Package 'arpcPriceBasis'

October 19, 2025

Type Package

Title Tools for Price and basis by Commodity and County

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Creator Francis Tsiboe

Description Provides tools to download, harmonize, and analyze agricultural price and basis data at the commodity and county level. Integrates multiple data sources (including DTN ProphetX and USDA NASS) and applies econometric and spatial calibration techniques to support policy evaluation, risk management research, and farm-level decision tools. Includes vendor-neutral helpers to evaluate Excel RTD formulas (e.g., DTN ProphetX and Bloomberg) from R via COM automation, with retry logic to handle transient ``Wait'' tokens from data providers.

License GPL-3 + file LICENSE

URL https://github.com/you/arpcPriceBasis

BugReports https://github.com/you/arpcPriceBasis/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports data.table, stringr, xml2, urbnmapr, sf, GWmodel, sp, methods, readxl, rlang, tidygeocoder, zipcodeR, zoo, rexcelbridge, purrr, readr, utils

Remotes github::UrbanInstitute/urbnmapr, github::dylan-turner25/rfcip, github::omegahat/RDCOMClient, github::ftsiboe/rexcelbridge

Suggests knitr, rmarkdown, tibble, tidyr, dplyr, RDCOMClient, rvest, mockery, lubridate, withr, testthat (>= 3.0.0)

LazyData true

Contents

`arpcPriceBasis_control` *Create control parameters for* `arpcPriceBasis`

Description

Centralized knobs for batching and Excel/ProphetX evaluation behavior.

Usage

```
arpcPriceBasis_control(
  dtn_prophetX_query_limit = 500L,
  continuous_integration_session = FALSE,
  excel_eval_max_attempts = 2L,
  excel_eval_retry_sleep = 1
)
```

Arguments

`dtn_prophetX_query_limit`
: integer(1). Max rows per DTN ProphetX batch (used for chunking). Must be > 0. Default: `3500L`.

`continuous_integration_session`
: logical(1). If `TRUE`, skip Excel/ ProphetX evaluation and other side effects; used for CI runs. Default: `FALSE`.

excel_eval_max_attempts
: integer(1). Max retry attempts when evaluating Excel RTD queries (per chunk). Must be >= 1. Default: 2L.

excel_eval_retry_sleep
: numeric(1). Sleep (seconds) between Excel RTD retry attempts. Must be >= 0. Default: 1.

Value

Named list of control values (validated and sanitized).

`bloomberg_bds_formula` *Build a Bloomberg BDS Excel formula (bulk/descriptor data)*

Description

Constructs a Bloomberg BDS formula string for bulk fields (returns a spilled table). Use with `rexcelbridge::rb_eval_single` to read the result.

Usage

```
bloomberg_bds_formula(security, field, overrides = NULL)
```

Arguments

security	Character scalar, e.g., "IBM US Equity" or an index/ticker.
field	Character scalar bulk field (e.g., "INDX_MEMBERS").
overrides	Optional named vector/list of overrides.

Value

A character string like:

```
=BDS("SPX Index","INDX_MEMBERS")
```

`build_dtn_queries` *Build ProphetX query rows and split for batching*

Description

Build ProphetX query rows and split for batching

Usage

```
build_dtn_queries(
  symbols,
  fields,
  target_dates = Sys.Date() - 1,
  time_scale = "daily",
  control = arpcPriceBasis_control()
)
```

Arguments

symbols	Filtered symbol table.
fields	Character vector of fields.
target_dates	Vector of query dates.
time_scale	ProphetX time_scale ("Daily","Weekly","Monthly").
control	a list of control parameters -> dtn_prophetX_query_limit Max rows per chunk.

Value

List of data.frames (chunks) with columns: symbol metadata, date, field, query.

See Also

Other DTN: clean_dtn_excel_file(), download_dtn_weekly_prices(), dtn_prophetX_formula(), fetch_dtn_elevators(), get_dtn_price(), get_dtn_price_by_symbol()

---

clean_dtn_excel_file *Clean a DTN Excel extract into a tidy long table*

Description

Reads a DTN elevator/basis Excel file and returns a tidy data frame with symbol metadata, a proper Date, OHLC fields, and fields parsed from the file name (resolution, commodity, variable, type). All worksheets are read; each sheet name is treated as a state abbreviation.

Usage

```
clean_dtn_excel_file(dtn_file_path)
```

Arguments

dtn_file_path	Character scalar. Path to a DTN Excel file to be cleaned.

Details

For each sheet:

1. read & transpose; 2) forward-fill id columns; 3) iterate over "draw" columns (4:ncol) to build OHLC blocks; 4) widen with pivot_wider() using a duplicate-safe aggregator; 5) keep rows with Close present; 6) standardize to lower case; 7) convert Excel-serial dates safely; 8) stack; 9) pivot to long;
2. parse filename into resolution/commodity/variable/type.

Value

A data.frame with at least: symbol, description, date (Date), field in ("close","high","low","open"), value (numeric), resolution, commodity, variable, type, state_abbreviation.

See Also

Other DTN: build_dtn_queries(), download_dtn_weekly_prices(), dtn_prophetX_formula(), fetch_dtn_elevators(), get_dtn_price(), get_dtn_price_by_symbol()

clear_arpcPriceBasis_cache *Clear the package cache of downloaded data files*

Description

Deletes the entire cache directory used by the **arpcPriceBasis** package to store downloaded data files. Useful if you need to force re-download of data, or free up disk space.

Usage

```
clear_arpcPriceBasis_cache()
```

Value

Invisibly returns `NULL`. A message is printed indicating which directory was cleared.

Examples

```
## Not run:
# Remove all cached data files so they will be re-downloaded on next use
clear_arpcCost_cache()

## End(Not run)
```

downloaded_nass_large_datasets *Download and cache USDA NASS Quick Stats large dataset files*

Description

`downloaded_nass_large_datasets()` retrieves a Quick Stats file from the USDA National Agricultural Statistics Service (NASS) https://www.nass.usda.gov/datasets/ page and saves it locally. If the file is already present in the target directory, it is not re-downloaded.

Usage

```
downloaded_nass_large_datasets(
  large_datasets,
  dir_dest = "./data-raw/fastscratch/nass/"
)
```

Arguments

`large_datasets`	`character list` The base name of the Quick Stats file to download. For example, use `"crops"` to fetch `qs.crops_YYYYMMDD.txt.gz` or include `"census2022"` (e.g. `"census2022"`) to fetch the gzipped 2022 census version (`qs.census2022.txt.gz`). any of: "census2002","census2007","census2012","census2017","census2022", "census2007zipcode","census2017zipcode", "animals_products","crops","demographics","economic
`dir_dest`	`character(1)` Path to a directory where downloaded files will be stored. Defaults to `"./data-raw/fastscratch/nass/"`.

Details

1. Prepends "qs." to the provided large_dataset. If large_dataset contains "census", appends ".txt.gz", otherwise NULL.
2. Ensures dir_dest exists (creates it if needed).
3. Scrapes the NASS datasets page (https://www.nass.usda.gov/datasets/) for links ending in .txt.gz.
4. Downloads the matching file into dir_dest if not already present.

Value

Invisibly returns the normalized file large_dataset (e.g. "qs.crops_YYYYMMDD.txt.gz" or "qs.censusYYYY.txt.gz") that was downloaded or already present.

See Also

Other USDA NASS Quick Stats: process_nass_dataset()

Examples

```
## Not run:
# Download the 'crops' dataset if not already cached:
downloaded_nass_large_datasets(large_dataset = "crops")

# Download the 2022 census version:
downloaded_nass_large_datasets(large_dataset = "census2022",
dir_dest = "./data-raw/fastscratch/nass/")

## End(Not run)
```

---

download_dtn_weekly_prices

Download Weekly DTN ProphetX Price Data by Crop and Market

Description

Generates a full worklist of crop-date-market combinations over a specified date range and downloads daily DTN ProphetX price data for each combination. The results are saved as individual .rds files organized by ISO year/week.

Usage

```
download_dtn_weekly_prices(
  crop = NULL,
  market = c("forward", "spot"),
  start_date = Sys.Date() - 14,
  end_date = Sys.Date() - 7,
  output_directory = NULL,
  work_station = 1L,
  number_of_stations = 1L,
  control = arpcPriceBasis_control()
)
```

Arguments

`crop`	Character (length 1 or vector). **Substring matched** (case-insensitive) against internal dataset `dtnEelevators$commodity` (e.g., "soy", "corn", "wheat"). If `crop = NULL`, all unique `commodity` values are used.
`market`	Character vector of market types. Defaults to `c("forward","spot")`. If `market = NULL`, all unique `dtnEelevators$market_type` values are used.
`start_date, end_date`	Date objects defining the inclusive range.
`output_directory`	Character path to save downloaded files. If `NULL`, the user is prompted interactively to enter a directory.
`work_station`	Integer vector of workstation ID that should run this shard.
`number_of_stations`	Integer vector for total number of workstations used to shard the workload (default: `1L` if not provided).
`control`	List of control parameters (see `arpcPriceBasis_control()`), including: • `continuous_integration_session` (logical) to short-circuit downloads • batching limits for query chunking (e.g., `dtn_prophetX_query_limit`)

Value

A character message summarizing attempted, succeeded, and failed downloads.

See Also

Other DTN: `build_dtn_queries()`, `clean_dtn_excel_file()`, `dtn_prophetX_formula()`, `fetch_dtn_elevators()`, `get_dtn_price()`, `get_dtn_price_by_symbol()`

`dtnEelevators` *Simulator Helper Datasets*

Description

A combined dataset for dtnEelevators

Usage

```
data(dtnEelevators)
```

Format

A data frame with 15722 rows and 4 columns covering Inf–Inf.

Source

Manual download from DTN

dtnRootSymbols *dtnRootSymbols*

Description

A combined dataset for dtnRootSymbols

Usage

```
data(dtnRootSymbols)
```

Format

A data frame with 47 rows and 5 columns covering Inf–Inf.

Source

DTN website

dtn_prophetX_formula *Build a DTN ProphetX AIHIST RTD formula*

Description

Constructs a single-cell Excel formula string for the DTN ProphetX `AIHIST` RTD function. This can be passed to `rb_eval_single()` to evaluate historical data directly from Excel via COM automation.

Usage

```
dtn_prophetX_formula(
  symbol,
  field,
  time_scale = "Daily",
  date = Sys.Date() - 1
)
```

Arguments

`symbol`	Character. ProphetX instrument symbol (e.g., `"BEANS.20254.B"`).
`field`	Character. Field(s) to request, such as `"Open"`, `"High"`, `"Low"`, `"Close"`, or `"Description"`.
`time_scale`	Character. Time scale such as `"Daily"`, `"Weekly"`, or `"Monthly"`.
`date`	Date or string convertible to Date. The Excel serial number is computed relative to 1899-12-30.

Details

- The returned string is not evaluated in R; it must be written into an Excel cell using `rb_eval_single()`.
- The `date` argument is converted to an Excel serial (days since 1899-12-30).
- Wrapping with `IFERROR(...,0)` ensures Excel returns `0` if the RTD call fails.

Value

A character string containing a valid Excel formula of the form:

```
=IFERROR(RTD("prophetx.rtdserver","","AIHIST",symbol,time_scale,"1",date,"",field,"XD"),0)
```

See Also

Other DTN: `build_dtn_queries()`, `clean_dtn_excel_file()`, `download_dtn_weekly_prices()`, `fetch_dtn_elevators()`, `get_dtn_price()`, `get_dtn_price_by_symbol()`

---

estimate_gwss_by_county

Estimate geographically weighted summary statistics (GWSS) for counties

---

Description

Computes Geographically Weighted Summary Statistics (GWSS) for a scalar, county-level variable observed in a subset of counties, then evaluates the statistics at **all** county locations (points-on-surface). This is useful for spatial smoothing and gap-filling (imputation) when some counties are missing observations.

Usage

```
estimate_gwss_by_county(
  data,
  fip_col,
  variable,
  distance_metric = "Euclidean",
  kernel = "gaussian",
  target_crs = 5070,
  draw_rate = 0.5,
  approach = "CV",
  adaptive = TRUE
)
```

Arguments

`data`	A `data.frame`/`data.table` with at least `fip_col` and `variable`.
`fip_col`	Character. Name of the county ID column in `data`; copied to `"county_fips"`.
`variable`	Character. Name of the numeric column in `data` to summarize.
`distance_metric`	Character. One of `gw_distance_metric_names()`. Default: `"Euclidean"`.

kernel	Character. One of "gaussian", "exponential", "bisquare", "boxcar", "tricube". Default: "gaussian".
target_crs	Integer EPSG used to project county geometries when longlat = FALSE. Default: **5070** (NAD83 / CONUS Albers, meters).
draw_rate	Numeric in (0, 1]. Fraction of observed counties used during bandwidth cross-validation. Default: **0.5** (50%).
approach	Character. Bandwidth selection approach passed to GWmodel::bw.gwr(). One of "CV", "AIC", "AICc". Default: "CV".
adaptive	Logical. Use adaptive (nearest-neighbour count) bandwidth instead of fixed distance. Default: TRUE.

Details

The function:

1. pulls U.S. counties from **urbnmapr** and projects to a suitable CRS;
2. copies data[[fip_col]] into "county_fips" and joins to the map;
3. fits GWSS using **only observed counties** (points) and selects an adaptive bandwidth by cross-validation on a random subsample sized by draw_rate;
4. evaluates GWSS at **all counties** and returns local summaries keyed by county_fips.

Inputs: data must contain a county identifier column referenced by fip_col and a numeric column referenced by variable. Internally, a column "county_fips" is created for joining with urbnmapr::get_urbn_map("counties").

Distance metric (distance_metric) defines (p, theta, longlat) for GWmodel::gw.dist(). Use gw_distance_metric_names() to list options. If longlat = TRUE (e.g., "Great Circle"), counties are transformed to EPSG:4326; otherwise to target_crs (default 5070, meters).

Bandwidth selection: CV is run on a uniform random subsample of size ceiling(draw_rate * n_obs), bounded to [5, n_obs - 1]. Call set.seed() beforehand for reproducibility. Returns NULL (with a message) if fewer than 5 counties have finite values.

Value

A data.table of GW summary statistics for **all counties**, with a county_fips column for merging back to polygons. Column names follow **GWmodel** conventions for the internal value variable (created from variable), e.g. value_LM, value_LSD, value_LCV, value_LSKe, value_LSSke, etc. Attributes attached: "bandwidth", "distance_params", "kernel", "approach", "adaptive". Returns NULL when there are < 5 observed counties.

Imputation workflow

```
set.seed(123)
gw <- estimate_gwss_by_county(
  data = my_data, fip_col = "county_fips", variable = "my_var"
)
sf_out <- counties_sf |>
  dplyr::left_join(gw[, c("county_fips","value_LM")], by = "county_fips") |>
 dplyr::mutate(my_var_imputed = dplyr::if_else(is.finite(my_var), my_var, value_LM))
```

`fetch_dtn_elevators` *Build a DTN elevator directory (IDs, location, and metadata) for selected crops*

Description

Generates and queries a candidate list of DTN elevator symbols across a user-specified range of elevator IDs, for both spot and forward markets, and returns a tidy directory of **active** elevators (i.e., those for which DTN returns valid coordinates) with name, city, state, ZIP, latitude, longitude, and description.

Usage

```
fetch_dtn_elevators(
  crop = NULL,
  dtn_elevator_id_range = 1:100000,
  control = arpcPriceBasis_control()
)
```

Arguments

`crop`	Character vector or length-1 string. **Substring matched** (case-insensitive) against internal dataset `dtnRootSymbols$commodity` (e.g., "soy", "corn", "wheat"). If `crop = NULL`, all unique `commodity` values are used.
`dtn_elevator_id_range`	Integer vector of DTN elevator ID candidates to try (default `1:100000`). These are combined with crop roots and market type to generate DTN symbols.
`control`	List of control parameters (see `arpcPriceBasis_control()`), including: • `continuous_integration_session` (logical) to short-circuit downloads. • `dtn_prophetX_query_limit` (integer) batching limit for Excel bridge queries. • `excel_eval_max_attempts`, `excel_eval_retry_sleep` retry controls.

Details

Symbols are constructed as `<dtn_commodity_root>.<ID>` for forward markets and `<dtn_commodity_root>$.<ID>` for spot markets (matching ProphetX help). The function first probes `DTNElevatorLongitude` only, to filter to symbols that exist, then fetches full metadata for that filtered set.

Value

A data.table with one row per discovered elevator and columns:

- `symbol` (character)
- `dtn_commodity_root` (character)
- `commodity` (character; from internal mapping)
- `market` (factor: `"spot"` or `"forward"`)
- `dtn_elevator_id` (integer)
- `DTNElevatorName`, `DTNElevatorCity`, `DTNElevatorState`, `DTNElevatorZip` (character)
- `DTNElevatorLatitude`, `DTNElevatorLongitude` (numeric)

- description (character; DTN field)

Returns an empty data.table when continuous_integration_session = TRUE or when no elevators are found.

See Also

Other DTN: build_dtn_queries(), clean_dtn_excel_file(), download_dtn_weekly_prices(), dtn_prophetX_formula(), get_dtn_price(), get_dtn_price_by_symbol()

---

first_trading_days *First trading (Mon-Fri) day for each month*

Description

First trading (Mon-Fri) day for each month

Usage

```
first_trading_days(x)
```

Arguments

x	Date vector.

Value

Sorted unique first trading days.

---

geocode_locations *Geocode location names (ZIP-centroid first, optional Census fallback)*

Description

Adds latitude and longitude coordinates to a dataset that contains a column (e.g., location_name) with address-like text such as "AGREX, MOBILE, AL 36602". The function first uses offline ZIP centroid coordinates from **zipcodeR**, and can optionally fall back to online geocoding using the U.S. Census API via **tidygeocoder** (method = "census").

Usage

```
geocode_locations(
  data,
  location_col,
  use_fallback = FALSE,
  fallback_method = "census",
  respect_existing_state = TRUE
)
```

Arguments

data	A data.frame or data.table.
location_col	Column in data that contains location strings such as "AGREX, MOBILE, AL 36602". Can be specified as a bare name or a string.
use_fallback	Logical; if TRUE, perform fallback geocoding for rows without ZIP-centroid coordinates using the U.S. Census API. Default = FALSE.
fallback_method	Character; the geocoding provider passed to tidygeocoder::geocode(). Default = "census".
respect_existing_state	If TRUE and state_abbreviation exists,

Details

The function performs the following steps:

1. Extracts a 5-digit ZIP code from the specified location column.
2. Joins ZIP-level latitude and longitude from **zipcodeR**.
3. Marks matched rows with geocode_method = "zip_centroid".
4. If use_fallback = TRUE, uses the U.S. Census API through **tidygeocoder** for rows where ZIP-based coordinates are missing, and updates geocode_method = "census".

This hybrid approach ensures high speed and reproducibility for most locations (via ZIP lookup) and full U.S. coverage when combined with the Census fallback.

Value

A modified version of data with additional columns:

- zip - Extracted 5-digit ZIP code.
- lat, lon - Latitude and longitude.
- geocode_method - The geocoding source ("zip_centroid" or "census").

---

get_census_harvested_area

Retrieve and Aggregate NASS Harvested Area Data

Description

Extracts, filters, and aggregates harvested area information for major field crops from pre-processed USDA NASS datasets (e.g., Census of Agriculture). Multiple crop-specific descriptors (e.g., "CORN, GRAIN - ACRES HARVESTED", "CORN, SILAGE - ACRES HARVESTED") are standardized into unified commodity groups, then aggregated to county level and (optionally) rolled up to state and national levels across selected years.

Usage

```
get_census_harvested_area(
  dir_source,
  census_years = c(2002, 2007, 2012, 2017, 2022),
  aggregation_level = c("STATE", "NATIONAL", "COUNTY"),
  map_crop_area = list(barley = "BARLEY - ACRES HARVESTED", corn =
   c("CORN, GRAIN - ACRES HARVESTED", "CORN, SILAGE - ACRES HARVESTED"), cotton =
    "COTTON - ACRES HARVESTED", oats = "OATS - ACRES HARVESTED", peanuts =
    "PEANUTS - ACRES HARVESTED", rice = "RICE - ACRES HARVESTED", sorghum =
    c("SORGHUM, GRAIN - ACRES HARVESTED", "SORGHUM, SILAGE - ACRES HARVESTED",
  "SORGHUM, SYRUP - ACRES HARVESTED"), soybeans = "SOYBEANS - ACRES HARVESTED", wheat =
    "WHEAT - ACRES HARVESTED")
)
```

Arguments

`dir_source`	Character. Path to the root directory containing the downloaded or pre-processed NASS datasets.
`census_years`	Numeric vector of Census of Agriculture years to include. Default: `c(2002, 2007, 2012, 2017, 2022)`.
`aggregation_level`	Character vector specifying one or more levels of geographic aggregation to include. One or more of `"COUNTY"`, `"STATE"`, `"NATIONAL"`. COUNTY data are always queried and used as the base for rollups.
`map_crop_area`	Named list mapping standardized crop names to their corresponding NASS `short_desc` values used for filtering harvested area items. Defaults cover barley, corn, cotton, oats, peanuts, rice, sorghum, soybeans, and wheat.

Details

For each requested census dataset, the function:

1. Calls `process_nass_dataset()` with filters for harvested area.
2. Normalizes and coerces numeric values.
3. Maps NASS descriptors to standardized `commodity_name`.
4. Aggregates to COUNTY, then (optionally) rolls up to STATE and NATIONAL.

Datasets that cannot be read/processed are skipped silently.

Value

A single `data.table` with aggregated harvested area and columns:

commodity_year Numeric; year of the commodity observation.

commodity_name Character; standardized crop identifier.

state_code State FIPS code (NA at NATIONAL level).

county_code County FIPS code (NA at STATE/NATIONAL levels).

value Total harvested area (acres).

agg_level One of `"COUNTY"`, `"STATE"`, `"NATIONAL"`.

`get_dtn_price` *Pull DTN Eelevators-average cash prices and reconstruct futures to compute basis*

Description

Builds and executes DTN ProphetX queries for one or more counties and returns a tidy table over a weekday-only date range for the requested crop(s) and market(s). Cash **Open/Close** are pulled directly from elevator symbols; a small sample of matching **basis (.B) instruments** (basis quoted in **cents**) is used to reconstruct **futures** in $/unit, from which **basis** is computed in **cents**.

Usage

```
get_dtn_price(
  crop,
  market = "forward",
  start_date = Sys.Date() - 1,
  end_date = Sys.Date() - 1,
  time_scale = "Daily",
  control = arpcPriceBasis_control()
)
```

Arguments

`crop`	Character (length 1 or vector). **Substring matched** (case-insensitive) against `dtnEelevators$commodity` (e.g., `"soy"`, `"corn"`, `"wheat"`).
`market`	Character (length 1 or vector). **Substring matched** (case-insensitive) against `dtnEelevators$market_type` (e.g., `"spot"`, `"forward"`).
`start_date, end_date`	Date. Inclusive range. Weekends are dropped by `get_target_dates()`. Defaults: yesterday to yesterday.
`time_scale`	Character. One of `"Daily"`, `"Weekly"`, `"Monthly"` (case-insensitive). Used for both target date construction and ProphetX queries.
`control`	List of control parameters (see `arpcPriceBasis_control()`), including: • `continuous_integration_session` (logical) to short-circuit downloads • batching limits for query chunking (e.g., `dtn_prophetX_query_limit`)

Details

- **Matching semantics**: Inputs are validated against the available **lowercased** values, but symbols are selected using **substring, case-insensitive** matching.
- **Futures reconstruction**: For each (`date`, `commodity`, `market_type`) group, we sample up to 10 matching symbols, fetch their `.B` basis series (in **cents**), and compute:

$$Futures = Cash - Basis/100$$

  Then we average across the sample for robustness.
- **Units**: `cash_price_*` and `futures_price_*` are in **$/unit**. `basis_*` are in **cents/unit**.
- **Geocoding**: `location_name` is geocoded (ZIP centroid first with optional Census fallback). Consider memoising `geocode_locations()` at package load for cross-call caching.

Value

A data.table with one row per (symbol, date) including:

- IDs: time_scale, commodity, symbol, description, market_type, date, parsed commodity_type, market_label, price_type, location_name
- Prices: cash_price_open, cash_price_close, futures_price_open, futures_price_close
- Basis (cents): basis_open, basis_close
- Geocodes: geocode_method, state_name, state_abbv, state_fips, county_fips, county_name, zip, lat, lon

Dependencies

Relies on dtnEelevators, get_target_dates(), get_dtn_price_by_symbol(), geocode_locations(), and DTN ProphetX connectivity.

See Also

Other DTN: build_dtn_queries(), clean_dtn_excel_file(), download_dtn_weekly_prices(), dtn_prophetX_formula(), fetch_dtn_elevators(), get_dtn_price_by_symbol()

---

get_dtn_price_by_symbol

Get DTN prices (Close/Open/Description) by symbol and date(s)

Description

Queries DTN for a set of symbols over one or more target_dates, returning a tidy data.table with **Close** prices and, when available, **Open** and **Description** fields. Internally, this:

1. builds per-symbol/day queries via build_dtn_queries(),
2. evaluates them with retry logic via rexcelbridge::rb_eval_with_retries(),
3. gathers additional fields (Open, Description),
4. coerces numeric fields, and
5. returns a wide table (one row per symbol/date/scale) with columns like open, close, and Description.

Usage

```
get_dtn_price_by_symbol(
  symbols,
  target_dates = Sys.Date() - 1,
  time_scale = "daily",
  control = arpcPriceBasis_control()
)
```

Arguments

symbols	Character vector of DTN symbols to request (e.g., "ZCZ25").
target_dates	Vector of dates (Date, POSIXct, or character coercible to Date) for which to request prices.
time_scale	Character scalar indicating the temporal scale for the query, typically "daily" (default). Passed through to build_dtn_queries().
control	List. Control parameters (see arpcPriceBasis_control()), including continuous_integration_s and batching limits. # CHANGE: documented control

Details

- **Control object**: If control$continuous_integration_session is TRUE, the function short-circuits and returns an empty data.table.
- **Additional fields**: After fetching "Close", the function also requests "Open" and "Description" by modifying the built queries. These are appended when available.
- **Type handling**: The function converts open and close to numeric and drops non-finite values. Rows with missing close are removed.
- **Empty results**: If nothing valid is returned (e.g., all NAs), the function returns data.table() (zero-row table).

Value

A data.table with one row per symbol/date/scale/contract (depending on what build_dtn_queries() emits), including at least:

- symbol (character)
- time_scale (character)
- target_date or similar date column emitted by build_dtn_queries()
- close (numeric) - Close price
- open (numeric, optional) - Open price if available
- Description (character, optional) - Contract/series description

Column names for IDs (e.g., date column) reflect whatever build_dtn_queries() provides; this function preserves them.

See Also

Other DTN: build_dtn_queries(), clean_dtn_excel_file(), download_dtn_weekly_prices(), dtn_prophetX_formula(), fetch_dtn_elevators(), get_dtn_price()

get_previous_weekdays_range *Get Previous Week (Monday-Friday) Range Based on Week of Year*

Description

Assumes weeks run Sunday-Saturday.

Usage

```
get_previous_weekdays_range(ref_date = Sys.Date())
```

Arguments

ref_date	A Date object or something coercible to Date (default = Sys.Date()).

Value

A named list with start_date (Monday) and end_date (Friday).

get_target_dates *Target dates by time scale (Daily/Weekly/Monthly)*

Description

Target dates by time scale (Daily/Weekly/Monthly)

Usage

```
get_target_dates(
  start_date = Sys.Date() - 7,
  end_date = Sys.Date(),
  time_scale = "Daily"
)
```

Arguments

start_date, end_date	Date range.
time_scale	One of "Daily","Weekly","Monthly".

Value

Vector of dates.

`gw_distance_metric_names` *List valid GW distance metric names*

Description

List valid GW distance metric names

Usage

```
gw_distance_metric_names()
```

Value

Character vector of valid preset names.

`gw_distance_metric_presets` *GW distance metric presets for GWmodel*

Description

Provides a curated set of distance metric presets (Minkowski family and great-circle) for **GWmodel**. Each preset specifies `(p, theta, longlat)` for `GWmodel::gw.dist()`.

Usage

```
gw_distance_metric_presets()
```

Value

A named list of presets, each entry a `list(p, theta, longlat)`.

`is_weekend` *Weekend flag (locale-agnostic)*

Description

Weekend flag (locale-agnostic)

Usage

```
is_weekend(x)
```

Arguments

`x`	Date vector.

Value

Logical vector; TRUE for weekend.

process_nass_dataset *Process a USDA NASS Quick Stats dataset by sector and statistic category*

Description

process_nass_dataset() downloads (if needed) and reads one or more NASS Quick Stats large datasets files for a given sector, filters the rows by the chosen statistic category plus any additional Quick Stats API parameters, converts and cleans the value column, aggregates it by taking its mean over all remaining grouping columns, and then renames that aggregated column to match the requested statistic.

Usage

```
process_nass_dataset(
  dir_source = "./data-raw/fastscratch/nass/",
  large_dataset,
  statisticcat_desc = NULL,
  nassqs_params = NULL
)
```

Arguments

dir_source	character(1) **Length 1.** Path to the directory where Quick Stats large datasets files are stored (and will be downloaded to via get_nass_large_datasets()). Defaults to "./data-raw/fastscratch/nass/".
large_dataset	character(1) The Quick Stats large_dataset to load (e.g. "crops"). one of: "census2002","census2007","census2012","census2017","census2022", "census2007zipcode","censu "animals_products","crops","demographics","economics","environmental"
statisticcat_desc	character(1) **Length 1.** The Quick Stats statisticcat_desc to filter on (e.g. "PRICE RECEIVED"). After aggregation, the resulting column of mean values will be renamed to gsub(" ", "_", statisticcat_desc).
nassqs_params	list or NULL A named list of additional Quick Stats API parameters to filter by (e.g. "domain_desc", "agg_level_desc", "year", etc.). Names must correspond to valid Quick Stats fields. If NULL (the default), only sector_desc + statisticcat_desc filtering is applied. Use rnassqs::nassqs_params() to list all valid parameter names.

Details

The full set of valid Quick Stats API parameter names can be retrieved with:

```
rnassqs::nassqs_params()
```

Value

A data.table where:

- All original columns have been lowercased.
- Rows have been filtered by nassqs_params.

- A value column has been converted to numeric (commas stripped), cleaned of non-finite entries, and then aggregated by mean over the remaining columns.
- That aggregated column is renamed to gsub(" ", "_", statisticcat_desc).
- Numeric code columns state_code, country_code, asd_code, plus commodity_year and commodity_name have been created.

See Also

- get_nass_large_datasets() for downloading the raw Quick Stats files

Other USDA NASS Quick Stats: downloaded_nass_large_datasets()

Examples

```
## Not run:
# National annual average price received for all CROPS in 2020:
dt1 <- process_nass_dataset(
  large_dataset       = "crops",
  statisticcat_desc = "PRICE RECEIVED",
  nassqs_params = list( agg_level_desc = "NATIONAL", year = 2020 ))

# State-level marketing-year average price for soybeans:
dt2 <- process_nass_dataset(
  large_dataset       = "crops",
  statisticcat_desc = "PRICE RECEIVED",
  nassqs_params     = list(
    agg_level_desc      = "STATE",
    short_desc          = "SOYBEANS - PRICE RECEIVED, MEASURED IN $ / BU",
    reference_period_desc = "MARKETING YEAR",
    freq_desc           = "ANNUAL"
  )
)

## End(Not run)
```

---

resolve_distance_metric

Resolve a GW distance metric preset

Description

Resolve a GW distance metric preset

Usage

```
resolve_distance_metric(name, stop_on_error = TRUE)
```

Arguments

name	Character scalar. One of gw_distance_metric_names().
stop_on_error	Logical. If TRUE, throw for unknown names; else NULL.

Value

list(p, theta, longlat) or NULL.

`split_into_chunks` *Split a data.frame into fixed-size chunks*

Description

Split a data.frame into fixed-size chunks

Usage

```
split_into_chunks(df, control = arpcPriceBasis_control())
```

Arguments

`df`	data.frame/data.table.
`control`	a list of control parameters -> dtn_prophetX_query_limit Max rows per chunk.

Value

List of data.frames.

Index