Package 'carbonr'

August 27, 2025

Title Calculate Carbon-Equivalent Emissions

Version 0.2.7

Description Provides a flexible tool for calculating carbon-equivalent emissions. Mostly using data from the UK Government's Greenhouse Gas Conversion Factors report <https://www.gov.uk/government/publications/greenhouse-gas-reporting-conversion-factors-2024>, it facilitates transparent emissions calculations for various sectors, including travel, accommodation, and clinical activities. The package is designed for easy integration into R workflows, with additional support for 'shiny' applications and community-driven extensions.

License LGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, purrr, rmarkdown, sp, testthat (>= 2.0.0), tibble

Config/testthat/edition 2

Imports airportr, checkmate, cowplot, dplyr, emojifont, ggplot2, ggpp, htmltools, lubridate, magrittr, readxl, rlang, shiny, shinydashboard, stringr, tidyr, tidyselect

Depends R (>= 4.1.0)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Lily Clements [aut, cre] (ORCID: <https://orcid.org/0000-0001-8864-0552>)

Maintainer Lily Clements <lily@idems.international>

Repository CRAN

Date/Publication 2025-08-27 20:00:02 UTC

Contents

`add_inputs` *Create multiple textInput functions in Shiny*

Description

For use in the `shiny_emissions()` function. Adding an unknown quantity of textInputs.

Usage

```
add_inputs(numeric_input, label, value)
```

Arguments

numeric_input	Name of numerical input that controls the number of items to add.
label	Label of new textInput.
value	Value of new textInput.

Value

Returns textInput for use in the shiny_emissions() function.

Examples

```
if(interactive()) {
ui <- shinydashboard::dashboardPage(header = shinydashboard::dashboardHeader(),
                                    sidebar = shinydashboard::dashboardSidebar(),
                                    shinydashboard::dashboardBody(
                                    shiny::fluidRow(
                                    shiny::column(12, align = "left",
                                    shiny::splitLayout(shinydashboard::box(width = NULL,
                                    shiny::numericInput("newbox_add",
                                                        "Number of new boxes:",
                                                        value = 0, min = 0),
                                    shiny::uiOutput("newbox_input")))))))
server <- function(input, output) {
  K_plane <- shiny::reactive({ input$newbox_add })
  output$newbox_input <- shiny::renderUI({ add_inputs(numeric_input = K_plane(),
                                                      label = "New Box:",
                                                      value = "textbox") })
}
shiny::shinyApp(ui, server)
}
```

---

airplane_emissions *Calculate CO2e emissions from an airplane journey*

Description

This function calculates the CO2e emissions between airports based on the provided parameters. The distances are calculated using the "airport_distance" function from the "airportr" package.

Usage

```
airplane_emissions(
  from,
  to,
  via = NULL,
  num_people = 1,
  radiative_force = TRUE,
  include_WTT = TRUE,
  round_trip = FALSE,
  class = c("Average passenger", "Economy class", "Business class",
    "Premium economy class", "First class")
)
```

Arguments

from	Three-letter IATA code corresponding to the departure airport. You can check the IATA code using the "airport_finder" function.
to	Three-letter IATA code corresponding to the destination airport. You can check the IATA code using the "airport_finder" function.
via	Optional. Vector of three-letter IATA codes corresponding to airports for any layovers or stops along the route.
num_people	Number of people taking the flight. Must be a single numerical value.
radiative_force	Logical. Determines whether radiative forcing should be taken into account. It is recommended to set this parameter as TRUE since emissions from airplanes at higher altitudes have a greater impact on climate change than those at ground level.
include_WTT	Logical. Determines whether emissions associated with extracting, refining, and transporting fuels should be included. It is recommended to set this parameter as TRUE.
round_trip	Logical. Determines if the flight is round trip (return) or one-way. Default is FALSE (one-way).
class	Class flown in. Options include "Average passenger", "Economy class", "Business class", "Premium economy class", and "First class".

Details

The distances are calculated using the "airport_distance" function from the "airportr" package. This means that the distances between locations uses the Haversine formula. This is calculated as the crow flies.

Value

Returns CO2e emissions in tonnes.

Examples

```
# Calculate emissions for a flight between Vancouver (YVR) and Toronto (YYZ)
airplane_emissions("YVR", "YYZ")
# Calculate emissions for a flight between London Heathrow (LHR)
# and Kisumu Airport (KIS), with layovers in Amsterdam (AMS) and Nairobi
# (NBO), flying in Economy class.
airplane_emissions("LHR", "KIS", via = c("AMS", "NBO"),
                   class = "Economy class")
```

---

airports *Table of airport detail data*

Description

This dataset is adapted from the `airportr` package. Full credit and acknowledgment go to the original authors of the `airportr` package for their contribution. A dataset containing names, codes, locations, altitude, and timezones for airports.

Usage

```
airports
```

Format

A data frame with 7698 rows and 14 variables:

OpenFlights ID OpenFlights database ID

Name Airport name, sometimes contains name of the city

City Name of the city served by the airport

IATA 3-letter IATA code

ICAO 4-letter ICAO code

Country Country name as in OpenFlights database. Note that country names may not be ISO 3166-1 standard.

Country Code ISO 3166-1 numeric country code

Country Code (Alpha-2) Two-letter ISO country code

Country Code (Alpha-3) Three-letter ISO country code

Latitude Latitude in decimal degrees

Longitude Longitude in decimal degrees

Altitude Altitude in feet

UTC Hours offset from UTC

DST Daylight Savings Time. One of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown)

Timezone Timezone in Olson format

Type Type of airport (e.g., large airport, medium airport, small airport)

Source Source of data, generally sourced from OurAirports

Source

https://cran.r-project.org/package=airportr

`airport_finder` *Find the airport code for an airport*

Description

Find the name, city, country, and IATA code of an airport. For use in the `airplane_emissions` function.

Usage

```
airport_finder(
  name,
  city,
  country,
  IATA_code,
  distance = 0.1,
  ignore.case = FALSE
)
```

Arguments

`name`	Name of the airport.
`city`	City that the airport is in.
`country`	Country that the airport is in.
`IATA_code`	The IATA code.
`distance`	Maximum distance allowed for a match between the name/country/city given, and that of the value in the data set.
`ignore.case`	If `FALSE`, the check for is case-sensitive. If `TRUE`, case is ignored.

Value

Data frame containing the name, city, country, and IATA code of an airport.

Examples

```
# Can get the IATA code from the name of an airport. Gets similar matches.
airport_finder(name = "Bristo")

# Can get the IATA code from the name and city of an airport
airport_finder(name = "Bristo", country = "United Kingdom")

# Can find the name and city of an airport given the IATA code
airport_finder(IATA_code = "BRS")
```

anaesthetic_emissions *Anaesthetic Emissions*

Description

Estimates the CO2e emissions associated with different anaesthetic agents.

Usage

```
anaesthetic_emissions(
  desflurane = 0,
  sevoflurane = 0,
  isoflurane = 0,
  N2O = 0,
  methoxyflurane = 0,
  propofol = 0
)
```

Arguments

desflurane	Amount of desflurane used in KG (default: 0).
sevoflurane	Amount of sevoflurane used in KG (default: 0).
isoflurane	Amount of isoflurane used in KG (default: 0).
N2O	Amount of nitrous oxide (N2O) used in KG (default: 0).
methoxyflurane	Amount of methoxyflurane used in KG (default: 0).
propofol	Amount of propofol used in KG (default: 0).

Details

These estimates are based on available literature and may vary depending on factors such as specific anaesthetic agents, usage conditions, and waste gas management practices.

Value

The total CO2e emissions in tonnes.

References

- McGain F, Muret J, Lawson C, Sherman JD. Environmental sustainability in anaesthesia and critical care. Br J Anaesth. 2020 Nov;125(5):680-692. DOI: 10.1016/j.bja.2020.06.055. Epub 2020 Aug 12. PMID: 32798068; PMCID: PMC7421303.
- ACS Sustainable Chem. Eng. 2019, 7, 7, 6580–6591. Publication Date: January 20, 2019. Link
- Sherman, Jodi MD*; Le, Cathy; Lamers, Vanessa; Eckelman, Matthew PhD. Life Cycle Greenhouse Gas Emissions of Anesthetic Drugs. Anesthesia & Analgesia 114(5):p 1086-1090, May 2012. DOI: 10.1213/ANE.0b013e31824f6940. Link

Examples

```
anaesthetic_emissions(desflurane = 200, sevoflurane = 30, N2O = 5)
```

`building_emissions` *Building emissions (UK govt schema; table-driven)*

Description

Building emissions (UK govt schema; table-driven)

Usage

```
building_emissions(
  water_supply = 0,
  water_trt = TRUE,
  water_unit = c("cubic metres", "million litres"),
  electricity_kWh = 0,
  electricity_TD = TRUE,
  electricity_WTT = TRUE,
  heat_kWh = 0,
  heat_TD = TRUE,
  heat_WTT = TRUE,
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

`water_supply`	numeric, amount of water in the given unit.
`water_trt`	logical, include treatment emissions (default TRUE).
`water_unit`	"cubic metres" or "million litres".
`electricity_kWh`	numeric kWh consumed.
`electricity_TD`	logical, include T&D losses (default TRUE).
`electricity_WTT`	logical, include WTT for electricity (default TRUE).
`heat_kWh`	numeric kWh of heat/steam (onsite; excludes district).
`heat_TD`	logical, include district heat distribution losses (default TRUE).
`heat_WTT`	logical, include WTT for heat/steam (default TRUE).
`units`	"kg" or "tonnes" for the result (default "kg").
`value_col`	which factor column to use: "value" or "value_2024" (default "value").
`strict`	logical, error if a required factor is missing (default TRUE).

Value

numeric total CO2e in requested units.

Examples

```
# specify emissions in an office
# Basic office: include water treatment, electricity TD+WTT, and heat TD+WTT
building_emissions(
  water_supply = 10, water_unit = "cubic metres", water_trt = TRUE,
  electricity_kWh = 100, electricity_TD = TRUE, electricity_WTT = TRUE,
  heat_kWh = 50, heat_TD = TRUE, heat_WTT = TRUE,
  units = "kg"
)

# Water only, in million litres, reported in tonnes, using 2024 factors
building_emissions(
  water_supply = 0.002, water_unit = "million litres", water_trt = TRUE,
  electricity_kWh = 0, heat_kWh = 0,
  value_col = "value_2024", units = "tonnes"
)

# Electricity without TD (but with WTT generation)
building_emissions(
  electricity_kWh = 10, electricity_TD = FALSE, electricity_WTT = TRUE,
  heat_kWh = 0, water_supply = 0
)

# Heat only, include WTT but exclude distribution
building_emissions(
  heat_kWh = 20, heat_TD = FALSE, heat_WTT = TRUE,
  water_supply = 0, electricity_kWh = 0
)
```

---

carbonr *carbonr: Calculate Carbon-Equivalent Emissions*

Description

The carbonr package provides a flexible tool for calculating carbon-equivalent emissions.

Author(s)

Maintainer: Lily Clements <lily@idems.international> (ORCID)

See Also

See the README on GitHub

carbon_price_credit *Calculate carbon price credit*

Description

This function calculates the carbon price credit for a given jurisdiction, year, period, and CO2e value. It uses CPI (Carbon Price Index) data to determine the carbon price for the specified jurisdiction and time period. The carbon price credit is calculated by multiplying the CO2e value by the corresponding carbon price.

Usage

```
carbon_price_credit(
  jurisdiction = NULL,
  year = NULL,
  period = 0,
  manual_price = NULL,
  co2e_val
)
```

Arguments

jurisdiction	A character string specifying the jurisdiction for which the carbon price credit should be calculated.
year	An optional numeric value specifying the year for which the carbon price credit should be calculated. If NULL, the most recent year available in the CPI data will be used.
period	An optional numeric value specifying the period within the specified year for which the carbon price credit should be calculated. If 1, the function will use the first period if it is available; if 2, the function will use the second period if it is available. If 0, the function will calculate the mean between the first and second period.
manual_price	An option to manually input a carbon price index to override the value in the World Bank Data. This should be a value of the carbon credit price per tCO2e.
co2e_val	A numeric value specifying the CO2e (carbon dioxide equivalent) value for which the carbon price credit should be calculated.

Value

The calculated carbon price credit in USD ($).

Examples

```
# Calculate carbon price credit for the United Kingdom in the year 2000,
# period 2, and CO2e value of 100
carbon_price_credit("United Kingdom", 2022, 2, co2e_val = 100)
```

```
# Or manually enter a value
carbon_price_credit(manual_price = 66.9, co2e_val = 100)
```

`check_CPI` *Check which jurisdictions are in the Carbon Credits data*

Description

Find jurisdictions available in the Carbon Credits data. If a jurisdiction is specified, find the years associated with that jurisdiction.

Usage

```
check_CPI(jurisdiction = NULL, period = FALSE)
```

Arguments

`jurisdiction`	(optional) A character string specifying the jurisdiction to filter the data by.
`period`	(logical) If TRUE, include the Period column in the output data frame.

Value

A vector or data frame containing the information.

Examples

```
which_jur <- check_CPI()
which_years <- check_CPI(jurisdiction = "Switzerland")
which_years_and_period <- check_CPI(jurisdiction = "Switzerland", period = TRUE)
```

`clinical_theatre_data` *Clinical Emissions: Data Frame and Plot*

Description

Calculate clinical theatre emissions row-by-row from a data frame. Each row is expanded into a call to `clinical_theatre_emissions()` using the columns you specify. Optionally, results can be combined with carbon price credit information and plotted.

Usage

```
clinical_theatre_data(
  data,
  time,
  date_format = "%d/%m/%Y",
  name,
  wet_clinical_waste,
  wet_clinical_waste_unit = c("tonnes", "kg"),
  desflurane = 0,
  sevoflurane = 0,
  isoflurane = 0,
  methoxyflurane = 0,
  N2O = 0,
  propofol = 0,
  water_supply = NULL,
  water_trt = TRUE,
  water_unit = c("cubic metres", "million litres"),
  electricity_kWh = NULL,
  electricity_TD = TRUE,
  electricity_WTT = TRUE,
  heat_kWh = NULL,
  heat_TD = TRUE,
  heat_WTT = TRUE,
  paper_vars = NULL,
  plastic_vars = NULL,
  metal_vars = NULL,
  electrical_vars = NULL,
  construction_vars = NULL,
  paper_waste = TRUE,
  plastic_waste = TRUE,
  metal_waste = TRUE,
  electrical_waste = TRUE,
  construction_waste = TRUE,
  paper_material_production = "Primary material production",
  metal_material_production = "Primary material production",
  construction_material_production = "Primary material production",
 paper_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill"),
 plastic_waste_disposal = c("Landfill", "Open-loop", "Closed-loop", "Combustion"),
 metal_waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  electrical_waste_disposal = c("Landfill", "Open-loop"),
 construction_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill",
    "Open-loop"),
  units = "kg",
  value_col = c("value", "value_2024"),
  strict = TRUE,
  include_cpi = FALSE,
  jurisdiction = NULL,
  year = NULL,
```

```
  period = 0,
  manual_price = NULL,
  gti_by = c("default", "month", "year"),
  overall_by = c("default", "month", "year"),
  single_sheet = FALSE
)
```

Arguments

data	Data frame containing all variables required for emissions calculation.
time	Column in data giving the time variable.
date_format	Character string of date format for time (default: "%d/%m/%Y").
name	Column in data giving the theatre identifier/name.
wet_clinical_waste	Numeric. Amount of (wet) clinical waste.
wet_clinical_waste_unit	Unit for wet_clinical_waste ("tonnes" or "kg").
desflurane	Amount of desflurane used in KG (default: 0).
sevoflurane	Amount of sevoflurane used in KG (default: 0).
isoflurane	Amount of isoflurane used in KG (default: 0).
methoxyflurane	Amount of methoxyflurane used in KG (default: 0).
N2O	Amount of nitrous oxide (N2O) used in KG (default: 0).
propofol	Amount of propofol used in KG (default: 0).
water_supply	numeric, amount of water in the given unit.
water_trt	logical, include treatment emissions (default TRUE).
water_unit	"cubic metres" or "million litres".
electricity_kWh	numeric kWh consumed.
electricity_TD	logical, include T&D losses (default TRUE).
electricity_WTT	logical, include WTT for electricity (default TRUE).
heat_kWh	numeric kWh of heat/steam (onsite; excludes district).
heat_TD	logical, include district heat distribution losses (default TRUE).
heat_WTT	logical, include WTT for heat/steam (default TRUE).
paper_vars	Named character vector mapping canonical paper keys ("board", "mixed", "paper") to column names in data. Each row's values are passed as a paper_use vector.
plastic_vars	Named character vector mapping canonical plastic keys (e.g. "average", "pet", "pp", "pvc", ...) to columns in data.
metal_vars	Named character vector mapping canonical metal keys (e.g. "aluminuim_cans", "steel_cans", "scrap") to columns in data.
electrical_vars	Named character vector mapping canonical electrical keys (e.g. "fridges", "freezers", "it", "alkaline_batteries") to columns in data.

`construction_vars`
: Named character vector mapping canonical construction keys (e.g. `"concrete"`, `"bricks"`, `"wood"`, `"metals"`) to columns in `data`.

`paper_waste`, `plastic_waste`, `metal_waste`, `electrical_waste`, `construction_waste`
: Logical. Whether the same tonnage is assumed to go to waste treatment for that material type (default: `TRUE`).

`paper_material_production`, `metal_material_production`, `construction_material_production`
: Column text string for material-use factor (default: `"Primary material production"`).

`paper_waste_disposal`, `plastic_waste_disposal`, `metal_waste_disposal`, `electrical_waste_disposal`, `construction_waste_disposal`
: Disposal route(s) to use for each material category. See `material_emissions()` for valid options.

`units`
: Units of result, `"kg"` or `"tonnes"`. Default: `"kg"`.

`value_col`
: Which column of `uk_gov_data` to use (`"value"` or `"value_2024"`).

`strict`
: Logical. If `TRUE`, error when a factor is missing; if `FALSE`, treat as zero.

`include_cpi`
: Logical. Whether to add carbon price credit calculations.

`jurisdiction`
: Jurisdiction string for CPI lookup (see `check_CPI()`).

`year`, `period`
: CPI year and period to use.

`manual_price`
: Optional numeric CPI value to override World Bank data.

`gti_by`
: Grouping type for GTI calculation (`"default"`, `"month"`, `"year"`).

`overall_by`
: Grouping type for overall output plot.

`single_sheet`
: `NULL`, `TRUE`, or `FALSE`. If not `NULL`, returns a list with the emissions table and a single-sheet display/plot.

Value

If `single_sheet = NULL`: a tibble with emissions (and `carbon_price_credit` if `include_cpi = TRUE`).

If `single_sheet = TRUE/FALSE`: a list with the emissions table and a plot object generated by `output_display()`.

Examples

```
df <- data.frame(
  time = c("10/04/2000","11/04/2000"),
  theatre = c("A","A"),
  clinical_waste = c(80,90),
  electricity_kwh = c(100,110),
  general_waste = c(65,55)
)

clinical_theatre_data(
  df,
  time = time,
```

```
  name = theatre,
  wet_clinical_waste = clinical_waste,
  wet_clinical_waste_unit = "kg",
  electricity_kWh = electricity_kwh,
  plastic_vars = c(average="general_waste"),
  units = "kg"
)
```

---

clinical_theatre_emissions *Clinical Emissions*

Description

Calculate CO2e from an operating theatre session by summing: (1) wet clinical waste; (2) building use (water/electricity/heat); (3) material purchases + end-of-life (paper, plastics, metal, electrical, construction); (4) anaesthetic agents.

Usage

```
clinical_theatre_emissions(
  wet_clinical_waste,
  wet_clinical_waste_unit = c("tonnes", "kg"),
  desflurane = 0,
  sevoflurane = 0,
  isoflurane = 0,
  methoxyflurane = 0,
  N2O = 0,
  propofol = 0,
  water_supply = 0,
  water_trt = TRUE,
  water_unit = c("cubic metres", "million litres"),
  electricity_kWh = 0,
  electricity_TD = TRUE,
  electricity_WTT = TRUE,
  heat_kWh = 0,
  heat_TD = TRUE,
  heat_WTT = TRUE,
  paper_use = stats::setNames(numeric(), character()),
  plastic_use = stats::setNames(numeric(), character()),
  metal_use = stats::setNames(numeric(), character()),
  electrical_use = stats::setNames(numeric(), character()),
  construction_use = stats::setNames(numeric(), character()),
  paper_waste = TRUE,
  plastic_waste = TRUE,
  metal_waste = TRUE,
  electrical_waste = TRUE,
  construction_waste = TRUE,
```

```
  paper_material_production = "Primary material production",
  metal_material_production = "Primary material production",
  construction_material_production = "Primary material production",
 paper_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill"),
 plastic_waste_disposal = c("Landfill", "Open-loop", "Closed-loop", "Combustion"),
 metal_waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  electrical_waste_disposal = c("Landfill", "Open-loop"),
 construction_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill",
    "Open-loop"),
  value_col = c("value", "value_2024"),
  units = "kg",
  strict = TRUE
)
```

Arguments

`wet_clinical_waste`	Numeric. Amount of (wet) clinical waste.
`wet_clinical_waste_unit`	Unit for `wet_clinical_waste` (`"tonnes"` or `"kg"`).
`desflurane`	Amount of desflurane used in KG (default: 0).
`sevoflurane`	Amount of sevoflurane used in KG (default: 0).
`isoflurane`	Amount of isoflurane used in KG (default: 0).
`methoxyflurane`	Amount of methoxyflurane used in KG (default: 0).
`N2O`	Amount of nitrous oxide (N2O) used in KG (default: 0).
`propofol`	Amount of propofol used in KG (default: 0).
`water_supply`	numeric, amount of water in the given unit.
`water_trt`	logical, include treatment emissions (default TRUE).
`water_unit`	"cubic metres" or "million litres".
`electricity_kWh`	numeric kWh consumed.
`electricity_TD`	logical, include T&D losses (default TRUE).
`electricity_WTT`	logical, include WTT for electricity (default TRUE).
`heat_kWh`	numeric kWh of heat/steam (onsite; excludes district).
`heat_TD`	logical, include district heat distribution losses (default TRUE).
`heat_WTT`	logical, include WTT for heat/steam (default TRUE).
`paper_use`, `plastic_use`, `metal_use`, `electrical_use`, `construction_use`	Named numeric vectors (tonnes) for each material family (see **Material vectors** above).
`paper_waste`, `plastic_waste`, `metal_waste`, `electrical_waste`, `construction_waste`	Logical. If `TRUE`, the same tonnage as the corresponding `*_use` is routed to waste treatment using the family's `*_waste_disposal` choice. Default `TRUE`.

`paper_material_production`, `metal_material_production`, `construction_material_production`
: Column Text choice for **material-use** factors (typically `"Primary material production"`).

`paper_waste_disposal`, `plastic_waste_disposal`, `metal_waste_disposal`, `electrical_waste_disposal`, `construction_waste_disposal`
: Disposal route to use for that family (see `material_emissions()` for allowed values).

`value_col`
: Which `uk_gov_data` column to use: `"value"` or `"value_2024"`. Default `"value"`.

`units`
: Output units: `"kg"` (default) or `"tonnes"`.

`strict`
: Logical. If `TRUE`, missing factors error inside delegated functions; if `FALSE`, treat as 0. Default `TRUE`.

Details

Wet clinical waste factor defaults to 0.879 tCO2e per tonne (NGA 2022). Building emissions are computed via `building_emissions()` with `units = "kg"`. Materials are computed via `material_emissions()` with `units = "kg"`. Anaesthetic emissions are computed via `anaesthetic_emissions()` (tonnes) and converted to kg to sum. The function sums all components in kg and returns in the `units` requested.

Value

Total CO2e in the units specified by `units` (kg or tonnes).

Inputs – Material vectors

For each material family, pass a **named numeric vector** of tonnages (in tonnes). Names must be canonical keys (case/space/punctuation is normalised internally, but using the canonical forms below is recommended):

- **Paper** `paper_use`: `board`, `mixed`, `paper`
- **Plastics** `plastic_use`: `average`, `average_film`, `average_rigid`, `hdpe`, `ldpe`, `lldpe`, `pet`, `pp`, `ps`, `pvc`
- **Metal** `metal_use`: use your package's canonical metal keys (e.g. `aluminium_cans`, `aluminium_foil`, `mixed_cans`, `scrap`, `steel_cans`)
- **Electrical** `electrical_use`: `fridges`, `freezers`, `large_electrical`, `it`, `small_electrical`, `alkaline_batteries`, `liion_batteries`, `nimh_batteries`
- **Construction** `construction_use`: e.g. `aggregates`, `average`, `asbestos`, `asphalt`, `bricks`, `concrete`, `insulation`, `metals`, `soils`, `mineral_oil`, `plasterboard`, `tyres`, `wood`

For each family you can also set a single **waste toggle** (e.g. `plastic_waste = TRUE`) to send the same tonnage to a single disposal route (e.g. `plastic_waste_disposal = "Landfill"`).

Examples

```
# Minimal example using vector-first materials
clinical_theatre_emissions(
  wet_clinical_waste = 100, wet_clinical_waste_unit = "kg",

  # Building (kWh)
  electricity_kWh = 250, heat_kWh = 120,

  # Materials: paper/plastic/metal/electrical/construction
  paper_use = c(paper = 20),
  paper_waste = TRUE, paper_waste_disposal = "Closed-loop",

  plastic_use = c(pet = 10),
  plastic_waste = TRUE, plastic_waste_disposal = "Landfill",

  metal_use = c(steel_cans = 0.2),
  metal_waste = TRUE, metal_waste_disposal = "Open-loop",

  electrical_use = c(alkaline_batteries = 0.05),
  electrical_waste = TRUE, electrical_waste_disposal = "Open-loop",

  construction_use = c(concrete = 1),
  construction_waste = FALSE,

  value_col = "value",
  units = "kg"
)
```

---

construction_emissions

Calculate construction emissions (UK govt schema)

Description

Computes embodied GHG emissions for construction materials using the UK government conversion factors table (uk_gov_data), specifically rows with Level 2 = "Construction". Factors are taken from the selected column (value or value_2024) and are assumed to be kg CO2e per tonne.

Usage

```
construction_emissions(
  use = stats::setNames(numeric(), character()),
  waste = TRUE,
 material_production = c("Primary material production", "Re-used", "Closed-loop source"),
 waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill", "Open-loop"),
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

use	Named numeric vector of material quantities in tonnes. Names are matched case/space/punctuation-insensitively to Level 3 (e.g., "Mineral oil", "mineral_oil", "MINERAL-OIL" all match). Missing/unknown materials are treated as zero.
waste	Logical. If TRUE, waste quantities are assumed equal to use (i.e., the same tonnage is sent to the chosen disposal route). If FALSE, no waste is applied (equivalent to zero for all materials).
material_production	Either: • a single string applied to all materials, e.g. "Primary material production", "Closed-loop source", or "Re-used"; or • a named character vector giving a choice per material name, e.g. c(concrete = "Closed-loop source", wood = "Re-used"). Synonyms are accepted for material production: "reused"/"re-used", "closed loop"/"closed-loop"/"closed-loop source".
waste_disposal	One of "Closed-loop", "Combustion", "Composting", "Landfill", or "Open-loop". Applied to all waste. If the chosen disposal route is not available for a material, behaviour depends on strict.
units	Output units: "kg" (default) or "tonnes".
value_col	Which factor column to use from uk_gov_data: "value" or "value_2024".
strict	Logical (default TRUE). If TRUE, error when a required factor is missing/invalid for any nonzero quantity (either material use or waste). If FALSE, treat missing factors as zero contribution.

Details

Material-production options (Column Text under Material use):

- Aggregates, Asphalt: "Primary material production", "Closed-loop source", "Re-used".
- Asbestos, Average Construction, Bricks: "Primary material production" only.
- Concrete, Insulation, Metals, Mineral Oil, Plasterboard: "Primary material production" or "Closed-loop source".
- Soils: "Closed-loop source" only.
- Tyres, Wood: "Primary material production" or "Re-used".

Waste-disposal options (Column Text under Waste disposal):

- "Closed-loop" is valid for aggregates, average, asphalt, concrete, insulation, metal, soils, mineral oil, plasterboard, tyres, wood.
- "Combustion" is valid for average, mineral oil, wood.
- "Composting" is valid for wood only.
- "Landfill" is valid for everything except average, mineral oil, tyres.
- "Open-loop" is valid for aggregates, average, asphalt, bricks, concrete (and any other materials where the table provides a factor).

These rules are enforced by the presence/absence of rows in uk_gov_data. If a requested material-route pair has no factor in the table, the lookup yields NA: with strict = TRUE a descriptive error is thrown; with strict = FALSE it contributes zero to the total.

Units: Factors are kg CO2e / tonne; if units = "tonnes", the result is divided by 1000.

Value

Numeric total emissions in the requested units.

Examples

```
# 1) Basic: primary production for all materials, landfill waste = use
construction_emissions(
  use = c(Aggregates = 1000, Concrete = 500, Wood = 2000),
  material_production = "Primary material production",
  waste_disposal = "Landfill",
  waste = TRUE,
  strict = FALSE,
  units = "kg"
)

# 2) Per-material production + synonyms ("closed loop" ->
# "Closed-loop source", "reused" -> "Re-used")
construction_emissions(
  use = c(aggregates = 100, concrete = 50, wood = 10),
  material_production = c(aggregates = "closed loop",
                          concrete = "Closed-loop source",
                          wood = "reused"),
  waste_disposal = "Landfill",
  waste = TRUE,
  units = "tonnes",
  value_col = "value_2024"
)

# 3)  Tolerant mode treats missing factors as zero:
construction_emissions(
  use = c(bricks = 10),
  material_production = "Re-used",
  strict = FALSE
)
```

---

degree_conversion *Convert degrees to radians*

Description

Convert degrees to radians.

Usage

```
degree_conversion(deg)
```

Arguments

deg	Degree value to convert.

Value

Value in radians.

Examples

```
# Convert 90 degrees into radians
degree_conversion(90)
```

---

distance_calc *Distance calculator*

Description

Calculate distances between locations in miles using the Haversine formula. This is calculated as the crow flies.

Usage

```
distance_calc(lat1, lat2, long1, long2)
```

Arguments

lat1	Latitude of the first location.
lat2	Latitude of the second location.
long1	Longitude of the first location.
long2	Longitude of the second location.

Value

Distance between locations in miles

Examples

```
# Distance between the London Eye and the Eiffel Tower in miles
distance_calc(51.5033, 48.8584, 0.1196, 2.2945)
```

`electrical_emissions` *Electrical emissions*

Description

Computes embodied GHG emissions for electrical items using `uk_gov_data` rows with Level 2 = "Electrical items". Material-use factors come from `Level 1 = "Material use"`, `Column Text = material_production` (your table currently provides Primary material production only). Waste factors come from `Level 1 = "Waste disposal"`, C
Factors are kg CO2e per tonne.

Usage

```
electrical_emissions(
  use = stats::setNames(numeric(), character()),
  waste = TRUE,
  material_production = "Primary material production",
  waste_disposal = c("Landfill", "Open-loop"),
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

`use`	Named numeric vector of quantities in tonnes. Canonical names supported: `fridges`, `freezers`, `large_electrical`, `it`, `small_electrical`, `alkaline_batteries`, `liion_batteries`, `nimh_batteries`. Aliases accepted (case/punct-insensitive), e.g. `"fridge"`, `"freezer"`, `"large"`, `"it equipment"`, `"small"`, `"alkaline"`, `"liion"`, `"ni mh"`, Unknown names warn and are ignored.
`waste`	Logical. If `TRUE`, waste tonnages equal `use`. If `FALSE`, no waste.
`material_production`	Either a single string for all items (e.g., `"Primary material production"`) or a named vector per item. Synonyms accepted: `"primary"`, `"closed loop"`, etc. (If a chosen option is absent in the table, behaviour depends on `strict`.)
`waste_disposal`	One of `"Landfill"` or `"Open-loop"`. Applied to all waste.
`units`	Output units: `"kg"` (default) or `"tonnes"`.
`value_col`	Which numeric column to use: `"value"` or `"value_2024"`.
`strict`	If `TRUE` (default), error when any nonzero `use`/waste needs a factor missing from the table. If `FALSE`, treat missing factors as `0`.

Value

Numeric total emissions in requested `units`.

Examples

```
# Primary production + landfill; waste = use
electrical_emissions(
  use = c(fridges = 1, freezers = 0.5, large_electrical = 0.2),
  waste_disposal = "Landfill",
  waste = TRUE,
  units = "kg"
)

# 2024 factors, no waste, report in tonnes
electrical_emissions(
  use = c(it = 0.01, liion_batteries = 0.002),
  value_col = "value_2024",
  waste = FALSE,
  units = "tonnes"
)
```

---

example_clinical_theatre

Clinical Theatre Data

Description

This dataset contains activities within a clinical theatre setting related to healthcare services or medical supplies. It is structured to facilitate analysis of healthcare operations, costs, and estimate CO2e emissions.

Usage

```
example_clinical_theatre
```

Format

A data frame with 4,386 rows and 6 columns, including:

time The date of transaction or activity in DDMMYYYY format.

prices The cost associated with each transaction or activity in AUD, related to services provided or medical supplies used.

quantities The amount or volume of a product or service provided, indicating the number of items used or procedures performed.

prodID A unique identifier for each type of product or service, such as medical supplies, medications, or surgical procedures.

retID A code identifying the department, supplier, or healthcare provider involved in the transaction, or the entity receiving the product or service.

description Categorises the clinical theatre activities into medical specialties or types of procedures, with 'ent' (Ear, Nose, and Throat), 'colorectal', 'hbt' (hematology or blood treatment), 'ortho' (orthopedics), 'paedsurg' (pediatric surgery), and 'gensurg' (general surgery).

Details

The dataset provides an overview of clinical theatre operations.

`ferry_emissions` *Calculate CO2e emissions from ferry journeys*

Description

A function that calculates CO2e emissions between ferry ports.

Usage

```
ferry_emissions(
  from,
  to,
  via = NULL,
  type = c("Foot", "Car", "Average"),
  num_people = 1,
  times_journey = 1,
  include_WTT = TRUE,
  round_trip = FALSE
)
```

Arguments

`from`	Port code for the port departing from. Use `seaport_finder` to find port code.
`to`	Port code for the port arriving from. Use `seaport_finder` to find port code.
`via`	Optional. Takes a vector containing the port code that the ferry travels through. Use `seaport_finder` to find port code.
`type`	Whether the journey is taken on foot or by car. Options are `"Foot"`, `"Car"`, `"Average"`.
`num_people`	Number of people taking the journey. Takes a single numerical value.
`times_journey`	Number of times the journey is taken.
`include_WTT`	logical. Recommended `TRUE`. Whether to include emissions associated with extracting, refining, and transporting fuels.
`round_trip`	Whether the journey is one-way or return.

Details

The distances are calculated using the Haversine formula. This is calculated as the crow flies.

Value

Returns CO2e emissions in tonnes for the ferry journey.

Examples

```
# Emissions for a ferry journey between Belfast and New York City
seaport_finder(city = "Belfast")
seaport_finder(city = "New York")
ferry_emissions(from = "BEL", to = "BOY")
```

`gg_value_box` *Create a Value Box for Reports*

Description

This function creates a value box for use in reports.

Usage

```
gg_value_box(values, information, icons)
```

Arguments

`values`	A vector of numeric values to be displayed in the value box.
`information`	A vector of strings providing information or labels for the values.
`icons`	A vector of Font Awesome unicode symbols to be displayed as icons.

Details

This function creates a value box with customizable values, information, and icons. The function takes inputs for the values, information, icons, and color of the value box. The values and information are provided as vectors, while the icons are specified using Font Awesome unicode symbols. The color of the value box can be customized using a factor variable. The resulting value box is a `ggplot2` object that can be further customized or combined with other plots or elements in a report.

Value

A `ggplot2` object with a value box for report use.

References

Modified from Stack Overflow post: https://stackoverflow.com/questions/47105282/valuebox-like-function-for-static-reports

Examples

```
# Create a value box with custom values and icons
gg_value_box(
  values = c(100, 500, 1000),
  information = c("Sales", "Revenue", "Customers"),
  icons = c("\U0000f155", "\U0000f155", "\U0000f0f7")
)
```

hotel_emissions *Calculate CO2e emissions from a hotel stay*

Description

Indirect emissions from a stay at a hotel. Values to calculate emissions are from UK government 2024 report.

Usage

```
hotel_emissions(location = "UK", nights = 1, rooms = 1)
```

Arguments

location	Location of the hotel stay. Current accepted locations are "UK", "UK (London)", "Australia", "Belgium", "Brazil", "Canada", "Chile", "China", "Colombia", "Costa Rica", "Egypt", "France", "Germany", "Hong Kong, China", "India", "Indonesia", "Italy", "Japan", "Jordan", "Korea", "Malaysia", "Maldives", "Mexico", "Netherlands", "Oman", "Philippines", "Portugal", "Qatar", "Russia", "Saudi Arabia", "Singapore", "South Africa", "Spain", "Switzerland", "Thailand", "Turkey", "United Arab Emirates", "United States", "Vietnam".
nights	Number of nights stayed in the hotel.
rooms	Number of rooms used in the hotel.

Value

Tonnes of CO2e emissions for a stay in a hotel.

Examples

```
# Emissions for a two night stay in Australia.
hotel_emissions(location = "Australia", nights = 2)
```

household_emissions *Calculate household material emissions (vector-first)*

Description

Sums emissions from paper, plastics, metals, electrical, construction (delegated to their calculators) plus household-specific streams: Glass/Clothing/Books (GCB), Organic (food/drink/compost), and Household residual waste.

Usage

```
household_emissions(
  paper_use = stats::setNames(numeric(), character()),
  plastic_use = stats::setNames(numeric(), character()),
  metal_use = stats::setNames(numeric(), character()),
  electrical_use = stats::setNames(numeric(), character()),
  construction_use = stats::setNames(numeric(), character()),
  paper_waste = TRUE,
  plastic_waste = TRUE,
  metal_waste = TRUE,
  electrical_waste = TRUE,
  construction_waste = TRUE,
  paper_material_production = "Primary material production",
  metal_material_production = "Primary material production",
  construction_material_production = "Primary material production",
 paper_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill"),
 plastic_waste_disposal = c("Landfill", "Open-loop", "Closed-loop", "Combustion"),
 metal_waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  electrical_waste_disposal = c("Landfill", "Open-loop"),
 construction_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill",
    "Open-loop"),
  gcb_use = stats::setNames(numeric(), character()),
  gcb_waste = TRUE,
  gcb_waste_disposal = c("Closed-loop", "Combustion", "Landfill"),
  organic_use = stats::setNames(numeric(), character()),
  organic_waste = TRUE,
  compost_waste_disposal = c("Anaerobic digestion", "Combustion", "Composting",
    "Landfill"),
  household_residual_waste = 0,
  hh_waste_disposal = c("Combustion", "Landfill"),
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

`paper_use`, `plastic_use`, `metal_use`, `electrical_use`, `construction_use`
: Named numeric vectors (tonnes) passed to the corresponding calculators.

`paper_waste`, `plastic_waste`, `metal_waste`, `electrical_waste`, `construction_waste`
: Logical; if `TRUE`, apply waste factors to the same tonnages as `*_use`.

`paper_material_production`, `metal_material_production`, `construction_material_production`
: Single string or per-material named vector for MU column text; forwarded.

`paper_waste_disposal`, `plastic_waste_disposal`, `metal_waste_disposal`, `electrical_waste_disposal`, `construction_waste_disposal`
: Waste route per family; forwarded.

gcb_use — Named numeric vector for Glass/Clothing/Books (keys: glass, clothing, books). E.g., gcb_use = c(glass = 3, books = 0.5)

gcb_waste — Logical; if TRUE, apply GCB waste factors to same tonnages.

gcb_waste_disposal — One of "Closed-loop", "Combustion", "Landfill".

organic_use — Named numeric vector for food, drink, compost_from_garden, compost_from_food_and_garden.

organic_waste — Logical; if TRUE, apply organic waste factors to same tonnages.

compost_waste_disposal — One of "Anaerobic digestion", "Combustion", "Composting", "Landfill".

household_residual_waste — Numeric (tonnes).

hh_waste_disposal — "Combustion" or "Landfill".

units — Output units: "kg" (default) or "tonnes".

value_col — Which factor column in uk_gov_data to use: "value" or "value_2024".

strict — If TRUE (default) error when a required factor is missing; if FALSE, treat as 0.

Value

Numeric total emissions in requested units.

Inputs

Provide named *_use vectors in tonnes and *_waste = TRUE/FALSE flags. Unknown names in gcb_use / organic_use are ignored with a warning.

Examples

```
household_emissions(
  gcb_use = c(glass = 3, books = 0.5),
  organic_use = c(food = 1, drink = 0.5),
  household_residual_waste = 0.8,
  gcb_waste = TRUE, organic_waste = TRUE,
  gcb_waste_disposal = "Closed-loop",
  compost_waste_disposal = "Anaerobic digestion",
  hh_waste_disposal = "Combustion",
  units = "kg"
)
```

import_CPI *Import CPI data from an Excel file*

Description

This function uses the downloaded data from the World Bank Carbon Pricing Dashboard (https://carbonpricingdashboard.wor
It imports data from an Excel file containing CPI (Carbon Price Index) data. It filters the data for ETS (Emissions Trading Scheme) instruments, performs necessary transformations, and returns a processed data frame.

Usage

```
import_CPI(path, sheet = "Data_Price", skip = 2)
```

Arguments

path	A character string specifying the file path of the Excel file.
sheet	A character string specifying the name of the sheet in the Excel file to read data from.
skip	An integer specifying the number of rows to skip while reading the Excel sheet.

Value

A processed data frame containing CPI data.

land_emissions *Calculate CO2e emissions from land-travel journeys*

Description

A function that calculates CO2e emissions on a journey on land.

Usage

```
land_emissions(
  distance,
  units = c("miles", "km"),
  num = 1,
 vehicle = c("Cars", "Motorbike", "Taxis", "Bus", "National rail", "International rail",
    "Light rail and tram", "London Underground", "Coach"),
  fuel = c("Petrol", "Diesel", "Unknown", "Battery Electric Vehicle",
    "Plug-in Hybrid Electric Vehicle"),
  car_type = c("Average car", "Small car", "Medium car", "Large car", "Mini",
   "Supermini", "Lower medium", "Upper medium", "Executive", "Luxury", "Sports",
    "Dual purpose 4X4", "MPV"),
```

```
  bike_type = c("Average", "Small", "Medium", "Large"),
  bus_type = c("Local bus (not London)", "Local London bus", "Average local bus"),
  taxi_type = c("Regular taxi", "Black cab"),
  TD = TRUE,
  include_WTT = TRUE,
  include_electricity = TRUE,
  owned_by_org = TRUE
)
```

Arguments

distance	Distance in km or miles of the journey made (this can be calculated with other tools, such as google maps.).
units	Units for the distance travelled. Options are "km" or "miles".
num	Number of passengers if vehicle is one of coach, tram, or tube. Otherwise, number of vehicles used.
vehicle	Vehicle used for the journey. Options are "Cars", "Motorbike", "Taxis", "Bus", "National rail", "International rail, "Coach", "Light rail and tram", "London Underground". Note: car, taxi, motorbike is per vehicle.
fuel	Fuel type used for the journey. For car, "Petrol", "Diesel", "Unknown", "Battery Electric Vehicle", "Plug-in Hybrid Electric Vehicle" are options. ##' "hybrid electric" and "battery electric" account for electricity kWh emissions.
car_type	Size/type of vehicle for car. Options are c("Average car", "Small car", "Medium car", "Large car" "Mini", "Supermini", "Lower medium", "Upper medium", "Executive", "Luxury", "Sports", "Dual purpose 4X4", "MPV"),. Small denotes up to a 1.4L engine, unless diesel which is up to 1.7L engine. Medium denotes 1.4-2.0L for petrol cars, 1.7-2.0L for diesel cars. Large denotes 2.0L+ engine.
bike_type	Size of vehicle for motorbike. Options are "Small", "Medium", "Large", or "Average". Sizes denote upto 125cc, 125cc-500cc, 500cc+ respectively.
bus_type	Options are "local_nL", "local_L", "local", or "average". These denote whether the bus is local but outside of London, local in London, local, or average.
taxi_type	Whether a taxi is regular or black cab. Options are "Regular taxi", "Black cab".
TD	logical.Whether to account for transmission and distribution (TD) for electric vehicles (only car and van)
include_WTT	logical. Well-to-tank (include_WTT) - whether to account for emissions associated with extraction, refining and transportation of the fuels (for non-electric vehicles).
include_electricity	logical. Whether to account for ... for electric vehicles (car and van).
owned_by_org	logical. Whether the vehicle used is owned by the organisation or not (only for car, motorbike).

Value

Tonnes of CO2e emissions per mile travelled.

Examples

```
# Emissions for a 100 mile car journey
land_emissions(distance = 100)

# Emissions for a 100 mile motorbike journey where the motorbike is 500+cc
land_emissions(distance = 100, vehicle = "Motorbike", bike_type = "Large")
```

`material_emissions` *Material (and waste) emissions — vector-first wrapper*

Description

Convenience wrapper that sums emissions from paper, plastics, metals, electrical items, construction materials, glass, and industrial waste. It forwards to the dedicated calculators: `paper_emissions()`, `plastic_emissions()`, `metal_emissions()`, `electrical_emissions()`, and `construction_emissions()`.

Usage

```
material_emissions(
  paper_use = stats::setNames(numeric(), character()),
  plastic_use = stats::setNames(numeric(), character()),
  metal_use = stats::setNames(numeric(), character()),
  electrical_use = stats::setNames(numeric(), character()),
  construction_use = stats::setNames(numeric(), character()),
  paper_waste = TRUE,
  plastic_waste = TRUE,
  metal_waste = TRUE,
  electrical_waste = TRUE,
  construction_waste = TRUE,
  paper_material_production = "Primary material production",
  metal_material_production = "Primary material production",
  construction_material_production = "Primary material production",
 paper_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill"),
 plastic_waste_disposal = c("Landfill", "Open-loop", "Closed-loop", "Combustion"),
 metal_waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  electrical_waste_disposal = c("Landfill", "Open-loop"),
 construction_waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill",
    "Open-loop"),
  glass = 0,
  glass_waste = TRUE,
 glass_waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  industrial_waste = 0,
  industrial_waste_disposal = c("Combustion", "Landfill"),
```

```
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

paper_use, plastic_use, metal_use, electrical_use, construction_use	Named numeric vectors of quantities in tonnes (defaults empty).
paper_waste, plastic_waste, metal_waste, electrical_waste, construction_waste, glass_waste	Logical flags: if TRUE, apply waste factors to the same tonnages as use.
paper_material_production, metal_material_production, construction_material_production	Either a single string (applied to all materials) or a named vector per material. Common values: "Primary material production", "Closed-loop source", "Closed-loop", "Open-loop", "Combustion", "Landfill" (availability depends on the table).
paper_waste_disposal, plastic_waste_disposal, metal_waste_disposal, electrical_waste_disposal, construction_waste_disposal, glass_waste_disposal	Waste route to use for the family. See the calculators' docs for valid choices. (Electrical typically: "Landfill", "Open-loop"; Construction supports "Closed-loop", "Combustion", "Composting", "Landfill", "Open-loop" with material-specific availability.)
glass	Numeric tonnage of glass (material use).
industrial_waste	Numeric tonnage of commercial and industrial waste (end-of-life only).
industrial_waste_disposal	"Combustion" or "Landfill".
units	Output units: "kg" or "tonnes" (default "kg").
value_col	Which factor column in uk_gov_data to use: "value" or "value_2024".
strict	If TRUE (default), error when a required factor is missing; if FALSE, treat missing factors as 0.

Details

For each family you provide a named use vector in tonnes plus a waste = TRUE/FALSE flag (waste tonnage equals use when TRUE). Unknown names are ignored by the underlying calculators (with warnings).

Value

Total emissions in the requested units.

Canonical names (examples)

- Paper: board, mixed, paper
- Plastics: average, average_film, average_rigid, hdpe, ldpe, lldpe, pet, pp, ps, pvc
- Metals: aluminium (cans/foil), mixed_cans, scrap, steel_cans
- Electrical: fridges, freezers, large_electrical, it, small_electrical, alkaline_batteries, liion_batteries, nimh_batteries
- Construction: use the material names supported by construction_emissions()

Backwards compatibility

Legacy scalar arguments (e.g. board, HDPE, fridges, aluminuim_cans, ...) are still accepted and are added into the corresponding *_use vectors. Legacy *_WD arguments (separate waste tonnages) are deprecated and ignored; supply *_waste = TRUE instead.

Examples

```
# Paper + Metals + Glass, with waste to the same tonnages
material_emissions(
  paper_use = c(board = 10, paper = 5),
  metal_use = c(aluminium = 0.4, steel_cans = 0.2),
  glass = 3, glass_waste = TRUE,
  paper_waste_disposal = "Closed-loop",
  metal_waste_disposal = "Landfill",
  glass_waste_disposal = "Closed-loop",
  units = "kg"
)
```

---

metal_emissions *Metal emissions (UK govt schema; table-driven, with material_production)*

Description

Computes embodied GHG emissions for metals using uk_gov_data rows with Level 2 = "Metal". Material-use factors come from Level 1 = "Material use", Column Text = material_production. Waste factors come from Level 1 = "Waste disposal", Column Text = waste_disposal. Factors are kg CO2e per tonne.

Usage

```
metal_emissions(
  use = stats::setNames(numeric(), character()),
  waste = TRUE,
  material_production = "Primary material production",
  waste_disposal = c("Closed-loop", "Combustion", "Landfill", "Open-loop"),
  units = c("kg", "tonnes"),
```

```
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

use	Named numeric vector of metal quantities in tonnes. Canonical names supported: aluminium, mixed_cans, scrap, steel_cans. Aliases accepted (case/punct/UK-US spelling-insensitive), e.g.: "Metal: aluminium cans and foil (excl. forming)", aluminum_cans, aluminium_foil, mixed, scrap_metal, etc. Unknown names warn and are ignored.
waste	Logical. If TRUE, waste tonnages are the same as use. If FALSE, no waste is applied.
material_production	Either a single string applied to all metals, or a named vector per metal type. Accepted values (matched leniently): "Primary material production", "Closed-loop source", "Closed-loop", "Open-loop", "Combustion", "Landfill". Synonyms accepted: "primary", "closed loop", "open loop", etc. Unspecified metals default to "Primary material production".
waste_disposal	One of "Closed-loop", "Combustion", "Landfill", "Open-loop". Applied to all waste.
units	Output units: "kg" (default) or "tonnes".
value_col	Which numeric column in uk_gov_data to use: "value" or "value_2024".
strict	If TRUE (default), error when any nonzero use/waste needs a factor absent in the table. If FALSE, treat missing factors as 0.

Value

Numeric total emissions in requested units.

Examples

```
# Primary for all; landfill; waste = use
metal_emissions(
  use = c(aluminium = 1.2, steel_cans = 0.4),
  material_production = "Primary material production",
  waste_disposal = "Landfill",
  waste = TRUE,
  units = "kg"
)

# Per-metal: aluminium closed-loop source, others primary; no waste; 2024 factors
metal_emissions(
  use = c(`Metal: aluminium cans and foil (excl. forming)` = 0.5, mixed_cans = 0.2),
  material_production = c(aluminium = "closed loop"),
  waste = FALSE,
  value_col = "value_2024",
  units = "tonnes"
)
```

`office_emissions` *Office emissions (uses building_emissions + homeworking factors)*

Description

Computes office emissions either from a per-person average (when `specify = FALSE`) or from specified utilities via `building_emissions()` (when `specify = TRUE`), plus optional **homeworking** emissions (Level 1 = "Homeworking").

Usage

```
office_emissions(
  specify = FALSE,
  office_num = 1,
  WFH_num = 0,
  WFH_hours = 0,
  WFH_type = c("Office Equipment", "Heating"),
  water_supply = 0,
  water_trt = TRUE,
  water_unit = c("cubic metres", "million litres"),
  electricity_kWh = 0,
  electricity_TD = TRUE,
  electricity_WTT = TRUE,
  heat_kWh = 0,
  heat_TD = TRUE,
  heat_WTT = TRUE,
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

`specify`	Logical. If `FALSE`, use an average per person; if `TRUE`, use the specified utility inputs (water/electricity/heat) via `building_emissions()`.
`office_num`	Number of individuals in the office (only used when `specify = FALSE`). Uses a constant of **2600 kg CO2e per person-year**.
`WFH_num`	Number of people working from home.
`WFH_hours`	Hours worked from home **per person**. If one of `WFH_num`/`WFH_hours` is zero but the other is > 0, the zero one is treated as 1.
`WFH_type`	Which homeworking components to include; any of `c("Office Equipment","Heating")`. Default is both.
`water_supply`	See `building_emissions()`.
`water_trt`	See `building_emissions()`.
`water_unit`	See `building_emissions()`.

`electricity_kWh`, `electricity_TD`, `electricity_WTT`
: See `building_emissions()`.

`heat_kWh`, `heat_TD`, `heat_WTT`
: See `building_emissions()`.

`units`
: Output units: `"kg"` (default) or `"tonnes"`.

`value_col`
: Which factor column to use in `uk_gov_data`: `"value"` or `"value_2024"`.

`strict`
: If `TRUE` (default), error when needed factors (e.g., homeworking) are missing. If `FALSE`, treat missing factors as 0.

Details

Factors are assumed **kg CO2e per unit** (e.g., per person-hour for homeworking), and the result is returned in requested `units`.

Value

Numeric total emissions in requested `units`.

Examples

```
# 1) Use specified utilities (building_emissions) + homeworking
office_emissions(
  specify = TRUE,
  electricity_kWh = 200, heat_kWh = 100,
  water_supply = 10, water_unit = "cubic metres",
  WFH_num = 5, WFH_hours = 8, WFH_type = c("Office Equipment","Heating")
)

# 2) Use per-person average for 12 staff, with WFH equipment only
office_emissions(
  specify = FALSE, office_num = 12,
  WFH_num = 6, WFH_hours = 4, WFH_type = "Office Equipment",
  units = "tonnes"
)
```

---

output_display *Display a grid of plots and tables*

Description

This function generates a grid of plots and tables, including a value box, data table, relative GPI plot, and total output plot.

Usage

```
output_display(
  data = x$data,
  time = time,
  date_format = c("%d/%m/%Y"),
  name = theatre,
  relative_gpi_val = emissions,
  gti_by = c("default", "month", "year"),
  plot_val = carbon_price_credit,
  plot_by = "default",
  pdf = TRUE
)
```

Arguments

data	The data frame containing the data.
time	The variable representing the time dimension.
date_format	The date format for the time variable (optional, default: "%d/%m/%Y").
name	The variable representing the grouping variable.
relative_gpi_val	The variable representing the relative GPI (Growth to Previous Index) value.
gti_by	The grouping type for calculating the GTI ("default", "month", "year").
plot_val	The variable to plot in the total output plot.
plot_by	The grouping type for the total output plot ("default", "month", "year").
pdf	Whether to export the plots to a PDF file (default: TRUE).

Details

The function utilises other auxiliary functions such as relative_gti() and total_output().

Value

A grid of plots and tables showing the value box, data table, relative GPI plot, and total output plot.

---

paper_emissions *Paper emissions*

Description

Computes embodied GHG emissions for paper using `uk_gov_data` rows with Level 2 = "Paper". Material-use factors come from `Level 1 = "Material use"`, `Column Text = material_production`. Waste factors come from `Level 1 = "Waste disposal"`, `Column Text = waste_disposal`. Factors are kg CO2e per tonne.

Usage

```
paper_emissions(
  use = stats::setNames(numeric(), character()),
  waste = TRUE,
  material_production = "Primary material production",
  waste_disposal = c("Closed-loop", "Combustion", "Composting", "Landfill"),
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

use	Named numeric vector of paper quantities in tonnes. Names matched case/space/punctuation-insensitively to `Level 3` (drops a leading `"Paper: "` prefix and trailing parentheses). Canonical names: `board`, `mixed`, `paper`. Unknown names warn and are ignored.
waste	Logical. If `TRUE`, waste tonnages are the same as `use`. If `FALSE`, no waste is applied.
material_production	Either a single string applied to all paper types (e.g., `"Primary material production"` or `"Closed-loop source"`), or a named vector per paper type, e.g. `c(board = "Closed-loop source", mixed = "Primary material production")`. If you provide a per-material vector for a subset, unspecified types default to `"Primary material production"`.
waste_disposal	One of `"Closed-loop"`, `"Combustion"`, `"Composting"`, `"Landfill"`.
units	Output units: `"kg"` (default) or `"tonnes"`.
value_col	Which numeric column in `uk_gov_data` to use: `"value"` or `"value_2024"`.
strict	If `TRUE` (default), error when any nonzero `use`/waste needs a factor that is absent in the table. If `FALSE`, treat missing factors as 0.

Value

Numeric total emissions in requested `units`.

Examples

```
# Closed-loop source for all paper types; landfill; waste = use
paper_emissions(
  use = c(board = 10, paper = 100),
  material_production = "Closed-loop source",
  waste_disposal = "Landfill",
  waste = TRUE
)

# Per-material: board closed-loop, mixed primary (default), no waste
paper_emissions(
  use = c(board = 5, mixed = 2),
```

```
  material_production = c(board = "closed loop"),
  waste = FALSE,
  value_col = "value_2024",
  units = "tonnes"
)
```

`plastic_emissions` *Calculate plastic emissions*

Description

Computes embodied GHG emissions for plastics using uk_gov_data rows with Level 2 = "Plastic". Material-use factors come from `Level 1 = "Material use"`, `Column Text = "Primary material production"`. Waste factors come from `Level 1 = "Waste disposal"`, `Column Text = waste_disposal`. Factors are assumed kg CO2e per tonne (`UOM = tonne`, `GHG/Unit = kg CO2e`).

Usage

```
plastic_emissions(
  use = stats::setNames(numeric(), character()),
  waste = TRUE,
  waste_disposal = c("Landfill", "Open-loop", "Closed-loop", "Combustion"),
  units = c("kg", "tonnes"),
  value_col = c("value", "value_2024"),
  strict = TRUE
)
```

Arguments

use	Named numeric vector of plastic quantities in tonnes. Names are matched (case/space/punctuation-insensitive) to `Level 3` after normalisation that also: 1. removes any prefix up to `": "` (e.g., `"Plastics: HDPE"` is `"HDPE"`), and 2. strips any trailing parenthetical (e.g., `"HDPE (bottles)"` is `"HDPE"`). Accepted types: average, average_film, average_rigid, hdpe, ldpe, lldpe, pet, pp, ps, pvc. Unknown names are ignored (treated as zero).
waste	Logical. If TRUE, the same tonnages as use are sent to the chosen waste route. If FALSE, no waste is applied.
waste_disposal	One of `"Landfill"`, `"Open-loop"`, `"Closed-loop"`, or `"Combustion"`. Applied to all waste. If a plastic lacks a factor for the chosen route, behaviour depends on strict.
units	Output units: `"kg"` (default) or `"tonnes"`.
value_col	Which numeric column to use in uk_gov_data: `"value"` or `"value_2024"`.
strict	If TRUE (default), error when any nonzero use or implied waste requires a factor that is absent in the table. If FALSE, treat missing factors as zero.

Details

Material use: Plastics generally use "Primary material production" as the Column Text. This function always uses that for material-use factors.

Waste disposal: Factors are taken from the specified waste_disposal route. Availability varies by plastic type and year; this is enforced by the presence/absence of rows in uk_gov_data. Missing pairs error under strict = TRUE or contribute zero under strict = FALSE.

Units: Factors are kg CO2e per tonne; if units = "tonnes", the total is divided by 1000.

Value

Numeric total emissions in requested units.

Examples

```
# 1) Basic: primary material production + landfill; waste tonnage = use
plastic_emissions(
  use = c(average_plastics = 100, hdpe = 50, pet = 25),  # tonnes
  waste_disposal = "Landfill",
  waste = TRUE,
  units = "kg"
)

# 2) Choose 2024 factors and report in tonnes; no waste applied
plastic_emissions(
  use = c(average_plastic_film = 10, average_plastic_rigid = 5, pp = 2),
  waste_disposal = "Closed-loop",
  waste = FALSE,
  value_col = "value_2024",
  units = "tonnes"
)

# 3) Strict behaviour: error if a required waste route is unavailable
## Not run:
plastic_emissions(
  use = c(ps = 1),
  waste_disposal = "Combustion",
  waste = TRUE,
  strict = TRUE
)
## End(Not run)
# Tolerant: treat missing waste factors as 0
plastic_emissions(
  use = c(ps = 1),
  waste_disposal = "Combustion",
  waste = TRUE,
  strict = FALSE
)
```

`rail_emissions` *Calculate CO2e emissions from a train journey*

Description

A function that calculates CO2e emissions between train stations in the UK.

Usage

```
rail_emissions(
  from = NULL,
  to = NULL,
  via = NULL,
  distance = 0,
  num_people = 1,
  times_journey = 1,
  include_WTT = TRUE,
  round_trip = FALSE
)
```

Arguments

`from`	Station departing from.
`to`	Station arriving to
`via`	Optional. Takes a vector containing the stations the train travels via.
`distance`	Distance travelled in kilometres (default `0`. Ignored if values are given in `from` and `to`).
`num_people`	Number of people taking the journey. Takes a single numerical value.
`times_journey`	Number of times the journey is taken.
`include_WTT`	logical. Recommended `TRUE`. Whether to include emissions associated with extracting, refining, and transporting fuels.
`round_trip`	Whether the journey is one-way or return.

Details

The distances are calculated using the Haversine formula. This is calculated as the crow flies. As a result, inputting the "via" journeys will make for a more reliable function.

Value

Returns CO2e emissions in tonnes for the train journey.

Examples

```
# Emissions for a train journey between Southampton Central and
# Manchester Piccadilly Station
rail_emissions("Southampton Central", "Manchester Piccadilly")

# Emissions for a train journey between Bristol Temple Meads and
# London Paddington via Bath, Swindon, and Reading
# Use the \code{rail_finder} function to find the name of London Paddington
rail_finder(region = "London")
# Then calculate emissions
rail_emissions("Bristol Temple Meads", "Paddington", via = c("Bath Spa",
"Swindon", "Reading"))

# Alternatively state the distance of a journey
rail_emissions(distance = 100)
```

---

rail_finder *Find the station code for a train station*

Description

Find the name, area, and code of a train station in the UK. For use in the `rail_emissions` function.

Usage

```
rail_finder(
  station,
  region,
  county,
  district,
  station_code,
  distance = 0.1,
  ignore.case = FALSE
)
```

Arguments

`station`	Name of train station.
`region`	Region the train station is in. One of `c("London", "Scotland", "Wales - Cymru", "North West", "West Midlands", "North East", "East", "South East", "East Midlands", "Yorkshire And The Humber", "South West", NA)`.
`county`	County the train station is in.
`district`	District the train station is in.
`station_code`	Code of the train station.
`distance`	Maximum distance allowed for a match between the name/country/city given, and that of the value in the data set.
`ignore.case`	If `FALSE`, the check for is case-sensitive. If `TRUE`, case is ignored.

Value

Data frame containing the station code, station name, region, county, district, latitude, and longitude of a train station in the UK.

Examples

```
# Can get the station code from the station. Gets similar matches.
rail_finder(station = "Bristo")

# Can get the code from the station and city.
rail_finder(station = "Bristo", county = "Bristol")

# Can find the name and district of a train station given the IATA code
rail_finder(station_code = "BRI")
```

---

raw_fuels *Raw Fuels Emissions*

Description

Raw Fuels Emissions

Usage

```
raw_fuels(
  num_people = 1,
  butane = 0,
  CNG = 0,
  LPG = 0,
  LNG = 0,
  natural_gas = 0,
  natural_gas_mineral = 0,
  other_petroleum_gas = 0,
  propane = 0,
  aviation = 0,
  aviation_fuel = 0,
  burning_oil = 0,
  diesel = 0,
  diesel_mineral = 0,
  fuel_oil = 0,
  gas_oil = 0,
  lubricants = 0,
  naptha = 0,
  petrol_biofuel = 0,
  petrol_mineral = 0,
  residual_oil = 0,
  distillate = 0,
```

```
refinery_miscellaneous = 0,
waste_oils = 0,
marine_gas = 0,
marine_fuel = 0,
coal_industrial = 0,
coal_electricity_gen = 0,
coal_domestic = 0,
coking_coal = 0,
petroleum_coke = 0,
coal_home_produced_gen = 0,
bioethanol = 0,
biodiesel = 0,
biomethane = 0,
biodiesel_cooking_oil = 0,
biodiesel_tallow = 0,
biodiesel_HVO = 0,
biopropane = 0,
bio_petrol = 0,
renewable_petrol = 0,
wood_log = 0,
wood_chips = 0,
wood_pellets = 0,
grass = 0,
biogas = 0,
landfill_gas = 0,
butane_units = c("kwh", "litres", "tonnes"),
CNG_units = c("kwh", "litres", "tonnes"),
LPG_units = c("kwh", "litres", "tonnes"),
LNG_units = c("kwh", "litres", "tonnes"),
natural_gas_units = c("kwh", "cubic metres", "tonnes"),
natural_gas_mineral_units = c("kwh", "cubic metres", "tonnes"),
other_petroleum_gas_units = c("kwh", "litres", "tonnes"),
propane_units = c("kwh", "litres", "tonnes"),
aviation_units = c("kwh", "litres", "tonnes"),
aviation_fuel_units = c("kwh", "litres", "tonnes"),
burning_oil_units = c("kwh", "litres", "tonnes"),
diesel_units = c("kwh", "litres", "tonnes"),
diesel_mineral_units = c("kwh", "litres", "tonnes"),
fuel_oil_units = c("kwh", "litres", "tonnes"),
gas_oil_units = c("kwh", "litres", "tonnes"),
lubricants_units = c("kwh", "litres", "tonnes"),
naptha_units = c("kwh", "litres", "tonnes"),
petrol_biofuel_units = c("kwh", "litres", "tonnes"),
petrol_mineral_units = c("kwh", "litres", "tonnes"),
residual_oil_units = c("kwh", "litres", "tonnes"),
distillate_units = c("kwh", "litres", "tonnes"),
refinery_miscellaneous_units = c("kwh", "litres", "tonnes"),
waste_oils_units = c("kwh", "tonnes"),
```

```
  marine_gas_units = c("kwh", "tonnes"),
  marine_fuel_units = c("kwh", "tonnes"),
  coal_industrial_units = c("kwh", "tonnes"),
  coal_electricity_gen_units = c("kwh", "tonnes"),
  coal_domestic_units = c("kwh", "tonnes"),
  coking_coal_units = c("kwh", "tonnes"),
  petroleum_coke_units = c("kwh", "tonnes"),
  coal_home_produced_gen_units = c("kwh", "tonnes"),
  bioethanol_units = c("litres", "GJ", "kg"),
  biodiesel_units = c("litres", "GJ", "kg"),
  biomethane_units = c("litres", "GJ", "kg"),
  biodiesel_cooking_oil_units = c("litres", "GJ", "kg"),
  biodiesel_tallow_units = c("litres", "GJ", "kg"),
  biodiesel_HVO_units = c("litres", "GJ", "kg"),
  biopropane_units = c("litres", "GJ", "kg"),
  bio_petrol_units = c("litres", "GJ", "kg"),
  renewable_petrol_units = c("litres", "GJ", "kg"),
  wood_log_units = c("kwh", "tonnes"),
  wood_chips_units = c("kwh", "tonnes"),
  wood_pellets_units = c("kwh", "tonnes"),
  grass_units = c("kwh", "tonnes"),
  biogas_units = c("kwh", "tonnes"),
  landfill_gas_units = c("kwh", "tonnes")
)
```

Arguments

num_people	Number of people to account for.
butane	amount of Butane used.
CNG	amount used. Compressed natural gas (CNG). A compressed version of the natural gas used in homes. An alternative transport fuel.
LPG	amount used. Liquid petroleum gas. Used to power cooking stoves or heaters off-grid and fuel some vehicles (e.g. fork-lift trucks and vans).
LNG	amount used. Liquefied natural gas. An alternative transport fuel.
natural_gas	amount used. Standard natural gas received through the gas mains grid network in the UK.
natural_gas_mineral	amount used. Natural gas (100% mineral blend) factor is natural gas not obtained through the grid and therefore does not contain any biogas content. It can be used for calculating bespoke fuel mixtures.
other_petroleum_gas	amount used. Consists mainly of ethane, plus other hydrocarbons, (excludes butane and propane).
propane	amount used.
aviation	amount used. Fuel for piston-engined aircraft - a high octane petrol (aka AVGAS).

`aviation_fuel` amount used. Fuel for turbo-prop aircraft and jets (aka jet fuel). Similar to kerosene used as a heating fuel, but refined to a higher quality.

`burning_oil` amount used. Main purpose is for heating/lighting on a domestic scale (also known as kerosene).

`diesel` amount used. Standard diesel bought from any local filling station (across the board forecourt fuel typically contains biofuel content).

`diesel_mineral` amount used. Diesel that has not been blended with biofuel (non-forecourt diesel).

`fuel_oil` amount used. Heavy oil used as fuel in furnaces and boilers of power stations, in industry, for industrial heating and in ships.

`gas_oil` amount used. Medium oil used in diesel engines and heating systems (also known as red diesel).

`lubricants` amount used. Waste petroleum-based lubricating oils recovered for use as fuels

`naptha` amount used. A product of crude oil refining - often used as a solvent.

`petrol_biofuel` amount used. Standard petrol bought from any local filling station (across the board forecourt fuel typically contains biofuel content).

`petrol_mineral` amount used. Petrol that has not been blended with biofuel (non forecourt petrol).

`residual_oil` amount used. Waste oils meeting the 'residual' oil definition contained in the 'Processed Fuel Oil Quality Protocol'.

`distillate` amount used. Waste oils meeting the 'distillate' oil definition contained in the 'Processed Fuel Oil Quality Protocol'.

`refinery_miscellaneous` amount used. Includes aromatic extracts, defoament solvents and other minor miscellaneous products

`waste_oils` amount used. Recycled oils outside of the 'Processed Fuel Oil Quality Protocol' definitions.

`marine_gas` amount used. Distillate fuels are commonly called "Marine gas oil". Distillate fuel is composed of petroleum fractions of crude oil that are separated in a refinery by a boiling or "distillation" process.

`marine_fuel` amount used. Residual fuels are called "Marine fuel oil". Residual fuel or "residuum" is the fraction that did not boil, sometimes referred to as "tar" or "petroleum pitch".

`coal_industrial` amount used. Coal used in sources other than power stations and domestic use.

`coal_electricity_gen` amount used. Coal used in power stations to generate electricity.

`coal_domestic` amount used. Coal used domestically.

`coking_coal` amount used. Coke may be used as a heating fuel and as a reducing agent in a blast furnace.

`petroleum_coke` amount used. Normally used in cement manufacture and power plants.

coal_home_produced_gen
: amount used. Coal used in power stations to generate electricity (only for coal produced in the UK).

bioethanol
: amount used. Renewable fuel derived from common crops (such as sugar cane and sugar beet).

biodiesel
: amount used. Renewable fuel almost exclusively derived from common natural oils (for example, vegetable oils).

biomethane
: amount used. The methane constituent of biogas. Biogas comes from anaerobic digestion of organic matter.

biodiesel_cooking_oil
: amount used. Renewable fuel almost exclusively derived from common natural oils (such as vegetable oils).

biodiesel_tallow
: amount used. Renewable fuel almost exclusively derived from common natural oils (such as vegetable oils).

biodiesel_HVO
: amount used.

biopropane
: amount used.

bio_petrol
: amount used.

renewable_petrol
: amount used.

wood_log
: amount used.

wood_chips
: amount used.

wood_pellets
: amount used. Compressed low quality wood (such as sawdust and shavings) made into pellet form

grass
: amount used.

biogas
: amount used. A naturally occurring gas from the anaerobic digestion of organic materials (such as sewage and food waste), or produced intentionally as a fuel from the anaerobic digestion of biogenic substances (such as energy crops and agricultural residues).

landfill_gas
: amount used. Gas collected from a landfill site. This may be used for electricity generation, collected and purified for use as a transport fuel, or be flared off

butane_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

CNG_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

LPG_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

LNG_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

natural_gas_units
: units that the gas is given in. Options are "tonnes", "cubic metres", "kwh".

natural_gas_mineral_units
: units that the gas is given in. Options are "tonnes", "cubic metres", "kwh".

other_petroleum_gas_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

propane_units
: units that the gas is given in. Options are "tonnes", "litres", "kwh".

aviation_units units that the fuel is given in. Options are "tonnes", "litres", "kwh".

aviation_fuel_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

burning_oil_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

diesel_units units that the fuel is given in. Options are "tonnes", "litres", "kwh".

diesel_mineral_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

fuel_oil_units units that the fuel is given in. Options are "tonnes", "litres", "kwh".

gas_oil_units units that the fuel is given in. Options are "tonnes", "litres", "kwh".

lubricants_units
: units that the fuel is given in. Options are "tonnes", "kwh".

naptha_units units that the fuel is given in. Options are "tonnes", "litres", "kwh".

petrol_biofuel_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

petrol_mineral_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

residual_oil_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

distillate_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

refinery_miscellaneous_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

waste_oils_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

marine_gas_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

marine_fuel_units
: units that the fuel is given in. Options are "tonnes", "litres", "kwh".

coal_industrial_units
: units that the fuel is given in. Options are "kwh", "tonnes".

coal_electricity_gen_units
: units that the fuel is given in. Options are "kwh", "tonnes".

coal_domestic_units
: units that the fuel is given in. Options are "kwh", "tonnes".

coking_coal_units
: units that the fuel is given in. Options are "kwh", "tonnes".

petroleum_coke_units
: units that the fuel is given in. Options are "kwh", "tonnes".

coal_home_produced_gen_units
: units that the fuel is given in. Options are "kwh", "tonnes".

bioethanol_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biodiesel_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biomethane_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biodiesel_cooking_oil_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biodiesel_tallow_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biodiesel_HVO_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

biopropane_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

bio_petrol_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

renewable_petrol_units
: units that the biofuel is given in. Options are "litres", "GJ", "kg".

wood_log_units
: units that the biomass is given in. Options are "tonnes", "kwh".

wood_chips_units
: units that the biomass is given in. Options are "tonnes", "kwh".

wood_pellets_units
: units that the biomass is given in. Options are "tonnes", "kwh".

grass_units
: units that the biomass is given in. Options are "tonnes", "kwh".

biogas_units
: units that the biogas is given in. Options are "tonnes", "kwh".

landfill_gas_units
: units that the biogas is given in. Options are "tonnes", "kwh".

Details

This function calculates CO2e emissions from a wide variety of fuels, considering different unit conversions for each type of fuel. It supports the calculation of emissions from commonly used fuels such as diesel, petrol, natural gas, and biodiesel, as well as more specific fuels like aviation fuel, marine fuel, and landfill gas.

Unit conversions are done internally based on the specified units for each type of fuel (e.g., kWh, litres, tonnes). The function is useful for assessing the carbon footprint associated with different fuel sources over a specified time period.

Value

A data frame with calculated emissions in tonnes of CO2e for each type of fuel input.

References

- DEFRA Conversion Factors for Greenhouse Gas (GHG) Reporting: https://www.gov.uk/government/collections/govern conversion-factors-for-company-reporting
- Descriptions from 2021 UK Government Report: https://www.gov.uk/government/publications/greenhouse-gas-reporting-conversion-factors-2021

Examples

```
# Calculate emissions for 100 litres of diesel and 500 kWh of natural gas:
raw_fuels(
  diesel = 100, diesel_units = "litres",
  natural_gas = 500, natural_gas_units = "kwh",
)

# Calculate emissions for 10 tonnes of aviation fuel:
raw_fuels(
  aviation_fuel = 10, aviation_fuel_units = "tonnes",
)
```

---

relative_gti *Relative GTI Plot*

Description

Calculate and plot the relative growth to index (GTI) over time

Usage

```
relative_gti(
  data = data,
  time = time,
  date_format = c("%d/%m/%Y"),
  name = theatre,
  val = emissions,
  gti_by = c("default", "month", "year")
)
```

Arguments

data	The data frame containing the data.
time	The variable representing the time dimension.
date_format	The date format for the time variable (optional, default: c("%d/%m/%Y")).
name	The variable representing the grouping variable.
val	The variable representing the value.
gti_by	The grouping type for calculating the GTI ("default", "month", "year").

Value

A ggplot2 object showing the relative GTI (Growth to Index) over time.

Examples

```
# Example dataset
emission_data <- data.frame(
  theatre = c("Theatre A", "Theatre A", "Theatre B", "Theatre B", "Theatre A", "Theatre B"),
   emissions = c(200, 250, 150, 180, 300, 220),
  date = c("01/01/2023", "01/02/2023", "01/01/2023", "01/02/2023", "01/03/2023", "01/03/2023")
   )

# Using the relative_gti function
relative_gti_plot <- relative_gti(
  data = emission_data,
  time = date,
  date_format = "%d/%m/%Y",  # Date format used in the dataset
  name = theatre,
  val = emissions,
  gti_by = "default"  # Calculating based on default time period
)

# Plot the relative GTI
print(relative_gti_plot)
```

---

seaports *Dataset of different seaports*

Description

A dataset containing the country, country code, city, city code, and coordinates of seaports

Usage

```
data(seaports)
```

Format

A data frame with 8736 rows and 7 variables:

country Name of the country with the port

city Name of the city with the port

country_code Code of the country name

port_code Code of the city port

latitude Latitude of the port

longitude Longitude of the port

Source

https://www.kaggle.com/therohk/world-seaport-airport-dataset-and-codes/script

seaport_finder *Check the code or name of a seaport*

Description

Find the name and/or code of a seaport. For use in the `ferry_emissions` function.

Usage

```
seaport_finder(city, country, port_code, distance = 0.1, ignore.case = FALSE)
```

Arguments

`city`	Name of the city.
`country`	Name of the country.
`port_code`	Name of the port.
`distance`	Maximum distance allowed for a match between the country/city given, and that of the value in the data set.
`ignore.case`	If `FALSE`, the check is case-sensitive. If `TRUE`, case is ignored.

Value

Data frame containing the country, city, country code, port code, latitude, and longitude of a seaport.

Examples

```
# Look up the city of Aberdeen to find the port_code for it
seaport_finder(city = "Aberdeen")

# Search for a country and city and it finds matches
seaport_finder(country = "United", city = "borunemouth", ignore.case = TRUE)
```

shiny_emissions *Run 'shiny' App to Calculate Carbon Emissions*

Description

Runs a GUI to the functions in the 'carbonr' package to calculate carbon-equivalent emissions.

Usage

```
shiny_emissions()
```

Value

'shiny' app to calculate carbon-equivalent emissions

Examples

```
if(interactive()){shiny_emissions()}
```

`stations` *Dataset of UK train stations*

Description

A dataset containing the city, station code, and coordinates of seaports

Usage

```
data(stations)
```

Format

A data frame with 2608 rows and 4 variables:

station Name of the station

station_code Code of the station

region Region of the station. One of `"London"`, `"Scotland"`, `"Wales - Cymru"`, `"North West"`, `"West Midlands"`, `"North East"`, `"East"`, `"South East"`, `"East Midlands"`, `"Yorkshire And The Humber"`, `"South West"`, `NA`

county County of the station

district District of the station

latitude Latitude of the station

longitude Longitude of the station

Source

https://www.data.gov.uk/dataset/ff93ffc1-6656-47d8-9155-85ea0b8f2251/naptan

https://www.theguardian.com/news/datablog/2011/may/19/train-stations-listed-rail#data

`total_output` *Calculate Total Output and Generate Plot*

Description

This function calculates the total output and generates a plot based on the specified parameters.

Usage

```
total_output(
  data = x$data,
  time = time,
  date_format = c("%d/%m/%Y"),
  name = theatre,
  val = carbon_price_credit,
  plot_by = c("default", "month", "year")
)
```

Arguments

`data`	The data frame containing the data.
`time`	The variable representing the time dimension.
`date_format`	The date format for the time variable (optional, default: c("%d/%m/%Y")).
`name`	The variable representing the grouping variable.
`val`	The variable to calculate the total output for (default: carbon_price_credit).
`plot_by`	The grouping type for the total output plot ("default", "month", "year").

Details

This function calculates the total output by grouping the data based on the specified parameters (grouping variable and time dimension). It then summarises the specified variable (CPI or emissions) using the sum function. The resulting data is used to create a line plot showing the total output over time, with each group represented by a different color. The plot can be grouped by the default grouping, month, or year, based on the plot_by parameter.

Value

A ggplot object showing the total output plot.

vehicle_emissions *Calculate CO2e emissions from land-travel journeys*

Description

A function that calculates CO2e emissions on a journey on land.

Usage

```
vehicle_emissions(
  distance,
  units = c("miles", "km"),
  num = 1,
  vehicle = c("Cars", "Motorbike"),
  fuel = c("Petrol", "Diesel", "Unknown", "Battery Electric Vehicle",
    "Plug-in Hybrid Electric Vehicle", "Hybrid", "CNG", "LPG"),
  car_type = c("Average car", "Small car", "Medium car", "Large car", "Mini",
   "Supermini", "Lower medium", "Upper medium", "Executive", "Luxury", "Sports",
    "Dual purpose 4X4", "MPV"),
  bike_type = c("Average", "Small", "Medium", "Large"),
  TD = TRUE,
  include_WTT = TRUE,
  include_electricity = TRUE,
  owned_by_org = TRUE
)
```

Arguments

distance	Distance in km or miles of the journey made (this can be calculated with other tools, such as google maps.).
units	Units for the distance travelled. Options are "km" or "miles".
num	Number of vehicles used.
vehicle	Vehicle used for the journey. Options are "Cars", "Motorbike".
fuel	Fuel type used for the journey. For car, "Petrol", "Diesel", "Unknown", "Battery Electric Vehicle", "Plug-in Hybrid Electric Vehicle" are options. Additionally "Hybrid" is an option if the car_type is one of "Average car", "Small car", "Medi Additionally "CNG" and "LPG" are options if the car_type is one of "Average car", "Medium car", "Lar Note that "Plug-in Hybrid Electric Vehicle" does not give values if car_type == "Mini" "hybrid electric" and "battery electric" account for electricity kWh emissions.
car_type	Size/type of vehicle for car. Options are c("Average car", "Small car", "Medium car", "Large car "Mini", "Supermini", "Lower medium", "Upper medium", "Executive", "Luxury", "Sports", "Dual purpose 4X4", "MPV"),. Small denotes up to a 1.4L engine, unless diesel which is up to 1.7L engine. Medium denotes 1.4-2.0L for petrol cars, 1.7-2.0L for diesel cars. Large denotes 2.0L+ engine.

bike_type	Size of vehicle for motorbike. Options are "Small", "Medium", "Large", or "Average". Sizes denote upto 125cc, 125cc-500cc, 500cc+ respectively.
TD	logical.Whether to account for transmission and distribution (TD) for electric vehicles (only car and van)
include_WTT	logical. Well-to-tank (include_WTT) - whether to account for emissions associated with extraction, refining and transportation of the fuels (for non-electric vehicles).
include_electricity	logical. Whether to account for ... for electric vehicles (car and van).
owned_by_org	logical. Whether the vehicle used is owned by the organisation or not (only for car, motorbike).

Value

Tonnes of CO2e emissions per mile travelled.

Examples

```
# Emissions for a 100 mile car journey
 vehicle_emissions(distance = 100)

# Emissions for a 100 mile motorbike journey where the motorbike is 500+cc
 vehicle_emissions(distance = 100, vehicle = "Motorbike", bike_type = "Large")
```

Index