# Pandas

Unidad 6

Apunte de cátedra
2do Cuatrimestre 2023

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

# Estructura de este apunte

Este apunte fue armado primero como un Google Colab antes de ser pasado a PDF. Esto significa que pueden ir al siguiente link y conseguir una versión más interactiva si así prefieren. Es importante destacar que no van a poder modificar el archivo porque solo tiene permisos de lectura, deben crear una copia primero. Una vez hecha la copia van a poder cambiar el código y probar cosas por su cuenta.

# ¿Qué es Pandas y para qué sirve?

Pandas es una biblioteca de Python que se utiliza para el análisis y la manipulación de datos en bruto. Es una herramienta poderosa que puede ayudar a los usuarios a limpiar, transformar y analizar datos de una manera rápida y eficiente.

La estructura de datos principal en Pandas se llama DataFrame. Es una tabla de datos bidimensional que se compone de filas y columnas, y se asemeja a una hoja de cálculo de Excel. Los DataFrames son útiles para almacenar y manipular grandes cantidades de datos y proporcionan una gran cantidad de funcionalidades, como la selección y filtrado de datos, la agregación de datos y la realización de operaciones matemáticas y estadísticas.

# Poniendo a punto Pandas

**1.** Importamos pandas con el alias `pd`.

```
import pandas as pd
```

**2.** Mostramos la versión de pandas que importamos.

```
pd.__version__
```

*Resultado:*

```
{"type":"string"}
```

**3.** Imaginemos que tenemos el siguiente diccionario `data` y la lista `labels`:

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat',
'dog', 'dog'],
        'age': [2.5, 3, 0.5, None, 5, 2, 4.5, None, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no',
'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

Vamos a crear un DataFrame `df` a partir del diccionario `data` usando `labels` como índice.

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat',
'dog', 'dog'],
        'age': [2.5, 3, 0.5, None, 5, 2, 4.5, None, 7, 3],
```

```
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no',
'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(data, index=labels)
df # Esto lo ponemos para mostrarlo
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  2.0       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
```

**4.** Mostramos información que resuma al DataFrame

`df.info()` nos devuelve información sobre el DataFrame incluidos sus índices y columnas, valores no nulos y uso de memoria

```
df.info()
```

*Resultado:*

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   animal    10 non-null     object
 1   age       8 non-null      float64
 2   visits    10 non-null     int64
 3   priority  10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

Otra forma de obtener algun tipo de información del DataFrame es utilizando la función `describe` la cual devuelve una serie con un resumen descriptivo que incluye el número de datos, su suma, el mínimo, el máximo, la media, la desviación típica y los cuartile

```
df.describe() # Otra forma
```

*Resultado:*

```
            age     visits
count  8.000000  10.000000
mean   3.437500   1.900000
std    2.007797   0.875595
min    0.500000   1.000000
25%    2.375000   1.000000
50%    3.000000   2.000000
75%    4.625000   2.750000
max    7.000000   3.000000
```

**5.** `df.head(n)` Devuelve las primeras `n` filas del DataFrame `df` En el caso de no especificarlo, nos devuelve las primeras 5. Mostramos las primeras 3 filas del DataFrame `df`.

```
df.head(3)
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
```

```
# Otra forma
df.iloc[:3]
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
```

**6.** Si queremos solamente mostrar las columnas 'animal' y 'age' columns del DataFrame `df`.

Vamos a utilizar `df.iloc[filas, columnas]` lo que devuelve un DataFrame con los elementos de las filas de la lista `filas` y las columnas de la lista `columnas`. Si queremos que muestre todas las columnas podemos poner `:` en su lugar.

```
df.loc[:, ['animal', 'age']]
```

*Resultado:*

```
  animal  age
a    cat  2.5
b    cat  3.0
c  snake  0.5
d    dog  NaN
e    dog  5.0
f    cat  2.0
g  snake  4.5
h    cat  NaN
```

```
i    dog  7.0
j    dog  3.0
```

Otra manera puede ser mediante el uso de doble corchete. En este caso no especificamos la cantidad de filas entonces nos muestra todas por defecto.

```
# Otra manera

df[['animal', 'age']]
```

*Resultado:*

```
  animal  age
a    cat  2.5
b    cat  3.0
c  snake  0.5
d    dog  NaN
e    dog  5.0
f    cat  2.0
g  snake  4.5
h    cat  NaN
i    dog  7.0
j    dog  3.0
```

**7.** Ahora seleccionemos las filas `[3, 4, 8]` *y* columnas `['animal', 'age']`.

```
df.loc[df.index[[3, 4, 8]], ['animal', 'age']]
```

*Resultado:*

```
  animal  age
d    dog  NaN
e    dog  5.0
i    dog  7.0
```

**8.** Para filtrar los elementos de nuestra tabla lo que hacemos es poner una condición dentro de los corchetes. Por ejemplo, podemos seleccionar solamente las filas que tengan número de visitas mayores que 3.

```
df[df['visits'] > 3] #No hay ninguno ja
```

*Resultado:*

```
Empty DataFrame
Columns: [animal, age, visits, priority]
Index: []
```

**9.** Seleccionamos las filas que le falten los datos de la edad, es decir que su valor sea `NaN`. En este caso la función `isnull()` se usa como condición que devuelve `True` si el valor de la columna es nulo o NaN.

```
df[df['age'].isnull()]
```

*Resultado:*

```
  animal  age  visits priority
d    dog  NaN       3      yes
h    cat  NaN       1      yes
```

**10.** Podemos poner condiciones más complejas dentro de los corchetes. Seleccionamos las filas donde el animal sea cat *y* la edad es menor a 3.

```
df[(df['animal'] == 'cat') & (df['age'] < 3)]
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
f    cat  2.0       3       no
```

**11.** La función `between(a, b)` elige los elementos que se encuentren en el rango [a,b]. Sería el equivalente de hacer `columna >= a & columna <= b`. Seleccionamos las filas con edad entre 2 y 4 (inclusive).

```
df[df['age'].between(2, 4)]
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
f    cat  2.0       3       no
j    dog  3.0       1       no
```

**14.** Si queremos cambiar el valor de celdas específicas usamos el = como si fuera una variable normal. Cambiemos la edad en la fila 'f' a 1.5.

```
df.loc['f', 'age'] = 1.5
```

**15.** La función `sum()` suma todos los valores de la columna seleccionada. Sumemos la cantidad de visitas de todas las filas de `df` (numero total de visitas).

```
df['visits'].sum()
```

*Resultado:*

```
19
```

**16.** Calculemos la media de las edades de los animales de `df`. En este caso `groupby` agrupa las filas según el animal y con la función `mean` se calcula el promedio de la edad.

```
df.groupby('animal')['age'].mean()
```

*Resultado:*

```
animal
cat      2.333333
dog      5.000000
```

```
snake    2.500000
Name: age, dtype: float64
```

**17.** Agreguemos una nueva fila 'k' a `df` con los valores que queramos para cada columna.

```
df.loc['k'] = ['dog', 5.5, 2, 'no']
df
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  1.5       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
k    dog  5.5       2       no
```

Ahora borremos la columna nueva para volver a tener el DataFrame original. La función `drop` recibe el índice de la fila que se desea eliminar.

```
df = df.drop('k')
df
```

*Resultado:*

```
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  1.5       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no
```

**18.** La función `value_counts()` devuelve cuantas veces aparece cada valor único para una columna específica. Contemos qué cantidad de animales hay en el `df`.

```
df['animal'].value_counts()
```

*Resultado:*

```
cat    4
dog    4
```

```
snake    2
Name: animal, dtype: int64
```

**19.** La función `sort_values(by, ascending)` ordena la tabla con el criterio que pidamos. En el parámetro `by` ponemos una lista con las columnas que queremos usar para ordenar y en `ascending` ponemos una lista de booleanos donde indicamos si queremos que el ordenamiento de la lista anterior sea ascendente o no. Entonces, ordenemos el `df` primero a partir de los valores de edad 'age' en orden *descendente* , luego por el valor de las visitas 'visits' en orden *ascendente* (la fila `i` debería estar primero, y la `d` última).

```
df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

*Resultado:*

```
  animal  age  visits priority
i    dog  7.0       2       no
e    dog  5.0       2       no
g  snake  4.5       1       no
j    dog  3.0       1       no
b    cat  3.0       3      yes
a    cat  2.5       1      yes
f    cat  1.5       3       no
c  snake  0.5       2       no
h    cat  NaN       1      yes
d    dog  NaN       3      yes
```

**20.** Con `map` podemos transformar los valores de columna entera, le pasamos un diccionario con elementos del tipo `{valor_viejo: valor_nuevo}`. La columna 'priority' contiene valores 'yes' y 'no'. Vamos a reemplazarlos por: 'yes' como `True` y 'no' como `False`.

```
df['priority'] = df['priority'].map({'yes': True, 'no': False})
df
```

*Resultado:*

```
  animal  age  visits  priority
a    cat  2.5       1      True
b    cat  3.0       3      True
c  snake  0.5       2     False
d    dog  NaN       3      True
e    dog  5.0       2     False
f    cat  1.5       3     False
g  snake  4.5       1     False
h    cat  NaN       1      True
i    dog  7.0       2     False
j    dog  3.0       1     False
```

**21.** En la columna 'animal' cambiemos 'snake' por 'python'. Podemos usar `map` como antes o podemos usar `replace` que recibe primero el valor viejo, después el valor nuevo, y realizará el reemplazo.

```
df['animal'] = df['animal'].replace('snake', 'python')
df
```

*Resultado:*

```
   animal  age  visits  priority
a     cat  2.5       1      True
b     cat  3.0       3      True
c  python  0.5       2     False
d     dog  NaN       3      True
e     dog  5.0       2     False
f     cat  1.5       3     False
g  python  4.5       1     False
h     cat  NaN       1      True
i     dog  7.0       2     False
j     dog  3.0       1     False
```

# 4. Biblioteca de Funciones de Pandas

Funciones de común utilización en pandas.

| Función | Definición | Ejemplo de Uso |
|---|---|---|
| read_csv() | Lee un archivo CSV y lo carga en un DataFrame | `df = pd.read_csv('data.csv')` |
| head() | Muestra las primeras filas del DataFrame. Se le puede pasar un número para pedirle una N cantidad de filas. | `df.head()`<br>`df.head(7)` |
| tail() | Muestra las últimas filas del | `df.tail()` |

| | | |
|---|---|---|
| | DataFrame | |
| shape | Devuelve la forma (número de filas y columnas) del DataFrame | `shape = df.shape` |
| info() | Muestra información sobre el DataFrame | `df.info()` |
| describe() | Genera estadísticas descriptivas del DataFrame | `df.describe()` |
| columns | Devuelve los nombres de las columnas del DataFrame | `columns = df.columns` |
| dtypes | Devuelve los tipos de datos de las columnas del DataFrame | `dtypes = df.dtypes` |
| dropna() | Elimina filas con valores faltantes del DataFrame | `df.dropna()` |
| fillna() | Rellena los valores faltantes del DataFrame con un valor dado | `df.fillna(0)` |

| | | |
|---|---|---|
| groupby() | Agrupa el DataFrame según una o varias columnas | `grouped = df.groupby('columna')` |
| sum() | Calcula la suma de los valores en el DataFrame | `total = df['columna'].sum()` |
| mean() | Calcula la media de los valores en el DataFrame | `average = df['columna'].mean()` |
| max() | Encuentra el valor máximo en el DataFrame | `max_value = df['columna'].max()` |
| min() | Encuentra el valor mínimo en el DataFrame | `min_value = df['columna'].min()` |
| unique() | Devuelve los valores únicos en una columna del DataFrame | `unique_values = df['columna'].unique()` |
| nunique() | Devuelve el número de valores únicos en una columna del DataFrame | `num_unique = df['columna'].nunique()` |

| | | |
|---|---|---|
| sort_values() | Ordena el DataFrame por una o varias columnas | `df.sort_values('columna')`<br>`df.sort_values(by=['columna1','columna2'], ascending=[True,False])` |
| merge() | Combina dos DataFrames en función de una o varias columnas | `merged_df = pd.merge(df1, df2, on='columna')` |
| to_csv() | Guarda el DataFrame en un archivo CSV | `df.to_csv('output.csv', index=False)` |
| ize() | Devuelve el número total de elementos en el objeto | `total_elements = df.size` |
| count() | Devuelve el número de elementos no nulos en el objeto | `non_null_count = df['columna'].count()` |
| map() | Aplica una función o un diccionario a los elementos del objeto | `mapped_values = df['columna'].map(func)` |
| value_counts() | Devuelve una serie que contiene recuentos de valores únicos | `value_counts = df['columna'].value_counts()` |

| | | |
|---|---|---|
| isnull() | Devuelve una máscara booleana que indica valores nulos | `is_null_mask = df['columna'].isnull()` |
| notnull() | Devuelve una máscara booleana que indica valores no nulos | `not_null_mask = df['columna'].notnull()` |
| reset_index() | Restablece los índices del DataFrame | `df_reset = df.reset_index()` |
| loc[ ] | Permite la indexación y selección de datos por etiquetas. | `df.loc[2]`<br>`df.loc[1:3, 'columna']` |
| iloc[ ] | Permite la indexación y selección de datos por posición. | `df.iloc[2]`<br>`df.iloc[1:3, 0]` |
| get() | Devuelve el valor correspondiente a una clave dada | `value = df.get('columna')` |

## 4.1. loc vs iloc

Ambos son métodos de indexación con los cuales podemos acceder a filas y columnas específicas.

La diferencia es que:

- **loc** se basa en etiquetas, o sea que utiliza las etiquetas de fila y columna para acceder a los datos.
- **iloc** se basa en números, es decir que utiliza las posiciones de los valores para acceder a los datos.

| | padron | nombre | apellido | edad | carrera | mail_fiuba | ingreso_en_pandemia |
|---|---|---|---|---|---|---|---|
| 1 | 109234 | Agustín | Ramírez | 18 | Química | AgustinRamirez | si |
| 2 | 110456 | María | Portillo | 19 | Civil | MariaPortillo | no |
| 3 | 102890 | Gustavo | Feliche | 20 | Mecánica | None | si |
| 4 | 100450 | Federico | Monte | 35 | Informática | FedericoMonte | si |
| 5 | 100430 | Felipe | Michael | 25 | Informática | FelipeMichael | no |
| 6 | 100455 | Manuela | Gomez | 20 | Informática | ManuelaGomez | no |
| 7 | 105689 | Laura | Ramírez | 24 | Sistemas | None | si |

Para el dataframe de la foto, usando **loc** para obtener nombre y edad se ve así:

```
nombres_edades_v2 = df.loc[:, ['nombre', 'edad']]
```

```
nombres_edades_v2
```

Pero usando **iloc** se ve así:

```
nombres_edades_v2 = df.iloc[:, [1,3]]
```

```
nombres_edades_v2
```

Los dos puntos (:) marcan que queremos mostrar todas las filas, pero se puede modificar para mostrar menos. Por ejemplo:

`:10` imprime del índice 0 al 10.