**PRÁCTICA INTRO PANDAS + Titanic**

Series

Un Series es un objeto **unidemensional**, similar a un array, una lista o una columna en una tabla, que tiene **asociado una etiqueta** para cada elemento. Por defecto, esta etiqueta es un número de 0 a N.

Se puede crear a partir de un diccionario de datos. Se accede con el método .values.

* Loc: devuelve el valor asociado a una etiqueta
* Iloc: devuelve el valor asociado a una determinada posición.

Filtrado de datos: S[s==3]: estamos seleccionando los valores que sean igual a 3. Tienen asociadas unas operaciones determinadas.

DataFrame

Conjunto tabular de filas y columnas. Columnas tienen nombres que las identifican de manera unívoca.

Generar a partir de diccionario:

d = {'localidad':['Pamplona', 'Pamplona', 'Pamplona', 'Tudela', 'Estella', 'Alsasua'],

'poblacion':[183964, 196166, 195853, 35388, 13702, 7490],

'año': [2001, 2014, 2015, 2015, 2015, 2015]}

data = pd.DataFrame(d)

Se pueden usar funciones determinadas sobre los dataframes, así como operaciones de filtrado sobre sus entradas que poseen. Ordenar por columnas, sumar valores por columnas, obtener media , mediana, etc.

**Merge**: permite unir dos datrafames por una columna común. De lo contrario, se puede concatenar. Se puede también, agrupar resultados por unas determinadas condiciones que imponga el usuario.

**Tablas** **pivotes**: Nos permiten agrupar por dos atributos a la vez. A la tabla pivote resultante, se le puede aplicar una operación simplificar los datos; u obtener algunas métricas de interés.

Sexo F M

Origen

GER 6 7

FRA 5 6  
ESP 3 4

De esta manera, se está agrupando por dos atributos a la vez. Ahora, podríamos realizar suma por filas, y dividr cada valor por eso, obteniendo así el porcentaje.

**Apply**, permite aplicar una función que haya sido definida por el usuario a un dataframe.

**PRÁCTICA IMBD**

Bins = determina el número de grupos. No deja de ser **binning**.

**Pd.cut()** permite discretizar los valores del dataframe. Luego etiquetamos los intervalos creados.

Se puede aplicar operaciones directamente con las columnas de las tablas pivotes. Tabla[‘F’].values – Tabla[‘H’].values = devuelve directamente la diferencia entre columnas.

**PRÁCTICA 1**

Con el método **describe** se puede obtener un informe estadístico de los datos. Como, por ejemplo, q1, q3, etc.

X['Superficie'][X['Superficie']>750]= X['Superficie'][X['Superficie']>750]/(10.7639) : sería una transformación de escala de los datos, algo muy importante.

---------------------------------------------------------- Acceso a elementos del dataframe-------------

numeroAtributos = len(dataframe.columns)

nombreAtributos = dataframe.columns.values

Existe métodos como std o mean, que pueden devolver la media por columnas o filas – o desviación-, si se combina con la palabra reservada *axis*.

>>> from sklearn.preprocessing import StandardScaler

>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]

>>> scaler = StandardScaler()

>>> print(scaler.fit(data))

StandardScaler()

>>> print(scaler.mean\_)

[0.5 0.5]

\*Al hacer fit, calcula internamente la media y la desviación.

\*preprocessing.MinMaxScaler() : clasificador que un atributo tiene el mínimo valor, y en otro, le máximo. Estandariza los datos de esa manera.

**PRÁCTICA 2: Validación de modelos**

-Librería model\_selection, nos permite elegir el mejor modelo dado un conjunto de valores e hiperparámetros.

**Train\_test\_split** : simula el proceso de hold-out.

Una vez que los tenemos, el conjunto de entrenamiento será utilizado para realizar el aprendizaje del clasificador con una configuración dada y cuyo rendimiento será evaluado con los ejemplos del conjunto de validación. La mejor configuración del clasificador será aquella que obtenga el mejor rendimiento con el conjunto de validación.

Para evitar realizar este proceso de forma manual, Scikit-learn ofrece una clase que nos ayuda a realizar dicho proceso. Esta clase también está dentro del paquete **model\_selection** y se llama [GridSearchCV](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html). Su llamada y sus parámetros principales son los siguientes:

El parámetro de salida es un conjunto de clasificadores (todas las posibles combinaciones de los híper-parámetros a utilizar). Al igual que hacemos con un solo clasificador, debemos entrenar todos ellos con los datos de train (llamada al método fit con la variable donde se almacena el conjunto de clasificadores). En este caso, la llamada al entrenamiento ya realiza internamente tanto el entrenamiento como la evaluación del rendimiento de los clasificadores en el conjunto de validación. Por este motivo, tras entrenarlos (llamada a fit), ya se ha calculado internamente la mejor configuración. La mejor configuración está almacenada en el campo best\_params\_ y su rendimiento asociado está almacenado en el campo best\_score\_. Podemos visualizarlos con print

[StratifiedShuffleSplit](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html" \t "_blank) : realiza un hold-out aleatorio, manteniendo los porcentajes de ejemplos que irán a train y test.

**Práctica 3: valores perdidos**

**Dtypes** nos devuelve el tipo de datos que son cada columna.

**isNull**() , nos devuelve el total de valores nulos de una columna.

Drop, nos permite eliminar determinados elementos de un dataframe.

La clase **simpleImputer**(), nos permite definir una estrategia para imputar valores a una sola variable.

**ColumnTransformer**(), permite poner diversas tuplas para imputar valores, dependiendo de las variables objetivo.

Pipe-Line

En esta sección vamos a ver una clases que permite establecer una secuencia de técnicas a aplicar a los datos. Es decir, podemos establecer todas las técnicas de pre-procesamiento a aplicar (y en el orden deseado) y finalizar con una técnica de predicción. Una vez que se ha establecido la secuencia, al realizar el aprendizaje, se aplicará el aprendizaje de cada componente de forma secuencial y por tanto, los componentes aprenderán sobre el resultado dado por los componentes previos de la secuencia (lo mismo pasa al realizar la predicción). De este modo, ahorramos código y minimizamos las opciones de cometer fallos de programación y incurrir en *data leakage* que afecten al resultado obtenido.

La clase que nos ofrece esta posibilidad se llama [*Pipeline*](http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline) y está dentro de la librería *pipeline* de Scikit-learn. La llamada al constructor de esta clase consiste en un conjunto de tuplas del tipo (nombreFase, objeto), cuyo significado es:

* nombreFase: string que establece el nombre de la fase. Por ejemplo 'tipoCodificacion', 'estandarizacion' o 'clasificador'
* objeto: variable en la que se almacena la llamada al constructor de lo que se desee hacer. Por ejemplo ce.TargetEncoder(smoothing=0.0000001), MinMaxScaler() o neighbors.KNeighborsRegressor()

Es decir, si quisiéramos combinar los procesos mencionados anteriormente deberíamos realizar la siguiente llamada:

pipeline = Pipeline([('tipoCodificacion', ce.TargetEncoder(smoothing=0.0000001)), ('estandarizacion', MinMaxScaler()), ('modelo', neighbors.KNeighborsRegressor())])

Hay que destacar que los objetos de todas las fases de la secuencia (excepto de la última) tienen que tener los métodos *fit* y *transform*, para que puedan aprender de los datos y transformarlos en consecuencia. El último objeto debe tener el método *fit*, para aprender de los datos, y el método *predict* para poder realizar nuevas predicciones. Es decir, el último objeto debe ser un modelo de clasificación o regresión.

El objeto combinado, la Pipeline generada, dispone de los siguientes métodos:

* fit: Recibe como parámetros de entrada los datos de entrada (X) y de salida (Y). Cada objeto de cada fase aprende en base a dichos datos.
* predict: Recibe como parámetro de entrada los datos de entrada (X). Realiza la predicción realizando lo siguiente:
  + Primero se aplican las transformaciones de datos por medio de los primeros objetos (llaman a sus respectivas funciones transform)
  + Finalmente, se aplica al objeto de la última fase (clasificador o modelo de regresión) para realizar la predicción correspondiente a los datos de entrada (llamada a su método predict).

**CODIFICAR + ESTANDARIZAR + PREDECIR (SE NECESITA MODELO)**

Se puede combinar un pipeline con grid search, para así, poder obtener los mejores parámetros para el modelo.

**Práctica 4**

min\_samples\_split: determina el número mínimo de ejemplos que debe tener un nodo interno para dividirlo más

min\_samples\_leaf: determina el número mínimo de ejemplos que debe tener un nodo para generar una hoja. Si no llega a este valor no se genera el nodo hoja.