Package 'airqualitytr'

November 3, 2025

Title Turkish Air Quality Data Access Tools

Version 0.1.0

Date 2025-10-27

Description Provides functions to download and explore air quality monitoring data from the Turkish Ministry of Environment, Urbanization and Climate Change. Enables researchers to access hourly and daily measurements of PM10, PM2.5, SO2, CO, NO2, NOX, NO, and O3 across monitoring stations throughout Turkey.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL https://github.com/emraher/airqualitytr

BugReports https://github.com/emraher/airqualitytr/issues

Depends R (>= 4.1.0)

Imports dplyr, httr, jsonlite, rlang, rvest, tibble, tidyr, tidyselect (>= 1.1.0), xml2 (>= 1.3.0)

Suggests cli, future, furrr, ggplot2, glue, httptest, knitr, lubridate, progressr, purrr, rmarkdown, scales, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Emrah Er [aut, cre] (ORCID: <https://orcid.org/0000-0001-9909-7479>)

Maintainer Emrah Er <eer@eremrah.com>

Contents

`airqualitytr-package`

airqualitytr: Turkish Air Quality Data Access Tools

Description

Provides functions to download and explore air quality monitoring data from the Turkish Ministry of Environment, Urbanization and Climate Change. Enables researchers to access hourly and daily measurements of PM10, PM2.5, SO2, CO, NO2, NOX, NO, and O3 across monitoring stations throughout Turkey.

The airqualitytr package provides functions to download and explore air quality monitoring data from the Turkish Ministry of Environment, Urbanization and Climate Change. It enables researchers to access hourly and daily measurements of PM10, PM2.5, SO2, CO, NO2, NOX, NO, and O3 across 300+ monitoring stations throughout Turkey.

Main Functions

- `download_air_quality_data()`: Download air quality data for specific stations
- `bulk_download_air_quality_data()`: Download data for multiple stations
- `list_stations()`: List all available monitoring stations
- `list_cities()`: List all cities with monitoring stations
- `list_parameters()`: List all available pollutant parameters
- `quality_report()`: Generate data quality assessment reports

Vignettes

Learn more about using airqualitytr:

- `vignette("getting-started", package = "airqualitytr")` - Introduction and basic usage
- `vignette("parallel-processing", package = "airqualitytr")` - Speed up downloads with parallel processing
- `vignette("data-quality", package = "airqualitytr")` - Assess data quality and completeness
- `vignette("long-term-data", package = "airqualitytr")` - Download and manage long-term datasets

Package Information

- GitHub: https://github.com/emraher/airqualitytr
- Report bugs: https://github.com/emraher/airqualitytr/issues

Author(s)

Maintainer: Emrah Er <eer@eremrah.com> (ORCID)

See Also

Useful links:

- https://github.com/emraher/airqualitytr
- Report bugs at https://github.com/emraher/airqualitytr/issues

---

`assess_completeness`

Assess Data Completeness

Description

Calculates completeness metrics for air quality data, including percentage of missing values, gap identification, and temporal coverage.

Usage

```
assess_completeness(data, by_parameter = TRUE, by_station = FALSE)
```

Arguments

`data`	A tibble containing air quality data with columns: time, parameter, value
`by_parameter`	Logical. If TRUE, calculates metrics separately for each parameter. Default is TRUE.
`by_station`	Logical. If TRUE and station_id column exists, calculates metrics separately for each station. Default is FALSE.

Value

A tibble containing completeness metrics:

parameter Parameter name (if by_parameter = TRUE)

station_id Station ID (if by_station = TRUE)

total_records Total number of records

missing_records Number of missing (NA) values

valid_records Number of valid (non-NA) values

completeness_pct Percentage of valid records

start_date First timestamp in data

end_date Last timestamp in data

expected_records Expected number of records based on frequency

coverage_pct Actual vs expected records percentage

Examples

```
## Not run:
data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10", "NO2", "O3"),
  start_datetime = "2025-03-01 00:00",
  end_datetime = "2025-03-31 23:59",
  frequency = "hourly"
)

# Get completeness summary
completeness <- assess_completeness(data)
print(completeness)

# By station (if multiple stations)
completeness_by_station <- assess_completeness(data, by_station = TRUE)

## End(Not run)
```

---

bulk_download_air_quality_data

Bulk Download Air Quality Data for All Stations

Description

Downloads time series air quality data (e.g., hourly measurements) for all available stations over a specified date range and for the given set of measurement parameters. This function retrieves station metadata via `list_stations()` and the available cities via `list_cities()`, then for each station downloads the measurement data using `download_air_quality_data()`.

Usage

```
bulk_download_air_quality_data(
  start_datetime,
  end_datetime,
  parameters = c("PM10", "CO", "NO2"),
  frequency = "hourly",
  show_progress = TRUE,
  parallel = FALSE,
  workers = NULL
)
```

Arguments

`start_datetime`	Start date-time in "YYYY-MM-DD HH:MM" format.
`end_datetime`	End date-time in "YYYY-MM-DD HH:MM" format.
`parameters`	A character vector of pollutant parameters (e.g., `c("PM10", "CO", "NO2")`). Default is `c("PM10", "CO", "NO2")`.
`frequency`	Either `"hourly"` or `"daily"`. Default is `"hourly"`.

`show_progress`
: Logical. If `TRUE` and the `cli` package is available, displays a progress bar. Default is `TRUE`.

`parallel`
: Logical. If `TRUE`, downloads will be performed in parallel using the `future` and `furrr` packages. You must set up a `future` plan before using this option (e.g., `future::plan(future::multisession)`). Default is `FALSE`.

`workers`
: Integer. Number of parallel workers to use when `parallel = TRUE`. If `NULL` (default), uses the number of workers specified in the active `future` plan.

Value

A tibble containing the combined time series data from all stations. The tibble includes:

time POSIXct timestamp of the measurement.

station_id Station identifier.

city_name City name for the station.

station_name Station name.

parameter Measurement parameter (e.g., "PM10").

value The corresponding measurement value.

... Additional metadata columns in snake_case (city_id, area_type, etc.).

Examples

```
# Download one hour of data for all stations (small example)
bulk_data <- bulk_download_air_quality_data(
  start_datetime = "2024-03-01 00:00",
  end_datetime = "2024-03-01 01:00",
  parameters = c("PM10"),
  frequency = "hourly"
)
```

---

check_invalid_values

Check for Invalid Values in Air Quality Data

Description

Identifies invalid, impossible, or suspicious values in air quality measurements. This includes negative values, extreme outliers, and measurements that exceed physically plausible limits.

Usage

```
check_invalid_values(data, parameter_limits = NULL, outlier_threshold = 5)
```

Arguments

`data`	A tibble containing air quality data with columns: time, parameter, value
`parameter_limits`	A named list of maximum plausible values for each parameter. If NULL (default), uses standard limits based on measurement ranges.
`outlier_threshold`	Number of standard deviations beyond which a value is considered an outlier. Default is 5.

Value

A tibble with the same structure as input data, plus additional columns:

is_negative Logical indicating if value is negative

exceeds_limit Logical indicating if value exceeds plausible maximum

is_outlier Logical indicating if value is a statistical outlier

is_valid Logical indicating if value passes all checks (TRUE = valid)

quality_flag Character describing the issue (if any)

Examples

```
## Not run:
data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10", "NO2"),
  start_datetime = "2025-03-01 00:00",
  end_datetime = "2025-03-07 23:59"
)

# Check data quality
checked_data <- check_invalid_values(data)

# See flagged values
flagged <- checked_data |> dplyr::filter(!is_valid)

## End(Not run)
```

`download_air_quality_data` *Download Air Quality Data with Additional Objects*

Description

Downloads time series air quality data for a specified station by simulating a form POST request to the Turkish Ministry's air quality data service. In addition to retrieving the raw time series (from the "Data" object), this function attempts to extract any other objects present in the JSON response (such as "Summaries", "Monitors", "StationIds", and "Parameters"). Station and city names are automatically retrieved from the metadata.

Usage

```
download_air_quality_data(
  station_id,
  parameters,
  start_datetime,
  end_datetime,
  frequency = c("hourly", "daily"),
  return_all = FALSE
)
```

Arguments

station_id	A character string with the station's unique identifier.
parameters	A character vector of pollutant parameters (e.g., c("PM10", "CO", "NO2")).
start_datetime	Start date-time in "YYYY-MM-DD HH:MM" format.
end_datetime	End date-time in "YYYY-MM-DD HH:MM" format.
frequency	Either "hourly" or "daily". Default is "hourly".
return_all	Logical. If TRUE (default), a named list with elements data, summaries, monitors, stations, and options is returned. If FALSE, only the tidied time series data is returned.

Value

If return_all = TRUE, a named list with the following elements:

data A tibble with the time series measurements (columns include time, station_id, city_name, station_name, parameter, and value). All column names are in snake_case format.

summaries A tibble with summary data (if available; otherwise an empty tibble). Column names are in snake_case format.

monitors A tibble with monitors metadata (if available; otherwise an empty tibble). Column names are in snake_case format.

stations A tibble with station metadata (if available; otherwise an empty tibble). Column names are in snake_case format.

options A tibble with available parameter/option settings (if available; otherwise empty). Column names are in snake_case format.

If return_all = FALSE, only the data tibble is returned with snake_case column names.

Examples

```
# Download one week of hourly PM10 data
data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10"),
  start_datetime = "2024-03-01 00:00",
  end_datetime = "2024-03-07 23:59",
  frequency = "hourly"
)
head(data)
```

```
# Download multiple parameters
multi_data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10", "NO2", "O3"),
  start_datetime = "2024-03-01 00:00",
  end_datetime = "2024-03-07 23:59",
  frequency = "hourly"
)
```

`download_all_data_for_station` *Download All Available Data for a Given Station*

Description

Downloads all available air quality measurements for a specified station by iterating over manageable date ranges. The function automatically retrieves station metadata from the `list_stations()` function and consolidates the chunked downloads into a single tibble.

Usage

```
download_all_data_for_station(
  station_id,
  parameters,
  frequency = c("hourly", "daily"),
  show_progress = TRUE,
  chunk_size = "1 year",
  parallel = FALSE,
  workers = NULL
)
```

Arguments

`station_id`	A character string representing the station's unique identifier.
`parameters`	A character vector of pollutant parameters (e.g., `c("PM10", "CO", "NO2")`).
`frequency`	Either `"hourly"` or `"daily"`. Default is `"hourly"`.
`show_progress`	Logical. If `TRUE`, displays informative messages during download. Default is `TRUE`.
`chunk_size`	A character string specifying the size of each download window passed to `seq.POSIXt()` (e.g., `"1 month"`, `"1 year"`). Defaults to `"1 year"`.
`parallel`	Logical. If `TRUE`, downloads will be performed in parallel using the `future` and `furrr` packages. You must set up a `future` plan before using this option (e.g., `future::plan(future::multisession)`). Default is `FALSE`.
`workers`	Integer. Number of parallel workers to use when `parallel = TRUE`. If `NULL` (default), uses the number of workers specified in the active `future` plan.

Value

A tibble containing the available time series air quality data for the given station. The tibble includes columns such as time, station_id, station_name, parameter, value, and other metadata columns retrieved from station information. All column names are in snake_case format.

Examples

```
# Download all available data for a station
all_data <- download_all_data_for_station(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10", "NO2"),
  frequency = "hourly",
  chunk_size = "3 months"
)
```

---

format_api_error *Enhanced Error Message Formatter*

Description

Formats API errors with helpful context and troubleshooting suggestions.

Usage

```
format_api_error(error_type, details = NULL, http_status = NULL)
```

Arguments

error_type	Character. Type of error (e.g., "http", "network", "timeout")
details	Additional error details
http_status	HTTP status code (if applicable)

Value

Formatted error message with suggestions

---

get_api_status *Get API Status Information*

Description

Retrieves current status information about the Turkish Ministry of Environment API, including response times and availability.

Usage

```
get_api_status()
```

Value

A list containing API status metrics

Examples

```
## Not run:
status <- get_api_status()
print(status)

## End(Not run)
```

`identify_gaps` *Identify Time Gaps in Air Quality Data*

Description

Finds gaps in time series data where measurements are missing for extended periods.

Usage

```
identify_gaps(data, min_gap_hours = 3, by_parameter = TRUE)
```

Arguments

`data`	A tibble containing air quality data with columns: time, parameter, value
`min_gap_hours`	Minimum gap duration in hours to report. Default is 3 hours.
`by_parameter`	Logical. If TRUE, identifies gaps separately for each parameter. Default is TRUE.

Value

A tibble containing identified gaps:

parameter Parameter name (if by_parameter = TRUE)

gap_start Start time of gap

gap_end End time of gap

gap_hours Duration of gap in hours

missing_records Estimated number of missing records

Examples

```
## Not run:
data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10"),
  start_datetime = "2025-03-01 00:00",
  end_datetime = "2025-03-31 23:59",
  frequency = "hourly"
)
```

```
# Find gaps of 6+ hours
gaps <- identify_gaps(data, min_gap_hours = 6)
print(gaps)

## End(Not run)
```

`list_cities` *List Available Cities*

Description

Retrieves a list of cities available for filtering air quality data.

Usage

```
list_cities()
```

Value

A tibble with columns `city_id` and `city_name`.

Examples

```
# Get all available cities
cities <- list_cities()
head(cities)
```

`list_parameters` *List Available Measurement Parameters*

Description

Retrieves a list of measurement parameters available for querying.

Usage

```
list_parameters()
```

Value

A tibble with details on available parameters.

Examples

```
# Get all available parameters
params <- list_parameters()
print(params)
```

list_stations *List Available Stations*

Description

Retrieves a metadata table of available stations from the air quality service.

Usage

```
list_stations()
```

Value

A tibble of station metadata.

Examples

```
# Get all available stations
stations <- list_stations()
head(stations)

# Filter stations by city
library(dplyr)
ankara_stations <- stations |>
  filter(grepl("Ankara", city_name, ignore.case = TRUE))
```

print.air_quality_report *Print Method for Air Quality Report*

Description

Print Method for Air Quality Report

Usage

```
## S3 method for class 'air_quality_report'
print(x, ...)
```

Arguments

x	An air quality report object from quality_report()
...	Additional arguments (not used)

print.api_diagnostic *Print Method for API Diagnostic*

Description

Print Method for API Diagnostic

Usage

```
## S3 method for class 'api_diagnostic'
print(x, ...)
```

Arguments

x	An API diagnostic object from test_api_connection()
...	Additional arguments (not used)

quality_report *Generate Data Quality Report*

Description

Produces a comprehensive data quality report including validity checks, completeness assessment, and gap identification.

Usage

```
quality_report(data, outlier_threshold = 5, min_gap_hours = 3)
```

Arguments

data	A tibble containing air quality data with columns: time, parameter, value
outlier_threshold	Number of standard deviations for outlier detection. Default is 5.
min_gap_hours	Minimum gap duration in hours to report. Default is 3.

Value

A list containing:

summary Overall data quality summary

completeness Completeness metrics by parameter

invalid_values Summary of invalid value counts by type

gaps Identified time gaps

flagged_data Data with quality flags (only invalid records)

Examples

```
## Not run:
data <- download_air_quality_data(
  station_id = "468478b7-ace5-4bd3-b89a-a9c1c2e53080",
  parameters = c("PM10", "NO2", "O3"),
  start_datetime = "2025-03-01 00:00",
  end_datetime = "2025-03-31 23:59",
  frequency = "hourly"
)

# Generate quality report
report <- quality_report(data)

# View summary
print(report$summary)

# View completeness by parameter
print(report$completeness)

# View flagged values
print(report$flagged_data)

## End(Not run)
```

`test_api_connection` *Test API Connection*

Description

Tests the connection to the Turkish Ministry of Environment API and verifies that all required endpoints are accessible. Useful for troubleshooting connection issues before attempting large downloads.

Usage

```
test_api_connection(timeout = 10, verbose = TRUE)
```

Arguments

`timeout`	Timeout in seconds for each test. Default is 10.
`verbose`	Logical. If TRUE, prints detailed diagnostic information. Default is TRUE.

Value

A list containing:

overall_status Character: "success", "partial", or "failure"

tests Tibble with results for each endpoint test

recommendations Character vector of troubleshooting recommendations

Examples

```
## Not run:
# Test API connection
test_result <- test_api_connection()

# Quick check
if (test_result$overall_status == "success") {
  message("API is accessible")
}

# Detailed diagnostics
print(test_result$tests)

## End(Not run)
```