**Pivot Tables**

Las tablas dinámicas son una herramienta útil para **agrupar** y aplicar **funciones agregadas** a nuestros datos para obtener información significativa sobre éstos.

***pd.pivot\_table(data, values=****None****, index=****None****, columns=****None****, aggfunc=****'mean'****, fill\_value=****None****, margins=****False****, dropna=****True****, margins\_name=****'All'****, observed=****False****, sort=****True****)***

* **values: columnas (adicionales al index) a agregar a la tabla dinámica (opcional)**

→ si no añadimos este campo, podemos hacer un “count”, “sum”, “mean”, etc, sobre el index seleccionado, PERO si en la tabla principal hay **columnas numéricas**, estas se pueden utilizar como valores en la tabla dinámica (sobre ellas podemos aplicar las mismas funciones “agg”)

* **index: son las columnas por las que vamos a agrupar y sobre las que se aplicarán los cálculos de las funciones agregadas**

→ puede ser una columna, array, dics … EXCEPTO LISTAS, ya que necesitan tener valores (pares KEY-VALUE dónde el **key** sería el **index**).

Si se pasa una matriz, debe tener la misma longitud que los datos.

* **columns: sus valores (o argumentos) pasan a ser el nombre de las columnas sobre las que nos gustaría calcular una estadística para cada valor del index** (se añaden a las columnas predeterminadas

* **aggfunc: la media por defecto → función o lista/dic de funciones a aplicar sobre el index**

Si se pasa la lista de funciones, la tabla dinámica resultante tendrá columnas jerárquicas cuyo nivel superior son los nombres de las funciones. Si se pasa el dict, el KEY es la columna para agregar y el valor es la función.

→ aggfunc = “max”

→ aggfunc = [“max”, “mean”]

→ aggfunc = {“nombre columna1”: “max” , “nombre de la columna2”: “mean”}

* **fill\_values: scalar, default None** Valor con el que reemplazar los valores faltantes

→ EJEMPLO: vehicles.pivot\_table(index=["Vehicle Class"], columns=["Cylinders"], values=["Combined MPG"], **fill\_value=0**)

* **margins: (True, False), default False**

Agregue todas las filas/columnas o no (p. ej., para subtotales/totales generales).

* **dropna: (True, False), default True →** No incluya columnas cuyas entradas sean todas NaN
* **margins\_name: str, default ‘All’**

Nombre de la fila/columna que contendrá los totales cuando los márgenes sean True.

* **observed: (True, False), default False**

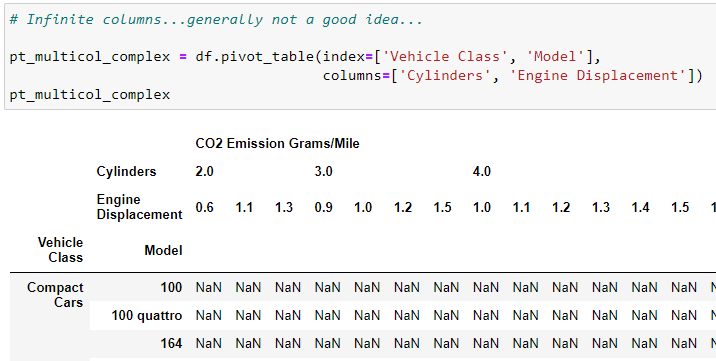
Esto solo aplica si alguno de los datos son Categóricos. Si es verdadero: solo muestra los valores observados. Si es falso: muestra todos los valores.

* **sort: (True, False), default True →** Especifica si se debe ordenar el resultado.

\* pd.options.display.max\_columns = None

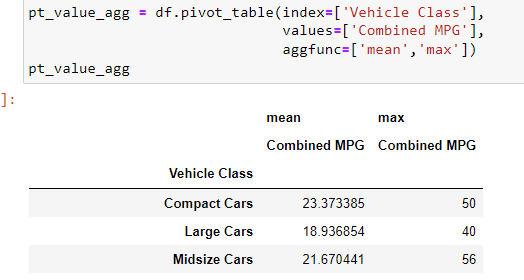
**EJEMPLOS:**

1. MULTICOLUMN → 3 niveles (el predeterminado con todas las columnas numéricas + los dos que añadimos)

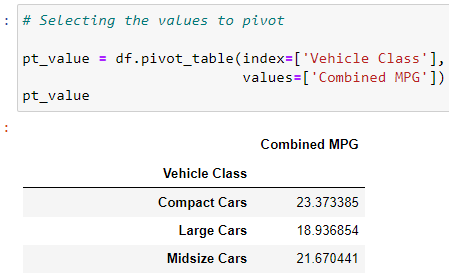
****

***\**** *Si vemos NaN en las celdas quiere decir que no hay filas en el DF original con esta combinación*

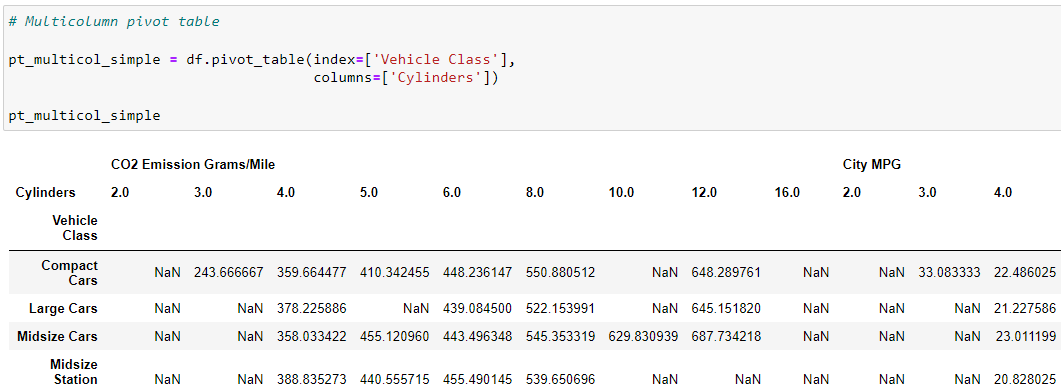
1. 2 funciones agregadas para una columna

****

1. Obtener una única columna gracias al campo “values”

****

**SLICING:**

****

1. **print(pt\_multicol\_simple.columns[0])**

****

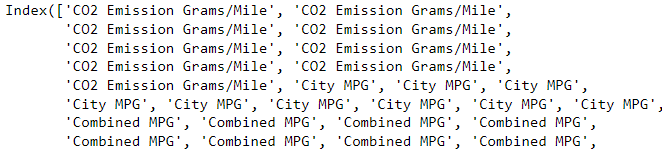
1. **print(pt\_multicol\_simple.columns[0][0])**

****

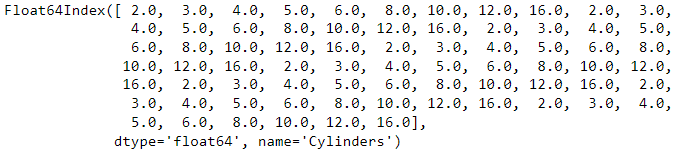
1. **print(pt\_multicol\_simple.columns[0][1])**

****

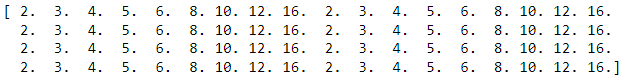
1. **print(pt\_multicol\_simple.columns.get\_level\_values(0))**

****

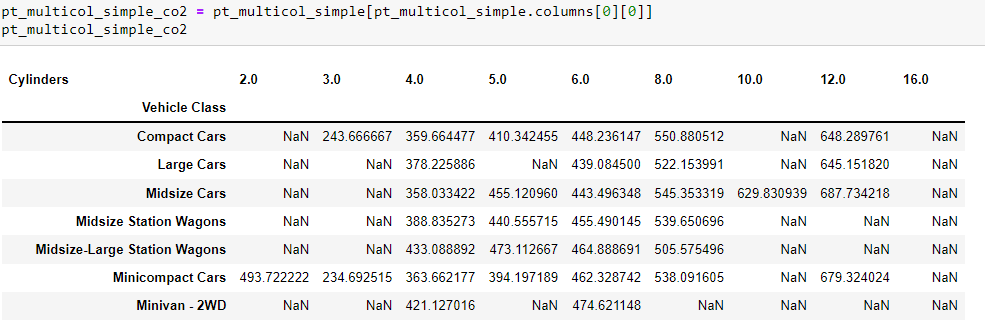
1. **print(pt\_multicol\_simple.columns.get\_level\_values(1))**

****

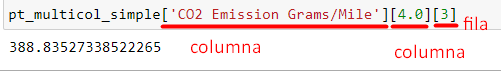
1. **print(pt\_multicol\_simple.columns.get\_level\_values(1).values)**

****

→ OBTENER UNA PARTE DE LA TABLA DINÁMICA A TRAVÉS DE SLICING

****

→ OBTENER UN VALOR CONCRETO



**df*.set\_index*(*keys*, *drop=****True***, *append=****False***, *inplace=****False***, *verify\_integrity=****False***)**

→ Con esa fórmula establecemos el index usando los valores de una/s COLUMNAS. Si usamos ARRAYS los index serán los KEYS.

* **keys:**

→ valores únicos **de una columna**

→ keys **de un array** (que tenga mismo length que el DataFrame)

→ una lista que contenga valores únicos de una columna y keys de un array (aquí “array” encuadra Series, Index, np.ndarray, and instances of Iterator).

* **drop: bool, default True**

Opción que elimina las columnas que van a pasar a ser el index

* **append: bool, default False**

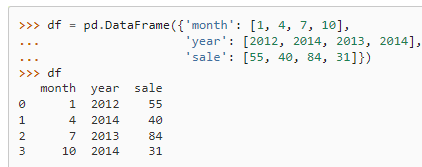
Opción que agrega columnas al index

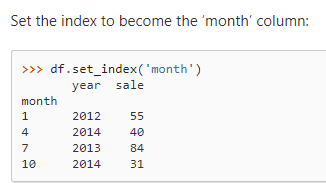
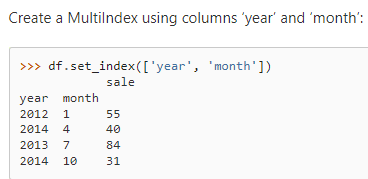
* **inplace: bool, default False**
  + inplace=True → te lo sustituye por el antiguo index
  + inplace=False → crea uno nuevo index y el antiguo lo guarda en una columna
* **verify\_integrity: bool, default False**

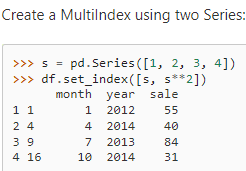
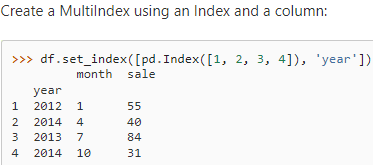
→ True: Comprueba el nuevo índice en busca de duplicados (implica más tiempo de procesado)

| ***See also***  [***DataFrame.reset\_index***](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.reset_index.html#pandas.DataFrame.reset_index)  ***Opposite of set\_index.***  [***DataFrame.reindex***](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.reindex.html#pandas.DataFrame.reindex)  ***Change to new indices or expand indices.***  [***DataFrame.reindex\_like***](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.reindex_like.html#pandas.DataFrame.reindex_like)  ***Change to same indices as other DataFrame.*** |
| --- |

**EJEMPLO:** para el siguiente dataframe vamos a crear nuevos index

**

*** ***

******

* **sort\_values(by=”**nombre columna**”, ascending=False)**

****