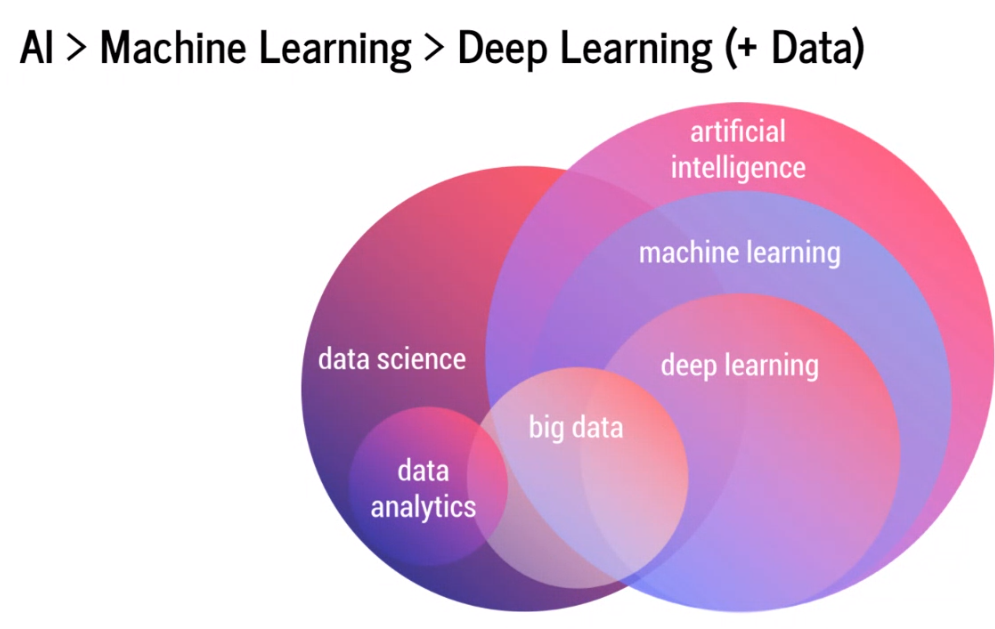
**Inteligencia Artificial**

#### **Artificial Intelligence**: Se enseña a las computadoras a tomar decisiones como los humanos, utilizando la percepción, la lógica y el aprendizaje. Desde machine learning, a deep learning o cualquier decisión.

#### **Machine Learning**: Se aplican **algoritmos matemáticos** a los datos para permitir que las computadoras **aprendan de datos anteriores** **(patrones)** para **hacer predicciones o decisiones** cuando se les muestran nuevos datos

#### **Deep Learning**: Es un tipo de **aprendizaje automático** que utiliza **redes neuronales** (estructura algorítmica) de muchas capas



* **modelo:** representación (simplificación) de una realidad, sobre la que nosotros necesitaremos recopilar información.
* Es diferente modelo de datos (lo que hemos estado utilizando siempre) a modelo de machine learning

\* **Inferir** es interpretar o explicar un fenómeno en base a una o varias observaciones. Una buena inferencia debe ser apoyada o comprobada con nuevas observaciones (si se calienta el agua, se evapora). De lo contrario, se convierte solo en una suposición o adivinanza. **Predecir:** Predecir es anunciar con anticipación la realización de un fenómeno (mañana se evaporará el agua).

#### **Machine Learning**

Los modelos de Machine Learning no implican causalidad.

Machine Learning ocurre en dos pasos:

1. TRAINING: es el proceso por el cual **el algoritmo aprende los patrones** en los datos históricos (llamados conjunto de entrenamiento). Está formado por un conjunto de **parámetros del algoritmo** que se ajustan para adaptarse mejor a esos datos en particular.

→ RESULTADO: Un **modelo entrenado**.

1. PREDICTION: es cuando este algoritmo entrenado **se expone a nuevos datos (estimador)** y se le pide que **prediga el resultado (estimación)**.

**3 tipos diferentes de algoritmos de aprendizaje**:

1. **APRENDIZAJE SUPERVISADO**: el algoritmo es entrenado con dataset en el que se le dice el resultado de cada input.

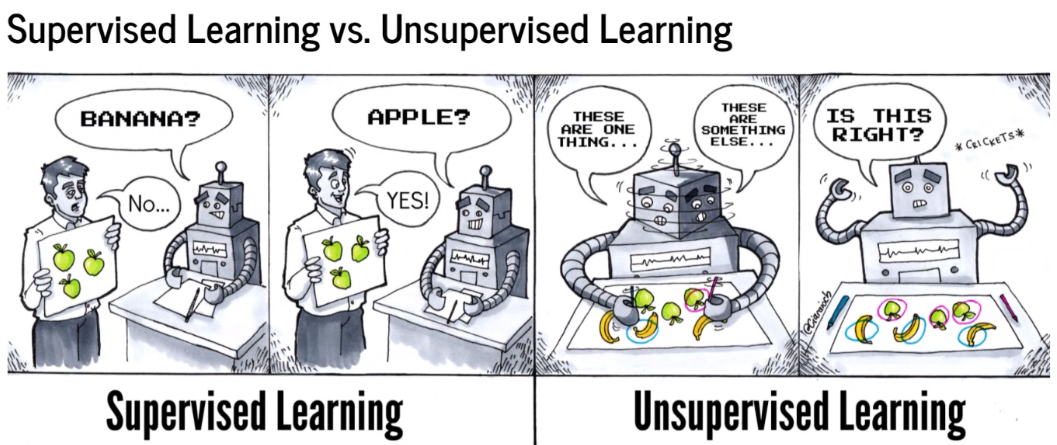
* **características** (atributos)
* **target / objetivo / resultado** (también se denomina verdad fundamental)

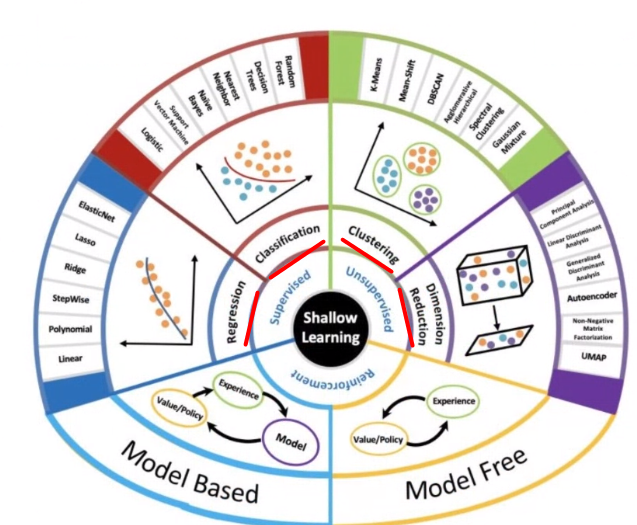
→ cuando un imput tiene características pero no resultados, el algoritmo tiene que predecir el resultado

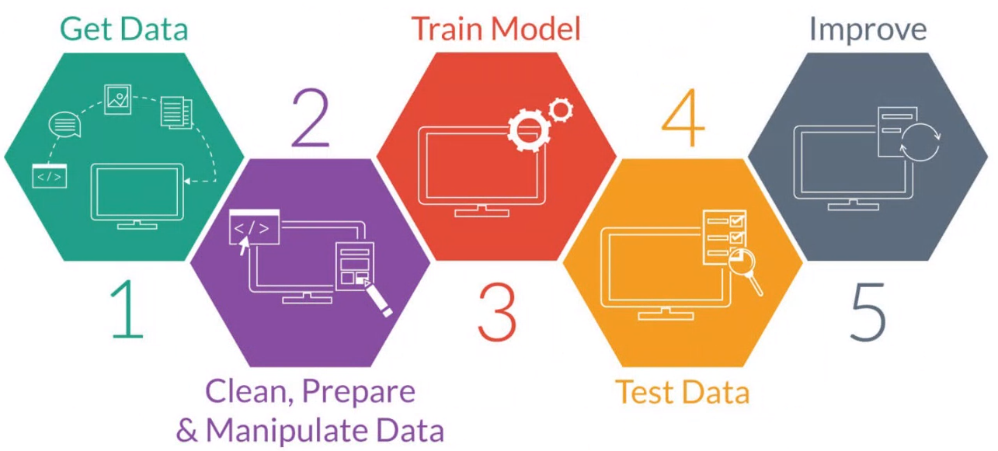
TIPOS DE ALGORITMOS:

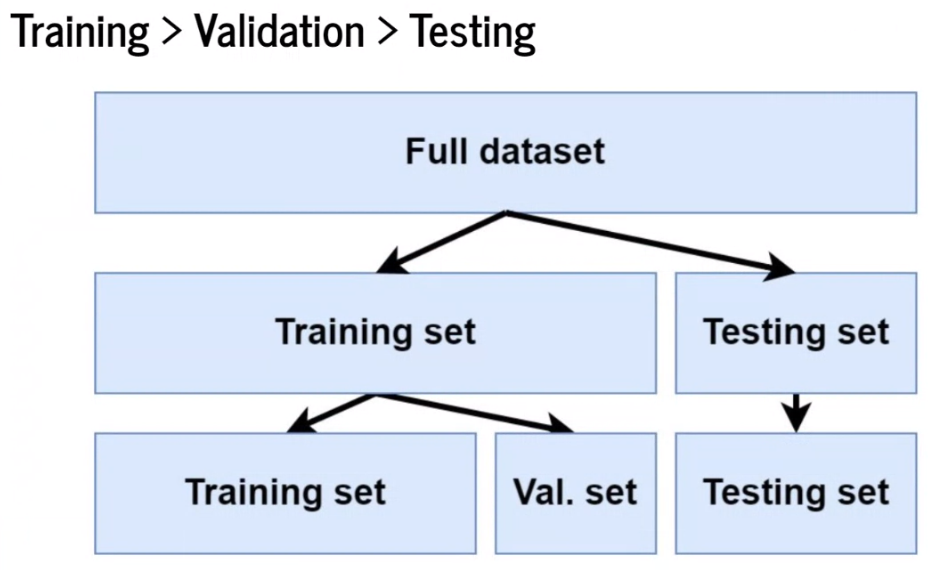
* *REGRESIÓN*: predecir una **variable** (objetivo) **continuo**. Ej.: predecir el precio de los diamantes en base a sus atributos.
* *CLASIFICACIÓN*: clasificar en base a atributos nuestro **objetivo discreto** (ej.: que nos diga si es perro o gato).

1. **APRENDIZAJE NO SUPERVISADO**: al algoritmo se le da **un conjunto de datos** pero **no se le dice el objetivo** (por lo tanto tampoco una respuesta), si no que se le pide que **investigue entre los atributos** y obtenga posibles respuestas ante falta de información o necesidades que tengamos.
2. **APRENDIZAJE REFORZADO**: una máquina aprende a través del proceso de **prueba y error**, en la que se le da un **feedback** con cada acción que realiza y por lo tanto aprende lo que es “bueno” y “malo”. Ej.: enseñar a un algoritmo a jugar a un videojuego.









\* todo tiene que ser números, no valen strings

\* lo que no aporta mejor quitarlo, sino son mas datos para el algoritmo al que se le puede confundir

**Scikit learn -** [API reference](https://scikit-learn.org/stable/modules/classes.html#)

**pip install -U scikit-learn**

**from sklearn.impute import SimpleImputer**

**from sklearn import datasets** → si queremos usar sus datasets públicos

(**EDA: Exploratory Data Analysis**) → ML Data Preparation

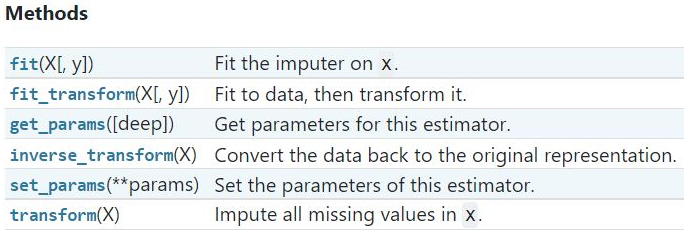
Lo primero que hay que hacer es PREPARAR el dataset:

* **Volumen y Calidad de los datos**: El DF debe de tener datos suficientes para entrenar al modelo, además, las variables independientes tienen que ser de calidad, si hay una cantidad excesiva de parámetros libres el modelo puede fallar.
* **Sentido de los datos**: ¿Hay valores negativos cuando solo deberían ser positivos?, ej precios, etc, ¿hay outliers (a dos desviaciones estándar de la media), boxplot?, ¿los datos tienen diferentes escalas? ¿filas duplicadas? ¿hay correlación entre las variables y el target? … → .describe(), rangos, isnull().sum(), histogram, boxplot
* **1 - Missing Values**: ver qué hacemos con los **valores nulos** (eliminar ese record), poner la media
* **\* - FEATURE SELECTION**: **eliminar columnas** altamente **correlacionadas**
* **\* - FEATURE ENGINEERING**:
  + **agrupar valores** de columnas que tienen muchos datos diferentes (pasándolo a variables categóricas, rangos, etc)
  + **ajustar las fechas** para que el algoritmo lo pueda leer
* **2 - Encoding**: pasar **atributos categóricos a numéricos**
* **3 - Scaling**: ver la importancia de cada atributo (ya que son numéricos, y algo con valores más altos podría tomar más importancia), una opción es **pasarlo todo** a la **misma escala**
* **4 - Data Imbalance**: **ver** la distribución de los datos y **si el target está bien balanceado** (que hay tantos 0 como 1 para que el algoritmo se pueda enseñar bien)

**1- Missing Values - SimpleImputer** (también se podría usar sino methods() de Pandas)**:**

Con la librería podemos limpiar y adaptar el dataset (el siguiente proceso se llama “**IMPUTATION OF MISSING VALUES**”):

[link doc](https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)



1º: **nueva\_variable** = **SimpleImputer(strategy**='mean', **missing\_values**=np.nan**)** → creamos un objeto (**clase**) de sklearn (Con esto estamos instanciando el modelo (ya que creamos esa estancia/clase)) al que vamos a aplicar media, moda, etc a los valores vacíos.



2º: **nueva\_variable** = **nueva\_variable.fit(**columna (variable a transformar)**)**

→ elegimos la variable a “transformar (IMPUTAR)”

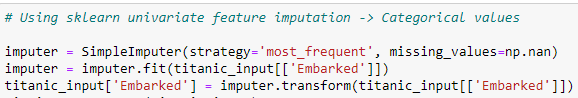


3º: **columna a transformar = nueva\_variable.transform(**columna (variable a transformar)**)**

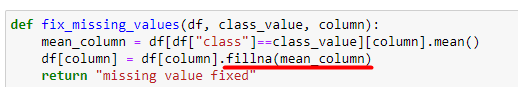
→ transformamos dicha variable



**\* OTRO EJEMPLO:** en vez de la media, cojemos la moda para una columna categórica

****

**\* EJEMPLO CON PANDAS: .fillna()**

****

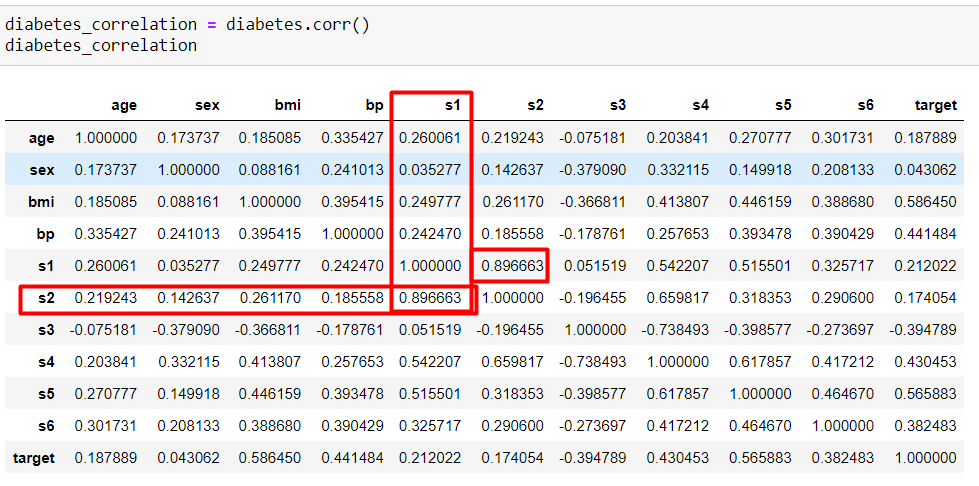
**\* - FEATURE (columnas) SELECTION**

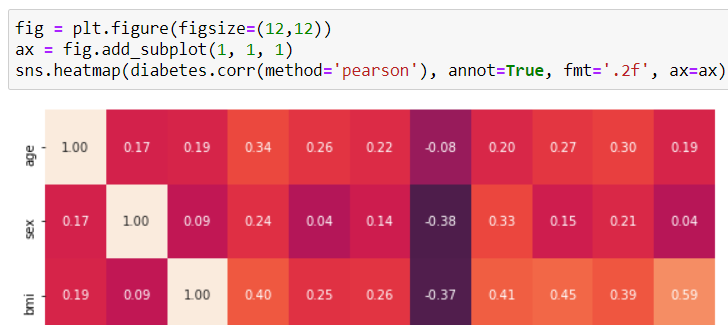
Es la tarea de decidir cómo representar una observación (una observación es un record, una fila del dataset) de cara al target (SI LA ELIMINAMOS O NO)

* ***MANUAL Feature selection - CORRELATION****:* El coeficiente describe tanto la fuerza como la dirección de la relación.
  + PEARSON: evalúa la relación lineal entre dos variables continuas
  + SPEARMAN: evalúa la relación monótona entre dos variables continuas u ordinales (una influye sobre la otra pero no a un ritmo constante

→ Para crear un buen modelo tenemos que dar los datos justos y necesarios. **NO podemos incluir columnas que estén altamente correlacionadas con el target**. Ej.: si el target es ingresos anuales, no podemos poner una columna de ingresos mensuales, porque tomaría demasiado peso en el algoritmo. **OBJETIVO: eliminar columnas altamente correlacionadas**

1. pandas.Series.corr → **S1.corr(S2, method=**’pearson’**)** [link](https://pandas.pydata.org/docs/reference/api/pandas.Series.corr.html)
2. pandas.DataFrame.corr → **DF.corr(method=**’pearson’**)** [link](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html)





→ DEBEMOS DE **ELIMINAR UNA DE LAS DOS COLUMNAS** (ya que estarían ofreciendo la misma información) EJ.: si tenemos MEN y WOMEN (booleanos), una columna ya muestra la info de ambas.

* ***AUTOMATED Feature selection* elimina las columnas con alta correlación de manera automática**

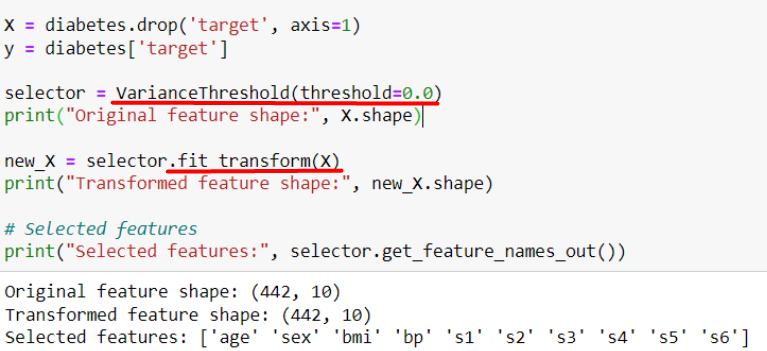
1. **VarianceThreshold** (Umbral de Varianza):

Este método toma un valor de umbral de varianza y elimina **cualquier función por debajo de este umbral**. ANALIZA SOLO EL “X” del dataset, no el “y” (target) **OBJETIVO: eliminar columnas altamente correlacionadas, a través de una varianza baja.**

Si el valor predeterminado para el umbral es 0, lo que hace que se elimine cualquier feature en el que todos sus valores sean iguales (es decir que entre ellos no exista varianza), si es 0,5 elimna aquellos que tengan una varianza por debajo de ella

→ **new\_variable\_instancied** = **VarianceThreshold(threshold=0.0)** [link](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html)

→ **new\_scaled\_data** = **new\_variable\_instancied.fit\_transform(X)**



1. **Univariate feature selection** (ANOVA univariante):

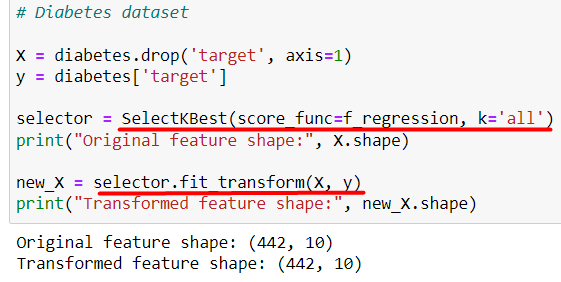
compara las medias entre los grupos que le interesan y determina si alguna de esas medias es estadísticamente significativamente diferente entre sí.

→ **new\_variable\_instancied** = **SelectKBest(score\_func**=f\_regression**, k**='all'**)** [link](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html)

* **score\_func** → por defecto es “f\_classif”, asique lo debemos de cambiar
* **k** → por defecto es 10, asiq mejor poner “all” o un número alto que queramos de test

→ **new\_scaled\_data** = **new\_variable\_instancied.fit\_transform(X, y)**

\* en este caso sí que uitliza el target, así que debemos de poner “X” e “y”

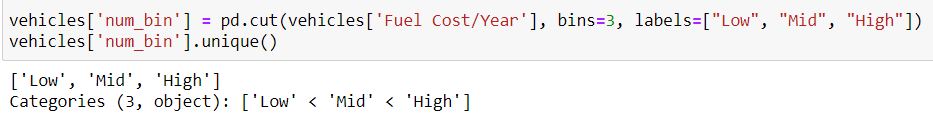


**\* - FEATURE (columnas) ENGINEERING**

**OBJETIVO:**

* **agrupar los datos de las columnas que tengan muchos valores diferentes**
* **ajustar los valores de fecha**
* ***MANUAL*:**

1. **BINNING:** Muchas veces, cuando hay una gran cantidad de datos diferentes dentro de un feature es más difícil para el algoritmo predecir, por lo que una opción es **AGRUPAR valores EN CATEGORÍAS**.



1. **ADJUST DATE:** La fecha es un atributo no numérico, por lo que podemosde hacer dos cosas:
   1. Pasarlo a un “ordinal number”:

* **pd.to\_datetime()**
* **.apply(lambda x: x.toordinal())**





* 1. Extraer la fecha de año, mes y dia en columnas diferentes

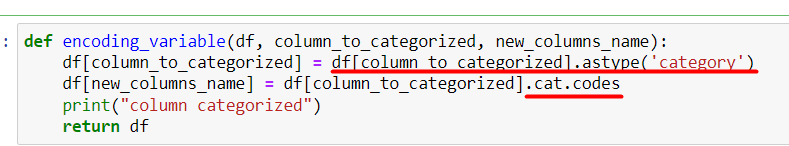


**2 - Encoding:** pasar variables categóricas a numéricas

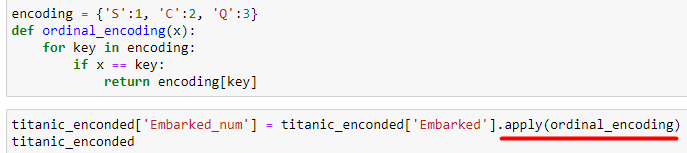
1. **ENCODINGS MÚLTIPLE**: a cada valor un número (problema, le estaríamos asignando un peso diferente a cada valor (a mayor número más peso → Si quieres **PONDERAR usa esto**)

→column**.astype(“category”)**

→column**.cat.codes**

****

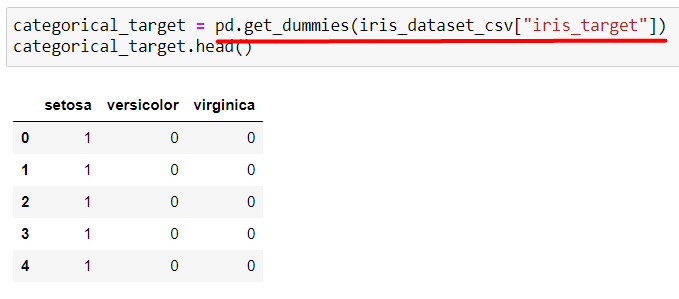
crear un **dic**, **iterar** y **aplicarlo** al df

****

1. **ONE SHOOT ENCODINGS**: valores booleanos, sin dar pesos por ponderaciones

→pd**.get\_dummies(**df**, [**columns**]**, **drop\_first=**True**)**

\* Si ponemos drop\_first = True elimina la primera columna pq realmente es redundante, si es hombre-mujer, te deja solo una columna con 1 y 0 en la misma columna.

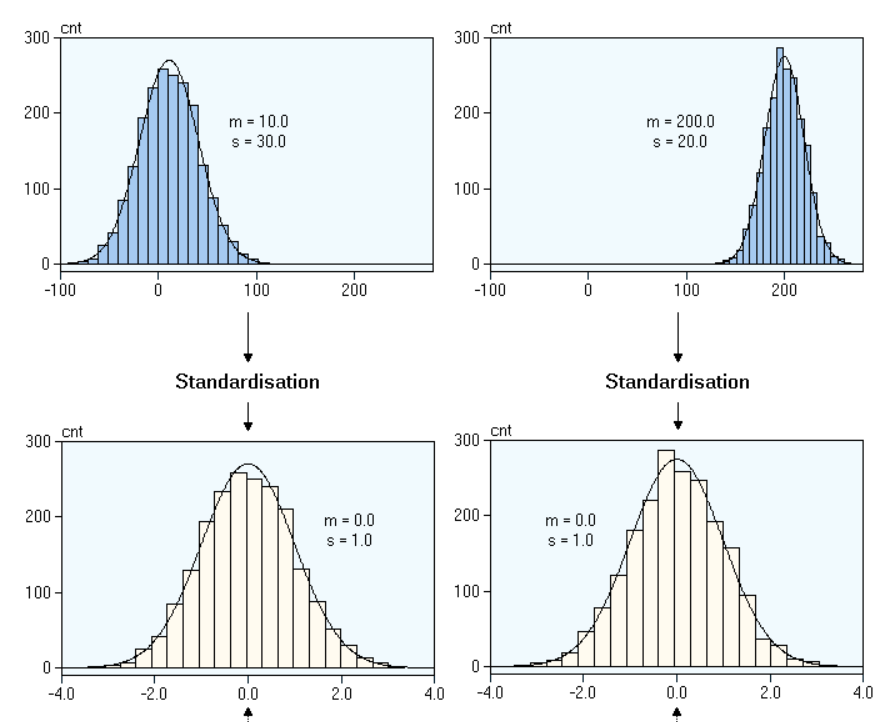
****

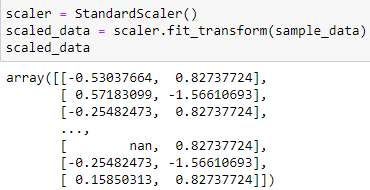
→[link para hacer lo mismo con SciKit Learn](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)

**3 - Scaling:** como cada número supone un peso (más alto, más peso) tendremos que pasar todas las columnas a la misma escala.

1. **ESTANDARIZACIÓN**: escalar a valores parecidos ***siguiendo una distribución normal*** (la media se quedará en 0 y la desviación estándar en 1)

→ **new\_variable\_instancied** = **StandardScaler()**

→ **new\_scaled\_data** = **new\_variable\_instancied.fit\_transform(**data**)**



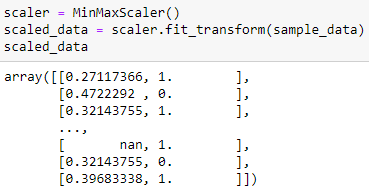
→ pd.DataFrame(new\_scaled\_data**, columns = ...**)



1. **NORMALIZACIÓN** (MinMax SCALING): escalar a valores a un rango entre 0 y 1

→ **new\_variable\_instancied** = **MinMaxScaler()**

→ **new\_scaled\_data** = **new\_variable\_instancied.fit\_transform(**data**)**



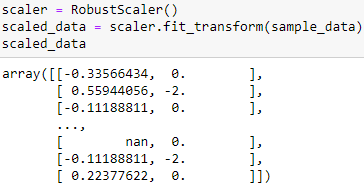
→ pd.DataFrame(new\_scaled\_data**, columns = ...**)



1. **ROBUST SCALER**: lo pasa todo a un rango quartil entre el 25% y 75%

→ **new\_variable\_instancied** = **RobustScaler()**

→ **new\_scaled\_data** = **new\_variable\_instancied.fit\_transform(**data**)**



→ pd.DataFrame(new\_scaled\_data**, columns = ...**)



**4 - Data Imbalance:** ver la distribución de los datos y si el target está bien balanceado (que hay tantos 0 como 1 para que el algoritmo se pueda enseñar bien)

1. OBTENER MÁS DATOS
2. **AJUSTAR LOS DATOS** (creando datos artificiales o disminuyendo lo desbalanceado)
   1. **over-sampling**: “inventarnos” nuevos records con valores que necesitamos del target( si tenemos pocos 0 CREAMOS nuevos records con target=1)
   2. **under-sampling**: “eliminar” records con valores que necesitamos del target para balancearlo (si tenemos muchos 0 ELIMINAMOS records con target=0)

**conda install -c conda-forge imbalanced-learn**

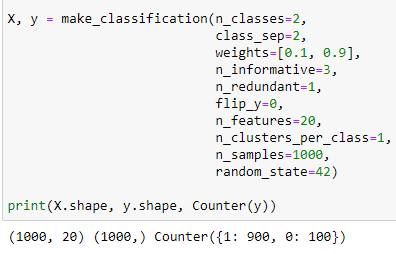
**nueva\_variable** = **SMOTE(**random\_state=42**)**

**X\_res, y\_res** = **nueva\_variable.fit\_resample(X, y)**

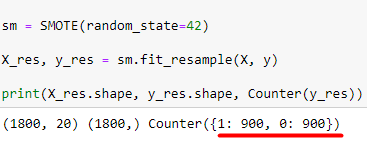
*print(X\_res.shape , y\_res.shape , Counter(y\_res))*

**EJEMPLO:**

* **dataset original desbalanceado**



* **dataset RESULTANTE balanceado:**



**5 - TEST and TRAINING SET:**

1. ***PREPARAR EL DATASET DE ENTRENAMIENTO Y TEST:***

Dividimos el Dataframe (o array) en 2, una parte se utilizará para entrenar al algoritmo y otra para testarlo. (normalmente 60-80% para entrenarlo)

Para dividirlo usamos el método **train\_test\_split()**  [link doc](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

**EJEMPLO** (En el siguiente ejemplo el 80% irá para entrenar y el 20% para testar):

**from sklearn.model\_selection import train\_test\_split**

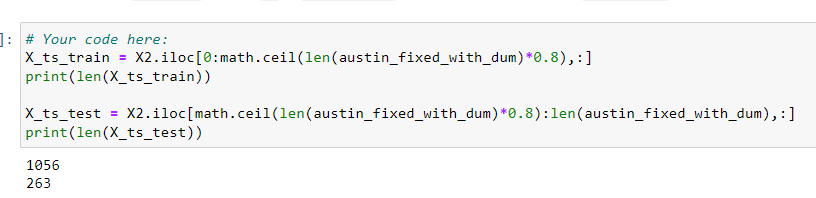
X\_train, X\_test, y\_train, y\_test **= train\_test\_split(X**, **y**, **test\_size**=0.2, **random\_state=42)**

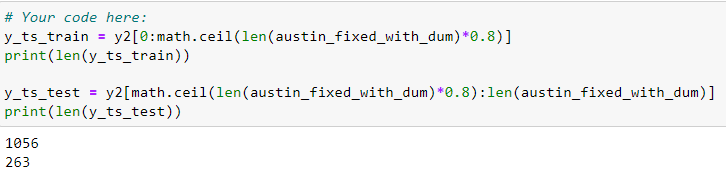
* **X** → una parte del dataset (FEATURES)
  + **X\_train** → features que el modelo conoce
  + **X\_test** → features que el modelo NO conoce
* **y** → la otra (target), normalmente una única columna)
  + **X\_train** → target que el modelo conoce
  + **X\_test** → target que el modelo NO conoce





Sino, se podría hacer así (**por orden**)





1. ***ENTRENAR Y TESTAR MODELOS DE REGRESIÓN LINEAL:***

**from sklearn import linear\_model**

**from sklearn.linear\_model import ElasticNet**

**from sklearn.linear\_model import Ridge**

**from sklearn.svm import SVR**

**from sklearn.linear\_model import SGDRegressor**

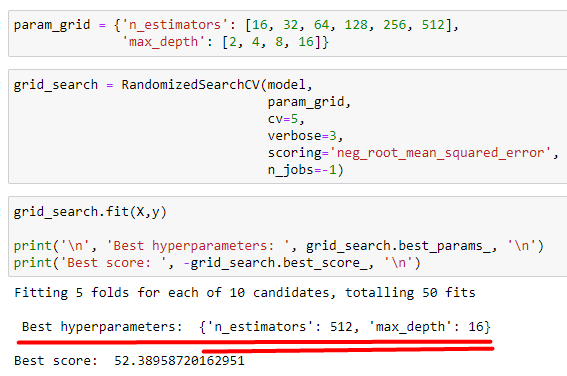
**from sklearn.linear\_model import LinearRegression**

PASOS:

1. SELECCIONAMOS el modelo

* **model** = **linear\_model.Lasso()**
* model = **ElasticNet()**
* model = **Ridge()**
* model = **SVR()** # --> este no es lineal
* model = **SGDRegressor()**
* model = **LinearRegression()**
* model = **RandomForestRegressor()**

*\* OPTIMIZACIÓN DE LOS HIPER PARÁMETROS*[***GridSearchCV()***](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) *ò* [***RandomizedSearchCV()***](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

**

1. ENTRENAMOS el modelo

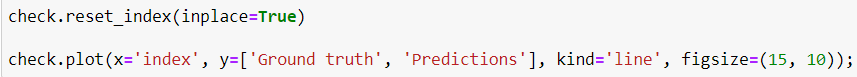
→ **model.fit(X\_train, y\_train)**

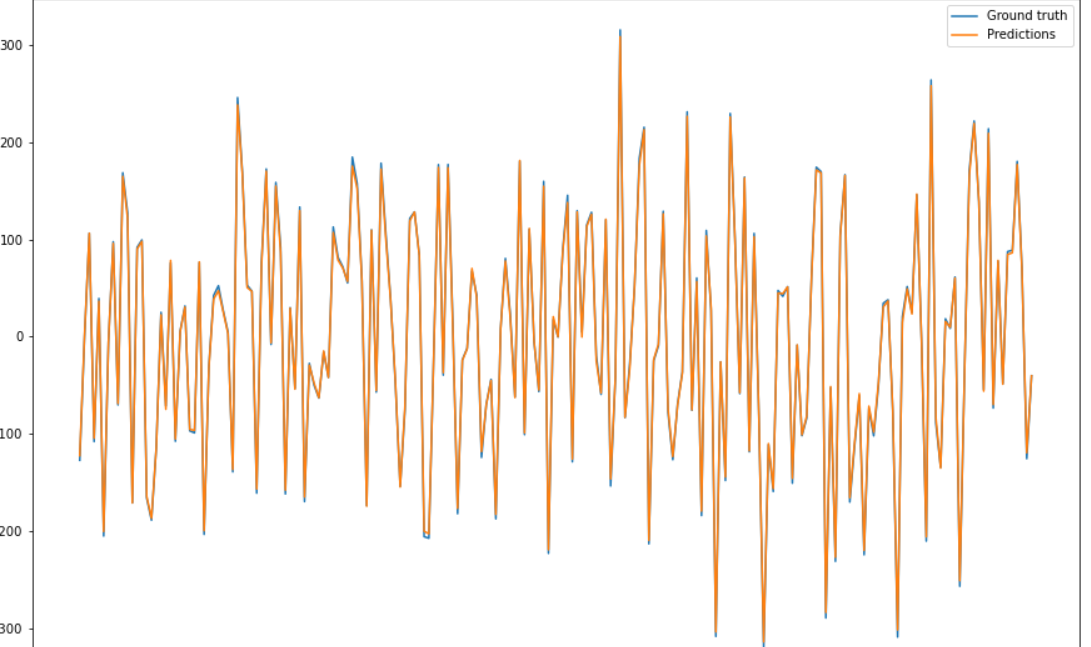
1. TESTAMOS el modelo

→ ***predictions*** = **model.predict(X\_test)**

1. podemos crear un DF, plotearlo, …



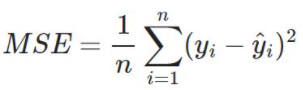




→ predicción perfecta (NO HAY DIFERENCIAS ENTRE LO PREDECIDO Y LO TESTADO)

1. ***EVALUAR LOS ERRORES*** *(nos quedaremos con el RMSE y con el CROSS-VALIDATION)*

* MSE (Mean Square Error): El error cuadrático medio mide la diferencia cuadrática promedio entre los valores estimados y el valor real **(es decir la media de los errores al cuadrado)**.

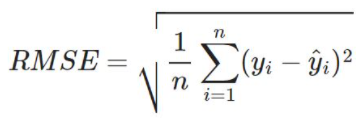


→ **mean\_squared\_error(**y\_test, y\_pred**)**

\* Es sensible a los outliers, al coger la media de los errores

NO LO VAMOS A UTILIZAR

* **RMSE (Root Mean Square Error):** es simplemente la raíz cuadrada del error cuadrático medio

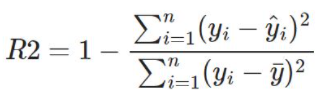


→ **mean\_squared\_error(**y\_test, y\_pred**)\*\*0.5**

\* Es sensible a los outliers, aun así **es mejor pq no está al cuadrado**

**SI LO VAMOS A UTILIZAR**

* R2 (R-squared): representa qué proporción de la varianza de una variable dependiente se explica por las variables independientes (p. ej., un r-cuadrado del 60 % revela que el 60 % de los datos se ajustan al modelo de regresión).



→ **r2\_score(**y\_test, y\_pred**)**

ò

→ **regressor.score(**X\_test, y\_test**)**

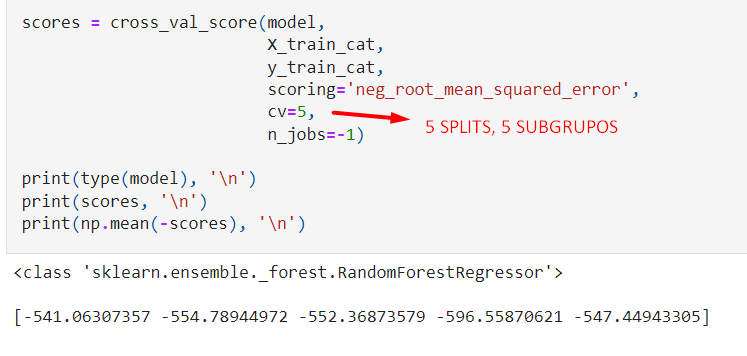
**\* de 0 a 1, donde 0 significa que el 0 % de los datos se ajustan al modelo de regresión y 1 significa que el 100 % de los datos se ajustan al modelo de regresión.**

* **Cross-validation:** Es un método que realiza directamente los splits sin tener que hacerlos nosotros manualmente. Hace GRUPOS DE SPLIT, sobre los cuales calcula su RMSE y devuelve un RMSE FINAL que consiste en la media de todos los RMSE

**from sklearn.model\_selection import cross\_val\_score**

[LINK](https://scikit-learn.org/stable/modules/cross_validation.html)

→ scores **= cross\_val\_score(model, X, y, scoring=**'neg\_root\_mean\_squared\_error'**, cv=**5**, n\_jobs=**1**)**



**SI LO VAMOS A UTILIZAR**

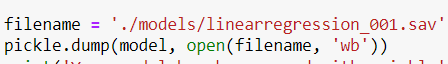
1. ***GUARDAR Y CARGAR EL MODELO (no el dataframe):***

Una vez que hemos entrenado y testado el modelo, lo podemos guardar para cargarlo directamente y no tenerlo que volver a entrenar

***import pickle***

***import joblib***

* pickle:
  + GUARDAR: **pickle.dump(model, open(**“location.sav”**, 'wb'))**



* + CARGAR: **loaded\_model** = **pickle.load(open(**“location.sav”**, 'rb'))**

****

* joblib
  + GUARDAR: **joblib.dump(model,** “location.sav”**)**

****

* + CARGAR: **loaded\_model = joblib.load(**“location.sav”**)**

****

**Algoritmos de CLASIFICACIÓN:**

* **One-Class Classification**
* Multiclass Classification
* Multilabel Classification (also known as Multi Output Classification, but not exactly the same)
* Multitask Classification (also known as Multiclass-multioutput classification

**ONE-CLASS CLASSIFICATION:**

El proceso de EDA es el mismo, por lo que vamos a pasar directamente a:

1. ***ENTRENAR Y TESTAR LOS MODELOS:***

[***SGDClassifier***](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html)

***from sklearn.linear\_model import SGDClassifier***

[***SVC***](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)

***from sklearn.svm import SVC***

[***DecisionTreeClassifier***](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

***from sklearn.tree import DecisionTreeClassifier***

[***LogisticRegression***](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

***from sklearn.linear\_model import LogisticRegression***

**PASOS:**

1. SELECCIONAMOS el modelo

* **model** = **SGDClassifier()**
* model = **SVC()**
* model = **Ridge()**
* model **= LogisticRegression(**solver='liblinear'**)**

1. ENTRENAMOS el modelo

→ **model.fit(X\_train, y\_train)**

1. TESTAMOS el modelo

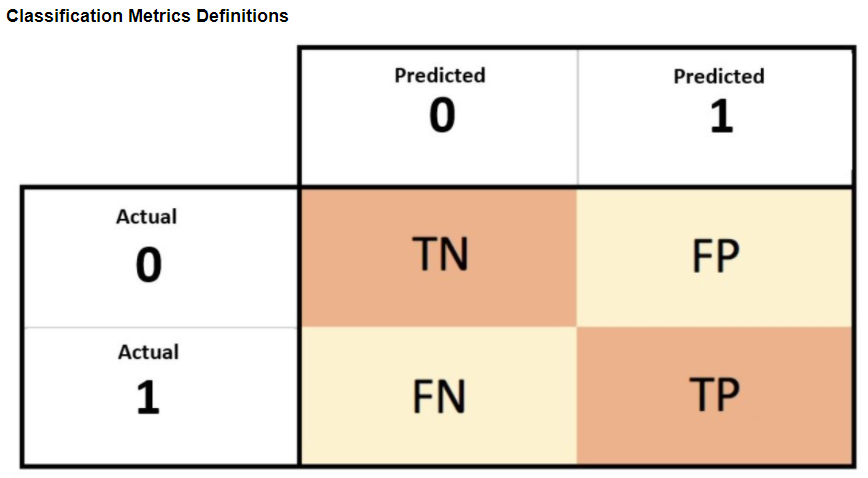
→ ***predictions*** = **model.predict(X\_test)**

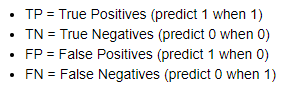
1. podemos crear un DF, plotearlo, …

→ check=pd.DataFrame({'Ground truth':y\_test, 'Predictions':predictions, 'Diff':y\_test-predictions})

check.tail(20)

------------------------------------------------------------------------------------------------------------------------------------------

******

******

**MÉTRICAS PARA SABER SI EL MODELO ES VÁLIDO:**

* ***ACCURACY***: Es la **proporción de predicciones** que fueron **correctas**.



→ **model.score(**X\_test, y\_test**)**

****

→ **accuracy\_score(**y\_test, predictions**)**

****

**→ *accuracy\_score(****y\_test, predictions,* ***normalize****=False****)***

****

* ***PRECISION***: Es la **proporción de positivos predecidos** que **realmente fueron positivos**.

****

→ **precision\_score(**y\_test, predictions**)**

****

* ***RECALL***: Es la **proporción de positivos** que fueron **predecidos correctamente**.

****

→ **recall\_score(**y\_test, predictions**)**

****

* ***F1 SCORE***: Es el **promedio** ponderado **entre “precision” y “recall”.**

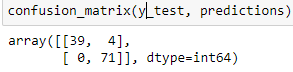
****

→ **f1\_score(**y\_test, predictions**)**

****

* ***CONFUSION MATRIX:*** Muestra de manera resumida y visual la cantidad de positivos, negativos, falsos positivos y falsos negativos

→ **confusion\_matrix(**y\_test, predictions**)**



fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(13,8))

ax = sns.heatmap(confusion\_matrix(y\_test, predictions), annot=True)

b, t = ax.get\_ylim()

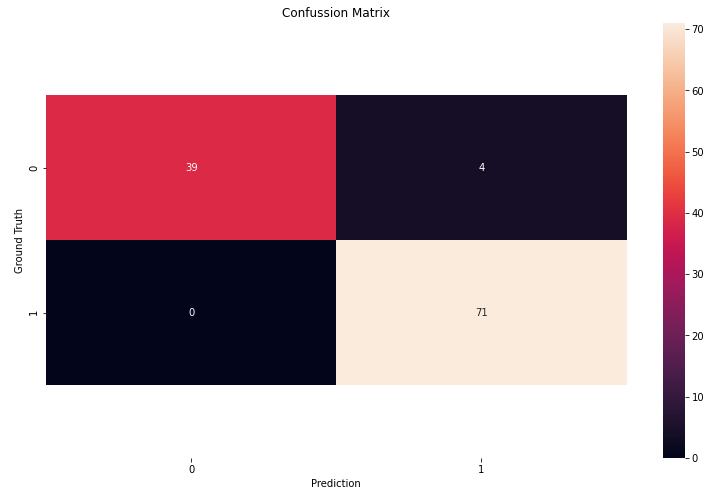
#ax.set\_ylim(b + 0.5, t - 0.5)

plt.title('Confussion Matrix')

plt.ylabel('Ground Truth')

plt.xlabel('Prediction')

plt.show();

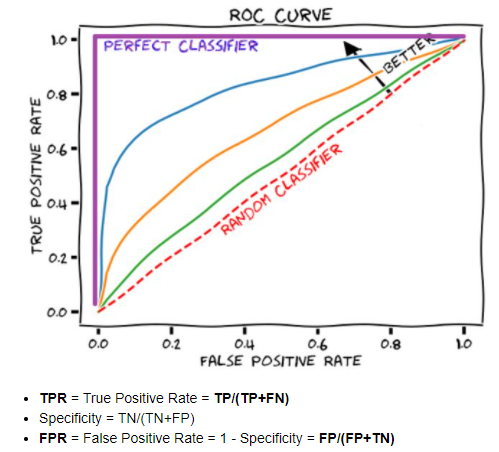


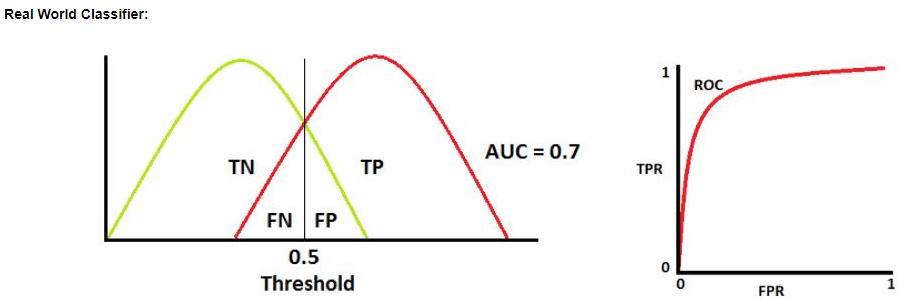
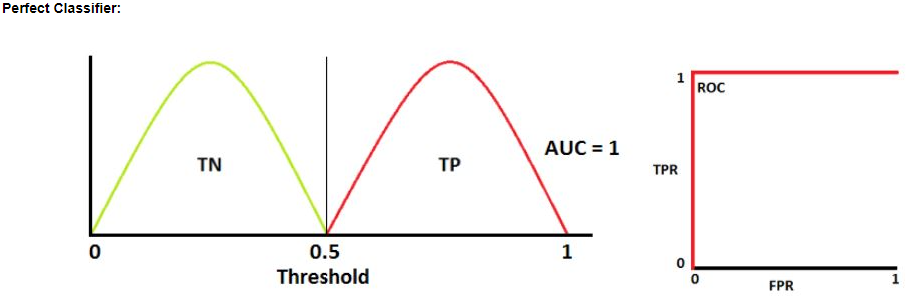
* ***ROC Curve and AUC:*** ROC (Receiver Operating Characteristic) and AUC (Area Under the Curve). Visualizan y evalúan el rendimiento del modelo.

→ **roc\_auc\_score(**y\_test, predictions**)**

****

**→ Cuanto más CERCANO A 1 mejor**

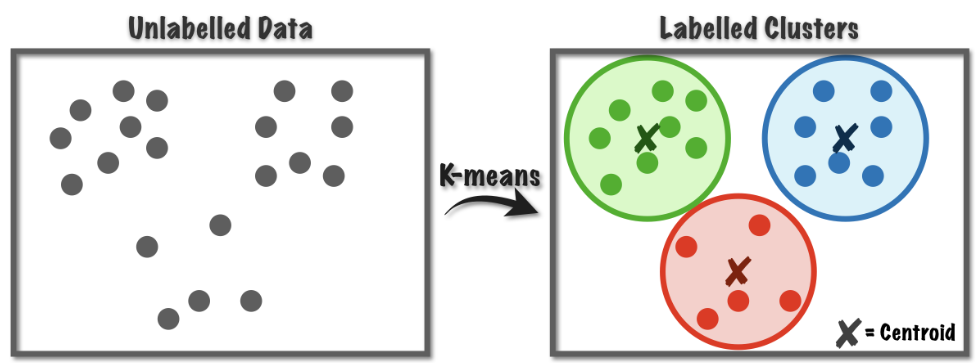
******

******

**Unsupervised learning**

**APRENDIZAJE NO SUPERVISADO**: al algoritmo se le da **un conjunto de datos** pero **no se le dice el objetivo** (por lo tanto tampoco una respuesta), si no que se le pide que **investigue entre los atributos** y obtenga posibles respuestas ante falta de información o necesidades que tengamos.

Se basa en el **CLUSTERING** (segmentación de atributos en grupos parecidos)



***pip install umap-learn***

***from sklearn.decomposition import PCA***

***import umap***

***conda install -c districtdatalabs yellowbrick***  ó ***pip install yellowbrick***

***from sklearn.cluster import KMeans***

***from sklearn.cluster import DBSCAN***

***from yellowbrick.cluster import KElbowVisualizer***

***from sklearn.datasets import load\_digits***