Coursework Two

Release 1.0

Team Birch

May 02, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

INSTALLATION GUIDE

To set up this project locally:

1. Clone the repository:

```
git clone git clone https://github.com/Boissek123/coursework_two.git
cd coursework_two
```

2. Install Poetry (if not already installed):

```
curl -sSL https://install.python-poetry.org | python3 -
```

3. Install dependencies:

```
poetry install
```

4. Create a *.env* file with the following variables:

```
DEEPSEEK_API=your_deepseek_api_key

MINIO_ENDPOINT=minio:9000
MINIO_ACCESS_KEY=ift_bigdata
MINIO_SECRET_KEY=minio_password
MINIO_SECURE=false

DB_NAME=fift
DB_USER=postgres
DB_PASSWORD=postgres
DB_HOST=postgres
DB_PORT=5439
SCHEMA_NAME=csr_reporting
TABLE_NAME=company_indicators
```

5. Run the extraction pipeline:

```
poetry run python Main.py
```

6. Database & Post-processing

   After extraction, three CSV-based scripts handle post-processing:

   - Data Cleaning ((*data_cleaning.py*)): cleans and reshapes raw CSR indicators CSV:

     ```
     poetry run python src/modules/output/data_cleaning.py
     ```

- Metadata Export & Merge ((*reports_export.py*)): exports and merges (*company_reports*) and (*company_static*) from Postgres:

  ```
  poetry run python src/modules/output/reports_export.py
  ```

- Postgres Load ((*db_load.py*)): creates schema/table and bulk-loads (*csr_indicators.csv*):

  ```
  poetry run python src/modules/output/db_load.py
  ```

Once complete, the cleaned and validated CSR indicator data will be available in the (*csr_reporting.company_indicators*) table in your Postgres database.

CHAPTER

TWO

USAGE INSTRUCTIONS

1. **Run the extraction pipeline**:

   ```
   poetry run python Main.py
   ```

   This will: - Connect to MinIO and stream CSR PDF reports - Chunk each PDF into text blocks - Query Deepseek API to extract indicator values - Append results to *logs/final_output.csv*

2. **Post-processing & Data Cleaning**

   After extraction, clean, merge and load your CSR data:

   a. **Data Cleaning**: normalize and split values in the raw CSV:

      ```
      poetry run python src/modules/output/data_cleaning.py
      ```

   b. **Metadata Export & Merge**: export *company_reports* and *company_static* tables from Postgres and merge:

      ```
      poetry run python src/modules/output/reports_export.py
      ```

   c. **Database Load**: create schema/table and bulk-load the cleaned CSV into Postgres:

      ```
      poetry run python src/modules/output/db_load.py
      ```

2.1 Troubleshooting Tips

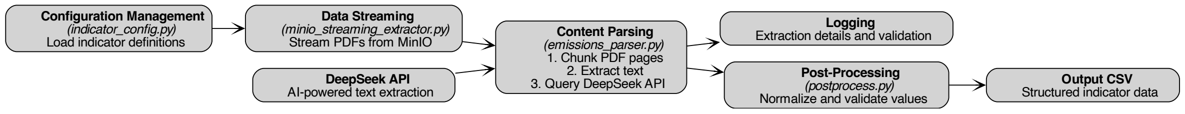
- **Deepseek API errors**: • Verify *DEEPSEEK_API* in your *.env* is correct. • Check your plan's rate limits.
- **MinIO connection issues**: • Confirm *MINIO_ENDPOINT*, *MINIO_ACCESS_KEY*, *MINIO_SECRET_KEY* and *MINIO_SECURE* in *.env*.
- **Database connectivity**: • Ensure Postgres is running and the *DB_** credentials in *.env* match your instance.
- **No indicators extracted**: • Inspect *logs/final_log.txt* for parsing warnings or failures.

CHAPTER

THREE

ARCHITECTURE OVERVIEW

3.1 Data Flow



3.2 Components

- src/modules/input/: Core extraction logic ((*emissions_parser.py*), (*minio_streaming_extractor.py*), (*indicator_config.py*), (*postprocess.py*)).
- src/modules/db/data_storage.py: Stores cleaned data into PostgreSQL ((*csr_reporting.company_indicators*)).
- src/modules/output/data_clean.py: Cleans and reshapes raw extracted CSV into standardized format.
- src/modules/output/data_export.py: Merges company metadata with report information for downstream use.
- config/indicators.yaml: Indicator definitions, aliases, and validation rules.
- logs/: Stores extraction output ((*final_output.csv*)) and processing logs.
- PostgreSQL: Schema (*csr_reporting*), (*table company_indicators*), (*alongside company_reports*) and (*company_static*) for metadata.

CHAPTER

FOUR

API REFERENCE

This page contains auto-generated API reference documentation[1].

4.1 modules

4.1.1 Submodules

modules.db

Submodules

modules.db.data_storage

Functions

store_clean_data_to_postgres(cleaned_csv_path)	Store cleaned CSR indicator data into a PostgreSQL database.

Module Contents

modules.db.data_storage.**store_clean_data_to_postgres**(*cleaned_csv_path: str*)

Store cleaned CSR indicator data into a PostgreSQL database.

This function reads a cleaned CSV file containing company sustainability indicators and writes the data into a PostgreSQL table *csr_reporting.company_indicators*. If the table or schema does not exist, they will be created. If the table already exists, the new data will be appended.

Environment variables must be set via a *.env* file or system environment for: - POSTGRES_USER - POSTGRES_PASSWORD - POSTGRES_HOST - POSTGRES_PORT - POSTGRES_DB

The CSV file is expected to have the following columns (in order): - company - year - annual_carbon_emissions_tonnes_co2 - annual_carbon_emissions_reduction_percent - annual_water_use_cubic_meters - annual_water_use_reduction_percent - renewable_energy_use_mwh - sustainable_materials_ratio_percent - waste_recycling_rate_percent

Parameters

cleaned_csv_path (*str*) – Path to the cleaned CSV file containing CSR indicator data.

Raises

- **psycopg2.Error** – If connection to the PostgreSQL database fails.

[1] Created with sphinx-autoapi

- **pandas.errors.EmptyDataError** – If the CSV file is empty or unreadable.
- **ValueError** – If column count or names in the CSV do not match expected schema.

modules.db.db_connection

modules.input

Submodules

modules.input.emissions_parser

Handles extraction of CSR indicators from PDF files using DeepSeek API.

Includes text parsing, relevance detection, DeepSeek querying, postprocessing, and CSV output.

Attributes

logger
UNIT_PATTERN
session

Functions

build_alias_map(indicator_config)	Builds a mapping from indicator aliases to canonical indicator names.
build_indicator_labels(indicator_config)	Builds a flat list of all indicator names and aliases.
extract_keywords(→ set)	Extracts all indicator names and aliases into a set of lowercase keywords.
is_relevant_chunk(→ bool)	Determines if a text chunk is relevant based on presence of keywords and units.
query_deepseek(→ str)	Sends extracted PDF text to DeepSeek API for CSR indicator extraction.
extract_indicators_from_bytes(company_name, pdf_bytes, ...)	Extracts CSR indicators from a PDF byte stream and saves results to CSV.

Module Contents

modules.input.emissions_parser.**logger**

modules.input.emissions_parser.**UNIT_PATTERN**

modules.input.emissions_parser.**build_alias_map**(*indicator_config*)

Builds a mapping from indicator aliases to canonical indicator names.

Parameters

indicator_config (*list*) – List of indicator group configurations.

Returns

Mapping from alias (lowercase) to canonical indicator name.

Return type
dict

`modules.input.emissions_parser.`**`build_indicator_labels`**`(`*indicator_config*`)`

Builds a flat list of all indicator names and aliases.

Parameters
`indicator_config` (*`list`*) – List of indicator group configurations.

Returns
List of indicator names and aliases.

Return type
list

`modules.input.emissions_parser.`**`extract_keywords`**`(`*indicator_config: list*`)` → set

Extracts all indicator names and aliases into a set of lowercase keywords.

Parameters
`indicator_config` (*`list`*) – List of indicator group configurations.

Returns
Set of keywords for indicator matching.

Return type
set

`modules.input.emissions_parser.`**`is_relevant_chunk`**`(`*text: str*, *keywords: set*`)` → bool

Determines if a text chunk is relevant based on presence of keywords and units.

Parameters

- **`text`** (*`str`*) – Text chunk to evaluate.
- **`keywords`** (*`set`*) – Set of indicator keywords.

Returns
True if the chunk is relevant, False otherwise.

Return type
bool

`modules.input.emissions_parser.`**`session`**

`modules.input.emissions_parser.`**`query_deepseek`**`(`*api_key: str*, *pdf_text: str*, *indicator_config: list*, *extract_header: bool = False*`)` → str

Sends extracted PDF text to DeepSeek API for CSR indicator extraction.

Parameters

- **`api_key`** (*`str`*) – DeepSeek API key.
- **`pdf_text`** (*`str`*) – Text extracted from the PDF.
- **`indicator_config`** (*`list`*) – Indicator configuration for prompts.
- **`extract_header`** (*`bool, optional`*) – Whether to only extract header metadata. Defaults to False.

Returns
DeepSeek API extracted text response.

Return type
str

`modules.input.emissions_parser.`**`extract_indicators_from_bytes`**`(`*company_name: str*, *pdf_bytes: io.BytesIO*, *config_path: pathlib.Path*, *output_csv: pathlib.Path*, *log_path: pathlib.Path*, *source_filename: str = 'unknown_file.pdf'*`)`

Extracts CSR indicators from a PDF byte stream and saves results to CSV.

Parameters

- **`company_name`** (*`str`*) – Company name for labeling extracted data.
- **`pdf_bytes`** (*`BytesIO`*) – Byte stream of the PDF file.
- **`config_path`** (*`Path`*) – Path to the indicators configuration YAML.
- **`output_csv`** (*`Path`*) – Path to save extracted CSV output.
- **`log_path`** (*`Path`*) – Path to save extraction logs.
- **`source_filename`** (*`str, optional`*) – Source file name for lineage tracking. Defaults to "unknown_file.pdf".

Returns

- Dictionary mapping indicator names to extracted values.
- List of extracted lineage records for audit purposes.

Return type

tuple[dict, list]

modules.input.indicator_config

Loads the sustainability indicator configuration from a YAML file.

Functions

`load_indicator_config`(→ list)	Loads the indicator configuration from a YAML file.

Module Contents

`modules.input.indicator_config.`**`load_indicator_config`**`(`*path: pathlib.Path*`)` → list

Loads the indicator configuration from a YAML file.

Parameters

`path` (*`Path`*) – Path to the YAML configuration file.

Returns

Parsed list of indicator group configurations.

Return type

list

modules.input.input_reader

modules.input.minio_streaming_extractor

Streams PDF files from MinIO storage and triggers CSR indicator extraction.

Attributes

logger

Functions

connect_to_minio_from_env()	Connects to MinIO server using environment variables.
stream_pdf_and_extract(minio_client, bucket, ...[, prefix])	Streams PDF files from a MinIO bucket, extracts CSR indicators, and saves results.

Module Contents

modules.input.minio_streaming_extractor.**logger**

modules.input.minio_streaming_extractor.**connect_to_minio_from_env**()

Connects to MinIO server using environment variables.

Returns
Connected MinIO client object.

Return type
Minio

modules.input.minio_streaming_extractor.**stream_pdf_and_extract**(*minio_client*, *bucket*, *config_path*, *output_csv*, *log_path*, *prefix=''*)

Streams PDF files from a MinIO bucket, extracts CSR indicators, and saves results.

Parameters

- **minio_client** (*Minio*) – Initialized MinIO client.
- **bucket** (*str*) – Bucket name containing CSR PDF reports.
- **config_path** (*Path*) – Path to indicators YAML configuration.
- **output_csv** (*Path*) – Path to save extracted results.
- **log_path** (*Path*) – Path to save extraction log.
- **prefix** (*str, optional*) – Object prefix filter inside bucket. Defaults to "".

Returns
None

modules.input.postprocess

Handles normalization, validation, and postprocessing of extracted CSR indicator values.

Attributes

logger

continues on next page

Table 7 – continued from previous page

UNIT_NORMALIZATION
SCALE_MULTIPLIERS
GALLON_TO_CUBIC_METERS
BLOCKED_UNITS

Functions

normalize_unit_and_number(→ str)	Normalizes raw extracted values to expected units.
extract_numeric(→ float)	Extracts numeric value from a string.
validate_value(→ dict)	Validates a numeric value against defined rules.
postprocess_value(→ dict)	Postprocesses and validates an extracted CSR indicator value.

Module Contents

modules.input.postprocess.**logger**

modules.input.postprocess.**UNIT_NORMALIZATION**

modules.input.postprocess.**SCALE_MULTIPLIERS**

modules.input.postprocess.**GALLON_TO_CUBIC_METERS = 0.00378541**

modules.input.postprocess.**BLOCKED_UNITS**

modules.input.postprocess.**normalize_unit_and_number**(*raw_value: str*, *expected_unit: str*) → str

Normalizes raw extracted values to expected units.

Parameters

- **raw_value** (*str*) – Raw extracted string.
- **expected_unit** (*str*) – Expected standardized unit.

Returns
Normalized value or “N/A” if invalid.

Return type
str

modules.input.postprocess.**extract_numeric**(*value: str*) → float

Extracts numeric value from a string.

Parameters
value (*str*) – Input value as string.

Returns
Extracted numeric value or None if parsing fails.

Return type
float

`modules.input.postprocess.`**`validate_value`**`(`*value: str*, *rules: dict*`)` → dict

Validates a numeric value against defined rules.

Parameters

- **`value`** (*`str`*) – Value to validate.
- **`rules`** (*`dict`*) – Validation rule set with min, max, warn_above keys.

Returns
Validation result containing flags and numeric value.

Return type
dict

`modules.input.postprocess.`**`postprocess_value`**`(`*raw_value: str*, *expected_unit: str*, *validation_rules: dict*, *expected_type: str = 'float'*, *aim: str = 'reduction'*`)` → dict

Postprocesses and validates an extracted CSR indicator value.

Parameters

- **`raw_value`** (*`str`*) – Raw extracted value string.
- **`expected_unit`** (*`str`*) – Expected unit for normalization.
- **`validation_rules`** (*`dict`*) – Validation rules for the value.
- **`expected_type`** (*`str, optional`*) – Expected data type (“float” or “int”). Defaults to “float”.
- **`aim`** (*`str, optional`*) – Aim of the indicator (“reduction” or “increase”). Defaults to “reduction”.

Returns
Processed result including normalized value, validation status, and warnings.

Return type
dict

modules.output

Submodules

modules.output.data_clean

This script processes CSR indicator data from multiple CSV files. It performs the following steps:

- Concatenates two sets of scraped CSR indicator data.
- Cleans the report year column.
- Cleans and standardizes renewable energy usage values.
- Removes outliers and handles units for several CSR indicators.
- Renames columns to include unit information.
- Processes fractional columns (e.g., percentages) and removes values > 100.
- Removes duplicate and invalid rows.
- Saves the cleaned CSR indicators to a CSV file.
- Standardizes company names by matching them with a standard company list, using fuzzy matching with a configurable threshold.

- Saves the final standardized company indicators to a CSV file.

Attributes

csr_indicators1
csr_indicators2
csr_indicators
csr_indicators
valid_units
renamed_columns
csr_indicators
columns_to_process
csr_indicators
csr_indicators
standard_df
messy_df
standard_names
messy_names
messy_df
messy_df

Functions

clean_energy_value(x)	Clean and standardize an energy usage value.
clean_and_strip_unit(cell, valid_unit)	Clean a cell value by extracting the numeric component if the specified valid unit is present.
match_company(name, choices[, threshold])	Match a company name to a list of standard names using fuzzy string matching.

Module Contents

modules.output.data_clean.**csr_indicators1 = None**

modules.output.data_clean.**csr_indicators2 = None** `modules.output.data_clean.`**`csr_indicators = None`**

`modules.output.data_clean.`**`csr_indicators`**

`modules.output.data_clean.`**`clean_energy_value`**`(`*x*`)`

Clean and standardize an energy usage value.

Attempts to convert a string representing energy usage to a standardized format with "MWh" as the unit. If the value is expressed in GWh or TWh, it is converted into MWh.

Parameters

`x` – A value (string or numeric) representing the energy usage.

Returns

A string in the format "[numeric value] MWh". If conversion fails, returns an empty string.

`modules.output.data_clean.`**`valid_units`**

`modules.output.data_clean.`**`clean_and_strip_unit`**`(`*cell*, *valid_unit*`)`

Clean a cell value by extracting the numeric component if the specified valid unit is present.

Parameters

- **`cell`** – The cell value to process.
- **`valid_unit`** – The unit that the cell value should contain.

Returns

The numeric part of the cell as a string if the unit is present; otherwise, an empty string.

`modules.output.data_clean.`**`renamed_columns`**

`modules.output.data_clean.`**`csr_indicators`**

`modules.output.data_clean.`**`columns_to_process = ['Sustainable Materials Usage Ratio (percent)', 'Waste Recycling Rate (percent)']`**

`modules.output.data_clean.`**`csr_indicators`**

`modules.output.data_clean.`**`csr_indicators`**

`modules.output.data_clean.`**`standard_df = None`**

`modules.output.data_clean.`**`messy_df = None`**

`modules.output.data_clean.`**`standard_names`**

`modules.output.data_clean.`**`messy_names`**

`modules.output.data_clean.`**`match_company`**`(`*name*, *choices*, *threshold=80*`)`

Match a company name to a list of standard names using fuzzy string matching.

Uses fuzzywuzzy's extractOne function to find the best match. If the score is above the threshold, the best match is returned; otherwise, None is returned.

Parameters

- **`name`** – The company name to match.
- **`choices`** – A list of standard company names.
- **`threshold`** – An integer threshold for a valid match (default is 80).

Returns

A matched company name if the similarity score exceeds the threshold; otherwise, None.

`modules.output.data_clean.`**`messy_df`**

`modules.output.data_clean.`**`messy_df`**

modules.output.data_export

This script connects to a PostgreSQL database, exports two tables as CSVs, cleans and merges them based on company identifiers, and saves the merged output.

Steps: 1. Connects to PostgreSQL using environment variables via *dotenv*. 2. Exports:

- *csr_reporting.company_reports* to *company_reports.csv*
- *csr_reporting.company_static* to *company_static.csv*

3. Cleans *symbol* and *security* columns to ensure consistent casing and whitespace.
4. Merges static metadata (*company_static*) with dynamic reports (*company_reports*) on symbol + security.
5. Drops duplicates and saves the final merged data as *company_information.csv*.

Attributes

conn
query1
df1
query2
df2
df1_cleaned
df1_cleaned
df1_subset
merged_df
df

Module Contents

`modules.output.data_export.`**`conn`**

`modules.output.data_export.`**`query1 = 'SELECT * FROM csr_reporting.company_reports'`**

`modules.output.data_export.`**`df1 = None`**

`modules.output.data_export.`**`query2 = 'SELECT * FROM csr_reporting.company_static'`** `modules.output.data_export.df2 = None`

`modules.output.data_export.df1_cleaned`

`modules.output.data_export.df1_cleaned`

`modules.output.data_export.df1_subset`

`modules.output.data_export.merged_df`

`modules.output.data_export.df`

modules.output.data_storage

This script ensures the existence of a PostgreSQL table *csr_reporting.company_indicators*, creates it if missing, and then loads data from *company_indicators.csv* into the table.

Workflow: 1. Connects to the PostgreSQL database using environment variables from *.env*. 2. Checks if the schema and table exist using *information_schema*. 3. Creates the schema and table if they don't exist. 4. Reads the cleaned CSR indicator data from a CSV file. 5. Standardizes column names for SQL compatibility. 6. Uses SQLAlchemy to insert the DataFrame into PostgreSQL using *to_sql*.

Attributes

schema_name
table_name
check_table_exists_query
conn
cur
table_exists
engine
csv_path
df

Module Contents

`modules.output.data_storage.schema_name = 'csr_reporting'`

`modules.output.data_storage.table_name = 'company_indicators'`

`modules.output.data_storage.check_table_exists_query = Multiline-String`

```
"""
SELECT EXISTS (
    SELECT 1
```

(continues on next page)

(continued from previous page)

```
    FROM information_schema.tables
    WHERE table_schema = 'csr_reporting'
    AND table_name = 'company_indicators'
);
"""
```

modules.output.data_storage.**conn**

modules.output.data_storage.**cur**

modules.output.data_storage.**table_exists**

modules.output.data_storage.**engine = None**

modules.output.data_storage.**csv_path = 'company_indicators.csv'**

modules.output.data_storage.**df = None**

modules.output.script_purposes

4.2 generate_data_catalogue

4.2.1 Attributes

config_path

4.2.2 Functions

load_indicator_config(→ list)
generate_catalogue_and_dictionary(config_path, output_dir)

4.2.3 Module Contents

generate_data_catalogue.**load_indicator_config**(*config_path: pathlib.Path*) → list

generate_data_catalogue.**generate_catalogue_and_dictionary**(*config_path: pathlib.Path*, *output_dir: pathlib.Path*)

generate_data_catalogue.**config_path**

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX