

# INTRODUCCIÓN A LA CIENCIA DE DATOS

---



---

# FEATURE ENGINEERING

---

# Definición

*La ingeniería de atributos es la aplicación del conocimiento de un campo (domain knowledge) a la creación de atributos que, al ser alimentados en modelos de machine learning, mejoren el nivel de precisión en las predicciones.*

# Importancia

Algunos proyectos de machine learning tienen éxito mientras que muchos otros no. Qué hace la diferencia? Fácilmente, uno de los factores más importantes son los atributos utilizados. Si tenemos muchos atributos independientes que se correlacionan bien con la clase a predecir el aprendizaje es muy sencillo.

Sin embargo, si la clase es una función muy compleja de los atributos seguramente nuestro modelo no será capaz de aprender. Frecuentemente, los datos de entrada sin procesar no son interpretables por nuestros modelos. En consecuencia, es imprescindible generar atributos que sí lo sean. Esta es la fase más prominente de machine learning.

Seguramente, la ingeniería de atributos es la única tarea donde la creatividad y la intuición son tan importantes como el conocimiento del campo.

- La ingeniería de atributos es un aspecto fundamental en el aprendizaje automático aplicado.
- Es el factor clave para determinar si un proyecto de machine learning tendrá éxito o no.
- Los mejores Kagglers mencionan que dedican sus mayores esfuerzos al diseño de atributos para evitar overfitting.

1. Brainstorming, familiarizarse con los datos.
2. Decidir qué atributos crear y cómo crearlos.
3. Crear los atributos.
4. Checkear/Testear cómo resultan los atributos en tu modelo.
5. Mejorar los atributos si es necesario.
6. Volver al paso 1, seguir probando ideas hasta mejorar el modelo.

---

# FEATURE CREATION

---

# Feature creation

Tres grandes formas de crear atributos:

1. Feature extraction
2. Feature transformation/mapping
3. Feature construction

# Feature extraction

La **extracción de atributos** comienza con un conjunto de valores y construye valores derivados (atributos) que son informativos y no redundantes (independientes o no correlacionados), facilitando el subsecuente aprendizaje y generalización.

En algunos casos, incluso conllevan a una mejor interpretación humana. La extracción de atributos está fuertemente relacionada a la reducción de dimensionalidad.

Por ejemplo, muchos algoritmos de factorización de matrices utilizan en cierta extensión el Análisis de Componentes Principales (PCA según sus siglas en inglés), para expresar inmensas matrices como el producto interno de dos matrices de menor dimensionalidad sin pérdida de información.

# Feature extraction

## Ejemplo relacionado a películas:

Supongamos que tenemos una tabla, como la del práctico 0, cuyas filas representan usuarios de Netflix y las columnas todas las películas que Netflix ofrece. Las celdas de esta tabla se encuentran llenas con los ratings que los usuarios le han asignado a las películas. Esta tabla en sí intenta representar los gustos de los usuarios.

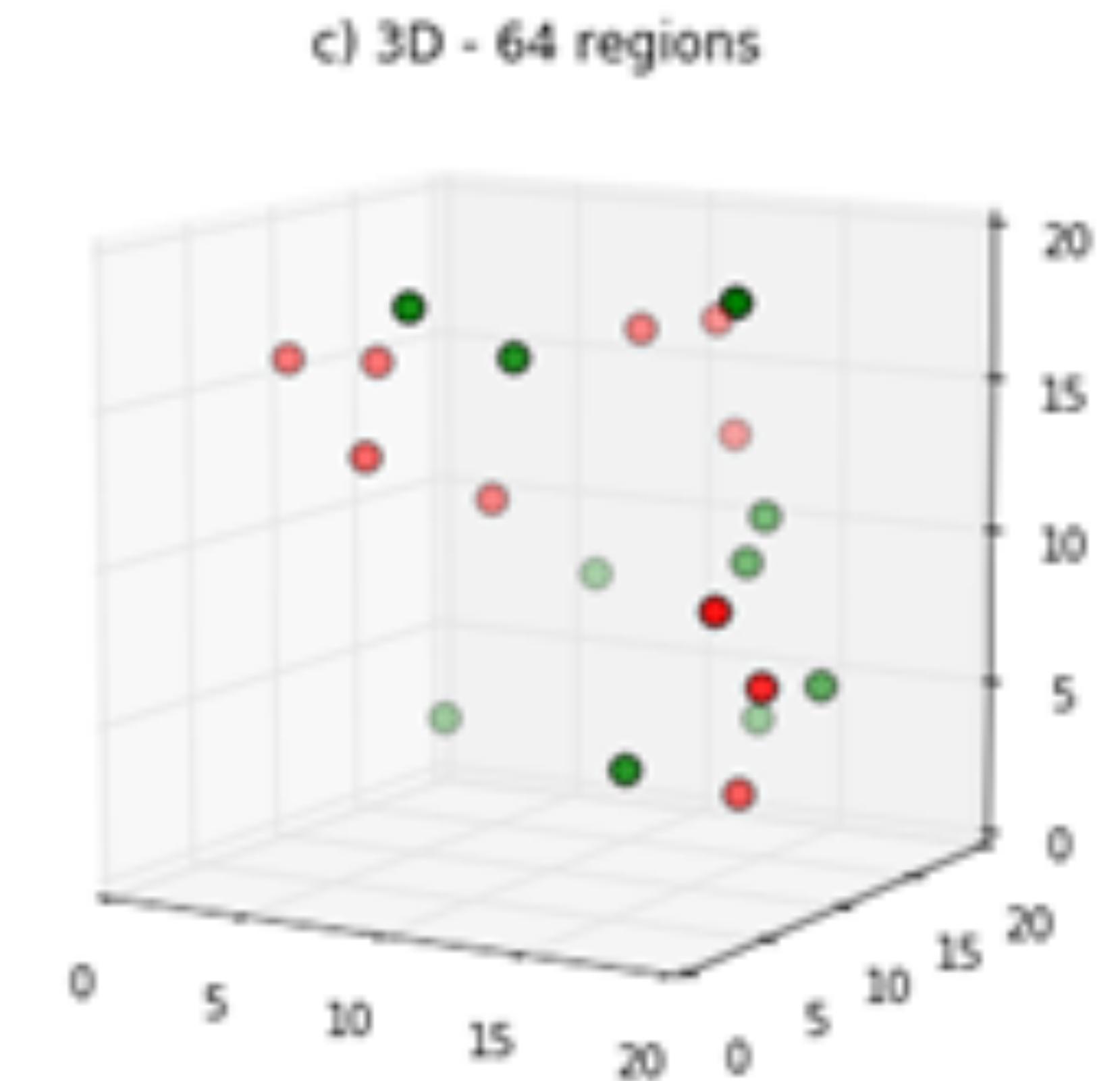
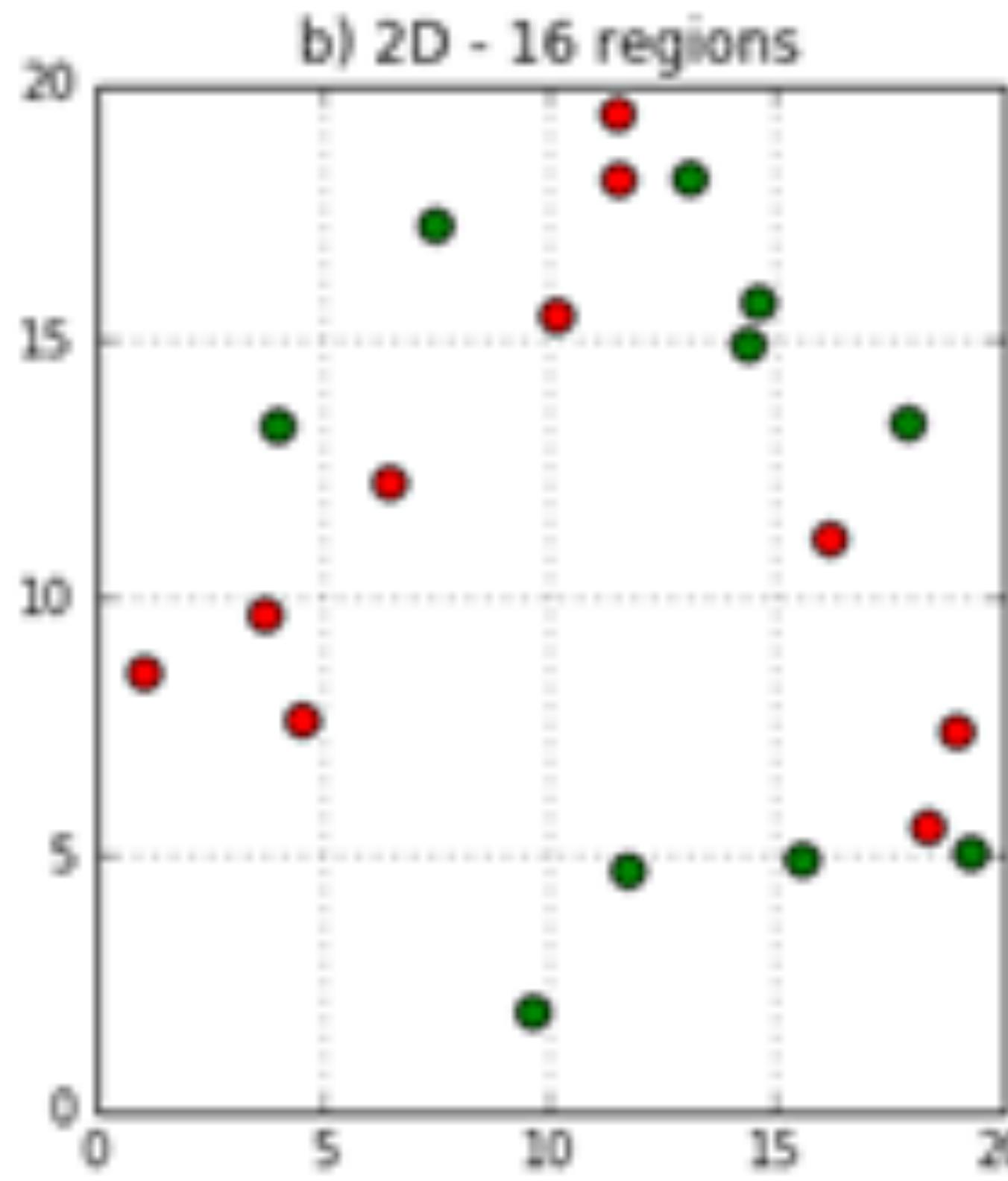
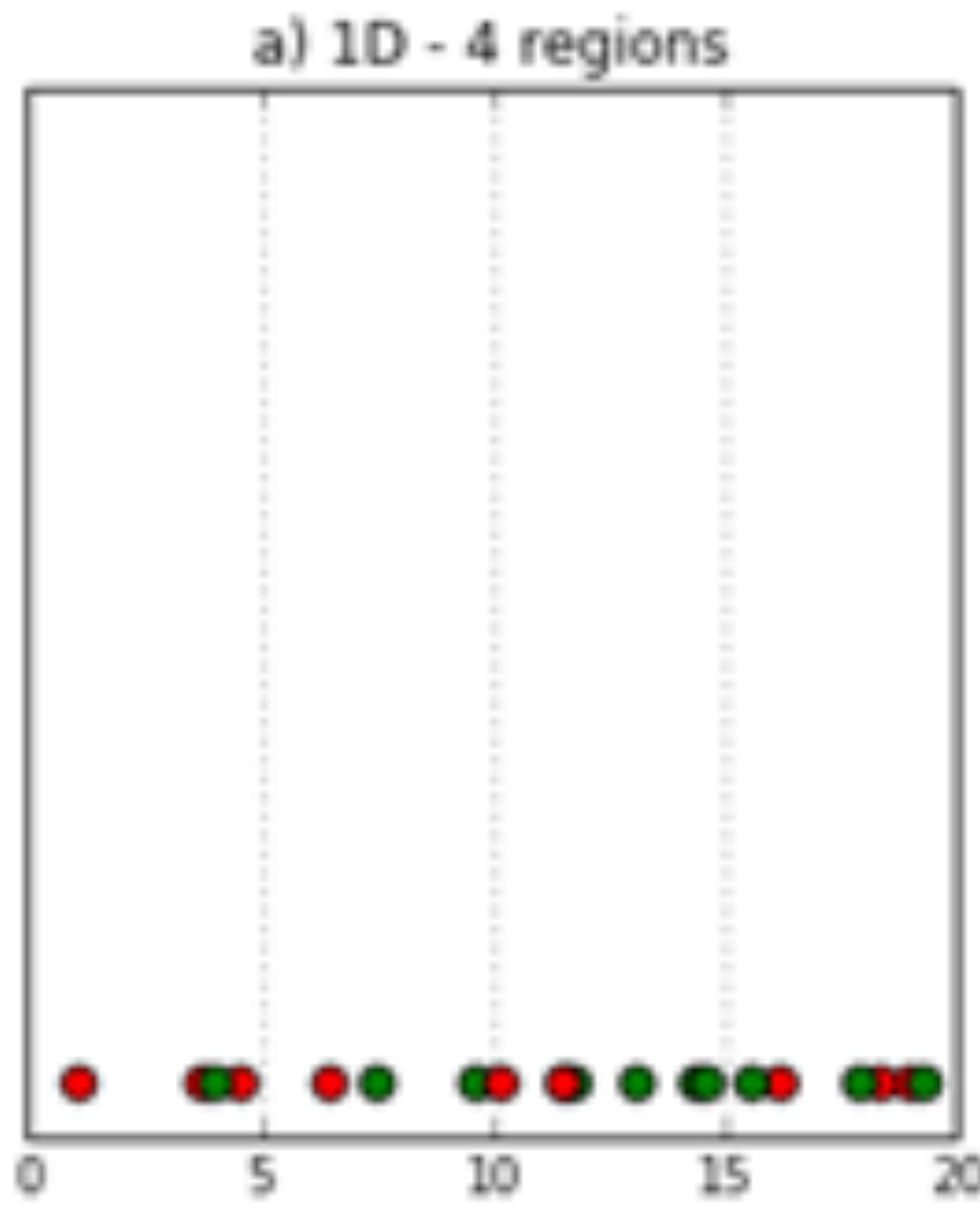
Al existir miles de películas en Netflix en consecuencia terminamos teniendo una representación extremadamente compleja de los gustos de los usuarios (cada columna/película es en sí un gusto). A qué me refiero a que es una representación compleja? Por poner un ejemplo, si observamos que un usuario le ha asignado 5 estrellas a películas como “El señor de los anillos” o “Corazón valiente” no tenemos que ser unos genios para saber que a este usuario quizá le guste una película como “Troya”. Es decir, existe una característica subyacente común a estas películas que no está explícita, como lo puede ser el género “Guerras” o “Batallas con espadas”.

Esta representación compleja acarrea muchos problemas. Por ejemplo, cuantas más películas, mayor complejidad e interpretabilidad de los gustos. Mayor cómputo en las predicciones. Baja densidad de información (*Curse of dimensionality*).

Algoritmos como PCA nos permiten “descubrir” estas características subyacentes (*latent features*). Qué ganamos con esto?

1. Simplificar nuestra representación de la realidad (en el ejemplo, los gustos del usuario).
2. Reducir la dimensión de nuestros datos (mayor eficiencia, menor procesamiento de datos, menor tiempo de cómputo).
3. Toclear el problema de la “Maldición de la dimensionalidad” sin una pérdida de información significativa.

# Curse of dimensionality



R	I1	I2	I3	I4	I5
U1	2	NA	5	2	NA
U2	1	3	NA	1	4
U3	5	NA	3	4	NA
U4	4	NA	4	5	NA
U5	NA	1	NA	NA	4

user item matrix

P	DIM1	DIM2
U1	-0.138	0.421
U2	-0.022	0.589
U3	0.690	0.586
U4	0.280	0.822
U5	0.917	0.842



Q	I1	I2	I3	I4	I5
DIM1	0.489	-0.827	-0.378	-0.108	0.216

item feature

user feature

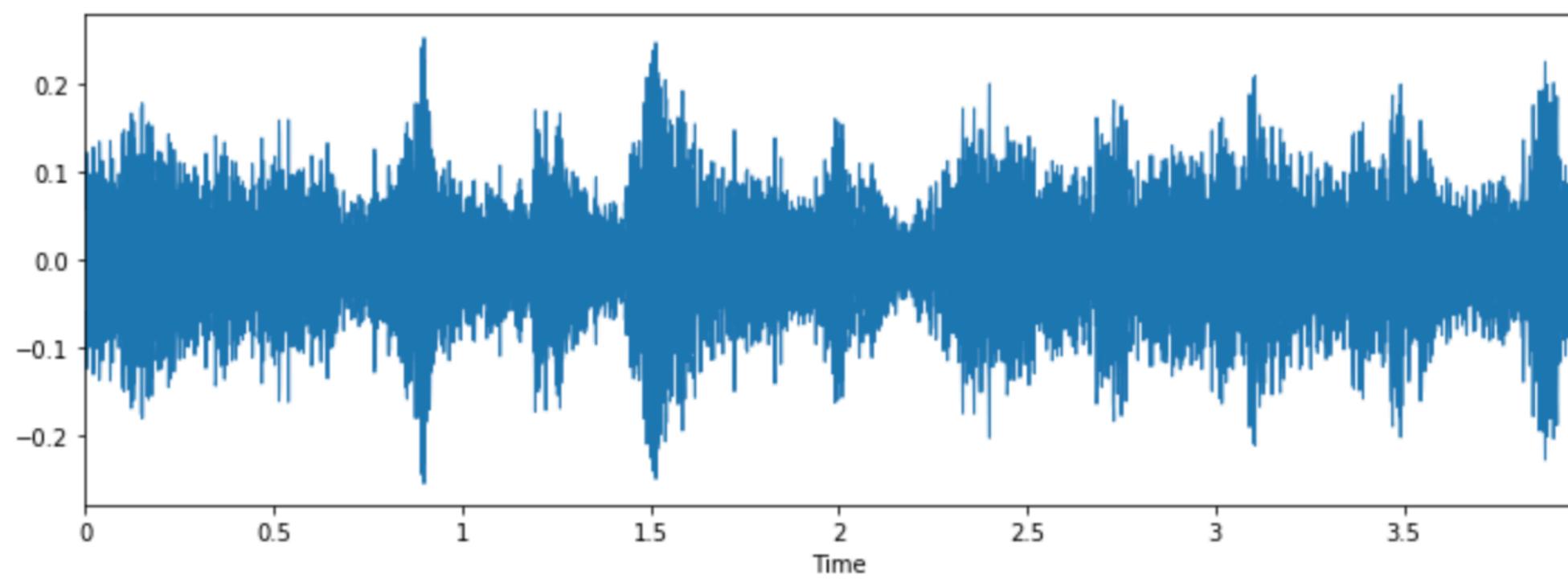
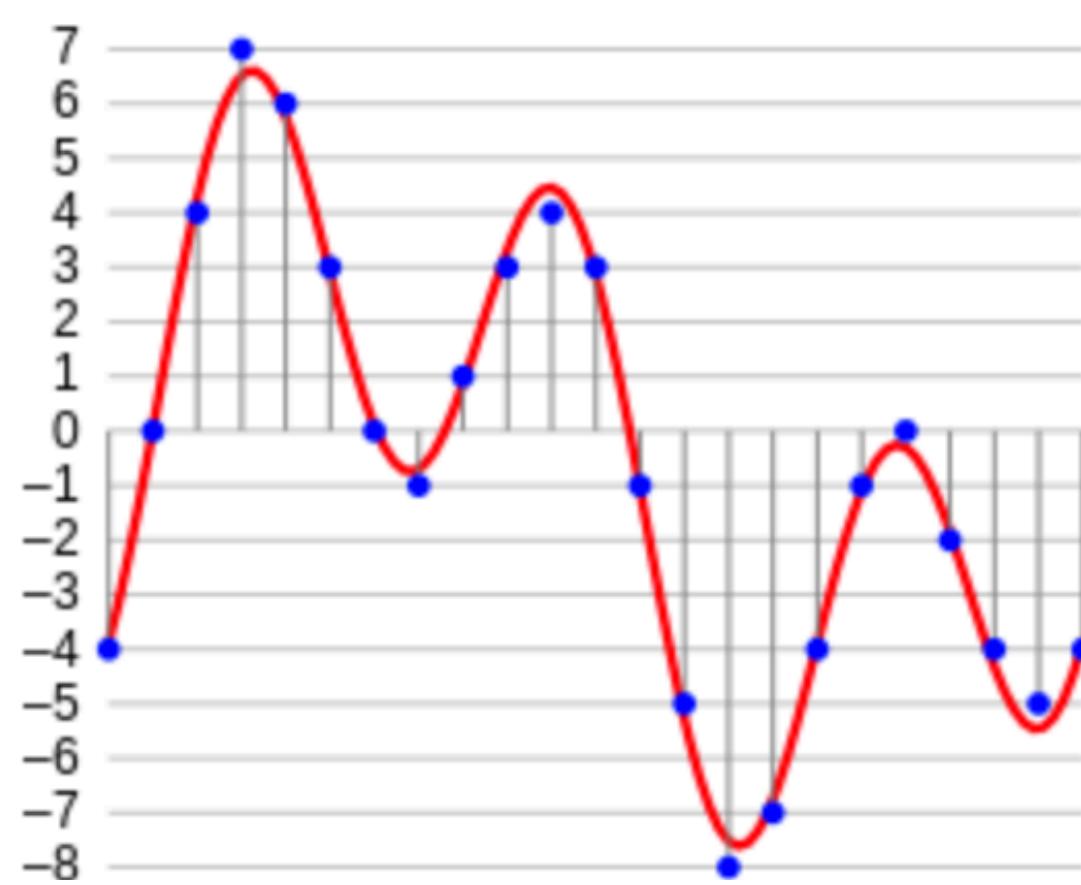
R(Pred)	I1	I2	I3	I4	I5
U1	2	3.009	5	2	3.746
U2	1	3	2.863	1	4
U3	5	2.929	3	4	4.702
U4	4	3.312	4	5	4.903
U5	3.355	1	3	2.794	4

predicted rating

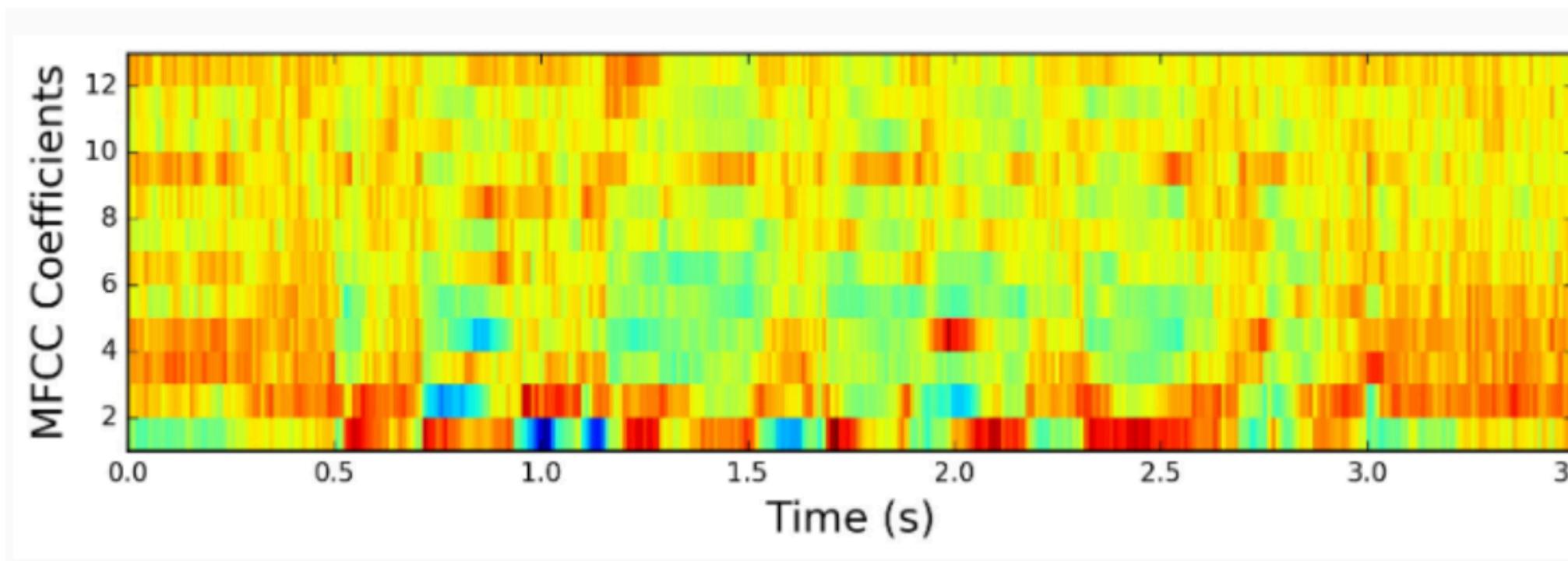
# Feature transformation

La **transformación de atributos** supone el mapeo de un conjunto de valores a un nuevo conjunto de valores por medio de una función o una transformada para hacer la representación de los datos más adaptables o más fáciles de procesar por un algoritmo.

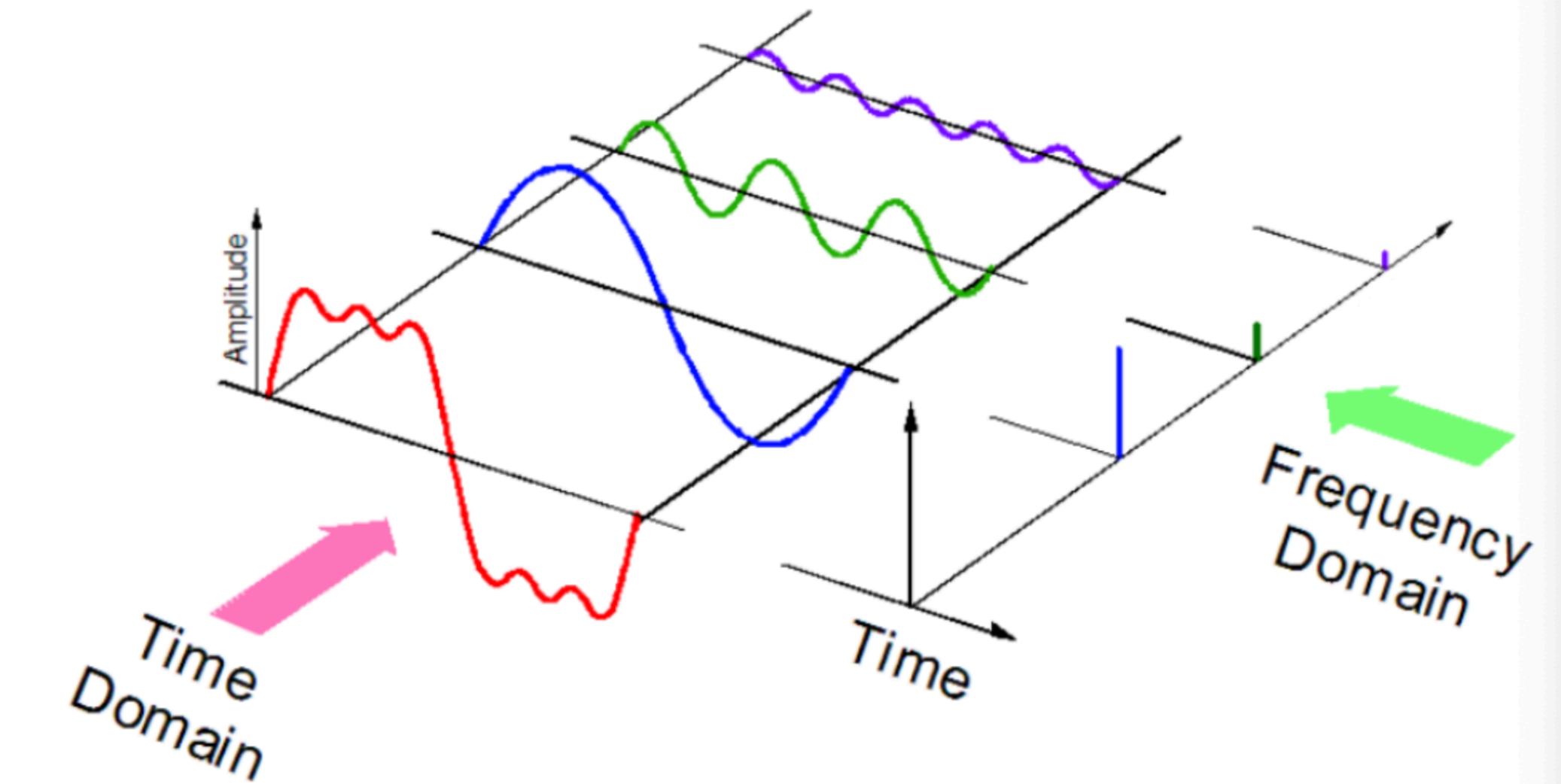
El re-escalamiento de valores es una aplicación frecuente de transformación de atributos. En otros campos de machine learning, como por ejemplo speech recognition, un audio es representado como un conjunto de valores de amplitudes en el tiempo para luego serles aplicado una transformada de Fourier y obtener espectogramas de Mel, los cuales son alimentados en redes neuronales para el reconocimiento del audio.



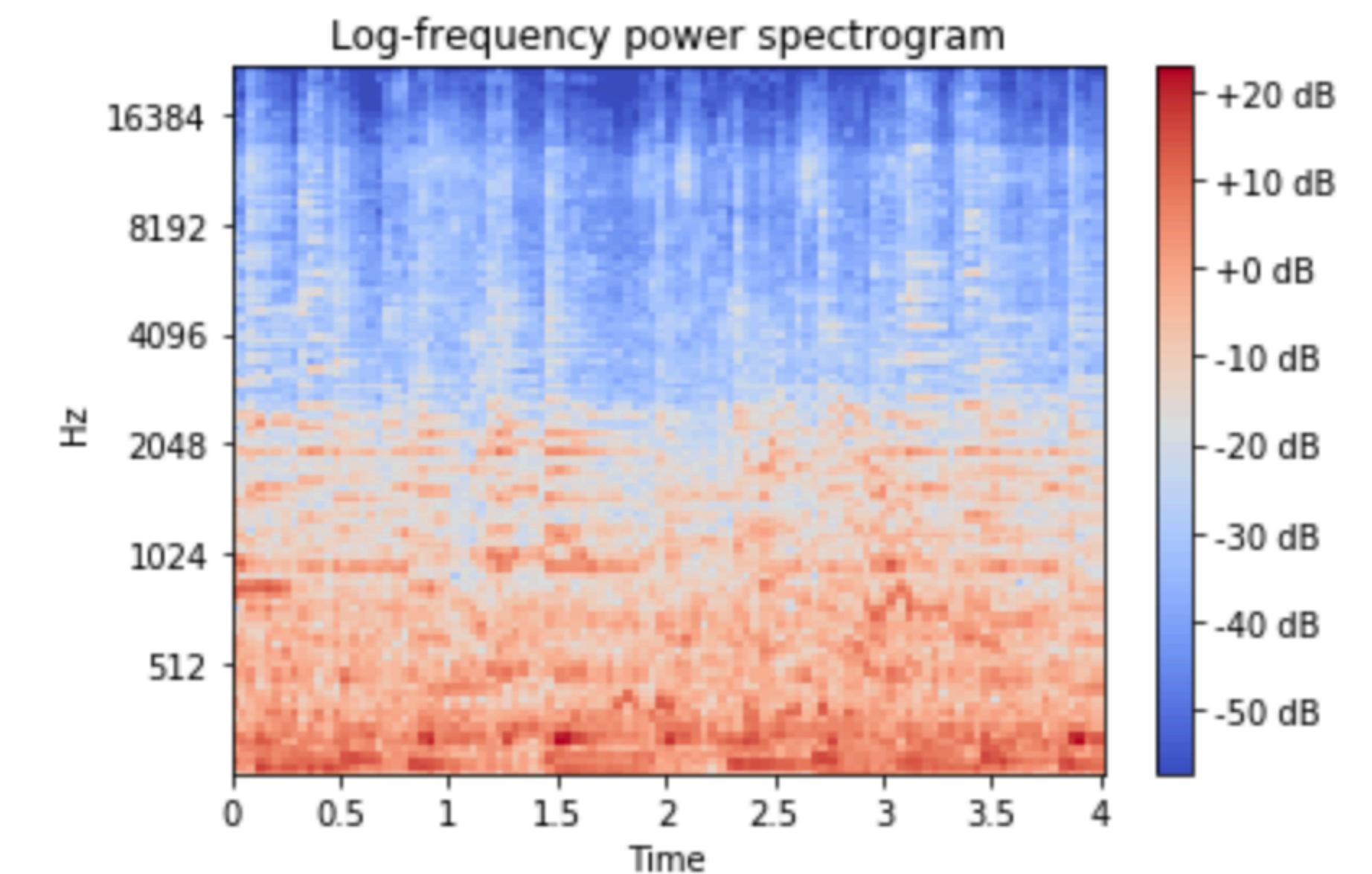
1) Expresar un sonido como un vector de amplitudes en el tiempo



3) Mel Frequency Cepstral Coefficients (MFCCs) en el tiempo



2) Transformada de Fourier cambia del dominio del tiempo al dominio de la frecuencia



4) Log-frequency power spectrogram

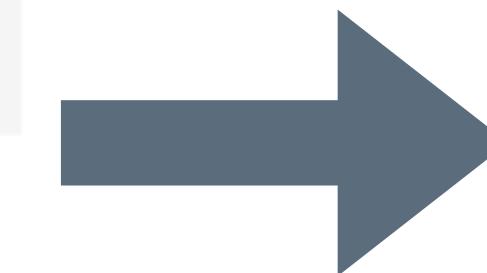
# Feature transformation

El concepto que se quiere ilustrar con este ejemplo, es que existen mecanismos para transformar los datos disponibles con el fin de resaltar las características que diferencian a nuestras instancias o registros. De esta forma, mediante un modelo predictivo de machine learning, podremos predecir con mayor precisión nuestra variable objetivo.

# Feature construction

Los datos no son adecuados para su análisis o para ser ingresados a un modelo predictivo, por lo que nuevos atributos deben ser creados.

ID_alumno	Materia
0	1 Matematica
1	1 Fisica
2	2 Matematica
3	3 Quimica
4	3 Matematica
5	2 Fisica
6	4 Algebra

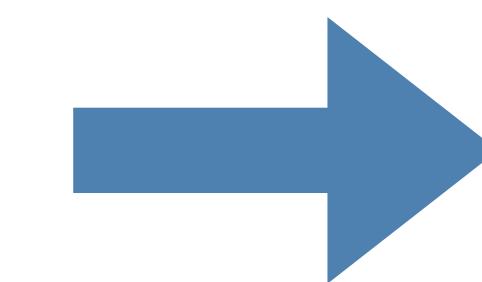


ID_alumno	Materia_Algebra	Materia_Fisica	Materia_Matematica	Materia_Quimica
0	1	0	1	0
1	2	0	1	0
2	3	0	0	1
3	4	1	0	0

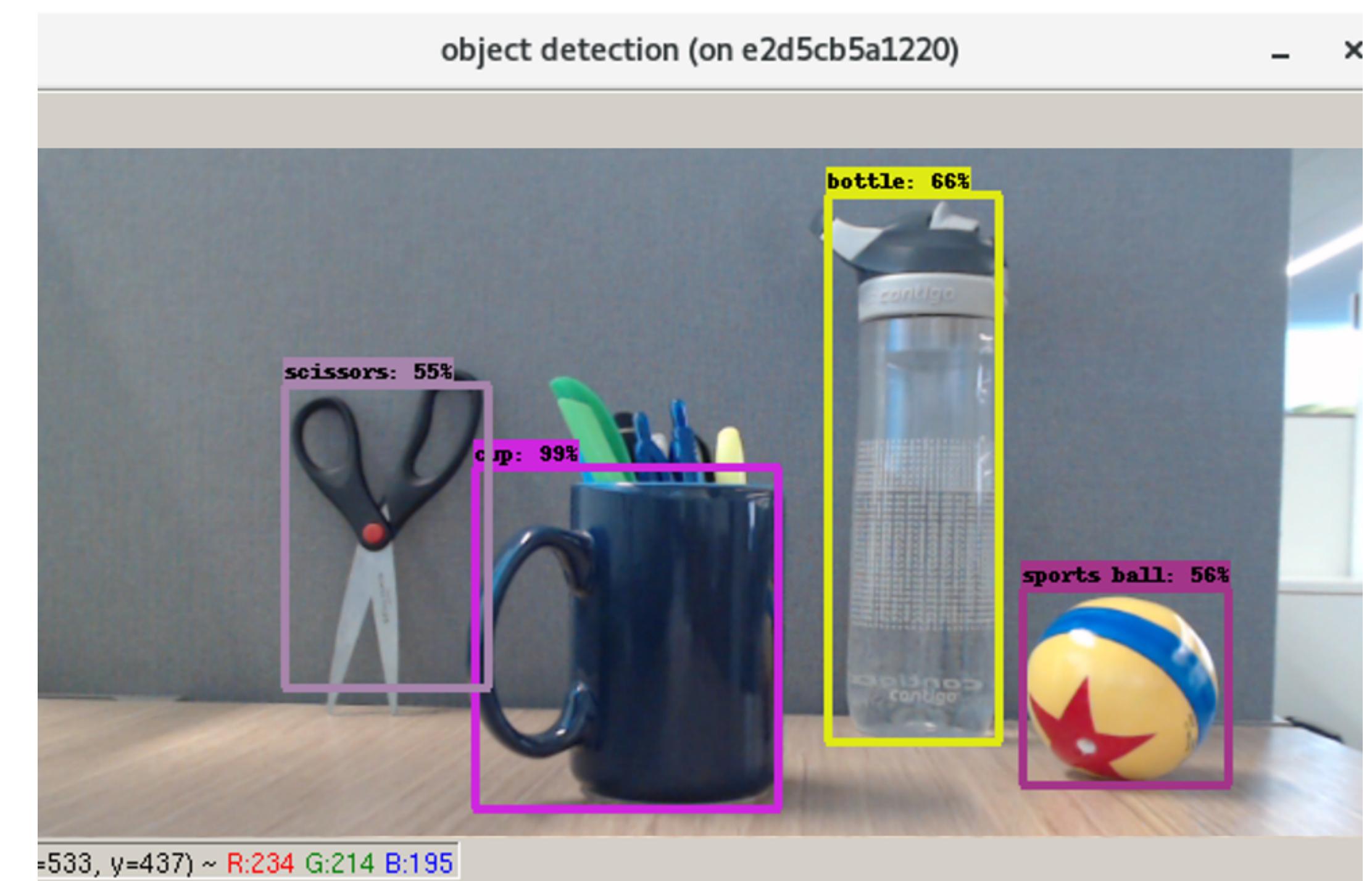
# Feature construction

Un ejemplo de feature construction es la detección de múltiples objetos contenidos en una imagen en problemas de Computer Vision. Frecuentemente, una columna objetivo puede contener listas con los objetos que aparecen en la imagen. Sin embargo, nosotros queremos un vector que contenga 1s en la posición asignada al objeto y 0s si el objeto no aparece. Supongamos que los objetos que queremos predecir son (en orden): [bottle, scissor, spoon, machete, bazooka, cup, ball]

[bottle,  
scissor,  
cup,  
ball]



[1,  
1,  
0,  
0,  
0,  
1,  
1]



# Discretización y binarización

Podemos convertir variables categóricas en variables numéricas o variables binarias para poder operar con ellas o para poder ingresarlas en distintos algoritmos de machine learning.

**Categóricas a numéricas:** si tenemos una variable categórica con N clases, podemos convertirla a una variable numérica discreta de 0 a N-1 valores.

Puede tener sentido esta estrategia cuando tenemos variables categóricas ordinales donde la diferencia numérica o distancia adquieran sentido.

Categorical Value	Integer Value
<i>awful</i>	0
<i>poor</i>	1
<i>OK</i>	2
<i>good</i>	3
<i>great</i>	4

Este método es también conocido como **ordinal encoding**, **label encoding** o **integer encoding**. Este método simplemente remplaza categorías que están expresadas como strings a categorías expresadas mediante números (enteros).

# Discretización y binarización

Podemos convertir variables categóricas en variables numéricas o variables binarias para poder operar con ellas o para poder ingresarlas en distintos algoritmos de machine learning.

**Categóricas a binarias:** si tenemos una variable categórica con N clases, podemos convertirla a variables binarias ( $m = \log_2(N)$ ). Ejemplo: para nuestra variable categórica, tenemos 5 clases:

1. Awful = 0 = 000 =  $0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
2. Poor = 1 = 001 =  $0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
3. OK = 2 = 010 =  $0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
4. Good = 3 = 011 =  $0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
5. Great = 4 = 100 =  $1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

Categorical Value	$x_1$	$x_2$	$x_3$
<i>awful</i>	0	0	0
<i>poor</i>	0	0	1
<i>OK</i>	0	1	0
<i>good</i>	0	1	1
<i>great</i>	1	0	0

# Discretización y binarización

Podemos convertir variables categóricas en variables numéricas o variables binarias para poder operar con ellas o para poder ingresarlas en distintos algoritmos de machine learning.

**Categóricas a binarias asimétricas:** si tenemos una variable categórica con N clases, podemos convertirla en N variables binarias asimétricas.

Puede tener sentido si tenemos variables categóricas nominales. Contra: aumenta la dimensionalidad de nuestro dataset.

Con frecuencia, no se crean nuevas columnas, sino una única con listas de longitud N, llenas de ceros excepto que tienen un 1 en la posición asignada a la clase.

Categorical Value	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
<i>awful</i>	1	0	0	0	0
<i>poor</i>	0	1	0	0	0
<i>OK</i>	0	0	1	0	0
<i>good</i>	0	0	0	1	0
<i>great</i>	0	0	0	0	1

**One-hot encoding:** consiste en codificar cada variable categórica con un conjunto de variables booleanas que adoptan los valores 0 y 1 , indicando si la categoría esta presente en la el registro o no.

# One-hot encoding

De forma general, conviene codificar variables categóricas creando  $k-1$  categorías, donde  $k$  es el número de categorías distintas, pues podemos representar toda la información usando una dimensión menos. Esto evita introducir información redundante.

Por ejemplo, si quisiésemos codificar la variable categórica “Sexo”, no necesitamos crear dos columnas nuevas “Masculino” y “Femenino” sino que basta con usar  $k-1 = 1$  columnas y podemos representar esta categoría. Si tuviésemos una columna “Resultado” que adopta 3 valores distintos “Piedra”, “Papel” o “Tijera” podemos codificarla usando dos columnas “Piedra” y “Tijera” y cuando no es ni “Piedra” ni “Tijera” (0s en las dos columnas) entonces por descarte es “Papel”.

Sin embargo, existen escenarios donde sí queremos hacer encoding para  $k$  variables. Por ejemplo, cuando usamos árboles de decisión, cuando hacemos feature selection o cuando queremos determinar la importancia de cada variable.

```
# k variables
pd.concat([resultado['Resultado'],
           pd.get_dummies(resultado['Resultado'])], axis=1).head()
```

	Resultado	Papel	Piedra	Tijera
0	Piedra	0	1	0
1	Papel	1	0	0
2	Tijera	0	0	1
3	Papel	1	0	0
4	Papel	1	0	0

```
# k-1 variables
pd.concat([resultado['Resultado'],
           pd.get_dummies(resultado['Resultado'], drop_first = True)], axis=1).head()
```

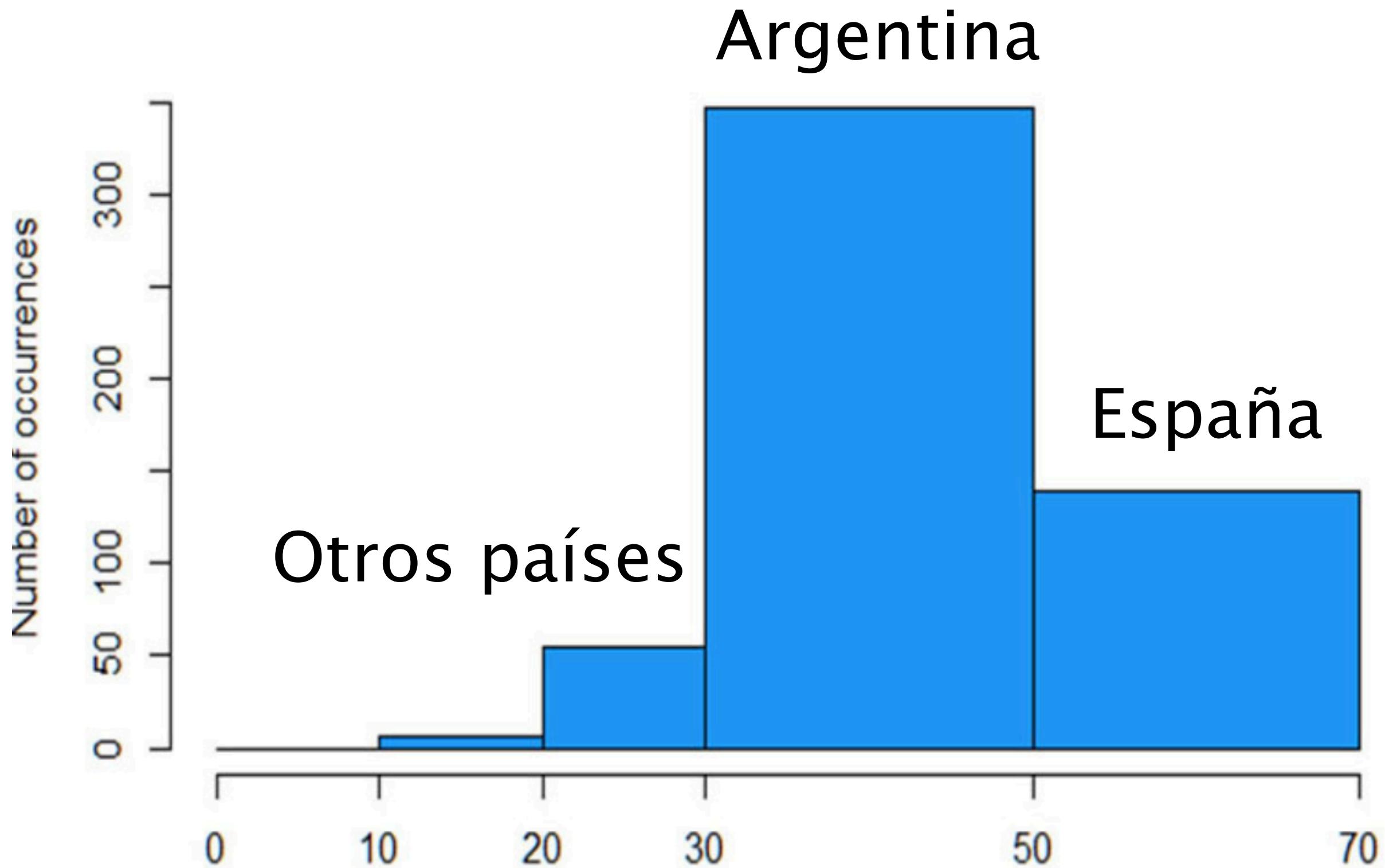
	Resultado	Piedra	Tijera
0	Piedra	1	0
1	Papel	0	0
2	Tijera	0	1
3	Papel	0	0
4	Papel	0	0

# One-hot encoding for top categories

Con frecuencia, las clases de nuestro dataset están desbalanceadas. Esto quiere decir, que las instancias para una o un conjunto de clases aparecen significativamente con mayor frecuencia respecto a otras. Por ejemplo, si evaluáramos los países de procedencia de los inmigrantes uruguayos, seguramente, estos sean “Argentina” y “España” y luego una gran cantidad de países con baja cantidad de inmigrantes.

One-hot encoding for top categories solo codifica las categorías predominantes y las categorías con bajas frecuencias las agrupa en un mismo conjunto. Siguiendo con el ejemplo, solo codificaríamos “Argentina” y “España” y el resto de los países los pondríamos en la misma bolsa.

Este método nos ayuda a no expandir nuestro espacio de dimensiones masivamente. Aunque perdemos, en cierta medida, información.



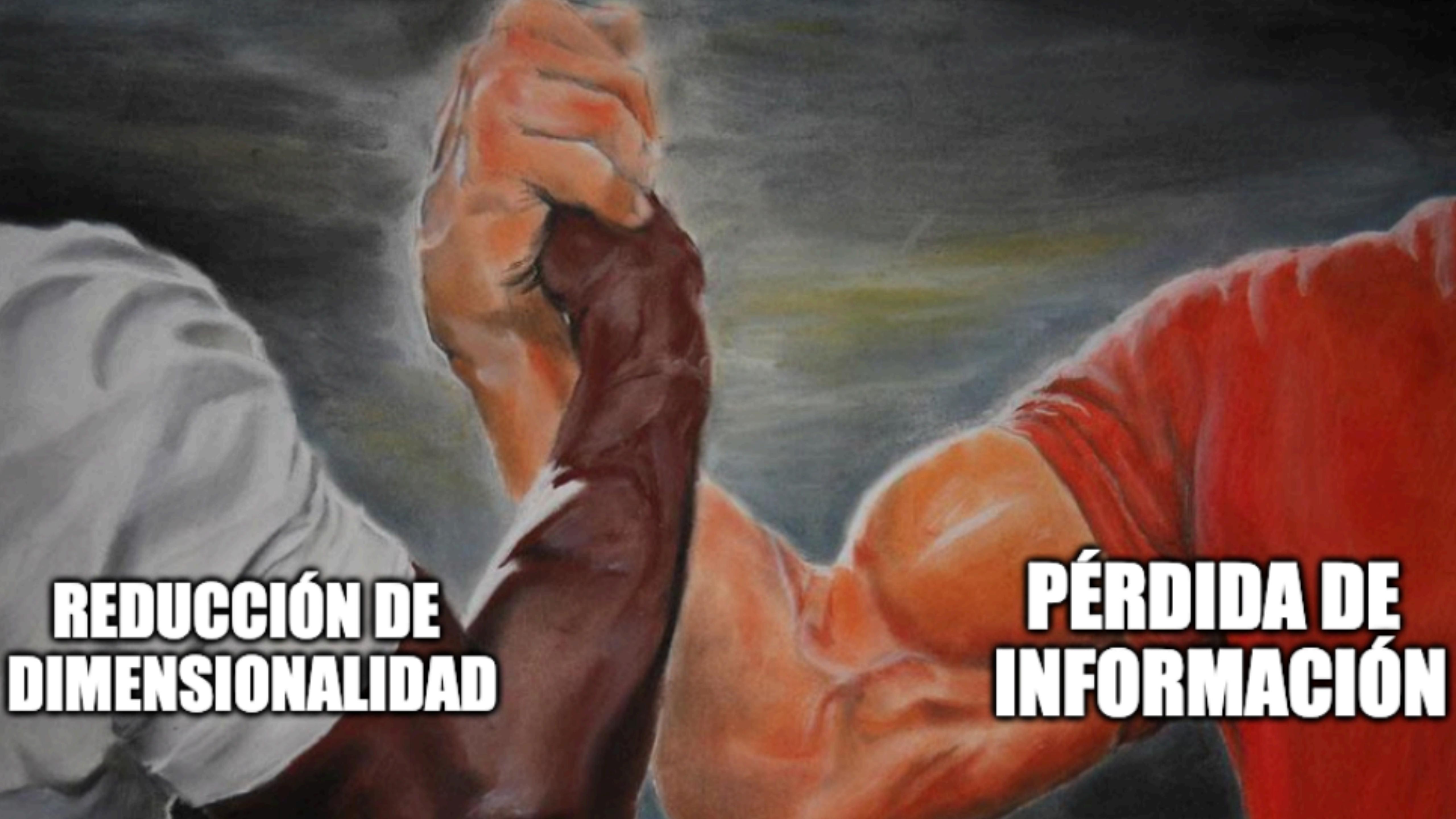
## Data Binning/Bucketing

Podemos convertir atributos numéricos a categóricos mediante una técnica llamada **binning**. Este método de pre-procesamiento nos permite agrupar variables continuas en grupos o bins. Intenta minimizar los errores en observaciones pequeñas, teniendo un efecto suavizante, reduciendo las chances de padecer overfitting.

Generalmente, las dos estrategias de binning son:

1. **Equal width binning:** creamos k categorías mediante intervalos de igual ancho. Si tenemos una variable continua en un rango de valores (A,B) y queremos k bins, entonces el ancho de cada bin es:
  - $W = (B-A)/k$
  - Ejemplo: si quisiésemos hacer bins para una variable "Edad" que adopta valores en el rango (0,100) años y queremos 4 bins, entonces tendríamos bins de longitud  $W = 25$  años. En concreto, estos serían [0,25), [25,50), [50,75), [75,100].
  - Esta técnica, por lo general, se ve afectada negativamente por la presencia de outliers.
2. **Equal frequency binning:** dividimos la variable en k bins tal que cada bin contenga la misma cantidad de registros (misma frecuencia). Generalmente, ya que no es usual obtener k bins con la misma cantidad de registros, los algoritmos de binning con igual frecuencia nos devuelven bins con aproximadamente la misma cantidad de valores para cada bin.

**La pregunta en este método es: qué cantidad de bins seleccionar?**



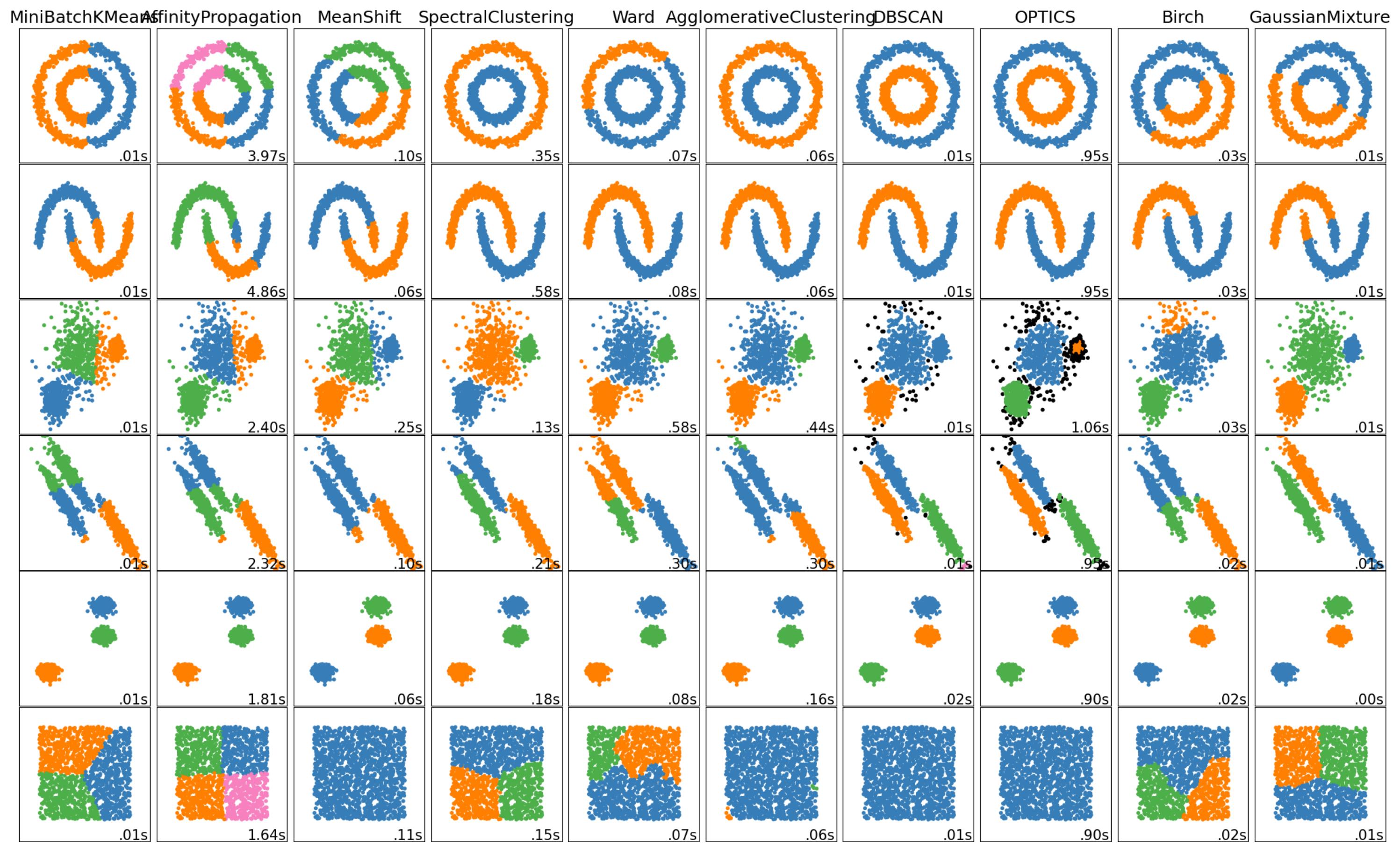
**REDUCCIÓN DE  
DIMENSIONALIDAD**

**PÉRDIDA DE  
INFORMACIÓN**

# Discretización y binarización

## Unsupervised learning methods (clustering)

Podemos discretizar una variable continua mediante algoritmos de clustering de aprendizaje no supervisado. Estos algoritmos encuentran grupos en espacios N dimensionales, devolviendo para cada punto la categoría del grupo al que el algoritmo cree que ese punto pertenece. Los mecanismos para determinar estos grupos son diversos y diferentes para cada algoritmo, aunque existen fuertes similitudes en algunos casos. En la imagen podemos ver algoritmos de clustering populares implementados en **sklearn** y como determinan grupos dependiendo de la nube de puntos que tengamos.



# Discretización y binarización

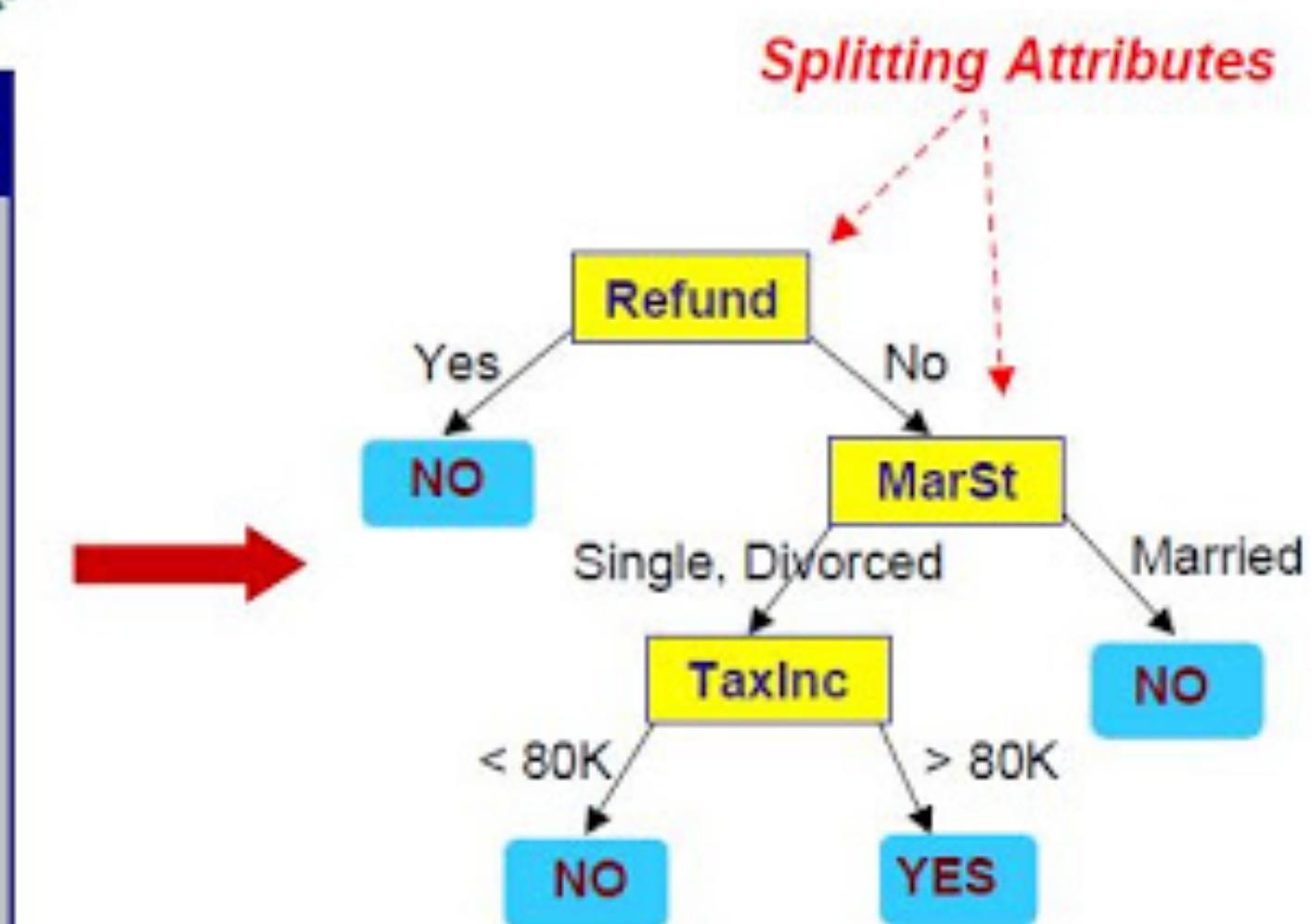
## Supervised learning methods (classification)

Podemos discretizar usando algoritmos de aprendizaje supervisado, usualmente algoritmos de clasificación, como árboles de decisión. Sin embargo, necesitamos tener un dataset de entrenamiento (training data) en el que las clases a determinar sean conocidas para poder entrenar el algoritmo. Luego, al ingresar nuevos registros que no tengan esta clase que queremos predecir simplemente ingresaremos el registro al algoritmo y este nos dirá a qué clase pertenece.

Veamos un ejemplo sencillo. Nuevamente, si tenemos una variable "Edad" que queremos discretizar, podemos tener un dataset de entrenamiento, donde cada registro tenga asociada una clase. Ejemplo: 5 años = "niño", 17 años = "adolescente", 40 años = "adulto", 80 años = "anciano", etc. Podemos entrar un árbol de decisión de manera que, si en el futuro tenemos un registro que solo tenga la edad numérica pero no la clase, este registro sea ingresado al algoritmo y éste se encargue de predecir la clase. El algoritmo ya habrá "visto" tantos ejemplos con clases que habrá aprendido a predecir la clase a partir de la edad numérica.

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

# Normalización (Min Max Scaling)

Existen numerosas definiciones de normalización. Sin embargo, cuando hablamos de datos, normalizar significa re-escalar los valores de una columna al rango [0,1].

El fin de normalizar datos numéricos es llevar distintas columnas que se encuentran en distintos rangos de valores a una escala común.

Frecuentemente, la normalización se aplica para que las columnas que tienen valores numéricos “grandes” no adquieran mayor relevancia por el simple hecho de estar en una escala distinta.

Al tomar medidas, siempre usamos algún tipo de escala (gramos, grados centígrados, metros cúbicos, etc). Sin embargo, las escalas son arbitrarias desde un punto de vista matemático.

# Normalización (Min Max Scaling)

La normalización es un procedimiento necesario cuando tenemos atributos numéricos con valores en distintos rangos. Especialmente, cuando estos rangos difieren en varios órdenes de magnitud. La idea central de la normalización es llevar todos los atributos a un rango común. El hecho de que una variable se encuentre en un rango determinado no debería resultar en que esa variable adquiera mayor importancia respecto a otras.

Ilustremos un posible problema con un ejemplo: supongamos 4 personas, tales que queremos comparar sus físicos a partir de dos atributos, “Altura” y “Peso”. Como sabemos, la altura y el peso de una persona pueden ser medidos en distintas escalas. La altura en metros, centímetros, pies y pulgadas, etc, mientras que el peso en kilogramos, gramos, libras, etc. El uso de una escala o unidad conlleva a que la variable adquiera distintos valores numéricos.

# Normalización (Min Max Scaling)

Consideremos las siguientes 4 personas:

1. Persona 1: [1.80 , 80]
2. Persona 2: [1.80 , 78.7]
3. Persona 3: [0.50 , 80]
4. Persona 4: [0.50 , 70]

Cómo las agruparíamos? La técnica más utilizada es a través del concepto de distancia. Existen numerosas maneras de calcular la distancia entre puntos del espacio. Una de las más populares es la distancia Euclídea. Cuanto más cercanos son los puntos en el espacio, más similares podemos decir que son (o no?).

# Normalización (Min Max Scaling)

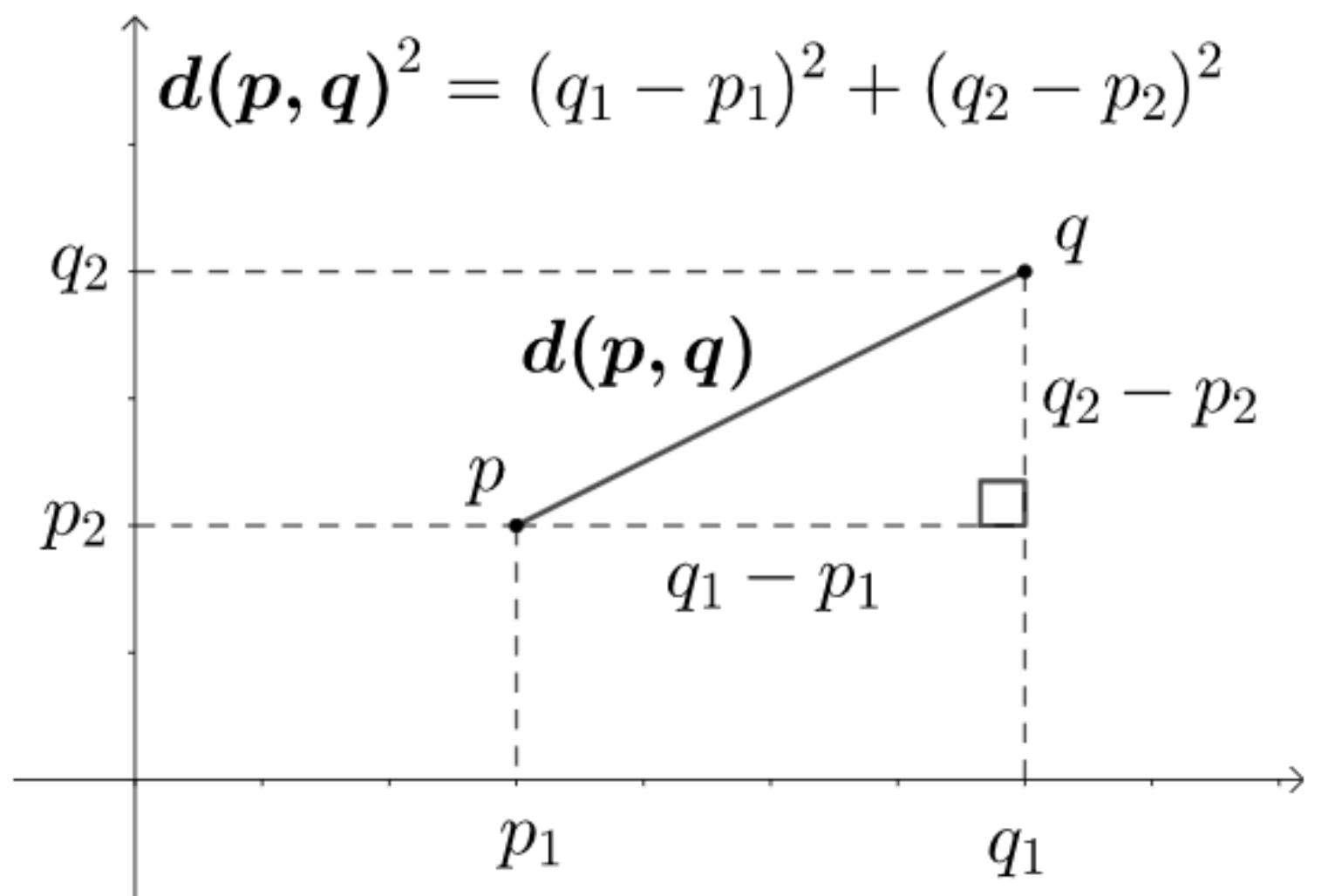
Si tenemos dos puntos P y Q:

- $Q = (q_1, q_2, q_3, \dots, q_n)$
- $P = (p_1, p_2, p_3, \dots, p_n)$

Entonces su distancia euclídea es:

$$\text{Distancia Euclídea} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

Si el ejemplo fuese en dos dimensiones:

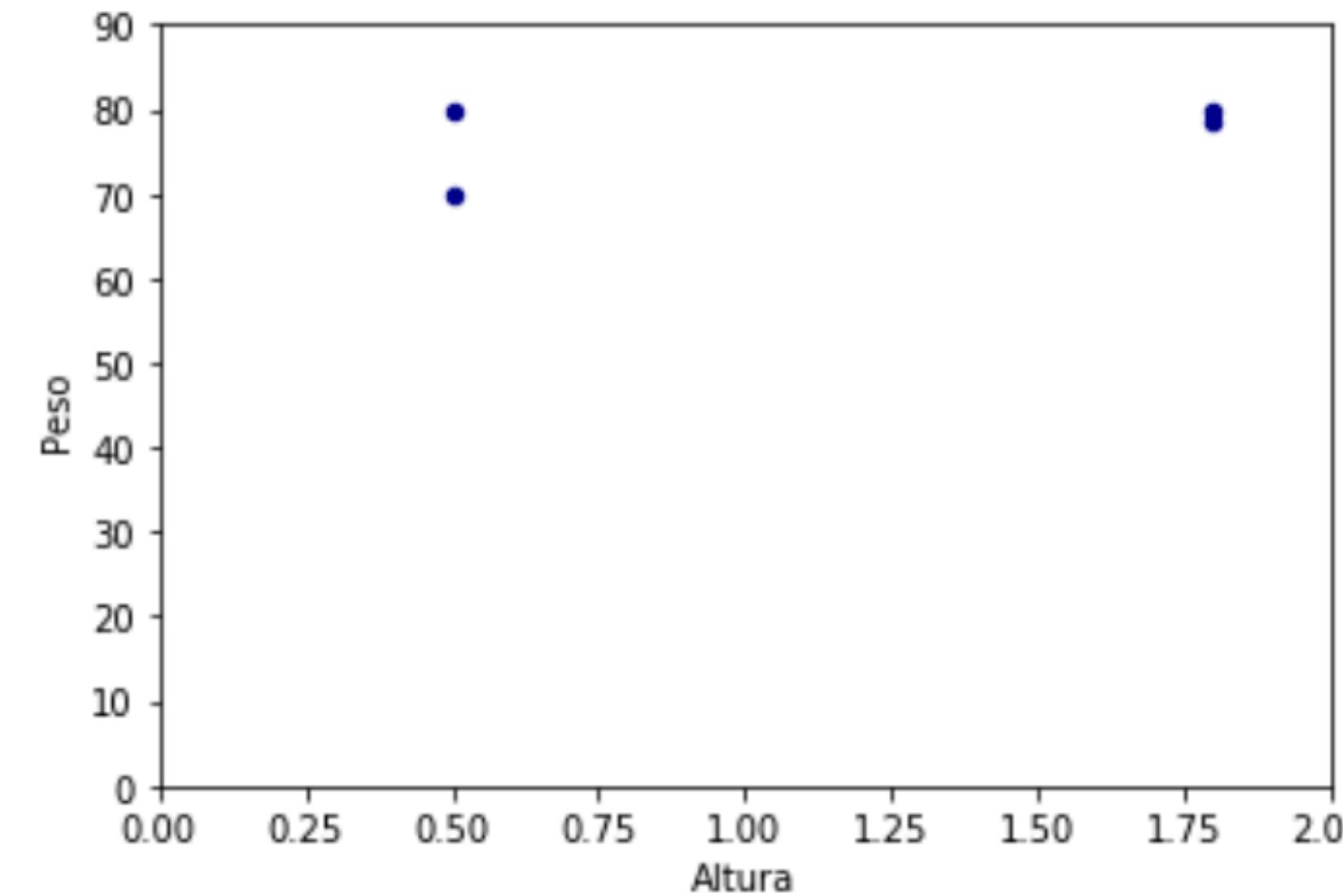


# Normalización (Min Max Scaling)

Dataframe

	<b>Altura</b>	<b>Peso</b>
0	1.8	80.0
1	1.8	78.7
2	0.5	80.0
3	0.5	70.0

Scatter plot



Pair-wise Euclidean distance

	0	1	2	3
0	0.000000	1.300000	1.300000	10.084146
1	1.300000	0.000000	1.838478	8.796590
2	1.300000	1.838478	0.000000	10.000000
3	10.084146	8.796590	10.000000	0.000000

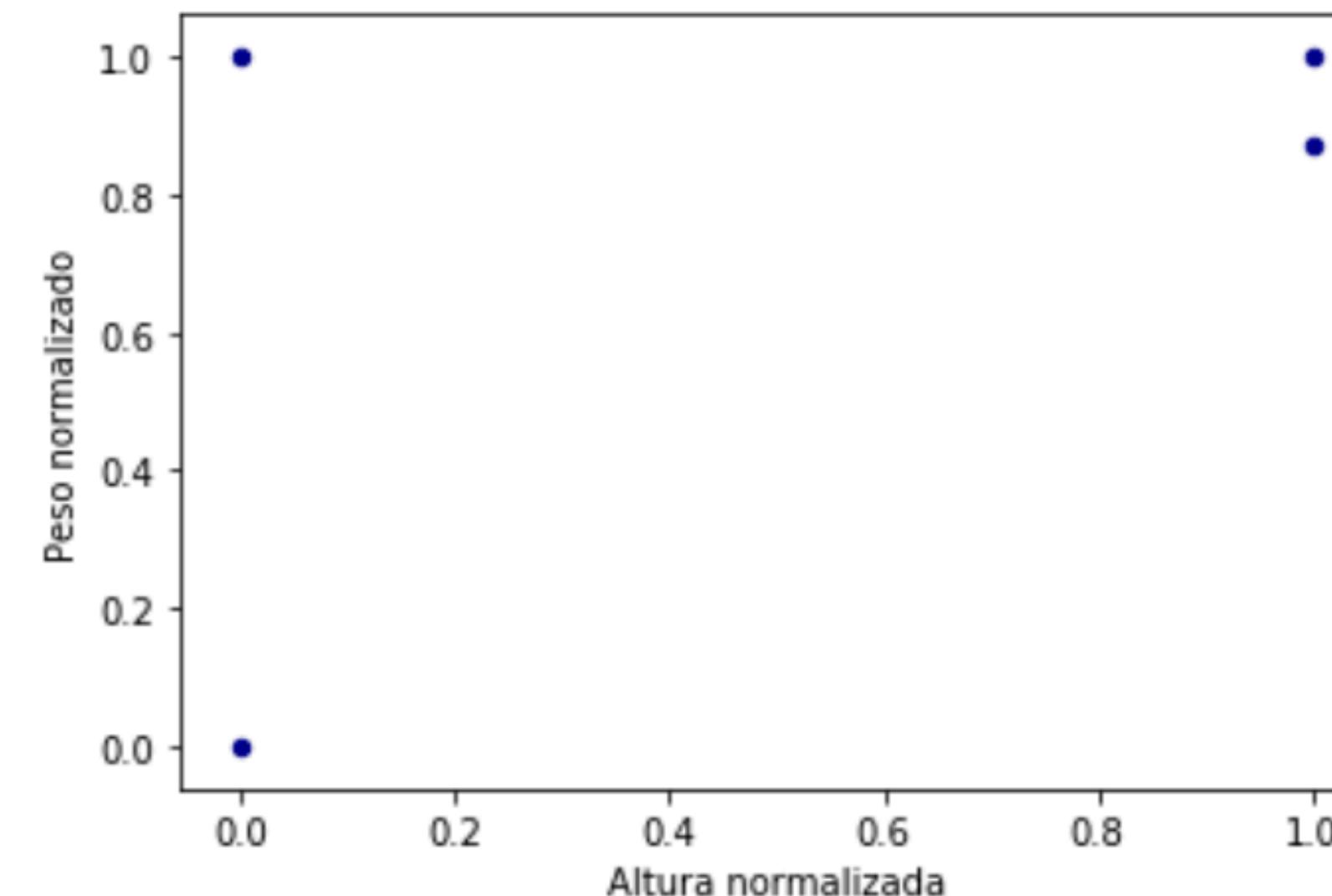
La distancia entre el punto 0 y el punto 1 es la misma que entre el punto 0 y el punto 2!

# Normalización (Min Max Scaling)

Dataframe  
normalizado

	<b>Altura</b>	<b>Peso</b>
0	1.0	1.00
1	1.0	0.87
2	0.0	1.00
3	0.0	0.00

Scatter plot



Pair-wise Euclidean distance

	0	1	2	3
0	0.000000	0.130000	1.000000	1.414214
1	0.130000	0.000000	1.008415	1.325481
2	1.000000	1.008415	0.000000	1.000000
3	1.414214	1.325481	1.000000	0.000000

La distancia entre el punto 0 y el punto 1 es la mínima! Esto es lo que buscábamos.

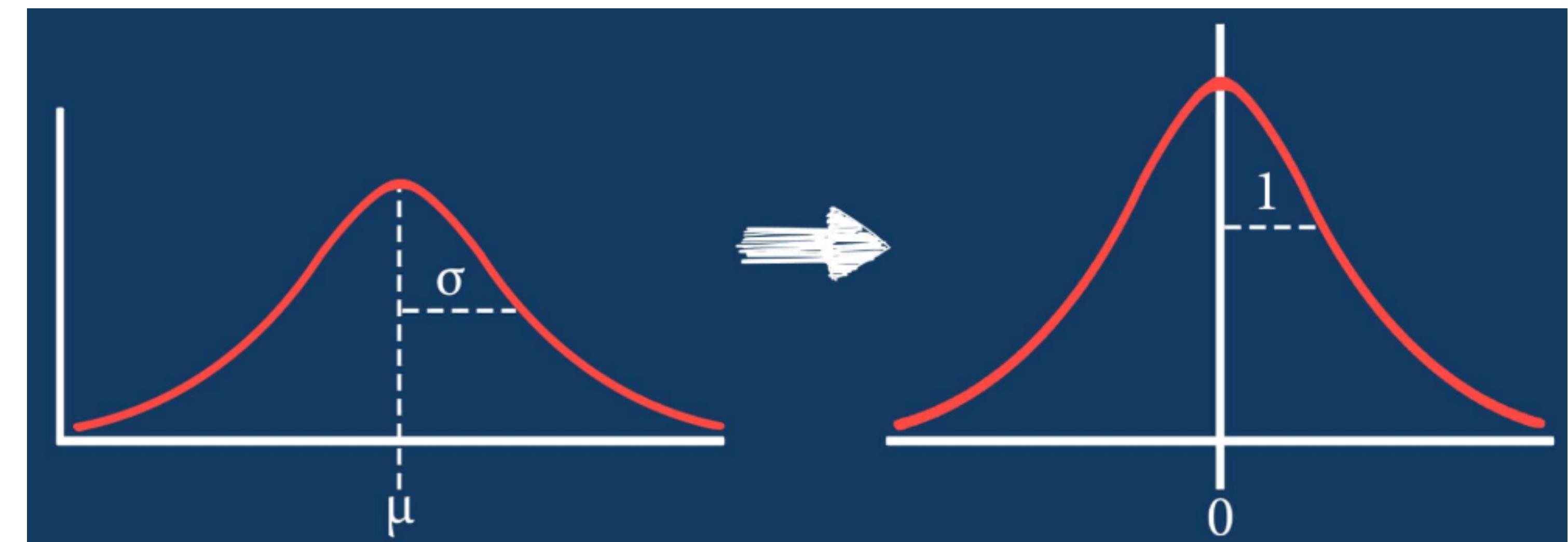
# Estandarización

La estandarización asume que la distribución de los datos es una distribución Gaussiana (campana de Gauss). Este supuesto, a efectos prácticos, casi nunca se cumple, pero esta técnica es más efectiva cuanto la distribución de los datos se asemejen más a una distribución gaussiana.

En concreto, la estandarización de datos consiste en “trasladar” la distribución de tal forma que el promedio de los datos sea 0 y la desviación estándar sea 1. Es la forma de re-escalar los datos que tienen una distribución gaussiana a una escala común sin distorsionar significativamente la distribución de los mismos.

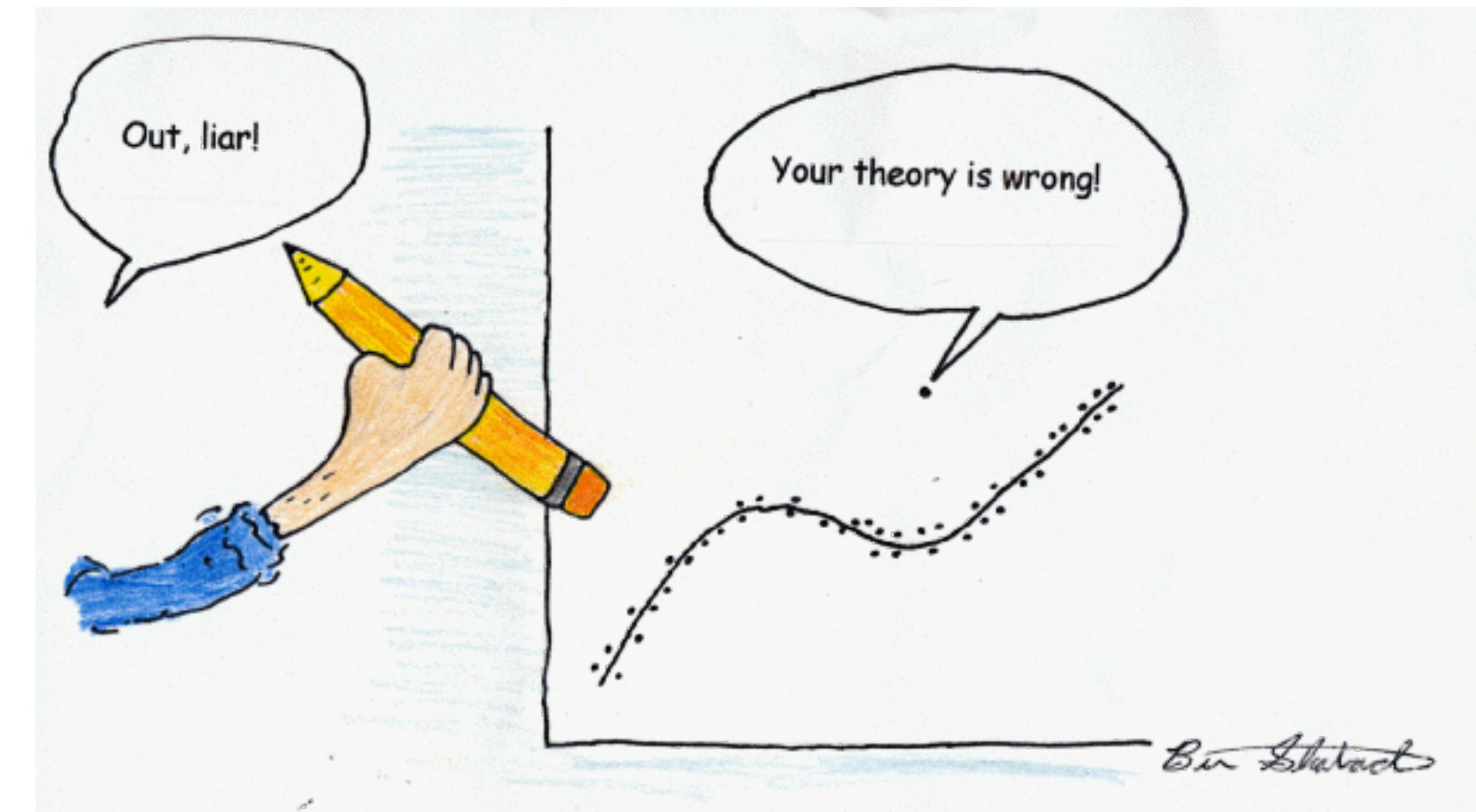
$$z = \frac{x-\mu}{\sigma}$$

- $\mu$  = media o promedio de los datos
- $\sigma$  = desviación estándar de los datos



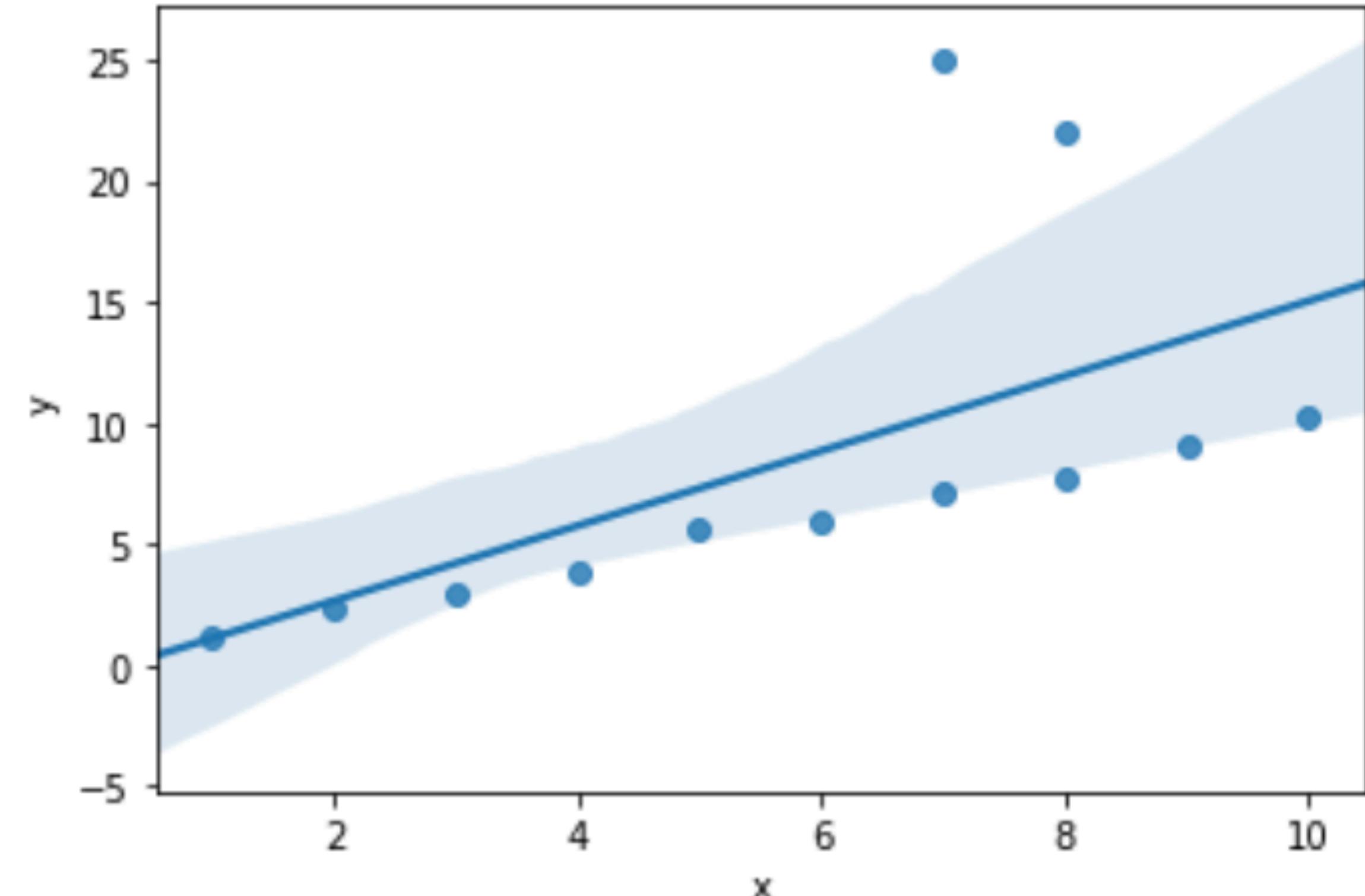
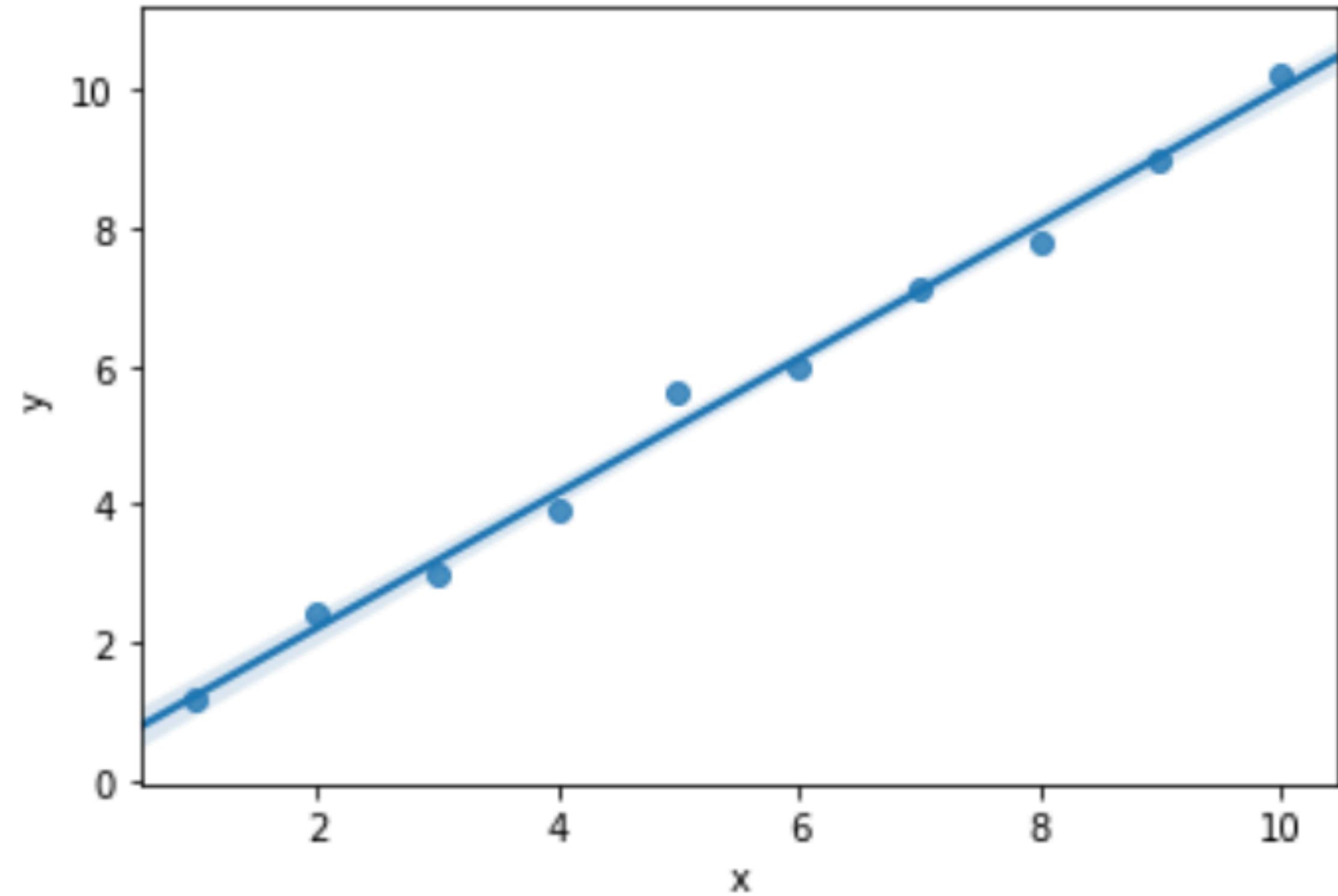
Un outlier es un registro significativamente distinto al resto de los datos.

“Un outlier es una observación que se desvía tanto del resto de las observaciones que genera sospecha de que fue generado a partir de mecanismos diferentes”. – D. Hawkins



Los outliers pueden afectar negativamente nuestros modelos predictivos. En ciertos casos, queremos deshacernos de ellos (ej: una anomalía en un instrumento de medición), en otros casos queremos prestarles particular atención (ej: una compra de gran porte puede ser una transacción fraudulenta). En última instancia, queremos conocer el mecanismo por el cual ese registro fue generado para determinar cuál enfoque es el adecuado para lidiar con el mismo.

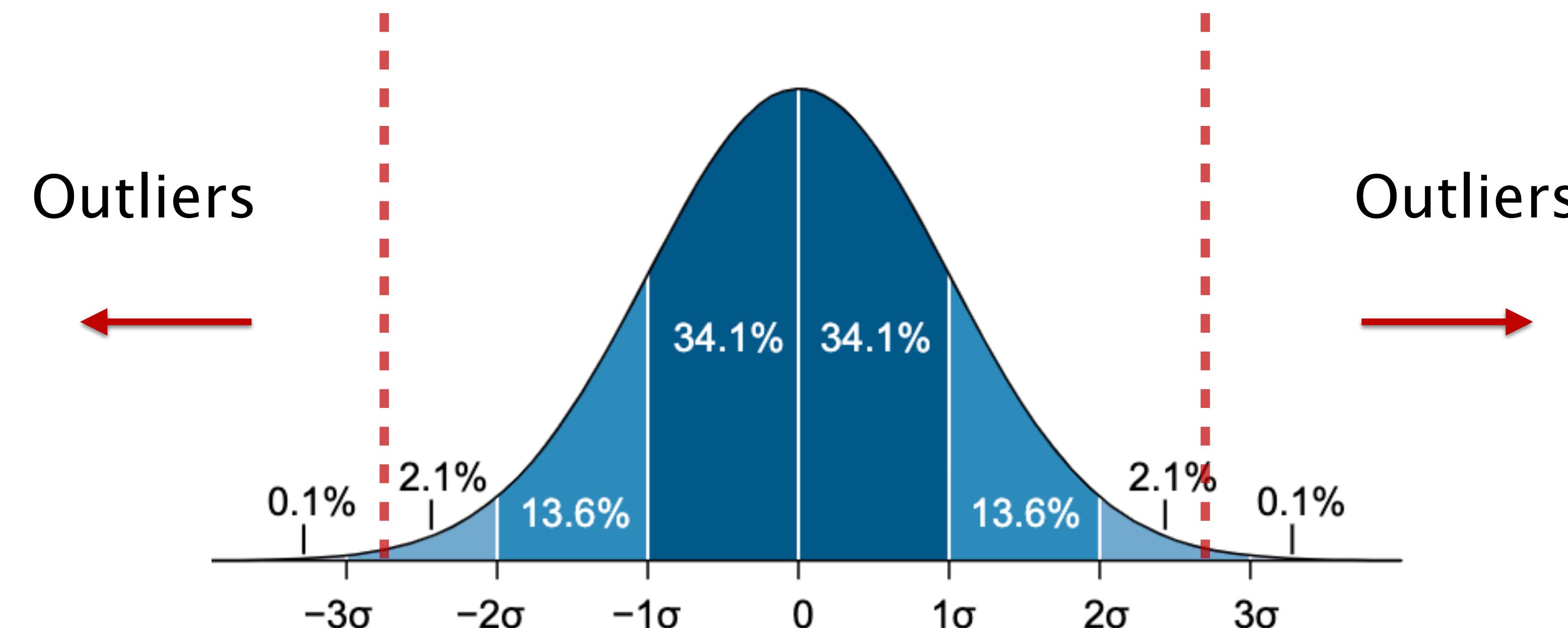
# Outliers



**Datos sin outliers vs. Datos con outliers:** nuestros modelos predictivos se ven afectados negativamente por estas anomalías

# Detección de outliers

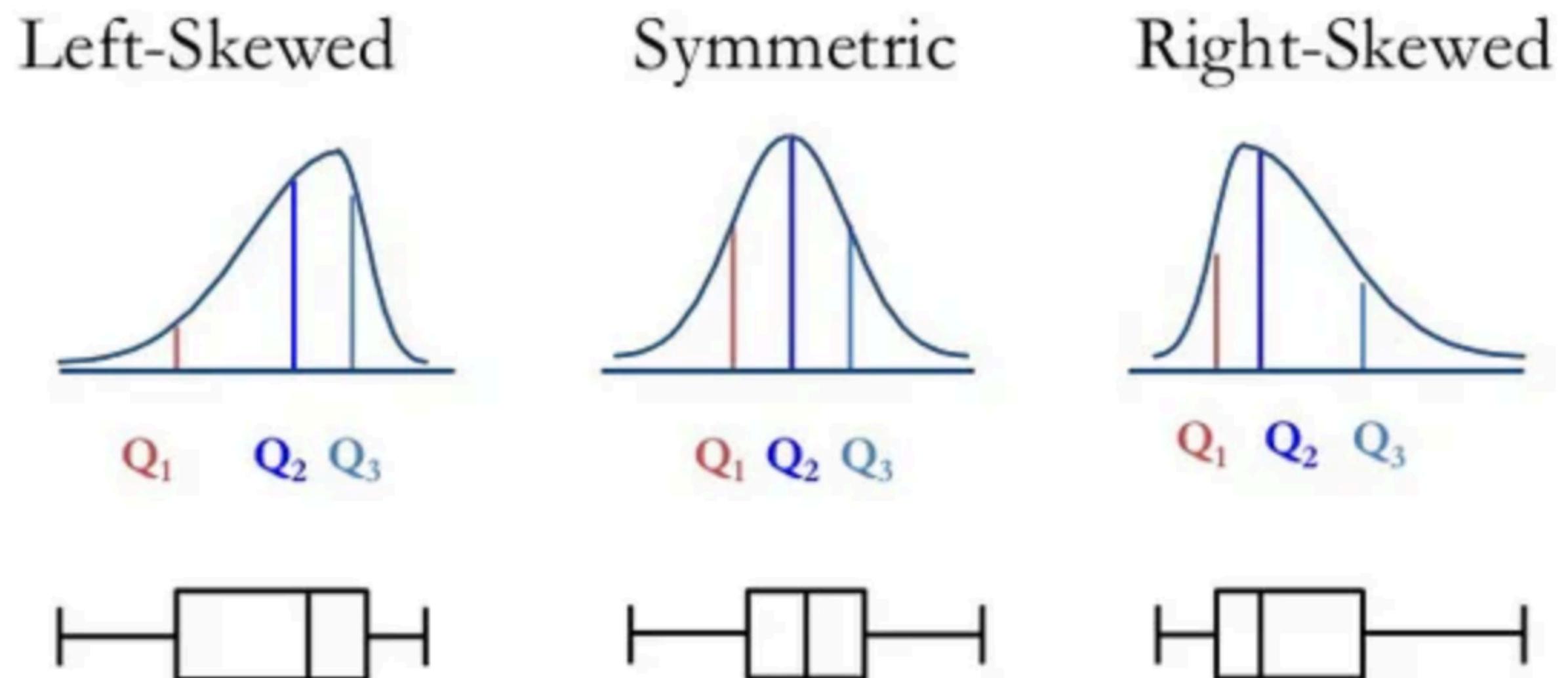
Cuando nuestros datos siguen una distribución normal, la manera de detectar outliers es fijándonos en aquellos datos que se encuentran aproximadamente a una distancia de  $\pm 3$  desviaciones estándar, ya que alrededor del 99% de nuestros valores caen dentro de este rango.



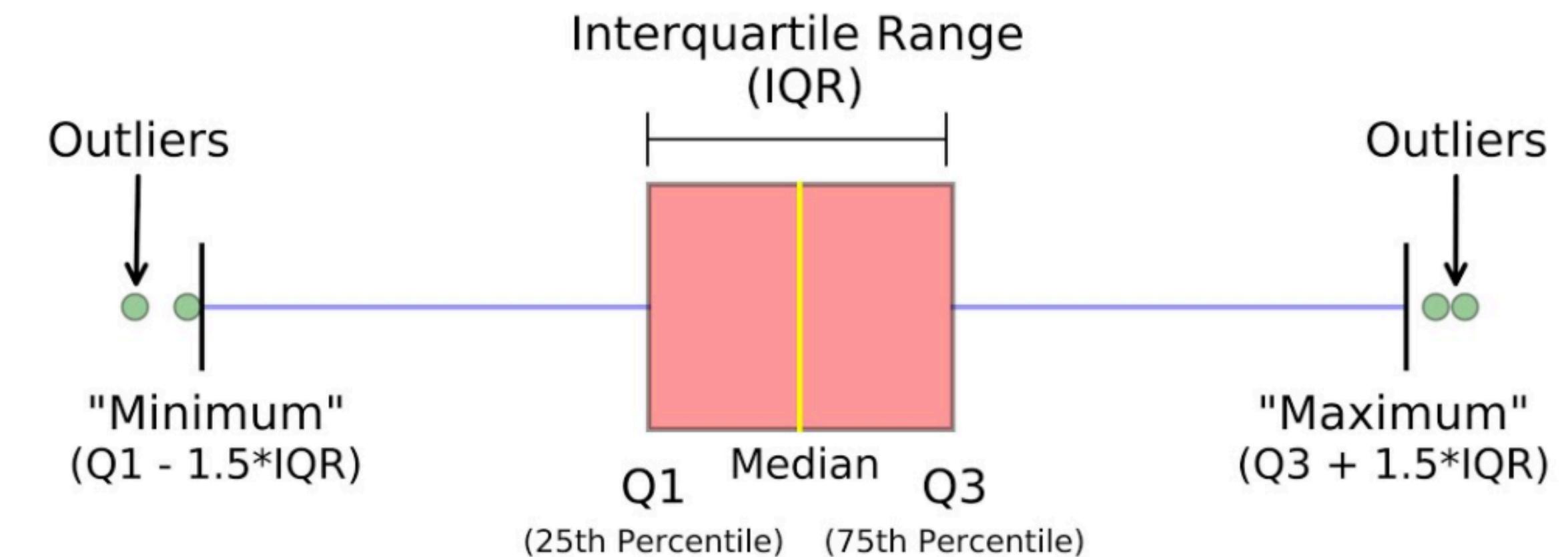
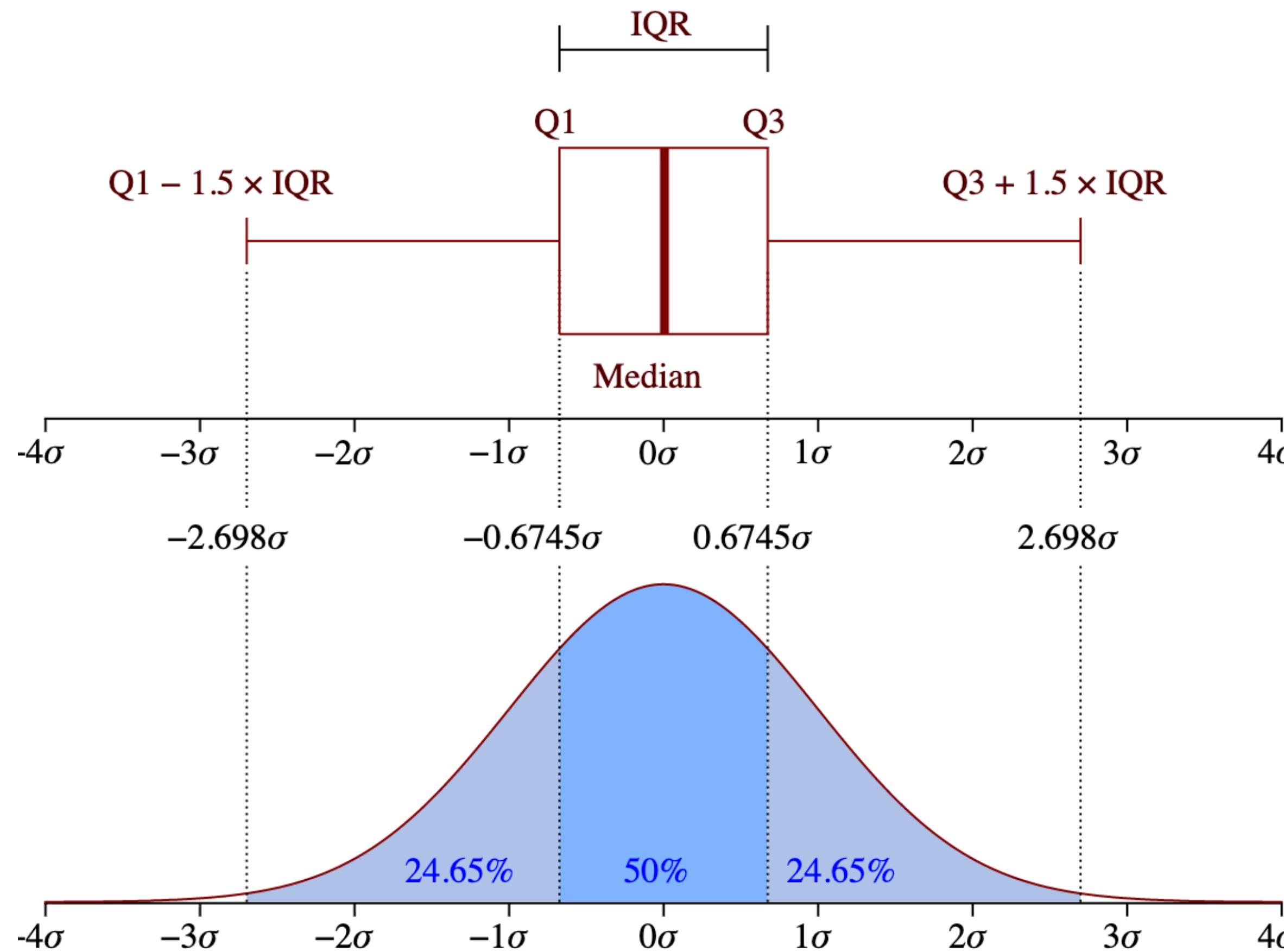
# Detección de outliers

Sin embargo, no siempre nuestros datos seguirán una distribución normal. El enfoque general consiste en determinar los cuartiles de nuestra distribución para luego determinar el rango intercuartil (Inter-Quartile Range). El IQR queda determinado por la resta del cuartil 25 y el cuartil 75.

- $Q_1$  = primer cuartil. Parte el primer 25% de los valores del 75% restante. También conocido como percentil 25.
- $Q_3$  = tercer cuartil. Parte el primer 75% de los valores del 25% restante. También conocido como percentil 75.
- $IQR = 75^{\text{th}} - 25^{\text{th}}$
- **Límite superior** =  $75^{\text{th}} + IQR \times 1.5$
- **Límite inferior** =  $25^{\text{th}} - IQR \times 1.5$



# Detección de outliers



Los outliers son aquellos valores que se encuentran por fuera de los límites superior e inferior.

## Cómo lidiar con outliers?

- **Trimming:** remover los outliers del dataset
- Tratar los outliers como **datos faltantes** y utilizar algún método de imputación de valores.
- **Discretización:** hacer binning y que los outliers entren los bins inferiores o superiores.
- **Censorización:** top/bottom coding. Establecer límites inferiores y superiores en nuestros datos y modificar los outliers para que adopten estos valores.

La imputación de valores faltantes consiste en asignar al valor faltante un estimado del mismo, ya sea estadístico o mediante un algoritmo de machine learning.

El objetivo es poder obtener más registros para poder entrenar un modelo de machine learning.

# Missing values

Usualmente, los datos extraídos son incompletos. Esto puede deberse a múltiples distintos factores, como por ejemplo:

- Inexistencia del dato
- Extravío del dato
- Error en el sistema/instrumento
- Falta de permisos
- Falta de disposición del usuario a dar el dato
- Pereza por parte del usuario

Tipos de datos faltantes:

- 1. Missingness completely at random:** la falta del valor no se debe a variables observadas o no observadas.
- 2. Missingness at random:** la falta del valor se debe a variables observadas.
- 3. Missingness that depend on unobserved predictors:** la falta del valor se debe a variables no observadas.
- 4. Missingness that depend on the missing value itself:** la falta del valor se debe a la variable misma.

Tipos de imputación de valores faltantes:

- Imputación de la media/moda
- Imputación aleatoria (mediante alguna distribución estadística)
- Imputación del fin de la distribución
- Imputación mediante regresiones lineales
- Hot-deck imputation
- Imputación mediante regresiones iterativas
- Métodos no supervisados (clustering)
- Métodos supervisados (regresiones y clasificadores)

Finalmente, cabe destacar que existen técnicas específicas de feature engineering para cada área de la inteligencia artificial, como por ejemplo Computer Vision, Natural Language Processing, Speech recognition, etc. Aunque las técnicas que hemos visto también hasta cierta extensión aplican a las áreas antes mencionadas, éstas técnicas y métodos están relacionados principalmente a datos estructurados.