Transform

NDBI046: Practical class 3

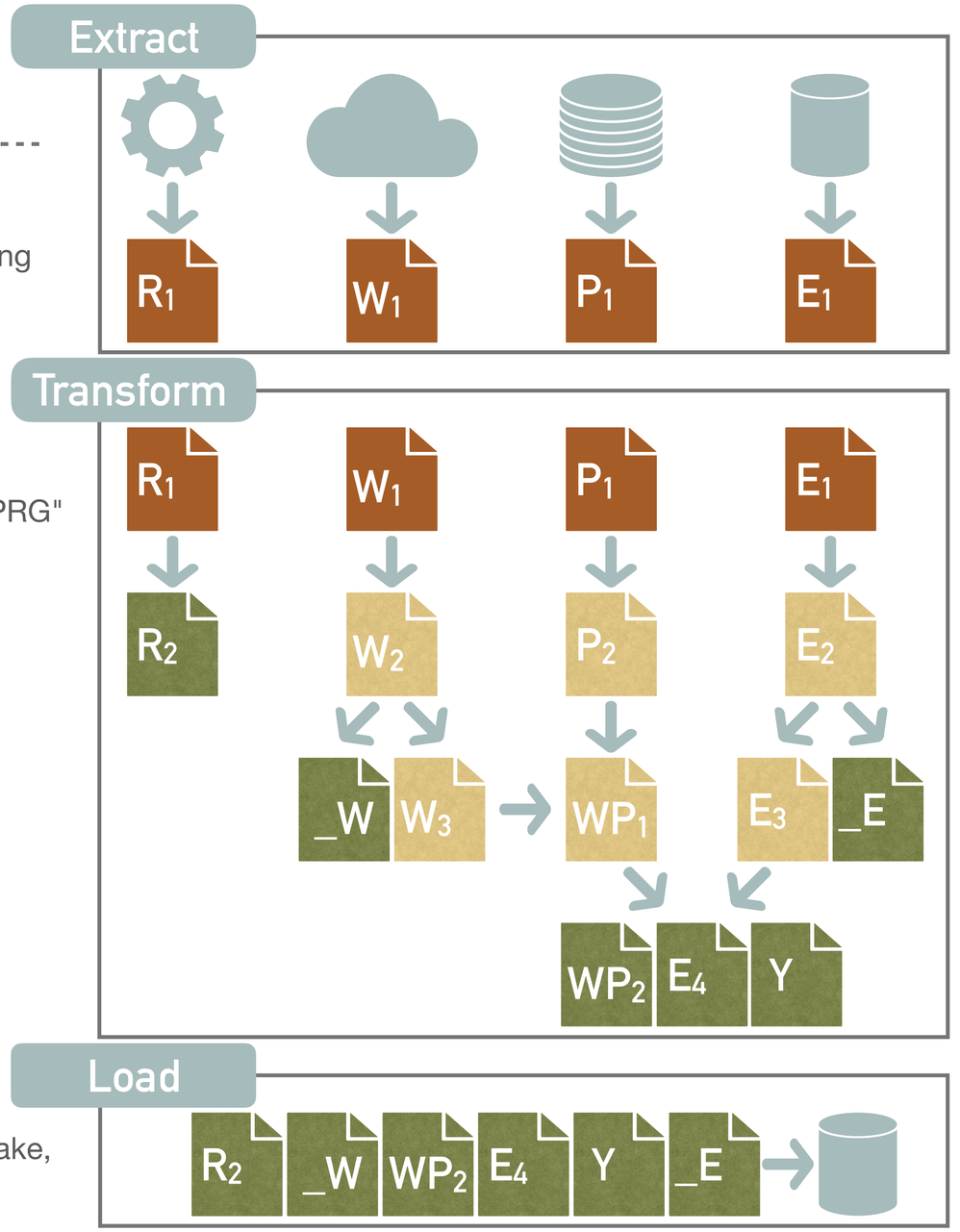
User Story

- The data analysts in our company asked us to prepare waste management datasets and load them to the data warehouse so that they could perform the required analyses
  - Specifically, they need datasets related to waste management in the Czech Republic at the level of individual regions and the extent of funds used to mitigate the environmental burden due to waste management
  - Their aim is to assess municipal and corporate waste production and address impacts, and they require that the amount of waste can be calculated per capita or per unit area
  - As a source of data, we are to use open data on waste from Czech open data portal, data published by the Czech Statistical Office and generally known facts can be extracted from Wikipedia

- **Data Engineer roles:**
  - *Extract datasets* from (various) sources
  - *Transform data* into a uniform form, detect and correct inconsistencies, etc.
  - *Load* the *data to* a *data warehouse* so that analysts can perform analyses

Extract Transform Load (ETL)

- **Data extraction** involves extracting data from homogeneous or heterogeneous sources
- **Data transformation** processes clean and transform data into a suitable format/structure for querying and analysis:
  - *Projection* of specific columns to load or excluding null columns from loading
  - *Selection*, i.e., filtering records based on specific criteria (e.g., a certain attribute value equality)
  - *Translation of coded values* from one format to another
  - *Encoding free-form values*, i.e., mapping values from one format to another, e.g., "Prague" to "PRG"
  - *Surrogate-key generation* to maintain data integrity
  - *Deriving a value* based on existing data
  - *Sorting* data based on specified columns to enhance search performance
  - *Joining data* from multiple sources, including lookup and merge operations, and removing duplicates
  - *Aggregation*, i.e., summarizing multiple rows of data
  - *Data normalization*, i.e., breaking down repeating fields into individual records
  - *Transposing or pivoting*, i.e., restructuring data from multiple columns into rows or vice versa
  - *Column splitting* into multiple columns, e.g., parsing comma-separated values
  - *Data validation* to verifying data integrity and ensuring that relevant data (from tables) is valid
  - *Data cleaning* to ensure that only proper data is passed, eliminating inconsistencies and errors
- **Data loading** involves the insertion of data into, e.g., operational data store, data warehouse, data lake, or data mart



Prerequisite: Setting up Python (Linux, macOS)

- *Check* which *version of Python* is installed (if any)
  - If Python 3 is not installed, download the (latest) version of Python[#1] and follow the installation

- Once installed, *create* any *folder for your NDBI046 project* and navigate to it, e.g., ~/Projects/python-ndbi046
- *Create* your *Python environment*, e.g., ndbi046_env
- *Activate* you Python environment

- *Install* the required *packages*
  - Download the requirements.txt file from the practical class website
  - You may also *install additional packages*
  - You may always export the list of installed packages to a file

- Exit the Python environment after completing the practical class (not before)
  - You can return to the environment at any time by activating it

checking the installed version

creating and activating a virtual environment

installation of the required packages

export of installed packages

list installed packages

deactivating a virtual environment

```
1  % python3 --version
2
3  % mkdir ~/Projects/python-ndbi046
4  % cd ~/Projects/python-ndbi046
5
6  % python3 -m venv ndbi046_env
7
8  % source ndbi046_env/bin/activate
9
10 (ndbi046_env) % pip install -r requirements.txt
11
12 (ndbi046_env) % pip install pandas
13
14 (ndbi046_env) % pip freeze > requirements.txt
15
16 (ndbi046_env) % deactivate
17
18 % cat requirements.txt
```

#1 https://www.python.org/downloads/

Prerequisite: Setting up Python (Windows)

- *Check* which *version of Python* is installed (if any)
  - If Python 3 is not installed, download the (latest) version of Python[#1] and follow the installation

- Once installed, *create* any *folder for your NDBI046 project* and navigate to it, e.g., C:\Projects\python-ndbi046
- *Create* your *Python environment*, e.g., ndbi046_env
- *Activate* you Python environment

- *Install* the required *packages*
  - Download the requirements.txt file from the practical class website
  - You may also *install additional packages*
  - You may always export the list of installed packages to a file

- Exit the Python environment after completing the practical class (not before)
  - You can return to the environment at any time by activating it

checking the installed version

creating and activating a virtual environment

installation of the required packages

export of installed packages

list installed packages

deactivating a virtual environment

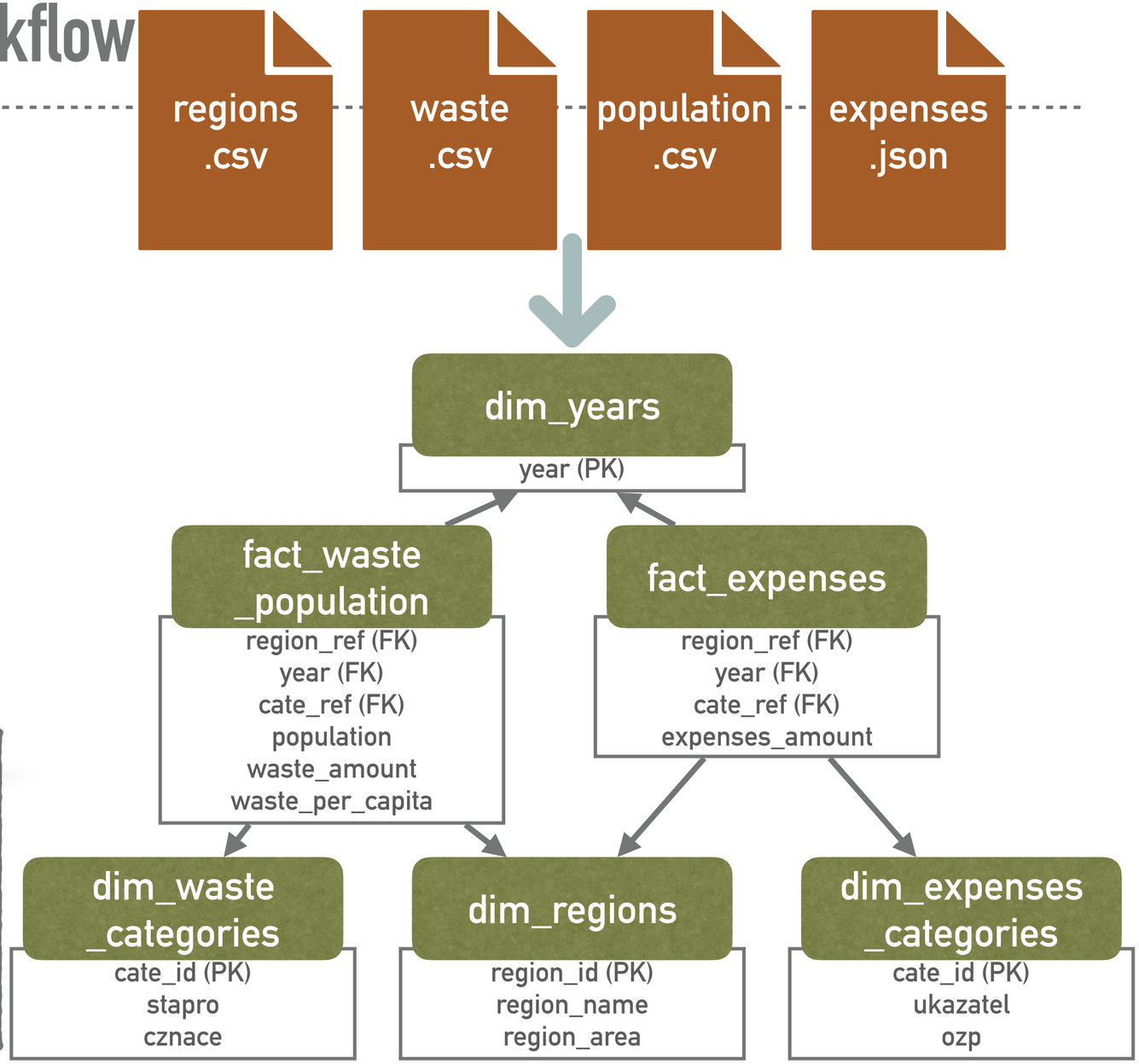
```
1  > python3 --version
2
3  > mkdir C:\Projects\python-ndbi046
4  > cd C:\Projects\python-ndbi046
5
6  > python3 -m venv ndbi046_env
7
8  > ndbi046_env\Scripts\activate.bat
9
10 (ndbi046_env) > pip install -r requirements.txt
11
12 (ndbi046_env) > pip install pandas
13
14 (ndbi046_env) > pip freeze > requirements.txt
15
16 (ndbi046_env) > deactivate
17
18 > type requirements.txt
```

#1 https://www.python.org/downloads/

Example 3.1: Data Transformation Workflow

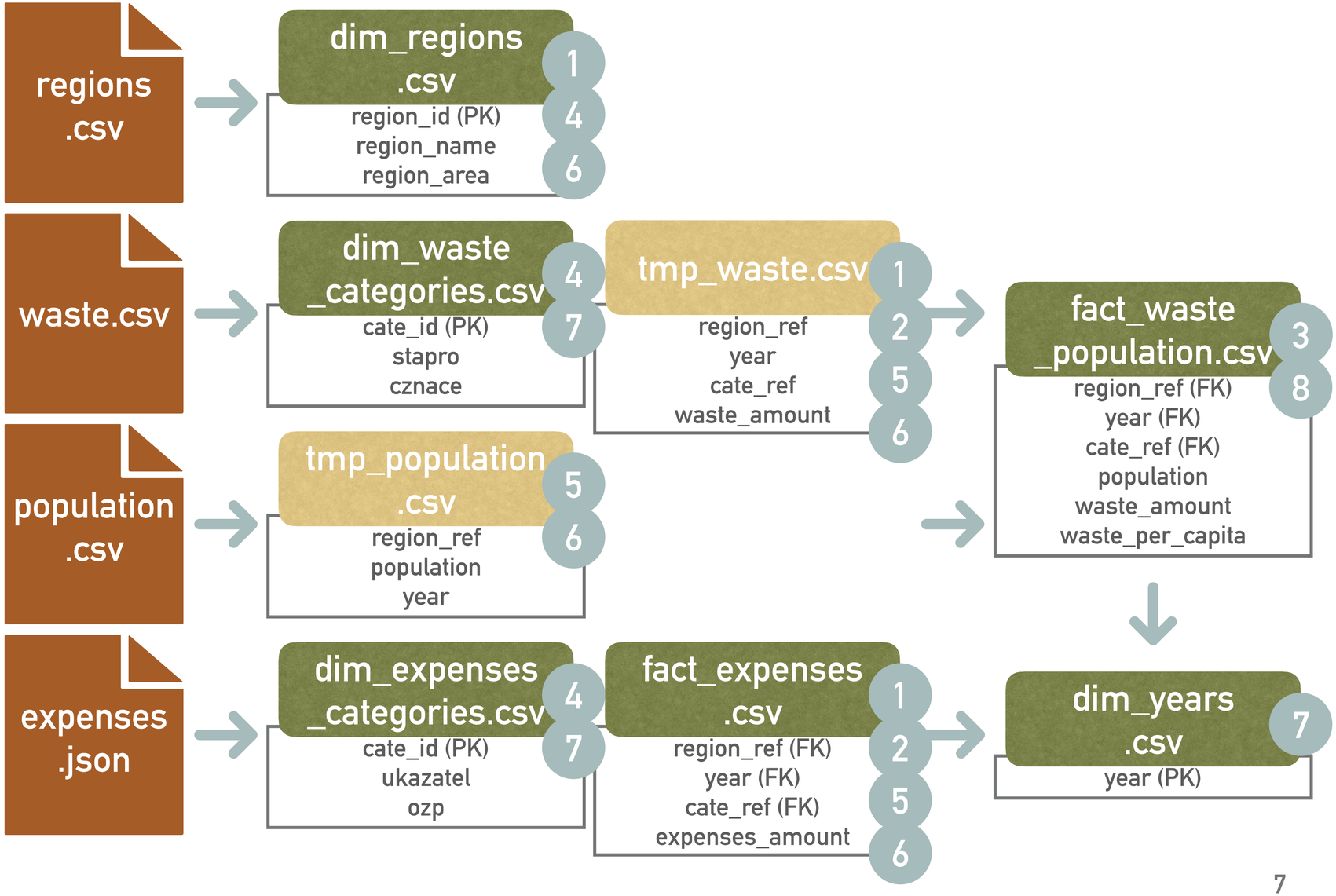
- Implementing the *requirements of data analysts* in order to efficiently evaluate queries *leads to* the design of a *galaxy schema* for the data warehouse
  - *Fact tables:* fact_waste_population, fact_expenses
  - *Dimensions*: dim_years, dim_regions, dim_waste_categories, dim_expenses_categories
    - Dimensions dim_waste_categories and dim_expenses_categories are created by normalising the repeating data in the original datasets
    - The dim_years dimension is created by identifying all (common) years for which we have some data, and allows, together with dim_regions, to efficiently join the corresponding rows of the fact tables
  - *Measures*: waste_amount, waste_per_capita, population, expenses_amount

- Having extracted datasets regions.csv, waste.csv, population.csv, and expenses.json from the last practical class, *propose a data transformation workflow* for the input datasets
  - In particular, e.g., determine the appropriate common data format, resolve inconsistencies in the data, and normalize the data.
  - The aim is to load the data into the data warehouse in a form corresponding to the proposed galaxy schema



Example 3.1: Data Transformation Workflow (Solution)

- We have identified the operations needed to transform the input datasets:
  1. *Projection* of selected columns
  2. *Selection* of rows (data filtering)
  3. *Joining* of datasets
  4. Adding a *surrogate key*
  5. *Substituting values* for a reference
  6. Adjust *number formatting*
  7. *Normalizing* repetitive data
  8. *Deriving* new values



Example 3.2: Transform Region Data (CSV)

- Based on the transformation workflow (see Example 3.1), we want to *convert the input dataset* dataset_regions.csv *into* the *dimension* dim_regions
  - The dimension will contain the attributes region_id (PK), region_name (i.e., region name), and region_area (i.e., region area)
- Ensure *data consistency* (between datasets)
  - For example, keep region naming and number formatting consistent across datasets

- Identify the steps to *transform the dataset* and then *write a Python script* to perform the transformation
  - Save the transformed result to the file dim_regions.csv

Název kraje (kraj)	Zkratka ČSÚ	Zkratka na RZ	Krajské město	Počet obyvatel	Rozloha (km^2)	Hustota zalidnění	HDP	HDP na obyv.
Hlavní město Praha	PHA	A	Praha	1 357 326	496,21	2 735	1 926,3	1 453,6
Středočeský kraj	STČ	S	Praha	1 439 391	10 928,50	132	775,7	557,6
...	...	...	...	...	...	...	...	...
Zlínský kraj	ZLK	Z	Zlín	580 531	3 963,04	146	304,8	524,9

region_id	region_name	region_area
1	Hlavní město Praha	496.21
2	Středočeský kraj	10928.50
...	...	...
14	Zlínský kraj	3963.04

- **Tip:** Utilize the Pandas framework for data loading and transformation

Example 3.2: Transform Region Data (CSV) (Solution)

- First, we identify the particular steps of the dataset transformation:
  - Addition of a *surrogate key* (e.g., the index of the row determines the value of the integer key)
  - *Projection* of columns region_id, Název kraje (kraj), and Rozloha (km^2)
  - *Renaming* columns Název kraje (kraj) → region_name a Rozloha (km^2) → region_area
  - Modification of the *number formatting* in the region_area column from Czech to English format

Název kraje (kraj)	Zkratka ČSÚ	Zkratka na RZ	Krajské město	Počet obyvatel	Rozloha (km^2)	Hustota zalidnění	HDP	HDP na obyv.
Hlavní město Praha	PHA	A	Praha	1 357 326	496,21	2 735	1 926,3	1 453,6
Středočeský kraj	STČ	S	Praha	1 439 391	10 928,50	132	775,7	557,6
...	...	...	...	...	...	...	...	...
Zlínský kraj	ZLK	Z	Zlín	580 531	3 963,04	146	304,8	524,9

region_id	region_name	region_area
1	Hlavní město Praha	496.21
2	Středočeský kraj	10928.50
...	...	...
14	Zlínský kraj	3963.04

an integer primary key

projection and column name change

adjusted number formatting

Example 3.2: Transform Region Data (CSV) (Solution)

```python
import logging
import sys
import pandas as pd

logging.basicConfig(level = logging.INFO, format="%(levelname)s: %(message)s")

def load_csv_to_dataframe(file_path: str) -> pd.DataFrame:
  pass

def add_surrogate_key(dataframe: pd.DataFrame, ident_name: str) -> pd.DataFrame:
  pass

def project_columns(dataframe: pd.DataFrame, columns: list) -> pd.DataFrame:
  pass

def rename_columns(dataframe: pd.DataFrame, column_names: dict) -> pd.DataFrame:
  pass

def format_numbers(dataframe: pd.DataFrame, column: str) -> pd.DataFrame:
  pass

def save_dataframe_to_csv(dataframe: pd.DataFrame, file_path: str) -> None:
  pass
```

Import the necessary libraries

Logging settings

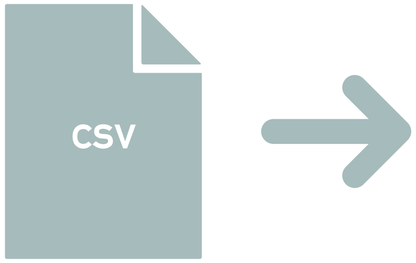
Decompose the task into individual subtasks with appropriate interface

Example 3.2: Transform Region Data (CSV) (Solution)

```python
def load_csv_to_dataframe(file_path: str) -> pd.DataFrame:
    try:
        dataframe = pd.read_csv(file_path)
        return dataframe
    except Exception as e:
        logging.error(f"Error loading CSV file: {e}")
        raise

def add_surrogate_key(dataframe: pd.DataFrame, ident_name: str) -> pd.DataFrame:
    try:
        dataframe[ident_name] = range(1, len(dataframe) + 1)
        return dataframe
    except Exception as e:
        logging.error(f"Error adding surrogate key: {e}")
        raise
```

loading a csv file using pandas

adding an integer surrogate key, where the row index determines its value



region_id	Název kraje (kraj)	Zkratka ČSÚ	Zkratka na RZ	Krajské město	Počet obyvatel	Rozloha (km^2)	Hustota zalidnění	HDP	HDP na obyv.
1	Hlavní město Praha	PHA	A	Praha	1 357 326	496,21	2 735	1 926,3	1 453,6
2	Středočeský kraj	STČ	S	Praha	1 439 391	10 928,50	132	775,7	557,6
...	...	...	...	...	...	...	...	...	...
14	Zlínský kraj	ZLK	Z	Zlín	580 531	3 963,04	146	304,8	524,9

Example 3.2: Transform Region Data (CSV) (Solution)

```python
def project_columns(dataframe: pd.DataFrame, columns: list) -> pd.DataFrame:
    try:
        return dataframe[columns]
    except Exception as e:
        logging.error(f"Error projecting columns: {e}")
        raise

def rename_columns(dataframe: pd.DataFrame, column_names: dict) -> pd.DataFrame:
    try:
        return dataframe.rename(columns=column_names, copy=False)
    except Exception as e:
        logging.error(f"Error renaming columns: {e}")
        raise
```

the operator [] performs the projection of the specified columns

rename the columns

region_id	Název kraje (kraj)	Zkratka ČSÚ	Zkratka na RZ	Krajské město	Počet obyvatel	Rozloha (km^2)	Hustota zalidnění	HDP	HDP na obyv.
1	Hlavní město Praha	PHA	A	Praha	1 357 326	496,21	2 735	1 926,3	1 453,6
2	Středočeský kraj	STČ	S	Praha	1 439 391	10 928,50	132	775,7	557,6
...	...	...	...	...	...	...	...	...	...
14	Zlínský kraj	ZLK	Z	Zlín	580 531	3 963,04	146	304,8	524,9

region_id	region_name	region_area
1	Hlavní město Praha	496,21
2	Středočeský kraj	10 928,50
...	...	...
14	Zlínský kraj	3 963,04

Example 3.2: Transform Region Data (CSV) (Solution)

```python
def format_numbers(dataframe: pd.DataFrame, column: str) -> pd.DataFrame:
    try:
        dataframe[column].str.replace(r"\s+", "", regex=True).str.replace(",", ".")
        return dataframe
    except Exception as e:
        logging.error(f"Error formatting numbers: {e}")
        raise

def save_dataframe_to_csv(dataframe: pd.DataFrame, file_path: str) -> None:
    try:
        dataframe.to_csv(file_path, index=False)
        logging.info(f"Result has been saved to the file: {file_path}")
    except Exception as e:
        logging.error(f"Failed to save the result to CSV file: {e}")
        raise
```

adjusting the thousands and decimal separators

saving the resulting dataframe to csv file

region_id	region_name	region_area
1	Hlavní město Praha	496,21
2	Středočeský kraj	10 928,50
...	...	...
14	Zlínský kraj	3 963,04

region_id	region_name	region_area
1	Hlavní město Praha	496.21
2	Středočeský kraj	10928.50
...	...	...
14	Zlínský kraj	3963.04

Example 3.2: Transform Region Data (CSV) (Solution)

```
53 if __name__ == "__main__":
54   if len(sys.argv) != 3:
55     logging.error("Usage: python script.py <input_file_path> <output_file_path>")
56     sys.exit(1)
57
58   input_file_path = sys.argv[1]
59   output_file_path = sys.argv[2]
60
61   try:
62     dataframe = load_csv_to_dataframe(input_file_path)
63     dataframe = add_surrogate_key(dataframe, "region_id")
64     dataframe = project_columns(
65       dataframe, ["region_id", "Název kraje  (kraj)", "Rozloha  (km²)"]
66     )
67     dataframe = rename_columns(
68       dataframe,
69       {"Název kraje  (kraj)": "region_name", "Rozloha  (km²)": "region_area"},
70     )
71     dataframe = format_numbers(dataframe, "region_area")
72     save_dataframe_to_csv(dataframe, output_file_path)
73   except Exception as e:
74     logging.error(f"Failed to process the data: {e}")
```

the program accepts two arguments: (1) the path to the input file and (2) the path to the output file

assembling a transformation from individual steps

Exercise 3.3: Transform Waste Data (CSV)

- Based on the transformation workflow (see Example 3.1), we want to *convert the input dataset* dataset_waste.csv *into dimension* dim_waste_categories *and* into a temporary dataset as a part of the *fact table* fact_waste_population
  - The dimension will contain the attributes cate_id (i.e., an identifier), stapro (i.e., industrial or municipal), and cznace (i.e., waste source specification)
  - The fact table will consist of attributes region_ref (i.e., a reference to dim_regions), year, cate_ref (i.e., a reference to dim_waste_categories), and waste_amount
- *Select* only the *rows* where the value in the column uzemi_cis equal 100 (i.e., we want only the rows containing data for one of the regions)
- *Normalize the* repeating *data* (i.e., columns stapro_txt and cznace_txt forming waste categories)
- Ensure *data consistency* (between datasets)
  - *Encode region names* according to the values in dim_regions.csv (see Example 3.2), where, e.g., "Hlavní město Praha" → 1
  - Keep *number formatting consistent* across datasets
- Identify the steps to *transform the datasets* and then *write a Python script* to perform the transformation
  - Save the transformed results into files tmp_waste.csv (transformed original dataset) and dim_waste_category.csv (normalized waste categories)

region_ref	year	cate_ref	waste_amount
4	2010	1	136904
10	2010	1	155268
...	...	...	...
7	2017	9	84349

idhod	hodnota	duvernost	stapro_kod	cznace_cis	cznace_kod	rok	uzemi_cis	uzemi_kod	stapro_txt	cznace_txt	uzemi_txt
745245377	136904	verejny	5319			2010	100	3042	Celková produkce komu...		Plzeňský kraj
745245701	155268	verejny	5319			2010	100	3107	Celková produkce komu...		Kraj Vysočina
...	...	...	...	...	...	...	...	...	...	...	...
793266427	84349	verejny	5760	5724	37390001	2017	100	3077	Produkce podnikového...	Činnost související...	Liberecký kraj

cate_id	stapro	cznace
1	Celková produkce komu...	
...		...
8	Produkce podnikového...	Těžba a dobývání
9	Produkce podnikového...	Činnost související...

- **Tip:** Utilize the Pandas framework for data loading and transformation

Exercise 3.4: Transform Population Data (CSV)

- Based on the transformation workflow (see Example 3.1), we want to *convert the input dataset* dataset_population.csv *into* a temporary dataset that will be a part of the *fact table* fact_waste_population
  - The temporary dataset will contain the attributes region_ref (i.e., a reference to dim_regions), population (i.e., number of residents), and year
- Ensure *data consistency* (between datasets)
  - *Encode region names* according to the values in dim_regions.csv (see Example 3.2), where, e.g., "Hlavní město Praha" → 1
  - Keep *number formatting consistent* across datasets

- Identify the steps to *transform the dataset* and then *write a Python script* to perform the transformation
  - Save the transformed result to the file tmp_population.csv

note the differences in the naming of the regions - we need to correct inconsistencies

kraj	pocet_obyvatel	rok
Hl. m. Praha	1251075	2014
Středočeský	1309139	2014
...	...	...
Moravskoslezský	1187776	2022

region_ref	population	year
1	1251075	2014
2	1309139	2014
...	...	...
13	1187776	2022

- **Tip:** Utilize the Pandas framework for data loading and transformation

Example 3.5: Join Waste and Population Data

- Based on the transformation workflow (see Example 3.1), we want to *join the temporary datasets* tmp_waste.csv and tmp_population.csv *into* a dataset that corresponds to the *fact table* fact_waste_population
  - Perform *inner join* according to the matching values in the columns (region_ref, year)
- Create a *derived column* waste_per_capita in the joined dataset, calculated *as a fraction* of the values from the columns value and population

- Identify the steps to *transform the datasets* and then *write a Python script* to perform the transformation
  - Save the transformed result to the file fact_waste_population.csv

region_ref	year	cate_ref	waste_amount
4	2010	1	136904
10	2010	1	155268
...	...	...	...
7	2017	9	84349

region_ref	population	year
1	1251075	2014
2	1309139	2014
...	...	...
13	1187776	2022

region_ref	year	cate_ref	population	waste_amount	waste_per_capita
4	2014	1	573993	144478	0.251706...
10	2014	1	510006	167954	0.329337...
...	...	...	...	...	...
7	2017	9	4440934	84349	0.191296...

- **Tip:** Utilize the Pandas framework for data loading and transformation

Example 3.5: Join Waste and Population Data (Solution)

```python
import logging
import sys
import pandas as pd

logging.basicConfig(level = logging.INFO, format="%(levelname)s: %(message)s")

def load_csv_to_dataframe(file_path: str) -> pd.DataFrame:
  pass

def join_dataframes(first: pd.DataFrame, second: pd.DataFrame, on: list, how: str
) -> pd.DataFrame:
  pass

def calculate_new_column_as_share(dataframe: pd.DataFrame, divident: str, divisor: str,
fraction: str) -> pd.DataFrame:
  pass

def save_dataframe_to_csv(dataframe: pd.DataFrame, file_path: str) -> None:
  pass
```

Import the necessary libraries

Logging settings

this has already been implemented in Example 3.2

this has already been implemented in Example 3.2

Example 3.5: Join Waste and Population Data (Solution)

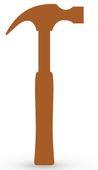
```python
def join_dataframes(first: pd.DataFrame, second: pd.DataFrame, on: list, how: str
) -> pd.DataFrame:
  try:
    joined_df = pd.merge(first, second, on=on, how=how)
    return joined_df
  except Exception as e:
    logging.error(f"Error joining data: {e}")
    raise

def calculate_new_column_as_share(dataframe: pd.DataFrame, divident: str, divisor: str,
fraction: str) -> pd.DataFrame:
  try:
    dataframe[fraction] = dataframe[divident] / dataframe[divisor]
    return dataframe
  except Exception as e:
    logging.error(f"Error calculating new column: {e}")
    raise
```

region_ref	year	cate_ref	waste_amount
4	2010	1	136904
10	2010	1	155268
...	...	...	...
7	2017	9	84349

region_ref	population	year
1	1251075	2014
2	1309139	2014
...	...	...
13	1187776	2022

region_ref	year	cate_ref	population	waste_amount	waste_per_capita
4	2014	1	573993	144478	0.251706...
10	2014	1	510006	167954	0.329337...
...	...	...	...	...	...
7	2017	9	4440934	84349	0.191296...

Example 3.5: Join Waste and Population Data (Solution)



```python
if __name__ == "__main__":
  if len(sys.argv) != 4:
    logging.error(
      "Usage: python script.py <input_file_path1> <input_file_path2> <output_file_path>"
    )
    sys.exit(1)

  input_file_path1 = sys.argv[1]
  input_file_path2 = sys.argv[2]
  output_file_path = sys.argv[3]

  try:
    first_dataframe = load_csv_to_dataframe(input_file_path1)
    second_dataframe = load_csv_to_dataframe(input_file_path2)

    data = join_dataframes(first_dataframe, second_dataframe, ["region_ref", "year"], "inner")
    data = calculate_new_column_as_share(
      data, "waste_amount", "population", "waste_per_capita"
    )

    save_dataframe_to_csv(data, output_file_path)
  except Exception as e:
    logging.error(f"Failed to process the data: {e}")
```

the program accepts three arguments: (1+2) the paths to the input files and (3) the path to the output file

assembling a transformation from individual steps

Exercise 3.6: Transform Expenses Data (JSON)

- Based on the transformation workflow (see Example 3.1), we want to *convert the input dataset* dataset_expenses.json *into dimension* dim_expenses_categories *and* into *fact table* fact_expenses
  - The dimension will contain the attributes cate_id (i.e., an identifier), ukazatel, and ozp
  - The fact table will consist of attributes region_ref (i.e., a reference to dim_regions), year, cate_ref (i.e., a reference to dim_expenses_categories), and expenses_amount (i.e., amount of waste)
- *Select* only the *rows* where the value in the column uzemi_cis equal 100 (i.e., we want only data for one of the regions) and where the value in the column ozp_txt is equal 'Ekologické nakládání s odpady' or 'Ekologické nakládání s odpadními vodami' (i.e., we want only the rows containing data related to waste management)
- *Normalize* the *repeating data* (i.e., columns ukazatel_txt and ozp_txt forming expenses categories)
- Ensure *data consistency* (between datasets)
  - *Encode region names* according to the values in dim_regions.csv (see Example 3.2), where, e.g., "Hlavní město Praha" → 1
  - Keep *number formatting consistent* across datasets
- *Resolve* the *absence of values* in the column value in an appropriate way
  - For example, by removing the entire row, determining a qualified estimate, approximating the value or finding the exact value

- Identify the steps to *transform the datasets* and then *write a Python script* to perform the transformation
  - Save the transformed results into files fact_expenses.csv (transformed original dataset) and dim_expenses_category.csv (normalized expenses categories)

- **Tip:** Utilize the Pandas framework for data loading and transformation

Exercise 3.7: Finalize Data (CSV)

- Based on the transformation workflow (see Example 3.1), we want to *create dataset* dim_years as a result of all (common) years in datasets fact_expenses.csv and fact_waste_population.csv
  - The dimension will contain single attribute year
- Decide whether it *is appropriate to remove rows* from the fact_expenses.csv and fact_waste_population.csv datasets that we cannot associate with corresponding rows from the other dataset via a pair of values (year, region_ref)
  - If you decide to remove rows and specify only common years, add the following functions to the script
  - *Provide a reason* for your decision (i.e., remove or keep all data)

- Identify the steps to *transform the datasets* and then *write a Python script* to perform the transformation
  - Save the transformed results into files dim_years.csv (as (common) years) and if appropriate also overwrite the input files fact_expenses.csv and fact_waste_population.csv

- **Tip:** Utilize the Pandas framework for data loading and transformation

References

Python

- Python 3.x (LATEST) documentation: https://docs.python.org/3/
- venv documentation. https://docs.python.org/3/library/venv.html
- Python W3Schools Tutorial: https://www.w3schools.com/python/
- Pandas W3Schools Tutorial: https://www.w3schools.com/python/pandas/default.asp