01.python_avanzado

March 2, 2023

1 1. FUNCIONES LIBRERIAS OS EN PYTHON

1.0.1 Los tipos de datos en Python incluyen:

Números: Enteros (int): son números enteros sin parte decimal, como 1, 2, 3, etc.

Punto flotante (float): son números con parte decimal, como 1.0, 2.5, 3.14, etc.

Cadenas (str): son secuencias de caracteres que pueden ser letras, números o cualquier otro tipo de caracteres. Se representan con comillas simples o dobles, como 'Hola' o "Mundo".

Listas (list): son colecciones ordenadas de objetos, que pueden incluir números, cadenas, listas o cualquier otro tipo de datos. Se representan con corchetes [], como [1, 2, 3] o ['Hola', 'Mundo'].

Diccionarios (dict): son colecciones no ordenadas de pares clave-valor, donde la clave y el valor pueden ser cualquier tipo de datos. Se representan con llaves {}, como {'clave1': 'valor1', 'clave2': 'valor2'}.

Tuplas (tuple): son similares a las listas, pero son inmutables, es decir, una vez creadas, no se pueden modificar. Se representan con paréntesis (), como (1, 2, 3) o ('Hola', 'Mundo').

Conjuntos (set): son colecciones no ordenadas y sin elementos repetidos. Se representan con llaves {}, como {1, 2, 3} o {'Hola', 'Mundo'}.

2 1.1. Funciones OS

[1]:
```
import os
import pandas as pd
import numpy as np
```

[2]:
```
print(os.name) # nombre del SO
```

```
posix
```

[3]:
```
print(os.getcwd()) # ruta actual
```

```
/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado
```

[4]:
```
os.chdir("/Users/adrian_gr/Desktop") # cambiamos el directorio
print(os.getcwd()) # ruta actual
```

```
/Users/adrian_gr/Desktop
```

[5]:
```
print(os.listdir("/Users/adrian_gr/Desktop")) # archivos en el escritorio
```

```
['.Rhistory', '..Rapp.history.icloud', '.DS_Store', '3.english', '1.rtp',
'5.bcg', '2.agr', '.gitignore', '.ipynb', '6. ML_Finance', '.ipynb_checkpoints',
'.git', '7.stata_datos', '4.cnmv', '.Rproj.user']
```

[6]:
```
print(os.listdir(os.getcwd())) # una forma mas elegante
```

```
['.Rhistory', '..Rapp.history.icloud', '.DS_Store', '3.english', '1.rtp',
'5.bcg', '2.agr', '.gitignore', '.ipynb', '6. ML_Finance', '.ipynb_checkpoints',
'.git', '7.stata_datos', '4.cnmv', '.Rproj.user']
```

[7]:
```
os.mkdir(".0000prueba_python") # creamos una nueva carpeta y vemos si esta
print(os.listdir(os.getcwd())) # esta
os.rmdir(".0000prueba_python") # vemos que está y le eliminamos
print(os.listdir(os.getcwd())) # ya no esta
```

```
['.Rhistory', '..Rapp.history.icloud', '.DS_Store', '3.english', '1.rtp',
'.0000prueba_python', '5.bcg', '2.agr', '.gitignore', '.ipynb', '6. ML_Finance',
'.ipynb_checkpoints', '.git', '7.stata_datos', '4.cnmv', '.Rproj.user']
['.Rhistory', '..Rapp.history.icloud', '.DS_Store', '3.english', '1.rtp',
'5.bcg', '2.agr', '.gitignore', '.ipynb', '6. ML_Finance', '.ipynb_checkpoints',
'.git', '7.stata_datos', '4.cnmv', '.Rproj.user']
```

[8]:
```
archivo_buscar = "prueba_python.xlsx"
ruta = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado/"

print(os.path.exists(ruta+archivo_buscar))
# nos dice si el archivo especificado esta la tuta dada
```

```
True
```

[9]:
```
directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"␣
 ↪# en que directorio buscar

archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

for archivo in archivos:
    ruta_completa = os.path.join(directorio, archivo)
    if os.path.isfile(ruta_completa) and archivo.startswith("prueba"):
        print(archivo)
```

```
prueba_1.xlsx
prueba_python.xlsx
prueba_3.xlsx
prueba_2.xlsx
```

[10]:
```
directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"␣
 ↪# en que directorio buscar

archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

lista_archivos = []
for archivo in archivos: # lo hacemos ahora sin que nos compruebe primero si␣
 ↪vuelve a estar en el dorectorio
    if archivo.startswith("prueba"):
        lista_archivos.append(archivo)
print(lista_archivos) # ahora me lo guarda en una lista
```

```
['prueba_1.xlsx', 'prueba_python.xlsx', 'prueba_3.xlsx', 'prueba_2.xlsx']
```

[11]:
```
directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"␣
 ↪# en que directorio buscar

archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

lista_archivos = []


for archivo in archivos:
    if "ueba" in archivo and "python" not in archivo:
        lista_archivos.append(archivo)

print(lista_archivos) ## igual pero que contenga en el nombre "ueba" y no la␣
 ↪palabra python
```

```
['prueba_1.xlsx', 'prueba_3.xlsx', 'prueba_2.xlsx']
```

[12]:
```
# Tenemos un archivo que se llama prueva_4 y que no está sancando por erro.␣
 ↪cambiar nombre y listar.

directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"␣
 ↪# en que directorio buscar
archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

for archivo in archivos:
    ruta_completa = os.path.join(directorio, archivo)
    if "prueva" in archivo:
        nuevo_nombre = archivo.replace("prueva", "prueba")
        nueva_ruta = os.path.join(directorio, nuevo_nombre)
        os.rename(ruta_completa, nueva_ruta)

## vemos que ahora saca tambien el prueba_4
```

```
directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"
 ↪# en que directorio buscar

archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

lista_archivos = []
for archivo in archivos:
    if archivo.startswith("prueba") and "python" not in archivo:
        lista_archivos.append(archivo)
print(lista_archivos) # ahora me lo guarda en una lista

os.rename(directorio + "/prueba_4.xlsx", directorio + "/prueva_4.xlsx") # lo
 ↪dejamos igua
```

```
['prueba_1.xlsx', 'prueba_4.xlsx', 'prueba_3.xlsx', 'prueba_2.xlsx']
```

[ ]:

3 1.2. Funciones Read/Write

Son funciones, a priori, para abrir y leer o escribir algunos tipos de archivos. No es tanto usado por Excel, si no cosas de txt, html, json, csv… No es uno de los temas que más vaya a indagar porque no creo que sea tan relevante

[13]:
```
# Crear una lista con algunos datos
data = ["adrian", "david", "carolina"]

# Abrir un archivo de texto para escritura
with open("/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado/
 ↪texto.txt", "w") as file:
    # Escribir los datos en el archivo de texto
    for line in data:
        file.write(line + "\n") # lo de la n se pone para que cada nombre se
 ↪ponga en una linea diferente
```

[14]:
```
ruta = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado/texto.
 ↪txt"


with open(ruta, "r") as f:
    contenido = f.read()
contenido # lo lee mal pq pone las n, pero no voy a darle más vueltas.
```

[14]:
```
'adrian\ndavid\ncarolina\n'
```

4 1.2.bis. Funciones para cargar archivos con pandas

[4]:
```
import pandas as pd
```

[16]:
```
os.chdir("/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado")
```

[17]:
```
# Vamos a abrir uno de los archivos excel y lo vamos a guardar como un csv

df = pd.read_excel("prueba_1.xlsx")

# Guardar el DataFrame como un archivo CSV
df.to_csv("archivo_csv.csv", index=False)
```

[18]:
```
df = pd.read_csv("archivo_csv.csv")
df
```

[18]:

	A	B	C
0	1	a	s
1	2	d	d
2	3	e	y
3	4	t	l

[19]:
```
# Especificar el nombre de las columnas a cargar
columnas = [columna for columna in pd.read_csv("archivo_csv.csv", nrows=0).
 ↪columns if not columna.startswith("B")]

# Cargar solo las columnas especificadas
df2 = pd.read_csv("archivo_csv.csv", usecols=columnas)
df2 ## de esta forma ya no carga la columna B
```

[19]:

	A	C
0	1	s
1	2	d
2	3	y
3	4	l

[20]:
```
directorio = "/Users/adrian_gr/Desktop/4.cnmv/04.practica/01. python_avanzado"␣
 ↪# en que directorio buscar
archivos = os.listdir(directorio) # que me saque la lista de todos los archivos

lista_archivos = []
for archivo in archivos:
    if "ueba" in archivo and "python" not in archivo: # que solo se quede con␣
 ↪los archivos que empiezan por "ueba"
        lista_archivos.append(archivo)
print(lista_archivos) # es la lista que finalmente se genera
```

```
completo = pd.DataFrame() # creamos un df vacio
for pr in lista_archivos:
    pr_n = pr.replace(".xlsx", "") # contruimos una local qeu es el nombre del␣
 ↪archivo sin xlsx
    pr_c = pr_n # pr_c es igual al nombre del archivo
    pr_n = pd.read_excel(pr) # abrimos el archivo
    pr_n['fuente'] = pr_c # hacemos una variable que contenga el nombre de cada␣
 ↪archivo
    completo = pd.concat([completo, pr_n], ignore_index=True) # los vamos␣
 ↪apendeando

completo
```

```
['prueba_1.xlsx', 'prueba_3.xlsx', 'prueba_2.xlsx']
```

[20]:

	A	B	C	fuente
0	1	a	s	prueba_1
1	2	d	d	prueba_1
2	3	e	y	prueba_1
3	4	t	l	prueba_1
4	1	A	s	prueba_3
5	2	A	d	prueba_3
6	3	A	y	prueba_3
7	4	A	l	prueba_3
8	1	a	B	prueba_2
9	2	d	B	prueba_2
10	3	e	B	prueba_2
11	4	t	B	prueba_2

[ ]:

5 1.3. Condicionales

[ ]:
```
if condition1:
    # statements to be executed if condition1 is True
elif condition2:
    # statements to be executed if condition1 is False and condition2 is True
else:
    # statements to be executed if both condition1 and condition2 are False
```

[35]:
```
# vamos a crear un df para hacer diferentes ejemplos
df = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                   'B': [10, 20, 30, 40, 50],
                   'C': [100, 90, 80, 70, 60]})
```

[36]:
```
# Example 1: Simple if-else statement to print a message
if df['A'].mean() > 2.5:
    print("The mean of column A is greater than 2.5")
else:
    print("The mean of column A is less than or equal to 2.5")
```

```
The mean of column A is greater than 2.5
```

[37]:
```
df['A'].mean()
```

[37]:
```
3.0
```

[38]:
```
# Example 2: if-else statement to change values in a column
if df['A'].mean() > 2.5: # como la media es 3, va a multiplicar la fila A por 2
    df['A'] = df['A'] * 2
else:
    df['A'] = df['A'] - 1
df
```

[38]:

	A	B	C
0	2	10	100
1	4	20	90
2	6	30	80
3	8	40	70
4	10	50	60

[39]:
```
# Example 3: Nested if-else statement to categorize values in a column
for i, row in df.iterrows(): # por cada elemento de la columna B, La B pq es la␣
 ↪que se especifica abajo
    if row['B'] > 30:
        df.at[i, 'B'] = "high"
    elif row['B'] > 20:
        df.at[i, 'B'] = "medium"
    else:
        df.at[i, 'B'] = "low"
df
```

[39]:

	A	B	C
0	2	low	100
1	4	low	90
2	6	medium	80
3	8	high	70
4	10	high	60

[40]:
```
# Example 4: Using multiple conditions in an if statement
for i, row in df.iterrows(): # ppr cada elemento de la columna que se␣
 ↪especifique, C en este caso
```

```
    if row['A'] > 3 and row['B'] == "high":
        df.at[i, 'C'] = df.at[i, 'C'] + 10
    elif row['A'] <= 3 and row['B'] == "low":
        df.at[i, 'C'] = df.at[i, 'C'] - 10
df
```

[40]:

	A	B	C
0	2	low	90
1	4	low	90
2	6	medium	80
3	8	high	80
4	10	high	70

[41]:
```
# Example 5: Using if-elif-else statement with multiple conditions
for i, row in df.iterrows():
    if row['A'] > 3 and row['B'] == "high":
        df.at[i, 'C'] = df.at[i, 'C'] + 20
    elif row['A'] <= 3 and row['B'] == "medium":
        df.at[i, 'C'] = df.at[i, 'C'] - 20
    else:
        df.at[i, 'C'] = df.at[i, 'C'] * 2
df
```

[41]:

	A	B	C
0	2	low	180
1	4	low	180
2	6	medium	160
3	8	high	100
4	10	high	90

[ ]:
```

```

6 1.4. Bucles

[42]:
```
# condicional con for
lista = [1, 2, 3, 4, 5]
for numero in lista:
    print(numero)
```

```
1
2
3
4
5
```

[43]:
```
# condicional con while
contador = 1
while contador <= 3:
    print(contador)
    contador = contador + 1
```

```
1
2
3
```

[44]:
```
# definimos df de prueba
# Crear un marco de datos de ejemplo
data = {'Nombre': ['Juan', 'Pedro', 'Maria', 'Ana', 'Luis'],
        'Edad': [25, 30, 28, 35, 40],
        'Pais': ['Argentina', 'Chile', 'Colombia', 'Peru', 'Uruguay']}
df = pd.DataFrame(data)
df
```

[44]:

	Nombre	Edad	Pais
0	Juan	25	Argentina
1	Pedro	30	Chile
2	Maria	28	Colombia
3	Ana	35	Peru
4	Luis	40	Uruguay

[ ]:
```
# Ejemplo 1
```

[45]:
```
for nombre in df['Nombre']:
    print(nombre)
```

```
Juan
Pedro
Maria
Ana
Luis
```

[49]:
```
aa = df['Nombre'].unique()
for  j in aa :
    print(j)
```

```
Juan
Pedro
Maria
Ana
Luis
```

[ ]:
```
# Ejemplo 2
```

[51]:
```
df["Edad"].sum()
```

[51]: 158

[52]:
```
suma_edades = 0
for edad in df['Edad']:
    suma_edades += edad ## que chulo esto como autosuma la propia variable
print("Suma de edades:", suma_edades)
```

```
Suma de edades: 158
```

[53]:
```
suma_edades = 0
for edad in df['Edad']:
    suma_edades = suma_edades + edad
print("Suma de edades:", suma_edades)
```

```
Suma de edades: 158
```

[54]:
```
# ejemplo 3
```

[57]:
```
i = 0
buscado = 'Maria'
encontrado = False
while i < len(df) and not encontrado: # que mire tantas veces como len del df y␣
↪econtrado sea falso. Es decir, en cuanto la encuntre pare
    if df.iloc[i]['Nombre'] == buscado: # si la primera fila de la variable␣
↪nombre es maria
        encontrado = True # si la encuentra encontrado es verdaderp
        print("Encontrado en la fila", i) # que diga en que fila lo ha␣
↪encontrado
    i += 1 # que vaya sumando 1 para no mirar siempre en la misma fila
#fila 2, pa la 1 es 0
```

```
Encontrado en la fila 2
```

[58]:
```
# Ejemplo 4
```

[64]:
```
paises = {} # crea un diccionario
for pais in df['Pais']: # para cada pais diferente en la variable pais
    if pais in paises: # si pais está ya en el diccionario
        paises[pais] += 1 # se mete en el valor de esa clave y le suma 1
    else: # si no esta ya guardado, lo añade le pone el valor 1
        paises[pais] = 1
print("Contador de países:", paises)
```

```
Contador de países: {'Argentina': 1, 'Chile': 1, 'Colombia': 1, 'Peru': 1,
'Uruguay': 1}
```

[65]:
```
#ejemplo 5
```

[66]:
```
filtrado = [] # crea una lista vacia
for i, row in df.iterrows(): # para cada fila
    if row['Edad'] >= 30: # si la edad es 30
        filtrado.append(row) # vaya apendeando la lista filtrado con esa fila␣
↪entera
df_filtrado = pd.DataFrame(filtrado) # convierte filtrado en un df
print("Marco de datos filtrado:")
print(df_filtrado)
```

```
Marco de datos filtrado:
  Nombre  Edad     Pais
1  Pedro    30    Chile
3    Ana    35     Peru
4   Luis    40  Uruguay
```

[77]:
```
# esta seria la forma de hacer un filtrado en python
df[(df["Edad"] >= 30) & (df["Nombre"] == "Ana")]
```

[77]:

	Nombre	Edad	Pais
3	Ana	35	Peru

7 1.4. Funciones

def: es una palabra clave que indica que se está definiendo una función.

nombre_de_la_función: es el nombre de la función, que debe ser único y representar de manera clara y concisa lo que hace la función.

argumentos: son los parámetros que se le pueden pasar a la función. Una función puede no tener argumentos o puede tener uno o más.

El cuerpo de la función: incluido entre los dos puntos y sangrado, contiene el código que se ejecuta cuando se llama a la función.

return es una palabra clave que se utiliza para devolver un resultado desde la función. Una función puede o no devolver un resultado.

[1]:
```
def nombre_de_la_función(argumentos):
    # Código que se ejecuta cuando se llama a la función
    # ...
    return resultado
```

[19]:
```
#Definimos el dataframe
data = {'nombre': ['Juan', 'Pedro', 'Ana', 'Juan', 'Pedro'],
        'edad': [25, 30, 35, 25, 30],
        'profesion': ['Ingeniero', 'Doctor', 'Abogado', 'Ingeniero', 'Doctor']}
df = pd.DataFrame(data)
```

```
df
```

[19]:
```
  nombre  edad  profesion
0   Juan    25  Ingeniero
1  Pedro    30     Doctor
2    Ana    35    Abogado
3   Juan    25  Ingeniero
4  Pedro    30     Doctor
```

[20]:
```
# Ejemplo 1
```

[21]:
```
# FIltrar en base a una condición
def filtro(datos, variable, valor):
    datos_variable_valor = datos[datos[variable] == valor]
    return datos_variable_valor
filtro(df, "profesion", "Ingeniero")
```

[21]:
```
  nombre  edad  profesion
0   Juan    25  Ingeniero
3   Juan    25  Ingeniero
```

[26]:
```
# Ejemplo 2
```

[27]:
```
def agrupar_filas(df, columna):
    return df.groupby(columna).size().reset_index(name='counts') # por la␣
 ↪columna especificada diga cuantas veces se repite cada uno de ellos y lo␣
 ↪ponga en la columna count

# Usar la función para agrupar las filas en el data frame por la columna␣
 ↪'nombre'
filas_agrupadas = agrupar_filas(df, 'nombre')
print("Filas agrupadas por nombre:")
print(filas_agrupadas)
```

```
Filas agrupadas por nombre:
  nombre  counts
0    Ana       1
1   Juan       2
2  Pedro       2
```

[28]:
```
#Ejemplo 3
```

[40]:
```
def agrupar_nombre(df):
    df_agrupado = df.groupby('nombre').mean()
    return df_agrupado
```

[41]:
```
df_agrupado = agrupar_nombre(df)
print("DataFrame agrupado por ciudad y con media de edad por grupo:")
print(df_agrupado)
```

```
DataFrame agrupado por ciudad y con media de edad por grupo:
        edad
nombre
Ana     35.0
Juan    25.0
Pedro   30.0
```

```
<ipython-input-40-d584fa4e8509>:2: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  df_agrupado = df.groupby('nombre').mean()
```

[42]:
```
# Ejemplo 4
```

[51]:
```
# Crear una columna en base a su sueldo y dependiendo de ese sueldo hacer algo␣
↪con la edad
def asignar_sueldo(profesion):
    if profesion == "Ingeniero":
        return 40000
    elif profesion == "Abogado":
        return 50000
    else:
        return 30000

df['sueldo'] = df['profesion'].apply(asignar_sueldo)
df
```

[51]:

	nombre	edad	profesion	sueldo
0	Juan	25	Ingeniero	40000
1	Pedro	30	Doctor	30000
2	Ana	35	Abogado	50000
3	Juan	25	Ingeniero	40000
4	Pedro	30	Doctor	30000

[52]:
```
df.drop("sueldo", axis=1, inplace=True)
df
```

[52]:

	nombre	edad	profesion
0	Juan	25	Ingeniero
1	Pedro	30	Doctor
2	Ana	35	Abogado
3	Juan	25	Ingeniero
4	Pedro	30	Doctor

[55]:
```
## TIPS: casi siempre lo mejor es usa rfunciones numpy y pandas pq es lo que␣
 ↪mas rápido se ejecuta
#para hacer algo similar a lo de arriba pero además cambiar la variable edad:
df['sueldo'] = np.where(df['profesion'] == "Ingeniero", 40000, # si es ingeniro␣
 ↪sueldo vale 40k
                        np.where(df['profesion'] == "Abogado", 50000, 30000)) #␣
 ↪si es abogado 50j, y si no 30K
df['edad'] = np.where(df['sueldo'] > 41000, df['edad'] * 2, df['edad'])
df
```

[55]:

	nombre	edad	profesion	sueldo
0	Juan	25	Ingeniero	40000
1	Pedro	30	Doctor	30000
2	Ana	70	Abogado	50000
3	Juan	25	Ingeniero	40000
4	Pedro	30	Doctor	30000

[ ]:
```
```

8 TEMA 2. NUMPY Y PANDAS

9 2.1 Numpy

Con NUmpy se suelen usar siempre array, más ráìdo de calculo. Las listas son mas utilizadas para cuando queramos meter dentro algo difrente a numero. En los bucles se utiliza mas listas que array

[57]:
```
# Generamos un aary de 1o numeros aleatorio
arr = np.random.rand(10)
arr
```

[57]:
```
array([0.85278903, 0.30769862, 0.59918087, 0.57871608, 0.85398955,
       0.32594768, 0.36436776, 0.59964229, 0.76076124, 0.25155039])
```

[59]:
```
# que me devulva el 8 valor
arr[8]
```

[59]:
```
0.7607612360655607
```

[60]:
```
# que me devuleva del 2 al 5. No coge el 5 como vemos
arr[2:5]
```

[60]:
```
array([0.59918087, 0.57871608, 0.85398955])
```

[61]:
```
# vamos a multiplicar el arr por 10
arr10 = arr * 10
arr10
```

[61]:
```
array([8.52789034, 3.0769862 , 5.99180875, 5.78716076, 8.53989553,
       3.25947675, 3.64367762, 5.99642287, 7.60761236, 2.51550388])
```

[62]:
```
# generamos otro arr y hacemos la resta
arr2 = np.random.rand(10)
arr - arr2
```

[62]:
```
array([ 0.70731461, -0.68006569,  0.35810493,  0.28650027,  0.04866547,
       -0.30052442, -0.49290048,  0.02775098,  0.12753573, -0.47659582])
```

[63]:
```
# calculamos elgunos estidicticos de arr10
print(np.max(arr10))
print(np.std(arr10))
```

```
8.539895531828234
2.1602243089172437
```

[69]:
```
# generamps un array de numero aleatorios menores que 0.5
arr_p = np.random.rand(5) * 0.5
arr_p
```

[69]:
```
array([0.05424562, 0.11342594, 0.21511431, 0.11588669, 0.0548554 ])
```

[71]:
```
# si queremos numeros enteros del 0 al 10
arr_e = np.random.randint(0, 10, size=(5))
arr_e
```

[71]:
```
array([6, 8, 1, 8, 8])
```

[73]:
```
# pero no uqeremos que ninguno se repita
arr_u = np.random.choice(np.arange(0,11), size = 5, replace = 0)
arr_u
```

[73]:
```
array([ 5,  3,  7, 10,  1])
```

[75]:
```
# un array solo de 1
arr1 = np.ones(5)
arr1
```

[75]:
```
array([1., 1., 1., 1., 1.])
```

[78]:
```
# creamos array de diferentes tipos para unirlos

arr_m = np.array([np.nan] * 5)
arr_0 = np.zeros(5)
arr_15 = np.full(5, 15)
arr_todo = np.concatenate((arr_m, arr_0, arr_15))
```

```
arr_todo
```

[78]:
```
array([nan, nan, nan, nan, nan,  0.,  0.,  0.,  0.,  0., 15., 15., 15.,
       15., 15.])
```

[80]:
```
# me quedo con lo que no es missing
mask = ~np.isnan(arr_todo) # virguilla (option+ñ)
arr_todo[mask]
```

[80]:
```
array([ 0.,  0.,  0.,  0.,  0., 15., 15., 15., 15., 15.])
```

[82]:
```
# ahora solo con los valores uperiores a 10
arr_todo[arr_todo > 10]
```

[82]:
```
array([15., 15., 15., 15., 15.])
```

[58]:
```
#Generamos matriz de 3x3 aleatoria
mt = np.random.rand(3,3)
mt
```

[58]:
```
array([[0.11934411, 0.94671774, 0.59080041],
       [0.08189107, 0.58200434, 0.72716797],
       [0.66630335, 0.87692779, 0.20774172]])
```

[3]:
```
# multiplicscción de matrices algebraicas
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])
result = np.dot(matrix_a, matrix_b)
result
```

[3]:
```
array([[19, 22],
       [43, 50]])
```

[6]:
```
# mstrix inversa
matrix_a_inv = np.linalg.inv(matrix_a)
print(matrix_a_inv)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

[7]:
```
# determinante y rango de una matriz
determinante = np.linalg.det(matrix_a)
print("Determinante: ", determinante)

rango = np.linalg.matrix_rank(matrix_a)
print("Rango: ", rango)
```

```
Determinante:  -2.0000000000000004
Rango:  2
```

[ ]:

10 2.2 Pandas

[8]:
```
# Creación a partir de un diccionario
data = {'nombre':['Juan', 'Pedro', 'María'],
        'edad':[30, 25, 28],
        'pais':['España', 'Francia', 'Italia']}
df = pd.DataFrame(data)
```

[10]:
```
# Creación a partir de una lista de diccionarios
data = [{'nombre': 'Juan', 'edad': 30, 'pais': 'España'},
        {'nombre': 'Pedro', 'edad': 25, 'pais': 'Francia'},
        {'nombre': 'María', 'edad': 28, 'pais': 'Italia'}]
df = pd.DataFrame(data)
df
```

[10]:

	nombre	edad	pais
0	Juan	30	España
1	Pedro	25	Francia
2	María	28	Italia

[17]:
```
# Ejemplos de seleeciones
    #- rango
df[0:2]
    #- varias columnas
df[['nombre', 'edad']]
    #-Filtros
df[df['edad'] > 25]
```

[17]:

	nombre	edad	pais
0	Juan	30	España
2	María	28	Italia

[18]:
```
# Tratamiento de variables
```

[19]:
```
# definimos df
data = {'col1': [1, 2, np.nan, 4, 5],
        'col2': [3, 4, 5, np.nan, 7],
        'col3': [7, np.nan, 9, 10, 11]}

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

[19]:
```
   col1  col2  col3
0   1.0   3.0   7.0
1   2.0   4.0   NaN
2   NaN   5.0   9.0
3   4.0   NaN  10.0
4   5.0   7.0  11.0
```

[20]:
```
# eliminar filas de missing
df_without_na = df.dropna() # elimina cualquier fila donde hay al menos un␣
 ↪missing
df_without_na
```

[20]:
```
   col1  col2  col3
0   1.0   3.0   7.0
4   5.0   7.0  11.0
```

[38]:
```
# eliminar filas de missing con un criterio
df = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6],
                   'B': [7, 8, None, 10, None, 12],
                   'C': [None, 14, 15, 16, 17, 18]})
df
```

[38]:
```
   A     B     C
0  1   7.0   NaN
1  2   8.0  14.0
2  3   NaN  15.0
3  4  10.0  16.0
4  5   NaN  17.0
5  6  12.0  18.0
```

[33]:
```
prop_missing = df.isna().mean(axis=1) # calcula la % de missing en cada fila,␣
 ↪axis 1 son las filas
# Elimina las filas con una proporción de valores faltantes mayor al 20%
df_filtered = df[prop_missing <= 0.2].reset_index(drop=True)
df_filtered # solo las filas donde % es menos del 20
```

[33]:
```
   A     B     C
0  2   8.0  14.0
1  4  10.0  16.0
2  6  12.0  18.0
```

[39]:
```
# borramos filas si es missing cierta variable
df.dropna(subset=["C"], inplace=True) # inplace es que el cambio es efectivo
df # al no poner reset_index(drop=True) el indice no se altera, y se ha␣
 ↪eliminado el index 0
```

[39]:
```
   A     B     C
1  2   8.0  14.0
2  3   NaN  15.0
3  4  10.0  16.0
4  5   NaN  17.0
5  6  12.0  18.0
```

[40]:
```
# por si queremos tener el indice desde 0
df.reset_index(drop=True, inplace=True)
df
```

[40]:
```
   A     B     C
0  2   8.0  14.0
1  3   NaN  15.0
2  4  10.0  16.0
3  5   NaN  17.0
4  6  12.0  18.0
```

[21]:
```
# imputacion missing
df.fillna(df.mean(), inplace=True) # pone en el missing el valor medio de la␣
 ↪columna
df
```

[21]:
```
   col1  col2   col3
0   1.0  3.00   7.00
1   2.0  4.00   9.25
2   3.0  5.00   9.00
3   4.0  4.75  10.00
4   5.0  7.00  11.00
```

[ ]:
```

```

10.0.1 2.2.1 GROUPBY

[27]:
```
# Group
data = {'col1': [1, 2, 3, 4, 1],
        'col2': [3, 4, 5, 6, 7],
        'col3': [7, 8, 9, 10, 11]}

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

[27]:
```
   col1  col2  col3
0     1     3     7
1     2     4     8
2     3     5     9
3     4     6    10
```

```
4     1     7    11
```

[28]:
```
# agrupar por col1 y calcular la suma de col2 y col3
grouped = df.groupby(['col1']).sum()
grouped # es el collapse sum de todas las filas por la col1
```

[28]:
```
      col2  col3
col1
1       10    18
2        4     8
3        5     9
4        6    10
```

[2]:
```
# creamos un df para hacer ciertas cosas

nombres = ['Juan', 'Pedro', 'Ana', 'María', 'José']
anios = [2020, 2021, 2022]
ventas = [
  [20, 30, np.nan],
  [np.nan, 50, 60],
  [40, np.nan, 70],
  [10, 20, np.nan],
  [50, np.nan, 30]
]

# Convertimos las listas en un dataframe
df = pd.DataFrame({'nombre': np.repeat(nombres, len(anios)),
                   'ventas': np.concatenate(ventas),
                   'anio': np.tile(anios, len(nombres))})

# Mostramos el resultado
df
```

[2]:
```
    nombre  ventas  anio
0     Juan    20.0  2020
1     Juan    30.0  2021
2     Juan     NaN  2022
3   Pedro      NaN  2020
4   Pedro    50.0  2021
5   Pedro    60.0  2022
6      Ana   40.0  2020
7      Ana    NaN  2021
8      Ana   70.0  2022
9   María    10.0  2020
10  María    20.0  2021
11  María     NaN  2022
12   José    50.0  2020
```

```
13   José     NaN  2021
14   José    30.0  2022
```

[ ]:
```
## ESTO ES VITAL

df['count'] = df['nombre'].map(df['nombre'].value_counts())
df
```

[3]:
```
# sacamos las ventas total por año y nombre
grouped = df.groupby(['nombre', 'anio']).sum().reset_index()
grouped # muy importante el reset index para seguir manteniendo la esrructura␣
 ↪de dataframe
```

[3]:

	nombre	anio	ventas
0	Ana	2020	40.0
1	Ana	2021	0.0
2	Ana	2022	70.0
3	José	2020	50.0
4	José	2021	0.0
5	José	2022	30.0
6	Juan	2020	20.0
7	Juan	2021	30.0
8	Juan	2022	0.0
9	María	2020	10.0
10	María	2021	20.0
11	María	2022	0.0
12	Pedro	2020	0.0
13	Pedro	2021	50.0
14	Pedro	2022	60.0

[4]:
```
grouped['aux'] = "AAA"
grouped
```

[4]:

	nombre	anio	ventas	aux
0	Ana	2020	40.0	AAA
1	Ana	2021	0.0	AAA
2	Ana	2022	70.0	AAA
3	José	2020	50.0	AAA
4	José	2021	0.0	AAA
5	José	2022	30.0	AAA
6	Juan	2020	20.0	AAA
7	Juan	2021	30.0	AAA
8	Juan	2022	0.0	AAA
9	María	2020	10.0	AAA
10	María	2021	20.0	AAA
11	María	2022	0.0	AAA
12	Pedro	2020	0.0	AAA

```
13  Pedro  2021    50.0  AAA
14  Pedro  2022    60.0  AAA
```

[5]:
```
grouped_per = grouped.groupby(['nombre']).sum().reset_index()
grouped_per.drop('anio', axis = 1, inplace = True)
grouped_per # muy importante el reset index para seguir manteniendo la␣
 ↪esrructura de dataframe
```

```
<ipython-input-5-7148c8522b5c>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  grouped_per = grouped.groupby(['nombre']).sum().reset_index()
```

[5]:

	nombre	ventas
0	Ana	110.0
1	José	80.0
2	Juan	50.0
3	María	30.0
4	Pedro	110.0

[16]:
```
grouped = grouped[['nombre', 'ventas']] # el collapse de stata auí hay que␣
 ↪hacerlo en dos pasos
grouped_per = grouped.groupby(['nombre']).sum().reset_index() # de esta forma␣
 ↪ya solo tenemos las columnas deseadas
grouped_per # muy importante el reset index para seguir manteniendo la␣
 ↪esrructura de dataframe
```

[16]:

	nombre	ventas
0	Ana	110.0
1	José	80.0
2	Juan	50.0
3	María	30.0
4	Pedro	110.0

[18]:
```
# si solo queremos quedarnos con las variable numericas
df_numeric = df.select_dtypes(include=[np.number])
df_numeric
```

[18]:

	ventas	anio
0	20.0	2020
1	30.0	2021
2	NaN	2022
3	NaN	2020
4	50.0	2021
5	60.0	2022
6	40.0	2020

```
7      NaN  2021
8     70.0  2022
9     10.0  2020
10    20.0  2021
11     NaN  2022
12    50.0  2020
13     NaN  2021
14    30.0  2022
```

[25]:
```
# vamos a trabajar ahora el nombre de las columnas y sus nombre
```

[26]:
```
# definimos un df tipo
data = {'nombre': ['nombre_1'],
        'empresa_nombre': [None],
        '1_nombre': [None],
        'nombre_1': [None]}
df = pd.DataFrame(data)
df
```

[26]:
```
     nombre empresa_nombre 1_nombre nombre_1
0  nombre_1           None     None     None
```

[27]:
```
# todas las columnas
for j in df.columns:
    print(j)
```

```
nombre
empresa_nombre
1_nombre
nombre_1
```

[28]:
```
# solo las que contienen el nombre en su nombre
cols = [col for col in df.columns if "nombre" in col]
for j in cols:
    print(j)
```

```
nombre
empresa_nombre
1_nombre
nombre_1
```

[29]:
```
# empiezan por nombre
nombre_columns = [col for col in df.columns if col.startswith("nombre")]
for j in nombre_columns:
        print(j)
```

```
nombre
nombre_1
```

[31]:
```
# acaban por nombre
nombre_columns = [col for col in df.columns if col.endswith("nombre")]
for j in nombre_columns:
    print(j)
```

```
nombre
empresa_nombre
1_nombre
```

[32]:
```
# cambiamos el nombre de algunas columnas
for col in df.columns:
    if 'nombre' in col: # si la columna tiene 'nombre'
        new_col = col.replace('nombre', 'Nombre') # el string col, cambien␣
 ↪nombre por Nombre
        df.rename(columns={col: new_col}, inplace=True) # el rename y que sea␣
 ↪permamante
df
```

[32]:

	Nombre	empresa_Nombre	1_Nombre	Nombre_1
0	nombre_1	None	None	None

[ ]:
```

```

2.2.2. MERGE

[102]:
```
import pandas as pd

df1 = pd.DataFrame({'Nombre': ['Juan', 'Carlos', 'Sofia', 'Juan', 'Manuel'],
                    'Empresa': ['Empresa1', 'Empresa2', 'Empresa3', 'Empresa4',␣
 ↪'Empresa5'],
                    'Sueldo': [1000, 2000, 1500, 1700, 1300]})

df2 = pd.DataFrame({'Nombre': ['Juan', 'Carlos', 'Juan', 'Sonia', 'Pedro'],
                    'Colegio': ['Colegio1', 'Colegio2', 'Colegio1', 'Colegio4',␣
 ↪'Colegio5'],
                    'Ciudad': ['Ciudad1', 'Ciudad2', 'Ciudad3', 'Ciudad4',␣
 ↪'Ciudad5']})
df1
```

[102]:

	Nombre	Empresa	Sueldo
0	Juan	Empresa1	1000
1	Carlos	Empresa2	2000
2	Sofia	Empresa3	1500
3	Juan	Empresa4	1700
4	Manuel	Empresa5	1300

[103]:
```
df2
```

[103]:
	Nombre	Colegio	Ciudad
0	Juan	Colegio1	Ciudad1
1	Carlos	Colegio2	Ciudad2
2	Juan	Colegio1	Ciudad3
3	Sonia	Colegio4	Ciudad4
4	Pedro	Colegio5	Ciudad5

[104]:
```
# el merge mas generico, se pierden todos los nombres que no están en ambos df
# ya vemos como se hacen todas las posibilidades de juan
df_merge = df1.merge(df2, on='Nombre')
df_merge
```

[104]:
	Nombre	Empresa	Sueldo	Colegio	Ciudad
0	Juan	Empresa1	1000	Colegio1	Ciudad1
1	Juan	Empresa1	1000	Colegio1	Ciudad3
2	Juan	Empresa4	1700	Colegio1	Ciudad1
3	Juan	Empresa4	1700	Colegio1	Ciudad3
4	Carlos	Empresa2	2000	Colegio2	Ciudad2

[105]:
```
# mantiene todo lo que está en IZQ pero perdemos lo que no solo esta en DER
df_merged = df1.merge(df2, on='Nombre', how='left')
df_merged
```

[105]:
	Nombre	Empresa	Sueldo	Colegio	Ciudad
0	Juan	Empresa1	1000	Colegio1	Ciudad1
1	Juan	Empresa1	1000	Colegio1	Ciudad3
2	Carlos	Empresa2	2000	Colegio2	Ciudad2
3	Sofia	Empresa3	1500	NaN	NaN
4	Juan	Empresa4	1700	Colegio1	Ciudad1
5	Juan	Empresa4	1700	Colegio1	Ciudad3
6	Manuel	Empresa5	1300	NaN	NaN

[106]:
```
# mantiene todo lo que está en IZDERQ pero perdemos lo que no solo esta en IZQ
df_merged = df1.merge(df2, on='Nombre', how='right')
df_merged
```

[106]:
	Nombre	Empresa	Sueldo	Colegio	Ciudad
0	Juan	Empresa1	1000.0	Colegio1	Ciudad1
1	Juan	Empresa4	1700.0	Colegio1	Ciudad1
2	Carlos	Empresa2	2000.0	Colegio2	Ciudad2
3	Juan	Empresa1	1000.0	Colegio1	Ciudad3
4	Juan	Empresa4	1700.0	Colegio1	Ciudad3
5	Sonia	NaN	NaN	Colegio4	Ciudad4
6	Pedro	NaN	NaN	Colegio5	Ciudad5

[107]:
```
# mantiene todo lo que está en IZQ pero perdemos lo que no solo esta en DER
df_merged = df1.merge(df2, on='Nombre', how='outer')
```

```
df_merged
```

[107]:

	Nombre	Empresa	Sueldo	Colegio	Ciudad
0	Juan	Empresa1	1000.0	Colegio1	Ciudad1
1	Juan	Empresa1	1000.0	Colegio1	Ciudad3
2	Juan	Empresa4	1700.0	Colegio1	Ciudad1
3	Juan	Empresa4	1700.0	Colegio1	Ciudad3
4	Carlos	Empresa2	2000.0	Colegio2	Ciudad2
5	Sofia	Empresa3	1500.0	NaN	NaN
6	Manuel	Empresa5	1300.0	NaN	NaN
7	Sonia	NaN	NaN	Colegio4	Ciudad4
8	Pedro	NaN	NaN	Colegio5	Ciudad5

[108]:
```
# vamos a identificar que nombre se repiten en df2
# no vale solo deja un df con esas columasn
df2_t = df2.groupby("Nombre").size().reset_index(name="aux")
df2_t
```

[108]:

	Nombre	aux
0	Carlos	1
1	Juan	2
2	Pedro	1
3	Sonia	1

[109]:
```
# bien pero habria que mirar como hacer el bys nombre colegio
df2['count_N'] = df2.groupby(['Nombre'])['Nombre'].transform('count')
df2['count_C'] = df2.groupby(['Colegio'])['Colegio'].transform('count')
df2
```

[109]:

	Nombre	Colegio	Ciudad	count_N	count_C
0	Juan	Colegio1	Ciudad1	2	2
1	Carlos	Colegio2	Ciudad2	1	1
2	Juan	Colegio1	Ciudad3	2	2
3	Sonia	Colegio4	Ciudad4	1	1
4	Pedro	Colegio5	Ciudad5	1	1

[110]:
```
# forma sencilla, ver si se puede hacer mejor en el futuro
df2['nombre_colegio'] = df2.Nombre + df2.Colegio
df2
```

[110]:

	Nombre	Colegio	Ciudad	count_N	count_C	nombre_colegio
0	Juan	Colegio1	Ciudad1	2	2	JuanColegio1
1	Carlos	Colegio2	Ciudad2	1	1	CarlosColegio2
2	Juan	Colegio1	Ciudad3	2	2	JuanColegio1
3	Sonia	Colegio4	Ciudad4	1	1	SoniaColegio4
4	Pedro	Colegio5	Ciudad5	1	1	PedroColegio5

[111]:
```
# solo vemos lo que se repiten
df2['count_T'] = df2.groupby(['nombre_colegio'])['nombre_colegio'].
 ↪transform('count')
df2[df2['count_T'] > 1]
```

[111]:

	Nombre	Colegio	Ciudad	count_N	count_C	nombre_colegio	count_T
0	Juan	Colegio1	Ciudad1	2	2	JuanColegio1	2
2	Juan	Colegio1	Ciudad3	2	2	JuanColegio1	2

[116]:
```
start_col = df2.columns.get_loc("count_N")
end_col = df2.columns.get_loc("count_T")
columns_to_drop = list(df2.columns[start_col:end_col+1])
df2.drop(columns_to_drop, axis=1, inplace=True)
df2
```

[116]:

	Nombre	Colegio	Ciudad
0	Juan	Colegio1	Ciudad1
1	Carlos	Colegio2	Ciudad2
2	Juan	Colegio1	Ciudad3
3	Sonia	Colegio4	Ciudad4
4	Pedro	Colegio5	Ciudad5

[73]:
```
# ahora vamos a dejar el df2 unico por nombre
df2
```

[73]:

	Nombre	Colegio	Ciudad
0	Juan	Colegio1	Ciudad1
1	Carlos	Colegio2	Ciudad2
2	Juan	Colegio1	Ciudad3
3	Sonia	Colegio4	Ciudad4
4	Pedro	Colegio5	Ciudad5

[76]:
```

```

[76]:

	Nombre	Colegio	Ciudad	count_N	count_C
0	Juan	Colegio1	Ciudad1	2	2
1	Carlos	Colegio2	Ciudad2	1	1
2	Juan	Colegio1	Ciudad3	2	2
3	Sonia	Colegio4	Ciudad4	1	1
4	Pedro	Colegio5	Ciudad5	1	1

[83]:
```
df2
```

[83]:

	Nombre	Colegio	Ciudad	count_N	count_C
0	Juan	Colegio1	Ciudad1	2	2
1	Carlos	Colegio2	Ciudad2	1	1
2	Juan	Colegio1	Ciudad3	2	2

```
3   Sonia  Colegio4  Ciudad4        1        1
4   Pedro  Colegio5  Ciudad5        1        1
```

[82]:
```
df["count"] = df.groupby(["Nombre","Colegio"]).size().reset_index(name="count")
df2
```

```
␣
↪---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call␣
↪last)

<ipython-input-82-95e413bbc7e0> in <module>
----> 1 df["count"] = df.groupby(["Nombre","Colegio"]).size().
↪reset_index(name="count")
      2 df2


~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py in␣
↪groupby(self, by, axis, level, as_index, sort, group_keys, squeeze, observed,␣
↪dropna)
   8397         axis = self._get_axis_number(axis)
   8398
-> 8399         return DataFrameGroupBy(
   8400             obj=self,
   8401             keys=by,


~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/groupby/groupby.
↪py in __init__(self, obj, keys, axis, level, grouper, exclusions, selection,␣
↪as_index, sort, group_keys, squeeze, observed, mutated, dropna)
    957             from pandas.core.groupby.grouper import get_grouper
    958
--> 959             grouper, exclusions, obj = get_grouper(
    960                 obj,
    961                 keys,


~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/groupby/grouper.
↪py in get_grouper(obj, key, axis, level, sort, observed, mutated, validate,␣
↪dropna)
    886                 in_axis, level, gpr = False, gpr, None
    887             else:
--> 888                 raise KeyError(gpr)
    889         elif isinstance(gpr, Grouper) and gpr.key is not None:
    890             # Add key to exclusions
```

```
KeyError: 'Colegio'
```

[75]:
```
df2.drop(["count_N", "count_C"], axis=1, inplace=True)
```

[ ]:
```
```

10.1 DE STATA A PYTHON

[22]:
```
# generar datos aleatorios
n_filas = 25
ingresos = np.random.randint(10000, 50000, size=n_filas)
paises = np.random.choice(['Argentina', 'Brasil', 'Chile', 'México', 'Perú'],␣
 ↪size=n_filas)
años = np.random.choice([2010, 2011, 2012, 2013, 2014], size=n_filas)

# crear DataFrame
df = pd.DataFrame({'ingresos': ingresos, 'pais': paises, 'año': años})

# filtrar por los paises y años que aparecen al menos 5 veces
paises_seleccionados = df['pais'].value_counts().index[df['pais'].
 ↪value_counts() >= 5]
años_seleccionados = df['año'].value_counts().index[df['año'].value_counts() >=␣
 ↪5]
df = df.loc[df['pais'].isin(paises_seleccionados) & df['año'].
 ↪isin(años_seleccionados)]

# imprimir el DataFrame resultante
df
data = df
data
```

[22]:

	ingresos	pais	año
1	27433	Perú	2012
2	16380	México	2010
3	14511	México	2010
4	40337	Argentina	2010
8	42469	México	2010
11	18036	Perú	2012
14	29759	Argentina	2010
17	14030	Perú	2012
19	24830	Argentina	2012
20	44500	Perú	2010
21	15341	Perú	2010
22	18994	Argentina	2012

```
24     20114     México  2012
```

[23]:
```
# Una variable que es la suma por paises

data['aux'] = data.groupby('pais')['ingresos'].transform('sum')
data
```

[23]:

	ingresos	pais	año	aux
1	27433	Perú	2012	119340
2	16380	México	2010	93474
3	14511	México	2010	93474
4	40337	Argentina	2010	113920
8	42469	México	2010	93474
11	18036	Perú	2012	119340
14	29759	Argentina	2010	113920
17	14030	Perú	2012	119340
19	24830	Argentina	2012	113920
20	44500	Perú	2010	119340
21	15341	Perú	2010	119340
22	18994	Argentina	2012	113920
24	20114	México	2012	93474

[24]:
```
# Numero de obs por paises

data['aux'] = data.groupby('pais')['ingresos'].transform('count')
data
```

[24]:

	ingresos	pais	año	aux
1	27433	Perú	2012	5
2	16380	México	2010	4
3	14511	México	2010	4
4	40337	Argentina	2010	4
8	42469	México	2010	4
11	18036	Perú	2012	5
14	29759	Argentina	2010	4
17	14030	Perú	2012	5
19	24830	Argentina	2012	4
20	44500	Perú	2010	5
21	15341	Perú	2010	5
22	18994	Argentina	2012	4
24	20114	México	2012	4

[25]:
```
# Adapatamos el bys v1 : gen aux = _n == 1

data['aux'] = data.groupby('pais')['ingresos'].transform(lambda x: x.index == x.
 ↪index.min())
data
```

[25]:

	ingresos	pais	año	aux
1	27433	Perú	2012	True
2	16380	México	2010	True
3	14511	México	2010	False
4	40337	Argentina	2010	True
8	42469	México	2010	False
11	18036	Perú	2012	False
14	29759	Argentina	2010	False
17	14030	Perú	2012	False
19	24830	Argentina	2012	False
20	44500	Perú	2010	False
21	15341	Perú	2010	False
22	18994	Argentina	2012	False
24	20114	México	2012	False

[26]:
```
# Queremos hacer el reshape, pero antes tenemos que hacer un collapse

df = data.groupby(['pais', 'año'])['ingresos'].sum().reset_index()
df
```

[26]:

	pais	año	ingresos
0	Argentina	2010	70096
1	Argentina	2012	43824
2	México	2010	73360
3	México	2012	20114
4	Perú	2010	59841
5	Perú	2012	59499

[27]:
```
# Reshape wide

data_wide = df.pivot(index='año', columns='pais', values='ingresos')
data_wide
```

[27]:

pais	Argentina	México	Perú
año			
2010	70096	73360	59841
2012	43824	20114	59499

[29]:
```
# Reshape long

data_long = data.melt(id_vars=['pais', 'año'], value_vars=['ingresos'])
data_long
```

[29]:

	pais	año	variable	value
0	Perú	2012	ingresos	27433
1	México	2010	ingresos	16380
2	México	2010	ingresos	14511

```
3   Argentina  2010  ingresos  40337
4      México  2010  ingresos  42469
5        Perú  2012  ingresos  18036
6   Argentina  2010  ingresos  29759
7        Perú  2012  ingresos  14030
8   Argentina  2012  ingresos  24830
9        Perú  2010  ingresos  44500
10       Perú  2010  ingresos  15341
11  Argentina  2012  ingresos  18994
12     México  2012  ingresos  20114
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Ejemplos examen CNMV

[ ]:
```
import pandas as pd
import pandas_datareader as pdr
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[14]:
```
# Vamos a generar un df y definiendo una función y bucles vamos a cambiar␣
 ↪ciertos string
import pandas as pd


gonzalez = ["Adrián", "David", "Victoria", "JOSE"]
```

```
fernandez = ["Carolina", "ROSÍ", "Alex"]
fernandez.extend([""] * 1)

# Creamos el dataframe a partir de las variables gonzalez y fernandez
df = pd.DataFrame({"gonzalez": gonzalez, "fernandez": fernandez})

# Imprimimos el dataframe para verificar que se haya creado correctamente
print(df)
```

```
   gonzalez fernandez
0    Adrián  Carolina
1     David      ROSÍ
2  Victoria      Alex
3      JOSE
```

[15]:
```
import unicodedata

def modificar_nombres(df):
  # Recorremos cada fila del dataframe
  for i, row in df.iterrows():
    # Recorremos cada columna del dataframe
    for col in df.columns:
      nombre = row[col]
      nombre = nombre.capitalize()  # Primera letra en mayúscula
      nombre = unicodedata.normalize("NFKD", nombre).encode("ascii", "ignore").
 ↪decode()  # Quitamos acentos
      if nombre == "Alex":
        nombre = "Alejandro"
      df.at[i, col] = nombre

# Llamamos a la función para modificar el dataframe
modificar_nombres(df)

# Imprimimos el dataframe para verificar que se hayan realizado las␣
 ↪modificaciones correctamente
print(df)
```

```
   gonzalez  fernandez
0    Adrian   Carolina
1     David       Rosi
2  Victoria  Alejandro
3      Jose
```

[16]:
```
import os
directorio_actual = os.getcwd()
print(directorio_actual)
```

```
/Users/adrian_gr/Desktop
```

[20]:
```
os.chdir("/Users/adrian_gr/Desktop/cnmv")
os.getcwd()
```

[20]:
```
'/Users/adrian_gr/Desktop/cnmv'
```

[23]:
```
import pandas as pd

# Abre el archivo
df = pd.read_excel("prueba_python.xlsx")
df
```

[23]:

	A	B	C
0	1	a	s
1	2	d	d
2	3	e	y
3	4	t	l

10.1.1 CONDICIONAL

Los condicionales son una parte fundamental de cualquier lenguaje de programación y te permiten controlar el flujo de ejecución de tu código según ciertas condiciones. En Python, hay varias estructuras de control de flujo que puedes utilizar para trabajar con condicionales. Aquí hay cinco cosas importantes que debes saber sobre la estructura de los condicionales en Python:

El bloque de código que se ejecuta cuando se cumple una condición se llama ramificación. En Python, puedes utilizar la sentencia if para especificar una condición y el bloque de código que se debe ejecutar cuando se cumple esa condición.

Puedes utilizar la sentencia elif (abreviatura de "else if") para especificar múltiples ramificaciones. Esto te permite verificar múltiples condiciones y ejecutar diferentes bloques de código según cuál de ellas se cumpla.

Si quieres especificar un bloque de código que se debe ejecutar cuando ninguna de las condiciones se cumple, puedes utilizar la sentencia else. Esto es útil cuando quieres cubrir todos los casos posibles.

Python utiliza valores booleanos (verdadero o falso) para evaluar las condiciones. Puedes utilizar operadores de comparación (como == para igualdad, < para menor que, etc.) para crear condiciones más complejas. También puedes utilizar operadores lógicos (como and y or) para combinar múltiples condiciones.

Es importante utilizar indentación para organizar el código y hacerlo más legible. Los bloques de código que deben ejecutarse cuando se cumple una condición deben indentarse respecto a la sentencia if, elif o else correspondiente.

[25]:
```
### EJEMPLO DE CONDICONAL ###
# Ejemplo de uso de condicionales

# Definimos una variable
edad = 17
```

```python
# Utilizamos una sentencia if para verificar si la edad es mayor o igual a 18
if edad >= 18:
    # Si se cumple la condición, ejecutamos este bloque de código
    print("Eres mayor de edad")

# Si la edad es menor que 18, no se ejecuta este bloque de código

# Utilizamos una sentencia elif para verificar si la edad es mayor o igual a 65
elif edad >= 65:
    # Si se cumple esta condición, ejecutamos este bloque de código
    print("Eres una persona mayor")

# Si ninguna de las condiciones anteriores se cumple, ejecutamos este bloque de␣
 ↪código
else:
    print("Eres una persona joven")
```

```
Eres una persona joven
```

[26]:
```python
# Ejemplo de condicional muy compleja

x = 5
y = 3
z = -1
if x > 0 and y < 0 and z == 0:
    print("x es un número positivo, y es un número negativo y z es cero")
elif x > 0 and y < 0:
    print("x es un número positivo y y es un número negativo")
elif x < 0 and y > 0 and z == 0:
    print("x es un número negativo, y es un número positivo y z es cero")
else:
    print("La condición no se cumple")
```

```
La condición no se cumple
```

10.1.2 FUNCIONES Y CONDIONALES

Para definir una función en Python, utilizas la palabra reservada def seguida del nombre de la función y una lista de parámetros entre paréntesis. Por ejemplo:

def mi_funcion(param1, param2): # Código de la función pass

Los parámetros son variables que se utilizan para pasar información a la función. Al llamar a la función, debes proporcionar valores para cada parámetro. Los parámetros tienen un valor por defecto, que se utiliza si no se proporciona un valor al llamar a la función.

Dentro de la función, puedes utilizar las variables de los parámetros como cualquier otra variable. Además, puedes utilizar otras variables locales dentro de la función, que solo estarán disponibles dentro de la función.

Para devolver un valor desde una función, utilizas la palabra reservada return. Esto te permite proporcionar un resultado a quien llame a la función.

Las funciones también pueden tener documentación, que se escribe en una cadena de documentación (o "docstring") al comienzo de la función. Esta docstring describe qué hace la función y cómo se utiliza. Puedes acceder a la docstring de una función utilizando la función help() o accediendo a la propiedad **doc** de la función.

[28]:
```
data = pd.read_excel("prueba_python.xlsx")
data
```

[28]:

	A	B	C
0	1	a	s
1	2	d	d
2	3	e	y
3	4	t	l

[29]:
```
# la columna A, elevar al cuadrado, si la suma es mayor que 9

def aplicar_funcion_condicional(datos, columna, funcion, condicion):
    """
    Aplica una función a una columna de un DataFrame solo si se cumple una␣
 ↪condición.
    """
    # Creamos una copia del DataFrame para no modificar el original
    df = datos.copy()

    # Verificamos si se cumple la condición. eval comprueba si la condición es␣
 ↪cierta
    if eval(condicion):
        # Si se cumple la condición, aplicamos la función
        df[columna] = df[columna].apply(funcion)

    # Devolvemos el DataFrame con la función aplicada
    return df
```

[32]:
```
# Aplicamos la función np.log1p a la columna "total_bill" solo si el valor␣
 ↪máximo de la columna es mayor que 10
df_transformado = aplicar_funcion_condicional(data, "A", np.square, "df['A'].
 ↪sum() > 9")
df_transformado # como cumple la condición lo hace
```

[32]:

	A	B	C
0	1	a	s
1	4	d	d
2	9	e	y
3	16	t	l

10.1.3 SQL

Claro, aquí van las cinco cosas más importantes que debes saber sobre la captura de datos de SQL Server utilizando SQLAlchemy y PyODBC en Python:

SQLAlchemy es una librería de Python que ofrece una interfaz de alto nivel para trabajar con bases de datos. Permite crear, leer, actualizar y eliminar (CRUD) datos en diversas bases de datos de manera muy sencilla, incluyendo SQL Server.

PyODBC es una librería de Python que permite conectarse a bases de datos utilizando ODBC, un estándar de conectividad que soporta una amplia variedad de bases de datos. Es muy útil para conectarse a bases de datos que no son soportadas nativamente por SQLAlchemy, como SQL Server.

Para conectarse a una base de datos de SQL Server utilizando SQLAlchemy y PyODBC, primero debes instalar los módulos necesarios: pip install sqlalchemy pyodbc. Luego, debes crear una cadena de conexión que indique el tipo de base de datos, el servidor, el usuario y la contraseña. Por ejemplo: 'mssql+pyodbc://mi_usuario:mi_contraseña@mi_servidor/mi_base_de_datos?driver=SQL+Server+Native+Cl

Una vez que tienes la cadena de conexión, puedes utilizar SQLAlchemy para crear un motor de base de datos y una sesión. El motor es el encargado de realizar las conexiones y las consultas a la base de datos, mientras que la sesión es el objeto encargado de administrar las transacciones. Por ejemplo:

Copy code from sqlalchemy import create_engine from sqlalchemy.orm import sessionmaker

11 Crea el motor de base de datos

engine = create_engine(mi_cadena_de_conexion)

12 Crea la sesión

Session = sessionmaker(bind=engine) session = Session() Una vez que tienes el motor y la sesión creados, puedes utilizar SQLAlchemy para crear una clase que represente una tabla de la base de datos y realizar consultas utilizando la sintaxis de Python. Por ejemplo, para seleccionar todos los datos de la tabla "clientes": from sqlalchemy import Column, Integer, String from sqlalchemy.ext.declarative import declarative_base

13 Crea la clase que representa la tabla "clientes"

Base = declarative_base() class Cliente(Base): **tablename** = "clientes" id = Column(Integer, primary_key=True) nombre = Column(String) apellidos = Column(String) pais = Column(String)

14 Selecciona todos los datos de la tabla "clientes"

clientes = session.query(Cliente).all()

15 Muestra los datos

for cliente in clientes: print(cliente.id, cliente.nombre, cliente.apellidos, cliente.pais)

[35]:
```
import pandas as pd

clientes = [
    (1, "Juan", "Pérez", "España"),
    (2, "Ana", "García", "España"),
    (3, "Pedro", "Sánchez", "España"),
    (4, "Sara", "Rodríguez", "España"),
    (5, "Mario", "Fernández", "Italia"),
    (6, "Laura", "Martínez", "Francia"),
]

facturas = [
    (1, 1, 50),
    (2, 1, 75),
    (3, 1, 25),
    (4, 2, 15),
    (5, 2, 50),
    (6, 3, 10),
    (7, 3, 20),
    (8, 3, 30),
    (9, 3, 40),
    (10, 3, 50),
    (11, 4, 25),
    (12, 4, 75),
    (13, 5, 10),
    (14, 5, 20),
    (15, 6, 15),
]

# Crea el DataFrame de clientes
df_clientes = pd.DataFrame(clientes, columns=["id", "nombre", "apellidos",␣
 ↪"pais"])

# Crea el DataFrame de facturas
df_facturas = pd.DataFrame(facturas, columns=["factura", "id", "importe"])
```

[45]:
```
df = pd.merge(df_facturas, df_clientes, on="id")
df
```

[45]:

	factura	id	importe	nombre	apellidos	pais
0	1	1	50	Juan	Pérez	España
1	2	1	75	Juan	Pérez	España
2	3	1	25	Juan	Pérez	España
3	4	2	15	Ana	García	España
4	5	2	50	Ana	García	España
5	6	3	10	Pedro	Sánchez	España
6	7	3	20	Pedro	Sánchez	España

```
7          8   3       30  Pedro    Sánchez   España
8          9   3       40  Pedro    Sánchez   España
9         10   3       50  Pedro    Sánchez   España
10        11   4       25   Sara  Rodríguez   España
11        12   4       75   Sara  Rodríguez   España
12        13   5       10  Mario  Fernández   Italia
13        14   5       20  Mario  Fernández   Italia
14        15   6       15  Laura   Martínez  Francia
```

[ ]: