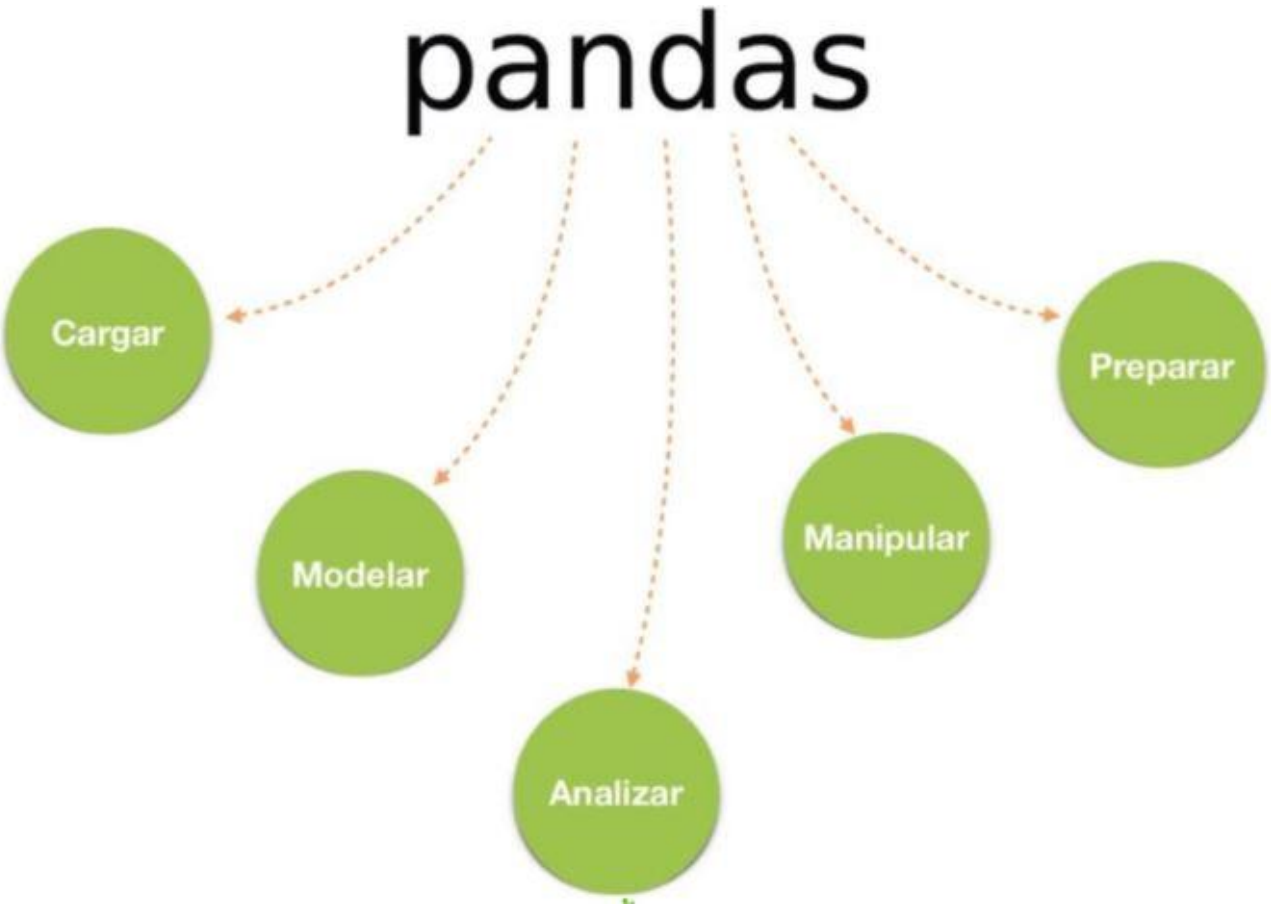
.a) Introducción

Pandas es una de las librerías más utilizadas para ciencia de datos en Python. Es fácil de usar, está basada en NumPy, con la que comparte muchas funciones y propiedades.

Con Pandas podemos leer y extraer de archivos, transformarlos y analizarlos, calcular estadísticas, correlaciones y más.



Para empezar a utilizar pandas, debemos instalar la librería (si estamos trabajando en un entorno local) con `pip install pandas`, y luego importarlo en nuestro código:

```
import pandas as pd
```

pd es un alias habitual cuando importamos esta librería.

El nombre PanDas deriva “panel data”, un término de econometría.

.b) Series y dataframes

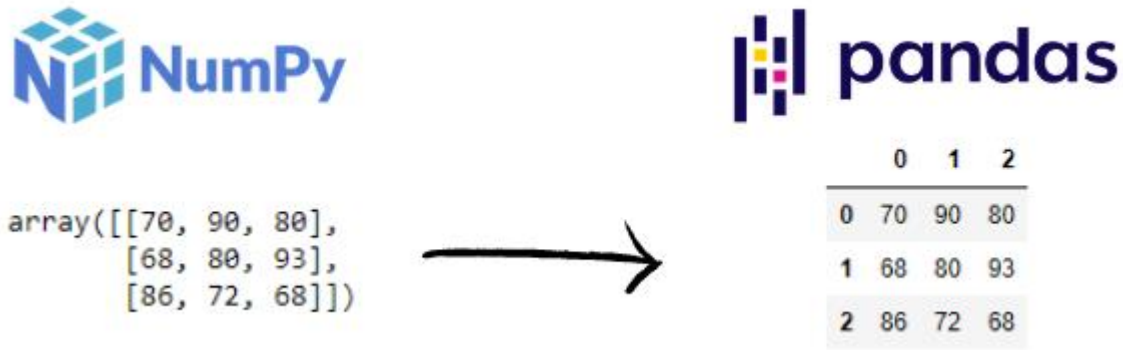
Los dos componentes primartios de pandas son las series y el dataframe.

Una serie es una columna y un dataframes es una tabla multidimensional a partir de una colección de series.

Por ejemplo, el siguiente dataframe esta formado por dos series: edades y alturas.

edades	alturas
18	172
18	180
25	175
43	167
36	170

Podemos pensar en las series como un vector o array unidimensional, mientras que un dataframes es un array multidimensional.



La principal diferencia entre las estructuras en numpy y pandas, es que los dataframes y series es que, además de almacenar los datos, nos proporcionan los 'labels', que nos permiten acceder a los datos a través de los nombres de las columnas y las filas.

.c) Creando un dataframe

Creemos un dataframe con nombres de columnas para explorar sus funciones. La manera más fácil de hacerlo es a partir de un diccionario:

```
data = {
        'edades' : [18, 18, 25, 43, 36],
        'alturas' : [172, 180, 175, 167, 170]
}
```

Cada clave es una columna, mientras que los valores de la lista son los datos de esa columna.

A continuación, debemos pasar el diccionario a dataframe, por medio del constructor de pandas:

```
df = pd.DataFrame(data)
```

El dataframe crea automáticamente un número como índice para cada una de las filas.

```
print(df)
```

```
   edades  alturas
0      18      172
1      18      180
2      25      175
3      43      167
4      36      170
```

Estos índices se pueden modificar al crear el dataframe:

```
df = pd.DataFrame(data, index=[ 'Juan' ,' Ana' ,' Clara' ,' Fabio' ,' Susana' ])
print(df)
```

```
         edades  alturas
Juan         18      172
Ana          18      180
Clara        25      175
Fabio        43      167
Susana       36      170
```

Accedemos a cada fila con la función loc(), que utiliza corchetes para especificar el índice.

```
print(df.loc[ 'Clara' ])
edades      25
alturas    175
Name: Clara, dtype: int64
```

.d) Indexando y slicing

Podemos seleccionar una columna específica con los corchetes. El resultado será un objeto de tipo serie:

```
print(df[ 'edades' ])
```

```
Juan      18
Ana       18
Clara     25
Fabio     43
Susana    36
Name: edades, dtype: int64
```

Si quisiéramos seleccionar varias columnas, podemos especificarlas en una lista. En este caso, el resultado será un dataframe:

```
print(df[['edades','alturas']])
```

```
Juan        18      172
Ana         18      180
Clara       25      175
Fabio       43      167
Susana      36      170
```

Podemos utilizar este método cuando solamente nos interesan algunas de las columnas.

.d.i. Slicing

Pandas utiliza la función iloc() para seleccionar datos basándose en su índice numérico. Funciona de la misma forma que en Python.

```
print(df.iloc[1]) # segunda fila
```

```
edades      18
alturas    180
Name: Ana, dtype: int64
```

```
print(df.iloc[:3]) #primeras 3 filas
```

```
        edades  alturas
Juan        18      172
Ana         18      180
Clara       25      175
```

```
print(df.iloc[1:3]) #filas 2 a 3
```

```
        edades  alturas
Ana         18      180
Clara       25      175
```

iloc() sigue las mismas reglas cque el slicing de listas en Python.

.e) Condicionales

Podemos seleccionar los datos basados en condiciones.

Por ejemplo, selecciones todas las filas con edades mayores a 18 y altura mayor a 170:

```
df2 = df[(df['edades'] > 18) & (df['alturas'] > 170)]
print(df2)
```

```
        edades  alturas
Clara       25      175
```

También podemos utilizar el operador | (or) para combinar condicionales.

Leyendo datos

Es muy común tener los datos en formato planilla o csv.

Pandas permite convertir archivos de estos tipos directamente en dataframes. Por ejemplo:

```
df3 = pd.read_csv('pandas¥estadistica.csv')
print(df3)
```

```
        Nombre    Apellido  Salario
0       Carlyn     Klimkov      256
1   Maximilien       Baily      250
2      Roselle   Schuelcke      252
3         Chas     Blacket      255
4   Batholomew       Dyble      252
..         ...         ...      ...
65     Vidovic      Purdom      316
66   Bernadine  Stonehouse      318
67      Brandy   Snozzwell      322
68         Shay       Doley      325
69     Abagail   Checketts      328

[70 rows x 3 columns]
```

Es importante proporcionar la ruta correcta al archivo para poder abrirlo. Podrás encontrar el archivo de ejemplo con el material de esta unidad.

Pandas también soporta archivos JSON, SQL, xlsx y otros formatos.

Explorando los datos

Podemos obtener las primeras filas de datos con la función head()

```
print(df3.head())
```

```
        Nombre   Apellido  Salario
0       Carlyn    Klimkov      256
1   Maximilien      Baily      250
2      Roselle  Schuelcke      252
3         Chas    Blacket      255
4   Batholomew       Dyble      252
```

Por defecto, devuelve las 5 primeras líneas. Podemos especificar el número de filas que necesitamos con un argumento, por ejemplo: `df3.head(10)`.

De igual manera, podemos obtener las últimas filas con la función tail().

La función info() nos proporciona información esencial sobre nuestro dataset, como número de filas, columnas y tipos de datos.

```
print(df3.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70 entries, 0 to 69
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Nombre    70 non-null     object
 1   Apellido  70 non-null     object
 2   Salario   70 non-null     int64
dtypes: int64(1), object(2)
memory usage: 1.8+ KB
None
```

.f) Creando columnas

Podemos agregar una columna a nuestro dataframe cuyo contenido dependa del valor de otra de las columnas, combinando funciones de pandas y numpy.

Supongamos que queremos agregar una columna adicional, que indique qué empleados recibirán un bono poniendo la palabra 'BONO' si le corresponde, o nada si no le corresponde. La condición es que su salario sea inferior a $280.000.

La funcion where() de numpy permite asignar un valor según la siguiente sintaxis:

np.where(condición, valor por True, valor por False)

Entonces:

```
df3['Bono'] = np.where(df3['Salario'] < 280, 'BONO', '')
print(df3['Bono'])
```

```
0     BONO
1     BONO
2     BONO
3     BONO
4     BONO
      ...
65
66
67
68
69
```

```
Name: Bono, Length: 70, dtype: object
```

.g) Estadística

Podemos obtener un resumen estadístico a partir de las columnas numéricas de nuestro dataset: cantidad de filas, promedio, desviación standard, mínimo, primer, segundo y tercer cuartil y valor máximo.

```
df3.describe()
```

```
            Salario
count    70.000000
mean    282.642857
std      18.553515
min     250.000000
25%     269.250000
50%     281.000000
75%     294.750000
max     328.000000
```

También podemos obtener estas estadísticas especificando al columna:

```
print(df['alturas'].describe())
```

```
count       5.000000
mean      172.800000
std         4.969909
min       167.000000
25%       170.000000
50%       172.000000
75%       175.000000
max       180.000000
Name: alturas, dtype: float64
```

.h) Agrupamiento

Podemos obtener la cantidad de personas que tienen el mismo sueldo con la función de agrupamiento:

```
print(df3.groupby('Bono')['Salario'].count())
```

```
Bono
        36
BONO    34
```

```
# 36 personas no recibirán bono, 34 personas lo recibirán
```

De manera similar, podemos obtener la suma de los salarios, mínimo, maximo, etc. aplicando las diferentes funciones al agrupamiento elegido.