

ADS507_Team2_Pipeline_Final_v2

February 27, 2023

1 Appendix A

1.1 Global setup

```
[1]: '''Setup Kaggle API citation:
      https://www.kaggle.com/docs/api'''

      # Load libraries
      import numpy as np
      import pandas as pd
      import pymysql as mysql
      import matplotlib.pyplot as plt
      import os
      import shutil
      import re
      import logging
      import time
      import kaggle
      import zipfile

      # Set pandas global options
      pd.options.display.max_rows = 17
```

1.2 Connect to Kaggle API

```
[2]: # Split up-1-level path & current working folder
      up1_path, curr_folder = os.path.split(os.getcwd())
      print(up1_path)
      print(curr_folder)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering
deliverables

1.2.1 Setup up connection

```
[3]: '''Setup Kaggle API connection citation: https://python.plainenglish.io/
      ↪how-to-use-the-kaggle-api-in-python-4d4c812c39c7'''
      # Create Kaggle API authentication instance
      from kaggle.api.kaggle_api_extended import KaggleApi
      api = KaggleApi()
      api.authenticate()
```

Connect to database: kaggle datasets download -d thedevastator/global-fossil-co2-emissions-by-country-2002-2022

```
[4]: # Assignment data placement folder
      emi_place_folder = 'data\Emissions'

      # Join up-1-level path to placement folder
      emi_place_folder_path = os.path.join(up1_path, emi_place_folder)
      print(emi_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Emissions

```
[5]: # Assign Kaggle API link details
      emi_kag_owner = 'thedevastator'
      emi_kag_dataset = 'global-fossil-co2-emissions-by-country-2002-2022'
      emi_kag_api_link = emi_kag_owner + '/' + emi_kag_dataset
```

```
[6]: # Access Kaggle API and download file(s)
      api.dataset_download_files(emi_kag_api_link, path=emi_place_folder_path)
```

```
[7]: # Unzip downloaded file
      emi_zip_file = emi_kag_dataset + '.zip'
      emi_zip_file_path = os.path.join(emi_place_folder_path, emi_zip_file)
      print(emi_zip_file_path)

      with zipfile.ZipFile(emi_zip_file_path, 'r') as zipref:
          zipref.extractall(emi_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Emissions\global-fossil-co2-emissions-by-country-2002-2022.zip

Connect to database: kaggle datasets download -d sevgisarak/temperature-change

```
[8]: # Assignment data placement folder
      tmp_place_folder = 'data\Temperature'

      # Join up-1-level path to placement folder
      tmp_place_folder_path = os.path.join(up1_path, tmp_place_folder)
      print(tmp_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Temperature

```
[9]: # Assign Kaggle API link details
tmp_kag_owner = 'sevgisarak'
tmp_kag_dataset = 'temperature-change'
tmp_kag_api_link = tmp_kag_owner + '/' + tmp_kag_dataset
```

```
[10]: # Access Kaggle API and download file(s)
api.dataset_download_files(tmp_kag_api_link, path=tmp_place_folder_path)
```

```
[11]: # Unzip downloaded file
tmp_zip_file = tmp_kag_dataset + '.zip'
tmp_zip_file_path = os.path.join(tmp_place_folder_path, tmp_zip_file)
print(tmp_zip_file_path)

with zipfile.ZipFile(tmp_zip_file_path, 'r') as zipref:
    zipref.extractall(tmp_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Temperature\temperature-change.zip

Connect to database: kaggle datasets download -d iamsouravbanerjee/world-population-dataset

```
[12]: # Assignment data placement folder
pop_place_folder = 'data\Population'

# Join up-1-level path to placement folder
pop_place_folder_path = os.path.join(up1_path, pop_place_folder)
print(pop_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Population

```
[13]: # Assign Kaggle API link details
pop_kag_owner = 'iamsouravbanerjee'
pop_kag_dataset = 'world-population-dataset'
pop_kag_api_link = pop_kag_owner + '/' + pop_kag_dataset
```

```
[14]: # Access Kaggle API and download file(s)
api.dataset_download_files(pop_kag_api_link, path=pop_place_folder_path)
```

```
[15]: # Unzip downloaded file
pop_zip_file = pop_kag_dataset + '.zip'
pop_zip_file_path = os.path.join(pop_place_folder_path, pop_zip_file)
print(pop_zip_file_path)

with zipfile.ZipFile(pop_zip_file_path, 'r') as zipref:
    zipref.extractall(pop_place_folder_path)
```

C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Population\world-population-dataset.zip

1.3 Load data into MySQL tables from CSV files

1.3.1 Get credentials from local path and connect to MySQL DB

```
[16]: '''Set local environment variables to hide user name & password citation:
https://www.geeksforgeeks.org/how-to-hide-sensitive-credentials-using-python/'''

user_name = os.environ['MySQLUSRAC']
user_pass = os.environ['MySQLPWDAC']

# Instantiate connection
db_conn = mysql.connect(host='localhost',
                        port=int(3306),
                        user=user_name,
                        passwd=user_pass,
                        db='507_final_proj')

# Create a cursor object
cursor = db_conn.cursor()
```

```
[17]: tbl_names = pd.read_sql('SHOW TABLES', db_conn)

display(tbl_names)
print(type(tbl_names))
```

```
Tables_in_507_final_proj
0          country_map
1      emissions_gross
2      emissions_tempy
3          etp_view
4              iso
5          iso_tempy
6      population
7      population_tempy
8      population_trans
9          temp_core1
10         temperature
11      temperature_tempy

<class 'pandas.core.frame.DataFrame'>
```

1.3.2 Setup log parameters

```
[18]: '''Logging citations (see additional code in following code blocks:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/howto/logging.html#logging-basic-example;
https://docs.python.org/3/howto/logging.html#logging-to-a-file;
https://docs.python.org/3/howto/logging-cookbook.html#using-a-rotating-log-file-handler;
```

```
https://docs.python.org/3/howto/logging-cookbook.
↳html#using-a-timed-rotating-file-handler'''
```

```
# Set up logging
logging.basicConfig(level=logging.INFO,
                    filename='pymysql.log',
                    filemode='a',
                    format='>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<\n%(asctime)s -\n
↳%(levelname)s - %(message)s')
```

```
[19]: '''Move files to from unpacked folder to MySQL unrestricted folder citation:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/library/os.html;
https://docs.python.org/3/library/shutil.html'''
```

```
import os

# Set the file paths
source_files = ['C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
↳Emissions/GCB2022v27_MtCO2_flat.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
↳Population/world_population.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
↳Temperature/FAOSTAT_data_1-10-2022.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
↳Temperature/FAOSTAT_data_11-24-2020.csv']
destination_folder = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads'

# Move the files to the destination folder and overwrite them if they already
↳exist
for source_file in source_files:
    file_name = os.path.basename(source_file)
    destination_file = os.path.join(destination_folder, file_name)
    os.replace(source_file, destination_file)
```

1.3.3 Update individual tables

Update iso table from CSV

```
[20]: '''Using cursor and loading into temp file:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://pynative.com/python-mysql-insert-data-into-database-table/'''
```

```
# Execute query and measure execution time
start_time = time.time()
```

```
# Wipe temp table
try:
```



```

# Insert new records into main table
try:
    ism_load_stmtn = """
    INSERT INTO iso
    (
    country_code,
    country,
    m49_code,
    iso2,
    iso3,
    year_start,
    year_end
    )
    SELECT
        tp.country_code,
        tp.country,
        tp.m49_code,
        tp.iso2,
        tp.iso3,
        tp.year_start,
        tp.year_end
    FROM iso_tempy AS tp
    LEFT JOIN iso AS mn
        ON tp.iso3 = mn.iso3
    WHERE mn.iso3 IS NULL
    """

    cursor.execute(ism_load_stmtn)
    logging.info(f'Successfully executed query:\n{ism_load_stmtn}\n\nRecords_
↳scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ism_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}_
↳seconds\n>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    cursor.execute(ist_dlt_tble_stmtn)
    logging.info(f'Successfully executed query:
↳\n{ist_dlt_tble_stmtn}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ist_dlt_tble_stmtn}\n\n{e}')
finally:

```

```

        end_time = time.time()
        logging.info(f'Time taken: {end_time - start_time:.3f}␣
↪seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<\n\n')

```

Update emissions_gross table from CSV

```

[21]: # Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    egt_dlt_tble_stmnt = """DELETE FROM emissions_tempy"""
    cursor.execute(egt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
↪\n{egt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
↪seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    egt_csv_load_stmnt = """
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
↪GCB2022v27_MtCO2_flat.csv'
    INTO TABLE emissions_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    IGNORE 1 ROWS
    (
    country,
    iso3,
    record_year,
    total,
    coal,
    oil,
    gas,
    cement,
    flaring,
    other,
    per_capita

```



```

WHERE mn.iso3 IS NULL AND mn.record_year IS NULL
"""
cursor.execute(egm_load_stmtn)
logging.info(f'Successfully executed query:\n{egm_load_stmtn}\n\nRecords_
↳scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}_
↳seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    egt_dlt_tble_stmtnt = """DELETE FROM emissions_tempy"""
    cursor.execute(egt_dlt_tble_stmtnt)
    logging.info(f'Successfully executed query:
↳\n{egt_dlt_tble_stmtnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egt_dlt_tble_stmtnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}_
↳seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')

```

Update population table from CSV

```

[22]: # Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    ppt_dlt_tble_stmtnt = """DELETE FROM population_tempy"""
    cursor.execute(ppt_dlt_tble_stmtnt)
    logging.info(f'Successfully executed query:
↳\n{ppt_dlt_tble_stmtnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppt_dlt_tble_stmtnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}_
↳seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time

```



```

ppm_load_stmtn = """
INSERT INTO population
(
pop_rank,
iso3,
country,
capital,
continent,
pop_2022,
pop_2020,
pop_2015,
pop_2010,
pop_2000,
pop_1990,
pop_1980,
pop_1970,
area,
density,
grow_rate,
pop_perc
)
SELECT
    tp.pop_rank,
    tp.iso3,
    tp.country,
    tp.capital,
    tp.continent,
    tp.pop_2022,
    tp.pop_2020,
    tp.pop_2015,
    tp.pop_2010,
    tp.pop_2000,
    tp.pop_1990,
    tp.pop_1980,
    tp.pop_1970,
    tp.area,
    tp.density,
    tp.grow_rate,
    tp.pop_perc
FROM population_tempy AS tp
LEFT JOIN population AS mn
    ON tp.iso3 = mn.iso3
WHERE mn.iso3 IS NULL
"""

cursor.execute(ppm_load_stmtn)
logging.info(f'Successfully executed query:\n{ppm_load_stmtn}\n\nRecords_
↳scanned: {cursor.rowcount}')

```

```
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppm_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}┐
↳seconds\n>>>>>>>>>>>><<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    ppt_dlt_tble_stmtnt = """DELETE FROM population_tempy"""
    cursor.execute(ppt_dlt_tble_stmtnt)
    logging.info(f'Successfully executed query:
↳\n{ppt_dlt_tble_stmtnt}\n\nRecords scanned: {cursor.rowcount}')
```

Update temperature table from CSV

```
[23]: '''Remove first row of CSV file:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/library/csv.html'''

import csv

# Open input and output files
input_file = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
↳FAOSTAT_data_1-10-2022.csv'
output_file = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
↳FAOSTAT_data_1-10-2022_new.csv'

with open(input_file, 'r') as csv_input_file, open(output_file, 'w',
↳newline='') as csv_output_file:
    # Create CSV reader and writer objects
    csv_reader = csv.reader(csv_input_file)
    csv_writer = csv.writer(csv_output_file)

    # Skip the first row of the input file
    next(csv_reader)

    # Write the remaining rows to the output file
```

```

for row in csv_reader:
    csv_writer.writerow(row)

```

```

[24]: # Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    tpt_dlt_tble_stmnt = """DELETE FROM temperature_tempy"""
    cursor.execute(tpt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
↳\n{tpt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}
↳seconds\n>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    tpt_csv_load_stmnt = """
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
↳FAOSTAT_data_1-10-2022_new.csv'
    INTO TABLE temperature_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    (
    domain_code,
    domain,
    area_code,
    country,
    element_code,
    element,
    month_code,
    month_name,
    year_code,
    record_year,
    unit,
    temp,
    flag,
    flag_desc
    )

```



```

        'pop_2015',
        'pop_2010',
        'pop_2000',
        'pop_1990',
        'pop_1980',
        'pop_1970']
var_names = [re.sub(r'^pop_', '', col) for col in cols_to_melt]
ppm_slct_all_df_melted = pd.melt(ppm_slct_all_df, id_vars=['pop_rank',
                                                         'iso3',
                                                         'country',
                                                         'capital',
                                                         'continent',
                                                         'area',
                                                         'density',
                                                         'grow_rate',
                                                         'pop_perc'],
                                value_vars=cols_to_melt,
                                var_name='year',
                                value_name='population')
#print(ppm_slct_all_df_melted.head())

# Insert the melted data into the MySQL table
insert_query = """
INSERT INTO population_trans (pop_rank, iso3, country, capital, continent,
    ↪area, density, grow_rate, pop_perc, year, population)
SELECT %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s FROM DUAL WHERE NOT EXISTS
    ↪(SELECT * FROM population_trans WHERE country = %s AND year = %s)
"""
for index, row in ppm_slct_all_df_melted.iterrows():
    variable = var_names[cols_to_melt.index(row['year'])] # get variable name
    ↪based on column name
    #print(index)
    #print(row)
    #print(variable)
    values = (row['pop_rank'],
              row['iso3'],
              row['country'],
              row['capital'],
              row['continent'],
              row['area'],
              row['density'],
              row['grow_rate'],
              row['pop_perc'],
              variable,
              row['population'],
              row['country'],
              variable)

```

```
#print(values)
cursor.execute(insert_query, values)
```

Standardize feature values in emissions_gross table based on mapping to iso

```
[26]: '''Update table col vals based on mapping to another table citation:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://pynative.com/python-mysql-insert-data-into-database-table/'''

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    egm_updt_country_stmt = """
    UPDATE emissions_gross AS t1
    INNER JOIN iso AS t2
        ON t1.iso3 = t2.iso3
    SET t1.country = t2.country
    WHERE t1.country <> t2.country AND t1.iso3 <> ''
    """

    cursor.execute(egm_updt_country_stmt)
    logging.info(f'Successfully executed query:
    ↵\n{egm_updt_country_stmt}\n\nRecords scanned: {cursor.rowcount}')
```

```
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_updt_country_stmt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
    ↵seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<\n\n')
```

```
# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    egm_add_cc_stmt = """
    UPDATE emissions_gross AS t1
    INNER JOIN iso AS t2
        ON t1.country = t2.country
    SET t1.country_code = t2.country_code
    """

    cursor.execute(egm_add_cc_stmt)
    logging.info(f'Successfully executed query:\n{egm_add_cc_stmt}\n\nRecords␣
    ↵scanned: {cursor.rowcount}')
```

```
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_add_cc_stmt}\n\n{e}')
```

```

finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
↪seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<\n\n')

```

Standardize feature values in population_trans table based on mapping to iso

```

[27]: # Execute query and measure execution time
start_time = time.time()

# Update table
try:
    ptm_updt_country_stmt = """
    UPDATE population_trans AS t1
    INNER JOIN iso AS t2
        ON t1.iso3 = t2.iso3
    SET t1.country = t2.country
    WHERE t1.country <> t2.country AND t1.iso3 <> ''
    """
    cursor.execute(ptm_updt_country_stmt)
    logging.info(f'Successfully executed query:
↪\n{ptm_updt_country_stmt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ptm_updt_country_stmt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
↪seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    ptm_add_cc_stmt = """
    UPDATE population_trans AS t1
    INNER JOIN iso AS t2
        ON t1.country = t2.country
    SET t1.country_code = t2.country_code
    """
    cursor.execute(ptm_add_cc_stmt)
    logging.info(f'Successfully executed query:\n{ptm_add_cc_stmt}\n\nRecords␣
↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ptm_add_cc_stmt}\n\n{e}')
finally:
    end_time = time.time()

```



```

end_time = time.time()
logging.info(f'Time taken: {end_time - start_time:.3f}␣
↳seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<\n\n')

```

```

[29]: # Extract data from each table - emissions
start_time = time.time()

try:
    emissions_query = "SELECT * FROM emissions_gross;"
    emissions_df = pd.read_sql(emissions_query, db_conn)

    display(emissions_df)
    logging.info(f'Successfully executed query:\n{emissions_query}\n\nRecords␣
↳scanned: {len(emissions_df)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{emissions_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
↳seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<\n\n')

```

	eg_id	country	iso3	record_year	total	coal \
0	1	Afghanistan	AFG	1750	0	
1	2	Afghanistan	AFG	1751	0	
2	3	Afghanistan	AFG	1752	0	
3	4	Afghanistan	AFG	1753	0	
4	5	Afghanistan	AFG	1754	0	
...	
63099	63100	Global	WLD	2017	36096.739276	14506.973805
63100	63101	Global	WLD	2018	36826.506600	14746.830688
63101	63102	Global	WLD	2019	37082.558969	14725.978025
63102	63103	Global	WLD	2020	35264.085734	14174.564010
63103	63104	Global	WLD	2021	37123.850352	14979.598083

	oil	gas	cement	flaring	other \
0					
1					
2					
3					
4					
...
63099	12242.627935	7144.928128	1507.923185	391.992176	302.294047
63100	12266.016285	7529.846784	1569.218392	412.115746	302.478706
63101	12345.653374	7647.528220	1617.506786	439.253991	306.638573
63102	11191.808551	7556.290283	1637.537532	407.583673	296.301685
63103	11837.159116	7921.829472	1672.592372	416.525563	296.145746

per_capita	country_code
------------	--------------

```
[63104 rows x 13 columns]
```

	domain_code		domain	area_code	country	element_code	\
0	ET	Temperature	change	2	Afghanistan	7271	
1	ET	Temperature	change	2	Afghanistan	7271	
2	ET	Temperature	change	2	Afghanistan	7271	
3	ET	Temperature	change	2	Afghanistan	7271	
4	ET	Temperature	change	2	Afghanistan	7271	
...	
244200	ET	Temperature	change	182	Réunion	7271	
244201	ET	Temperature	change	182	Réunion	7271	
244202	ET	Temperature	change	182	Réunion	7271	
244203	ET	Temperature	change	182	Réunion	7271	
244204	ET	Temperature	change	182	Réunion	7271	

22

3	Temperature change	7001	January	1964
4	Temperature change	7001	January	1965
...
244200	Temperature change	7003	March	2005
244201	Temperature change	7008	August	1971
244202	Temperature change	7008	August	1999
244203	Temperature change	7011	November	1975
244204	Temperature change	7020	Meteorological year	2008

	record_year	unit	temp	flag	flag_desc	country_code
0	1961	?C	0.746	Fc	Calculated data	2
1	1962	?C	0.009	Fc	Calculated data	2
2	1963	?C	2.695	Fc	Calculated data	2
3	1964	?C	-5.277	Fc	Calculated data	2
4	1965	?C	1.827	Fc	Calculated data	2
...
244200	2005	?C	1.401	Fc	Calculated data	182
244201	1971	?C	-0.504	Fc	Calculated data	182
244202	1999	?C	1.11	Fc	Calculated data	182
244203	1975	?C	0.706	Fc	Calculated data	182
244204	2008	?C	0.652	Fc	Calculated data	182

[244205 rows x 15 columns]

```
[31]: # Extract data from each table - population_trans
start_time = time.time()

try:
    worldpop_query = "SELECT * FROM population_trans;"
    worldpop_df = pd.read_sql(worldpop_query, db_conn)

    display(worldpop_df)
    logging.info(f'Successfully executed query:\n{worldpop_query}\n\nRecords_
↳scanned: {len(worldpop_df)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{worldpop_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}_
↳seconds\n>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<\n\n')
```

	pop_trans_id	pop_rank	iso3	country \
0	1	36	AFG	Afghanistan
1	2	138	ALB	Albania
2	3	34	DZA	Algeria
3	4	213	ASM	American Samoa
4	5	203	AND	Andorra
...

3547	3548	3	USA	United States of America
3548	3549	234	VAT	Holy See
3549	3550	51	VEN	Venezuela (Bolivarian Republic of)
3550	3551	16	VNM	Viet Nam
3551	3552	226	WLF	Wallis and Futuna Islands

	capital	continent	area	density	grow_rate	pop_perc	\
0	Kabul	Asia	652230	63.0587	1.0257	0.52	
1	Tirana	Europe	28748	98.8702	0.9957	0.04	
2	Algiers	Africa	2381741	18.8531	1.0164	0.56	
3	Pago Pago	Oceania	199	222.4774	0.9831	0	
4	Andorra la Vella	Europe	468	170.5641	1.01	0	
...	
3547	Washington, D.C.	North America	9372610	36.0935	1.0038	4.24	
3548	Vatican City	Europe	1	510	0.998	0	
3549	Caracas	South America	916445	30.882	1.0036	0.35	
3550	Hanoi	Asia	331212	296.4472	1.0074	1.23	
3551	Mata-Utu	Oceania	142	81.493	0.9953	0	

	year	population	country_code
0	2022	41128771	2
1	2022	2842321	3
2	2022	44903225	4
3	2022	44273	5
4	2022	79824	6
...
3547	1970	200328340	231
3548	1970	752	94
3549	1970	11355475	236
3550	1970	41928849	237
3551	1970	9377	243

[3552 rows x 13 columns]

```
[32]: # Transformation step - this is for country, year, total, temperature, and
      ↪ population only

      # Extract data from each table - emissions
      start_time = time.time()

      try:
          t1_stmnt = """
          SELECT
              e.Country,
              e.Record_year,
              e.Total,
              t.temp AS Temperature,
```



```
p.population AS Population
FROM emissions_gross e
JOIN temperature t
    ON e.country = t.country AND e.record_year = t.record_year
JOIN population_trans p
    ON e.country = p.country
ORDER BY e.country, e.record_year"""
Transform1 = pd.read_sql(t1_stmtnt, db_conn)
display(Transform1)
logging.info(f'Successfully executed query:\n{t1_stmtnt}\n\nRecords scanned: {len(Transform1)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{t1_stmtnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f} seconds\n>>>>>>>>>><<<<<<<<<<<\n\n')
```

```
[3432232 rows x 5 columns]
```

```
[33]: # General transformation to include as much raw data to show relationships
      ↪ between all datasets for emissions, temperature, and population.
```



```
[34]: # Main Transformation - depicts emission factors, temperature, and population
      ↪factors

# Extract data from each table - emissions
start_time = time.time()

try:
    t3_stmtnt = """
        SELECT
            e.country,
            e.record_year,
            e.total,
            e.coal,
            e.oil,
            e.gas,
            e.cement,
            e.flaring,
            e.other,
            e.per_capita,
            t.temp AS Temperature,
            p.density,
            p.grow_rate,
            p.pop_perc
        FROM emissions_gross e
        JOIN temperature t
            ON e.country = t.country AND e.record_year = t.record_year
        JOIN population_trans p
            ON e.country = p.country
        """

    Transform3 = pd.read_sql(t3_stmtnt, db_conn)
    display(Transform3)
    logging.info(f'Successfully executed query:\n{t3_stmtnt}\n\nRecords scanned:
    ↪{len(Transform3)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{t3_stmtnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}
    ↪seconds\n>>>>>>>>>>>>>><<<<<<<<<<<<<<\n\n')
```

3432228	Viet Nam	1997	44.516863	21.527760	18.398448	1.139597
3432229	Viet Nam	1997	44.516863	21.527760	18.398448	1.139597
3432230	Viet Nam	1997	44.516863	21.527760	18.398448	1.139597
3432231	Viet Nam	1997	44.516863	21.527760	18.398448	1.139597

	cement	flaring	other	per_capita	Temperature	density	grow_rate	\
0	0.021806	0		0.055835	0.746	63.0587	1.0257	
1	0.021806	0		0.055835	0.746	63.0587	1.0257	
2	0.021806	0		0.055835	0.746	63.0587	1.0257	
3	0.021806	0		0.055835	0.746	63.0587	1.0257	
4	0.021806	0		0.055835	0.746	63.0587	1.0257	
...	
3432227	3.451058	0		0.585297	0.303	296.4472	1.0074	
3432228	3.451058	0		0.585297	0.303	296.4472	1.0074	
3432229	3.451058	0		0.585297	0.303	296.4472	1.0074	
3432230	3.451058	0		0.585297	0.303	296.4472	1.0074	
3432231	3.451058	0		0.585297	0.303	296.4472	1.0074	

	pop_perc
0	0.52
1	0.52
2	0.52
3	0.52
4	0.52
...	...
3432227	1.23
3432228	1.23
3432229	1.23
3432230	1.23
3432231	1.23

[3432232 rows x 14 columns]

[35]: *# Create a view for data security purposes and hide complexity of queries*

```
view_drp_stmt = """DROP VIEW IF EXISTS etp_view"""
cursor.execute(view_drp_stmt)

# Create a cursor object
cursor = db_conn.cursor()

# Execute query and measure execution time
start_time = time.time()

# Execute the CREATE VIEW query
try:
    create_view_query = """
```


[11765 rows x 4 columns]

1.3.5 Commit changes and close cursor and connection instances

```
[37]: # Commit the changes to the database  
db_conn.commit()  
  
# Close the cursor and database connection  
cursor.close()  
db_conn.close()
```