data_cleaning_and_feature_selection

August 4, 2025

1 Introduction

1.1 Improting required libraries

[65]:
```python
# Core libraries for data manipulation
import numpy as np             # For numerical operations
import pandas as pd            # For structured data (DataFrame) manipulation

# Regular expressions for pattern matching
import re

# Visualization libraries
import matplotlib.pyplot as plt   # For basic plotting
import seaborn as sns             # For statistical plots and visual styles

# Dask for out-of-core and parallel data processing (large datasets)
import dask.dataframe as dd

# Glob for file path matching (e.g., loading multiple CSV files at once)
import glob
```

1.2 Code to check and review each CSV

[66]:
```python
def eval_df(dataframe):
    """
    Evaluate the structure and quality of a pandas DataFrame.

    This function prints:
    - Data types and memory usage
    - Columns with missing values
    - Count of duplicate rows
    - Summary statistics (numeric and categorical)

    Parameters:
    -----------
    dataframe : pd.DataFrame
        The DataFrame to evaluate.
    """
```

```python
    # Display data types and basic memory usage
    print("\n DATA TYPES & MEMORY USAGE")
    print("-" * 40)
    print(dataframe.info())

    # Check and display missing values per column
    print("\n MISSING VALUES PER COLUMN")
    print("-" * 40)
    missing_values = dataframe.isnull().sum()
    print(missing_values[missing_values > 0])

    # Check for duplicate rows
    duplicates = dataframe.duplicated().sum()
    print("\n DUPLICATE ROWS FOUND")
    print("-" * 40)
    print(f"{duplicates} duplicate rows found.")

    # Display summary statistics (for both numeric and object types)
    print("\n SUMMARY STATISTICS")
    print("-" * 40)
    print(dataframe.describe())
```

1.3 Get the sample countries

- this would include those that have mobility data, policy data, covid data and country statistics data

[67]:
```python
# To get a common code for the countries I would use the ISO 3166-1 alpha-3
# Load country and country code reference data
COUNTRY_CODE = pd.read_csv("data/country_codes.csv")

# Preview the first few rows
COUNTRY_CODE.head()
```

[67]:

	id	alpha2	alpha3	en
0	4	af	afg	Afghanistan
1	8	al	alb	Albania
2	12	dz	dza	Algeria
3	20	ad	and	Andorra
4	24	ao	ago	Angola

[68]:
```python
len(COUNTRY_CODE)
```

[68]: 193

[69]:
```python
# Capitalise all codes in alpha2 and alpha3
COUNTRY_CODE['alpha2'] = COUNTRY_CODE['alpha2'].str.upper()
```

```
COUNTRY_CODE['alpha3'] = COUNTRY_CODE['alpha3'].str.upper()
```

[70]:
```
COUNTRY_CODE = COUNTRY_CODE.rename(columns = {
    'alpha2': 'code2',
    'alpha3' : 'Code',
    'en' : 'Country'
})
```

1.3.1 Get the mobility data

[71]:
```
# Columns to include (mobility trends + region info + date)
usecols = [
    "country_region_code",
    "country_region",
    "sub_region_1",
    "sub_region_2",
    "metro_area",
    "date",
    "retail_and_recreation_percent_change_from_baseline",
    "grocery_and_pharmacy_percent_change_from_baseline",
    "parks_percent_change_from_baseline",
    "transit_stations_percent_change_from_baseline",
    "workplaces_percent_change_from_baseline",
    "residential_percent_change_from_baseline"
]

# Explicit dtypes to avoid Dask inference issues
dtype_fix = {
    'sub_region_1': 'object',
    'sub_region_2': 'object',
    'metro_area': 'object',
}

# Load the dataset with parsing and type fixes
df = dd.read_csv(
    'data/global_mobility_report.csv',
    usecols=usecols,
    dtype=dtype_fix,
    parse_dates=['date'],
    assume_missing=True
)

# Filter: only country-level data + dates up to the end of 2022
missing_cols = ['sub_region_1', 'sub_region_2', 'metro_area']
filtered = df[
    df[missing_cols].isnull().all(axis=1) &
    (df['date'] <= '2022-12-31')
```

```python
]

# Safely preview the filtered result
print(filtered.sample(frac = 0.0001, random_state = 1).compute())
```

```
       country_region_code country_region sub_region_1 sub_region_2  \
18274                   AR      Argentina         <NA>         <NA>
456350                  BJ          Benin         <NA>         <NA>
109303                  CL          Chile         <NA>         <NA>
497226                  FR         France         <NA>         <NA>
19372                   GT      Guatemala         <NA>         <NA>
160328                  LI  Liechtenstein         <NA>         <NA>
91450                   KG     Kyrgyzstan         <NA>         <NA>
268085                  ML           Mali         <NA>         <NA>
480291                  NP          Nepal         <NA>         <NA>
533080                  RW         Rwanda         <NA>         <NA>
450650                  TJ     Tajikistan         <NA>         <NA>
532980                  US  United States         <NA>         <NA>
410679                  UY        Uruguay         <NA>         <NA>

       metro_area       date  \
18274        <NA> 2020-11-28
456350       <NA> 2022-02-22
109303       <NA> 2022-03-22
497226       <NA> 2020-08-08
19372        <NA> 2021-08-02
160328       <NA> 2021-06-06
91450        <NA> 2021-07-08
268085       <NA> 2021-10-05
480291       <NA> 2021-07-08
533080       <NA> 2020-08-13
450650       <NA> 2020-04-29
532980       <NA> 2021-01-03
410679       <NA> 2021-10-12

        retail_and_recreation_percent_change_from_baseline  \
18274                                               -44.0
456350                                               38.0
109303                                               -6.0
497226                                              -22.0
19372                                                -6.0
160328                                                NaN
91450                                               -19.0
268085                                               43.0
480291                                              -11.0
533080                                              -14.0
450650                                              -32.0
```

```
532980                                            -27.0
410679                                            -15.0

        grocery_and_pharmacy_percent_change_from_baseline  \
18274                                             -11.0
456350                                            113.0
109303                                             19.0
497226                                             -7.0
19372                                              17.0
160328                                              NaN
91450                                              -4.0
268085                                             86.0
480291                                             33.0
533080                                            -12.0
450650                                            -20.0
532980                                            -16.0
410679                                              4.0

        parks_percent_change_from_baseline  \
18274                                -65.0
456350                               156.0
109303                               -11.0
497226                               155.0
19372                                 -9.0
160328                                 NaN
91450                                 -4.0
268085                               114.0
480291                                 3.0
533080                                 3.0
450650                               -22.0
532980                               -24.0
410679                               -39.0

        transit_stations_percent_change_from_baseline  \
18274                                           -38.0
456350                                           -4.0
109303                                           13.0
497226                                            2.0
19372                                           -21.0
160328                                          -52.0
91450                                            -5.0
268085                                           62.0
480291                                           -3.0
533080                                           -9.0
450650                                          -37.0
532980                                          -33.0
410679                                            0.0
```

```
        workplaces_percent_change_from_baseline  \
18274                                      -5.0
456350                                     35.0
109303                                     32.0
497226                                    -14.0
19372                                     -18.0
160328                                      NaN
91450                                     -44.0
268085                                      3.0
480291                                    -21.0
533080                                    -25.0
450650                                    -12.0
532980                                    -17.0
410679                                     23.0

        residential_percent_change_from_baseline
18274                                       11.0
456350                                       6.0
109303                                       5.0
497226                                      -2.0
19372                                        7.0
160328                                       NaN
91450                                       -6.0
268085                                     -11.0
480291                                       8.0
533080                                       9.0
450650                                       8.0
532980                                       7.0
410679                                       1.0
```

[72]:
```python
# --- New Code to Check Latest Date per Country ---
# Group by country and find the maximum (latest) date for each
latest_dates = filtered.groupby('country_region')['date'].max()

# Compute the result (this triggers the Dask computation)
latest_dates_computed = latest_dates.compute()

# Optional: Sort the results by date to see which countries have the most␣
↪recent data
print("\n--- Latest Date per Country (Sorted) ---")
print(latest_dates_computed.sort_values(ascending=False))
```

```
--- Latest Date per Country (Sorted) ---
country_region
Cambodia     2022-10-15
Sri Lanka    2022-10-15
Slovakia     2022-10-15
```

```
Singapore    2022-10-15
Senegal      2022-10-15
                ...
Benin        2022-10-15
Barbados     2022-10-15
Bangladesh   2022-10-15
Australia    2022-10-15
Ukraine      2022-02-23
Name: date, Length: 135, dtype: datetime64[ns]
```

[73]:
```python
# --- New Code to Check Latest Date per Country ---
# Group by country and find the maximum (latest) date for each
latest_dates = filtered.groupby('country_region')['date'].min()

# Compute the result (this triggers the Dask computation)
latest_dates_computed = latest_dates.compute()

# Optional: Sort the results by date to see which countries have the most␣
↪recent data
print("\n--- Earliest Date per Country (Sorted) ---")
print(latest_dates_computed.sort_values(ascending=False))
```

```
--- Earliest Date per Country (Sorted) ---
country_region
Cambodia               2020-02-15
Panama                 2020-02-15
Trinidad and Tobago    2020-02-15
Slovakia               2020-02-15
Singapore              2020-02-15
                          ...
Brazil                 2020-02-15
Benin                  2020-02-15
Barbados               2020-02-15
Bangladesh             2020-02-15
Jordan                 2020-02-15
Name: date, Length: 135, dtype: datetime64[ns]
```

[74]:
```python
# I woul like to get what happened in the first two years of the pandemic
filtered = filtered[
    filtered['date'] <= '2022-02-15'
]
```

[75]:
```python
print(filtered.sample(frac = 0.0001, random_state = 1).compute())
```

```
       country_region_code      country_region sub_region_1 sub_region_2  \
35060                   AU           Australia         <NA>         <NA>
177745                  CM            Cameroon         <NA>         <NA>
```

```
157822                    DO  Dominican Republic         <NA>          <NA>
212703                    HU             Hungary         <NA>          <NA>
130970                    LB             Lebanon         <NA>          <NA>
266250                    MD             Moldova         <NA>          <NA>
58358                     PT            Portugal         <NA>          <NA>
431777                    SV         El Salvador         <NA>          <NA>
430657                    VE           Venezuela         <NA>          <NA>

       metro_area       date  \
35060        <NA> 2021-10-18
177745       <NA> 2021-06-21
157822       <NA> 2020-12-05
212703       <NA> 2021-12-07
130970       <NA> 2020-07-17
266250       <NA> 2022-01-26
58358        <NA> 2020-08-17
431777       <NA> 2020-05-17
430657       <NA> 2020-12-23

        retail_and_recreation_percent_change_from_baseline  \
35060                                               -15.0
177745                                                6.0
157822                                              -29.0
212703                                                7.0
130970                                              -10.0
266250                                              -14.0
58358                                                -1.0
431777                                              -82.0
430657                                                6.0

        grocery_and_pharmacy_percent_change_from_baseline  \
35060                                                6.0
177745                                              33.0
157822                                              -4.0
212703                                              24.0
130970                                               4.0
266250                                               3.0
58358                                               12.0
431777                                             -65.0
430657                                              44.0

        parks_percent_change_from_baseline  \
35060                                -17.0
177745                               -11.0
157822                               -32.0
212703                                22.0
130970                                28.0
266250                               -29.0
```

```
58358                                    93.0
431777                                  -74.0
430657                                    9.0

        transit_stations_percent_change_from_baseline  \
35060                                           -53.0
177745                                           35.0
157822                                          -23.0
212703                                           -9.0
130970                                          -42.0
266250                                          -19.0
58358                                           -35.0
431777                                          -76.0
430657                                           19.0

        workplaces_percent_change_from_baseline  \
35060                                     -18.0
177745                                    -12.0
157822                                    -17.0
212703                                    -12.0
130970                                    -23.0
266250                                    -32.0
58358                                     -44.0
431777                                    -48.0
430657                                    -20.0

        residential_percent_change_from_baseline
35060                                        9.0
177745                                      -2.0
157822                                       8.0
212703                                       5.0
130970                                       1.0
266250                                       1.0
58358                                       11.0
431777                                      23.0
430657                                       9.0
```

[76]:
```python
# rename country_region_code and country_region
filtered = filtered.rename(columns = {
    'country_region_code' : 'code2',
    'country_region' : 'Country'
})
```

[77]:
```python
# get a dataframe for the unique countries in the mobility data
countries_in_mob = filtered[['code2', 'Country']].drop_duplicates().compute()
countries_in_mob
```

[77]:
```
        code2                   Country
308409     BA  Bosnia and Herzegovina
547709     JO                  Jordan
161748     LT               Lithuania
222006     LU              Luxembourg
253027     LY                   Libya
…       …                     …
53823      KE                   Kenya
317749     MY                Malaysia
578462     NL             Netherlands
242319     PY                Paraguay
504341     ZM                  Zambia

[135 rows x 2 columns]
```

[78]:
```
# check if there are missing
countries_in_mob[countries_in_mob['code2'].isna()]
```

[78]:
```
        code2  Country
355842   <NA>  Namibia
```

Python has read NA code for Nambia as an empty cell

[79]:
```
# Create the condition
condition = (filtered['Country'] == 'Namibia') & (filtered['code2'].isna())

# Use .where to replace values where the condition is False
# This means: keep 'code2' where condition is False, otherwise use 'NA'
# Note: .where keeps the original where the condition is FALSE, and replaces
 ↪where it's TRUE
# So we need to negate the condition for .where, or use .mask
# .mask is the opposite of .where: it replaces where the condition is TRUE
filtered['code2'] = filtered['code2'].mask(condition, 'NA')
```

[80]:
```
# get a dataframe for the unique countries in the mobility data
countries_in_mob = filtered[['code2', 'Country']].drop_duplicates().compute()
# check if there are missing
countries_in_mob[countries_in_mob['code2'].isna()]
```

[80]:
```
Empty DataFrame
Columns: [code2, Country]
Index: []
```

[81]:
```
# Get unique code2 that is in countries_in_mob but not in COUNTRY_CODE
not_in_cc = np.setdiff1d(countries_in_mob['code2'].unique(),
 ↪COUNTRY_CODE['code2'].unique())
not_in_cc
```

[81]:
```
array(['AW', 'HK', 'PR', 'RE', 'TW'], dtype=object)
```

[82]:
```
countries_in_mob[countries_in_mob['code2'].isin(not_in_cc)]
```

[82]:
```
       code2      Country
307435    AW        Aruba
261599    RE      Réunion
128171    HK    Hong Kong
498936    PR  Puerto Rico
460440    TW       Taiwan
```

[83]:
```
# Check if there are items Country in countries_in_mob and COUNTRY_CODE that␣
↪have the same code2 but different name
```

All of the five are either not a country, not a soverign nation, or not recognised as independent so for this analysis I would remove them for now

[84]:
```
# Check if there are items in countries_in_mob and COUNTRY_CODE that have the␣
↪same 'code2' but different 'Country' names

# Perform an inner join on 'code2' to find matching codes
merged_check = countries_in_mob[['code2', 'Country']].merge(
    COUNTRY_CODE[['code2', 'Country']],
    on='code2',
    how='inner',
    suffixes=('_mob', '_cc')  # Add suffixes to distinguish the 'Country'␣
↪columns
)

# Filter for rows where the country names are different
mismatched_names = merged_check[merged_check['Country_mob'] !=␣
↪merged_check['Country_cc']]

# Display the results
if not mismatched_names.empty:
    print("Found entries with the same 'code2' but different 'Country' names:")
    print(mismatched_names[['code2', 'Country_mob', 'Country_cc']])
else:
    print("No entries found with the same 'code2' but different 'Country' names.
↪")
```

```
Found entries with the same 'code2' but different 'Country' names:
   code2      Country_mob                                        Country_cc
9     MD          Moldova                              Moldova, Republic of
20    VE        Venezuela                Venezuela, Bolivarian Republic of
24    BO          Bolivia                   Bolivia, Plurinational State of
28    TZ         Tanzania                       Tanzania, United Republic of
30    GB   United Kingdom  United Kingdom of Great Britain and Northern I…
```

```
38      LA             Laos                    Lao People's Democratic Republic
60      VN          Vietnam                                           Viet Nam
68      MM  Myanmar (Burma)                                           Myanmar
76      TR           Turkey                                           Türkiye
86      KR      South Korea                                 Korea, Republic of
100     BS      The Bahamas                                           Bahamas
105     US    United States                          United States of America
109     RU           Russia                                Russian Federation
123     CV       Cape Verde                                        Cabo Verde
```

[85]:
```python
COUNTRY_CODE.head(10)
```

[85]:
```
   id code2 Code              Country
0   4    AF  AFG          Afghanistan
1   8    AL  ALB              Albania
2  12    DZ  DZA              Algeria
3  20    AD  AND              Andorra
4  24    AO  AGO               Angola
5  28    AG  ATG  Antigua and Barbuda
6  32    AR  ARG            Argentina
7  51    AM  ARM              Armenia
8  36    AU  AUS            Australia
9  40    AT  AUT              Austria
```

Based on the output, the countries are the same

[86]:
```python
# create a version of the COUNTRY_CODE the has no country to be merged to the␣
↪filtered dask dataframe
country_code_wo_country = COUNTRY_CODE[['code2', 'Code']]

# merge the COUNTRY_CODE to the filtered dataframe to add the three letter code
filtered_mobility = dd.merge(
    filtered,
    country_code_wo_country,
    on = 'code2',
    how = 'left'
)

# Get a dataframe for the unique countries in the mobility data, excluding rows␣
↪with <NA> in 'code2'
countries_in_mob = (
    filtered_mobility[['Code', 'Country']]
    .drop_duplicates()
    .dropna(subset=['Code'])  # Remove rows where 'code2' is <NA>
    .compute()
)

countries_in_mob
```

[86]:

```
     Code                 Country
2877  ARG               Argentina
2883  GNB           Guinea-Bissau
732   ITA                   Italy
5856  TUR                  Turkey
2196  UKR                 Ukraine
…     …                       …
2193  POL                  Poland
0     ARE    United Arab Emirates
3609  AUT                 Austria
732   KEN                   Kenya
2825  ROU                 Romania

[130 rows x 2 columns]
```

[87]:

```python
# remove duplicates, and sort the results alphabetically by Country
result = (
    filtered_mobility[['code2', 'Code', 'Country']]
    .drop_duplicates()
    .compute()
    .sort_values(by='Country')  # Sort alphabetically by the 'Country' column
)

result.style.set_table_attributes(
    'style="height:300px; overflow-y:scroll; display:block;"'
)
```

[87]: <pandas.io.formats.style.Styler at 0x719527fd89e0>

[88]:

```python
# It seems that there might be missing code
result[result['Code'].isna()]
```

[88]:

```
     code2  Code      Country
732     AW  <NA>        Aruba
3590    HK  <NA>    Hong Kong
2925    PR  <NA>  Puerto Rico
2196    RE  <NA>      Réunion
732     TW  <NA>       Taiwan
```

[89]:

```python
print(filtered_mobility.sample(frac = 0.0001, random_state = 1).compute())
```

```
      code2            Country sub_region_1 sub_region_2 metro_area  \
611      AU          Australia         <NA>         <NA>       <NA>
2688     CM           Cameroon         <NA>         <NA>       <NA>
1758     DO  Dominican Republic        <NA>         <NA>       <NA>
7179     HU            Hungary         <NA>         <NA>       <NA>
6009     LB            Lebanon         <NA>         <NA>       <NA>
```

```
12385    MD             Moldova         <NA>         <NA>       <NA>
184      PT            Portugal         <NA>         <NA>       <NA>
3020     SV         El Salvador         <NA>         <NA>       <NA>
1044     VE           Venezuela         <NA>         <NA>       <NA>

            date  retail_and_recreation_percent_change_from_baseline  \
611   2021-10-18                                              -15.0
2688  2021-06-21                                                6.0
1758  2020-12-05                                              -29.0
7179  2021-12-07                                                7.0
6009  2020-07-17                                              -10.0
12385 2022-01-26                                              -14.0
184   2020-08-17                                               -1.0
3020  2020-05-17                                              -82.0
1044  2020-12-23                                                6.0

       grocery_and_pharmacy_percent_change_from_baseline  \
611                                                  6.0
2688                                                33.0
1758                                                -4.0
7179                                                24.0
6009                                                 4.0
12385                                                3.0
184                                                 12.0
3020                                               -65.0
1044                                                44.0

       parks_percent_change_from_baseline  \
611                                 -17.0
2688                                -11.0
1758                                -32.0
7179                                 22.0
6009                                 28.0
12385                               -29.0
184                                  93.0
3020                                -74.0
1044                                  9.0

       transit_stations_percent_change_from_baseline  \
611                                            -53.0
2688                                            35.0
1758                                           -23.0
7179                                            -9.0
6009                                           -42.0
12385                                          -19.0
184                                            -35.0
3020                                           -76.0
1044                                            19.0
```

```
       workplaces_percent_change_from_baseline  \
611                                      -18.0
2688                                     -12.0
1758                                     -17.0
7179                                     -12.0
6009                                     -23.0
12385                                    -32.0
184                                      -44.0
3020                                     -48.0
1044                                     -20.0

       residential_percent_change_from_baseline Code
611                                         9.0  AUS
2688                                       -2.0  CMR
1758                                        8.0  DOM
7179                                        5.0  HUN
6009                                        1.0  LBN
12385                                       1.0  MDA
184                                        11.0  PRT
3020                                       23.0  SLV
1044                                        9.0  VEN
```

[90]:

```
# Drop unnecessary columns in filtered_mobility
filtered_mobility = filtered_mobility.drop(columns = [
    'code2',
    'sub_region_1',
    'sub_region_2',
    'metro_area'
], axis = 1)

# Rename columns to be shorter
filtered_mobility = filtered_mobility.rename(columns = {
    'retail_and_recreation_percent_change_from_baseline' :␣
 ↪'retail_and_recreation',
    'grocery_and_pharmacy_percent_change_from_baseline' :␣
 ↪'grocery_and_pharmacy',
    'parks_percent_change_from_baseline' : 'parks',
    'transit_stations_percent_change_from_baseline' : 'transit_stations',
    'workplaces_percent_change_from_baseline' : 'workplaces',
    'residential_percent_change_from_baseline' : 'residential',
    'date' : 'Date'
})

print(filtered_mobility.sample(frac = 0.0001, random_state = 1).compute())
```

```
                  Country       Date  retail_and_recreation  \
611             Australia 2021-10-18                  -15.0
```

```
2688               Cameroon 2021-06-21                        6.0
1758    Dominican Republic 2020-12-05                       -29.0
7179                Hungary 2021-12-07                        7.0
6009                Lebanon 2020-07-17                      -10.0
12385               Moldova 2022-01-26                      -14.0
184                Portugal 2020-08-17                       -1.0
3020            El Salvador 2020-05-17                      -82.0
1044              Venezuela 2020-12-23                        6.0

       grocery_and_pharmacy  parks  transit_stations  workplaces  residential  \
611                     6.0  -17.0             -53.0       -18.0          9.0
2688                   33.0  -11.0              35.0       -12.0         -2.0
1758                   -4.0  -32.0             -23.0       -17.0          8.0
7179                   24.0   22.0              -9.0       -12.0          5.0
6009                    4.0   28.0             -42.0       -23.0          1.0
12385                   3.0  -29.0             -19.0       -32.0          1.0
184                    12.0   93.0             -35.0       -44.0         11.0
3020                  -65.0  -74.0             -76.0       -48.0         23.0
1044                   44.0    9.0              19.0       -20.0          9.0

      Code
611    AUS
2688   CMR
1758   DOM
7179   HUN
6009   LBN
12385  MDA
184    PRT
3020   SLV
1044   VEN
```

1.3.2 Get national policy

[91]:
```
# Check the oxford covid19 government response tracker
national_policy = pd.read_csv('data/oxcgt.csv')
national_policy.head()
```

[91]:
```
  CountryName CountryCode  RegionName  RegionCode Jurisdiction      Date  \
0       Aruba         ABW         NaN         NaN    NAT_TOTAL  20200101
1       Aruba         ABW         NaN         NaN    NAT_TOTAL  20200102
2       Aruba         ABW         NaN         NaN    NAT_TOTAL  20200103
3       Aruba         ABW         NaN         NaN    NAT_TOTAL  20200104
4       Aruba         ABW         NaN         NaN    NAT_TOTAL  20200105

   C1M_School closing  C1M_Flag  C2M_Workplace closing  C2M_Flag  …  \
0                   0       NaN                      0       NaN  …
1                   0       NaN                      0       NaN  …
```

```
2                   0       NaN                      0       NaN  …
3                   0       NaN                      0       NaN  …
4                   0       NaN                      0       NaN  …

   V3_Vaccine Financial Support (summary)  V4_Mandatory Vaccination (summary)  \
0                                       0                                 NaN
1                                       0                                 NaN
2                                       0                                 NaN
3                                       0                                 NaN
4                                       0                                 NaN

   ConfirmedCases  ConfirmedDeaths  MajorityVaccinated  PopulationVaccinated  \
0             0.0              0.0                  NV                   0.0
1             0.0              0.0                  NV                   0.0
2             0.0              0.0                  NV                   0.0
3             0.0              0.0                  NV                   0.0
4             0.0              0.0                  NV                   0.0

   StringencyIndex_Average  GovernmentResponseIndex_Average  \
0                      0.0                              0.0
1                      0.0                              0.0
2                      0.0                              0.0
3                      0.0                              0.0
4                      0.0                              0.0

   ContainmentHealthIndex_Average  EconomicSupportIndex
0                             0.0                   0.0
1                             0.0                   0.0
2                             0.0                   0.0
3                             0.0                   0.0
4                             0.0                   0.0

[5 rows x 56 columns]
```

[92]:
```
eval_df(national_policy)
```

```
 DATA TYPES & MEMORY USAGE
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202760 entries, 0 to 202759
Data columns (total 56 columns):
 #   Column
Non-Null Count   Dtype
---  ------
--------------   -----
 0   CountryName
```

```
202760 non-null  object
 1   CountryCode
202760 non-null  object
 2   RegionName
0 non-null       float64
 3   RegionCode
0 non-null       float64
 4   Jurisdiction
202760 non-null  object
 5   Date
202760 non-null  int64
 6   C1M_School closing
202760 non-null  int64
 7   C1M_Flag
128263 non-null  float64
 8   C2M_Workplace closing
202760 non-null  int64
 9   C2M_Flag
133824 non-null  float64
 10  C3M_Cancel public events
202760 non-null  int64
 11  C3M_Flag
138576 non-null  float64
 12  C4M_Restrictions on gatherings
202760 non-null  int64
 13  C4M_Flag
125800 non-null  float64
 14  C5M_Close public transport
202760 non-null  int64
 15  C5M_Flag
67355 non-null   float64
 16  C6M_Stay at home requirements
202760 non-null  int64
 17  C6M_Flag
89648 non-null   float64
 18  C7M_Restrictions on internal movement
202760 non-null  int64
 19  C7M_Flag
70722 non-null   float64
 20  C8EV_International travel controls
202760 non-null  int64
 21  E1_Income support
202760 non-null  int64
 22  E1_Flag
99391 non-null   float64
 23  E2_Debt/contract relief
202760 non-null  int64
 24  E3_Fiscal measures
```

```
106288 non-null  float64
 25  E4_International support
106370 non-null  float64
 26  H1_Public information campaigns
202760 non-null  int64
 27  H1_Flag
191745 non-null  float64
 28  H2_Testing policy
202760 non-null  int64
 29  H3_Contact tracing
202760 non-null  int64
 30  H4_Emergency investment in healthcare
106245 non-null  float64
 31  H5_Investment in vaccines
198475 non-null  float64
 32  H6M_Facial Coverings
202760 non-null  int64
 33  H6M_Flag
167295 non-null  float64
 34  H7_Vaccination policy
202760 non-null  int64
 35  H7_Flag
126150 non-null  float64
 36  H8M_Protection of elderly people
202760 non-null  int64
 37  H8M_Flag
114519 non-null  float64
 38  V1_Vaccine Prioritisation (summary)
202760 non-null  int64
 39  V2A_Vaccine Availability (summary)
202760 non-null  int64
 40  V2B_Vaccine age eligibility/availability age floor (general population
summary)  118396 non-null  object
 41  V2C_Vaccine age eligibility/availability age floor (at risk summary)
120360 non-null  object
 42  V2D_Medically/ clinically vulnerable (Non-elderly)
127138 non-null  float64
 43  V2E_Education
127138 non-null  float64
 44  V2F_Frontline workers  (non healthcare)
127138 non-null  float64
 45  V2G_Frontline workers  (healthcare)
127138 non-null  float64
 46  V3_Vaccine Financial Support (summary)
202760 non-null  int64
 47  V4_Mandatory Vaccination (summary)
90957 non-null   float64
 48  ConfirmedCases
```

```
201664 non-null  float64
 49  ConfirmedDeaths
201664 non-null  float64
 50  MajorityVaccinated
200568 non-null  object
 51  PopulationVaccinated
200568 non-null  float64
 52  StringencyIndex_Average
202760 non-null  float64
 53  GovernmentResponseIndex_Average
202760 non-null  float64
 54  ContainmentHealthIndex_Average
202760 non-null  float64
 55  EconomicSupportIndex
202760 non-null  float64
dtypes: float64(30), int64(20), object(6)
memory usage: 86.6+ MB
None

  MISSING VALUES PER COLUMN
----------------------------------------
RegionName
202760
RegionCode
202760
C1M_Flag
74497
C2M_Flag
68936
C3M_Flag
64184
C4M_Flag
76960
C5M_Flag
135405
C6M_Flag
113112
C7M_Flag
132038
E1_Flag
103369
E3_Fiscal measures
96472
E4_International support
96390
H1_Flag
11015
H4_Emergency investment in healthcare
```

```
96515
H5_Investment in vaccines
4285
H6M_Flag
35465
H7_Flag
76610
H8M_Flag
88241
V2B_Vaccine age eligibility/availability age floor (general population summary)
84364
V2C_Vaccine age eligibility/availability age floor (at risk summary)
82400
V2D_Medically/ clinically vulnerable (Non-elderly)
75622
V2E_Education
75622
V2F_Frontline workers  (non healthcare)
75622
V2G_Frontline workers  (healthcare)
75622
V4_Mandatory Vaccination (summary)
111803
ConfirmedCases
1096
ConfirmedDeaths
1096
MajorityVaccinated
2192
PopulationVaccinated
2192
dtype: int64

  DUPLICATE ROWS FOUND
----------------------------------------
0 duplicate rows found.

  SUMMARY STATISTICS
----------------------------------------
       RegionName  RegionCode          Date  C1M_School closing  \
count         0.0         0.0  2.027600e+05       202760.000000
mean          NaN         NaN  2.021066e+07            1.193199
std           NaN         NaN  8.174621e+03            1.139102
min           NaN         NaN  2.020010e+07            0.000000
25%           NaN         NaN  2.020098e+07            0.000000
50%           NaN         NaN  2.021070e+07            1.000000
75%           NaN         NaN  2.022040e+07            2.000000
max           NaN         NaN  2.022123e+07            3.000000
```

```
            C1M_Flag  C2M_Workplace closing       C2M_Flag  \
count  128263.000000          202760.000000  133824.000000
mean        0.835268               1.149645       0.797630
std         0.370940               0.988767       0.401768
min         0.000000               0.000000       0.000000
25%         1.000000               0.000000       1.000000
50%         1.000000               1.000000       1.000000
75%         1.000000               2.000000       1.000000
max         1.000000               3.000000       1.000000

       C3M_Cancel public events       C3M_Flag  \
count             202760.000000  138576.000000
mean                   1.103250       0.858857
std                    0.851878       0.348170
min                    0.000000       0.000000
25%                    0.000000       1.000000
50%                    1.000000       1.000000
75%                    2.000000       1.000000
max                    2.000000       1.000000

       C4M_Restrictions on gatherings  …  \
count                   202760.000000  …
mean                         2.010209  …
std                          1.710198  …
min                          0.000000  …
25%                          0.000000  …
50%                          3.000000  …
75%                          4.000000  …
max                          4.000000  …

       V2G_Frontline workers  (healthcare)  \
count                        127138.000000
mean                              1.803316
std                               0.471447
min                               0.000000
25%                               2.000000
50%                               2.000000
75%                               2.000000
max                               2.000000

       V3_Vaccine Financial Support (summary)  \
count                           202760.000000
mean                                 3.100533
std                                  2.411897
min                                  0.000000
25%                                  0.000000
50%                                  5.000000
```

```
75%                                    5.000000
max                                    5.000000

       V4_Mandatory Vaccination (summary)  ConfirmedCases  ConfirmedDeaths  \
count                        90957.000000    2.016640e+05     2.016640e+05
mean                             0.278681    1.340886e+06     1.955307e+04
std                              0.448353    5.583371e+06     7.556113e+04
min                              0.000000    0.000000e+00     0.000000e+00
25%                              0.000000    5.146750e+03     6.400000e+01
50%                              0.000000    5.987900e+04     8.650000e+02
75%                              1.000000    5.218538e+05     7.470000e+03
max                              1.000000    1.007653e+08     1.092764e+06

       PopulationVaccinated  StringencyIndex_Average  \
count         200568.000000            202760.000000
mean              22.603420                42.675426
std               29.597555                24.930305
min                0.000000                 0.000000
25%                0.000000                22.220000
50%                2.330000                42.590000
75%               46.570000                62.040000
max              105.750000               100.000000

       GovernmentResponseIndex_Average  ContainmentHealthIndex_Average  \
count                    202760.000000                   202760.000000
mean                         44.857776                       46.699253
std                          19.649721                       19.865910
min                           0.000000                        0.000000
25%                          31.250000                       33.330000
50%                          46.880000                       48.720000
75%                          60.000000                       62.020000
max                          91.150000                       93.450000

       EconomicSupportIndex
count         202760.000000
mean              31.968029
std               32.962193
min                0.000000
25%                0.000000
50%               25.000000
75%               62.500000
max              100.000000

[8 rows x 50 columns]
```

[93]:
```
# List of measures to focus on
measures_to_focus = [
```

```
    'CountryName',
    'CountryCode',
    'Date',
    'ConfirmedCases',
    'ConfirmedDeaths',
    'PopulationVaccinated',
    'StringencyIndex_Average', # Index that encompasses containment and closure␣
 ↪policies and public information campaigns
    'ContainmentHealthIndex_Average', # Index that involve both␣
 ↪StringencyIndex_Average plus health system polices
    'EconomicSupportIndex' # Index encompasses by economic policies
]

# Filter national_policy to include only the selected high-priority measures
national_policy = national_policy[measures_to_focus]

# Convert the 'Date' column to datetime using .loc to avoid␣
 ↪SettingWithCopyWarning
# This explicitly targets the 'Date' column for all rows (:)
national_policy.loc[:, 'Date'] = pd.to_datetime(
    national_policy['Date'].astype(str),
    format='%Y%m%d'
)

# Rename the columns in the national_policy columns
national_policy = national_policy.rename(columns = {
    'CountryName' : 'Country',
    'CountryCode' : 'Code'
})

national_policy
```

[93]:

```
         Country Code       Date  ConfirmedCases  ConfirmedDeaths  \
0          Aruba  ABW 2020-01-01             0.0              0.0
1          Aruba  ABW 2020-01-02             0.0              0.0
2          Aruba  ABW 2020-01-03             0.0              0.0
3          Aruba  ABW 2020-01-04             0.0              0.0
4          Aruba  ABW 2020-01-05             0.0              0.0
…              …   …          …               …                …
202755  Zimbabwe  ZWE 2022-12-27        259981.0           5637.0
202756  Zimbabwe  ZWE 2022-12-28        259981.0           5637.0
202757  Zimbabwe  ZWE 2022-12-29        259981.0           5637.0
202758  Zimbabwe  ZWE 2022-12-30        259981.0           5637.0
202759  Zimbabwe  ZWE 2022-12-31        259981.0           5637.0

        PopulationVaccinated  StringencyIndex_Average  \
0                       0.00                     0.00
```

```
1                       0.00                      0.00
2                       0.00                      0.00
3                       0.00                      0.00
4                       0.00                      0.00
…                       …                         …
202755                 29.11                     29.48
202756                 29.11                     29.48
202757                 29.11                     29.48
202758                 29.11                     29.48
202759                 29.11                     29.48

        ContainmentHealthIndex_Average  EconomicSupportIndex
0                                 0.00                   0.0
1                                 0.00                   0.0
2                                 0.00                   0.0
3                                 0.00                   0.0
4                                 0.00                   0.0
…                                  …                   …
202755                           41.65                   0.0
202756                           41.65                   0.0
202757                           41.65                   0.0
202758                           41.65                   0.0
202759                           41.65                   0.0

[202760 rows x 9 columns]
```

[94]:
```
# To mirror the mobility data, I would limit the max date to 2022-02-15
national_policy = national_policy[
    national_policy['Date'] <= '2022-02-15'
]
```

[95]:
```
# Create a Country and Code of the national_policy
np_df = national_policy[['Code']].drop_duplicates()

# Get the intersection of np_df and the countries_in_mob
study_sample = pd.merge(
    countries_in_mob,
    np_df,
    how = 'inner',
    on = ['Code'])

print(study_sample.sort_values(by = 'Country'))
```

```
    Code      Country
50   AFG  Afghanistan
113  AGO       Angola
0    ARG    Argentina
```

```
29   AUS    Australia
124  AUT      Austria
..   …          …
110  VEN    Venezuela
4    VNM      Vietnam
70   YEM        Yemen
111  ZMB       Zambia
112  ZWE     Zimbabwe

[127 rows x 2 columns]
```

1.3.3 Get the country stats

[96]:
```
COUNTRY_CODE
```

[96]:
```
      id code2 Code                            Country
0      4    AF  AFG                        Afghanistan
1      8    AL  ALB                            Albania
2     12    DZ  DZA                            Algeria
3     20    AD  AND                            Andorra
4     24    AO  AGO                             Angola
..   …   …   …                                 …
188  862    VE  VEN  Venezuela, Bolivarian Republic of
189  704    VN  VNM                           Viet Nam
190  887    YE  YEM                              Yemen
191  894    ZM  ZMB                             Zambia
192  716    ZW  ZWE                           Zimbabwe

[193 rows x 4 columns]
```

[97]:
```
country_stat = COUNTRY_CODE
# Define path to folder containing country statistics CSVs
folder_path = 'data/country_stat/'
csv_files = glob.glob(folder_path + "*.csv")

# Merge each additional CSV file into the main country_stat DataFrame
# Handle potential encoding issues for each file
for file in csv_files:
    print(f"Attempting to read: {file}")
    try:
        # Try UTF-8 first (most common standard)
        df = pd.read_csv(file, encoding='utf-8')
    except UnicodeDecodeError:
        # If UTF-8 fails, try common alternatives
        try:
            df = pd.read_csv(file, encoding='latin1') # ISO 8859-1
            print(f"  Successfully read {file} with 'latin1' encoding.")
```

```
        except UnicodeDecodeError:
            try:
                df = pd.read_csv(file, encoding='cp1252') # Windows-1252
                print(f"  Successfully read {file} with 'cp1252' encoding.")
            except UnicodeDecodeError:
                # If all common encodings fail, raise an error with the filename
                raise UnicodeDecodeError(f"Failed to read {file} with common␣
↪encodings (utf-8, latin1, cp1252). "
                                         "Please check the file's encoding.")

    # Drop 'Country' column if it's duplicated (assuming 'Code' is the unique␣
↪key)
    df.drop(columns=["Country"], inplace=True)

    # Merge on country code
    country_stat = pd.merge(country_stat, df, on="Code", how="outer")

# Evaluate the merged DataFrame for structure, completeness, and duplicates
eval_df(country_stat)
```

```
Attempting to read: data/country_stat/urbanization.csv
Attempting to read: data/country_stat/corruption_perception_index.csv
Attempting to read: data/country_stat/gdp_per_capita.csv
Attempting to read: data/country_stat/geographic_data.csv
Attempting to read: data/country_stat/hospital_beds.csv
  Successfully read data/country_stat/hospital_beds.csv with 'latin1' encoding.
Attempting to read: data/country_stat/unemployment.csv
Attempting to read: data/country_stat/political_regime.csv
Attempting to read: data/country_stat/gini_index.csv
Attempting to read: data/country_stat/population_density.csv
Attempting to read: data/country_stat/extreme_poverty.csv
Attempting to read: data/country_stat/median-age.csv
Attempting to read: data/country_stat/land-area-km.csv

 DATA TYPES & MEMORY USAGE
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 317 entries, 0 to 316
Data columns (total 19 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   id                           193 non-null    float64
 1   code2                        193 non-null    object
 2   Code                         247 non-null    object
 3   Country                      193 non-null    object
 4   urban_population             197 non-null    float64
 5   corruption_perception_index  180 non-null    float64
```

```
 6   gdp_per_capita             271 non-null    float64
 7   land_boundaries            312 non-null    float64
 8   coastline                  310 non-null    float64
 9   num_border_countries       310 non-null    float64
 10  border_countries           176 non-null    object
 11  hospital_beds_per_1000     167 non-null    float64
 12  unemployment               187 non-null    float64
 13  political_regime           174 non-null    object
 14  gini_index                 169 non-null    float64
 15  population_density         200 non-null    float64
 16  poverty                    159 non-null    float64
 17  median_age                 201 non-null    float64
 18  land_area_sqkm             196 non-null    float64
dtypes: float64(14), object(5)
memory usage: 47.2+ KB
None

  MISSING VALUES PER COLUMN
----------------------------------------
id                             124
code2                          124
Code                            70
Country                        124
urban_population               120
corruption_perception_index    137
gdp_per_capita                  46
land_boundaries                  5
coastline                        7
num_border_countries             7
border_countries               141
hospital_beds_per_1000         150
unemployment                   130
political_regime               143
gini_index                     148
population_density             117
poverty                        158
median_age                     116
land_area_sqkm                 121
dtype: int64

  DUPLICATE ROWS FOUND
----------------------------------------
0 duplicate rows found.

  SUMMARY STATISTICS
----------------------------------------
               id  urban_population  corruption_perception_index  \
count  193.000000        197.000000                   180.000000
```

```
mean   433.279793        59.427223                          43.166667
std    254.431053        23.161899                          18.960264
min      4.000000        13.250000                           9.000000
25%    212.000000        41.612000                          29.000000
50%    430.000000        60.308000                          39.500000
75%    659.000000        78.099000                          56.000000
max    894.000000       100.000000                          87.000000

       gdp_per_capita  land_boundaries      coastline  num_border_countries  \
count      271.000000       312.000000     310.000000            310.000000
mean     26985.167085      1747.996795    2602.079032              2.106452
std      26661.273006      3088.111848   12848.775646              2.548867
min        623.400000         0.000000       0.000000              0.000000
25%       6856.675700         0.000000      58.900000              0.000000
50%      18256.998000       156.000000     190.500000              1.000000
75%      42198.130000      2432.000000    1146.750000              4.000000
max     166907.800000     22457.000000  202080.000000             14.000000

       hospital_beds_per_1000  unemployment  gini_index  population_density  \
count              167.000000    187.000000  169.000000          200.000000
mean                 2.951976      7.293941    0.375518          294.578268
std                  2.679764      5.673796    0.079872         1414.506600
min                  0.170000      0.100000    0.232323            0.136699
25%                  1.110000      3.448000    0.317809           31.278362
50%                  2.300000      5.206000    0.356654           84.848432
75%                  4.055000     10.334500    0.425342          209.366408
max                 22.020000     28.468000    0.630258        18297.025000

          poverty  median_age  land_area_sqkm
count  159.000000  201.000000    1.960000e+02
mean    10.755403   29.041841    6.618971e+05
std     17.961751    9.253179    1.823404e+06
min      0.000000   14.368000    2.084000e+00
25%      0.266830   20.903000    2.308750e+04
50%      1.401915   28.361000    1.203750e+05
75%     15.240933   36.543000    5.151600e+05
max     78.942020   54.642000    1.637687e+07
```

[98]:

```python
# Drop the id and code2 columns
country_stat = country_stat.drop(columns = ['id', 'code2'], axis = 1)
```

[99]:

```python
# Create a Country and Code of the country stat
cs_df = country_stat[['Code']].drop_duplicates()

# Get the intersection of np_df and the countries_in_mob
study_sample = pd.merge(
    study_sample,
```

```
    cs_df,
    how = 'inner',
    on = ['Code'])

print(study_sample.sort_values(by = 'Country'))
```

```
    Code      Country
50   AFG  Afghanistan
113  AGO       Angola
0    ARG    Argentina
29   AUS    Australia
124  AUT      Austria
..   …          …
110  VEN    Venezuela
4    VNM      Vietnam
70   YEM        Yemen
111  ZMB       Zambia
112  ZWE     Zimbabwe

[127 rows x 2 columns]
```

[100]:
```
country_stat = country_stat[country_stat['Code'].isin(study_sample['Code'])]
eval_df(country_stat)
```

```
 DATA TYPES & MEMORY USAGE
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
Index: 127 entries, 1 to 246
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Code                         127 non-null    object
 1   Country                      127 non-null    object
 2   urban_population             127 non-null    float64
 3   corruption_perception_index  124 non-null    float64
 4   gdp_per_capita               126 non-null    float64
 5   land_boundaries              127 non-null    float64
 6   coastline                    127 non-null    float64
 7   num_border_countries         127 non-null    float64
 8   border_countries             112 non-null    object
 9   hospital_beds_per_1000       126 non-null    float64
 10  unemployment                 126 non-null    float64
 11  political_regime             124 non-null    object
 12  gini_index                   125 non-null    float64
 13  population_density           127 non-null    float64
 14  poverty                      115 non-null    float64
 15  median_age                   127 non-null    float64
```

```
 16  land_area_sqkm               127 non-null    float64
dtypes: float64(13), object(4)
memory usage: 17.9+ KB
None

  MISSING VALUES PER COLUMN
----------------------------------------
corruption_perception_index     3
gdp_per_capita                  1
border_countries               15
hospital_beds_per_1000          1
unemployment                    1
political_regime                3
gini_index                      2
poverty                        12
dtype: int64

  DUPLICATE ROWS FOUND
----------------------------------------
0 duplicate rows found.

  SUMMARY STATISTICS
----------------------------------------
       urban_population  corruption_perception_index  gdp_per_capita  \
count        127.000000                   124.000000      126.000000
mean          62.862701                    46.217742    29550.430979
std           22.370051                    19.204742    28943.024152
min           13.250000                    15.000000      623.400000
25%           47.401000                    30.750000     8125.092000
50%           66.177000                    41.000000    18379.695000
75%           81.456000                    60.000000    44282.128750
max          100.000000                    87.000000   166907.800000

       land_boundaries      coastline  num_border_countries  \
count       127.000000     127.000000            127.000000
mean       3190.928346    5059.715748              3.629921
std        3482.265985   19347.715515              2.448852
min           0.000000       0.000000              0.000000
25%         877.000000      62.250000              2.000000
50%        2237.000000     823.000000              4.000000
75%        4368.075000    2790.000000              5.000000
max       22407.000000  202080.000000             14.000000

       hospital_beds_per_1000  unemployment  gini_index  population_density  \
count              126.000000    126.000000  125.000000          127.000000
mean                 2.907857      6.571746    0.381391          227.507278
std                  2.433828      4.960589    0.083967          735.066124
min                  0.170000      0.100000    0.232323            2.075187
```

```
25%                  1.067500      3.433500    0.318931            33.333070
50%                  2.230000      4.968500    0.362465            84.937030
75%                  4.177500      8.730750    0.433141           208.888695
max                 12.880000     28.468000    0.630258          7896.325700

         poverty  median_age  land_area_sqkm
count  115.000000  127.000000    1.270000e+02
mean     8.515624   30.512756    8.072397e+05
std     15.720207    9.175992    2.070121e+06
min      0.000000   14.699000    1.600000e+02
25%      0.243869   22.996500    5.517500e+04
50%      1.040000   29.063000    2.300800e+05
75%      6.626446   39.189500    6.158150e+05
max     74.528350   47.262000    1.637687e+07
```

[101]:
```
columns_w_missing = [
    'Code',
    'Country',
    'corruption_perception_index',
    'gdp_per_capita',
    'border_countries',
    'hospital_beds_per_1000',
    'unemployment',
    'political_regime',
    'gini_index',
    'poverty'
]
```

Evaluation of the Missing values:
No data for Liechtenstein, for the rest I used the value that is closest to the 2019 value.

[102]:
```
# Create a copy to explore missing data safely
missing_df = country_stat[columns_w_missing]

# Add a column to count missing values per country
missing_df.loc[:, 'missing_count'] = missing_df.isna().sum(axis=1)

# Filter only countries with at least one missing value and sort by the number␣
↪of missing entries
missing_df = missing_df[missing_df['missing_count'] > 0].
↪sort_values(by='missing_count', ascending=True)

# Display in a scrollable format (useful for large result sets)
missing_df
```

[102]:
```
    Code                         Country  corruption_perception_index  \
1    AFG                     Afghanistan                         16.0
```

```
13   AUS                        Australia                          77.0
50   CPV                       Cabo Verde                          58.0
31   BRB                         Barbados                          62.0
121  KWT                           Kuwait                          40.0
117  KHM                         Cambodia                          20.0
113  JPN                            Japan                          73.0
110  JAM                          Jamaica                          43.0
125  LBY                            Libya                          18.0
128  LKA                        Sri Lanka                          38.0
144  MLT                            Malta                          54.0
235  VEN  Venezuela, Bolivarian Republic of                        16.0
188  SAU                     Saudi Arabia                          53.0
174  PHL                      Philippines                          34.0
167  OMN                             Oman                          52.0
152  MUS                        Mauritius                          52.0
72   FJI                             Fiji                           NaN
166  NZL                      New Zealand                          87.0
191  SGP                        Singapore                          85.0
22   BHR                          Bahrain                          42.0
221  TTO              Trinidad and Tobago                          40.0
23   BHS                          Bahamas                          64.0
27   BLZ                           Belize                           NaN
127  LIE                    Liechtenstein                           NaN

     gdp_per_capita                                   border_countries  \
1          2927.245  China 91 km; Iran 921 km; Pakistan 2,670 km; T…
13        56981.395                                                NaN
50         9111.630                                                NaN
31        18572.510                                                NaN
121       51122.332                  Iraq 254 km; Saudi Arabia 221 km
117        6448.885      Laos 555 km; Thailand 817 km; Vietnam 1158 km
113       44976.508                                                NaN
110       10216.065                                                NaN
125       14333.423  Algeria 989 km; Chad 1,050 km; Egypt 1,115 km;…
128       14637.444                                                NaN
144       54667.383                                                NaN
235             NaN  Brazil 2,137 km; Colombia 2,341 km; Guyana 789 km
188       56365.508  Iraq 811 km; Jordan 731 km; Kuwait 221 km; Oma…
174        9452.294                                                NaN
167       38292.387      Saudi Arabia 658 km; UAE 609 km; Yemen 294 km
152       25739.355                                                NaN
72        13567.474                                                NaN
166       47523.230                                                NaN
191      119572.270                                                NaN
22        56749.960                                                NaN
221       34349.130                                                NaN
23        32495.334                                                NaN
```

```
27       11803.088                  Guatemala 266 km; Mexico 276 km
127     166907.800                  Austria 34 km; Switzerland 41 km

     hospital_beds_per_1000  unemployment       political_regime  gini_index  \
1                      0.38        11.185  electoral_autocracies    0.410000
13                     3.84         5.159    liberal_democracies    0.343326
50                     1.97        12.128  electoral_democracies    0.423811
31                     5.74         8.412    liberal_democracies    0.340680
121                    2.00         2.251  electoral_autocracies    0.529000
117                    0.74         0.119  electoral_autocracies    0.454000
113                   12.88         2.351    liberal_democracies    0.329849
110                    1.73         4.987    liberal_democracies    0.356387
125                    3.20        19.050     closed_autocracies    0.441000
128                    4.00         4.670  electoral_democracies    0.376638
144                    4.11         3.616  electoral_democracies    0.310418
235                    0.93         5.876  electoral_autocracies    0.446984
188                    2.15         5.636     closed_autocracies    0.544000
174                    0.98         2.237  electoral_autocracies    0.378117
167                    1.10         2.040     closed_autocracies    0.443000
152                    3.63         6.331  electoral_democracies    0.367612
72                     1.89         4.373  electoral_autocracies    0.307069
166                    2.55         4.109    liberal_democracies    0.346000
191                    2.60         3.100  electoral_autocracies    0.337000
22                     1.74         1.223     closed_autocracies    0.557000
221                    1.90         3.523    liberal_democracies    0.533000
23                     2.70         9.336                    NaN    0.533000
27                     1.03         9.053                    NaN         NaN
127                     NaN           NaN                    NaN         NaN

      poverty  missing_count
1         NaN              1
13   0.497094              1
50   4.564231              1
31   1.677398              1
121       NaN              1
117       NaN              1
113  1.221445              1
110  0.056063              1
125       NaN              1
128  0.958613              1
144  0.304682              1
235  9.712060              1
188       NaN              1
174  5.057057              1
167       NaN              1
152  0.125314              1
72   1.318269              2
```

```
166        NaN              2
191        NaN              2
22         NaN              2
221        NaN              2
23         NaN              3
27    1.040000              3
127        NaN              6
```

[103]:
```python
# Drop border countries
missing_df = missing_df.drop(columns = 'border_countries', axis = 1)

# Add a column to count missing values per country
missing_df.loc[:, 'missing_count'] = missing_df.isna().sum(axis=1)

# Filter only countries with at least one missing value and sort by the number␣
↪of missing entries
missing_df = missing_df[missing_df['missing_count'] > 0].
↪sort_values(by='missing_count', ascending=True)

# Display in a scrollable format (useful for large result sets)
missing_df
```

[103]:
```
    Code                           Country  corruption_perception_index  \
1    AFG                       Afghanistan                         16.0
121  KWT                            Kuwait                         40.0
117  KHM                          Cambodia                         20.0
125  LBY                             Libya                         18.0
235  VEN  Venezuela, Bolivarian Republic of                         16.0
188  SAU                      Saudi Arabia                         53.0
167  OMN                              Oman                         52.0
72   FJI                              Fiji                          NaN
166  NZL                       New Zealand                         87.0
191  SGP                         Singapore                         85.0
22   BHR                           Bahrain                         42.0
221  TTO               Trinidad and Tobago                         40.0
23   BHS                           Bahamas                         64.0
27   BLZ                            Belize                          NaN
127  LIE                     Liechtenstein                          NaN

     gdp_per_capita  hospital_beds_per_1000  unemployment  \
1          2927.245                    0.38        11.185
121       51122.332                    2.00         2.251
117        6448.885                    0.74         0.119
125       14333.423                    3.20        19.050
235             NaN                    0.93         5.876
188       56365.508                    2.15         5.636
167       38292.387                    1.10         2.040
```

```
72        13567.474                    1.89         4.373
166       47523.230                    2.55         4.109
191      119572.270                    2.60         3.100
22        56749.960                    1.74         1.223
221       34349.130                    1.90         3.523
23        32495.334                    2.70         9.336
27        11803.088                    1.03         9.053
127      166907.800                     NaN           NaN

          political_regime  gini_index   poverty  missing_count
1    electoral_autocracies    0.410000       NaN              1
121  electoral_autocracies    0.529000       NaN              1
117  electoral_autocracies    0.454000       NaN              1
125     closed_autocracies    0.441000       NaN              1
235  electoral_autocracies    0.446984  9.712060              1
188     closed_autocracies    0.544000       NaN              1
167     closed_autocracies    0.443000       NaN              1
72   electoral_autocracies    0.307069  1.318269              1
166    liberal_democracies    0.346000       NaN              1
191  electoral_autocracies    0.337000       NaN              1
22      closed_autocracies    0.557000       NaN              1
221    liberal_democracies    0.533000       NaN              1
23                     NaN    0.533000       NaN              2
27                     NaN         NaN  1.040000              3
127                    NaN         NaN       NaN              6
```

[104]:
```
# for this analysis we will remove all the rows that have missing
study_sample = study_sample[~study_sample['Code'].isin(missing_df['Code'])]
```

2 Create country stat, national policy and mobility data that includes only countries in the study sample

[105]:
```
country_stat = country_stat[country_stat['Code'].isin(study_sample['Code'])]
country_stat
```

[105]:
```
    Code               Country  urban_population  corruption_perception_index  \
2    AGO                Angola            66.177                         26.0
6    ARE  United Arab Emirates            86.789                         71.0
7    ARG             Argentina            91.991                         45.0
13   AUS             Australia            86.124                         77.0
14   AUT               Austria            58.515                         77.0
..   …                     …                 …                          …
238  VNM              Viet Nam            36.628                         37.0
243  YEM                 Yemen            37.273                         15.0
244  ZAF          South Africa            66.856                         44.0
245  ZMB                Zambia            44.072                         34.0
```

```
246  ZWE              Zimbabwe           32.210                         24.0

     gdp_per_capita  land_boundaries  coastline  num_border_countries  \
2         8274.5430          5369.00     1600.0                   4.0
6        68887.8400          1066.00     1318.0                   2.0
7        26629.5530         11968.00     4989.0                   5.0
13       56981.3950             0.00    25760.0                   0.0
14       65312.0230          2524.00        0.0                   8.0
..              …                …          …                     …
238      11628.6140          4616.00     3444.0                   3.0
243        623.4000          1601.00     1906.0                   2.0
244      14370.2380          5244.00     2798.0                   6.0
245       3591.5642          6043.15        0.0                   8.0
246       3294.8062          3229.00        0.0                   4.0

                                      border_countries  \
2    Democratic Republic of the Congo 2,646 km (of …
6                     Oman 609 km; Saudi Arabia 457 km
7    Bolivia 942 km; Brazil 1,263 km; Chile 6,691 k…
13                                                 NaN
14   Czech Republic 402 km; Germany 801 km; Hungary…
..                                                 …
238   Cambodia 1,158 km; China 1,297 km; Laos 2,161 km
243                 Oman 294 km; Saudi Arabia 1,307 km
244  Botswana 1,969 km; Lesotho 1,106 km; Mozambiqu…
245  Angola 1,065 km; Botswana 0.15 km; Democratic …
246  Botswana 834 km; Mozambique 1,402 km; South Af…

     hospital_beds_per_1000  unemployment      political_regime  gini_index  \
2                      0.75        16.497  electoral_autocracies    0.512640
6                      1.87         2.331     closed_autocracies    0.263990
7                      3.71         9.843  electoral_democracies    0.433141
13                     3.84         5.159    liberal_democracies    0.343326
14                     7.19         4.560    liberal_democracies    0.302104
..                      …            …                      …          …
238                    2.55         1.681  electoral_autocracies    0.367902
243                    0.71        17.202     closed_autocracies    0.367071
244                    2.30        28.468  electoral_democracies    0.630258
245                    2.00         5.542  electoral_autocracies    0.514831
246                    2.00         7.373  electoral_autocracies    0.502564

     population_density    poverty  median_age  land_area_sqkm
2             25.969065  31.122005      16.302       1246700.0
6            132.045270   0.000000      30.834         71020.0
7             16.433529   1.684649      30.763       2736690.0
13             3.312877   0.497094      36.543       7692020.0
14           107.620880   0.640639      42.433         82520.0
```

```
..              …          …          …            …
238        310.034420   0.653778     30.586      313429.0
243         66.502680  19.802757     18.017      527970.0
244         49.120743  20.492558     26.873     1213090.0
245         24.904613  64.349754     16.763      743390.0
246         39.476223  39.754530     17.187      386850.0

[112 rows x 17 columns]
```

[106]:

```
national_policy = national_policy[national_policy['Code'].
  ↪isin(study_sample['Code'])]
national_policy
```

[106]:

```
         Country Code       Date  ConfirmedCases  ConfirmedDeaths  \
2192      Angola  AGO 2020-01-01             0.0              0.0
2193      Angola  AGO 2020-01-02             0.0              0.0
2194      Angola  AGO 2020-01-03             0.0              0.0
2195      Angola  AGO 2020-01-04             0.0              0.0
2196      Angola  AGO 2020-01-05             0.0              0.0
…            …   …        …             …               …
202436  Zimbabwe  ZWE 2022-02-11      231214.0           5374.0
202437  Zimbabwe  ZWE 2022-02-12      231299.0           5374.0
202438  Zimbabwe  ZWE 2022-02-13      231381.0           5374.0
202439  Zimbabwe  ZWE 2022-02-14      231603.0           5374.0
202440  Zimbabwe  ZWE 2022-02-15      231603.0           5374.0

        PopulationVaccinated  StringencyIndex_Average  \
2192                    0.00                     0.00
2193                    0.00                     0.00
2194                    0.00                     0.00
2195                    0.00                     0.00
2196                    0.00                     0.00
…                        …                        …
202436                 20.48                    51.45
202437                 20.51                    51.45
202438                 20.52                    51.45
202439                 20.53                    51.45
202440                 20.54                    51.45

        ContainmentHealthIndex_Average  EconomicSupportIndex
2192                              0.00                   0.0
2193                              0.00                   0.0
2194                              0.00                   0.0
2195                              0.00                   0.0
2196                              0.00                   0.0
…                                  …                    …
202436                           61.05                   0.0
```

```
202437                              61.05                   0.0
202438                              61.05                   0.0
202439                              61.05                   0.0
202440                              61.05                   0.0

[87024 rows x 9 columns]
```

[107]:
```
filtered_mobility = filtered_mobility[filtered_mobility['Code'].
 ↪isin(study_sample['Code'])]
print(filtered_mobility.sample(frac = 0.0001, random_state = 1).compute())
```

```
          Country        Date  retail_and_recreation  grocery_and_pharmacy  \
2925     Barbados  2022-02-13                   32.0                  17.0
2688     Cameroon  2021-06-21                    6.0                  33.0
3070      Estonia  2020-07-06                   15.0                  14.0
637       Georgia  2021-12-28                   34.0                  76.0
3067  South Korea  2020-07-03                   -8.0                   5.0
17011      Mexico  2020-09-15                  -24.0                   1.0
3763       Serbia  2020-10-23                   -7.0                  13.0
3106  El Salvador  2020-08-11                  -47.0                 -24.0

       parks  transit_stations  workplaces  residential Code
2925    51.0               3.0        15.0         -3.0  BRB
2688   -11.0              35.0       -12.0         -2.0  CMR
3070   116.0              -3.0       -37.0          4.0  EST
637     30.0               9.0        -4.0         -3.0  GEO
3067    23.0              -5.0        -1.0          2.0  KOR
17011  -27.0             -40.0       -31.0          9.0  MEX
3763    18.0              -2.0       -15.0         -1.0  SRB
3106   -42.0             -56.0       -45.0         20.0  SLV
```

3 Evaluate each dataframe to get the features that would be used in the analysis

3.1 Mobility data

[108]:
```
# Basic info on the mobility data set
print(f'Shape: {filtered_mobility.shape[0].compute()} rows')
print(f'Columns: {filtered_mobility.columns.tolist()}')
print('Data types')
print(filtered_mobility.dtypes)
```

```
Shape: 81888 rows
Columns: ['Country', 'Date', 'retail_and_recreation', 'grocery_and_pharmacy',
'parks', 'transit_stations', 'workplaces', 'residential', 'Code']
Data types
Country                  string[pyarrow]
```

```
Date                          datetime64[ns]
retail_and_recreation                float64
grocery_and_pharmacy                 float64
parks                                float64
transit_stations                     float64
workplaces                           float64
residential                          float64
Code                         string[pyarrow]
dtype: object
```

[109]:
```python
# Check missing data patterns
missing_counts = filtered_mobility.isnull().sum().compute()
missing_percent = (missing_counts / len(filtered_mobility)) * 100

print("Missing data percentage:")
for col, pct in missing_percent.items():
    print(f"{col}: {pct:.2f}%")
```

```
Missing data percentage:
Country: 0.00%
Date: 0.00%
retail_and_recreation: 0.28%
grocery_and_pharmacy: 0.19%
parks: 0.67%
transit_stations: 0.44%
workplaces: 0.16%
residential: 0.08%
Code: 0.00%
```

[110]:
```python
# Assuming 'Country' column exists in filtered_mobility
# Group by 'Country', calculate missing percentages for each group, and compute␣
↪the result
missing_percent_per_country = (
    filtered_mobility
    .groupby(['Country','Code'])  # Group the Dask DataFrame by 'Country'
    .apply(              # Apply a function to each group
        lambda group: (group.isnull().sum() / len(group)) * 100,
        meta=dict([(col, 'f8') for col in filtered_mobility.columns]) # Provide␣
↪meta for Dask
    )
    .compute()           # Compute the result (triggers Dask execution)
)

# Filter the result to show only countries with at least one column having␣
↪missing data
# A country has missing data if the sum of missing percentages across its␣
↪columns is greater than 0
```

```python
# Or, more simply, if the maximum missing percentage for the country is greater␣
 ↪than 0
countries_with_missing = missing_percent_per_country[
    missing_percent_per_country.max(axis=1) > 0
]

# Optional: Sort countries by the maximum percentage of missing data
sorted_by_max_missing = countries_with_missing.loc[countries_with_missing.
 ↪max(axis=1).sort_values(ascending=False).index]

# Display the results for countries with missing data
if not countries_with_missing.empty:
    print("Missing data percentage per Country (Countries with ANY missing␣
 ↪data):")
    print(sorted_by_max_missing)
else:
    print("No countries found with missing data in any column.")
```

```
Missing data percentage per Country (Countries with ANY missing data):
                       Country  Date  retail_and_recreation  \
Country          Code
Vietnam          VNM       0.0   0.0               0.000000
Gabon            GAB       0.0   0.0               3.415301
Cape Verde       CPV       0.0   0.0               3.017833
Papua New Guinea PNG       0.0   0.0               3.017833
Mongolia         MNG       0.0   0.0               2.595628
Benin            BEN       0.0   0.0               1.775956
Luxembourg       LUX       0.0   0.0               0.000000
Angola           AGO       0.0   0.0               0.000000
Burkina Faso     BFA       0.0   0.0               3.005464
Rwanda           RWA       0.0   0.0               3.415301
Zimbabwe         ZWE       0.0   0.0               0.000000
Botswana         BWA       0.0   0.0               3.415301
Haiti            HTI       0.0   0.0               3.415301
Barbados         BRB       0.0   0.0               0.409836
Mali             MLI       0.0   0.0               0.819672
Namibia          NAM       0.0   0.0               0.000000
Yemen            YEM       0.0   0.0               0.136612
Togo             TGO       0.0   0.0               0.000000
Zambia           ZMB       0.0   0.0               0.000000
Niger            NER       0.0   0.0               3.415301
Estonia          EST       0.0   0.0               0.000000
Malta            MLT       0.0   0.0               0.000000
Slovenia         SVN       0.0   0.0               0.000000
Lebanon          LBN       0.0   0.0               0.000000
Mauritius        MUS       0.0   0.0               0.000000
Norway           NOR       0.0   0.0               0.000000
```

Uganda	UGA	0.0	0.0	0.000000
Poland	POL	0.0	0.0	0.000000
Switzerland	CHE	0.0	0.0	0.000000
Latvia	LVA	0.0	0.0	0.000000

		grocery_and_pharmacy	parks	transit_stations \
Country	Code			
Vietnam	VNM	0.000000	0.000000	0.000000
Gabon	GAB	0.683060	10.245902	3.415301
Cape Verde	CPV	3.017833	3.017833	3.017833
Papua New Guinea	PNG	3.017833	3.017833	3.017833
Mongolia	MNG	3.415301	3.551913	0.000000
Benin	BEN	0.000000	3.415301	3.415301
Luxembourg	LUX	0.273224	3.415301	0.000000
Angola	AGO	0.000000	3.415301	3.415301
Burkina Faso	BFA	0.000000	3.415301	3.415301
Rwanda	RWA	2.459016	3.415301	0.000000
Zimbabwe	ZWE	0.000000	3.415301	0.409836
Botswana	BWA	3.415301	3.415301	3.415301
Haiti	HTI	0.273224	0.000000	3.142077
Barbados	BRB	0.546448	3.415301	3.415301
Mali	MLI	0.000000	3.415301	3.415301
Namibia	NAM	0.000000	3.415301	3.415301
Yemen	YEM	0.000000	3.415301	3.415301
Togo	TGO	0.000000	3.415301	3.415301
Zambia	ZMB	0.000000	3.415301	0.000000
Niger	NER	3.415301	3.415301	3.415301
Estonia	EST	0.000000	2.459016	0.000000
Malta	MLT	0.000000	2.185792	0.000000
Slovenia	SVN	0.000000	1.912568	0.000000
Lebanon	LBN	0.000000	0.000000	1.912568
Mauritius	MUS	0.000000	0.546448	0.000000
Norway	NOR	0.409836	0.000000	0.000000
Uganda	UGA	0.000000	0.000000	0.000000
Poland	POL	0.136612	0.000000	0.000000
Switzerland	CHE	0.136612	0.000000	0.000000
Latvia	LVA	0.000000	0.136612	0.000000

		workplaces	residential	Code
Country	Code			
Vietnam	VNM	16.256831	0.000000	0.0
Gabon	GAB	0.000000	0.000000	0.0
Cape Verde	CPV	0.000000	4.115226	0.0
Papua New Guinea	PNG	0.411523	3.978052	0.0
Mongolia	MNG	0.000000	0.000000	0.0
Benin	BEN	0.000000	0.000000	0.0
Luxembourg	LUX	0.409836	0.273224	0.0
Angola	AGO	0.000000	0.000000	0.0

```
Burkina Faso     BFA     0.000000     0.000000    0.0
Rwanda           RWA     0.000000     0.000000    0.0
Zimbabwe         ZWE     0.000000     0.000000    0.0
Botswana         BWA     0.409836     0.000000    0.0
Haiti            HTI     0.000000     0.000000    0.0
Barbados         BRB     0.000000     0.546448    0.0
Mali             MLI     0.000000     0.000000    0.0
Namibia          NAM     0.000000     0.000000    0.0
Yemen            YEM     0.000000     0.000000    0.0
Togo             TGO     0.000000     0.000000    0.0
Zambia           ZMB     0.000000     0.000000    0.0
Niger            NER     0.000000     0.000000    0.0
Estonia          EST     0.000000     0.000000    0.0
Malta            MLT     0.000000     0.000000    0.0
Slovenia         SVN     0.000000     0.000000    0.0
Lebanon          LBN     0.000000     0.000000    0.0
Mauritius        MUS     0.000000     0.000000    0.0
Norway           NOR     0.000000     0.000000    0.0
Uganda           UGA     0.000000     0.273224    0.0
Poland           POL     0.000000     0.000000    0.0
Switzerland      CHE     0.000000     0.000000    0.0
Latvia           LVA     0.000000     0.000000    0.0
```

Based on this, it seems Vietnam and Gabon have more than 10% missing in one of their mobility features so I would look more into it

[111]:
```
# Check Vietnam and Gabon
vietnam = filtered_mobility[filtered_mobility['Code'] == 'VNM'].compute()
gabon = filtered_mobility[filtered_mobility['Code'] == 'GAB'].compute()
```

[112]:
```
vietnam
```

[112]:
```
      Country       Date  retail_and_recreation  grocery_and_pharmacy  parks  \
1464  Vietnam 2020-02-15                   -6.0                  -4.0  -11.0
1465  Vietnam 2020-02-16                   -9.0                  -7.0   -9.0
1466  Vietnam 2020-02-17                   -9.0                  -7.0   -7.0
1467  Vietnam 2020-02-18                  -11.0                  -4.0   -8.0
1468  Vietnam 2020-02-19                   -9.0                  -9.0   -9.0
…         …          …                     …                     …     …
2191  Vietnam 2022-02-11                   -9.0                   5.0  -10.0
2192  Vietnam 2022-02-12                   -9.0                   8.0  -11.0
2193  Vietnam 2022-02-13                   -7.0                  10.0   -2.0
2194  Vietnam 2022-02-14                    1.0                  23.0    2.0
2195  Vietnam 2022-02-15                  -12.0                  16.0   -6.0

      transit_stations  workplaces  residential Code
1464              -9.0        -4.0          7.0  VNM
1465             -11.0        -7.0          7.0  VNM
```

```
1466             -11.0       5.0         6.0  VNM
1467              -9.0       7.0         6.0  VNM
1468             -12.0       7.0         4.0  VNM
…                  …         …         …   …
2191             -29.0       NaN         0.0  VNM
2192             -26.0       NaN         1.0  VNM
2193             -26.0       NaN         1.0  VNM
2194             -27.0       NaN        -4.0  VNM
2195             -27.0       NaN        -3.0  VNM

[732 rows x 9 columns]
```

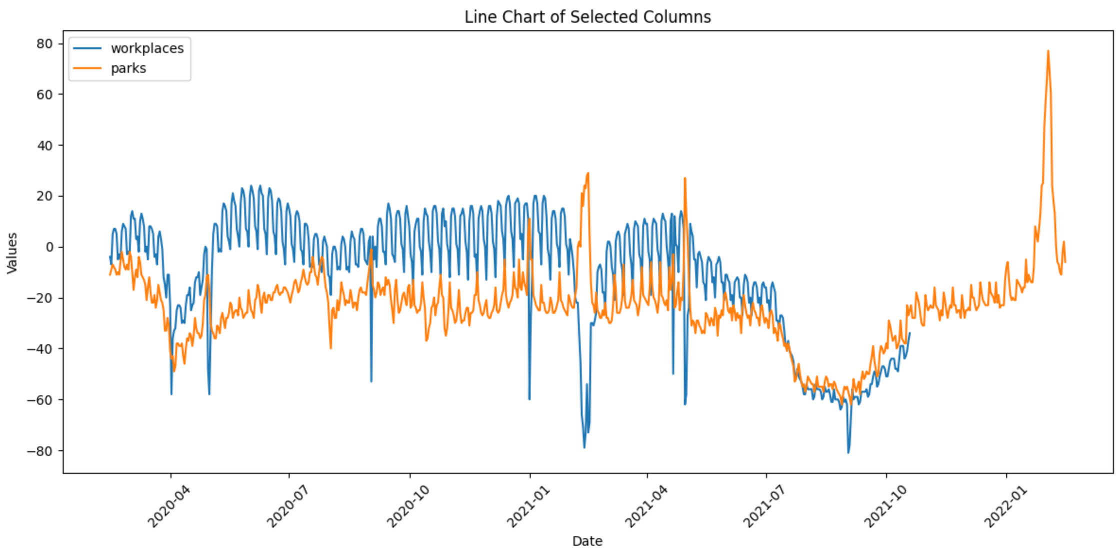
[116]:

```
def plot_type(df, date_column, columns, type):
    """
    Plot selected columns from a DataFrame over time using line or stackplot.

    Parameters:
    - df: DataFrame with a 'date' column
    - date_column: a column for the date
    - columns: list of column names to plot
    - type: 'line' or 'stackplot'
    """
    plt.figure(figsize=(12, 6))

    if type == 'line':
        return [plt.plot(df[date_column], df[col], label=col) for col in␣
 ↪columns]

    elif type == 'stackplot':
        return plt.stackplot(df[date_column], *[df[col] for col in columns],␣
 ↪labels=columns)

def graph_columns(df, date_column, columns, type):
    """
    Generate a time series graph with title and styling.

    Parameters:
    - df: DataFrame
    - columns: list of columns to plot
    - type: 'line' or 'stackplot'
    """
    plot_type(df, date_column, columns, type)
    plt.title(f"{type.capitalize()} Chart of Selected Columns")
    plt.xlabel("Date")
    plt.ylabel("Values")
    plt.legend()
    plt.xticks(rotation=45)
```

```
    plt.tight_layout()
    plt.show()
```

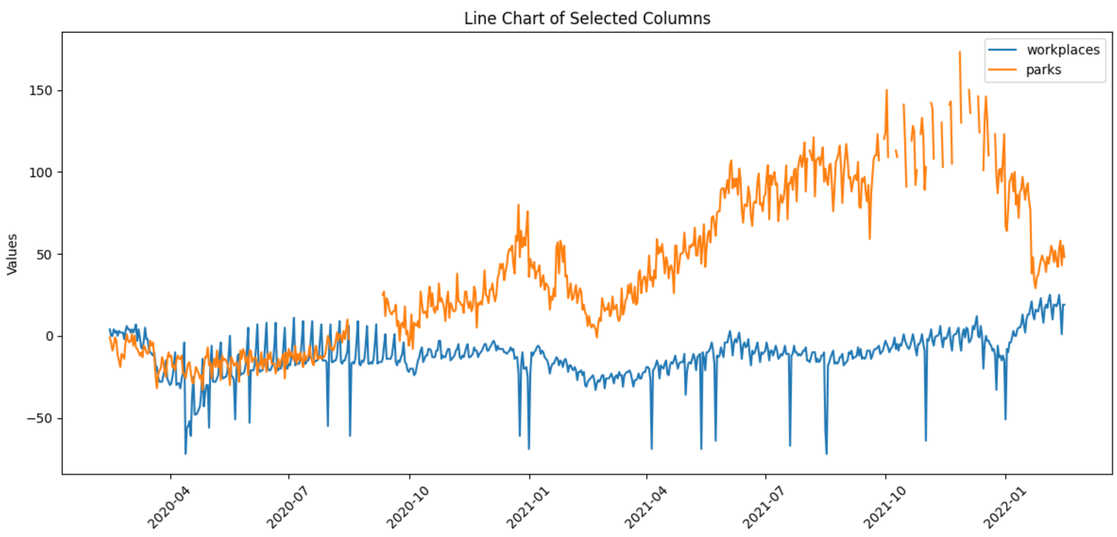
[117]:
```
print("Vietnam")
graph_columns(vietnam,'Date', ['workplaces', 'parks'], "line")
```

```
Vietnam
```



[118]:
```
print("Gabon")
graph_columns(gabon, 'Date', ['workplaces', 'parks'], "line")
```

```
Gabon
```



Based on the graphs, workplace data may not have been collected since around the last quarter of 2021 and parks are missing in some dates in Gabon. In addition, there seems to be a high degrea of day to day variabilty within the week as noticed by our reference study. With this I would use weekly median value to address the variability and might also address the missing values

3.1.1 Get the weekly median for the mobility values

[119]:
```
# 1. Extract year and week number from the 'date' column
# Using isocalendar() is the recommended way as it handles week boundaries␣
↪correctly
# This adds new columns 'year' and 'week'
filtered_mobility_with_week = filtered_mobility.assign(
    year=filtered_mobility['Date'].dt.isocalendar().year,
    week=filtered_mobility['Date'].dt.isocalendar().week
)

# Create the combined 'year_and_week' column (e.g., "2020-W01")
# Using string formatting for clarity and consistency
filtered_mobility_with_week = filtered_mobility_with_week.assign(
    year_and_week=(
        filtered_mobility_with_week['year'].astype(str) +
        "-W" +
        filtered_mobility_with_week['week'].astype(str).str.zfill(2) # Pad week␣
↪with leading zero
    )
)

# 2. Define the mobility columns to aggregate
mobility_cols = [
    'retail_and_recreation',
    'grocery_and_pharmacy',
    'parks',
    'transit_stations',
    'workplaces',
    'residential'
]

# 3. Group by Country, Code, and year_and_week, then calculate the median for␣
↪mobility columns
# Select the relevant columns for grouping and aggregation
grouping_cols = ['Country', 'Code', 'year_and_week']
aggregation_dict = {col: 'median' for col in mobility_cols}

median_mobility_by_week = (
    filtered_mobility_with_week[grouping_cols + mobility_cols]
```

```
    .groupby(grouping_cols)
    .agg(aggregation_dict, split_out=1) # split_out can help with performance␣
↪on large groups
    # .median() # Alternative to .agg(), but .agg() is more explicit for␣
↪multiple columns
)

# 4. Reset index to make 'Country', 'Code', 'year_and_week' regular columns
median_mobility_by_week = median_mobility_by_week.reset_index()

# 5. Compute the result (triggers Dask execution)
# The result will be a pandas DataFrame
weekly_mobility = median_mobility_by_week.compute()

# Display the final result
print("Median mobility values by Country, Code, and Year-Week:")
print(weekly_mobility.head(20)) # Show first 20 rows as an example
```

```
Median mobility values by Country, Code, and Year-Week:
   Country Code year_and_week  retail_and_recreation  grocery_and_pharmacy  \
0    Japan  JPN      2020-W09                   -3.0                   7.0
1    Japan  JPN      2020-W10                  -10.0                   2.0
2    Japan  JPN      2020-W11                   -7.0                   3.0
3    Japan  JPN      2020-W12                   -3.0                   4.0
4    Japan  JPN      2020-W14                  -14.0                   6.0
5    Japan  JPN      2020-W18                  -35.0                  -5.0
6    Japan  JPN      2020-W19                  -34.0                  -5.0
7    Japan  JPN      2020-W20                  -32.0                  -1.0
8    Japan  JPN      2020-W21                  -27.0                  -1.0
9    Japan  JPN      2020-W23                  -18.0                   1.0
10   Japan  JPN      2020-W24                  -16.0                  -1.0
11   Japan  JPN      2020-W25                  -11.0                   2.0
12   Japan  JPN      2020-W26                  -11.0                   1.0
13   Japan  JPN      2020-W28                  -12.0                  -1.0
14   Japan  JPN      2020-W31                  -13.0                   1.0
15   Japan  JPN      2020-W35                  -13.0                   0.0
16   Japan  JPN      2020-W36                  -13.0                  -1.0
17   Japan  JPN      2020-W39                  -12.0                   0.0
18   Japan  JPN      2020-W40                   -9.0                   1.0
19   Japan  JPN      2020-W41                  -11.0                  -2.0

    parks  transit_stations  workplaces  residential
0    -4.0             -10.0         1.0          2.0
1    -8.0             -18.0        -4.0          5.0
2    10.0             -17.0        -4.0          4.0
3    18.0             -15.0        -4.0          3.0
4     9.0             -25.0       -10.0          7.0
```

```
5     4.0              -50.0        -30.0         15.0
6     3.0              -56.0        -27.0         16.0
7     0.0              -44.0        -23.0         13.0
8    -8.0              -42.0        -21.0         12.0
9     2.0              -28.0        -13.0          8.0
10  -10.0              -27.0        -13.0          8.0
11    6.0              -23.0        -12.0          6.0
12   -7.0              -21.0        -12.0          6.0
13  -18.0              -22.0        -12.0          7.0
14   -3.0              -22.0        -12.0          6.0
15   -5.0              -24.0        -12.0          6.0
16  -10.0              -24.0        -12.0          6.0
17  -10.0              -20.0        -10.0          6.0
18    4.0              -17.0         -9.0          4.0
19   -4.0              -18.0         -9.0          4.0
```

[120]:
```
len(weekly_mobility)
```

[120]:
```
11862
```

[121]:
```
# Check missing data patterns
missing_counts = weekly_mobility.isnull().sum()
missing_percent = (missing_counts / len(weekly_mobility)) * 100

print("Missing data percentage:")
for col, pct in missing_percent.items():
    print(f"{col}: {pct:.2f}%")
```

```
Missing data percentage:
Country: 0.00%
Code: 0.00%
year_and_week: 0.00%
retail_and_recreation: 0.18%
grocery_and_pharmacy: 0.13%
parks: 0.46%
transit_stations: 0.34%
workplaces: 0.14%
residential: 0.00%
```

[122]:
```
# Note: For pandas groupby.apply, we don't use the 'meta' argument.
missing_percent_per_group = (
    weekly_mobility
    .groupby(['Country', 'Code'])  # Group the pandas DataFrame by 'Country'␣
 ↪and 'Code'
    .apply(              # Apply a function to each group
        lambda group: (group.isnull().sum() / len(group)) * 100
        # Do NOT include meta=... here for pandas
```

```
    )
    # No .compute() needed for pandas operations
)

# Filter the result to show only groups with at least one column having missing␣
↪data
# A group has missing data if the maximum missing percentage across its columns␣
↪is greater than 0
countries_with_missing = missing_percent_per_group[
    missing_percent_per_group.max(axis=1) > 0
]

sorted_by_max_missing = countries_with_missing.loc[countries_with_missing.
↪max(axis=1).sort_values(ascending=False).index]

# Display the results for groups with missing data
if not countries_with_missing.empty:
    print("Missing data percentage per Country-Code group (Groups with ANY␣
↪missing data):")
    print(sorted_by_max_missing)
else:
    print("No Country-Code groups found with missing data in any column.")
```

```
Missing data percentage per Country-Code group (Groups with ANY missing data):
                       Country  Code  year_and_week  retail_and_recreation  \
Country          Code
Vietnam          VNM       0.0   0.0            0.0               0.000000
Angola           AGO       0.0   0.0            0.0               0.000000
Benin            BEN       0.0   0.0            0.0               0.000000
Barbados         BRB       0.0   0.0            0.0               0.000000
Botswana         BWA       0.0   0.0            0.0               2.830189
Burkina Faso     BFA       0.0   0.0            0.0               0.000000
Gabon            GAB       0.0   0.0            0.0               2.830189
Cape Verde       CPV       0.0   0.0            0.0               2.830189
Luxembourg       LUX       0.0   0.0            0.0               0.000000
Mali             MLI       0.0   0.0            0.0               0.000000
Mongolia         MNG       0.0   0.0            0.0               0.000000
Haiti            HTI       0.0   0.0            0.0               2.830189
Namibia          NAM       0.0   0.0            0.0               0.000000
Niger            NER       0.0   0.0            0.0               2.830189
Rwanda           RWA       0.0   0.0            0.0               2.830189
Papua New Guinea PNG       0.0   0.0            0.0               2.830189
Togo             TGO       0.0   0.0            0.0               0.000000
Yemen            YEM       0.0   0.0            0.0               0.000000
Zambia           ZMB       0.0   0.0            0.0               0.000000
Zimbabwe         ZWE       0.0   0.0            0.0               0.000000
```

```
                   grocery_and_pharmacy     parks  transit_stations  \
Country           Code
Vietnam           VNM             0.000000  0.000000          0.000000
Angola            AGO             0.000000  2.830189          2.830189
Benin             BEN             0.000000  2.830189          2.830189
Barbados          BRB             0.000000  2.830189          2.830189
Botswana          BWA             2.830189  2.830189          2.830189
Burkina Faso      BFA             0.000000  2.830189          2.830189
Gabon             GAB             0.000000  2.830189          2.830189
Cape Verde        CPV             2.830189  2.830189          2.830189
Luxembourg        LUX             0.000000  2.830189          0.000000
Mali              MLI             0.000000  2.830189          2.830189
Mongolia          MNG             2.830189  2.830189          0.000000
Haiti             HTI             0.000000  0.000000          0.943396
Namibia           NAM             0.000000  2.830189          2.830189
Niger             NER             2.830189  2.830189          2.830189
Rwanda            RWA             0.000000  2.830189          0.000000
Papua New Guinea PNG             2.830189  2.830189          2.830189
Togo              TGO             0.000000  2.830189          2.830189
Yemen             YEM             0.000000  2.830189          2.830189
Zambia            ZMB             0.000000  2.830189          0.000000
Zimbabwe          ZWE             0.000000  2.830189          0.000000

                   workplaces  residential
Country           Code
Vietnam           VNM    16.037736          0.0
Angola            AGO     0.000000          0.0
Benin             BEN     0.000000          0.0
Barbados          BRB     0.000000          0.0
Botswana          BWA     0.000000          0.0
Burkina Faso      BFA     0.000000          0.0
Gabon             GAB     0.000000          0.0
Cape Verde        CPV     0.000000          0.0
Luxembourg        LUX     0.000000          0.0
Mali              MLI     0.000000          0.0
Mongolia          MNG     0.000000          0.0
Haiti             HTI     0.000000          0.0
Namibia           NAM     0.000000          0.0
Niger             NER     0.000000          0.0
Rwanda            RWA     0.000000          0.0
Papua New Guinea PNG     0.000000          0.0
Togo              TGO     0.000000          0.0
Yemen             YEM     0.000000          0.0
Zambia            ZMB     0.000000          0.0
Zimbabwe          ZWE     0.000000          0.0

/tmp/ipykernel_931/2281753613.py:5: DeprecationWarning: DataFrameGroupBy.apply
operated on the grouping columns. This behavior is deprecated, and in a future
```

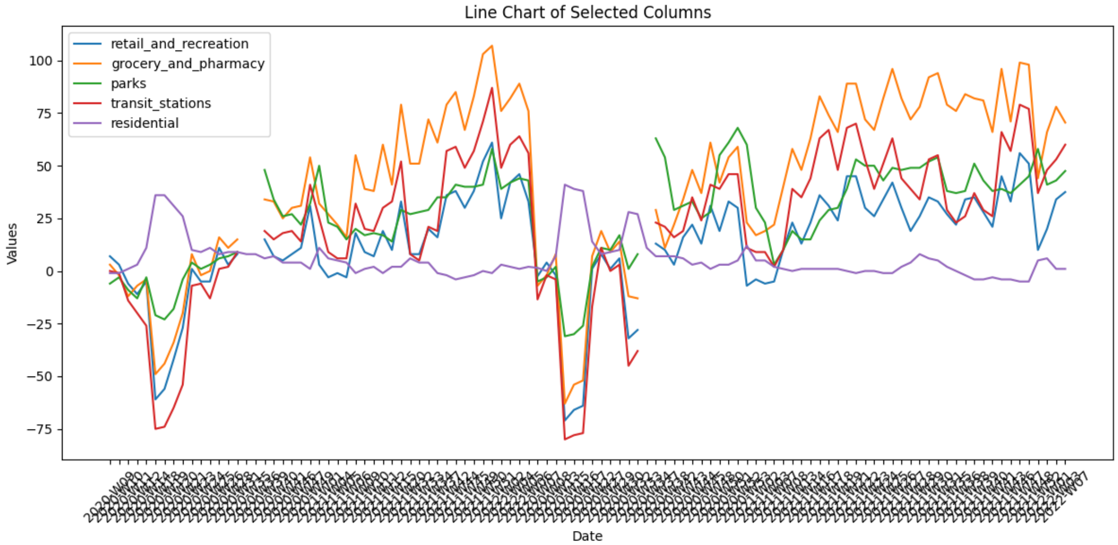
```
version of pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  .apply(              # Apply a function to each group
```

Vietnam still has attribute that has more that 16% missing so I will remove it for the analysis. There are still countries that have missing values in some of their parameters. I will check Botswana as it has 2.83% missing in three of its parameters

[124]:
```
to_keep = study_sample['Code'] != "VNM"
study_sample = study_sample[to_keep]
weekly_mobility = weekly_mobility[weekly_mobility['Code'].
 ↪isin(study_sample['Code'])]
print(weekly_mobility)
```

```
      Country Code year_and_week  retail_and_recreation  grocery_and_pharmacy  \
0       Japan  JPN      2020-W09                   -3.0                   7.0
1       Japan  JPN      2020-W10                  -10.0                   2.0
2       Japan  JPN      2020-W11                   -7.0                   3.0
3       Japan  JPN      2020-W12                   -3.0                   4.0
4       Japan  JPN      2020-W14                  -14.0                   6.0
…         …   …           …                     …                    …
11857  Jordan  JOR      2021-W48                   15.0                  50.0
11858  Jordan  JOR      2021-W52                   16.0                  47.0
11859  Jordan  JOR      2022-W01                   11.0                  46.0
11860  Jordan  JOR      2022-W03                    0.0                  33.0
11861  Jordan  JOR      2022-W07                   11.5                  41.5

       parks  transit_stations  workplaces  residential
0       -4.0             -10.0         1.0          2.0
1       -8.0             -18.0        -4.0          5.0
2       10.0             -17.0        -4.0          4.0
3       18.0             -15.0        -4.0          3.0
4        9.0             -25.0       -10.0          7.0
…      …                 …          …          …
11857    8.0               4.0         3.0          3.0
11858    5.0             -10.0        -1.0          3.0
11859   10.0             -12.0         2.0          3.0
11860   -5.0             -30.0        -2.0          9.0
11861    5.5             -23.0        -1.5          6.0

[11756 rows x 9 columns]
```

[132]:
```
# Check Botswana
country_to_check = weekly_mobility[weekly_mobility['Code'] == 'BWA']
graph_columns(country_to_check, 'year_and_week', ['retail_and_recreation',␣
 ↪'grocery_and_pharmacy', 'parks', 'transit_stations', 'residential'], "line")
```



Missing Data Handling: We identified short-term missing data periods (<3% of observations per country) in mobility variables, typically lasting 1-3 consecutive weeks. Given the low frequency and short duration of these gaps, we employed linear interpolation within each country to maintain data integrity while preserving sample size. This approach is appropriate because: (1) mobility patterns exhibit temporal smoothness, (2) the missing data percentage is minimal, and (3) interpolation preserves the temporal correlation structure of the data. We validated interpolated values to ensure they remained within reasonable bounds relative to surrounding observations.

3.1.2 Interpolate missing values

[134]:
```python
def simple_mobility_interpolation(df):
    """Simple and robust interpolation for your mobility data"""

    # Define mobility variables
    mobility_variables = [
        'retail_and_recreation',
        'grocery_and_pharmacy',
        'parks',
        'transit_stations',
        'workplaces',
        'residential'
    ]

    # Create result dataframe
    result_df = df.copy()

    print("Starting interpolation...")
    print(f"Original missing values: {result_df[mobility_variables].isnull().
 ↪sum().sum()}")
```

```
    # Interpolate each variable using groupby transform (most reliable method)
    for var in mobility_variables:
        if var in result_df.columns:
            print(f"Interpolating {var}...")
            # This is the most robust approach for group-wise interpolation
            result_df[var] = result_df.groupby(['Country', 'Code'])[var].
↪transform(
                lambda x: x.interpolate(method='linear', limit_direction='both')
            )

    # Handle any remaining NAs (countries with all missing values for a␣
↪variable)
    for var in mobility_variables:
        if var in result_df.columns:
            result_df[var] = result_df[var].fillna(method='ffill').
↪fillna(method='bfill')

    final_missing = result_df[mobility_variables].isnull().sum().sum()
    print(f"Final missing values: {final_missing}")
    print("Interpolation complete!")

    return result_df

# Apply the simpler approach
interpolated_mobility_data = simple_mobility_interpolation(weekly_mobility)
```

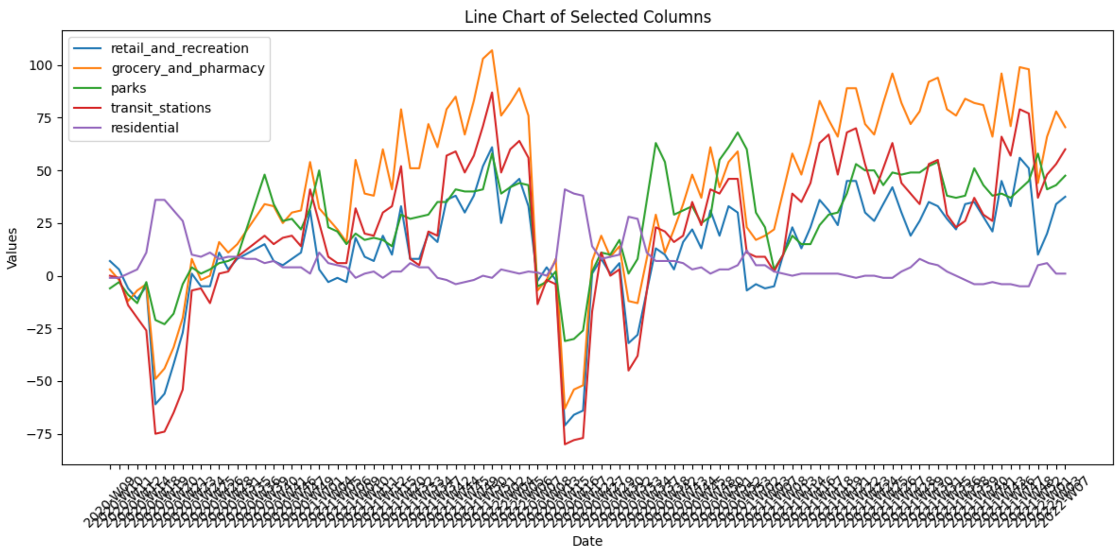
```
Starting interpolation…
Original missing values: 130
Interpolating retail_and_recreation…
Interpolating grocery_and_pharmacy…
Interpolating parks…
Interpolating transit_stations…
Interpolating workplaces…
Interpolating residential…
Final missing values: 0
Interpolation complete!
```

[135]:
```
# Check missing data patterns
missing_counts = interpolated_mobility_data.isnull().sum()
missing_percent = (missing_counts / len(interpolated_mobility_data)) * 100

print("Missing data percentage:")
for col, pct in missing_percent.items():
    print(f"{col}: {pct:.2f}%")
```

```
Missing data percentage:
```

```
Country: 0.00%
Code: 0.00%
year_and_week: 0.00%
retail_and_recreation: 0.00%
grocery_and_pharmacy: 0.00%
parks: 0.00%
transit_stations: 0.00%
workplaces: 0.00%
residential: 0.00%
```

[137]:
```
# Check Botswana
country_to_check =␣
↪interpolated_mobility_data[interpolated_mobility_data['Code'] == 'BWA']
graph_columns(country_to_check, 'year_and_week', ['retail_and_recreation',␣
↪'grocery_and_pharmacy', 'parks', 'transit_stations', 'residential'], "line")
```



3.2 Checking for outliers

[139]:
```
# Check for extreme values in mobility variables
mobility_cols = [
    'retail_and_recreation',
    'grocery_and_pharmacy',
    'parks',
    'transit_stations',
    'workplaces',
    'residential'
]
```

```
# Compute quantiles for outlier detection
for col in mobility_cols:
    quantiles = interpolated_mobility_data[col].quantile([0.01, 0.25, 0.50, 0.
↪75, 0.99])
    print(f"{col} - 1st percentile: {quantiles[0.01]:.2f}, 1st quartile:
↪{quantiles[0.25]:0.2f}, median:{quantiles[0.50]:0.2f}, 3rd quartile:
↪{quantiles[0.75]:0.2f}, 99th percentile: {quantiles[0.99]:.2f}")

    # Winsorize extreme values or set reasonable bounds
    # Mobility changes rarely exceed ±100%
    #filtered_mobility[col] = filtered_mobility[col].clip(lower=-100, upper=100)
```

```
retail_and_recreation - 1st percentile: -78.00, 1st quartile:-26.00,
median:-10.00, 3rd quartile:3.00, 99th percentile: 70.00
grocery_and_pharmacy - 1st percentile: -54.45, 1st quartile:-5.00, median:5.00,
3rd quartile:22.00, 99th percentile: 120.00
parks - 1st percentile: -67.00, 1st quartile:-19.00, median:-1.00, 3rd
quartile:27.00, 99th percentile: 193.00
transit_stations - 1st percentile: -77.00, 1st quartile:-35.00, median:-17.00,
3rd quartile:1.00, 99th percentile: 67.00
workplaces - 1st percentile: -68.00, 1st quartile:-29.00, median:-18.00, 3rd
quartile:-6.00, 99th percentile: 36.45
residential - 1st percentile: -10.00, 1st quartile:1.00, median:6.00, 3rd
quartile:11.00, 99th percentile: 32.00
```

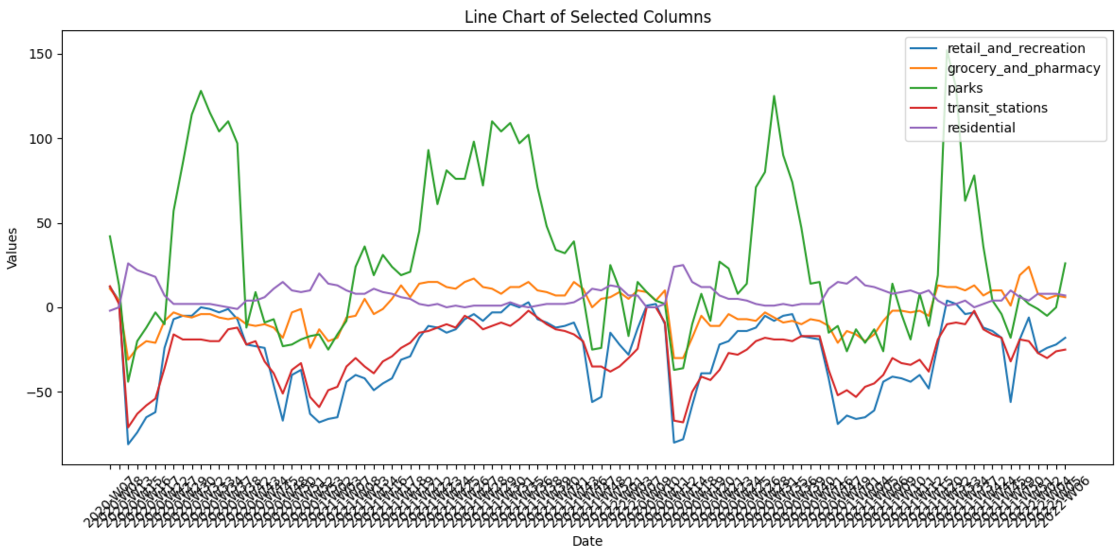
[143]:
```
outlier_threshold = 100

# 2. Initialize a dictionary to store results for each column
countries_with_outliers = {}

# 3. Check each mobility column for outliers and get max values
for column in mobility_cols:
    # Filter the DataFrame to rows where the value exceeds the threshold
    df_outliers_for_column = interpolated_mobility_data[
        interpolated_mobility_data[column] > outlier_threshold
    ]

    # If there are outliers, find the max value and corresponding country code␣
↪for each country
    if not df_outliers_for_column.empty:
        # Group by 'Code' and find the maximum value for this column within␣
↪each group
        max_values_per_country = (
            df_outliers_for_column.groupby('Code')[column].max().reset_index()
        )
        # Convert to list of tuples (Country_Code, Max_Value)
        countries_with_outliers[column] = list(
```

```
            max_values_per_country.itertuples(index=False, name=None)
        )
    else:
        countries_with_outliers[column] = []

# 4. Display the results
print(f"Countries with outlier values (>{outlier_threshold}) in each mobility␣
↪parameter:")
print("-" * 80)
for column, country_data_list in countries_with_outliers.items():
    # Extract the base name of the column for cleaner display (optional)
    # display_name = column.replace('_percent_change_from_baseline', '')
    print(f"\n{column}:")

    if country_data_list:
        # Sort the list by Country Code for consistent output
        for country_code, max_value in sorted(country_data_list, key=lambda x:␣
↪x[0]):
            print(f"  - {country_code}: {max_value:.2f}")
    else:
        print("  - None found")
```

```
Countries with outlier values (>100) in each mobility parameter:
--------------------------------------------------------------------------------

retail_and_recreation:
  - IRQ: 103.00
  - MNG: 103.00
  - YEM: 106.00

grocery_and_pharmacy:
  - BEN: 129.00
  - BFA: 181.00
  - BWA: 107.00
  - CIV: 135.00
  - EGY: 118.00
  - IRQ: 172.00
  - MAR: 105.00
  - MNG: 194.00
  - NPL: 122.00
  - PNG: 120.00
  - YEM: 127.00
  - ZWE: 105.00

parks:
  - AGO: 102.50
  - AUT: 152.00
```

```
  - BEL: 142.00
  - BEN: 138.00
  - BFA: 143.00
  - BGR: 111.00
  - CAN: 183.00
  - CHE: 152.00
  - CZE: 134.00
  - DEU: 156.00
  - DNK: 389.00
  - ESP: 102.00
  - EST: 236.00
  - FIN: 368.00
  - FRA: 241.00
  - GAB: 151.50
  - GBR: 113.00
  - GRC: 308.00
  - HRV: 610.00
  - HUN: 170.00
  - IRL: 131.00
  - ITA: 209.00
  - LTU: 226.00
  - LUX: 161.00
  - LVA: 195.00
  - MLI: 150.50
  - MNG: 101.00
  - NLD: 251.00
  - NOR: 250.00
  - POL: 212.00
  - PRT: 160.00
  - SVK: 186.00
  - SVN: 221.00
  - SWE: 367.00
  - TGO: 135.00
  - TUR: 116.00
  - YEM: 136.00

transit_stations:
  - MNG: 131.50
  - ZWE: 101.00

workplaces:
  - None found

residential:
  - None found
```

[145]:
```
# Check Botswana
country_to_check =␣
↪interpolated_mobility_data[interpolated_mobility_data['Code'] == 'AUT']
graph_columns(country_to_check, 'year_and_week', ['retail_and_recreation',␣
↪'grocery_and_pharmacy', 'parks', 'transit_stations', 'residential'], "line")
```



Great question! Looking at your distributions, I can see some notable outliers, especially in grocery_and_pharmacy (120%) and parks (193%). Let me help you manage these outliers and understand their impact on your composite mobility variable.

Identifying Problematic Outliers Extreme Values Analysis: Parks: 99th percentile = 193% (very extreme!) Grocery & Pharmacy: 99th percentile = 120% (quite high) Others: Within reasonable ranges (-78% to 67%) Recommended Approach: Winsorization Winsorization is the best approach because it:

Caps extreme values rather than removing them Preserves sample size Maintains data distribution shape Reduces influence on composite index

[152]:
```
def manage_mobility_outliers(df):
    """Manage outliers in mobility data using winsorization"""

    # Define reasonable caps based on your distributions
    outlier_caps = {
        'retail_and_recreation': (-80, 75),     # Slightly beyond 1st/99th␣
↪percentiles
        'grocery_and_pharmacy': (-60, 100),      # Cap the 120% extreme values
        'parks': (-70, 150),                    # Significantly cap the 193%␣
↪values
        'transit_stations': (-80, 70),
        'workplaces': (-70, 40),
```

```
        'residential': (-15, 35)
    }

    df_cleaned = df.copy()

    print("Managing outliers...")
    print("Variable                    Before Cap Range   After Cap Range   ␣
↪Observations Capped")
    print("-" * 75)

    for var, (lower_cap, upper_cap) in outlier_caps.items():
        if var in df_cleaned.columns:
            # Count observations that will be capped
            below_count = (df_cleaned[var] < lower_cap).sum()
            above_count = (df_cleaned[var] > upper_cap).sum()
            total_capped = below_count + above_count

            # Apply winsorization
            df_cleaned[var] = df_cleaned[var].clip(lower=lower_cap,␣
↪upper=upper_cap)

            # Show impact
            if total_capped > 0:
                print(f"{var:<30} [{df[var].min():4.0f}, {df[var].max():4.0f}] ␣
↪ [{lower_cap:4.0f}, {upper_cap:4.0f}]    {total_capped:>6}")

    return df_cleaned

# Apply outlier management
mobility_data_cleaned = manage_mobility_outliers(interpolated_mobility_data)
```

```
Managing outliers…
Variable                    Before Cap Range   After Cap Range    Observations
Capped
---------------------------------------------------------------------------
retail_and_recreation          [ -92,  106]     [ -80,   75]        180
grocery_and_pharmacy           [ -87,  194]     [ -60,  100]        312
parks                          [ -93,  610]     [ -70,  150]        349
transit_stations               [ -92,  132]     [ -80,   70]        141
workplaces                     [ -87,   92]     [ -70,   40]        156
residential                    [ -15,   44]     [ -15,   35]         57
```

[153]:
```
len(interpolated_mobility_data)
```

[153]: 11756

[155]:
```python
# Quick before/after comparison of key statistics
print("=== KEY IMPROVEMENTS ===")
print("Variable: Max Reduction | Std Dev Reduction")
print("-" * 45)

for var in mobility_cols:
    orig_max = interpolated_mobility_data[var].max()
    clean_max = mobility_data_cleaned[var].max()
    orig_std = interpolated_mobility_data[var].std()
    clean_std = mobility_data_cleaned[var].std()

    max_reduction = orig_max - clean_max
    std_reduction = orig_std - clean_std

    print(f"{var}: {max_reduction:>6.1f} | {std_reduction:>6.2f}")
```

```
=== KEY IMPROVEMENTS ===
Variable: Max Reduction | Std Dev Reduction
---------------------------------------------
retail_and_recreation:   31.0 |   0.43
grocery_and_pharmacy:   94.0 |   2.09
parks:  460.0 |   6.57
transit_stations:   61.5 |   0.35
workplaces:   52.0 |   0.40
residential:    9.0 |   0.06
```

[158]:
```python
def compare_composite_before_after(df_before, df_after):
    """Compare composite mobility index before and after outlier management"""

    # Create composite indices
    weights = {
        'retail_and_recreation': 0.25,
        'grocery_and_pharmacy': 0.15,
        'parks': 0.10,
        'transit_stations': 0.20,
        'workplaces': 0.30
    }

    def create_composite(df_chunk):
        weighted_sum = sum(df_chunk[col] * weight for col, weight in weights.
↪items())
        return weighted_sum / sum(weights.values())

    composite_before = df_before.apply(create_composite, axis=1)
    composite_after = df_after.apply(create_composite, axis=1)

    # Compare distributions
```

```python
    print("=== COMPOSITE MOBILITY INDEX COMPARISON ===")
    print("Metric           Before     After     Change")
    print("-" * 45)

    metrics = ['min', '25%', '50%', '75%', 'max', 'mean', 'std']

    for metric in metrics:
        if metric == 'min':
            before_val = composite_before.min()
            after_val = composite_after.min()
        elif metric == 'max':
            before_val = composite_before.max()
            after_val = composite_after.max()
        elif metric == 'mean':
            before_val = composite_before.mean()
            after_val = composite_after.mean()
        elif metric == 'std':
            before_val = composite_before.std()
            after_val = composite_after.std()
        else:
            pct = float(metric.replace('%', '')) / 100
            before_val = composite_before.quantile(pct)
            after_val = composite_after.quantile(pct)

        change = after_val - before_val
        print(f"{metric:<12} {before_val:>8.2f} {after_val:>8.2f} {change:>8.
↪2f}")

    # Check extreme values
    extreme_before = (composite_before > 50) | (composite_before < -50)
    extreme_after = (composite_after > 50) | (composite_after < -50)

    print(f"\nExtreme composite values (>50 or <-50):")
    print(f"  Before: {extreme_before.sum()} observations")
    print(f"  After:  {extreme_after.sum()} observations")

# Run comparison
compare_composite_before_after(interpolated_mobility_data,␣
↪mobility_data_cleaned)
```

```
=== COMPOSITE MOBILITY INDEX COMPARISON ===
Metric           Before     After     Change
---------------------------------------------
min            -88.40    -73.00     15.40
25%            -22.00    -22.00      0.00
50%             -8.72     -8.75     -0.03
75%              3.70      3.50     -0.20
```

```
max            111.40    71.45   -39.95
mean            -8.42    -8.68    -0.26
std             23.78    23.04    -0.74

Extreme composite values (>50 or <-50):
  Before: 742 observations
  After:  713 observations
```

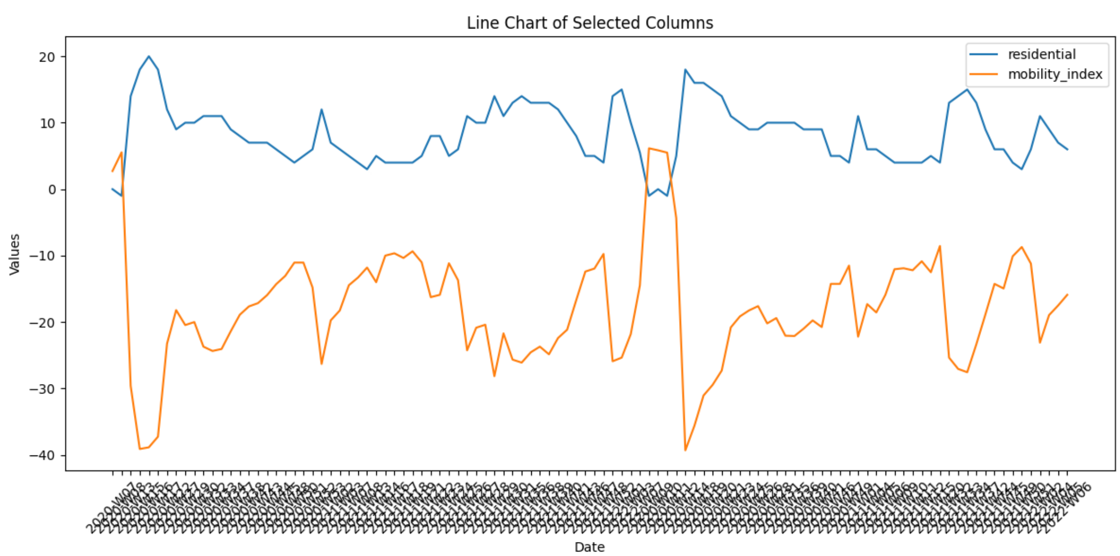
3.3 Create a composite mobility index

[160]:
```python
# Create composite indices
weights = {
    'retail_and_recreation': 0.25,
    'grocery_and_pharmacy': 0.15,
    'parks': 0.10,
    'transit_stations': 0.20,
    'workplaces': 0.30
}

def create_composite(df_chunk):
    weighted_sum = sum(df_chunk[col] * weight for col, weight in weights.
 ↪items())
    return weighted_sum / sum(weights.values())

mobility_data_cleaned['mobility_index'] =  mobility_data_cleaned.
 ↪apply(create_composite, axis = 1)
mobility_data_cleaned
```

[160]:
```
      Country Code year_and_week  retail_and_recreation  grocery_and_pharmacy  \
0       Japan  JPN      2020-W09                   -3.0                   7.0
1       Japan  JPN      2020-W10                  -10.0                   2.0
2       Japan  JPN      2020-W11                   -7.0                   3.0
3       Japan  JPN      2020-W12                   -3.0                   4.0
4       Japan  JPN      2020-W14                  -14.0                   6.0
…         …   …           …                     …                    …
11857  Jordan  JOR      2021-W48                   15.0                  50.0
11858  Jordan  JOR      2021-W52                   16.0                  47.0
11859  Jordan  JOR      2022-W01                   11.0                  46.0
11860  Jordan  JOR      2022-W03                    0.0                  33.0
11861  Jordan  JOR      2022-W07                   11.5                  41.5

        parks  transit_stations  workplaces  residential  mobility_index
0        -4.0             -10.0         1.0          2.0           -1.80
1        -8.0             -18.0        -4.0          5.0           -7.80
2        10.0             -17.0        -4.0          4.0           -4.90
3        18.0             -15.0        -4.0          3.0           -2.55
4         9.0             -25.0       -10.0          7.0           -9.70
```

```
...     ...             ...          ...          ...          ...             
11857    8.0             4.0         3.0          3.0           13.75
11858    5.0           -10.0        -1.0          3.0            9.25
11859   10.0           -12.0         2.0          3.0            8.85
11860   -5.0           -30.0        -2.0          9.0           -2.15
11861    5.5           -23.0        -1.5          6.0            4.60

[11756 rows x 10 columns]
```

[221]:
```
# Check Botswana
country_to_check = mobility_data_cleaned[mobility_data_cleaned['Code'] == 'AUS']
graph_columns(country_to_check, 'year_and_week', [
    #'retail_and_recreation',
    #'grocery_and_pharmacy',
    #'parks',
    #'transit_stations',
    #'workplaces',
    'residential',
    'mobility_index'], "line")
```



[168]:
```
# save to csv
mobility_data_cleaned.to_csv('data/cleaned/Mobility.csv')
```

4 National policy

[169]:
```python
# update the national policy dataframe with changes on the study sample
national_policy = national_policy[national_policy['Code'].
 ↪isin(study_sample['Code'])]
national_policy
```

[169]:
```
          Country Code        Date  ConfirmedCases  ConfirmedDeaths  \
2192       Angola  AGO  2020-01-01             0.0              0.0
2193       Angola  AGO  2020-01-02             0.0              0.0
2194       Angola  AGO  2020-01-03             0.0              0.0
2195       Angola  AGO  2020-01-04             0.0              0.0
2196       Angola  AGO  2020-01-05             0.0              0.0
…             …   …          …              …                …
202436   Zimbabwe  ZWE 2022-02-11        231214.0           5374.0
202437   Zimbabwe  ZWE 2022-02-12        231299.0           5374.0
202438   Zimbabwe  ZWE 2022-02-13        231381.0           5374.0
202439   Zimbabwe  ZWE 2022-02-14        231603.0           5374.0
202440   Zimbabwe  ZWE 2022-02-15        231603.0           5374.0

        PopulationVaccinated  StringencyIndex_Average  \
2192                    0.00                     0.00
2193                    0.00                     0.00
2194                    0.00                     0.00
2195                    0.00                     0.00
2196                    0.00                     0.00
…                        …                       …
202436                 20.48                    51.45
202437                 20.51                    51.45
202438                 20.52                    51.45
202439                 20.53                    51.45
202440                 20.54                    51.45

        ContainmentHealthIndex_Average  EconomicSupportIndex
2192                              0.00                   0.0
2193                              0.00                   0.0
2194                              0.00                   0.0
2195                              0.00                   0.0
2196                              0.00                   0.0
…                                  …                    …
202436                           61.05                   0.0
202437                           61.05                   0.0
202438                           61.05                   0.0
202439                           61.05                   0.0
202440                           61.05                   0.0

[86247 rows x 9 columns]
```

4.0.1 Check for missing

[171]:
```
# Check missing data patterns
missing_counts = national_policy.isnull().sum()
missing_percent = (missing_counts / len(national_policy)) * 100

print("Missing data percentage:")
for col, pct in missing_percent.items():
    print(f"{col}: {pct:.2f}%")
```

```
Missing data percentage:
Country: 0.00%
Code: 0.00%
Date: 0.00%
ConfirmedCases: 0.00%
ConfirmedDeaths: 0.00%
PopulationVaccinated: 0.00%
StringencyIndex_Average: 0.00%
ContainmentHealthIndex_Average: 0.00%
EconomicSupportIndex: 0.00%
```

4.0.2 Check for outliers

[173]:
```
# Check for extreme values in mobility variables
policy_cols = [
    'ConfirmedCases',
    'ConfirmedDeaths',
    'PopulationVaccinated',
    'StringencyIndex_Average',
    'ContainmentHealthIndex_Average',
    'EconomicSupportIndex'
]

# Compute quantiles for outlier detection
for col in policy_cols:
    quantiles = national_policy[col].quantile([0.01, 0.25, 0.50, 0.75, 0.99])
    print(f"{col} - 1st percentile: {quantiles[0.01]:.2f}, 1st quartile:
 ↪{quantiles[0.25]:0.2f}, median:{quantiles[0.50]:0.2f}, 3rd quartile:
 ↪{quantiles[0.75]:0.2f}, 99th percentile: {quantiles[0.99]:.2f}")
```

```
ConfirmedCases - 1st percentile: 0.00, 1st quartile:5064.00, median:82130.00,
3rd quartile:511385.00, 99th percentile: 20900049.88
ConfirmedDeaths - 1st percentile: 0.00, 1st quartile:91.00, median:1329.00, 3rd
quartile:10512.00, 99th percentile: 426153.38
PopulationVaccinated - 1st percentile: 0.00, 1st quartile:0.00, median:0.00, 3rd
quartile:9.49, 99th percentile: 80.01
StringencyIndex_Average - 1st percentile: 0.00, 1st quartile:37.96,
median:52.78, 3rd quartile:69.91, 99th percentile: 96.30
```

```
ContainmentHealthIndex_Average - 1st percentile: 0.00, 1st quartile:44.05,
median:56.37, 3rd quartile:66.37, 99th percentile: 83.45
EconomicSupportIndex - 1st percentile: 0.00, 1st quartile:0.00, median:50.00,
3rd quartile:75.00, 99th percentile: 100.00
```

[175]:
```
national_policy
```

[175]:
```
          Country Code       Date  ConfirmedCases  ConfirmedDeaths  \
2192       Angola  AGO 2020-01-01             0.0              0.0
2193       Angola  AGO 2020-01-02             0.0              0.0
2194       Angola  AGO 2020-01-03             0.0              0.0
2195       Angola  AGO 2020-01-04             0.0              0.0
2196       Angola  AGO 2020-01-05             0.0              0.0
…             …   …          …              …                …
202436   Zimbabwe  ZWE 2022-02-11        231214.0           5374.0
202437   Zimbabwe  ZWE 2022-02-12        231299.0           5374.0
202438   Zimbabwe  ZWE 2022-02-13        231381.0           5374.0
202439   Zimbabwe  ZWE 2022-02-14        231603.0           5374.0
202440   Zimbabwe  ZWE 2022-02-15        231603.0           5374.0

        PopulationVaccinated  StringencyIndex_Average  \
2192                    0.00                     0.00
2193                    0.00                     0.00
2194                    0.00                     0.00
2195                    0.00                     0.00
2196                    0.00                     0.00
…                        …                        …
202436                 20.48                    51.45
202437                 20.51                    51.45
202438                 20.52                    51.45
202439                 20.53                    51.45
202440                 20.54                    51.45

        ContainmentHealthIndex_Average  EconomicSupportIndex
2192                              0.00                   0.0
2193                              0.00                   0.0
2194                              0.00                   0.0
2195                              0.00                   0.0
2196                              0.00                   0.0
…                                  …                    …
202436                           61.05                   0.0
202437                           61.05                   0.0
202438                           61.05                   0.0
202439                           61.05                   0.0
202440                           61.05                   0.0

[86247 rows x 9 columns]
```

There seems no outliers, since Covid and Vaccination statistics are different from policy, I would seperate them

[177]:
```
covid_stats = national_policy[['Country', 'Code', 'Date','ConfirmedCases',␣
↪'ConfirmedDeaths', 'PopulationVaccinated']]
policy_index = national_policy[['Country', 'Code', 'Date',␣
↪'StringencyIndex_Average', 'ContainmentHealthIndex_Average',␣
↪'EconomicSupportIndex']]
```

4.0.3 Create a pure health systems index

[180]:
```
def create_pure_health_index(df):
    """Create pure health index accounting for different indicator counts"""

    # Your mathematically correct formula
    df_result = df.copy()
    df_result['pure_health_index'] = (
        (df_result['ContainmentHealthIndex_Average'] * 14) -
        (df_result['StringencyIndex_Average'] * 9)
    ) / 5

    # Validate the results
    print("=== PURE HEALTH INDEX VALIDATION ===")
    print(f"Pure Health Index Range: {df_result['pure_health_index'].min():.2f}␣
↪to {df_result['pure_health_index'].max():.2f}")
    print(f"Pure Health Index Mean: {df_result['pure_health_index'].mean():.
↪2f}")
    print(f"Pure Health Index Std Dev: {df_result['pure_health_index'].std():.
↪2f}")

    # Check for negative values (should be minimal)
    negative_count = (df_result['pure_health_index'] < 0).sum()
    if negative_count > 0:
        print(f"Warning: {negative_count} negative values (likely rounding␣
↪errors)")
        # Clip negative values to 0
        df_result['pure_health_index'] = df_result['pure_health_index'].
↪clip(lower=0)

    # Verify the calculation makes sense
    countries_with_health_policies = (df_result['pure_health_index'] > 0).sum()
    print(f"Countries/periods with health policies:␣
↪{countries_with_health_policies}")

    return df_result

# Apply the formula
```

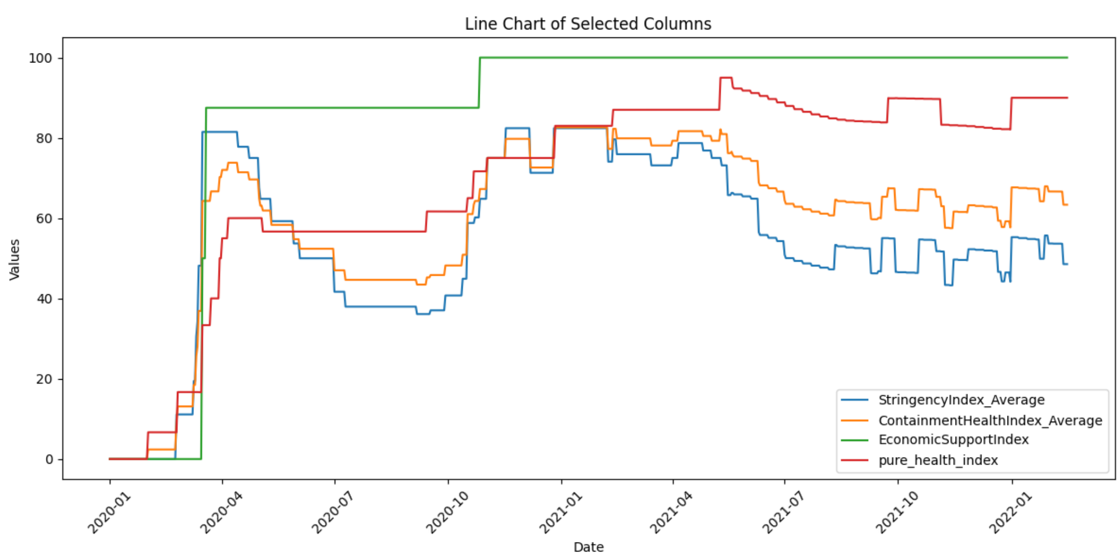
```
policy_data_with_pure_health = create_pure_health_index(policy_index)
```

```
=== PURE HEALTH INDEX VALIDATION ===
Pure Health Index Range: -0.02 to 97.49
Pure Health Index Mean: 53.92
Pure Health Index Std Dev: 22.95
Warning: 621 negative values (likely rounding errors)
Countries/periods with health policies: 81997
```

[181]:
```
# Since policies can not be negative and -0.02 most likely a rounding error, I␣
↪would clip it to 0
policy_data_with_pure_health['pure_health_index'] =␣
↪policy_data_with_pure_health['pure_health_index'].clip(lower=0)
```

4.0.4 Check for day to day variability

[183]:
```
# Check Botswana
country_to_check =␣
↪policy_data_with_pure_health[policy_data_with_pure_health['Code'] == 'AUT']
graph_columns(country_to_check, 'Date', [
    'StringencyIndex_Average',
    'ContainmentHealthIndex_Average',
    'EconomicSupportIndex',
    'pure_health_index'
    ], "line")
```



The indices appear relatively smooth over time, with no extreme outliers or heavy skewness. There are no visible outliers that would significantly distort the average.

4.0.5 Create a weekly policy indices

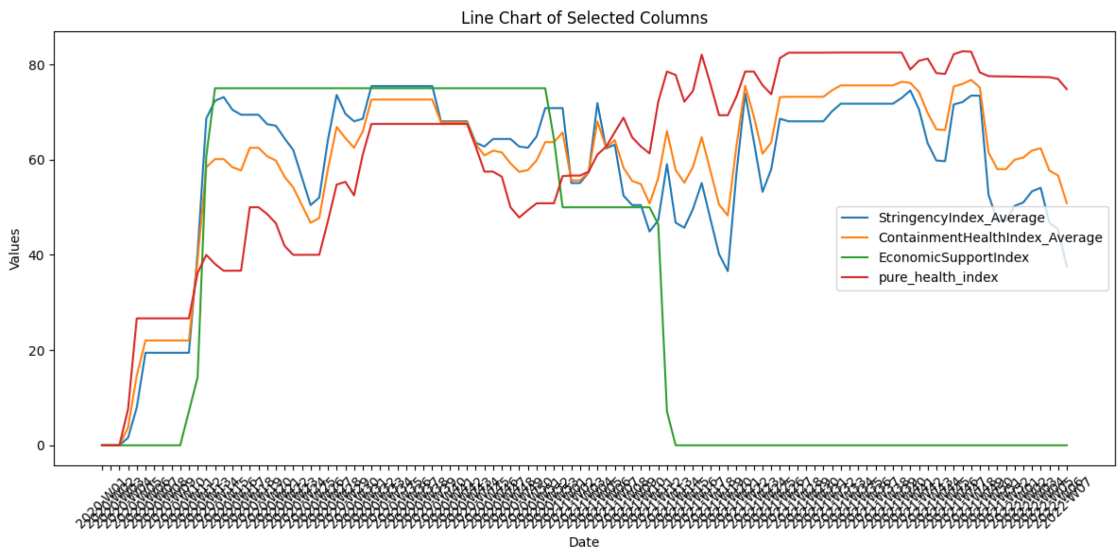
[185]:
```
# 1. Extract year and week number from the 'date' column
# Using isocalendar() is the recommended way as it handles week boundaries␣
↪correctly
# This adds new columns 'year' and 'week'
policy_with_week = policy_data_with_pure_health.assign(
    year=policy_data_with_pure_health['Date'].dt.isocalendar().year,
    week=policy_data_with_pure_health['Date'].dt.isocalendar().week
)

# Create the combined 'year_and_week' column (e.g., "2020-W01")
# Using string formatting for clarity and consistency
policy_with_week = policy_with_week.assign(
    year_and_week=(
        policy_with_week['year'].astype(str) +
        "-W" +
        policy_with_week['week'].astype(str).str.zfill(2) # Pad week with␣
↪leading zero
    )
)

# 2. Define the policy columns to aggregate
policy_cols = [
    'StringencyIndex_Average',
    'ContainmentHealthIndex_Average',
    'EconomicSupportIndex',
    'pure_health_index'
]

# 3. Group by Country, Code, and year_and_week, then calculate the mean for␣
↪policy columns
# Select the relevant columns for grouping and aggregation
grouping_cols = ['Country', 'Code', 'year_and_week']
aggregation_dict = {col: 'mean' for col in policy_cols}

mean_policy_by_week = (
    policy_with_week[grouping_cols + policy_cols]
    .groupby(grouping_cols)
    .agg(aggregation_dict, split_out=1) # split_out can help with performance␣
↪on large groups
    # .median() # Alternative to .agg(), but .agg() is more explicit for␣
↪multiple columns
)

# 4. Reset index to make 'Country', 'Code', 'year_and_week' regular columns
mean_policy_by_week = mean_policy_by_week.reset_index()
```

```python
# Display the final result
print("Mean policy values by Country, Code, and Year-Week:")
print(mean_policy_by_week.head(20)) # Show first 20 rows as an example
```

```
Mean policy values by Country, Code, and Year-Week:
   Country Code year_and_week  StringencyIndex_Average  \
0   Angola  AGO      2020-W01                 0.000000
1   Angola  AGO      2020-W02                 0.000000
2   Angola  AGO      2020-W03                 0.000000
3   Angola  AGO      2020-W04                 0.000000
4   Angola  AGO      2020-W05                 0.000000
5   Angola  AGO      2020-W06                 3.177143
6   Angola  AGO      2020-W07                 5.560000
7   Angola  AGO      2020-W08                 5.560000
8   Angola  AGO      2020-W09                 5.955714
9   Angola  AGO      2020-W10                 8.330000
10  Angola  AGO      2020-W11                 8.330000
11  Angola  AGO      2020-W12                 9.521429
12  Angola  AGO      2020-W13                54.760000
13  Angola  AGO      2020-W14                90.740000
14  Angola  AGO      2020-W15                90.740000
15  Angola  AGO      2020-W16                90.740000
16  Angola  AGO      2020-W17                83.860000
17  Angola  AGO      2020-W18                78.700000
18  Angola  AGO      2020-W19                77.512857
19  Angola  AGO      2020-W20                76.722857

    ContainmentHealthIndex_Average  EconomicSupportIndex  pure_health_index
0                         0.000000              0.000000           0.000000
1                         0.000000              0.000000           0.000000
2                         0.000000              0.000000           0.000000
3                         0.000000              0.000000           0.000000
4                         0.000000              0.000000           0.000000
5                         2.040000              0.000000           0.000000
6                         3.570000              0.000000           0.000000
7                         3.570000              0.000000           0.000000
8                         5.525714              0.000000           4.755143
9                         7.740000              0.000000           6.678000
10                        7.740000              0.000000           6.678000
11                        8.502857              0.000000           6.669429
12                       37.582857              0.000000           6.664000
13                       62.417143              0.000000          11.436000
14                       66.157143             28.571429          21.908000
15                       68.450000             50.000000          28.328000
16                       64.027143             60.714286          28.328000
17                       60.710000             75.000000          28.328000
```

```
18                         59.947143             75.000000          28.328857
19                         59.440000             75.000000          28.330857
```

[201]:
```
# Check Botswana
country_to_check = mean_policy_by_week[mean_policy_by_week['Code'] == 'AUS']
graph_columns(country_to_check, 'year_and_week', [
    'StringencyIndex_Average',
    'ContainmentHealthIndex_Average',
    'EconomicSupportIndex',
    'pure_health_index'
    ], "line")
```



[192]:
```
len(mean_policy_by_week['Country'].unique())
```

[192]: 111

[199]:
```
mean_policy_by_week.groupby(['Country', 'Code']).count()
```

[199]:
```
                          year_and_week  StringencyIndex_Average  \
Country        Code
Angola         AGO                  112                      112
Argentina      ARG                  112                      112
Australia      AUS                  112                      112
Austria        AUT                  112                      112
Bangladesh     BGD                  112                      112
…                                     …                        …
United States  USA                  112                      112
Uruguay        URY                  112                      112
Yemen          YEM                  112                      112
```

```
Zambia        ZMB              112                      112
Zimbabwe      ZWE              112                      112

                    ContainmentHealthIndex_Average  EconomicSupportIndex  \
Country       Code
Angola        AGO                             112                   112
Argentina     ARG                             112                   112
Australia     AUS                             112                   112
Austria       AUT                             112                   112
Bangladesh    BGD                             112                   112
…                                             …                   …
United States USA                             112                   112
Uruguay       URY                             112                   112
Yemen         YEM                             112                   112
Zambia        ZMB                             112                   112
Zimbabwe      ZWE                             112                   112

                    pure_health_index
Country       Code
Angola        AGO                 112
Argentina     ARG                 112
Australia     AUS                 112
Austria       AUT                 112
Bangladesh    BGD                 112
…                                 …
United States USA                 112
Uruguay       URY                 112
Yemen         YEM                 112
Zambia        ZMB                 112
Zimbabwe      ZWE                 112

[111 rows x 5 columns]
```

[202]:
```
# Save the policy data frame
mean_policy_by_week.to_csv('data/cleaned/National_policy.csv')
```

4.1 Covid stat

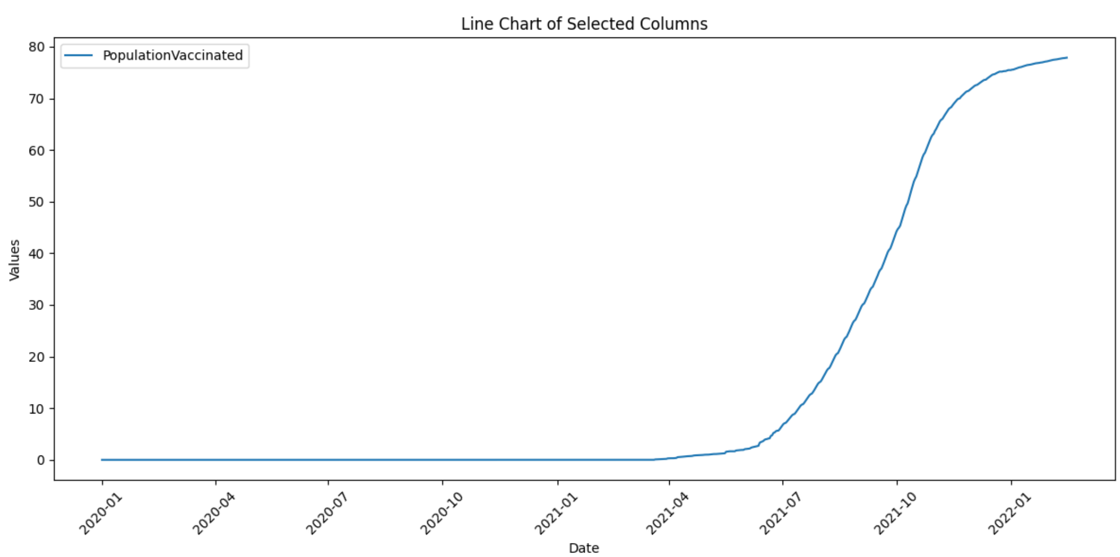
[203]:
```
covid_stats
```

[203]:
```
         Country Code        Date  ConfirmedCases  ConfirmedDeaths  \
2192      Angola  AGO  2020-01-01             0.0              0.0
2193      Angola  AGO  2020-01-02             0.0              0.0
2194      Angola  AGO  2020-01-03             0.0              0.0
2195      Angola  AGO  2020-01-04             0.0              0.0
2196      Angola  AGO  2020-01-05             0.0              0.0
…            …    …           …               …                …
```

```
202436  Zimbabwe  ZWE 2022-02-11        231214.0          5374.0
202437  Zimbabwe  ZWE 2022-02-12        231299.0          5374.0
202438  Zimbabwe  ZWE 2022-02-13        231381.0          5374.0
202439  Zimbabwe  ZWE 2022-02-14        231603.0          5374.0
202440  Zimbabwe  ZWE 2022-02-15        231603.0          5374.0

        PopulationVaccinated
2192                    0.00
2193                    0.00
2194                    0.00
2195                    0.00
2196                    0.00
…                      …
202436                 20.48
202437                 20.51
202438                 20.52
202439                 20.53
202440                 20.54

[86247 rows x 6 columns]
```

[214]:

```
# Check Botswana
country_to_check = covid_stats[covid_stats['Code'] == 'AUS']
graph_columns(country_to_check, 'Date', [
    #'ConfirmedCases',
    #'ConfirmedDeaths',
    'PopulationVaccinated'
    ], "line")
```



[205]:
```python
population = pd.read_csv('data/population.csv')
```

[206]:
```python
population
```

[206]:
```
                    Country Name Country Code         2019
0                          Aruba          ABW     109203.0
1    Africa Eastern and Southern          AFE  675950189.0
2                    Afghanistan          AFG   37856121.0
3     Africa Western and Central          AFW  463365429.0
4                         Angola          AGO   32375632.0
..                           ...          ...          ...
254                       Kosovo          XKX    1788891.0
255                  Yemen, Rep.          YEM   35111408.0
256                 South Africa          ZAF   59587885.0
257                       Zambia          ZMB   18513839.0
258                     Zimbabwe          ZWE   15271368.0

[259 rows x 3 columns]
```

[208]:
```python
population = population.rename(columns = {
    'Country Name' : 'Country',
    'Country Code' : 'Code',
    '2019' : 'Population'
})

population = population[population['Code'].isin(study_sample['Code'])]

population = population.drop(columns = 'Country')

population
```

[208]:
```
    Code   Population
4    AGO   32375632.0
8    ARE    9445785.0
9    ARG   44973465.0
13   AUS   25334826.0
14   AUT    8879920.0
..   ...          ...
244  USA  330226227.0
255  YEM   35111408.0
256  ZAF   59587885.0
257  ZMB   18513839.0
258  ZWE   15271368.0

[111 rows x 2 columns]
```

[209]:
```python
def create_weekly_covid_simple(covid_df, population_df):
    """Simplified approach using pandas resample"""

    # Prepare data
    covid_df['Date'] = pd.to_datetime(covid_df['Date'])

    # Merge with population
    covid_merged = covid_df.merge(
        population_df[['Code', 'Population']],
        on='Code',
        how='left'
    )

    # Set date as index for resampling
    covid_indexed = covid_merged.set_index('Date')

    weekly_data_list = []

    # Process each country
    for country_code in covid_merged['Code'].unique():
        country_data = covid_indexed[covid_indexed['Code'] == country_code].
↪copy()

        if len(country_data) > 0:
            # Resample to weekly (take last value of each week)
            weekly_country = country_data.resample('W').last()

            # Calculate differences for incidence
            weekly_country['case_incidence'] = weekly_country['ConfirmedCases'].
↪diff()
            weekly_country['death_incidence'] =␣
↪weekly_country['ConfirmedDeaths'].diff()

            # First week will be NaN, use the cumulative value
            weekly_country['case_incidence'].iloc[0] =␣
↪weekly_country['ConfirmedCases'].iloc[0]
            weekly_country['death_incidence'].iloc[0] =␣
↪weekly_country['ConfirmedDeaths'].iloc[0]

            # Calculate per 100k
            population = weekly_country['Population'].iloc[0]
            weekly_country['case_incidence_per_100k'] =␣
↪(weekly_country['case_incidence'] / population) * 100000
            weekly_country['death_incidence_per_100k'] =␣
↪(weekly_country['death_incidence'] / population) * 100000
```

```python
            # Create year_and_week format
            weekly_country['year_and_week'] = (
                weekly_country.index.isocalendar().year.astype(str) +
                "-W" +
                weekly_country.index.isocalendar().week.astype(str).str.zfill(2)
            )

            # Clean and select columns
            weekly_country_clean = weekly_country.reset_index()[[
                'Country', 'Code', 'year_and_week',
                'case_incidence_per_100k', 'death_incidence_per_100k',
                'PopulationVaccinated'
            ]].rename(columns={'PopulationVaccinated': 'percent_vaccinated'})

            # Ensure non-negative values
            weekly_country_clean['case_incidence_per_100k'] =␣
↪weekly_country_clean['case_incidence_per_100k'].clip(lower=0)
            weekly_country_clean['death_incidence_per_100k'] =␣
↪weekly_country_clean['death_incidence_per_100k'].clip(lower=0)

            weekly_data_list.append(weekly_country_clean)

    return pd.concat(weekly_data_list, ignore_index=True)

covid19_final = create_weekly_covid_simple(covid_stats, population)
covid19_final
```

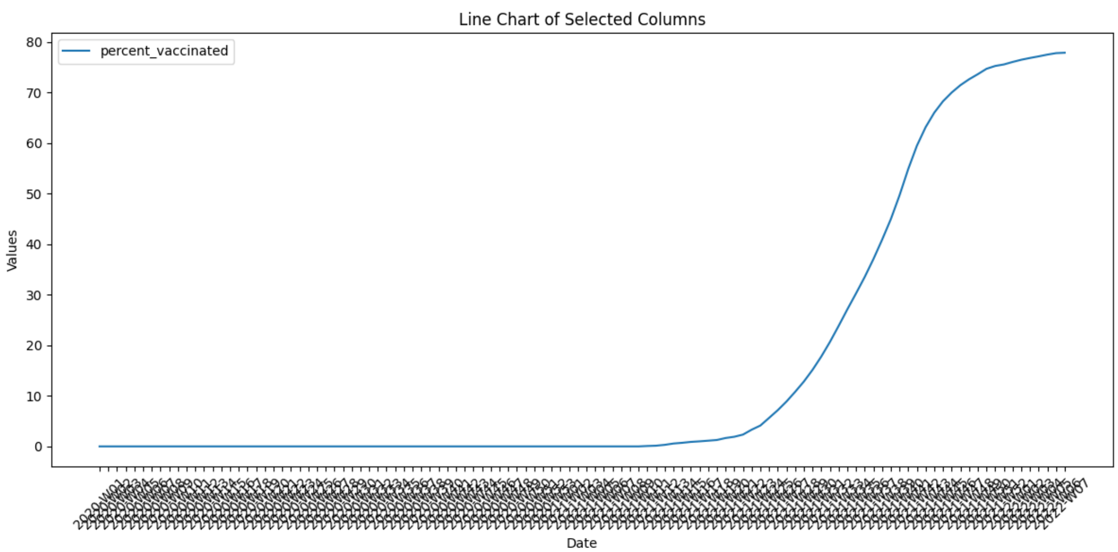
[209]:

```
        Country Code year_and_week  case_incidence_per_100k  \
0        Angola  AGO      2020-W01                 0.000000
1        Angola  AGO      2020-W02                 0.000000
2        Angola  AGO      2020-W03                 0.000000
3        Angola  AGO      2020-W04                 0.000000
4        Angola  AGO      2020-W05                 0.000000
…           …   …           …                        …
12427  Zimbabwe  ZWE      2022-W03                14.248887
12428  Zimbabwe  ZWE      2022-W04                 7.897131
12429  Zimbabwe  ZWE      2022-W05                 6.168406
12430  Zimbabwe  ZWE      2022-W06                 6.410690
12431  Zimbabwe  ZWE      2022-W07                 1.453701

       death_incidence_per_100k  percent_vaccinated
0                      0.000000                0.00
1                      0.000000                0.00
2                      0.000000                0.00
3                      0.000000                0.00
4                      0.000000                0.00
…                          …                   …
```

```
12427                     0.307765              20.04
12428                     0.281573              20.20
12429                     0.163705              20.37
12430                     0.078578              20.52
12431                     0.000000              20.54

[12432 rows x 6 columns]
```

[218]:

```
# Check Botswana
country_to_check = covid19_final[covid19_final['Code'] == 'AUS']
graph_columns(country_to_check, 'year_and_week', [
    #'case_incidence_per_100k',
    #'death_incidence_per_100k',
    'percent_vaccinated'
    ], "line")
```



[219]:

```
# Save Covid19_final
covid19_final.to_csv('data/cleaned/Covid_data.csv')
```

4.2 Country Stat

[223]:

```
country_stat = country_stat[country_stat['Code'].isin(study_sample['Code'])]
country_stat
```

[223]:

```
    Code                 Country  urban_population  \
2    AGO                  Angola            66.177
6    ARE    United Arab Emirates            86.789
7    ARG               Argentina            91.991
```

```
13   AUS                  Australia           86.124
14   AUT                    Austria           58.515
..   …                            …                …
231  USA  United States of America            82.459
243  YEM                      Yemen           37.273
244  ZAF               South Africa           66.856
245  ZMB                     Zambia           44.072
246  ZWE                   Zimbabwe           32.210

     corruption_perception_index  gdp_per_capita  land_boundaries  coastline  \
2                           26.0       8274.5430          5369.00     1600.0
6                           71.0      68887.8400          1066.00     1318.0
7                           45.0      26629.5530         11968.00     4989.0
13                          77.0      56981.3950             0.00    25760.0
14                          77.0      65312.0230          2524.00        0.0
..                             …               …                …        …
231                         69.0      69511.7660         12002.00    19924.0
243                         15.0        623.4000          1601.00     1906.0
244                         44.0      14370.2380          5244.00     2798.0
245                         34.0       3591.5642          6043.15        0.0
246                         24.0       3294.8062          3229.00        0.0

     num_border_countries                                  border_countries  \
2                     4.0  Democratic Republic of the Congo 2,646 km (of …
6                     2.0                 Oman 609 km; Saudi Arabia 457 km
7                     5.0  Bolivia 942 km; Brazil 1,263 km; Chile 6,691 k…
13                    0.0                                               NaN
14                    8.0  Czech Republic 402 km; Germany 801 km; Hungary…
..                      …                                                 …
231                   2.0  Canada 8,891 km (including 2,475 km with Alask…
243                   2.0               Oman 294 km; Saudi Arabia 1,307 km
244                   6.0  Botswana 1,969 km; Lesotho 1,106 km; Mozambiqu…
245                   8.0  Angola 1,065 km; Botswana 0.15 km; Democratic …
246                   4.0  Botswana 834 km; Mozambique 1,402 km; South Af…

     hospital_beds_per_1000  unemployment       political_regime  gini_index  \
2                      0.75        16.497  electoral_autocracies    0.512640
6                      1.87         2.331     closed_autocracies    0.263990
7                      3.71         9.843  electoral_democracies    0.433141
13                     3.84         5.159    liberal_democracies    0.343326
14                     7.19         4.560    liberal_democracies    0.302104
..                        …             …                      …           …
231                    2.75         3.669    liberal_democracies    0.415335
243                    0.71        17.202     closed_autocracies    0.367071
244                    2.30        28.468  electoral_democracies    0.630258
245                    2.00         5.542  electoral_autocracies    0.514831
246                    2.00         7.373  electoral_autocracies    0.502564
```

```
     population_density    poverty  median_age  land_area_sqkm
2             25.969065  31.122005      16.302       1246700.0
6            132.045270   0.000000      30.834         71020.0
7             16.433529   1.684649      30.763       2736690.0
13             3.312877   0.497094      36.543       7692020.0
14           107.620880   0.640639      42.433         82520.0
..                  …          …         …              …
231           36.927360   0.999171      37.002       9147420.0
243           66.502680  19.802757      18.017        527970.0
244           49.120743  20.492558      26.873       1213090.0
245           24.904613  64.349754      16.763        743390.0
246           39.476223  39.754530      17.187        386850.0

[111 rows x 17 columns]
```

4.2.1 Check for missing

[224]:
```python
# Check missing data patterns
missing_counts = country_stat.isnull().sum()
missing_percent = (missing_counts / len(country_stat)) * 100

print("Missing data percentage:")
for col, pct in missing_percent.items():
    print(f"{col}: {pct:.2f}%")
```

```
Missing data percentage:
Code: 0.00%
Country: 0.00%
urban_population: 0.00%
corruption_perception_index: 0.00%
gdp_per_capita: 0.00%
land_boundaries: 0.00%
coastline: 0.00%
num_border_countries: 0.00%
border_countries: 8.11%
hospital_beds_per_1000: 0.00%
unemployment: 0.00%
political_regime: 0.00%
gini_index: 0.00%
population_density: 0.00%
poverty: 0.00%
median_age: 0.00%
land_area_sqkm: 0.00%
```

border countries have more than 8% missing so I will drop it

```
[225]: country_stat = country_stat.drop(columns = 'border_countries')
```

4.2.2 Check for outliers

```
[226]: # Check for extreme values in numerical variables
num_cols = [
    'urban_population',
    'corruption_perception_index',
    'gdp_per_capita',
    'land_boundaries',
    'coastline',
    'num_border_countries',
    'hospital_beds_per_1000',
    'unemployment',
    'gini_index',
    'population_density',
    'poverty',
    'median_age',
    'land_area_sqkm'
]

# Compute quantiles for outlier detection
for col in num_cols:
    quantiles = country_stat[col].quantile([0.01, 0.25, 0.50, 0.75, 0.99])
    print(f"{col} - 1st percentile: {quantiles[0.01]:.2f}, 1st quartile:
 ↪{quantiles[0.25]:0.2f}, median:{quantiles[0.50]:0.2f}, 3rd quartile:
 ↪{quantiles[0.75]:0.2f}, 99th percentile: {quantiles[0.99]:.2f}")
```

```
urban_population - 1st percentile: 16.60, 1st quartile:47.76, median:65.76, 3rd
quartile:80.50, 99th percentile: 97.78
corruption_perception_index - 1st percentile: 18.20, 1st quartile:31.00,
median:41.00, 3rd quartile:60.00, 99th percentile: 85.90
gdp_per_capita - 1st percentile: 1537.26, 1st quartile:7384.21, median:18106.02,
3rd quartile:41918.00, 99th percentile: 111515.05
land_boundaries - 1st percentile: 0.00, 1st quartile:1160.50, median:2420.00,
3rd quartile:4401.00, 99th percentile: 15919.30
coastline - 1st percentile: 0.00, 1st quartile:51.30, median:823.00, 3rd
quartile:2790.00, 99th percentile: 53009.70
num_border_countries - 1st percentile: 0.00, 1st quartile:2.00, median:4.00, 3rd
quartile:5.00, 99th percentile: 9.90
hospital_beds_per_1000 - 1st percentile: 0.21, 1st quartile:1.08, median:2.30,
3rd quartile:4.37, 99th percentile: 12.15
unemployment - 1st percentile: 0.42, 1st quartile:3.60, median:5.01, 3rd
quartile:8.63, 99th percentile: 20.33
gini_index - 1st percentile: 0.24, 1st quartile:0.32, median:0.35, 3rd
quartile:0.42, 99th percentile: 0.60
population_density - 1st percentile: 3.23, 1st quartile:37.46, median:88.66, 3rd
```

```
quartile:178.91, 99th percentile: 1205.61
poverty - 1st percentile: 0.00, 1st quartile:0.24, median:1.01, 3rd
quartile:6.63, 99th percentile: 65.02
median_age - 1st percentile: 15.09, 1st quartile:22.38, median:29.06, 3rd
quartile:39.77, 99th percentile: 45.89
land_area_sqkm - 1st percentile: 586.70, 1st quartile:65747.00,
median:230800.00, 3rd quartile:616035.00, 99th percentile: 9111548.00
```

[227]:
```
# Save the file
country_stat.to_csv('data/cleaned/Country_stat.csv')
```