, Mathematica y SageMath, un estilo de interfaz computacional que se originó con **Mathematica** en la década de 1980.



PYTHON

Jupyter Notebook

jupyter Read Ingredients and products example Last Checkpoint: 09/08/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [1]:

```
import pandas as pd
import numpy as np

pd.set_option("max_colwidth", None)
pd.set_option("max_seq_items", None)
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', None)
```

In [2]:

```
def explode_product_df(df: pd.DataFrame, col: str, add_internal_id:bool = False, add_sku_parent: bool = True,
                      multiply:bool = False, quantity_col: str = "quantity", order_column: bool = False) -> tuple:

    if not col in df.columns:
        return -1, pd.DataFrame()

    df_temp = df.copy()

    if add_internal_id:
        df_temp["parent_internal_id"] = 0
        df_temp["internal_id"] = np.arange(1, df_temp.shape[0] + 1)
        counter_id = df_temp.shape[0] + 1

    first_level = df_temp.drop([col], axis=1)
    first_level[col] = np.nan

    df_list = [first_level]

    def add_data_from_parent_to_list(lists, col, data):
        for dic in lists:
            dic[col] = data
        return lists
```



HERRAMIENTAS DE DESARROLLO

Vamos a ver tres formas de instalar nuestro entorno de desarrollo:

- **Modo bebe**



- **Modo novato**



- **Modo gore**



HERRAMIENTAS DE DESARROLLO



Modo bebé

Google Colab permite escribir y ejecutar Python en el navegador:

- Sin configurar
- Fácil de compartir
- Acceso a GPUs sin cargo



Es una Jupyter Notebook que corre en una máquina virtual de Google Cloud:

- Es gratuito
- Ofrece 12 GB de RAM y 100 GB de disco.

Las notebooks quedan en Google Drive, fácil de compartir.

El problema es que no todo lo de esta materia y futuros cursos funcionan en Google Colab

HERRAMIENTAS DE DESARROLLO



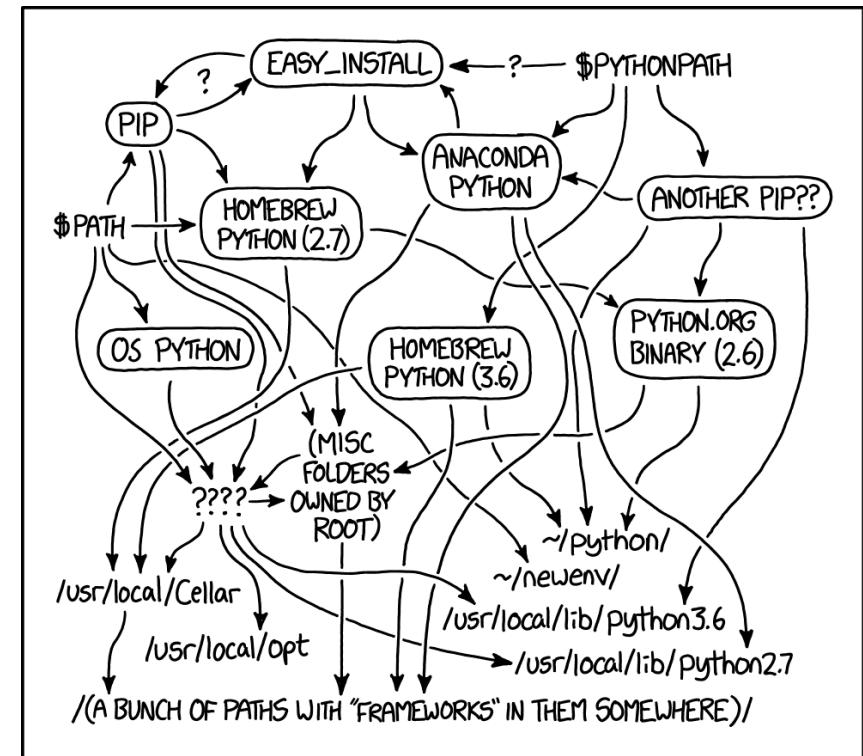
Modo novato

Para poder utilizar Python debemos preparar nuestra máquina con las herramientas necesarias.

Para arrancar a usar Python necesitamos instalar:

- Tener instalado al menos una versión Python
- **Gestión de paquetes.** Tener una herramienta que nos permita gestionar las librerías que queremos instalar.
- **Entornos virtuales.** Cuando trabajamos en distintos proyectos, no todos ellos requieren los mismos paquetes ni siquiera la misma versión de Python. Para ello podemos usar entornos virtuales que nos permiten instalar paquetes específicos al proyecto, también establecer qué versión de Python usaremos.

No es recomendable usar el Python que usa el sistema, sobre todo en Linux, ya que, si modificamos algo de esta versión, podemos romper parte de nuestro OS.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Fuente: <https://xkcd.com/1987/>

HERRAMIENTAS DE DESARROLLO



Modo novato

La forma más fácil de obtener todo esto es usar una herramienta que nos resuelva todo este problema y además que este pensado principalmente para ciencia de datos.

Esta herramienta es [Anaconda](#)

Anaconda es una distribución de Python y R, enfocado en computación científica (ciencia de datos, procesamiento en gran escala y aprendizaje automático).

Está orientado a simplificar el despliegue y administración de los paquetes de software.

La versión de los paquetes en Anaconda es llevada a cabo por el sistema de manejo de paquete conda.

Anaconda está disponible en Linux, MacOS y Windows.



ANACONDA®

HERRAMIENTAS DE DESARROLLO



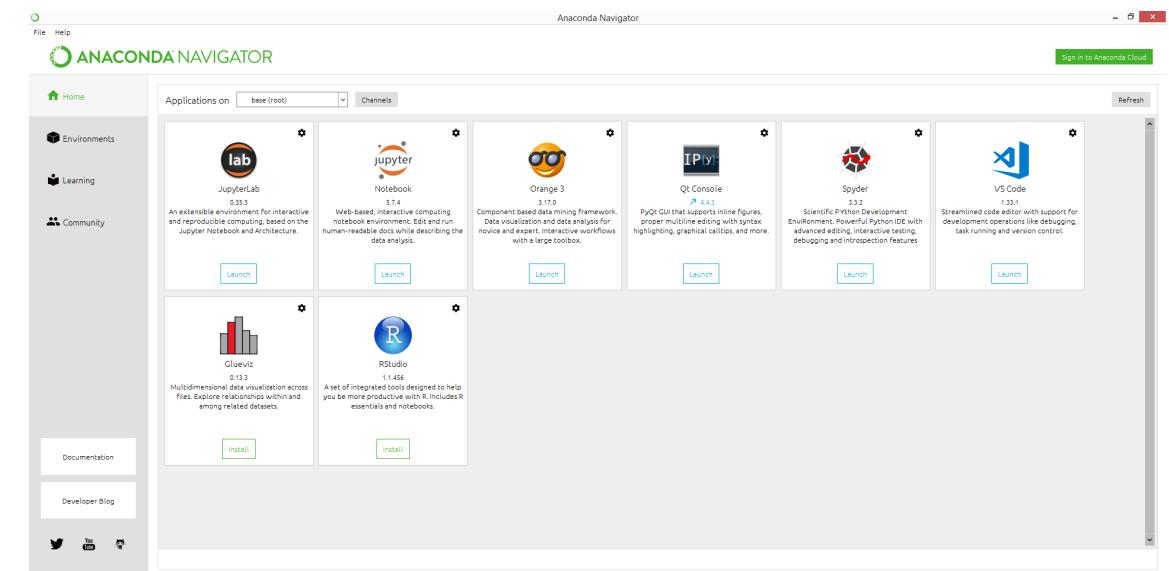
Modo novato

Anaconda también maneja entornos virtuales usando **conda**.

Además, nos provee un UI que permite configurar sin necesidad de pasar por consola.

Gran desventaja, pesa 5 Gb. Se puede evitar si se usa Miniconda (400 Mb), pero perdemos la UI.

Te sentís un usuario más avanzado, lee la [documentación de Anaconda](#) para entender los diferentes sabores que se presentan y como usar el CLI (conda).



HERRAMIENTAS DE DESARROLLO



Modo gore

A partir de ahí, instalar un entorno de desarrollo de Python es similar, tenés que:

- Tener instalado al menos una versión Python
- **Gestión de paquetes**
- **Entornos virtuales**

Nada más que ahora veamos herramientas más poderosas y generales.

HERRAMIENTAS DE DESARROLLO



Modo gore

Instalar Python, dependiendo de que OS tengas vas a tener muchas opciones:

- Instalar [Python](#) desde la página. Se pueden descargar más de una versión, hoy en día se recomienda de ≥ 3.9 . Nosotros vamos a usar la 3.10.
- Instalar con tu gestor de paquete (Linux o Windows con [WSL](#)) o [Homebrew](#) (MacOS). Por ejemplo, Ubuntu tiene [este repositorio](#) o Archlinux usando [AUR](#).
- Usando un gestor de versiones de Python:
 - [pyenv](#) para Linux/MacOS
 - [pyenv-win](#) para Windows

HERRAMIENTAS DE DESARROLLO



Modo gore

Gestión de paquetes:

pip es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Muchos de estos paquetes pueden ser encontrados en [PyPI](#). Python incluyen pip por defecto.

Existen otros gestores:

- [uv](#)
- [Mamba](#)
- [PDM](#)
- Etc.

HERRAMIENTAS DE DESARROLLO



Modo gore

Entornos virtuales:

Hay múltiples formas de hacerlo, una de ellas es [virtualenv](#).

Otros que existen son:

- [virtualenvwrapper](#)
- [pyenv-virtualenv](#)

HERRAMIENTAS DE DESARROLLO



Modo gore

Hoy en día una herramienta que es muy popular en el desarrollo de Python es [Poetry](#).



Poetry es una herramienta para gestión y empaquetado de dependencias en Python. Permite declarar las bibliotecas de las que depende el proyecto y se encarga de administrarla.

HERRAMIENTAS DE DESARROLLO



Modo gore

Para arrancar un nuevo proyecto, hacemos:

```
poetry new test-project
```

También podemos elegir que versión de Python usar, pero no instala automáticamente el intérprete de Python, eso hay que encargarse uno. Lo bueno es que, al indicarle una versión, se encarga de que los paquetes que se instalen sean compatibles con las versiones especificadas.

Eso se hace modificando el archivo `pyproject.toml`:

```
[tool.poetry.dependencies]
python = "^3.9.0"
```

HERRAMIENTAS DE DESARROLLO



Modo gore

Dado que Poetry no maneja versiones y entornos virtuales, ese es tu problema.

Primero podemos cambiar la versión de Python usando:

```
poetry env use python3.10
```

Al hacer esto se encarga de crear un entorno virtual para este proyecto para esa versión de Python.

Una vez hecho esto, podemos instalar paquetes haciendo:

```
poetry add matplotlib
```

Y con esto solo estamos viendo la superficie de [Poetry](#), es una herramienta muy poderosa. Dado que sos un usuario hardcore no vas a tener problema de leer la documentación.

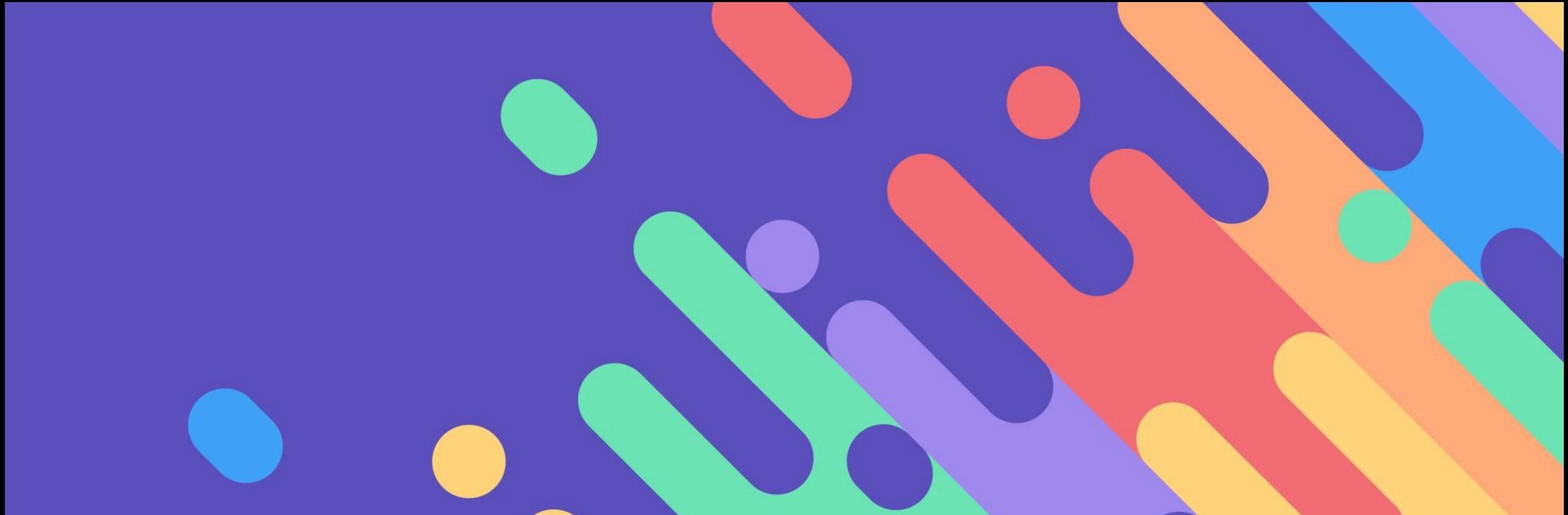
En el repositorio de esta materia se provee un `pyproject.toml` para instalar todas las dependencias que iremos a usar.

PYTHON

Jupyter Notebook



- Jupyter posee diferentes kernels, los cuales son procesos específicos de algún lenguaje de programación que ejecuta el código contenido en las celdas de un notebook de Jupyter.
- Para cada ambiente virtual, podemos crear un kernel, y cuando elijamos ese kernel, estaremos usando ese entorno virtual en Jupyter.
- Para ello, nuestro ambiente debe tener instalado la librería ipykernel. En Anaconda:
`conda install -c conda-forge ipykernel`
- Luego creamos el kernel haciendo
`python -m ipykernel install --user --name=myenv`
- Listo, si ejecutamos Jupyter va a aparecer entre los kernel disponibles con el nombre `myenv`.



LIBRERÍAS ASOCIADAS

NUMPY

NumPy (NUMerical PYthon) es el paquete fundamental para la computación científica con Python. Contiene entre otras cosas:

- Estructura de datos principal: el arreglo N-dimensional
- Se puede integrar código en C/C++ y Fortran.
- Capacidades muy eficientes de álgebra lineal, transformación de Fourier y números aleatorios.

NUMPY

El principal beneficio de NumPy es que permite una generación y manejo de datos extremadamente rápido.

NumPy tiene su propia estructura de datos incorporada llamado arreglo que es similar a la lista normal de Python, pero puede almacenar y operar con datos de manera mucho más eficiente.

Una forma que logra ser eficiente es que los arrays sus elementos tienen que ser del mismo tipo.

Los arrays son **mutables** por defectos, pero se puede cambiar este comportamiento.

SCIPY

La biblioteca SciPy ("Scientific Python") está construida sobre NumPy y ofrece funciones científicas y estadísticas por encima de las funciones puramente matemáticas, por ejemplo:

- Rutinas de álgebra lineal avanzada
- Optimización de funciones matemáticas
- Procesamiento de señales
- Distribuciones matemáticas

STATSMODELS

Una librería estadística que está en auge hoy en día, reemplazando a Scipy en lo que refiere a estadística es Statsmodels.

Es un módulo de Python que proporciona clases y funciones para la estimación de muchos modelos estadísticos diferentes, así como para realizar pruebas estadísticas y exploración de datos estadísticos.

PANDAS

El nombre de Pandas se deriva del término “Panel Data” y es la librería de análisis de datos de Python que:

- Define nuevas estructuras de datos basadas en los arrays de Numpy.
- Permite leer y escribir fácilmente ficheros en formato.
- Permite acceder a los datos mediante índices o nombres para filas y columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite trabajar con series temporales.
- Realiza estas operaciones de manera eficiente.

PANDAS

Serie

Son estructuras 1D similares a los arrays de una dimensión. Son homogéneas, es decir, sus elementos tienen que ser del mismo tipo, y su tamaño es inmutable, es decir, no se puede cambiar, aunque sí su contenido.

- Una Serie es similar a la columna de una tabla y es equivalente a una columna de un DataFrame.
- Dispone de un índice que asocia un nombre a cada elemento de la serie y a través de la cual se accede al elemento.

PANDAS

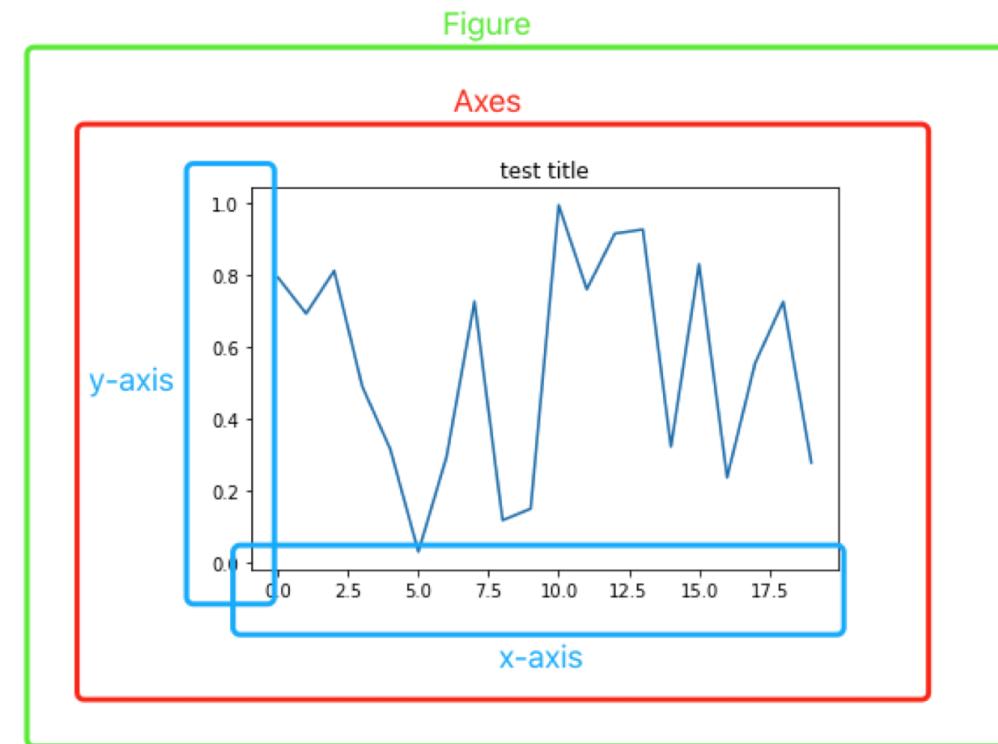
DataFrame

Un objeto del tipo DataFrame es una colección bidimensional (2D) ordenada en columnas con nombres y tipos, parecido a una tabla de Excel, en donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos.

Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas.

MATPLOTLIB

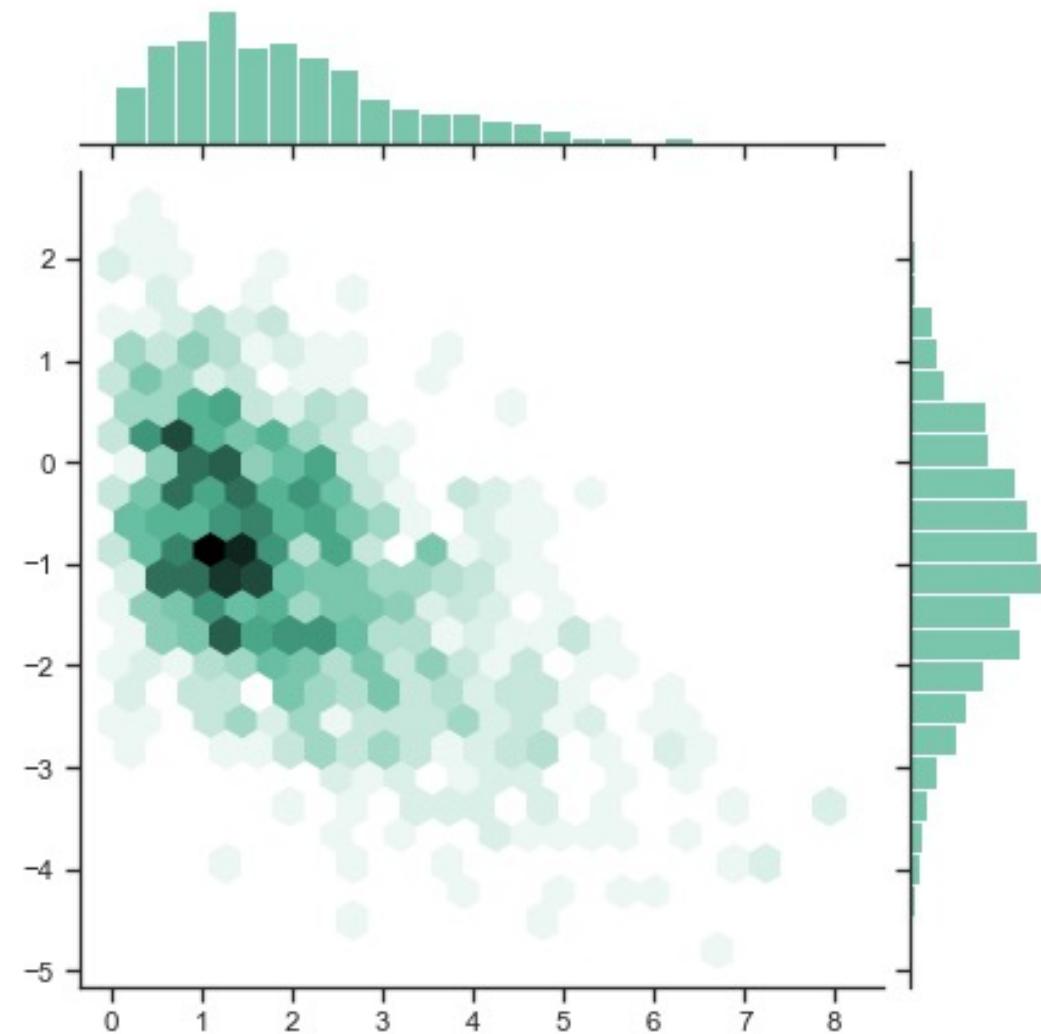
Matplotlib es una biblioteca de graficado para Python y NumPy. Proporciona una API orientada a objetos para incrustar gráficos en aplicaciones que utilizan kits de herramientas GUI de uso general como Tkinter, wxPython, Qt o GTK.



SEABORN

Seaborn es una biblioteca de visualización de datos de Python basada en matplotlib.

Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.



SCIKIT-LEARN

Es una biblioteca muy útil cuando se trabaja en algoritmos de Machine Learning en Python. Proporciona herramientas de procesamiento de datos y una gama de algoritmos de aprendizaje en Python.

Es la puerta de entrada al aprendizaje automático en Python.

PYTORCH

PyTorch es una librería de Deep Learning. Es mantenida por Meta y es la principal competencia de otra famosa librería (tensorflow de Google).

Tiene un API para Python, como también para C++. PyTorch nos da toda una infraestructura de:

- Computación de tensores. Es similar a los arrays de Numpy pero con posibilidad de usarlos en GPU.
- Redes neuronales profundas.