18_Practica_Obligatoria_Pandas_I

November 28, 2023

```
edad_gonzalo = 30
edad_gonzalo_en_estreno = edad_gonzalo - (2023 - serie_lanzamiento)
print(edad_gonzalo_en_estreno)
```

✓ 0.0s

```
Toy Story 4            22
Los Incríbles 2        25
Buscando a Dory        23
Toy Story 3            17
Caco                   24
Inside Out             22
Monsters University    20
Up                     13
Name: lanzamiento, dtype: int64
```

0.1 PRACTICA OBLIGATORIA: Iniciación a Pandas

- La práctica obligatoria de esta unidad consiste en varios ejercicios de programación libre a completar. Descarga este notebook en tu ordenador y trabaja en local. Ten en cuenta que tendrás que descar los directorios de imágenes y datos adicionales, si los hubiera.
- Recuerda que debes subirla a tu repositorio personal antes de la sesión en vivo para que puntúe adecuadamente.

- Recuerda también que no es necesario que esté perfecta, sólo es necesario que se vea el esfuerzo.
- Esta práctica se resolverá en la sesión en vivo correspondiente y la solución se publicará en el repo del curso.

1 #1 Series

Importa pandas y numpy de la forma que hemos visto hasta ahora:

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

1. A partir de las listas siguientes, crea tres series que tengan como índices los títulos de las películas. Guárdalas cada una en una variable, que usarás a lo largo de la práctica.m

[2]:
```
titulos = ["Toy Story 4", "Los Incríbles 2", "Buscando a Dory", "Toy Story 3",␣
 ↪"Caco", "Inside Out", "Monsters University", "Up"]
lanzamiento = [2015, 2018, 2016, 2010, 2017, 2015, 2013, 2006]
recaudaciones = [1073, 1242, 1029, 1067, 807, 857, 744, 735]  # En millones de␣
 ↪dólares
espectadores = [74.91, 93.42, 76.72, 81.35, 62.75, 68.27, 54.74, 54.34] # En␣
 ↪millones, estimación hecha por aficionados
```

[ ]:
```
```

[3]:
```
# creo un diccionario por si me hiciera falta mas adelante, con todo (PRUEBA␣
 ↪POR SI ME HACIA FALTA MAS ADELANTE)
dict_completo = {}
for i in range(len(titulos)):
    dict_completo[titulos[i]] = {
        "año_lanzamiento": lanzamiento[i],
        "recaudacion(millones $)": recaudaciones[i],
        "personas(millones habitantes": espectadores[i]
        }
#rint(dict_completo)
```

[16]:
```
##1##
# hago arrays mediante su posicion en la lista y despues le hago su dataframe␣
 ↪con indice por peliculas
serie_1_np = np.array(lanzamiento)
serie_1 = pd.DataFrame(serie_1_np, columns = ["año lan."], index =[titulos])
serie_1
```

[16]:

	año lan.
Toy Story 4	2015
Los Incríbles 2	2018
Buscando a Dory	2016
Toy Story 3	2010
Caco	2017
Inside Out	2015
Monsters University	2013
Up	2006

[17]:
```
# hago arrays mediante su posicion en la lista y despues le hago su dataframe␣
 ↪con indice por peliculas y columna
serie_2_np = np.array(recaudaciones)
serie_2 = pd.DataFrame(serie_2_np, columns = ["millones $)"],index =[titulos])
serie_2
```

[17]:

	millones $)
Toy Story 4	1073
Los Incríbles 2	1242

```
Buscando a Dory             1029
Toy Story 3                 1067
Caco                         807
Inside Out                   857
Monsters University          744
Up                           735
```

[18]:
```
# hago arrays mediante su posicion en la lista y despues le hago su dataframe␣
 ↪con indice por peliculas y columna año
serie_3_np = np.array(espectadores)
serie_3 = pd.DataFrame(serie_3_np, columns = ["millones hab."], index␣
 ↪=[titulos])
serie_3
```

[18]:
```
                     millones hab.
Toy Story 4                  74.91
Los Incríbles 2              93.42
Buscando a Dory              76.72
Toy Story 3                  81.35
Caco                         62.75
Inside Out                   68.27
Monsters University          54.74
Up                           54.34
```

[ ]:
```
```

2. El método sort_values() ordena de forma ascendente una serie. Pruébalo con la serie de recaudaciones. Busca el argumento que te permita hacer la ordenación en orden inverso y crea una nueva serie para las recaudaciones que esté ordenada de mayor recaudación a menor.

[19]:
```
##2##
# extructura de este metodo: DataFrame o Serie.sort_values(by, axis=0,␣
 ↪ascending=True, inplace=False, ignore_index=False)(by: especifica columna/s␣
 ↪que se desae ordenar, pudiendo ser nombre de columnas o listas de nombres,
# axis= si va ordenar a lo largo de filas(0) o columnas(1), ascending=␣
 ↪ascendiente con True y lo contrario con False, inplace: si debe hacer la␣
 ↪ordenacion en el original o debe crear una copia del dataframe,
# Ignore_index: si es True, restablecera los indices despues de la ordenacion )
df_recaudaciones = pd.Series(recaudaciones) # ccreamos la serie Pandas␣
 ↪recaudaciones
recaudaciones_values_asc = df_recaudaciones.sort_values()
recaudaciones_values_des = df_recaudaciones.sort_values(ascending= False)
print(recaudaciones_values_asc)
print("\n")
print(recaudaciones_values_des)
```

```
7     735
6     744
```

```
4     807
5     857
2    1029
3    1067
0    1073
1    1242
dtype: int64


1    1242
0    1073
3    1067
2    1029
5     857
4     807
6     744
7     735
dtype: int64
```

[ ]:

3. Utilizando la serie de recaudaciones obtenida en el apartado anterior, recorrela de forma que muestres para cada película su recaudación y ya de paso su año de lanzamiento y el número de espectadores. (Ojo el que corresponda de forma correcta, no te vale el índice de un enumerate por ejemplo, pero no lo necesitas, recuerda que las Series vienen con su índice explícito incorporado y todas las que hayas creado tienen el mismo). Muestra también el precio medio de la entrada que tuvo cada película.

[20]:
```
# FORMA SIN INDICES (INCORRECTA), YA QUE NO DA LOS INDICES CORRECTAMENTE
#crero los Series de los elementos titulos, lanzamiento, recaudaciones y␣
 ↪espectadores
df_titulos = pd.Series(titulos)
df_lanzamientos= pd.Series(lanzamiento)
df_recaudaciones = pd.Series(recaudaciones)
df_espectadores = pd.Series(espectadores)
# para ordenar los cuatro dataframe usando un metodo similar en numpy pero en␣
 ↪pandas que ordenas los dataframe a lo largo del eje que elijas 0 = filas o 1␣
 ↪= columnas
df_mezclado = pd.
 ↪concat([df_titulos,df_lanzamientos,df_recaudaciones,df_espectadores], axis=1)
```

[31]:
```
##3##
# FORMA CON INDICES, CORRECTA PARA VALORAR
#creo una cuarta serie con valores inventados de los tickects de entrada en␣
 ↪dolares; primero pongo la lista normal, la paso a array y despues hago su␣
 ↪dataframe ordando por coumnas y filas
precio= round(df_recaudaciones/df_espectadores, 2)
```

```
serie_4_pn = np.array(precio)
serie_4 = pd.DataFrame(serie_4_pn, columns = ["ticket $"], index =[titulos])
# finalmente creo la tabla con todos los valores
dt_completo= pd.concat([serie_1, serie_2, serie_3, serie_4], axis=1)
dt_completo
```

[31]:

	año lan.	millones $)	millones hab.	ticket $
Toy Story 4	2015	1073	74.91	14.32
Los Incríbles 2	2018	1242	93.42	13.29
Buscando a Dory	2016	1029	76.72	13.41
Toy Story 3	2010	1067	81.35	13.12
Caco	2017	807	62.75	12.86
Inside Out	2015	857	68.27	12.55
Monsters University	2013	744	54.74	13.59
Up	2006	735	54.34	13.53

[ ]:

4. Contesta: (Usando código, of course) 4.1 ¿De que año es la película con menor recaudación? 4.2 ¿Cómo se llama la película con más de 1000 millones de recaudación y menos de 75 millones de espectadores? 4.3 Gonzalo tiene ahora 30 años, ¿con qué años fue a ver cada película, teniendo en cuenta que fue al estreno de todas?

[10]:
```
#(PRUEBA)#
#pelicula año emnor recaudacion
peli_agno_menor_recaudacion = df_lanzamientos.values[7]
#print(peli_agno_menor_recaudacion)
#print("\n")
#película con más de 1000 millones de recaudación y menos de 75 millones de␣
 ↪espectadores
peli_1000_menos_75 = df_titulos.iloc[0:1]
#print(peli_1000_menos_75)
#print("\n")
```

[34]:
```
########FORMA CORRECTA A VALORAR########
##4.1#
# aplico la funcion min para hallar los valores minimos y uso la funion idxmin,␣
 ↪para saber el indice correspondiente al min, o el idxmax para concer el␣
 ↪maximo
minimo_pelicula= dt_completo.min()
print(minimo_pelicula)
print("\n")
##4.2#
# hago el dataframe de titulos para que me de el titulo de la pelicula con␣
 ↪ayuda de otra funcion
titulo_np = np.array(titulos)
dt_titulo = pd.DataFrame(titulo_np, columns= ["Peliculas"])
```

```python
# usando la funcion de panda iat(), conseguimos indexar el indice y conseguir␣
 ↪el valor pelicula que le corresponde a los valores minimos (valor =␣
 ↪dataframe.iat[fila, columna])
peli_min=dt_titulo.iat[7, 0]

print(f"la pelicula {peli_min}) fue la que obtuvo valores minimos en la tabla:
 ↪\n{minimo_pelicula}")
print("\n")
##4.3#
#Gonzalo tiene ahora 30 años, ¿con qué años fue a ver cada película, teniendo␣
 ↪en cuenta que fue al estreno de todas?
#hago una funcion para calcular el año de nacimiento de gonzalo y restarlo a␣
 ↪los años de lanzamiento
def ver_pelis(edad, lanzamiento):
    agno_nacimiento = 2023 - edad
    df_lanzamientos= pd.Series(lanzamiento)
    #df_lanzamientos_ord = df_lanzamientos.sort_values()
    for i in range(8):
        resultado = df_lanzamientos.values[:] - agno_nacimiento
    return resultado
#imprimo por pantalla
print(f"Las peliculas de la lista: {(titulos)}\n , lanzadas en los años␣
 ↪{lanzamiento}\nfueron vistas con las siguientes edades por Gonzalo,teniendo␣
 ↪en cuenta tiene 30 años: {ver_pelis(30,lanzamiento)}")
print("\n")
```

```
año lan.          2006.00
millones $)        735.00
millones hab.       54.34
ticket $            12.55
dtype: float64


la pelicula Up) fue la que obtuvo valores minimos en la tabla:
año lan.          2006.00
millones $)        735.00
millones hab.       54.34
ticket $            12.55
dtype: float64


Las peliculas de la lista: ['Toy Story 4', 'Los Incríbles 2', 'Buscando a Dory',
'Toy Story 3', 'Caco', 'Inside Out', 'Monsters University', 'Up']
 , lanzadas en los años [2015, 2018, 2016, 2010, 2017, 2015, 2013, 2006]
fueron vistas con las siguientes edades por Gonzalo,teniendo en cuenta tiene 30
```

```
años: [22 25 23 17 24 22 20 13]
```

```
edad_gonzalo = 30
edad_gonzalo_en_estreno = edad_gonzalo - (2023 - serie_lanzamiento)
print(edad_gonzalo_en_estreno)
```

✓ 0.0s

```
Toy Story 4            22
Los Incríbles 2        25
Buscando a Dory        23
Toy Story 3            17
Caco                   24
Inside Out             22
Monsters University    20
Up                     13
Name: lanzamiento, dtype: int64
```

[ ]:

5. Corrije los siguientes datos erróneos (pero no repitas los apartados anteriores) en las series (no en las listas iniciales) de forma que tus variables contengan los valores correctos:

- Toy Story 4 es de y Up de 2009
- La recaudación de Monsters University fue de 754 millones.
- Hay 2 nombres que han sufrido el efecto del conocido "error Jaime", corrígelo en todas las series (recuerda que los indices son inmutables, tendrás que hacer algo de código)

[45]:
```
##5##
#5.1#cambiamos mediante un cilo for los años,
# el elemento pop solo borrara el 1 elemnto 2015, por lo que no afectara al␣
 ↪segundo
for i in lanzamiento:
    if i == 2015:
        lanzamiento.pop(0)
        lanzamiento.insert(0, 2009)# aqui uso insert para colocar el nuevo␣
 ↪elelento en el lugar del otro
    elif i == 2006:
        lanzamiento.pop(-1)
        lanzamiento.append(2009)# aqui us append porque ira al final de la lista
#impre la  lista correcta de años
print(lanzamiento)
print("\n")
#5.2 # mediante un pop elimino la cantidad erronea e inserto en su lugar la␣
 ↪cantidad corecta
recaudaciones.pop(6)
```

```
recaudaciones.insert(-1,754)
#imprimo recaudaciones con las cantidades correctas
print(recaudaciones)
print("\n")
#5.3
#1 mediante cilo for elimino de la lista titulos  los 2 elementos e inserto en␣
 ↪las mismas posiciones los elementos corectos, al igual que hago con las␣
 ↪series 1 y 2
for i in titulos:
    if i == "Los Incríbles 2":
        titulos.pop(1)
        titulos.insert(1,"Los Increíbles 2")
        serie_1.pop(1)
        serie_1.append("Los Increíbles 2")
    elif i == "Caco":
        titulos.pop(4)
        titulos.insert(4, "Coco")
        serie_2.pop(2)
        serie_2.append("Coco")
#imprimo resultados con los datos correctos
print(titulos)
print(serie_1)
print(serie_2)
```

```
[2009, 2018, 2016, 2010, 2017, 2015, 2013, 2009]


[1073, 1242, 1029, 1067, 807, 857, 754, 735]


['Toy Story 4', 'Los Increíbles 2', 'Buscando a Dory', 'Toy Story 3', 'Coco',
'Inside Out', 'Monsters University', 'Up']
                     año lan.
Toy Story 4              2015
Los Incríbles 2          2018
Buscando a Dory          2016
Toy Story 3              2010
Caco                     2017
Inside Out               2015
Monsters University      2013
Up                       2006
                     millones $)
Toy Story 4                1073
Los Incríbles 2            1242
Buscando a Dory            1029
Toy Story 3                1067
Caco                        807
Inside Out                  857
```

```
Monsters University          744
Up                           735
```

[ ]:

[ ]:

2 #2 DataFrame

1. Crea un `DataFrame` con las series anteriores, ya corregidas, como columnas (usando la serie ordenada de recaudaciones) y que cómo indice de filas tenga las películas.

[37]:
```
#SIN USAR SORT_VALUES ####
# vamos a ordenar en orden descendente los años de lanzamiento , lo epectadores␣
 ↪y el valor medio de los teckets de entrada
#años
lanzamientos_values_des_= df_lanzamientos.sort_values(ascending= False)
#print(ordenar_values_des_agno)
#millones $
#print(recaudaciones_values_des)
#hab
espectadores_values_des= df_espectadores.sort_values(ascending= False)
#print(ordenar_values_des_hab)
# tickets
df_precio = pd.Series(precio)
precio_values_des= df_precio.sort_values(ascending= False)
#print(ordenar_values_des_precio)
#creo una lista de titulos conforme a la ordenacion descendiente
titulos_ord = ['Los Increíbles 2', 'Coco', 'Buscando a Dory', 'Inside Out',␣
 ↪'Monsters University', 'Toy Story 3', 'Toy Story 4', 'Up']
titulos_ord_np = np.array(titulos_ord)
dt_tirulos_ord = pd.DataFrame(titulos_ord_np)
# creo un nuevo dataframe de la serie 1 con la lista ordenada
serie_1_np_ord = np.array(lanzamientos_values_des_)
serie_1_ord =pd.DataFrame(serie_1_np_ord, columns = ["año lan."], index␣
 ↪=[titulos_ord])

# creo un nuevo dataframe de la serie 2 con la lista ordenada
serie_2_np_ord = np.array(recaudaciones_values_des)
serie_2_ord = pd.DataFrame(serie_2_np_ord, columns = ["millones $"], index␣
 ↪=[titulos_ord])

# creo un nuevo dataframe de la serie 3 con la lista ordenada
serie_3_np_ord = np.array(espectadores_values_des)
serie_3_ord = pd.DataFrame(serie_3_np_ord, columns = ["millones hab."], index␣
 ↪=[titulos_ord])
```

```
# creo un nuevo dataframe de la serie 3 con la lista ordenada
serie_4_np_ord = np.array(precio_values_des)
serie_4_ord = pd.DataFrame(serie_4_np_ord, columns = ["Tickets $"], index␣
 ↪=[titulos_ord])

#creo el conjunto las 3 series ordenadas
dt_conjunto_3_series = pd.concat([serie_1_ord, serie_2_ord, serie_3_ord,␣
 ↪serie_4_ord], axis=1)
dt_conjunto_3_series
```

[37]:

	año lan.	millones $	millones hab.	Tickets $
Los Increíbles 2	2018	1242	93.42	14.32
Coco	2017	1073	81.35	13.59
Buscando a Dory	2016	1067	76.72	13.53
Inside Out	2015	1029	74.91	13.41
Monsters University	2015	857	68.27	13.29
Toy Story 3	2013	807	62.75	13.12
Toy Story 4	2010	744	54.74	12.86
Up	2006	735	54.34	12.55

[36]:
```
#USANDO SORT_VALUES#####
precio= round(df_recaudaciones/df_espectadores, 2)
serie_4_pn = np.array(precio)
serie_4 = pd.DataFrame(serie_4_pn, columns = ["ticket $"], index =[titulos])
# finalmente creo la tabla con todos los valores
dt_completo= pd.concat([serie_1, serie_2, serie_3, serie_4], axis=1)
peliculas_completo= dt_completo.sort_values(by=[ "año lan.", "millones $)",␣
 ↪"millones hab.", "ticket $"], axis =0, ascending = True)

peliculas_completo
```

[36]:

	año lan.	millones $)	millones hab.	ticket $
Up	2006	735	54.34	13.53
Toy Story 3	2010	1067	81.35	13.12
Monsters University	2013	744	54.74	13.59
Inside Out	2015	857	68.27	12.55
Toy Story 4	2015	1073	74.91	14.32
Buscando a Dory	2016	1029	76.72	13.41
Caco	2017	807	62.75	12.86
Los Incríbles 2	2018	1242	93.42	13.29

[ ]:
```

```

2. Muestra los datos de las películas que tengan más de 10 años.

[44]:
```
# Crear un diccionario con los con los titulos y con los años de lanzamiento
dict_titulo_año= {
```

```
    'Película': ['Toy Story 4', 'Los Increíbles 2', 'Buscando a Dory', 'Toy␣
 ↪Story 3', 'Coco', 'Inside Out', 'Monsters University', 'Up'],
    'Año de Lanzamiento': [2009, 2018, 2016, 2010, 2017, 2015, 2013, 2009]
}
#creo un nuevo dataframe con ambos valores
dt_decada = pd.DataFrame(dict_titulo_año)

# año actual
agno_actual = 2023

# accedo al dataframe y a la columna del año delanzamiento, flitrando a␣
 ↪peliculas menores del año actual menos 10
peliculas_mas_una_decada = dt_decada[dt_decada['Año de Lanzamiento'] <␣
 ↪agno_actual - 10]

# Mostrar las películas que cumplen con el criterio
display("Películas con más de una decada son:")
display(peliculas_mas_una_decada)
```

```
'Películas con más de una decada son:'
```

	Película	Año de Lanzamiento
0	Toy Story 4	2009
3	Toy Story 3	2010
7	Up	2009

[ ]:

3. Muestra los datos de las películas que superen los 800 millones de recaudación y los 65 millones de habitantes

[43]:
```
# Crear un diccionario con los títulos, recaudación y número de habitantes
dict_titulo_millones = {
    'Película': ['Toy Story 4', 'Los Increíbles 2', 'Buscando a Dory', 'Toy␣
 ↪Story 3', 'Coco', 'Inside Out', 'Monsters University', 'Up'],
    'millones de recaudacion': [1073, 1242, 1029, 1067, 807, 857, 754, 735],
    'millones de habitantes': [74.91, 93.42, 76.72, 81.35, 62.75, 68.27, 54.74,␣
 ↪54.34]
}

# Crear un DataFrame con el diccionario
dt_titulo_pasta_hab = pd.DataFrame(dict_titulo_millones)

# Accedo al contedo del dataframe, accediendo a las columnas ",illones de␣
 ↪habitantes" y "millones de recaudacion, Filtrar películas con más de 65␣
 ↪millones de habitantes y más de 800 millones de recaudación
```

```
peliculas_mas_65_800 = dt_titulo_pasta_hab[(dt_titulo_pasta_hab["millones de␣
 ↪habitantes"] > 65) & (dt_titulo_pasta_hab["millones de recaudacion"] > 800)]

# accedo al contenido del dataframe, mediante la funcion shape, con el que se␣
 ↪obtene la forma (número de filas y columnas, shape[0] seran las filas y␣
 ↪shape[1] las columnas, y filtro aqui si las filas son mayor que cero
if peliculas_mas_65_800.shape[0] > 0:# si hay filas, revisa su contenido␣
 ↪accediendo y comprobando la condicion establecida anteriormente
    display("Las películas que han recaudado más de 800 millones y han sido␣
 ↪vistas por más de 65 millones de personas son:")
    display(peliculas_mas_65_800)
else:
    display("No hay películas que cumplan con los criterios especificados.")
```

```
'Las películas que han recaudado más de 800 millones y han sido vistas por más␣
 ↪de 65 millones de personas son:'
```

	Película	millones de recaudacion	millones de habitantes
0	Toy Story 4	1073	74.91
1	Los Increíbles 2	1242	93.42
2	Buscando a Dory	1029	76.72
3	Toy Story 3	1067	81.35
5	Inside Out	857	68.27

[ ]:

4. Añade una columna "Ingreso_por_espectador" que contenga eso... el ingreso por espectador de cada película

[37]:
```
# para realziar el calculo covierto la listas recaudacion y espectadore sa a␣
 ↪arrays para poder dividir ambas listas
recaudaciones_np =np.array(recaudaciones)
espectadores_np =np.array(espectadores)
#resultado de la division de ambas listas arrays
ingresos_por_espectador = recaudaciones_np / espectadores_np
ingresos_por_espectador
```

[37]:
```
array([14.32385529, 13.29479769, 13.41240876, 13.11616472, 12.86055777,
       12.55309799, 13.77420533, 13.52594774])
```

[42]:
```
# Crear un diccionario con los títulos, recaudación y número de habitantes,␣
 ↪añadiendole ingresos por espectador, obtenido en la celda anterior
dict_titulo_millones = {
    'Película': ['Toy Story 4', 'Los Increíbles 2', 'Buscando a Dory', 'Toy␣
 ↪Story 3', 'Coco', 'Inside Out', 'Monsters University', 'Up'],
    'millones de recaudacion': [1073, 1242, 1029, 1067, 807, 857, 754, 735],
    'millones de habitantes': [74.91, 93.42, 76.72, 81.35, 62.75, 68.27, 54.74,␣
 ↪54.34],
```

```
    'ingresos por espectador': [14.32385529, 13.29479769, 13.41240876, 13.
 ↪11616472, 12.86055777,
       12.55309799, 13.77420533, 13.52594774]
}

# Crear un DataFrame con el diccionario
dt_titulo_pasta_hab_precio = pd.DataFrame(dict_titulo_millones)

# Accedo al contedo del dataframe, accediendo a las columnas ",illones de␣
 ↪habitantes" y "millones de recaudacion, Filtrar películas con más de 65␣
 ↪millones de habitantes y más de 800 millones de recaudación
peliculas_mas_65_800_precio =␣
 ↪dt_titulo_pasta_hab_precio[(dt_titulo_pasta_hab_precio["millones de␣
 ↪habitantes"] > 65) & (dt_titulo_pasta_hab_precio["millones de recaudacion"]␣
 ↪> 800) &  (dt_titulo_pasta_hab_precio["ingresos por espectador"])]


# accedo al contenido del dataframe, mediante la funcion shape, con el que se␣
 ↪obtene la forma (número de filas y columnas, shape[0] seran las filas y␣
 ↪shape[1] las columnas, y filtro aqui si las filas son mayor que cero
if peliculas_mas_65_800_precio .shape[0] > 0:# si hay filas, revisa su␣
 ↪contenido accediendo y comprobando la condicion establecida anteriormente
    display("Las películas que han recaudado más de 800 millones y  han sido␣
 ↪vistas por más de 65 millones de personas  son:")
    display(peliculas_mas_65_800_precio)
else:
    display("No hay películas que cumplan con los criterios especificados.")
```

```
'Las películas que han recaudado más de 800 millones y  han sido vistas por más␣
 ↪de 65 millones de personas  son:'
```

	Película	millones de recaudacion	millones de habitantes	\
0	Toy Story 4	1073	74.91	
1	Los Increíbles 2	1242	93.42	
2	Buscando a Dory	1029	76.72	
3	Toy Story 3	1067	81.35	
5	Inside Out	857	68.27	

	ingresos por espectador
0	14.323855
1	13.294798
2	13.412409
3	13.116165
5	12.553098

[ ]:

5. Igual que existe sort_values para `Series`, existe para sort_values para `DataFrame`, solo que

además le puedes indicar más de una columna. Ordena el `DataFrame` para que muestre las películas ordenadas de forma ascendente por Año. Es decir que la variable que contenga el `DataFrame` termine teniendo el dataframe ordenado después de ejecutar el código.

NOTA PERSONAL = extructura dataframe.sort_values: **by**: especifica columna/s que se desae ordenar, pudiendo ser nombre de columnas o listas de nombres, **axis**= si va ordenar a lo largo de filas(0) o columnas(1), **ascending**= ascendiente con True y lo contrario con False, **inplace**: si debe hacer la ordenacion en el original o debe crear una copia del dataframe,**Ignore_index**: si es True, restablecera los indices despues de la ordenacion ##### (by, axis=0, ascending=True, inplace=False, ignore_index=False)

[99]:
```
peliculas_asc_año= dt_decada.sort_values(by=[ "Año de Lanzamiento",␣
↪"Película"], axis =0, ascending = True)

peliculas_asc_año
```

[99]:

	Película	Año de Lanzamiento
0	Toy Story 4	2009
7	Up	2009
3	Toy Story 3	2010
6	Monsters University	2013
5	Inside Out	2015
2	Buscando a Dory	2016
4	Coco	2017
1	Los Increíbles 2	2018

6. Ordena el `DataFrame` ahora para que quede ordenado de mayor a menor recaudación por espectador.

[45]:
```
peliculas_asc_dinero= dt_titulo_pasta_hab_precio.sort_values(by=[ "ingresos por␣
↪espectador", "Película", "millones de recaudacion", "millones de␣
↪habitantes"], axis =0, ascending = False)

peliculas_asc_dinero
```

[45]:

	Película	millones de recaudacion	millones de habitantes	ingresos por espectador
0	Toy Story 4	1073	74.91	14.323855
6	Monsters University	754	54.74	13.774205
7	Up	735	54.34	
2	Buscando a Dory	1029	76.72	
1	Los Increíbles 2	1242	93.42	
3	Toy Story 3	1067	81.35	
4	Coco	807	62.75	
5	Inside Out	857	68.27	

```
7                13.525948
2                13.412409
1                13.294798
3                13.116165
4                12.860558
5                12.553098
```

[ ]:

7. Deshaz las modificaciones de el punto 5 de la parte #1

[97]:
```
eliculas_asc_año= dt_decada.sort_values(by=[ "Año de Lanzamiento", "Película"],␣
↪axis =0, ascending = True)

peliculas_asc_año
```

[97]:
```
              Película  Año de Lanzamiento
0          Toy Story 4                2009
7                   Up                2009
3          Toy Story 3                2010
6  Monsters University                2013
5           Inside Out                2015
2      Buscando a Dory                2016
4                 Coco                2017
1     Los Increíbles 2                2018
```

8. Finalmente, queremos que el título sea una columna más y no el índice, investiga el método set_index y el reset_index para que el título pase a ser una columna y el año el nombre de las filas (o índice del DataFrame)

la funcion **set_index** tiene la siguiente sintaxis : **DF.set_index(keys, drop=True, append=False, inplace=False)**, siendo Keys representa la nueva/s etiqueta/s que se quieren poner como indice, pudiendo ser 1 o varias columnas. drop tiene 2 estados booleanos, siendo su valor predeterminado *True*, realizando que la columna que antes era indice se elimine. Si es *False*, se conservaran como columnas en el DF, agragando como columna a la columna que has puesto de indice(columna repetida). append tiene 2 estados booleanos, siendo su valor predeterminado *True*, por el cual las columnas usadas como indice se conservaran en el DF y se agragaran a las existentes como un indice multiple si ya hubiera un indice. inpalce tiene 2 estados booleanos, siendo su valor predeterminado *False*. Si se establece en True, se aplicarán los cambios directamente al DataFrame, en lugar de crear un nuevo DataFrame con el índice modificado.

Lafuncion **reset_index** iene la siguiente sintaxis : **DF.reset_index(level=None, drop=False, inplace=False)**, siendo level Especifica el nivel del índice para restablecer. Si se omite, todos los niveles del índice se restablecerán. Puede ser una etiqueta única o una lista de etiquetas si el índice es jerárquico. drop tiene 2 estados booleanos, siendo su valor predeterminado *False*, el indice se mantendra como columna del DF, si es *True* lo eliminara. inplace igual que en el set_index

[47]:
```
#hago copia del Df original
dt_completo_e = dt_completo.copy()
```

[52]:
```
cambio_indice_año = dt_completo_e.set_index ("año lan.", drop= True,append =␣
↪True, inplace =False)
cambio_indice_año
```

[52]:

	año lan.	millones $)	millones hab.	ticket $
Toy Story 4	2015	1073	74.91	14.32
Los Incríbles 2	2018	1242	93.42	13.29
Buscando a Dory	2016	1029	76.72	13.41
Toy Story 3	2010	1067	81.35	13.12
Caco	2017	807	62.75	12.86
Inside Out	2015	857	68.27	12.55
Monsters University	2013	744	54.74	13.59
Up	2006	735	54.34	13.53

[50]:
```
cambio_indice_añoR= dt_completo_e.reset_index(names= "Pelicula")
cambio_indice_añoR
```

[50]:

	Pelicula	año lan.	millones $)	millones hab.	ticket $
0	Toy Story 4	2015	1073	74.91	14.32
1	Los Incríbles 2	2018	1242	93.42	13.29
2	Buscando a Dory	2016	1029	76.72	13.41
3	Toy Story 3	2010	1067	81.35	13.12
4	Caco	2017	807	62.75	12.86
5	Inside Out	2015	857	68.27	12.55
6	Monsters University	2013	744	54.74	13.59
7	Up	2006	735	54.34	13.53

[36]:

[ ]:

[ ]:

[ ]:

18_Practica_Obligatoria_Pandas_II

November 28, 2023

0.1 PRACTICA OBLIGATORIA: Trabajando datos con Pandas

- La práctica obligatoria de esta unidad consiste en dos partes en las que se trabaja sobre un mismo conjunto de datos muy parecido al visto en las sesiones de trabajo personal y en la segunda unidad. Descarga este notebook en tu ordenador y trabaja en local. Ten en cuenta que tendrás que descar los directorios de imágenes y datos adicionales, si los hubiera.
- Recuerda que debes subirla a tu repositorio personal antes de la sesión en vivo para que puntúe adecuadamente.

- Recuerda también que no es necesario que esté perfecta, sólo es necesario que se vea el esfuerzo.
- Esta práctica se resolverá en la sesión en vivo correspondiente y la solución se publicará en el repo del curso.

0.2 #0 Carga de datos y primera exploración

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

df_viajes_aereos = pd.read_csv("./data/dataset_viajes.csv", index_col =␣
 ↪"Id_vuelo")
```

1. Haz una primera exploración de los datos. Muestra la información general, la descripción de las variables numéricas, las columnas y muestra la distribución de datos de tres columnas escogidas por ti.

[2]:
```
df_viajes_aereos.head()
```

[2]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Air_PaGi_10737	Airnar	París	Ginebra	411.0	Boeing 737	NaN	51.0
Fly_BaRo_10737	FlyQ	Bali	Roma	12738.0	Boeing 737	33479.13254400001	1167.0
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103.0	Airbus A380	NaN	626.0
Mol_PaCi_10737	MoldaviAir	París	Cincinnati	6370.0	Boeing 737	17027.01	503.0
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480.0	Boeing 747	86115.744	518.0

[3]:
```
df_viajes_aereos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Air_PaGi_10737 to Air_PaCi_10737
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  1000 non-null   object
 1   Origen      1000 non-null   object
 2   Destino     1000 non-null   object
 3   Distancia   872 non-null    float64
 4   avion       1000 non-null   object
 5   consumo_kg  862 non-null    object
 6   duracion    853 non-null    float64
dtypes: float64(2), object(5)
memory usage: 62.5+ KB
```

[4]:
```
df_viajes_aereos.describe()
```

[4]:
```
          Distancia    duracion
count    872.000000   853.00000
mean    8107.309633   650.29660
std     5500.759386   458.82867
min      344.000000    42.00000
25%     3073.000000   224.00000
50%     6877.000000   567.00000
75%    12553.000000  1053.00000
max    20029.000000  1721.00000
```

[5]:
```
df_viajes_aereos.dtypes
```

[5]:
```
Aircompany     object
Origen         object
Destino        object
Distancia     float64
avion          object
consumo_kg     object
duracion      float64
dtype: object
```

[6]:
```
df_viajes_aereos.columns
```

[6]:
```
Index(['Aircompany', 'Origen', 'Destino', 'Distancia', 'avion', 'consumo_kg',
       'duracion'],
      dtype='object')
```

[7]:
```
df_viajes_aereos[["duracion", "Distancia", "consumo_kg"]].value_counts()
```

[7]:
```
duracion  Distancia  consumo_kg
433.0     6206.0     75328.428            4
1359.0    15262.0    39341.52892800001    3
845.0     12383.0    151736.3288          3
70.0      698.0      1751.98              3
1305.0    16589.0    205422.781408        3
                                         ..
442.0     5566.0     14877.918            1
                     14742.6642           1
                     14472.1566           1
                     13931.1414           1
1721.0    20029.0    55679.01768000001    1
Name: count, Length: 555, dtype: int64
```

2. Si fueras el analista de datos de la empresa que tiene esta información, ¿sobre qué columna querrías saber más? [No tiene solución única, es para adelantar lo que haremos en el siguiente bloque]

1. Investigaria y analizaria las columnas tipo de avion en relacion a su consumo y el tiempo de duracion en relacion a la distancia, con la finalidad de optimzar los recursos economicos de la

empresa en el gasto de combustible, lo que mejoraria en competitividad y poder lanzar vuelos mas baratos y con una mayor cantidad de ganancias para dicha empresa.

2. Ademas, desde un punto de vista tecnco, me interesaria mucho por las comlunas "duracion", "Distancia" y "consumo_kg" ya que presentan mucho valores nulos e incidiria en la conlumna consumo_kg , ya que a diferencia de las columnas duracion y distancia que son tipo float64, ésta presenta un tipo object(mezcla de tipos), lo cual seguramente sea debido que algun punto se ha cambiado a alguna coma(str) o similar, ademas de tener valores nulos.

0.3 #1 Limpieza de Datos

1. Revisa los datos y contesta a las siguientes preguntas:

   1.1 ¿Existen filas duplicadas?¿Cuántas?¿Hay alguna compañía que lo sufra más que otra o más o menos están igual?

   1.2 ¿Hay columnas con datos faltantes o nulos? ¿Cuáles? ¿Qué porcentaje representan esos valores del total? (Pista: Para contestar a esta pregunta explora los argumentos del método `value_counts`)

   1.3 ¿Hay alguna columna "sucia"? Es decir, ¿existe alguna columna cuyos datos tuvieramos que modificar para tener una mejor comprensión y manajeabilidad de los mismos? ¿Cuáles?

1.1A Existen filas duplicadas?¿Cuántas?

si, existen un total de 54 elementos duplicados, siendo reseñable que la mayoria estan en las columnas de las compañias aereas, tras el analisis realizado que se resume en:La compañia *MoldaviAir* es la que mayores duplicados presenta destando con 20 elementos duplicados, despues existe un grupo de 2 compañias formadas por *PamPangea y TabarAir*, con valores muy proximos: $10, 14$, respectivamente y destacando por el minimo de duplicados encontramos *Airnarr*, la cual, no presenta ningun duplicado, estando muy cerca de esta ultima *FlyQ*, la cual unicamente presenta 4 duplicados, siendo la suma total de los elementos duplicados de las compañias aereas citadas 48 de los 54 totales, representado un 88.88% de elementos totales duplicados en el DataFrame "viajes_aereos".

[8]:
```
df_viajes_aereos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Air_PaGi_10737 to Air_PaCi_10737
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Aircompany  1000 non-null   object 
 1   Origen      1000 non-null   object 
 2   Destino     1000 non-null   object 
 3   Distancia   872 non-null    float64
 4   avion       1000 non-null   object 
 5   consumo_kg  862 non-null    object 
 6   duracion    853 non-null    float64
dtypes: float64(2), object(5)
memory usage: 62.5+ KB
```

[9]:
```
df_viajes_aereos.columns
```

[9]:
```
Index(['Aircompany', 'Origen', 'Destino', 'Distancia', 'avion', 'consumo_kg',
       'duracion'],
      dtype='object')
```

[10]:
```
print(len(df_viajes_aereos.loc[df_viajes_aereos.duplicated(keep = False)]))
df_viajes_aereos.loc[df_viajes_aereos.duplicated(keep = False)]
```

```
54
```

[10]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103.0	Airbus A380
Pam_GiNu_11320	PamPangea	Ginebra	Nueva York	6206.0	Airbus A320
Mol_MeLo_10747	MoldaviAir	Melbourne	Londres	16900.0	Boeing 747
Mol_CiPa_11380	MoldaviAir	Cincinnati	París	6370.0	Airbus A380
Tab_LoNu_11380	TabarAir	Londres	Nueva York	5566.0	Airbus A380
Mol_BaLo_10747	MoldaviAir	Bali	Londres	12553.0	Boeing 747
Pam_NuBa_10747	PamPangea	Nueva York	Bali	16589.0	Boeing 747
Pam_BaPa_11380	PamPangea	Bali	París	11980.0	Airbus A380
Mol_LoCa_11320	MoldaviAir	Londres	Cádiz	1716.0	Airbus A320
Tab_LoNu_11380	TabarAir	Londres	Nueva York	5566.0	Airbus A380
Tab_NuGi_11380	TabarAir	Nueva York	Ginebra	6206.0	Airbus A380
Pam_NuBa_10747	PamPangea	Nueva York	Bali	16589.0	Boeing 747
Tab_LoNu_10737	TabarAir	Londres	Nueva York	5566.0	Boeing 737
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785.0	Airbus A320
Mol_MeLo_11380	MoldaviAir	Melbourne	Londres	16900.0	Airbus A380
Pam_LoBa_10747	PamPangea	Londres	Bali	12553.0	Boeing 747
Tab_RoLo_10737	TabarAir	Roma	Los Angeles	10077.0	Boeing 737
Mol_MeCi_10737	MoldaviAir	Melbourne	Cincinnati	15262.0	Boeing 737
Tab_RoLo_10737	TabarAir	Roma	Los Angeles	10077.0	Boeing 737
Mol_PaCa_11320	MoldaviAir	París	Cádiz	1447.0	Airbus A320
Air_GiBa_11380	Airnar	Ginebra	Bali	12383.0	Airbus A380
Pam_GiNu_11320	PamPangea	Ginebra	Nueva York	6206.0	Airbus A320
Air_BaGi_11380	Airnar	Bali	Ginebra	12383.0	Airbus A380
Pam_MeLo_10747	PamPangea	Melbourne	Londres	16900.0	Boeing 747
Pam_BaPa_11380	PamPangea	Bali	París	11980.0	Airbus A380
Tab_GiLo_10747	TabarAir	Ginebra	Londres	739.0	Boeing 747
Mol_CiPa_11380	MoldaviAir	Cincinnati	París	6370.0	Airbus A380
Tab_GiLo_10747	TabarAir	Ginebra	Londres	739.0	Boeing 747
Air_GiCi_10747	Airnar	Ginebra	Cincinnati	6969.0	Boeing 747
Pam_MeLo_10747	PamPangea	Melbourne	Londres	16900.0	Boeing 747
Air_BaGi_11380	Airnar	Bali	Ginebra	12383.0	Airbus A380
Mol_MeLo_10747	MoldaviAir	Melbourne	Londres	16900.0	Boeing 747
Mol_LoCa_11320	MoldaviAir	Londres	Cádiz	1716.0	Airbus A320
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103.0	Airbus A380
Mol_MeLo_11380	MoldaviAir	Melbourne	Londres	16900.0	Airbus A380
Fly_BaBa_11380	FlyQ	Barcelona	Bali	13058.0	Airbus A380
Mol_LoCa_11320	MoldaviAir	Londres	Cádiz	1716.0	Airbus A320

Mol_MeCi_10737	MoldaviAir	Melbourne	Cincinnati	15262.0	Boeing 737
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785.0	Airbus A320
Mol_BaCi_10737	MoldaviAir	Bali	Cincinnati	15011.0	Boeing 737
Mol_LoCa_11320	MoldaviAir	Londres	Cádiz	1716.0	Airbus A320
Fly_NuBa_11380	FlyQ	Nueva York	Bali	16589.0	Airbus A380
Tab_NuGi_11380	TabarAir	Nueva York	Ginebra	6206.0	Airbus A380
Air_GiBa_11380	Airnar	Ginebra	Bali	12383.0	Airbus A380
Fly_NuBa_11380	FlyQ	Nueva York	Bali	16589.0	Airbus A380
Pam_LoBa_10747	PamPangea	Londres	Bali	12553.0	Boeing 747
Air_GiCi_10747	Airnar	Ginebra	Cincinnati	6969.0	Boeing 747
Mol_PaCa_11320	MoldaviAir	París	Cádiz	1447.0	Airbus A320
Mol_LoPa_11320	MoldaviAir	Londres	París	344.0	Airbus A320
Mol_LoPa_11320	MoldaviAir	Londres	París	344.0	Airbus A320
Mol_BaLo_10747	MoldaviAir	Bali	Londres	12553.0	Boeing 747
Fly_BaBa_11380	FlyQ	Barcelona	Bali	13058.0	Airbus A380
Mol_BaCi_10737	MoldaviAir	Bali	Cincinnati	15011.0	Boeing 737
Tab_LoNu_10737	TabarAir	Londres	Nueva York	5566.0	Boeing 737

Id_vuelo	consumo_kg	duracion
Tab_GiLo_11380	NaN	626.0
Pam_GiNu_11320	16200.1424	569.0
Mol_MeLo_10747	192980.9648	1326.0
Mol_CiPa_11380	78055.432	444.0
Tab_LoNu_11380	70133.8264	391.0
Mol_BaLo_10747	137829.4294	856.0
Pam_NuBa_10747	185751.412496	1305.0
Pam_BaPa_11380	150952.792	818.0
Mol_LoCa_11320	4737.876	144.0
Tab_LoNu_11380	70133.8264	391.0
Tab_NuGi_11380	NaN	433.0
Pam_NuBa_10747	185751.412496	1305.0
Tab_LoNu_10737	13795.8876	442.0
Tab_LoLo_11320	NaN	756.0
Mol_MeLo_11380	207242.1312	1326.0
Pam_LoBa_10747	144520.1784	856.0
Tab_RoLo_10737	25221.7233	785.0
Mol_MeCi_10737	39341.52892800001	1359.0
Tab_RoLo_10737	25221.7233	785.0
Mol_PaCa_11320	3704.6094	124.0
Air_GiBa_11380	150304.854	NaN
Pam_GiNu_11320	16200.1424	569.0
Air_BaGi_11380	151736.3288	845.0
Pam_MeLo_10747	204222,5744	1326.0
Pam_BaPa_11380	150952.792	818.0
Tab_GiLo_10747	8665,514000000001	69.0
Mol_CiPa_11380	78055.432	444.0

```
Tab_GiLo_10747  8665,514000000001      69.0
Air_GiCi_10747  74289.54000000001     484.0
Pam_MeLo_10747         204222,5744    1326.0
Air_BaGi_11380         151736.3288     845.0
Mol_MeLo_10747         192980.9648    1326.0
Mol_LoCa_11320            4737.876     144.0
Tab_GiLo_11380                 NaN     626.0
Mol_MeLo_11380         207242.1312    1326.0
Fly_BaBa_11380       166407.809152    1070.0
Mol_LoCa_11320            4393.3032     144.0
Mol_MeCi_10737  39341.52892800001    1359.0
Tab_LoLo_11320                 NaN     756.0
Mol_BaCi_10737        38694.515184    1340.0
Mol_LoCa_11320            4393.3032     144.0
Fly_NuBa_11380       205422.781408    1305.0
Tab_NuGi_11380                 NaN     433.0
Air_GiBa_11380          150304.854       NaN
Fly_NuBa_11380       205422.781408    1305.0
Pam_LoBa_10747         144520.1784     856.0
Air_GiCi_10747  74289.54000000001     484.0
Mol_PaCa_11320            3704.6094     124.0
Mol_LoPa_11320              906.612      44.0
Mol_LoPa_11320              906.612      44.0
Mol_BaLo_10747         137829.4294     856.0
Fly_BaBa_11380       166407.809152    1070.0
Mol_BaCi_10737        38694.515184    1340.0
Tab_LoNu_10737          13795.8876     442.0
```

[11]:
```
df_viajes_aereos ["Aircompany"].unique()
```

[11]:
```
array(['Airnar', 'FlyQ', 'TabarAir', 'MoldaviAir', 'PamPangea'],
      dtype=object)
```

[12]:
```
a_FlyQ= df_viajes_aereos.loc[(df_viajes_aereos.Aircompany =="FlyQ") &␣
 ↪(df_viajes_aereos.duplicated(keep= False))]
#display(a_FlyQ)
print(len(a_FlyQ))
#### FlyQ tiene 4 duplicados totales
```

```
4
```

[13]:
```
a_TabarAir= df_viajes_aereos.loc[(df_viajes_aereos.Aircompany =="TabarAir") &␣
 ↪(df_viajes_aereos.duplicated(keep= False))]
#display(a_TabarAir)
print(len(a_TabarAir))
####TabarAir tiene 43 duplicados totales
```

```
14
```

[14]:
```
a_MoldaviAir= df_viajes_aereos.loc[(df_viajes_aereos.Aircompany =="MoldaviAir")␣
 ↪& (df_viajes_aereos.duplicated(keep= False))]
#display(a_MoldaviAir)
print(len(a_MoldaviAir))
#### MoldaviAir tiene 55 duplicados totales
```

20

[15]:
```
a_Airnarr= df_viajes_aereos.loc[(df_viajes_aereos.Aircompany =="Airnarr") &␣
 ↪(df_viajes_aereos.duplicated(keep= False))]
#display(a_Airnarr)
print(len(a_Airnarr))
#### Airnarr tiene 0 duplicados.
```

0

[16]:
```
a_PamPangea= df_viajes_aereos.loc[(df_viajes_aereos.Aircompany =="PamPangea") &␣
 ↪(df_viajes_aereos.duplicated(keep= False))]
#display(a_PamPangea)
print(len(a_PamPangea))
```

10

2. Es hora de limpiar. No necesariamente seguiremos estos pasos pero primero, deshazte de las filas duplicadas. Quédate con las últimas copias.

[17]:
```
ultimas =df_viajes_aereos.loc[df_viajes_aereos.duplicated(keep = "last")]
print(len(df_viajes_aereos))# El DF antes del borrado presentaba 1000 filas y␣
 ↪ahora quedan 973 filas, eliminando 77 filas duplicadas, quedandose con las␣
 ↪ultimas como buenas
print(ultimas)
```

1000

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion
Tab_GiLo_11380	TabarAir	Ginebra	Los Angeles	9103.0	Airbus A380
Pam_GiNu_11320	PamPangea	Ginebra	Nueva York	6206.0	Airbus A320
Mol_MeLo_10747	MoldaviAir	Melbourne	Londres	16900.0	Boeing 747
Mol_CiPa_11380	MoldaviAir	Cincinnati	París	6370.0	Airbus A380
Tab_LoNu_11380	TabarAir	Londres	Nueva York	5566.0	Airbus A380
Mol_BaLo_10747	MoldaviAir	Bali	Londres	12553.0	Boeing 747
Pam_NuBa_10747	PamPangea	Nueva York	Bali	16589.0	Boeing 747
Pam_BaPa_11380	PamPangea	Bali	París	11980.0	Airbus A380
Mol_LoCa_11320	MoldaviAir	Londres	Cádiz	1716.0	Airbus A320
Tab_NuGi_11380	TabarAir	Nueva York	Ginebra	6206.0	Airbus A380
Tab_LoNu_10737	TabarAir	Londres	Nueva York	5566.0	Boeing 737
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785.0	Airbus A320
Mol_MeLo_11380	MoldaviAir	Melbourne	Londres	16900.0	Airbus A380
Pam_LoBa_10747	PamPangea	Londres	Bali	12553.0	Boeing 747
Tab_RoLo_10737	TabarAir	Roma	Los Angeles	10077.0	Boeing 737

```
Mol_MeCi_10737  MoldaviAir    Melbourne   Cincinnati    15262.0   Boeing 737
Mol_PaCa_11320  MoldaviAir        París       Cádiz     1447.0  Airbus A320
Air_GiBa_11380      Airnar      Ginebra        Bali    12383.0  Airbus A380
Air_BaGi_11380      Airnar         Bali      Ginebra    12383.0  Airbus A380
Pam_MeLo_10747   PamPangea    Melbourne      Londres    16900.0   Boeing 747
Tab_GiLo_10747    TabarAir      Ginebra      Londres      739.0   Boeing 747
Air_GiCi_10747      Airnar      Ginebra   Cincinnati     6969.0   Boeing 747
Fly_BaBa_11380        FlyQ    Barcelona         Bali    13058.0  Airbus A380
Mol_LoCa_11320  MoldaviAir      Londres       Cádiz     1716.0  Airbus A320
Mol_BaCi_10737  MoldaviAir         Bali   Cincinnati    15011.0   Boeing 737
Fly_NuBa_11380        FlyQ   Nueva York         Bali    16589.0  Airbus A380
Mol_LoPa_11320  MoldaviAir      Londres        París      344.0  Airbus A320

                     consumo_kg  duracion
Id_vuelo
Tab_GiLo_11380              NaN     626.0
Pam_GiNu_11320         16200.1424     569.0
Mol_MeLo_10747        192980.9648    1326.0
Mol_CiPa_11380          78055.432     444.0
Tab_LoNu_11380          70133.8264     391.0
Mol_BaLo_10747        137829.4294     856.0
Pam_NuBa_10747      185751.412496    1305.0
Pam_BaPa_11380         150952.792     818.0
Mol_LoCa_11320           4737.876     144.0
Tab_NuGi_11380              NaN     433.0
Tab_LoNu_10737         13795.8876     442.0
Tab_LoLo_11320              NaN     756.0
Mol_MeLo_11380        207242.1312    1326.0
Pam_LoBa_10747        144520.1784     856.0
Tab_RoLo_10737         25221.7233     785.0
Mol_MeCi_10737  39341.52892800001    1359.0
Mol_PaCa_11320          3704.6094     124.0
Air_GiBa_11380         150304.854       NaN
Air_BaGi_11380        151736.3288     845.0
Pam_MeLo_10747        204222,5744    1326.0
Tab_GiLo_10747  8665,514000000001      69.0
Air_GiCi_10747  74289.54000000001     484.0
Fly_BaBa_11380      166407.809152    1070.0
Mol_LoCa_11320          4393.3032     144.0
Mol_BaCi_10737       38694.515184    1340.0
Fly_NuBa_11380      205422.781408    1305.0
Mol_LoPa_11320             906.612      44.0
```

[18]:
```
df_viajes_aereos.drop_duplicates(keep = "last" , inplace=True)

df_viajes_aereos.loc[df_viajes_aereos.duplicated(keep = False)]
```

[18]:
```
Empty DataFrame
Columns: [Aircompany, Origen, Destino, Distancia, avion, consumo_kg, duracion]
Index: []
```

[19]:
```
df_viajes_aereos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  973 non-null    object
 1   Origen      973 non-null    object
 2   Destino     973 non-null    object
 3   Distancia   845 non-null    float64
 4   avion       973 non-null    object
 5   consumo_kg  838 non-null    object
 6   duracion    827 non-null    float64
dtypes: float64(2), object(5)
memory usage: 60.8+ KB
```

3. Vamos con los nulos o NaN localizados en 1.2. Elimina las filas que tengan más de dos nulos, las podemos considerar demasiado ruidosas y las quitamos.

[20]:
```
df_viajes_aereos["Distancia"].value_counts(dropna = False)
# esta columna presenta un total de 128 NaN , por lo que procederemos al␣
 ↪elimmado de als que tiene mas 2 dos nulos
```

[20]:
```
Distancia
NaN        128
6206.0      39
11980.0     36
12383.0     35
6877.0      34
739.0       30
6969.0      28
9103.0      27
15011.0     27
16900.0     26
3073.0      26
6370.0      25
7480.0      24
1447.0      24
5566.0      24
12553.0     24
344.0       23
411.0       22
911.0       22
```

```
16925.0     21
2779.0      21
6284.0      20
16589.0     20
6624.0      18
15262.0     17
12738.0     16
12798.0     16
16674.0     14
10077.0     14
1433.0      14
20029.0     14
13058.0     13
1716.0      13
698.0       13
5835.0      12
8785.0      11
6170.0      11
12845.0     11
3944.0      10
859.0       10
660.0        9
16082.0      7
6763.0       7
1725.0       6
9099.0       6
9373.0       5
Name: count, dtype: int64
```

[21]:
```
df_viajes_aereos["duracion"].value_counts(dropna = False)
```

[21]:
```
duracion
NaN       146
845.0      25
818.0      25
1326.0     21
433.0      20
          ...
533.0       1
731.0       1
799.0       1
151.0       1
488.0       1
Name: count, Length: 117, dtype: int64
```

4. Ahora con los nulos en las columnas numéricas, ¿hay alguna columna que creas que puedes reconstruir completamente? ¿Cuál?

[22]:
```
df_viajes_aereos["consumo_kg"].value_counts(dropna = False)
```

[22]:
```
consumo_kg
NaN                  135
75328.428              4
18400.052              4
8799.1252              4
194854.5664            3
                    …
9583.92                1
45826.91360000001      1
213653.076             1
68140.63880000002      1
16872,219              1
Name: count, Length: 704, dtype: int64
```

[23]:
```
#confirmada la existencia de valores NaN,  La manera de identificar las filas␣
 ↪con valores faltantes es a través del  metodo isna()
#aplicado a las columnas en las que sabemos que hay valores nulos

df_viajes_aereos.loc[df_viajes_aereos["Distancia"].isna()]
```

[23]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Air_GiCa_11380	Airnar	Ginebra	Cádiz	NaN	Airbus A380	NaN	135.0
Tab_LoCi_10737	TabarAir	Los Angeles	Cincinnati	NaN	Boeing 737	7915,433400000001	NaN
Mol_LoBa_11380	MoldaviAir	Londres	Bali	NaN	Airbus A380	156721.6944	856.0
Air_BaGi_11380	Airnar	Bali	Ginebra	NaN	Airbus A380	146010.4296	845.0
Pam_MeBa_10737	PamPangea	Melbourne	Bali	NaN	Boeing 737	7158.148200000001	NaN
…	…	…	…	…	…	…	…
Pam_LoPa_11320	PamPangea	Londres	París	NaN	Airbus A320	949.784	NaN
Air_CiPa_11380	Airnar	Cincinnati	París	NaN	Airbus A380	79528.176	444.0
Tab_RoNu_11380	TabarAir	Roma	Nueva York	NaN	Airbus A380	85062.98840000002	478.0
Pam_BaPa_11380	PamPangea	Bali	París	NaN	Airbus A380		
Mol_PaMe_10737	MoldaviAir	París	Melbourne	NaN	Boeing 737		

```
Pam_BaPa_11380           138488.8     818.0
Mol_PaMe_10737  45766.96020000001    1485.0

[128 rows x 7 columns]
```

[24]:
```
df_viajes_aereos.loc[df_viajes_aereos["duracion"].isna()]
```

[24]:
```
                Aircompany       Origen     Destino  Distancia        avion  \
Id_vuelo
Mol_CaMe_10737  MoldaviAir        Cádiz   Melbourne    20029.0   Boeing 737
Tab_LoCi_10737    TabarAir  Los Angeles  Cincinnati        NaN   Boeing 737
Pam_MeBa_10737   PamPangea    Melbourne        Bali        NaN   Boeing 737
Fly_GiCi_10737        FlyQ      Ginebra  Cincinnati     6969.0   Boeing 737
Mol_BaMe_11320  MoldaviAir         Bali   Melbourne     2779.0  Airbus A320
…                     …           …          …         …           …
Mol_PaCi_11380  MoldaviAir        París  Cincinnati     6370.0  Airbus A380
Pam_LoPa_11320   PamPangea      Londres       París        NaN  Airbus A320
Mol_MePa_11380  MoldaviAir    Melbourne       París    16925.0  Airbus A380
Tab_LoGi_10737    TabarAir      Londres     Ginebra      739.0   Boeing 737
Air_CaCi_11320      Airnar        Cádiz  Cincinnati     6624.0  Airbus A320

                      consumo_kg  duracion
Id_vuelo
Mol_CaMe_10737  53148.15324000001       NaN
Tab_LoCi_10737  7915,433400000001       NaN
Pam_MeBa_10737  7158.148200000001       NaN
Fly_GiCi_10737                NaN       NaN
Mol_BaMe_11320  7114.795799999999       NaN
…                            …       …
Mol_PaCi_11380          79528.176       NaN
Pam_LoPa_11320            949.784       NaN
Mol_MePa_11380        215687.8672       NaN
Tab_LoGi_10737          1867.6008       NaN
Air_CaCi_11320       18328.766976       NaN

[146 rows x 7 columns]
```

[25]:
```
df_viajes_aereos.loc[df_viajes_aereos["consumo_kg"].isna()]
```

[25]:
```
                Aircompany      Origen      Destino  Distancia        avion  \
Id_vuelo
Air_PaGi_10737     Airnar       París      Ginebra      411.0   Boeing 737
Air_GiCa_11380     Airnar     Ginebra        Cádiz        NaN  Airbus A380
Tab_RoLo_10747   TabarAir        Roma  Los Angeles    10077.0   Boeing 747
Air_BaCa_10737     Airnar        Bali        Cádiz    12798.0   Boeing 737
Air_BaCi_10737     Airnar        Bali   Cincinnati    15011.0   Boeing 737
…                     …           …          …         …           …
```

```
Pam_LoNu_10737  PamPangea     Londres   Nueva York     5566.0   Boeing 737
Tab_CiRo_10747   TabarAir  Cincinnati         Roma     7480.0   Boeing 747
Air_CiPa_11380     Airnar  Cincinnati        París     6370.0  Airbus A380
Pam_BaPa_10737  PamPangea        Bali        París    11980.0   Boeing 737
Air_BaPa_11380     Airnar        Bali        París    11980.0  Airbus A380

                consumo_kg  duracion
Id_vuelo
Air_PaGi_10737         NaN      51.0
Air_GiCa_11380         NaN     135.0
Tab_RoLo_10747         NaN     691.0
Air_BaCa_10737         NaN    1171.0
Air_BaCi_10737         NaN    1340.0
…                      …        …
Pam_LoNu_10737         NaN     442.0
Tab_CiRo_10747         NaN     518.0
Air_CiPa_11380         NaN     444.0
Pam_BaPa_10737         NaN    1029.0
Air_BaPa_11380         NaN     818.0

[135 rows x 7 columns]
```

[26]:
```
# vamos aver intersecciones entre las 3
a_Distancia_NaN = df_viajes_aereos["Distancia"].isna()
a_duracion_NaN = df_viajes_aereos["duracion"].isna()
a_consumo_NaN = df_viajes_aereos["consumo_kg"].isna()
#comparamos las condicones
cruce =df_viajes_aereos.loc[a_Distancia_NaN & a_duracion_NaN & a_consumo_NaN]
# veremos intrsecciones de filas y columnas o solo en columnas o solo en filas
display(cruce)
```

```
                Aircompany      Origen  Destino  Distancia        avion  \
Id_vuelo
Pam_PaGi_10747   PamPangea       París  GInEbRa        NaN   Boeing 747
Mol_MeCa_10747  MoldaviAir   Melbourne    Cádiz        NaN   Boeing 747
Fly_CiRo_11380        FlyQ  Cincinnati     Roma        NaN  Airbus A380

                consumo_kg  duracion
Id_vuelo
Pam_PaGi_10747         NaN       NaN
Mol_MeCa_10747         NaN       NaN
Fly_CiRo_11380         NaN       NaN
```

Como observamos el cruce de las 3 filas con valores NaN se cruzan en solo 3 elementos

[27]:
```
# una vez comprobado los valores NaN, procedemos al eliminado de las filas que␣
 ↪tengan mas de dos nulos
nulos = 2
```

```
# filtra en el DF con el metodo isnull, el cual cambiara los valores que no son␣
 ↪NaN a false y el resto a True, despues suma los elementos en el eje␣
 ↪horonzintal(True),
# y si el recuento de NaNs por cada fila es >= al valor de la variable nulos␣
 ↪permitidos, procedera a elimnarlas
df_viajes_aereos_Nan_2 = df_viajes_aereos.loc[df_viajes_aereos.isna().
 ↪sum(axis=1) <= nulos]


# Imprime el DataFrame resultante
print(f"El DataFrame después de eliminar filas con más de dos NaN(not a number)␣
 ↪es: \n {df_viajes_aereos_Nan_2}")
print(len(df_viajes_aereos_Nan_2))
```

```
El DataFrame después de eliminar filas con más de dos NaN(not a number) es:
                 Aircompany      Origen     Destino  Distancia       avion  \
Id_vuelo
Air_PaGi_10737      Airnar       París     Ginebra      411.0  Boeing 737
Fly_BaRo_10737        FlyQ        Bali        Roma    12738.0  Boeing 737
Mol_PaCi_10737  MoldaviAir       París  Cincinnati     6370.0  Boeing 737
Tab_CiRo_10747    TabarAir  Cincinnati        Roma     7480.0  Boeing 747
Mol_CaMe_10737  MoldaviAir       Cádiz   Melbourne    20029.0  Boeing 737
…                     …         …          …        …          …
Pam_LoNu_10747   PamPangea     Londres  Nueva York     5566.0  Boeing 747
Mol_MeLo_10747  MoldaviAir   Melbourne     Londres    16900.0  Boeing 747
Mol_BaPa_10747  MoldaviAir        Bali       París    11980.0  Boeing 747
Air_CaCi_10747      Airnar       Cádiz  Cincinnati     6624.0  Boeing 747
Air_PaCi_10737      Airnar       París  Cincinnati     6370.0  Boeing 737

                       consumo_kg  duracion
Id_vuelo
Air_PaGi_10737                NaN      51.0
Fly_BaRo_10737  33479.13254400001    1167.0
Mol_PaCi_10737           17027.01     503.0
Tab_CiRo_10747          86115.744     518.0
Mol_CaMe_10737  53148.15324000001       NaN
…                             …       …
Pam_LoNu_10747          62300,238     391.0
Mol_MeLo_10747        194854.5664    1326.0
Mol_BaPa_10747         128983.868     818.0
Air_CaCi_10747         72024.0768     461.0
Air_PaCi_10737          16872,219     503.0

[970 rows x 7 columns]
970
```

[ ]:

5. Sustituye los valores nulos por valores medios en las columnas numéricas que no puedas reconstruir. Se valorará hacerlo con más precisión que utilizar la media global, ¿se te ocurre como hacerlo más preciso?

[28]:
```
df_viajes_aereos.describe()
```

[28]:

	Distancia	duracion
count	845.000000	827.000000
mean	8058.820118	647.540508
std	5495.837382	459.060596
min	344.000000	42.000000
25%	3073.000000	224.000000
50%	6877.000000	549.000000
75%	12553.000000	1049.000000
max	20029.000000	1721.000000

[29]:
```
df_viajes_aereos.loc[df_viajes_aereos["Distancia"].isna(), "Distancia"] =␣
↪df_viajes_aereos["Distancia"].mean()
df_viajes_aereos["Distancia"].info()# ya no tiene valores NaN
```

```
<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: Distancia
Non-Null Count  Dtype
--------------  -----
973 non-null    float64
dtypes: float64(1)
memory usage: 15.2+ KB
```

[30]:
```
df_viajes_aereos.loc[df_viajes_aereos["duracion"].isna(), "duracion"] =␣
↪df_viajes_aereos["duracion"].mean()
df_viajes_aereos["duracion"].info()
```

```
<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: duracion
Non-Null Count  Dtype
--------------  -----
973 non-null    float64
dtypes: float64(1)
memory usage: 15.2+ KB
```

6. En el caso de las columna no numéricas con nulos, ¿crees que puedes reconstruirlas? Si no es así, bórralas. [Se puede, pero no es sencillo]

[31]:
```
df_viajes_aereos.columns
```

[31]:
```
Index(['Aircompany', 'Origen', 'Destino', 'Distancia', 'avion', 'consumo_kg',
       'duracion'],
      dtype='object')
```

[32]:
```
df_viajes_aereos["Origen"].info()# no tiene NaN
print()
df_viajes_aereos["Destino"].info()# no tiene NaN
print()
df_viajes_aereos["consumo_kg"].info()# tene 135 NaN, que pueden ser valores␣
 ↪nulos, o alguna coma en vez de un punto en el decimal
#.mode()
```

```
<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: Origen
Non-Null Count  Dtype
--------------  -----
973 non-null    object
dtypes: object(1)
memory usage: 15.2+ KB

<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: Destino
Non-Null Count  Dtype
--------------  -----
973 non-null    object
dtypes: object(1)
memory usage: 15.2+ KB

<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: consumo_kg
Non-Null Count  Dtype
--------------  -----
838 non-null    object
dtypes: object(1)
memory usage: 15.2+ KB
```

[33]:
```
# al manipular un DF a veces da problemas y es interesante resetar la etiqueta␣
 ↪de valores indice. Aqui nos pasa eso , al preguntar por valores NaN me daba␣
 ↪error (df_viajes_aereos.loc[df_viajes_aereos.consumo_kg.isna()])por  lo que␣
 ↪primero reseteamos el index
# para intentar corregir esta anomalia y poder operar con la columna de cosumo␣
 ↪con NaN
df_reset = df_viajes_aereos.reset_index(drop=True)# no queremos crear nueva␣
 ↪columna solo descartar indice anterior
```

```
# Extraer la columna 'Nombre' como una Serie
consumo_kg = df_reset["consumo_kg"]

# Utiliza isna() para verificar los valores NaN en la Serie
NaN = consumo_kg.isna()

# Imprime los valores resultantes
print(f"Valores NaN en la columna 'consumo_kg':\n {NaN}")
```

```
Valores NaN en la columna 'consumo_kg':
 0       True
1      False
2      False
3      False
4      False
       …
968    False
969    False
970    False
971    False
972    False
Name: consumo_kg, Length: 973, dtype: bool
```

[34]:
```
print(len(df_viajes_aereos.loc[df_viajes_aereos.consumo_kg.isna()]))
df_viajes_aereos.loc[df_viajes_aereos.consumo_kg.isna()]# tiene esta columna
 ↪135 valores NaN
```

```
135
```

[34]:

Id_vuelo	Aircompany	Origen	Destino	Distancia	avion	consumo_kg	duracion
Air_PaGi_10737	Airnar	París	Ginebra	411.000000	Boeing 737	NaN	51.0
Air_GiCa_11380	Airnar	Ginebra	Cádiz	8058.820118	Airbus A380	NaN	135.0
Tab_RoLo_10747	TabarAir	Roma	Los Angeles	10077.000000	Boeing 747		
Air_BaCa_10737	Airnar	Bali	Cádiz	12798.000000	Boeing 737		
Air_BaCi_10737	Airnar	Bali	Cincinnati	15011.000000	Boeing 737		
…	…	…	…	…	…		
Pam_LoNu_10737	PamPangea	Londres	Nueva York	5566.000000	Boeing 737		
Tab_CiRo_10747	TabarAir	Cincinnati	Roma	7480.000000	Boeing 747		
Air_CiPa_11380	Airnar	Cincinnati	París	6370.000000	Airbus A380		
Pam_BaPa_10737	PamPangea	Bali	París	11980.000000	Boeing 737		
Air_BaPa_11380	Airnar	Bali	París	11980.000000	Airbus A380		

```
Tab_RoLo_10747        NaN     691.0
Air_BaCa_10737        NaN    1171.0
Air_BaCi_10737        NaN    1340.0
…                     …       …
Pam_LoNu_10737        NaN     442.0
Tab_CiRo_10747        NaN     518.0
Air_CiPa_11380        NaN     444.0
Pam_BaPa_10737        NaN    1029.0
Air_BaPa_11380        NaN     818.0

[135 rows x 7 columns]
```

[35]:
```
df_viajes_aereos["consumo_kg"].value_counts(dropna = False)
```

[35]:
```
consumo_kg
NaN                  135
75328.428              4
18400.052              4
8799.1252              4
194854.5664            3
                    …
9583.92                1
45826.91360000001      1
213653.076             1
68140.63880000002      1
16872,219              1
Name: count, Length: 704, dtype: int64
```

[36]:
```
# una vez comprobado los valores NaN, y al no poder ni hacer la media ni la␣
 ↪moda, y observar que  tiene caracteres no compatible ("," en vez de ".",
#asi que procedemos ello

# Reemplazar comas por puntos en los valores decimales de la columna␣
 ↪'consumo_kg'
df_viajes_aereos['consumo_kg'] = df_viajes_aereos['consumo_kg'].str.
 ↪replace(',', '.')

# Convertir la columna 'consumo_kg' a tipo numérico, a traves de la funcion␣
 ↪to_numeric() que cambia argumentos a tipo numerico, manejando los posibles␣
 ↪errores que pueda
# dar por tipo ,se establen los valores no numericos como NaN, en lugar de␣
 ↪generar error con (errors = "coerce")
df_viajes_aereos['consumo_kg'] = pd.to_numeric(df_viajes_aereos['consumo_kg'],␣
 ↪errors='coerce')
# Calculo la media de consumo_kg
media_consumo = df_viajes_aereos['consumo_kg'].mean()
# Reemplaza los NaN con la media
```

```
df_viajes_aereos.loc[df_viajes_aereos["consumo_kg"].isna(), "consumo_kg"] =␣
 ↪df_viajes_aereos["consumo_kg"].mean()
# Imprime el DataFrame resultante
print(df_viajes_aereos.consumo_kg)
```

```
Id_vuelo
Air_PaGi_10737     66630.083667
Fly_BaRo_10737     33479.132544
Mol_PaCi_10737     17027.010000
Tab_CiRo_10747     86115.744000
Mol_CaMe_10737     53148.153240
                       …
Pam_LoNu_10747     62300.238000
Mol_MeLo_10747    194854.566400
Mol_BaPa_10747    128983.868000
Air_CaCi_10747     72024.076800
Air_PaCi_10737     16872.219000
Name: consumo_kg, Length: 973, dtype: float64
```

[37]:
```
print(len(df_viajes_aereos.consumo_kg))
```

```
973
```

[38]:
```
df_viajes_aereos["consumo_kg"].info()
```

```
<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: consumo_kg
Non-Null Count  Dtype
--------------  -----
973 non-null    float64
dtypes: float64(1)
memory usage: 15.2+ KB
```

7. Ahora sobre el `DataFrame` resultante, corrije las columnas que tengan valores ”sucios” (comas equivocadas, valores extraños que no nulos, etc)

En el punto anteriro ya he realziado este punto 7, cambiando las comas que tenia la columna consumo_kg por puntos y asi poder sustituir por la media , los 135 valores NaN que tenia anteriormente

0.4 #2 Análisis de Datos

Una vez ”limpios” ya podemos analizar los datos y/o manipularlos. De hecho, nos piden que:

1. Clasifiquemos los vuelos en Larga Distancia (LD), Media Distancia (MD) y Regionales (R). Con los siguientes límites: más de 9000km, entre 1500 y 9000 kms y resto de vuelos. Añade una columna al `DataFrame` que tenga esa clasificación por vuelo.

[39]:
```
df_viajes_aereos.head()
```

[39]:
```
                Aircompany      Origen     Destino  Distancia       avion  \
Id_vuelo
Air_PaGi_10737      Airnar       París     Ginebra      411.0  Boeing 737
Fly_BaRo_10737        FlyQ        Bali        Roma    12738.0  Boeing 737
Mol_PaCi_10737  MoldaviAir       París  Cincinnati     6370.0  Boeing 737
Tab_CiRo_10747    TabarAir  Cincinnati        Roma     7480.0  Boeing 747
Mol_CaMe_10737  MoldaviAir       Cádiz   Melbourne    20029.0  Boeing 737

                  consumo_kg     duracion
Id_vuelo
Air_PaGi_10737  66630.083667    51.000000
Fly_BaRo_10737  33479.132544  1167.000000
Mol_PaCi_10737  17027.010000   503.000000
Tab_CiRo_10747  86115.744000   518.000000
Mol_CaMe_10737  53148.153240   647.540508
```

[40]:
```
# para ello, vamos a utilizar la funcion apply, y para ello primero haremo suna␣
 ↪funcion para subduivir los valor de la columna Distancia

def avioneta(Distancia):
    if Distancia > 9000.0:
            clase= "LD"
    elif Distancia >= 1500.0:
        clase = "MD"
    else:
        clase= "R"
    return clase

#aplicamos la funcion al DF

df_viajes_aereos["Clase"] = df_viajes_aereos["Distancia"].apply(avioneta)
```

[41]:
```
#df_viajes_aereos
```

2. Crea ahora una columna que recoja el índice de contaminación de cada viaje calculado como el consumo de keroseno por kilometro recorrdio. Llama IC a esa columna.

[42]:
```
df_viajes_aereos["IC"] = (df_viajes_aereos.consumo_kg)/df_viajes_aereos.
 ↪Distancia
df_viajes_aereos["IC"].info()
```

```
<class 'pandas.core.series.Series'>
Index: 973 entries, Air_PaGi_10737 to Air_PaCi_10737
Series name: IC
Non-Null Count  Dtype
--------------  -----
973 non-null    float64
dtypes: float64(1)
```

```
memory usage: 15.2+ KB
```

[43]:
```
#df_viajes_aereos
```

3. Con vistas a un futuro impuesto basado en emisiones, nos piden obtener las 3 empresas más contaminantes. Hazlo de forma manual y utilizando alguno de los métodos siguientes: `nlargest`, `nsmallest` [Nota: Podemos considerar como más contaminante las que mayor IC global tiene]

Las tres empresas aereas mas contaminantes del dataframe son:

1. Las empresas Airnar y PamPangea con el mismo valor de IC = 162.116992
2. La empresa FlyQ con un valor de IC = 100.954672
3. La empresa TabarAir con un valor de IC = 95.458573

Las tres empresas aereas menos contaminantes del dataframe son:

1. las empresas MoldaviAir y PamPangea con valores de IC = 0.114642
2. La empresa Airnar con valores de iC = 0.139531
3. La empresa FlyQ con valores de Ic = 0.212943

[44]:
```
# los 3 valores maximos
masca = df_viajes_aereos["IC"].nlargest(3)
print(masca)
print()
# los tres valores minimos
menos_masca = df_viajes_aereos["IC"].nsmallest(3)
print(menos_masca)
#como al info me la da con el indice de vuelos y no empresas, voy a realziar un␣
 ↪subconjunto con  dos columnas y de hay los ordenar descen y ascendente
#para obtener los 3 mayores y los 3 menores
sub_lista= df_viajes_aereos[["Aircompany","IC"]]
#display(sub_lista)
#sub_lista_ord = sub_lista.sort_values(by= "IC", ascending = False)
sub_lista_ord = sub_lista.sort_values(by= "IC", ascending = True)
display(sub_lista_ord.head())
#sub_lista_ord.head(30)
```

```
Id_vuelo
Air_PaGi_10737    162.116992
Air_PaGi_11380    162.116992
Pam_GiPa_10747    162.116992
Name: IC, dtype: float64

Id_vuelo
Mol_LoPa_11320    0.113571
Mol_PaLo_11320    0.114642
Pam_LoPa_11320    0.117856
Name: IC, dtype: float64
```

	Aircompany	IC

```
Id_vuelo
Mol_LoPa_11320  MoldaviAir  0.113571
Mol_PaLo_11320  MoldaviAir  0.114642
Pam_LoPa_11320   PamPangea  0.117856
Pam_GiPa_11320   PamPangea  0.134411
Air_GiPa_11320      Airnar  0.139531
```

[ ]:

4. Continuando con nuestro análisis para futuras acciones impositivas, muestra ahora las empresas más contaminantes por categoría de vuelo.

Las tres empresas aereas mas contaminantes por clase de vuelo del dataframe son:

1. **En MD:** 1 - la empresa MoldaviAir con valores de IC = 38.828720 _____ 2 - La empresa Airnar con valores de IC = 38.626135 _____ 3 - Las empresa PamPangea con valores de IC = 26.511707
2. **En LD:** 1 - La empresas MoldaviAir y PamPangea con valores de IC = 13.224640 _____ 2 - La empresa TabarAir con valores de IC = 13.124924 _____ 3 - La empresa FlyQ con valores de IC= 13.104416
3. **En R:** 1 - Las empresas Airnar, PamPangea con valores de IC = 162.116992 _____ 2 - La empresa FlyQ con valores de IC = 100.954672 _____ 3 - La empresa TabarAir con valores de IC =95.458573

Las tres empresas aereas menos contaminantes por clase de vuelo del dataframe son:

1. **En MD:** 1 - La empresa MoldaviAir con valores de IC = 0.113571 _____ 2 - La empresa PamPangea con valores de IC = 0.117856 _____ 3 - La empresa Airnar con valores de IC = 0.139531
2. **En LD:** 1 - La empresa TabarAir con valores de IC = 2.454300 _____ 2 - La empresa Airnar con valores de IC = 2.478600 _____ 3 - La empresa TabarAir con valores de IC = 2.502900
3. **En R:** 1 - Las empresas TabarAir, MoldaviAir con valores de IC = 2.430000 _____ 2 - Las empresas PamPangea, Airnar y FlayQ con valores de IC= 2.454300 _____ 3 - Las empresas FlyQ, TabarAir y MoldaviAir con valores de IC = 2.478600

[45]:
```
#como al info me la da con el indice de vuelos y no empresas, voy a realziar un␣
 ↪subconjunto con  dos columnas y de hay los ordenar descen y ascendente
#para obtener los 3 mayores y los 3 menores
sub_lista= df_viajes_aereos[["Aircompany","Clase","IC"]]
#display(sub_lista)
#sub_lista_ord = sub_lista.sort_values(by= "IC", ascending = False)
sub_lista_ord = sub_lista.sort_values(by= "IC", ascending = True)
display(sub_lista_ord.head())
#sub_lista_ord.tail(900)
```

```
                Aircompany Clase        IC
Id_vuelo
Mol_LoPa_11320  MoldaviAir    MD  0.113571
Mol_PaLo_11320  MoldaviAir    MD  0.114642
```

```
Pam_LoPa_11320   PamPangea    MD  0.117856
Pam_GiPa_11320   PamPangea    MD  0.134411
Air_GiPa_11320      Airnar    MD  0.139531
```

[ ]:

5. Obtener los destinos con vuelos más contaminantes (utiliza el IC acumulado por destino y divídelo por el número de viajes a ese destino)

--Los vuelos mas contaminates por destino teniendo en cuenta su IC son:

1. El destino a **Barcelona** con valores de IC= 100.954672 y numero de vuelos = **26**(21 como Barcelona y 5 como BARcelOnA(FlyQ)), siendo su *tasa* IC/n_vuelos = $3,88$
2. El destino a **Roma** con valores de IC= 95.458573 y numero de vuelos = **83**, siendo su *tasa* IC/n_vuelos = 162,12 = 1.15
3. El destino a **Paris** con valores de IC= 162.116992 y numero de vuelos = **108**, siendo su *tasa* IC/n_vuelos = 162,12 = 1.50
4. El destino a **Ginebra** con valores de IC= 162.116992 y numero de vuelos = **100**, siendo su *tasa* IC/n_vuelos = $1,62$

[46]:
```
subconj = df_viajes_aereos[["Destino", "IC"]]

subconj_ord = subconj.sort_values(by= "IC", ascending = False)
```

[47]:
```
subconj_ord.head(60)
```

[47]:

Id_vuelo	Destino	IC
Air_PaGi_10737	Ginebra	162.116992
Pam_GiPa_10747	París	162.116992
Air_PaGi_11380	GINEbRA	162.116992
Fly_BaGi_11380	Ginebra	100.954672
Fly_GiBa_10747	BARcelOnA	100.954672
Tab_GiRo_10737	Roma	95.458573
Fly_RoGi_10747	Ginebra	95.458573
Tab_GiLo_11380	Londres	90.162495
Pam_GiLo_11380	Londres	90.162495
Tab_LoGi_11380	GINEBrA	90.162495
Tab_GiLo_10747	Londres	90.162495
Fly_BaRo_10737	Roma	77.567036
Fly_RoBa_10747	Barcelona	77.567036
Fly_BaRo_10747	Roma	77.567036
Tab_NuCi_10747	Cincinnati	73.139499
Tab_CiNu_10737	Nueva York	73.139499
Fly_NuCi_11320	Cincinnati	73.139499
Tab_LoRo_10747	Roma	46.496918
Air_PaCa_11320	Cádiz	46.047052
Mol_PaCa_11320	Cádiz	46.047052
Air_CaPa_11380	París	46.047052

```
Mol_LoCa_11380        Cádiz   38.828720
Mol_LoCa_10737        Cádiz   38.828720
Mol_CaLo_11320      Londres   38.828720
Air_CaGi_11380      Ginebra   38.626135
Air_CaGi_11320      Ginebra   38.626135
Mol_CaMe_11380    Melbourne   32.867878
Pam_MeGi_11380      GINeBrA   27.113526
Pam_MePa_11380        París   26.511707
Mol_MeLo_11380      Londres   25.716188
Pam_LoMe_11380    Melbourne   25.464068
Pam_LoMe_10747    Melbourne   25.109007
Mol_PaMe_10747    Melbourne   24.214812
Pam_MePa_10747        París   24.214812
Mol_MeLo_10747      Londres   24.179044
Pam_NuMe_10747    Melbourne   24.114910
Pam_MePa_10747        París   23.981977
Mol_MeBa_10737         Bali   23.976281
Mol_BaMe_10737    MELboURnE   23.976281
Pam_NuBa_10747         Bali   23.277668
Tab_LoCi_10747   Cincinnati   21.682422
Tab_LoCi_11320   Cincinnati   21.682422
Air_CiLo_10747  Los Angeles   21.682422
Air_LoCi_10737   Cincinnati   21.682422
Air_CiLo_11320  Los Angeles   21.682422
Fly_BaBa_11380         Bali   21.038760
Mol_LoBa_11380         Bali   19.447226
Air_CaBa_10747         Bali   19.190562
Mol_CaBa_10747         Bali   19.014501
Air_LoBa_10747         Bali   18.554211
Air_BaGi_11380      Ginebra   18.118090
Mol_BaLo_11380      Londres   18.006691
Pam_BaGi_11380      Ginebra   17.940462
Mol_BaLo_10747      Londres   17.933168
Air_PaBa_11380         Bali   17.700291
Mol_LoBa_10747         Bali   17.435025
Mol_BaPa_10747        París   17.431519
Pam_GiBa_10747         Bali   17.198910
Air_BaGi_10747      GInEBrA   17.198910
Pam_BaPa_11380        París   17.184749
```

[ ]:

[48]:
```
numero_destino_G = df_viajes_aereos.loc[df_viajes_aereos["Destino"] ==␣
 ↪"Ginebra"]
#print(numero_destino_G)
print(numero_destino_G["Destino"].value_counts())
print()
```

```
numero_destino_P = df_viajes_aereos.loc[df_viajes_aereos["Destino"] == "París"]
numero_destino_P
print(numero_destino_P["Destino"].value_counts())
print()
numero_destino_B = df_viajes_aereos.loc[df_viajes_aereos["Destino"] ==␣
 ↪"BARcelOnA"]
numero_destino_B2 = df_viajes_aereos.loc[df_viajes_aereos["Destino"] ==␣
 ↪"Barcelona"]
#print(numero_destino_B2)
print(numero_destino_B["Destino"].value_counts())
print(numero_destino_B2["Destino"].value_counts())
print()
numero_destino_R = df_viajes_aereos.loc[df_viajes_aereos["Destino"] == "Roma"]
numero_destino_R
print(numero_destino_R["Destino"].value_counts())
```

```
Destino
Ginebra    100
Name: count, dtype: int64

Destino
París    108
Name: count, dtype: int64

Destino
BARcelOnA    1
Name: count, dtype: int64
Destino
Barcelona    21
Name: count, dtype: int64

Destino
Roma    83
Name: count, dtype: int64
```

[ ]:

6. Para cada uno de los dos destinos anteriores encuentra la compañía que más viaja a cada uno (por seperado) y la que más viaja a los dos (considerandolos juntos, es decir sumando los viajes de cada compañía a los dos). Si no has sabido hacer el apartado 5. escoge dos ciudades destino para hacer este apartado.

6.1 La compañia que mas numero de vuelos a los destinos del punto anterior (Barcelona, Roma , París y Ginebra) fue Paris con un total de vuelos de 108, calculo realizado en el punto anterior. **6.2** La empresa con los 2 vuelos mas contaminantes en relacion IC/n_vuelos_destino(Barcelona y Roma), han sido fletado por las empresas: 1. El vuelo Fly_GiBa_10747 con destino a **Barcelona** pertenece a la empresa **FlyQ**, la cual ha tenido a nivel mundial un total de **171 vuelos** a todos los destinos. 2. El vuelo Tab_GiRo_10737 con destino a **Roma** pertenece a la empresa **TabarAir**, la cual ha tenido a nivel mundial un total de **222 vuelos** a todos los destinos.

CONCLUSION.- La compañia que mas ha viajado a nivel mundial, que se encuentra entre las empresas con vuelos mas contaminantes es **TabarAir**

[49]:
```
subconj2 = df_viajes_aereos[["Aircompany", "Destino", "IC" ]]

subconj2_ord = subconj2.sort_values(by= "IC", ascending = False)
#print(subconj2_ord)
numero_total = df_viajes_aereos.loc[df_viajes_aereos["Aircompany"] =="FlyQ"]
numero_total
print(numero_total["Destino"].value_counts())
print()
numero_total2 = df_viajes_aereos.loc[df_viajes_aereos["Aircompany"]␣
 ↪=="TabarAir"]
numero_total2
print(numero_total2["Destino"].value_counts())
```

```
Destino
Roma          39
Nueva York    33
Bali          26
Ginebra       22
Barcelona     21
Cincinnati    19
GINebra        2
BaRCelONa      2
BARcelOnA      1
GinEbra        1
GInebRA        1
GINEbRa        1
BARCEloNA      1
GInEBrA        1
BarcELONa      1
Name: count, dtype: int64

Destino
Roma          44
Nueva York    42
Londres       39
Los Angeles   32
Cincinnati    28
Ginebra       28
GIneBra        1
GINebRa        1
GiNEbRa        1
GinebrA        1
GINEBra        1
GInebra        1
GINEBrA        1
```

```
GiNeBra         1
GInEBra         1
Name: count, dtype: int64
```

0.5 #3 Ampliación y reanálisis [BONUS]

Esta parte no cuenta para la valoración de la práctica pero se recomienda encarecidamente que la INTENTES para poder sacar más provecho a la sesión en vivo. Para empezar, carga dos nuevos dataframes con una ampliación de más viajes y con más información sobre estos, respectivamente

[50]:
```
df_viajes_extra = pd.read_csv("./data/dataset_viajes_extra.csv", index_col =␣
↪"Id_vuelo")
df_ingresos = pd.read_csv("./data/dataset_ingresos.csv")
```

1. Emplea la función concat de pandas que permite concatenar dataframes, de manera que añadas los viajes de df_viajes_extra al dataframe que hayas utilizado hasta ahora.

[51]:
```
display(df_viajes_extra)# antes de la concatenacion
print(df_viajes_extra.info())
print()
print("Antes de la concatenacion el df viejes_extra no tenia ningun NaN")
print()
df_nuevos_viajes = pd.concat([df_viajes_aereos + df_viajes_extra])
display(df_nuevos_viajes)#despues de la concatenacion
df_nuevos_viajes.info()
print()
print("Depues de la concatenacion, el dataframe nuevos viajes contiene un total␣
↪de 1098 elementos, de los cuales 626 son nulos y en las clases 'IC' y␣
↪'Clase' son todos NaN")
```

	Aircompany	Origen	Destino	Distancia	avion	\
Id_vuelo						
Fly_BaNu_10737	FlyQ	Barcelona	Nueva York	6170	Boeing 737	
Mol_LoPa_10747	MoldaviAir	Londres	París	344	Boeing 747	
Fly_RoBa_11380	FlyQ	Roma	Bali	12738	Airbus A380	
Tab_RoGi_10737	TabarAir	Roma	Ginebra	698	Boeing 737	
Air_BaGi_10737	Airnar	Bali	Ginebra	12383	Boeing 737	
…	…	…	…	…	…	
Tab_LoLo_11320	TabarAir	Los Angeles	Londres	8785	Airbus A320	
Mol_CiLo_10737	MoldaviAir	Cincinnati	Londres	6284	Boeing 737	
Fly_RoCi_11320	FlyQ	Roma	Cincinnati	7480	Airbus A320	
Tab_RoLo_10747	TabarAir	Roma	Londres	1433	Boeing 747	
Air_PaLo_10737	Airnar	París	Los Angeles	9099	Boeing 737	

	consumo_kg	duracion
Id_vuelo		
Fly_BaNu_10737	15143.031000	488
Mol_LoPa_10747	3740.380800	42
Fly_RoBa_11380	159031.382400	869

```
Tab_RoGi_10737    1797.908400        73
Air_BaGi_10737   31607.260776      1140
…                         …       …
Tab_LoLo_11320   24766.953120       756
Mol_CiLo_10737   16491.729600       497
Fly_RoCi_11320   19721.049920       662
Tab_RoLo_10747   15734.053400       115
Air_PaLo_10737   22331.675700       711

[200 rows x 7 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 200 entries, Fly_BaNu_10737 to Air_PaLo_10737
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  200 non-null    object
 1   Origen      200 non-null    object
 2   Destino     200 non-null    object
 3   Distancia   200 non-null    int64
 4   avion       200 non-null    object
 5   consumo_kg  200 non-null    float64
 6   duracion    200 non-null    int64
dtypes: float64(1), int64(2), object(4)
memory usage: 12.5+ KB
None
```

Antes de la concatenacion el df viejes_extra no tenia ningun NaN

```
                 Aircompany  Clase     Destino  Distancia  IC    Origen  \
Id_vuelo
Air_BaCa_10737           NaN    NaN         NaN        NaN NaN       NaN
Air_BaCa_10737           NaN    NaN         NaN        NaN NaN       NaN
Air_BaCa_10737           NaN    NaN         NaN        NaN NaN       NaN
Air_BaCa_10747  AirnarAirnar    NaN  CádizCádiz    25596.0 NaN  BaliBali
Air_BaCa_10747  AirnarAirnar    NaN  CádizCádiz    25596.0 NaN  BaliBali
…                      …    …         …        …  ..       …
Tab_RoNu_10747           NaN    NaN         NaN        NaN NaN       NaN
Tab_RoNu_11320           NaN    NaN         NaN        NaN NaN       NaN
Tab_RoNu_11380           NaN    NaN         NaN        NaN NaN       NaN
Tab_RoNu_11380           NaN    NaN         NaN        NaN NaN       NaN
Tab_RoNu_11380           NaN    NaN         NaN        NaN NaN       NaN

                           avion     consumo_kg  duracion
Id_vuelo
Air_BaCa_10737               NaN            NaN       NaN
Air_BaCa_10737               NaN            NaN       NaN
Air_BaCa_10737               NaN            NaN       NaN
```

```
Air_BaCa_10747  Boeing 747Boeing 747  297955.869120    2106.0
Air_BaCa_10747  Boeing 747Boeing 747  293699.356704    2106.0
…                                …            …       …
Tab_RoNu_10747                   NaN            NaN       NaN
Tab_RoNu_11320                   NaN            NaN       NaN
Tab_RoNu_11380                   NaN            NaN       NaN
Tab_RoNu_11380                   NaN            NaN       NaN
Tab_RoNu_11380                   NaN            NaN       NaN

[1098 rows x 9 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1098 entries, Air_BaCa_10737 to Tab_RoNu_11380
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Aircompany  472 non-null    object
 1   Clase       0 non-null      float64
 2   Destino     472 non-null    object
 3   Distancia   472 non-null    float64
 4   IC          0 non-null      float64
 5   Origen      472 non-null    object
 6   avion       472 non-null    object
 7   consumo_kg  472 non-null    float64
 8   duracion    472 non-null    float64
dtypes: float64(5), object(4)
memory usage: 85.8+ KB
```

```
Depues de la concatenacion, el dataframe nuevos viajes contiene un total de 1098
elementos, de los cuales 626 son nulos y en las clases 'IC' y 'Clase' son todos
NaN
```

2. Comprueba si existen nulos en las columnas IC y todas las columnas que hayas añadido a tu dataframe previo al concat, y después del concat (utiliza info o value_counts con el argumento dropna con el valor adecuado). Tendrá que haberlos ya que el dataset de viajes extra no tenía esas columnas.

Si, Las dos columnas creadas durante las gestiones realizadas con el DataFrame(DF) ”df_viajes_aereos”, las cuales contia valores str y float, despues de la concatecion con el DF ”df_viajes_extra”, todas las columnas esta llenas de elementos Nulos, como se puede observar tras la comprobacion, hecho producdo por que el DF viajes extra es de mayor tamaña al viajes aereos, con lo que el mas pequeño se une al mas grande y si existen algunas columnas que no estan en el otro, les asiganara automaticamte valores NaN en el nuevo DF creado tras la concatenacion

[52]:
```
df_nuevos_viajes["IC"].value_counts(dropna = False)
print(df_nuevos_viajes["IC"].value_counts(dropna = False))
print()
df_nuevos_viajes["Clase"].value_counts(dropna = False)
print(df_nuevos_viajes["Clase"].value_counts(dropna = False))
```

```
IC
NaN    1098
Name: count, dtype: int64

Clase
NaN    1098
Name: count, dtype: int64
```

3. Calcula la columna IC para los vuelos que no la tengan informada (o sea que tengan NaN)

[53]:
```
# IC = consumo_kg / Distancia, al ser toda la clumna NaN en ic se calculara␣
 ↪pora todos los vuelos

df_nuevos_viajes["Distancia"].value_counts(dropna = False)# total de NaN son 626
print()
df_nuevos_viajes["IC"].value_counts(dropna = False)# 1098 NaN (todas la columna)
print()
df_nuevos_viajes["consumo_kg"].value_counts(dropna = False)# hay un total de␣
 ↪626 NaN
```

[53]:
```
consumo_kg
NaN               626
116134.418800       3
68245.924032        3
162663.072000       2
136148.454000       2
                 ...
303337.636800       1
307755.175200       1
159123.325192       1
295146.797088       1
7589.311300         1
Name: count, Length: 438, dtype: int64
```

[54]:
```
# una vez comprobado los valores NaN delas columnas necesarias, procedemos a␣
 ↪realizar el cambio de valores NaN de las columnas Distancia y consumo a sus␣
 ↪valores medios

# Calculo la media de consumo_kg
media_consumo = df_nuevos_viajes['consumo_kg'].mean()
# me muestra todos los NaN cambiados a True
NaN_True= df_nuevos_viajes.consumo_kg.isna()
# Reemplaza los NaN-False con la media
df_nuevos_viajes.loc[NaN_True, "consumo_kg"] = df_nuevos_viajes["consumo_kg"].
 ↪mean()
```

```
# Imprime el DataFrame resultante
print(df_nuevos_viajes.consumo_kg)
```

```
Id_vuelo
Air_BaCa_10737    148290.396029
Air_BaCa_10737    148290.396029
Air_BaCa_10737    148290.396029
Air_BaCa_10747    297955.869120
Air_BaCa_10747    293699.356704
                        …
Tab_RoNu_10747    148290.396029
Tab_RoNu_11320    148290.396029
Tab_RoNu_11380    148290.396029
Tab_RoNu_11380    148290.396029
Tab_RoNu_11380    148290.396029
Name: consumo_kg, Length: 1098, dtype: float64
```

[55]:
```
#vamos a realizar los cambios de Nan a valor media en la columna Distancia

#calculo de la media
media_Distancia = df_nuevos_viajes['Distancia'].mean()
# me muestra todos los NaN cambiados a True
NaN_True2= df_nuevos_viajes.Distancia.isna()
# Reemplaza los NaN con la media
df_nuevos_viajes.loc[NaN_True2, "Distancia"] = df_nuevos_viajes["Distancia"].
 ↪mean()
# Imprime el DataFrame resultante
print(df_nuevos_viajes.Distancia)
```

```
Id_vuelo
Air_BaCa_10737    17684.77658
Air_BaCa_10737    17684.77658
Air_BaCa_10737    17684.77658
Air_BaCa_10747    25596.00000
Air_BaCa_10747    25596.00000
                       …
Tab_RoNu_10747    17684.77658
Tab_RoNu_11320    17684.77658
Tab_RoNu_11380    17684.77658
Tab_RoNu_11380    17684.77658
Tab_RoNu_11380    17684.77658
Name: Distancia, Length: 1098, dtype: float64
```

[56]:
```
# ahore que tenemos las columnas distancia y consumo completas, realizaremos el␣
 ↪IC, borrando primero la columna IC con NaNs

# eliminado
df_nuevos_viajes.drop(columns =["IC"])
```

```
# hallo el IC y creo la columna nueva
df_nuevos_viajes["IC"]= df_nuevos_viajes["consumo_kg"] /␣
 ↪df_nuevos_viajes["Distancia"]
```

[56]:

Id_vuelo	Aircompany	Clase	Destino	Distancia	IC	Origen	avion	consumo_kg	duracion
Air_BaCa_10737	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Air_BaCa_10737	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Air_BaCa_10737	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Air_BaCa_10747	AirnarAirnar	NaN	CádizCádiz	25596.00000	11.640720	BaliBali	Boeing 747Boeing 747	297955.869120	2106.0
Air_BaCa_10747	AirnarAirnar	NaN	CádizCádiz	25596.00000	11.474424	BaliBali	Boeing 747Boeing 747	293699.356704	2106.0
…	…	…	…	…	…	…	…	…	…
Tab_RoNu_10747	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Tab_RoNu_11320	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Tab_RoNu_11380	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Tab_RoNu_11380	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN
Tab_RoNu_11380	NaN	NaN	NaN	17684.77658	8.385200	NaN	NaN	148290.396029	NaN

[1098 rows x 9 columns]

4. Como ya no las vamos a necesitar o bien deshazte de las columnas Origen, Destino y Duracion o create otro dataframe que conserve todas las columnas menos esa.

[93]:
```
#df_nuevos_viajes.drop(columns =["Origen"]) #### ELIMINADA
```

[ ]:
```
#df_nuevos_viajes.drop(columns =["duracion"]) ### ELIMINADA
```

[91]:
```
#df_nuevos_viajes.drop(columns =["Destino"]) ### ELIMINADA
```

[91]:

Id_vuelo	Aircompany	Clase	Distancia	IC
Air_BaCa_10737	NaN	NaN	17684.77658	8.385200
Air_BaCa_10737	NaN	NaN	17684.77658	8.385200
Air_BaCa_10737	NaN	NaN	17684.77658	8.385200

```
Air_BaCa_10747  AirnarAirnar    NaN  25596.00000  11.640720
Air_BaCa_10747  AirnarAirnar    NaN  25596.00000  11.474424
…                         …    …          …         …
Tab_RoNu_10747           NaN    NaN  17684.77658   8.385200
Tab_RoNu_11320           NaN    NaN  17684.77658   8.385200
Tab_RoNu_11380           NaN    NaN  17684.77658   8.385200
Tab_RoNu_11380           NaN    NaN  17684.77658   8.385200
Tab_RoNu_11380           NaN    NaN  17684.77658   8.385200

                                avion     consumo_kg  duracion
Id_vuelo
Air_BaCa_10737                   NaN  148290.396029       NaN
Air_BaCa_10737                   NaN  148290.396029       NaN
Air_BaCa_10737                   NaN  148290.396029       NaN
Air_BaCa_10747  Boeing 747Boeing 747  297955.869120    2106.0
Air_BaCa_10747  Boeing 747Boeing 747  293699.356704    2106.0
…                                …            …       …
Tab_RoNu_10747                   NaN  148290.396029       NaN
Tab_RoNu_11320                   NaN  148290.396029       NaN
Tab_RoNu_11380                   NaN  148290.396029       NaN
Tab_RoNu_11380                   NaN  148290.396029       NaN
Tab_RoNu_11380                   NaN  148290.396029       NaN

[1098 rows x 7 columns]
```

5. Investiga el método `merge` y después ejecuta la siguiente línea haciendo las sustituciones necesarias

[104]:
```
df_final = df_nuevos_viajes.merge(df_ingresos, left_index = True, right_on =␣
↪"Id_vuelo", how = "left")

df_final
```

[104]:
```
       Aircompany  Clase     Destino    Distancia         IC  \
241           NaN    NaN         NaN  17684.77658   8.385200
241           NaN    NaN         NaN  17684.77658   8.385200
241           NaN    NaN         NaN  17684.77658   8.385200
337  AirnarAirnar    NaN  CádizCádiz  25596.00000  11.640720
337  AirnarAirnar    NaN  CádizCádiz  25596.00000  11.474424
..            …    …          …          …         …
229           NaN    NaN         NaN  17684.77658   8.385200
7             NaN    NaN         NaN  17684.77658   8.385200
260           NaN    NaN         NaN  17684.77658   8.385200
260           NaN    NaN         NaN  17684.77658   8.385200
260           NaN    NaN         NaN  17684.77658   8.385200

                  avion     consumo_kg  duracion        Id_vuelo   ingresos
241                 NaN  148290.396029       NaN  Air_BaCa_10737  460592.06
```

```
241                    NaN  148290.396029       NaN  Air_BaCa_10737  460592.06
241                    NaN  148290.396029       NaN  Air_BaCa_10737  460592.06
337  Boeing 747Boeing 747  297955.869120    2106.0  Air_BaCa_10747  778562.16
337  Boeing 747Boeing 747  293699.356704    2106.0  Air_BaCa_10747  778562.16
..                     …            …       …             …       …
229                    NaN  148290.396029       NaN  Tab_RoNu_10747  387692.61
7                      NaN  148290.396029       NaN  Tab_RoNu_11320  279534.74
260                    NaN  148290.396029       NaN  Tab_RoNu_11380  423549.43
260                    NaN  148290.396029       NaN  Tab_RoNu_11380  423549.43
260                    NaN  148290.396029       NaN  Tab_RoNu_11380  423549.43

[1098 rows x 10 columns]
```

6. ¿Qué columna has añadido?¿Qué le ha ocurrido al índice de filas?

Ha añadido un total de 3 columnas nuevas, siendo estas las columnas ingresos, duracion y destino.

El indice al fusionar con la funcion merge el df_nuevos_viajes con df_ingresos por la izquierda, añadiendo nuevos elementos relacionados directamente a ””Id_vuelo”, que se encontraba como indice en df_nuevos viajes antes de la fusion, al fusionarse el indice ha pasado a ser columna para relacionarse con los nuevos elemntos, dejando como undice el nuemrico por defecto

7. Reconstruye el índice haciendo uso de `set_index`, y de la columna ”Id_vuelo”.

[109]:
```
df_final.set_index("Id_vuelo", inplace=True)
```

[111]:
```
df_final
```

[111]:
```
                   Aircompany  Clase     Destino    Distancia         IC  \
Id_vuelo
Air_BaCa_10737            NaN    NaN         NaN  17684.77658   8.385200
Air_BaCa_10737            NaN    NaN         NaN  17684.77658   8.385200
Air_BaCa_10737            NaN    NaN         NaN  17684.77658   8.385200
Air_BaCa_10747  AirnarAirnar      NaN  CádizCádiz  25596.00000  11.640720
Air_BaCa_10747  AirnarAirnar      NaN  CádizCádiz  25596.00000  11.474424
…                        …    …          …           …        …
Tab_RoNu_10747            NaN    NaN         NaN  17684.77658   8.385200
Tab_RoNu_11320            NaN    NaN         NaN  17684.77658   8.385200
Tab_RoNu_11380            NaN    NaN         NaN  17684.77658   8.385200
Tab_RoNu_11380            NaN    NaN         NaN  17684.77658   8.385200
Tab_RoNu_11380            NaN    NaN         NaN  17684.77658   8.385200

                                 avion     consumo_kg  duracion   ingresos
Id_vuelo
Air_BaCa_10737                    NaN  148290.396029       NaN  460592.06
Air_BaCa_10737                    NaN  148290.396029       NaN  460592.06
Air_BaCa_10737                    NaN  148290.396029       NaN  460592.06
Air_BaCa_10747  Boeing 747Boeing 747  297955.869120    2106.0  778562.16
Air_BaCa_10747  Boeing 747Boeing 747  293699.356704    2106.0  778562.16
```

```
…                                     …           …         …          …
Tab_RoNu_10747                          NaN  148290.396029       NaN  387692.61
Tab_RoNu_11320                          NaN  148290.396029       NaN  279534.74
Tab_RoNu_11380                          NaN  148290.396029       NaN  423549.43
Tab_RoNu_11380                          NaN  148290.396029       NaN  423549.43
Tab_RoNu_11380                          NaN  148290.396029       NaN  423549.43

[1098 rows x 9 columns]
```

8. Ahora que tenemos los ingresos medios por ruta:
   8.1 Obten los ingresos por compañía totales.
   8.2 Obten los ingresos por compañía y origen.
   8.3 Obten los ingresos por compañía y destino.

[132]:
```
subconj3 = df_final[["Aircompany", "Destino","ingresos"]]



# al haber muchas comñias con NaN vamos a ver el numero
df_final["Aircompany"].value_counts(dropna = False) # total 626 NaN
# vamos halllar la moda de la columna
moda = df_final["Aircompany"].mode()[0]# la moda es TabarAirTabarAir(0)
print(moda)
#vamos a cambiar los Nan por la moda
NaN_final= df_final.Aircompany.isna()# muestra las filas donde estan los NAN
# cambiado por la moda
df_final.loc[NaN_final, "Aircompany"] = moda

df_final.Aircompany
```

```
TabarAirTabarAir
```

[132]:
```
Id_vuelo
Air_BaCa_10737    TabarAirTabarAir
Air_BaCa_10737    TabarAirTabarAir
Air_BaCa_10737    TabarAirTabarAir
Air_BaCa_10747        AirnarAirnar
Air_BaCa_10747        AirnarAirnar
                        …
Tab_RoNu_10747    TabarAirTabarAir
Tab_RoNu_11320    TabarAirTabarAir
Tab_RoNu_11380    TabarAirTabarAir
Tab_RoNu_11380    TabarAirTabarAir
Tab_RoNu_11380    TabarAirTabarAir
Name: Aircompany, Length: 1098, dtype: object
```

[138]:
```
# Obten los ingresos por compañía totales
#compañias:
compañias= df_final["Aircompany"].unique()
```

```
print(compañias)
#agrupo toda la informacion de cada compañia
dinero_company_moda = df_final.groupby("TabarAirTabarAir")
dinero_company_Airna = df_final.groupby("AirnarAirnar")
dinero_company_FlyQ = df_final.groupby("FlyQFlyQ")
dinero_company_Moldavia = df_final.groupby("MoldaviAirMoldaviAir")
dinero_company_PamPam = df_final.groupby("PamPangeaPamPange")
#sumo ingresos
dinero1 = dinero_company_moda["ingresos"].sum()
dinero2 = dinero = dinero_company_Airna["ingresos"].sum()
dinero3 = dinero = dinero_company_FlyQ["ingresos"].sum()
dinero4 = dinero_company_Moldavia["ingresos"].sum()
dinero5 = dinero_company_PamPam["ingresos"].sum()
# ingresos totales
dinero1 = dinero_company_moda.sort_values(ascending=False)
dinero2 = dinero_company_Airna.sort_values(ascending=False)
dinero3 = dinero_company_FlyQ.sort_values(ascending=False)
dinero4= dinero_company_Moldavia.sort_values(ascending=False)
dinero5 = dinero_company_PamPam.sort_values(ascending=False)
# mostrar
print(dinero1)
print()
print(dinero2)
print()
print(dinero3)
print()
print(dinero4)
print()
print(dinero5)
print()
```

```
['TabarAirTabarAir' 'AirnarAirnar' 'FlyQFlyQ' 'MoldaviAirMoldaviAir'
 'PamPangeaPamPangea']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
e:\Cursos\BC_Data_Science\Repositorio\ONLINE_DS_THEBRIDGE_V\SPRING 4\UNIT␣
 ↪2\PRACTICA 2\18_Practica_Obligatoria_Pandas_II.ipynb Celda 115 line 6

     <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/PRACTICA%202/
 ↪18_Practica_Obligatoria_Pandas_II.ipynb#Y244sZmlsZQ%3D%3D?line=3'>4</a>␣
 ↪print(compañias)
     <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/PRACTICA%202/
 ↪18_Practica_Obligatoria_Pandas_II.ipynb#Y244sZmlsZQ%3D%3D?line=4'>5</a>␣
 ↪#agrupo toda la informacion de cada compañia
```

```
----> <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/PRACTICA%202/
 ↪18_Practica_Obligatoria_Pandas_II.ipynb#Y244sZmlsZQ%3D%3D?line=5'>6</a>␣
 ↪dinero_company_moda = df_final.groupby("TabarAirTabarAir")
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/PRACTICA%202/
 ↪18_Practica_Obligatoria_Pandas_II.ipynb#Y244sZmlsZQ%3D%3D?line=6'>7</a>␣
 ↪dinero_company_Airna = df_final.groupby("AirnarAirnar")
      <a href='vscode-notebook-cell:/e%3A/Cursos/BC_Data_Science/Repositorio/
 ↪ONLINE_DS_THEBRIDGE_V/SPRING%204/UNIT%202/PRACTICA%202/
 ↪18_Practica_Obligatoria_Pandas_II.ipynb#Y244sZmlsZQ%3D%3D?line=7'>8</a>␣
 ↪dinero_company_FlyQ = df_final.groupby("FlyQFlyQ")

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\frame.
 ↪py:8869, in DataFrame.groupby(self, by, axis, level, as_index, sort,␣
 ↪group_keys, observed, dropna)
   8866 if level is None and by is None:
   8867     raise TypeError("You have to supply one of 'by' and 'level'")
-> 8869 return DataFrameGroupBy(
   8870     obj=self,
   8871     keys=by,
   8872     axis=axis,
   8873     level=level,
   8874     as_index=as_index,
   8875     sort=sort,
   8876     group_keys=group_keys,
   8877     observed=observed,
   8878     dropna=dropna,
   8879 )

File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\groupby\
 ↪py:1278, in GroupBy.__init__(self, obj, keys, axis, level, grouper,␣
 ↪exclusions, selection, as_index, sort, group_keys, observed, dropna)
   1275 self.dropna = dropna
   1277 if grouper is None:
-> 1278     grouper, exclusions, obj = get_grouper(
   1279         obj,
   1280         keys,
   1281         axis=axis,
   1282         level=level,
   1283         sort=sort,
   1284         observed=False if observed is lib.no_default else observed,
   1285         dropna=self.dropna,
   1286     )
   1288 if observed is lib.no_default:
   1289     if any(ping._passed_categorical for ping in grouper.groupings):
```

```
File c:
 ↪\Users\victo\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\groupby\
 ↪py:1009, in get_grouper(obj, key, axis, level, sort, observed, validate,␣
 ↪dropna)
   1007         in_axis, level, gpr = False, gpr, None
   1008     else:
-> 1009         raise KeyError(gpr)
   1010 elif isinstance(gpr, Grouper) and gpr.key is not None:
   1011     # Add key to exclusions
   1012     exclusions.add(gpr.key)

KeyError: 'TabarAirTabarAir'
```

9. Se dice que se va a imponer un impuesto del 0.2% de los ingresos a las dos empresas más contaminantes, ¿cuánto tendrían que pagar considerando sólo los viajes de tu dataframe?

10. Dados el rumor del apartado 9, las empresas quieren saber que vuelos les rentan más y cuales menos. Suponiendo que en "ingresos" va realmente el beneficio, obten el beneficio por km de cada viaje y luego el beneficio por km. medio para cada Compañía.

11. Las empresas quieren saber más sobre sus ingresos con vistas a saber que rutas son las menos rentables y las más susceptibles de pagar impuestos: 11.1 Obten por comapañía y ruta: ingresos totales, consumo medio, IC medio, numero de viajes (usa `agg`) 11.2 ¿Cuáles son las dos rutas menos rentables por compañía?¿Y en total?

[ ]:

[ ]:

[ ]:

[ ]: