main

June 16, 2023

1 Descripción del dataset

Este dataset concierne a las redes de metro globales. Se estudia los kilómetros, paradas, el uso, año de construcción y de última ampliación así como aspectos relacionados con la ciudad donde se encuentra: población, localización geográfica, país.

Tras limpiar estos datos se analizan aspectos como el número de viajes por habitante, latitudes con alta densidad de uso (este asiático), ciudades con mas estaciones, etc.

Las bases de datos originales están localizables en: - del metro https://www.kaggle.com/datasets/drahulsingh/metro-systems-worldwide - de ciudades https://simplemaps.com/data/world-cities

2 Integración y selección de los datos de interés a analizar

[1]:
```
import numpy as np
import pandas as pd
from itertools import chain

import matplotlib.pyplot as plt
from matplotlib import ticker

plt.style.use('seaborn-v0_8-whitegrid')
import seaborn as sns

import warnings
import copy
```

2.1 Carga de bases de datos

Cargamos dataset del metro (de https://www.kaggle.com/datasets/drahulsingh/metro-systems-worldwide).

[2]:
```
df_metro = pd.read_csv("../dataset/Metro-Systems-Worldwide.csv")
```

[3]:
```
df_metro.rename(columns={'Country\r\nregion': 'Country',
                    'Service\r\nopened': 'Year opened',
                    'Last\r\nexpanded': 'Last year expanded',
```

```
                    'Annual ridership\r\n(millions)': 'Annual ridership␣
↪(millions)'
                   }, inplace=True)
```

[4]:
```
df_metro.head()
```

[4]:
```
           City    Country                      Name  Year opened  \
0       Algiers    Algeria             Algiers Metro         2011
1  Buenos Aires  Argentina  Buenos Aires Underground         1913
2       Yerevan    Armenia             Yerevan Metro         1981
3        Sydney  Australia              Sydney Metro         2019
4        Vienna    Austria            Vienna U-Bahn          1978

  Last year expanded  Stations      System length Annual ridership (millions)
0               2018        19  18.5 km (11.5 mi)                 45.3 (2019)
1               2019        78  56.7 km (35.2 mi)                 74.0 (2020)
2               1996        10   12.1 km (7.5 mi)                 23.3 (2022)
3                  -        13       36 km (22 mi)                16.3 (2022)
4               2017        98  83.3 km (51.8 mi)                459.8 (2019)
```

Cargamos dataset de ciudades (de https://simplemaps.com/data/world-cities):

[5]:
```
df_cities = pd.read_csv("../dataset/worldcities.csv")
df_cities =␣
↪df_cities[["city_ascii","country","admin_name","population","lat","lng"]]
df_cities.head()
```

[5]:
```
  city_ascii    country   admin_name  population      lat       lng
0      Tokyo      Japan        Tōkyō  37732000.0  35.6897  139.6922
1    Jakarta  Indonesia      Jakarta  33756000.0  -6.1750  106.8275
2      Delhi      India        Delhi  32226000.0  28.6100   77.2300
3  Guangzhou      China    Guangdong  26940000.0  23.1300  113.2600
4     Mumbai      India  Mahārāshtra  24973000.0  19.0761   72.8775
```

2.2 Combinar ambos datasets

2.2.1 Campos del merge

Pero hay ciudades homónimas en muchos países (en ambos dataframes):

[6]:
```
df_cities["city_ascii"].value_counts(), df_metro["City"].value_counts()
```

[6]:
```
(Santa Cruz      17
 San Fernando    16
 Santa Ana       15
 Santa Maria     14
 San Juan        14
                 ..
 Arsenyev         1
```

```
Panjakent        1
Kleve            1
Venkatagiri      1
Nordvik          1
Name: city_ascii, Length: 41140, dtype: int64,
Seoul            3
Tokyo            3
New York City    3
London           2
Manila           2
                ..
Medellín         1
Prague           1
Copenhagen       1
Santo Domingo    1
Hanoi            1
Name: City, Length: 194, dtype: int64)
```

Incluso mas complicado, hay ciudades repetidas en dataset del metro:

[7]:
```
unique_cities_metro = df_metro.groupby(['City', 'Country']).agg({'Name': lambda␣
↪x: x.index.tolist()}).reset_index()
unique_cities_metro.rename(columns={'Name': 'Original indexes'}, inplace=True)
unique_cities_metro['Count'] = [len(k) for k in unique_cities_metro['Original␣
↪indexes']]
mask_cities_metro = unique_cities_metro['Count']>1
unique_cities_metro[mask_cities_metro]
```

[7]:

	City	Country	Original indexes	Count
11	Bangkok	Thailand	[173, 174]	2
94	London	United Kingdom	[185, 186]	2
100	Manila	Philippines	[150, 151]	2
120	New York City	United States	[194, 195, 196]	3
130	Philadelphia	United States	[197, 198]	2
151	Seoul	South Korea	[136, 137, 138]	3
174	Tokyo	Japan	[124, 125, 126]	3
190	Yokohama	Japan	[127, 128]	2

[8]:
```
# repeated cities vs sum
unique_cities_metro[mask_cities_metro]['Count'].shape[0] , \
unique_cities_metro[mask_cities_metro]['Count'].sum()
```

[8]: (8, 19)

Comprobamos que se debe que en una misma ciudad-país puede haber varios sistemas de metro:

[9]:
```
indexes = list(chain.
↪from_iterable(unique_cities_metro[mask_cities_metro]['Original indexes']))
```

```
df_metro.iloc[indexes ,:]
```

```
[9]:            City         Country                               Name  \
173         Bangkok        Thailand                       BTS Skytrain
174         Bangkok        Thailand        Metropolitan Rapid Transit
185          London  United Kingdom               London Underground
186          London  United Kingdom            Docklands Light Railway
150          Manila     Philippines  Manila Light Rail Transit System
151          Manila     Philippines   Manila Metro Rail Transit System
194   New York City   United States              New York City Subway
195   New York City   United States             Staten Island Railway
196   New York City   United States                              PATH
197    Philadelphia   United States                             SEPTA
198    Philadelphia   United States                    PATCO Speedline
136           Seoul     South Korea          Seoul Metropolitan Subway
137           Seoul     South Korea                Korail metro lines
138           Seoul     South Korea      Shinbundang Line (Neo Trans)
124           Tokyo           Japan                       Toei Subway
125           Tokyo           Japan                       Tokyo Metro
126           Tokyo           Japan                       Rinkai Line
127        Yokohama           Japan          Yokohama Municipal Subway
128        Yokohama           Japan                  Minatomirai Line

     Year opened Last year expanded  Stations        System length  \
173         1999               2021        60    68.2 km (42.4 mi)
174         2004               2019        53        71 km (44 mi)
185         1863               2021       272      402 km (250 mi)
186         1987               2011        45        34 km (21 mi)
150         1984               2021        33    37.2 km (23.1 mi)
151         1999               2000        13    16.9 km (10.5 mi)
194         1904               2017       424      399 km (248 mi)
195         1925               2017        21    22.5 km (14.0 mi)
196         1908               1937        13    22.2 km (13.8 mi)
197         1907               1973        72    59.1 km (36.7 mi)
198         1936               1980        13    22.9 km (14.2 mi)
136         1974               2022       279  345.3 km (214.6 mi)
137         1994               2022        86   151.7 km (94.3 mi)
138         2011               2022        16    33.4 km (20.8 mi)
124         1960               2002        99   109.0 km (67.7 mi)
125         1927               2020       142  195.1 km (121.2 mi)
126         1996               2002         8     12.2 km (7.6 mi)
127         1972               2008        40    53.4 km (33.2 mi)
128         2004               2008         6      4.1 km (2.5 mi)

    Annual ridership (millions)
173                236.9 (2020)
174                 95.3 (2020)
```

```
185                1,026 (2022)
186                 39.9 (2020)
150                218.2 (2019)
151                 96.9 (2019)
194              1,793.1 (2022)
195                  3.8 (2022)
196                 45.5 (2022)
197                 41.2 (2022)
198                  4.9 (2022)
136              2,127.2 (2020)
137                426.4 (2019)
138                122.5 (2019)
124              1,174.9 (2019)
125              2,757.4 (2019)
126                 95.0 (2019)
127                243.2 (2019)
128                 80.6 (2019)
```

Por suerte no hay la triple combinanción ciudad-país-nombre del metro. Así tras hacer el merge no hemos de eliminar la columna `Name`.

[10]:
```
unique_cities_metro_name = df_metro.groupby(['City', 'Country','Name']).size().
  ↪reset_index(name='Count')
mask = unique_cities_metro_name['Count']>1
unique_cities_metro_name[mask]
```

[10]:
```
Empty DataFrame
Columns: [City, Country, Name, Count]
Index: []
```

Por otro lado, también hay varias ciudades hómonimas en mismo país:

[11]:
```
unique_cities_cities = df_cities.groupby(['city_ascii', 'country']).size().
  ↪reset_index(name='Count')
mask_cities_cities = unique_cities_cities['Count']>1
unique_cities_cities[mask_cities_cities]
```

[11]:

	city_ascii	country	Count
55	Abasolo	Mexico	2
56	Abay	Kazakhstan	2
80	Aberdeen	United States	4
97	Abington	United States	2
169	Acatlan	Mexico	2
…	…	…	…
42236	Zhangjiazhuang	China	2
42239	Zhangping	China	2
42262	Zheleznogorsk	Russia	2
42282	Zhijiang	China	2

```
42301        Zhongshan           China      2

[1452 rows x 3 columns]
```

Pero de todas esas, solo nos competen las que tienen metro:

[12]:
```python
pd.merge(unique_cities_metro, unique_cities_cities[mask_cities_cities],
              left_on=["City","Country"],
              right_on=["city_ascii","country"])[['City','Country','Count_y']]
```

[12]:
```
        City        Country  Count_y
0   Changsha          China        2
1  Cleveland  United States        3
2   Dongguan          China        2
3     Fuzhou          China        2
4    Gwangju    South Korea        2
5     Jaipur          India        2
6      Miami  United States        2
7     Suzhou          China        2
8    Taizhou          China        2
9       Wuxi          China        2
```

Vamos a suponer en cada conflicto que la ciudad con metro será la de mayor población. Es decir, vuelvo a guardar como ya hice en `unique_cities_metro` los índices del dataset original. Vuelvo a mergear. Y en cada registro (conflicto) guardo índices de ciudades mas pequeñas. Las que finalmente elimino del dataset de ciudades.

[13]:
```python
unique_cities_cities = df_cities.groupby(['city_ascii', 'country']).agg({'lat':␣
  ↪lambda x: x.index.tolist()}).reset_index()
unique_cities_cities.rename(columns={'lat': 'Original indexes'}, inplace=True)
unique_cities_cities['Count'] = [len(k) for k in unique_cities_cities['Original␣
  ↪indexes']]
mask_cities_cities = unique_cities_cities['Count']>1
unique_cities_cities[mask_cities_cities]
```

[13]:
```
          city_ascii        country              Original indexes  Count
55           Abasolo         Mexico                 [7800, 36261]      2
56              Abay     Kazakhstan                [22412, 38952]      2
80          Aberdeen  United States  [20999, 26822, 29107, 29794]      4
97          Abington  United States                [14521, 28967]      2
169          Acatlan         Mexico                [22509, 24113]      2
…                  …              …                             …    …
42236  Zhangjiazhuang         China                [15064, 25281]      2
42239      Zhangping          China                 [2782, 44147]      2
42262   Zheleznogorsk        Russia                 [5920, 12556]      2
42282       Zhijiang          China                  [1574, 5287]      2
42301      Zhongshan          China                 [275, 21491]      2
```

```
[1452 rows x 4 columns]
```

[14]:
```python
df_merge = pd.merge(unique_cities_metro,␣
 ↪unique_cities_cities[mask_cities_cities],
              left_on=["City","Country"],
              right_on=["city_ascii","country"])
df_merge[['City','Country','Count_y','Original indexes_y']]
```

[14]:
```
        City        Country  Count_y   Original indexes_y
0   Changsha          China        2          [151, 1165]
1  Cleveland  United States        3  [474, 13250, 37437]
2   Dongguan          China        2          [38, 12924]
3     Fuzhou          China        2           [186, 207]
4    Gwangju    South Korea        2          [527, 2273]
5     Jaipur          India        2         [254, 38513]
6      Miami  United States        2         [100, 34184]
7     Suzhou          China        2           [125, 172]
8    Taizhou          China        2           [135, 383]
9       Wuxi          China        2          [241, 1026]
```

[15]:
```python
df_merge.shape[0] , df_merge[['Count_y']].sum().values[0] # cities unique vs␣
 ↪cities total
```

[15]:
```
(10, 21)
```

[16]:
```python
indexes = list(chain.from_iterable(df_merge['Original indexes_y']))
df_cities.iloc[indexes ,:][['country','city_ascii','population']]
```

[16]:
```
             country city_ascii  population
151            China   Changsha   4766296.0
1165           China   Changsha    717700.0
474    United States  Cleveland   1683059.0
13250  United States  Cleveland     72589.0
37437  United States  Cleveland     11285.0
38             China   Dongguan  10646000.0
12924          China   Dongguan     75135.0
186            China     Fuzhou   4047200.0
207            China     Fuzhou   3671192.0
527      South Korea    Gwangju   1490092.0
2273     South Korea    Gwangju    310278.0
254            India     Jaipur   3073350.0
38513          India     Jaipur     10259.0
100    United States      Miami   5711945.0
34184  United States      Miami     12997.0
125            China     Suzhou   5352924.0
172            China     Suzhou   4330000.0
135            China    Taizhou   5031000.0
```

```
383            China    Taizhou   2162461.0
241            China       Wuxi   3245179.0
1026           China       Wuxi    853197.0
```

[17]:
```python
indexes_to_remove = list()
for indexes in df_merge['Original indexes_y']:
    result_dict = dict(df_cities.loc[indexes, 'population'])
    max_value = max(result_dict.values())
    indexes_with_smaller_values = [key for key, value in result_dict.items() if␣
↪value < max_value]
    indexes_to_remove.append(indexes_with_smaller_values)
indexes_to_remove = list(chain.from_iterable(indexes_to_remove))
```

[18]:
```python
len(indexes_to_remove) # cities to remove
```

[18]: 11

[19]:
```python
df_cities.shape
```

[19]: (44691, 6)

[20]:
```python
df_cities.drop(indexes_to_remove, axis=0, inplace=True)
```

[21]:
```python
df_cities.shape
```

[21]: (44680, 6)

[22]:
```python
# comprobación
unique_cities_cities = df_cities.groupby(['city_ascii', 'country']).size().
↪reset_index(name='Count')
mask_cities_cities = unique_cities_cities['Count']>1
pd.merge(unique_cities_metro, unique_cities_cities[mask_cities_cities],
              left_on=["City","Country"],
              right_on=["city_ascii","country"])
```

[22]:
```
Empty DataFrame
Columns: [City, Country, Original indexes, Count_x, city_ascii, country,
Count_y]
Index: []
```

2.2.2 pd.merge()

[23]:
```python
df = pd.merge(df_metro, df_cities,
              left_on=["City","Country"],
              right_on=["city_ascii","country"],
              how='left').drop('city_ascii', axis=1)
df.drop('Country', axis=1, inplace=True)
```

[24]:
```
df.head()
```

[24]:
```
          City                        Name  Year opened Last year expanded  \
0      Algiers               Algiers Metro         2011               2018
1  Buenos Aires  Buenos Aires Underground         1913               2019
2      Yerevan               Yerevan Metro         1981               1996
3       Sydney                Sydney Metro         2019                  -
4       Vienna              Vienna U-Bahn          1978               2017

   Stations      System length Annual ridership (millions)    country  \
0        19  18.5 km (11.5 mi)                 45.3 (2019)    Algeria
1        78  56.7 km (35.2 mi)                 74.0 (2020)  Argentina
2        10   12.1 km (7.5 mi)                 23.3 (2022)    Armenia
3        13      36 km (22 mi)                 16.3 (2022)  Australia
4        98  83.3 km (51.8 mi)                459.8 (2019)    Austria

                      admin_name  population      lat       lng
0                          Alger   3415811.0  36.7539    3.0589
1  Buenos Aires, Ciudad Autónoma de  16710000.0 -34.5997  -58.3819
2                        Yerevan   1075800.0  40.1814   44.5144
3                New South Wales   4840600.0 -33.8678  151.2100
4                           Wien   1973403.0  48.2083   16.3725
```

Comprobamos que las 205 ciudades con metros se corresponden 1 a 1 en el dataframe mergeado:

[25]:
```
[k.shape[0] for k in [df_metro,df]]
```

[25]:
```
[205, 205]
```

3 Limpieza de los datos

Estudiamos NaN

[26]:
```
def df_check_nan_null(df):
    print('** NA **')
    # note: read your csv with empty cells interpreted as empty strings by␣
 ↪simply setting keep_default_na=False
    na_cols = df.isnull().sum()
    print("\nNon NaN cols:\n", na_cols[na_cols != 0])

    print('\n\n** Blancos **')
    blank_cols = (df == "").sum()
    blank_cols = blank_cols[blank_cols != 0]
    print("\nNon zero cols:\n", blank_cols)

    return na_cols, blank_cols
```

[27]:
```
na_cols, blank_cols = df_check_nan_null(df);
```

```
** NA **

Non NaN cols:
 Annual ridership (millions)    16
country                        17
admin_name                     19
population                     17
lat                            17
lng                            17
dtype: int64


** Blancos **

Non zero cols:
 Series([], dtype: int64)
```

16 registros del dataset del metro no recogen los millones de pasajeros.

3.1 Mergeo sin ASCII

Los otros campos con 17 registros muy probablemente se deban a un incorrecto mergeado:

[28]:
```python
def df_check_nan(df, na_cols, col_index=[1]): #[0,1]
    for k in col_index:
        col = na_cols[na_cols>0].index[k]
        print(col)
        mask = df[[col]].isnull().values
        index = df[mask].index
        display(df[mask].reset_index())
    return mask, index;
mask_nan, index_nan_merged_df = df_check_nan(df, na_cols);
```

```
country

    index              City                      Name  Year opened  \
0      10          Brasília   Federal District Metro         2001
1      16         São Paulo          São Paulo Metro         1974
2      38         Hong Kong     Mass Transit Railway         1979
3      59            Ürümqi             Ürümqi Metro         2018
4      65            Xuzhou             Xuzhou Metro         2019
5      67          Medellín           Medellín Metro         1995
6      68            Prague             Prague Metro         1974
7      94            Kanpur             Kanpur Metro         2021
8     103           Isfahan    Isfahan Urban Railway         2015
9     158   Nizhny Novgorod    Nizhny Novgorod Metro         1985
10    172           Taoyuan            Taoyuan Metro         2017
11    179             İzmir              İzmir Metro         2000
12    194     New York City     New York City Subway         1904
```

```
13   195             New York City   Staten Island Railway         1925
14   196             New York City                    PATH         1908
15   199  San Francisco (Bay Area)                    BART         1972
16   201          Washington, D.C.        Washington Metro         1976

   Last year expanded  Stations         System length  \
0                2020        27    42.38 km (26.33 mi)
1                2021        89     104.4 km (64.9 mi)
2                2022        99    209.0 km (129.9 mi)
3                2019        21  27.615 km (17.159 mi)
4                2021        51    64.35 km (39.99 mi)
5                2012        27      31.3 km (19.4 mi)
6                2015        58      65.4 km (40.6 mi)
7                   -         9      8.98 km (5.58 mi)
8                2018        20      20.2 km (12.6 mi)
9                2018        15    21.82 km (13.56 mi)
10                  -        22      53.1 km (33.0 mi)
11               2014        17          20 km (12 mi)
12               2017       424        399 km (248 mi)
13               2017        21      22.5 km (14.0 mi)
14               1937        13      22.2 km (13.8 mi)
15               2020        47    191.5 km (119.0 mi)
16               2023        98        206 km (128 mi)

   Annual ridership (millions) country admin_name  population  lat  lng
0                42.8 (2019)     NaN        NaN         NaN  NaN  NaN
1            1,104.149 (2022)     NaN        NaN         NaN  NaN  NaN
2             1,616.30 (2021)     NaN        NaN         NaN  NaN  NaN
3               19.11 (2020)     NaN        NaN         NaN  NaN  NaN
4               20.94 (2020)     NaN        NaN         NaN  NaN  NaN
5               215.2 (2022)     NaN        NaN         NaN  NaN  NaN
6               251.4 (2020)     NaN        NaN         NaN  NaN  NaN
7                        NaN     NaN        NaN         NaN  NaN  NaN
8                        NaN     NaN        NaN         NaN  NaN  NaN
9                29.9 (2022)     NaN        NaN         NaN  NaN  NaN
10               28.0 (2019)     NaN        NaN         NaN  NaN  NaN
11                100 (2019)     NaN        NaN         NaN  NaN  NaN
12            1,793.1 (2022)     NaN        NaN         NaN  NaN  NaN
13                3.8 (2022)     NaN        NaN         NaN  NaN  NaN
14               45.5 (2022)     NaN        NaN         NaN  NaN  NaN
15               39.6 (2022)     NaN        NaN         NaN  NaN  NaN
16               93.0 (2022)     NaN        NaN         NaN  NaN  NaN
```

Hemos combinado los dataframes en función del código ASCII, pero el CSV del metro no recoge el ASCII (siempre), sino que a veces es en idioma original. Probamos a combinar estos 17 registros con el campo `city` en vez de `city_ascii` de `df_cities`. Lo cual funciona con Brasília, São Paulo, etc.

[29]:
```
df_cities_orig = pd.read_csv("../dataset/worldcities.csv")
df_nan_merged = pd.merge(df[mask_nan], df_cities_orig,
              left_on=["City"],
              right_on=["city"],
              how='left',
              suffixes=('_x', '')).drop('city', axis=1)

# Keep columns from the second DataFrame in case of homonyms
columns_to_keep = [col for col in df_nan_merged if not col.endswith('_x')]
df_nan_merged = df_nan_merged[columns_to_keep][df.columns]
```

Sobreescribimos en dataframe mergeado original los anteriores. Y comprobamos que NaN aún faltan:

[30]:
```
df.iloc[index_nan_merged_df,:] = df_nan_merged
df.iloc[index_nan_merged_df,:]
```

[30]:
```
                      City                   Name  Year opened  \
10                Brasília  Federal District Metro         2001
16               São Paulo        São Paulo Metro         1974
38               Hong Kong    Mass Transit Railway         1979
59                  Ürümqi           Ürümqi Metro         2018
65                 Xuzhou           Xuzhou Metro         2019
67                Medellín        Medellín Metro         1995
68                  Prague           Prague Metro         1974
94                  Kanpur           Kanpur Metro         2021
103                Isfahan   Isfahan Urban Railway         2015
158        Nizhny Novgorod  Nizhny Novgorod Metro         1985
172               Taoyuan          Taoyuan Metro         2017
179                  İzmir            İzmir Metro         2000
194          New York City     New York City Subway         1904
195          New York City   Staten Island Railway         1925
196          New York City                    PATH         1908
199  San Francisco (Bay Area)                  BART         1972
201         Washington, D.C.       Washington Metro         1976

    Last year expanded  Stations           System length  \
10                2020        27     42.38 km (26.33 mi)
16                2021        89      104.4 km (64.9 mi)
38                2022        99     209.0 km (129.9 mi)
59                2019        21   27.615 km (17.159 mi)
65                2021        51     64.35 km (39.99 mi)
67                2012        27       31.3 km (19.4 mi)
68                2015        58       65.4 km (40.6 mi)
94                   -         9       8.98 km (5.58 mi)
103               2018        20       20.2 km (12.6 mi)
158               2018        15     21.82 km (13.56 mi)
172                  -        22       53.1 km (33.0 mi)
```

```
179                2014        17          20 km (12 mi)
194                2017       424         399 km (248 mi)
195                2017        21       22.5 km (14.0 mi)
196                1937        13       22.2 km (13.8 mi)
199                2020        47     191.5 km (119.0 mi)
201                2023        98         206 km (128 mi)

    Annual ridership (millions)    country        admin_name  population  \
10                42.8 (2019)     Brazil  Distrito Federal   3039444.0
16           1,104.149 (2022)     Brazil         São Paulo  23086000.0
38            1,616.30 (2021)  Hong Kong               NaN   7450000.0
59               19.11 (2020)      China          Xinjiang   4335017.0
65               20.94 (2020)        NaN               NaN         NaN
67               215.2 (2022)   Colombia         Antioquia   2529403.0
68               251.4 (2020)    Czechia             Praha   1335084.0
94                        NaN        NaN               NaN         NaN
103                       NaN        NaN               NaN         NaN
158               29.9 (2022)        NaN               NaN         NaN
172               28.0 (2019)        NaN               NaN         NaN
179                100 (2019)     Turkey             İzmir   4320519.0
194            1,793.1 (2022)        NaN               NaN         NaN
195                3.8 (2022)        NaN               NaN         NaN
196               45.5 (2022)        NaN               NaN         NaN
199               39.6 (2022)        NaN               NaN         NaN
201               93.0 (2022)        NaN               NaN         NaN

         lat       lng
10  -15.7939  -47.8828
16  -23.5500  -46.6333
38   22.3000  114.2000
59   43.8225   87.6125
65       NaN       NaN
67    6.2308  -75.5906
68   50.0875   14.4214
94       NaN       NaN
103      NaN       NaN
158      NaN       NaN
172      NaN       NaN
179  38.4200   27.1400
194      NaN       NaN
195      NaN       NaN
196      NaN       NaN
199      NaN       NaN
201      NaN       NaN
```

[31]:
```
mask_nan, index_nan_merged_df = df_check_nan(df,na_cols);
```

```
country
```

```
   index                        City                    Name  Year opened  \
0     65                      Xuzhou           Xuzhou Metro         2019
1     94                      Kanpur           Kanpur Metro         2021
2    103                     Isfahan  Isfahan Urban Railway         2015
3    158            Nizhny Novgorod  Nizhny Novgorod Metro          1985
4    172                     Taoyuan          Taoyuan Metro         2017
5    194               New York City   New York City Subway         1904
6    195               New York City  Staten Island Railway         1925
7    196               New York City                   PATH         1908
8    199  San Francisco (Bay Area)                    BART         1972
9    201           Washington, D.C.       Washington Metro         1976

  Last year expanded  Stations        System length  \
0               2021        51  64.35 km (39.99 mi)
1                  -         9    8.98 km (5.58 mi)
2               2018        20    20.2 km (12.6 mi)
3               2018        15  21.82 km (13.56 mi)
4                  -        22    53.1 km (33.0 mi)
5               2017       424      399 km (248 mi)
6               2017        21    22.5 km (14.0 mi)
7               1937        13    22.2 km (13.8 mi)
8               2020        47  191.5 km (119.0 mi)
9               2023        98      206 km (128 mi)

  Annual ridership (millions) country admin_name  population  lat  lng
0               20.94 (2020)     NaN        NaN         NaN  NaN  NaN
1                        NaN     NaN        NaN         NaN  NaN  NaN
2                        NaN     NaN        NaN         NaN  NaN  NaN
3                29.9 (2022)     NaN        NaN         NaN  NaN  NaN
4                28.0 (2019)     NaN        NaN         NaN  NaN  NaN
5             1,793.1 (2022)     NaN        NaN         NaN  NaN  NaN
6                 3.8 (2022)     NaN        NaN         NaN  NaN  NaN
7                45.5 (2022)     NaN        NaN         NaN  NaN  NaN
8                39.6 (2022)     NaN        NaN         NaN  NaN  NaN
9                93.0 (2022)     NaN        NaN         NaN  NaN  NaN
```

3.2 Registros individuales

Ya hemos comprobado que columna ASCII no coincide con `City` del dataset del metro en estos 10 registros. Intetamos solventarlos individualmente.

3.2.1 Xuzhou

La web https://en.wikipedia.org/wiki/Xuzhou_Metro nos indica: - la provincia: Jiangsu Province - about 67 kilometres (42 mi) - In its first year of operation the ridership was 7.5 million (2019) - Number of stations: 54

Datos muy parecidos a los del dataset del metro:

[32]:
```
df_metro[df_metro["City"] == "Xuzhou"]
```

[32]:
```
      City Country          Name  Year opened Last year expanded  Stations  \
65  Xuzhou   China  Xuzhou Metro         2019               2021        51

          System length Annual ridership (millions)
65  64.35 km (39.99 mi)                 20.94 (2020)
```

El mandarin se transcibe muy asiduamente mal, vamos a filtrar ciudades con misma terminación:

[33]:
```
mask = df_cities_orig['city_ascii'].str.endswith('uzhou')
df_cities_orig[mask]
```

[33]:
```
          city city_ascii      lat       lng country iso2 iso3 admin_name  \
125     Suzhou     Suzhou  33.6333  116.9683   China   CN  CHN      Anhui
172     Suzhou     Suzhou  31.3000  120.6194   China   CN  CHN    Jiangsu
178     Luzhou     Luzhou  28.8918  105.4409   China   CN  CHN    Sichuan
181    Liuzhou    Liuzhou  24.3264  109.4281   China   CN  CHN    Guangxi
186     Fuzhou     Fuzhou  27.9814  116.3577   China   CN  CHN    Jiangxi
188    Zhuzhou    Zhuzhou  27.8407  113.1469   China   CN  CHN      Hunan
190    Chuzhou    Chuzhou  32.3062  118.3115   China   CN  CHN      Anhui
207     Fuzhou     Fuzhou  26.0769  119.2917   China   CN  CHN     Fujian
257     Wuzhou     Wuzhou  23.4833  111.3167   China   CN  CHN    Guangxi
506     Huzhou     Huzhou  30.8925  120.0875   China   CN  CHN   Zhejiang
983     Quzhou     Quzhou  28.9545  118.8763   China   CN  CHN   Zhejiang
13398   Guzhou     Guzhou  25.9452  108.5238   China   CN  CHN    Guizhou
15554  Jiuzhou    Jiuzhou  39.5054  116.5642   China   CN  CHN      Hebei
30087   Luzhou     Luzhou  23.3687  114.5196   China   CN  CHN  Guangdong

      capital  population          id
125       NaN   5352924.0  1156871297
172     minor   4330000.0  1156029196
178     minor   4218427.0  1156582079
181     minor   4157934.0  1156360785
186       NaN   4047200.0  1156915325
188     minor   4020800.0  1156041962
190       NaN   3987054.0  1156036420
207     admin   3671192.0  1156188037
257     minor   3061100.0  1156620133
506     minor   1558826.0  1156335543
983       NaN    902767.0  1156180376
13398     NaN     70098.0  1156435005
15554     NaN     49616.0  1156799658
30087     NaN     15890.0  1156708150
```

Vemos que el segundo resultado también es el único de la misma provincia que lo de Wikipedia:

[34]:
```
mask2 = df_cities_orig["admin_name"] == "Jiangsu"
match = df_cities_orig[mask*mask2]
match
```

[34]:
```
       city city_ascii   lat       lng country iso2 iso3 admin_name capital  \
172  Suzhou     Suzhou  31.3  120.6194   China   CN  CHN    Jiangsu   minor

     population          id
172   4330000.0  1156029196
```

Ya teníamos datos de una ciudad homónima del mismo país, pero comprobamos que ni `Name` (del sistema de metro de la ciudad) ni `admin_name`, así como otros campos, no son de la ciudad que nos compete.

[35]:
```
df.loc[df["City"] == "Suzhou", :]
```

[35]:
```
      City                 Name  Year opened Last year expanded  Stations  \
55  Suzhou  Suzhou Rail Transit         2012               2021       154

          System length Annual ridership (millions) country admin_name  \
55  208.2 km (129.4 mi)               308.57 (2020)   China      Anhui

    population      lat       lng
55   5352924.0  33.6333  116.9683
```

Como el nombre de la ciudad y su país son iguales en ambos registros, y no hay tercer campo con el que filtrar la unión de datasets, pues es en vano simplemente corregir el nombre de la ciudad para después mergear. Directamente añado valores NaN:

[36]:
```
cols_cities = "country,admin_name,population,lat,lng".split(",")
```

[37]:
```
df.loc[df["City"] == "Xuzhou", cols_cities] = match[cols_cities].values
```

Comparamos ambas ciudades: "Suzhou, Anhui, China" vs "Xuzhou, Jiangsu, China". Como latitud y longitud denotan, están muy próximas: según Google Maps a menos de 100 km (link).

[38]:
```
df[df["City"].isin(["Xuzhou", "Suzhou"])]
```

[38]:
```
      City                 Name  Year opened Last year expanded  Stations  \
55  Suzhou  Suzhou Rail Transit         2012               2021       154
65  Xuzhou         Xuzhou Metro         2019               2021        51

          System length Annual ridership (millions) country admin_name  \
55  208.2 km (129.4 mi)               308.57 (2020)   China      Anhui
65  64.35 km (39.99 mi)                20.94 (2020)   China    Jiangsu

    population      lat       lng
55   5352924.0  33.6333  116.9683
65   4330000.0  31.3000  120.6194
```

3.2.2 Kanpur

Los datos del metro de Wikipedia coinciden aprox. con los de nuestro dataset: https://en.wikipedia.org/wiki/Kanpur_Metro

[39]:
```
df_metro[df_metro["City"] == "Kanpur"]
```

[39]:
```
      City Country          Name  Year opened Last year expanded  Stations  \
94  Kanpur   India  Kanpur Metro         2021                  -         9

         System length Annual ridership (millions)
94  8.98 km (5.58 mi)                          NaN
```

Parece que lo han transcrito mal. Añadiendo una "h" tras la K. Entre `Khanpur` y `Khānpur` cogemos la ciudad claramente mas grande.

[40]:
```
mask = df_cities_orig['city_ascii'].str.endswith('npur') &␣
 ↪df_cities_orig['city_ascii'].str.startswith('K')
df_cities_orig[mask]
```

[40]:
```
                city   city_ascii      lat      lng   country iso2 iso3  \
3946         Khanpur      Khanpur  28.6453  70.6567  Pakistan   PK  PAK
10027        Khānpur      Khanpur  25.8572  85.9330     India   IN  IND
10613      Kanchanpur   Kanchanpur  25.6636  85.2703     India   IN  IND
31014      Kaleyānpur   Kaleyanpur  26.4297  84.9327     India   IN  IND
34532       Kishunpur    Kishunpur  25.3272  87.7173     India   IN  IND
35143   Khānjahānpur  Khanjahanpur  25.6055  86.0927     India   IN  IND
36694        Kamānpur     Kamanpur  18.6667  79.5000     India   IN  IND
37944       Kishunpur    Kishunpur  25.7947  86.8237     India   IN  IND
38149       Kalyānpur    Kalyanpur  26.4802  84.1789     India   IN  IND
40066      Kanchanpur   Kanchanpur  24.6096  84.2361     India   IN  IND

           admin_name capital  population          id
3946           Punjab     NaN    160308.0  1586169401
10027           Bihār     NaN     12066.0  1356667678
10613           Bihār     NaN      8616.0  1356097827
31014           Bihār     NaN     13704.0  1356786428
34532           Bihār     NaN     11237.0  1356155445
35143           Bihār     NaN     10899.0  1356018473
36694  Andhra Pradesh     NaN     11048.0  1356269111
37944           Bihār     NaN      9963.0  1356145353
38149           Bihār     NaN      9802.0  1356695468
40066           Bihār     NaN      9758.0  1356197058
```

[41]:
```
mask = df_cities_orig['city'] == 'Khanpur'
match = df_cities_orig[mask]
match
```

```
[41]:          city city_ascii      lat      lng   country iso2 iso3 admin_name  \
      3946  Khanpur    Khanpur  28.6453  70.6567  Pakistan   PK  PAK     Punjab

           capital  population          id
      3946     NaN    160308.0  1586169401
```

Pero hay error en el país:

```
[42]: match = match.copy()
      match.loc[:, 'country'] = "India"
      match
```

```
[42]:          city city_ascii      lat      lng country iso2 iso3 admin_name  \
      3946  Khanpur    Khanpur  28.6453  70.6567   India   PK  PAK     Punjab

           capital  population          id
      3946     NaN    160308.0  1586169401
```

```
[43]: df.loc[df["City"] == "Kanpur", cols_cities] = match[cols_cities].values
```

3.2.3 Isfahan

La web del metro (https://en.wikipedia.org/wiki/Isfahan_Metro) nos hipervincula con la Wikipedia de la ciudad, ésta indica que la ciudad tiene 2.2 millones de habitantes

```
[44]: df_metro[df_metro["City"] == "Isfahan"]
```

```
[44]:         City Country                   Name  Year opened Last year expanded  \
      103  Isfahan    Iran  Isfahan Urban Railway         2015               2018

          Stations      System length Annual ridership (millions)
      103       20  20.2 km (12.6 mi)                         NaN
```

Por tanto, la ciudad es `Esfahan`:

```
[45]: mask = df_cities_orig['city_ascii'].str.endswith('ahan')
      df_cities_orig[mask]
```

```
[45]:                 city      city_ascii      lat       lng      country  \
      377          Eşfahān         Esfahan  32.6447   51.6675         Iran
      4979        Behbahān        Behbahan  30.5958   50.2417         Iran
      6862   Pinagkaisahan   Pinagkaisahan  14.5229  121.0555  Philippines
      7478        Pinyahan        Pinyahan  14.6400  121.0461  Philippines
      7608         Ardahan         Ardahan  41.1111   42.7022       Turkey
      8176        Mukdahan        Mukdahan  16.5431  104.7228     Thailand
      13012    Pinamungahan    Pinamungahan  10.2667  123.5833  Philippines
      18238       Simbahan        Simbahan   6.3000  120.5833  Philippines
      19399       Diplahan        Diplahan   7.6911  122.9853  Philippines
      20752          Nāhan           Nahan  30.5500   77.3000        India
```

```
25831            Hanahan            Hanahan  32.9302  -80.0027  United States
31755           Dargahān           Dargahan  26.9636   56.0622           Iran
38115    Phibun Mangsahan  Phibun Mangsahan  15.2482  105.2296       Thailand
38369         Kūcheşfahān        Kuchesfahan  37.2783   49.7728           Iran
41817          Dānesfahān         Danesfahan  35.8108   49.7422           Iran
42529            Harahan            Harahan  29.9374  -90.2031  United States

      iso2 iso3          admin_name capital  population          id
377     IR  IRN             Eşfahān   admin   2219343.0  1364023865
4979    IR  IRN           Khūzestān   minor    122604.0  1364393434
6862    PH  PHL              Makati     NaN     57343.0  1608216406
7478    PH  PHL              Quezon     NaN     28129.0  1608055244
7608    TR  TUR             Ardahan   admin     42226.0  1792379425
8176    TH  THA            Mukdahan   admin     33102.0  1764994534
13012   PH  PHL                Cebu     NaN     75131.0  1608414270
18238   PH  PHL                Sulu     NaN     36374.0  1608320501
19399   PH  PHL   Zamboanga Sibugay     NaN     32585.0  1608903309
20752   IN  IND    Himāchal Pradesh     NaN     28899.0  1356417528
25831   US  USA      South Carolina     NaN     20381.0  1840014256
31755   IR  IRN           Hormozgān     NaN     14525.0  1364735795
38115   TH  THA     Ubon Ratchathani   minor     10842.0  1764591980
38369   IR  IRN               Gīlān     NaN     10026.0  1364862389
41817   IR  IRN              Qazvīn     NaN      9434.0  1364115128
42529   US  USA           Louisiana     NaN      9137.0  1840013997
```

[46]:
```python
mask = df_cities_orig['city_ascii'] == 'Esfahan'
match = df_cities_orig[mask]
match
```

[46]:
```
        city city_ascii      lat      lng country iso2 iso3 admin_name  \
377  Eşfahān    Esfahan  32.6447  51.6675    Iran   IR  IRN    Eşfahān

    capital  population          id
377   admin   2219343.0  1364023865
```

[47]:
```python
df.loc[df["City"] == "Isfahan", cols_cities] = match[cols_cities].values
```

3.2.4 Nizhny Novgorod

[48]:
```python
df_metro[df_metro["City"] == "Nizhny Novgorod"]
```

[48]:
```
                City Country                    Name  Year opened  \
158  Nizhny Novgorod  Russia  Nizhny Novgorod Metro         1985

    Last year expanded  Stations          System length  \
158               2018        15  21.82 km (13.56 mi)
```

```
    Annual ridership (millions)
158                  29.9 (2022)
```

[49]:
```
mask = df_cities_orig['city_ascii'].str.endswith('Novgorod')
df_cities_orig[mask]
```

[49]:
```
                  city        city_ascii      lat      lng country iso2 iso3  \
606   Nizhniy Novgorod  Nizhniy Novgorod  56.3269  44.0075  Russia   RU  RUS
2959  Velikiy Novgorod  Velikiy Novgorod  58.5210  31.2758  Russia   RU  RUS

                   admin_name capital  population          id
606   Nizhegorodskaya Oblast’   admin   1264075.0  1643012126
2959    Novgorodskaya Oblast’   admin    222594.0  1643774241
```

[50]:
```
mask = df_cities_orig['city_ascii'] == 'Nizhniy Novgorod'
match = df_cities_orig[mask]
match
```

[50]:
```
                 city        city_ascii      lat      lng country iso2 iso3  \
606  Nizhniy Novgorod  Nizhniy Novgorod  56.3269  44.0075  Russia   RU  RUS

                  admin_name capital  population          id
606  Nizhegorodskaya Oblast’   admin   1264075.0  1643012126
```

[51]:
```
df.loc[df["City"] == "Nizhny Novgorod", cols_cities] = match[cols_cities].values
```

3.2.5 Taoyuan

[52]:
```
df_metro[df_metro["City"] == "Taoyuan"]
```

[52]:
```
        City Country          Name  Year opened Last year expanded  Stations  \
172  Taoyuan  Taiwan  Taoyuan Metro         2017                  -        22

         System length Annual ridership (millions)
172  53.1 km (33.0 mi)                 28.0 (2019)
```

[53]:
```
mask = df_cities_orig['iso2'] == "TW"
df_cities_orig[mask]
```

[53]:
```
            city city_ascii      lat       lng country iso2 iso3 admin_name  \
47        Taipei     Taipei  25.0375  121.5625  Taiwan   TW  TWN     Taipei
262     Taichung   Taichung  24.1439  120.6794  Taiwan   TW  TWN   Taichung
292    Kaohsiung  Kaohsiung  22.6150  120.2975  Taiwan   TW  TWN  Kaohsiung
437       Tainan     Tainan  22.9833  120.1833  Taiwan   TW  TWN     Tainan
486      Zhongli    Zhongli  24.9650  121.2168  Taiwan   TW  TWN    Taoyuan
…            …          …        …        …      …   …    …          …
41269       Fuli       Fuli  23.1333  121.2833  Taiwan   TW  TWN    Hualien
41356      Xinpi      Xinpi  22.4880  120.5814  Taiwan   TW  TWN   Pingtung
```

```
42650  Nanzhuang  Nanzhuang  24.5699  121.0157  Taiwan   TW  TWN     Miaoli
43176      Beibu      Beibu  24.6639  121.0681  Taiwan   TW  TWN    Hsinchu
44143    Jianshi    Jianshi  24.5761  121.3081  Taiwan   TW  TWN    Hsinchu

       capital  population          id
47     primary   9078000.0  1158881289
262      admin   3033885.0  1158689622
292      admin   2733566.0  1158331334
437      admin   1874686.0  1158061376
486        NaN   1632616.0  1158025380
…          …          …          …
41269      NaN      9681.0  1158634303
41356      NaN      9540.0  1158537415
42650      NaN      9029.0  1158569080
43176      NaN      8647.0  1158656385
44143      NaN      9532.0  1158994660

[159 rows x 11 columns]
```

[54]:
```python
df_cities_orig.loc[mask & df_cities_orig['city_ascii'].str.startswith('Tao')]
```

[54]:
```
                  city        city_ascii      lat       lng country iso2 iso3  \
1701  Taoyuan District  Taoyuan District  24.9913  121.3143  Taiwan   TW  TWN

     admin_name capital  population          id
1701    Taoyuan   admin    443273.0  1158127875
```

[55]:
```python
mask = df_cities_orig['city_ascii'] == 'Taoyuan District'
match = df_cities_orig[mask]
match
```

[55]:
```
                  city        city_ascii      lat       lng country iso2 iso3  \
1701  Taoyuan District  Taoyuan District  24.9913  121.3143  Taiwan   TW  TWN

     admin_name capital  population          id
1701    Taoyuan   admin    443273.0  1158127875
```

[56]:
```python
df.loc[df["City"] == "Taoyuan", cols_cities] = match[cols_cities].values
```

3.2.6 New York City

[57]:
```python
mask = df_cities_orig['city_ascii'].str.startswith('New York')
match = df_cities_orig[mask]
match
```

[57]:
```
        city city_ascii      lat      lng        country iso2 iso3 admin_name  \
11  New York   New York  40.6943 -73.9249  United States   US  USA   New York
```

```
   capital  population          id
11     NaN  18972871.0  1840034016
```

[58]:
```
df.loc[df["City"] == "New York City", cols_cities] = match[cols_cities].values
```

3.2.7 San Francisco (Bay Area)

[59]:
```
mask = df_cities_orig['iso2'] == "US"
match = df_cities_orig.loc[mask & df_cities_orig['city_ascii'].str.
 ↪startswith('San Francisco')]
match
```

[59]:
```
              city     city_ascii      lat       lng        country iso2 iso3  \
237  San Francisco  San Francisco  37.7558 -122.4449  United States   US  USA

     admin_name capital  population          id
237  California     NaN   3290197.0  1840021543
```

[60]:
```
df.loc[df["City"] == "San Francisco (Bay Area)", cols_cities] =␣
 ↪match[cols_cities].values
```

3.2.8 Washington, D.C.

[61]:
```
mask = df_cities_orig['admin_name'] == "District of Columbia"
match = df_cities_orig.loc[mask & df_cities_orig['city_ascii'].str.
 ↪startswith('Washington')]
match
```

[61]:
```
           city  city_ascii      lat      lng        country iso2 iso3  \
149  Washington  Washington  38.9047 -77.0163  United States   US  USA

               admin_name  capital  population          id
149  District of Columbia  primary   4810669.0  1840006060
```

[62]:
```
df.loc[df["City"] == "Washington, D.C.", cols_cities] = match[cols_cities].
 ↪values
```

3.3 Comprobación

Nos faltan 3 identificadores municipales. Pero su relevancia es escasa.

[63]:
```
na_cols, blank_cols = df_check_nan_null(df);
```

```
** NA **

Non NaN cols:
 Annual ridership (millions)    16
```

```
admin_name                  3
dtype: int64


** Blancos **

Non zero cols:
 Series([], dtype: int64)
```

[64]:
```
df_check_nan(df, na_cols);
```

```
admin_name

   index       City                  Name  Year opened Last year expanded  \
0     38  Hong Kong  Mass Transit Railway         1979               2022
1    129     Almaty          Almaty Metro         2011               2022
2    163  Singapore    Mass Rapid Transit         1987               2022

   Stations        System length Annual ridership (millions)     country  \
0        99  209.0 km (129.9 mi)              1,616.30 (2021)   Hong Kong
1        11     13.4 km (8.3 mi)                 17.1 (2022)  Kazakhstan
2       134  230.2 km (143.0 mi)                766.5 (2021)   Singapore

  admin_name  population      lat       lng
0        NaN   7450000.0  22.3000  114.2000
1        NaN   1916822.0  43.2775   76.8958
2        NaN   5983000.0   1.3000  103.8000
```

3.4 Correción de campos

[65]:
```
def check_no_numeric_rows(df=df, col='Ridership (millions)'):
    non_float_rows = df[~pd.to_numeric(df[col], errors="coerce").notnull()]
    return non_float_rows
```

3.4.1 Solo NaN como tal

El campo `Last year expanded` registra - si no ha habido expansión de la red de metro alguna desde su construcción.

[66]:
```
col = "Last year expanded"
check_no_numeric_rows(col=col).head(3)
```

[66]:
```
         City                  Name  Year opened Last year expanded  Stations  \
3      Sydney          Sydney Metro         2019                  -        13
6       Dhaka      Dhaka Metro Rail         2022                  -         9
29  Dongguan  Dongguan Rail Transit         2016                  -        15

       System length Annual ridership (millions)     country  \
3      36 km (22 mi)                 16.3 (2022)   Australia
```

```
6    11.7 km (7.3 mi)                         NaN  Bangladesh
29  37.7 km (23.4 mi)                 35.06 (2020)       China

         admin_name  population       lat       lng
3   New South Wales   4840600.0 -33.8678  151.2100
6             Dhaka  18627000.0  23.7639   90.3889
29        Guangdong  10646000.0  23.0475  113.7493
```

[67]:
```
df[col].replace("-", np.nan, inplace=True)
df[col] = df[col].astype(float)
check_no_numeric_rows(col=col).head(3)
```

[67]:
```
        City                  Name  Year opened  Last year expanded  \
3     Sydney          Sydney Metro         2019                 NaN
6      Dhaka      Dhaka Metro Rail         2022                 NaN
29  Dongguan  Dongguan Rail Transit         2016                 NaN

    Stations      System length Annual ridership (millions)     country  \
3         13       36 km (22 mi)                 16.3 (2022)   Australia
6          9   11.7 km (7.3 mi)                          NaN  Bangladesh
29        15  37.7 km (23.4 mi)                 35.06 (2020)       China

         admin_name  population       lat       lng
3   New South Wales   4840600.0 -33.8678  151.2100
6             Dhaka  18627000.0  23.7639   90.3889
29        Guangdong  10646000.0  23.0475  113.7493
```

Hay 22 metros sin ampliaciones

[68]:
```
df_check_nan_null(df);
```

```
** NA **

Non NaN cols:
 Last year expanded             22
Annual ridership (millions)    16
admin_name                      3
dtype: int64


** Blancos **

Non zero cols:
 Series([], dtype: int64)
```

Campo de año de construcción siempre aparece:

[69]:
```
check_no_numeric_rows(col="Year opened").head(3)
```

[69]:
```
Empty DataFrame
Columns: [City, Name, Year opened, Last year expanded, Stations, System length,
Annual ridership (millions), country, admin_name, population, lat, lng]
Index: []
```

3.4.2 Uso del metro: ridership

[70]:
```python
col = 'Annual ridership (millions)'
df[['Ridership (millions)', 'Ridership Year']] = df[col].str.extract(r'^(\d+\.
↪\d+)\s+\((\d+)\)$')
df.drop(col, axis=1, inplace=True)

col = 'Ridership (millions)'
df[col] = df[col].astype(float)

df.head(3)
```

[70]:
```
           City                        Name  Year opened  Last year expanded  \
0       Algiers               Algiers Metro         2011              2018.0
1  Buenos Aires  Buenos Aires Underground         1913              2019.0
2       Yerevan               Yerevan Metro         1981              1996.0

   Stations       System length    country                        admin_name  \
0        19  18.5 km (11.5 mi)    Algeria                             Alger
1        78  56.7 km (35.2 mi)  Argentina  Buenos Aires, Ciudad Autónoma de
2        10   12.1 km (7.5 mi)    Armenia                           Yerevan

   population      lat      lng  Ridership (millions) Ridership Year
0   3415811.0  36.7539   3.0589                  45.3           2019
1  16710000.0 -34.5997 -58.3819                  74.0           2020
2   1075800.0  40.1814  44.5144                  23.3           2022
```

[71]:
```python
col = 'Ridership (millions)'
df[col] = df[col].astype(float)
```

[72]:
```python
check_no_numeric_rows().head()
```

[72]:
```
          City              Name  Year opened  Last year expanded  Stations  \
6        Dhaka  Dhaka Metro Rail         2022                 NaN         9
15    Salvador    Salvador Metro         2014              2018.0        19
16   São Paulo   São Paulo Metro         1974              2021.0        89
22     Beijing    Beijing Subway         1971              2023.0       370
26     Chengdu     Chengdu Metro         2010              2020.0       284

          System length     country admin_name  population      lat      lng  \
6      11.7 km (7.3 mi)  Bangladesh      Dhaka  18627000.0  23.7639  90.3889
15    32.5 km (20.2 mi)      Brazil      Bahia   2886698.0 -12.9747  -38.4767
```

```
16   104.4 km (64.9 mi)      Brazil  São Paulo  23086000.0 -23.5500  -46.6333
22  785.7 km (488.2 mi)       China    Beijing  18522000.0  39.9040  116.4075
26  518.5 km (322.2 mi)       China    Sichuan  14645000.0  30.6600  104.0633

    Ridership (millions) Ridership Year
6                    NaN            NaN
15                   NaN            NaN
16                   NaN            NaN
22                   NaN            NaN
26                   NaN            NaN
```

Añado el uso por habitante de cada ciudad:

[73]:
```
df['Ridership per capita'] = df['Ridership (millions)']*1e6 / df['population']
```

3.4.3 Longitud de vías

[74]:
```
col = 'System length'
df[col] = df[col].str.extract(r'^([\d.]+)')
df[col] = df[col].astype(float)
df.tail(3)
```

[74]:
```
         City            Name  Year opened  Last year expanded  Stations  \
202  Tashkent  Tashkent Metro         1977              2023.0        48
203   Caracas   Caracas Metro         1983              2015.0        49
204     Hanoi     Hanoi Metro         2021                 NaN        12

     System length     country        admin_name  population      lat  \
202           59.1  Uzbekistan          Toshkent   2571668.0  41.3111
203           67.2   Venezuela  Distrito Capital   2245744.0  10.4806
204           13.1     Vietnam            Hà Nội   8246600.0  21.0283

          lng  Ridership (millions) Ridership Year  Ridership per capita
202   69.2797                 136.7           2022             53.156162
203  -66.9036                   NaN            NaN                   NaN
204  105.8542                   NaN            NaN                   NaN
```

[75]:
```
check_no_numeric_rows(col=col)
```

[75]:
```
Empty DataFrame
Columns: [City, Name, Year opened, Last year expanded, Stations, System length,
country, admin_name, population, lat, lng, Ridership (millions), Ridership Year,
Ridership per capita]
Index: []
```

4 Análisis de los datos

Comprobamos que los campos enteros ó decimales son así:

[76]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 205 entries, 0 to 204
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   City                   205 non-null    object
 1   Name                   205 non-null    object
 2   Year opened            205 non-null    int64
 3   Last year expanded     183 non-null    float64
 4   Stations               205 non-null    int64
 5   System length          205 non-null    float64
 6   country                205 non-null    object
 7   admin_name             202 non-null    object
 8   population             205 non-null    float64
 9   lat                    205 non-null    float64
 10  lng                    205 non-null    float64
 11  Ridership (millions)   140 non-null    float64
 12  Ridership Year         140 non-null    object
 13  Ridership per capita   140 non-null    float64
dtypes: float64(7), int64(2), object(5)
memory usage: 24.0+ KB
```

4.1 Distribución de campos numéricos

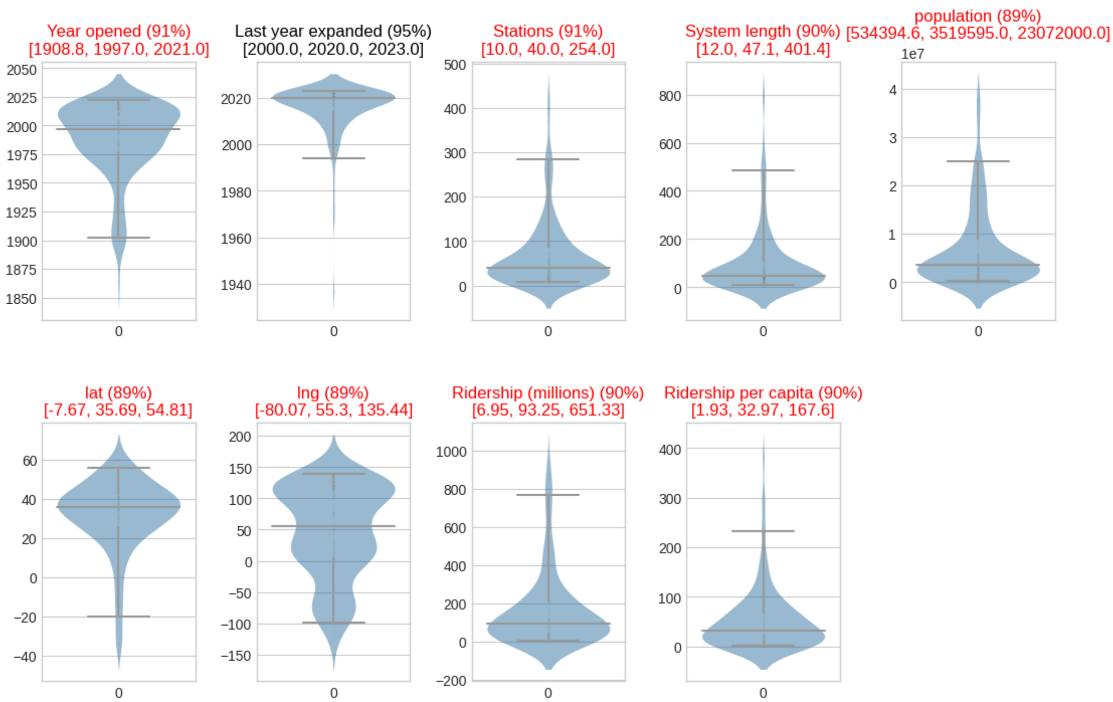
Estudiamos la estadística descriptiva de las variables continuas. Si título en rojo entonces menos del 95% de valores se encuentran entre percentiles 2.5 y 97.5. Entre corchetes se muestra: [percentil 5, media, percentil 95].

[77]:
```
def box_violin_plot(df, cols):
    fig_cols = 5
    fig_rows = int(np.ceil(len(cols)/fig_cols))

    fig = plt.figure(figsize=(13, 4*fig_rows))
    fig.subplots_adjust(hspace=0.4, wspace=0.4)

    for i,col in enumerate(cols):
            x = df[col].dropna().values

            percentiles = [np.percentile(x, k) for k in [5,50,95]]
            percent_range_5_95 = (1 - ((sum(x<percentiles[0]) +␣
 ↪sum(x>percentiles[-1]))/len(x)))*100
            color = 'k'
            if percent_range_5_95< 95: color='r'

            ax = fig.add_subplot(fig_rows, fig_cols, i+1)
```

```
            sns.boxplot(x, showfliers=False, showbox=False, whis=[2.5,97.5],␣
↪color='w')
            sns.violinplot(x, inner='point', linewidth=.01)
            plt.setp(ax.collections, alpha=.5)
            ax.set_title(f"{col} ({percent_range_5_95:.0f}%)\n{[round(g,2) for␣
↪g in percentiles ]}", color=color)
    return fig
```

[78]:
```
cols_num = df.select_dtypes(include=[np.number]).columns
```

[79]:
```
with warnings.catch_warnings():
    box_violin_plot(df, cols_num);
```



Se observa que: - el boom de los metros se inicio en la década de los 50 - la gran mayoria de redes metropolitanas no llegan al centenar de estaciones - una silueta distributiva casi idéntica al nº de estaciones es la de la longitud (ver correlaciones en siguiente apartado) - la mayoria de ciudades no llegan a la decena de millones de habitantes - el uso per capita toma rangos tan variado como case 2 para el percentil 5 y menos de 165 para el percentil 95, es decir, unas 165/5=33 veces mas. Muy dispar.

Estos gráficos nos ayudan también a identificar valores extremos (outliers). Por ejemplo que el nº de estaciones o la pobación tuviera valores negativos, latitudes fuera de rango [-90,90], etc. Nada de esto sucede si nos fijamos en los boxplots, aunque podemos codificarlo para cerciorarnos:

[80]:
```
df[cols_num].describe()
```

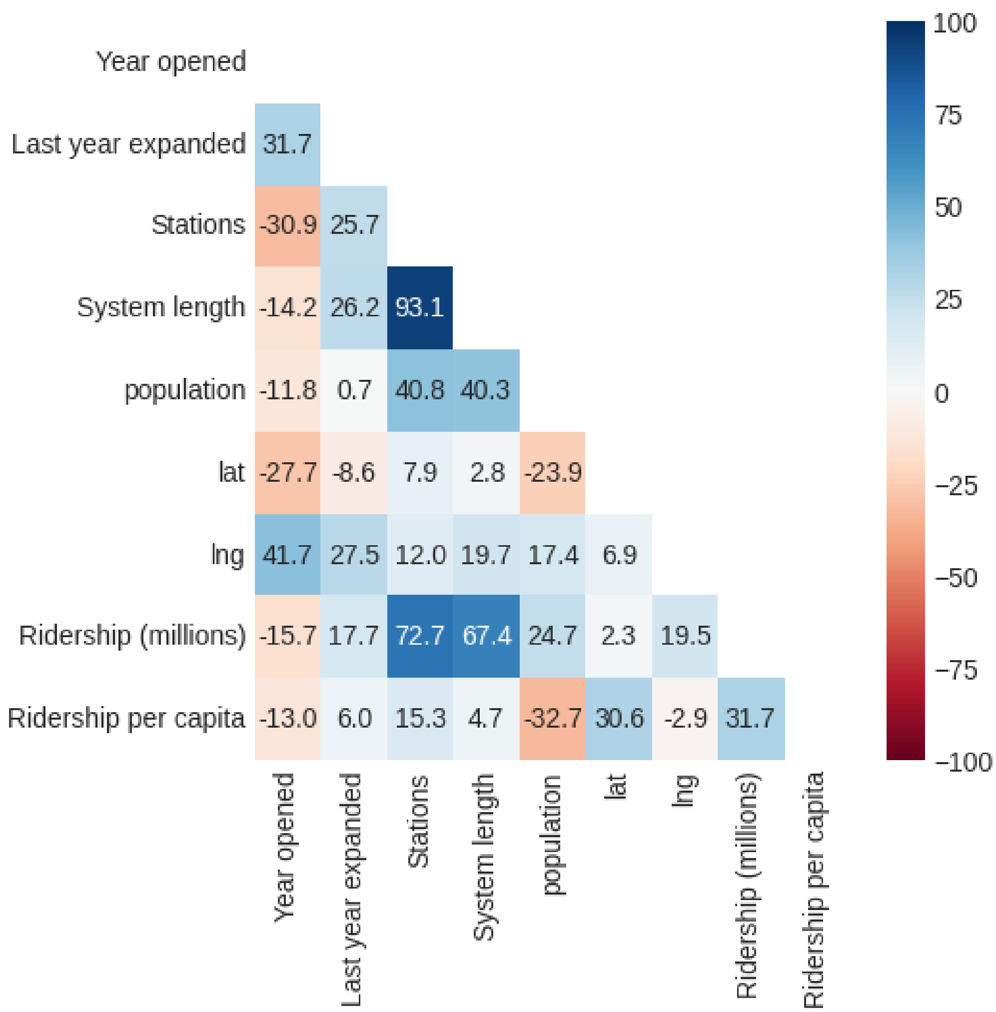
[80]:
```
       Year opened  Last year expanded    Stations  System length  \
count   205.000000          183.000000  205.000000     205.000000
mean   1988.907317         2015.918033   69.243902      99.689698
std      32.145902           10.565799   76.443360     132.231555
min    1863.000000         1937.000000    2.000000       4.100000
25%    1977.000000         2014.000000   21.000000      28.000000
50%    1997.000000         2020.000000   40.000000      47.100000
75%    2014.000000         2022.000000   88.000000     109.000000
max    2023.000000         2023.000000  424.000000     795.500000

         population         lat         lng  Ridership (millions)  \
count  2.050000e+02  205.000000  205.000000            140.000000
mean   6.776130e+06   31.989243   48.288138            161.934500
std    7.816394e+06   18.127545   73.158234            196.443906
min    3.538400e+04  -34.599700 -123.100000              0.400000
25%    1.683059e+06   25.286700    4.500000             36.500000
50%    3.519595e+06   35.689700   55.297200             93.250000
75%    8.911000e+06   42.318800  116.407500            209.207500
max    3.773200e+07   60.170800  151.210000            935.200000

       Ridership per capita
count            140.000000
mean              52.219741
std               61.779523
min                0.049944
25%               11.294210
50%               32.967561
75%               69.845015
max              380.850837
```

4.2 Correlaciones

[81]:
```
def nor(x):
    range_vals = max(x) - min(x)
    if not range_vals: return x
    return (x - min(x))/range_vals

def plot_corr(df, cols, annotbool=True, figsize=(5, 5)):
    corr = df[cols].corr()*100

    plt.figure(figsize=figsize)
    sns.heatmap(corr, annot=annotbool, mask=np.triu(corr), fmt=".1f",␣
 ↪cmap='RdBu', vmin=-100, vmax=100)
    return corr;
```

[82]:
```
df_norm = copy.deepcopy(df.loc[:,cols_num])
```

```
for col in cols_num:
    df_norm[col] = nor(df_norm[col])

corr = plot_corr(df_norm, cols_num);
```



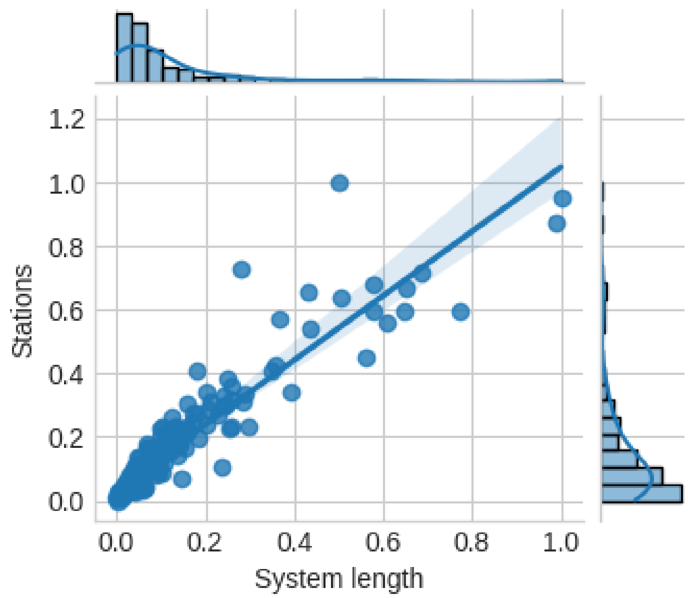
Como los violinplots ya dejaban entrever, el número de estaciones es fuertemente correlativo con la longitud del metro (93.1%).

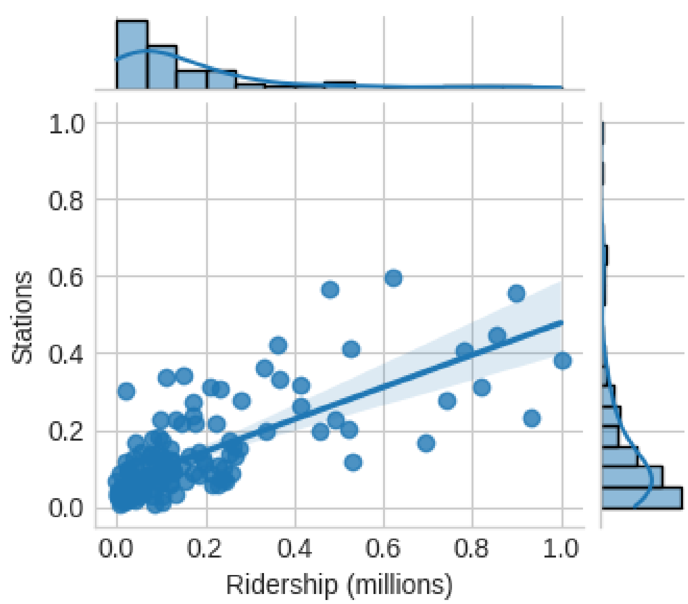
El número de viajeros (`Ridership (millions)`) se correlaciona con los anteriores dos campos al 73 y 67% respectivamnete.

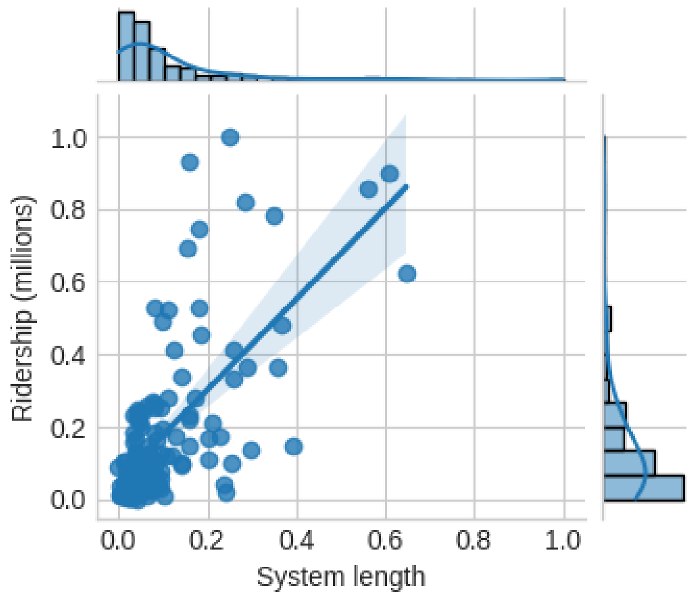
Estas tres correlaciones son lógicas: - a mas estaciones mas distancia total - a mas estaciones (y mayores distancias abarcables) pues mas viajeros se benefician del sistema ferroviario subterráneo

[83]:
```
for k in ['System length', 'Ridership (millions)']:
    g = sns.jointplot(data=df_norm, x=k, y='Stations', kind='reg')
```

```
    g.fig.set_size_inches((3.5, 3))
g = sns.jointplot(data=df_norm, x='System length', y='Ridership (millions)',␣
  ↪kind='reg')
g.fig.set_size_inches((3.5, 3));
```







El resto de variables numérica están poco correlacionadas. A resaltar el 42% entre la longitud geográfica `lng` y el año de inaguración, que como se grafica en siguiente apartado, denota la incipiente construcción de líneas de metro nuevas en China y el lejano oriente en general. A como ciudades grandes requieren mas estaciones y cobertura espacial (y viceversa en pequeñas), es decir, correlaciones superiores al 40% entre `population` y `estations` como entre `population` y `System length`.

Si eliminamos los outliers (aquellos que se alejan mas de 3 desviaciones estándar de la media) las regresiones anteriores (excepto una) no se darán tan fuertes, como se denota al rechequear las correlaciones:

[84]:
```
def remove_outliers(col, limit=3):
    mn = np.nanmean(col)
    out = limit * np.nanstd(col)
    mask = (col < (mn - out)) | (col > (mn + out))
    return col[~mask]
```

[85]:
```
df_norm_no_outliers = df_norm.copy()
df_norm_no_outliers = df_norm_no_outliers.apply(remove_outliers)
```

Los outliers no se han eliminado, sino reemplazados por NaN:

[86]:
```
df_norm.shape[0], df_norm_no_outliers.shape[0]
```

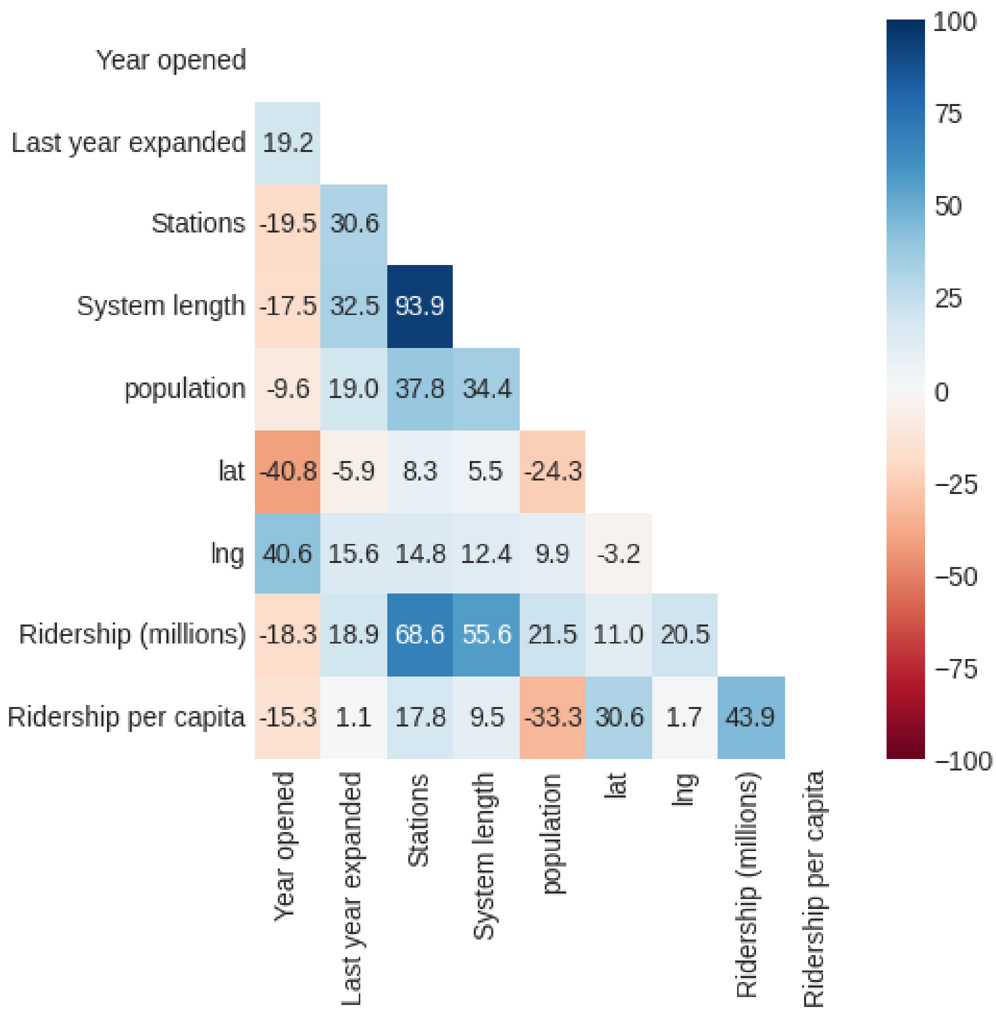
[86]:
```
(205, 205)
```

[87]:
```
df_check_nan_null(df_norm_no_outliers);
```

```
** NA **

Non NaN cols:
 Year opened              2
Last year expanded      26
Stations                 5
System length            6
population               5
lat                      6
Ridership (millions)    70
Ridership per capita    68
dtype: int64


** Blancos **

Non zero cols:
 Series([], dtype: int64)
```

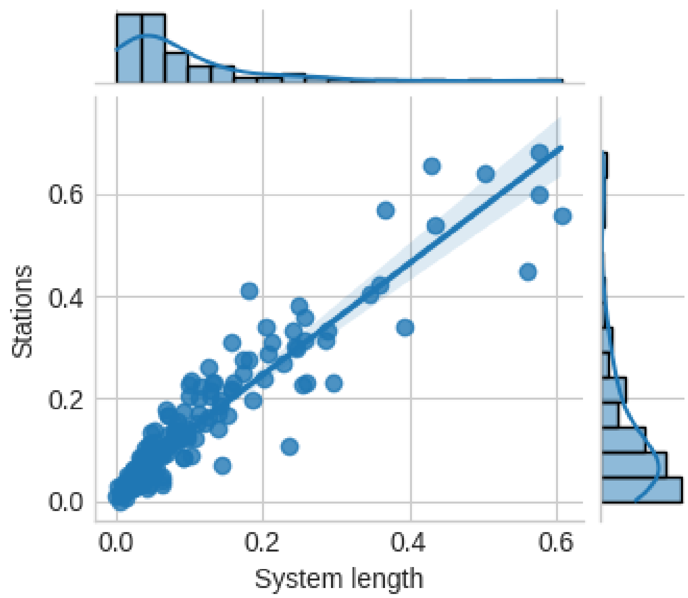
Las correlaciones sin considerar outliers son:

[88]:
```
corr_no_outliers = plot_corr(df_norm_no_outliers, cols_num);
```



Lo que reafirma la codependencia kilómetros-estaciones (94%):

[89]:
```
g = sns.jointplot(data=df_norm_no_outliers, x='System length', y='Stations',␣
    ↪kind='reg')
g.fig.set_size_inches((3.5, 3));
```

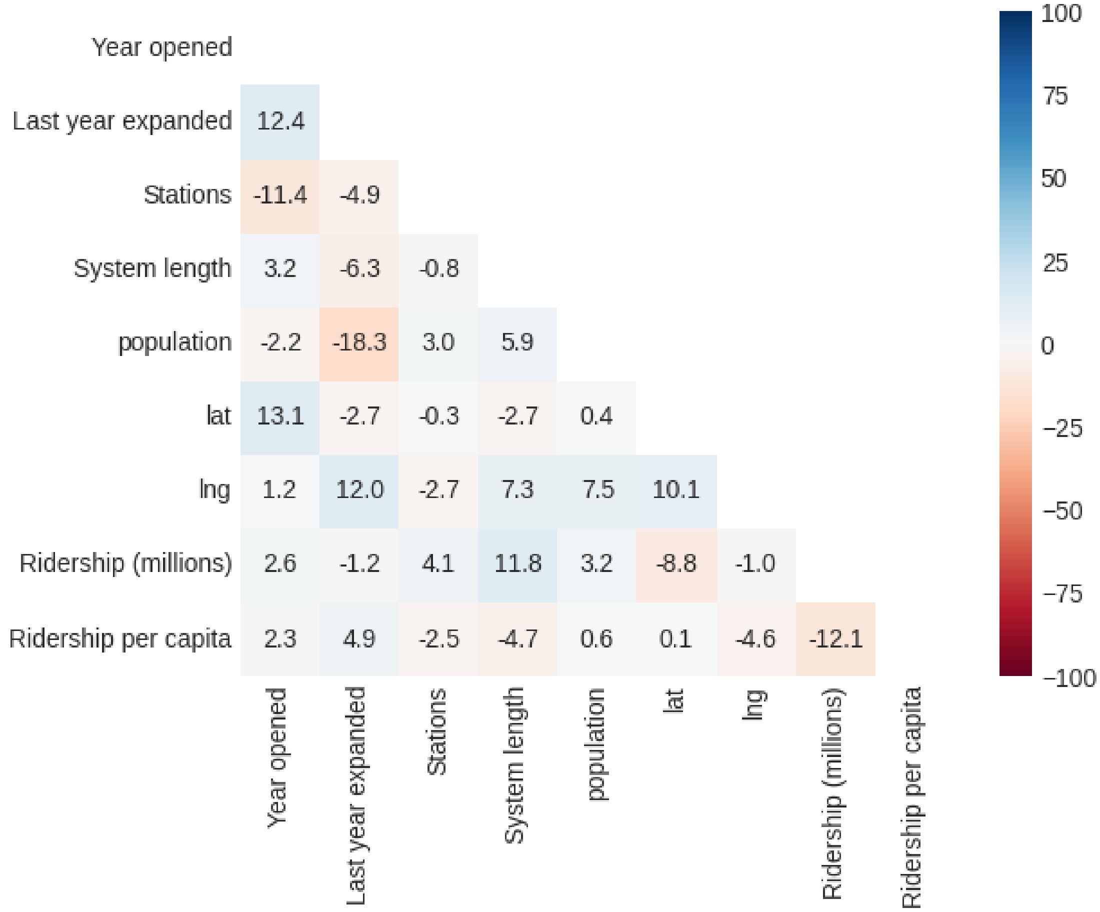


La diferencia entre tabla de correlaciones con y sin outliers es moderada.

[90]:
```
corr_diff = corr - corr_no_outliers
corr_diff.describe().loc[['mean', 'min', 'max']].applymap(lambda x: format(x, ".
↪1f"))
```

[90]:
```
     Year opened Last year expanded Stations System length population   lat  \
mean         2.4               -0.5     -1.7           1.5       -0.0   1.0
min        -11.4              -18.3    -11.4          -6.3      -18.3  -8.8
max         13.1               12.4      4.1          11.8        7.5  13.1

       lng Ridership (millions) Ridership per capita
mean   3.3                 -0.2                 -1.8
min   -4.6                -12.1                -12.1
max   12.0                 11.8                  4.9
```

[91]:
```
sns.heatmap(corr_diff, annot=True, mask=np.triu(corr), fmt=".1f", cmap='RdBu',␣
↪vmin=-100, vmax=100);
```



4.3 Regresión lineal

Para aplicar el modelo de regresión lineal se ha de trabajar con datasets sin NaN, pero los outliers pasaron a ser NaN, y con que fuese outlier en un campo ya he de eliminar la tupla entera:

[92]:
```
df_norm_no_outliers_no_nan = df_norm_no_outliers.dropna()
```

[93]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept=True)

y_col = 'System length'
X = df_norm_no_outliers_no_nan.drop(y_col, axis=1).values
y = np.array(df_norm_no_outliers_no_nan[y_col])[:, np.newaxis]

X.shape, y.shape
```

[93]:
```
((112, 8), (112, 1))
```

Aplico el modelo

[94]:
```
model.fit(X, y)
```

[94]:
```
LinearRegression()
```

Este modelo indica que la aproximación $\hat{y}$ al campo `System length` es tal que:

$$\hat{y} = -0.07 + 0.067 \cdot x_{\text{Last year expanded}} + 0.01 \cdot x_{\text{Year opened}} + 0.79 \cdot x_{\text{lng}} + 0.003 \cdot x_{\text{lat}} + 0.02 \cdot x_{\text{Ridership (millions)}} - 0.01 \cdot x_{\text{Stations}}$$

[95]:
```
pd.DataFrame(data=model.coef_, columns=list(set(cols_num).
 ↪difference(set([y_col]))))
```

[95]:
```
        lat  Year opened  Stations  population       lng  \
0  0.067262     0.010513  0.794723    0.002668  0.020939

   Ridership (millions)  Last year expanded  Ridership per capita
0              -0.01282            0.007279             -0.081639
```

[96]:
```
model.intercept_
```

[96]:
```
array([-0.07242396])
```

Con un coefficient de determinacion (R^2) de solo 93.5%, es decir, la regresión lineal es muy pobre para representar este campo, ó puede que dependa de otras varaibles no recogidas en nuestro dataset como PIB anual del país, inversión en infraestructuras per cápita, etc.

[97]:
```
import statsmodels.api as sm
results = sm.OLS(y, X).fit()
print(results.summary(),'\n')
# individual results parameters can be accessed
print('Parameters: ', results.params)
print('R2: ', results.rsquared)
```

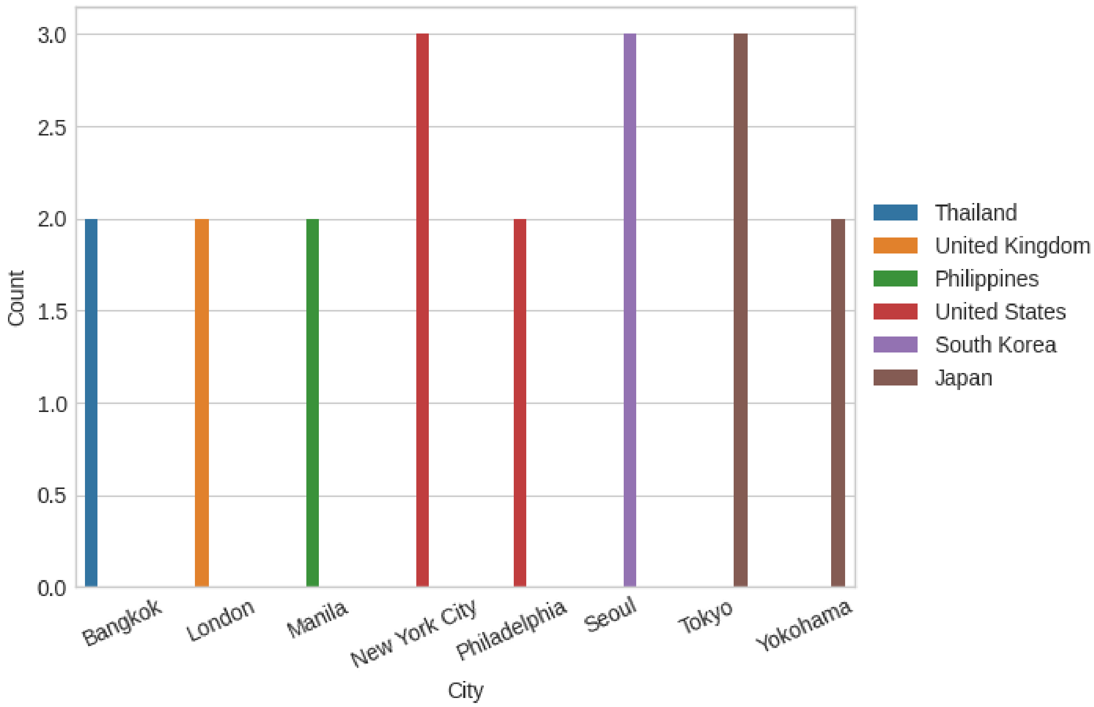
```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                      y   R-squared (uncentered):
0.935
Model:                            OLS   Adj. R-squared (uncentered):
0.930
Method:                 Least Squares   F-statistic:
187.1
Date:                Fri, 16 Jun 2023   Prob (F-statistic):
3.96e-58
Time:                        21:20:36   Log-Likelihood:
227.91
No. Observations:                 112   AIC:
```

```
-439.8
Df Residuals:                     104   BIC:
-418.1
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.0553      0.022      2.495      0.014       0.011       0.099
x2            -0.0482      0.029     -1.656      0.101      -0.106       0.010
x3             0.8075      0.044     18.402      0.000       0.720       0.895
x4            -0.0063      0.029     -0.221      0.825      -0.063       0.051
x5             0.0076      0.021      0.357      0.722      -0.035       0.050
x6            -0.0088      0.013     -0.668      0.505      -0.035       0.017
x7             0.0053      0.035      0.153      0.879      -0.063       0.074
x8            -0.0888      0.039     -2.293      0.024      -0.166      -0.012
==============================================================================
Omnibus:                       64.171   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              344.509
Skew:                           1.863   Prob(JB):                     1.55e-75
Kurtosis:                      10.742   Cond. No.                         29.3
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.

Parameters:  [ 0.055267   -0.0481955   0.80750771 -0.00634602  0.00760379
-0.00881157
  0.00527578 -0.08876115]
R2:  0.9350200760687406
```

4.4 Estudios diversos

Ciudades con mas de un sistema de metro:

[98]:
```
unique_cities = df.groupby(['City', 'country']).size().reset_index(name='Count')
mask_cities = unique_cities['Count'] > 1

ax = sns.barplot(data=unique_cities[mask_cities], x='City', y='Count',␣
 ↪hue='country')
ax.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.xticks(rotation=25);
```



Agrupamos datos de metros de una misma ciudad:

[99]:
```
# total number of stations and system length for each city-country pair
city_country_totals = df.groupby(['City', 'country']).agg({'Stations': 'sum',
                                                           'System length':␣
 ↪'sum',
                                                           'Year opened': 'min',
                                                           'Last year expanded':
 ↪ 'max',
                                                           'lat': 'mean',
                                                           'lng': 'mean',
                                                           'Ridership␣
 ↪(millions)': 'sum',
                                                           'Ridership per␣
 ↪capita': 'sum'}).reset_index()
```

Por ejemlo Nueva York tiene 3 redes de metro, con un total de 458 estaciones:

[100]:
```
df[df['City']=="New York City"]
```

[100]:
```
              City                   Name  Year opened  Last year expanded  \
194  New York City   New York City Subway         1904              2017.0
195  New York City  Staten Island Railway         1925              2017.0
196  New York City                   PATH         1908              1937.0
```

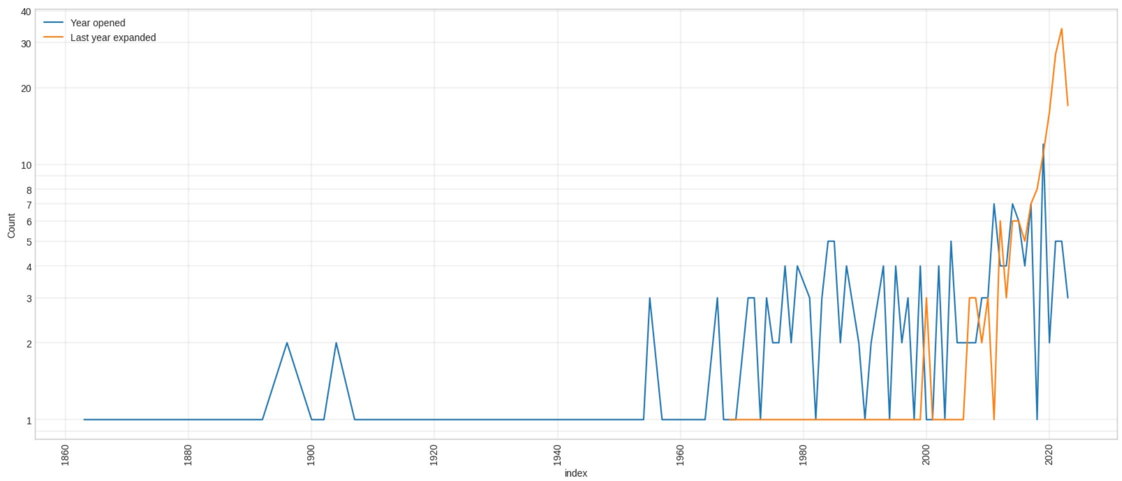
```
     Stations  System length        country admin_name  population      lat  \
194       424          399.0  United States   New York  18972871.0  40.6943
195        21           22.5  United States   New York  18972871.0  40.6943
196        13           22.2  United States   New York  18972871.0  40.6943

         lng  Ridership (millions) Ridership Year  Ridership per capita
194 -73.9249                   NaN            NaN                   NaN
195 -73.9249                   3.8           2022              0.200286
196 -73.9249                  45.5           2022              2.398161
```

[101]:
```python
city_country_totals[city_country_totals['City']=="New York City"]
```

[101]:
```
               City        country  Stations  System length  Year opened  \
120  New York City  United States       458          443.7         1904

     Last year expanded      lat      lng  Ridership (millions)  \
120              2017.0  40.6943 -73.9249                  49.3

     Ridership per capita
120              2.598447
```

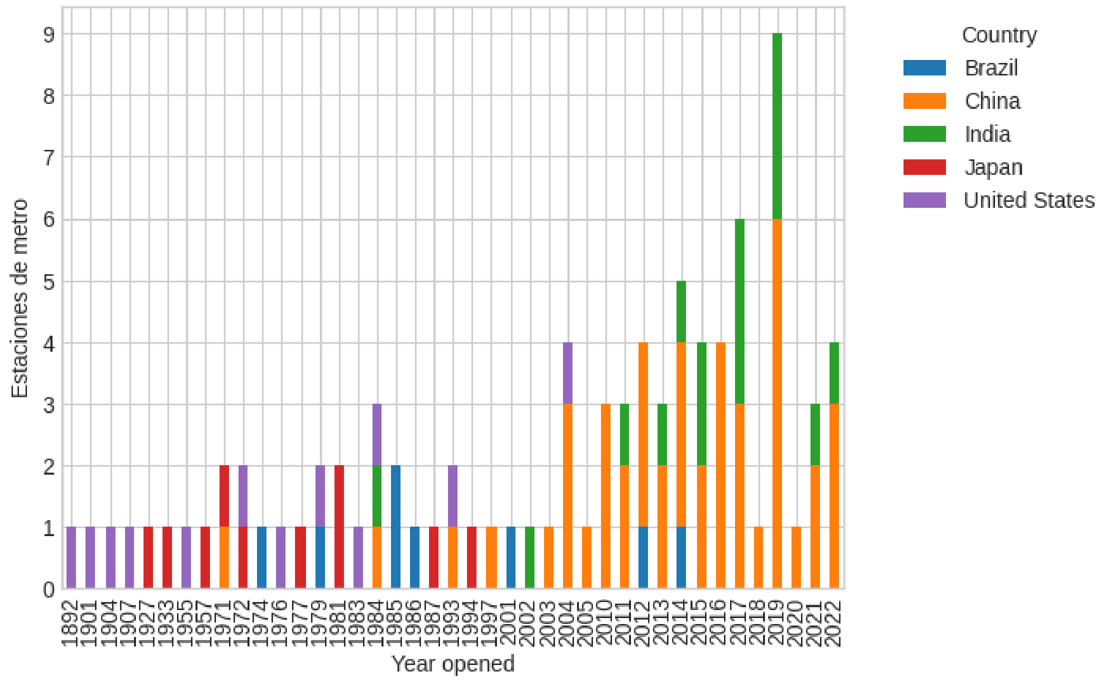
Redes de metro: cuantas se crean y cuando han sido por último expandidas.

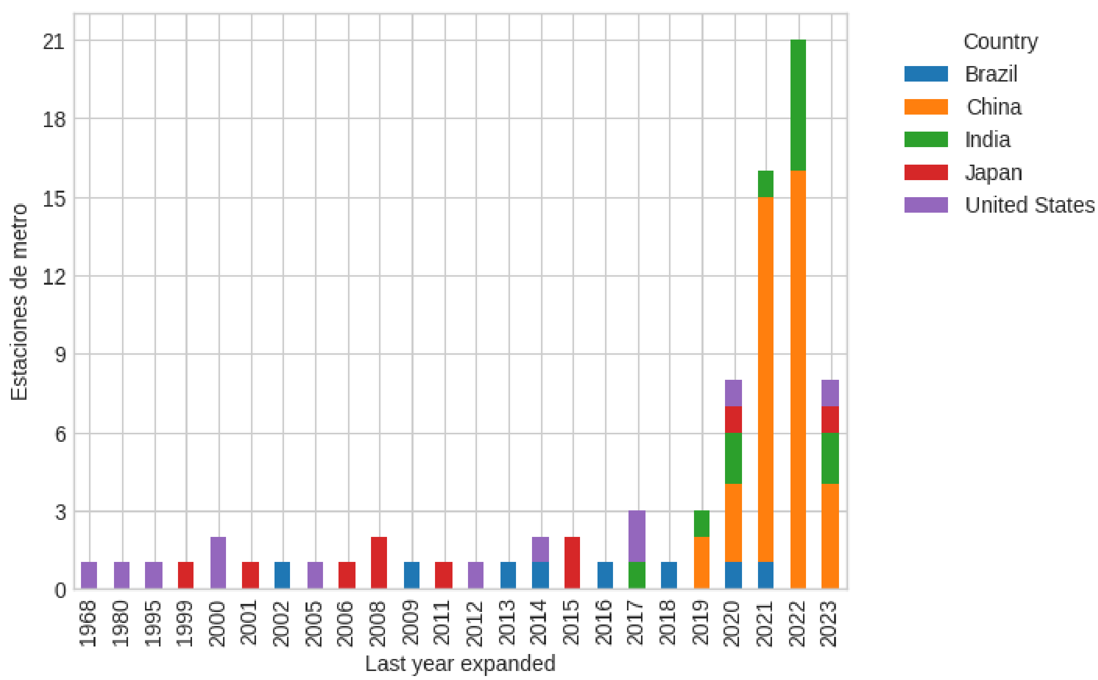
[102]:
```python
fig, ax = plt.subplots(figsize=(20, 8))
cols = ['Year opened', 'Last year expanded']
for col in cols:
    df_aux = city_country_totals[col].value_counts().reset_index(name='Count')

    l = sns.lineplot(data=df_aux, x='index', y='Count', markers=True,␣
 ↪dashes=False,ax=ax,label=col)
    l.set(yscale="log")

plt.xticks(rotation=90)
plt.grid(visible=True, which='both', color='black', linewidth=0.075)

# Customize y-ticks
ax.set_yscale("log")
ax.yaxis.set_major_locator(ticker.LogLocator(base=10, subs=range(1,9)))
ax.yaxis.set_major_formatter(ticker.ScalarFormatter())
```



Cantidad de redes de metro que se han abierto cada año (`Year opened`), pero solo de los 5 países con mas redes. Similar con `Last year expanded`.

[103]:
```python
def plot_year(df=city_country_totals, col='Year opened', top_n=5):

    # occurrences of each country
    country_counts = df['country'].value_counts()

    # top 5 countries
    top_countries = country_counts.head(top_n).index

    # filter
    filtered_df = df[df['country'].isin(top_countries)]

    # group the filtered DataFrame
    grouped = filtered_df.groupby([col, 'country']).size().unstack()

    # plot
    ax = grouped.plot(kind='bar', stacked=True)
    ax.yaxis.set_major_locator(ticker.MaxNLocator(integer=True))
    ax.set_xticklabels([int(float(label.get_text()))
                        if float(label.get_text()).is_integer()
                        else label.get_text()
                        for label in ax.get_xticklabels()])
    plt.xlabel(col)
    plt.ylabel('Estaciones de metro')
    plt.legend(title='Country', bbox_to_anchor=(1.05, 1), loc='best')
    plt.plot()

[plot_year(col=k) for k in ['Year opened', 'Last year expanded']];
```

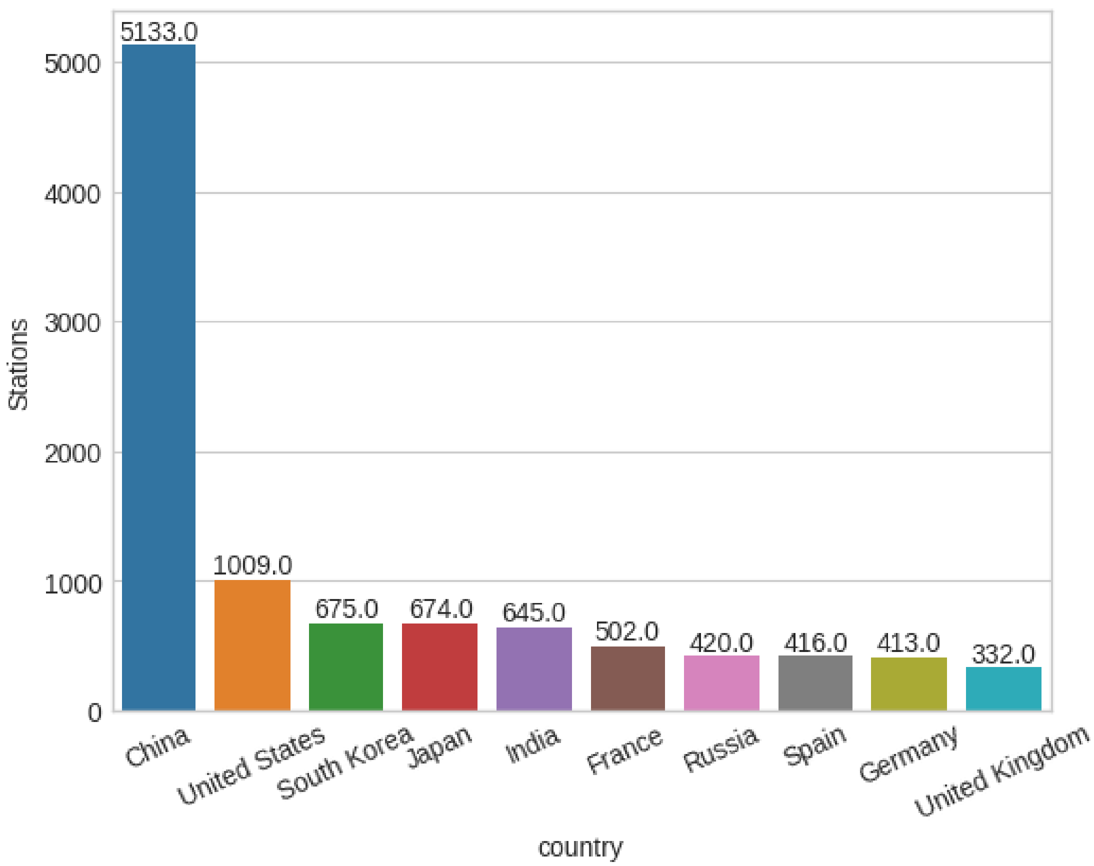




Países con mas estaciones:

[104]:
```
def df_count_values(df=city_country_totals, col='Stations', top_n=10,␣
↪mean=False):
    # stations for each country

    if not mean:
        df = df.groupby('country')[col].sum().reset_index()
    else:
        df = df.groupby('country')[col].mean().reset_index()

    # top 10 countries with the most stations
    top_countries = df.nlargest(top_n, col)

    ax = sns.barplot(data=top_countries, x='country', y=col)
    for container in ax.containers:
        ax.bar_label(container, fmt='%.1f');
    plt.xticks(rotation=25)

    return top_countries;
```

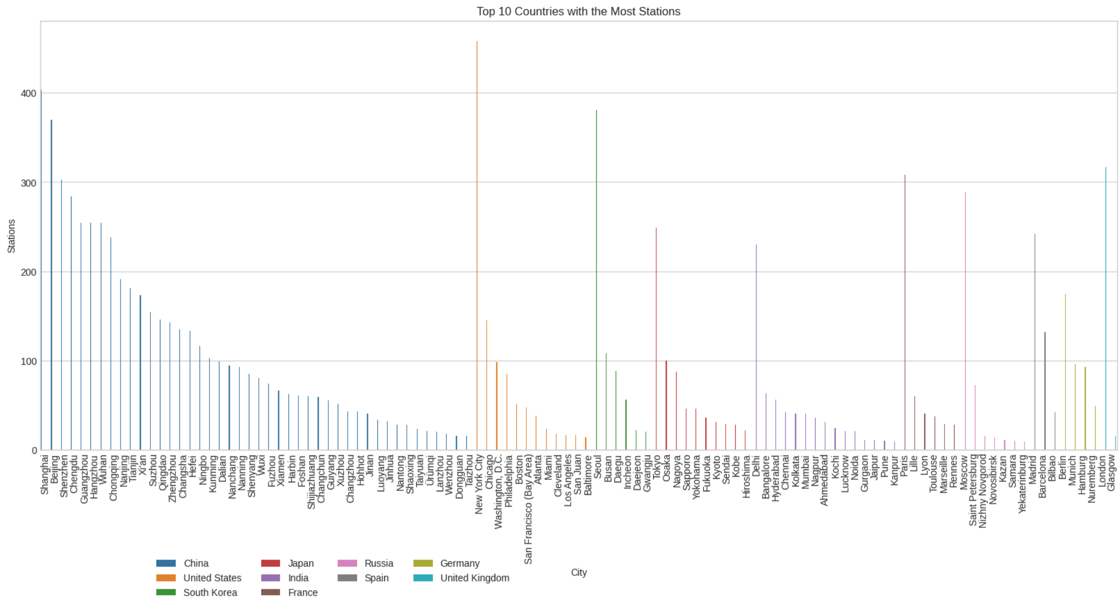
[105]:
```
top_countries = df_count_values();
```



En eje de abcisas situamos a la izquierda los países con mas estaciones, y dentro de cada país también ordenamos sus ciudades según el número de estaciones descendetes.

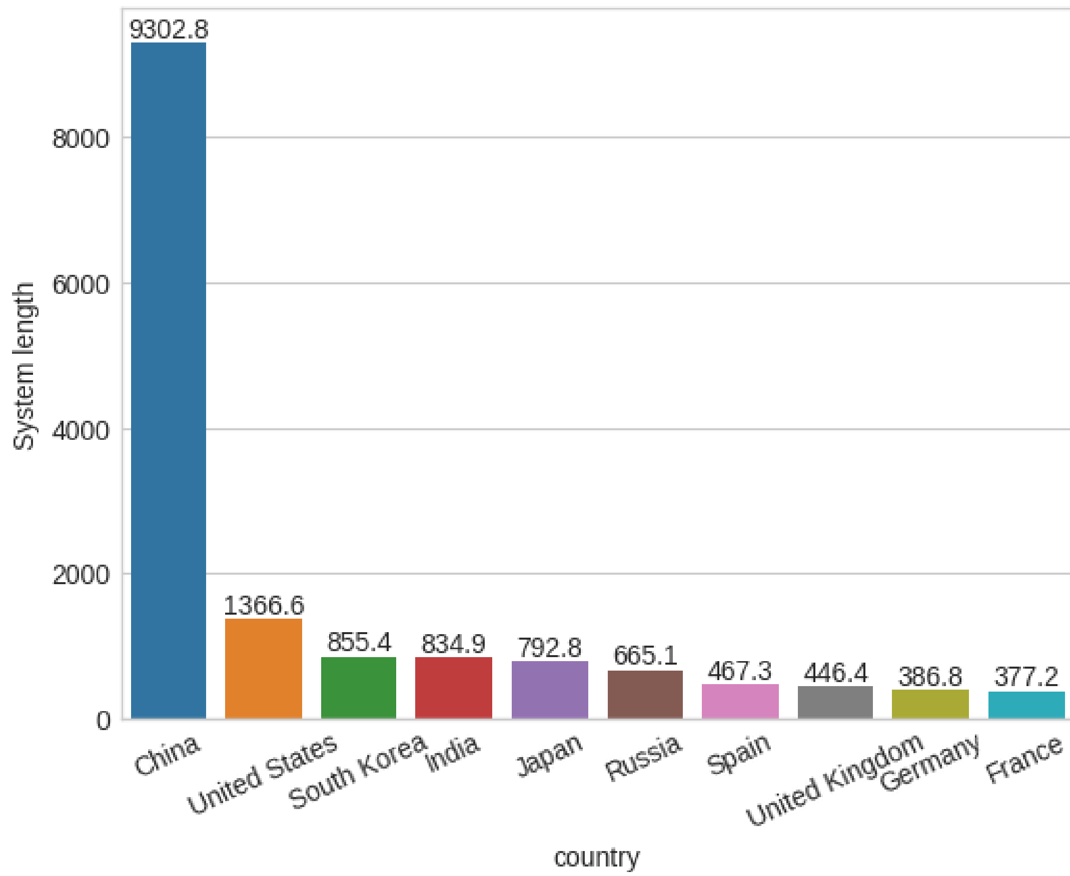
[106]:
```
def df_plot_each_city(df=city_country_totals, col='Stations',␣
↪top_countries=top_countries):

    # Filter the original DataFrame for the selected countries
    filtered_df = df[df['country'].isin(top_countries['country'])]

    # sort cities in each country
    filtered_df = filtered_df.sort_values(by=['country',col], ascending=[True,␣
↪False])

    # sort countries not unsorting its cities
    boolVar = True
    for k in top_countries['country']:
        df_k = filtered_df[filtered_df['country'] == k]
        if boolVar:
            df_aux = df_k.copy()
            boolVar = False
        else:
            df_aux = pd.concat([df_aux,df_k])

    # bar plot
    plt.figure(figsize=(20, 8))
    ax = sns.barplot(data=df_aux, x='City', y=col, hue='country')

    plt.xlabel('City')
    plt.ylabel(col)
    plt.title(f'Top 10 Countries with the Most {col}')
    plt.legend(title='Country')
    ax.legend(loc='center left', bbox_to_anchor=(0.1, -0.3),fancybox=True,␣
↪shadow=True, ncol=4)
    plt.xticks(rotation=90);
```

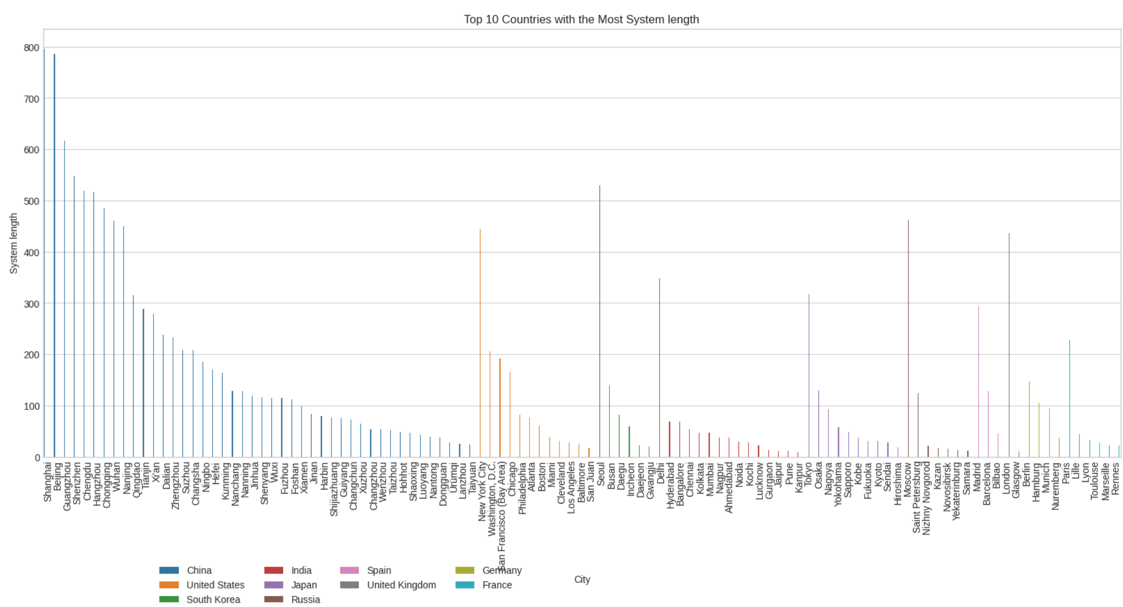
[107]:
```
df_plot_each_city()
```



Países con mas kilómetros. Francia es la sexta con mas estaciones, pero la décima en kilómetros:

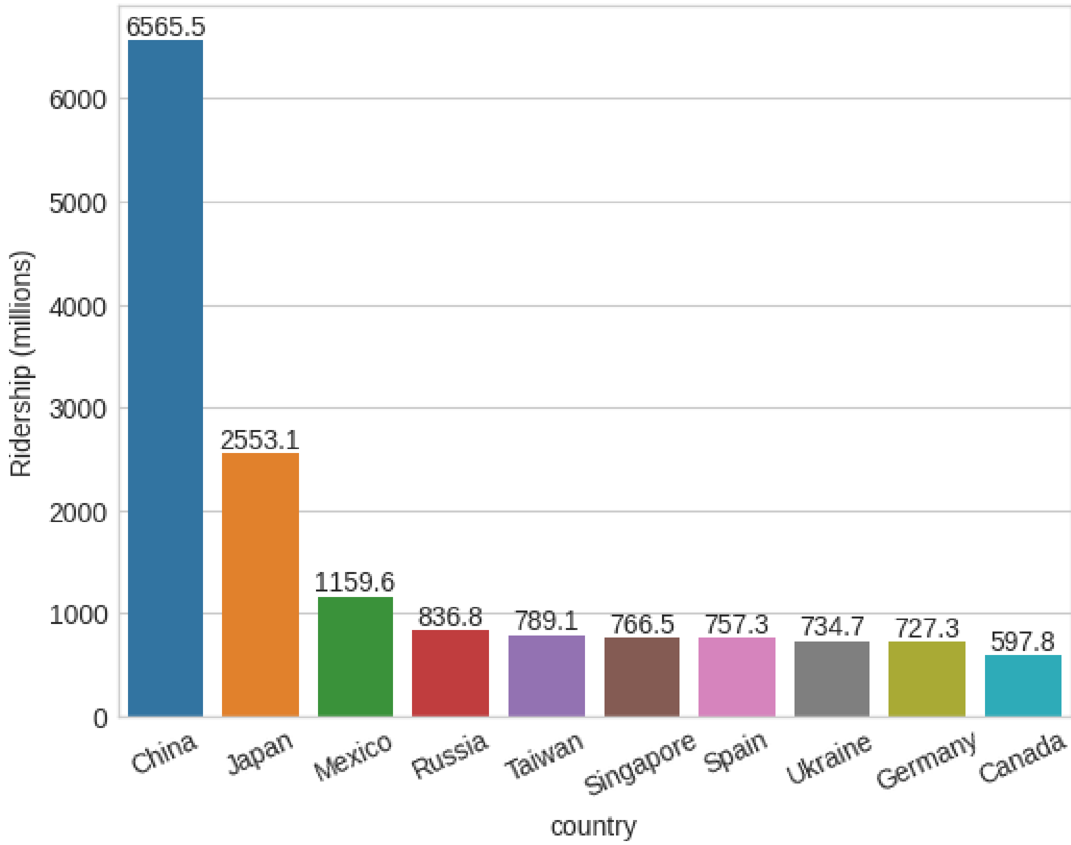
[108]:
```
col="System length"
top_countries = df_count_values(col=col);
df_plot_each_city(col=col, top_countries=top_countries);
```

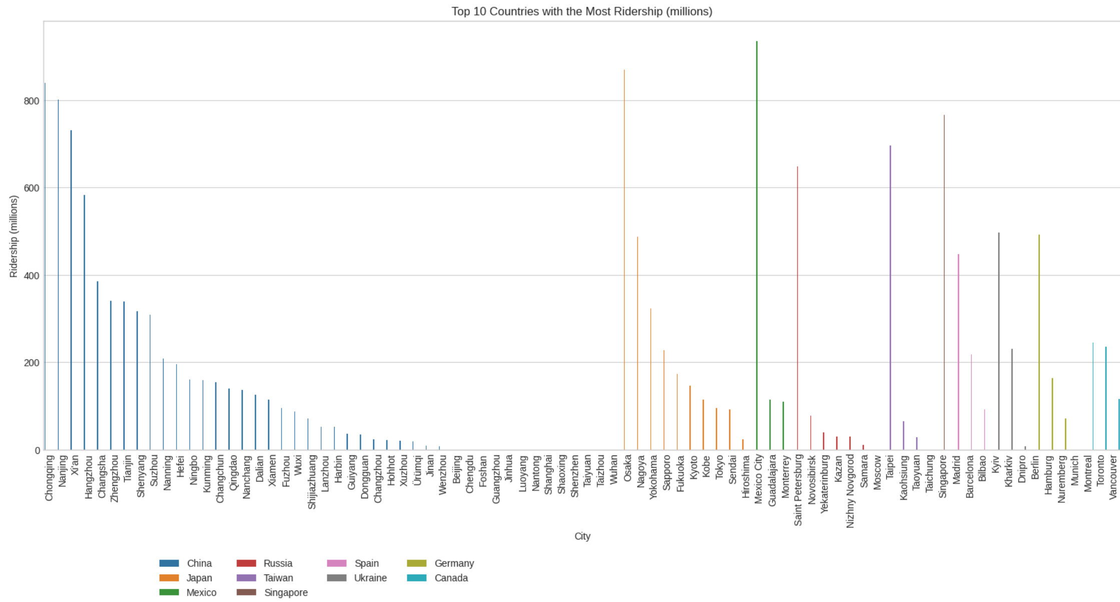

9302.8
1366.6
855.4
834.9
792.8
665.1
467.3
446.4
386.8
377.2
8000
6000
4000
2000
0
System length
China
United States
South Korea
India
Japan
Russia
Spain
United Kingdom
Germany
France
country


Top 10 Countries with the Most System length
System length
800
700
600
500
400
300
200
100
0
City
China
United States
South Korea
India
Japan
Russia
Spain
United Kingdom
Germany
France

Países con mas viajeros (en el año recogido en la base de datos):

[109]:
```
col="Ridership (millions)"
top_countries = df_count_values(col=col);
df_plot_each_city(col=col, top_countries=top_countries);
```



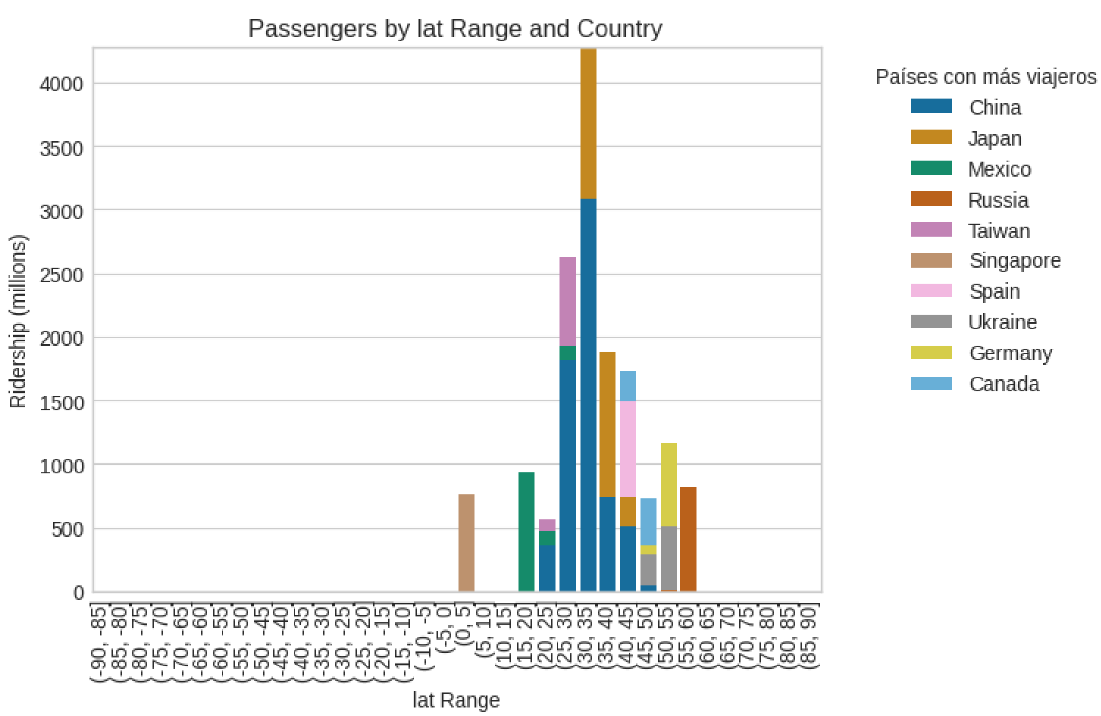


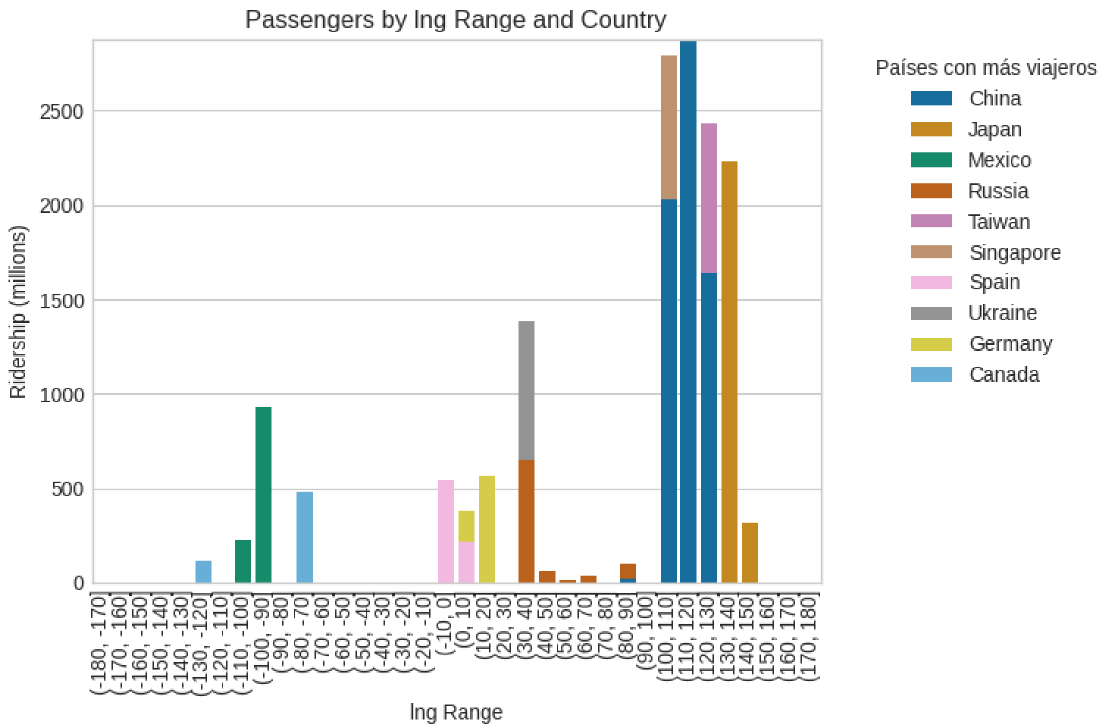
Distribución de pasajeros por latitud, después por longitud. Solo países con mas viajeros. Con estos 2 gráficas y nuestra imaginación podemos comprender como se distribuye la cuantía de viajeros respecto a Greenwich y respecto al ecuador.

Líneas de latitud oscilan entre -90 y +90 grados, las coordenadas de longitud están entre -180 y +180 grados.

[110]:
```
def plot_lat_lng(df=city_country_totals, col='lat',␣
↪top_countries=top_countries):
    # Define the latitude/longitude ranges for binning
    if col=='lat':
        latlng_ranges = np.arange(-90, 91, 5)
    else:
        latlng_ranges = np.arange(-180, 181, 10)


    # Filter the original DataFrame for the selected countries
    df = df[df['country'].isin(top_countries['country'])]

    # Set up subplots
    fig, ax = plt.subplots()

    # Initialize empty array for stacked heights
    stacked_heights = np.zeros(len(latlng_ranges)-1)

    # Define a color palette for each country
    colors = sns.color_palette('colorblind', len(top_countries))
```

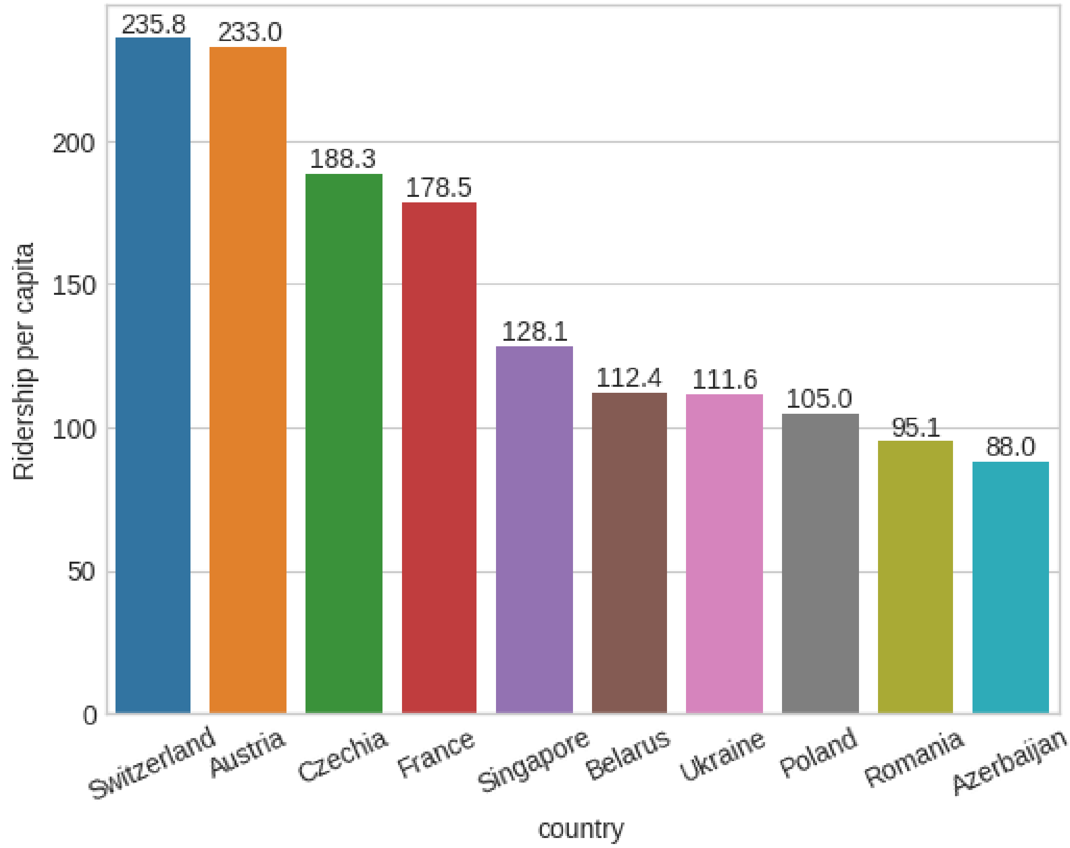
```
    # Iterate over countries and plot stacked bar plots
    for i, country in enumerate(top_countries['country']):
        data = df[df['country'] == country][[col, 'Ridership (millions)']]
        data[f'{col}_range'] = pd.cut(data[col], latlng_ranges)  # Bin the latitude values into ranges
        grouped_data = data.groupby(f'{col}_range')['Ridership (millions)'].sum()  # Group by lat_range and sum the ridership
        grouped_data = pd.DataFrame(grouped_data)

        sns.barplot(data=grouped_data, x=grouped_data.index, y='Ridership (millions)',
                    ax=ax, bottom=stacked_heights, label=country, color=colors[i])
        stacked_heights += grouped_data['Ridership (millions)'].values

    # Customize plot
    plt.xlabel(f'{col} Range')
    plt.ylabel('Ridership (millions)')
    plt.title(f'Passengers by {col} Range and Country')
    plt.legend(title='Países con más viajeros', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xticks(rotation=90)
    plt.show()


[plot_lat_lng(col=k) for k in ['lat', 'lng']];
```

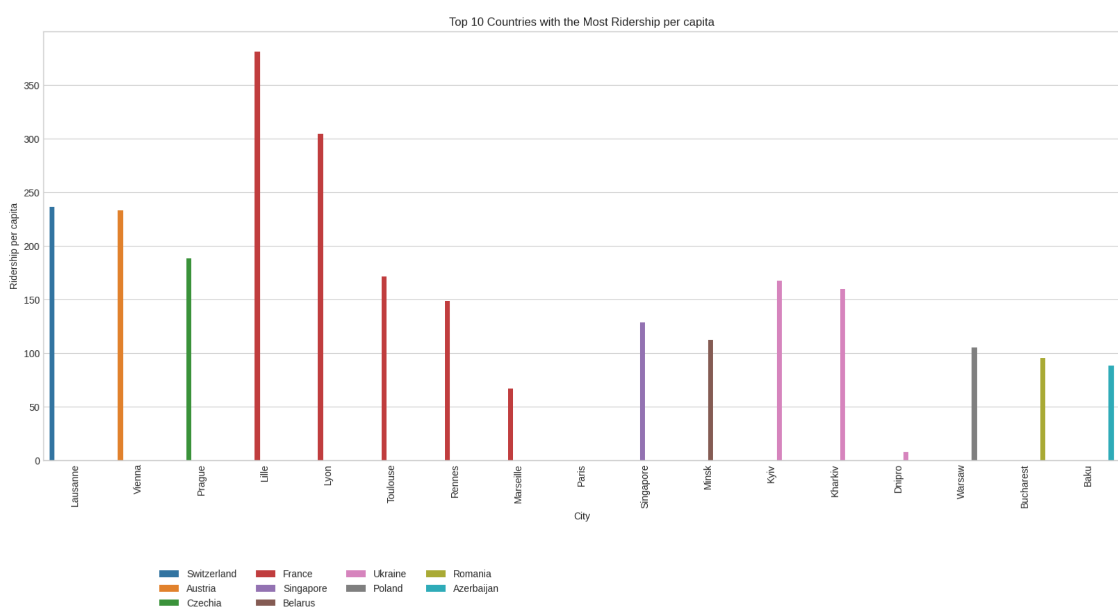




Países con mas viajeros per capita (en el año recogido en la base de datos).

[111]:
```
col="Ridership per capita"
top_countries = df_count_values(col=col, mean=True);
df_plot_each_city(col=col, top_countries=top_countries);
```

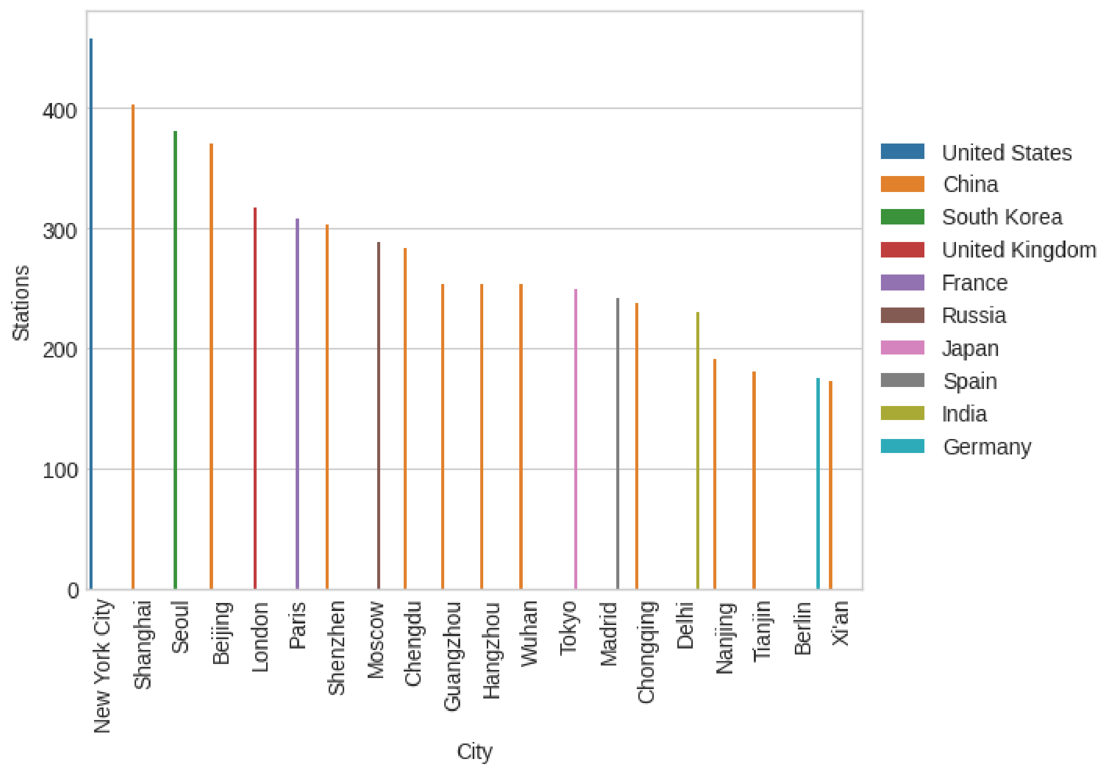

235.8
233.0
188.3
178.5
128.1
112.4
111.6
105.0
95.1
88.0
Ridership per capita
0
50
100
150
200
Switzerland
Austria
Czechia
France
Singapore
Belarus
Ukraine
Poland
Romania
Azerbaijan
country


Top 10 Countries with the Most Ridership per capita
Ridership per capita
0
50
100
150
200
250
300
350
Lausanne
Vienna
Prague
Lille
Lyon
Toulouse
Rennes
Marseille
Paris
Singapore
Minsk
Kyiv
Kharkiv
Dnipro
Warsaw
Bucharest
Baku
City
Switzerland
Austria
Czechia
France
Singapore
Belarus
Ukraine
Poland
Romania
Azerbaijan

Ciudades con mas estaciones:

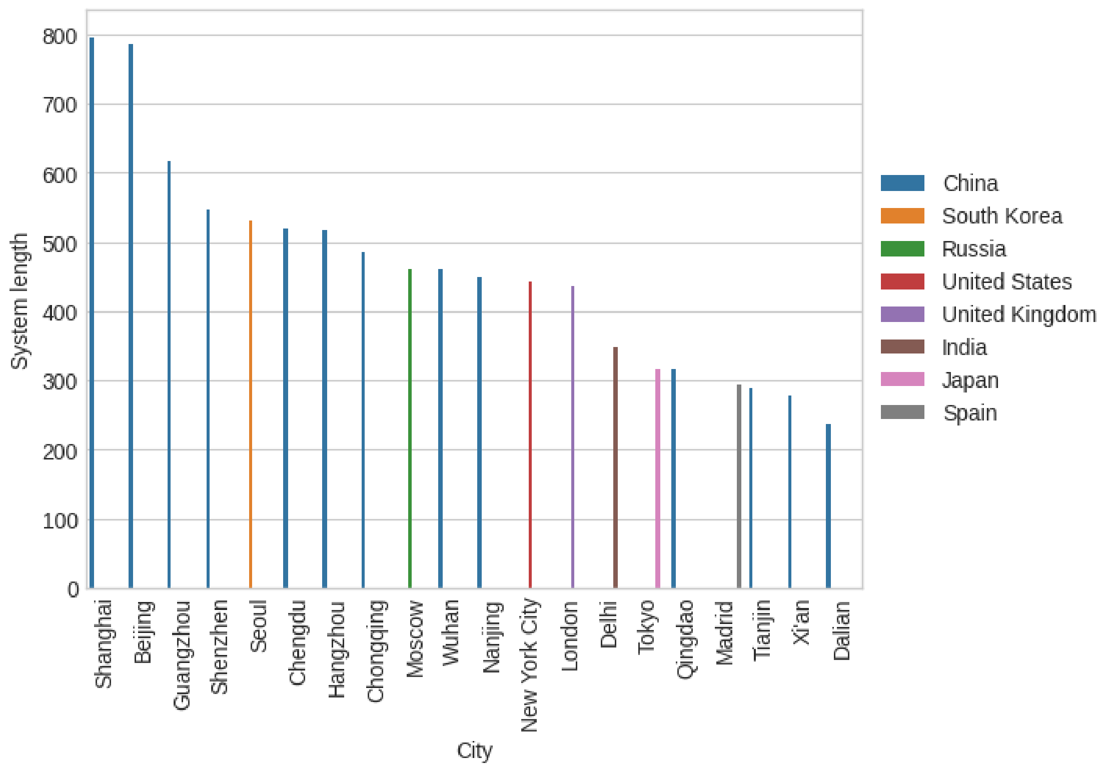
[112]:
```python
def plot_top_cities(df=df, col='Stations'):
    top_cities = df.groupby(['City', 'country'])[col].sum().nlargest(20).
 ↪reset_index(name=col)
    ax = sns.barplot(data=top_cities, x='City', y=col, hue='country')
    ax.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
    plt.xticks(rotation=90);
```

[113]:
```python
plot_top_cities()
```



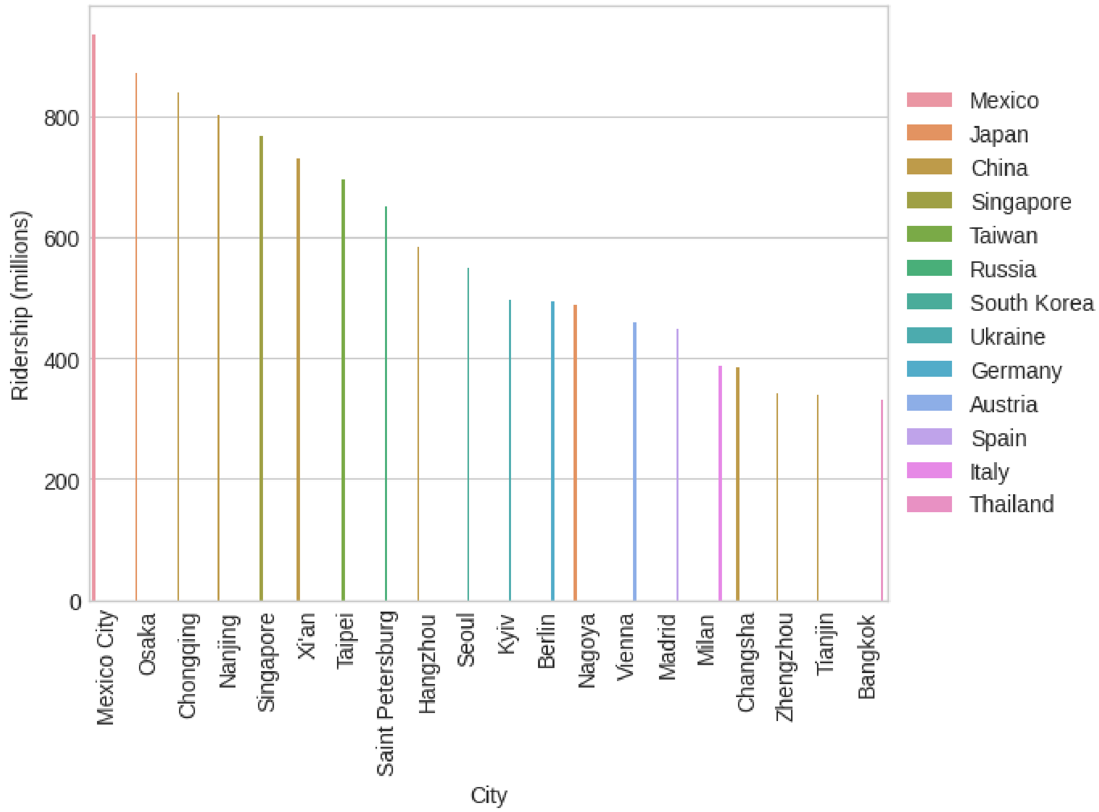
Ciudades con mas kilómetros:

[114]:
```python
plot_top_cities(col="System length")
```



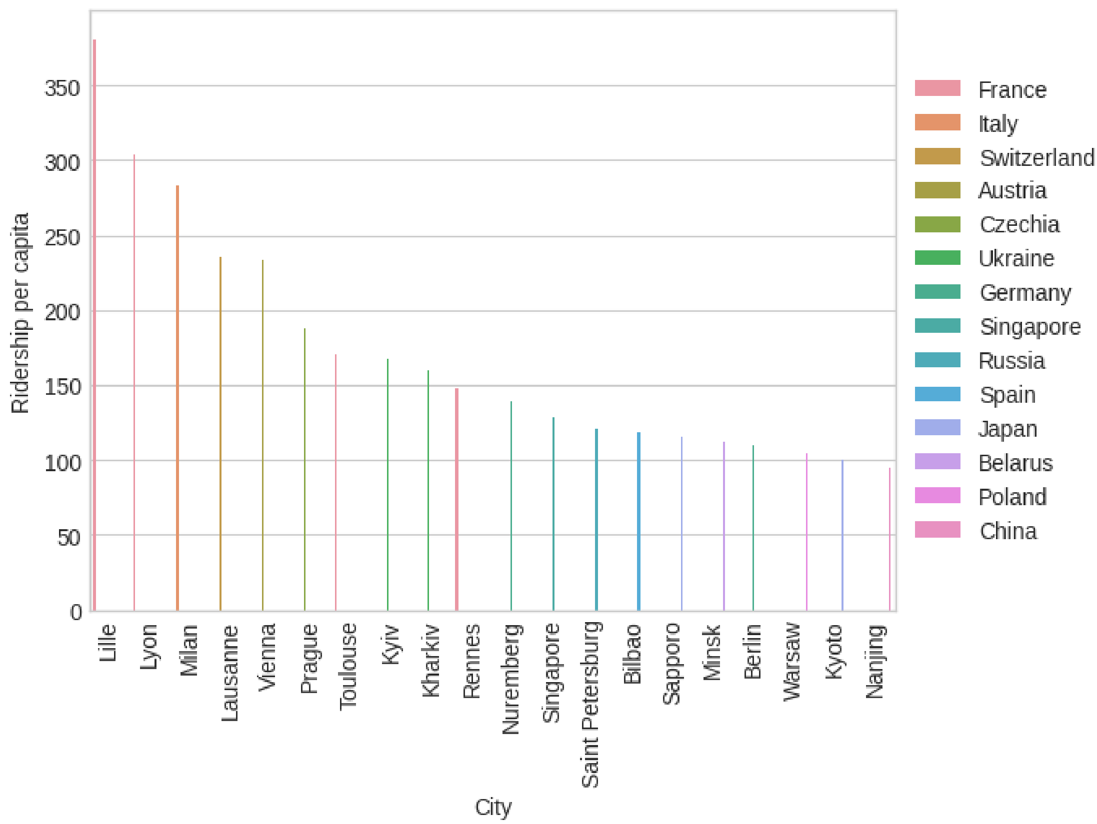
Ciudades con mas viajeros por año (en el año recogido en la base de datos):

[115]:
```
plot_top_cities(col="Ridership (millions)")
```



Ciudades con mas viajeros per capita (en el año recogido en la base de datos).

[116]:
```
plot_top_cities(col="Ridership per capita")
```



5 Resolución del problema y conclusiones.

La distribución de campos numéricos (violinplots) resumen muy visualmente la base de datos: no muchos datos (solo 205 registros) pero muy variados.

Los campos han sido preprocesados correctamente, restando solo 16 registros de viajeros no indicados. Ya que los NaN de la variable `Last year expanded` simplemente indican que hay 22 metros sin ampliaciones.

Los heatmaps de correlaciones con ó sin outliers indican una alta falta de correlación excepto entre estas 3 parejas de campos: `System length-Stations`, `System length-Ridership (millions)` y `Stations-Ridership (millions)`. Precisamente por tener esta ventana se ha elegido la variable `System length` para su estudio de predición como regresión lineal en función del resto de variables numéricas. El moderado coef. de determinación del 93% no es suficientemente alto como para considerar el modelo satisfactorio. Si se desease predecir esa variable se recomienda: - recurrir a modelos mas complejos como árboles de decisión dónde también participen variables categóricas como el país - aumentar el número de registro buscando en internet mas lejos del dataset de kaggle si hay mas metros a día de hoy - recurrir a otras bases de datos con datos relativos a la ciudad/país como PIB anual del país, inversión en infraestructuras per cápita, etc. ó al metro como coste de mantenimineto, personal operario, grado de satisfación de los viajeros,...

Por otro lado, los múltiples gráficos de barras posteriores a la regresión sí nos han aportado muy valiosa información: - China está acelerando la construcción de líneas de metro desde el principio de siglo (la India potente también pero desde 2011), es donde mas se usa en términos absolutos, y desde 2019 es donde mas líneas se amplían - los países con mas estaciones, kilómetros de vías y viajeros siguen un patrón común de sus redes de metro, donde una megaurbe (Nueva York, Seul, Tokyo, Dehli,…) puntua claramente en cada uno de estos aspectos al resto de urbes de su país. Este sorpaso es mínimo el doble, llegando en algunas situaciones a ser triple. Después, el resto de metros del país (ciudades 3º, 4º, etc.) reducen sus cuantías de los 3 citados campos ya de una manera mas progresiva. Ésto no sucede en China (lider absoluto de estos 3 parámetros) donde Shangai, Pekín, Shenzhen ó Chengdu compiten mas a la par por la supremacía de este servicio - en cuanto al uso per cápita (pasajeros anuales entre población urbana) destacan países fuera de top absolutos como Suiza (con Lausanne), Austria (con Viena) y Rep. Checa (con Praga) - si analizamos las líneas sin prefiltrar que tengan que pertenecer a la cima en el aspecto en estudio, se observa que: - de las 20 con mas estaciones solo 5 son europeas: 5º Londres, 6º París, 8º Moscú, 14º Madrid y 19º Berlín - de las 20 mas largas solo 3 son europeas: 9º Moscú, 13º Londrés y 17º Madrid - de las 20 mas usadas solo 5 son europeas: 11º Kiev, 12º Berlín, 14º Viena, 15º Madrid, 16º Milán (aunque es un dato a revisar, ya que los datos pueden ser muy antiguos, ó por ejemplo no se han registrado los viajeros de Pekín) - de las 20 mas usadas per cápita 16 son europeas, con especial asiduididaz se suben al metros los ciudadanos de Lille (Francia). Esta gráfica ha de tener las mismas consideraciones que la anterior.

Conclusiones: - el lejano oriente (y la India-Irán) ya es lider en transporte suburbano y su tendencia es fuertemente creciente - si extrapolamos datos del metro: en Estados Unidos y Europa suele haber unas pocas ciudades por país donde se concentra el desarollo, mientras en los dragones asiáticos no es tan pronunciado el contraste - Europa lidera las ciudades donde cada habitante mas usa el metro

6 Exportación del código y de los datos producidos

Guardamos el DataFrame mergeado y limpiado (pero no normalizado) en un archivo CSV.

[117]:
```
file_output = "../dataset/df_metro_cities.csv"
df.to_csv(file_output, index=False)
```

Lo cargamos para comprobar que se guardó satisfactoriamente.

[118]:
```
pd.read_csv(file_output).tail(3)
```

[118]:
```
         City           Name  Year opened  Last year expanded  Stations  \
202  Tashkent  Tashkent Metro         1977              2023.0        48
203   Caracas   Caracas Metro         1983              2015.0        49
204     Hanoi     Hanoi Metro         2021                 NaN        12

     System length     country        admin_name  population      lat  \
202           59.1  Uzbekistan          Toshkent   2571668.0  41.3111
203           67.2   Venezuela  Distrito Capital   2245744.0  10.4806
204           13.1     Vietnam            Hà Nội   8246600.0  21.0283

          lng  Ridership (millions)  Ridership Year  Ridership per capita
```

```
202   69.2797                  136.7        2022.0             53.156162
203  -66.9036                    NaN           NaN                   NaN
204  105.8542                    NaN           NaN                   NaN
```

[ ]: