Tema 3. Preprocesamiento de datos

Dr. Manuel Castillo-Cara
Intelligent Ubiquitous Technologies – Smart Cities (IUT-SCi)
Web: www.smartcityperu.org

Índice

- Análisis Exploratorio de datos.
- Preprocesamiento de datos.
- Métodos de remuestreo.

Smart City

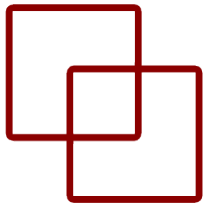
LAB CTIC UNI

Análisis Exploratorio de datos

1. Definición

- Tenemos que comprender nuestros datos, es decir, ¿de qué se trata? Por ejemplo:
  - ¿Cuántos registros hay?
  - ¿Están todas las filas completas ó tenemos campos con valores nulos? En caso que haya demasiados nulos: ¿Queda el resto de información inútil?
  - ¿Que datos son discretos y cuales continuos? Muchas veces sirve obtener el tipo de datos: texto, int, double, float.
  - Si es un problema supervisado (binaria/multiclase, balanceo de la clase) o un problema no supervisado.
  - ¿Cuales parecen ser características importantes? ¿Cuales podemos descartar?
  - ¿Siguen alguna distribución? ¿Hay correlación entre características?
  - ¿Cuales son los Outliers? (unos pocos datos aislados que difieren drásticamente del resto y «contaminan» ó desvían las distribuciones)
  - ¿Tenemos posible sesgo de datos?
- Veamos los pasos a seguir.

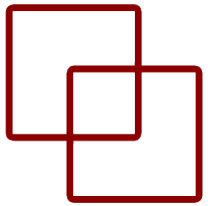
2. Cargar un CSV y librerías



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns

dataset='data/countries.csv'
df = pd.read_csv(dataset, sep=";") #index_col=0
df
```

	alpha_2	alpha_3	area	capital	continent	currency_code	currency_name	eqivalent_fips_code	fips	geoname_id	languages
0	AD	AND	468.0	Andorra la Vella	EU	EUR	Euro	NaN	AN	3041565	ca
1	AE	ARE	82880.0	Abu Dhabi	AS	AED	Dirham	NaN	AE	290557	ar-AE,fa,en,hi,ur
2	AF	AFG	647500.0	Kabul	AS	AFN	Afghani	NaN	AF	1149361	fa-AF,ps,uz-AF,tk

3. Información básica (I)



- Lo primero que tenemos que conocer es el tamaño de nuestro dataset y los nombres de las columnas que tiene.

```
print('Cantidad de Filas y columnas:',df.shape)
print('Nombre columnas:',df.columns)
```

```
Cantidad de Filas y columnas: (252, 19)
Nombre columnas: Index(['alpha_2', 'alpha_3', 'area', 'capital', 'continent', 'currency_code',
       'currency_name', 'eqivalent_fips_code', 'fips', 'geoname_id',
       'languages', 'name', 'neighbours', 'numeric', 'phone', 'population',
       'postal_code_format', 'postal_code_regex', 'tld'],
      dtype='object')
```

3. Información básica (II)

- Las columnas que no tenga el tipo correcto deberemos transformarlas.
  - Cambiar el tipo de dato.
- Cuando tenemos características numéricas y categóricas deberemos uniformizarlo
  - One-Hot Encoding
- Ver si alguna columna tiene valores NaN que deberemos de tratar.

```
print(df.isnull().sum())
```

```
alpha_2                 1
alpha_3                 0
area                    0
capital                 6
continent              42
currency_code           1
currency_name           1
eqivalent_fips_code   251
fips                    3
geoname_id              0
languages               3
name                    0
neighbours             87
numeric                 0
phone                   5
population              0
postal_code_format     98
postal_code_regex     100
tld                     2
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252 entries, 0 to 251
Data columns (total 19 columns):
alpha_2                 251 non-null object
alpha_3                 252 non-null object
area                    252 non-null float64
capital                 246 non-null object
continent               210 non-null object
currency_code           251 non-null object
currency_name           251 non-null object
eqivalent_fips_code     1 non-null object
fips                    249 non-null object
geoname_id              252 non-null int64
languages               249 non-null object
name                    252 non-null object
neighbours              165 non-null object
numeric                 252 non-null int64
phone                   247 non-null object
population              252 non-null int64
postal_code_format      154 non-null object
postal_code_regex       152 non-null object
tld                     250 non-null object
dtypes: float64(1), int64(3), object(15)
memory usage: 37.5+ KB
```

3. Información básica (III)

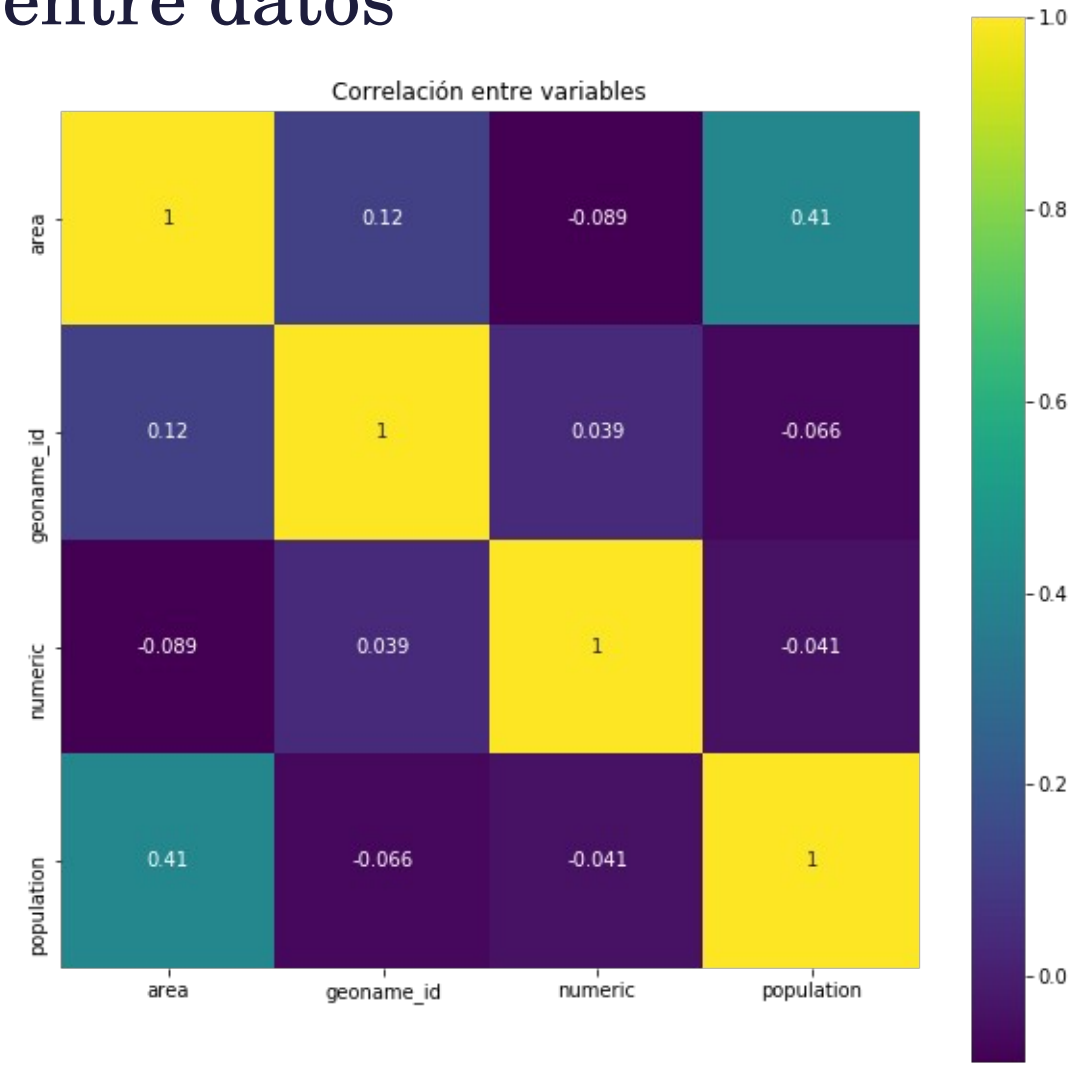
- Conocer los rangos de nuestros datos.

```
df.describe()
```

	area	geoname_id	numeric	population
count	2.520000e+02	2.520000e+02	252.000000	2.520000e+02
mean	5.952879e+05	2.427870e+06	434.309524	2.727679e+07
std	1.904818e+06	1.632093e+06	254.663139	1.164127e+08
min	0.000000e+00	4.951800e+04	0.000000	0.000000e+00
25%	1.098000e+03	1.163774e+06	217.000000	1.879528e+05
50%	6.489450e+04	2.367967e+06	436.000000	4.268583e+06
75%	3.622245e+05	3.478296e+06	652.500000	1.536688e+07
max	1.710000e+07	8.505033e+06	894.000000	1.330044e+09

4. Visualización

- Ver la correlación entre datos



4.1. Extracción de información (I)

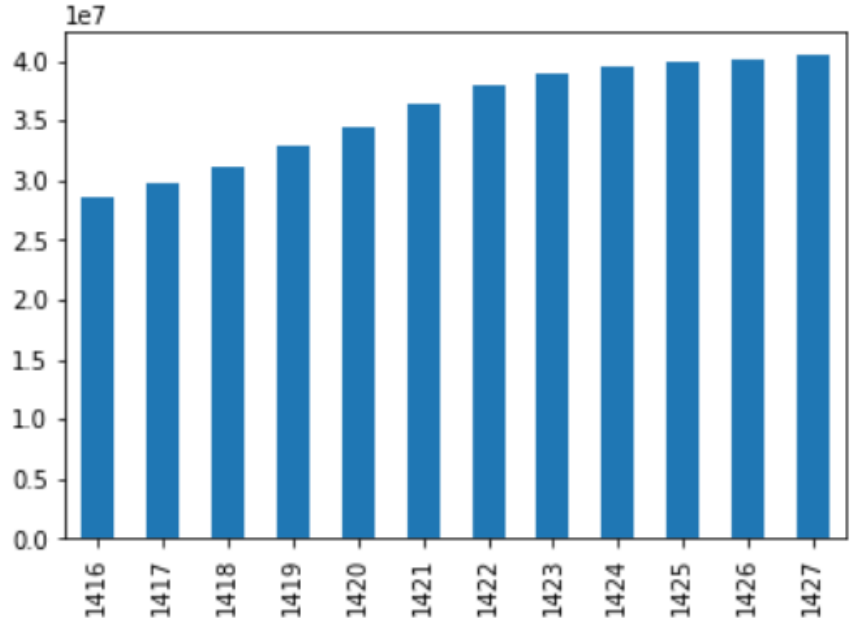
- Crecimiento en España

```
df_pop_es = df_pop[df_pop["country"] == 'Spain' ]
print(df_pop_es.head())
```

```
     country  year  population
1416   Spain  1952    28549870
1417   Spain  1957    29841614
1418   Spain  1962    31158061
1419   Spain  1967    32850275
1420   Spain  1972    34513161
```

```
df_pop_es.drop(['country'],axis=1)['population'].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f21aec21fd0>
```

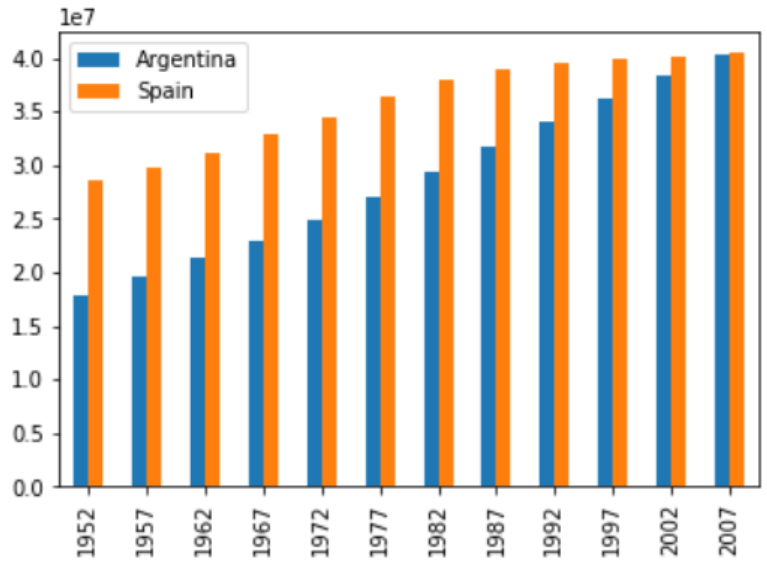


4.2. Extracción de información (II)

- Comparar con otros países

```
df_pop_ar = df_pop[(df_pop["country"] == 'Argentina')]
anios = df_pop_es['year'].unique()
pop_ar = df_pop_ar['population'].values
pop_es = df_pop_es['population'].values

df_plot = pd.DataFrame({'Argentina': pop_ar,
                        'Spain': pop_es},
                          index=anios)
df_plot.plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f21b1e69a90>
```

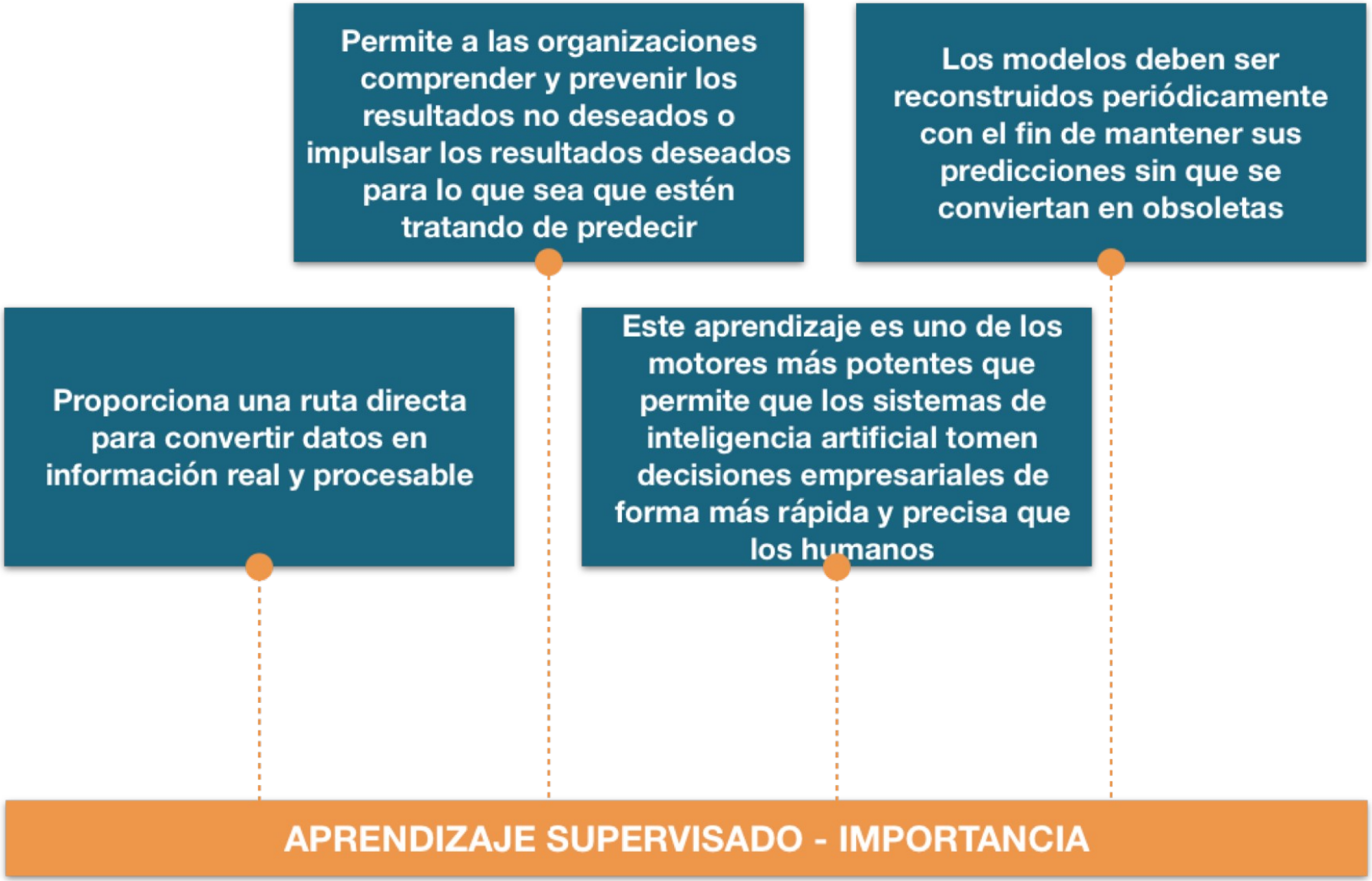


Smart City

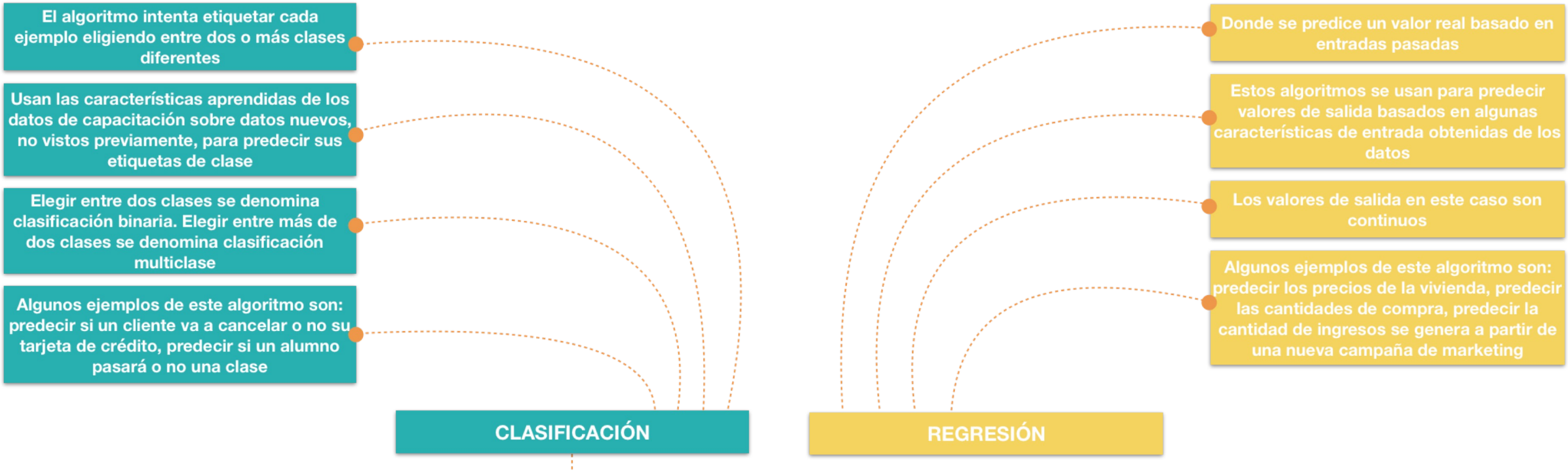
LAB CTIC UNI

Preprocesamiento de datos

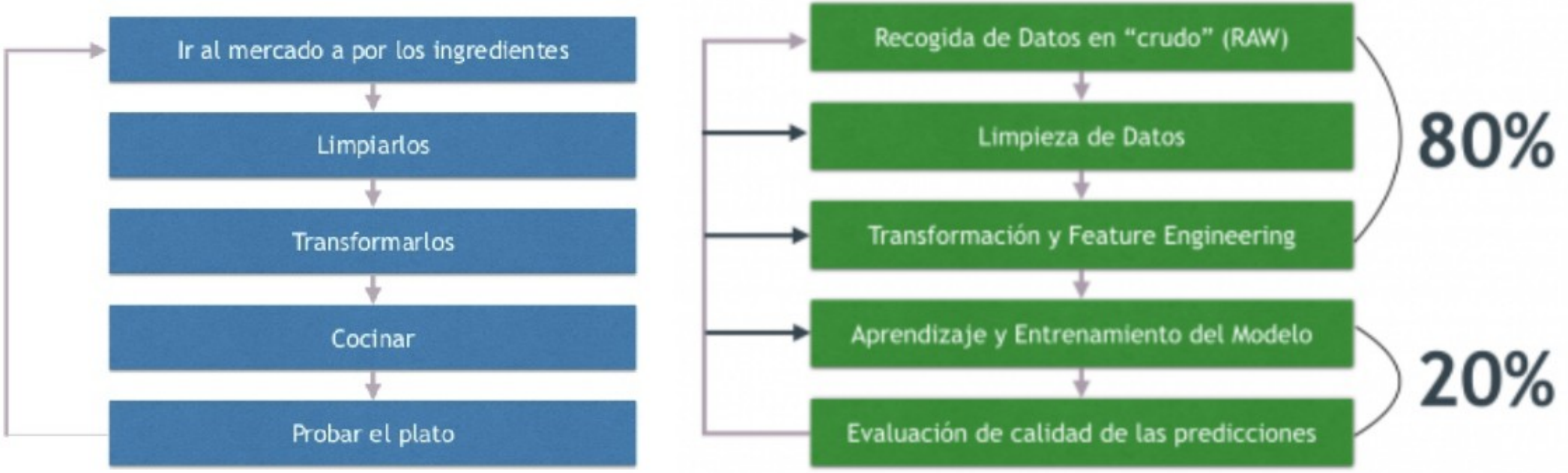
1. Aprendizaje supervisado



2. Clasificación Vs. Regresión



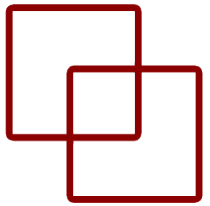
3. Tratamiento de datos



(a) Analogía con la vida diaria. (b) Proceso completo de machine learning.

FIGURA 4.1: Imagen en la que puede verse el proceso completo de machine learning

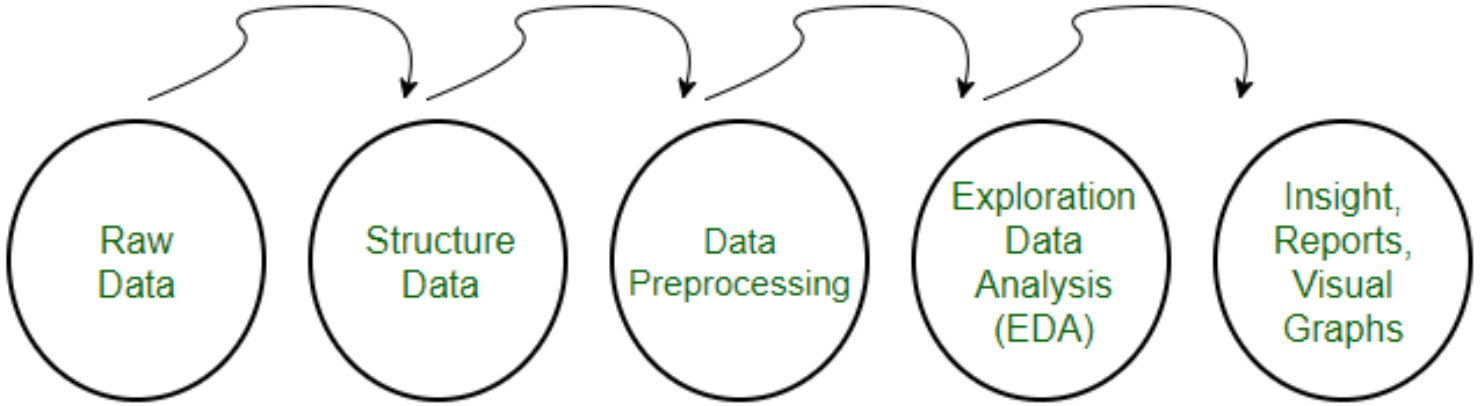
4. Necesidad de preprocesamiento



- Los datos en bruto deben ser procesados.
  - Por ejemplo, algoritmos tipo árbol se comportan mejor con atributos de tipo nominal.
- Problema empírico → necesidad de probar diferentes transformaciones de datos.
- Algunas reglas generales:
  - Los métodos basados en instancias son más efectivos si los atributos de entrada tienen la misma escala.
  - Los métodos de regresión pueden funcionar mejor si los atributos de entrada están estandarizados.

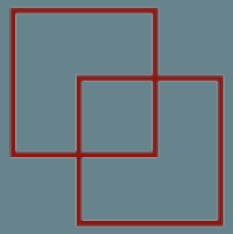
5. Transformación de datos

- Cuando tengamos mucha varianza en los diferentes atributos de un conjunto de datos. Por ejemplo:
  - Edad tiene 2 dígitos y sueldo 5 o más dígitos.
  - El modelo le otorgará más importancia a salario que a edad.
- Por tanto, debemos unificar las diferentes variables.



6. Transformación de los datos

- Utilizaremos el paquete "*sklearn.preprocessing*" de Python.
- Dos métodos
  - **Ajuste y transformación múltiple** es el enfoque preferido.
    - Llama a la función ***fit()*** para preparar los parámetros de la transformación una vez en sus datos.
    - Luego, puede usar la función ***transform()*** en los mismos datos para prepararlos para el modelado y nuevamente en el conjunto de datos de prueba o validación o los nuevos datos que puede ver en el futuro.
  - **Ajuste y transformación combinados** es una conveniencia que puede usar para tareas únicas. Esto puede ser útil si está interesado en trazar o resumir los datos transformados y se utilizará la función ***fit_transform()***.
- Existe una gran cantidad de funciones que podemos aplicar en esta fase de preprocesamiento según la necesidad de nuestros datos.



Smart City

LAB CTIC UNI

Métodos de transformación de datos

1. Métodos de transformación

preprocessing.Binarizer([threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold
preprocessing.FunctionTransformer([func, …])	Constructs a transformer from an arbitrary callable.
preprocessing.KBinsDiscretizer([n_bins, …])	Bin continuous data into intervals.
preprocessing.KernelCenterer()	Center a kernel matrix
preprocessing.LabelBinarizer([neg_label, …])	Binarize labels in a one-vs-all fashion
preprocessing.LabelEncoder	Encode target labels with value between 0 and n_classes-1.
preprocessing.MultiLabelBinarizer([classes, …])	Transform between iterable of iterables and a multilabel format
preprocessing.MaxAbsScaler([copy])	Scale each feature by its maximum absolute value.
preprocessing.MinMaxScaler([feature_range, copy])	Transform features by scaling each feature to a given range.
preprocessing.Normalizer([norm, copy])	Normalize samples individually to unit norm.
preprocessing.OneHotEncoder([categories, …])	Encode categorical features as a one-hot numeric array.
preprocessing.OrdinalEncoder([categories, dtype])	Encode categorical features as an integer array.
preprocessing.PolynomialFeatures([degree, …])	Generate polynomial and interaction features.
preprocessing.PowerTransformer([method, …])	Apply a power transform featurewise to make data more Gaussian-like.
preprocessing.QuantileTransformer([…])	Transform features using quantiles information.
preprocessing.RobustScaler([with_centering, …])	Scale features using statistics that are robust to outliers.
preprocessing.StandardScaler([copy, …])	Standardize features by removing the mean and scaling to unit variance

preprocessing.add_dummy_feature(X[, value])	Augment dataset with an additional dummy feature.
preprocessing.binarize(X[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix
preprocessing.label_binarize(y, classes[, …])	Binarize labels in a one-vs-all fashion
preprocessing.maxabs_scale(X[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
preprocessing.minmax_scale(X[, …])	Transform features by scaling each feature to a given range.
preprocessing.normalize(X[, norm, axis, …])	Scale input vectors individually to unit norm (vector length).
preprocessing.quantile_transform(X[, axis, …])	Transform features using quantiles information.
preprocessing.robust_scale(X[, axis, …])	Standardize a dataset along any axis
preprocessing.scale(X[, axis, with_mean, …])	Standardize a dataset along any axis
preprocessing.power_transform(X[, method, …])	Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

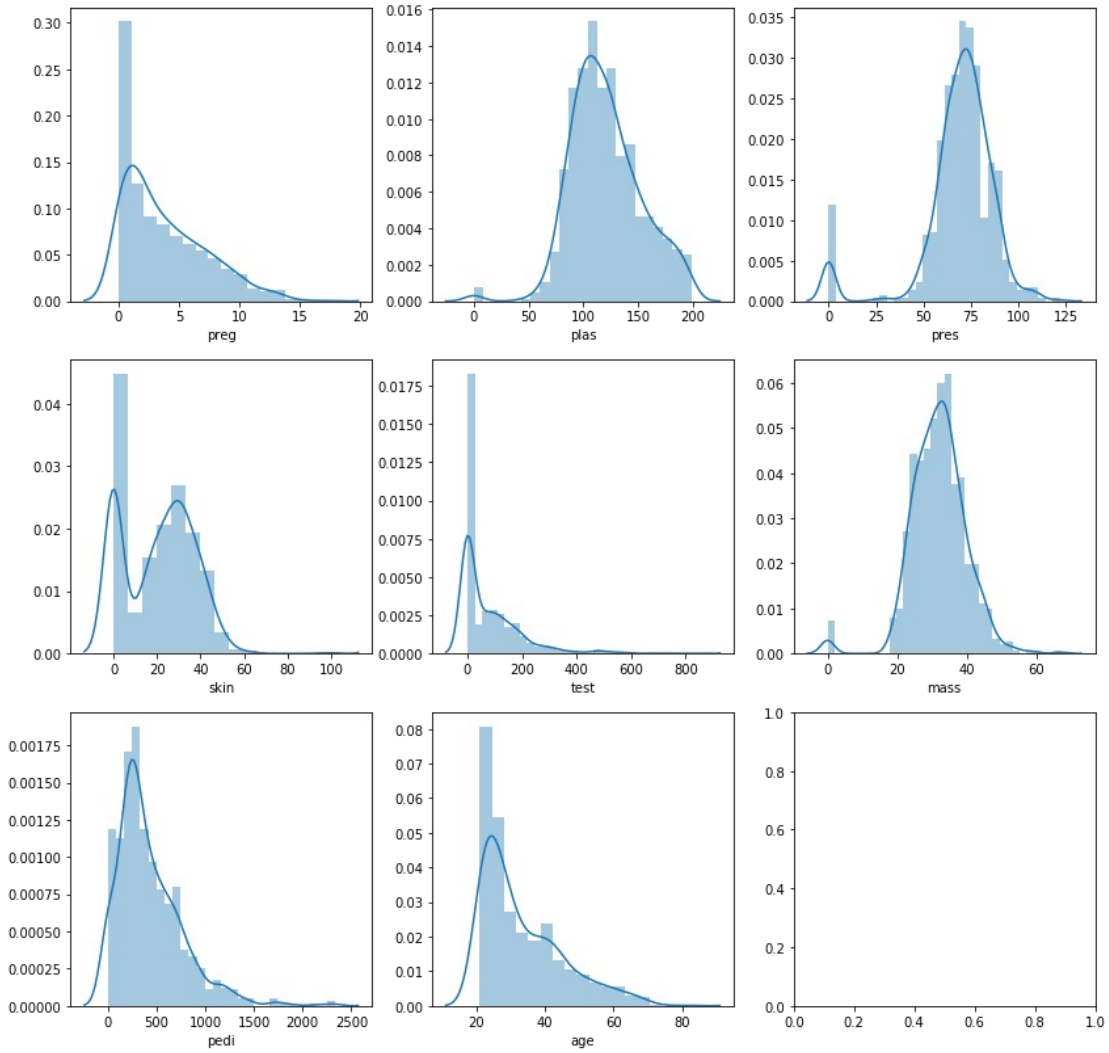
2. Escalamiento (I)

- Esta transformación es útil para los algoritmos de optimización utilizados en el núcleo de los algoritmos de aprendizaje automático como Gradiente Descendiente.
- También es útil para algoritmos que ponderan entradas como Regression y Neural Networks y algoritmos que usan medidas de distancia como k-Nearest Neighbours.
- Puede reescalar sus datos usando la clase *MinMaxScaler*.
- Después de reescalar puede ver que todos los valores están en el rango [0,1].

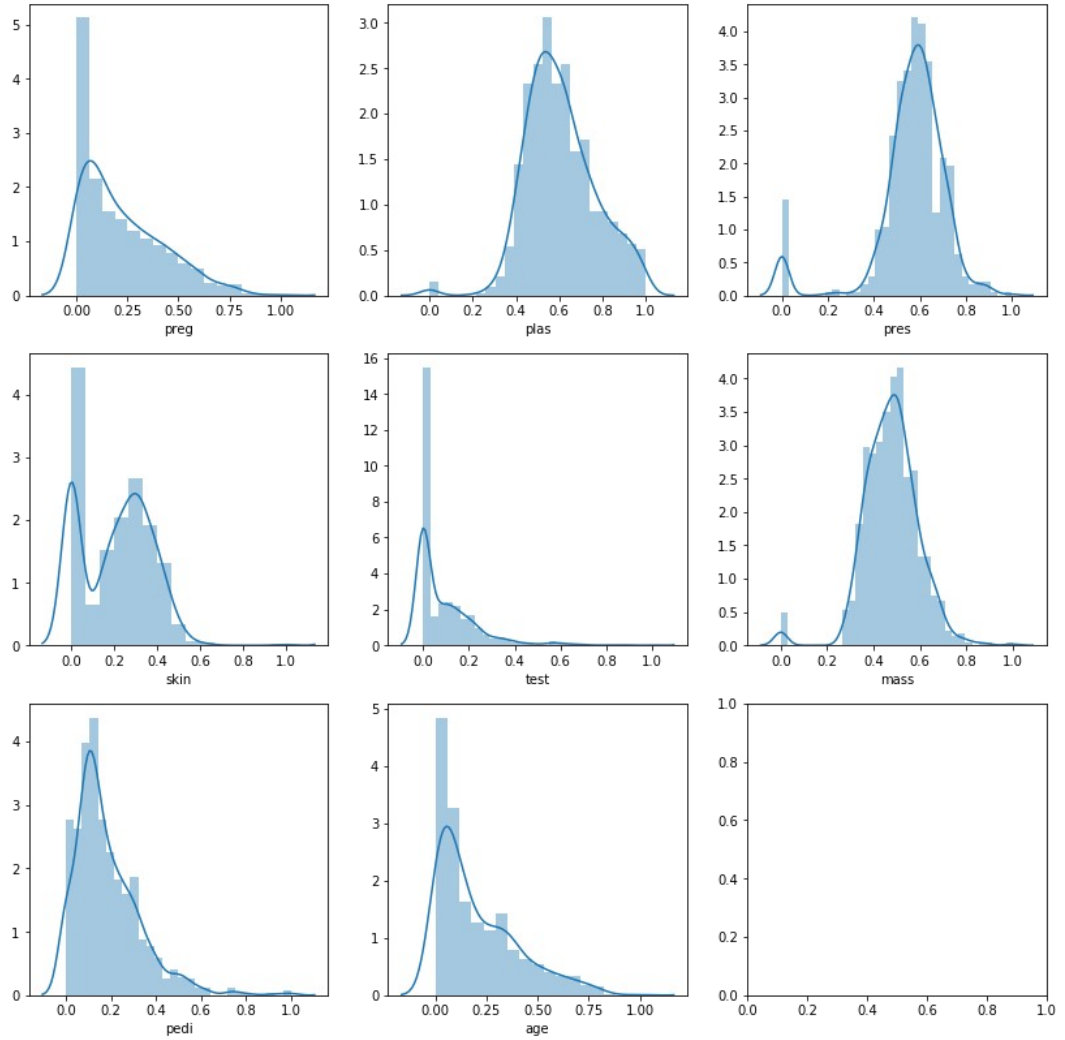
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
np.set_printoptions(precision=3)
print(names)
print(rescaledX[0:5,:])
print(type(rescaledX))
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[[0.353 0.744 0.59  0.354 0.    0.501 0.269 0.483]
 [0.059 0.427 0.541 0.293 0.    0.396 0.151 0.167]
 [0.471 0.92  0.525 0.    0.    0.347 0.289 0.183]
 [0.059 0.447 0.541 0.232 0.111 0.419 0.072 0.   ]
 [0.    0.688 0.328 0.354 0.199 0.642 0.982 0.2  ]]
<class 'numpy.ndarray'>
```

2. Escalamiento (II)



ANTES



DESPUÉS

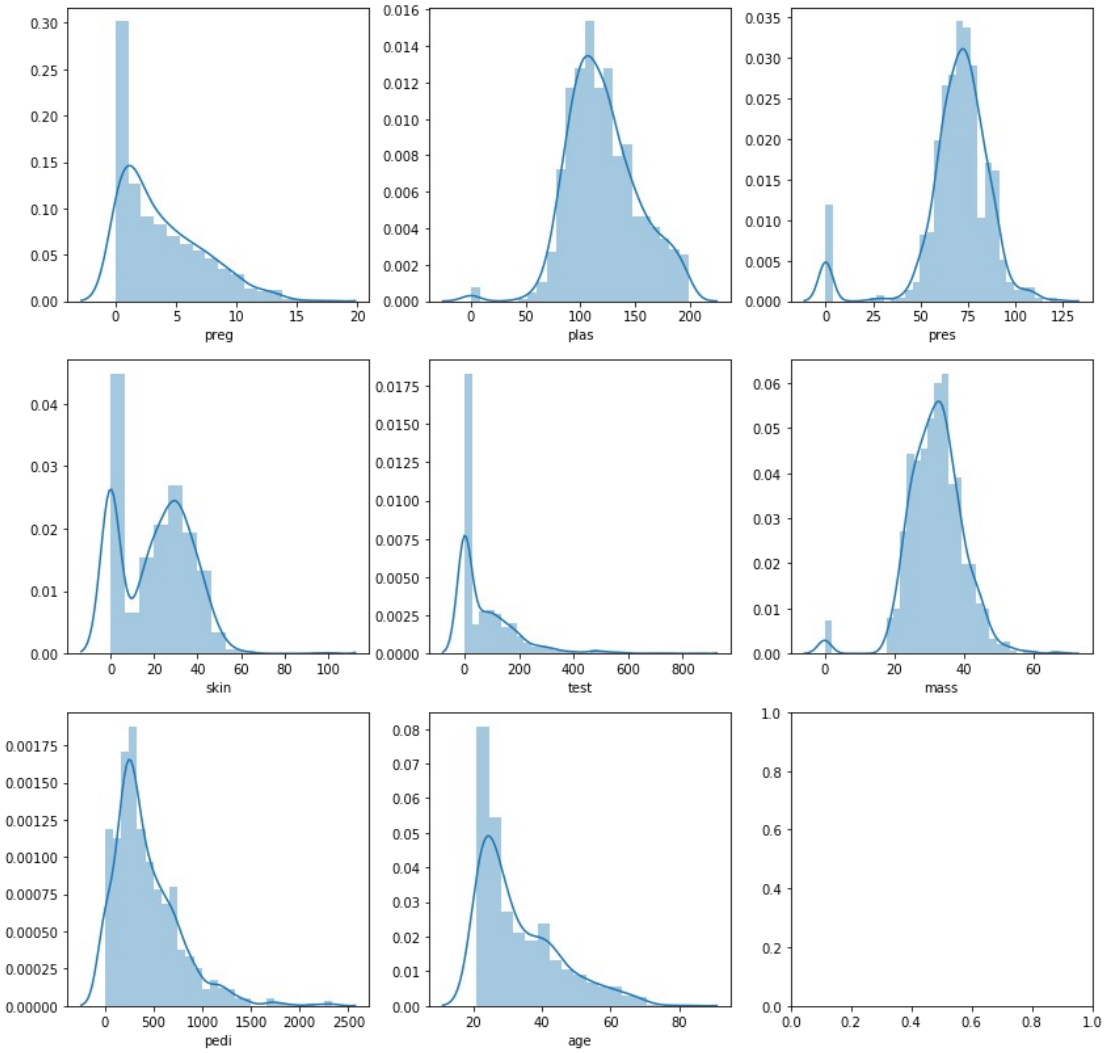
3. Estandarización de datos (I)

- Es más adecuada para técnicas que asumen una distribución gaussiana en las variables de entrada y funcionan mejor con datos reescalados, como LiR, LoR y LDA.
- Puede estandarizar datos utilizando la clase *StandardScaler*.
- Los valores para cada atributo ahora tienen un valor medio de 0 y una desviación estándar de 1.

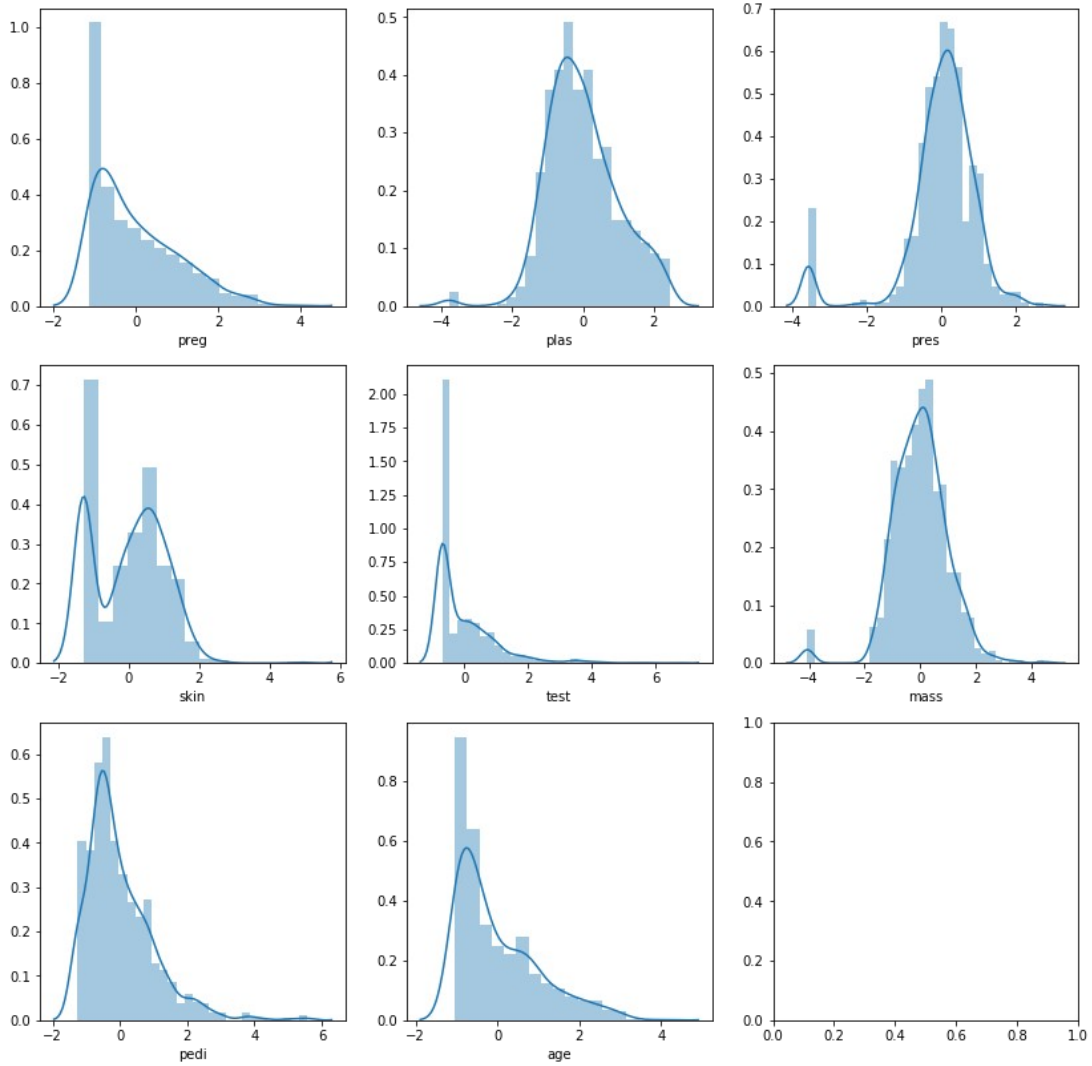
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
np.set_printoptions(precision=3)
print(names)
print(rescaledX[0:5,:])
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.584  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.227 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.716 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.768 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.466 -0.02 ]]
```

3. Estandarización de datos (II)



ANTES



DESPUÉS

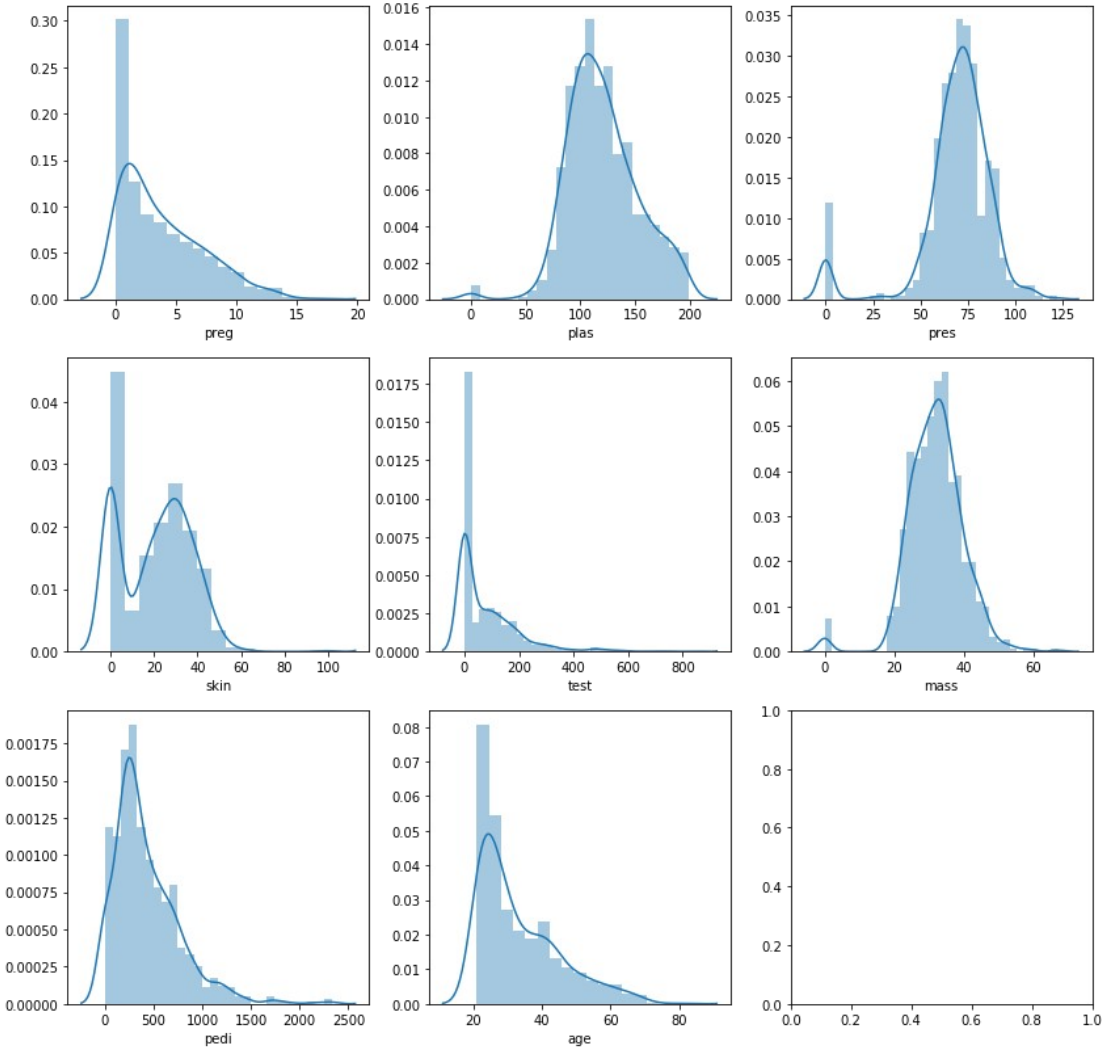
4. Normalización de datos (I)

- Los valores de los datos se pueden escalar en el rango de [0, 1], es decir, con el valor mínimo a 0 y el valor máximo a 1.
- Este método de preprocesamiento puede ser útil para conjuntos de datos dispersos (muchos ceros) con atributos de escalas variables.
  - Cuando se utilizan algoritmos que ponderan valores de entrada como NN y algoritmos que usan medidas de distancia como k-NN.
- Puede normalizar datos en Python con la clase *Normalizer*.

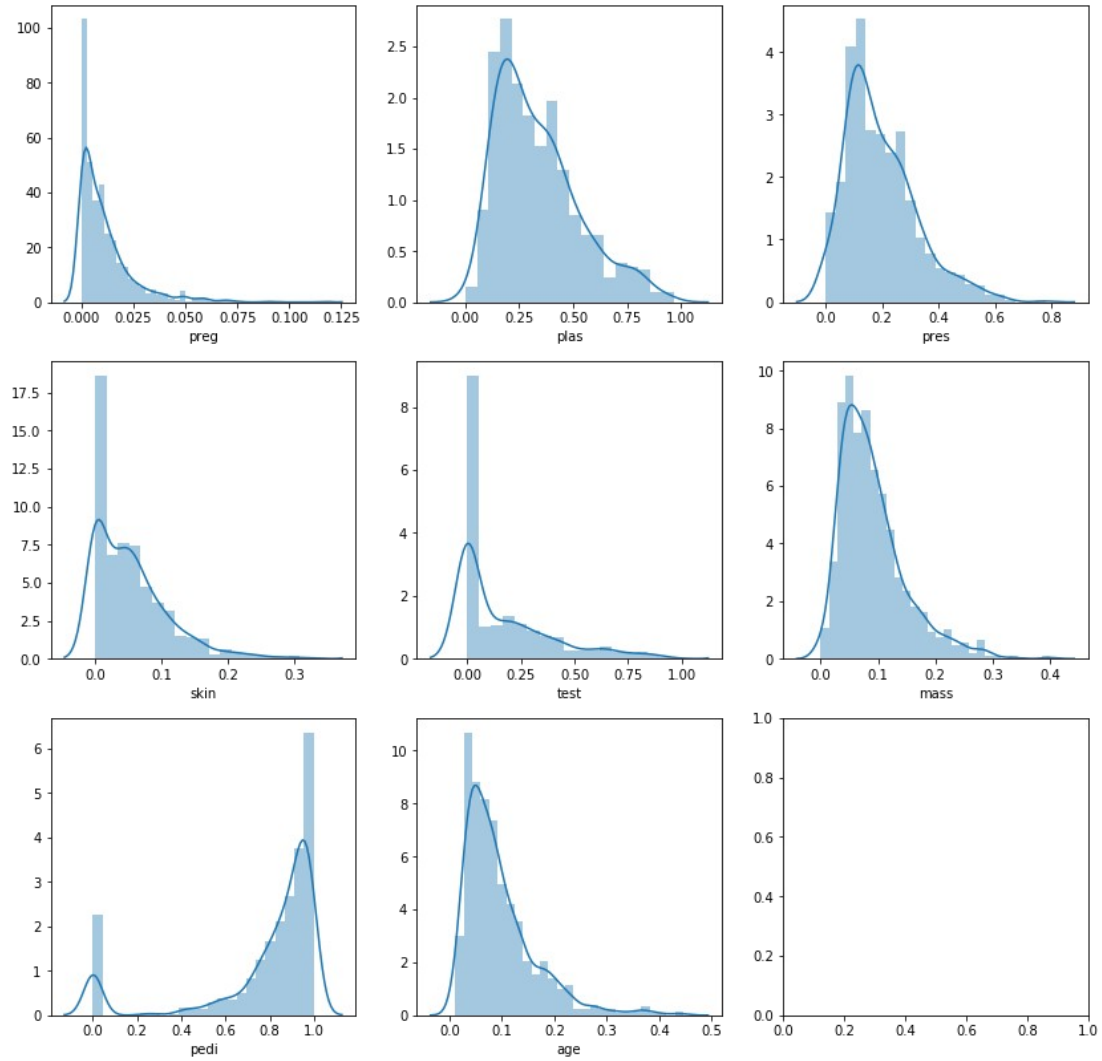
```
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
np.set_printoptions(precision=3)
print(names)
print(normalizedX[0:5,:])
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[[0.009 0.227 0.11  0.054 0.    0.052 0.962 0.077]
 [0.003 0.229 0.178 0.078 0.    0.072 0.947 0.084]
 [0.011 0.261 0.091 0.    0.    0.033 0.959 0.046]
 [0.004 0.395 0.293 0.102 0.417 0.125 0.741 0.093]
 [0.    0.06  0.017 0.015 0.073 0.019 0.995 0.014]]
```

4. Normalización de datos (II)



ANTES



DESPUÉS

5. Binarización

- Puede crear nuevos atributos binarios en Python usando la clase Binarizer
  .
- Puede ver que todos los valores iguales o menores que 0 están marcados con 0 y todos los que están por encima de 0 están marcados con 1.

```
from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
np.set_printoptions(precision=3)
print(names)
print(binaryX[0:5,:])
```

```
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
[[1. 1. 1. 1. 0. 1. 1. 1.]
 [1. 1. 1. 1. 0. 1. 1. 1.]
 [1. 1. 1. 0. 0. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 1. 1. 1. 1. 1. 1. 1.]]
```

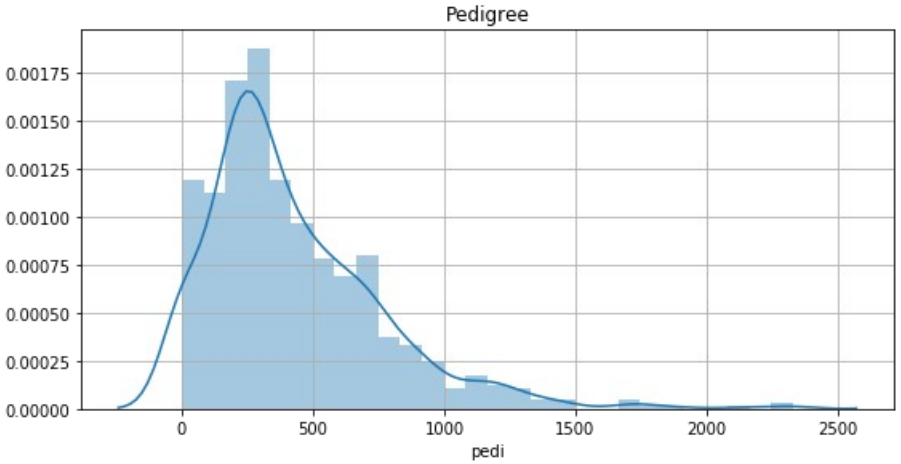
6. Box-Cox (I)

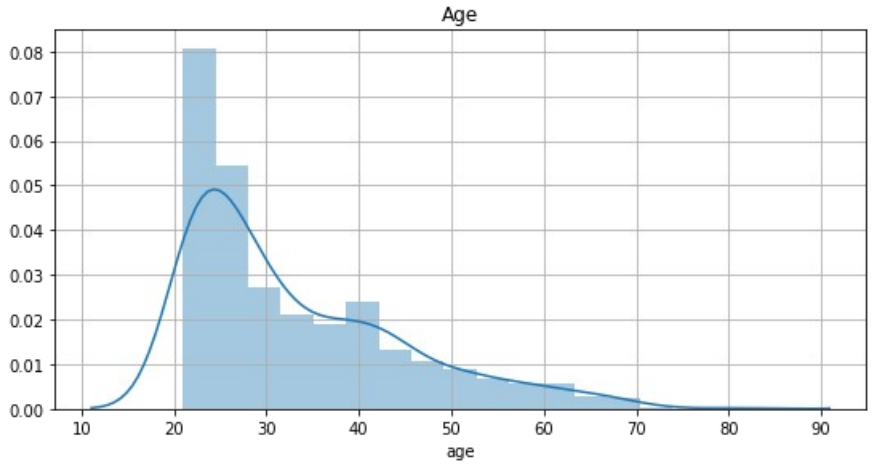
- Los atributos representan un sesgo o inclinación (Gaussiana desplazada).
  - Box-Cox asume todos los atributos positivos.
  - Aplica la transformación a los atributos que parecen tener sesgo.
  - Corrige la no linealidad en la relación (mejorar correlación entre las variables).

```
from sklearn.preprocessing import PowerTransformer
# extract features with skew
features = data[['pedi', 'age']]
#instatiate
pt = PowerTransformer(method='box-cox', standardize=True,)
#Fit the data to the powertransformer
skl_boxcox = pt.fit(features)
#Lets get the Lambdas that were found
#print (skl_boxcox.lambdas_)
calc_lambdas = skl_boxcox.lambdas_
#Transform the data
skl_boxcox = pt.transform(features)
#Pass the transformed data into a new dataframe
df_features = pd.DataFrame(data=skl_boxcox, columns=['pedi', 'age'])
# Pass to the original dataframe the transform columns
data.drop(['age'], axis=1, inplace=True)
data.drop(['pedi'], axis=1, inplace=True)
# Concatenar ambos dataframes
df_data = pd.concat([data, df_features], axis=1)
cols = df_data.columns.tolist()
# Pasa e útimo elemento al primero de la lista (2 veces)
cols = cols[-1:] + cols[:-1]
cols = cols[-1:] + cols[:-1]
# Sobreescrimos
df_data = df_data[cols]
df_data
```

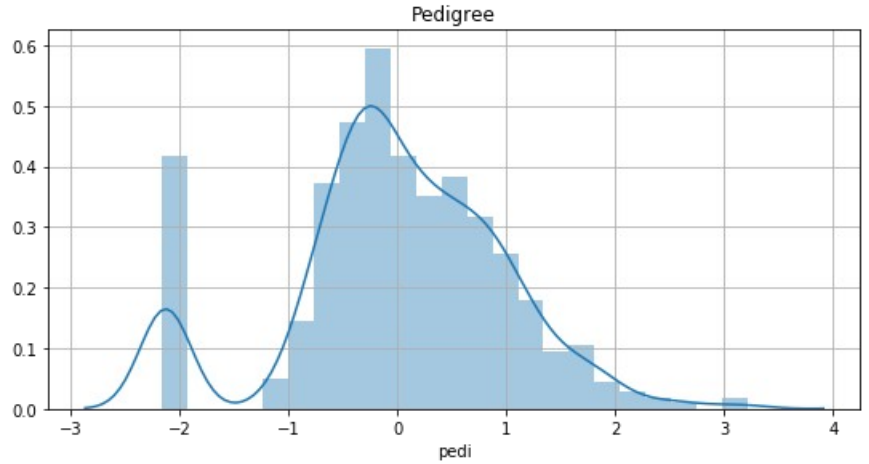
6. Box-Cox (II)

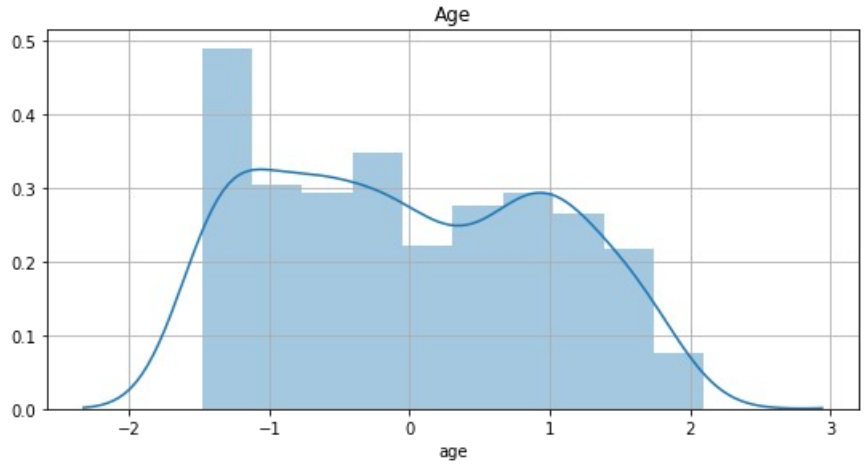
ANTES





DESPUÉS





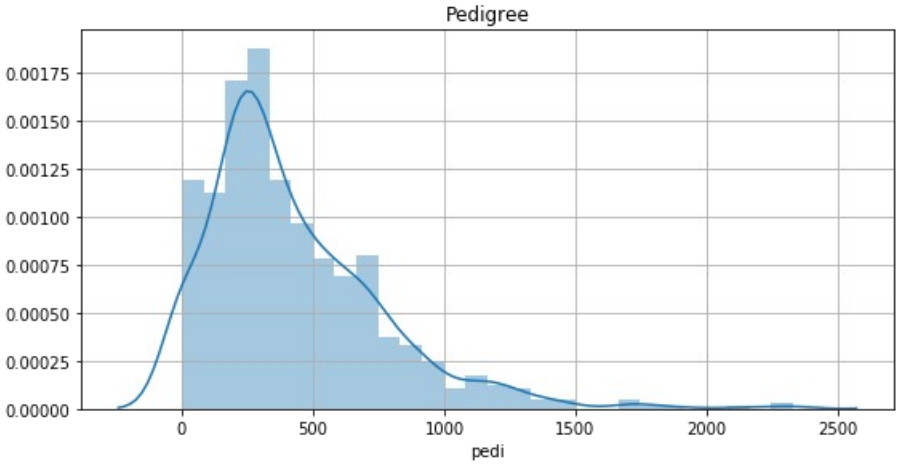
7. Yeo-Johnson (I)

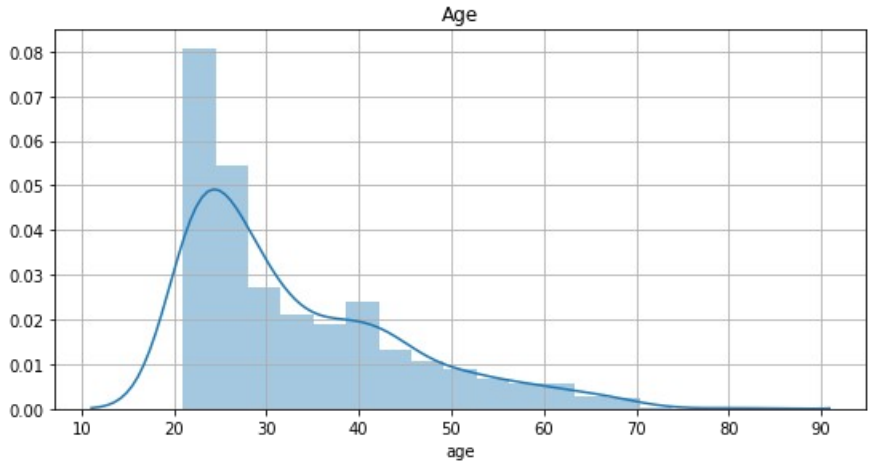
- Igual que Box-Cox pero soporta valores en bruto que son iguales a cero y negativos.

```
# extract features with skew
features = data[['pedi', 'age']]
#instatiate
pt = PowerTransformer(method='yeo-johnson', standardize=True,)
#Fit the data to the powertransformer
skl_boxcox = pt.fit(features)
#Lets get the Lambdas that were found
#print (skl_boxcox.lambdas_)
calc_lambdas = skl_boxcox.lambdas_
#Transform the data
skl_boxcox = pt.transform(features)
#Pass the transformed data into a new dataframe
df_features = pd.DataFrame(data=skl_boxcox, columns=['pedi', 'age'])

# Pass to the original dataframe the transform columns
data.drop(['age'], axis=1, inplace=True)
data.drop(['pedi'], axis=1, inplace=True)
# Concatenar ambos dataframes
df_data = pd.concat([data, df_features], axis=1)
cols = df_data.columns.tolist()
```

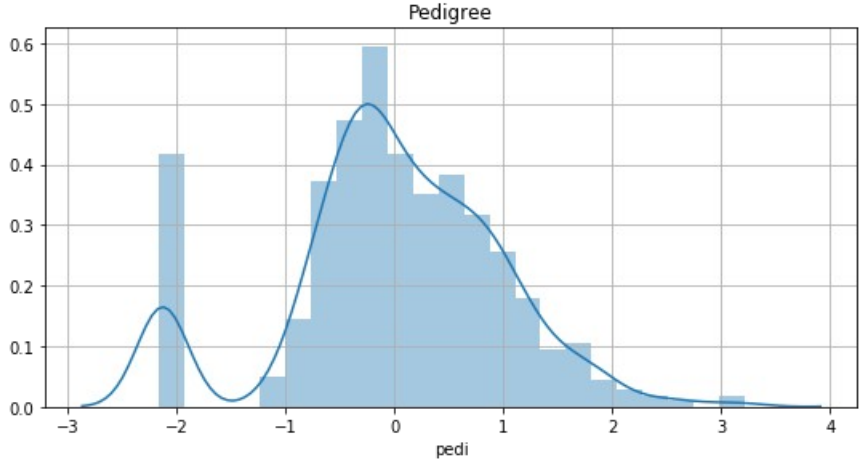
7. Yeo-Johnson (II)

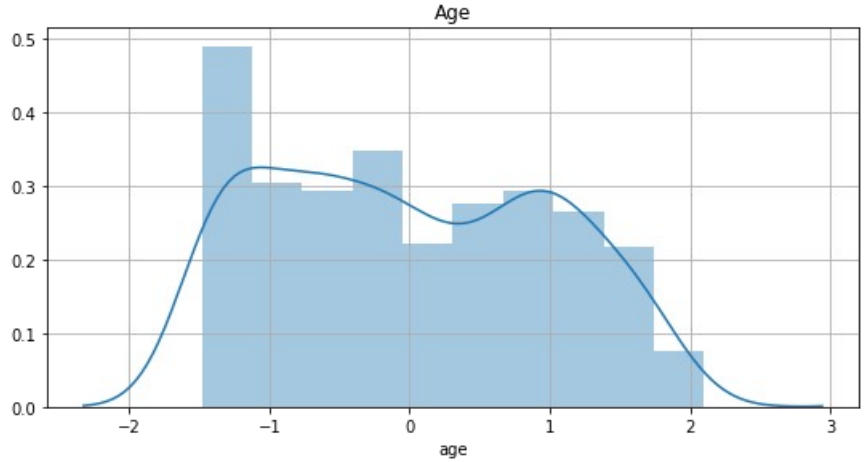
ANTES

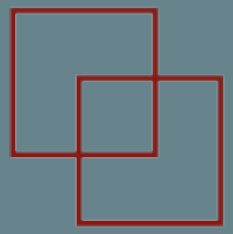




DESPUÉS







Smart City

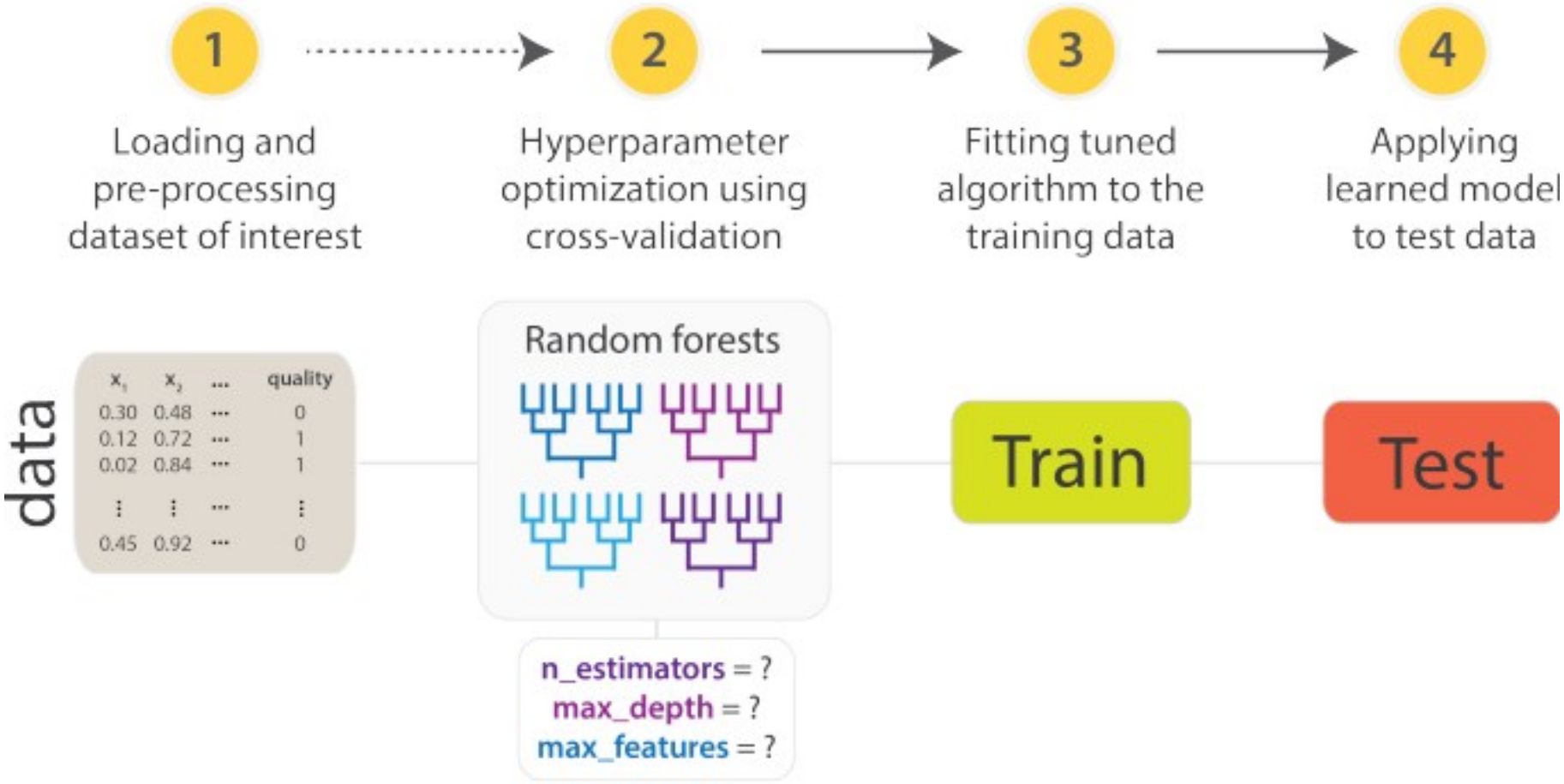
LAB CTIC UNI

Métodos de remuestreo

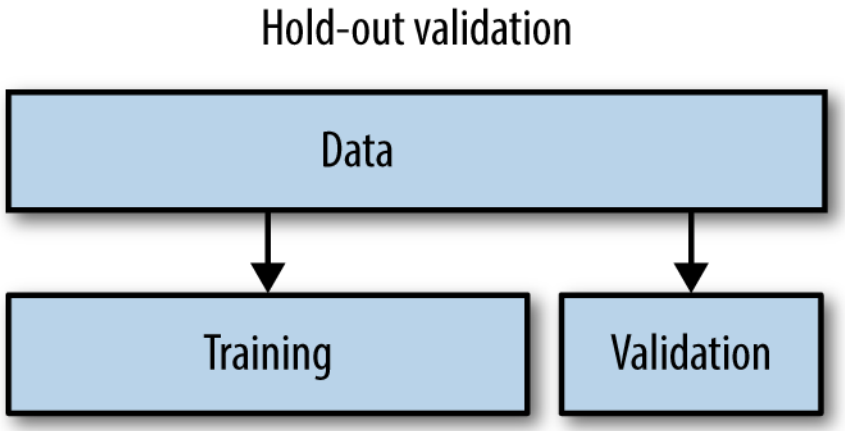
1. Introducción

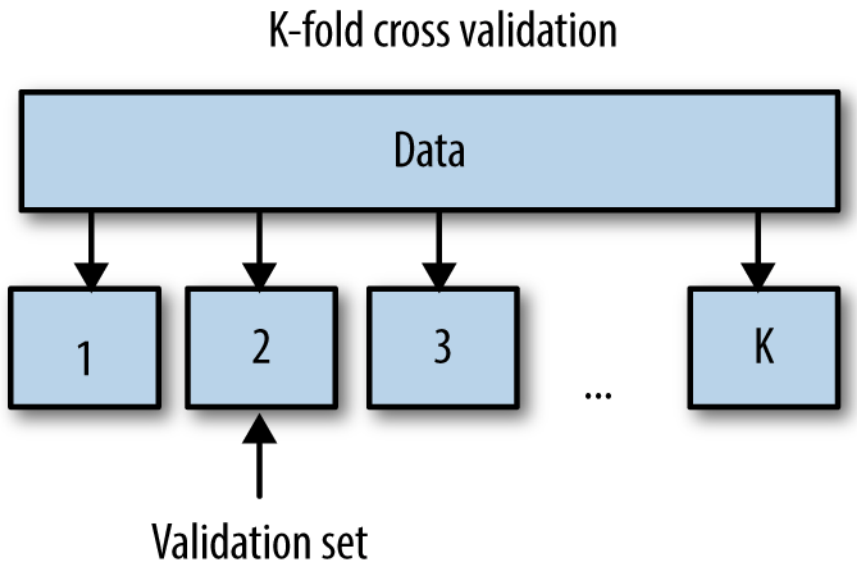
- Objetivo: Evaluar los algoritmos.
- Dividir los datos para la evaluación.
- Cuatro enfoques de división:
  - Cómo dividir un conjunto de datos en subconjuntos por porcentaje para entrenamiento/validación.
  - Cómo evaluar la robustez del modelo utilizando la validación cruzada, k-fold, con y sin repeticiones.
  - Cómo evaluar la robustez del modelo usando una validación cruzada dejando uno fuera (LOOCV).
  - División en train/test repetidos aleatoriamente.

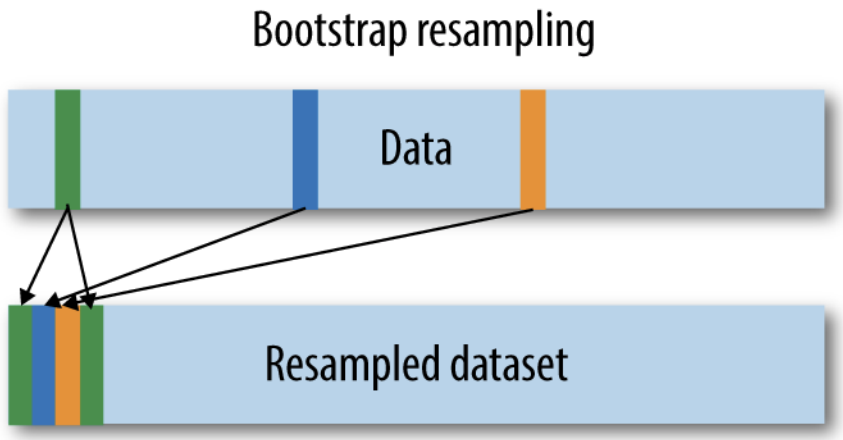
2. Estimar el Accuracy



3. Técnicas para remuestreo

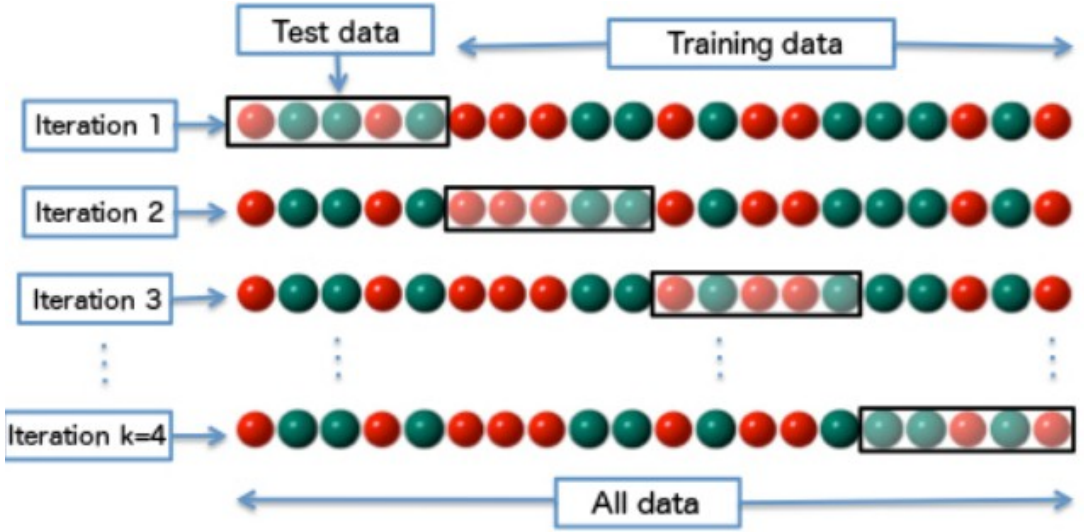






3.1. Validación cruzada

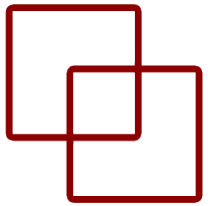
- División en k subconjuntos de datos, es decir, k-fold.
- Los valores comunes para k son de 5 y 10.




```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.60% (5.16%)
```

3.2. Validación cruzada con repeticiones



- Igual que validación cruzada pero en vez de hacer una ejecución hacemos x repeticiones obteniendo la salida media de todas ellas.

```
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

seed = 7
num_folds = 10
num_repeated = 5
repeatedcv = RepeatedKFold(n_splits=num_folds, n_repeats= num_repeated, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
results = cross_val_score(model, X, Y, cv=repeatedcv)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.56% (4.22%)
```

3.3. Validación cruzada dejando uno fuera

- Se omite una instancia de datos y se construye un modelo en todas las demás instancias de datos en el conjunto de entrenamiento, repitiéndose este proceso para todas las instancias de datos.


```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

loocv = LeaveOneOut()
model = LogisticRegression(solver='lbfgs', max_iter=1000)
results = cross_val_score(model, X, Y, cv=loocv)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.60% (41.69%)
```

3.4. División por porcentaje

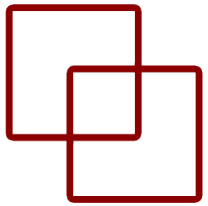
- Realiza una división del conjunto de datos por porcentaje directo, por ejemplo, 67% entrenamiento y 33% validación.

FIGURA 4.2: División de los datos de entrenamiento/validación.

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                    test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print(f"Accuracy: {result*100:,.2f}%")
```

```
Accuracy: 78.74%
```

3.5. División por repetidos aleatoriamente



- Realiza una división del conjunto de datos por porcentaje directo pero repitiendo la evaluación *x* veces.

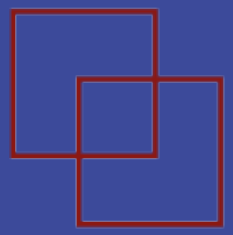
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

n_splits = 10
test_size = 0.33
seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='lbfgs', max_iter=1000)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 76.54% (2.24%)
```

4. Qué técnica usar

- Por lo general, la validación cruzada de *k-fold* es el estándar defacto para evaluar el rendimiento de un algoritmo en datos no etiquetados con *k* configurado en 3, 5 o 10.
- El uso de una división de *train*/*test* es buena para la velocidad cuando se usa un algoritmo lento y produce estimaciones de rendimiento con un sesgo más bajo cuando se usan conjuntos de datos grandes.
- Las técnicas como LOOCV y las divisiones aleatorias repetidas pueden ser intermedios útiles cuando se trata de equilibrar la variación en el rendimiento estimado, la velocidad de entrenamiento del modelo y el tamaño del conjunto de datos.
- En caso de duda, utilice la validación cruzada con un *k-fold* de 10.

¡GRACIAS!



Smart City

LAB CTIC UNI

Dr. Manuel Castillo-Cara
Intelligent Ubiquitous Technologies – Smart Cities (IUT-SCi)
Web: www.smartcityperu.org