

TRƯỜNG ĐẠI HỌC FPT - CƠ SỞ TP.HCM -



FPT UNIVERSITY

TIỂU LUẬN BÁO CÁO DỰ ÁN

**ĐỀ TÀI: PHÂN TÍCH DỮ LIỆU DÂN CƯ QUỐC GIA VÀ
XÂY DỰNG MÔ HÌNH DỰ ĐOÁN**

**MÔN: KHOA HỌC DỮ LIỆU VỚI PYTHON VÀ SQL
(ADY201m)**

LỚP: AI1903

GIẢNG VIÊN BỘ MÔN: ĐOÀN NGUYỄN THÀNH HÒA

NHÓM SINH VIÊN THỰC HIỆN:

- | | |
|---------------------------|----------------|
| 1. Nguyễn Tân Khôi Nguyên | MSSV: SE200526 |
| 2. Trần Quang Thái | MSSV: SE200901 |
| 3. Phan Tân Phước | MSSV: SE201023 |
| 4. Trịnh Tiến Khải | MSSV: SE196461 |

TP.Hồ Chí Minh, ngày 14 tháng 10 năm 2025

Mục lục báo cáo

A. Giới thiệu về đề tài:	4
1. Khái quát về đề tài:	4
2. Các vấn đề gặp phải khi phân tích dữ liệu và xu hướng dân cư:	4
3. Hướng giải quyết các vấn đề:	4
4. Các vấn đề cần phân tích:	4
B. Các bước tiền xử lý dữ liệu:	5
1. Thu thập dữ liệu:	5
2. Mô tả dữ liệu:	5
3. Tiền xử lý dữ liệu:	5
C. Truy vấn dữ liệu trên hệ quản trị cơ sở dữ liệu SQL server:	12
1. Khái quát về SQL server:	12
2. Thao tác đẩy dữ liệu lên SQL server.	12
3. Truy vấn các vấn đề trên SQL server.	15
3.1 Quan sát xu hướng tăng trưởng dân số theo thời gian:	15
3.2 Tìm vùng có mật độ dân số lớn nhất mỗi năm:	15
3.3 Xem vùng nào có tốc độ tăng dân số tự nhiên cao/thấp nhất:	16
3.4 Kiểm tra sự cân bằng nhân khẩu học dựa vào tỷ suất sinh thô và chết thô:	17
3.5 Đánh giá quá trình đô thị hóa của các vùng:	17
3.6 So sánh thất nghiệp giữa thành thị và nông thôn:	17
3.7 Xem vùng có tăng dân số mạnh có tỷ lệ thất nghiệp cao không:.....	18
3.8 Xem vùng nào thu hút hay mất dân cư:.....	19
D. Phân tích dữ liệu với Pandas (Python):	20
1. Truy vấn và phân tích các vấn đề trên thư viện Pandas:	20
1.1 Quan sát xu hướng tăng trưởng dân số theo thời gian:.....	20
1.2 Tìm vùng có mật độ dân số lớn nhất mỗi năm:	21
1.3 Xem vùng nào có tốc độ tăng dân số tự nhiên cao/thấp nhất:	22
1.4 Kiểm tra sự cân bằng nhân khẩu học dựa vào tỷ suất sinh thô và chết thô:	23
1.5 Đánh giá quá trình đô thị hóa của các vùng:	23
1.6 So sánh thất nghiệp giữa thành thị và nông thôn:	24
1.7 Xem vùng có tăng dân số mạnh có tỷ lệ thất nghiệp cao không:.....	25
1.8 Xem vùng nào thu hút hay mất dân cư:.....	26

2. So sánh với SQL server.	27
E. Trực quang hóa dữ liệu với Python:	28
1. Import thư viện và đọc dữ liệu:	28
2. Chuẩn bị dữ liệu:	28
3. Phân tích hàm thông kê mô tả:	29
4. Viết các hàm vẽ biểu đồ:.....	30
5. Trực quang hóa dữ liệu:	33
F. Mô hình hồi quy và dự đoán trong dữ liệu:	46
1. Phân tích hồi quy và mô hình XGBoost Regressor:	46
1.1 Tổng quan:	46
1.2 Thư viện và vai trò:	46
1.3 Các lớp/hàm chính:	46
1.4 Mã nguồn hỗ trợ:.....	49
1.5 Phân tích kết quả dự đoán (R^2):	49
1.6 Đánh giá mô hình:	50
2. Phân tích kết quả dự đoán bằng ARIMA.....	50
2.1 Tổng quan:	50
2.2 Thư viện và vai trò:	50
2.3 Các hàm và class trong file:	50
2.4 Các cell mã nguồn khác:.....	52
2.5 Kết quả dự đoán:	52

A. Giới thiệu đề tài:

1. Khái quát về đề tài:

- Đề tài tập trung khai thác và phân tích sâu dữ liệu dân cư quốc gia trong giai đoạn 2011 - 2024 để nhận diện các xu hướng biến động chính, bao gồm: sự thay đổi về quy mô và mật độ dân số, tốc độ đô thị hóa (qua sự dịch chuyển dân cư), và diễn biến tỷ lệ thất nghiệp qua các năm. Bằng cách kết nối các yếu tố này, đề tài sẽ làm rõ tác động qua lại giữa tăng trưởng dân số, đô thị hóa và vấn đề việc làm, từ đó đánh giá ảnh hưởng của chúng lên quy hoạch đô thị và phân bổ nguồn lực.
- Dữ liệu sẽ được xử lý và phân tích bằng các công cụ mạnh như SQL và Python. Dựa trên các xu hướng đã phân tích, đề tài sẽ áp dụng các mô hình học máy và dự đoán để đưa ra các kịch bản dự báo các chỉ số về dân cư, mức độ đô thị hóa và tỉ lệ thất nghiệp cho giai đoạn 2025 – 2030. Kết quả phân tích không chỉ cung cấp một bức tranh toàn cảnh về dân cư Việt Nam mà còn là cơ sở khoa học hỗ trợ việc hoạch định chính sách, phát triển đô thị và phân bổ nguồn lực quốc gia một cách hiệu quả hơn.

2. Các vấn đề gặp phải khi phân tích dữ liệu và xu hướng dân cư:

Trong quá trình phân tích dữ liệu dân cư quốc gia, nhóm gặp một số khó khăn chính:

- Kiểu dữ liệu không đồng nhất: Các bảng thống kê có những giá trị không đồng nhất dữ liệu với cột mà giá trị áy đang thể hiện.
- Thiếu dữ liệu ở một số năm hoặc khu vực: Một số giai đoạn không có số liệu cụ thể cho từng vùng do bị phân tách thành thị - nông thôn hoặc lỗi dữ liệu làm mất thời gian xử lý tìm kiếm dữ liệu và ảnh hưởng đến độ chính xác của mô hình dự đoán.

3. Hướng giải quyết các vấn đề:

Kiểu dữ liệu không đồng nhất:

- Làm sạch dữ liệu với thư viện Pandas: Dùng thư viện Pandas(Python) để kiểm tra tuần tự các dòng giá trị không đồng nhất kiểu dữ liệu hoặc các giá trị Null.
- Xây dựng nền tảng cơ sở dữ liệu(các bảng) sau khi đã được kết hợp và chuẩn hóa để phục vụ cho việc nghiên cứu dữ liệu .

Thiếu dữ liệu ở một số năm hoặc khu vực:

- Giới hạn phạm vi phân tích: Bởi vì những dữ liệu trước năm 2011 thiếu nhiều, nên nhóm tiến hành phân tích ở giai đoạn có đủ thông tin (giai đoạn cụ thể: 2011-2024), tránh sai lệch mô hình.

4. Các vấn đề cần phân tích:

1. Quan sát xu hướng tăng trưởng dân số theo thời gian.
2. Tìm vùng có mật độ dân số lớn nhất mỗi năm.
3. Xem vùng nào có tốc độ tăng dân số tự nhiên cao/thấp nhất.
4. Kiểm tra sự cân bằng nhân khẩu học dựa vào tỷ suất sinh thô và chết thô.
5. Đánh giá quá trình đô thị hóa của các vùng.
6. So sánh thất nghiệp giữa thành thị và nông thôn.
7. Xem vùng có tăng dân số mạnh có tỷ lệ thất nghiệp cao không.
8. Xem vùng nào thu hút hay mất dân cư.

B.Các bước tiền xử lý dữ liệu:

1. Thu thập dữ liệu:

Dữ liệu được thu thập từ các bảng thống kê được công bố trực tuyến tại trang web chính thức của Cơ quan thống kê quốc gia – thuộc Cục thống kê của Bộ Tài Chính <https://www.nso.gov.vn/>.

Quá trình thu thập được thực hiện thủ công:

- Tải các file Excel được công bố chính thức.
- Lưu trữ dữ liệu dưới dạng .xlsx thống nhất để dễ xử lý.

2. Mô tả dữ liệu:

Tên cột	Ý nghĩa	Đơn vị
Vùng	Tên tỉnh và thành phố trực thuộc trung ương	
Khu vực	Khu vực địa lý của từng tỉnh	
Năm	Thời gian	Năm
Diện tích	Diện tích của một tỉnh hoặc thành phố	km ²
Tổng dân số tỉnh	Thể hiện quy mô dân số thực tế của một năm	Nghìn người
Dân số trung bình	Phản ánh quy mô dân số trung bình trong cả năm	Nghìn người
Mật độ dân số	Tổng số người trong phạm vi 1km ²	Người/ km ²
Tổng dân số thành thị	Tổng số dân ở thành thị	Nghìn người
Tổng dân số nông thôn	Tổng số dân ở nông thôn	Nghìn người
Nam	Tổng dân số có giới tính nam	Nghìn người
Nữ	Tổng dân số có giới tính nam	Nghìn người
Tỷ số giới tính	Mức chênh lệch giới tính	Số nam/ 100 nữ
Tỷ suất sinh thô	Số trẻ em được sinh trên 1000 dân	%/00
Tỷ suất chết thô	Số người chết trên 1000 dân	%/00
Tỷ lệ tăng tự nhiên	Mức tăng dân số tự nhiên, không tính di cư	%/00
Tỷ suất nhập cư	Số người nhập cư trong 1000 dân	%/00
Tỷ suất xuất cư	Số người xuất cư trong 1000 dân	%/00
Tỷ suất di cư thuần	Thể hiện mức chênh lệch di cư. Nếu dương thì dân số tăng do nhập cư, và ngược lại do xuất cư	%/00
Tỉ lệ thất nghiệp	Tỷ lệ người đang không có việc làm trong độ tuổi lao động	/%
Tỉ lệ thất nghiệp ở thành thị	Tình trạng thất nghiệp ở thành thị	/%
Tỉ lệ thất nghiệp ở nông thôn	Tình trạng thất nghiệp ở nông thôn	/%
Tổng dân số cả nước	Tổng dân số cả nước trong 1 năm	Nghìn người
Phần trăm dân số	Tỷ lệ tổng số dân tại 1 tỉnh hoặc thành phố so với tổng dân số cả nước trong một năm	/%

Bảng 1: Ý nghĩa của từng cột

3. Tiền xử lí dữ liệu:

❖ Mục tiêu của tiền xử lí dữ liệu: Chuẩn hóa, làm sạch và định dạng lại dữ liệu từ Cơ quan thống kê quốc gia thành dạng chuẩn hỗ trợ cho việc phân tích.

❖ Các bước cụ thể:

1. Xác định cấu trúc đích.
2. Làm sạch dữ liệu (00_Data_Cleaning.ipynb):

- Mục tiêu: Làm sạch, chuẩn hóa cấu trúc và khắc phục các lỗi mã hóa hoặc cấu trúc định dạng không tương thích.
- Các bước thực hiện:
 - + Bước 1: Đọc dữ liệu thô được lưu từ website <https://www.nso.gov.vn/>.
 - + Bước 2: Trích xuất các dữ liệu theo các tiêu chí: Vùng, Năm, và các chỉ tiêu khác.
 - + Bước 3: Xử lý hoàn chỉnh dữ liệu.
 - + Bước 3: Ghép toàn bộ các dữ liệu được tách theo cột Vùng.
 - + Bước 4: Kiểm tra lại dữ liệu sau làm sạch.
 - + Bước 4: Chuẩn hóa tên cột và xuất dữ liệu sạch.
- Tổng quan file xử lý dữ liệu 00_Data_Cleaning.ipynb:

```

raw_data = pd.read_excel('Datasets/Crawl_data/data.xlsx')
def formatting(i): # dong muon format
    row = raw_data.iloc[i] # du lieu tai dong do 2011 -> 2024
    year_data = [] # co du lieu tai cac nam do
    cnt = 0
    for i in range(1,42,3): # 1 -> 42
        if 2011 + cnt < 2025:
            a = [row[f'{2011 + cnt}']] # a = [row['2011']]
        else: break
        cnt += 1
        for j in range(1,3):
            a.append(row[f'Unnamed: {i+j}']) # a = []
    year_data.append(a) # year = [data 2011] index = 0
    region = row['Unnamed: 0'] # string -> list
    years = range(2011,2011+len(year_data)) #
    index = pd.MultiIndex.from_product(
        [[region],years],names = ['Vùng','Năm']) # dat ten columns
)
return pd.DataFrame(year_data,index=index,columns = list('abc'))
Raw_data = formatting(1)
for i in range(2,71):
    Raw_data = pd.concat([Raw_data,formatting(i)])
Raw_data = Raw_data.rename(columns={'a': 'Diện tích(Km2)', 'b': 'Dân số trung bình (Nghìn người)', 'c': 'Mật độ dân số (Người/km2)'})
Raw_data.to_excel('Datasets/Raw_data/Area_population_density.xlsx')

```

Mã nguồn 1: Formatting data.xlsx

```

raw_data = pd.read_excel('Datasets/Crawl_data/Data_02.xlsx',engine='openpyxl')
def get_data(str1,str2,row):
    if str2 == 'end':
        return np.array([data for data in row[str1:]])
    cols = row.index
    start = cols.get_loc(str1)
    end = cols.get_loc(str2)
    return row.iloc[start:end].to_numpy()
def formatting_2(i):
    row = raw_data.iloc[i]

    sum_data = get_data('Tổng số', 'Nam',row)
    man_data = get_data('Nam', 'Nữ',row)
    woman_data = get_data('Nữ', 'Thành thị',row)
    city_data = get_data('Thành thị', 'Nông thôn',row)
    rural_data = get_data('Nông thôn', 'end',row)

```

```

year_data = np.column_stack((sum_data,man_data,woman_data,city_data,rural_data))

year = [i for i in range(2011,2025)]
region = row['Vùng']

index = pd.MultiIndex.from_product([[region],year],names=[ 'Vùng', 'Năm'])
return pd.DataFrame(year_data,index=index,columns=list('abcde'))

Raw_data_2 = formatting_2(1)
for i in range(2,71):
    Raw_data_2 = pd.concat([Raw_data_2,formatting_2(i)])
Raw_data_2 = Raw_data_2.fillna(method='ffill')
Raw_data_2 = Raw_data_2.rename(columns={'a': 'Tổng số', 'b': 'Nam', 'c': 'Nữ','d':'Tổng dân số thành thị','e':'Tổng dân số nông thôn'})
Raw_data_2.to_excel('Datasets/Raw_data/Average_population_by_sex_urban_rural_residence.xlsx')

```

Mã nguồn 2: Formatting Data_02.xlsx

```

raw_data_05 = pd.read_excel('Datasets/Crawl_data/Data_05.xlsx')

def formatting_data_05(i):
    """Format one region's data row into long format like 'Sex_ratio_locality_true'"""
    row = raw_data_05.loc[i]
    region = row['Unnamed: 0']

    # --- Collect all yearly data ---
    year_data = []
    for col in row.index[1:]:
        # Extract only valid numeric year columns
        year_str = str(col)
        year = ''.join(filter(str.isdigit, year_str))
        if year.isdigit():
            value = row[col]
            if pd.notna(value) and value != '..': # Skip missing or placeholder
                year_data.append([region, int(year), float(value)])

    # Convert to DataFrame for that region
    return pd.DataFrame(year_data, columns=['Vùng', 'Năm', 'Tỷ lệ nam nữ'])

# === COMBINE ALL REGIONS ===
formatted_data_05 = formatting_data_05(0)
for i in range(1, len(raw_data_05)):
    formatted_data_05 = pd.concat([formatted_data_05, formatting_data_05(i)], ignore_index=True)

# === OUTPUT ===
formatted_data_05.reset_index(drop=True, inplace=True)
formatted_data_05.to_excel('Datasets/Raw_data/Sex_ratio_locality.xlsx', index=False)

```

Mã nguồn 3: Formatting Data_05.xlsx

```

raw_data_08 = pd.read_excel('Datasets/Crawl_data/Data_08.xlsx')
def formatting_data_08(i):
    row = raw_data_08.loc[i]
    year_data = []
    Crude_birth_rate = get_data('Tỷ suất sinh thô','Tỷ suất chết thô',row)
    Crude_death_rate = get_data('Tỷ suất chết thô','Tỷ lệ tăng tự nhiên',row)
    Natural_birht_rate = get_data('Tỷ lệ tăng tự nhiên','end',row)

```

```

year_data = np.column_stack([Crude_birth_rate,Crude_death_rate,Natural_birht_rate])
year = [i for i in range(2011,2025)]
region = row['Unnamed: 0']

index = pd.MultiIndex.from_product([[region],year],names=[ 'Vùng','Năm'])
return pd.DataFrame(year_data,index=index,columns=list(['Tỷ suất sinh thô','Tỷ suất
chết thô','Tỷ lệ tăng tự nhiên']))
data_08 = formatting_data_08(1)
for i in range(2,71):
    data_08 = pd.concat([data_08,formatting_data_08(i)])
data_08.to_excel('Datasets/Raw_data/Crude_birth_death_rate_natural_growth_rate_locality.xls
x')

```

Mã nguồn 4: Formatting Data_08.xlsx

```

raw_data_15 = pd.read_excel('Datasets/Crawl_data/Data_15.xlsx')
def get_data(str1,str2,row):
    if str2 == 'end':
        return np.array([data for data in row[str1:]])
    cols = row.index
    start = cols.get_loc(str1)
    end = cols.get_loc(str2)
    return row.iloc[start:end].to_numpy()
def formatting_data_15(i):
    row = raw_data_15.loc[i]
    year_data = []
    a = get_data('Tỷ suất nhập cư','Tỷ suất xuất cư',row)
    b = get_data('Tỷ suất xuất cư','Tỷ suất di cư thuận',row)
    c = get_data('Tỷ suất di cư thuận','end',row)

    year_data = np.column_stack([a,b,c])
    year = [i for i in range(2011,2025)]
    region = row['Unnamed: 0']

    index = pd.MultiIndex.from_product([[region],year],names=[ 'Vùng','Năm'])
    return pd.DataFrame(year_data,index=index,columns=list(['Tỷ suất nhập cư','Tỷ suất nhập
cư','Tỷ suất di cư thuận']))
formatting_data_15(1)
data_15 = formatting_data_15(1)
for i in range(2,71):
    data_15 = pd.concat([data_15,formatting_data_15(i)])
data_15.to_excel('Datasets/Raw_data/Immigration_emigration_net_migration_rates_locality.xls
x')

```

Mã nguồn 5: Formatting Data_15.xlsx

```

raw_data_25 = pd.read_excel('Datasets/Crawl_data/Data_25.xlsx')
raw_data_25.to_excel('Datasets/Raw_data/Unemployed_Data.xlsx',index=False)

```

Mã nguồn 6: Formatting Data_25.xlsx

3. Ghép các bảng dữ liệu (*01_Data_concated.ipynb*):

- Mục tiêu: Đảm bảo dữ liệu không trùng, không lỗi, chuẩn hóa chữ cái tiếng việt, đổi tên cột sau khi ghép bảng và loại bỏ các dữ liệu Vùng có giá trị là Cả nước và các khu vực địa lý để về đủ 63 tỉnh thành Việt Nam.
- Các bước cụ thể:
 - + Bước 1: Đọc các dữ liệu đã được làm sạch.

```

file_path = [r'Datasets/Raw_data/Area_population_density.xlsx',
             r'Datasets/Raw_data/Average_population_by_sex_urban_rural_residence.xlsx',
             r'Datasets/Raw_data/Crude_birth_death_rate_natural_growth_rate_locality.xlsx',
             r'Datasets/Raw_data/Immigration_emigration_net_migration_rates_locality.xlsx',
             r'Datasets/Raw_data/Sex_ratio_locality.xlsx',
             r'Datasets/Raw_data/Unemployed_data.xlsx']

df1 = pd.read_excel(file_path[0], engine='openpyxl')
df2 = pd.read_excel(file_path[1], engine='openpyxl')
df3 = pd.read_excel(file_path[2], engine='openpyxl')
df4 = pd.read_excel(file_path[3], engine='openpyxl')
df5 = pd.read_excel(file_path[4], engine='openpyxl')
df6 = pd.read_excel(file_path[5], engine='openpyxl')

```

Mã nguồn 7: Đọc dữ liệu đã làm sạch

- + Bước 2: Xử lý các dữ liệu thiếu do merge cell trong excel tạo các khoảng trống sau khi làm sạch ở cột vùng nên sử dụng ffill để điền các giá trị còn thiếu.

```

df1 = df1.ffill()
df2 = df2.ffill()
df3 = df3.ffill()
df4 = df4.ffill()
df5 = df5.ffill()
df6 = df6.ffill()

```

Mã nguồn 8: Xử lý các dữ liệu thiếu

- + Bước 3: Chuẩn hóa kí tự tiếng Việt(Unicode NFC)
- Tạo file chuẩn hóa dữ liệu tiếng Việt, loại bỏ các ký tự vô hình và thống nhất các ký tự sai hệ chữ. Sử dụng các thư viện re(loại bỏ ký tự ẩn), unicodedata(chuẩn hóa dạng Unicode NFD/NFC), và pandas(xử lý dataframe).

```

import re
import unicodedata
import pandas as pd

```

```

def debug_unicode(a, b):
    for i, (ca, cb) in enumerate(zip(a, b)):
        if ca != cb:
            print(f"Khác tại vị trí {i}: '{ca}' ({ord(ca)}) vs '{cb}' ({ord(cb)})")

# Các ký tự/vết vô hình hay gấp
INVISIBLES = [
    "\ufe0f",      # BOM
    "\u200b",      # zero-width space
    "\u200c",      # ZWNJ
    "\u200d",      # ZWJ
    "\u2060",      # word joiner
]
# thay thế chúng bằng rỗng
INV_RE = re.compile("|".join(map(re.escape, INVISIBLES)))

# Homoglyph & ký tự "sai hệ chữ" thường gấp
HOMOGLYPHS = {
    "Đ": "Đ", "đ": "đ",           # ETH (Icelandic) do đó D gạch (VN)
    "İ": "İ", "ı": "ı",           # Turkish dotted/ dotless I do đó Latin
}

```

```

": "", ", ":" , ", "€": "", "€€": "", "€€€": "", "„": "", "„„": "", "„„„": "", # nhiều Loại gạch nối
"-": "-", "-": "-", "-": "-", "-": "-", "-": "-", "-": "-", "# NBSP do đó space thường
}
PAIR = {
    "oá": "óa",
    "oà": "òa",
    "oả": "òa",
    "oã": "õa",
    "oạ": "ọa",
}
# Dấu tiếng Việt dưới dạng “tổ hợp” (nếu có)
COMBINING_TONE = {"\u0300", "\u0301", "\u0303", "\u0309", "\u0323"} # huyền, sắc, ngã,
hỏi, nặng
COMBINING_SHAPES = {"\u0302", "\u0306", "\u031B"} # ^, ˇ, ˘ (ô/â/ă/ơ/u)

def normalize_vi_text(s: str) -> str:
    if s == 'Huế': s = 'Thừa Thiên Huế'
    if s == 'Bắc Trung Bộ và Duyên hải miền Trung': s = 'Bắc Trung Bộ và duyên hải miền
Trung'
    if s == 'TP.Hồ Chí Minh': s = 'TP. Hồ Chí Minh'
    if not isinstance(s, str):
        return s

    # 1) Bỏ ký tự ẩn + chuẩn hoá khoảng trắng
    s = INV_RE.sub("", s)
    s = s.replace("\xa0", " ") # NBSP
    s = s.strip()
    # gom nhiều khoảng trắng về 1
    s = re.sub(r"\s+", " ", s)

    # 2) Thay homoglyphs sang ký tự Latin/Việt chuẩn và chuẩn hóa vị trí của dấu
    if HOMOGLYPHS:
        s = "".join(HOMOGLYPHS.get(ch, ch) for ch in s)
    for k,v in PAIR.items():
        s = re.sub(rf'\b{k}\b', v, s)
        s = re.sub(rf'{k}\b', v, s)
    # 3) “NFD do đó xử lý do đó NFC” để gộp dấu đúng chuẩn
    #     - NFD tách dấu ra; NFC gộp lại theo thứ tự chuẩn Unicode
    s = unicodedata.normalize("NFD", s)

    # Đề phòng dữ liệu có dấu/shape dính nhầm vào ký tự khác,
    # ta chỉ giữ các dấu kết hợp hợp lệ, bỏ dấu rác hiêm gặt.
    cleaned = []
    for ch in s:
        # Bỏ các combining không phải dấu/shape tiếng Việt (rất hiêm)
        if unicodedata.combining(ch) and (ch not in COMBINING_TONE | COMBINING_SHAPES):
            continue
        cleaned.append(ch)
    s = "".join(cleaned)

    # 4) Gộp Lại thành dạng chuẩn (precomposed) - cực quan trọng khi merge
    s = unicodedata.normalize("NFC", s)
    return s

def normalize_vi_df(df: pd.DataFrame) -> pd.DataFrame:
    # Chuẩn hoá tiêu đề cột

```

```

df.columns = [normalize_vi_text(c) if isinstance(c, str) else c for c in df.columns]
# Chuẩn hóa toàn bộ cột kiểu object (string)
for col in df.select_dtypes(include="object").columns:
    df[col] = df[col].map(normalize_vi_text)
return df

```

Mã nguồn 9: File chuẩn hóa ký tự tiếng Việt(Vietnamese_text_normalize.py)

- Gọi hàm chuẩn hóa ký tự tiếng Việt của từng Dataframe.

```

df1 = normalize_vi_df(df1)
df2 = normalize_vi_df(df2)
df3 = normalize_vi_df(df3)
df4 = normalize_vi_df(df4)
df5 = normalize_vi_df(df5)
df6 = normalize_vi_df(df6)

```

Mã nguồn 10: Chuẩn hóa ký tự tiếng Việt cho từng Dataframe

- + Bước 4: Kết hợp các Dataframe theo cột Vùng và Năm

```

data12 = normalize_vi_df(pd.merge(df1,df2, on=['Vùng','Năm']))
data34 = normalize_vi_df(pd.merge(df3,df4, on=['Vùng','Năm']))
data56 = normalize_vi_df(pd.merge(df5,df6, on=['Vùng','Năm']))
semi_data = pd.merge(data12,data34, on=['Vùng','Năm'])
full_data = pd.merge(semi_data,data56, on=['Vùng','Năm'])

```

Mã nguồn 11: Kết hợp các Dataframe

- + Bước 5: Đổi tên các cột bị trùng:

```

full_data.columns
full_data = full_data.rename(columns={'Tổng số_x':'Tổng dân số',
                                      'Thành thị_x':'Tổng dân số thành thị',
                                      'Nông thôn_x':'Tổng dân số nông thôn',
                                      'Tổng số_y': 'Tỉ lệ thất nghiệp',
                                      'Thành thị_y':'Tỉ lệ thất nghiệp ở thành thị',
                                      'Nông thôn_y':'Tỉ lệ thất nghiệp ở nông thôn'
                                     })

```

Mã nguồn 12: Đổi tên các cột bị trùng

- + Bước 6: Loại bỏ các dữ liệu ở cột Vùng có giá trị là Cả nước và các khu vực địa lý để còn lại 63 tỉnh thành cho việc phân tích và tránh sai dữ liệu sử dụng các hàm thống kê.

```

eliminate = ['CẢ NƯỚC',
             'Đồng bằng sông Hồng',
             'Trung du và miền núi phía Bắc',
             'Bắc Trung Bộ và duyên hải miền Trung',
             'Tây Nguyên',
             'Đông Nam Bộ',
             'Đồng bằng sông Cửu Long']

for region in eliminate:
    full_data = full_data[full_data['Vùng'] != region ]

```

Mã nguồn 13: Loại bỏ một số dữ liệu ở cột Vùng

- + Bước 7:Xuất dữ liệu sang CSV với encoding UTF-8 để tích hợp hầu hết với các tool phân tích dữ liệu.

```
full_data.to_csv('Datasets/Clean_data/Locality_Analyst.csv', encoding='utf-8-sig')
```

Mã nguồn 14: Xuất file phân tích cuối cùng

C. Truy vấn dữ liệu trên hệ quản trị cơ sở dữ liệu SQL server:

1. Khái quát về SQL server:

- SQL server (hay Microsoft SQL server) là một hệ quản trị cơ sở dữ liệu do Microsoft phát triển, sử dụng ngôn ngữ truy vấn SQL. SQL (Structured Query Language) là ngôn ngữ truy vấn có cấu trúc dùng để tương tác lên hệ quản trị cơ sở dữ liệu như SQL server, MySQL, PostgreSQL hay Oracle.
- Với SQL server, người dùng có thể sử dụng ngôn ngữ SQL để thực hiện các thao tác như lưu trữ, truy xuất, cập nhật, thêm, xóa hay tìm kiếm dữ liệu trong cơ sở dữ liệu một cách nhanh chóng và dễ dàng.
- Phần mềm này đóng vai trò trọng tâm trong việc lưu trữ dữ liệu tập trung và hỗ trợ truy vấn dữ liệu, thông tin nhanh chóng. Ngoài ra, còn giúp các ứng dụng, websites hoặc các tổ chức dễ dàng trong việc quản lý và khai thác dữ liệu của mình.

2. Thao tác đẩy dữ liệu lên SQL server.

- Tổng quan thao tác đẩy dữ liệu lên SQL server từ Python có 5 bước:
 1. Kết nối đến SQL server.
 2. Đọc dữ liệu từ file CSV.
 3. Tạo bảng (table) trên SQL server.
 4. Chuyển đổi dữ liệu.
 5. Đưa dữ liệu lên SQL server.
- Chi tiết quy trình:
 - + Bước 1: Kết nối đến SQL server.
 - Đoạn mã kết nối đến SQL server và thực hiện Import các thư viện cần thiết:

```
import pyodbc as odbc
import pandas as pd
from USER import DRIVERS, SERVER, DATABASE
import numpy as np

conn_str = f"""
    DRIVER={DRIVERS};
    SERVER={SERVER};
    DATABASE={DATABASE};
    Trusted_Connection=yes;
"""

conn = odbc.connect(conn_str)
cursor = conn.cursor()
```

Đoạn mã 13: Kết nối đến SQL server.

- Giải thích đoạn mã:
 - Sử dụng thư viện **pyodbc** để dễ dàng kết nối với SQL server thông qua giao thức ODBC(Open Database Connectivity). Giao thức ODBC là một chuẩn quốc tế cho phép các ứng dụng kết nối và tương tác với nhiều hệ quản trị cơ sở dữ liệu khác nhau như: SQL server, MySQL, PostgreSQL,...
 - Các giá trị DRIVERS, SERVER, DATABASE được lưu trong file **USER.py** để tránh lộ thông tin cấu hình.
 - Sau khi mở kênh kết nối đến SQL server thông qua hàm **connect()**, tạo một con trỏ (**cursor**) làm trung gian giữa SQL server và Python thông qua hàm **cursor()**. Cursor đảm nhận việc

gửi các câu lệnh SQL sang cho SQL server xử lí và tiếp nhận kết quả được trả về bao gồm: dữ liệu, trạng thái, lỗi,... . Có thể nói ngắn gọn, *cursor()* là công cụ thực thi lệnh SQL trong Python, đóng vai trò như “người phiên dịch” giữa mã Python và máy chủ SQL server.

+ Bước 2: Đọc dữ liệu từ file CSV:

```
data = pd.read_csv('Datasets/Clean_data/Locality_Analyst.csv')
```

Đoạn mã 14: Đọc dữ liệu file CSV

- Giải thích đoạn mã: Sử dụng Pandas để đọc file csv thông qua hàm *read_csv()*.

+ Bước 3: Tạo bảng (table) trên SQL server:

- Đoạn mã SQL được viết trên Python thông qua con trỏ Cursor:

```
cursor.execute("""
    IF OBJECT_ID('dbo.population_data', 'U') IS NOT NULL
        DROP TABLE dbo.population_data;

    CREATE TABLE dbo.population_data (
        region NVARCHAR(100),
        year INT,
        area_km2 FLOAT,
        avg_population_thousand FLOAT,
        population_density FLOAT,
        total_population FLOAT,
        male FLOAT,
        female FLOAT,
        urban_population FLOAT,
        rural_population FLOAT,
        crude_birth_rate FLOAT,
        crude_death_rate FLOAT,
        natural_growth_rate FLOAT,
        immigration_rate FLOAT,
        emigration_rate FLOAT,
        net_migration_rate FLOAT,
        sex_ratio FLOAT,
        area_type NVARCHAR(100),
        unemployment_rate FLOAT,
        urban_unemployment_rate FLOAT,
        rural_unemployment_rate FLOAT,
        population_percentage FLOAT
    );
""")

conn.commit()
cursor.close()
```

Đoạn mã 15: Tạo bảng SQL

- Giải thích đoạn mã:
 - Các hàm thao tác dữ liệu: Hàm *execute()* dùng để gửi các câu lệnh SQL đến SQL server; Hàm *commit()* dùng để xác nhận và lưu vĩnh viễn các thay đổi trong SQL, Hàm *close()* dùng để đóng kết nối sau khi hoàn thành công việc.
 - Lệnh SQL: Nếu *dbo.population_data* đã có tồn tại thì thực hiện xóa bảng đó và bắt đầu thực thi câu lệnh tạo bảng bên dưới.

+ Bước 4: Chuyển đổi kiểu dữ liệu:

- Đoạn mã dùng để ép kiểu dữ liệu:

```
data = data.replace('...', np.nan)

col = ['Năm', 'Diện tích(Km2)', 'Dân số trung bình (Nghìn người)',
```

```

'Mật độ dân số (Người/km2)', 'Tổng dân số', 'Nam', 'Nữ',
'Tổng dân số thành thị', 'Tổng dân số nông thôn', 'Tỷ suất sinh thô',
'Tỷ suất chết thô', 'Tỷ lệ tăng tự nhiên', 'Tỷ suất nhập cư',
'Tỷ suất xuất cư', 'Tỷ suất di cư thuần', 'Tỷ số giới tính',
'Tỉ lệ thất nghiệp', 'Tỉ lệ thất nghiệp ở thành thị',
'Tỉ lệ thất nghiệp ở nông thôn', 'Phần trăm dân số']

```

```

data[col] = data[col].apply(pd.to_numeric, errors='coerce')

data = data.replace(np.nan, None)
data['Vùng'] = data['Vùng'].astype('string')

```

Đoạn mã 16: Ép kiểu dữ liệu

- Giải thích đoạn mã:
- Mục đích việc ép kiểu dữ liệu là chắc chắn tất cả các ô dữ liệu hoàn toàn khớp với kiểu dữ liệu đã khai báo để tạo bảng trong đoạn mã tạo bảng trước đó.
- Việc thay thế các dòng dữ liệu có giá trị '..' thành nan và chuyển tiếp thành None để tránh việc SQL server không đọc được các giá trị nan và có sự sai lệch kiểu dữ liệu.
- Ép kiểu [col] thành dạng số để đảm bảo không có sự sai lệch với kiểu dữ liệu trên SQL server. Và với phần tùy chọn tham số *errors='coerce'*, nó giúp việc xử lý các giá trị ngoài giá trị số thành nan để được chuyển thành giá trị None đầy vào SQL sẽ không lỗi.

+ Bước 5: Đưa dữ liệu lên SQL server:

- Đoạn mã dùng để đưa dữ liệu lên SQL server:

```

cursor = conn.cursor()
for index, row in data.iterrows():
    cursor.execute("""
        INSERT INTO Population_Data (
            region, year, area_km2, avg_population_thousand,
            population_density, total_population, male, female,
            urban_population, rural_population, crude_birth_rate,
            crude_death_rate, natural_growth_rate, immigration_rate,
            emigration_rate, net_migration_rate, sex_ratio, area_type,
            unemployment_rate, urban_unemployment_rate, rural_unemployment_rate,
            population_percentage
        )
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (
        row['Vùng'],
        row['Năm'],
        row['Diện tích(Km2)'],
        row['Dân số trung bình (Nghìn người)'],
        row['Mật độ dân số (Người/km2)'],
        row['Tổng dân số'],
        row['Nam'],
        row['Nữ'],
        row['Tổng dân số thành thị'],
        row['Tổng dân số nông thôn'],
        row['Tỷ suất sinh thô'],
        row['Tỷ suất chết thô'],
        row['Tỷ lệ tăng tự nhiên'],
        row['Tỷ suất nhập cư'],
        row['Tỷ suất xuất cư'],
        row['Tỷ suất di cư thuần'],
        row['Tỷ số giới tính'],
        row['Khu vực'],
        row['Tỉ lệ thất nghiệp'],
    ))

```

```

        row['Tỉ lệ thất nghiệp ở thành thị'],
        row['Tỉ lệ thất nghiệp ở nông thôn'],
        row['Phần trăm dân số']
    ))

```

```

conn.commit()
cursor.close()

```

Đoạn mã 17: Đẩy dữ liệu lên SQL

- Giải thích đoạn mã:
- Khởi tạo lại con trỏ sau khi đóng kết nối con trỏ ở đoạn mã trước.
- Câu lệnh `for index, row in data.iterrows()` dùng vòng lặp `for` để duyệt từng dòng (`row`) của một Dataframe. Hàm `iterrows()` dùng để lặp qua từng dòng của bảng và mỗi lần trả về `index` của dòng đó và một series các dữ liệu của dòng đó được gọi là `row`.
- Dấu “?” trong `VALUES()` là placeholder-là vị trí gắn các giá trị thật sau này. Mỗi lần duyệt các giá trị “?” sẽ được thay đổi theo giá trị tương ứng với các `row` bên dưới.

3. Truy vấn các vấn đề trên SQL server.

- Các vấn đề mà nhóm cần phân tích đã được nêu ra trước đó là:

3.1 Quan sát xu hướng tăng trưởng dân số theo thời gian:

Câu truy vấn:

```
select region, year, avg_population_thousand from population_data
```

Đoạn mã 18: Truy vấn vấn đề 1

Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

	region	year	avg_population_thousand
1	Hà Nội	2011	6825.8
2	Hà Nội	2012	6991.4
3	Hà Nội	2013	7128.4
4	Hà Nội	2014	7285.5
5	Hà Nội	2015	7433.6
6	Hà Nội	2016	7590.8
7	Hà Nội	2017	7742.2
8	Hà Nội	2018	7914.5
9	Hà Nội	2019	8093.9
10	Hà Nội	2020	8246.5
11	Hà Nội	2021	8330.8
12	Hà Nội	2022	8435.7
13	Hà Nội	2023	8587.1
14	Hà Nội	2024	8717.6

Hình ảnh 1: Kết quả vấn đề 1 SQL

3.2 Tìm vùng có mật độ dân số lớn nhất mỗi năm:

Câu truy vấn:

```

with max_population as(
    select year, max(population_density) max_total
    from population_data
)

```

```

        group by year)
select ld.region, ld.year, mp.max_total
from population_data ld, max_population mp
where ld.year = mp.year and ld.population_density = mp.max_total

```

Đoạn mã 19: Truy vấn vấn đề 2

Kết quả truy vấn (TP.HCM):

	region	year	max_total
1	TP. Hồ Chí Minh	2024	4554.6
2	TP. Hồ Chí Minh	2023	4513.1
3	TP. Hồ Chí Minh	2022	4481
4	TP. Hồ Chí Minh	2021	4375
5	TP. Hồ Chí Minh	2020	4404
6	TP. Hồ Chí Minh	2019	4385
7	TP. Hồ Chí Minh	2018	4289.9
8	TP. Hồ Chí Minh	2017	4196.4
9	TP. Hồ Chí Minh	2016	4113.3
10	TP. Hồ Chí Minh	2015	3964.6
11	TP. Hồ Chí Minh	2014	3882.6
12	TP. Hồ Chí Minh	2013	3805.1
13	TP. Hồ Chí Minh	2012	3717.2
14	TP. Hồ Chí Minh	2011	3633.1

Hình ảnh 2: Kết quả vấn đề 2 SQL

3.3 Xem vùng nào có tốc độ tăng dân số tự nhiên cao/thấp nhất:

Câu truy vấn:

```

with avg_natural_growth_rate as(
    select region, avg(natural_growth_rate) avg_value
    from population_data
    group by region
)
select *
from avg_natural_growth_rate
where avg_value = (select max(avg_value) from avg_natural_growth_rate)
or avg_value = (select min (avg_value) from avg_natural_growth_rate)

```

Đoạn mã 20: Truy vấn vấn đề 3

Kết quả truy vấn (Bến Tre & Điện Biên):

	region	avg_value
1	Bến Tre	3.33071428571429
2	Điện Biên	15.53

Hình ảnh 3: Kết quả vấn đề 3 SQL

3.4 Kiểm tra sự cân bằng nhân khẩu học dựa vào tỷ suất sinh thô và chết thô:

- ❖ Vì SQL server không hỗ trợ hàm thống kê tương quan (Correlation) và thực hiện trên SQL server quá phức tạp nên nhóm xin được phân tích bằng Pandas ở mục D.1

3.5 Đánh giá quá trình đô thị hóa của các vùng:

Câu truy vấn:

```
select region, year, round((urban_population / total_population *100), 2) urbanization  
from population_data
```

Đoạn mã 21: Truy vấn vấn đề 5

Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

	region	year	urbanization
1	Hà Nội	2011	42.49
2	Hà Nội	2012	42.52
3	Hà Nội	2013	42.43
4	Hà Nội	2014	49.19
5	Hà Nội	2015	49.11
6	Hà Nội	2016	49.18
7	Hà Nội	2017	49.21
8	Hà Nội	2018	49.34
9	Hà Nội	2019	49.42
10	Hà Nội	2020	49.25
11	Hà Nội	2021	49.16
12	Hà Nội	2022	49.06
13	Hà Nội	2023	49.06
14	Hà Nội	2024	49.11

Hình ảnh 4: Kết quả vấn đề 5 SQL

3.6 So sánh thất nghiệp giữa thành thị và nông thôn:

Câu truy vấn:

```
select REGION, area_type, Round(AVG(unemployment_rate),2) avg_total,  
      round(AVG(urban_unemployment_rate),2) avg_urban,  
      round(AVG(rural_unemployment_rate),2) avg_rural  
from population_data  
group by REGION, area_type  
order by region
```

Đoạn mã 22: Truy vấn vấn đề 6

Kết quả truy vấn (Với dữ liệu có 63 tỉnh/thành, nhóm xin phép lấy 12 tỉnh/thành đầu làm kết quả mẫu):

	REGION	area_type	avg_total	avg_urban	avg_rural
1	An Giang	Đồng bằng sông Cửu Long	2.82	2.16	3.58
2	Bà Rịa - Vũng Tàu	Đông Nam Bộ	3.15	1.8	3.19
3	Bắc Giang	Trung du và miền núi phía Bắc	2.83	2.23	2.74
4	Bắc Kạn	Trung du và miền núi phía Bắc	3	2.05	3.21
5	Bạc Liêu	Đồng bằng sông Cửu Long	3.31	2.06	3.27
6	Bắc Ninh	Đồng bằng sông Hồng	2.88	2.46	2.2
7	Bến Tre	Đồng bằng sông Cửu Long	2.66	1.72	3.89
8	Bình Định	Bắc Trung Bộ và duyên hải miền Trung	2.58	2.12	2.7
9	Bình Dương	Đông Nam Bộ	2.63	2.11	3.01
10	Bình Phước	Đông Nam Bộ	2.59	1.7	3.72
11	Bình Thuận	Bắc Trung Bộ và duyên hải miền Trung	2.66	2.1	3.1
12	Cà Mau	Đồng bằng sông Cửu Long	2.8	2.11	3.63

Hình ảnh 5: Kết quả vấn đề 6 SQL

3.7 Xem vùng có tăng dân số mạnh có tỷ lệ thất nghiệp cao không:

Câu truy vấn:

```
with subtable as(
    select region, year, total_population, ROUND(
        (total_population - LAG(total_population) OVER (PARTITION BY region ORDER BY year))
        / NULLIF(LAG(total_population) OVER (PARTITION BY region ORDER BY year), 0))
    * 100, 2) population_growth_pct, unemployment_rate
    from population_data )
SELECT region, ROUND(AVG(population_growth_pct), 2) population_growth_pct,
ROUND(AVG(unemployment_rate), 2) unemployment_rate
FROM subtable
GROUP BY region
ORDER BY population_growth_pct DESC;
```

Đoạn mã 23: Truy vấn vấn đề 7

Kết quả truy vấn (Với dữ liệu có 63 tỉnh/thành, nhóm xin phép lấy 12 tỉnh/thành đầu làm kết quả mẫu):

	region	population_growth_pct	unemployment_rate
1	Bình Dương	4.13	2.63
2	Bắc Ninh	2.86	2.88
3	Đắk Nông	2.38	2.42
4	Kon Tum	2.23	3.37
5	Đà Nẵng	2.14	3.23
6	Điện Biên	1.92	3.06
7	Hà Nội	1.9	2.63
8	Đồng Nai	1.82	3.29
9	Lai Châu	1.8	3.19
10	TP. Hồ Chí Minh	1.76	2.78
11	Lào Cai	1.68	3.24
12	Bắc Giang	1.65	2.83

Hình ảnh 6: Kết quả vấn đề 7 SQL

3.8 Xem vùng nào thu hút hay mất dân cư:

Câu truy vấn:

```
select region , year, immigration_rate, emigration_rate, round((immigration_rate - emigration_rate),2) net_migration_rates
from population_data
```

Đoạn mã 24: Truy vấn vấn đề 8

Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

	region	year	immigration_rate	emigration_rate	net_migration_rates
1	Hà Nội	2011	11	6.4	4.6
2	Hà Nội	2012	6.1	3.3	2.8
3	Hà Nội	2013	7.7	7.4	0.3
4	Hà Nội	2014	7.5	7.8	-0.3
5	Hà Nội	2015	4.7	4.1	0.6
6	Hà Nội	2016	4.6	2.6	2
7	Hà Nội	2017	3	3.3	-0.3
8	Hà Nội	2018	4.7	2.6	2.1
9	Hà Nội	2019	8.8	2.5	6.3
10	Hà Nội	2020	6.26	2.53	3.73
11	Hà Nội	2021	7.68	1.75	5.93
12	Hà Nội	2022	4.79	2.84	1.95
13	Hà Nội	2023	5.85	1.96	3.89
14	Hà Nội	2024	5.91	4.47	1.44

Hình ảnh 7: Kết quả vấn đề 8 SQL

D.Phân tích dữ liệu với Pandas (Python):

1. Truy vấn và phân tích các vấn đề trên thư viện Pandas:

1.1 Quan sát xu hướng tăng trưởng dân số theo thời gian:

- Câu truy vấn:

```
Locality_data[['Vùng', 'Năm', 'Dân số trung bình (Nghìn người)']].copy()
```

Đoạn mã 25: Truy vấn vấn đề 1

- Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

	Vùng	# Năm	# Dân số trung bình (Nghìn người)
0	Hà Nội	2011	6825.8
1	Hà Nội	2012	6991.4
2	Hà Nội	2013	7128.4
3	Hà Nội	2014	7285.5
4	Hà Nội	2015	7433.6
5	Hà Nội	2016	7590.8
6	Hà Nội	2017	7742.2
7	Hà Nội	2018	7914.5
8	Hà Nội	2019	8093.9
9	Hà Nội	2020	8246.5
10	Hà Nội	2021	8330.8
11	Hà Nội	2022	8435.7
12	Hà Nội	2023	8587.1
13	Hà Nội	2024	8717.6

Hình ảnh 8: Kết quả vấn đề 1 Pandas

- Đánh giá kết quả:
- + Dân số Việt Nam tăng liên tục qua toàn bộ giai đoạn.
- + Tổng quy mô từ 88,145.4 nghìn (2011) lên 101,343.9 nghìn (2024), tăng khoảng 13,198.5 nghìn người (Khoảng 13.2 triệu người).
- + Nhịp tăng theo năm duy trì quanh 1.1–1.2% giai đoạn 2012–2020, sau đó có sự chững lại từ 2021 - 2023 (còn khoảng 0.84 - 0.98%) vì COVID-19 và dịch chuyển nhân khẩu học.
- + Năm 2024 ghi nhận phục hồi nhẹ lên khoảng 1.03%.
- + Tóm lại, xu hướng chung là tăng bền vững nhưng có dấu hiệu giảm tốc gần đây; đóng góp tăng trưởng đến từ cả tăng tự nhiên và di cư cơ học (đặc biệt về các trung tâm công nghiệp/đô thị).

1.2 Tìm vùng có mật độ dân số lớn nhất mỗi năm:

- Câu truy vấn:

```
dt2 = Locality_data[['Vùng', 'Năm', 'Mật độ dân số (Người/km2)']].copy()
idx = dt2.groupby('Năm')['Mật độ dân số (Người/km2)'].idxmax()
dt2_max = dt2.loc[idx]
dt2_max
```

Đoạn mã 26: Truy vấn vấn đề 2

- Kết quả truy vấn (TP.HCM):

#	Vùng	Năm	Mật độ dân số (Người/km ²)
686	TP. Hồ Chí Minh	2011	3633.1
687	TP. Hồ Chí Minh	2012	3717.2
688	TP. Hồ Chí Minh	2013	3805.1
689	TP. Hồ Chí Minh	2014	3882.6
690	TP. Hồ Chí Minh	2015	3964.6
691	TP. Hồ Chí Minh	2016	4113.3
692	TP. Hồ Chí Minh	2017	4196.4
693	TP. Hồ Chí Minh	2018	4289.9
694	TP. Hồ Chí Minh	2019	4385.0
695	TP. Hồ Chí Minh	2020	4404.0
696	TP. Hồ Chí Minh	2021	4375.0
697	TP. Hồ Chí Minh	2022	4481.0
698	TP. Hồ Chí Minh	2023	4513.1
699	TP. Hồ Chí Minh	2024	4554.6

Hình ảnh 9: Kết quả ván đề 2 Pandas

- Đánh giá kết quả:
- + TP.HCM đứng **đầu bảng cả 14/14 năm (2011–2024)** về *mật độ dân số (Người/km²)*.
- + Mật độ dao động từ khoảng **3.633** đến **khoảng 4.555** người/km², cho thấy xu hướng tăng rõ rệt theo thời gian.
- + Với việc mật độ dân số liên tục cao các năm tạo áp lực **hạ tầng, nhà ở, giao thông và dịch vụ công** tại TP.HCM ngày càng lớn và tăng dần; cần có chiến lược **giãn dân/dô thị vệ tinh** và nâng cấp năng lực hạ tầng để hấp thụ dòng di cư tiếp tục mạnh.

1.3 Xem vùng nào có tốc độ tăng dân số tự nhiên cao/thấp nhất:

- Câu truy vấn:

```
idx = Locality_data.groupby('Vùng')['Tỷ lệ tăng tự nhiên'].mean()
idx.loc[[idx.idxmax(), idx.idxmin()]]
```

Đoạn mã 27: Truy vấn ván đề 3

- Kết quả truy vấn (Bến Tre & Điện Biên):

A Vùng	# Tỷ lệ tăng tự nhiên
Điện Biên	15.53
Bến Tre	3.330714285714286

Hình ảnh 10: Kết quả ván đề 3 Pandas

- Đánh giá kết quả:
- + **Cao nhất: Điện Biên** với **khoảng 15.53%/năm** (trung bình), phản ánh mức sinh còn cao và cơ cấu dân số trẻ.
- + **Thấp nhất: Bến Tre** với **khoảng 3.33%/năm**, cho thấy mức sinh thấp và/hoặc già hóa dân số mạnh hơn mặt bằng chung.

- + **Khoảng cách chênh lệch (khoảng 12.2%)** giữa hai cực thể hiện **phân hóa nhân khẩu học vùng miền**; các tỉnh miền núi phía Bắc thường giữ mức tăng tự nhiên cao, trong khi một số tỉnh Đồng bằng sông Cửu Long ở mức thấp.

1.4 Kiểm tra sự cân bằng nhân khẩu học dựa vào tỷ suất sinh thô và chết thô:

- Câu truy vấn:

```
Locality_data['Tỷ suất chết thô'] = Locality_data['Tỷ suất chết thô'] * (-1)
Locality_data[['Tỷ suất sinh thô', 'Tỷ suất chết thô']].corr(method='pearson')
```

Đoạn mã 28: Truy vấn vấn đề 4

- Kết quả truy vấn (Bến Tre & Điện Biên):

...	# Tỷ suất sinh thô	# Tỷ suất chết thô
Tỷ suất sinh thô	1.0	0.020730964467089923
Tỷ suất chết thô	0.020730964467089923	1.0

Hình ảnh 11: Kết quả vấn đề 4 Pandas

- Đánh giá kết quả:
- + **Mức độ liên hệ:** Tương quan Pearson giữa **sinh thô** và **(-) chết thô** $\approx +0.021 \Rightarrow$ **gần như không có quan hệ tuyến tính**; còn giữa **sinh thô** và **chết thô** $\approx -0.021 \Rightarrow$ **cũng xấp xỉ 0**.
- + **Phân bố mức độ:** Trung bình giai đoạn toàn bộ dữ liệu: **sinh thô khoảng 16.06% (min 9.5; max 27.2)** và **chết thô khoảng 6.77% (min 2.72; max 14.4)**. **Độ trai của sinh thô rộng hơn** chết thô, nên biến thiên tăng tự nhiên chủ yếu đến từ **sự khác biệt về mức sinh**.
- + **Kết luận ngắn:** Nếu chỉ xét cặp **sinh–chết**, **không có mô hình “sinh cao và chết thấp” mang tính tuyến tính rõ rệt** trên toàn quốc. Chênh lệch tăng tự nhiên giữa các địa phương chủ yếu do **mức sinh khác nhau**, trong khi **mức chết biến thiên hẹp hơn** và không đồng biến/ nghịch biến rõ với mức sinh.

1.5 Đánh giá quá trình đô thị hóa của các vùng:

- Câu truy vấn:

```
dt3 = Locality_data[['Vùng', 'Năm', 'Tổng dân số', 'Tổng dân số thành thị']].copy()
dt3['Tỷ lệ dân số thành thị(%)'] = round(dt3['Tổng dân số thành thị'] / dt3['Tổng dân số'] * 100, 2)
dt3[['Vùng', 'Năm', 'Tỷ lệ dân số thành thị(%)']]
```

Đoạn mã 29: Truy vấn vấn đề 5

- Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

#	Vùng	Năm	Tỷ lệ dân số thành thị(%)
0	Hà Nội	2011	42.49
1	Hà Nội	2012	42.52
2	Hà Nội	2013	42.43
3	Hà Nội	2014	49.19
4	Hà Nội	2015	49.11
5	Hà Nội	2016	49.18
6	Hà Nội	2017	49.21
7	Hà Nội	2018	49.34
8	Hà Nội	2019	49.42
9	Hà Nội	2020	49.25
10	Hà Nội	2021	49.16
11	Hà Nội	2022	49.06
12	Hà Nội	2023	49.06
13	Hà Nội	2024	49.11

Hình ảnh 12: Kết quả ván đề 5 Pandas

- Đánh giá kết quả:
- + **Cá nước:** tăng 7.09% từ **31.40% (2011)** lên **38.49% (2024)**
- + Tỉnh thành có **đô thị hóa TB cao:** Đà Nẵng (**khoảng87.20%**), TP.HCM (**khoảng80.01%**), Bình Dương (**khoảng77.31%**), Cần Thơ (**khoảng68.97%**), Quảng Ninh (**khoảng62.24%**).
- + Tỉnh thành có **đô thị hóa TB thấp:** Bắc Giang (**khoảng14.15%**), Hưng Yên (**khoảng14.13%**), Sơn La (**khoảng13.91%**), Thái Bình (**khoảng10.78%**), Bến Tre (**khoảng10.38%**).
- + **Lưu ý biến động:** đa số **tăng**, nhưng có nơi **giảm nhẹ** (ví dụ: TP.HCM **-4.92 điểm %**, Kon Tum **-0.83**, Khánh Hòa **-0.45**). Bởi vì những tỉnh này đang có xu hướng đẩy ra các vệ tinh trung tâm xung quanh để giảm tải áp lực lên một khu vực.

1.6 So sánh thất nghiệp giữa thành thị và nông thôn:

- Câu truy vấn:

```
dt4 = Locality_data.copy()
dt4 = dt4.groupby(['Vùng'])[['Tỉ lệ thất nghiệp', 'Tỉ lệ thất nghiệp ở thành thị', 'Tỉ lệ thất nghiệp ở nông thôn']].mean().round(2)
dt4
```

Đoạn mã 30: Truy vấn ván đề 6

- Kết quả truy vấn (Với dữ liệu có 63 tỉnh/thành, nhóm xin phép lấy 12 tỉnh/thành đầu làm kết quả mẫu):

A) Vùng	...	# Tỉ lệ thất nghiệp	# Tỉ lệ thất nghiệp ở thành thị	# Tỉ lệ thất nghiệp ở nông thôn
An Giang		2.82	2.16	3.58
Bà Rịa - Vũng Tàu		3.15	1.8	3.19
Bình Dương		2.63	2.11	3.01
Bình Phước		2.59	1.7	3.72
Bình Thuận		2.66	2.1	3.1
Bình Định		2.58	2.12	2.7
Bạc Liêu		3.31	2.06	3.27
Bắc Giang		2.83	2.23	2.74
Bắc Kạn		3.0	2.05	3.21
Bắc Ninh		2.88	2.46	2.2
Bến Tre		2.66	1.72	3.89
Cao Bằng		2.9	2.79	3.48
Cà Mau		2.8	2.11	3.63

Hình ảnh 13: Kết quả vấn đề 6 SQL

- Đánh giá kết quả:
- + **Mặt bằng chung:** Thất nghiệp TB khoảng 2.96%; trong đó **thành thị** khoảng 2.24% và **nông thôn** khoảng 3.31% ⇒ **nông thôn cao hơn khoảng 1.06%**.
- + **Ý nghĩa:** Chênh lệch bền vững (**nông thôn > thành thị**) cho thấy thị trường lao động đô thị linh hoạt hơn (đa dạng ngành, dịch vụ), còn nông thôn nhạy hơn với mùa vụ, chuyển dịch cơ cấu, và kỹ năng.

1.7 Xem vùng có tăng dân số mạnh có tỷ lệ thất nghiệp cao không:

- Câu truy vấn:

```
Locality_data['Tăng dân số(%)'] = Locality_data.groupby('Vùng')[['Tổng dân số']].pct_change()*100
Locality_data.groupby('Vùng')[['Tăng dân số(%)','Tỉ lệ thất nghiệp']].mean().round(2).sort_values(by='Tăng dân số(%)', ascending=False)
```

Đoạn mã 31: Truy vấn vấn đề 7

- Kết quả truy vấn (Với dữ liệu có 63 tỉnh/thành, nhóm xin phép lấy 12 tỉnh/thành đầu làm kết quả mẫu):

A) Vùng	# Tăng dân số(%)	# Tỉ lệ thất nghiệp
Bình Dương	4.13	2.63
Bắc Ninh	2.86	2.88
Đắk Nông	2.38	2.42
Kon Tum	2.23	3.37
Đà Nẵng	2.14	3.23
Điện Biên	1.92	3.06
Hà Nội	1.9	2.63
Đồng Nai	1.82	3.29
Lai Châu	1.8	3.19
TP. Hồ Chí Minh	1.76	2.78
Lào Cai	1.68	3.24
Bắc Giang	1.65	2.83

Hình ảnh 14: Kết quả vấn đề 7 SQL

- Đánh giá kết quả:

- + **Mặt bằng chung:** Tăng dân số trung bình năm ($pct_change \approx 0.95\%$, thất nghiệp TB $\approx 2.96\%$).
- + **Tỉnh thành tăng trưởng dân số cao (+%):** Bình Dương (4.13%, thất nghiệp 2.63%), Bắc Ninh (2.86%, 2.88%), Đăk Nông (2.38%, 2.42%), Kon Tum (2.23%, 3.37%), Đà Nẵng (2.14%, 3.23%) do đó Nhóm công nghiệp/đô thị hóa mạnh dẫn đầu về tăng dân số.
- + **Tỉnh thành tăng trưởng dân số thấp (-%):** Cà Mau, Hậu Giang, Đồng Tháp, Sóc Trăng, An Giang (từ -0.24% đến -0.71%) do đó Phản ánh dòng di cư ra ngoài và/hoặc mức sinh giảm ở một số tỉnh DBSCL.
- + **Liên hệ với thất nghiệp:** $r \approx +0.03$ giữa *tăng dân số (%)* và *thất nghiệp* \Rightarrow hầu như không có quan hệ tuyến tính.
- + **Hàm ý:** Tăng dân số cao không đồng nghĩa thất nghiệp cao/thấp; biến động dân số chủ yếu chịu tác động di cư và hấp lực công nghiệp–dịch vụ, trong khi thất nghiệp phụ thuộc cấu trúc thị trường lao động từng địa phương

1.8 Xem vùng nào thu hút hay mất dân cư:

- Câu truy vấn:

```
dt5 = Locality_data[['Vùng', 'Tỷ suất nhập cư', 'Tỷ suất xuất cư', 'Tỷ suất di cư thuần']].copy()
dt5.groupby('Vùng')[['Tỷ suất nhập cư', 'Tỷ suất xuất cư', 'Tỷ suất di cư thuần']].mean()
dt5.round(3)
```

Đoạn mã 32: Truy vấn vấn đề 8

- Kết quả truy vấn (Với dữ liệu quá nhiều, nhóm xin phép lấy thành phố Hà Nội làm kết quả mẫu):

A] Vùng	# Tỷ suất nhập cư	# Tỷ suất xuất cư	# Tỷ suất di cư thuần
0 Hà Nội	11.0	6.4	4.7
1 Hà Nội	6.1	3.3	2.8
2 Hà Nội	7.7	7.4	0.3
3 Hà Nội	7.5	7.8	-0.4
4 Hà Nội	4.7	4.1	0.6
5 Hà Nội	4.6	2.6	2.0
6 Hà Nội	3.0	3.3	-0.3
7 Hà Nội	4.7	2.6	2.1
8 Hà Nội	8.8	2.5	6.3
9 Hà Nội	6.26	2.53	3.73
10 Hà Nội	7.68	1.75	5.93
11 Hà Nội	4.79	2.84	1.95
12 Hà Nội	5.85	1.96	3.89
13 Hà Nội	5.91	4.47	1.44

Hình ảnh 15: Kết quả vấn đề 8 SQL

- Đánh giá kết quả:
- + **Mặt bằng chung (TB toàn bộ dữ liệu):** $nhập cư \approx 4.962\%$, $xuất cư \approx 6.665\% \Rightarrow di cư thuần TB \approx -1.702\%$ (toàn quốc có xu hướng **mất dân nhẹ** do di cư ra ngoài).
- + **Thu hút dân cư (Top 5 di cư thuần TB):** Bình Dương (+36.48%), Bắc Ninh (+18.44%), TP.HCM (+9.39%), Đà Nẵng, Bà Rịa–Vũng Tàu do đó trung tâm công nghiệp/đô thị – việc làm.
- + **Mất dân cư (Bottom 5 di cư thuần TB):** Sóc Trăng (-11.62%), Cà Mau (-10.89%), An Giang (-10.63%), Bạc Liêu, Hậu Giang do đó khu vực DBSCL nổi bật xu hướng **di cư ra ngoài**.
- + **Kết luận:** Di cư thuần dương tập trung ở **cực công nghiệp – dịch vụ**, trong khi âm chủ yếu ở **nông nghiệp/đồng bằng hạ lưu**, phản ánh chênh lệch cơ hội việc làm & thu nhập giữa các vùng.

2. So sánh với SQL server.

Mục tiêu Phân tích	SQL (04_SQL_query.sql)	Pandas (05_Pandas_query.ipynb)	Ghi chú so sánh
1. Lấy dữ liệu dân số	select region, year, avg_population_thousand from population_data	Locality_data[['Vùng', 'Năm', 'Dân số trung bình (Nghìn người)']]	Cả hai đều là thao tác chọn cột cơ bản. Cú pháp SELECT...FROM của SQL so với cú pháp df[...] của Pandas.
2. Mật độ dân số cao nhất theo năm	Dùng CTE (Common Table Expression) (max_population) để tìm giá trị max cho mỗi năm. Sau đó JOIN CTE này ngược lại bảng chính.	Dùng .groupby('Năm') kết hợp với .idxmax() để tìm <i>chỉ số</i> (index) của hàng có giá trị lớn nhất. Sau đó dùng .loc[idx] để chọn các hàng đó.	Pandas ngắn gọn hơn đáng kể. Phương thức idxmax() là một shortcut rất mạnh, giúp tránh việc phải tự join (self-join) hoặc dùng CTE như SQL.
3. Tỷ lệ tăng tự nhiên cao/ thấp nhất	Dùng CTE (avg_natural_growth_rate) để tính trung bình theo vùng. Sau đó truy vấn CTE đó 2 lần với MAX() và MIN() trong mệnh đề WHERE.	Tính trung bình cho tất cả các vùng: idx = Locality_data.groupby('Vùng')['Tỷ lệ tăng tự nhiên'].mean(). Sau đó dùng .idxmax() và .idxmin() để tìm và chọn trực tiếp.	Tương tự Tác vụ 2, các phương thức idxmax()/idxmin() của Pandas cho phép tìm và truy xuất kết quả trực tiếp, trong khi SQL phải tính toán rồi truy vấn lại kết quả đó.
4. Tương quan sinh/c hết	<i>Không có trong file SQL.</i>corr(method='pearson')	Đây là điểm mạnh rõ rệt của Pandas. Các phân tích thống kê phức tạp (như hệ số tương quan Pearson) được tích hợp sẵn. Làm điều này trong SQL (đặc biệt là T-SQL) sẽ cực kỳ phức tạp, phải viết lại toàn bộ công thức toán học.
5. Tính toán đô thị hóa (Cột mới)	select, round((urban_population / total_population *100), 2) urbanization from ...	dt3['Tỷ lệ dân số thành thị(%)'] = round(dt3['Tổng dân số thành thị'] / dt3['Tổng dân số'] * 100, 2)	SQL tính toán và trả về cột mới <i>trong kết quả truy vấn</i> . Pandas <i>tạo ra cột mới</i> và lưu nó vào DataFrame trong bộ nhớ, cho phép sử dụng cột này ở các bước sau.
6. Thất nghiệp trung bình theo vùng	select REGION, area_type, Round(AVG(unemployment_rate),2) ... group by REGION, area_type	dt4 = dt4.groupby(['Vùng'])[['Tỷ lệ thất nghiệp', ...]].mean().round(2)	Cú pháp rất tương đồng về mặt khái niệm: GROUP BY của SQL tương ứng với .groupby() của Pandas, và AVG() tương ứng với .mean().
7. Tăng dân số &	Dùng Window Function (LAG(...)) OVER (PARTITION BY region ORDER BY year)) để lấy dân số của năm trước.	Dùng phương thức .pct_change() (percent change), được tối ưu hóa cho chuỗi thời gian, sau khi đã .groupby('Vùng').	Cả hai đều có các công cụ mạnh mẽ cho phân tích chuỗi/phân nhóm. LAG...OVER PARTITION BY của SQL là tiêu chuẩn.

Thất nghiệp			.pct_change() của Pandas (hoặc .shift() để tương đương LAG) rất ngắn gọn và rõ nghĩa.
8. Phân tích di cư	select ..., round((immigration_rate - emigration_rate),2) net_migration_rates from ...	dt5.groupby('Vùng')[...].mean()	Hai file của bạn đang thực hiện hai việc khác nhau ở đây. SQL (Tác vụ 8) chỉ tính toán một cột mới <i>trên mỗi hàng</i> (giống Tác vụ 5). Pandas (Tác vụ 8) lại thực hiện một phép <i>tổng hợp</i> (giống Tác vụ 6) để tính <i>trung bình</i> di cư theo Vùng.

Bảng 2: So sánh SQL server và Pandas

E. Trực quang hóa dữ liệu với Python:

- Quy trình trực quang hóa dữ liệu trải qua 5 bước:

1. Import thư viện và đọc dữ liệu:

- Đoạn mã:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

Data = pd.read_csv('Datasets/Clean_data/Locality_Analyst.csv')
```

Đoạn mã 33: Đọc dữ liệu Trực quang hóa

2. Chuẩn bị dữ liệu:

- Đoạn mã:

```
Data['Tỷ lệ dân số thành thị(%)'] = round(Data['Tổng dân số thành thị'] / Data['Tổng dân số'] * 100, 2)
Data['Tỷ lệ dân số nông thôn(%)'] = round(Data['Tổng dân số nông thôn'] / Data['Tổng dân số'] * 100, 2)

Data_mean = round(Data.groupby('Vùng', as_index=False).mean(numeric_only=True), 2)
Data_mean = Data_mean.drop(columns='Năm')

year_mean = (Data.drop(columns=['Vùng', 'Khu vực'])).copy()
year_mean = round(Data.groupby('Năm', as_index=False).mean(numeric_only=True), 2)
```

Đoạn mã 34: Chuẩn bị dữ liệu Trực quang hóa

- Giải thích:
 - Thêm cột tỷ lệ (%) cho tổng dân số thành thị hoặc nông thôn sẽ giúp các biểu đồ trực quang được rõ ràng đối với các tỉnh thành.
 - Tạo dataframe Data_mean dùng để tính trung bình 14 năm theo các vùng. Việc này giúp xem các biểu đồ theo cột Vùng không bị loạn bởi 1 tỉnh thay vì phải biểu diễn 14 năm thì chỉ cần giá trị trung bình để dễ dàng phân tích.

- Tạo dataframe year_mean tương tự với Data_mean nhưng thay vì biểu diễn theo cột Vùng (hay tỉnh thành) thì year_mean dùng để biểu diễn theo cột Năm

3. Phân tích hàm thống kê mô tả:

- Đoạn mã:

```
Data.describe()
```

Đoạn mã 35: Phân tích hàm thống kê mô tả

- Kết quả:

	# Năm	# Diện tích(Km2)	# Dân số trung bình (Nghìn)	# Mật độ dân số (Người/km ²)	# Tổng dân số
count	882.0	882.0	882.0	882.0	882.0
mean	2017.5	5255.39149659864	1505.208843574149	504.97040816326535	1505.2002834467119
std	4.033416039651957	3650.3962008102835	1378.4151018709342	641.6995271387846	1378.4168372504269
min	2011.0	822.7	298.7	43.3	298.69
25%	2014.0	2359.575	868.6500000000001	137.0	868.6500000000001
50%	2017.5	4701.2	1208.05	269.8	1208.065
75%	2021.0	6871.5	1631.95	667.8	1631.94
max	2024.0	16493.7	9543.6	4554.6	9543.63

Hình ảnh 16: Phân tích hàm thống kê mô tả (1)

# Nam	# Nữ	# Tổng dân số thành thị	# Tổng dân số nông thôn	# Tỷ suất sinh thô	# Tỷ suất chết thô
882.0	882.0	882.0	882.0	882.0	882.0
748.2335600907029	756.9670748299319	524.7049092970522	980.4955102040817	16.057120181405896	6.768038548752834
675.7918160825956	702.8341799261601	970.6641298930931	651.7186569142065	3.1602544986635546	1.4280537332424261
150.92	147.77	50.55	126.71	9.5	2.72
434.8775	433.395	174.7974999999999	589.3925	13.7	5.0
605.2	605.3	266.65	810.2449999999999	15.969999999999999	6.795
814.0899999999999	817.4225	483.2274999999996	1167.4375	18.1	7.6
4626.65	4916.98	7428.6	4436.79	27.2	14.4

Hình ảnh 17: Phân tích hàm thống kê mô tả (2)

# Tỷ lệ tăng tự nhiên	# Tỷ suất nhập cư	# Tỷ suất xuất cư	# Tỷ suất di cư thuần	# Tỷ lệ nam nữ
882.0	882.0	882.0	882.0	882.0
9.284773242630386	4.962108843537415	6.6651020408163255	-1.7023809523809523	99.53073696145123
3.503374861412685	7.479989413877968	3.8438646133172134	7.9939088758030294	2.5623778553375907
-0.94	0.1	0.8	-23.76	90.52
7.0	1.5	3.8625	-5.0	97.7025
9.129999999999999	2.9	6.08	-2.365	99.515
11.6	5.2	8.4525	-0.2025	101.35
19.8	70.2	25.79	58.6	108.08

Hình ảnh 18: Phân tích hàm thống kê mô tả (3)

# Tí lệ thất nghiệp	# Tí lệ thất nghiệp ở thành...	# Tí lệ thất nghiệp ở nông...	# Tí lệ dân số thành thị(%)	# Tí lệ dân số nông thôn(%)
882.0	882.0	882.0	882.0	882.0
2.9641950113378686	2.2442290249433103	3.3071768707483	28.96208616780045	71.03785714285715
1.3840706706686903	1.1417353051734065	1.5140905611878213	17.197121157905904	17.197272104047734
0.47	0.31	0.64	9.76	12.19
1.74	1.214999999999999	2.01	16.9125	67.14
3.02	2.285	3.325	23.335	76.665
4.21	3.17	4.57	32.86	83.0875
5.25	4.27	6.01	87.81	90.24

Hình ảnh 19: Phân tích hàm thống kê mô tả (4)

- Giai thích:

Thống kê	Ý nghĩa
count	Đếm số dòng có dữ liệu (không bị thiếu - NaN)
mean	Giá trị trung bình (mean)
std	Độ lệch chuẩn (standard deviation) – cho biết mức độ phân tán dữ liệu
min	Giá trị nhỏ nhất

25%	Phân vị thứ 25 (Q1) — 25% dữ liệu nhỏ hơn giá trị này
50%	Phân vị thứ 50 (Q2) — cũng là trung vị (median)
75%	Phân vị thứ 75 (Q3) — 75% dữ liệu nhỏ hơn giá trị này
max	Giá trị lớn nhất

Bảng 3: Giải thích hàm thống kê

4. Viết các hàm vẽ biểu đồ:

1. Biểu đồ scatter:

- Đoạn mã:

```
def visual_scatter(var: str):
    label = 'Tổng dân số'
    plt.scatter(Data[var], Data[label] , alpha=0.3)
    plt.xlabel(var)
    plt.ylabel(label)
```

Đoạn mã 36: Biểu đồ scatter

- Giải thích:

- + Mục đích: Dùng để vẽ biểu đồ phân tán giữa trục X là một biến bất kì tùy thuộc vào tham số *var* và trục Y là cột “Tổng dân số” được làm cố định. Điều này có nghĩa là nó giúp chúng ta xem được mối tương quan giữa cột “Tổng dân số” và các cột muôn tham chiếu.
- + Hàm *plt.scatter(Data[var], Data[label] , alpha=0.3)* vẽ biểu đồ phân tán giữa *Data[var]* (trục X) và *Data['Tổng dân số']* (trục Y), với độ trong suốt *alpha=0.3*(30% độ trong suốt) để dễ quan sát các điểm chồng nhau.

2. Biểu đồ cột ngang:

- Đoạn mã:

```
def column_chart_vertical(temp: str):
    df = Data_mean.sort_values('Vùng').copy()
    x = df['Vùng']
    y = pd.to_numeric(df[temp], errors="coerce")

    plt.figure(figsize=(8,12))
    plt.barh(x, y, height=0.6)
    plt.xlabel(temp); plt.ylabel('Vùng')
    plt.title(f"{temp} theo {'Vùng'}")
    plt.grid(axis="x")
    plt.margins(y = 0.01)
    plt.tight_layout(); plt.show()
```

Đoạn mã 37: Biểu đồ cột ngang

- Giải thích:

- + Mục đích: Dùng để vẽ biểu đồ cột ngang để hiển thị giá trị trung bình của tham số *temp* theo từng vùng. Kết hợp sử dụng datasets mới (*Data_mean*) là bản tổng hợp trung bình theo vùng từ file *Locality_Analyst.csv* thì mỗi vùng (như Hà Nội, Cà Mau, Bình Định,...) sẽ là một hàng.
- + Biểu đồ gồm : trục Y (ngang) thể hiện tên vùng và trục X (dọc) là giá trị của biến *temp*.
- + *plt.figure(figsize=(8,12))*: Tạo vùng vẽ kích thước 8x12 inch – biểu đồ dài theo chiều dọc.
- + *plt.barh(x, y, height=0.6)*: Vẽ biểu đồ cột ngang (barh)mỗi vùng là một thanh, chiều dài theo giá trị y và khoảng cách giữa các cột là 0.6 inch.
- + Có thêm *plt.grid(axis="x")* và *plt.tight_layout()* để biểu đồ rõ ràng, cân đối. Giúp so sánh nhanh giữa các vùng về một chỉ tiêu cụ thể.

3. Biểu đồ cột ngang kép:

- Đoạn mã:

```
def double_column_chart_vertical(var, temp: str):
    df = Data_mean.sort_values('Vùng').copy()
    x_labels = df['Vùng']
    y_var = pd.to_numeric(df[var], errors="coerce")
    y_temp = pd.to_numeric(df[temp], errors='coerce')

    plt.figure(figsize=(8,20))
    x = np.arange(len(x_labels))
    height=0.4
    plt.barh(x - height/2,y_var, height=height, color='#4A90E2', label=var)
    plt.barh(x + height/2,y_temp, height=height, color='#E67E22', label=temp)

    plt.yticks(ticks=x, labels=x_labels)
    plt.xlabel('Vùng'); plt.ylabel('Giá trị tổng quát')
    plt.title(f'Giá trị {var} và {temp} theo từng tỉnh')
    plt.grid(axis='x')
    plt.legend()
    plt.margins(y=0.035)
    plt.tight_layout()
    plt.show()
```

Đoạn mã 38: Biểu đồ cột ngang kép

- Giải thích:

- + Mục đích: Hàm này được thiết kế để so sánh trực quan hai tham số đầu vào với định lượng khác nhau giữa các tỉnh thành (Vùng). Mỗi vùng được biểu diễn bằng hai thanh song song giúp người xem so sánh mức độ chênh lệch hoặc tương quan giữa hai chỉ tiêu trong cùng một vùng.
- + Sử dụng 1 biến lưu trữ cho dataframe “Vùng” là *x_labels* để dễ tính toán khoảng cách phù hợp giữa các thanh “Vùng”. Sau khi sử dụng hàm *np.arange(len(x_labels))*, ta sẽ có số lượng cột “Vùng” để phân bổ 2 thanh ngang cho mỗi thanh “Vùng” phù hợp.
- + Cấu hình vị trí các thanh song song: X là đại diện cho vị trí từng vùng trên trục tung và 2 câu lệnh *plt.barh()* giúp vẽ 2 thanh ngang song song cho mỗi vùng với một thanh xanh lam cho tham số *var* và một thanh cam cho tham số *temp*. Các thanh được tránh chồng lấp lên nhau dựa vào công thức dịch lên xuống: $x \pm \text{height}/2$
- + Lệnh *plt.yticks()* giúp hiển thị lại tên vùng tương ứng với từng cặp cột và *plt.legend()* thêm chú thích màu để phân biệt hai biến so sánh.

4. Biểu đồ cột dọc:

- Đoạn mã:

```
def column_chart_horizontal(temp: str):
    df = year_mean.sort_values('Năm').copy()
    x = pd.to_numeric(df['Năm'], errors="coerce")
    y = pd.to_numeric(df[temp], errors="coerce")

    plt.figure(figsize=(12,6))
    plt.bar(x, y)
    plt.xlabel('Năm'); plt.ylabel(temp)
    plt.title(f'{temp} theo {\'Năm\'}')
    plt.grid(axis="y")
    plt.tight_layout()
    plt.show()
```

Đoạn mã 40: Biểu đồ cột dọc

- Giải thích:

- + Mục đích: Dùng để biểu diễn sự thay đổi của một chỉ tiêu (tham số temp) theo thời gian (theo từng năm) bằng biểu đồ cột đứng. Mục tiêu là giúp người xem quan sát được xu hướng tăng giảm qua các năm.
- + Hàm sử dụng datasets mới (year_mean) là bản tổng hợp trung bình năm từ file *Locality_Analyst.csv* thì mỗi năm sẽ là một cột.
- + *plt.bar(x, y)*: Vẽ biểu đồ cột dọc (bar) với mỗi năm là một cột, chiều cao theo giá trị y.

5. Biểu đồ cột dọc kép:

- Đoạn mã:

```
def double_column_chart(var, temp: str):
    df = year_mean.sort_values('Năm').copy()
    x = pd.to_numeric(df['Năm'], errors="coerce")
    y_var = pd.to_numeric(df[var], errors="coerce")
    y_temp = pd.to_numeric(df[temp], errors='coerce')

    plt.figure(figsize=(20,8))
    width=0.4
    plt.bar(x - width/2,y_var, width=width, color="#4A90E2", label=var)
    plt.bar(x + width/2,y_temp, width=width, color='#E67E22', label=temp)

    plt.xlabel('Năm'); plt.ylabel('Giá trị tổng quát')
    plt.title(f'Giá trị {var} và {temp} theo năm')
    plt.grid(axis='y')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Đoạn mã 40: Biểu đồ cột dọc kép

- Giải thích:
- + Mục đích: Dùng để so sánh hai biến định lượng khác nhau theo thời gian (theo năm) bằng biểu đồ cột kép với trục X biểu diễn các năm và trục Y thể hiện giá trị của hai tham số cần so sánh. Mỗi năm sẽ biểu thị hai cột song song giúp người xem đánh giá được sự khác biệt và tương quan giữa hai chỉ tiêu qua từng năm. Ngoài ra, còn giúp người xem quan sát được xu hướng biến động của mỗi tham số theo thời gian.
- + Cấu hình vị trí các cột song song: X là đại diện cho vị trí từng năm trên trục hoành và 2 câu lệnh *plt.bar()* giúp vẽ 2 cột song song tương ứng cho một năm. Các cột được tránh chồng lấp lên nhau dựa vào công thức dịch sang trái/phải: $x \pm \text{width}/2$
- + Sự khác biệt khi so với cột ngang là không cần tạo một biến đếm các phần tử. Bởi vì ở cột ngang là sử dụng cột “Vùng” với các giá trị ở kiểu dữ liệu String nên thư viện **Matplotlib** không hiểu đó là tọa độ nên cần tạo ra biến đếm các chỉ số tương ứng cho từng vùng. Còn ở cột dọc biểu diễn cho cột Năm-là dữ liệu số nên các cột sẽ tự động tách nhau theo giá trị số năm mà không cần biến đếm.

6. Biểu đồ ma trận tương quan:

- Đoạn mã:

```
def plot_correlation_heatmap(df: pd.core.frame.DataFrame, title_name: str='Train
correlation') -> None:

    corr = df.corr()
    fig, axes = plt.subplots(figsize=(12, 8))
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    sns.heatmap(corr, mask=mask, linewidths=.5, cmap='RdBu_r', annot=True,
    annot_kws={"size": 8})
    plt.title(title_name)
```

```
plt.show()
```

Đoạn mã 41: Biểu đồ ma trận tương quan

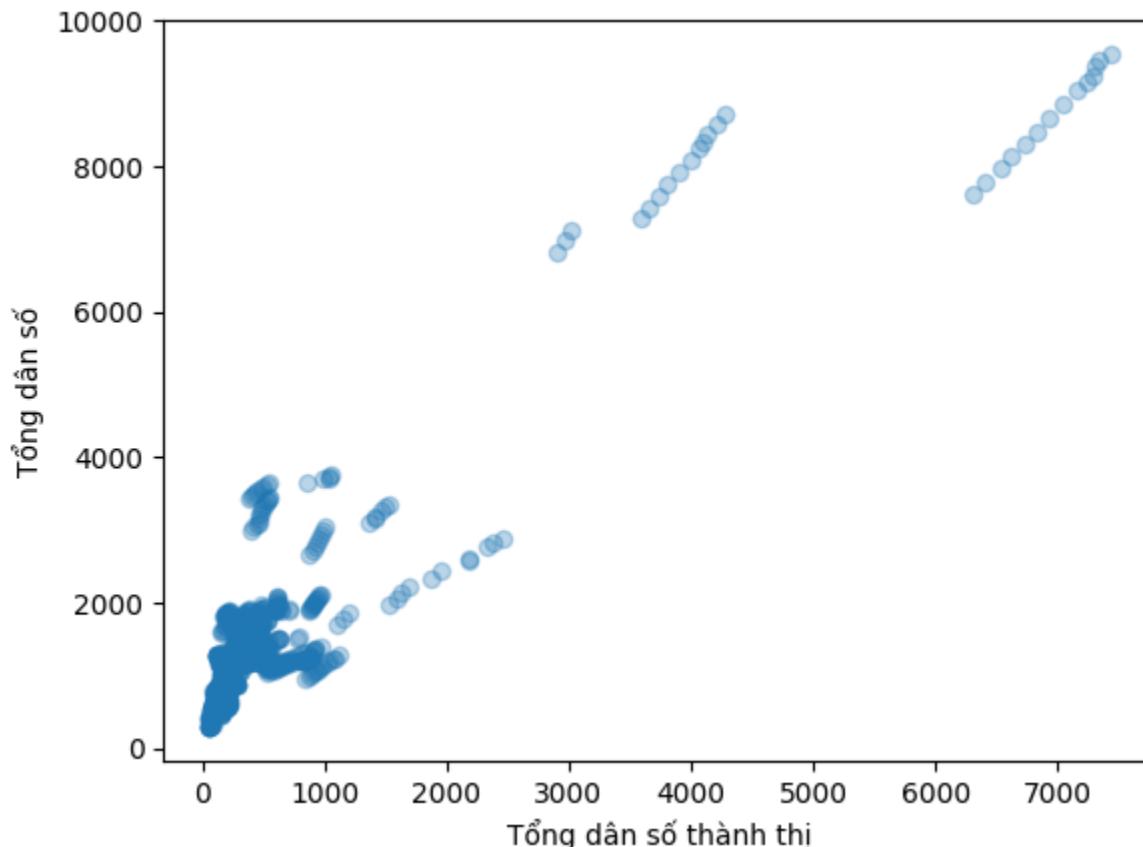
- Giải thích:
 - + Mục đích: Dùng để vẽ bản đồ nhiệt (heatmap) biểu diễn mức độ tương quan giữa các biến số trong Dataframe, giúp người xem nhanh chóng nhận biết mối liên hệ tuyến tính mạnh hay yếu (dương hay âm) giữa các biến định lượng.
 - + Hàm `df.corr()` dùng để tính ma trận hệ số tương quan Pearson giữa các cột số trong tham số đầu vào Dataframe và trả kết quả là ma trận vuông có giá trị [-1, 1].
 - + Hàm `plt.subplots()` làm hàm khởi tạo không gian vẽ và trả về 2 đối tượng là fig và axes. Đối tượng Figure, đại diện cho toàn bộ khung hình và đối tượng Axes, là vùng vẽ biểu đồ bên trong figure (nơi các trục, điểm, màu được hiển thị).
 - + Hai câu lệnh `mask = np.zeros_like(corr); mask[np.triu_indices_from(mask)] = True`: dùng tạo mặt nạ (mask) để ẩn nửa trên của ma trận tương quan (do ma trận đối xứng, chỉ cần vẽ một nửa là đủ)
 - + Ý nghĩa các tham số trong hàm `sns.heatmap()`: `mask=mask` là ẩn phần đối xứng trên, `linewidths=.5` là kẻ viền giữa các ô cho dễ đọc, `cmap='RdBu_r'` là màu đỏ–xanh, thể hiện tương quan âm–dương, `annot=True` là hiển thị giá trị số lên từng ô và `annot_kws={"size": 8}` là các chữ số hiển thị trên mỗi ô có kích thước font = 8.

5. Trục quang hóa dữ liệu:

Với mỗi biểu đồ nhóm xin lấy hai kết quả mẫu để đánh giá, ngoại trừ Biểu đồ ma trận tương quan:

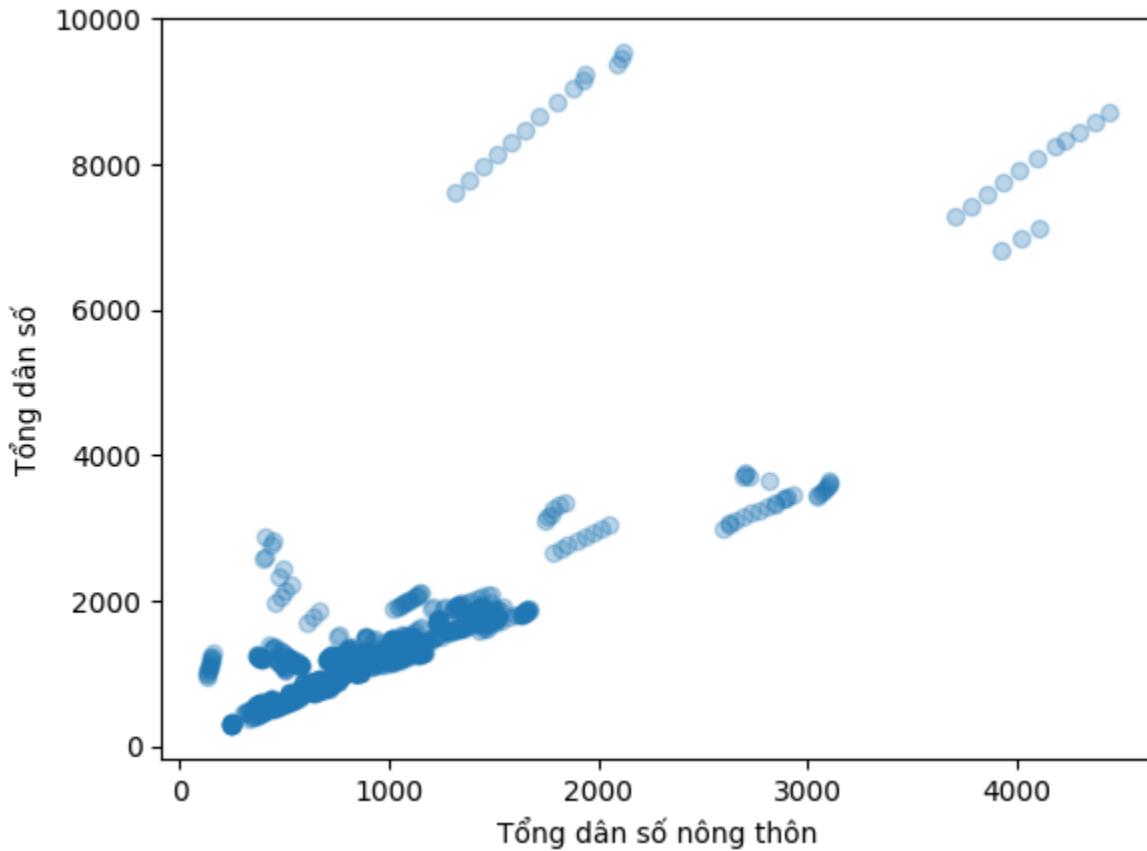
1. Biểu đồ scatter:

- So với cột “Tổng dân số thành thị”:
- + Kết quả:



Hình ảnh 20: Biểu đồ scatter (1)

- + Đánh giá: “Tổng dân số”so với “Tổng dân số thành thị”
 - Quan hệ tuyến tính dương rất mạnh (điểm nằm gần một dải thẳng).
 - Có cụm outlier ở góc phải trên rất có thể là các tỉnh đông dân như Hồ Chí Minh hoặc Hà Nội.
 - Kết luận: Tỉnh thành có dân số thành thị lớn gần như chắc chắn có tổng dân số lớn.
- So với cột “Tổng dân số nông thôn”
- + Kết quả:



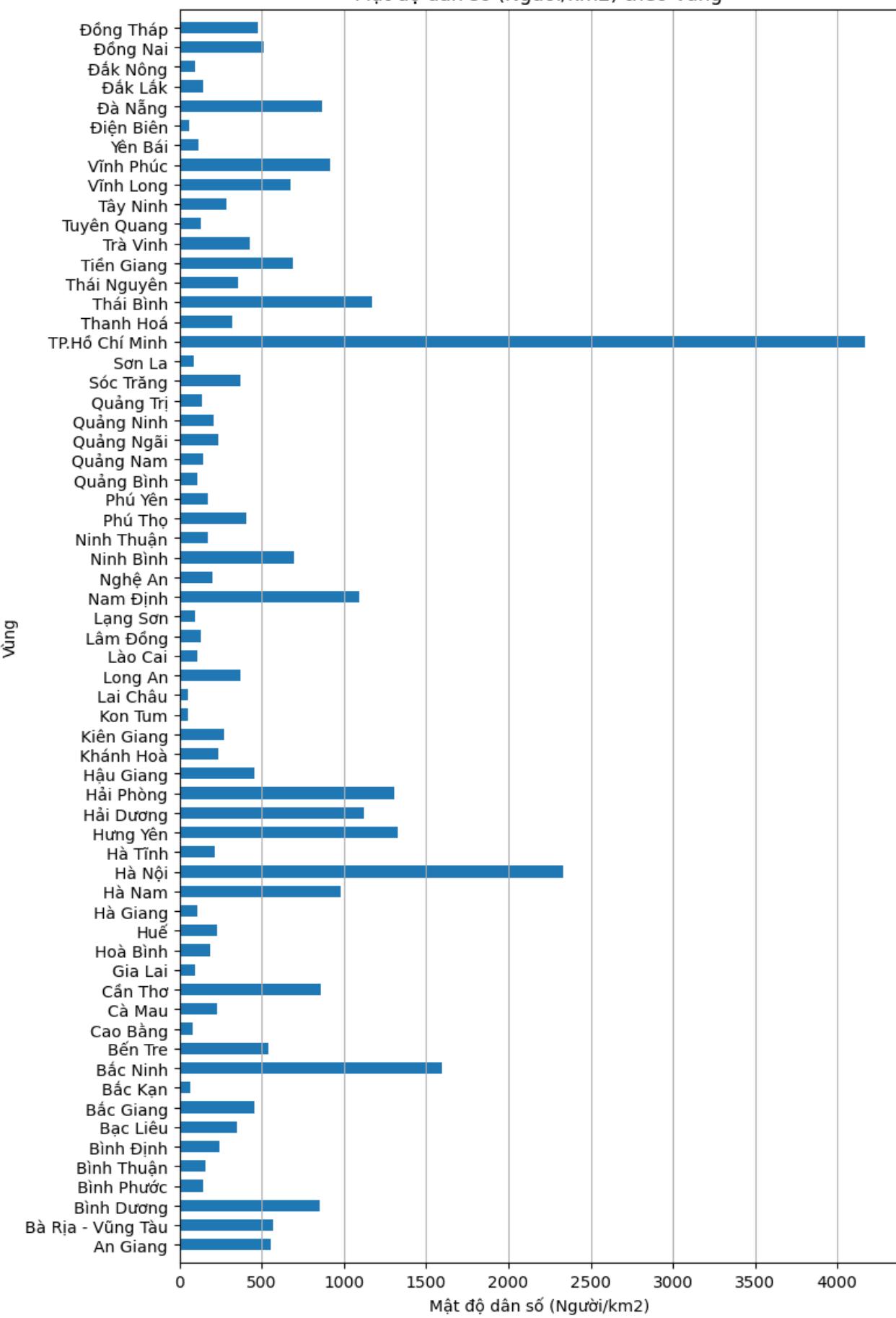
Hình ảnh 21: Biểu đồ scatter (2)

- + Đánh giá: “Tổng dân số”so với “Tổng dân số nông thôn”
 - Cũng dương mạnh nhưng **phân tán rộng hơn** so với thành thị (độ dốc nhỏ hơn).
 - Nhiều tỉnh có quy mô nông thôn tương đương nhưng tổng dân số khác nhau do đó cấu trúc đô thị hóa tạo khác biệt.

2. Biểu đồ cột ngang:

- So với cột “Mật độ dân số (Người/km2)”
- + Kết quả:

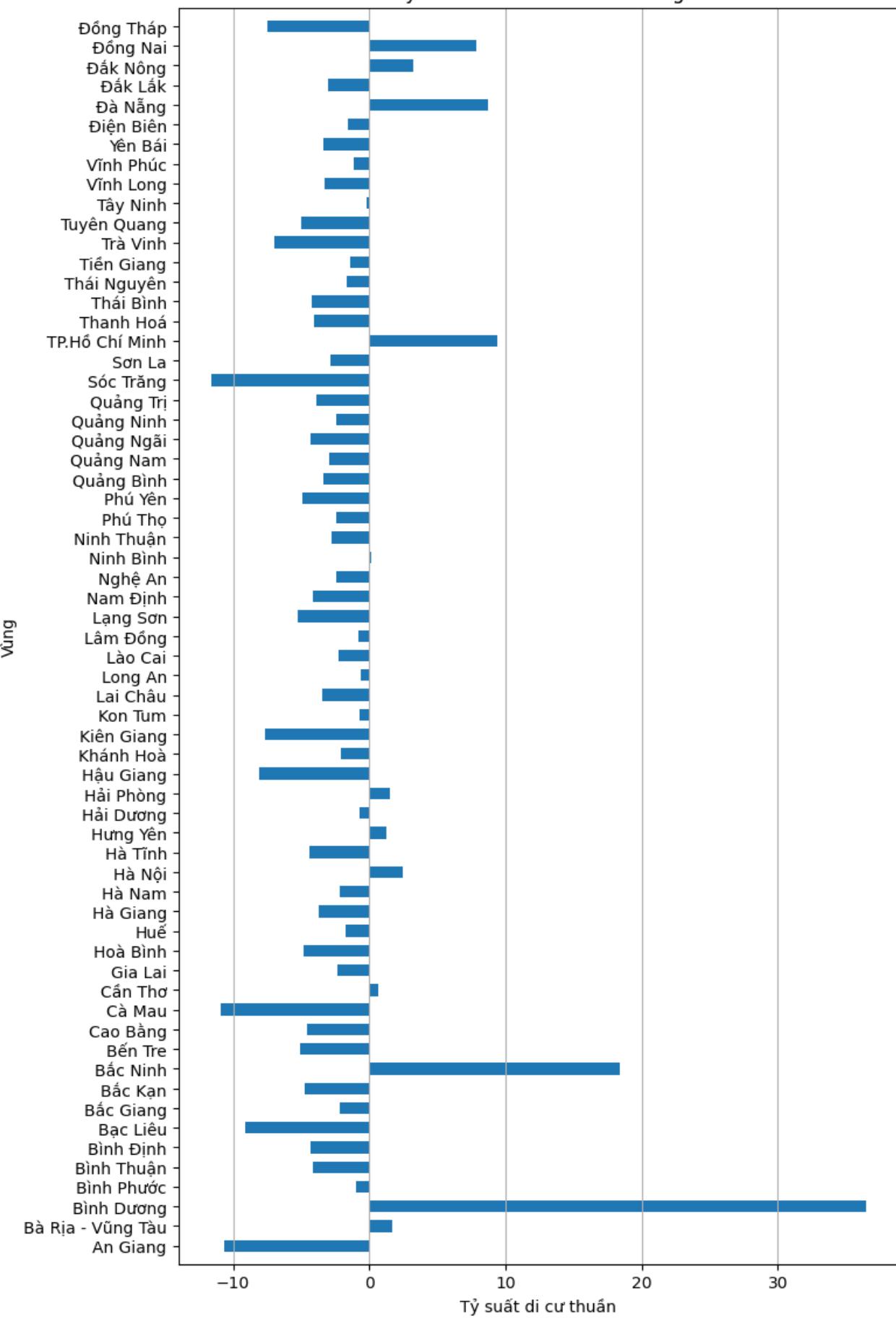
Mật độ dân số (Người/km²) theo Vùng



Hình ảnh 21: Biểu đồ cột ngang (1)

- + Đánh giá:
 - Nhóm phân hóa rất mạnh: TP.HCM (khoảng 4.1k) và Hà Nội (khoảng 2.3–2.4k) là ngoại lai dương; nhóm cao tiếp theo: Bắc Ninh, Hải Phòng, Hưng Yên, Nam Định (khoảng 1.0–1.6k).
 - Phần lớn tỉnh còn lại ở mức thấp-trung bình (<1k).
- So với cột “Tỷ suất di cư thuần”
- + Kết quả:

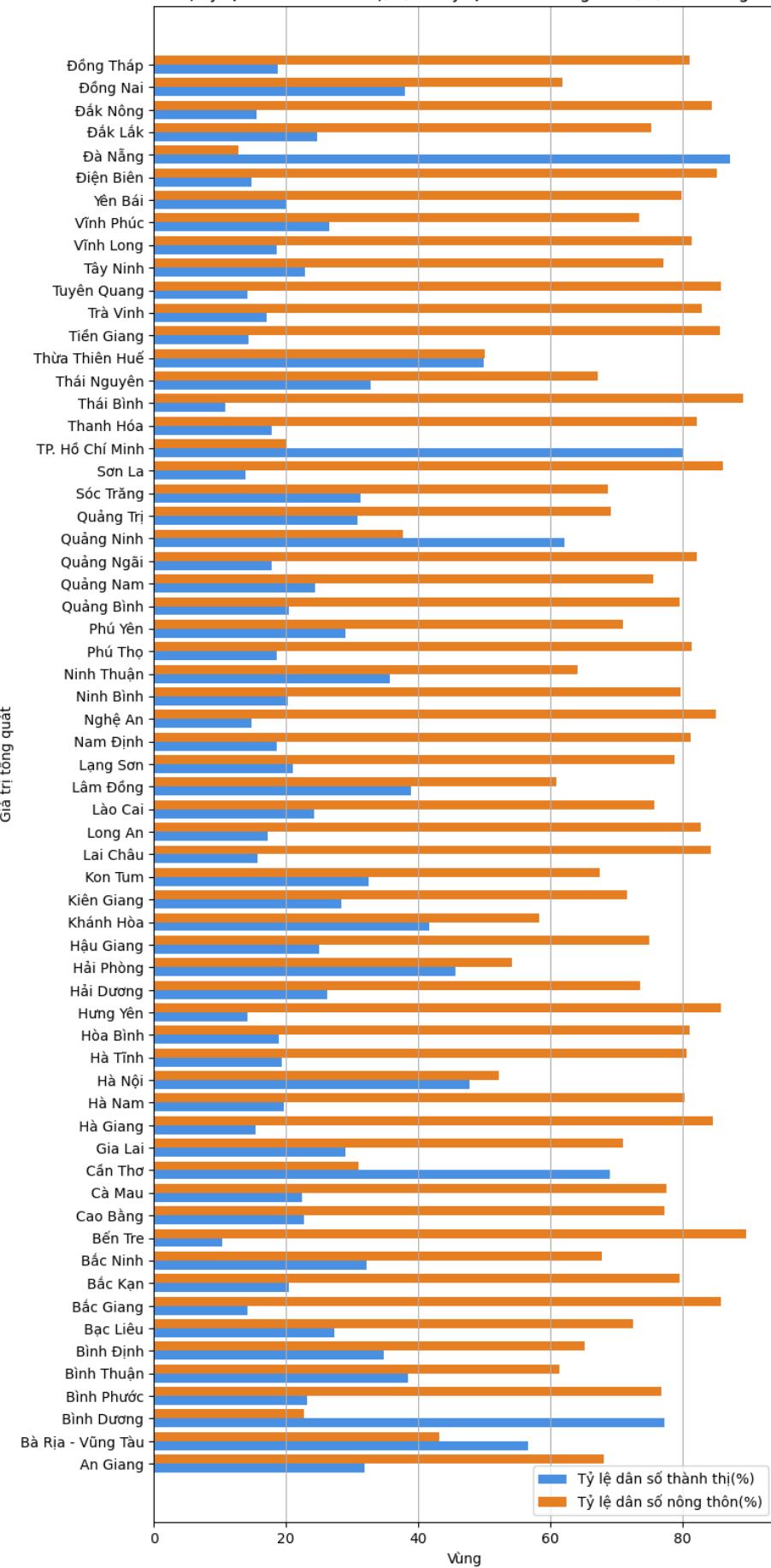
Tỷ suất di cư thuần theo Vùng



Hình ảnh 22: Biểu đồ cột ngang (2)

- + Đánh giá:
 - **Dương mạnh:** Bình Dương (khoảng 35–36%), Bắc Ninh (khoảng 18–19%), TP.HCM (khoảng 10%) do đó thu hút lao động/nhập cư ròng.
 - **Âm:** nhiều tỉnh Đồng bằng sông cửu long như An Giang, Cần Thơ, Sóc Trăng, Bạc Liêu, Cà Mau... do đó **xuất cư ròng**.
- 3. Biểu đồ cột ngang kép:
 - So với “Tỷ lệ dân số thành thị(%)” và “Tỷ lệ dân số nông thôn(%)”
 - + Kết quả:

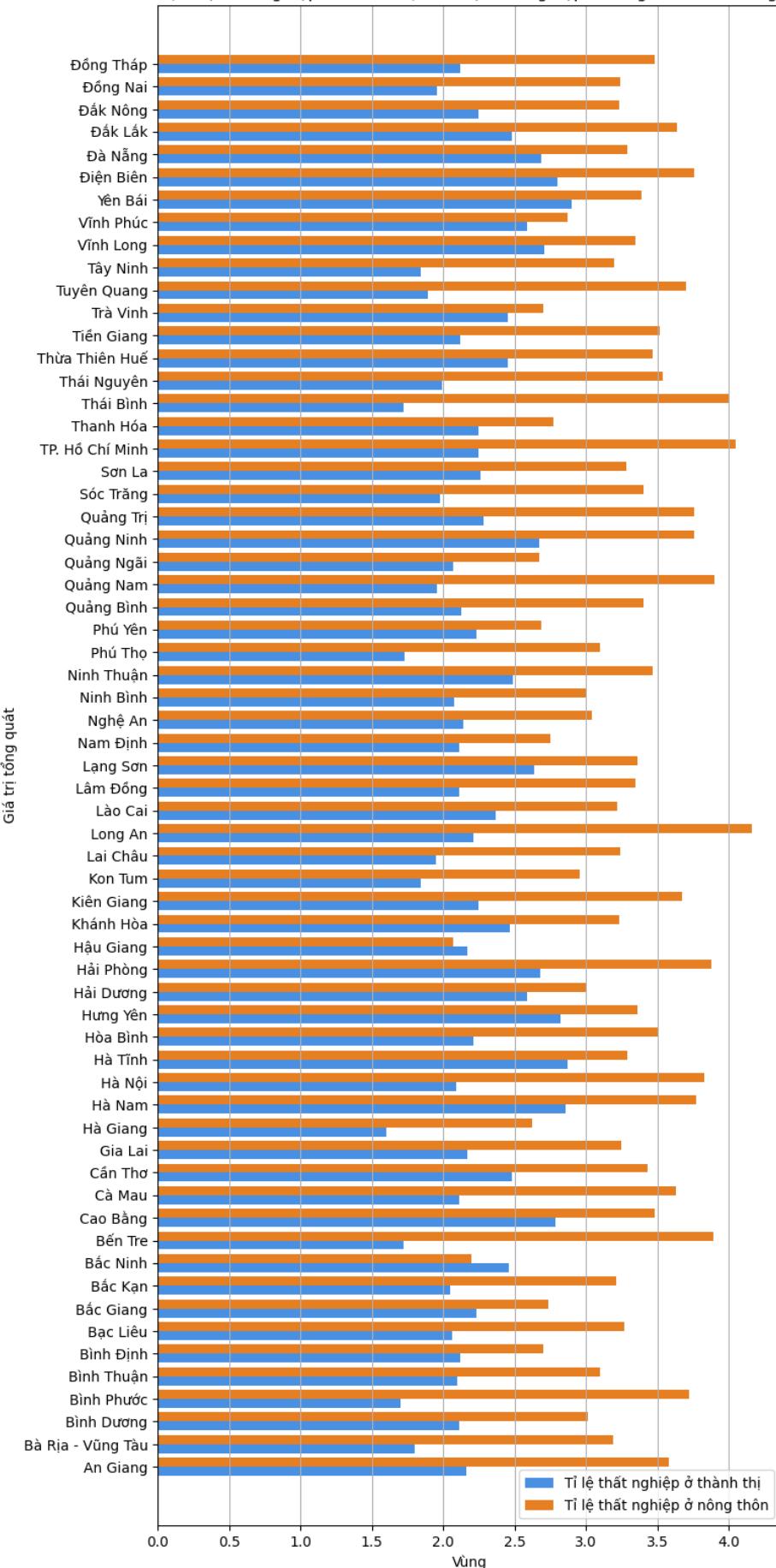
Giá trị Tỷ lệ dân số thành thị(%) và Tỷ lệ dân số nông thôn(%) theo từng tỉnh



Hình ảnh 23: Biểu đồ cột ngang kép(1)

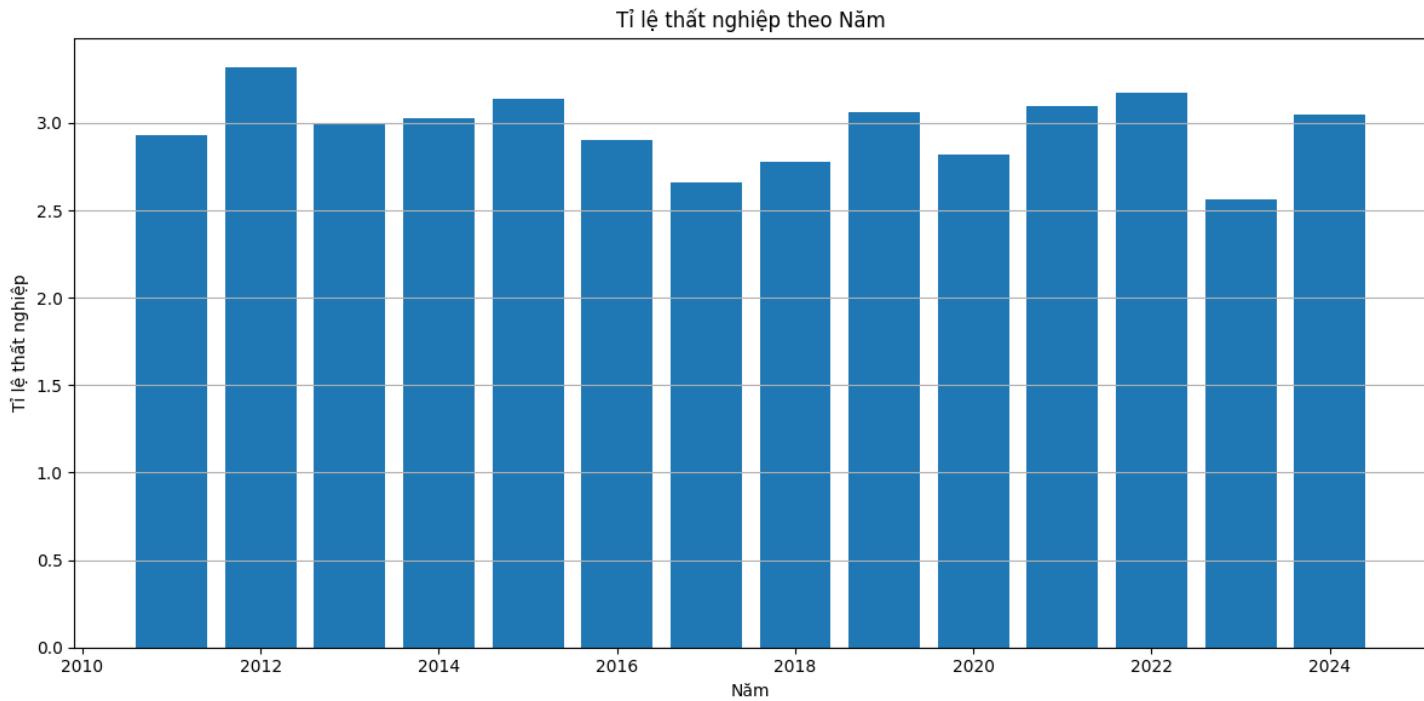
- + Đánh giá:
 - Hai tỷ lệ bù 100%, điều này phản ánh mức đô thị hóa.
 - Đô thị hóa rất cao: TP.Hồ Chí Minh, Đà Nẵng, Bình Dương, Quảng Ninh.
 - Nông thôn trội (>70–80%): nhiều tỉnh miền núi/ĐBSCL.
 - Liên vùng: các tỉnh thuộc Đông Nam Bộ và đô thị trung tâm cao hơn rõ; các tỉnh thuộc miền núi, Tây Nguyên và ĐBSCL thiên nông thôn.
 - Hàm ý: nơi đô thị hóa cao do đó mật độ lớn, nhập cư dương, áp lực hạ tầng; nơi nông thôn trội do đó cần đầu tư đô thị hóa, dịch vụ & CN nhẹ để giảm áp lực lên các nơi có đô thị hóa cao.
- So với “Tỉ lệ thất nghiệp ở thành thị” và “Tỉ lệ thất nghiệp ở nông thôn”
- + Kết quả:

Giá trị Tỉ lệ thất nghiệp ở thành thị và Tỉ lệ thất nghiệp ở nông thôn theo từng tỉnh



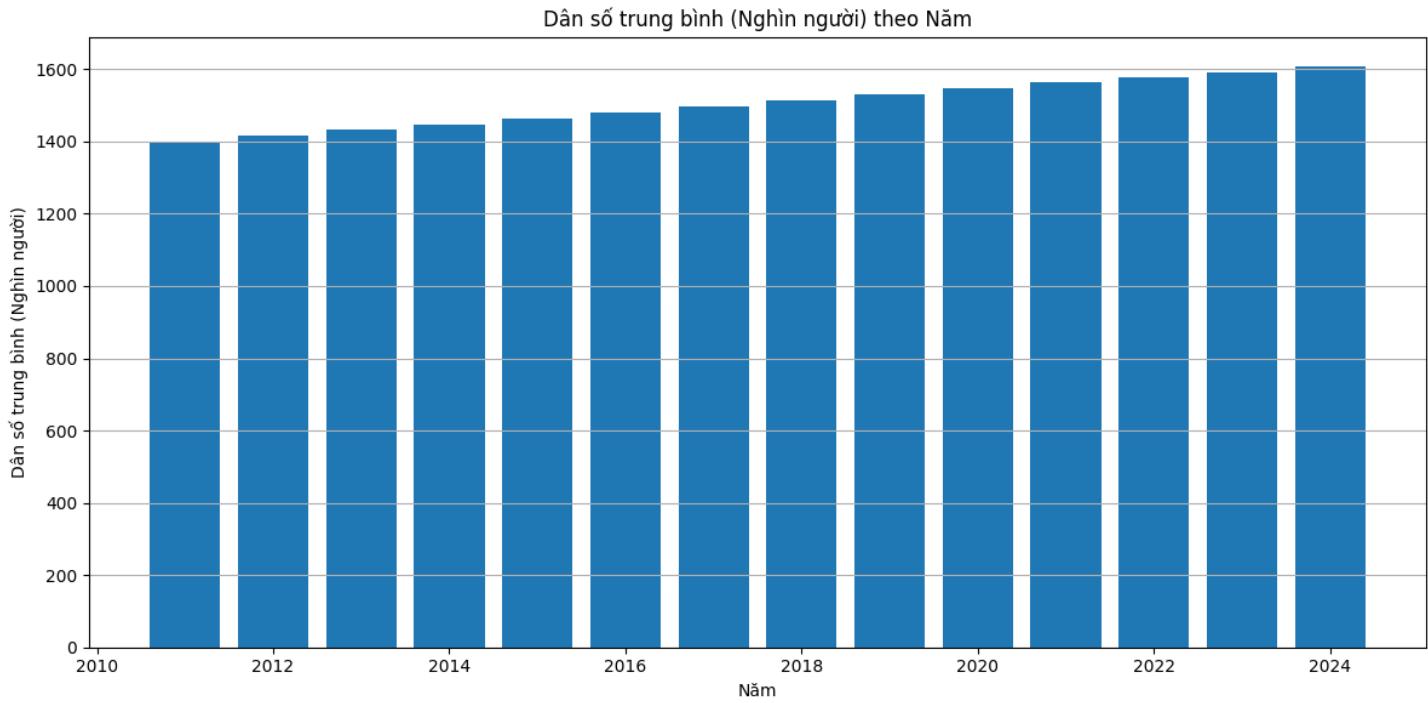
Hình ảnh 24: Biểu đồ cột ngang kép(2)

- + Đánh giá:
 - Toàn cục: nông thôn > thành thị ở đa số tỉnh (chênh khoảng 0.3 đến 1.5%).
 - Cao nhất nông thôn (khoảng 3.3 đến 4%): nhiều tỉnh DBSCL và một số duyên hải/miền núi (như Long An, Hậu Giang, An Giang...).
 - Thấp nhất thành thị (khoảng 1.7 đến 2.2%): rải ở tỉnh công nghiệp/dịch vụ (như Bà Rịa-Vũng Tàu, Tây Ninh...).
 - Ngoại lệ: có nơi chênh >1.5 điểm % do đó rủi ro thất nghiệp nông thôn cao.
 - Hàm ý: tạo việc phi nông nghiệp ở nông thôn, gắn khu công nghiệp – đào tạo nghề.
- 4. Biểu đồ cột dọc:
- So với “Tỉ lệ thất nghiệp”
- + Kết quả:



Hình ảnh 25: Biểu đồ cột dọc(1)

- + Đánh giá:
 - Mức độ và biên độ: dao động trong khoảng 2.6%–3.3% do đó thị trường lao động khá ổn định ở mức trung bình khoảng 3%.
 - Các mốc đáng chú ý:
 - Cao quanh 2012, 2015, 2021–2022 (khoảng 3.1–3.3%).
 - Thấp ở 2017 (khoảng 2.7%) và 2023 (khoảng 2.6%), sau đó tăng lên năm 2024 (khoảng 3.0%).
 - Giai đoạn 2020–2022 có xu hướng cao hơn (khả năng liên quan các cú sốc vĩ mô/dịch bệnh), 2023 hạ nhiệt, 2024 quay về mức quanh 3%.
- So với “Dân số trung bình (Nghìn người)”
- + Kết quả:

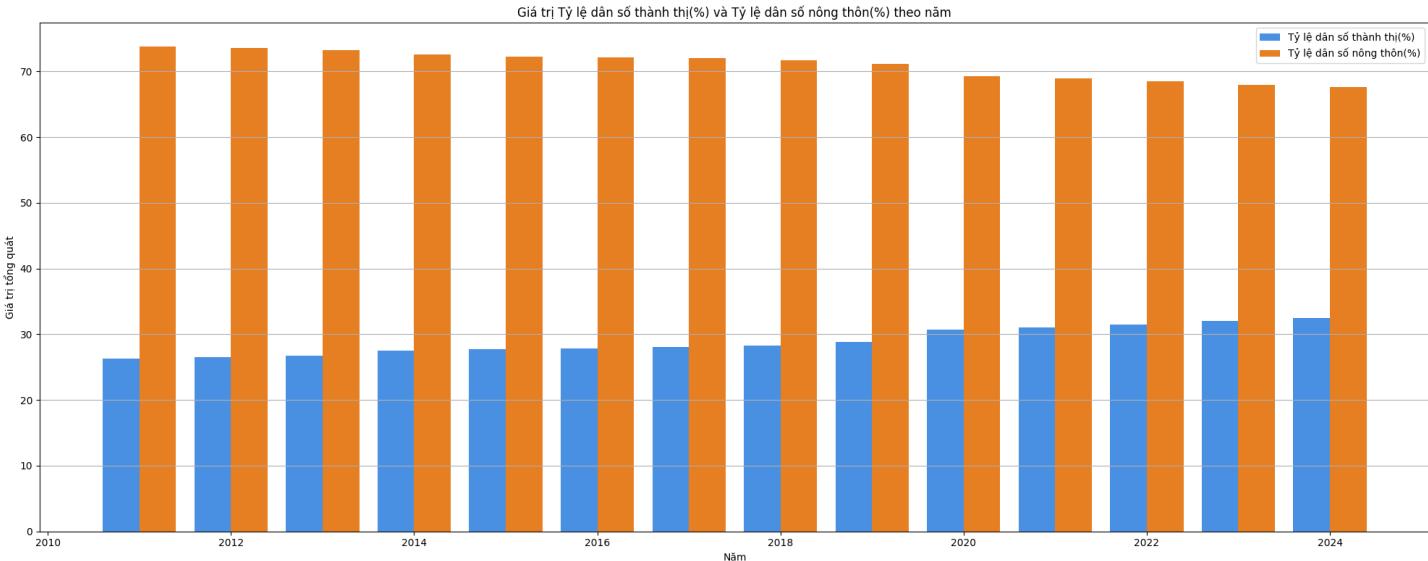


Hình ảnh 26: Biểu đồ cột dọc(2)

- + Đánh giá:
 - Xu hướng: tăng ổn định liên tục từ khoảng 1.40 triệu (2011) lên khoảng 1.60 triệu (2024).
 - Tốc độ dài hạn (ước lượng): Tốc độ tăng trưởng kép khoảng +1.1%/năm trong giai đoạn 2011-2024 (tăng đều, không có năm gãy mạnh).
 - Hambi ý: mặc dù trước đó bạn cho thấy tỷ suất sinh thô giảm, tổng dân vẫn tăng do đó khả năng đóng góp của nhập cư ròng và/hoặc đà quá độ dân số (quy mô nền lớn) là đáng kể.

5. Biểu đồ cột dọc kép:

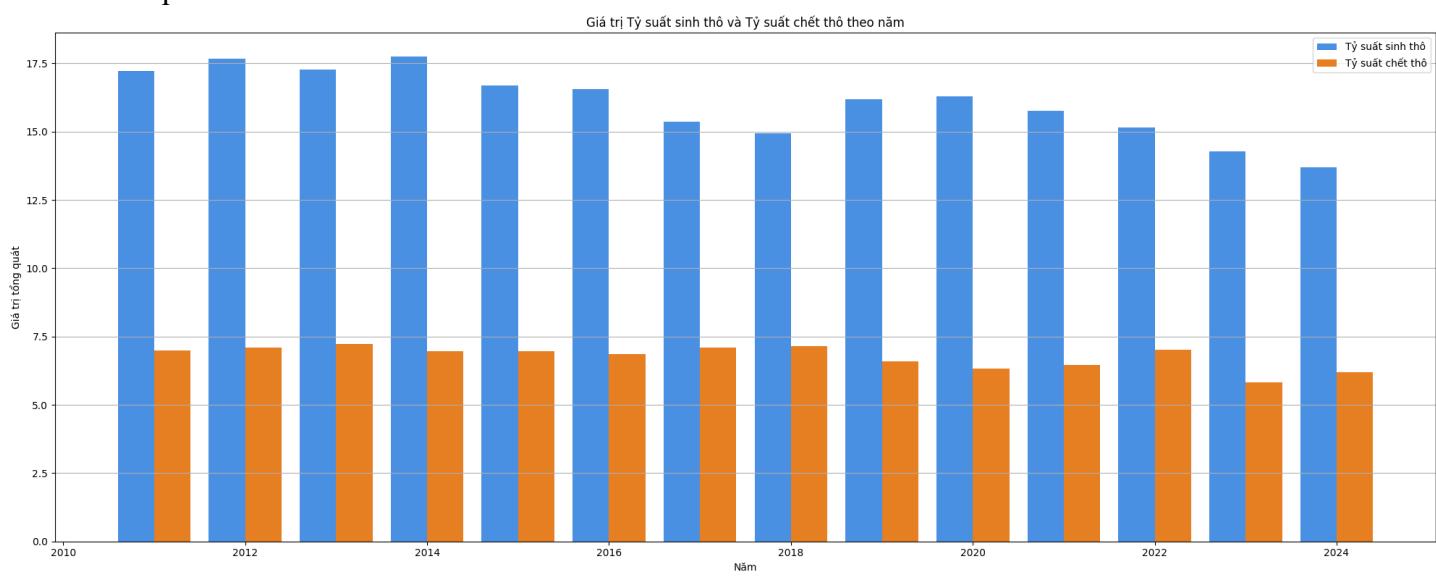
- So với “Tỷ lệ dân số thành thị(%)” và “Tỷ lệ dân số nông thôn(%)”
- + Kết quả:



Hình ảnh 27: Biểu đồ cột dọc kép(1)

- + Đánh giá:

- Thành thị tăng đều (khoảng 26% do đó khoảng 32–33% giai đoạn 2011–2024); nông thôn giảm đối xứng.
 - Nhip tăng đô thị hóa ổn định, không có cú sốc lớn (tăng khoảng **khoảng** 0.5 điểm %/năm).
 - Kết luận: đô thị hóa tiếp tục mở rộng, cơ cấu dân cư chuyển dịch về thành thị.
- So với “Tỷ suất sinh thô” và “Tỷ suất chết thô”
+ Kết quả:

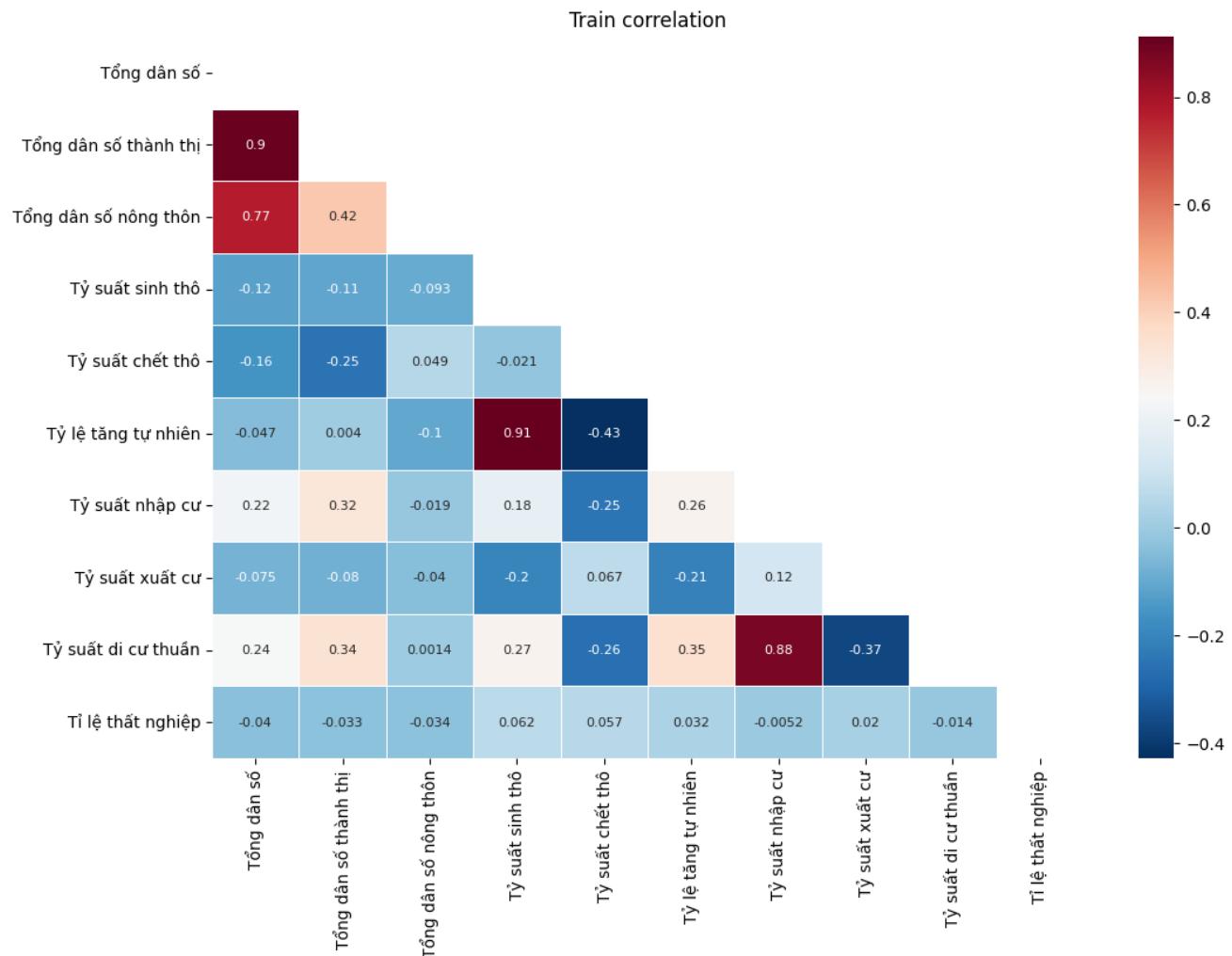


Hình ảnh 28: Biểu đồ cột dọc kép(2)

- + Đánh giá:
- Sinh thô giảm rõ (khoảng 17.5% do đó khoảng 13.6%).
 - Chết thô dao động hẹp quanh khoảng 6–7%.
 - Tăng tự nhiên (sinh–chết) thu hẹp từ khoảng 10–11% xuống khoảng 7–8%.
 - Kết luận: đà tăng dân số tự nhiên chậm lại; dân số tăng tương lai ngày càng phụ thuộc nhiều hơn vào di cư/đô thị hóa.

6. Biểu đồ ma trận tương quan:

- Kết quả:



Hình ảnh 28: Biểu đồ ma trận tương quan

- Đánh giá:

1. Quy mô dân số:

- Tổng dân số và Tổng dân số thành thị: 0.90 (rất mạnh, dương).
- Tổng dân số và Tổng dân số nông thôn: 0.77 (mạnh, dương).
- Thành thị và Nông thôn: 0.42 (trung bình yếu) do đó cơ cấu không dân số không đều.

2. Sinh–chết–tăng tự nhiên:

- Tỷ lệ tăng tự nhiên và Tỷ suất sinh thô: 0.91 (rất mạnh, dương).
- Tỷ lệ tăng tự nhiên và Tỷ suất chết thô: -0.43 (trung bình, âm).
do đó Đúng kỳ vọng: tăng tự nhiên chủ yếu do sinh tăng, chết giảm.

3. Di cư:

- Tỷ suất di cư thuần và Tỷ suất nhập cư: 0.88 (rất mạnh, dương).
- Tỷ suất di cư thuần và Tỷ suất xuất cư: -0.37 (âm, vừa).
do đó Di cư thuần tăng khi nhập cư tăng và/hoặc xuất cư giảm.

4. Quy mô ↔ di cư:

- Tổng dân số thành thị và Tỷ suất nhập cư: 0.32 (dương, nhẹ) do đó đô thị hút nhập cư.
- Liên hệ với Tổng dân số nhìn chung yếu ($|r| \leq$ khoảng 0.25).

5. Thất nghiệp:

- Tỉ lệ thất nghiệp có tương quan rất thấp với hầu hết biến ($|r| \approx 0-0.06$) do đó không đồng biến tuyến tính rõ với quy mô, sinh–chết hay di cư.

F. Mô hình hồi quy và dự đoán trong dữ liệu:

1. Phân tích hồi quy và mô hình XGBoost Regressor:

1.1 Tổng quan:

- Phân tích hồi quy (Regression Analysis) là một phương pháp thống kê và học máy được sử dụng để mô hình hóa mối quan hệ giữa một biến phụ thuộc và một hoặc nhiều biến độc lập. Mục tiêu của hồi quy là dự đoán giá trị của biến phụ thuộc dựa trên các đặc trưng đầu vào, đồng thời hiểu được mức độ ảnh hưởng của từng biến tới kết quả.
- Bối cảnh bài toán
 - Trong bài thực hành Machine Learning, mô hình hồi quy được áp dụng để dự đoán tổng dân số ('Tổng dân số') của từng vùng qua các năm. Dữ liệu đầu vào bao gồm nhiều đặc trưng liên quan đến nhân khẩu học như tỷ lệ sinh, di cư, mật độ dân số, tỷ lệ thành thị và nông thôn,... Bằng cách học từ dữ liệu giai đoạn 2011–2021, mô hình sẽ ước lượng quy luật biến động dân số và dự đoán cho các năm sau (2022–2024).
- Mô hình sử dụng
 - Mô hình được sử dụng là XGBoost Regressor (XGBRegressor), một thuật toán hồi quy phi tuyến dựa trên kỹ thuật Gradient Boosting. XGBoost kết hợp nhiều cây quyết định nhỏ, mỗi cây học từ phần sai số còn lại của cây trước đó, giúp giảm lỗi dự đoán và tăng độ chính xác. Pipeline xử lý dữ liệu (num_pipeline) được sử dụng để chuẩn hóa, xử lý giá trị thiếu và biến đổi dữ liệu trước khi đưa vào mô hình.
- Quy trình dự đoán
 - Dữ liệu được chia thành hai tập: tập huấn luyện (các năm \leq 2021) và tập kiểm thử (các năm $>$ 2021). Mô hình được huấn luyện trên tập train và dự đoán giá trị dân số của tập test. Kết quả được so sánh giữa giá trị thực tế (y_{test}) và giá trị dự đoán (y_{pred}), đồng thời đánh giá bằng các chỉ số như R^2 hoặc sai số trung bình (MSE).

1.2 Thư viện và vai trò:

- **numpy (np):** Tính toán số học mảng, vector, ma trận; kết hợp cột/hàng, thao tác số liệu nhanh.
- **pandas (pd):** Xử lý dữ liệu dạng bảng (DataFrame): đọc CSV, lọc theo cột, nối/ghép, xuất kết quả.
- **matplotlib.pyplot (plt):** Vẽ đồ thị đường, scatter, histogram để trực quan hóa kết quả dự đoán.
- **scikit-learn:** Chuẩn hóa/tiền xử lý (Pipeline, TransformerMixin), tách dữ liệu, đánh giá cross-validation.
- **xgboost.XGBRegressor:** Thuật toán hồi quy Gradient Boosting trên cây quyết định; mô hình chính để dự đoán biến liên tục.

1.3 Các lớp/hàm chính:

- ❖ Class **CombineAttributesAdder/StrEncoding** (tiền xử lý/biến đổi đặc trưng)
- Mục đích: Chuẩn hóa hoặc tạo thêm đặc trưng (feature engineering) để cải thiện khả năng học của mô hình.

```
class CombineAttributesAdder(BaseEstimator,TransformerMixin):  
  
    def fit(self,X,y=None):  
        X = X.copy()  
        return self
```

```

def transform(self,X):
    X = X.copy()
    X = X.sort_values(['Vùng','Năm'])
    X['prev_density'] = X.groupby('Vùng')['Mật độ dân số (Người/km2)'].shift(1)
    X['growth_density'] = X.groupby('Vùng')['Mật độ dân số (Người/km2)'].pct_change()
    return X.reset_index(drop=True)

class StrEncoding (BaseEstimator,TransformerMixin):
    def __init__(self):
        self.le_region = LabelEncoder()
        self.le_area = LabelEncoder()

    def fit(self,X,y=None):
        X = X.copy()
        self.le_region.fit(X['Vùng'])
        self.le_area.fit(X['Khu vực'])
        return self

    def transform(self,X):
        X = X.copy()
        X['Vùng'] = self.le_region.fit_transform(X['Vùng'])
        X['Khu vực'] = self.le_area.fit_transform(X['Khu vực'])
        return X

num_pipeline = Pipeline([
    ('Labelencode',StrEncoding()),
    ('attribs_adder',CombineAttributesAdder())
])

```

Đoạn mã 42: Class CombineAttributesAdder/StrEncoding

- Giải thích:
- + Mục tiêu pipeline: Mã hoá Vùng, Khu vực thành số; tạo đặc trưng theo thời gian cho mật độ dân số.
- + CombineAttributesAdder:
 - Sắp ['Vùng','Năm']; tạo:
 - prev_density: mật độ năm trước trong cùng vùng (groupby().shift(1)).
 - growth_density: % tăng mật độ theo năm trong cùng vùng (groupby().pct_change()).
 - Hàng đầu mỗi vùng → NaN. reset_index() thay đổi thứ tự gốc.
- + StrEncoding:
 - fit() học mapping LabelEncoder cho Vùng, Khu vực.
 - Lỗi nghiêm trọng: trong transform() dùng fit_transform dẫn đến tái học mapping, rò rỉ & không nhất quán. Do đó phải dùng transform: self.le_region.transform(...), self.le_area.transform(...).
 - LabelEncoder mang thứ bậc giá.
- + num_pipeline: sắp xếp thứ tự: Encode đến Add features. Đầu ra: dữ liệu đã mã hoá + prev_density, growth_density.
- ❖ Class **XGBModel** (mô hình hoá hồi quy với XGBoost)
- **Mục đích:** Xây mô hình dự báo “Tổng dân số” bằng XGBoost cho một vùng/tỉnh cụ thể (hoặc toàn bộ), với quy trình: tiền xử lý + tạo đặc trưng → tách train/test theo thời gian → huấn luyện → dự báo & lưu CSV → đánh giá CV.

```

class XGBModel:
    def __init__(self,region_name,features):
        self.region = region_name
        self.features = features
        self.XGBmodel = XGBRegressor()

```

```

        n_estimators = 500,
        learning_rate = 0.05,
        max_depth = 6,
        subsample = 0.8,
        colsample_bytree = 0.8,
        random_state = 42
    )

def get_data(self):
    if self.region == 'All': return num_pipeline.fit_transform(Data)
    return num_pipeline.fit_transform(Data[Data['Vùng'] == self.region])

def data_splits(self,data):
    train = data[data['Năm'] <= 2021]
    test = data[data['Năm'] > 2021]

    return train[self.features], train['Tổng dân số'], test[self.features], test['Tổng
dân số']

def train_XGBoost_model(self):
    df = self.get_data()
    X_train,y_train, X_test,y_test = self.data_splits(df)
    self.XGBmodel.fit(X_train,y_train)
    y_pred = self.XGBmodel.predict(X_test)

    return
pd.DataFrame(np.column_stack([y_pred,y_test]),columns=['y_pred','y_test']).to_csv(f'Result/
Test_set/XGBModel/{self.region}_predict_result.csv')

def CrossValScore(self,scoring):
    X = num_pipeline.fit_transform(Data)
    prepared = X[self.features]
    label = X['Tổng dân số']
    return cross_val_score(self.XGBmodel,prepared,label,scoring=scoring,cv=5)
features = ['Vùng','Khu vực','Năm','Tỷ lệ nam nữ', 'Nam', 'Nữ',
'Tổng dân số thành thị', 'Tổng dân số nông thôn', 'Diện tích(Km2)',
'Dân số trung bình (Nghìn người)', 'Mật độ dân số (Người/km2)',
'Tỷ suất sinh thô', 'Tỷ suất chết thô', 'Tỷ lệ tăng tự nhiên',
'Tỷ suất nhập cư', 'Tỷ suất xuất cư', 'Tỷ suất di cư thuận',
'Tỉ lệ thất nghiệp', 'Tỉ lệ thất nghiệp ở thành thị',
'Tỉ lệ thất nghiệp ở nông thôn','prev_density','growth_density']
PredictModel = XGBModel('Hà Nội',features)
PredictModel.train_XGBoost_model()
PredictModel.CrossValScore('r2')

```

Đoạn mã 43: Class XGBModel

- Giải thích:
- + `__init__(region_name, features)`: lưu tên vùng cần dự báo, danh sách đặc trưng và khởi tạo **XGBRegressor** (500 cây, lr=0.05, max_depth=6, subsample/colsample=0.8, random_state=42).
- + `get_data()`:
 - Gọi `num_pipeline.fit_transform(...)` để tiền xử lý dữ liệu (mã hoá Vùng, Khu vực, tạo prev_density, growth_density, sắp xếp, v.v.).
 - Nếu region='All' dùng toàn bộ Data; ngược lại lọc theo vùng trước khi xử lý.
- + `data_splits(data)`:
 - Chia theo thời gian: train = năm ≤ 2021, test = năm > 2021.

- Trả về: X_train, y_train, X_test, y_test với nhãn mục tiêu là 'Tổng dân số' và cột đầu vào là self.features.
- + *train_XGBoost_model()*:
 - Lấy dữ liệu đã chuẩn bị → tách train/test → fit mô hình trên train → predict trên test.
 - Ghép y_pred & y_test rồi lưu CSV:
Result/Test_set/XGBModel/{region}_predict_result.csv.
- + *CrossValScore(scoring)*:
 - Tiền xử lý toàn bộ Data qua num_pipeline, lấy prepared = X[self.features], label = X['Tổng dân số'].
 - Chạy cross-validation 5-fold với thước đo scoring (ví dụ 'r2'), trả về mảng điểm số.
- + features: tập hợp các biến nhân khẩu-xã hội (giới tính, đô thị/nông thôn, diện tích, mật độ, sinh-chết-tăng tự nhiên, di cư, thất nghiệp) + đặc trưng tạo thêm: prev_density, growth_density.

1.4 Mã nguồn hỗ trợ:

❖ Import thư viện

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.model_selection import cross_val_score
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
```

Đoạn mã 44:XGBModel (thư viện)

❖ Các cell mã nguồn khác

```
Data = pd.read_csv('Datasets/Clean_data/Locality_Analyst.csv')
```

Đoạn mã 45:XGBModel (đọc dữ liệu)

1.5 Phân tích kết quả dự đoán (R^2):

- Kết quả R^2 từ đánh giá chéo (5 giá trị): [0.95, 0.99, 0.99, 0.91, 0.99]

```
PredictModel = XGBModel('Hà Nội',features)
PredictModel.train_XGBoost_model()
PredictModel.CrossValScore('r2')
```

```
array([0.94732175, 0.9476873 , 0.99812334, 0.98676983, 0.99443233])
```

Hình ảnh 29: Kết quả dự đoán XGBModel

- + Trung bình $R^2 = 0.97$
- + Độ lệch chuẩn (mẫu) = 0.038
- + Min = 0.91 | Max = 0.99

- + Khoảng tin cậy 95% (xấp xỉ) cho trung bình R^2 : [0.92, 0.99]
- **Điễn giải:**
 - $R^2 \approx 0.95$ –0.99 cho thấy mô hình giải thích được khoảng 95%–99.9% phương sai của biến mục tiêu, chất lượng dự đoán rất cao.

1.6 Đánh giá mô hình:

- Hiệu năng rất cao (CV $R^2 \sim 0.95$ –0.998) \Rightarrow mô hình khớp dữ liệu mạnh và ổn định.
- Nguy cơ target leakage: dùng “Tổng dân số thành thị/nông thôn” (gần như cấu phần của mục tiêu “Tổng dân số”) nên điểm số bị “đẹp” bất thường.
- Rò rỉ trong pipeline: `LabelEncoder.fit_transform` trong `transform` và `fit` tiền xử lý trước CV có thể thổi phồng kết quả.
- **Kết luận:** model mạnh trên tập hiện tại, nhưng cần siết quy trình và đặc trưng để phản ánh năng lực dự báo thật.

2. Phân tích kết quả dự đoán bằng ARIMA

2.1 Tổng quan:

- Sau khi mô hình hồi quy được huấn luyện ở file 06_Machine_learning.ipynb, file này sử dụng mô hình để dự đoán dữ liệu mới, đánh giá kết quả và trực quan hóa.
- **Bối cảnh bài toán:**
 - + Trong bài thực hành Machine Learning, mô hình ARIMA (AutoRegressive Integrated Moving Average) được áp dụng để dự đoán tổng dân số (“Tổng dân số”) của từng vùng theo thời gian. Dữ liệu dân số được thu thập từ năm 2011 đến 2021, và mục tiêu là dự đoán xu hướng biến động dân số cho các năm 2022–2024.
 - + Phương pháp này giúp mô hình nắm bắt quy luật tăng trưởng, chu kỳ và độ trễ của dân số qua thời gian, phục vụ cho các bài toán dự báo dài hạn.
- **Mô hình sử dụng:**
 - + Mô hình ARIMA (AutoRegressive Integrated Moving Average) là một trong những phương pháp phổ biến nhất trong phân tích và dự báo chuỗi thời gian (time series forecasting). Khác với các mô hình hồi quy truyền thống dựa trên nhiều biến độc lập, ARIMA chỉ dựa vào chính các giá trị quá khứ của chuỗi dữ liệu để dự đoán giá trị tương lai.
- **Quy trình dự đoán:**
 - Tiên xử lý và kiểm định tính dừng \rightarrow Xác định tham số (p, d, q) \rightarrow Huấn luyện mô hình \rightarrow Dự đoán và đánh giá

2.2 Thư viện và vai trò:

- **numpy (np):** Xử lý mảng, vector, tính toán số học.
- **pandas (pd):** Xử lý dữ liệu dạng bảng, đọc và ghi file CSV.
- **matplotlib.pyplot (plt):** Vẽ biểu đồ trực quan kết quả dự đoán.
- **seaborn (sns):** Trực quan hóa dữ liệu nâng cao, thể hiện tương quan và sai số.
- **xgboost.XGBRegressor:** Thuật toán hồi quy mạnh mẽ dựa trên Gradient Boosting.
- **sklearn.metrics:** Tính các chỉ số đánh giá mô hình như R^2 , MAE, MSE,...

2.3 Các hàm và class trong file:

❖ Class: ARIMAModel:

- Xây dựng mô hình chuỗi thời gian ARIMA để dự báo “Tổng dân số” theo năm cho một vùng cụ thể hoặc toàn bộ, sau đó vẽ đồ thị dự báo và xuất file CSV kết quả.

```

class ARIMAModel:
    def __init__(self,region_name):
        self.region = region_name

    def get_data(self):
        if self.region == 'All': return Data[['Năm','Tổng dân số']]
        return Data[Data['Vùng'] == self.region][['Năm','Tổng dân số']]

    def train_ARIMA_model(self,data_end_year,predict_year):
        Series = self.get_data()
        Series = Series.set_index('Năm')
        model = ARIMA(Series,order=(1,1,1))
        model_fit = model.fit()
        forecast = model_fit.forecast(steps=predict_year-data_end_year)
        forecast.index = range(Series.index.max() + 1, Series.index.max() + len(forecast) + 1)
        self.Draw_predict(data_end_year,predict_year,Series,forecast)
        return

    pd.DataFrame(forecast.values,columns=['y_pred']).to_csv(f'Result/Test_set/ARIMAModel/{self.region}_predict_result.csv')

    def Draw_predict(self,year,end_year,Series,Forecast):
        plt.figure(figsize=(8,4))
        plt.plot(Series.index,Series['Tổng dân số'],label='Thực tế',marker='o',linestyle = '-')
        plt.plot(Forecast.index,Forecast,label='Dự báo',marker='o',linestyle='--',color = 'red')
        plt.title(f'Dự báo dân số {self.region} từ năm {year} - {end_year}')
        plt.xticks(range(Series.index.min(),Forecast.index.max(),3))
        plt.legend()
        plt.savefig(f'Result/Test_set/ARIMAModel/Predict_plot/Dự báo dân số {self.region} từ năm {year} - {end_year}')
        plt.close()

```

Đoạn mã 46:Class ARIMAModel

- Giải thích:
- + `__init__(region_name)`: lưu tên vùng/tỉnh cần dự báo.
- + `get_data()`:
 - Nếu All → lấy cột ['Năm','Tổng dân số'] toàn bộ.
 - Nếu chỉ định vùng → lọc theo Vùng, giữ ['Năm','Tổng dân số'].
- + `train_ARIMA_model(data_end_year, predict_year)`:
 - Lấy dữ liệu, đặt năm làm index.
 - Khởi tạo **ARIMA(1,1,1)**:
 - (p=1) thành phần tự hồi quy bậc 1,
 - (d=1) sai phân bậc 1 (khử xu thế để đạt gần đúng),
 - (q=1) trung bình trượt bậc 1.
 - fit() mô hình trên toàn bộ chuỗi hiện có.
 - forecast(steps = predict_year - data_end_year) → dự báo số năm tương lai.
 - Gán index cho đoạn dự báo nối tiếp sau năm cuối cùng của chuỗi.
 - Gọi `Draw_predict(...)` để vẽ & lưu biểu đồ (thực tế vs dự báo).
 - Xuất CSV Result/Test_set/ARIMAModel/{region}_predict_result.csv với cột y_pred.
- + `Draw_predict(year, end_year, Series, Forecast)`:
 - Vẽ đường thực tế (liền) và dự báo (đứt, màu đỏ),
 - Đặt tiêu đề “Dự báo dân số {region} từ {year}-{end_year}”,
 - Lưu hình vào thư mục Predict_plot.

2.4 Các cell mã nguồn khác:

- **Phần import thư viện:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
```

Đoạn mã 47: ARIMAModel (import thư viện)

- **Đọc file dữ liệu:**

```
Data = pd.read_csv('Datasets/Clean_data/Locality_Analyst.csv')
```

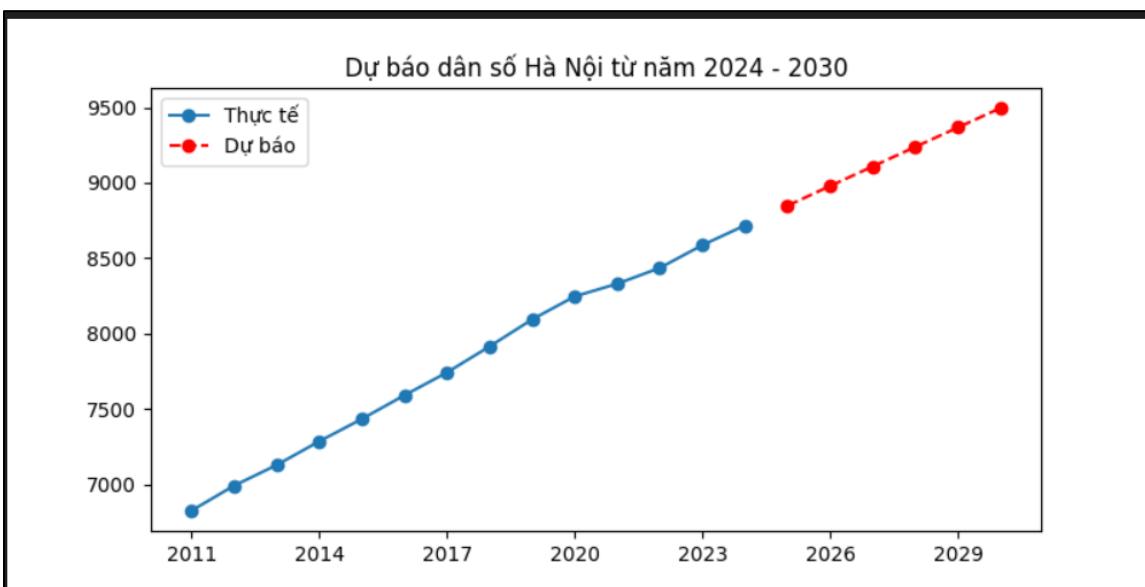
Đoạn mã 48: ARIMAModel (đọc dữ liệu)

- **Gọi class/function training và dự đoán dữ liệu:**

```
ARIMAModel = ARIMAModel('Hà Nội')
ARIMAModel.train_ARIMA_model(data_end_year=2024,predict_year=2030)
```

Đoạn mã 49: ARIMAModel (Gọi mô hình dự đoán)

2.5 Kết quả dự đoán:



Hình ảnh 30: Kết quả mô hình dữ đoán ARIMA

- ❖ Đánh giá nhanh mô hình/kết quả ARIMA(1,1,1) với thủ đô Hà Nội từ năm 2025 đến 2030:

- **Hình dạng dự báo:** đường đồ tăng gần như là tuyến tính, bám sát xu thế quá khứ → đúng kiểu của ARIMA(d=1): dự báo bằng xu hướng trung bình gần đây.
- **Tính hợp lý ngắn-trung hạn:** dân số Hà Nội tăng đều nhiều năm gần đây, dự báo này hợp lý 3–5 năm đầu.
- **Rủi ro/giới hạn:**
 - + Không có khoảng tin cậy → khó đánh giá độ chắc chắn; thực tế sai số sẽ mở rộng theo năm.
 - + Mô hình chỉ nội sinh (không dùng sinh-chết-di cư, chính sách...) ⇒ khó phản ứng cú sốc (di cư ròng, điều chỉnh địa giới, biến động kinh tế).
 - + Dùng một cấu hình ARIMA(1,1,1) cố định; chưa chứng minh tối ưu (so AIC/BIC, kiểm định dùng).

- + Fit toàn bộ quá khứ rồi dự báo tương lai, chưa thấy backtest cuộn (rolling) nên độ tin cậy ngoài mău chưa được định lượng (MAPE/RMSE).
- **Kết luận:** Dự báo hợp lý và mượt cho xu hướng tổng quát, nhưng có thể lạc quan/quá mượt ở giai đoạn 2025–2030