3. Data Manipulation

July 7, 2022

1 MANIPULACIÓN DE DATOS CON PANDAS

[1]:
```
### PANDAS

# Las funciones más comunes de Pandas para explorar datos son:

# df.head()
# df.info()
# df.shape
# df.describe()
# df.values
# df.columns
# df.index
```

1.1 Orden y subconjuntos

[2]:
```
### ORDEN Y SUBCONJUNTOS

# Para cambiar el orden de las filas:
# df.sort_values("Rooms")
# df.sort_values("Rooms", ascending = False)

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

df = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/homes.csv")
df1 = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/cities.csv")

# Para mostrar solo los X más grandes o chicos:
print(df.nlargest(6, "Living"))
print(df.nsmallest(6, "Rooms"))

# También se puede ordenar por más de una columna:
# df.sort_values(["Rooms", "Living"], ascending  = [True, False])
print(df.nsmallest(6, ["Rooms", "Living"]))
```

```python
# Para seleccionar columnas específicas:
df2 = df[["Rooms", "Taxes"]]

# A lo cual se le puede introducir operadores lógicos:
df3 = df[df["Living"] >35]
df3

# Y para seleccionar con base en texto:
df4 = df1[df1["State"] == "OH"]
df4

# Y con fechas:
# df[df["date_of_birth"] < "2015-01-01"]

# Combinando condiciones
df[(df["Rooms"] < 15) & (df["Living"] > 35)]

# .isin():
# is_black_or_brown = dogs["color"].isin(["Black", "Brown"])
# dogs[is_black_or_brown]
# Alternativamente: df[df["col"].isin(["value_1", "value_2"])].
```

```
   Sell   List  Living  Rooms  Beds  Baths    Age  Acres    Taxes
28  567  625.0    64.0   11.0   4.0    4.0    4.0   0.85  12192.0
43  212  230.0    39.0   12.0   5.0    3.0  202.0   4.29   3648.0
38  265  270.0    36.0   10.0   6.0    3.0   33.0   1.20   5853.0
8   271  285.0    30.0   10.0   5.0    2.0   30.0   0.53   5702.0
47  247  252.0    29.0    9.0   4.0    2.0    4.0   1.25   4626.0
0   142  160.0    28.0   10.0   5.0    3.0   60.0   0.28   3167.0
   Sell   List  Living  Rooms  Beds  Baths   Age  Acres   Taxes
9    89   90.0    10.0    5.0   3.0    1.0  43.0   0.30  2054.0
2   129  132.0    13.0    6.0   3.0    1.0  41.0   0.33  1471.0
33  148  153.0    13.0    6.0   3.0    2.0  22.0   0.39  3950.0
45  129  135.0    10.0    6.0   3.0    1.0  15.0   1.00  2438.0
3   138  140.0    17.0    7.0   3.0    1.0  22.0   0.46  3204.0
5   135  140.0    18.0    7.0   4.0    3.0   9.0   0.57  3028.0
   Sell   List  Living  Rooms  Beds  Baths   Age  Acres   Taxes
9    89   90.0    10.0    5.0   3.0    1.0  43.0   0.30  2054.0
45  129  135.0    10.0    6.0   3.0    1.0  15.0   1.00  2438.0
2   129  132.0    13.0    6.0   3.0    1.0  41.0   0.33  1471.0
33  148  153.0    13.0    6.0   3.0    2.0  22.0   0.39  3950.0
34  152  159.0    15.0    7.0   3.0    1.0  25.0   0.59  3055.0
11   87   90.0    16.0    7.0   3.0    1.0  50.0   0.65  1445.0
```

[2]:

	Sell	List	Living	Rooms	Beds	Baths	Age	Acres	Taxes
28	567	625.0	64.0	11.0	4.0	4.0	4.0	0.85	12192.0
38	265	270.0	36.0	10.0	6.0	3.0	33.0	1.20	5853.0

```
43   212   230.0     39.0    12.0    5.0     3.0   202.0    4.29    3648.0
```

[3]:
```python
### NUEVAS COLUMNAS

df["Rooms/10"] = df["Rooms"]/10

df["NuevaColumna"] = df["Beds"]/df["Rooms"]*100

print(df.head())
```

```
  Sell   List  Living  Rooms  Beds  Baths   Age  Acres   Taxes  Rooms/10  \
0  142  160.0    28.0   10.0   5.0    3.0  60.0   0.28  3167.0       1.0
1  175  180.0    18.0    8.0   4.0    1.0  12.0   0.43  4033.0       0.8
2  129  132.0    13.0    6.0   3.0    1.0  41.0   0.33  1471.0       0.6
3  138  140.0    17.0    7.0   3.0    1.0  22.0   0.46  3204.0       0.7
4  232  240.0    25.0    8.0   4.0    3.0   5.0   2.05  3613.0       0.8

   NuevaColumna
0     50.000000
1     50.000000
2     50.000000
3     42.857143
4     50.000000
```

1.2 Estadísticas de resumen

[4]:
```python
### ESTADÍSTICAS DE RESUMEN

df["Rooms"].mean()

# .median(), .mode(), .min(), .max(), .var(), .std(), .sum(), .quantile()

# df["date"].min()
```

[4]: 8.06

[5]:
```python
# Se pueden crear estadísticas personalizadas:

def pct30(column):
   return column.quantile(0.3)

# df["Rooms"].agg(pct30)

# O en varias columnas:

df[["Rooms", "Living"]].agg(pct30)
```

```
[5]: Rooms      7.7
     Living    17.0
     dtype: float64
```

```python
[6]: # Y además, usar varias estadísticas personalizadas:

     def pct40(column):
         return column.quantile(0.4)

     df["Rooms"].agg([pct30, pct40])
```

```
[6]: pct30    7.7
     pct40    8.0
     Name: Rooms, dtype: float64
```

```python
[7]: # Para una suma acumulativa:

     df["Rooms"].cumsum()

     # Otras estadísticas acumulativas son: .cummax(), .cummin(), .cumprod()
```

```
[7]: 0      10.0
     1      18.0
     2      24.0
     3      31.0
     4      39.0
     5      46.0
     6      54.0
     7      62.0
     8      72.0
     9      77.0
     10     85.0
     11     92.0
     12    100.0
     13    108.0
     14    116.0
     15    124.0
     16    133.0
     17    140.0
     18    147.0
     19    155.0
     20    164.0
     21    172.0
     22    181.0
     23    190.0
     24    199.0
     25    207.0
```

```
26    215.0
27    222.0
28    233.0
29    241.0
30    248.0
31    257.0
32    265.0
33    271.0
34    278.0
35    285.0
36    295.0
37    303.0
38    313.0
39    321.0
40    330.0
41    338.0
42    346.0
43    358.0
44    366.0
45    372.0
46    379.0
47    388.0
48    396.0
49    403.0
50      NaN
Name: Rooms, dtype: float64
```

[8]:

```
### EJEMPLO VENTAS

sales = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/sales_subset.csv")

print(sales.head())
print(sales.info())

print(sales["weekly_sales"].mean())
print(sales["weekly_sales"].median())

print(sales["date"].max())
print(sales["date"].min())

def iqr(column):
    return column.quantile(0.75)-column.quantile(0.25)

print(sales[["temperature_c", "fuel_price_usd_per_l", "unemployment"]].
 ↪agg([iqr, np.median]))
```

```python
sales_1_1 = sales.sort_values("date")
sales_1_1["cum_weekly_sales"] = sales_1_1["weekly_sales"].cumsum()
sales_1_1["cum_max_sales"] = sales_1_1["weekly_sales"].cummax()
print(sales_1_1.head())
```

```
   Unnamed: 0  store type  department        date  weekly_sales  is_holiday  \
0           0      1    A           1  2010-02-05      24924.50       False
1           1      1    A           1  2010-03-05      21827.90       False
2           2      1    A           1  2010-04-02      57258.43       False
3           3      1    A           1  2010-05-07      17413.94       False
4           4      1    A           1  2010-06-04      17558.09       False

   temperature_c  fuel_price_usd_per_l  unemployment
0       5.727778              0.679451         8.106
1       8.055556              0.693452         8.106
2      16.816667              0.718284         7.808
3      22.527778              0.748928         7.808
4      27.050000              0.714586         7.808
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10774 entries, 0 to 10773
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            10774 non-null  int64
 1   store                 10774 non-null  int64
 2   type                  10774 non-null  object
 3   department            10774 non-null  int64
 4   date                  10774 non-null  object
 5   weekly_sales          10774 non-null  float64
 6   is_holiday            10774 non-null  bool
 7   temperature_c         10774 non-null  float64
 8   fuel_price_usd_per_l  10774 non-null  float64
 9   unemployment          10774 non-null  float64
dtypes: bool(1), float64(4), int64(3), object(2)
memory usage: 768.2+ KB
None
23843.950148505668
12049.064999999999
2012-10-26
2010-02-05
        temperature_c  fuel_price_usd_per_l  unemployment
iqr         16.583333              0.073176         0.565
median      16.966667              0.743381         8.099
      Unnamed: 0  store type  department        date  weekly_sales  \
0              0      1    A           1  2010-02-05      24924.50
6437        6437     19    A          13  2010-02-05      38597.52
1249        1249      2    A          31  2010-02-05       3840.21
```

```
6449        6449     19     A           14   2010-02-05      17590.59
6461        6461     19     A           16   2010-02-05       4929.87

      is_holiday  temperature_c  fuel_price_usd_per_l  unemployment  \
0          False       5.727778              0.679451         8.106
6437       False      -6.133333              0.780365         8.350
1249       False       4.550000              0.679451         8.324
6449       False      -6.133333              0.780365         8.350
6461       False      -6.133333              0.780365         8.350

      cum_weekly_sales  cum_max_sales
0             24924.50       24924.50
6437          63522.02       38597.52
1249          67362.23       38597.52
6449          84952.82       38597.52
6461          89882.69       38597.52
```

1.3 Conteos

[9]:
```
### CONTEOS

# Se pueden eliminar duplicados:

homelessness = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/
 ↪homelessness.csv")

print(homelessness.drop_duplicates(subset = "region"))

# homelessness_1 = homelessness.drop_duplicates(subset = ["region", "state"])
# print(homelessness_1)

# Para contar por categoría:
print(homelessness["region"].value_counts())
print(homelessness["region"].value_counts(normalize = True))
```

```
    Unnamed: 0              region        state  individuals  family_members  \
0            0  East South Central      Alabama       2570.0           864.0
1            1             Pacific       Alaska       1434.0           582.0
2            2            Mountain      Arizona       7259.0          2606.0
3            3  West South Central     Arkansas       2280.0           432.0
6            6         New England  Connecticut       2280.0          1696.0
7            7      South Atlantic     Delaware        708.0           374.0
13          13  East North Central     Illinois       6752.0          3891.0
15          15  West North Central         Iowa       1711.0          1038.0
30          30        Mid-Atlantic   New Jersey       6048.0          3350.0

    state_pop
0     4887681
```

```
1      735139
2     7158024
3     3009733
6     3571520
7      965479
13   12723071
15    3148618
30    8886025
South Atlantic        9
Mountain              8
West North Central    7
New England           6
Pacific               5
East North Central    5
East South Central    4
West South Central    4
Mid-Atlantic          3
Name: region, dtype: int64
South Atlantic        0.176471
Mountain              0.156863
West North Central    0.137255
New England           0.117647
Pacific               0.098039
East North Central    0.098039
East South Central    0.078431
West South Central    0.078431
Mid-Atlantic          0.058824
Name: region, dtype: float64
```

[10]:

```
### EJEMPLO VENTAS

store_types = sales.drop_duplicates(subset = ["store", "type"])
print(store_types.head())

store_depts = sales.drop_duplicates(subset = ["store", "department"])
print(store_depts.head())

holiday_dates = sales[sales["is_holiday"] == True].drop_duplicates(subset =␣
 ↪"date")
print(holiday_dates["date"])

store_counts = store_types["type"].value_counts()
print(store_counts)

store_props = store_types["type"].value_counts(normalize = True)
print(store_props)
```

```python
dept_counts_sorted = store_depts["department"].value_counts(ascending = False)
print(dept_counts_sorted)

dept_props_sorted = store_depts["department"].value_counts(ascending=False,␣
 ↪normalize=True)
print(dept_props_sorted)
```

```
      Unnamed: 0  store type  department        date  weekly_sales  \
0              0      1    A           1  2010-02-05      24924.50
901          901      2    A           1  2010-02-05      35034.06
1798        1798      4    A           1  2010-02-05      38724.42
2699        2699      6    A           1  2010-02-05      25619.00
3593        3593     10    B           1  2010-02-05      40212.84

      is_holiday  temperature_c  fuel_price_usd_per_l  unemployment
0          False       5.727778              0.679451         8.106
901        False       4.550000              0.679451         8.324
1798       False       6.533333              0.686319         8.623
2699       False       4.683333              0.679451         7.259
3593       False      12.411111              0.782478         9.765
    Unnamed: 0  store type  department        date  weekly_sales  is_holiday  \
0            0      1    A           1  2010-02-05      24924.50       False
12          12      1    A           2  2010-02-05      50605.27       False
24          24      1    A           3  2010-02-05      13740.12       False
36          36      1    A           4  2010-02-05      39954.04       False
48          48      1    A           5  2010-02-05      32229.38       False

    temperature_c  fuel_price_usd_per_l  unemployment
0        5.727778              0.679451         8.106
12       5.727778              0.679451         8.106
24       5.727778              0.679451         8.106
36       5.727778              0.679451         8.106
48       5.727778              0.679451         8.106
498     2010-09-10
691     2011-11-25
2315    2010-02-12
6735    2012-09-07
6810    2010-12-31
6815    2012-02-10
6820    2011-09-09
Name: date, dtype: object
A    11
B     1
Name: type, dtype: int64
A    0.916667
B    0.083333
Name: type, dtype: float64
1     12
```

```
55     12
72     12
71     12
67     12
       ..
37     10
48      8
50      6
39      4
43      2
Name: department, Length: 80, dtype: int64
1      0.012917
55     0.012917
72     0.012917
71     0.012917
67     0.012917
         …
37     0.010764
48     0.008611
50     0.006459
39     0.004306
43     0.002153
Name: department, Length: 80, dtype: float64
```

1.4 Estadísticas por grupo

[11]:
```
### ESTADÍSTICAS POR GRUPO

print(homelessness[homelessness["region"] == "Pacific"]["individuals"].mean())
print(homelessness[homelessness["state"] == "California"]["individuals"].mean())

# Pero es má fácil usar .groupby:

print(homelessness.groupby("region")["individuals"].mean())

print(homelessness.groupby("region")["state_pop"].agg([min, max, sum]))

# Y agrupando en dos grupos:

print(sales.groupby(["type", "department"])["weekly_sales"].mean())

# Dos grupos y dos columnas:

print(sales.groupby(["type", "department"])[["weekly_sales", "temperature_c"]].
 ↪mean())
```

```
28427.2
109008.0
```

```
region
East North Central     5081.200000
East South Central     3117.000000
Mid-Atlantic          18012.666667
Mountain               3561.375000
New England            2150.500000
Pacific               28427.200000
South Atlantic         5806.666667
West North Central     1995.857143
West South Central     6710.500000
Name: individuals, dtype: float64
                        min       max       sum
region
East North Central  5807406  12723071  46886387
East South Central  2981020   6771631  19101485
Mid-Atlantic        8886025  19530351  41217298
Mountain             577601   7158024  24511745
New England          624358   6882635  14829322
Pacific              735139  39461588  53323075
South Atlantic       701547  21244317  65229624
West North Central   758080   6121623  21350241
West South Central  3009733  28628666  40238324
type  department
A     1             30961.725379
      2             67600.158788
      3             17160.002955
      4             44285.399091
      5             34821.011364
                        …
B     94              161.445833
      95            77082.102500
      96             9528.538333
      97             5828.873333
      98              217.428333
Name: weekly_sales, Length: 157, dtype: float64
                 weekly_sales  temperature_c
type department
A    1           30961.725379      15.258754
     2           67600.158788      15.258754
     3           17160.002955      15.258754
     4           44285.399091      15.258754
     5           34821.011364      15.258754
…                         …            …
B    94            161.445833      21.379167
     95          77082.102500      21.216204
     96           9528.538333      21.216204
     97           5828.873333      21.216204
     98            217.428333      21.163426
```

```
[157 rows x 2 columns]
```

[12]:
```python
### EJEMPLO VENTAS

sales_all = sales["weekly_sales"].sum()

sales_A = sales[sales["type"] == "A"]["weekly_sales"].sum()

sales_B = sales[sales["type"] == "B"]["weekly_sales"].sum()

sales_C = sales[sales["type"] == "C"]["weekly_sales"].sum()

sales_propn_by_type = [sales_A, sales_B, sales_C] / sales_all
print(sales_propn_by_type)

sales_by_type = sales.groupby("type")["weekly_sales"].sum()

sales_propn_by_type = sales_by_type/sum(sales_by_type)
print(sales_propn_by_type)

###

sales_by_type_is_holiday = sales.groupby(["type",␣
↪"is_holiday"])["weekly_sales"].sum()
print(sales_by_type_is_holiday)

###

import numpy as np

sales_stats = sales.groupby("type")["weekly_sales"].agg([min, max, np.mean, np.
↪median])
print(sales_stats)

unemp_fuel_stats = sales.groupby("type")[["unemployment",␣
↪"fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
print(unemp_fuel_stats)
```

```
[0.9097747 0.0902253 0.       ]
type
A    0.909775
B    0.090225
Name: weekly_sales, dtype: float64
type  is_holiday
A     False         2.336927e+08
      True          2.360181e+04
B     False         2.317678e+07
```

```
      True          1.621410e+03
Name: weekly_sales, dtype: float64
         min        max          mean    median
type
A    -1098.0  293966.05  23674.667242  11943.92
B     -798.0  232558.51  25696.678370  13336.08
     unemployment                        fuel_price_usd_per_l            \
              min    max      mean median                 min       max
type
A           3.879  8.992  7.972611  8.067            0.664129  1.107410
B           7.170  9.765  9.279323  9.199            0.760023  1.107674


          mean    median
type
A     0.744619  0.735455
B     0.805858  0.803348
```

1.5 Tablas dinámicas

[13]:
```python
### TABLAS DINÁMICAS

# Puede ser equivalente a las estadísticas de resumen de groupby:

print(sales.pivot_table(values = "weekly_sales", index = "type"))

import numpy as np

print(sales.pivot_table(values = "temperature_c", index = "type", aggfunc = np.
 ↪median))

# Para múltiples estadísticas:

print(homelessness.pivot_table(values = "individuals", index = "region",␣
 ↪aggfunc = [np.mean, np.median]))

# Y para agrupar dos variables:

print(sales.pivot_table(values = "weekly_sales", index = "type", columns =␣
 ↪"store"))

# Para sustituir los NAs y agregar las estadísticas de totales por grupo:

print(sales.pivot_table(values = "weekly_sales", index = "type", columns =␣
 ↪"store", fill_value = 0, margins = True))
```

```
      weekly_sales
type
```

```
A     23674.667242
B     25696.678370
      temperature_c
type
A         16.455556
B         21.688889
                        mean      median
                  individuals individuals
region
East North Central   5081.200000      5209.0
East South Central   3117.000000      2652.5
Mid-Atlantic        18012.666667      8163.0
Mountain             3561.375000      1926.5
New England          2150.500000      1142.5
Pacific             28427.200000     11139.0
South Atlantic       5806.666667      3928.0
West North Central   1995.857143      1711.0
West South Central   6710.500000      2681.5
store           1            2            4            6           10  \
type
A      20896.941787 26517.435162 26126.986071 21561.186477          NaN
B               NaN          NaN          NaN          NaN  25696.67837

store           13           14           19           20           27  \
type
A      25664.149474 30384.003017 19930.838157 28382.766385 24207.474711
B               NaN          NaN          NaN          NaN          NaN

store           31           39
type
A      18178.932225 18414.938423
B               NaN          NaN
store            1            2            4            6          10  \
type
A      20896.941787 26517.435162 26126.986071 21561.186477     0.00000
B          0.000000     0.000000     0.000000     0.000000 25696.67837
All    20896.941787 26517.435162 26126.986071 21561.186477 25696.67837

store           13           14           19           20           27  \
type
A      25664.149474 30384.003017 19930.838157 28382.766385 24207.474711
B          0.000000     0.000000     0.000000     0.000000     0.000000
All    25664.149474 30384.003017 19930.838157 28382.766385 24207.474711

store           31           39          All
type
A      18178.932225 18414.938423 23674.667242
B          0.000000     0.000000 25696.678370
```

```
All     18178.932225  18414.938423  23843.950149
```

1.6 Índices explícitos

[14]:
```python
### ÍNDICES EXPLÍCITOS

# Se puede configurar una columna como índice:

homelessness_ind = homelessness.set_index("region")

# Lo cual cambia ligeramente el dataset, haciendo que la nueva columna índice
 ↪se alinee a la izquierda
# Lo cual puede revertirse mediante:

homelessness_ind.reset_index()
```

[14]:
```
                region  Unnamed: 0                 state  individuals  \
0   East South Central           0               Alabama       2570.0
1              Pacific           1                Alaska       1434.0
2             Mountain           2               Arizona       7259.0
3   West South Central           3              Arkansas       2280.0
4              Pacific           4            California     109008.0
5             Mountain           5              Colorado       7607.0
6          New England           6           Connecticut       2280.0
7       South Atlantic           7              Delaware        708.0
8       South Atlantic           8  District of Columbia       3770.0
9       South Atlantic           9               Florida      21443.0
10      South Atlantic          10               Georgia       6943.0
11             Pacific          11                Hawaii       4131.0
12            Mountain          12                 Idaho       1297.0
13  East North Central          13              Illinois       6752.0
14  East North Central          14               Indiana       3776.0
15  West North Central          15                  Iowa       1711.0
16  West North Central          16                Kansas       1443.0
17  East South Central          17              Kentucky       2735.0
18  West South Central          18             Louisiana       2540.0
19         New England          19                 Maine       1450.0
20      South Atlantic          20              Maryland       4914.0
21         New England          21         Massachusetts       6811.0
22  East North Central          22              Michigan       5209.0
23  West North Central          23             Minnesota       3993.0
24  East South Central          24           Mississippi       1024.0
25  West North Central          25              Missouri       3776.0
26            Mountain          26               Montana        983.0
27  West North Central          27              Nebraska       1745.0
28            Mountain          28                Nevada       7058.0
29         New England          29         New Hampshire        835.0
```

30	Mid-Atlantic	30	New Jersey	6048.0
31	Mountain	31	New Mexico	1949.0
32	Mid-Atlantic	32	New York	39827.0
33	South Atlantic	33	North Carolina	6451.0
34	West North Central	34	North Dakota	467.0
35	East North Central	35	Ohio	6929.0
36	West South Central	36	Oklahoma	2823.0
37	Pacific	37	Oregon	11139.0
38	Mid-Atlantic	38	Pennsylvania	8163.0
39	New England	39	Rhode Island	747.0
40	South Atlantic	40	South Carolina	3082.0
41	West North Central	41	South Dakota	836.0
42	East South Central	42	Tennessee	6139.0
43	West South Central	43	Texas	19199.0
44	Mountain	44	Utah	1904.0
45	New England	45	Vermont	780.0
46	South Atlantic	46	Virginia	3928.0
47	Pacific	47	Washington	16424.0
48	South Atlantic	48	West Virginia	1021.0
49	East North Central	49	Wisconsin	2740.0
50	Mountain	50	Wyoming	434.0

	family_members	state_pop
0	864.0	4887681
1	582.0	735139
2	2606.0	7158024
3	432.0	3009733
4	20964.0	39461588
5	3250.0	5691287
6	1696.0	3571520
7	374.0	965479
8	3134.0	701547
9	9587.0	21244317
10	2556.0	10511131
11	2399.0	1420593
12	715.0	1750536
13	3891.0	12723071
14	1482.0	6695497
15	1038.0	3148618
16	773.0	2911359
17	953.0	4461153
18	519.0	4659690
19	1066.0	1339057
20	2230.0	6035802
21	13257.0	6882635
22	3142.0	9984072
23	3250.0	5606249

```
24          328.0    2981020
25         2107.0    6121623
26          422.0    1060665
27          676.0    1925614
28          486.0    3027341
29          615.0    1353465
30         3350.0    8886025
31          602.0    2092741
32        52070.0   19530351
33         2817.0   10381615
34           75.0     758080
35         3320.0   11676341
36         1048.0    3940235
37         3337.0    4181886
38         5349.0   12800922
39          354.0    1058287
40          851.0    5084156
41          323.0     878698
42         1744.0    6771631
43         6111.0   28628666
44          972.0    3153550
45          511.0     624358
46         2047.0    8501286
47         5880.0    7523869
48          222.0    1804291
49         2167.0    5807406
50          205.0     577601
```

[15]:

```python
# .doc filtra valores con base en un índice:

print(homelessness_ind.loc[["Pacific", "Mountain"]])

sales_ind = sales.set_index(["type", "department"])

print(sales_ind.loc["A", 1])
```

```
          Unnamed: 0       state  individuals  family_members  state_pop
region
Pacific            1      Alaska       1434.0           582.0     735139
Pacific            4  California     109008.0         20964.0   39461588
Pacific           11      Hawaii       4131.0          2399.0    1420593
Pacific           37      Oregon      11139.0          3337.0    4181886
Pacific           47  Washington      16424.0          5880.0    7523869
Mountain           2     Arizona       7259.0          2606.0    7158024
Mountain           5    Colorado       7607.0          3250.0    5691287
Mountain          12       Idaho       1297.0           715.0    1750536
Mountain          26     Montana        983.0           422.0    1060665
Mountain          28      Nevada       7058.0           486.0    3027341
```

```
Mountain          31  New Mexico       1949.0           602.0    2092741
Mountain          44        Utah       1904.0           972.0    3153550
Mountain          50     Wyoming        434.0           205.0     577601
                 Unnamed: 0  store        date  weekly_sales  is_holiday  \
type department
A    1                    0      1  2010-02-05      24924.50       False
     1                    1      1  2010-03-05      21827.90       False
     1                    2      1  2010-04-02      57258.43       False
     1                    3      1  2010-05-07      17413.94       False
     1                    4      1  2010-06-04      17558.09       False
…                       …    …          …           …         …
     1                 9906     39  2010-09-03      15019.76       False
     1                 9907     39  2010-10-01      18819.37       False
     1                 9908     39  2010-11-05      31729.41       False
     1                 9909     39  2010-12-03      24716.60       False
     1                 9910     39  2011-01-07      11141.04       False

                 temperature_c  fuel_price_usd_per_l  unemployment
type department
A    1                5.727778              0.679451         8.106
     1                8.055556              0.693452         8.106
     1               16.816667              0.718284         7.808
     1               22.527778              0.748928         7.808
     1               27.050000              0.714586         7.808
…                          …                   …            …
     1               27.850000              0.680772         8.360
     1               22.633333              0.687640         8.476
     1               16.455556              0.710359         8.476
     1               11.972222              0.715378         8.476
     1               11.522222              0.786176         8.395

[132 rows x 8 columns]

<ipython-input-15-f59278c82d08>:7: PerformanceWarning: indexing past lexsort
depth may impact performance.
  print(sales_ind.loc["A", 1])
```

[16]:

```python
### EJEMPLO TEMPERATURAS

temperatures = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/
 ↪temperatures.csv")

temperatures_ind = temperatures.set_index("city")

print(temperatures_ind.head())

print(temperatures_ind.reset_index())
```

```python
print(temperatures_ind.reset_index(drop = True))

###

cities = ["Moscow", "Saint Petersburg"]

print(temperatures[temperatures["city"].isin(cities)])

# Alternativamente:

print(temperatures_ind.loc[cities])

###

temperatures_ind = temperatures.set_index(["country", "city"])

rows_to_keep = [("Brazil", "Rio De Janeiro"), ("Pakistan", "Lahore")]

print(temperatures_ind.loc[rows_to_keep])

###

print(temperatures_ind.sort_index())

print(temperatures_ind.sort_index(level="city"))

print(temperatures_ind.sort_index(level=["country", "city"], ascending = [True,␣
↪False]))
```

```
         Unnamed: 0        date        country  avg_temp_c
city
Abidjan           0  2000-01-01  Côte D'Ivoire      27.293
Abidjan           1  2000-02-01  Côte D'Ivoire      27.685
Abidjan           2  2000-03-01  Côte D'Ivoire      29.061
Abidjan           3  2000-04-01  Côte D'Ivoire      28.162
Abidjan           4  2000-05-01  Côte D'Ivoire      27.547
          city  Unnamed: 0        date        country  avg_temp_c
0      Abidjan           0  2000-01-01  Côte D'Ivoire      27.293
1      Abidjan           1  2000-02-01  Côte D'Ivoire      27.685
2      Abidjan           2  2000-03-01  Côte D'Ivoire      29.061
3      Abidjan           3  2000-04-01  Côte D'Ivoire      28.162
4      Abidjan           4  2000-05-01  Côte D'Ivoire      27.547
…         …          …         …              …          …
16495     Xian       16495  2013-05-01          China      18.979
16496     Xian       16496  2013-06-01          China      23.522
16497     Xian       16497  2013-07-01          China      25.251
16498     Xian       16498  2013-08-01          China      24.528
16499     Xian       16499  2013-09-01          China         NaN
```

```
[16500 rows x 5 columns]
       Unnamed: 0        date        country  avg_temp_c
0               0  2000-01-01  Côte D'Ivoire      27.293
1               1  2000-02-01  Côte D'Ivoire      27.685
2               2  2000-03-01  Côte D'Ivoire      29.061
3               3  2000-04-01  Côte D'Ivoire      28.162
4               4  2000-05-01  Côte D'Ivoire      27.547
…             …         …             …          …
16495       16495  2013-05-01          China      18.979
16496       16496  2013-06-01          China      23.522
16497       16497  2013-07-01          China      25.251
16498       16498  2013-08-01          China      24.528
16499       16499  2013-09-01          China         NaN

[16500 rows x 4 columns]
       Unnamed: 0        date              city country  avg_temp_c
10725       10725  2000-01-01            Moscow  Russia      -7.313
10726       10726  2000-02-01            Moscow  Russia      -3.551
10727       10727  2000-03-01            Moscow  Russia      -1.661
10728       10728  2000-04-01            Moscow  Russia      10.096
10729       10729  2000-05-01            Moscow  Russia      10.357
…             …         …               …     …          …
13360       13360  2013-05-01  Saint Petersburg  Russia      12.355
13361       13361  2013-06-01  Saint Petersburg  Russia      17.185
13362       13362  2013-07-01  Saint Petersburg  Russia      17.234
13363       13363  2013-08-01  Saint Petersburg  Russia      17.153
13364       13364  2013-09-01  Saint Petersburg  Russia         NaN

[330 rows x 5 columns]
                  Unnamed: 0        date country  avg_temp_c
city
Moscow                 10725  2000-01-01  Russia      -7.313
Moscow                 10726  2000-02-01  Russia      -3.551
Moscow                 10727  2000-03-01  Russia      -1.661
Moscow                 10728  2000-04-01  Russia      10.096
Moscow                 10729  2000-05-01  Russia      10.357
…                          …         …     …          …
Saint Petersburg       13360  2013-05-01  Russia      12.355
Saint Petersburg       13361  2013-06-01  Russia      17.185
Saint Petersburg       13362  2013-07-01  Russia      17.234
Saint Petersburg       13363  2013-08-01  Russia      17.153
Saint Petersburg       13364  2013-09-01  Russia         NaN

[330 rows x 4 columns]
                         Unnamed: 0        date  avg_temp_c
country  city
Brazil   Rio De Janeiro       12540  2000-01-01      25.974
```

```
         Rio De Janeiro       12541  2000-02-01      26.699
         Rio De Janeiro       12542  2000-03-01      26.270
         Rio De Janeiro       12543  2000-04-01      25.750
         Rio De Janeiro       12544  2000-05-01      24.356
…                               …        …         …
Pakistan Lahore                8575  2013-05-01      33.457
         Lahore                8576  2013-06-01      34.456
         Lahore                8577  2013-07-01      33.279
         Lahore                8578  2013-08-01      31.511
         Lahore                8579  2013-09-01         NaN

[330 rows x 3 columns]
                    Unnamed: 0        date  avg_temp_c
country     city
Afghanistan Kabul         7260  2000-01-01       3.326
            Kabul         7261  2000-02-01       3.454
            Kabul         7262  2000-03-01       9.612
            Kabul         7263  2000-04-01      17.925
            Kabul         7264  2000-05-01      24.658
…                           …        …         …
Zimbabwe    Harare        5605  2013-05-01      18.298
            Harare        5606  2013-06-01      17.020
            Harare        5607  2013-07-01      16.299
            Harare        5608  2013-08-01      19.232
            Harare        5609  2013-09-01         NaN

[16500 rows x 3 columns]
                       Unnamed: 0        date  avg_temp_c
country       city
Côte D'Ivoire Abidjan           0  2000-01-01      27.293
              Abidjan           1  2000-02-01      27.685
              Abidjan           2  2000-03-01      29.061
              Abidjan           3  2000-04-01      28.162
              Abidjan           4  2000-05-01      27.547
…                              …        …         …
China         Xian          16495  2013-05-01      18.979
              Xian          16496  2013-06-01      23.522
              Xian          16497  2013-07-01      25.251
              Xian          16498  2013-08-01      24.528
              Xian          16499  2013-09-01         NaN

[16500 rows x 3 columns]
                    Unnamed: 0        date  avg_temp_c
country     city
Afghanistan Kabul         7260  2000-01-01       3.326
            Kabul         7261  2000-02-01       3.454
            Kabul         7262  2000-03-01       9.612
            Kabul         7263  2000-04-01      17.925
```

```
            Kabul         7264  2000-05-01      24.658
…                        …         …         …
Zimbabwe    Harare        5605  2013-05-01      18.298
            Harare        5606  2013-06-01      17.020
            Harare        5607  2013-07-01      16.299
            Harare        5608  2013-08-01      19.232
            Harare        5609  2013-09-01         NaN

[16500 rows x 3 columns]
```

1.7 Slicing

[17]:
```python
### SLICING

# Recuérdese que las posiciones en Python empiezan en 0

print(temperatures[0:5])

homelessness_srt = homelessness.set_index(["region", "state"]).sort_index()

print(homelessness_srt.loc["Mountain":"Pacific"])

# Esto solo funciona con los niveles exteriores del índice (región), no con los␣
↪interiores (state)

# Para ordenar fechas:

sales = sales.set_index("date").sort_index()
print(sales.head())
```

```
   Unnamed: 0        date     city        country  avg_temp_c
0           0  2000-01-01  Abidjan  Côte D'Ivoire      27.293
1           1  2000-02-01  Abidjan  Côte D'Ivoire      27.685
2           2  2000-03-01  Abidjan  Côte D'Ivoire      29.061
3           3  2000-04-01  Abidjan  Côte D'Ivoire      28.162
4           4  2000-05-01  Abidjan  Côte D'Ivoire      27.547
                           Unnamed: 0  individuals  family_members  state_pop
region      state
Mountain    Arizona                 2       7259.0          2606.0    7158024
            Colorado                5       7607.0          3250.0    5691287
            Idaho                  12       1297.0           715.0    1750536
            Montana                26        983.0           422.0    1060665
            Nevada                 28       7058.0           486.0    3027341
            New Mexico             31       1949.0           602.0    2092741
            Utah                   44       1904.0           972.0    3153550
            Wyoming                50        434.0           205.0     577601
New England Connecticut             6       2280.0          1696.0    3571520
            Maine                  19       1450.0          1066.0    1339057
```

```
                Massachusetts           21        6811.0           13257.0    6882635
                New Hampshire           29         835.0             615.0    1353465
                Rhode Island            39         747.0             354.0    1058287
                Vermont                 45         780.0             511.0     624358
Pacific         Alaska                   1        1434.0             582.0     735139
                California               4      109008.0           20964.0   39461588
                Hawaii                  11        4131.0            2399.0    1420593
                Oregon                  37       11139.0            3337.0    4181886
                Washington              47       16424.0            5880.0    7523869
            Unnamed: 0  store type  department  weekly_sales  is_holiday  \
date
2010-02-05           0      1    A           1      24924.50       False
2010-02-05        6437     19    A          13      38597.52       False
2010-02-05        1249      2    A          31       3840.21       False
2010-02-05        6449     19    A          14      17590.59       False
2010-02-05        6461     19    A          16       4929.87       False

            temperature_c  fuel_price_usd_per_l  unemployment
date
2010-02-05       5.727778              0.679451         8.106
2010-02-05      -6.133333              0.780365         8.350
2010-02-05       4.550000              0.679451         8.324
2010-02-05      -6.133333              0.780365         8.350
2010-02-05      -6.133333              0.780365         8.350
```

[18]:

```python
## Y para cortar por fechas:

print(sales.loc["2010-02-05":"2010-02-10"])

# Algo útil es que puede cortarse por fechas parciales:

print(sales.loc["2010":"2011"])

# Y cortar por número de renglón o columna:

print(sales.iloc[2:5, 1:4])
```

```
            Unnamed: 0  store type  department  weekly_sales  is_holiday  \
date
2010-02-05           0      1    A           1      24924.50       False
2010-02-05        6437     19    A          13      38597.52       False
2010-02-05        1249      2    A          31       3840.21       False
2010-02-05        6449     19    A          14      17590.59       False
2010-02-05        6461     19    A          16       4929.87       False
…                  …    …   …         …           …          …
2010-02-05        9555     31    A          52        842.92       False
2010-02-05        9177     31    A          16       2561.38       False
2010-02-05         180      1    A          17      13223.76       False
```

```
2010-02-05       10126     39    A           21       6843.57       False
2010-02-05       10378     39    A           45          3.77       False

            temperature_c  fuel_price_usd_per_l  unemployment
date
2010-02-05       5.727778              0.679451         8.106
2010-02-05      -6.133333              0.780365         8.350
2010-02-05       4.550000              0.679451         8.324
2010-02-05      -6.133333              0.780365         8.350
2010-02-05      -6.133333              0.780365         8.350
…                   …                   …           …
2010-02-05       3.916667              0.679451         8.324
2010-02-05       3.916667              0.679451         8.324
2010-02-05       5.727778              0.679451         8.106
2010-02-05       6.833333              0.679451         8.554
2010-02-05       6.833333              0.679451         8.554

[869 rows x 9 columns]
            Unnamed: 0  store type  department  weekly_sales  is_holiday  \
date
2010-02-05           0      1    A           1      24924.50       False
2010-02-05        6437     19    A          13      38597.52       False
2010-02-05        1249      2    A          31       3840.21       False
2010-02-05        6449     19    A          14      17590.59       False
2010-02-05        6461     19    A          16       4929.87       False
…                …    …  …         …            …          …
2010-12-17         501      1    A          45         22.94       False
2010-12-17        6792     19    A          45         23.00       False
2010-12-24         521      1    A          47         89.00       False
2010-12-24        1788      2    A          99       -147.00       False
2010-12-31        6810     19    A          47       -449.00        True

            temperature_c  fuel_price_usd_per_l  unemployment
date
2010-02-05       5.727778              0.679451         8.106
2010-02-05      -6.133333              0.780365         8.350
2010-02-05       4.550000              0.679451         8.324
2010-02-05      -6.133333              0.780365         8.350
2010-02-05      -6.133333              0.780365         8.350
…                   …                   …           …
2010-12-17       9.911111              0.757910         7.838
2010-12-17      -2.872222              0.872032         8.067
2010-12-24      11.294444              0.762401         7.838
2010-12-24       9.983333              0.762401         8.163
2010-12-31      -1.861111              0.881278         8.067

[9613 rows x 9 columns]
            store type  department
```

```
date
2010-02-05      2     A           31
2010-02-05     19     A           14
2010-02-05     19     A           16
```

[19]:

```
### EJEMPLO TEMPERATURAS

temperatures_srt = temperatures_ind.sort_index()

print(temperatures_srt.loc["Pakistan":"Russia"])

print(temperatures_srt.loc["Lahore":"Moscow"]) # no tiene sentido

print(temperatures_srt.loc[("Pakistan", "Lahore"):("Russia", "Moscow")])

###

print(temperatures_srt.loc[("India", "Hyderabad"):("Iraq", "Baghdad")])

print(temperatures_srt.loc[:, "date":"avg_temp_c"])

# Subconjunto de filas y columnas:

print(temperatures_srt.loc[("India", "Hyderabad"):("Iraq", "Baghdad"), "date":
 ↪"avg_temp_c"])
```

```
                           Unnamed: 0        date  avg_temp_c
country  city
Pakistan Faisalabad              4785  2000-01-01      12.792
         Faisalabad              4786  2000-02-01      14.339
         Faisalabad              4787  2000-03-01      20.309
         Faisalabad              4788  2000-04-01      29.072
         Faisalabad              4789  2000-05-01      34.845
…                                   …           …           …
Russia   Saint Petersburg       13360  2013-05-01      12.355
         Saint Petersburg       13361  2013-06-01      17.185
         Saint Petersburg       13362  2013-07-01      17.234
         Saint Petersburg       13363  2013-08-01      17.153
         Saint Petersburg       13364  2013-09-01         NaN

[1155 rows x 3 columns]
                    Unnamed: 0        date  avg_temp_c
country city
Mexico  Mexico           10230  2000-01-01      12.694
        Mexico           10231  2000-02-01      14.677
        Mexico           10232  2000-03-01      17.376
        Mexico           10233  2000-04-01      18.294
        Mexico           10234  2000-05-01      18.562
```

```
…                           …          …          …
Morocco Casablanca        3130  2013-05-01      19.217
        Casablanca        3131  2013-06-01      23.649
        Casablanca        3132  2013-07-01      27.488
        Casablanca        3133  2013-08-01      27.952
        Casablanca        3134  2013-09-01         NaN

[330 rows x 3 columns]
                 Unnamed: 0        date  avg_temp_c
country  city
Pakistan Lahore        8415  2000-01-01      12.792
         Lahore        8416  2000-02-01      14.339
         Lahore        8417  2000-03-01      20.309
         Lahore        8418  2000-04-01      29.072
         Lahore        8419  2000-05-01      34.845
…                        …          …          …
Russia   Moscow       10885  2013-05-01      16.152
         Moscow       10886  2013-06-01      18.718
         Moscow       10887  2013-07-01      18.136
         Moscow       10888  2013-08-01      17.485
         Moscow       10889  2013-09-01         NaN

[660 rows x 3 columns]
                   Unnamed: 0        date  avg_temp_c
country city
India   Hyderabad        5940  2000-01-01      23.779
        Hyderabad        5941  2000-02-01      25.826
        Hyderabad        5942  2000-03-01      28.821
        Hyderabad        5943  2000-04-01      32.698
        Hyderabad        5944  2000-05-01      32.438
…                          …          …          …
Iraq    Baghdad          1150  2013-05-01      28.673
        Baghdad          1151  2013-06-01      33.803
        Baghdad          1152  2013-07-01      36.392
        Baghdad          1153  2013-08-01      35.463
        Baghdad          1154  2013-09-01         NaN

[2145 rows x 3 columns]
                          date  avg_temp_c
country     city
Afghanistan Kabul   2000-01-01       3.326
            Kabul   2000-02-01       3.454
            Kabul   2000-03-01       9.612
            Kabul   2000-04-01      17.925
            Kabul   2000-05-01      24.658
…                          …          …
Zimbabwe    Harare  2013-05-01      18.298
            Harare  2013-06-01      17.020
```

```
             Harare  2013-07-01      16.299
             Harare  2013-08-01      19.232
             Harare  2013-09-01         NaN

[16500 rows x 2 columns]
                         date  avg_temp_c
country city
India   Hyderabad  2000-01-01      23.779
        Hyderabad  2000-02-01      25.826
        Hyderabad  2000-03-01      28.821
        Hyderabad  2000-04-01      32.698
        Hyderabad  2000-05-01      32.438
…                         …         …
Iraq    Baghdad    2013-05-01      28.673
        Baghdad    2013-06-01      33.803
        Baghdad    2013-07-01      36.392
        Baghdad    2013-08-01      35.463
        Baghdad    2013-09-01         NaN

[2145 rows x 2 columns]
```

[20]:
```python
temperatures_bool = temperatures[(temperatures["date"] >= "2010") &
 ↪(temperatures["date"] < "2012")]
print(temperatures_bool)

temperatures_ind = temperatures.set_index("date").sort_index()

print(temperatures_ind.loc["2010":"2011"])

print(temperatures_ind.loc["2010-08":"2011-02"])
```

```
       Unnamed: 0        date     city        country  avg_temp_c
120           120  2010-01-01  Abidjan  Côte D'Ivoire      28.270
121           121  2010-02-01  Abidjan  Côte D'Ivoire      29.262
122           122  2010-03-01  Abidjan  Côte D'Ivoire      29.596
123           123  2010-04-01  Abidjan  Côte D'Ivoire      29.068
124           124  2010-05-01  Abidjan  Côte D'Ivoire      28.258
…            …          …       …            …         …
16474       16474  2011-08-01     Xian          China      23.069
16475       16475  2011-09-01     Xian          China      16.775
16476       16476  2011-10-01     Xian          China      12.587
16477       16477  2011-11-01     Xian          China       7.543
16478       16478  2011-12-01     Xian          China      -0.490

[2400 rows x 5 columns]
            Unnamed: 0        city    country  avg_temp_c
date
2010-01-01        4905  Faisalabad   Pakistan      11.810
```

```
2010-01-01       10185   Melbourne  Australia     20.016
2010-01-01        3750   Chongqing      China      7.921
2010-01-01       13155   São Paulo     Brazil     23.738
2010-01-01        5400   Guangzhou      China     14.136
…                …          …         …         …
2010-12-01        6896     Jakarta  Indonesia     26.602
2010-12-01        5246       Gizeh      Egypt     16.530
2010-12-01       11186      Nagpur      India     19.120
2010-12-01       14981      Sydney  Australia     19.559
2010-12-01       13496    Salvador     Brazil     26.265

[1200 rows x 4 columns]
            Unnamed: 0          city        country  avg_temp_c
date
2010-08-01        2602      Calcutta          India      30.226
2010-08-01       12337          Pune          India      24.941
2010-08-01        6562         Izmir         Turkey      28.352
2010-08-01       15637       Tianjin          China      25.543
2010-08-01        9862        Manila    Philippines      27.101
…                …             …             …         …
2011-01-01        4257 Dar Es Salaam       Tanzania      28.541
2011-01-01       11352       Nairobi          Kenya      17.768
2011-01-01         297   Addis Abeba       Ethiopia      17.708
2011-01-01       11517       Nanjing          China       0.144
2011-01-01       11847      New York  United States      -4.463

[600 rows x 4 columns]
```

[21]:

```python
print(temperatures.iloc[22,1])

print(temperatures.iloc[0:5,])

print(temperatures.iloc[:,2:4])

print(temperatures.iloc[0:5,2:4])
```

```
2001-11-01
   Unnamed: 0        date     city        country  avg_temp_c
0           0  2000-01-01  Abidjan  Côte D'Ivoire      27.293
1           1  2000-02-01  Abidjan  Côte D'Ivoire      27.685
2           2  2000-03-01  Abidjan  Côte D'Ivoire      29.061
3           3  2000-04-01  Abidjan  Côte D'Ivoire      28.162
4           4  2000-05-01  Abidjan  Côte D'Ivoire      27.547
          city        country
0      Abidjan  Côte D'Ivoire
1      Abidjan  Côte D'Ivoire
2      Abidjan  Côte D'Ivoire
3      Abidjan  Côte D'Ivoire
```

```
4      Abidjan  Côte D'Ivoire
…        …            …
16495     Xian          China
16496     Xian          China
16497     Xian          China
16498     Xian          China
16499     Xian          China

[16500 rows x 2 columns]
      city        country
0  Abidjan  Côte D'Ivoire
1  Abidjan  Côte D'Ivoire
2  Abidjan  Côte D'Ivoire
3  Abidjan  Côte D'Ivoire
4  Abidjan  Côte D'Ivoire
```

1.8 Cálculos con tablas dinámicas

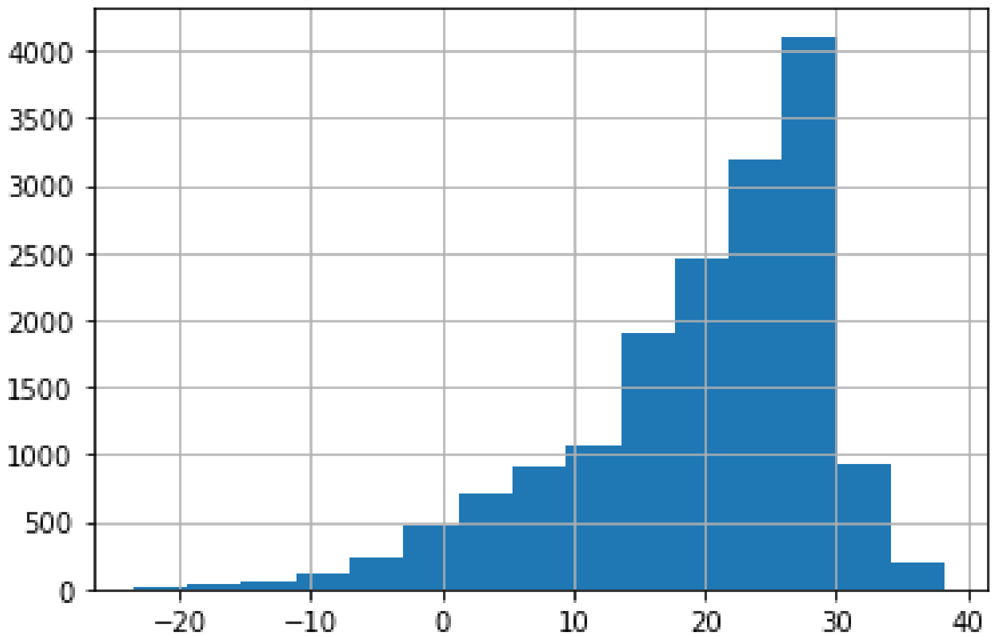
[22]:

```python
# Add a year column to temperatures
#temperatures["year"] = temperatures["date"].dt.year

# Pivot avg_temp_c by country and city vs year
#temp_by_country_city_vs_year = temperatures.pivot_table('avg_temp_c',␣
↪index=['country', 'city'],columns='year')

# See the result
#print(temp_by_country_city_vs_year)

# Subset for Egypt to India
#temp_by_country_city_vs_year.loc['Egypt':'India']

# Subset for Egypt, Cairo to India, Delhi
#temp_by_country_city_vs_year.loc[('Egypt','Cairo'):('India','Delhi')]

# Subset for Egypt, Cairo to India, Delhi, and 2005 to 2010
#temp_by_country_city_vs_year.loc[('Egypt','Cairo'):('India','Delhi'),'2005':
↪'2010']

# Get the worldwide mean temp by year
#mean_temp_by_year = temp_by_country_city_vs_year.mean(axis = 'index')

# Filter for the year that had the highest mean temp
#print(mean_temp_by_year[mean_temp_by_year == mean_temp_by_year.max()])

# Get the mean temp by city
#mean_temp_by_city = temp_by_country_city_vs_year.mean(axis = 'columns')
```

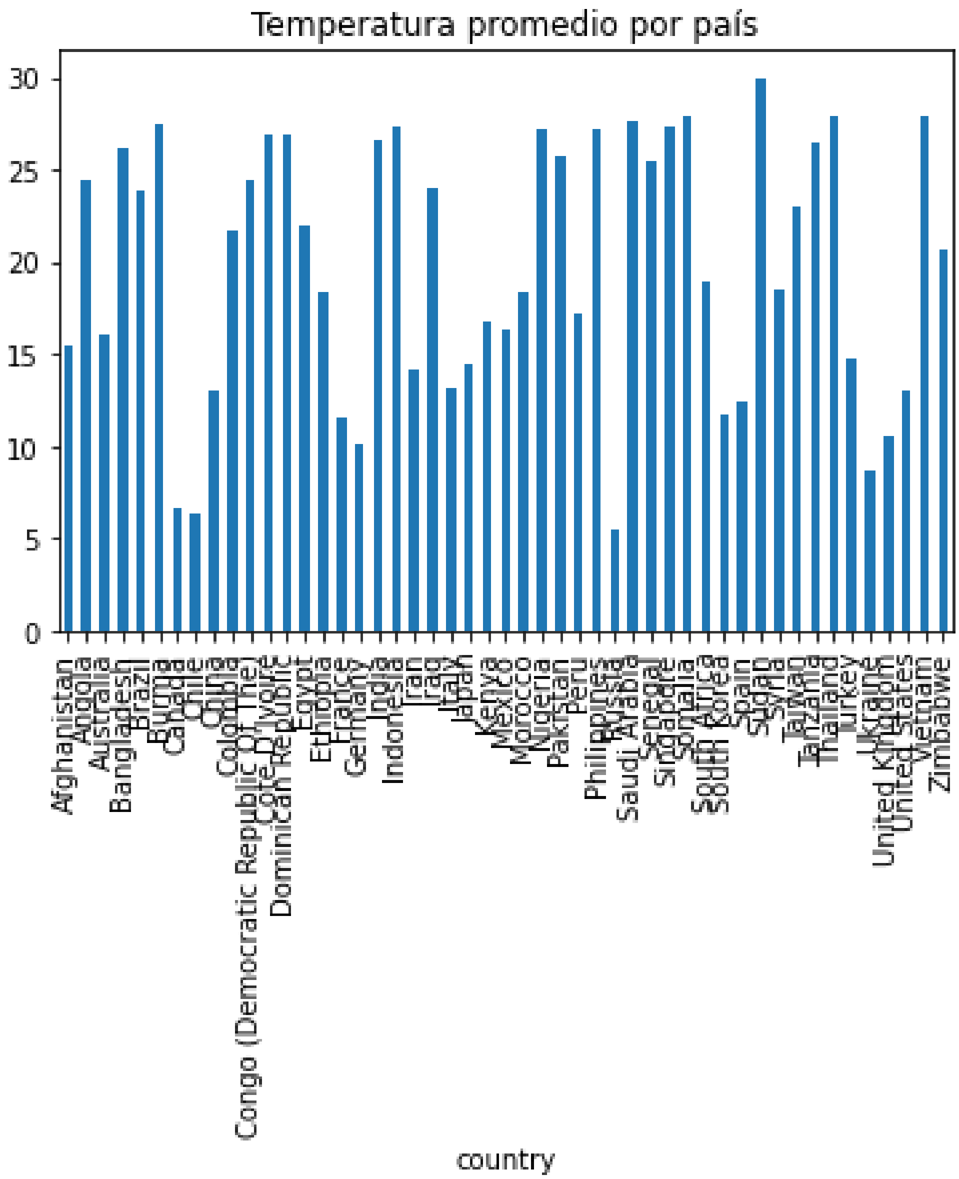
```
# Filter for the city that had the lowest mean temp
#print(mean_temp_by_city[mean_temp_by_city == mean_temp_by_city.min()])
```

1.9 Visualización de datos

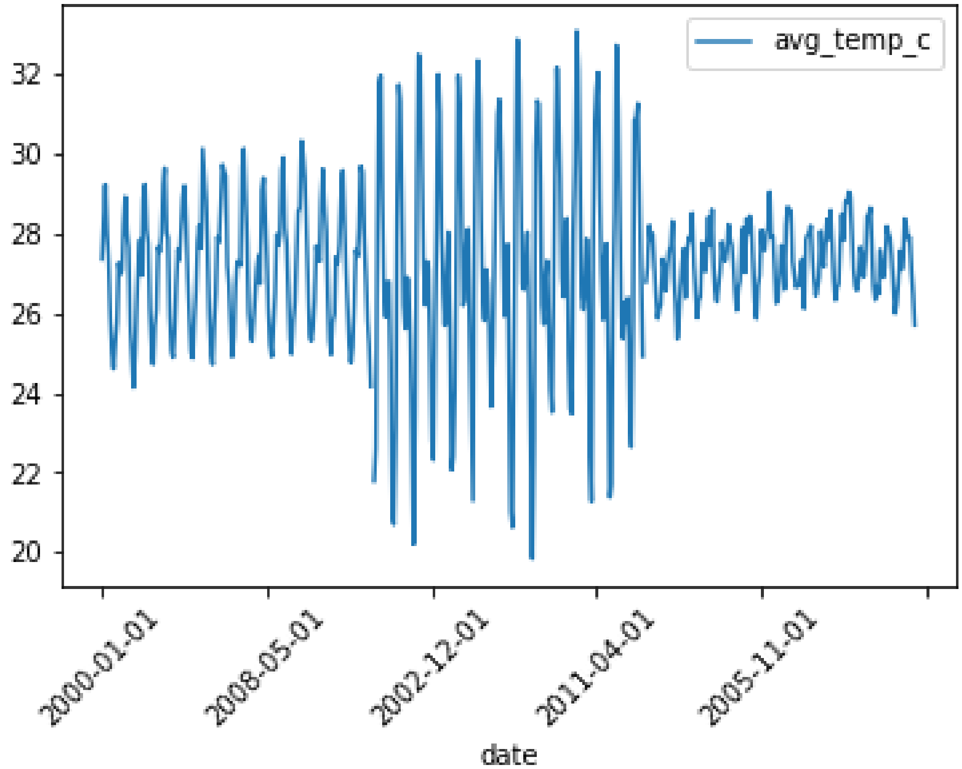
[23]:
```
### Histogramas

temperatures["avg_temp_c"].hist(bins = 15)
plt.show()
```



[24]:
```
avg_temp_by_country = temperatures.groupby("country")["avg_temp_c"].mean()
avg_temp_by_country.plot(kind = "bar", title = "Temperatura promedio por país")
plt.show()
```

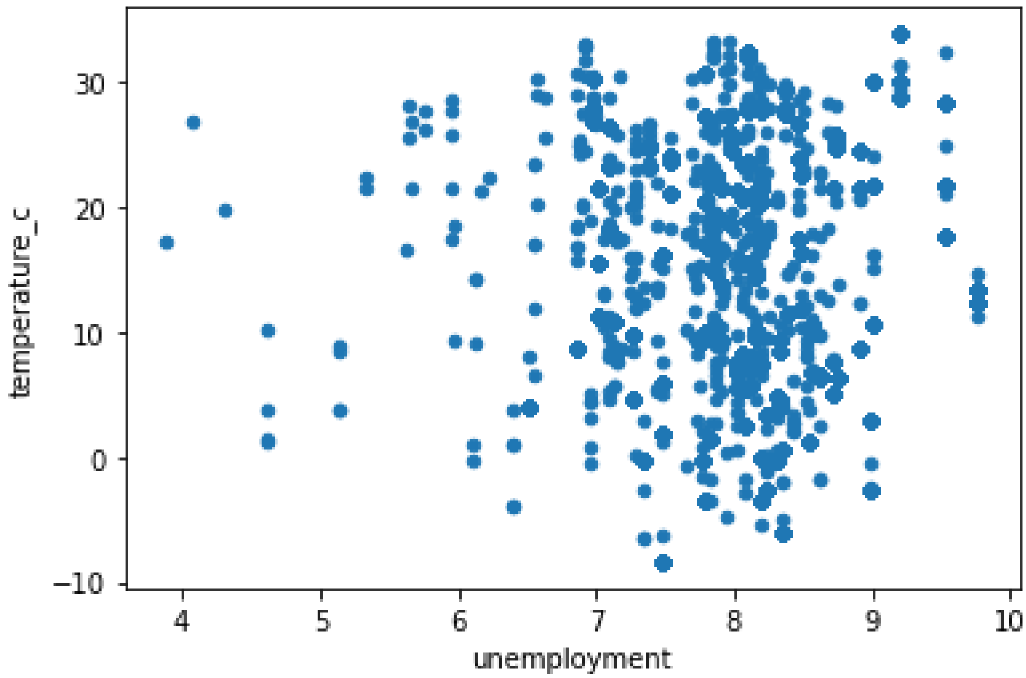



```
[25]: ### Gráficos de línea

nigeria = temperatures[temperatures["country"] == "Nigeria"]

nigeria.plot(x = "date", y = "avg_temp_c", kind = "line", rot = 45)
plt.show()
```



[26]:
```
### Scatter plots

sales = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/sales_subset.csv")

sales.plot("unemployment", "temperature_c", kind = "scatter")
```

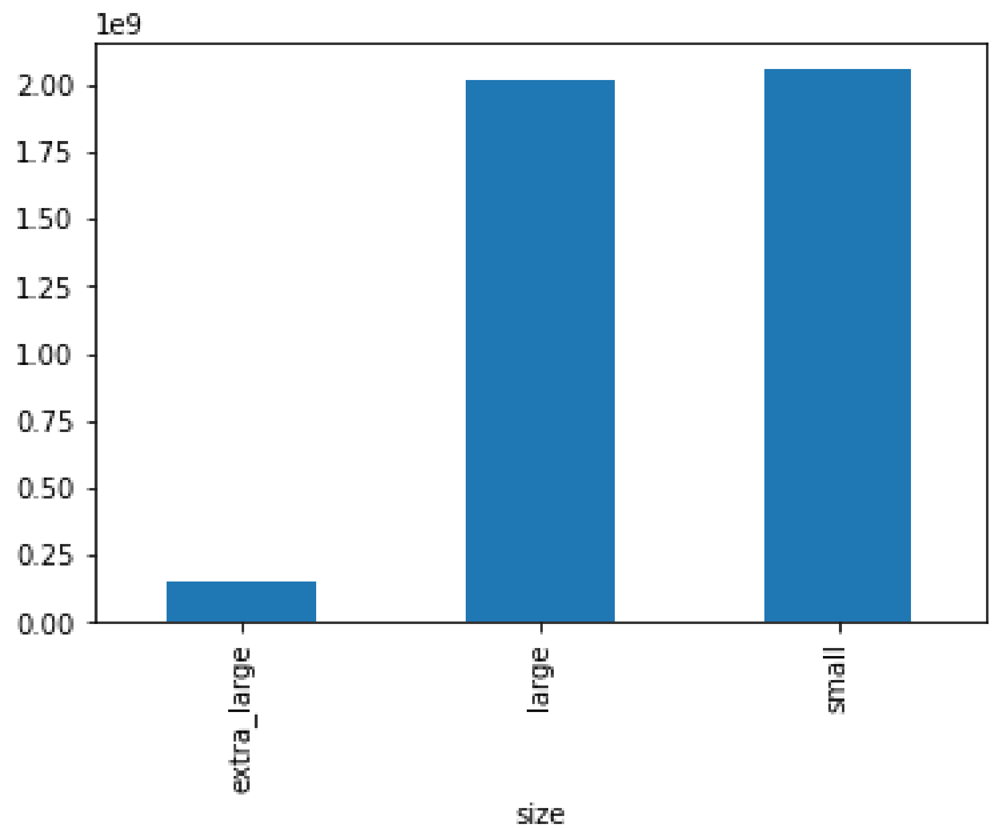
[26]:
```
<AxesSubplot:xlabel='unemployment', ylabel='temperature_c'>
```

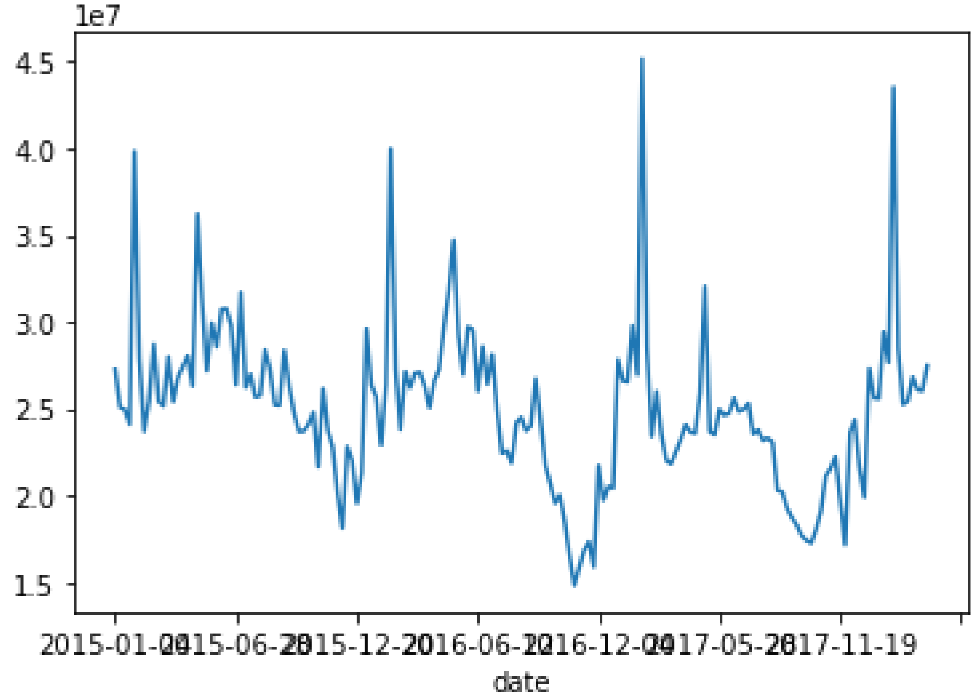


[27]:
```
### Ejemplo aguacates

import pickle

with open('C:/Users/marco/Data Camp Python/Datasets/avoplotto.pkl', 'rb') as f:
    avocados = pickle.load(f)

print(avocados.head())

nb_sold_by_size = avocados.groupby("size")["nb_sold"].sum()

nb_sold_by_size.plot(kind = "bar")
plt.show()

###

nb_sold_by_date = avocados.groupby("date")["nb_sold"].sum()

nb_sold_by_date.plot(kind = "line")
plt.show()

###
```

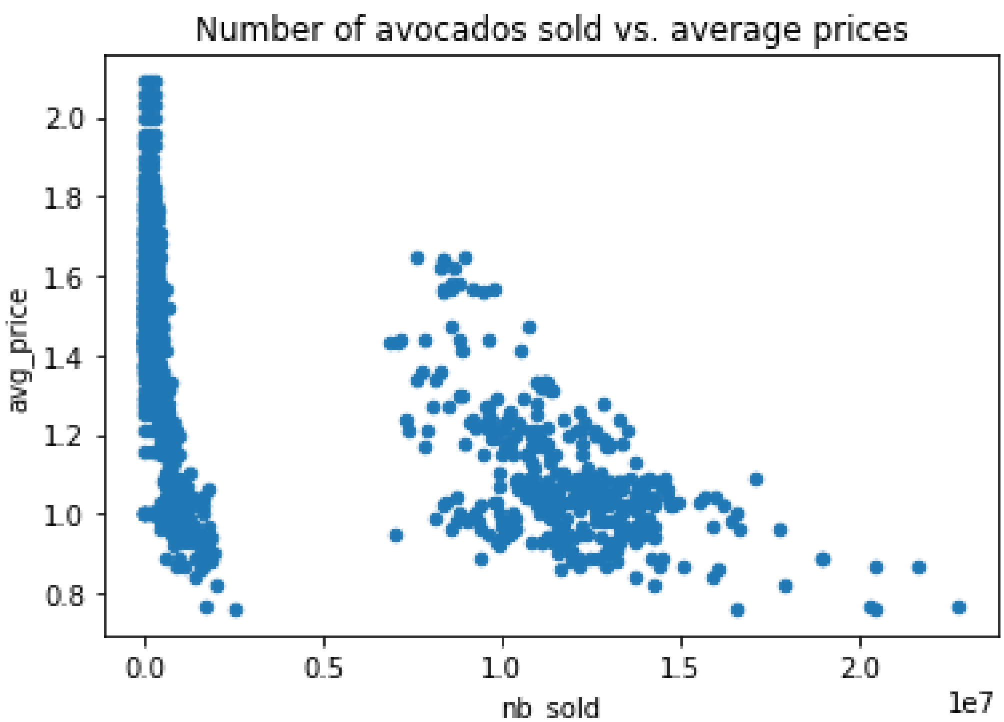
```
avocados.plot(x = "nb_sold", y = "avg_price", kind = "scatter", title = "Number␣
↪of avocados sold vs. average prices")
plt.show()

###

avocados[avocados["type"] == "conventional"]["avg_price"].hist(bins = 20, alpha␣
↪= 0.5)

avocados[avocados["type"] == "organic"]["avg_price"].hist(bins = 20, alpha = 0.
↪5)

plt.legend(["conventional", "organic"])

plt.show()
```

```
         date          type  year  avg_price   size     nb_sold
0  2015-12-27  conventional  2015       0.95  small  9626901.09
1  2015-12-20  conventional  2015       0.98  small  8710021.76
2  2015-12-13  conventional  2015       0.93  small  9855053.66
3  2015-12-06  conventional  2015       0.89  small  9405464.36
4  2015-11-29  conventional  2015       0.99  small  8094803.56
```

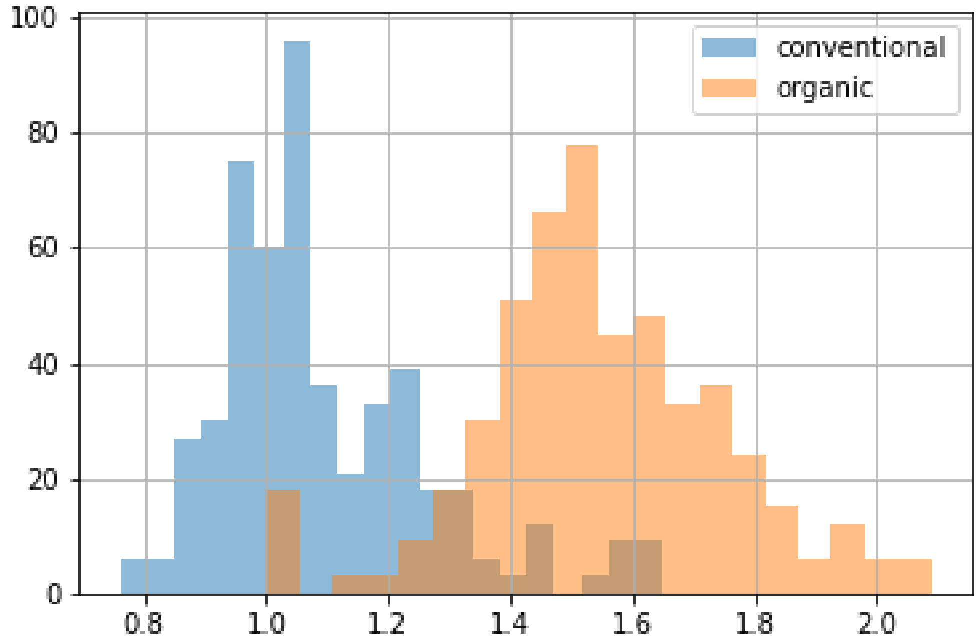




1e7
4.5
4.0
3.5
3.0
2.5
2.0
1.5
date
2015-01-04
2017-11-19



Number of avocados sold vs. average prices
avg_price
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.0
0.5
1.0
1.5
2.0
nb_sold
1e7



1.10 Missing values

[28]:
```
# Es buena idea explorar los NAs de un dataframe:

print(sales.isna())

print(temperatures.isna().any())

print(df1.isna().sum())

# Se pueden eliminar:

print(sales.dropna())

# O sustituir con ceros:

print(temperatures.fillna(0))
```

```
       Unnamed: 0  store   type  department   date  weekly_sales  is_holiday  \
0           False  False  False       False  False         False       False
1           False  False  False       False  False         False       False
2           False  False  False       False  False         False       False
3           False  False  False       False  False         False       False
4           False  False  False       False  False         False       False
```

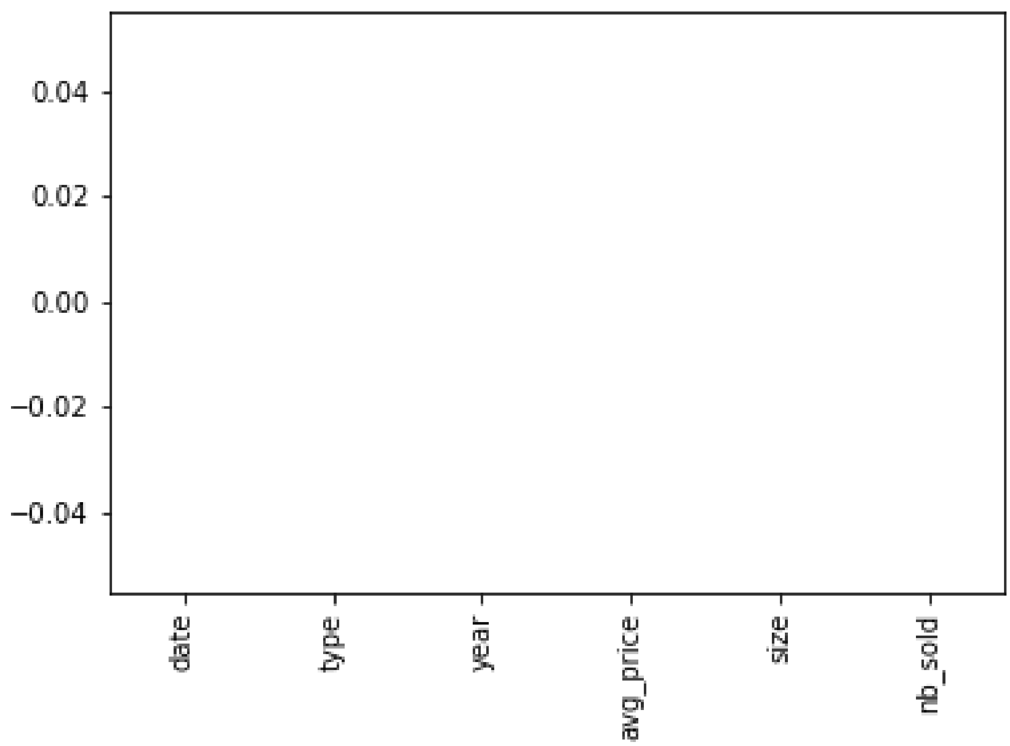
```
…           …     …     …         …     …           …         …
10769       False  False  False       False  False         False       False
10770       False  False  False       False  False         False       False
10771       False  False  False       False  False         False       False
10772       False  False  False       False  False         False       False
10773       False  False  False       False  False         False       False

       temperature_c  fuel_price_usd_per_l  unemployment
0              False                 False         False
1              False                 False         False
2              False                 False         False
3              False                 False         False
4              False                 False         False
…              …                   …            …
10769          False                 False         False
10770          False                 False         False
10771          False                 False         False
10772          False                 False         False
10773          False                 False         False

[10774 rows x 10 columns]
Unnamed: 0    False
date          False
city          False
country       False
avg_temp_c     True
dtype: bool
LatD     0
LatM     0
LatS     0
NS       0
LonD     0
LonM     0
LonS     0
EW       0
City     0
State    0
dtype: int64
       Unnamed: 0  store type  department        date  weekly_sales  \
0               0      1    A           1  2010-02-05      24924.50
1               1      1    A           1  2010-03-05      21827.90
2               2      1    A           1  2010-04-02      57258.43
3               3      1    A           1  2010-05-07      17413.94
4               4      1    A           1  2010-06-04      17558.09
…           …     …   …         …        …          …
10769       10769     39    A          99  2011-12-09        895.00
10770       10770     39    A          99  2012-02-03        350.00
10771       10771     39    A          99  2012-06-08        450.00
```

```
10772        10772     39     A          99  2012-07-13          0.06
10773        10773     39     A          99  2012-10-05        915.00

       is_holiday  temperature_c  fuel_price_usd_per_l  unemployment
0           False       5.727778              0.679451         8.106
1           False       8.055556              0.693452         8.106
2           False      16.816667              0.718284         7.808
3           False      22.527778              0.748928         7.808
4           False      27.050000              0.714586         7.808
…             …              …                   …            …
10769       False       9.644444              0.834256         7.716
10770       False      15.938889              0.887619         7.244
10771       False      27.288889              0.911922         6.989
10772       False      25.644444              0.860145         6.623
10773       False      22.250000              0.955511         6.228

[10774 rows x 10 columns]
       Unnamed: 0        date     city         country  avg_temp_c
0               0  2000-01-01  Abidjan  Côte D'Ivoire      27.293
1               1  2000-02-01  Abidjan  Côte D'Ivoire      27.685
2               2  2000-03-01  Abidjan  Côte D'Ivoire      29.061
3               3  2000-04-01  Abidjan  Côte D'Ivoire      28.162
4               4  2000-05-01  Abidjan  Côte D'Ivoire      27.547
…             …         …       …             …         …
16495       16495  2013-05-01     Xian          China      18.979
16496       16496  2013-06-01     Xian          China      23.522
16497       16497  2013-07-01     Xian          China      25.251
16498       16498  2013-08-01     Xian          China      24.528
16499       16499  2013-09-01     Xian          China       0.000

[16500 rows x 5 columns]
```

[29]:
```python
### Ejemplo aguacates

print(avocados.isna())

print(avocados.isna().any())

avocados.isna().sum().plot(kind = "bar")
plt.show()

###

avodados_complete = avocados.dropna()
```

```
       date   type   year  avg_price   size  nb_sold
0     False  False  False      False  False    False
1     False  False  False      False  False    False
```

```
2     False  False  False      False  False    False
3     False  False  False      False  False    False
4     False  False  False      False  False    False
…     …    …    …         …    …       …
1009  False  False  False      False  False    False
1010  False  False  False      False  False    False
1011  False  False  False      False  False    False
1012  False  False  False      False  False    False
1013  False  False  False      False  False    False

[1014 rows x 6 columns]
date         False
type         False
year         False
avg_price    False
size         False
nb_sold      False
dtype: bool
```



1.11 Creando dataframes

[30]:
```python
### Diccionarios

my_dict = {
    "title" : "Charlotte's Web",
    "author" : "E.B. White",
    "published" : 1952
}

print(my_dict["title"])

# Se puede crear un dataframe a partir de una lista de diccionarios (renglón␣
 ↪por renglón) o a partir de un diccionario de listas (columna por columna)

# Lista de diccionarios:

list_of_dicts = [
    {"name": "Ginger", "breed": "Dachshund", "height_cm" : 22,
    "weight_kg": 10, "date_of_birth": "2019-03-14"},
    {"name": "Scout", "breed": "Dalmatian", "height_cm" : 59,
    "weight_kg": 25, "date_of_birth": "2019-05-09"}
]

new_dogs = pd.DataFrame(list_of_dicts)
print(new_dogs)

# Diccionario de listas:

dict_of_lists = {
    "name": ["Ginger", "Scout"],
    "breed": ["Dachshund", "Dalmatian"],
    "height_cm": [22, 59],
    "weight_kg": [10, 25],
    "date_of_birth": ["2019-03-14", "2019-05-09"]
}

new_dogs1 = pd.DataFrame(dict_of_lists)
print(new_dogs1)
```

```
Charlotte's Web
     name      breed  height_cm  weight_kg date_of_birth
0  Ginger  Dachshund         22         10    2019-03-14
1   Scout  Dalmatian         59         25    2019-05-09
     name      breed  height_cm  weight_kg date_of_birth
0  Ginger  Dachshund         22         10    2019-03-14
1   Scout  Dalmatian         59         25    2019-05-09
```

[31]:
```
### Ejemplo aguacates

avocados_list = [
    {"date": "2019-11-03", "small_sold": 10376832, "large_sold": 7835071},
    {"date": "2019-11-10", "small_sold": 10717154, "large_sold": 8561348}
]

avocados_2019 = pd.DataFrame(avocados_list)

print(avocados_2019)

###

avocados_dict = {
    "date": ["2019-11-17", "2019-12-01"],
    "small_sold": [10859987, 9291634],
    "large_sold": [7674135, 6238096]
}

avocados_20191 = pd.DataFrame(avocados_dict)

print(avocados_20191)
```

```
         date  small_sold  large_sold
0  2019-11-03    10376832     7835071
1  2019-11-10    10717154     8561348
         date  small_sold  large_sold
0  2019-11-17    10859987     7674135
1  2019-12-01     9291634     6238096
```

2 COMBINANDO DATOS CON PANDAS

2.1 UNIONES BÁSICAS

Las siguientes tablas están relacionadas por la columna “ward”

[32]:
```
import pickle

with open('C:/Users/marco/Data Camp Python/Datasets/ward.p', 'rb') as f:
    wards = pickle.load(f)
print(wards.head())
print(wards.shape)

with open('C:/Users/marco/Data Camp Python/Datasets/census.p', 'rb') as f:
    census = pickle.load(f)
print(census.head())
print(census.shape)
```

```
  ward             alderman                          address    zip
0     1  Proco "Joe" Moreno        2058 NORTH WESTERN AVENUE  60647
1     2        Brian Hopkins       1400 NORTH  ASHLAND AVENUE  60622
2     3           Pat Dowell         5046 SOUTH STATE STREET  60609
3     4     William D. Burns  435 EAST 35TH STREET, 1ST FLOOR  60616
4     5   Leslie A. Hairston            2325 EAST 71ST STREET  60649
(50, 4)
  ward  pop_2000  pop_2010 change                                  address  \
0     1     52951     56149     6%              2765 WEST SAINT MARY STREET
1     2     54361     55805     3%                 WM WASTE MANAGEMENT 1500
2     3     40385     53039    31%                     17 EAST 38TH STREET
3     4     51953     54589     5%  31ST ST HARBOR BUILDING LAKEFRONT TRAIL
4     5     55302     51455    -7%  JACKSON PARK LAGOON SOUTH CORNELL DRIVE

     zip
0  60647
1  60622
2  60653
3  60653
4  60637
(50, 6)
```

[33]:
```python
# El método merge toma el primer dataframe y lo fusiona con el segundo (inner␣
↪join):

import pandas as pd

ward_census = wards.merge(census, on = "ward", suffixes = ("_ward", "_cen")) #␣
↪donde sufixxes se usa para diferencias columnas iguales entre ambos

ward_census.head(4)
```

[33]:
```
  ward             alderman                     address_ward zip_ward  \
0     1  Proco "Joe" Moreno        2058 NORTH WESTERN AVENUE    60647
1     2        Brian Hopkins       1400 NORTH  ASHLAND AVENUE    60622
2     3           Pat Dowell         5046 SOUTH STATE STREET    60609
3     4     William D. Burns  435 EAST 35TH STREET, 1ST FLOOR    60616

  pop_2000  pop_2010 change                              address_cen zip_cen
0     52951     56149     6%              2765 WEST SAINT MARY STREET   60647
1     54361     55805     3%                 WM WASTE MANAGEMENT 1500   60622
2     40385     53039    31%                     17 EAST 38TH STREET   60653
3     51953     54589     5%  31ST ST HARBOR BUILDING LAKEFRONT TRAIL   60653
```

[34]:
```python
# Ejemplo

with open('C:/Users/marco/Data Camp Python/Datasets/taxi_owners.p', 'rb') as f:
```

```
    taxi_owners = pickle.load(f)

with open('C:/Users/marco/Data Camp Python/Datasets/taxi_vehicles.p', 'rb') as␣
↪f:
    taxi_veh = pickle.load(f)

print(taxi_owners.head())
print(taxi_veh.head())

taxi_own_veh = taxi_owners.merge(taxi_veh, on = "vid", suffixes = ("_own",␣
↪"_veh"))
print(taxi_own_veh.columns)
```

```
     rid   vid           owner                 address    zip
0  T6285  6285  AGEAN TAXI LLC     4536 N. ELSTON AVE.  60630
1  T4862  4862    MANGIB CORP.  5717 N. WASHTENAW AVE.  60659
2  T1495  1495   FUNRIDE, INC.     3351 W. ADDISON ST.  60618
3  T4231  4231    ALQUSH CORP.  6611 N. CAMPBELL AVE.  60645
4  T5971  5971  EUNIFFORD INC.     3351 W. ADDISON ST.  60618
    vid    make   model  year fuel_type                owner
0  2767  TOYOTA   CAMRY  2013    HYBRID       SEYED M. BADRI
1  1411  TOYOTA    RAV4  2017    HYBRID          DESZY CORP.
2  6500  NISSAN  SENTRA  2019  GASOLINE       AGAPH CAB CORP
3  2746  TOYOTA   CAMRY  2013    HYBRID  MIDWEST CAB CO, INC
4  5922  TOYOTA   CAMRY  2013    HYBRID       SUMETTI CAB CO
Index(['rid', 'vid', 'owner_own', 'address', 'zip', 'make', 'model', 'year',
       'fuel_type', 'owner_veh'],
      dtype='object')
```

[35]:
```
print(taxi_own_veh['fuel_type'].value_counts())
```

```
HYBRID                    2792
GASOLINE                   611
FLEX FUEL                   89
COMPRESSED NATURAL GAS      27
Name: fuel_type, dtype: int64
```

2.1.1 Relaciones una-varias

En una relación 1-1, cada fila de la tabla A está relacionada con una y solo una fila de la tabla B En una relación una-varias, cada fila de A está relacionada con una o más filas de B.

[36]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/licenses.p', 'rb') as f:
    licenses = pickle.load(f)
print(licenses.head())

ward_licenses = wards.merge(licenses, on = "ward", suffixes = ("_ward", "_lic"))
print(ward_licenses.head())
```

```
# Donde hay varios registros por ward, dado que existen varios negocios por␣
↪ward.
```

```
  account ward  aid                     business              address    zip
0  307071    3  743        REGGIE'S BAR & GRILL       2105 S STATE ST  60616
1      10   10  829                  HONEYBEERS   13200 S HOUSTON AVE  60633
2   10002   14  775                 CELINA DELI     5089 S ARCHER AVE  60632
3   10005   12  NaN  KRAFT FOODS NORTH AMERICA         2005 W 43RD ST  60609
4   10044   44  638  NEYBOUR'S TAVERN & GRILLE  3651 N SOUTHPORT AVE  60613
  ward            alderman              address_ward zip_ward account  aid  \
0    1  Proco "Joe" Moreno  2058 NORTH WESTERN AVENUE    60647   12024  NaN
1    1  Proco "Joe" Moreno  2058 NORTH WESTERN AVENUE    60647   14446  743
2    1  Proco "Joe" Moreno  2058 NORTH WESTERN AVENUE    60647   14624  775
3    1  Proco "Joe" Moreno  2058 NORTH WESTERN AVENUE    60647   14987  NaN
4    1  Proco "Joe" Moreno  2058 NORTH WESTERN AVENUE    60647   15642  814

                business              address_lic zip_lic
0    DIGILOG ELECTRONICS        1038 N ASHLAND AVE   60622
1       EMPTY BOTTLE INC    1035 N WESTERN AVE 1ST   60622
2  LITTLE MEL'S HOT DOG     2205 N CALIFORNIA AVE   60647
3     MR. BROWN'S LOUNGE    2301 W CHICAGO AVE 1ST   60622
4           Beat Kitchen  2000-2100 W DIVISION ST   60622
```

[37]:
```python
with open('C:/Users/marco/Data Camp Python/Datasets/business_owners.p', 'rb')␣
↪as f:
    biz_owners = pickle.load(f)

# Merge the licenses and biz_owners table on account
licenses_owners = pd.merge(licenses, biz_owners, on="account")

# Group the results by title then count the number of accounts
counted_df = licenses_owners.groupby("title").agg({'account':'count'})

# Sort the counted_df in desending order
sorted_df = counted_df.sort_values(by = "account", ascending = False)

# Use .head() method to print the first few rows of sorted_df
print(sorted_df.head())
```

```
                 account
title
PRESIDENT           6259
SECRETARY           5205
SOLE PROPRIETOR     1658
OTHER               1200
VICE PRESIDENT       970
```

2.1.2 Uniendo varios dataframes

[38]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/stations.p', 'rb') as f:
    stations = pickle.load(f)
with open('C:/Users/marco/Data Camp Python/Datasets/cta_ridership.p', 'rb') as␣
↪f:
    ridership = pickle.load(f)
with open('C:/Users/marco/Data Camp Python/Datasets/cta_calendar.p', 'rb') as f:
    cal = pickle.load(f)

# Merge the ridership, cal, and stations tables
ridership_cal_stations = ridership.merge(cal, on=['year','month','day']).
↪merge(stations, on = "station_id")
print(ridership_cal_stations.head())

# Create a filter to filter ridership_cal_stations
filter_criteria = ((ridership_cal_stations['month'] == 7)
                   & (ridership_cal_stations['day_type'] == "Weekday")
                   & (ridership_cal_stations['station_name'] == "Wilson"))

# Use .loc and the filter to select for rides
print(ridership_cal_stations.loc[filter_criteria, 'rides'].sum())
```

```
  station_id  year  month  day  rides        day_type        station_name  \
0      40010  2019      1    1    576  Sunday/Holiday  Austin-Forest Park
1      40010  2019      1    2   1457         Weekday  Austin-Forest Park
2      40010  2019      1    3   1543         Weekday  Austin-Forest Park
3      40010  2019      1    4   1621         Weekday  Austin-Forest Park
4      40010  2019      1    5    719        Saturday  Austin-Forest Park

                 location
0  (41.870851, -87.776812)
1  (41.870851, -87.776812)
2  (41.870851, -87.776812)
3  (41.870851, -87.776812)
4  (41.870851, -87.776812)
140005
```

[39]:
```
# Merge licenses and zip_demo, on zip; and merge the wards on ward

with open('C:/Users/marco/Data Camp Python/Datasets/zip_demo.p', 'rb') as f:
    zip_demo = pickle.load(f)

licenses_zip_ward = licenses.merge(zip_demo, on = "zip").merge(wards, on =␣
↪"ward")

# Print the results by alderman and show median income
print(licenses_zip_ward.groupby("alderman").agg({'income':'median'}))
```

alderman	income
Ameya Pawar	66246
Anthony A. Beale	38206
Anthony V. Napolitano	82226
Ariel E. Reyboras	41307
Brendan Reilly	110215
Brian Hopkins	87143
Carlos Ramirez-Rosa	66246
Carrie M. Austin	38206
Chris Taliaferro	55566
Daniel "Danny" Solis	41226
David H. Moore	33304
Deborah Mell	66246
Debra L. Silverstein	50554
Derrick G. Curtis	65770
Edward M. Burke	42335
Emma M. Mitts	36283
George Cardenas	33959
Gilbert Villegas	41307
Gregory I. Mitchell	24941
Harry Osterman	45442
Howard B. Brookins, Jr.	33304
James Cappleman	79565
Jason C. Ervin	41226
Joe Moore	39163
John S. Arena	70122
Leslie A. Hairston	28024
Margaret Laurino	70122
Marty Quinn	67045
Matthew J. O'Shea	59488
Michael R. Zalewski	42335
Michael Scott, Jr.	31445
Michelle A. Harris	32558
Michelle Smith	100116
Milagros "Milly" Santiago	41307
Nicholas Sposato	62223
Pat Dowell	46340
Patrick Daley Thompson	41226
Patrick J. O'Connor	50554
Proco "Joe" Moreno	87143
Raymond A. Lopez	33959
Ricardo Munoz	31445
Roberto Maldonado	68223
Roderick T. Sawyer	32558
Scott Waguespack	68223
Susan Sadlowski Garza	38417
Tom Tunney	88708

```
Toni L. Foulkes              27573
Walter Burnett, Jr.          87143
William D. Burns            107811
Willie B. Cochran            28024
```

[40]:

```
# Merge land_use and census and merge result with licenses including suffixes
with open('C:/Users/marco/Data Camp Python/Datasets/land_use.p', 'rb') as f:
    land_use = pickle.load(f)

land_cen_lic = land_use.merge(census, on = "ward").merge(licenses, on = "ward",␣
 ↪suffixes = ("_cen", "_lic"))

# Group by ward, pop_2010, and vacant, then count the # of accounts
pop_vac_lic = land_cen_lic.groupby(["ward", "pop_2010", "vacant"],
                                   as_index=False).agg({'account':'count'})

# Sort pop_vac_lic and print the results
sorted_pop_vac_lic = pop_vac_lic.sort_values(["vacant", "account", "pop_2010"],
                                             ascending=[False, True, True])

# Print the top few rows of sorted_pop_vac_lic
print(sorted_pop_vac_lic.head())
```

```
   ward  pop_2010  vacant  account
47    7     51581      19       80
12   20     52372      15      123
1    10     51535      14      130
16   24     54909      13       98
7    16     51954      13      156
```

2.2 OTROS TIPOS DE UNIÓN

El left join devuelve todas las filas de la tabla A y solo aquellas filas de la tabla B donde coincidan las columnas clave.

[41]:

```
with open('C:/Users/marco/Data Camp Python/Datasets/movies.p', 'rb') as f:
    movies = pickle.load(f)

print(movies.head())
print(movies.shape)

with open('C:/Users/marco/Data Camp Python/Datasets/taglines.p', 'rb') as f:
    taglines = pickle.load(f)
print(taglines.head())
print(taglines.shape)
```

```
      id                 title  popularity release_date
0    257          Oliver Twist   20.415572   2005-09-23
1  14290  Better Luck Tomorrow    3.877036   2002-01-12
```

```
2  38365             Grown Ups   38.864027   2010-06-24
3   9672              Infamous    3.680896   2006-11-16
4  12819       Alpha and Omega   12.300789   2010-09-17
(4803, 4)
       id                                          tagline
0   19995                     Enter the World of Pandora.
1     285  At the end of the world, the adventure begins.
2  206647                            A Plan No One Escapes
3   49026                                 The Legend Ends
4   49529            Lost in our world, found in another.
(3955, 2)
```

[42]:
```
movies_taglines = movies.merge(taglines, on = "id", how = "left") # el how␣
↪default es "inner"
print(movies_taglines.head())
print(movies_taglines.shape)
```

```
      id                 title  popularity release_date  \
0    257          Oliver Twist   20.415572   2005-09-23
1  14290  Better Luck Tomorrow    3.877036   2002-01-12
2  38365             Grown Ups   38.864027   2010-06-24
3   9672              Infamous    3.680896   2006-11-16
4  12819       Alpha and Omega   12.300789   2010-09-17

                                          tagline
0                                             NaN
1             Never underestimate an overachiever.
2  Boys will be boys. . . some longer than others.
3                There's more to the story than you know
4                        A Pawsome 3D Adventure
(4803, 5)
```

[43]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/financials.p', 'rb') as f:
    financials = pickle.load(f)

# Merge movies and financials with a left join
movies_financials = movies.merge(financials, on = "id", how = "left")

# Count the number of rows in the budget column that are missing
number_of_missing_fin = movies_financials['budget'].isnull().sum()

# Print the number of movies missing financials
print(number_of_missing_fin)
```

```
1574
```

[44]:
```
toy_story = movies[movies['title'].str.contains("Toy Story")]
```

```
# Merge the toy_story and taglines tables with a left join
toystory_tag = toy_story.merge(taglines, on = "id", how = "left")

# Print the rows and shape of toystory_tag
print(toystory_tag)
print(toystory_tag.shape)

# Merge the toy_story and taglines tables with a inner join
toystory_tag = toy_story.merge(taglines, on = "id", how = "inner")

# Print the rows and shape of toystory_tag
print(toystory_tag)
print(toystory_tag.shape)
```

```
      id        title  popularity release_date                  tagline
0  10193  Toy Story 3   59.995418   2010-06-16  No toy gets left behind.
1    863  Toy Story 2   73.575118   1999-10-30        The toys are back!
2    862    Toy Story   73.640445   1995-10-30                       NaN
(3, 5)
      id        title  popularity release_date                  tagline
0  10193  Toy Story 3   59.995418   2010-06-16  No toy gets left behind.
1    863  Toy Story 2   73.575118   1999-10-30        The toys are back!
(2, 5)
```

2.2.1 Otras uniones

El right join revolverá todas las filas de la tabla B y solo las filas de la tabla A que tengan valores coincidentes.

[45]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/movie_to_genres.p', 'rb')␣
↪as f:
    movie_to_genres = pickle.load(f)

m = movie_to_genres["genre"] == "TV Movie"
tv_genre = movie_to_genres[m]
print(tv_genre)

# Movies será la tabla A y la funsionaremos con la tabla B, tv_genre

tv_movies = movies.merge(tv_genre, how = "right", left_on = "id", right_on =␣
↪"movie_id") # donde los dos últimos argumentos indican qué
# columnas clave de cada tabla usar para el merge
print(tv_movies.head())
```

```
      movie_id     genre
4998     10947  TV Movie
5994     13187  TV Movie
7443     22488  TV Movie
```

```
10061     78814  TV Movie
10790    153397  TV Movie
10835    158150  TV Movie
11096    205321  TV Movie
11282    231617  TV Movie
       id                      title  popularity release_date  movie_id  \
0   10947         High School Musical   16.536374   2006-01-20     10947
1   13187  A Charlie Brown Christmas    8.701183   1965-12-09     13187
2   22488           Love's Abiding Joy    1.128559   2006-10-06     22488
3   78814          We Have Your Husband    0.102003   2011-11-12     78814
4  153397                     Restless    0.812776   2012-12-07    153397

      genre
0  TV Movie
1  TV Movie
2  TV Movie
3  TV Movie
4  TV Movie
```

El outer join devolverá todas las filas de ambas tablas, independientemente si hay una coincidencia entre ellas o no.

[46]:

```python
m1 = movie_to_genres["genre"] == "Family"
family = movie_to_genres[m1].head(3)

m2 = movie_to_genres["genre"] == "Comedy"
comedy = movie_to_genres[m2].head(3)

family_comedy = family.merge(comedy, on = "movie_id", how = "outer", suffixes =
 ↪("_fam", "_com"))
print(family_comedy)
```

```
   movie_id genre_fam genre_com
0        12    Family       NaN
1        35    Family    Comedy
2       105    Family       NaN
3         5       NaN    Comedy
4        13       NaN    Comedy
```

[47]:

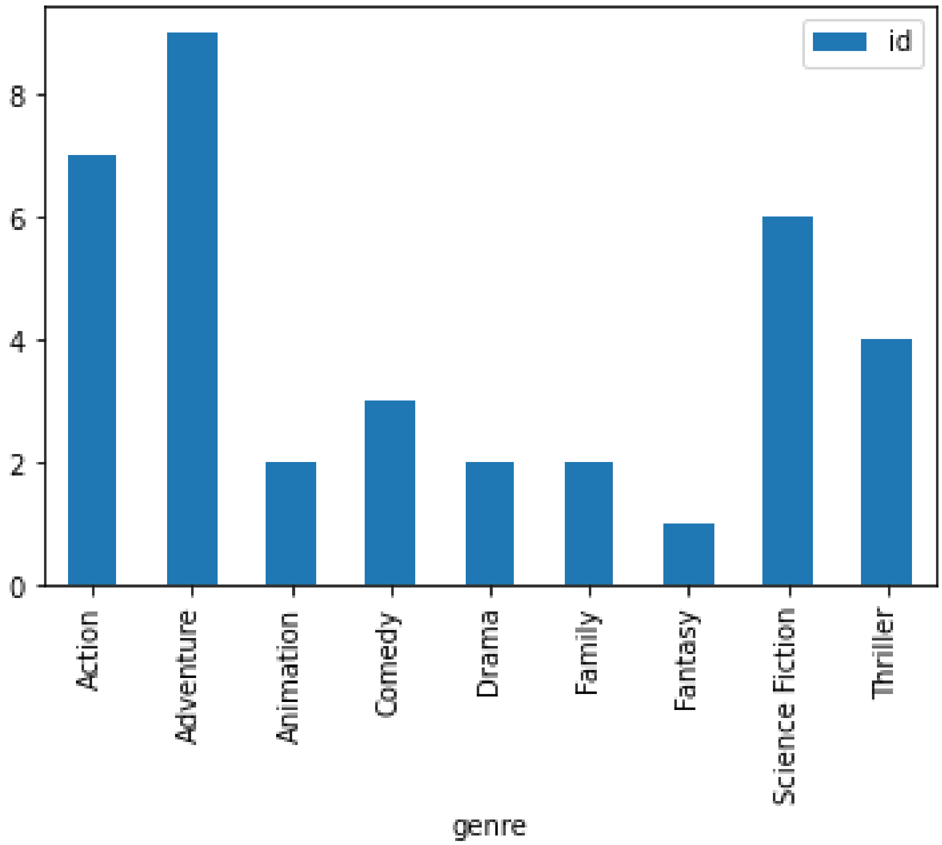
```python
# Ejemplo

action_movies = movie_to_genres[movie_to_genres['genre'] == 'Action']
scifi_movies = movie_to_genres[movie_to_genres['genre'] == 'Science Fiction']

# Merge action_movies to the scifi_movies with right join
action_scifi = action_movies.merge(scifi_movies, on='movie_id', how='right',
                                   suffixes=('_act','_sci'))
```

```
print(action_scifi.head())

# From action_scifi, select only the rows where the genre_act column is null
scifi_only = action_scifi[action_scifi['genre_act'].isnull()]

# Merge the movies and scifi_only tables with an inner join
movies_and_scifi_only = movies.merge(scifi_only, how = "inner", left_on = "id",␣
↪right_on = "movie_id")

# Print the first few rows and shape of movies_and_scifi_only
print(movies_and_scifi_only.head())
print(movies_and_scifi_only.shape)
```

```
   movie_id genre_act        genre_sci
0        11    Action  Science Fiction
1        18    Action  Science Fiction
2        19       NaN  Science Fiction
3        38       NaN  Science Fiction
4        62       NaN  Science Fiction
      id                         title  popularity release_date  movie_id  \
0  18841  The Lost Skeleton of Cadavra    1.680525   2001-09-12     18841
1  26672     The Thief and the Cobbler    2.439184   1993-09-23     26672
2  15301      Twilight Zone: The Movie   12.902975   1983-06-24     15301
3   8452                   The 6th Day   18.447479   2000-11-17      8452
4   1649    Bill & Ted's Bogus Journey   11.349664   1991-07-19      1649

  genre_act        genre_sci
0       NaN  Science Fiction
1       NaN  Science Fiction
2       NaN  Science Fiction
3       NaN  Science Fiction
4       NaN  Science Fiction
(258, 7)
```

[48]:
```
pop_movies = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRYrnc2ncyu2tA-DmL79DOa0WPw9OMwQG7CZHwzFWrhSAqrK97VJnpeuX3nj-3D86jbWOvkaFWIOLcW/
↪pub?gid=478296409&single=true&output=csv')

# Use right join to merge the movie_to_genres and pop_movies tables
genres_movies = movie_to_genres.merge(pop_movies, how='right',
                                      left_on = "movie_id",
                                      right_on = "id")

# Count the number of genres
genre_count = genres_movies.groupby('genre').agg({'id':'count'})

# Plot a bar chart of the genre_count
```

```
genre_count.plot(kind='bar')
plt.show()
```



[49]:
```
iron_1_actors = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRYrnc2ncyu2tA-DmL79D0a0WPw90MwQG7CZHwzFWrhSAqrK97VJnpeuX3nj-3D86jbW0vkaFWIOLcW/
↪pub?gid=1555829608&single=true&output=csv')

iron_2_actors = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRYrnc2ncyu2tA-DmL79D0a0WPw90MwQG7CZHwzFWrhSAqrK97VJnpeuX3nj-3D86jbW0vkaFWIOLcW/
↪pub?gid=940658024&single=true&output=csv')

# Merge iron_1_actors to iron_2_actors on id with outer join using suffixes
iron_1_and_2 = iron_1_actors.merge(iron_2_actors,
                                     on = "id",
                                     how = "outer",
                                     suffixes=("_1", "_2"))

# Create an index that returns true if name_1 or name_2 are null
m = ((iron_1_and_2['name_1'].isnull()) | (iron_1_and_2['name_2'].isnull()))
```

```
# Print the first few rows of iron_1_and_2
print(iron_1_and_2[m].head())
```

```
                     character_1      id           name_1 character_2 name_2
0                         Yinsen   17857       Shaun Toub         NaN    NaN
2   Obadiah Stane / Iron Monger    1229     Jeff Bridges         NaN    NaN
3                    War Machine   18288  Terrence Howard         NaN    NaN
5                           Raza   57452      Faran Tahir         NaN    NaN
8                     Abu Bakaar  173810    Sayed Badreya         NaN    NaN
```

2.2.2 Self-joins

[50]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/sequels.p', 'rb') as f:
    sequels = pickle.load(f)
print(sequels.head())

original_sequels = sequels.merge(sequels, left_on = "sequel", right_on = "id",␣
 ↪suffixes = ("_org", "_seq"))
print(original_sequels.head())
```

```
      id         title  sequel
0  19995        Avatar    <NA>
1    862     Toy Story     863
2    863   Toy Story 2   10193
3    597       Titanic    <NA>
4  24428  The Avengers    <NA>
   id_org                                         title_org  sequel_org  \
0     862                                         Toy Story         863
1     863                                       Toy Story 2       10193
2     675          Harry Potter and the Order of the Phoenix         767
3     121              The Lord of the Rings: The Two Towers         122
4     120  The Lord of the Rings: The Fellowship of the Ring         121

   id_seq                                      title_seq  sequel_seq
0     863                                    Toy Story 2       10193
1   10193                                    Toy Story 3        <NA>
2     767         Harry Potter and the Half-Blood Prince        <NA>
3     122  The Lord of the Rings: The Return of the King        <NA>
4     121          The Lord of the Rings: The Two Towers         122
```

[51]:
```
# Ejemplo

with open('C:/Users/marco/Data Camp Python/Datasets/crews.p', 'rb') as f:
    crews = pickle.load(f)

# Merge the crews table to itself
crews_self_merged = crews.merge(crews, on='id', how='inner',
```

```
                                suffixes=('_dir','_crew'))

# Create a boolean index to select the appropriate rows
boolean_filter = ((crews_self_merged['job_dir'] == 'Director') &
                  (crews_self_merged['job_crew'] != 'Director'))
direct_crews = crews_self_merged[boolean_filter]

# Print the first few rows of direct_crews
print(direct_crews.head())
```

```
        id department_dir   job_dir       name_dir department_crew  \
156  19995      Directing  Director  James Cameron         Editing
157  19995      Directing  Director  James Cameron           Sound
158  19995      Directing  Director  James Cameron      Production
160  19995      Directing  Director  James Cameron         Writing
161  19995      Directing  Director  James Cameron             Art

           job_crew          name_crew
156          Editor  Stephen E. Rivkin
157  Sound Designer  Christopher Boyes
158         Casting          Mali Finn
160          Writer      James Cameron
161    Set Designer    Richard F. Mays
```

2.2.3 Uniones e índices

[52]:
```
with open('C:/Users/marco/Data Camp Python/Datasets/ratings.p', 'rb') as f:
    ratings = pickle.load(f)

# Merge to the movies table the ratings table on the index
movies_ratings = movies.merge(ratings, on = "id", how = "left")

# Print the first few rows of movies_ratings
print(movies_ratings.head())

# Merge sequels and financials on index id
sequels_fin = sequels.merge(financials, on='id', how='left')

# Self merge with suffixes as inner join with left on sequel and right on id
# orig_seq = sequels_fin.merge(sequels_fin, how='inner', left_on='sequel',
                            # right_on='id', right_index=True,
                            # suffixes=('_org','_seq'))

# Add calculation to subtract revenue_org from revenue_seq
# orig_seq['diff'] = orig_seq['revenue_seq'] - orig_seq['revenue_org']

# Select the title_org, title_seq, and diff
# titles_diff = orig_seq[['title_org','title_seq','diff']]
```

```python
# Print the first rows of the sorted titles_diff
# print(titles_diff.sort_values('diff', ascending=False).head())
```

```
      id                 title  popularity release_date  vote_average  \
0    257          Oliver Twist   20.415572   2005-09-23           6.7
1  14290  Better Luck Tomorrow    3.877036   2002-01-12           6.5
2  38365             Grown Ups   38.864027   2010-06-24           6.0
3   9672              Infamous    3.680896   2006-11-16           6.4
4  12819       Alpha and Omega   12.300789   2010-09-17           5.3

   vote_count
0       274.0
1        27.0
2      1705.0
3        60.0
4       124.0
```

2.3 UNIONES Y CONCATENACIONES AVANZADAS

2.3.1 Filtrando uniones

Este proceso se refiere a la filtración de observaciones de una tabla basado en si estas se emparejan o no con una observación de otra tabla.

[53]:
```python
employees = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRDM7SeHLamufa04sSbA6WQrRa7foTL68Z9ggRK42HnjLLcN1m9V_fG5a0eBXmGpXlyuSpFQETLCgjh/
↪pub?gid=0&single=true&output=csv')
print(employees.head())
top_cust = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRDM7SeHLamufa04sSbA6WQrRa7foTL68Z9ggRK42HnjLLcN1m9V_fG5a0eBXmGpXlyuSpFQETLCgjh/
↪pub?gid=805548867&single=true&output=csv')
print(top_cust.head())


# Merge employees and top_cust
empl_cust = employees.merge(top_cust, on='srid',
                                 how='left', indicator=True)

# Select the srid column where _merge is left_only
srid_list = empl_cust.loc[empl_cust['_merge'] == 'left_only', 'srid']

# Get employees not working with top customers
print(employees[employees["srid"].isin(srid_list)])
```

```
   srid    lname    fname                title   hire_date  \
0     1    Adams   Andrew      General Manager 2002-08-14
1     2  Edwards       cy        Sales Manager 2002-05-01
2     3  Peacock     Jane  Sales Support Agent 2002-04-01
```

```
3      4     Park  Margaret   Sales Support Agent 2003-05-03
4      5  Johnson     Steve   Sales Support Agent 2003-10-17

                      email
0    andrew@chinookcorp.com
1        cy@chinookcorp.com
2      jane@chinookcorp.com
3  margaret@chinookcorp.com
4     steve@chinookcorp.com
   cid  srid      fname        lname               phone                 fax  \
0    1     3       Luís    Gonçalves  +55 (12) 3923-5555  +55 (12) 3923-5566
1    2     5     Leonie       Köhler    +49 0711 2842222                 NaN
2    3     3   François     Tremblay   +1 (514) 721-4711                 NaN
3    4     4      Bjørn       Hansen     +47 22 44 22 22                 NaN
4    5     4   František  Wichterlová    +420 2 4172 5555    +420 2 4172 5555

                      email
0      luisg@embraer.com.br
1     leonekohler@surfeu.de
2       ftremblay@gmail.com
3     bjorn.hansen@yahoo.no
4  frantisekw@jetbrains.com
   srid     lname    fname            title  hire_date  \
0     1     Adams   Andrew  General Manager 2002-08-14
1     2   Edwards       cy    Sales Manager 2002-05-01
5     6  Mitchell  Michael       IT Manager 2003-10-17
6     7      King   Robert         IT Staff 2004-01-02
7     8  Callahan    Laura         IT Staff 2004-03-04

                     email
0   andrew@chinookcorp.com
1       cy@chinookcorp.com
5  michael@chinookcorp.com
6   robert@chinookcorp.com
7    laura@chinookcorp.com
```

[54]:
```
non_mus_tcks = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=0&single=true&output=csv')

top_invoices = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=123344532&single=true&output=csv')

genres = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=1996578410&single=true&output=csv')
```

```python
# Merge the non_mus_tck and top_invoices tables on tid
tracks_invoices = non_mus_tcks.merge(top_invoices, on='tid')

# Use .isin() to subset non_mus_tcks to rows with tid in tracks_invoices
top_tracks = non_mus_tcks[non_mus_tcks['tid'].isin(tracks_invoices['tid'])]

# Group the top_tracks by gid and count the tid rows
cnt_by_gid = top_tracks.groupby(['gid'], as_index=False).agg({'tid':"count"})

# Merge the genres table to cnt_by_gid on gid and print
print(cnt_by_gid.merge(genres, on='gid'))
```

```
   gid  tid
0   19    4
1   21    2
2   22    1
```

2.3.2 Concatenando dataframes verticalmente

Para esto, es necesario usar el método de Pandas, .concat(), donde axis = 0 se refiere a la unión vertical.

[55]:
```python
tracks_master = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
 ↪pub?gid=536776690&single=true&output=csv')

tracks_ride = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
 ↪pub?gid=467918101&single=true&output=csv')

tracks_st = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
 ↪pub?gid=406201744&single=true&output=csv')

# Concatenate the tracks
tracks_from_albums = pd.concat([tracks_master, tracks_ride, tracks_st],␣
 ↪sort=True)
print(tracks_from_albums)

# Concatenate the tracks so the index goes from 0 to n-1
tracks_from_albums1 = pd.concat([tracks_master, tracks_ride, tracks_st],
                                ignore_index = True,
                                sort=True)
print(tracks_from_albums1)

# Concatenate the tracks, show only columns names that are in all tables
tracks_from_albums2 = pd.concat([tracks_master, tracks_ride, tracks_st],
```

```python
                                join = "inner",
                                sort=True)
print(tracks_from_albums2)
```

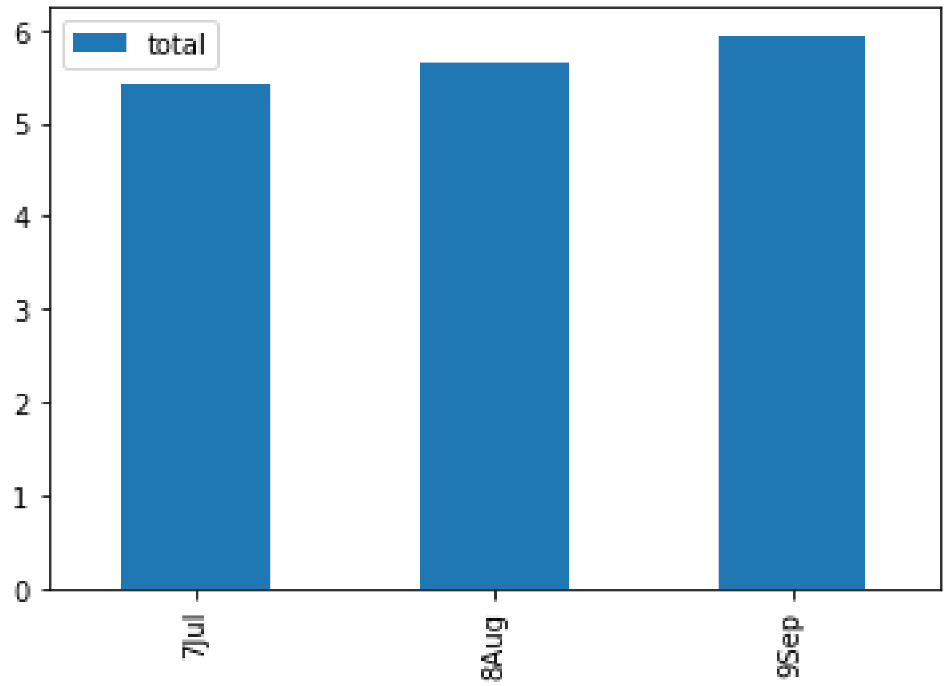
```
   aid  gid  mtid                   name   tid  u_price
0  155    3     1                Frantic  1882     0.99
1  155    3     1              St. Anger  1883     0.99
2  155    3     1     Some Kind Of Monster  1884     0.99
3  155    3     1           Dirty Window  1885     0.99
4  155    3     1          Invisible Kid  1886     0.90
0  154    3     1     Fight Fire With Fire  1874     0.99
1  154    3     1      Ride The Lightning  1875     0.99
2  154    3     1  For Whom The Bell Tolls  1876     0.99
3  154    3     1          Fade To Black  1877     0.99
4  154    3     1       Trapped Under Ice  1878     0.99
0  155    3     1                Frantic  1882     0.99
1  155    3     1              St. Anger  1883     0.99
2  155    3     1     Some Kind Of Monster  1884     0.99
3  155    3     1           Dirty Window  1885     0.99
4  155    3     1          Invisible Kid  1886     0.99
    aid  gid  mtid                   name   tid  u_price
0   155    3     1                Frantic  1882     0.99
1   155    3     1              St. Anger  1883     0.99
2   155    3     1     Some Kind Of Monster  1884     0.99
3   155    3     1           Dirty Window  1885     0.99
4   155    3     1          Invisible Kid  1886     0.90
5   154    3     1     Fight Fire With Fire  1874     0.99
6   154    3     1      Ride The Lightning  1875     0.99
7   154    3     1  For Whom The Bell Tolls  1876     0.99
8   154    3     1          Fade To Black  1877     0.99
9   154    3     1       Trapped Under Ice  1878     0.99
10  155    3     1                Frantic  1882     0.99
11  155    3     1              St. Anger  1883     0.99
12  155    3     1     Some Kind Of Monster  1884     0.99
13  155    3     1           Dirty Window  1885     0.99
14  155    3     1          Invisible Kid  1886     0.99
   aid  gid  mtid                   name   tid  u_price
0  155    3     1                Frantic  1882     0.99
1  155    3     1              St. Anger  1883     0.99
2  155    3     1     Some Kind Of Monster  1884     0.99
3  155    3     1           Dirty Window  1885     0.99
4  155    3     1          Invisible Kid  1886     0.90
0  154    3     1     Fight Fire With Fire  1874     0.99
1  154    3     1      Ride The Lightning  1875     0.99
2  154    3     1  For Whom The Bell Tolls  1876     0.99
3  154    3     1          Fade To Black  1877     0.99
4  154    3     1       Trapped Under Ice  1878     0.99
0  155    3     1                Frantic  1882     0.99
```

```
1  155     3      1                St. Anger  1883      0.99
2  155     3      1      Some Kind Of Monster  1884      0.99
3  155     3      1             Dirty Window  1885      0.99
4  155     3      1            Invisible Kid  1886      0.99
```

[56]:

```
inv_jul = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=386761321&single=true&output=csv')

inv_aug = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=1621349124&single=true&output=csv')

inv_sep = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxpO3UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=1409417594&single=true&output=csv')

# Concatenate the tables and add keys
inv_jul_thr_sep = pd.concat([inv_jul, inv_aug, inv_sep],
                            keys=["7Jul", "8Aug", "9Sep"])

print(inv_jul_thr_sep)

# Group the invoices by the index keys and find avg of the total column
avg_inv_by_month = inv_jul_thr_sep.groupby(level=0).agg({"total": "mean"})

# Bar plot of avg_inv_by_month
avg_inv_by_month.plot(kind = "bar")
plt.show()
```

```
         iid  cid  invoice_date  total       bill_ctry
7Jul 0    42   51         40000   1.98          Sweden
     1    43   53         40000   1.98              UK
     2    44   55         40001   3.96       Australia
     3    45   59         40002   5.94           India
     4    46    6         40005   8.91  Czech Republic
…       …   …           …     …           …
9Sep 28  387   29         41520   3.96          Canada
     29  388   33         41521   5.94          Canada
     30  389   39         41524   8.91          France
     31  390   48         41529  13.86     Netherlands
     32  391    3         41537   0.99          Canada

[103 rows x 5 columns]
```



[57]:
```
invoice_items = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zOOZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=385551090&single=true&output=csv')

# Use the .append() method to combine the tracks tables
metallica_tracks = tracks_ride.append([tracks_master, tracks_st], sort=False)

# Merge metallica_tracks and invoice_items
tracks_invoices = metallica_tracks.merge(invoice_items, on = "tid")

# For each tid and name sum the quantity sold
tracks_sold = tracks_invoices.groupby(['tid','name']).agg({"quantity": "sum"})

# Sort in decending order by quantity and print the results
print(tracks_sold.sort_values(["quantity"], ascending = False))
```

```
                              quantity
tid  name
1876 For Whom The Bell Tolls         2
1882 Frantic                         2
1884 Some Kind Of Monster            2
1886 Invisible Kid                   2
1875 Ride The Lightning              1
1877 Fade To Black                   1
```

2.3.3 Integridad

Al combinar tablas, pueden surgir problemas como relaciones una-varias o varias-varias no intencionales; o bien duplicidad de observaciones.

[58]:
```
classic_18 = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxp03UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=728027621&single=true&output=csv')
classic_19 = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxp03UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=1451981&single=true&output=csv')
pop_18 = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vRGKVF7aP5rJvJX4SzQpU1io--bQ8aa8DPn9i34nSrTr5uHGlCNwxp03UxCW-1hW6NwgZeC-TgvdRg5/
↪pub?gid=813843167&single=true&output=csv')
pop_19 = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=0&single=true&output=csv')

# Concatenate the classic tables vertically
classic_18_19 = pd.concat([classic_18, classic_19], ignore_index = True)

# Concatenate the pop tables vertically
pop_18_19 = pd.concat([pop_18, pop_19], ignore_index=True)

# Merge classic_18_19 with pop_18_19
classic_pop = classic_18_19.merge(pop_18_19, on = "tid")

# Using .isin(), filter classic_18_19 rows where tid is in classic_pop
popular_classic = classic_18_19[classic_18_19["tid"].isin(classic_pop["tid"])]

# Print popular chart
print(popular_classic)
```

```
    pid   tid
3    12  3479
10   12  3439
21   12  3445
23   12  3449
48   12  3437
50   12  3435
```

2.4 UNIÓN DE DATOS ORDENADOS Y FECHAS

merge_ordered() puede fusionar series de tiempo y otros datos ordenados. Además, sirve para completar NAs

[59]:

```
gdp = pd.read_csv('https://assets.datacamp.com/production/repositories/5486/
↪datasets/6ef405912a3801f3ae59d2dd57573f80d598c1fb/WorldBank_GDP.csv')

gdp = gdp.astype(str)

gdp.columns= gdp.columns.str.lower()

sp500 = pd.read_csv('https://assets.datacamp.com/production/repositories/5486/
↪datasets/6666955f71f936ab5fc3b0ee1eb595e19c126c01/S&P500.csv')

sp500 = sp500.astype(str)

sp500.columns= sp500.columns.str.lower()
```

[60]:
```
# Use merge_ordered() to merge gdp and sp500 on year and date
gdp_sp500 = pd.merge_ordered(gdp, sp500, left_on="year", right_on="date",
                             how="left")

# Print gdp_sp500
print(gdp_sp500.head())
```

```
    country name country code     indicator name  year              gdp  \
0          China          CHN  GDP (current US$)  2010   6087160000000.0
1        Germany          DEU  GDP (current US$)  2010   3417090000000.0
2          Japan          JPN  GDP (current US$)  2010   5700100000000.0
3  United States          USA  GDP (current US$)  2010  14992100000000.0
4          China          CHN  GDP (current US$)  2011   7551500000000.0

   date returns
0  2010   12.78
1  2010   12.78
2  2010   12.78
3  2010   12.78
4  2011     0.0
```

[61]:
```
# Use merge_ordered() to merge gdp and sp500, interpolate missing value
gdp_sp500 = pd.merge_ordered(gdp, sp500, left_on = "year", right_on = "date",
how = "left", fill_method = "ffill")


# Print gdp_sp500
print (gdp_sp500.head())

# Subset the gdp and returns columns
gdp_returns = gdp_sp500[["gdp", "returns"]]
gdp_returns = gdp_returns.astype(float)
# Print gdp_returns correlation
```

```
print (gdp_returns.corr())
```

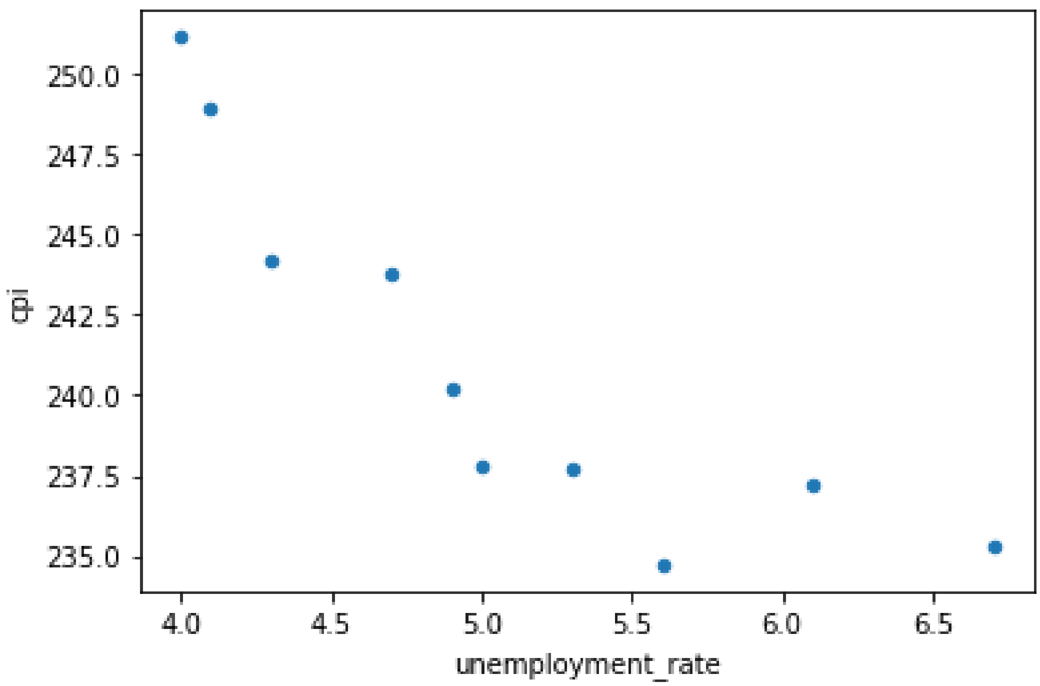
```
    country name country code     indicator name  year               gdp  \
0          China          CHN  GDP (current US$)  2010   6087160000000.0
1        Germany          DEU  GDP (current US$)  2010   3417090000000.0
2          Japan          JPN  GDP (current US$)  2010   5700100000000.0
3  United States          USA  GDP (current US$)  2010  14992100000000.0
4          China          CHN  GDP (current US$)  2011   7551500000000.0

   date returns
0  2010   12.78
1  2010   12.78
2  2010   12.78
3  2010   12.78
4  2011     0.0
              gdp   returns
gdp      1.000000  0.040669
returns  0.040669  1.000000
```

[62]:
```
inflation = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
 ↪pub?gid=1637506110&single=true&output=csv')
unemployment = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
 ↪pub?gid=214585587&single=true&output=csv')
```

[63]:
```
# Use merge_ordered() to merge inflation, unemployment with inner join
inflation_unemploy = pd.merge_ordered(inflation, unemployment, on = "date", how␣
 ↪= "inner")

# Print inflation_unemploy
print(inflation_unemploy)

# Plot a scatter plot of unemployment_rate vs cpi of inflation_unemploy
inflation_unemploy.plot(kind = "scatter", x = "unemployment_rate", y = "cpi")
plt.show()
```

```
         date      cpi     seriesid                  data_type  \
0  2014-01-01  235.288  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
1  2014-06-01  237.231  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
2  2015-01-01  234.718  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
3  2015-06-01  237.684  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
4  2016-01-01  237.833  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
5  2016-06-01  240.167  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
6  2017-01-01  243.780  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
7  2017-06-01  244.182  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
8  2018-01-01  248.884  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
9  2018-06-01  251.134  CUSR0000SA0  SEASONALLY ADJUSTED INDEX
```

```
   unemployment_rate
0                6.7
1                6.1
2                5.6
3                5.3
4                5.0
5                4.9
6                4.7
7                4.3
8                4.1
9                4.0
```



[64]:
```
gdp = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
 ↪pub?gid=1956073838&single=true&output=csv')
pop = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
 ↪pub?gid=305007125&single=true&output=csv')
```

[65]:
```
# Merge gdp and pop on date and country with fill and notice rows 2 and 3
ctry_date = pd.merge_ordered(gdp, pop, on = ["date", "country"],␣
 ↪fill_method='ffill')
```

```python
# Print ctry_date
print(ctry_date)

# Merge gdp and pop on country and date with fill
date_ctry = pd.merge_ordered(gdp, pop, on = ["country", "date"], fill_method =␣
 ↪"ffill")

# Print date_ctry
print(date_ctry)
```

```
          date    country         gdp  series_code_x       pop series_code_y
0   1990-01-01  Australia  158051.132  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
1   1990-01-01     Sweden   79837.846  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
2   1990-04-01  Australia  158263.582  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
3   1990-04-01     Sweden   80582.286  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
4   1990-07-01  Australia  157329.279  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
5   1990-07-01     Sweden   79974.360  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
6   1990-09-01  Australia  158240.678  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
7   1990-09-01     Sweden   80106.497  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
8   1991-01-01  Australia  156195.954  NYGDPMKTPSAKD  17284000   SP.POP.TOTL
9   1991-01-01     Sweden   79524.242  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
10  1991-04-01  Australia  155989.033  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
11  1991-04-01     Sweden   79073.059  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
12  1991-07-01  Australia  156635.858  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
13  1991-07-01     Sweden   79084.770  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
14  1991-09-01  Australia  156744.057  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
15  1991-09-01     Sweden   79740.606  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
16  1992-01-01  Australia  157916.081  NYGDPMKTPSAKD  17495000   SP.POP.TOTL
17  1992-01-01     Sweden   79390.922  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
18  1992-04-01  Australia  159047.827  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
19  1992-04-01     Sweden   79060.283  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
20  1992-07-01  Australia  160658.176  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
21  1992-07-01     Sweden   78904.605  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
22  1992-09-01  Australia  163960.221  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
23  1992-09-01     Sweden   76996.837  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
24  1993-01-01  Australia  165097.495  NYGDPMKTPSAKD  17667000   SP.POP.TOTL
25  1993-01-01     Sweden   75783.588  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
26  1993-04-01  Australia  166027.059  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
27  1993-04-01     Sweden   76708.548  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
28  1993-07-01  Australia  166203.179  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
29  1993-07-01     Sweden   77662.018  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
30  1993-09-01  Australia  169279.348  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
31  1993-09-01     Sweden   77703.304  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
          date    country         gdp  series_code_x       pop series_code_y
0   1990-01-01  Australia  158051.132  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
1   1990-04-01  Australia  158263.582  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
2   1990-07-01  Australia  157329.279  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
3   1990-09-01  Australia  158240.678  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
```

```
4   1991-01-01  Australia  156195.954  NYGDPMKTPSAKD  17284000   SP.POP.TOTL
5   1991-04-01  Australia  155989.033  NYGDPMKTPSAKD  17284000   SP.POP.TOTL
6   1991-07-01  Australia  156635.858  NYGDPMKTPSAKD  17284000   SP.POP.TOTL
7   1991-09-01  Australia  156744.057  NYGDPMKTPSAKD  17284000   SP.POP.TOTL
8   1992-01-01  Australia  157916.081  NYGDPMKTPSAKD  17495000   SP.POP.TOTL
9   1992-04-01  Australia  159047.827  NYGDPMKTPSAKD  17495000   SP.POP.TOTL
10  1992-07-01  Australia  160658.176  NYGDPMKTPSAKD  17495000   SP.POP.TOTL
11  1992-09-01  Australia  163960.221  NYGDPMKTPSAKD  17495000   SP.POP.TOTL
12  1993-01-01  Australia  165097.495  NYGDPMKTPSAKD  17667000   SP.POP.TOTL
13  1993-04-01  Australia  166027.059  NYGDPMKTPSAKD  17667000   SP.POP.TOTL
14  1993-07-01  Australia  166203.179  NYGDPMKTPSAKD  17667000   SP.POP.TOTL
15  1993-09-01  Australia  169279.348  NYGDPMKTPSAKD  17667000   SP.POP.TOTL
16  1990-01-01     Sweden   79837.846  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
17  1990-04-01     Sweden   80582.286  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
18  1990-07-01     Sweden   79974.360  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
19  1990-09-01     Sweden   80106.497  NYGDPMKTPSAKD   8558835   SP.POP.TOTL
20  1991-01-01     Sweden   79524.242  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
21  1991-04-01     Sweden   79073.059  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
22  1991-07-01     Sweden   79084.770  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
23  1991-09-01     Sweden   79740.606  NYGDPMKTPSAKD   8617375   SP.POP.TOTL
24  1992-01-01     Sweden   79390.922  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
25  1992-04-01     Sweden   79060.283  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
26  1992-07-01     Sweden   78904.605  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
27  1992-09-01     Sweden   76996.837  NYGDPMKTPSAKD   8668067   SP.POP.TOTL
28  1993-01-01     Sweden   75783.588  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
29  1993-04-01     Sweden   76708.548  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
30  1993-07-01     Sweden   77662.018  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
31  1993-09-01     Sweden   77703.304  NYGDPMKTPSAKD   8718561   SP.POP.TOTL
```

2.4.1 merge_asof()

Similar a un left join, pero empareja columnas de valores similares, no exactamente iguales. NOTA: los datos deben estar ordenados.

[66]:
```
jpm = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zOOZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=653764921&single=true&output=csv')

jpm["date_time"] = pd.to_datetime(jpm["date_time"])

wells = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zOOZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1331252858&single=true&output=csv')

wells["date_time"] = pd.to_datetime(wells["date_time"])
```

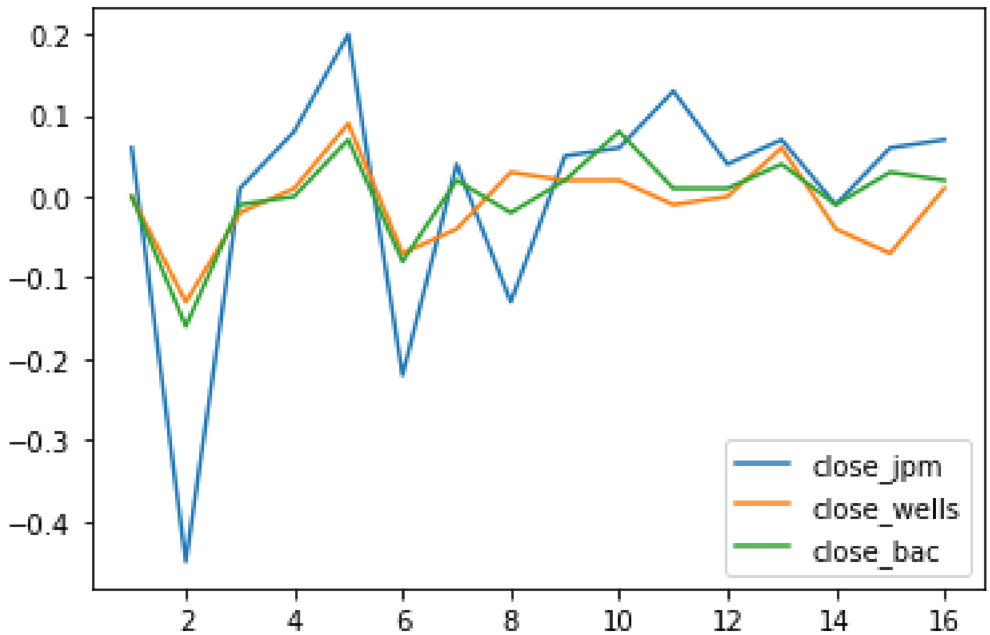
```
bac = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1413472476&single=true&output=csv')

bac["date_time"] = pd.to_datetime(bac["date_time"])

print(jpm.head())
print(wells.head())
print(bac.head())
```

```
            date_time  close
0 2017-11-17 15:35:17  98.12
1 2017-11-17 15:40:04  98.18
2 2017-11-17 15:45:01  97.73
3 2017-11-17 15:50:55  97.74
4 2017-11-17 15:55:00  97.82
            date_time  close
0 2017-11-17 15:35:08  54.32
1 2017-11-17 15:40:00  54.32
2 2017-11-17 15:45:32  54.19
3 2017-11-17 15:50:07  54.17
4 2017-11-17 15:55:00  54.18
            date_time  close
0 2017-11-17 15:35:17  26.55
1 2017-11-17 15:40:06  26.55
2 2017-11-17 15:45:05  26.39
3 2017-11-17 15:50:34  26.38
4 2017-11-17 15:55:06  26.38
```

[67]:
```
# Use merge_asof() to merge jpm and wells
jpm_wells = pd.merge_asof(jpm, wells, on = "date_time", suffixes=('',␣
↪'_wells'), direction = "nearest")


# Use merge_asof() to merge jpm_wells and bac
jpm_wells_bac = pd.merge_asof(jpm_wells, bac, on = "date_time", suffixes =␣
↪("_jpm", "_bac"), direction = "nearest")
print(jpm_wells_bac.head())

# Compute price diff
price_diffs = jpm_wells_bac.diff()
print(price_diffs.head())

# Plot the price diff of the close of jpm, wells and bac only
price_diffs.plot(y=["close_jpm", "close_wells", "close_bac"])
plt.show()
```

```
            date_time  close_jpm  close_wells  close_bac
```

```
0 2017-11-17 15:35:17      98.12        54.32      26.55
1 2017-11-17 15:40:04      98.18        54.32      26.55
2 2017-11-17 15:45:01      97.73        54.19      26.39
3 2017-11-17 15:50:55      97.74        54.17      26.38
4 2017-11-17 15:55:00      97.82        54.18      26.38
         date_time  close_jpm  close_wells  close_bac
0              NaT        NaN          NaN        NaN
1 0 days 00:04:47       0.06         0.00       0.00
2 0 days 00:04:57      -0.45        -0.13      -0.16
3 0 days 00:05:54       0.01        -0.02      -0.01
4 0 days 00:04:05       0.08         0.01       0.00
```



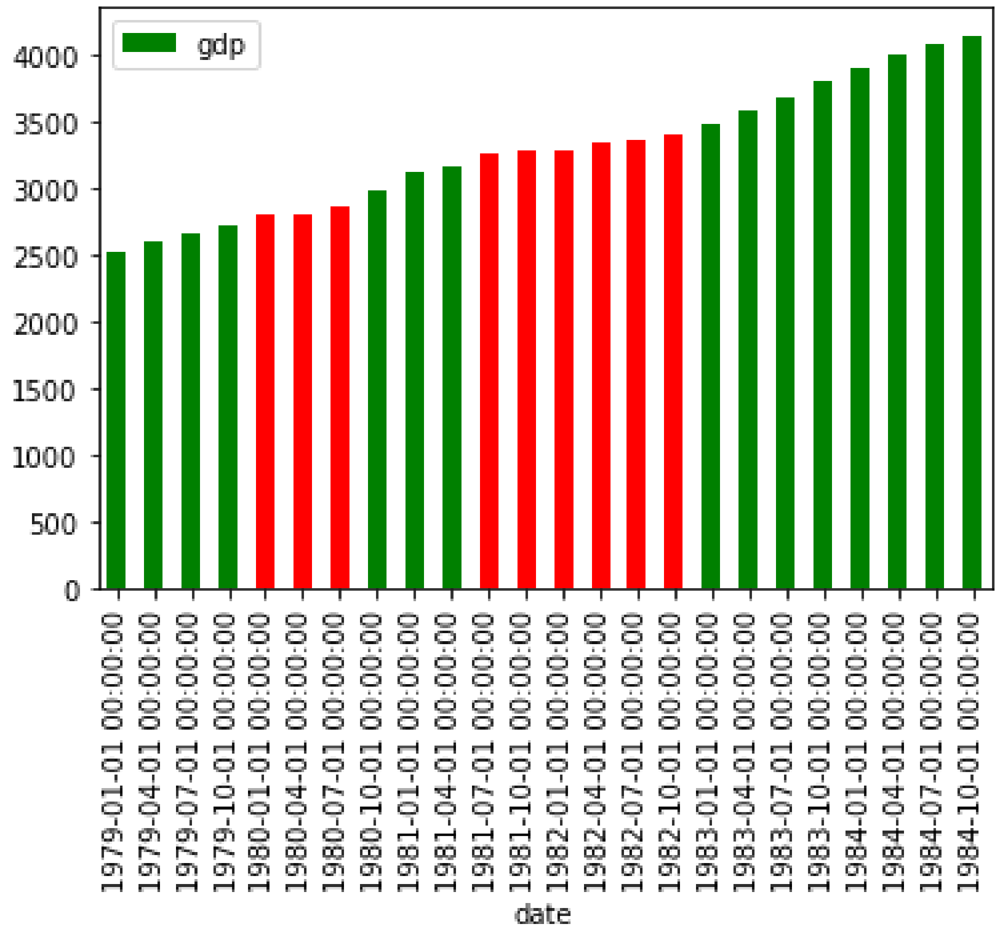
[68]:
```
gdp = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zOOZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1788266164&single=true&output=csv')
gdp["date"] = pd.to_datetime(gdp["date"])

recession = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zOOZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=978287109&single=true&output=csv')
recession["date"] = pd.to_datetime(recession["date"])

print(gdp.head())
print(recession.head())
```

```
        date       gdp
```

```
0 1979-01-01  2526.610
1 1979-04-01  2591.247
2 1979-07-01  2667.565
3 1979-10-01  2723.883
4 1980-01-01  2789.842
        date econ_status
0 1980-01-01   recession
1 1980-08-01      normal
2 1981-07-01   recession
3 1982-12-01      normal
4 1990-07-01   recession
```

[69]:

```python
# Merge gdp and recession on date using merge_asof()
gdp_recession = pd.merge_asof(gdp, recession, on = "date")

# Create a list based on the row value of gdp_recession['econ_status']
is_recession = ['r' if s=='recession' else 'g' for s in␣
 ↪gdp_recession['econ_status']]

# Plot a bar chart of gdp_recession
gdp_recession.plot(kind="bar", y="gdp", x="date", color=is_recession, rot=90)
plt.show()
```



2.4.2 Selección de datos con query()

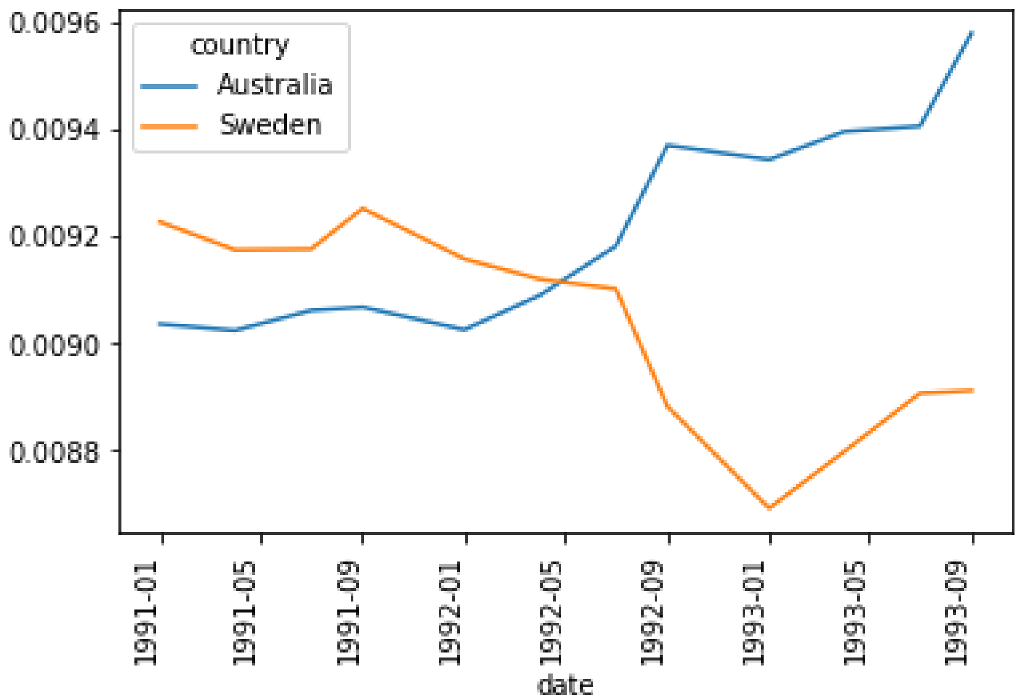
Su sintaxis es .query("CONDICIÓN DE SELECCIÓN")

[70]:
```
gdp = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1348453958&single=true&output=csv')
gdp["date"] = pd.to_datetime(gdp["date"])

pop = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1727314797&single=true&output=csv')
pop["date"] = pd.to_datetime(pop["date"])
```

[71]:
```
# Merge gdp and pop on date and country with fill
gdp_pop = pd.merge_ordered(gdp, pop, on=['country','date'], fill_method='ffill')

# Add a column named gdp_per_capita to gdp_pop that divides the gdp by pop
```

```python
gdp_pop['gdp_per_capita'] = gdp_pop['gdp'] / gdp_pop['pop']
print(gdp_pop.head())

# Pivot data so gdp_per_capita, where index is date and columns is country
gdp_pivot = gdp_pop.pivot_table('gdp_per_capita', 'date', 'country')
print(gdp_pivot.head())

# Select dates equal to or greater than 1991-01-01
recent_gdp_pop = gdp_pivot.query('date >= "1991-01-01"')

# Plot recent_gdp_pop
recent_gdp_pop.plot(rot=90)
plt.show()
```

```
        date    country         gdp  series_code_x       pop series_code_y  \
0 1990-01-01  Australia  158051.132  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
1 1990-04-01  Australia  158263.582  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
2 1990-07-01  Australia  157329.279  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
3 1990-09-01  Australia  158240.678  NYGDPMKTPSAKD  17065100   SP.POP.TOTL
4 1991-01-01  Australia  156195.954  NYGDPMKTPSAKD  17284000   SP.POP.TOTL

   gdp_per_capita
0        0.009262
1        0.009274
2        0.009219
3        0.009273
4        0.009037
country     Australia    Sweden
date
1990-01-01   0.009262  0.009328
1990-04-01   0.009274  0.009415
1990-07-01   0.009219  0.009344
1990-09-01   0.009273  0.009360
1991-01-01   0.009037  0.009228
```



2.4.3 Método melt()

Como sabemos, este método sirve para convertir un dataframe de formato wide a long. Mientras que un formato wide es más legible, a veces el formato long es más fácil para trabajar y leer computacionalmente.

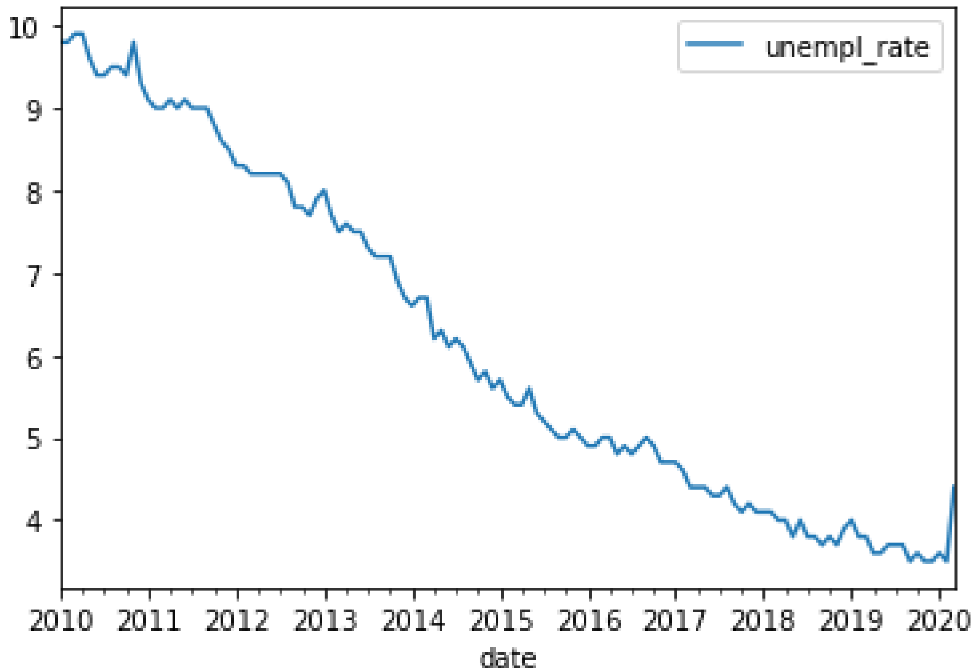
[72]:
```
ur_wide = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
 ↪2PACX-1vSFa60oGruMG6TlJT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
 ↪pub?gid=854767693&single=true&output=csv')
print(ur_wide.head())
```

```
   year  jan  feb  mar  apr  may  jun  jul  aug  sep  oct  nov  dec
0  2010  9.8  9.8  9.9  9.9  9.6  9.4  9.4  9.5  9.5  9.4  9.8  9.3
1  2011  9.1  9.0  9.0  9.1  9.0  9.1  9.0  9.0  9.0  8.8  8.6  8.5
2  2012  8.3  8.3  8.2  8.2  8.2  8.2  8.2  8.1  7.8  7.8  7.7  7.9
3  2013  8.0  7.7  7.5  7.6  7.5  7.5  7.3  7.2  7.2  7.2  6.9  6.7
4  2014  6.6  6.7  6.7  6.2  6.3  6.1  6.2  6.1  5.9  5.7  5.8  5.6
```

[73]:
```
# unpivot everything besides the year column
ur_tall = ur_wide.melt(id_vars = ["year"], var_name = "month", value_name =␣
 ↪"unempl_rate")
print(ur_tall.head(15))
ur_tall = ur_tall[ur_tall['unempl_rate'] != "nan"]
ur_tall = ur_tall.dropna()
```

```python
# Create a date column using the month and year columns of ur_tall
ur_tall = ur_tall.astype(str)
ur_tall['date'] = pd.to_datetime(ur_tall['month'] + '-' + ur_tall["year"])
print(ur_tall.head(15))

# Sort ur_tall by date in ascending order
ur_sorted = ur_tall.sort_values("date")

# Plot the unempl_rate by date
ur_sorted['unempl_rate'] = pd.to_numeric(ur_sorted['unempl_rate'])
ur_sorted.plot(x = "date", y = "unempl_rate")
plt.show()
```

```
    year month  unempl_rate
0   2010   jan          9.8
1   2011   jan          9.1
2   2012   jan          8.3
3   2013   jan          8.0
4   2014   jan          6.6
5   2015   jan          5.7
6   2016   jan          4.9
7   2017   jan          4.7
8   2018   jan          4.1
9   2019   jan          4.0
10  2020   jan          3.6
11  2010   feb          9.8
12  2011   feb          9.0
13  2012   feb          8.3
14  2013   feb          7.7
    year month unempl_rate       date
0   2010   jan         9.8 2010-01-01
1   2011   jan         9.1 2011-01-01
2   2012   jan         8.3 2012-01-01
3   2013   jan         8.0 2013-01-01
4   2014   jan         6.6 2014-01-01
5   2015   jan         5.7 2015-01-01
6   2016   jan         4.9 2016-01-01
7   2017   jan         4.7 2017-01-01
8   2018   jan         4.1 2018-01-01
9   2019   jan         4.0 2019-01-01
10  2020   jan         3.6 2020-01-01
11  2010   feb         9.8 2010-02-01
12  2011   feb         9.0 2011-02-01
13  2012   feb         8.3 2012-02-01
14  2013   feb         7.7 2013-02-01
```



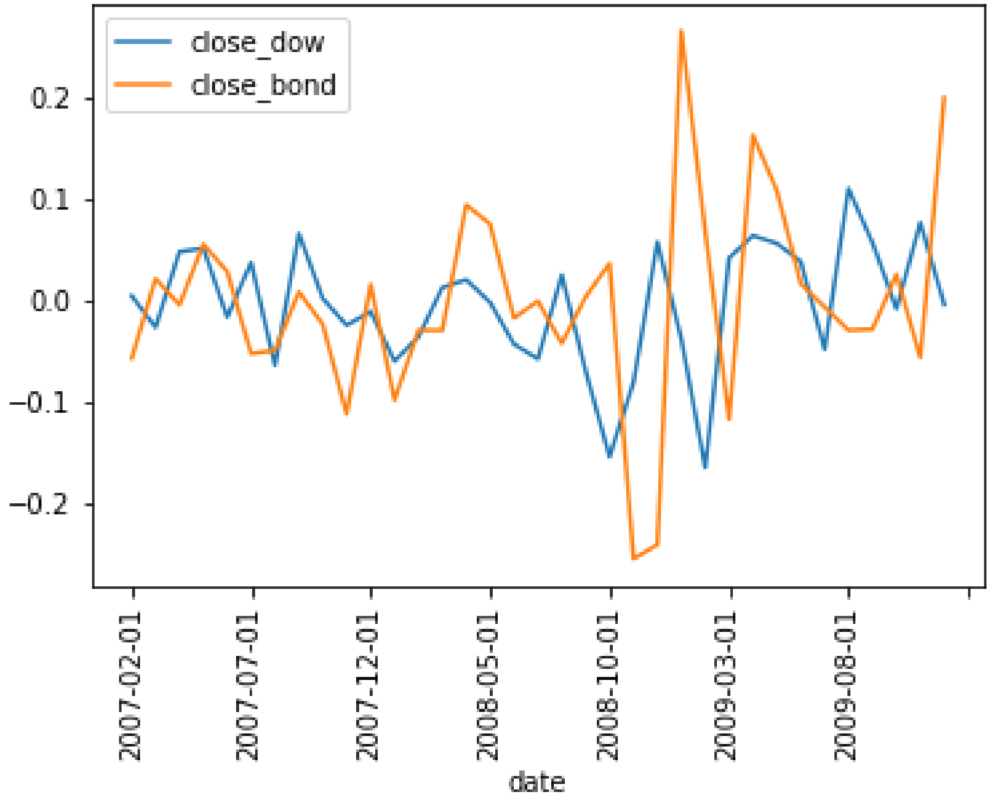
[74]:
```
ten_yr = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6T1JT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=1113683844&single=true&output=csv')
print(ten_yr.head())
dji = pd.read_csv('https://docs.google.com/spreadsheets/d/e/
↪2PACX-1vSFa6OoGruMG6T1JT9Hn5UYuABJ-o6nvwbXU5EkSSOHq0r2zO0ZP84KFfOZcdKIzeSuNwF-6MsgGzX_/
↪pub?gid=648434883&single=true&output=csv')
print(dji.head())
```

```
  metric  2007-02-01  2007-03-01  2007-04-01  2007-05-01  2007-06-01  \
0   open       0.033      -0.060       0.025      -0.004       0.061
1   high      -0.007      -0.041       0.022       0.031       0.080
2    low      -0.016      -0.008       0.031      -0.002       0.059
3  close      -0.057       0.022      -0.004       0.056       0.029

   2007-07-01  2007-08-01  2007-09-01  2007-10-01  …  2009-03-01  \
0       0.027      -0.059      -0.046       0.014  …       0.046
1      -0.022      -0.060      -0.038       0.004  …      -0.004
2      -0.027      -0.052      -0.043       0.003  …      -0.062
3      -0.052      -0.049       0.009      -0.023  …      -0.117

   2009-04-01  2009-05-01  2009-06-01  2009-07-01  2009-08-01  2009-09-01  \
0      -0.103       0.191       0.107       0.024      -0.007      -0.047
1       0.041       0.187       0.068      -0.062       0.032      -0.090
2       0.069       0.168       0.123      -0.055       0.040      -0.036
```

```
3       0.164       0.109       0.017      -0.006      -0.029      -0.028

   2009-10-01  2009-11-01  2009-12-01
0      -0.032       0.034      -0.051
1       0.012      -0.004       0.099
2      -0.051       0.030       0.007
3       0.026      -0.056       0.201

[4 rows x 36 columns]
        date     close
0  2007-02-01  0.005094
1  2007-03-01 -0.026140
2  2007-04-01  0.048530
3  2007-05-01  0.052010
4  2007-06-01 -0.016070
```

[75]:
```
# Use melt on ten_yr, unpivot everything besides the metric column
bond_perc = ten_yr.melt(id_vars = "metric", var_name = "date", value_name =␣
↪"close")
print(bond_perc.head())

# Use query on bond_perc to select only the rows where metric=close
bond_perc_close = bond_perc.query('metric == "close"')
print(bond_perc_close.head())

# Merge (ordered) dji and bond_perc_close on date with an inner join
dow_bond = pd.merge_ordered(dji, bond_perc_close, on = "date", suffixes =␣
↪("_dow", "_bond"), how = "inner")


# Plot only the close_dow and close_bond columns
dow_bond.plot(y = ["close_dow", "close_bond"], x='date', rot=90)
plt.show()
```

```
  metric        date  close
0   open  2007-02-01  0.033
1   high  2007-02-01 -0.007
2    low  2007-02-01 -0.016
3  close  2007-02-01 -0.057
4   open  2007-03-01 -0.060
   metric        date  close
3   close  2007-02-01 -0.057
7   close  2007-03-01  0.022
11  close  2007-04-01 -0.004
15  close  2007-05-01  0.056
19  close  2007-06-01  0.029
```



3 ANÁLISIS DE POLÍTICAS CON PANDAS

3.1 PREPARACIÓN DE DATOS

[76]:
```
import pandas as pd

ri = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/police.csv")

print(ri.head())

### 1. LOCALIZAR NAs

print(ri.isna().sum(axis = 0))
print(ri.shape) # county_name solo contiene NAs

### 2. Eliminar columna

ri.drop(["county_name", "state"], axis = "columns", inplace = True)
print(ri.shape)

# Count the number of missing values in each column
```

```python
print(ri.isnull().sum())

# Drop all rows that are missing 'driver_gender'
ri.dropna(subset=['driver_gender'], inplace=True)

# Count the number of missing values in each column (again)
print(ri.isnull().sum())

# Examine the shape of the DataFrame
print(ri.shape)
```

```
  state   stop_date stop_time  county_name driver_gender driver_race  \
0    RI  2005-01-04     12:55          NaN             M       White
1    RI  2005-01-23     23:15          NaN             M       White
2    RI  2005-02-17     04:15          NaN             M       White
3    RI  2005-02-20     17:15          NaN             M       White
4    RI  2005-02-24     01:20          NaN             F       White

                    violation_raw  violation  search_conducted search_type  \
0  Equipment/Inspection Violation  Equipment             False         NaN
1                        Speeding   Speeding             False         NaN
2                        Speeding   Speeding             False         NaN
3                Call for Service      Other             False         NaN
4                        Speeding   Speeding             False         NaN

    stop_outcome is_arrested stop_duration  drugs_related_stop district
0       Citation       False      0-15 Min               False  Zone X4
1       Citation       False      0-15 Min               False  Zone K3
2       Citation       False      0-15 Min               False  Zone X4
3  Arrest Driver        True     16-30 Min               False  Zone X1
4       Citation       False      0-15 Min               False  Zone X3
state                     0
stop_date                 0
stop_time                 0
county_name           91741
driver_gender          5205
driver_race            5202
violation_raw          5202
violation              5202
search_conducted          0
search_type           88434
stop_outcome           5202
is_arrested            5202
stop_duration          5202
drugs_related_stop        0
district                  0
dtype: int64
(91741, 15)
```

```
(91741, 13)
stop_date                 0
stop_time                 0
driver_gender          5205
driver_race            5202
violation_raw          5202
violation              5202
search_conducted          0
search_type           88434
stop_outcome           5202
is_arrested            5202
stop_duration          5202
drugs_related_stop        0
district                  0
dtype: int64
stop_date                 0
stop_time                 0
driver_gender             0
driver_race               0
violation_raw             0
violation                 0
search_conducted          0
search_type           83229
stop_outcome              0
is_arrested               0
stop_duration             0
drugs_related_stop        0
district                  0
dtype: int64
(86536, 13)
```

[77]:

```
### 3. COMPROBAR LOS TIPOS DE DATOS

print(ri.dtypes)

# Examine the head of the 'is_arrested' column
print(ri.is_arrested.head())

# Change the data type of 'is_arrested' to 'bool'
ri['is_arrested'] = ri.is_arrested.astype(bool)

# Check the data type of 'is_arrested'
print(ri.is_arrested.dtype)
```

```
stop_date             object
stop_time             object
driver_gender         object
driver_race           object
```

```
violation_raw         object
violation             object
search_conducted        bool
search_type           object
stop_outcome          object
is_arrested           object
stop_duration         object
drugs_related_stop      bool
district              object
dtype: object
0    False
1    False
2    False
3     True
4    False
Name: is_arrested, dtype: object
bool
```

[78]:
```
### 4. CREANDO UN DATETIMEINDEX

# Usaremos stop_date y stop_time combinándolos y convirtiéndolos a formato␣
↪datetime

print(ri.head())

# Concatenate 'stop_date' and 'stop_time' (separated by a space)
combined = ri.stop_date.str.cat(ri.stop_time, sep = " ")

# Convert 'combined' to datetime format
ri['stop_datetime'] = pd.to_datetime(combined)

# Examine the data types of the DataFrame
print(ri.head())
print(ri.dtypes)
```

```
    stop_date stop_time driver_gender driver_race  \
0  2005-01-04     12:55             M       White
1  2005-01-23     23:15             M       White
2  2005-02-17     04:15             M       White
3  2005-02-20     17:15             M       White
4  2005-02-24     01:20             F       White

                     violation_raw  violation  search_conducted search_type  \
0  Equipment/Inspection Violation  Equipment             False         NaN
1                        Speeding   Speeding             False         NaN
2                        Speeding   Speeding             False         NaN
3                Call for Service      Other             False         NaN
4                        Speeding   Speeding             False         NaN
```

```
    stop_outcome  is_arrested stop_duration  drugs_related_stop district
0       Citation        False      0-15 Min               False  Zone X4
1       Citation        False      0-15 Min               False  Zone K3
2       Citation        False      0-15 Min               False  Zone X4
3  Arrest Driver         True     16-30 Min               False  Zone X1
4       Citation        False      0-15 Min               False  Zone X3
    stop_date stop_time driver_gender driver_race  \
0  2005-01-04     12:55             M       White
1  2005-01-23     23:15             M       White
2  2005-02-17     04:15             M       White
3  2005-02-20     17:15             M       White
4  2005-02-24     01:20             F       White

                      violation_raw  violation  search_conducted search_type  \
0  Equipment/Inspection Violation  Equipment             False         NaN
1                        Speeding   Speeding             False         NaN
2                        Speeding   Speeding             False         NaN
3                Call for Service      Other             False         NaN
4                        Speeding   Speeding             False         NaN

    stop_outcome  is_arrested stop_duration  drugs_related_stop district  \
0       Citation        False      0-15 Min               False  Zone X4
1       Citation        False      0-15 Min               False  Zone K3
2       Citation        False      0-15 Min               False  Zone X4
3  Arrest Driver         True     16-30 Min               False  Zone X1
4       Citation        False      0-15 Min               False  Zone X3

        stop_datetime
0 2005-01-04 12:55:00
1 2005-01-23 23:15:00
2 2005-02-17 04:15:00
3 2005-02-20 17:15:00
4 2005-02-24 01:20:00
stop_date                     object
stop_time                     object
driver_gender                 object
driver_race                   object
violation_raw                 object
violation                     object
search_conducted                bool
search_type                   object
stop_outcome                  object
is_arrested                     bool
stop_duration                 object
drugs_related_stop              bool
district                      object
stop_datetime         datetime64[ns]
```

```
dtype: object
```

[79]:
```
# Para crear el índice:

# Set 'stop_datetime' as the index
ri.set_index("stop_datetime", inplace=True)

# Examine the index
print(ri.index)

# Examine the columns
print(ri.columns)
```

```
DatetimeIndex(['2005-01-04 12:55:00', '2005-01-23 23:15:00',
               '2005-02-17 04:15:00', '2005-02-20 17:15:00',
               '2005-02-24 01:20:00', '2005-03-14 10:00:00',
               '2005-03-29 21:55:00', '2005-04-04 21:25:00',
               '2005-07-14 11:20:00', '2005-07-14 19:55:00',
               …
               '2015-12-31 13:23:00', '2015-12-31 18:59:00',
               '2015-12-31 19:13:00', '2015-12-31 20:20:00',
               '2015-12-31 20:50:00', '2015-12-31 21:21:00',
               '2015-12-31 21:59:00', '2015-12-31 22:04:00',
               '2015-12-31 22:09:00', '2015-12-31 22:47:00'],
              dtype='datetime64[ns]', name='stop_datetime', length=86536,
freq=None)
Index(['stop_date', 'stop_time', 'driver_gender', 'driver_race',
       'violation_raw', 'violation', 'search_conducted', 'search_type',
       'stop_outcome', 'is_arrested', 'stop_duration', 'drugs_related_stop',
       'district'],
      dtype='object')
```

3.2 RELACIÓN ENTRE GÉNERO Y POLÍTICAS

[80]:
```
### 5. GÉNERO VS. DELITOS DE TRÁFICO

print(ri.stop_outcome.value_counts())

print(ri.stop_outcome.value_counts(normalize = True)) # %

print(ri.driver_race.value_counts())

# Para comparar las violaciones de un solo grupo racial:

asian = ri[ri.driver_race == "Asian"]
asian.stop_outcome.value_counts(normalize = True)
```

```
Citation            77091
Warning              5136
```

```
Arrest Driver        2735
No Action             624
N/D                   607
Arrest Passenger      343
Name: stop_outcome, dtype: int64
Citation            0.890855
Warning             0.059351
Arrest Driver       0.031605
No Action           0.007211
N/D                 0.007014
Arrest Passenger    0.003964
Name: stop_outcome, dtype: float64
White       61870
Black       12285
Hispanic     9727
Asian        2389
Other         265
Name: driver_race, dtype: int64
```

[80]:
```
Citation            0.922980
Warning             0.045207
Arrest Driver       0.017581
No Action           0.008372
N/D                 0.004186
Arrest Passenger    0.001674
Name: stop_outcome, dtype: float64
```

[81]:
```
# Count the unique values in 'violation'
print(ri.violation.value_counts())

# Express the counts as proportions
print(ri.violation.value_counts(normalize = True))

# Create a DataFrame of female drivers
female = ri[ri.driver_gender == "F"]

# Create a DataFrame of male drivers
male = ri[ri.driver_gender == "M"]

# Compute the violations by female drivers (as proportions)
print(female.violation.value_counts(normalize = True))

# Compute the violations by male drivers (as proportions)
print(male.violation.value_counts(normalize = True))
```

```
Speeding               48423
Moving violation       16224
Equipment              10921
```

```
Other                    4409
Registration/plates      3703
Seat belt                2856
Name: violation, dtype: int64
Speeding               0.559571
Moving violation       0.187483
Equipment              0.126202
Other                  0.050950
Registration/plates    0.042791
Seat belt              0.033004
Name: violation, dtype: float64
Speeding               0.658114
Moving violation       0.138218
Equipment              0.105199
Registration/plates    0.044418
Other                  0.029738
Seat belt              0.024312
Name: violation, dtype: float64
Speeding               0.522243
Moving violation       0.206144
Equipment              0.134158
Other                  0.058985
Registration/plates    0.042175
Seat belt              0.036296
Name: violation, dtype: float64
```

[82]:
```
# ¿El género influye en quién recibe una multa?

female_and_speeding = ri[(ri.driver_gender == "F") & (ri.violation ==␣
 ↪"Speeding")]

male_and_speeding = ri[(ri.driver_gender == "M") & (ri.violation == "Speeding")]

print(female_and_speeding.stop_outcome.value_counts(normalize=True))

print(male_and_speeding.stop_outcome.value_counts(normalize=True))
```

```
Citation            0.952192
Warning             0.040074
Arrest Driver       0.005752
N/D                 0.000959
Arrest Passenger    0.000639
No Action           0.000383
Name: stop_outcome, dtype: float64
Citation            0.944595
Warning             0.036184
Arrest Driver       0.015895
Arrest Passenger    0.001281
```

```
No Action            0.001068
N/D                  0.000976
Name: stop_outcome, dtype: float64
```

[83]:

```python
# ¿El género afecta a los vehículos registrados?

import numpy as np

# El porcentaje de paradas que resultan en un arresto es:

print(ri.is_arrested.value_counts(normalize = True))

# Alternativamente:

print(ri.is_arrested.mean()) # que arroja el porcentaje de Trues

# Para analizar la tasa de arresto por distrito:

print(ri.district.unique())

ri[ri.district == "Zone K1"].is_arrested.mean()

# Y para todos:

print(ri.groupby("district").is_arrested.mean())

# Y por género:

print(ri.groupby(["district", "driver_gender"]).is_arrested.mean())
print(ri.groupby(["driver_gender", "district"]).is_arrested.mean())
```

```
False    0.964431
True     0.035569
Name: is_arrested, dtype: float64
0.0355690117407784
['Zone X4' 'Zone K3' 'Zone X1' 'Zone X3' 'Zone K1' 'Zone K2']
district
Zone K1    0.024349
Zone K2    0.030801
Zone K3    0.032311
Zone X1    0.023494
Zone X3    0.034871
Zone X4    0.048038
Name: is_arrested, dtype: float64
district  driver_gender
Zone K1   F                0.019169
          M                0.026588
Zone K2   F                0.022196
```

```
         M               0.034285
Zone K3  F               0.025156
         M               0.034961
Zone X1  F               0.019646
         M               0.024563
Zone X3  F               0.027188
         M               0.038166
Zone X4  F               0.042149
         M               0.049956
Name: is_arrested, dtype: float64
driver_gender  district
F              Zone K1     0.019169
               Zone K2     0.022196
               Zone K3     0.025156
               Zone X1     0.019646
               Zone X3     0.027188
               Zone X4     0.042149
M              Zone K1     0.026588
               Zone K2     0.034285
               Zone K3     0.034961
               Zone X1     0.024563
               Zone X3     0.038166
               Zone X4     0.049956
Name: is_arrested, dtype: float64
```

[84]:

```
# Check the data type of 'search_conducted'
print(ri.search_conducted.dtype)

# Calculate the search rate by counting the values
print(ri.search_conducted.value_counts(normalize = True))

# Calculate the search rate by taking the mean
print(ri.search_conducted.mean())
```

```
bool
False    0.961785
True     0.038215
Name: search_conducted, dtype: float64
0.0382153092354627
```

[85]:

```
# Calculate the search rate for female drivers
print(ri[ri.driver_gender == "F"].search_conducted.mean())

# Calculate the search rate for male drivers
print(ri[ri.driver_gender == "M"].search_conducted.mean())

# Calculate the search rate for both groups simultaneously
print(ri.groupby("driver_gender").search_conducted.mean())
```

```
0.019180617481282074
0.04542557598546892
driver_gender
F    0.019181
M    0.045426
Name: search_conducted, dtype: float64
```

[86]:

```python
# Calculate the search rate for each combination of gender and violation
print(ri.groupby(["driver_gender", "violation"]).search_conducted.mean())

# Reverse the ordering to group by violation before gender
print(ri.groupby(["violation", "driver_gender"]).search_conducted.mean())
```

```
driver_gender  violation
F              Equipment              0.039984
               Moving violation       0.039257
               Other                  0.041018
               Registration/plates    0.054924
               Seat belt              0.017301
               Speeding               0.008309
M              Equipment              0.071496
               Moving violation       0.061524
               Other                  0.046191
               Registration/plates    0.108802
               Seat belt              0.035119
               Speeding               0.027885
Name: search_conducted, dtype: float64
violation            driver_gender
Equipment            F                0.039984
                     M                0.071496
Moving violation     F                0.039257
                     M                0.061524
Other                F                0.041018
                     M                0.046191
Registration/plates  F                0.054924
                     M                0.108802
Seat belt            F                0.017301
                     M                0.035119
Speeding             F                0.008309
                     M                0.027885
Name: search_conducted, dtype: float64
```

[87]:

```python
# ¿El género influye en quién es cacheado?

print(ri.search_type.value_counts())

# Para comprobar si una cadena está presenta en cada elemento de una columna␣
↪dada:
```

```python
ri["inventory"] = ri.search_type.str.contains("Inventory", na = False)

ri.inventory.sum()
```

```
Incident to Arrest                                          1290
Probable Cause                                               924
Inventory                                                    219
Reasonable Suspicion                                         214
Protective Frisk                                             164
Incident to Arrest,Inventory                                 123
Incident to Arrest,Probable Cause                            100
Probable Cause,Reasonable Suspicion                           54
Incident to Arrest,Inventory,Probable Cause                   35
Probable Cause,Protective Frisk                               35
Incident to Arrest,Protective Frisk                           33
Inventory,Probable Cause                                      25
Protective Frisk,Reasonable Suspicion                         19
Incident to Arrest,Inventory,Protective Frisk                 18
Incident to Arrest,Probable Cause,Protective Frisk            13
Inventory,Protective Frisk                                    12
Incident to Arrest,Reasonable Suspicion                        8
Incident to Arrest,Probable Cause,Reasonable Suspicion         5
Probable Cause,Protective Frisk,Reasonable Suspicion           5
Incident to Arrest,Inventory,Reasonable Suspicion              4
Incident to Arrest,Protective Frisk,Reasonable Suspicion       2
Inventory,Reasonable Suspicion                                 2
Inventory,Protective Frisk,Reasonable Suspicion                1
Inventory,Probable Cause,Protective Frisk                      1
Inventory,Probable Cause,Reasonable Suspicion                  1
Name: search_type, dtype: int64
```

[87]: 441

[88]:
```python
# Count the 'search_type' values
print(ri.search_type.value_counts())

# Check if 'search_type' contains the string 'Protective Frisk'
ri['frisk'] = ri.search_type.str.contains('Protective Frisk', na=False)

# Check the data type of 'frisk'
print(ri.frisk.dtype)

# Take the sum of 'frisk'
print(ri.frisk.sum())
```

```
Incident to Arrest                                          1290
Probable Cause                                               924
```

```
Inventory                                                    219
Reasonable Suspicion                                         214
Protective Frisk                                             164
Incident to Arrest,Inventory                                 123
Incident to Arrest,Probable Cause                            100
Probable Cause,Reasonable Suspicion                           54
Incident to Arrest,Inventory,Probable Cause                   35
Probable Cause,Protective Frisk                               35
Incident to Arrest,Protective Frisk                           33
Inventory,Probable Cause                                      25
Protective Frisk,Reasonable Suspicion                         19
Incident to Arrest,Inventory,Protective Frisk                 18
Incident to Arrest,Probable Cause,Protective Frisk            13
Inventory,Protective Frisk                                    12
Incident to Arrest,Reasonable Suspicion                        8
Incident to Arrest,Probable Cause,Reasonable Suspicion         5
Probable Cause,Protective Frisk,Reasonable Suspicion           5
Incident to Arrest,Inventory,Reasonable Suspicion              4
Incident to Arrest,Protective Frisk,Reasonable Suspicion       2
Inventory,Reasonable Suspicion                                 2
Inventory,Protective Frisk,Reasonable Suspicion                1
Inventory,Probable Cause,Protective Frisk                      1
Inventory,Probable Cause,Reasonable Suspicion                  1
Name: search_type, dtype: int64
bool
303
```

[89]:
```
# Create a DataFrame of stops in which a search was conducted
searched = ri[ri.search_conducted == True]

# Calculate the overall frisk rate by taking the mean of 'frisk'
print(searched.frisk.mean())

# Calculate the frisk rate for each gender
print(searched.groupby("driver_gender").frisk.mean())
```

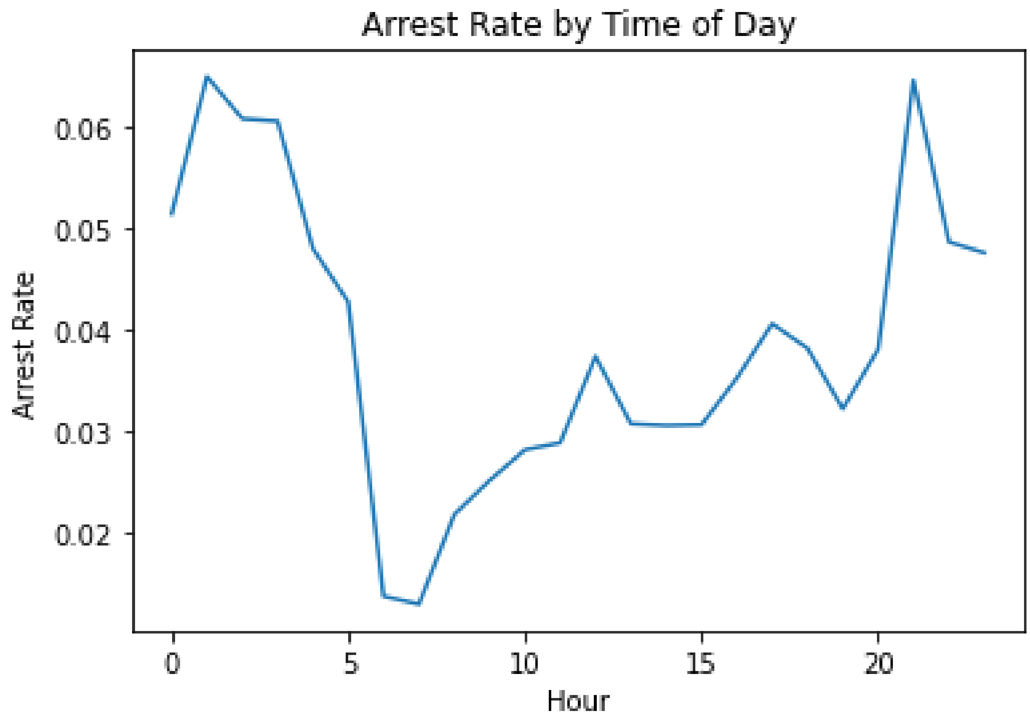
```
0.09162382824312065
driver_gender
F    0.074561
M    0.094353
Name: frisk, dtype: float64
```

3.3 ANÁLISIS EXPLORATORIO

[90]:
```
### 6. DELITOS Y TEMPORALIDAD

import matplotlib.pyplot as plt
```

```python
# Calculate the overall arrest rate
print(ri.is_arrested.mean())

# Calculate the hourly arrest rate
print(ri.groupby(ri.index.hour).is_arrested.mean())

# Save the hourly arrest rate
hourly_arrest_rate = ri.groupby(ri.index.hour).is_arrested.mean()

# Create a line plot of 'hourly_arrest_rate'
hourly_arrest_rate.plot()

# Add the xlabel, ylabel, and title
plt.xlabel("Hour")
plt.ylabel("Arrest Rate")
plt.title("Arrest Rate by Time of Day")

# Display the plot
plt.show()
```

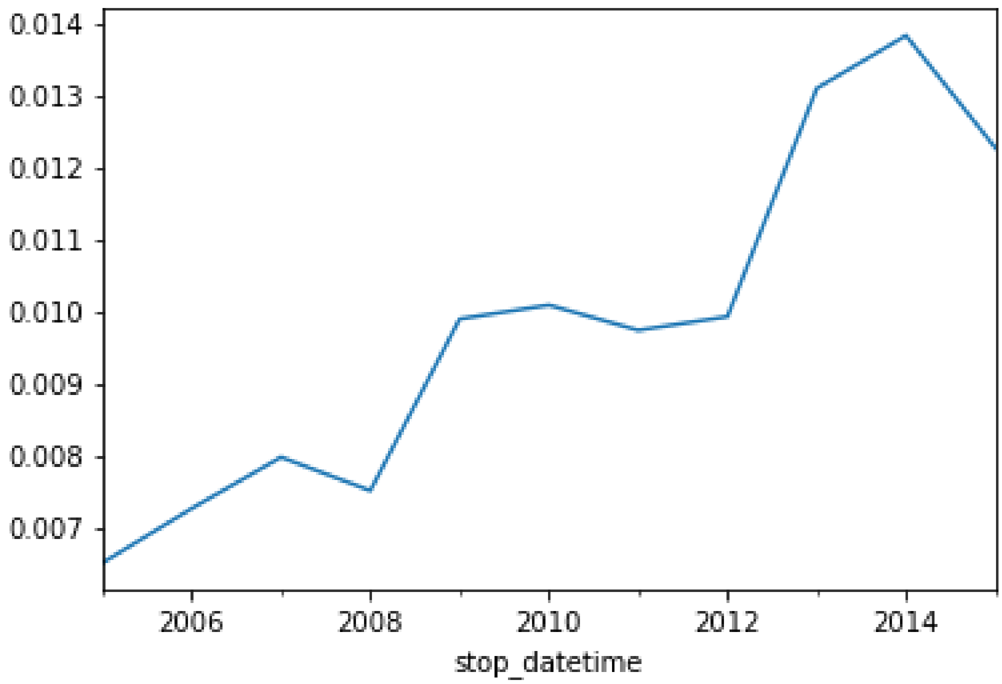
```
0.0355690117407784
stop_datetime
0     0.051431
1     0.064932
2     0.060798
3     0.060549
4     0.048000
5     0.042781
6     0.013813
7     0.013032
8     0.021854
9     0.025206
10    0.028213
11    0.028897
12    0.037399
13    0.030776
14    0.030605
15    0.030679
16    0.035281
17    0.040619
18    0.038204
19    0.032245
20    0.038107
21    0.064541
22    0.048666
23    0.047592
Name: is_arrested, dtype: float64
```



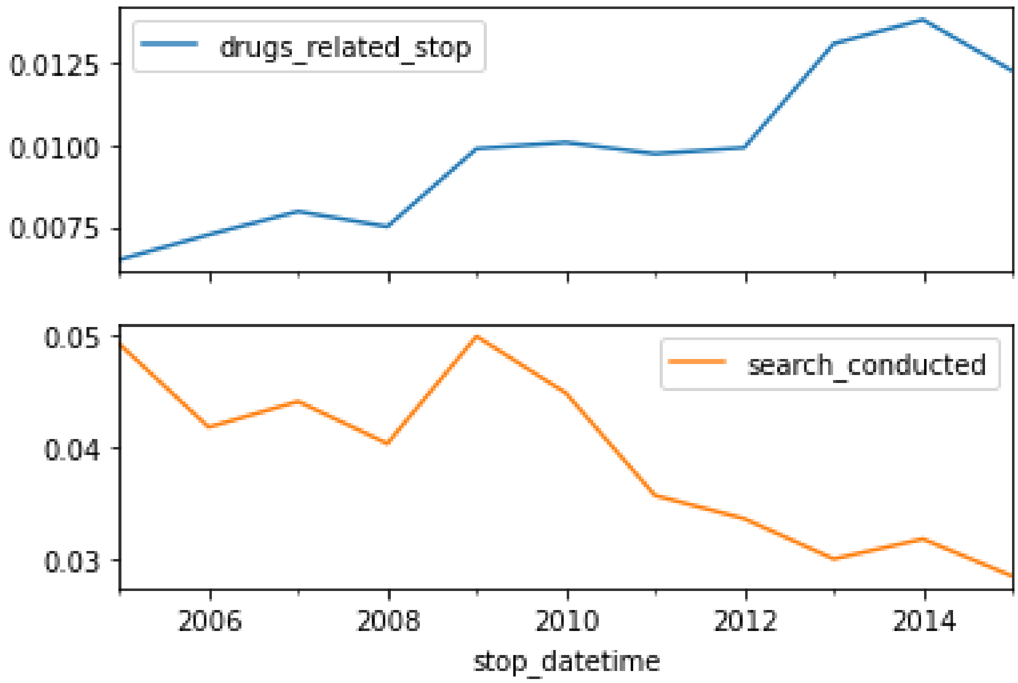
[91]:
```
# Delitos relacionados con drogas

# Calculate the annual rate of drug-related stops
print(ri.drugs_related_stop.resample('A').mean()) # resample("A") remuestrea al␣
↪último día del mes, y "A" al último día del año

# Save the annual rate of drug-related stops
annual_drug_rate = ri.drugs_related_stop.resample('A').mean()

# Create a line plot of 'annual_drug_rate'
annual_drug_rate.plot()

# Display the plot
plt.show()
```

```
stop_datetime
2005-12-31    0.006501
2006-12-31    0.007258
2007-12-31    0.007970
2008-12-31    0.007505
2009-12-31    0.009889
2010-12-31    0.010081
2011-12-31    0.009731
2012-12-31    0.009921
```

```
2013-12-31    0.013094
2014-12-31    0.013826
2015-12-31    0.012266
Freq: A-DEC, Name: drugs_related_stop, dtype: float64
```



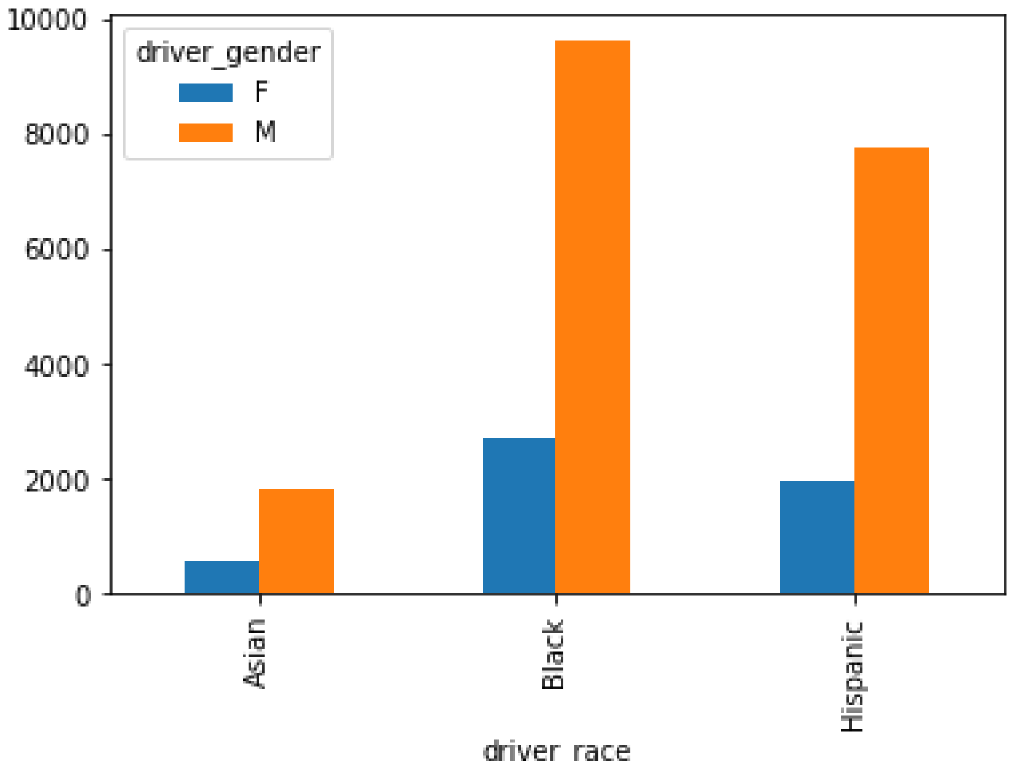
[92]:
```
# Calculate and save the annual search rate
annual_search_rate = ri.search_conducted.resample('A').mean()

# Concatenate 'annual_drug_rate' and 'annual_search_rate'
annual = pd.concat([annual_drug_rate,annual_search_rate], axis='columns')

# Create subplots from 'annual'
annual.plot(subplots=True)

# Display the subplots
plt.show()
```

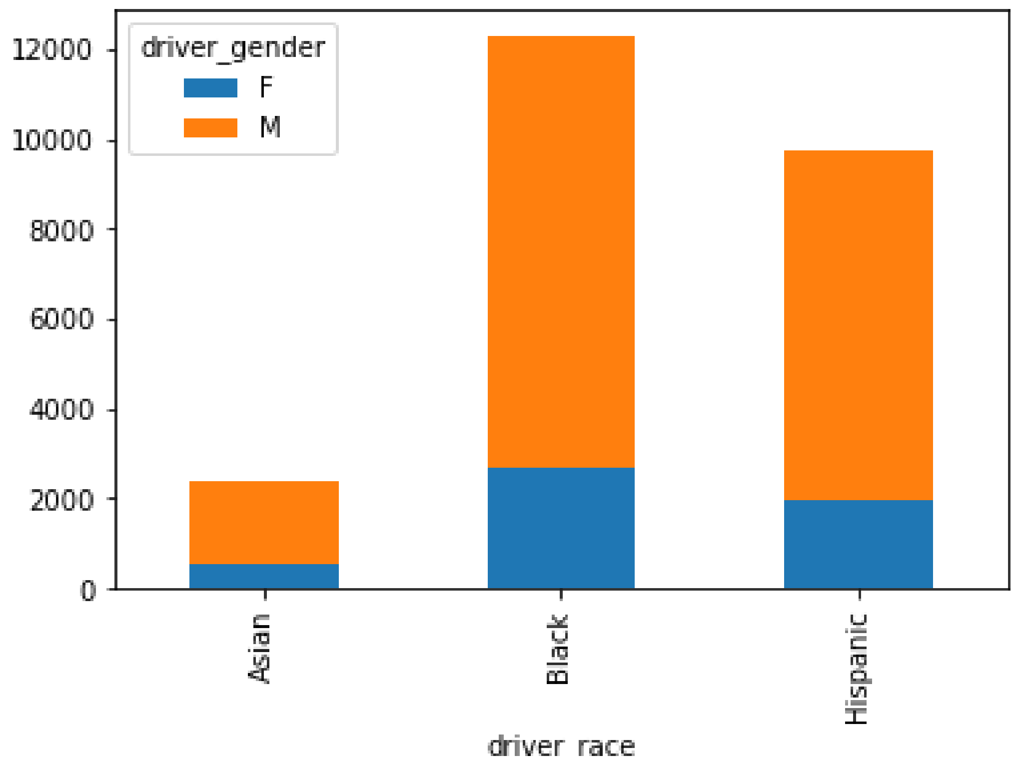


[93]:
```python
# ¿Qué violaciones son capturadas en cada distrito?

# Para computar una tabla de frecuencias:

table = pd.crosstab(ri.driver_race, ri.driver_gender)
print(table)

# Para seleccionar observaciones específicas:

print(table.loc["Asian": "Hispanic"])
table = table.loc["Asian": "Hispanic"]

table.plot(kind = "bar")
plt.show()

table.plot(kind = "bar", stacked = True)
plt.show()
```

```
driver_gender      F      M
driver_race
Asian            551   1838
Black           2681   9604
Hispanic        1953   7774
Other             53    212
White          18536  43334
```

```
driver_gender     F     M
driver_race
Asian           551  1838
Black          2681  9604
Hispanic       1953  7774
```



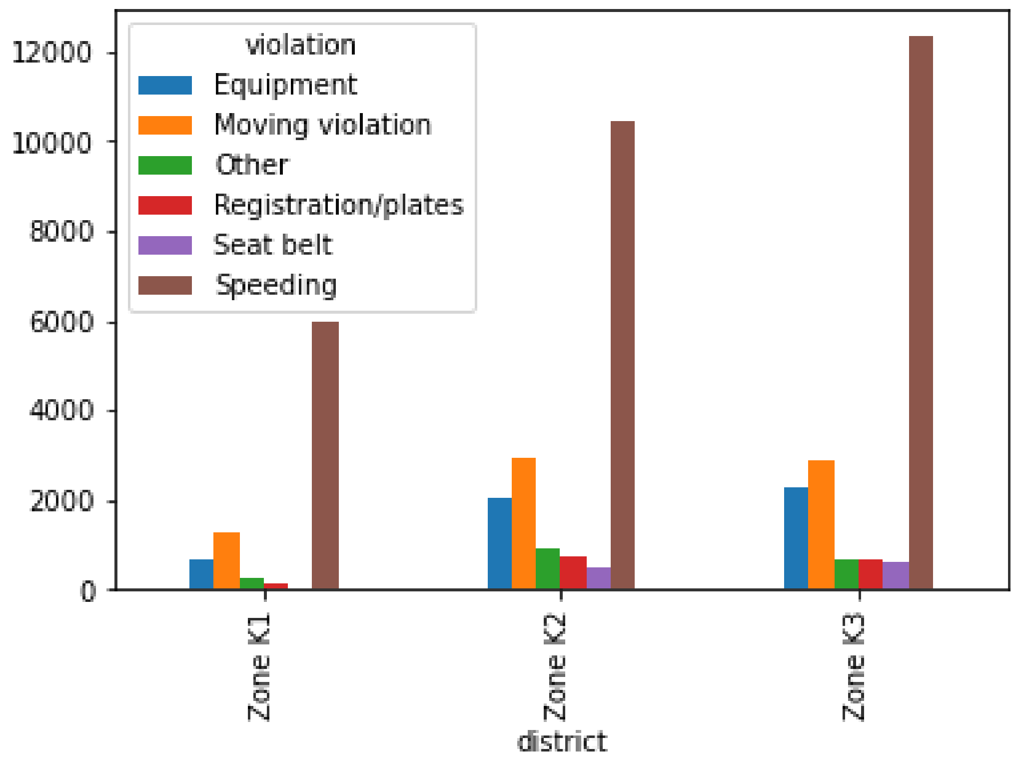


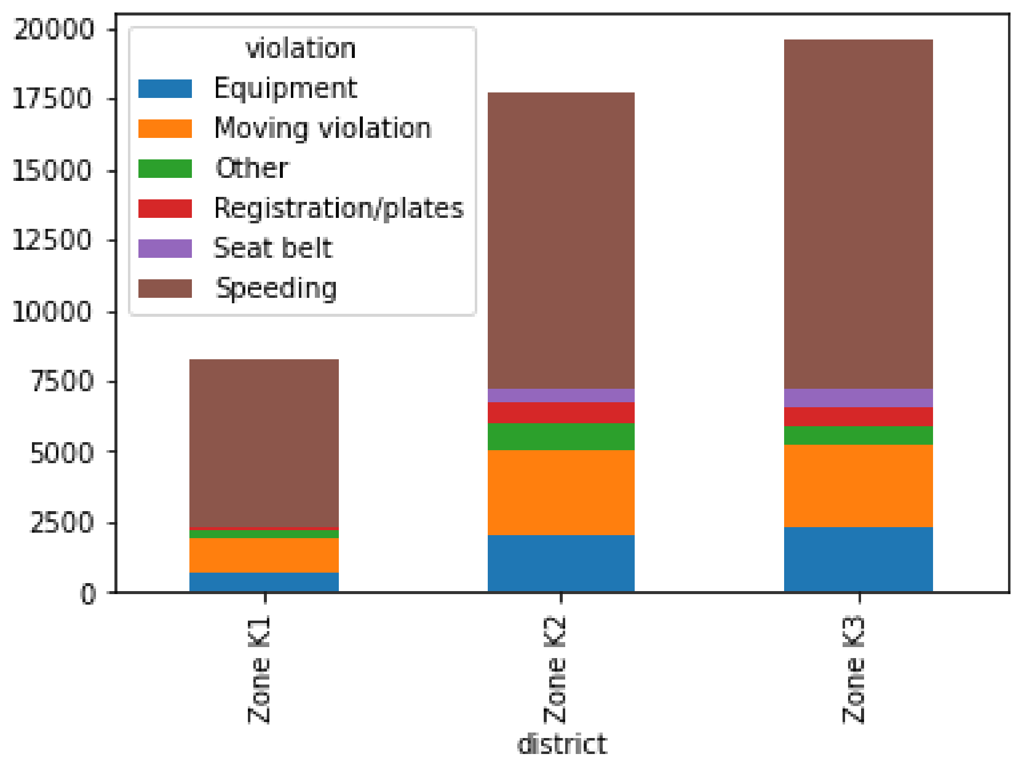
[94]:
```python
# Create a frequency table of districts and violations
print(pd.crosstab(ri.district, ri.violation))

# Save the frequency table as 'all_zones'
all_zones = pd.crosstab(ri.district, ri.violation)

# Select rows 'Zone K1' through 'Zone K3'
print(all_zones.loc["Zone K1": "Zone K3"])

# Save the smaller table as 'k_zones'
k_zones = all_zones.loc["Zone K1": "Zone K3"]
```

```
violation  Equipment  Moving violation  Other  Registration/plates  Seat belt  \
district
Zone K1          672              1254    290                  120          0
Zone K2         2061              2962    942                  768        481
Zone K3         2302              2898    705                  695        638
Zone X1          296               671    143                   38         74
Zone X3         2049              3086    769                  671        820
Zone X4         3541              5353   1560                 1411        843

violation  Speeding
district
```

```
Zone K1        5960
Zone K2       10448
Zone K3       12322
Zone X1        1119
Zone X3        8779
Zone X4        9795
violation  Equipment  Moving violation  Other  Registration/plates  Seat belt  \
district
Zone K1          672              1254    290                  120          0
Zone K2         2061              2962    942                  768        481
Zone K3         2302              2898    705                  695        638

violation  Speeding
district
Zone K1        5960
Zone K2       10448
Zone K3       12322
```

[95]:
```
# Create a bar plot of 'k_zones'
k_zones.plot(kind = "bar")

# Display the plot
plt.show()

# Create a stacked bar plot of 'k_zones'
k_zones.plot(kind = "bar", stacked = True)

# Display the plot
plt.show()
```


violation
Equipment
Moving violation
Other
Registration/plates
Seat belt
Speeding
12000
10000
8000
6000
4000
2000
0
Zone K1
Zone K2
Zone K3
district

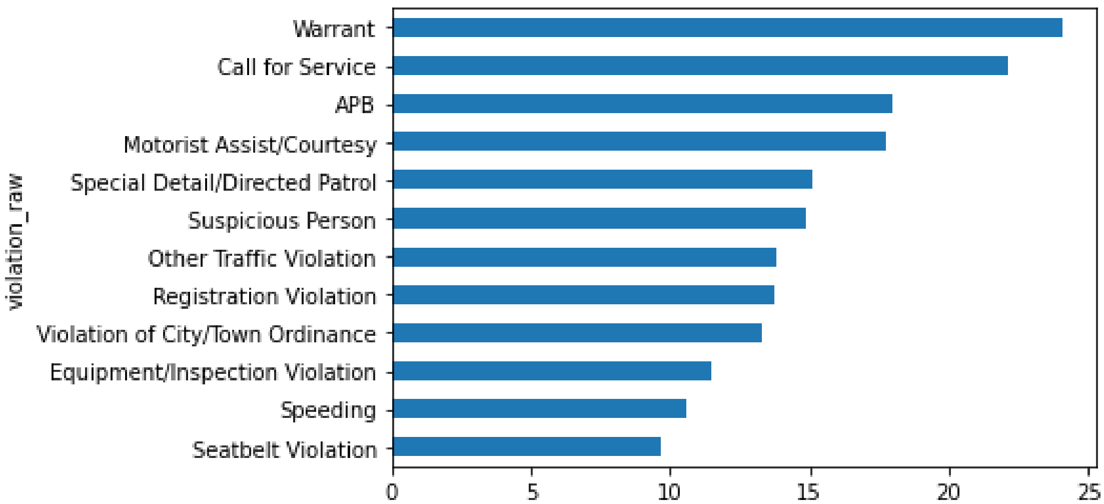

violation
Equipment
Moving violation
Other
Registration/plates
Seat belt
Speeding
20000
17500
15000
12500
10000
7500
5000
2500
0
Zone K1
Zone K2
Zone K3
district

[96]:
```python
# ¿Cuánto tiempo se puede estar detenido por un delito?

# Print the unique values in 'stop_duration'
print(ri.stop_duration.unique())

# Create a dictionary that maps strings to integers
mapping = {'0-15 Min':8,'16-30 Min':23,'30+ Min':45}

# Convert the 'stop_duration' strings to integers using the 'mapping'
ri['stop_minutes'] = ri.stop_duration.map(mapping)

# Print the unique values in 'stop_minutes'
print(ri.stop_minutes.unique())
```

```
['0-15 Min' '16-30 Min' '30+ Min']
[ 8 23 45]
```

[97]:
```python
 # Calculate the mean 'stop_minutes' for each value in 'violation_raw'
print(ri.groupby('violation_raw').stop_minutes.mean())

# Save the resulting Series as 'stop_length'
stop_length = ri.groupby('violation_raw').stop_minutes.mean()

# Sort 'stop_length' by its values and create a horizontal bar plot
stop_length.sort_values().plot(kind='barh')

# Display the plot
plt.show()
```

```
violation_raw
APB                                 17.967033
Call for Service                    22.124371
Equipment/Inspection Violation      11.445655
Motorist Assist/Courtesy            17.741463
Other Traffic Violation             13.844490
Registration Violation              13.736970
Seatbelt Violation                   9.662815
Special Detail/Directed Patrol      15.123632
Speeding                            10.581562
Suspicious Person                   14.910714
Violation of City/Town Ordinance    13.254144
Warrant                             24.055556
Name: stop_minutes, dtype: float64
```



3.4 CLIMA Y POLICÍA

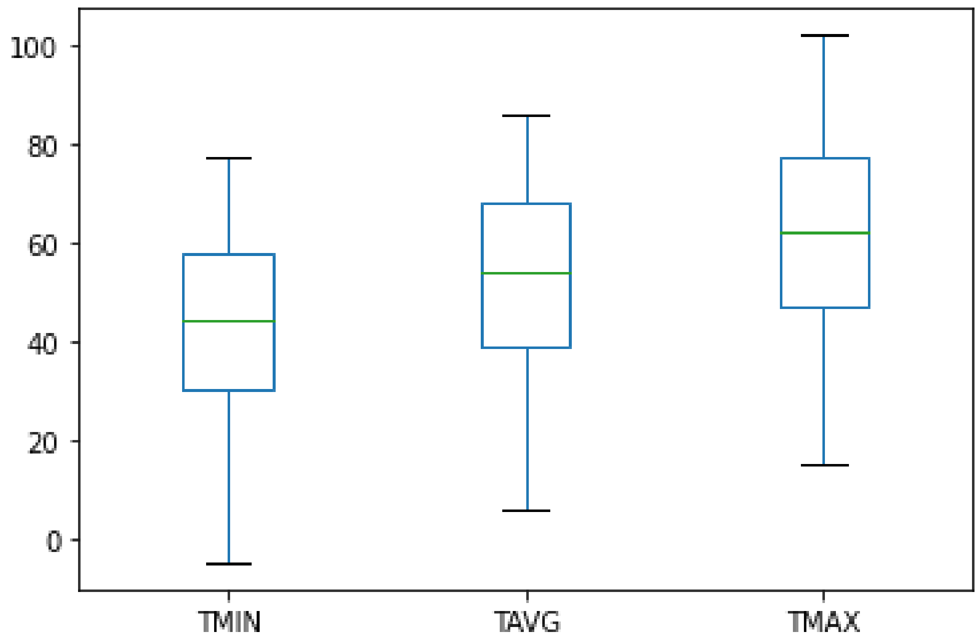
[98]:
```
### 7. EXPLORANDO EL DATASET DE WEATHER

weather = pd.read_csv("C:/Users/marco/Data Camp Python/Datasets/weather.csv")

print(weather.head())
```

```
       STATION        DATE  TAVG  TMIN  TMAX  AWND  WSF2  WT01  WT02  WT03  \
0  USW00014765  2005-01-01  44.0    35    53  8.95  25.1   1.0   NaN   NaN
1  USW00014765  2005-01-02  36.0    28    44  9.40  14.1   NaN   NaN   NaN
2  USW00014765  2005-01-03  49.0    44    53  6.93  17.0   1.0   NaN   NaN
3  USW00014765  2005-01-04  42.0    39    45  6.93  16.1   1.0   NaN   NaN
4  USW00014765  2005-01-05  36.0    28    43  7.83  17.0   1.0   NaN   NaN

   …  WT11  WT13  WT14  WT15  WT16  WT17  WT18  WT19  WT21  WT22
0  …   NaN   1.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
1  …   NaN   NaN   NaN   NaN   1.0   NaN   1.0   NaN   NaN   NaN
2  …   NaN   1.0   NaN   NaN   1.0   NaN   NaN   NaN   NaN   NaN
3  …   NaN   1.0   1.0   NaN   1.0   NaN   NaN   NaN   NaN   NaN
4  …   NaN   1.0   NaN   NaN   1.0   NaN   1.0   NaN   NaN   NaN

[5 rows x 27 columns]
```

[99]:
```
# Describe the temperature columns
print(weather[['TMIN','TAVG','TMAX']].describe())

# Create a box plot of the temperature columns
weather[['TMIN','TAVG','TMAX']].plot(kind='box')
```

```python
# Display the plot
plt.show()
```

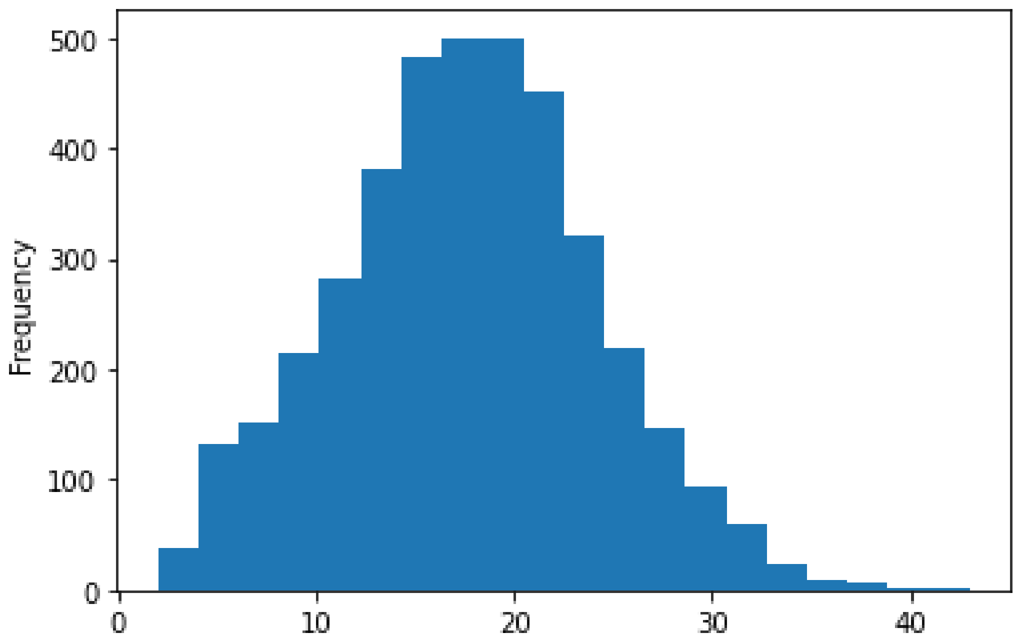
```
              TMIN         TAVG         TMAX
count  4017.000000  1217.000000  4017.000000
mean     43.484441    52.493016    61.268608
std      17.020298    17.830714    18.199517
min      -5.000000     6.000000    15.000000
25%      30.000000    39.000000    47.000000
50%      44.000000    54.000000    62.000000
75%      58.000000    68.000000    77.000000
max      77.000000    86.000000   102.000000
```



[100]:
```python
# Create a 'TDIFF' column that represents temperature difference
weather['TDIFF'] = weather.TMAX - weather.TMIN


# Describe the 'TDIFF' column
print(weather['TDIFF'].describe())

# Create a histogram with 20 bins to visualize 'TDIFF'
weather.TDIFF.plot(kind='hist',bins=20)

# Display the plot
plt.show()
```

```
count     4017.000000
```

```
mean        17.784167
std          6.350720
min          2.000000
25%         14.000000
50%         18.000000
75%         22.000000
max         43.000000
Name: TDIFF, dtype: float64
```



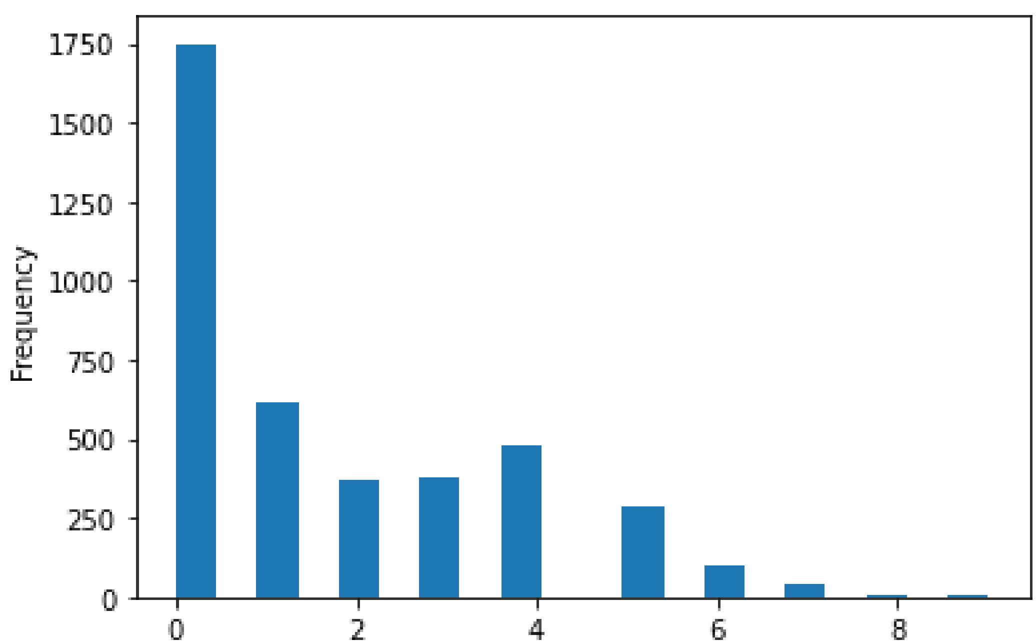
[101]:
```python
# Categorizando el clima

temp = weather.loc[:, "TAVG": "TMAX"]

print(temp.head())

# La suma de columnas está definida como sigue:

print(temp.sum())

# Y una suma por renglón:

print(temp.sum(axis = "columns").head()) # donde axis indica qué se está sumando
```

```
   TAVG  TMIN  TMAX
0  44.0    35    53
1  36.0    28    44
```

```
2  49.0    44    53
3  42.0    39    45
4  36.0    28    43
TAVG      63884.0
TMIN     174677.0
TMAX     246116.0
dtype: float64
0    132.0
1    108.0
2    146.0
3    126.0
4    107.0
dtype: float64
```

[102]:

```
# Copy 'WT01' through 'WT22' to a new DataFrame
WT = weather.loc[:, "WT01": "WT22"]

# Calculate the sum of each row in 'WT'
weather['bad_conditions'] = WT.sum(axis = "columns")

# Replace missing values in 'bad_conditions' with '0'
weather['bad_conditions'] = weather.bad_conditions.fillna(0).astype('int')

# Create a histogram to visualize 'bad_conditions'
weather.bad_conditions.plot(kind = "hist", bins = 20)

# Display the plot
plt.show()
```



[103]:
```
# Count the unique values in 'bad_conditions' and sort the index
print(weather.bad_conditions.value_counts().sort_index())

# Create a dictionary that maps integers to strings
mapping = {0:'good', 1:'bad', 2:'bad', 3:'bad', 4:'bad',5:'worse',6:'worse',7:
 ↪'worse',8:'worse',9:'worse'}

# Convert the 'bad_conditions' integers to strings using the 'mapping'
weather['rating'] = weather.bad_conditions.map(mapping)

# Count the unique values in 'rating'
print(weather.rating.value_counts())
```

```
0    1749
1     613
2     367
3     380
4     476
5     282
6     101
7      41
8       4
9       4
Name: bad_conditions, dtype: int64
bad      1836
good     1749
worse     432
Name: rating, dtype: int64
```

[104]:
```
# Specify the logical order of the weather ratings
cats = pd.CategoricalDtype(['good', 'bad', 'worse'], ordered=True)

# Change the data type of 'rating' to category
weather['rating'] = weather.rating.astype(cats)

# Examine the head of 'rating'
print(weather.rating.head())
```

```
0    bad
1    bad
2    bad
3    bad
4    bad
Name: rating, dtype: category
Categories (3, object): ['good' < 'bad' < 'worse']
```

[105]:
```
# Combinando datasets

# Reset the index of 'ri'
ri.reset_index(inplace=True)

# Examine the head of 'ri'
print(ri.head())

# Create a DataFrame from the 'DATE' and 'rating' columns
weather_rating = weather[['DATE','rating']]

# Examine the head of 'weather_rating'
print(weather_rating.head())
```

```
        stop_datetime   stop_date stop_time driver_gender driver_race  \
0 2005-01-04 12:55:00  2005-01-04     12:55             M       White
1 2005-01-23 23:15:00  2005-01-23     23:15             M       White
2 2005-02-17 04:15:00  2005-02-17     04:15             M       White
3 2005-02-20 17:15:00  2005-02-20     17:15             M       White
4 2005-02-24 01:20:00  2005-02-24     01:20             F       White

                    violation_raw  violation  search_conducted search_type  \
0  Equipment/Inspection Violation  Equipment             False         NaN
1                        Speeding   Speeding             False         NaN
2                        Speeding   Speeding             False         NaN
3                Call for Service      Other             False         NaN
4                        Speeding   Speeding             False         NaN

    stop_outcome  is_arrested stop_duration  drugs_related_stop district  \
0       Citation        False      0-15 Min               False  Zone X4
1       Citation        False      0-15 Min               False  Zone K3
2       Citation        False      0-15 Min               False  Zone X4
3  Arrest Driver         True     16-30 Min               False  Zone X1
4       Citation        False      0-15 Min               False  Zone X3

   inventory  frisk  stop_minutes
0      False  False             8
1      False  False             8
2      False  False             8
3      False  False            23
4      False  False             8
        DATE rating
0  2005-01-01    bad
1  2005-01-02    bad
2  2005-01-03    bad
3  2005-01-04    bad
4  2005-01-05    bad
```

[106]:
```python
# Examine the shape of 'ri'
print(ri.shape)

# Merge 'ri' and 'weather_rating' using a left join
ri_weather = pd.merge(left=ri, right=weather_rating, left_on='stop_date',␣
 ↪right_on='DATE', how='left')

# Examine the shape of 'ri_weather'
print(ri_weather.shape)

# Set 'stop_datetime' as the index of 'ri_weather'
ri_weather.set_index('stop_datetime', inplace=True)
```

```
(86536, 17)
(86536, 19)
```

[107]:
```python
# ¿El clima afecta la tasa de arrestos?

# Calculate the overall arrest rate
print(ri_weather.is_arrested.mean())


# Calculate the arrest rate for each 'rating'
print(ri_weather.groupby('rating').is_arrested.mean())


# Calculate the arrest rate for each 'violation' and 'rating'
print(ri_weather.groupby(['violation','rating']).is_arrested.mean())
```

```
0.0355690117407784
rating
good     0.033715
bad      0.036261
worse    0.041667
Name: is_arrested, dtype: float64
violation            rating
Equipment            good      0.059007
                     bad       0.066311
                     worse     0.097357
Moving violation     good      0.056227
                     bad       0.058050
                     worse     0.065860
Other                good      0.076966
                     bad       0.087443
                     worse     0.062893
Registration/plates  good      0.081574
                     bad       0.098160
                     worse     0.115625
```

```
Seat belt            good      0.028587
                     bad       0.022493
                     worse     0.000000
Speeding             good      0.013405
                     bad       0.013314
                     worse     0.016886
Name: is_arrested, dtype: float64
```

[108]:
```
# Save the output of the groupby operation from the last exercise
arrest_rate = ri_weather.groupby(['violation', 'rating']).is_arrested.mean()

# Print the 'arrest_rate' Series
print(arrest_rate)

# Print the arrest rate for moving violations in bad weather
print(arrest_rate.loc['Moving violation','bad'])

# Print the arrest rates for speeding violations in all three weather conditions
print(arrest_rate.loc['Speeding'])
```

```
violation            rating
Equipment            good      0.059007
                     bad       0.066311
                     worse     0.097357
Moving violation     good      0.056227
                     bad       0.058050
                     worse     0.065860
Other                good      0.076966
                     bad       0.087443
                     worse     0.062893
Registration/plates  good      0.081574
                     bad       0.098160
                     worse     0.115625
Seat belt            good      0.028587
                     bad       0.022493
                     worse     0.000000
Speeding             good      0.013405
                     bad       0.013314
                     worse     0.016886
Name: is_arrested, dtype: float64
0.05804964058049641
rating
good     0.013405
bad      0.013314
worse    0.016886
Name: is_arrested, dtype: float64
```

```
[109]: # Unstack the 'arrest_rate' Series into a DataFrame
print(arrest_rate.unstack())

# Create the same DataFrame using a pivot table
print(ri_weather.pivot_table(index='violation', columns='rating',␣
 ↪values='is_arrested'))
```

```
rating                   good       bad     worse
violation
Equipment            0.059007  0.066311  0.097357
Moving violation     0.056227  0.058050  0.065860
Other                0.076966  0.087443  0.062893
Registration/plates  0.081574  0.098160  0.115625
Seat belt            0.028587  0.022493  0.000000
Speeding             0.013405  0.013314  0.016886
rating                   good       bad     worse
violation
Equipment            0.059007  0.066311  0.097357
Moving violation     0.056227  0.058050  0.065860
Other                0.076966  0.087443  0.062893
Registration/plates  0.081574  0.098160  0.115625
Seat belt            0.028587  0.022493  0.000000
Speeding             0.013405  0.013314  0.016886
```

4 INTRODUCCIÓN A LAS BASES DE DATOS EN PYTHON

4.1 BASES DE DATOS RELACIONALES

SQLAlchemy permite integrar las funcionalidades de SQL a Python.

4.1.1 Introducción a SQL

```
[110]: from sqlalchemy import create_engine
from sqlalchemy import inspect

engine = create_engine("sqlite:///C:/Users/marco/Data Camp Python/Datasets/
 ↪census.sqlite")

connection = engine.connect()

insp = inspect(engine)
print(insp.get_table_names())

# Para accesar a una dataset SQL desde Python, se usa una "reflección"

from sqlalchemy import MetaData, Table

metadata = MetaData()
```

```python
census = Table("census", metadata, autoload = True, autoload_with = engine)

print(repr(census))

# Print the column names
print(census.columns.keys())

# Print full metadata of census
print(repr(metadata.tables['census']))

# Import select
from sqlalchemy import select

# Reflect census table via engine: census
census = Table('census', metadata, autoload=True, autoload_with=engine)

# Build select statement for census table: stmt
stmt = select([census])

# Print the emitted statement to see the SQL string
print(stmt)

# Execute the statement on connection and fetch 10 records: result
results = connection.execute(stmt).fetchmany(size=10)

# Execute the statement and print the results
print(results)

# Get the first row of the results by using an index: first_row
first_row = results[0]

# Print the first row of the results
print(first_row)

# Print the first column of the first row by using an index
print(first_row[0])

# Print the 'state' column of the first row by using its name
print(first_row['state'])
```

```
['census', 'data', 'state_fact']
Table('census', MetaData(), Column('state', VARCHAR(length=30), table=<census>),
Column('sex', VARCHAR(length=1), table=<census>), Column('age', INTEGER(),
table=<census>), Column('pop2000', INTEGER(), table=<census>), Column('pop2008',
INTEGER(), table=<census>), schema=None)
['state', 'sex', 'age', 'pop2000', 'pop2008']
Table('census', MetaData(), Column('state', VARCHAR(length=30), table=<census>),
```

```
Column('sex', VARCHAR(length=1), table=<census>), Column('age', INTEGER(),
table=<census>), Column('pop2000', INTEGER(), table=<census>), Column('pop2008',
INTEGER(), table=<census>), schema=None)
SELECT census.state, census.sex, census.age, census.pop2000, census.pop2008
FROM census
[('Illinois', 'M', 0, 89600, 95012), ('Illinois', 'M', 1, 88445, 91829),
('Illinois', 'M', 2, 88729, 89547), ('Illinois', 'M', 3, 88868, 90037),
('Illinois', 'M', 4, 91947, 91111), ('Illinois', 'M', 5, 93894, 89802),
('Illinois', 'M', 6, 93676, 88931), ('Illinois', 'M', 7, 94818, 90940),
('Illinois', 'M', 8, 95035, 86943), ('Illinois', 'M', 9, 96436, 86055)]
('Illinois', 'M', 0, 89600, 95012)
Illinois
Illinois
```

4.2 FILTROS, ORDEN Y AGRUPAMIENTO EN QUERIES

[111]:
```
engine = create_engine("sqlite:///C:/Users/marco/Data Camp Python/Datasets/
↪census.sqlite")
connection = engine.connect()
metadata = MetaData()
census = Table('census', metadata, autoload=True, autoload_with=engine)
insp = inspect(engine)
print(insp.get_table_names())

# Create a select query: stmt
stmt = select([census])

# Add a where clause to filter the results to only those for New York :␣
↪stmt_filtered
stmt = stmt.where(census.columns.state == 'New York')

# Execute the query to retrieve all the data returned: results
results = connection.execute(stmt).fetchall()

# Loop over the results and print the age, sex, and pop2000
for result in results:
    print(result.age, result.sex, result.pop2000)

# Define a list of states for which we want results
states = ['New York', 'California', 'Texas']

# Create a query for the census table: stmt
stmt = select([census])

# Append a where clause to match all the states in_ the list states
stmt = stmt.where(census.columns.state.in_(states))
```

```python
# Loop over the ResultProxy and print the state and its population in 2000
for result in connection.execute(stmt):
    print(result.state, result.pop2000)

# Import and_
from sqlalchemy import and_

# Build a query for the census table: stmt
stmt = select([census])

# Append a where clause to select only non-male records from California using␣
↪and_
stmt = stmt.where(
    # The state of California with a non-male sex
    and_(census.columns.state == 'California',
         census.columns.sex != 'M'
         )
)

# Loop over the ResultProxy printing the age and sex
for result in connection.execute(stmt):
    print(result.age, result.sex)
```

```
['census', 'data', 'state_fact']
0 M 126237
1 M 124008
2 M 124725
3 M 126697
4 M 131357
5 M 133095
6 M 134203
7 M 137986
8 M 139455
9 M 142454
10 M 145621
11 M 138746
12 M 135565
13 M 132288
14 M 132388
15 M 131959
16 M 130189
17 M 132566
18 M 132672
19 M 133654
20 M 132121
21 M 126166
22 M 123215
23 M 121282
```

```
24 M 118953
25 M 123151
26 M 118727
27 M 122359
28 M 128651
29 M 140687
30 M 149558
31 M 139477
32 M 138911
33 M 139031
34 M 145440
35 M 156168
36 M 153840
37 M 152078
38 M 150765
39 M 152606
40 M 159345
41 M 148628
42 M 147892
43 M 144195
44 M 139354
45 M 141953
46 M 131875
47 M 128767
48 M 125406
49 M 124155
50 M 125955
51 M 118542
52 M 118532
53 M 124418
54 M 95025
55 M 92652
56 M 90096
57 M 95340
58 M 83273
59 M 77213
60 M 77054
61 M 72212
62 M 70967
63 M 66461
64 M 64361
65 M 64385
66 M 58819
67 M 58176
68 M 57310
69 M 57057
70 M 57761
71 M 53775
```

```
72 M 53568
73 M 51263
74 M 48440
75 M 46702
76 M 43508
77 M 40730
78 M 37950
79 M 35774
80 M 32453
81 M 26803
82 M 25041
83 M 21687
84 M 18873
85 M 88366
0 F 120355
1 F 118219
2 F 119577
3 F 121029
4 F 125247
5 F 128227
6 F 128428
7 F 131161
8 F 133646
9 F 135746
10 F 138287
11 F 131904
12 F 129028
13 F 126571
14 F 125682
15 F 125409
16 F 122770
17 F 123978
18 F 125307
19 F 127956
20 F 129184
21 F 124575
22 F 123701
23 F 124108
24 F 122624
25 F 127474
26 F 123033
27 F 128125
28 F 134795
29 F 146832
30 F 152973
31 F 144001
32 F 143930
33 F 144653
```

34 F 151147
35 F 159228
36 F 159999
37 F 157911
38 F 156103
39 F 159284
40 F 163331
41 F 155353
42 F 153688
43 F 151615
44 F 146774
45 F 148318
46 F 139802
47 F 138062
48 F 134107
49 F 134399
50 F 136630
51 F 130843
52 F 130196
53 F 136064
54 F 106579
55 F 104847
56 F 101857
57 F 108406
58 F 94346
59 F 88584
60 F 88932
61 F 82899
62 F 82172
63 F 77171
64 F 76032
65 F 76498
66 F 70465
67 F 71088
68 F 70847
69 F 71377
70 F 74378
71 F 70611
72 F 70513
73 F 69156
74 F 68042
75 F 68410
76 F 64971
77 F 61287
78 F 58911
79 F 56865
80 F 54553
81 F 46381

```
82 F 45599
83 F 40525
84 F 37436
85 F 226378
New York 126237
New York 124008
New York 124725
New York 126697
New York 131357
New York 133095
New York 134203
New York 137986
New York 139455
New York 142454
New York 145621
New York 138746
New York 135565
New York 132288
New York 132388
New York 131959
New York 130189
New York 132566
New York 132672
New York 133654
New York 132121
New York 126166
New York 123215
New York 121282
New York 118953
New York 123151
New York 118727
New York 122359
New York 128651
New York 140687
New York 149558
New York 139477
New York 138911
New York 139031
New York 145440
New York 156168
New York 153840
New York 152078
New York 150765
New York 152606
New York 159345
New York 148628
New York 147892
New York 144195
```

```
New York 139354
New York 141953
New York 131875
New York 128767
New York 125406
New York 124155
New York 125955
New York 118542
New York 118532
New York 124418
New York 95025
New York 92652
New York 90096
New York 95340
New York 83273
New York 77213
New York 77054
New York 72212
New York 70967
New York 66461
New York 64361
New York 64385
New York 58819
New York 58176
New York 57310
New York 57057
New York 57761
New York 53775
New York 53568
New York 51263
New York 48440
New York 46702
New York 43508
New York 40730
New York 37950
New York 35774
New York 32453
New York 26803
California 252494
California 247978
California 250644
California 257443
California 266855
California 272801
California 274899
California 277580
California 283553
California 285478
```

```
California 284518
California 269009
California 262671
California 254889
California 253023
California 251962
California 249220
California 255482
California 252607
California 248356
California 250156
California 238235
California 235718
California 239698
California 240655
California 250964
California 245324
California 251413
California 260869
California 276142
California 293816
California 273159
California 268484
California 263472
California 269607
California 286895
California 284414
California 280861
California 281214
California 278802
California 290332
California 267684
California 268045
California 261885
California 252175
California 255340
California 239126
California 229057
California 219293
California 214700
California 219017
California 203068
California 200466
California 207237
California 160674
California 158483
California 150235
California 150046
```

```
California 133017
California 124106
California 121984
California 114331
California 110491
California 102859
California 99345
California 100052
California 91053
California 89634
California 88258
California 87840
California 88575
California 80843
California 79376
California 76365
California 73697
California 72885
California 69738
California 65865
California 62867
California 58012
California 51806
California 43254
California 40083
California 34144
California 30384
California 136442
California 239605
California 236543
California 240010
California 245739
California 254522
California 260264
California 261296
California 264083
California 270447
California 271482
California 270567
California 256656
California 249887
California 242724
California 240752
California 240170
California 233186
California 235767
California 234949
California 233477
```

```
California 233532
California 223990
California 222035
California 227742
California 228401
California 238602
California 233133
California 240008
California 249185
California 266010
California 278894
California 260916
California 256168
California 252784
California 256283
California 276234
California 277592
California 276277
California 275129
California 276094
California 283554
California 265614
California 265895
California 263355
California 255016
California 256779
California 244172
California 236211
California 226391
California 221928
California 225414
California 212545
California 208500
California 215228
California 168388
California 166675
California 158368
California 160423
California 142287
California 133235
California 132033
California 123328
California 120982
California 114959
California 111942
California 113547
California 104910
California 103883
```

```
California 102061
California 103181
California 106514
California 99453
California 100574
California 99772
California 99390
California 99277
California 95046
California 90193
California 86911
California 81990
California 75849
California 65410
California 61518
California 54748
California 50746
California 294583
New York 25041
New York 21687
New York 18873
New York 88366
New York 120355
New York 118219
New York 119577
New York 121029
New York 125247
New York 128227
New York 128428
New York 131161
New York 133646
New York 135746
New York 138287
New York 131904
New York 129028
New York 126571
New York 125682
New York 125409
New York 122770
New York 123978
New York 125307
New York 127956
New York 129184
New York 124575
New York 123701
New York 124108
New York 122624
New York 127474
```

```
New York 123033
New York 128125
New York 134795
New York 146832
New York 152973
New York 144001
New York 143930
New York 144653
New York 151147
New York 159228
New York 159999
New York 157911
New York 156103
New York 159284
New York 163331
New York 155353
New York 153688
New York 151615
New York 146774
New York 148318
New York 139802
New York 138062
New York 134107
New York 134399
New York 136630
New York 130843
New York 130196
New York 136064
New York 106579
New York 104847
New York 101857
New York 108406
New York 94346
New York 88584
New York 88932
New York 82899
New York 82172
New York 77171
New York 76032
New York 76498
New York 70465
New York 71088
New York 70847
New York 71377
New York 74378
New York 70611
New York 70513
New York 69156
```

New York 68042
New York 68410
New York 64971
New York 61287
New York 58911
New York 56865
New York 54553
New York 46381
New York 45599
New York 40525
New York 37436
New York 226378
Texas 172223
Texas 165635
Texas 165337
Texas 164292
Texas 165785
Texas 166278
Texas 167170
Texas 169210
Texas 171199
Texas 170521
Texas 173734
Texas 167859
Texas 166474
Texas 166014
Texas 166081
Texas 167257
Texas 165881
Texas 171567
Texas 170011
Texas 164671
Texas 163295
Texas 153946
Texas 150839
Texas 152673
Texas 153769
Texas 156739
Texas 153181
Texas 155480
Texas 161048
Texas 165852
Texas 167982
Texas 158505
Texas 153855
Texas 151149
Texas 155095
Texas 164514

Texas 167136
Texas 168668
Texas 167261
Texas 169195
Texas 173212
Texas 164647
Texas 163690
Texas 161774
Texas 154542
Texas 154603
Texas 145891
Texas 141254
Texas 133710
Texas 129998
Texas 128278
Texas 123298
Texas 120815
Texas 126031
Texas 95701
Texas 95537
Texas 93337
Texas 91482
Texas 82603
Texas 76614
Texas 73441
Texas 69422
Texas 67820
Texas 63502
Texas 62593
Texas 62994
Texas 57324
Texas 55581
Texas 54657
Texas 53235
Texas 52902
Texas 49046
Texas 46608
Texas 44784
Texas 42390
Texas 40487
Texas 37785
Texas 35332
Texas 33199
Texas 29635
Texas 27357
Texas 21864
Texas 20249
Texas 16946

Texas 15033
Texas 69392
Texas 164724
Texas 158669
Texas 157386
Texas 157374
Texas 158236
Texas 158722
Texas 160506
Texas 162126
Texas 163788
Texas 163500
Texas 165717
Texas 160176
Texas 159167
Texas 158693
Texas 158580
Texas 159654
Texas 155841
Texas 158372
Texas 156767
Texas 156778
Texas 156625
Texas 147729
Texas 144433
Texas 147865
Texas 146961
Texas 151098
Texas 148823
Texas 151810
Texas 158452
Texas 165252
Texas 164600
Texas 155658
Texas 150518
Texas 148996
Texas 152593
Texas 163350
Texas 167597
Texas 168463
Texas 168421
Texas 169355
Texas 171412
Texas 164244
Texas 163809
Texas 162822
Texas 155226
Texas 155427

```
Texas 149105
Texas 144081
Texas 136873
Texas 133610
Texas 133121
Texas 127211
Texas 125058
Texas 129694
Texas 99379
Texas 100403
Texas 97778
Texas 95755
Texas 87189
Texas 82764
Texas 79048
Texas 75160
Texas 74358
Texas 70332
Texas 70089
Texas 71266
Texas 65074
Texas 64383
Texas 63639
Texas 62713
Texas 64996
Texas 59894
Texas 58527
Texas 57708
Texas 56446
Texas 55989
Texas 52656
Texas 48993
Texas 47681
Texas 44609
Texas 42132
Texas 35378
Texas 33852
Texas 30076
Texas 27961
Texas 171538
0 F
1 F
2 F
3 F
4 F
5 F
6 F
7 F
```

```
8 F
9 F
10 F
11 F
12 F
13 F
14 F
15 F
16 F
17 F
18 F
19 F
20 F
21 F
22 F
23 F
24 F
25 F
26 F
27 F
28 F
29 F
30 F
31 F
32 F
33 F
34 F
35 F
36 F
37 F
38 F
39 F
40 F
41 F
42 F
43 F
44 F
45 F
46 F
47 F
48 F
49 F
50 F
51 F
52 F
53 F
54 F
55 F
```

```
56 F
57 F
58 F
59 F
60 F
61 F
62 F
63 F
64 F
65 F
66 F
67 F
68 F
69 F
70 F
71 F
72 F
73 F
74 F
75 F
76 F
77 F
78 F
79 F
80 F
81 F
82 F
83 F
84 F
85 F
```

[112]:
```python
# Ordenando datos

# Build a query to select the state column: stmt
stmt = select([census.columns.state])

# Order stmt by the state column
stmt = stmt.order_by(census.columns.state)

# Execute the query and store the results: results
results = connection.execute(stmt).fetchall()

# Print the first 10 results
print(results[:10])

# Import desc
from sqlalchemy import desc
```

```python
# Build a query to select the state column: stmt
stmt = select([census.columns.state])

# Order stmt by state in descending order: rev_stmt
rev_stmt = stmt.order_by(desc(census.columns.state))

# Execute the query and store the results: rev_results
rev_results = connection.execute(rev_stmt).fetchall()

# Print the first 10 rev_results
print(rev_results[:10])

# Build a query to select state and age: stmt
stmt = select([census.columns.state, census.columns.age])

# Append order by to ascend by state and descend by age
stmt = stmt.order_by(census.columns.state, desc(census.columns.age))

# Execute the statement and store all the records: results
results = connection.execute(stmt).fetchall()

# Print the first 20 results
print(results[:20])
```

```
[('Alabama',), ('Alabama',), ('Alabama',), ('Alabama',), ('Alabama',),
('Alabama',), ('Alabama',), ('Alabama',), ('Alabama',), ('Alabama',)]
[('Wyoming',), ('Wyoming',), ('Wyoming',), ('Wyoming',), ('Wyoming',),
('Wyoming',), ('Wyoming',), ('Wyoming',), ('Wyoming',), ('Wyoming',)]
[('Alabama', 85), ('Alabama', 85), ('Alabama', 84), ('Alabama', 84), ('Alabama',
83), ('Alabama', 83), ('Alabama', 82), ('Alabama', 82), ('Alabama', 81),
('Alabama', 81), ('Alabama', 80), ('Alabama', 80), ('Alabama', 79), ('Alabama',
79), ('Alabama', 78), ('Alabama', 78), ('Alabama', 77), ('Alabama', 77),
('Alabama', 76), ('Alabama', 76)]
```

[113]:
```python
# Import func
from sqlalchemy import func

# Build a query to select the state and count of ages by state: stmt
stmt = select([census.columns.state, func.count(census.columns.age)])

# Group stmt by state
stmt = stmt.group_by(census.columns.state)

# Execute the statement and store all the records: results
results = connection.execute(stmt).fetchall()
```

```python
# Print results
print(results)

# Print the keys/column names of the results returned
print(results[0].keys())

# Import func
from sqlalchemy import func

# Build an expression to calculate the sum of pop2008 labeled as population
pop2008_sum = func.sum(census.columns.pop2008).label('population')

# Build a query to select the state and sum of pop2008: stmt
stmt = select([census.columns.state, pop2008_sum])

# Group stmt by state
stmt = stmt.group_by(census.columns.state)

# Execute the statement and store all the records: results
results = connection.execute(stmt).fetchall()

# Print results
print(results)

# Print the keys/column names of the results returned
print(results[0].keys())
```

```
[('Alabama', 172), ('Alaska', 172), ('Arizona', 172), ('Arkansas', 172),
('California', 172), ('Colorado', 172), ('Connecticut', 172), ('Delaware', 172),
('District of Columbia', 172), ('Florida', 172), ('Georgia', 172), ('Hawaii',
172), ('Idaho', 172), ('Illinois', 172), ('Indiana', 172), ('Iowa', 172),
('Kansas', 172), ('Kentucky', 172), ('Louisiana', 172), ('Maine', 172),
('Maryland', 172), ('Massachusetts', 172), ('Michigan', 172), ('Minnesota',
172), ('Mississippi', 172), ('Missouri', 172), ('Montana', 172), ('Nebraska',
172), ('Nevada', 172), ('New Hampshire', 172), ('New Jersey', 172), ('New
Mexico', 172), ('New York', 172), ('North Carolina', 172), ('North Dakota',
172), ('Ohio', 172), ('Oklahoma', 172), ('Oregon', 172), ('Pennsylvania', 172),
('Rhode Island', 172), ('South Carolina', 172), ('South Dakota', 172),
('Tennessee', 172), ('Texas', 172), ('Utah', 172), ('Vermont', 172),
('Virginia', 172), ('Washington', 172), ('West Virginia', 172), ('Wisconsin',
172), ('Wyoming', 172)]
RMKeyView(['state', 'count_1'])
[('Alabama', 4649367), ('Alaska', 664546), ('Arizona', 6480767), ('Arkansas',
2848432), ('California', 36609002), ('Colorado', 4912947), ('Connecticut',
3493783), ('Delaware', 869221), ('District of Columbia', 588910), ('Florida',
18257662), ('Georgia', 9622508), ('Hawaii', 1250676), ('Idaho', 1518914),
('Illinois', 12867077), ('Indiana', 6373299), ('Iowa', 3000490), ('Kansas',
2782245), ('Kentucky', 4254964), ('Louisiana', 4395797), ('Maine', 1312972),
```

```
('Maryland', 5604174), ('Massachusetts', 6492024), ('Michigan', 9998854),
('Minnesota', 5215815), ('Mississippi', 2922355), ('Missouri', 5891974),
('Montana', 963802), ('Nebraska', 1776757), ('Nevada', 2579387), ('New
Hampshire', 1314533), ('New Jersey', 8670204), ('New Mexico', 1974993), ('New
York', 19465159), ('North Carolina', 9121606), ('North Dakota', 634282),
('Ohio', 11476782), ('Oklahoma', 3620620), ('Oregon', 3786824), ('Pennsylvania',
12440129), ('Rhode Island', 1046535), ('South Carolina', 4438870), ('South
Dakota', 800997), ('Tennessee', 6202407), ('Texas', 24214127), ('Utah',
2730919), ('Vermont', 620602), ('Virginia', 7648902), ('Washington', 6502019),
('West Virginia', 1812879), ('Wisconsin', 5625013), ('Wyoming', 529490)]
RMKeyView(['state', 'population'])
```

[114]:

```
# SQLAlchemy y Pandas para visualización

# import pandas
import pandas as pd

# Create a DataFrame from the results: df
df = pd.DataFrame(results)

# Set column names
df.columns = results[0].keys()

# Print the Dataframe
print(df)
```

```
                   state  population
0                Alabama     4649367
1                 Alaska      664546
2                Arizona     6480767
3               Arkansas     2848432
4             California    36609002
5               Colorado     4912947
6            Connecticut     3493783
7               Delaware      869221
8   District of Columbia      588910
9                Florida    18257662
10               Georgia     9622508
11                Hawaii     1250676
12                 Idaho     1518914
13              Illinois    12867077
14               Indiana     6373299
15                  Iowa     3000490
16                Kansas     2782245
17              Kentucky     4254964
18             Louisiana     4395797
19                 Maine     1312972
20              Maryland     5604174
```

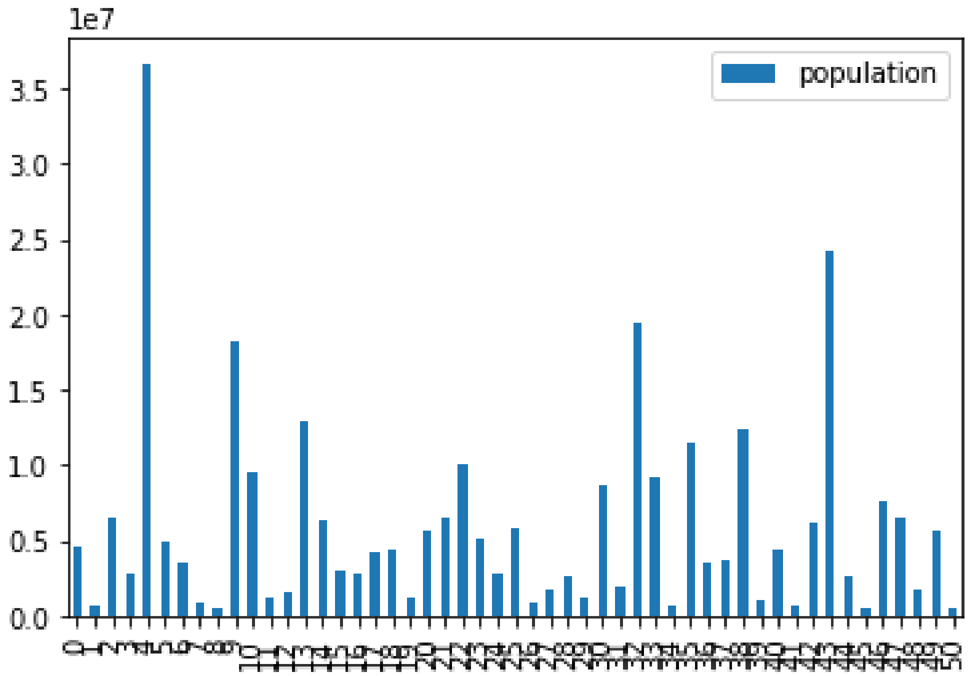
```
21         Massachusetts     6492024
22              Michigan     9998854
23             Minnesota     5215815
24           Mississippi     2922355
25              Missouri     5891974
26               Montana      963802
27              Nebraska     1776757
28                Nevada     2579387
29         New Hampshire     1314533
30            New Jersey     8670204
31            New Mexico     1974993
32              New York    19465159
33        North Carolina     9121606
34          North Dakota      634282
35                  Ohio    11476782
36              Oklahoma     3620620
37                Oregon     3786824
38          Pennsylvania    12440129
39          Rhode Island     1046535
40        South Carolina     4438870
41          South Dakota      800997
42             Tennessee     6202407
43                 Texas    24214127
44                  Utah     2730919
45               Vermont      620602
46              Virginia     7648902
47            Washington     6502019
48         West Virginia     1812879
49             Wisconsin     5625013
50               Wyoming      529490
```

[115]:

```
# Import pyplot as plt from matplotlib
from matplotlib import pyplot as plt

# Create a DataFrame from the results: df
df = pd.DataFrame(results)

# Set Column names
df.columns = results[0].keys()

# Print the DataFrame
print(df)

# Plot the DataFrame
df.plot.bar()

plt.show()
```

	state	population
0	Alabama	4649367
1	Alaska	664546
2	Arizona	6480767
3	Arkansas	2848432
4	California	36609002
5	Colorado	4912947
6	Connecticut	3493783
7	Delaware	869221
8	District of Columbia	588910
9	Florida	18257662
10	Georgia	9622508
11	Hawaii	1250676
12	Idaho	1518914
13	Illinois	12867077
14	Indiana	6373299
15	Iowa	3000490
16	Kansas	2782245
17	Kentucky	4254964
18	Louisiana	4395797
19	Maine	1312972
20	Maryland	5604174
21	Massachusetts	6492024
22	Michigan	9998854
23	Minnesota	5215815
24	Mississippi	2922355
25	Missouri	5891974
26	Montana	963802
27	Nebraska	1776757
28	Nevada	2579387
29	New Hampshire	1314533
30	New Jersey	8670204
31	New Mexico	1974993
32	New York	19465159
33	North Carolina	9121606
34	North Dakota	634282
35	Ohio	11476782
36	Oklahoma	3620620
37	Oregon	3786824
38	Pennsylvania	12440129
39	Rhode Island	1046535
40	South Carolina	4438870
41	South Dakota	800997
42	Tennessee	6202407
43	Texas	24214127
44	Utah	2730919
45	Vermont	620602
46	Virginia	7648902

```
47             Washington     6502019
48          West Virginia     1812879
49              Wisconsin     5625013
50                Wyoming      529490
```



4.3 QUERIES SQL AVANZADAS

[116]:
```python
# Build query to return state names by population difference from 2008 to 2000:␣
↪stmt
stmt = select([census.columns.state, (census.columns.pop2008-census.columns.
↪pop2000).label('pop_change')])

# Append group by for the state: stmt_grouped
stmt_grouped = stmt.group_by(census.columns.state)

# Append order by for pop_change descendingly: stmt_ordered
stmt_ordered = stmt_grouped.order_by(desc('pop_change'))

# Return only 5 results: stmt_top5
stmt_top5 = stmt_ordered.limit(5)

# Use connection to execute stmt_top5 and fetch all results
results = connection.execute(stmt_top5).fetchall()
```

```
# Print the state and population change for each record
for result in results:
    print('{}:{}'.format(result.state, result.pop_change))
```

```
Texas:40137
California:35406
Florida:21954
Arizona:14377
Georgia:13357
```

[117]:
```
# import case, cast and Float from sqlalchemy
from sqlalchemy import case, cast, Float

# Build an expression to calculate female population in 2000
female_pop2000 = func.sum(
    case([
        (census.columns.sex == 'F', census.columns.pop2000)
    ], else_=0))

# Cast an expression to calculate total population in 2000 to Float
total_pop2000 = cast(func.sum(census.columns.pop2000), Float)

# Build a query to calculate the percentage of women in 2000: stmt
stmt = select([female_pop2000 / total_pop2000 * 100])

# Execute the query and store the scalar result: percent_female
percent_female = connection.execute(stmt).scalar()

# Print the percentage
print(percent_female)
```

```
51.09467432293413
```

4.4 CREACIÓN Y MANIPULACIÓN DE BASES DE DATOS PROPIAS

[118]:
```
# Import Table, Column, String, Integer, Float, Boolean from sqlalchemy
from sqlalchemy import Table, Column, String, Integer, Float, Boolean

# Define a new table with a name, count, amount, and valid column: data
data = Table('data', metadata,
             Column('name', String(255)),
             Column('count', Integer()),
             Column('amount', Float()),
             Column('valid', Boolean())
)

# Use the metadata to create the table
metadata.create_all(engine)
```

```python
# Print table details
print(repr(data))
```

```
Table('data', MetaData(), Column('name', String(length=255), table=<data>),
Column('count', Integer(), table=<data>), Column('amount', Float(),
table=<data>), Column('valid', Boolean(), table=<data>), schema=None)
```