UNIÓN

Librerías:

```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
```

CONCAT:

Une dos o más DataFrame de **forma horizontal** (por los mismos índices, donde no coincidan rellena filas con NaN) **o vertical** (por defecto)

```
df_concat = pd.concat(objs, axis=0, join='outer', ignore_index=False)
```

- objs: lista de DataFrames a unir
- axis: 0 por filas, en vertical. 1 por columnas, en horizontal
- ignore_index: True reestablece el índice

MERGE:

Combina dos DataFrame por una o más **columnas comunes**

```
df_merge = df_left.merge(df_right, how='inner'/'left' on=None/left_on=None, right_on=None)
```

- df_left: primer DataFrame
- df_right: segundo DataFrame
- how: (op)métodos de unión: por defecto inner
  - inner: filas comunes por las columnas de unión. Como inner join en MySQL.
  - left: se conservan todas las filas del DataFrame de la izquierda. Left join
- on: si las columnas de unión se llaman igual
- left_on: columna de unión del primer DataFrame
- right_on: columna de unión del segundo DataFrame

JOIN:

Combina dos DataFrame por las **etiquetas de índice.** El primer DataFrame tiene el índice y el segundo una columna igual que el índice del primero.

```
df_join = df_left.join(df_right, on='nombre', how='tipo_de_join', lsuffix='', rsuffix='')
```

- df_left: DataFrame del que coge índice
- df_right: DF con el que unir, usando columna igual que el índice del primero
- on: columna e índice comunes
- lsuffix: un alias para las columnas del DF de la izquierda
- rsuffix: un alias para las columnas del DF de la derecha

LIMPIEZA

df = df.rename(colums= {‘key’: ‘value’}, index= {‘key’: ‘value’}, inplace=False)

Renombrar columnas

- colums: key nombre actual, value nuevo nombre
- index: (opcional) etiqueta del índice, key actual, value nuevo
- inplace: (op) False nuevo DataFrame por defecto, no poner si se crea nueva variable. True sobreescribe

Modificar valores de una columna **reemplazando** símbolos o espacios:

df[‘columna’] = df[‘columna’].str.replace(‘.’, ‘ ‘)

Modifica en todos los valores de la columna sustituyendo el . por espacio.

Dict comprehension para **unificar nombres** de todas las **columnas** del DataFrame:

1. nuevas_columnas = {columna: columna.lower().replace(".", "") for columna in df.columns}
2. df.rename(columns = nuevas_columnas, inplace = True)

Modifica todas las columnas poniendo sus nombres en minúsculas y sustituye el . por nada, para que el nombre de la columna vaya todo junto.

dataframe.set_index(keys, drop=True, inplace=False)

Establecer una **columna** o columnas **como índice** del DataFrame

- keys: columna o lista de columnas que serán índice
- drop: (op) True por defecto, las columnas utilizadas como índice se eliminarán de DataFrame True sobreescribe
- inplace: (op) False nuevo DataFrame por defecto, no poner si se crea nueva variable. True sobreescribe

Modificar valores de una columna:

Sintaxis:

df ["columna"].str.lower()

Métodos:

.str.lower(): minúsculas

.str.upper(): mayúsculas

.str.capitalize(): primera letra mayúscula

.str.strip(): elimina espacios en blanco del principio y final

.str.split("-"): output con una lista por fila con sus elementos separados donde el –

Para **sobreescribir** con los métodos:
df[["new_columna1", "new_columna2"]] = df[“columna_modificar”].str.split("-", expand=True).get([1, 2]):

- new_columna: columnas que queremos crear
- columna_modificar: columna de la que crear las nuevas
- “-“: elemento donde hacer la separación
- expand=True: sobreescribe el DataFrame
- get[1,2]: nº de índice de los elementos de la columna actual con los que crear las nuevas

Cambiar tipo de valor de una columna.
Modificarlo a tipo **fecha**:
df[‘columna’] = pd.to_datetime(df[‘columna’])

FILTRADO

Operadores de comparación: >, <, >=, <=, ==, !=
Crear una condición y aplicarla a columnas. Se pueden aplicar diferentes condiciones con & o |.

1. Crear condición: condición = df[‘columna’] == ‘x’
2. Aplicar condición df_nuevo = df[condición]
3. df_nuevo es igual pero con la columna de la condición modificada

isin(): seleccionar filas que contienen valores específicos en una columna. Un valor o lista de valores
df[‘columna’].isin(valores)

1. Crear filtro: filtro = [‘valor1’, ‘valor2’]
2. Aplicar filtro df_nuevo = df[df[‘columna’].isin(filtro)]
3. df_nuevo es igual pero con las filas que contienen los valores del filtro

between(): filtrar por un rango
nuevo_df = df[df[‘columna’].between(inicio, fin, inclusive=both/left/right/neither)]

- both: incluye los valores de inicio y fin
- left: incluye inicio pero no fin
- right: incluye fin pero no inicio

- neither: no incluye ni inicio ni fin

Para **filtrar** por **rango** de **fechas**:

1. variables con la fechas inicio = pd.to_datetime('2013-01-01') fin = pd.to_datetime('2013-01-31')
2. filtrar con between df_nuevo = df[df["columna"].between(inicio, fin, inclusive = "both")]

str.contains(): filtrar por palabras. Devuelve un booleano.

df[‘columna’].str.contains(pat, case=True, na=nan, regex=True)

- pat: patrón de texto a buscar
- case: (op) True distingue mayúsculas y minúsculas
- na=nan: (op)
- regex: (op) True se interpreta como regex