

Creación

Métodos comunes para crear DataFrames y atributos habituales

```
pd.DataFrame(dict) # Desde diccionario
pd.read_csv(file) # Desde un csv
pd.read_excel(file) # Desde un excel
pd.read_json(json) # Desde un json
pd.read_html(uri) # Desde una web
pd.read_sql(sql) # Desde una base de datos
pd.read_clipboard() # Desde el portapapeles
pd.read_table(file) # Desde un archivo delimitado tsv
pd.read_parquet() # Desde un archivo en formato parquet
pd.read_gbg
```

Limpieza de datos

Ayudan a identificar datos inválidos

```
pd.isna(obj) # Detecta los valores inválidos en un array
pd.isnull(obj) # Detecta los nulos en un array
pd.notna(obj) # Detecta los valores válidos en un array
pd.notnull(obj) # Detecta todos los valores no nulos en un array
pd.unique(obj) # Devuelve un array con valores únicos. También existe para series (serie.unique()) e índices (index.unique()), y variaciones (nuunique, is_unique)
sr.hasnans # Informa si la serie tiene NA
sd.dropna() # Elimina los valores inválidos
sd.fillna(val) # Rellena los valores inválidos
sd.interpolate() # Interpola los valores según distintos métodos
sd.drop_duplicates() # Elimina los duplicados
sd.duplicated() # Máscara con los duplicados
sr.is_monotonic # Indica si es una progresión creciente/decreciente
sr.nonzero() # Devuelve los índices de los elementos que no son cero
sd.drop() # Elimina las filas o columnas del objeto
```

Operadores lógicos

Operadores lógicos para usar en cualquier expresión booleana

```
& # And
| # Or
~ # Not
^ # Xor
sd.any() # Any
sd.all # All
```

Selección

Selecciona contenido del DataFrame

```
df.iloc[row_indexer,column_indexer] # Selecciona por índices de filas y columnas
df.loc[row_indexer,column_indexer] # Selecciona por etiquetas
df.iat[row,column] # Método análogo a iloc para obtener un valor concreto
df.at[row,column] # Método análogo a loc para obtener un valor concreto
df[] # Permite mezclar las selecciones y realizar filtrados
```

Transformaciones avanzadas

Transformaciones de las Series/DataFrames

```
pd.melt(df) # Descompone un Dataframe, según la columnas que digamos
pd.pivot(index,col,val) # Crea una tabla auxiliar a partir de 3 columnas
pd.pivot_table(df) # Crea una tabla auxiliar con el DataFrame.
Guarda los distintos niveles de la tabla con un índice múltiple.
También disponible en la clase DataFrame
pd.merge(left, right) # Fusiona 2 DataFrames como si fuera un join de base de datos.
También disponible en la clase DataFrame
sd.reindex() # Nos permite utilizar un índice con nuevas etiquetas
sd.join(obj) # Fusiona las columnas de 2 DataFrames en base a una clave/columna
sd.append(to_append) # Añade más columnas al DataFrame
sd.resample(rule) # Permite realizar un remuestro en función del tiempo
pd.concat(obj) # Concatena pandas en el eje que se decida
```

Ordenación

Ordenación de valores e índices

```
sd.sort_values() # Ordena los valores
sd.sort_index() # Ordena el índice
```

Consulta de datos

Obtener información de los datos almacenados

```
sd.sample() # Selecciona filas al azar.
Se pueden indicar % o número de filas/columnas
sd.isin(list) # Devuelve un Dataframe que indica si cada celda contiene alguno de los elementos que se pasan
df.query(expresion) # Permite obtener una parte del Dataframe a partir de una expresión.
Se puede conseguir algo similar con df[]
sd.filter(list|expr) # Filtra las columnas a mostrar.
Se puede utilizar o una lista o una expresión regular
sd.head(n=5) # Obtiene el comienzo del Dataframe.
Se pueden indicar el número de filas.
sd.tail(n=5) # Obtiene el final del Dataframe.
Se puede indicar el número de filas
sd.where(cond) # Es equivalente a df[cond] pero devolviendo un Dataframe con la misma forma que el original
sd.items() # Iterador perezoso de elementos. Equivalente a sd.iteritems()
sr.item() # Devuelve el primer elemento de la Serie
sd.keys() # Columnas del objeto
sd.pop(item) # Elimina un elemento del conjunto y lo devuelve
```

Operadores Binarios

Son operaciones entre 2 Series o DataFrames

```
sd.add(sd) # Suma a nivel de elemento
sd.sub(sd) # Resta a nivel de elemento
sd.mul(sd) # Multiplicación a nivel de elemento
sd.div(sd) # División a nivel de elemento
sd.mod(sd) # Módulo a nivel de elemento
sd.pow(sd) # Potencia a nivel de elemento
sd.combine(sd,func) # Combina 2 objetos aplicando la función a sus elementos
sd.round() # Redondea con el número de decimales que indiquemos
sd.lt() # Operador lógico <
sd.gt() # Operador lógico >
sd.le() # Operador lógico <=
sd.ge() # Operador lógico >=
sd.ne() # Operador lógico !=
sd.eq() # Operador lógico ==
sd.product() # Devuelve el producto de sus valores según el eje que indiquemos
sd.dot(sd) # Devuelve el producto matricial
```

Exportación

Permite exportar los datos a un fichero

```
sd.to_excel() # En formato excel
df.to_csv() # En formato csv
sd.to_dict() # En formato diccionario python
sd.to_json() # En formato json
sd.to_sql(tab, con) # A una base de datos indicando tabla y cadena de conexión
sd.to_string() # En formato cadena de texto
sd.to_clipboard() # Al portapapeles
Series.to_latex() # En formato latex
```

Gráficas

Permiten obtener gráficos de los datos del DataFrame

```
df.plot() # Gráfico
df.hist() # Histograma de los datos
```

Agrupaciones

Permite agrupar los datos y aplicar funciones
También se pueden utilizar cualquiera de las funciones de estadística

```
sd.groupby() # Agrupa datos por un criterio
sd.apply(func) # Aplica una función a los datos sobre el eje que indiquemos
df.applymap(lambda x: x*2) # Aplica una operación a todos los elementos del DataFrame
sd.rolling() # Crea ventanas que se desplazan para el procesamiento de los datos
sd.agg(func) # Agrega los datos aplicando la función del parámetro
sd.transform(func) # Devuelve una serie/Dataframe después de aplicarle la función
sd.pipe(func) # Permite encadenar llamadas a funciones
sd.expanding(periods) # Crea una ventana creciente con los periodos que indiquemos
```

Funciones estadísticas

Funciones que nos permiten calcular la estadística sobre columnas
o DataFrames completos

```
sd.sum() # Calcula la suma total
sd.count() # Calcula el número de elementos
sd.max() # Calcula el valor máximo
sd.min() # Calcula el valor mínimo
sd.std() # Calcula la desviación típica
sd.mean() # Calcula la media
sd.median() # Calcula la mediana
sr.value_counts() # Calcula los valores que hay de cada tipo
sd.abs() # Devuelve una Serie con el valor absoluto de cada elemento
sd.cov(sr) # Calcula la covarianza con otro objeto.
El parámetro es obligatorio en las Series
sd.corr(sr) # Calcula la correlación con otro objeto.
El parámetro es obligatorio sólo en las Series
sd.mad() # Devuelve la media de la desviación absoluta de los valores
sd.nlargest(n,col) # Devuelve los N elementos más altos.
En el DataFrame hay que indicar la columna
sd.nsmallest(n, col) # Devuelve los N elementos más pequeños.
En el DataFrame hay que indicar la columna
sd.pct_change() # Devuelve la Serie con los cambios porcentuales
sd.rank() # Rango de elementos
sd.cumsum() # Suma acumulada
sd.cummax() # Máximo acumulado
sd.cummin() # Mínimo acumulado
sd.cumprod() # Producto acumulado
sd.quantile() # Devuelve un elemento en el percentil indicado
```

Modificación

Permite modificar nuestros elementos

```
sd.rename() # Permite cambiar el nombre o las etiquetas del índice
sd.replace(to_replace) # Reemplaza los valores de panda según el parámetro
sd.update(sd) # Actualiza los valores según el objeto del parámetro
sd.shift() # Desplazamos los valores tantas posiciones como indiquemos (por defecto 1)
```

Metainformación

Nos da información sobre el modelo que estamos manejando

```
sd.index # Etiquetas del índice
sd.values # ndarray con los valores
sr.dtype # Informa del tipo de datos de la Serie. Es equivalente a sr.dtypes
sd.shape # Informa del número de filas de la Serie. También se pueda usar con DataFrame y devuelve filas y columnas
sd.size # Número de elementos
sr.data # Puntero a los datos
sr.name # Nombre de la Serie
sr.put(indices,values) # Efectúa un put sobre los índices con los values suministrados
```

```
# En esta guía utilizaremos 4 posibles notaciones:
# pd : Aplica a la librería de pandas
# df : Aplica únicamente a DataFrames
# sr : Aplica a series (pueden ser series únicas o columnas de un DataFrame)
# sd : Aplica tanto a series como DataFrames
# En todos los ejemplos salvo que se diga lo contrario pondremos únicamente
los atributos obligatorios
```

```
sr1 = pd.Series(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
```

Índice	Valor
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j

```
df1 = pd.DataFrame({'A': range(0, 50, 5),
                    'B': 10**2,
                    'C': sr_a.values},
                    index=pd.date_range(start='2019-01-01', freq="D", periods=10))
```

Índice	A	B	C
2019-01-01	0	100	a
2019-01-02	5	100	b
2019-01-03	10	100	c
2019-01-04	15	100	d
2019-01-05	20	100	e
2019-01-06	25	100	f
2019-01-07	30	100	g
2019-01-08	35	100	h
2019-01-09	40	100	i
2019-01-10	45	100	j

```
df1.iloc[2, 2]
```

Category	Item	Value	Unit
Category 1	Item 1.1	10	kg
	Item 1.2	20	kg
	Item 1.3	30	kg
	Item 1.4	40	kg
	Item 1.5	50	kg
	Item 1.6	60	kg
	Item 1.7	70	kg
	Item 1.8	80	kg
	Item 1.9	90	kg
	Item 1.10	100	kg
Category 2	Item 2.1	10	kg
	Item 2.2	20	kg
	Item 2.3	30	kg
	Item 2.4	40	kg
	Item 2.5	50	kg
	Item 2.6	60	kg
	Item 2.7	70	kg
	Item 2.8	80	kg
	Item 2.9	90	kg
	Item 2.10	100	kg
Category 3	Item 3.1	10	kg
	Item 3.2	20	kg
	Item 3.3	30	kg
	Item 3.4	40	kg
	Item 3.5	50	kg
	Item 3.6	60	kg
	Item 3.7	70	kg
	Item 3.8	80	kg
	Item 3.9	90	kg
	Item 3.10	100	kg
Category 4	Item 4.1	10	kg
	Item 4.2	20	kg
	Item 4.3	30	kg
	Item 4.4	40	kg
	Item 4.5	50	kg
	Item 4.6	60	kg
	Item 4.7	70	kg
	Item 4.8	80	kg
	Item 4.9	90	kg
	Item 4.10	100	kg

```
df1.iat[0, 2]
```

[illegible]

```
df1[df1.C > 'i']
```

[illegible]

```
df1.isin(['a', 'b', 'g', 'h', 'i', 100])
```

[illegible]

```
df1.query("'a' < C < 'd'")
```

[illegible]

```
df1.loc[pd.to_datetime('2018/01/01'), ['B', 'C']]
```

[illegible]

```
df1.at[pd.to_datetime('2018/01/03'), ['C']]
```

[illegible]

```
df1.sample(n=4)
```

[illegible]

```
df1.where(df1.C>'c')
```

[illegible]

```
df1.filter(items=['A', 'C'])
```

[illegible]

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df2.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

df3

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df3.pivot(index='foo', columns='bar', values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6