

# INTRODUCCIÓN



Aprendizaje  
Automático  
CEIoT - 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

## Repositorio de la materia:

- [https://github.com/facundolucianna/Apre\\_Aut\\_CEIoT](https://github.com/facundolucianna/Apre_Aut_CEIoT)

## Consultas

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

## Correo

- Facundo Adrián Lucianna: [facundolucianna@gmail.com](mailto:facundolucianna@gmail.com)

---

# EVALUACIÓN

---

3 Trabajos Prácticos con entrega a 14 días posteriores de cuando fue presentado. Los trabajos pueden ser presentados individualmente o grupalmente (máxima 4 personas por grupo)

---

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 formulario de Google o correo electrónico, ya sea el envío del contenido o el link a repositorio (de GitHub o GitLab) con el trabajo.

Criterio de evaluación:

[https://github.com/facundolucianna/Apre\\_Aut\\_CEIoT/blob/main/CriteriosAprobacion.md](https://github.com/facundolucianna/Apre_Aut_CEIoT/blob/main/CriteriosAprobacion.md)

---

```

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 store information
df_operation_table['log_currency'] = aux.get_currency(currency)

# Rename columns to make easier to use
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
Aprendizaje Automático – CEIoT – FIUBA
return 200, None, df_operation_table

```



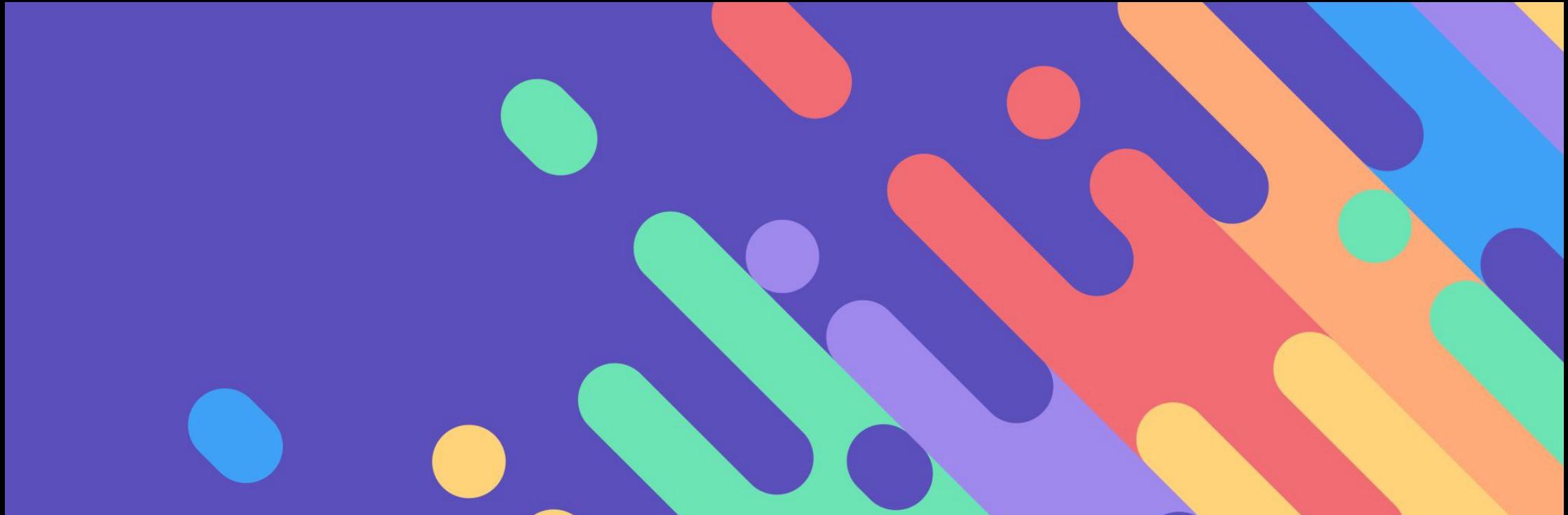
# HERRAMIENTAS

- **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

---

# BIBLIOGRAFÍA

- Python Data Science Handbook - Jake VanderPlas  
<https://jakevdp.github.io/PythonDataScienceHandbook/>
- Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python - Peter Bruce (Ed. O'Reilly)
- The Elements of Statistical Learning - Trevor Hastie (Ed. Springer)
- An Introduction to Statistical Learning - Gareth James (Ed. Springer)
- Pattern Recognition And Machine Learning - Christopher Bishop (Ed. Springer)
- Deep Learning - Ian Goodfellow <https://www.deeplearningbook.org/>



---

# APRENDIZAJE AUTOMÁTICO

---

# APRENDIZAJE AUTOMÁTICO

Aprendizaje automático se entiende a:

Una computadora observa algunos datos, construye un modelo basado en estos datos, y usa el modelo como una hipótesis sobre el mundo y como una pieza de software que puede resolver problemas.

*¿Pero porque queremos que una computadora aprenda? ¿Por qué no programar el modelo directamente?*

- *No se puede anticipar todas las posibles situaciones futuras.*
- *No se tiene idea de cómo programar una solución por uno mismo.*



---

# APRENDIZAJE AUTOMÁTICO – EJEMPLOS

## Detección de SPAM

El detector de SPAM es una herramienta que hoy en día damos por sentado que todo software de correo electrónico debe tener. Este problema se resuelve usando Aprendizaje Automático.

Una forma de resolverlo es analizando las palabras y signos de puntuación de los correos. Este es un problema que llamamos de *aprendizaje supervisado*, cuyo resultado es un indicador si el corre es SPAM o no. Esto se llama *problema de clasificación*.

	george	you	your	hp	free	hpl	!	our	re
<i>Spam</i>	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13
<i>Email</i>	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42

---

# APRENDIZAJE AUTOMÁTICO – EJEMPLOS

## Detección de SPAM

El detector de SPAM es una herramienta que hoy en día damos por sentado que todo software de correo electrónico debe tener. Este problema se resuelve usando Aprendizaje Automático.

En este problema no todo error al predecir es lo mismo. *Queremos evitar filtrar el buen correo electrónico*, mientras que dejar pasar el spam no es deseable pero sus consecuencias son menos graves.

	george	you	your	hp	free	hpl	!	our	re
<i>Spam</i>	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13
<i>Email</i>	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42

# APRENDIZAJE AUTOMÁTICO – EJEMPLOS

## Cáncer de próstata

Tenemos mediciones del antígeno prostático específico (PSA) y una serie de medidas clínicas, en 97 hombres que estaban a punto de recibir una prostatectomía radical.

Como objetivo se busca predecir el log de PSA usando un numero de mediciones a partir de una serie de mediciones que incluyen el registro del volumen del cáncer, el registro del peso de la próstata, la edad, el registro de la cantidad de hiperplasia prostática benigna, la invasión de la vesícula seminal, entre otros.

Este es un problema de *aprendizaje supervisado*, conocido como *problema de regresión*, porque la medición del resultado es *cuantitativa*.

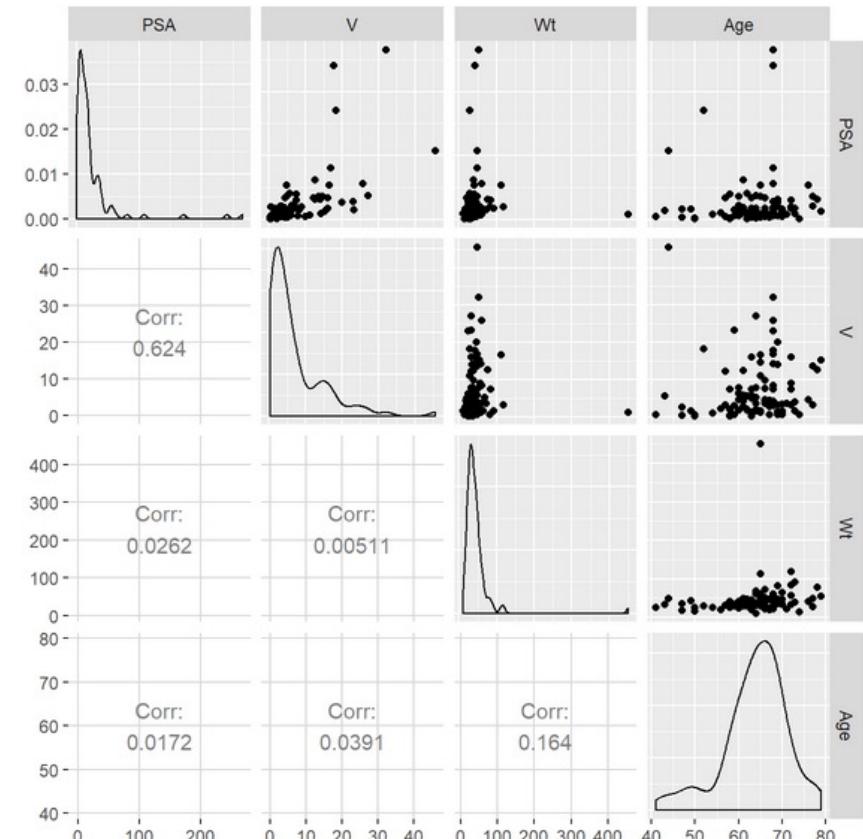
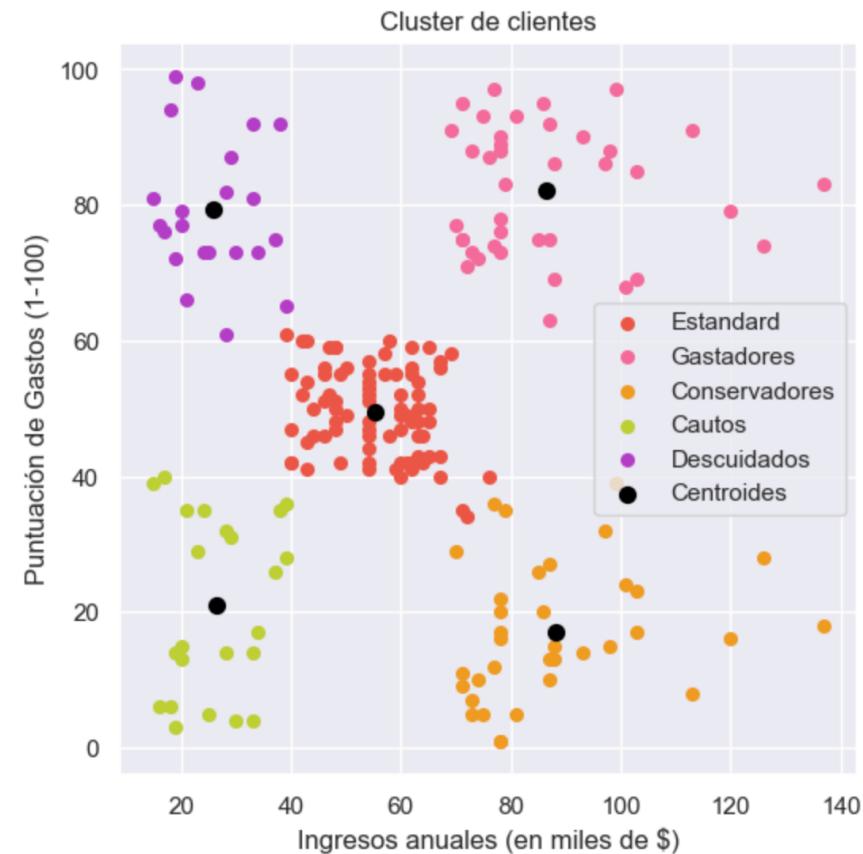


Imagen obtenida de [ADA2: Homework 06, Ch 03 A Taste of Model Selection for Multiple Regression](#)

# APRENDIZAJE AUTOMÁTICO – EJEMPLOS

## Descubrimiento de segmentos de clientes.

Una empresa de retail quiere entender cómo se segmentan sus clientes en información demográfica y gustos dados su interior de compra. Usando estos datos se usa un modelo que agrupa los datos en diferentes grupos por *similitud*. Una vez establecidos estos grupos, la empresa puede establecer estrategias de marketing diferentes para cada uno de estos grupos. Este tipo de problema es de tipo *no supervisado* y particularmente de *clustering*.



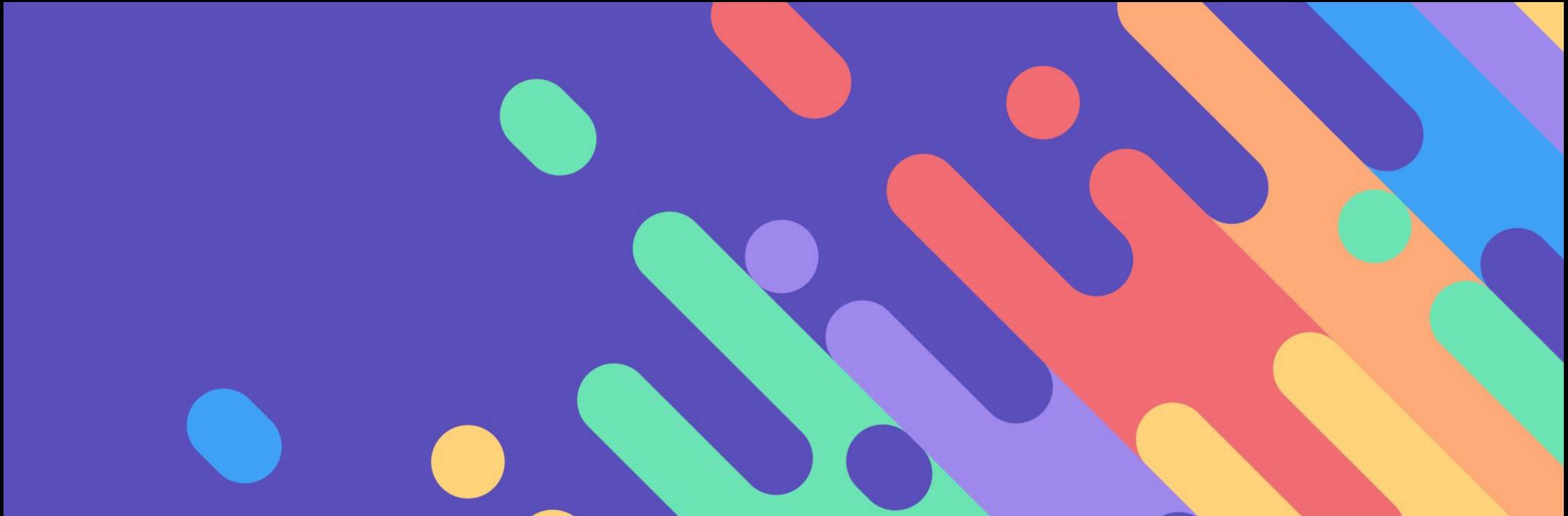
---

# APRENDIZAJE AUTOMÁTICO – EJEMPLOS

## Sistema de recomendación de videos

Una empresa dedicada a compartir videos necesita mantener a los usuarios comprometidos en la plataforma. Necesita que cuando un nuevo video termina, recomendarle al usuario el siguiente video a ver, de tal forma que la persona siga deseando mantenerse en la plataforma. Por lo que es vital construir un sistema de recomendación que se utilice la información del usuario en la plataforma.

Este tipo de algoritmos se llaman de *recomendación* y es de tipo *no supervisado*.



---

# DATOS

---

# DATOS

Lo más importante en Aprendizaje Automático (y en Data Science en general) son los

## Datos

Nos permite describir un objeto al que podemos llamar *entidad*.

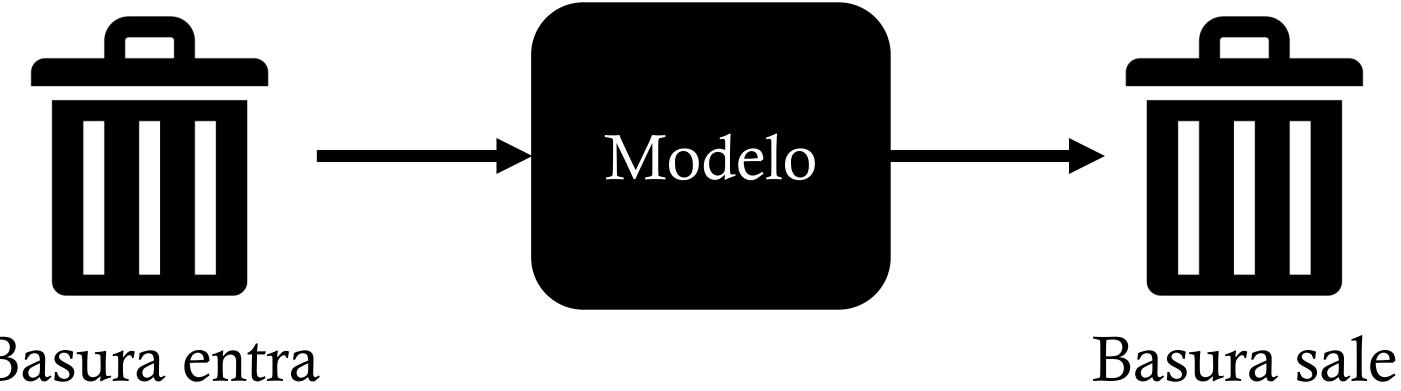
Esta entidad y su información puede ser diferente a pesar de que describa un mismo objeto. La forma en que se elija representar los datos no solo afecta la forma en que se construyen sus sistemas, sino también los problemas que sus sistemas pueden resolver.

Por ejemplo, queremos representar un auto:

- En un problema para compra y venta de autos, podemos representarlo con el fabricante, modelo, año, color, y su precio.
- En un problema de un sistema de seguimiento policial, podemos representarlo por quien es el dueño, patente y su historia de direcciones registradas.

# DATOS

---



---

# DATOS

Los datos se pueden encontrar en dos formas:

- **Datos estructurados:** Tienen un formato estandarizado que permite tanto al software como a las personas acceder a estos de forma eficaz. Por lo general, se trata de datos tabulares con filas y columnas que definen claramente sus atributos. Las computadoras pueden procesar eficazmente los datos estructurados en busca de información dado que se trata de información cuantitativa.
- **Datos no estructurados:** Son información sin un *modelo de datos* establecido o son datos que no están ordenados de una manera predefinida. Por ejemplo, archivos de texto, video, informes, e-mails, imágenes.

---

# DATOS

## Datos estructurados

En Aprendizaje Automatico en general se usan estructuras dos dimensiones y notaciones vectoriales para referirnos a los datos:

- Cada fila del array es una **muestra**, **observación** o dato puntual.
- Cada columna es una **característica** (**feature** o **atributo**), de la observación.
- En el caso más general hay una columna, que llamaremos **objetivo**, **label**, **etiqueta** o **respuesta**, y que será el valor que se pretende predecir.

# DATOS

	Atributos/features					Objetivo
	Position	Experience	Skill	Country	City	Salary (\$)
<b>Observación →</b>	Developer	0	1	Argentina	Buenos Aires	103100
	Data Scientist	2	2	Uruguay	Montevideo	104900
	Developer	3	1	Argentina	Chivilcoy	106800
	QA Eng	2	2	Colombia	Bogotá	108700
	Product Manager	1	5	Perú	Lima	110400
	Developer	7	5	Paraguay	Asunción	112300
	Cloud Eng	5	2	Argentina	Buenos Aires	116100

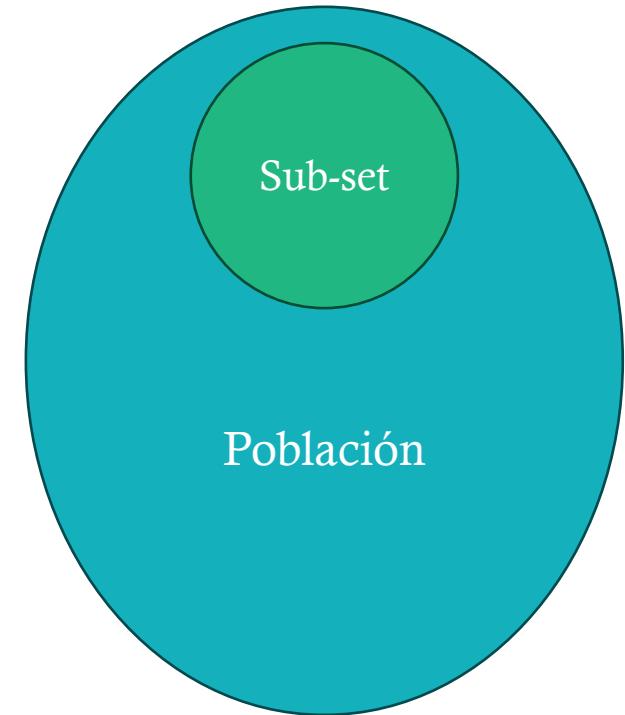
# DATOS

---

Cuando modelamos usando Aprendizaje Automático tenemos un variable dependiente **target  $\mathbf{y}$**  (que podemos conocer o no) dado un conjunto de predictores  **$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$** , el cual podemos describir como un vector  **$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p)$** .

*El desafío en Aprendizaje Automático es elegir las entradas correctas.*

El conjunto total de estas observaciones se llama **población**, lo cual casi nunca podemos tener, sino que tenemos un sub-set de estas observaciones.



---

# DATOS

Cada atributo puede tener diferentes formas, una medida de altura es diferente a el tipo de música que más le gusta a un usuario.

Tenemos diferentes tipos de variables:

**Variables numéricas:** Son aquellas que representan números y con ellas se pueden realizar operaciones aritméticas.

- **Discretas:** Son números enteros, cosas que se pueden contar. 1, 2, 3 empleados, 568 personas.
- **Continuas:** Números reales. El valor dado a una observación para una variable continua puede incluir valores tan pequeños como lo permita el instrumento de medición o la representación numérica. Altura, peso, costo, precio...

---

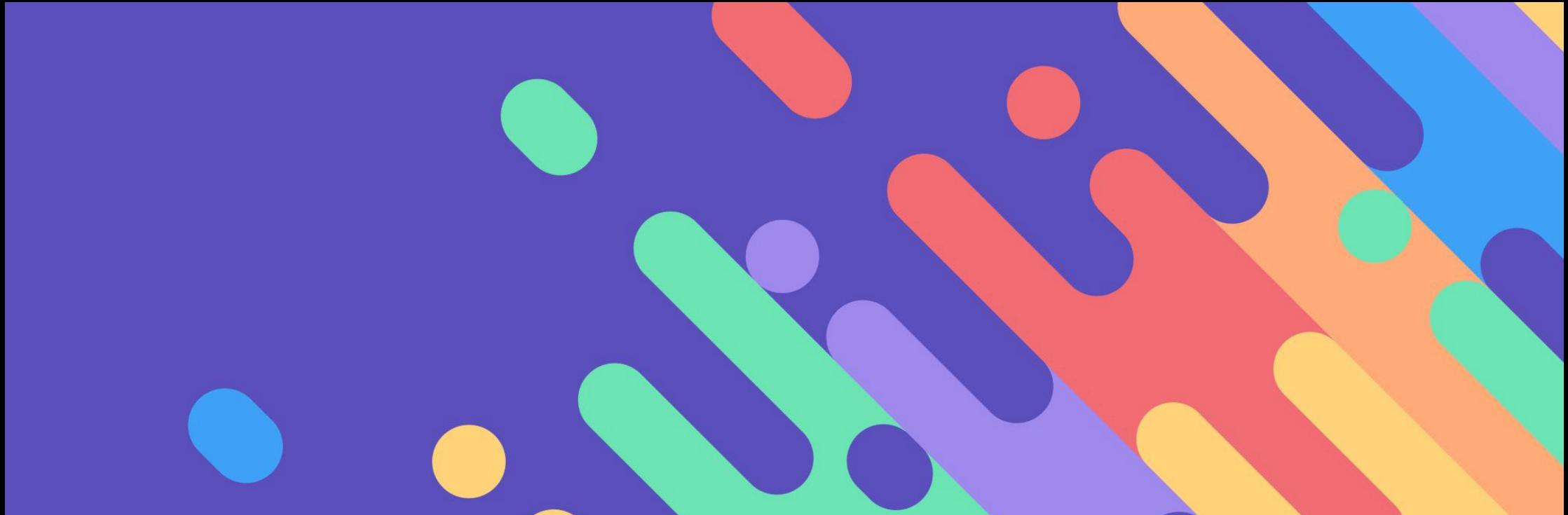
# DATOS

**Variables categóricas:** Es una variable que puede tomar uno de un número limitado, y por lo general fijo, de posibles valores.

- **Nominal:** Valores que toman corresponden a nombres de categorías, clases o estados de las cosas. Estado civil (soltero, casado, divorciado), Spam en e-mails (Binario, es spam o no). Tipo de cerveza (Ale, Pale, Stout, etc.)
- **Ordinal:** Similar al nominal, con la diferencia de poder aplicar un orden sobre estas categorías.

Estado de satisfacción: Me disgusta mucho, me disgusta, neutro, me gusta, me gusta mucho.

No tiene que haber una equidistancia entre las opciones.

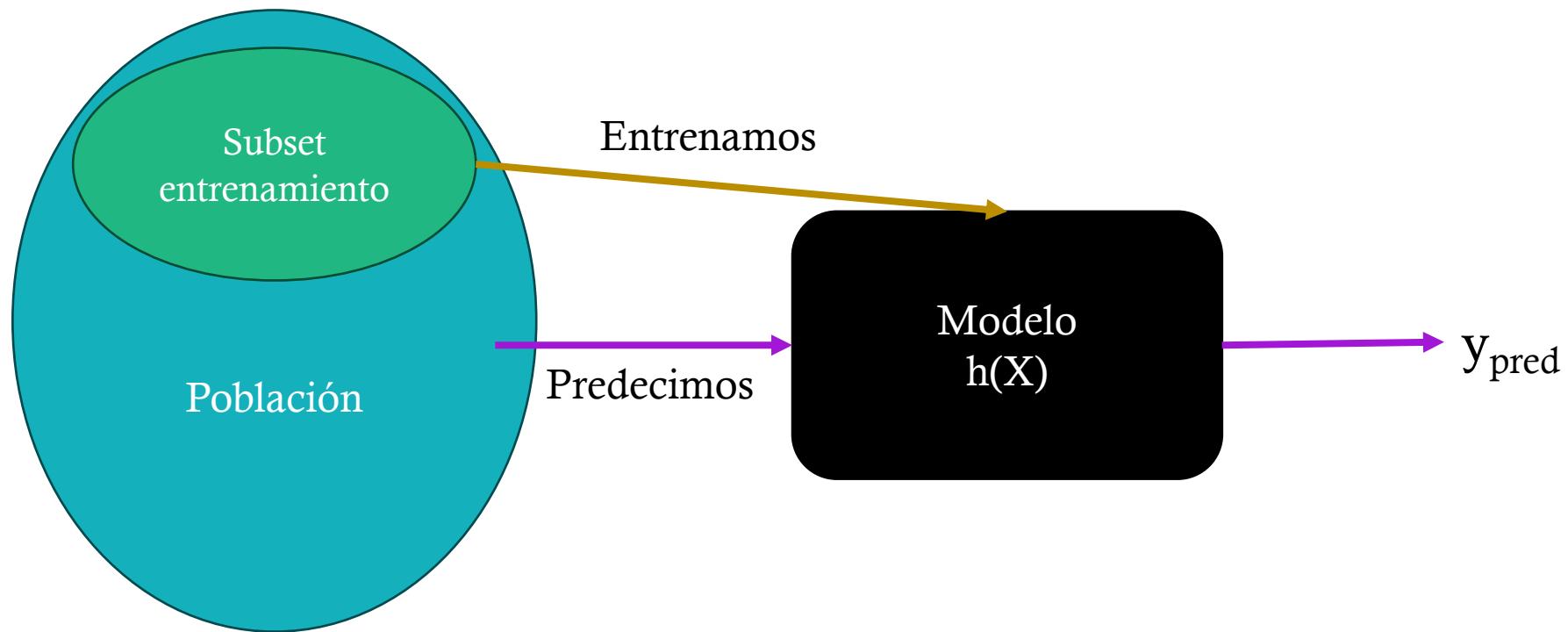


---

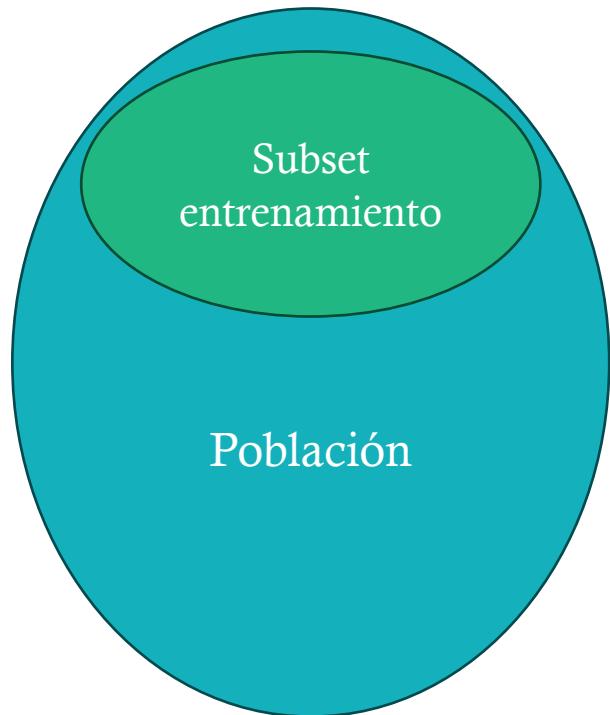
# APRENDIZAJE AUTOMÁTICO

# FORMAS DE APRENDIZAJE

Un esquema de aplicar Aprendizaje Automático nos queda...



# FORMAS DE APRENDIZAJE



Un paso importante que nos delata este grafico es que, en la realidad no podemos saber exactamente qué tan bien funcionará un modelo predictivo en la práctica porque no conocemos la verdadera distribución de los datos.

Siempre vamos a trabajar con sub-sets que podemos estimar y optimizar el rendimiento del modelo en un conjunto conocido de datos de entrenamiento.

El rendimiento sobre este conjunto conocido de datos de entrenamiento se denomina **riesgo empírico**.

# FORMAS DE APRENDIZAJE

Como vimos en los ejemplos hay diferentes tipos de aprendizaje, los cuales depende de la forma que tiene principalmente **y**.

- **Aprendizaje supervisado:** El modelo observa pares de entradas-salidas y aprende la relación entre ellos. Es decir, en este tipo de aprendizaje, conocemos el valor de  $y$  y se lo enseñamos al modelo.
  - Los modelos aprenden de los resultados conocidos y realizan ajustes en sus parámetros interiores para adaptarse a los datos de entrada.
  - Una vez que el modelo es entrenado adecuadamente, y los parámetros internos son coherentes con los datos de entrada y los resultados de los datos de entrenamiento, el modelo podrá realizar predicciones adecuadas ante nuevos datos



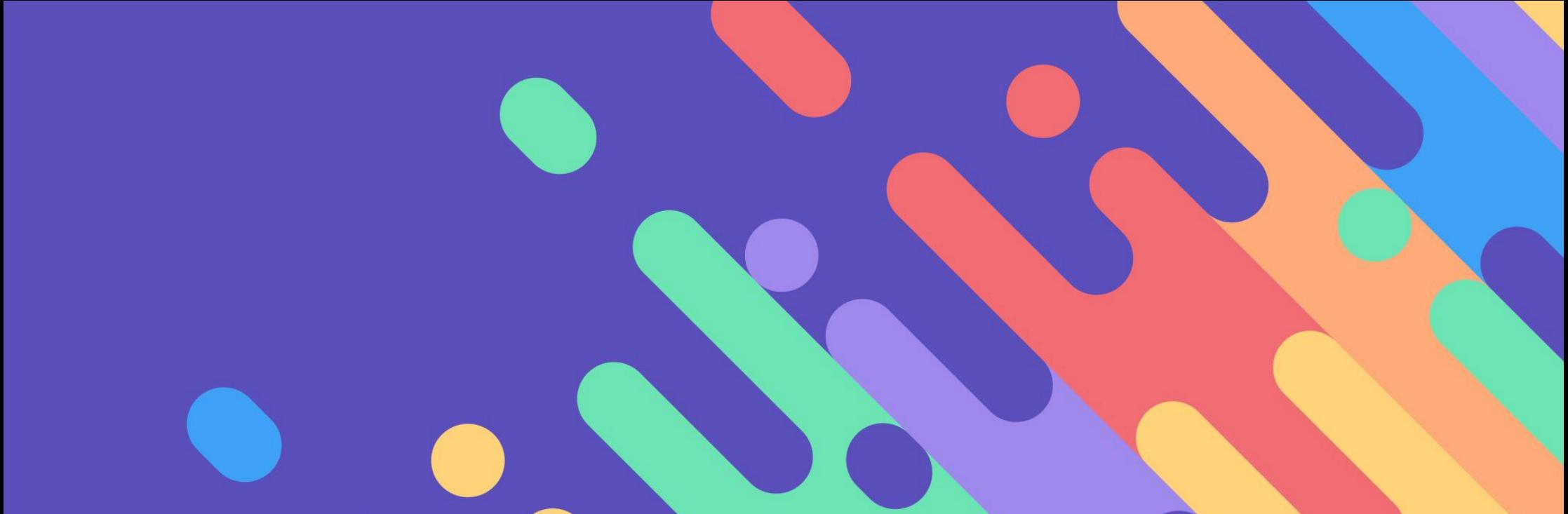
Image by [vectorjuice](#)

---

# FORMAS DE APRENDIZAJE

Como vimos en los ejemplos hay diferentes tipos de aprendizaje, los cuales depende de la forma que tiene principalmente **y**.

- **Aprendizaje no supervisado:** El modelo aprende patrones de la entrada sin ninguna retroalimentación. Es decir, no contamos con **y** de antemano.
- **Aprendizaje por refuerzo:** El agente aprende con una serie de refuerzos: recompensas y castigos. Depende del agente decidir cuál de las acciones anteriores al refuerzo fue la más responsable de él y modificar sus acciones para apuntar a más recompensas en el futuro.



---

# APRENDIZAJE SUPERVISADO

---

# APRENDIZAJE SUPERVISADO

Mas formalmente podemos definir a la tarea de aprendizaje supervisado como:

Dado un **set de entrenamiento** de N observaciones de pares de entradas y salida:

$$(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)$$

Donde cada par está generado por una función desconocida  $y = f(X)$ ,

Se descubre una función **h** que aproxima a la verdadera función f.

La función **h** es la que llamamos a la hipótesis de la población o el **modelo** que provienen de un espacio de hipótesis.

Llamamos a las salidas **y** como **ground truth**.

---

# APRENDIZAJE SUPERVISADO

## *¿Como elegimos el espacio de hipótesis?*

Es posible que tengamos algún conocimiento previo sobre el proceso que generó los datos. O podemos realizar un **análisis de datos exploratorio**: examinar los datos con pruebas estadísticas y visualizaciones para tener una idea de los datos y una idea de qué espacio de hipótesis podría ser apropiado. O simplemente podemos probar múltiples espacios de hipótesis y evaluar cuál funciona mejor.

## *¿Y cómo elegimos un modelo dentro del espacio de hipótesis?*

Podríamos esperar una hipótesis consistente: una **h** tal que cada  $X$  en el conjunto de entrenamiento tenga  $h(x) = y$ . Si las salidas son de valor continuo es muy difícil tener una salida exacta. En esos casos se usa una **función de ajuste** para la cual cada  $h(x_i)$  esté cerca de  $y_i$ .

---

---

# APRENDIZAJE SUPERVISADO

## Generalización

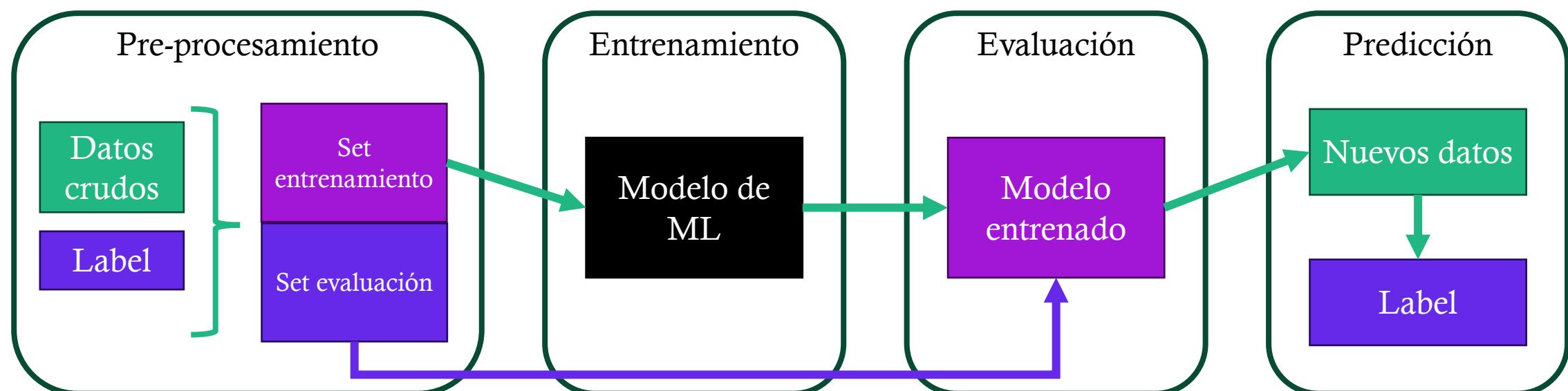
La verdadera medida de si un modelo funciona correctamente no es con respecto al set de entrenamiento, sino que tan bien maneja entradas que nunca vio.

Eso lo podemos ver con un segundo set de pares  $(X_i, y_i)$  llamada **conjunto de prueba**.

Se dice que  $h$  **generaliza** bien si predice con precisión los resultados de este conjunto.

# APRENDIZAJE SUPERVISADO

## Generalización



---

# APRENDIZAJE SUPERVISADO

## Tipos de aprendizaje supervisado

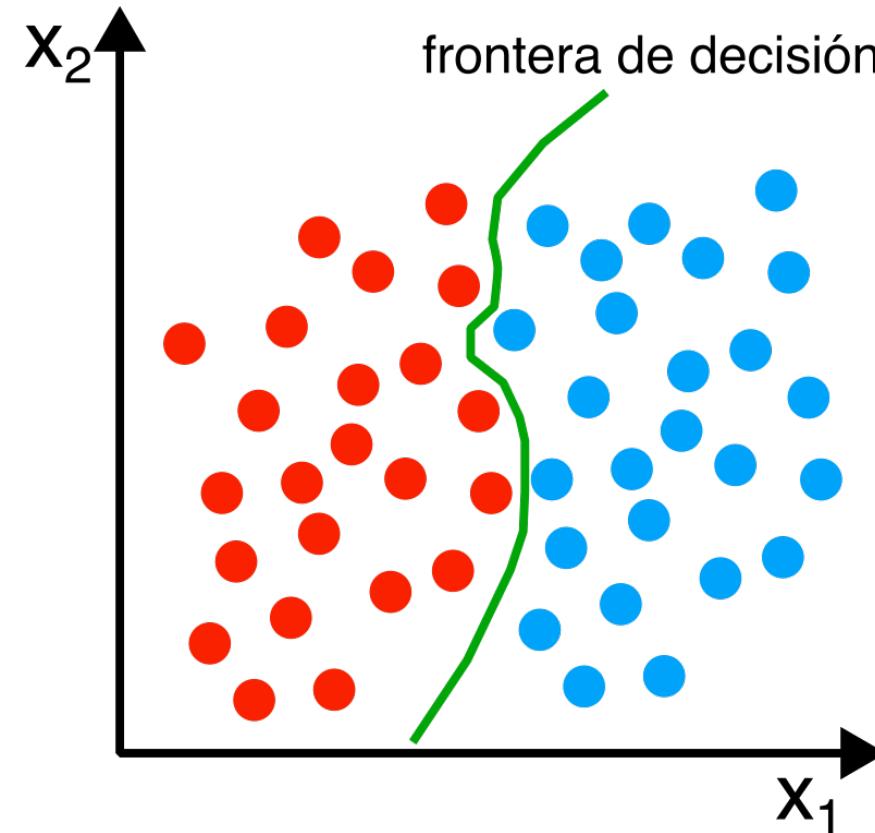
Si el target **y** es una *variable categórica* o toma valores discretos, este tipo de problema se llama un problema de clasificación. A su vez tenemos dos subvariantes:

- **Clasificación binaria** (Por ejemplo, es SPAM o no SPAM).
- **Clasificación multi-clase:** Múltiple clases, como por ejemplo clasificación del nivel socioeconómico de una persona (alta, media y baja).
  - Una variante de este tipo es cuando la cardinalidad es muy alta, es decir, se tienen muchísimas clases.

# APRENDIZAJE SUPERVISADO

Tipos de aprendizaje supervisado

Clasificación



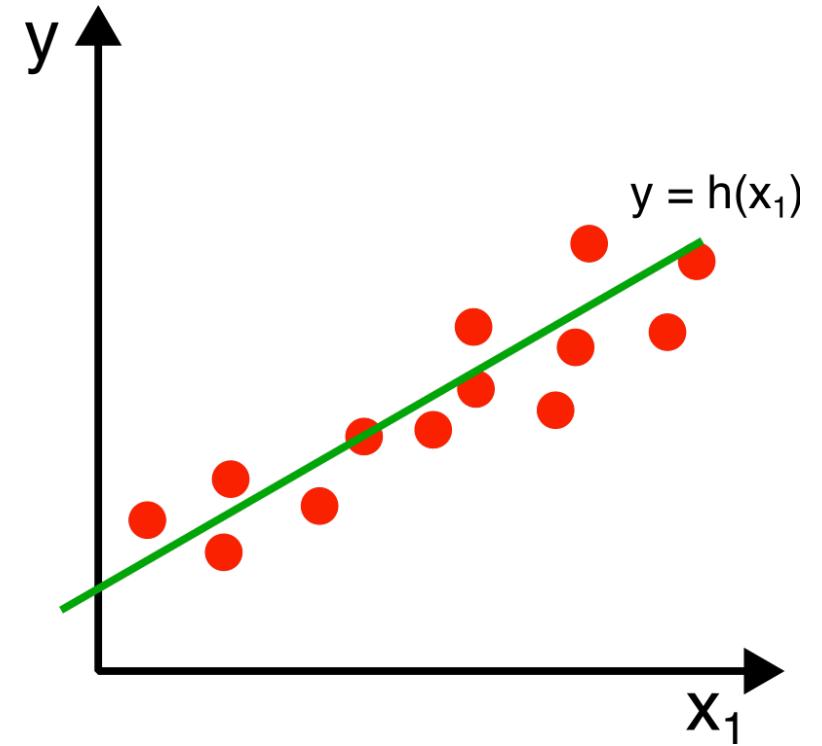
# APRENDIZAJE SUPERVISADO

## Tipos de aprendizaje supervisado

Si el target  $y$  es una *variable numérica*, este tipo de problema se llama un problema de regresión.

Se centra en estudiar las relaciones entre una variable dependiente de una o más variables independientes.

Es importante notar que, en Aprendizaje Automático, cuando buscamos una  $h(X)$  estamos armando un modelo puramente empírico. Es decir, nos basamos 100% en los datos medidos. En contraste con los modelos basados en propiedades fundamentales.



---

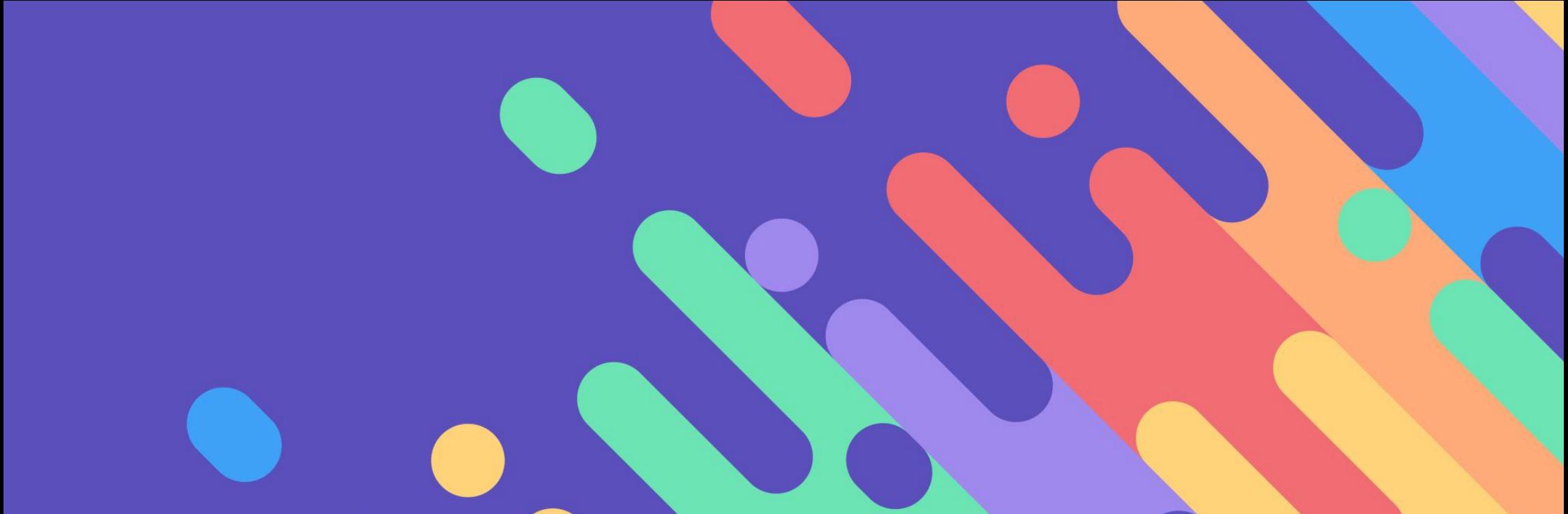
# APRENDIZAJE SUPERVISADO

Tipos de aprendizaje supervisado

## Regresión vs. Clasificación

Regresión y clasificación son problemas muy similares entre sí. En ambos buscamos predecir una variable, la diferencia radica en que regresión predice una variable numérica y clasificación una categórica.

*No es infrecuente encontrar que se puede resolver problemas como clasificación o regresión.*

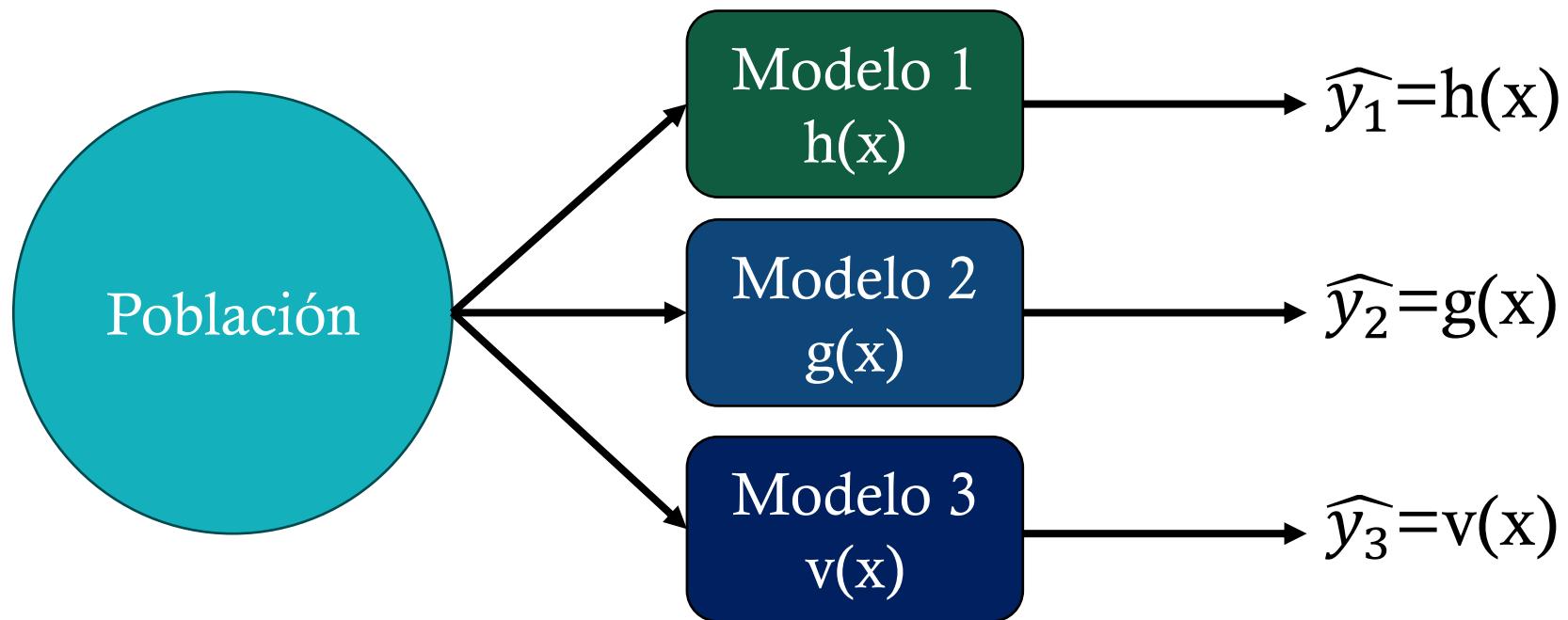


---

# SESGO Y VARIANZA

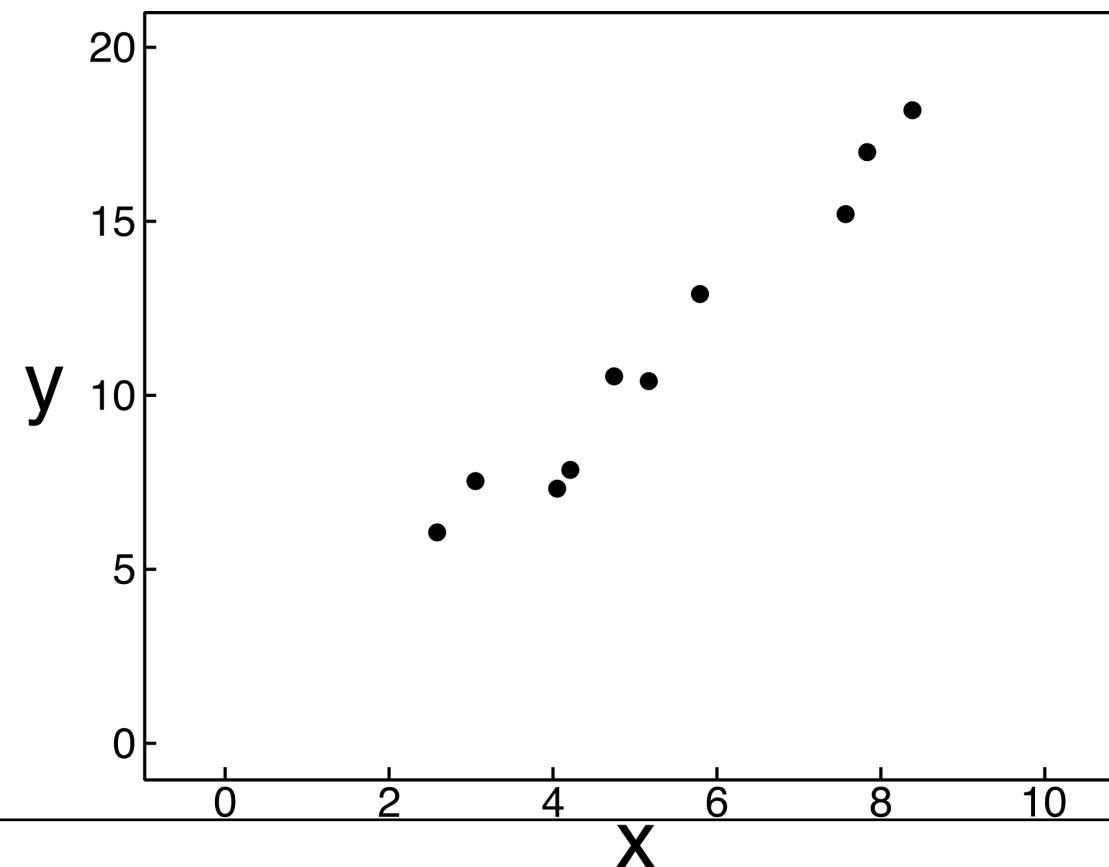
# SESGO Y VARIANZA

Supongamos que tenemos tres modelos en un problema de regresión. Tenemos una sola entrada  $\mathbf{x}$  y una salida  $\mathbf{y}$  que depende de  $\mathbf{x}$  con una relación  $f(\mathbf{x})$  que queremos modelar:



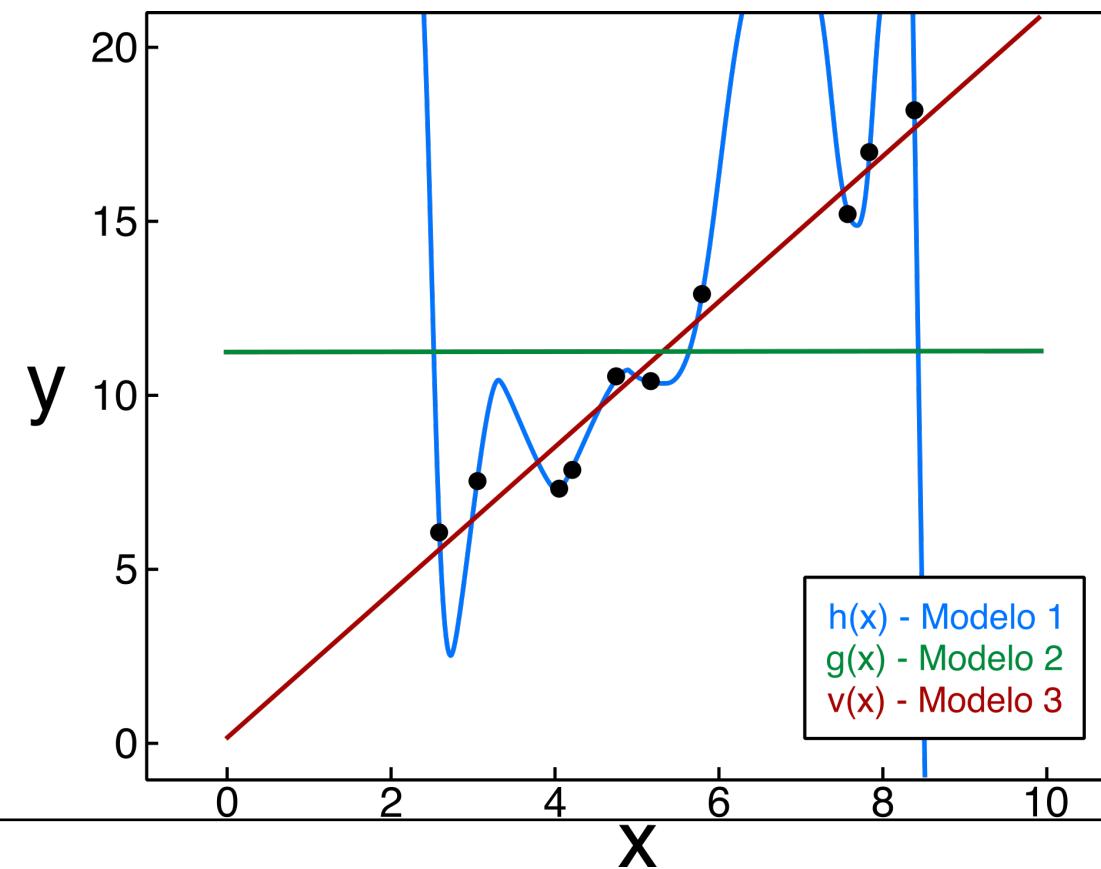
# SESGO Y VARIANZA

Entrenamos con un **set de entrenamiento** y nos queda:



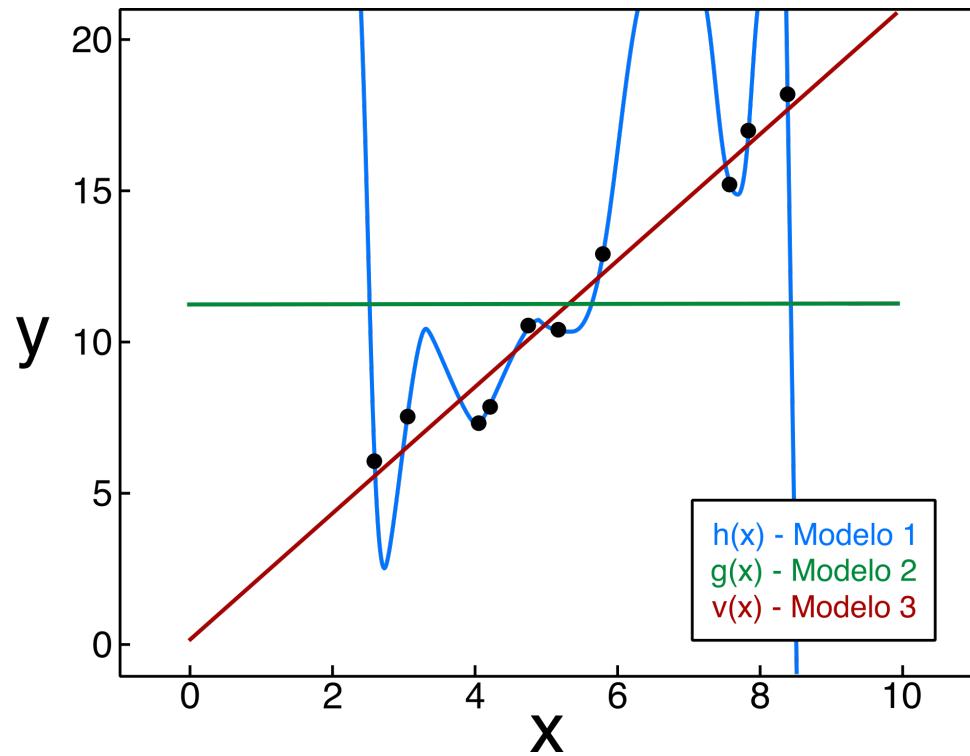
# SESGO Y VARIANZA

Entrenamos con un **set de entrenamiento** y nos queda:



# SESGO Y VARIANZA

Entrenamos con un **set de entrenamiento** y nos queda:



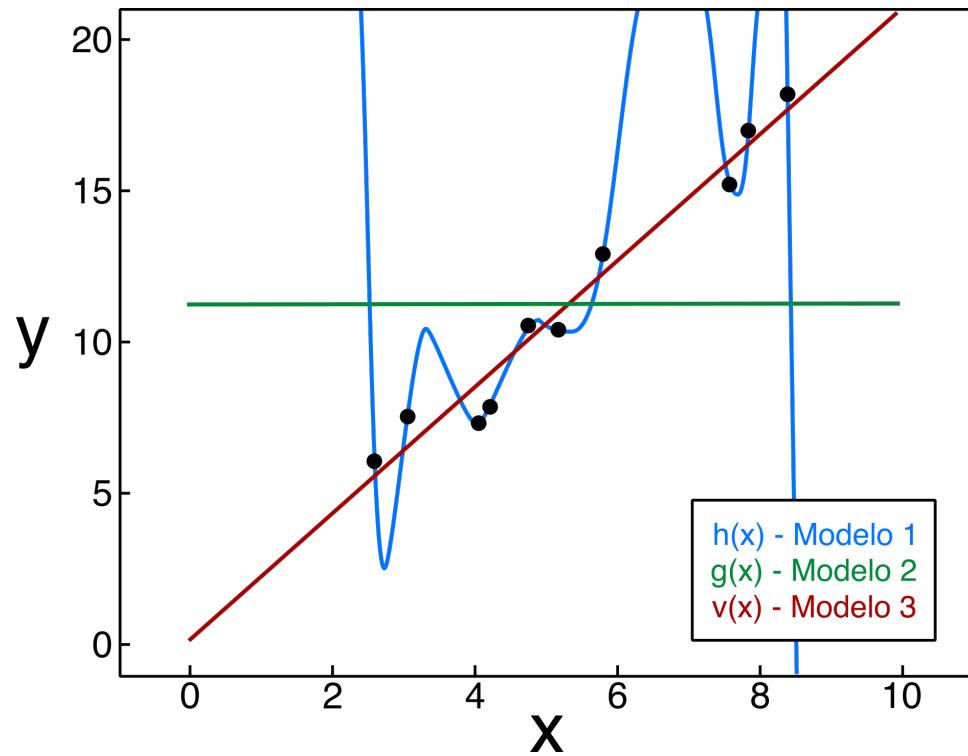
Midamos el error, para ello calculamos de la siguiente forma:

$$MAE_p = \sum_{i=1}^N (y_i - \hat{y}_p)$$

- Modelo 1:  $MAE_1 = 0$
- Modelo 2:  $MAE_2 = 5$
- Modelo 3:  $MAE_3 = 1.25$

# SESGO Y VARIANZA

Entrenamos con un **set de entrenamiento** y nos queda:



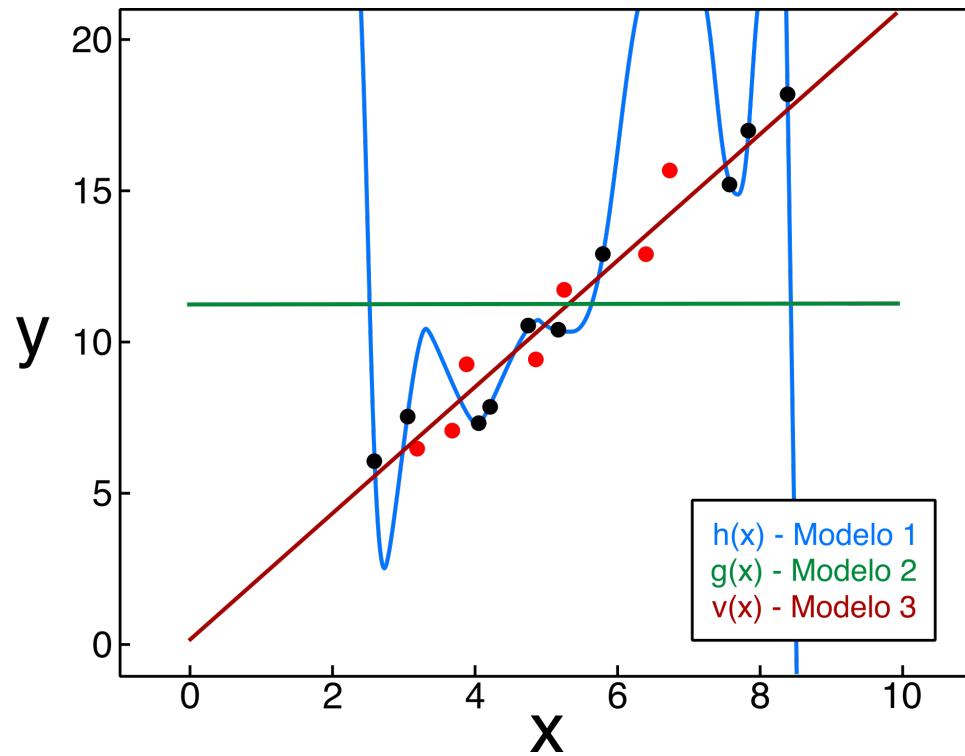
Midamos el error, para ello calculamos de la siguiente forma:

$$MAE_p = \sum_{i=1}^N (y_i - \hat{y}_p)$$

- Modelo 1:  $MAE_1 = 0$
- Modelo 2:  $MAE_2 = 5$
- Modelo 3:  $MAE_3 = 1.25$

# SESGO Y VARIANZA

¿Pero realmente es el mejor? Evaluemos con el **set de evaluación**



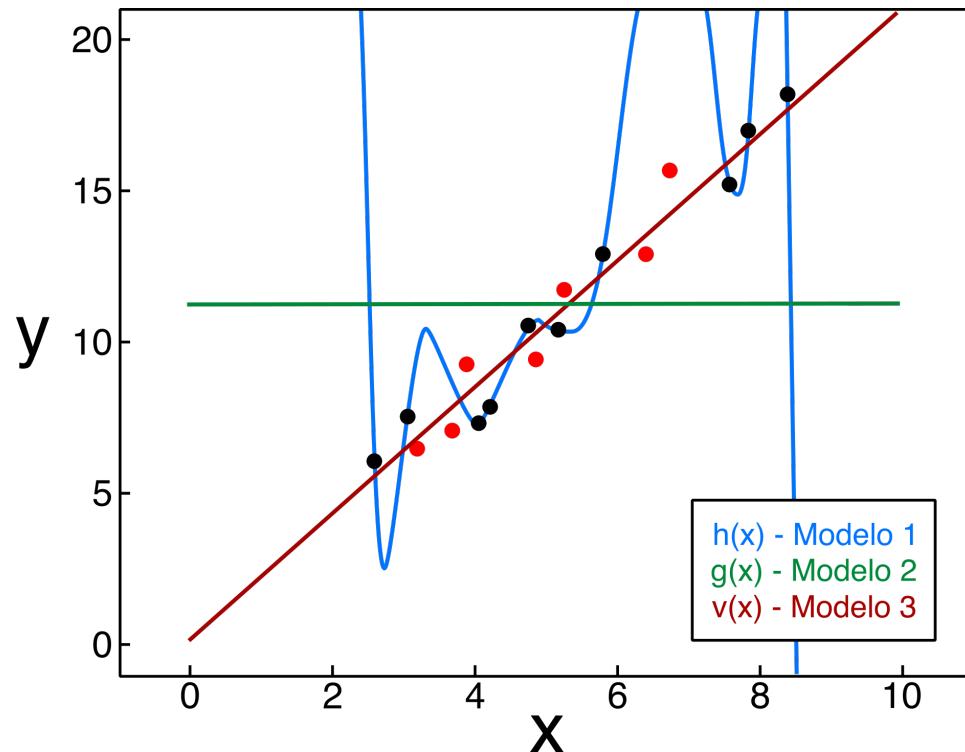
Midamos el error...

$$MAE_p = \sum_{i=1}^N (y_i - \hat{y}_p)$$

- Modelo 1:  $MAE_1 = 90$
- Modelo 2:  $MAE_2 = 5.2$
- **Modelo 3:  $MAE_3 = 1.75$**

# SESGO Y VARIANZA

¿Pero realmente es el mejor? Evaluemos con el **set de evaluación**

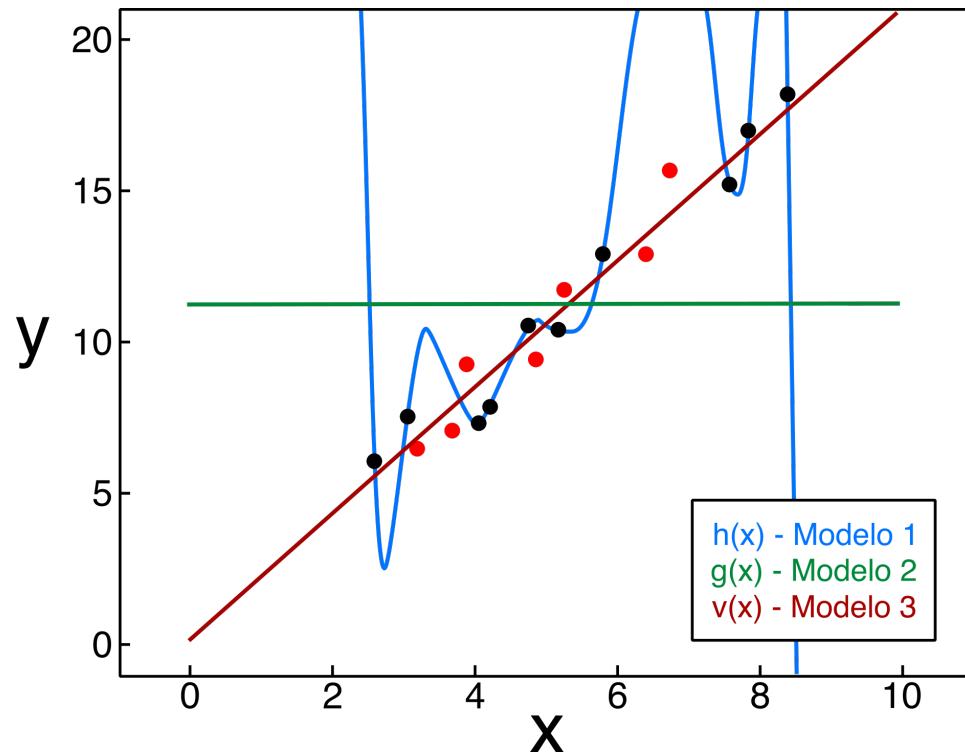


- El modelo 1 rinde muy bien con los datos de entrenamiento, pero muy mal con el de evaluación. Esto es lo que se llama **sobreajuste**, el modelo aprende tan bien que hasta aprende el ruido de los datos.  
Este modelo tiene demasiado parámetros, por lo que es demasiado complejo y por consiguiente **no generaliza**.
- Cuando se elige un modelo, se busca que:

*parametros << cantidad de obs. de entrenamiento*

# SESGO Y VARIANZA

¿Pero realmente es el mejor? Evaluemos con el **set de evaluación**



- El modelo 2 rinde pobemente con el set de entrenamiento y el de evaluación. Este modelo tiene no logra capturar el comportamiento buscado.

Cuando esto ocurre es lo que llamamos **subajuste**.

- El modelo 3 es el mejor modelo, el resultado entre el de evaluación y entrenamiento es similar, el modelo **generaliza**, sin tomar el ruido.

---

# SESGO Y VARIANZA

El error que se comete con un modelo que sobre-ajusta y sub-ajusta es diferente.

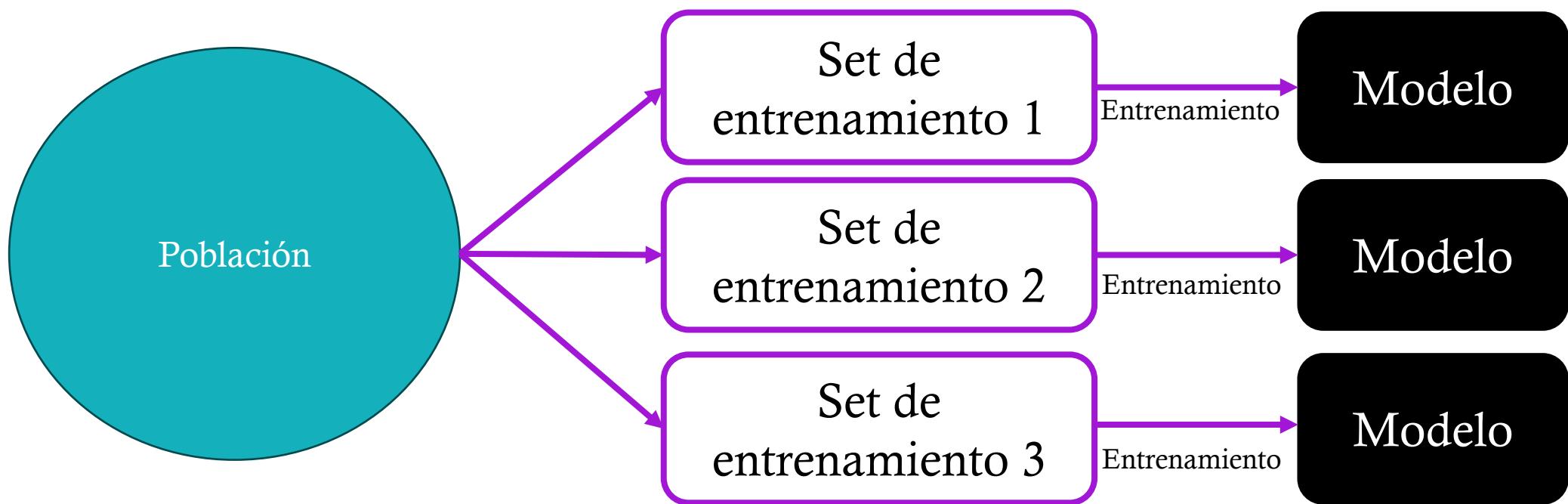
- El error de sobreajuste le llamamos **error de varianza**
- El error de subajuste le llamamos **error de sesgo**

Todo modelo el error total tiene como parte a los dos errores.

Estos errores son complementarios entre sí. Si queremos bajar uno, el otro va a subir.

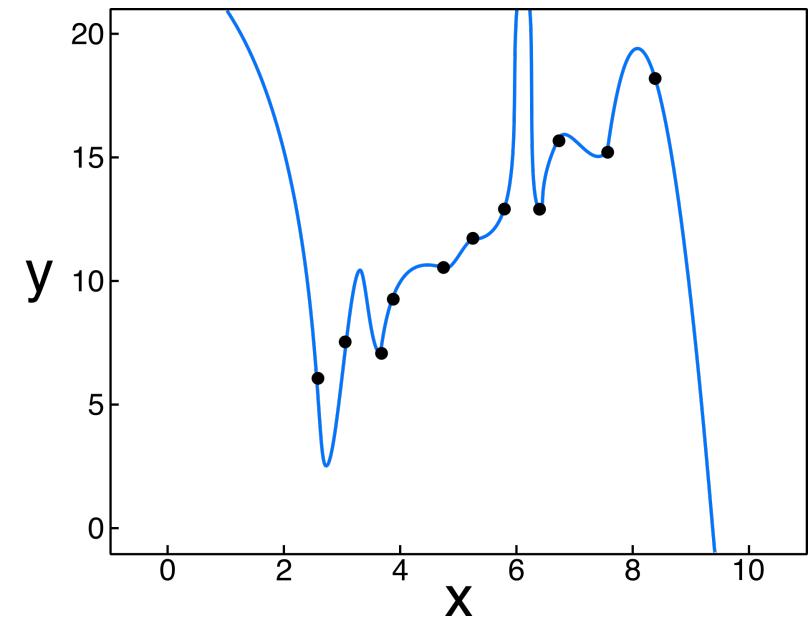
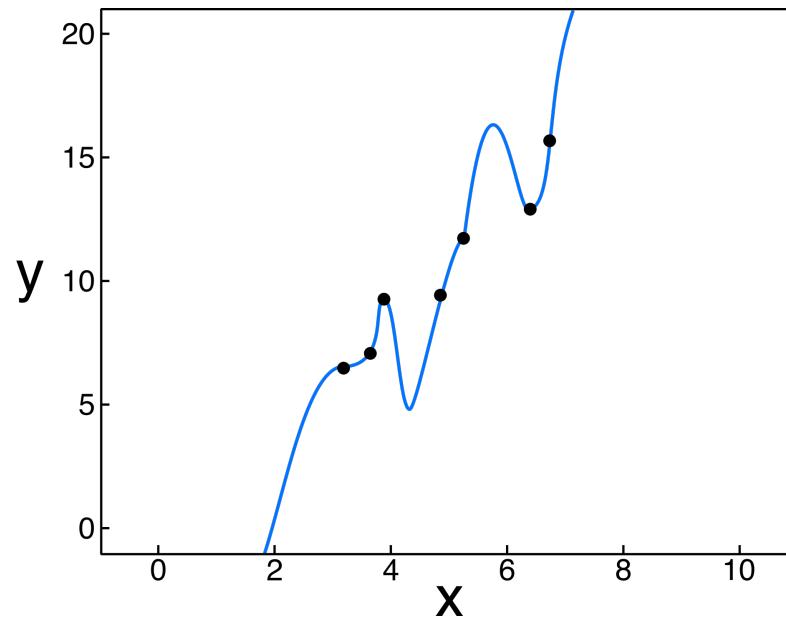
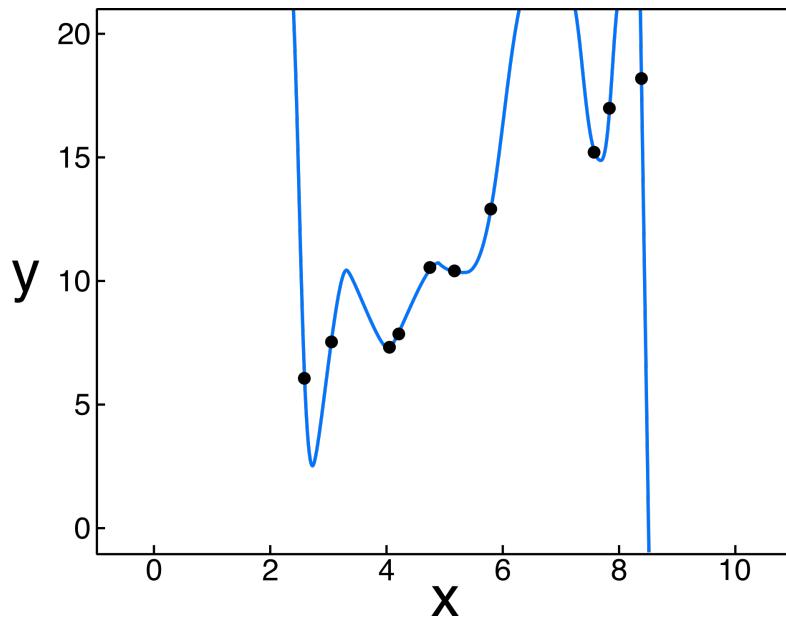
# SESGO Y VARIANZA

Estos errores lo podemos ver si tomamos diferentes **sets de entrenamiento** y entrenamos el modelo con cada uno de estos.



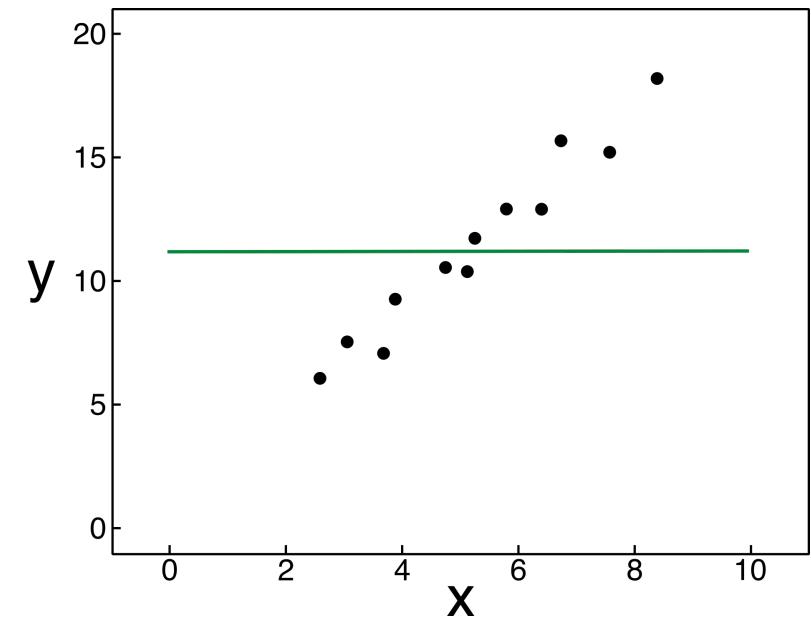
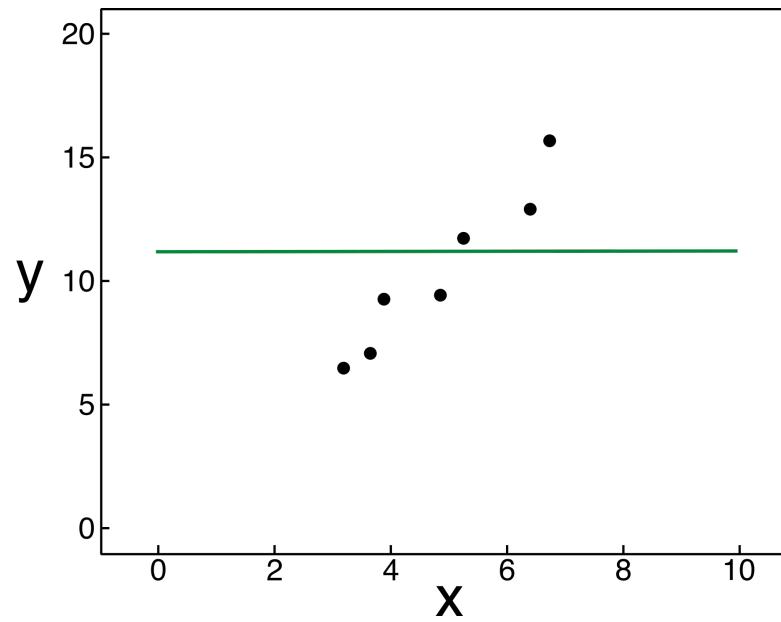
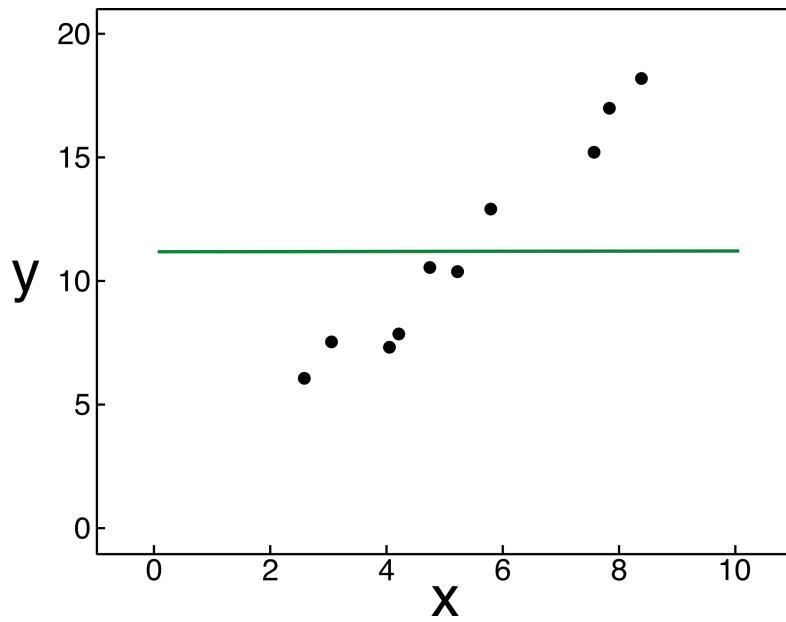
# SESGO Y VARIANZA

Si tenemos un modelo que sobreajusta, diferentes sets de entrenamiento nos darán modelos muy diferentes:



# SESGO Y VARIANZA

Si tenemos un modelo que subajusta, diferentes sets de entrenamiento nos darán modelos muy similares:



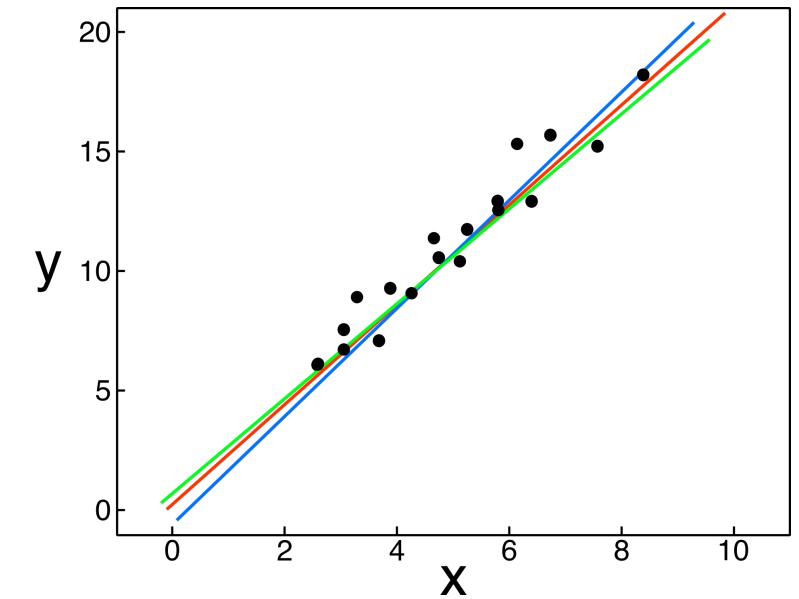
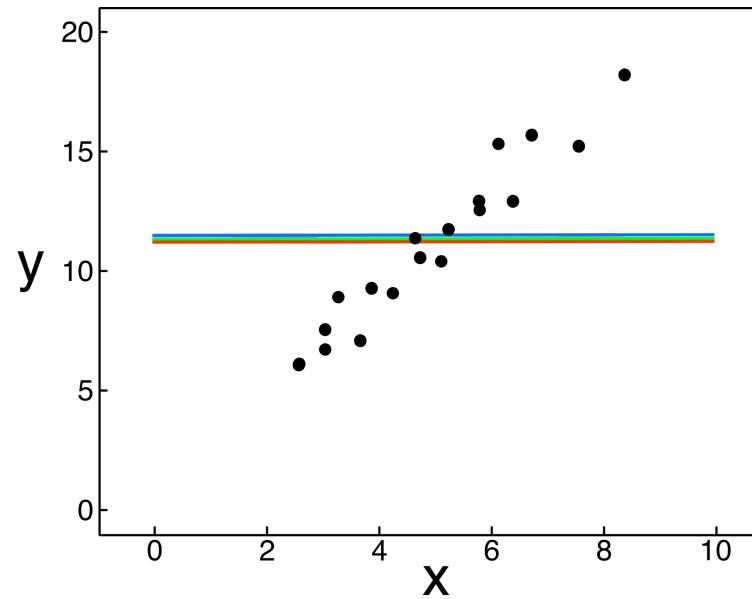
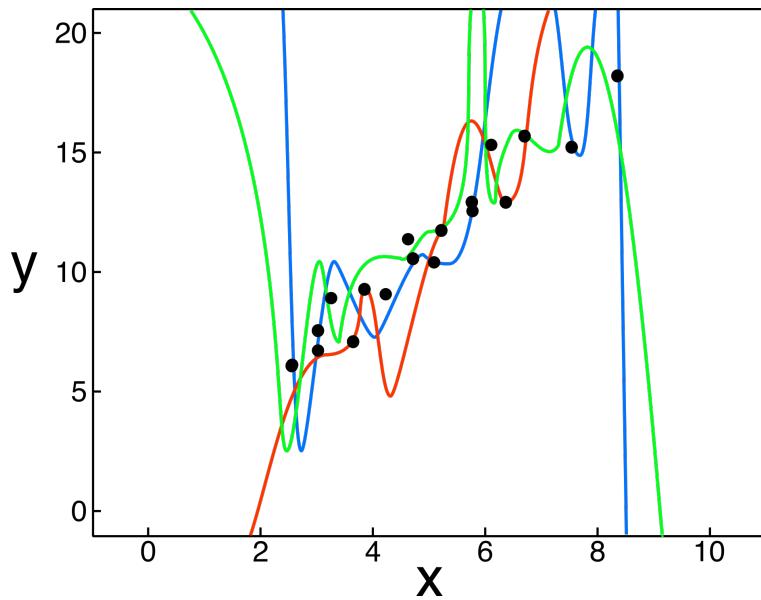
# SESGO Y VARIANZA

---

- El modelo más simple cometerá muchos errores en prácticamente cualquier conjunto de entrenamiento, lo que significa que tiene **un error alto de sesgo**. Sin embargo, cuando evaluamos con el set de evaluación a estos modelos, el resultado siempre es similar. Entonces decimos que tiene **un error de varianza bajo**. *Un sesgo alto y una varianza baja suelen corresponder a un subajuste.*
- Por otro lado, el modelo complejo se adapta perfectamente a cada conjunto de entrenamiento. Tiene **un error de sesgo muy bajo** pero un **error de varianza muy alto** (ya que dos conjuntos de entrenamiento cualesquiera probablemente darían lugar a modelos muy diferentes). *Esto corresponde a un sobreajuste.*

# SESGO Y VARIANZA

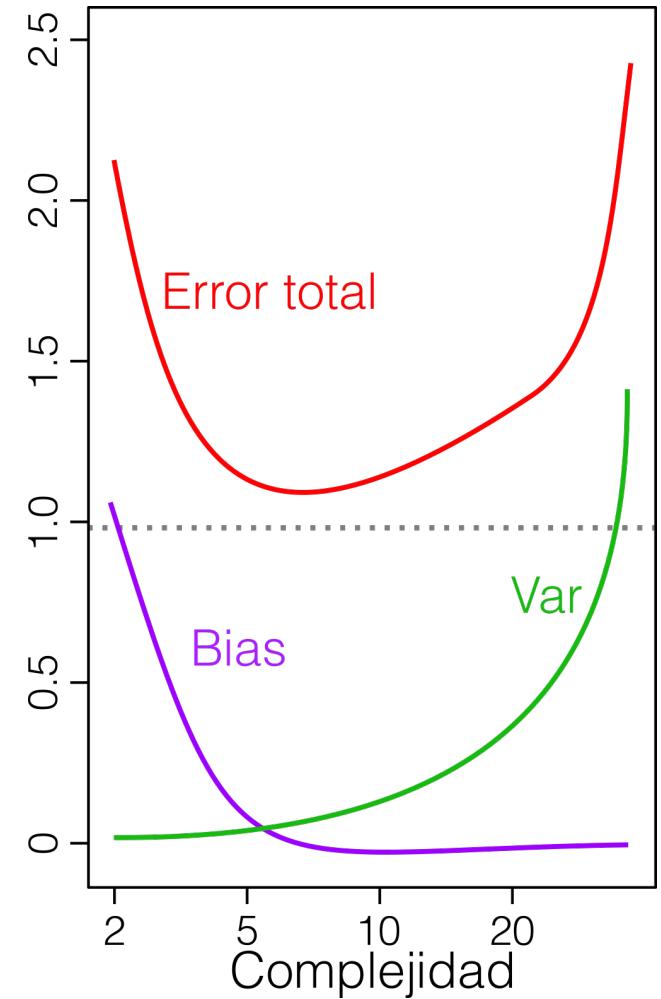
- Si vemos para cada caso, vemos que el modelo lineal, aunque tiene varianza, el error es chico, ya que tiene poca varianza y poco sesgo.

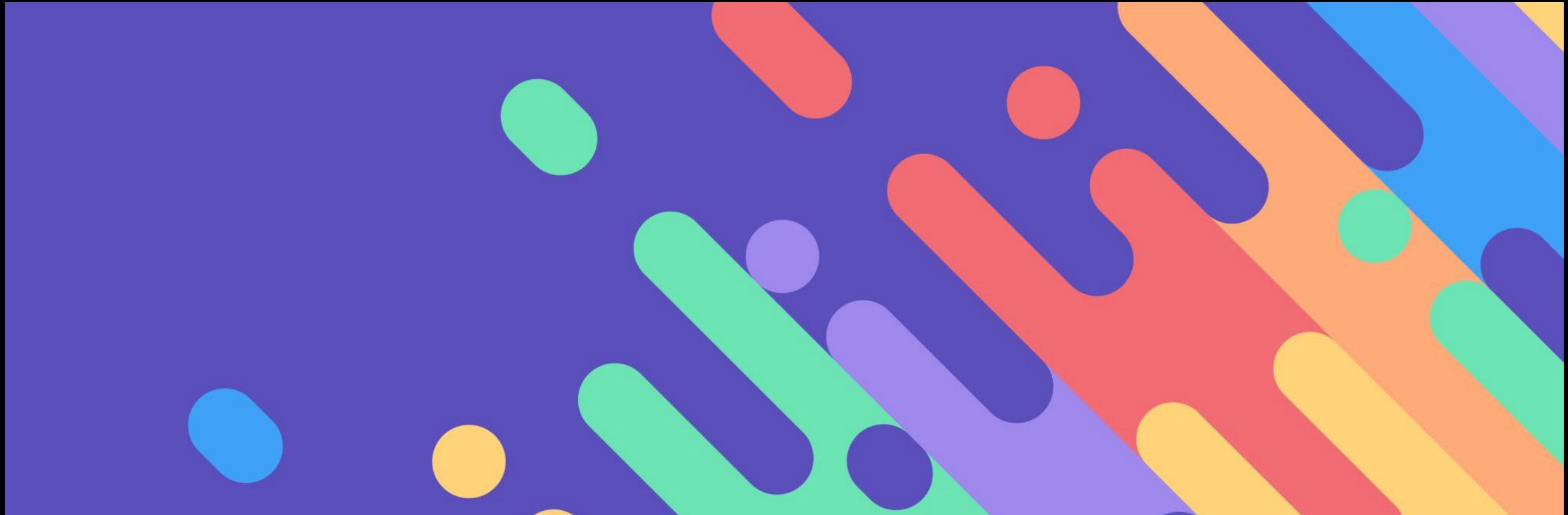


# SESGO Y VARIANZA

*Como regla general,*

- Cuando más complejo es el modelo, la varianza va a aumentar y el sesgo va a disminuir.
- Cuando aumentamos la complejidad de este, el sesgo tiende a disminuir más rápido de lo que la variabilidad aumenta, disminuyendo el error.
- Llega un punto en donde el efecto de la variabilidad es apreciable, aumentando el valor del error de nuevo.





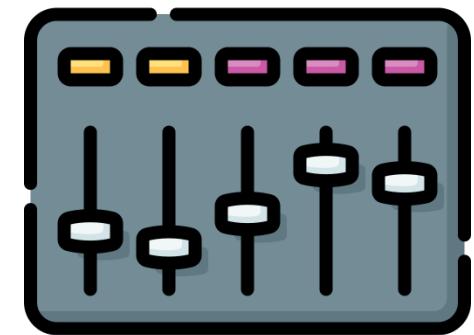
---

# ESTRATEGIAS PARA DISMINUIR EL RIESGO EMPÍRICO

# ESTRATEGIAS

Pensemos ahora a los modelos como cajas grises, todavía no sabemos cómo funcionan, pero tenemos una idea que tienen parámetros, estos son de dos tipos:

- **Parámetros que se entrena**n: Son los parámetros que un modelo aprende cuando lo entrenamos. Principalmente son números como coeficientes de un polinomio o los pesos sinápticos de una red, o variables categóricas en arboles de decisión. Ya vimos que su número debía ser mucho menor que la cantidad de datos de entrenamiento para evitar el sobreajuste.
- **Hiper-parámetros**: Son parámetros *internos* que no dependen de los datos. Estos deben ser definidos previo al entrenar. Por ejemplo, una red neuronal tiene que definirse la función de activación o el orden del polinomio.



Consola iconos creados por [Freepik](#) - Flaticon

# ESTRATEGIAS



Elegir estos hiper-parámetros es un desafío importante, ya que, para tener el mejor modelo, deberíamos saber de ante-mano, pero para saberlo, debemos entrenar.

Entonces lo que se hace es entrenar muchos modelos con diferentes hiperparámetros, y ver cuál es el mejor.

¡Pero si usamos al **set de evaluación**, vamos a encontrar el mejor para este con riesgo de que **no generalizemos!**

---

# ESTRATEGIAS

Entonces debemos usar un tercer dataset que sacamos del de entrenamiento, llamado **set de validación**.

- Este set nos permite evaluar diferentes valores de hiper-parámetros
- Ver si el modelo está sobre-ajustando.
- Evaluar el modelo mientras aprende. En Deep Learning es normal usarlo para evaluar en el medio, el entrenamiento de estas redes es con gradiente descendiente, y es importante ver cada cierta iteración cómo evoluciona.

Con este set evitamos usar el de evaluación y “*“hacer trampa”*”.

---

# ESTRATEGIAS

Esto mejora la generalización, pero es muy sensible a la selección del conjunto de validación. Además, quitamos datos que nos podría haber servido para entrenar.

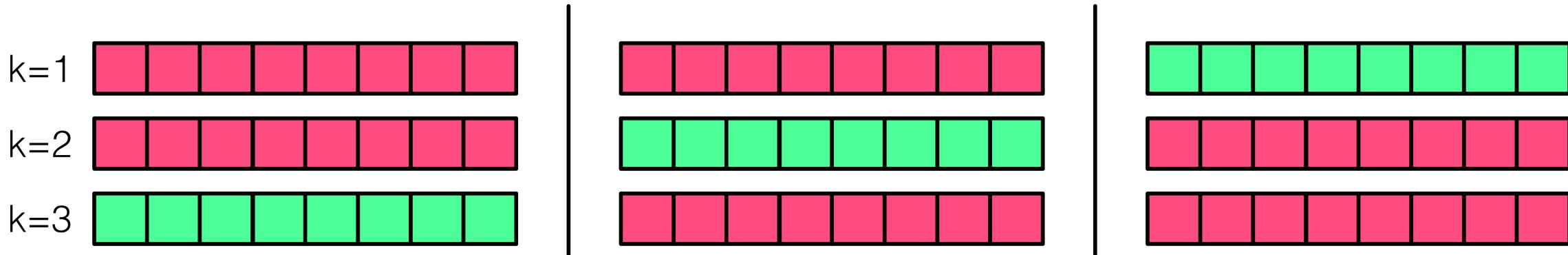
Una estrategia que nos evita esto es usar **Validación cruzada**.

# ESTRATEGIAS

## Validación cruzada

La más conocida es **Validación cruzada K-Fold**.

Los datos de entrenamientos se dividen en K sub-conjuntos. Con esta separación, realizamos el proceso de validación que vimos previamente, pero K veces. Cada vez vamos eligiendo set de validación a un conjunto K y a los restantes K-1 como entrenamiento. Una vez finalizado, los errores medidos se promedian para obtener la efectividad total del modelo.



---

# ESTRATEGIAS

## Validación cruzada K-Fold.

### Beneficios:

- Estrategia sistemática para encontrar los hiperparámetros,
- Nos permite evaluar al menos una vez cada punto de entrenamiento
- Nos permite evaluar problemas de sobreajuste
- Sin tocar el conjunto de testeo.

### Desventaja:

- Es un proceso muy lento. Cada modelo se debe entrenar K veces.

*En procesos lentos y con set de entrenamientos muy grandes, conviene usar un set de validación.*

---

# ESTRATEGIAS

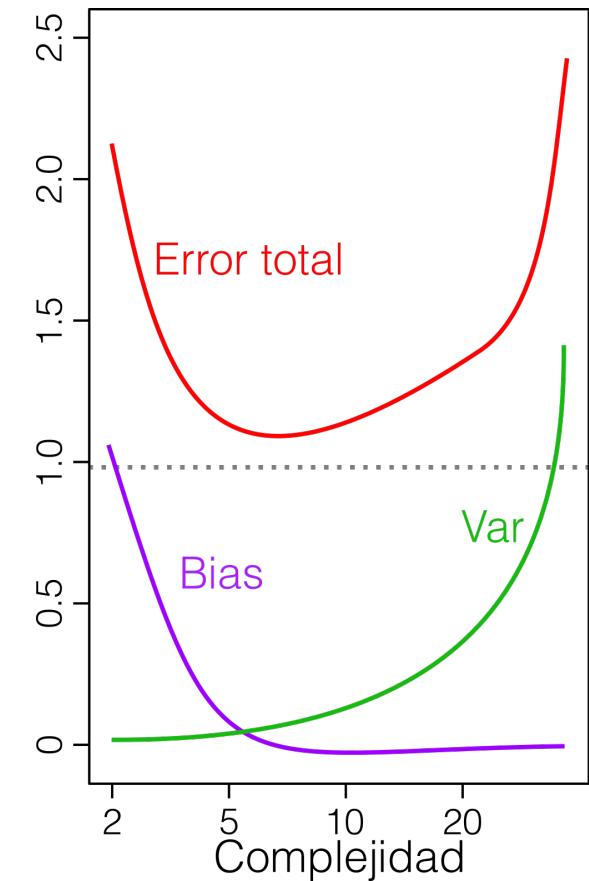
## Regularización

La estrategia anterior versa sobre los datos, pero también podemos trabajar sobre los modelos. Una forma es usando **métodos de regularización**.

La **regularización** es un método que nos permite restringir el proceso de estimación, se usa para evitar un posible sobre ajuste del modelo (overfitting).

La forma en que funciona esto es restringiendo que valores pueden adoptar los parámetros de entrenamiento del modelo o achicándolos.

Lo que buscamos en ese caso, es reducir el **error de varianza**, ya que el modelo no se va a poder mover tan libremente, pero aumentamos el **error de sesgo**, pero de tan suerte que lo que disminuye la varianza es mayor al aumento del sesgo.



# ESTRATEGIAS

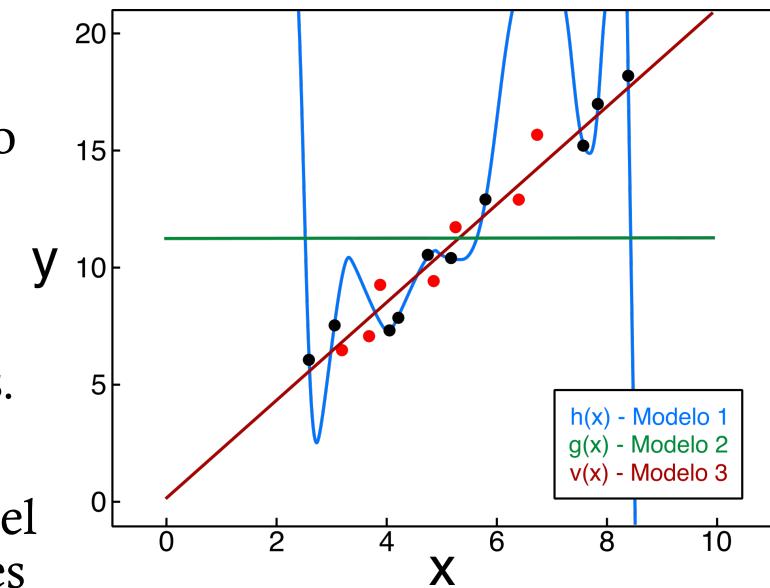
## Regularización

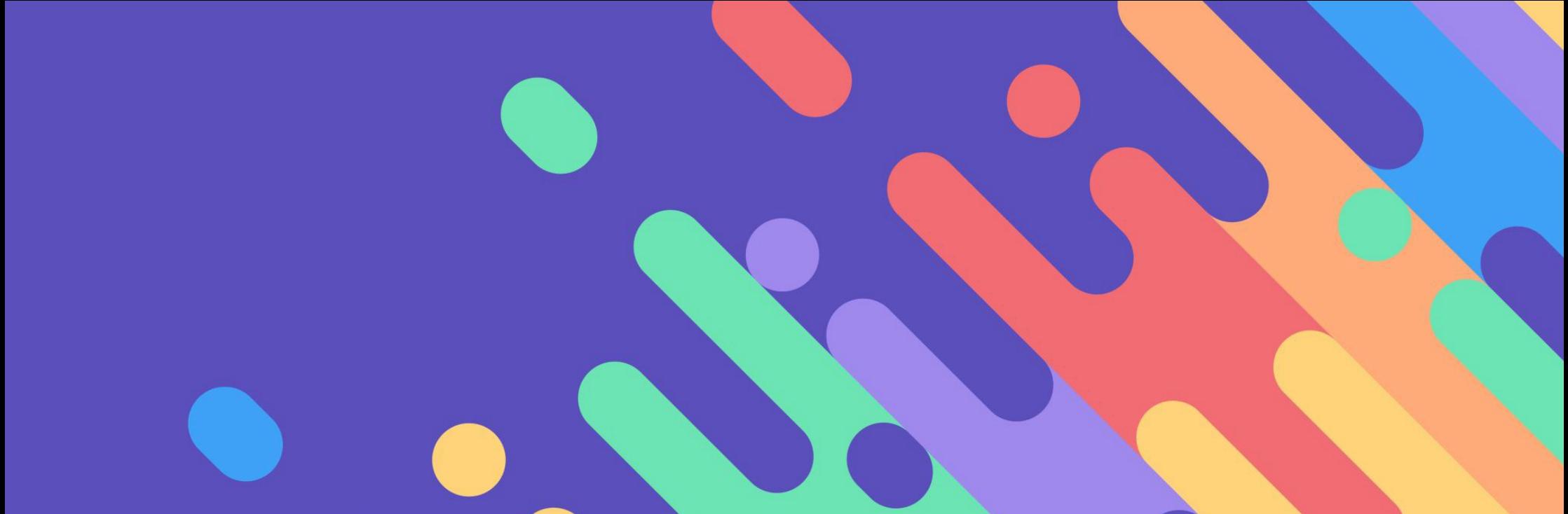
La estrategia anterior versa sobre los datos, pero también podemos trabajar sobre los modelos. Una forma es usando **métodos de regularización**.

La **regularización** es un método que nos permite restringir el proceso de estimación, se usa para evitar un posible sobre ajuste del modelo (overfitting).

La forma en que funciona esto es restringiendo que valores pueden adoptar los parámetros de entrenamiento del modelo o achicándolos.

Lo que buscamos en ese caso, es reducir el **error de varianza**, ya que el modelo no se va a poder mover tan libremente, pero aumentamos el **error de sesgo**, pero de tan suerte que lo que disminuye la varianza es mayor al aumento del sesgo.





---

# INSTALANDO LAS HERRAMIENTAS DE DESARROLLO

# 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.

# HERRAMIENTAS DE DESARROLLO



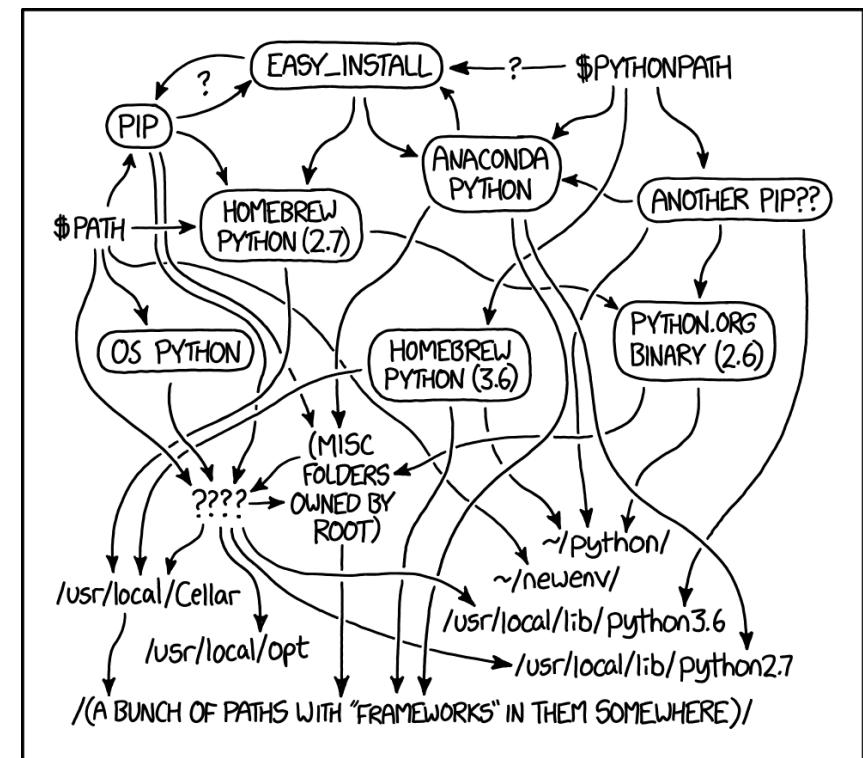
## Modo novato

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



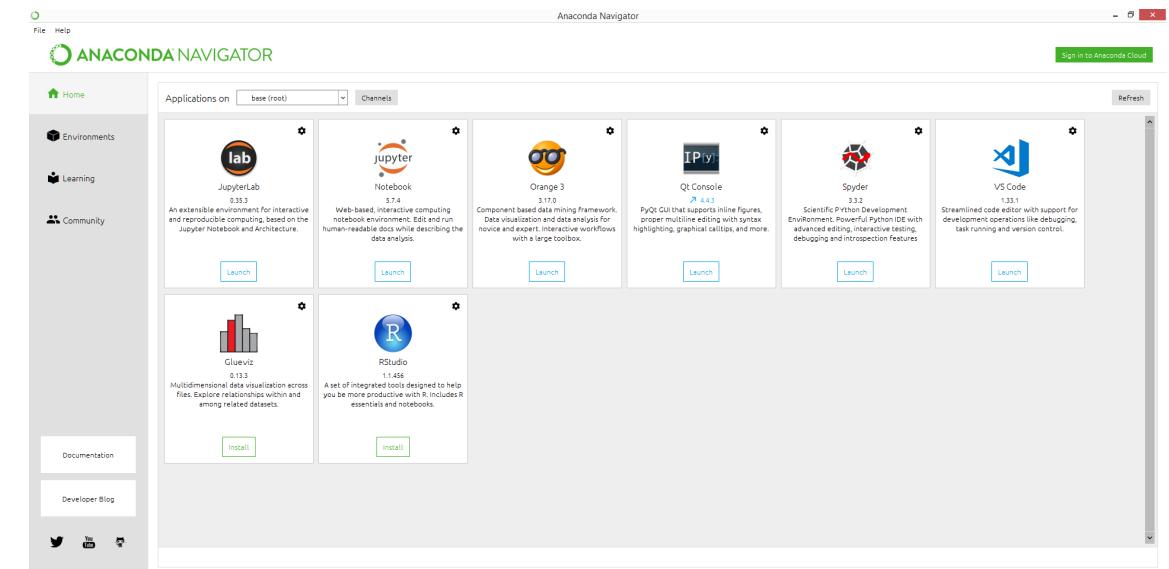
## Modo novato

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  $\geq 3.9$ . 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
```

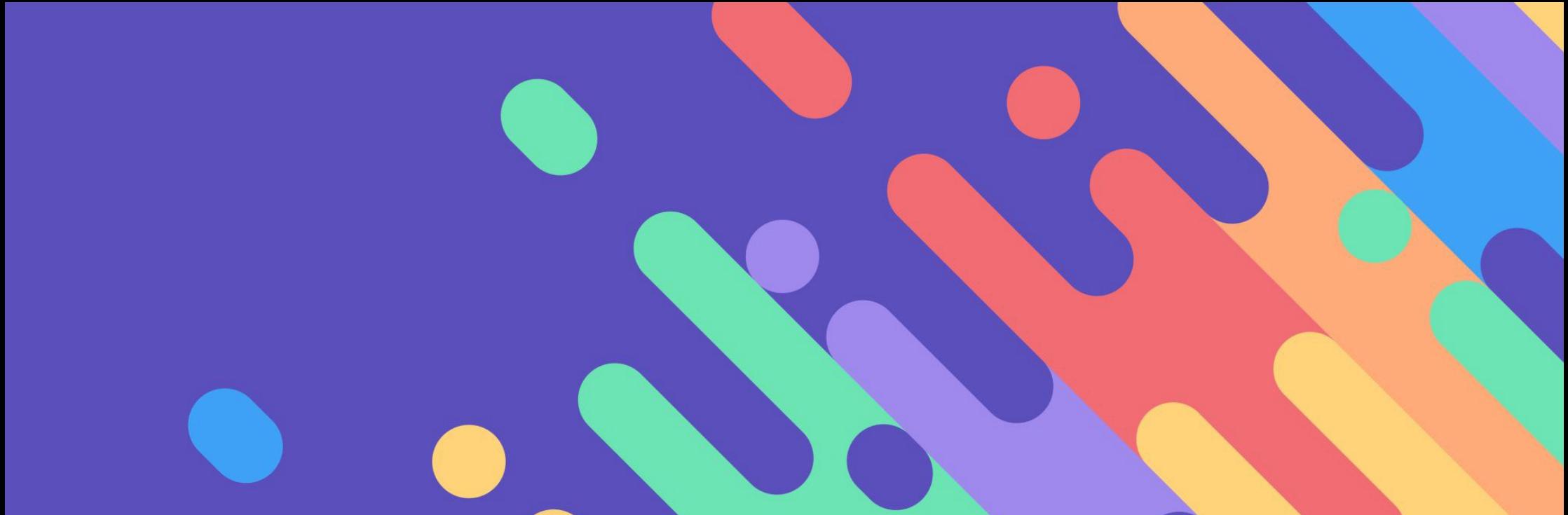
Al hacer esto se encarga de crear un entorno virtual para este proyecto para esa versión de Python.

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.



---

## 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

---

# 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

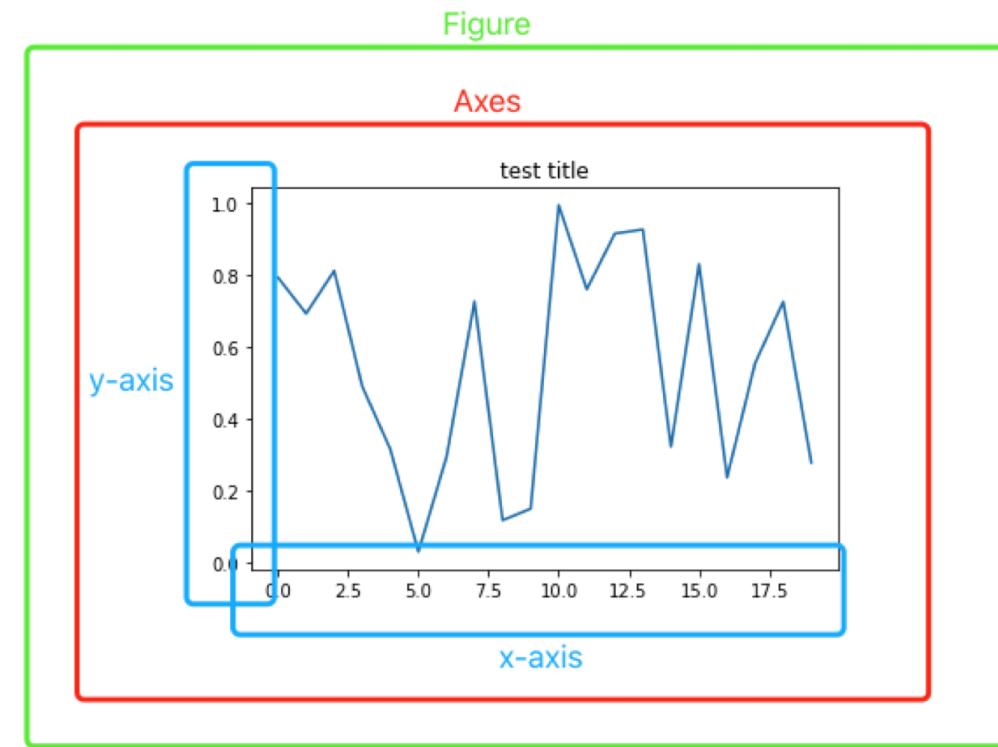
## DataFrame

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

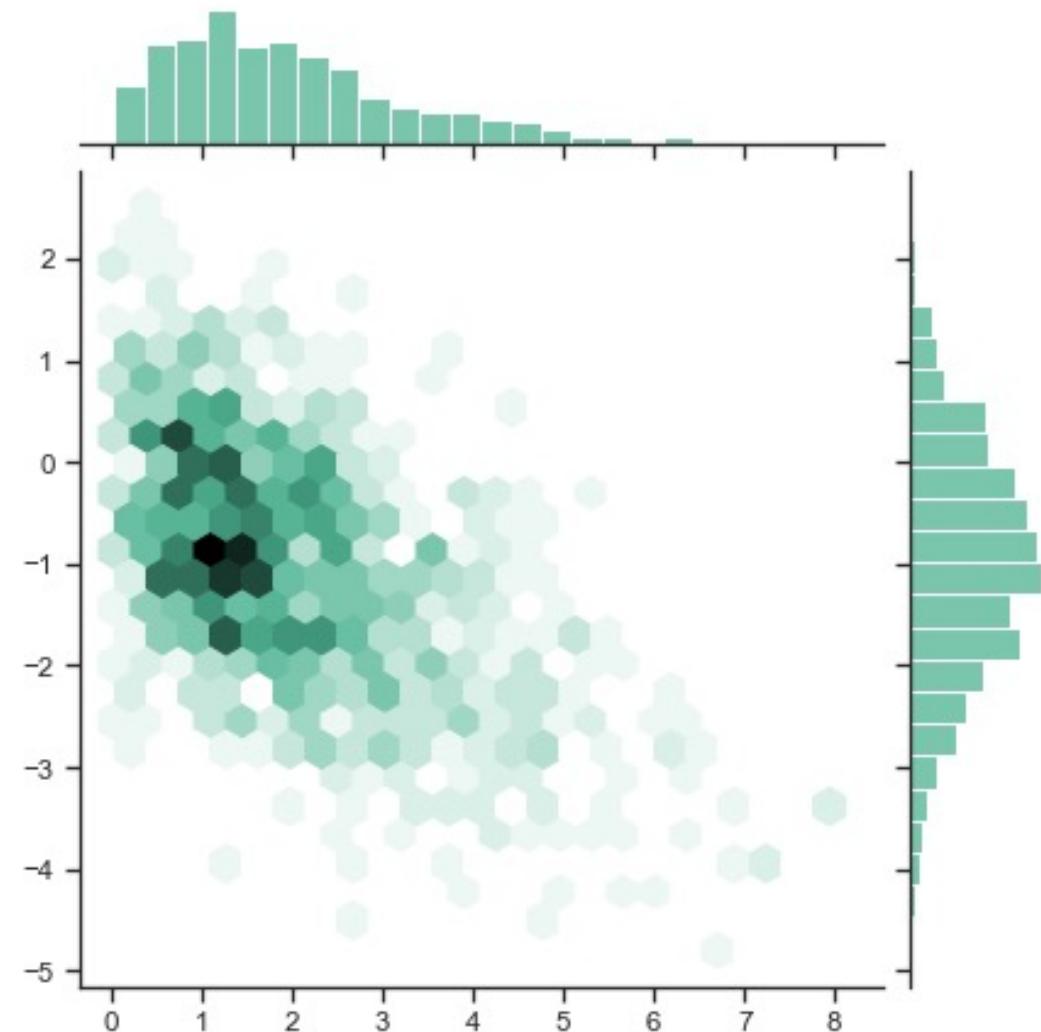
Matplotlib es una biblioteca de graficado para Python y NumPy. Proporciona una API orientada a objetos para incrustar gráficos en aplicaciones que utilizan kits de herramientas GUI de uso general como Tkinter, wxPython, Qt o GTK.



# 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.