Package 'sensortowerR'

December 30, 2025

Title Interface to 'Sensor Tower' Mobile App Intelligence API

Version 0.9.4

Description Interface to the 'Sensor Tower' API <https://app.sensortower.com/api/docs/app_analysis> for mobile app analytics and market intelligence. Provides functions to retrieve app metadata, publisher information, download and revenue estimates, active user metrics, category rankings, and market trends. The package includes data processing utilities to clean and aggregate metrics across platforms, automatic app name resolution, and tools for generating professional analytics dashboards. Supports both iOS and Android app ecosystems with unified data structures for cross-platform analysis.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, glue, httr, httr2, jsonlite, lubridate, openssl, purrr, rlang, stats, stringr, tibble, tidyr, utils

Suggests gt, gtExtras, knitr, pkgbuild, rcmdcheck, rmarkdown, rhub, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Author Phillip Black [aut, cre]

Maintainer Phillip Black <pblack@gameeconomistconsulting.com>

Depends R (>= 4.1.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2025-12-30 19:20:02 UTC

Contents

`calculate_yoy_growth` *Calculate year-over-year growth rates*

Description

Helper function to calculate YoY growth rates from the output of st_yoy_metrics

Usage

```
calculate_yoy_growth(yoy_data, baseline_year = NULL)
```

Arguments

`yoy_data`	Output from st_yoy_metrics
`baseline_year`	The year to use as baseline (default: earliest year)

Value

A tibble with growth rates relative to baseline year

clean_numeric_column *Clean numeric column by removing special characters*

Description

Clean numeric column by removing special characters

Usage

```
clean_numeric_column(x)
```

Arguments

x	Vector to clean

Value

Numeric vector

filter_helpers *Custom Filter Helper Functions*

Description

Functions to validate, test, and manage Sensor Tower custom field filters

find_column *Column validation and mapping helpers*

Description

Helper functions to handle column name variations and missing columns Get available columns matching a pattern

Usage

```
find_column(data, pattern, prefer = NULL)
```

Arguments

data	Data frame to check
pattern	Regular expression pattern to match
prefer	Character vector of preferred column names (first match wins)

Value

First matching column name or NULL

formatting_helpers *Formatting Helper Functions*

Description

Functions for formatting numeric values in a human-readable way, particularly useful for revenue, download counts, and percentages.

format_arpu *Format ARPU (Average Revenue Per User)*

Description

Format ARPU (Average Revenue Per User)

Usage

```
format_arpu(val, digits = 2)
```

Arguments

val	Numeric ARPU value
digits	Number of decimal places (default: 2)

Value

Formatted ARPU string

Examples

```
format_arpu(5.234)    # "$5.23"
format_arpu(0.99)     # "$0.99"
```

format_currency *Format currency values with appropriate suffixes*

Description

Format currency values with appropriate suffixes

Usage

```
format_currency(val, digits = 2)
```

Arguments

val	Numeric value to format as currency
digits	Number of digits after decimal for millions/billions (default: 2)

Value

Formatted currency string

Examples

```
format_currency(1234567)      # "$1.23M"
format_currency(1234567890)   # "$1.23B"
format_currency(123)          # "$123"
```

format_downloads *Format download counts with appropriate suffixes*

Description

Format download counts with appropriate suffixes

Usage

```
format_downloads(val)
```

Arguments

val	Numeric value to format as downloads

Value

Formatted download count string

Examples

```
format_downloads(1234567)      # "1.2M"
format_downloads(1234567890)   # "1.2B"
```

format_large_number *Format large numbers with K/M/B suffixes*

Description

Format large numbers with K/M/B suffixes

Usage

```
format_large_number(val, digits = 1, prefix = "")
```

Arguments

val	Numeric value to format
digits	Number of digits after decimal for millions/billions (default: 1)
prefix	Optional prefix (e.g., "$" for currency)

Value

Formatted string with appropriate suffix

Examples

```
format_large_number(1234567)      # "1.2M"
format_large_number(1234567890)   # "1.2B"
format_large_number(1234567, prefix = "$") # "$1.2M"
```

format_market_share *Format market share as percentage*

Description

Format market share as percentage

Usage

```
format_market_share(val, digits = 1)
```

Arguments

val	Numeric value as decimal (0-1 scale)
digits	Number of decimal places (default: 1)

Value

Formatted market share percentage string

Examples

```
format_market_share(0.234)    # "23.4%"
format_market_share(0.05)     # "5.0%"
```

---

format_percent *Format percentages*

Description

Format percentages

Usage

```
format_percent(val, digits = 1)
```

Arguments

val	Numeric value to format as percentage (0-100 scale)
digits	Number of decimal places (default: 1)

Value

Formatted percentage string

Examples

```
format_percent(23.456)    # "23.5%"
format_percent(0.234, digits = 2)    # "0.23%"
```

---

format_retention *Format retention rates*

Description

Format retention rates

Usage

```
format_retention(val, digits = 1)
```

Arguments

val	Numeric value as decimal (0-1 scale)
digits	Number of decimal places (default: 1)

Value

Formatted retention percentage string

Examples

```
format_retention(0.234)    # "23.4%"
format_retention(0.85)     # "85.0%"
```

---

format_users *Format user counts (DAU/MAU/WAU)*

Description

Format user counts (DAU/MAU/WAU)

Usage

```
format_users(val)
```

Arguments

val	Numeric value to format as user count

Value

Formatted user count string

Examples

```
format_users(1234567)      # "1.2M"
format_users(1234)         # "1.2K"
```

---

format_vector *Create a vector of formatted values*

Description

Create a vector of formatted values

Usage

```
format_vector(
  values,
  type = c("currency", "downloads", "percent", "users", "retention", "arpu"),
  ...
)
```

Arguments

values	Numeric vector to format
type	Type of formatting: "currency", "downloads", "percent", "users"
...	Additional arguments passed to formatting function

Value

Character vector of formatted values

Examples

```
format_vector(c(1234567, 2345678), "currency")
format_vector(c(0.234, 0.456), "percent")
```

---

get_column_spec *Get column specification for common Sensor Tower metrics*

Description

Get column specification for common Sensor Tower metrics

Usage

```
get_column_spec(metric_type = NULL, time_period = NULL, region = NULL)
```

Arguments

metric_type	Type of metric (revenue, downloads, retention, etc.)
time_period	Time period (30d, 180d, alltime, etc.)
region	Region (us, ww)

Value

List of column specifications

lookup_category_names *Helper function to look up category names*

Description

Helper function to look up category names

Usage

```
lookup_category_names(category_ids, platform = "ios")
```

Arguments

category_ids	Character vector of category IDs
platform	Character string. "ios" or "android"

Value

Character vector of category names

map_region_columns *Map region-specific columns intelligently*

Description

Map region-specific columns intelligently

Usage

```
map_region_columns(data, requested_region = "US")
```

Arguments

data	Data frame with Sensor Tower data
requested_region	Region requested (e.g., "US", "WW")

Value

Data frame with mapped columns

require_column *Validate column exists with fallback options*

Description

Validate column exists with fallback options

Usage

```
require_column(data, primary, fallbacks = NULL, operation = "operation")
```

Arguments

data	Data frame
primary	Primary column name
fallbacks	Character vector of fallback column names
operation	Description of operation for error message

Value

Column name that exists, or stops with error

select_columns_safe *Select columns safely with fallbacks*

Description

Select columns safely with fallbacks

Usage

```
select_columns_safe(data, columns)
```

Arguments

data	Data frame
columns	Named list of column specifications

Value

Data frame with selected/renamed columns

select_robust *Safe column selection with automatic fallbacks*

Description

Safe column selection with automatic fallbacks

Usage

```
select_robust(data, ...)
```

Arguments

data	Data frame
...	Column specifications (can be character vectors)

Value

Data frame with selected columns

st_analyze_filter *Analyze Custom Filter Performance*

Description

Provides a summary analysis of apps matching a custom filter, including top performers, growth metrics, and category breakdown.

Usage

```
st_analyze_filter(
  filter_id,
  measure = "DAU",
  regions = "US",
  top_n = 10,
  auth_token = NULL
)
```

Arguments

filter_id	Character. The custom fields filter ID to analyze
measure	Character. Metric to analyze: "DAU", "revenue", or "units"
regions	Character vector. Region codes (default "US")
top_n	Integer. Number of top apps to show (default 10)
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

A list containing summary statistics and top apps

`st_api_diagnostics` *Diagnose API Issues*

Description

This function helps diagnose common API issues by testing various ID formats and endpoints to determine the best approach for fetching data.

Usage

```
st_api_diagnostics(
  app_id,
  verbose = TRUE,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN")
)
```

Arguments

`app_id`	Character string. The app ID to diagnose (can be unified, iOS, or Android)
`verbose`	Logical. Show detailed diagnostic output. Default is TRUE.
`auth_token`	Character string. Your Sensor Tower API authentication token.

Value

A list with diagnostic results including: - ‘id_type‘: Detected ID type - ‘platform_ids‘: Resolved platform-specific IDs - ‘endpoint_results‘: Results from testing various endpoints - ‘recommendations‘: Suggested approach for this app

Examples

```
## Not run:
# Diagnose Star Trek Fleet Command
diagnosis <- st_api_diagnostics("5ba4585f539ce75b97db6bcb")

# Check iOS app
diagnosis <- st_api_diagnostics("1427744264")

## End(Not run)
```

st_app_details *Fetch Detailed App Metadata*

Description

Retrieves comprehensive metadata for one or more apps including descriptions, screenshots, ratings, publisher information, and more. This function provides rich app store listing data for apps when you already know their IDs.

Usage

```
st_app_details(
  app_ids,
  os,
  include_developer_contacts = TRUE,
  auth_token = NULL
)
```

Arguments

app_ids	Character vector. App IDs to fetch details for. - For iOS: numeric app IDs (e.g., "553834731") - For Android: bundle IDs (e.g., "com.king.candycrushsaga") - For unified: unified app IDs Maximum 100 apps per request.
os	Character string. Required. Operating system: "ios", "android", or "unified".
include_developer_contacts	Logical. Include developer contact information (email, address). Defaults to TRUE.
auth_token	Character string. Sensor Tower API authentication token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

Value

A [tibble][tibble::tibble] containing detailed app metadata with columns: - ‘app_id‘: The app’s store ID - ‘app_name‘: The app’s display name - ‘publisher_name‘: Publisher/developer name - ‘publisher_id‘: Publisher ID - ‘categories‘: App store categories - ‘description‘: Full app description - ‘subtitle‘: App subtitle (iOS) or short description (Android) - ‘rating‘: Current average rating - ‘rating_count‘: Total number of ratings - ‘rating_current_version‘: Rating for current version - ‘rating_count_current_version‘: Rating count for current version - ‘content_rating‘: Age rating/content rating - ‘price‘: App price - ‘currency‘: Price currency - ‘release_date‘: Initial release date - ‘last_update‘: Last update date - ‘version‘: Current version - ‘size_bytes‘: App size in bytes - ‘screenshots‘: List of screenshot URLs - ‘icon_url‘: App icon URL - ‘publisher_email‘: Developer email (if include_developer_contacts = TRUE) - ‘publisher_address‘: Developer address (if include_developer_contacts = TRUE) - ‘publisher_country‘: Developer country - Additional platform-specific fields

API Endpoint Used

- ‘GET /v1/{os}/apps‘

Examples

```
## Not run:
# Get details for a single iOS app
candy_crush <- st_app_details(
  app_ids = "553834731",
  os = "ios"
)

# Get details for multiple Android apps
android_games <- st_app_details(
  app_ids = c("com.king.candycrushsaga", "com.supercell.clashofclans"),
  os = "android"
)

# Get details without developer contacts
apps <- st_app_details(
  app_ids = c("553834731", "1053012308"),
  include_developer_contacts = FALSE
)

## End(Not run)
```

---

st_app_enriched *Fetch Enriched Metrics for Specific Apps*

Description

Retrieves comprehensive metrics including retention, MAU, DAU, demographics, and other aggregate tags for specific apps by their unified app IDs.

Usage

```
st_app_enriched(
  unified_app_ids,
  os = "unified",
  regions = "WW",
  auth_token = NULL
)
```

Arguments

`unified_app_ids`
: Character vector. One or more unified app IDs (24-character hex strings). Required. Use ‘st_app_info()‘ to find these.

os	Character string. Operating system context for the request. Must be "unified" (default), "ios", or "android".
regions	Character vector. Region codes for data filtering. Defaults to "WW" (worldwide).
auth_token	Optional. Character string. Your Sensor Tower API token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

Details

This function is designed for the common workflow of: 1. Search for apps by name using ‘st_app_info()‘ 2. Get their unified IDs 3. Fetch enriched metrics for those specific apps using this function

Value

A [tibble][tibble::tibble] with enriched metrics including: - **Identification**: ‘unified_app_id‘, ‘unified_app_name‘ - **Active Users**: ‘dau_30d_us‘, ‘dau_30d_ww‘, ‘wau_4w_us‘, ‘wau_4w_ww‘, ‘mau_month_us‘, ‘mau_month_ww‘ - **Retention**: ‘retention_1d_us/ww‘, ‘retention_7d_us/ww‘, ‘retention_14d_us/ww‘, ‘retention_30d_us/ww‘, ‘retention_60d_us/ww‘ - **Demographics**: ‘genders_us‘, ‘genders_ww‘, ‘age_us‘, ‘age_ww‘, ‘male_share_us‘, ‘female_share_us‘ - **Revenue/Downloads**: ‘revenue_30d_ww‘, ‘revenue_90d_ww‘, ‘revenue_alltime_us/ww‘, ‘downloads_30d_ww‘, ‘downloads_alltime_us/ww‘ - **Monetization**: ‘rpd_alltime_us/ww‘, ‘arpu_90d_us/ww‘ - **Launch**: ‘release_date_us/ww‘, ‘earliest_release_date‘

Recommended Workflow

“‘ # Step 1: Search for apps by name apps <- st_app_info("Royal Match")

Step 2: Get unified IDs app_ids <- apps$unified_app_id

Step 3: Fetch enriched metrics metrics <- st_app_enriched(app_ids) “‘

Data Availability Notes

- **IMPORTANT: Geographic Limitations** - All enriched metrics are only available for **US market (‘_us‘ suffix)** and **Worldwide aggregates (‘_ww‘ suffix)**. Per-country data (e.g., GB, DE, FR, JP) is NOT available through this endpoint. For per-country data, use [st_sales_report()] for revenue/downloads or [st_batch_metrics()] for MAU/DAU time-series. - Retention data (D1, D7, D14, D30, D60) is aggregated for the "last quarter" - not time-series data. D90 retention is NOT available through the API. - Demographics (age/gender) are primarily available for US market only. - Not all metrics are available for all apps - smaller apps may have NA values. - This returns **snapshot data**, not time-series. For historical trends, use [st_batch_metrics()] or [st_sales_report()].

See Also

[st_app_info()] for searching apps by name, [st_app_lookup()] for resolving app IDs, [st_sales_report()] for time-series revenue/download data, [st_batch_metrics()] for time-series DAU/WAU/MAU data

Examples

```
## Not run:
# Get enriched data for specific apps
royal_match <- st_app_info("Royal Match")
enriched <- st_app_enriched(royal_match$unified_app_id)

# Get data for multiple apps at once
game_ids <- c("5f16a8019f7b275235017614", "660af7c66237390ce7c829fc")
multi_enriched <- st_app_enriched(game_ids)

# View retention metrics
multi_enriched %>%
  select(unified_app_name, starts_with("retention"))

## End(Not run)
```

---

st_app_info *Fetch Unified App or Publisher Information from Sensor Tower*

Description

This function retrieves information about apps or publishers from the Sensor Tower API based on a search term. It targets the ‘/v1/{app_store}/search_entities‘ endpoint and fetches IDs and names for unified app or publisher entities.

Usage

```
st_app_info(
  term,
  app_store = "unified",
  entity_type = "app",
  limit = 20,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  return_all_fields = FALSE
)
```

Arguments

term	Character string. The search term for the app or publisher.
app_store	Character string. The app store to search. Defaults to "unified".
entity_type	Character string. The type of entity to search for. Either "app" (default) or "publisher".
limit	Numeric. The maximum number of results to return. Defaults to 20.
auth_token	Character string. Your Sensor Tower API authentication token.
return_all_fields	Boolean. If TRUE, returns all available fields from the API response. Defaults to FALSE.

Value

A [tibble][tibble::tibble] with entity information:

For apps (‘entity_type = "app"‘): - ‘unified_app_id‘: The unified app ID (24-char hex) - ‘unified_app_name‘: The app name - ‘category_details‘: (when available) Nested tibble with category info

For publishers (‘entity_type = "publisher"‘): - ‘unified_publisher_id‘: The unified publisher ID (24-char hex) - ‘unified_publisher_name‘: The publisher name

Use the returned publisher ID with ‘st_publisher_apps()‘ to get the publisher’s apps.

Examples

```
## Not run:
# Search for an app by name
app_info <- st_app_info(term = "Clash of Clans")
print(app_info)

# Search for a publisher by name
publisher_info <- st_app_info(term = "Lilith", entity_type = "publisher")
print(publisher_info)

# Get publisher's apps
lilith_apps <- st_publisher_apps(
  unified_id = publisher_info$unified_publisher_id[1],
  aggregate_related = TRUE
)

# ---- Piping Workflow Examples ----
library(dplyr)

# Pipe-friendly workflow: Find publisher -> Get apps -> Fetch sales
lilith_sales <- st_app_info("Lilith", entity_type = "publisher") %>%
  slice(1) %>%
  pull(unified_publisher_id) %>%
  st_publisher_apps(aggregate_related = TRUE) %>%
  pull(unified_app_id) %>%
  st_unified_sales_report(
    countries = "WW",
    start_date = "2024-01-01",
    end_date = "2024-12-31",
    date_granularity = "monthly"
  )

## End(Not run)
```

---

st_app_lookup *Look up app information by any ID type*

Description

This function looks up app information using any type of app ID - unified, iOS, or Android. It returns the unified ID and platform-specific IDs that can be used with other API functions.

Usage

```
st_app_lookup(
  app_id,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = FALSE
)
```

Arguments

app_id	Character string. Can be: - Sensor Tower unified app ID (24-char hex like "5ba4585f539ce75b97db6bcb") - iOS app ID (numeric like "943599237") - Android package name (like "com.bandainamcogames.dbzdokkanww")
auth_token	Character string. Your Sensor Tower API authentication token.
verbose	Logical. Whether to show progress messages. Default is FALSE.

Details

The function automatically detects the ID type: - 24-character hex strings are treated as unified IDs - Numeric strings are treated as iOS app IDs - Strings starting with com/net/org/io are treated as Android package names

Value

A list with components: - ‘unified_app_id‘: The Sensor Tower unified app ID - ‘ios_app_id‘: iOS app ID if found - ‘android_app_id‘: Android app ID if found - ‘app_name‘: App name if found - ‘publisher_name‘: Publisher name if found Returns NULL if app cannot be found.

Examples

```
## Not run:
# Look up Star Trek Fleet Command
app_ids <- st_app_lookup("5ba4585f539ce75b97db6bcb")

# Use the IDs with st_ytd_metrics
if (!is.null(app_ids)) {
  metrics <- st_ytd_metrics(
    ios_app_id = app_ids$ios_app_id,
    android_app_id = app_ids$android_app_id,
    years = 2025,
    metrics = "revenue",
    countries = "WW"
  )
}

## End(Not run)
```

st_app_tag *Fetch Apps by Custom Fields and Tags*

Description

Retrieves apps filtered by custom fields and tags from Sensor Tower. This function uses the /v1/app_tag/apps endpoint.

Usage

```
st_app_tag(
  app_id_type,
  custom_fields_filter_id,
  name = NULL,
  value = NULL,
  global = TRUE,
  last_known_id = NULL,
  auth_token = NULL,
  base_url = "https://api.sensortower.com"
)
```

Arguments

app_id_type	Character string. Operating System. Must be one of "itunes" (iOS) or "unified". Required.
custom_fields_filter_id	Character string. ID of a Sensor Tower custom field filter. Required. Use the filter ID from relevant endpoint.
name	Optional. Character string. Name of Custom or Global Field. Defaults to "Stock Ticker".
value	Optional. Character string. Tag value for custom or global field provided. Leave blank to fetch all possible apps.
global	Optional. Logical. Filter by global or organization custom fields. Defaults to TRUE (false means organization custom fields).
last_known_id	Optional. Character string. Supply last_known_id from previous request to get next page. Leave blank to get first page.
auth_token	Optional. Character string. Your Sensor Tower API token.
base_url	Optional. Character string. The base URL for the API.

Value

A [tibble][tibble::tibble] with app data including IDs and metadata.

st_batch_metrics *Batch Fetch Metrics for Multiple Apps*

Description

Efficiently fetch metrics for multiple apps by batching API calls and automatically handling platform-specific requirements. The OS parameter controls which platform's data is returned for all apps.

Usage

```
st_batch_metrics(
  os,
  app_list,
  metrics = c("revenue", "downloads"),
  date_range = list(start_date = Sys.Date() - 90, end_date = Sys.Date() - 1),
  countries,
  granularity,
  parallel = FALSE,
  cache_dir = NULL,
  verbose = TRUE,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  max_cores = 2,
  max_concurrent_requests = 2,
  retry = TRUE,
  max_retries = 3,
  publisher_ids = NULL
)
```

Arguments

os	Character. Required. Operating system: "ios", "android", or "unified". This determines which platform's data is returned for all apps.
app_list	List or data frame containing app information. Can be: - Character vector of app IDs - Data frame with columns: app_id, app_name (optional), platform (optional) - List of lists with app_id and optional metadata
metrics	Character vector. Metrics to fetch. Supported values: - "revenue" - App revenue estimates - "downloads" - App download estimates - "dau" - Daily Active Users - "wau" - Weekly Active Users - "mau" - Monthly Active Users
date_range	List with start_date and end_date, or "ytd" for year-to-date
countries	Character vector. Country codes. Required.
granularity	Character. Date granularity (default "monthly")
parallel	Logical. Use parallel processing (default FALSE)
cache_dir	Character. Directory for caching results (optional)
verbose	Logical. Show progress messages (default TRUE)

auth_token	Character string. Your Sensor Tower API authentication token.
max_cores	Integer. Maximum number of cores to use for parallel processing.
max_concurrent_requests	Integer. Max concurrent requests (deprecated/unused).
retry	Logical. Whether to retry failed requests.
max_retries	Integer. Max retries.
publisher_ids	Character vector. Publisher IDs to fetch data for.

Value

A tibble with all metrics for all apps.

---

st_build_filter_url *Build Sensor Tower Filter URL*

Description

Constructs a URL for the Sensor Tower web interface where you can create custom filters. Visit this URL, configure your filters, and then copy the custom_fields_filter_id from the resulting URL.

Usage

```
st_build_filter_url(
  os = "unified",
  category = NULL,
  countries = NULL,
  base_url = "https://app.sensortower.com/top-charts"
)
```

Arguments

os	Character string. Operating system filter. One of "ios", "android", or "unified". Defaults to "unified".
category	Optional. Category ID to pre-select.
countries	Optional. Character vector of country codes to pre-select.
base_url	Character string. Base URL for Sensor Tower. Defaults to "https://app.sensortower.com/top-charts".

Value

Character string. The constructed URL.

Examples

```
## Not run:
# Build URL for iOS games in US
url <- st_build_filter_url(os = "ios", category = 6014, countries = "US")

# Open in browser
browseURL(url)

## End(Not run)
```

`st_build_web_url` *Create Sensor Tower Web URL from Parameters*

Description

Builds a Sensor Tower web interface URL from API parameters. This is the reverse of st_parse_web_url().

Usage

```
st_build_web_url(
  os = "unified",
  measure = "revenue",
  category = NULL,
  regions = "US",
  start_date = NULL,
  end_date = NULL,
  custom_fields_filter_id = NULL,
  custom_tags_mode = NULL,
  ...
)
```

Arguments

`os`	Operating system
`measure`	Measure type
`category`	Category ID
`regions`	Region codes (converted to country parameters)
`start_date`	Start date
`end_date`	End date
`custom_fields_filter_id`	Custom filter ID
`custom_tags_mode`	Custom tags mode
`...`	Additional parameters

Value

Character string URL

`st_cache_info` *Show App ID Cache Statistics*

Description

Display information about the current app ID cache

Usage

```
st_cache_info()
```

Value

No return value, called for side effects (displaying cache statistics).

`st_categories` *List Available Sensor Tower Categories*

Description

Returns a tibble of app categories recognized by the Sensor Tower API, mapping category IDs to category names for different platforms (iOS/Android). Useful for finding valid inputs for the ‘category‘ parameter in other functions.

Usage

```
st_categories(platform = NULL)
```

Arguments

`platform`	Optional character string. Filter results for a specific platform ("ios" or "android"). If NULL (default), returns categories for both platforms.

Value

A tibble with columns ‘platform‘ (character, "ios" or "android"), ‘category_id‘ (character, e.g., "6014"), and ‘category_name‘ (character, e.g., "Games").

Examples

```
# Get all categories
all_cats <- st_categories()
head(all_cats)

# Get only iOS categories
ios_cats <- st_categories(platform = "ios")
head(ios_cats)

# Find game categories on iOS
ios_games <- subset(st_categories("ios"), grepl("Game", category_name))
head(ios_games)
```

---

st_category_rankings *Fetch App Store Category Rankings*

Description

Retrieves the top ranking apps for a specific category and chart type from the App Store or Google Play Store. This provides the official store rankings as they appear in the actual app stores.

Usage

```
st_category_rankings(
  os,
  category = NULL,
  chart_type = NULL,
  country = "US",
  date = NULL,
  limit = 100,
  offset = 0,
  custom_fields_filter_id = NULL,
  custom_tags_mode = NULL,
  auth_token = NULL
)
```

Arguments

os	Character string. Required. Operating system: "ios", "android", or "unified".
category	Character or numeric. Category ID to fetch rankings for. Use ‘st_categories()‘ to find valid category IDs. Required unless ‘custom_fields_filter_id‘ is provided.
chart_type	Character string. The chart type to retrieve. Options vary by OS: - iOS: "topfreeapplications", "toppaidapplications", "topgrossingapplications", etc. - Android: "topselling_free", "topselling_paid", "topgrossing", etc. Defaults to "topfreeapplications" for iOS, "topselling_free" for Android.
country	Character string. Two-letter country code (e.g., "US", "GB"). Defaults to "US".

`date` Date or character string in "YYYY-MM-DD" format. Date for rankings. Defaults to NULL (uses today's date).

`limit` Numeric. Number of results to return (1-400). Defaults to 100.

`offset` Numeric. Offset for pagination. Defaults to 0.

`custom_fields_filter_id` Optional. Character string. ID of a Sensor Tower custom field filter to apply. Use filter IDs from the web interface at app.sensortower.com. When provided, this filter will be applied to the results. The 'category' parameter becomes optional when using a custom filter.

`custom_tags_mode` Optional. Character string. Required if 'os' is 'unified' and 'custom_fields_filter_id' is provided. Specifies how the custom filter applies to unified apps. Options: "include", "exclude", "include_unified_apps". The "include_unified_apps" option includes all platform versions when any version matches the filter.

`auth_token` Character string. Sensor Tower API authentication token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

Value

A [tibble][tibble::tibble] containing ranking data with columns: - 'rank': The app's position in the chart - 'app_id': The app's store ID - 'category': The category ID - 'country': The country code - 'date': The ranking date - 'chart_type': The chart type - 'os': The operating system

API Endpoint Used

- 'GET /v1/{os}/ranking'

Note

The API returns only app IDs, not names. To get app names and other metadata, use the app IDs with 'st_app_details()'.

Examples

```
## Not run:
# Get top free games in the US
top_games <- st_category_rankings(
  os = "ios",
  category = 6014,  # Games category
  chart_type = "topfreeapplications",
  country = "US",
  limit = 50
)

# Get top grossing apps in UK for a specific date
top_grossing <- st_category_rankings(
  os = "android",
  category = "game",
  chart_type = "topgrossing",
```

```
  country = "GB",
  date = "2024-01-15",
  limit = 100
)

# Use custom filter instead of category
filtered_rankings <- st_category_rankings(
  os = "ios",
  custom_fields_filter_id = "60746340241bc16eb8a65d76",
  chart_type = "topgrossingapplications",
  country = "US",
  limit = 50
)

# With unified OS and custom filter
unified_rankings <- st_category_rankings(
  os = "unified",
  custom_fields_filter_id = "60746340241bc16eb8a65d76",
  custom_tags_mode = "include_unified_apps",
  chart_type = "topfreeapplications"
)

## End(Not run)
```

`st_clear_app_cache` *Clear App Name Cache*

Description

Clears the internal cache of app name lookups. Useful for testing or when you want to refresh app name data.

Usage

```
st_clear_app_cache()
```

Value

No return value, called for side effects (clearing the cache).

st_clear_id_cache *Clear App ID Cache*

Description

Clears the in-memory and on-disk cache of app ID mappings

Usage

```
st_clear_id_cache(disk = TRUE)
```

Arguments

disk	Logical. Also remove the on-disk cache file (default TRUE).

Value

No return value, called for side effects (clearing the cache).

st_combine_filters *Combine Multiple Filters*

Description

Creates a compound filter by combining multiple filter criteria.

Usage

```
st_combine_filters(filter_ids, operator = c("AND", "OR"), auth_token = NULL)
```

Arguments

filter_ids	Character vector. Existing filter IDs to combine
operator	Character. How to combine filters ("AND" or "OR")
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the combined filter ID

Note

OR operations may not be supported by all Sensor Tower endpoints

Examples

```
## Not run:
# Get Word Puzzle games (AND operation)
word_filter <- st_filter_by_genre(sub_genres = "Word")
puzzle_filter <- st_filter_by_genre(genres = "Puzzle")
combined <- st_combine_filters(
  filter_ids = c(word_filter, puzzle_filter),
  operator = "AND"
)

## End(Not run)
```

---

st_compare_filter_results

Compare Results With and Without Custom Filter

Description

Compares the results of API calls with a custom filter versus a standard category filter to understand what the custom filter is doing.

Usage

```
st_compare_filter_results(
  filter_id,
  category = 6014,
  os = "ios",
  regions = "US",
  limit = 20
)
```

Arguments

filter_id	Character string. The custom filter ID to test
category	Category ID to use for comparison
os	Character string. Operating system. Defaults to "ios".
regions	Character string. Region code. Defaults to "US".
limit	Integer. Number of results to fetch. Defaults to 20.

Value

List containing both result sets and comparison statistics

st_create_simple_filter *Create or Get Filter ID for Custom Criteria*

Description

Creates a custom fields filter or retrieves existing filter ID if the same criteria already exists. This is a convenience wrapper around st_custom_fields_filter.

Usage

```
st_create_simple_filter(
  field_name,
  field_values,
  global = TRUE,
  exclude = FALSE,
  auth_token = NULL
)
```

Arguments

field_name	Character. Name of the custom field to filter by
field_values	Character vector. Values to filter for
global	Logical. Whether this is a global field (TRUE) or organization field (FALSE)
exclude	Logical. Whether to exclude apps matching criteria (FALSE = include)
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get filter ID for Word games
filter_id <- st_create_simple_filter(
  field_name = "Game Sub-genre",
  field_values = "Word"
)

# Get filter ID for multiple genres
filter_id <- st_create_simple_filter(
  field_name = "Game Genre",
  field_values = c("Puzzle", "Word")
)

## End(Not run)
```

st_custom_fields *Custom Fields Filter Functions*

Description

Functions to work with Sensor Tower custom fields filters.

st_custom_fields_filter *Create a Custom Fields Filter*

Description

Creates a custom fields filter ID by posting filter criteria to Sensor Tower. This filter ID can then be used with other endpoints to query filtered data.

Usage

```
st_custom_fields_filter(
  custom_fields,
  auth_token = NULL,
  base_url = "https://api.sensortower.com"
)
```

Arguments

custom_fields	List. A list of custom field criteria with the following structure: - exclude: Logical. Whether to exclude apps matching this criteria - global: Logical. Whether this is a global field (TRUE) or organization field (FALSE) - name: Character. The name of the custom field (e.g., "Free", "Release Date (US)") - values: Character vector. Values to filter by (can be empty) - true: Logical. For boolean fields, the value to match
auth_token	Optional. Character string. Your Sensor Tower API token.
base_url	Optional. Character string. The base URL for the API.

Value

Character string containing the custom fields filter ID

Examples

```
## Not run:
# Create a filter for free apps
filter_id <- st_custom_fields_filter(
  custom_fields = list(
    list(
      exclude = FALSE,
      global = TRUE,
      name = "Free",
      values = list(),
      true = TRUE
    )
  )
)

## End(Not run)
```

`st_custom_fields_filter_by_id` *Get Custom Fields Filter Details by ID*

Description

Retrieves the custom field names and tag values associated with a Custom Fields Filter ID.

Usage

```
st_custom_fields_filter_by_id(
  id,
  auth_token = NULL,
  base_url = "https://api.sensortower.com"
)
```

Arguments

`id`	Character string. The custom fields filter ID to query.
`auth_token`	Optional. Character string. Your Sensor Tower API token.
`base_url`	Optional. Character string. The base URL for the API.

Value

A list containing the custom fields filter details

Examples

```
## Not run:
# Get details for a specific filter ID
filter_details <- st_custom_fields_filter_by_id(
  id = "6009d417241bc16eb8e07e9b"
)

## End(Not run)
```

`st_custom_fields_utils`

Custom Fields Utility Functions

Description

Utility functions for common custom fields filtering scenarios in Sensor Tower. These functions provide pre-built filters for frequently used queries.

`st_custom_fields_values`

Get Custom Fields Values

Description

Retrieves a list of all accessible custom fields and their possible values. This is useful for discovering what custom fields are available to filter by.

Usage

```
st_custom_fields_values(
  term = NULL,
  auth_token = NULL,
  base_url = "https://api.sensortower.com"
)
```

Arguments

`term`	Optional. Character string. Search term to filter field names.
`auth_token`	Optional. Character string. Your Sensor Tower API token.
`base_url`	Optional. Character string. The base URL for the API.

Value

A tibble containing custom fields and their possible values

Examples

```
## Not run:
# Get all custom fields
fields <- st_custom_fields_values()

# Search for specific fields
date_fields <- st_custom_fields_values(term = "date")

## End(Not run)
```

`st_custom_fields_workflow` *Custom Fields Filter Workflow Helper Functions*

Description

Helper functions to streamline working with custom fields filters in Sensor Tower. These functions combine the custom fields endpoints with data retrieval functions to provide a complete workflow.

`st_demographics` *Fetch Demographics Data for Apps*

Description

Retrieves user demographics (age and gender breakdown) for specific apps from the Sensor Tower Usage Intelligence API. This function queries the demographics endpoint directly using platform-specific app IDs.

Usage

```
st_demographics(
  unified_app_id = NULL,
  ios_app_id = NULL,
  android_app_id = NULL,
  os = NULL,
  country = "US",
  date_granularity = "all_time",
  start_date = NULL,
  end_date = NULL,
  auth_token = NULL,
  verbose = TRUE
)
```

Arguments

unified_app_id	Character string. Sensor Tower unified app ID (24-character hex). Will be resolved to platform-specific IDs automatically.
ios_app_id	Character string. iOS app ID (numeric, e.g., "1234567890").
android_app_id	Character string. Android package name (e.g., "com.example.app").
os	Character string. Operating system: "ios" or "android". Required if using platform-specific IDs. When using unified_app_id, defaults to trying both platforms.
country	Character string. Country code (e.g., "US", "GB"). Default is "US". Only single country supported per request.
date_granularity	Character string. Either "all_time" (default) or "quarterly". All-time data goes back to Q4 2015. Quarterly data begins Q1 2021.
start_date	Date or character string. Start date for quarterly data in "YYYY-MM-DD" format. Ignored for all_time granularity.
end_date	Date or character string. End date for quarterly data in "YYYY-MM-DD" format. Ignored for all_time granularity.
auth_token	Optional. Character string. Your Sensor Tower API token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.
verbose	Logical. If TRUE, prints progress messages.

Value

A [tibble][tibble::tibble] with demographics metrics including: - **app_id**: The platform-specific app ID - **os**: Platform (ios or android) - **country**: Country code - **female_percent**: Percentage of female users (0-100) - **male_percent**: Percentage of male users (0-100) - **average_age**: Average user age - **age_13_17**, **age_18_24**, **age_25_34**, **age_35_44**, **age_45_54**, **age_55_64**, **age_65_plus**: Age group percentages - **confidence**: Data confidence level

Data Availability

- Quarterly data begins Q1 2021 - All-time data goes back to Q4 2015 - Demographics are primarily available for US market - Data availability depends on app's user base size

Recommended Workflow

‘‘‘ # Step 1: Search for app by name app <- st_app_info("Royal Match")

Step 2: Get demographics using unified ID demographics <- st_demographics(unified_app_id = app$unified_app_id[1]) ‘‘‘

See Also

[st_app_info()] for searching apps by name, [st_app_lookup()] for resolving app IDs, [st_retention()] for retention metrics

Examples

```
## Not run:
# Get demographics for an app using unified ID
demo <- st_demographics(unified_app_id = "5f16a8019f7b275235017614")

# Get demographics for iOS app directly
demo <- st_demographics(
  ios_app_id = "553834731",
  os = "ios",
  country = "US"
)

## End(Not run)
```

`st_discover_fields` *Discover Available Custom Fields*

Description

Searches and displays available custom fields that can be used for filtering. This is helpful for discovering what fields are available before creating filters.

Usage

```
st_discover_fields(search_term = NULL, show_values = FALSE, auth_token = NULL)
```

Arguments

`search_term`	Optional. Character string to search for in field names
`show_values`	Logical. Whether to show possible values for each field
`auth_token`	Optional. Character string. Your Sensor Tower API token.

Value

A tibble with custom fields information

Examples

```
## Not run:
# Find all game-related fields
game_fields <- st_discover_fields("game")

# Find all date fields
date_fields <- st_discover_fields("date")

# Show all fields with their values
all_fields <- st_discover_fields(show_values = TRUE)
```

```
## End(Not run)
```

`st_extract_filter_id` *Extract Filter ID from Sensor Tower URL*

Description

Extracts the custom_fields_filter_id parameter from a Sensor Tower URL. This is helpful when copying URLs from the web interface.

Usage

```
st_extract_filter_id(url)
```

Arguments

`url`	Character string. A Sensor Tower URL containing custom_fields_filter_id

Value

Character string. The extracted filter ID, or NULL if not found.

Examples

```
## Not run:
url <- "https://app.sensortower.com/top-charts?custom_fields_filter_id=687df26ac5a19ebcfe817d7f"
filter_id <- st_extract_filter_id(url)

## End(Not run)
```

`st_extract_url_params` *Extract All Parameters from Sensor Tower URL*

Description

Extracts and displays all parameters from a Sensor Tower web URL in a readable format. Useful for understanding complex URLs.

Usage

```
st_extract_url_params(url)
```

Arguments

`url`	Character string. A Sensor Tower web interface URL

Value

Data frame with parameter names and values

Examples

```
## Not run:
url <- "https://app.sensortower.com/market-analysis/top-apps?os=unified&measure=DAU"
params_df <- st_extract_url_params(url)
View(params_df)

## End(Not run)
```

---

st_filter_by_date *Create Date-Based Filter*

Description

Creates a filter for apps based on release date criteria.

Usage

```
st_filter_by_date(
  released_after = NULL,
  released_before = NULL,
  region = "US",
  auth_token = NULL
)
```

Arguments

`released_after`	Date or character. Apps released after this date
`released_before`	Date or character. Apps released before this date
`region`	Character. Region for release date ("US", "WW", "JP", "CN")
`auth_token`	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get apps released in 2024
filter_id <- st_filter_by_date(
  released_after = "2024-01-01",
  released_before = "2024-12-31",
  region = "US"
)

## End(Not run)
```

`st_filter_by_genre` *Create Genre-Based Filter*

Description

Creates a filter for apps in specific game genres or sub-genres.

Usage

```
st_filter_by_genre(
  genres = NULL,
  sub_genres = NULL,
  exclude_genres = FALSE,
  auth_token = NULL
)
```

Arguments

`genres`	Character vector. Game genres to filter (e.g., "Puzzle", "Action")
`sub_genres`	Character vector. Game sub-genres to filter (e.g., "Word", "Match-3")
`exclude_genres`	Logical. Whether to exclude these genres (FALSE = include)
`auth_token`	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get Puzzle and Word games
filter_id <- st_filter_by_genre(
  genres = "Puzzle",
  sub_genres = "Word"
)
```

```
# Exclude Action and Shooter games
filter_id <- st_filter_by_genre(
  genres = c("Action", "Shooter"),
  exclude_genres = TRUE
)

## End(Not run)
```

---

st_filter_by_monetization *Create Monetization-Based Filter*

Description

Creates a filter for apps based on their monetization model.

Usage

```
st_filter_by_monetization(
  free_only = NULL,
  has_iap = NULL,
  has_ads = NULL,
  has_subscription = NULL,
  auth_token = NULL
)
```

Arguments

free_only	Logical. Only free apps
has_iap	Logical. Has in-app purchases
has_ads	Logical. Contains ads
has_subscription	Logical. Has subscription model
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get free apps with IAP but no ads
filter_id <- st_filter_by_monetization(
  free_only = TRUE,
  has_iap = TRUE,
  has_ads = FALSE
```

```
)

## End(Not run)
```

---

st_filter_by_publisher *Create Publisher-Based Filter*

Description

Creates a filter for apps from specific publishers.

Usage

```
st_filter_by_publisher(publisher_names, exclude = FALSE, auth_token = NULL)
```

Arguments

publisher_names	Character vector. Publisher names to filter
exclude	Logical. Whether to exclude these publishers (FALSE = include)
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get apps from specific publishers
filter_id <- st_filter_by_publisher(
  publisher_names = c("Electronic Arts", "Activision")
)

## End(Not run)
```

st_filter_by_sdk *Create SDK-Based Filter*

Description

Creates a filter for apps using specific SDKs or technologies.

Usage

```
st_filter_by_sdk(sdk_names, exclude = FALSE, auth_token = NULL)
```

Arguments

sdk_names	Character vector. SDK names to filter (e.g., "Unity", "Firebase")
exclude	Logical. Whether to exclude apps with these SDKs
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

Character string containing the filter ID

Examples

```
## Not run:
# Get Unity-based games
filter_id <- st_filter_by_sdk(sdk_names = "Unity")

# Get apps using both Firebase and AdMob
filter_id <- st_filter_by_sdk(
  sdk_names = c("Firebase", "AdMob")
)

## End(Not run)
```

st_game_summary *Fetch Game Market Summary Data*

Description

Retrieves aggregated download and revenue estimates by game categories, countries, and date ranges. This provides a market overview of game performance across different segments.

Usage

```
st_game_summary(
  categories = 7001,
  countries,
  os,
  date_granularity,
  start_date,
  end_date,
  auth_token = NULL,
  base_url = "https://api.sensortower.com",
  enrich_response = TRUE
)
```

Arguments

categories	Character string or numeric vector. Game category IDs to analyze. Defaults to 7001 (a popular game category). Use ‘st_categories()‘ to find valid category IDs.
countries	Character vector or comma-separated string. Country codes (e.g., ‘"US"‘, ‘c("US", "GB")‘, ‘"WW"‘ for worldwide) to analyze. Required.
os	Character string. Operating System. Must be one of "ios", "android", or "unified". Required.
date_granularity	Character string. Time granularity for aggregation. Must be one of "daily", "weekly", "monthly", or "quarterly". Required.
start_date	Character string or Date object. Start date for the query in "YYYY-MM-DD" format. Required.
end_date	Character string or Date object. End date for the query in "YYYY-MM-DD" format, inclusive. Required.
auth_token	Optional. Character string. Your Sensor Tower API token.
base_url	Optional. Character string. The base URL for the API.
enrich_response	Optional. Logical. If ‘TRUE‘ (default), enriches the response with readable column names and processes the data.

Value

A [tibble][tibble::tibble] with game market summary data including: - **Category information**: Game category details - **Geographic data**: Country-level breakdowns - **Downloads**: Unified iOS (iPhone + iPad combined) and Android download estimates - **Revenue**: Unified iOS (iPhone + iPad combined) and Android revenue estimates - **Time series**: Data broken down by specified granularity

Automatic Data Combination: For iOS and unified platforms, iPhone and iPad data are automatically combined into single "iOS Downloads" and "iOS Revenue" columns for simplified analysis.

API Endpoint Used

- **Game Summary**: ‘GET /v1/{os}/games_breakdown‘

Field Mappings and Processing

The API returns abbreviated field names which are automatically mapped to descriptive names and processed: - **iOS**: ‘iu‘ + ‘au‘ = iOS Downloads (combined), ‘ir‘ + ‘ar‘ = iOS Revenue (combined) - **Android**: ‘u‘ = Android Downloads, ‘r‘ = Android Revenue - **Common**: ‘cc‘ = Country Code, ‘d‘ = Date, ‘aid‘ = App ID

iPhone and iPad data are automatically combined for simplified analysis.

See Also

[st_categories()], [st_top_charts()], [st_metrics()]

Examples

```
## Not run:
# Basic game market summary (last 30 days, worldwide)
game_summary <- st_game_summary()

# Specific categories and countries
rpg_summary <- st_game_summary(
  categories = c(7001, 7002),
  countries = c("US", "GB", "DE"),
  date_granularity = "weekly"
)

# Monthly summary for iOS games in the US
ios_monthly <- st_game_summary(
  os = "ios",
  countries = "US",
  date_granularity = "monthly",
  start_date = "2024-01-01",
  end_date = "2024-06-30"
)

## End(Not run)
```

---

st_generate_example_filter_ids

Generate Example Filter IDs for Testing

Description

Generates example filter IDs in the correct format for testing purposes. Note: These are randomly generated and won't work with the actual API unless they happen to match a real filter in your account.

Usage

```
st_generate_example_filter_ids(n = 5, seed = NULL)
```

Arguments

n	Integer. Number of example IDs to generate. Defaults to 5.
seed	Integer. Random seed for reproducibility. Optional.

Value

Character vector of example filter IDs

Examples

```
# Generate example IDs
st_generate_example_filter_ids(3)

# Generate with seed for reproducibility
st_generate_example_filter_ids(3, seed = 123)
```

---

st_get_app_names *Get App Names from Publisher Apps Result*

Description

Helper function to create a name lookup table from the result of ‘st_publisher_apps()‘. This handles canonical ID mapping automatically, so you can join sales data (which uses canonical IDs) back to app names.

Usage

```
st_get_app_names(apps_df, include_canonical = TRUE)
```

Arguments

apps_df	A tibble returned by ‘st_publisher_apps()‘.
include_canonical	Logical. If TRUE, includes mappings for canonical IDs that were resolved during aggregation. Defaults to TRUE.

Value

A tibble with columns ‘unified_app_id‘ and ‘app_name‘ suitable for joining with sales data or other API results.

Examples

```
## Not run:
# Get apps with canonical ID resolution
apps <- st_publisher_apps("647eb849d9d91f31a54f1792", aggregate_related = TRUE)

# Get name lookup table
name_lookup <- st_get_app_names(apps)

# Use with sales data
sales <- st_unified_sales_report(apps$unified_app_id, ...)
sales_with_names <- sales %>%
  left_join(name_lookup, by = "unified_app_id")

## End(Not run)
```

---

st_get_filtered_apps *Get Top Apps with Custom Filter*

Description

Retrieves top apps using a custom fields filter. This combines filter creation with data retrieval in a single workflow.

Usage

```
st_get_filtered_apps(
  field_name = NULL,
  field_values = NULL,
  filter_id = NULL,
  measure = "DAU",
  regions = "US",
  date = NULL,
  end_date = NULL,
  limit = 100,
  enrich_response = TRUE,
  auth_token = NULL,
  ...
)
```

Arguments

field_name	Character. Name of the custom field to filter by (or NULL to use filter_id)
field_values	Character vector. Values to filter for (or NULL to use filter_id)
filter_id	Character. Existing filter ID to use (alternative to field_name/values)
measure	Character. Metric to measure: "DAU", "WAU", "MAU", "revenue", or "units"
regions	Character vector. Region codes (e.g., "US", "WW")

date	Character or Date. Start date for the query
end_date	Optional. Character or Date. End date for the query
limit	Integer. Maximum number of apps to return (default 100)
enrich_response	Logical. Whether to enrich with additional metrics
auth_token	Optional. Character string. Your Sensor Tower API token.
...	Additional parameters passed to st_top_charts

Value

A tibble with top apps data

Examples

```
## Not run:
# Get top Word games by DAU
word_games <- st_get_filtered_apps(
  field_name = "Game Sub-genre",
  field_values = "Word",
  measure = "DAU",
  regions = "US",
  limit = 20
)

# Use existing filter ID
apps <- st_get_filtered_apps(
  filter_id = "603697f4241bc16eb8570d37",
  measure = "revenue",
  regions = "US"
)

## End(Not run)
```

---

st_get_filter_collection

Get Pre-Built Filter Collections

Description

Returns commonly used filter IDs for quick access to pre-defined app segments.

Usage

```
st_get_filter_collection(
  collection = c("top_genres", "monetization_models", "platform_exclusive",
    "market_segments"),
  auth_token = NULL
)
```

Arguments

collection	Character. Name of the collection: - "top_genres": Major game genres - "monetization_models": Different monetization approaches - "platform_exclusive": Platform-specific apps - "market_segments": Market segment filters
auth_token	Optional. Character string. Your Sensor Tower API token.

Value

A named list of filter IDs

Examples

```
## Not run:
# Get filter IDs for top game genres
genre_filters <- st_get_filter_collection("top_genres")

# Use a filter from the collection
puzzle_apps <- st_get_filtered_apps(
  filter_id = genre_filters$puzzle,
  measure = "DAU",
  regions = "US"
)

## End(Not run)
```

---

st_get_unified_mapping

Get Unified ID Mapping for Apps

Description

Retrieves the mapping between platform-specific app IDs and unified app IDs. This function handles cases where platform IDs from st_top_charts may not be directly searchable, using app names as a fallback resolution method.

Usage

```
st_get_unified_mapping(
  app_ids,
  app_names = NULL,
  os = "unified",
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN")
)
```

Arguments

app_ids	Character vector of app IDs (can be iOS, Android, or unified hex IDs)
app_names	Character vector of app names (optional, helps with resolution)
os	Character string. Operating system: "ios", "android", or "unified"
auth_token	Character string. Sensor Tower API authentication token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

Details

This function uses an ID-first approach (no name-based resolution): 1. For hex IDs (24-char), uses st_app_lookup to get platform IDs 2. For platform IDs, first tries to look them up via st_app_lookup 3. If direct lookup fails, searches the unified index using the platform ID as the term and matches exact IDs within nested ios_apps/android_apps 4. Returns the best available mapping for each app using IDs only

Note: Platform IDs from st_top_charts may be regional or legacy IDs that aren't directly searchable. In these cases, name-based search provides the most reliable resolution to unified IDs.

Value

A data frame with columns: - 'input_id': The original ID provided - 'unified_app_id': The unified app ID (hex format) - 'unified_app_name': The unified app name - 'ios_app_id': iOS app ID (if available) - 'android_app_id': Android app ID (if available) - 'publisher_id': Publisher ID - 'publisher_name': Publisher name

Examples

```
## Not run:
# Get mapping with app names for better resolution
mapping <- st_get_unified_mapping(
  app_ids = c("943599237", "com.bandainamcogames.dbzdokkan"),
  app_names = c("Dragon Ball Z Dokkan Battle", "Dragon Ball Z Dokkan Battle"),
  os = "unified"
)

## End(Not run)
```

---

st_gt_dashboard *Create FiveThirtyEight-styled GT Dashboard from Top Charts Data*

Description

Creates a professional, FiveThirtyEight-themed GT table dashboard from Sensor Tower top charts data with customizable styling and metric options.

Usage

```
st_gt_dashboard(
  data,
  title = "Top Mobile Games",
  subtitle = NULL,
  ranking_metric = "revenue_180d_ww",
  show_demographics = TRUE,
  show_engagement = TRUE,
  show_retention = TRUE,
  retention_region = "us",
  show_rpd = TRUE,
  bar_charts = TRUE,
  bar_chart_columns = NULL,
  heatmap_retention = TRUE,
  compact_mode = TRUE,
  width = 1800,
  height = 700,
  save_path = NULL,
  icon_cache_dir = "inst/images/app_icons",
  raw = FALSE,
 color_scheme = list(revenue = "#FF6600", downloads = "#008FD5", engagement = "#9C27B0",
  rpd = "#4CAF50", retention_low = "#FFCDD2", retention_mid = "#C8E6C9", retention_high
   = "#4CAF50")
)
```

Arguments

data	Data frame from st_top_charts() or similar Sensor Tower function
title	Character string for the table title (default: "Top Mobile Games")
subtitle	Character string for subtitle. If NULL, auto-generates based on data
ranking_metric	Character string specifying which metric to use for ranking. Options: "revenue_180d_ww", "revenue_30d_ww", "downloads_180d_ww", "downloads_30d_ww", etc. (default: "revenue_180d_ww")
show_demographics	Logical, whether to show demographic columns (age, gender) (default: TRUE)
show_engagement	Logical, whether to show engagement metrics (DAU, WAU, MAU) (default: TRUE)
show_retention	Logical, whether to show retention metrics (default: TRUE)
retention_region	Character string for retention region ("us", "ww", etc.) (default: "us")
show_rpd	Logical, whether to show Revenue Per Download (default: TRUE)
bar_charts	Logical, whether to show bar chart visualizations (default: TRUE)
bar_chart_columns	Character vector of column patterns to add bar charts to. If NULL, applies to all numeric columns except RPD and retention.

heatmap_retention	Logical, whether to apply heatmap to retention columns (default: TRUE)
compact_mode	Logical, whether to use compact row heights (default: TRUE)
width	Numeric, table width in pixels (default: 1800)
height	Numeric, table height in pixels (default: 700)
save_path	Character string, path to save the table image. If NULL, returns GT object
icon_cache_dir	Character string, directory to cache app icons (default: "inst/images/app_icons")
raw	Logical, whether to return a minimally styled table without custom formatting, bar charts, or heatmaps (default: FALSE)
color_scheme	List with color codes for different metric types: - revenue: Revenue metrics color (default: "#FF6600") - downloads: Downloads metrics color (default: "#008FD5") - engagement: Engagement metrics color (default: "#9C27B0") - rpd: RPD metrics color (default: "#4CAF50") - retention_low: Low retention color (default: "#FFCDD2") - retention_mid: Mid retention color (default: "#C8E6C9") - retention_high: High retention color (default: "#4CAF50")

Value

GT object (if save_path is NULL) or saves image and returns path

Examples

```
## Not run:
# Basic usage - one line after st_top_charts()
top_rpgs <- st_top_charts(category = 7014, measure = "revenue")
st_gt_dashboard(top_rpgs)

# Raw mode for minimal styling
st_gt_dashboard(top_rpgs, raw = TRUE)

# Customize the dashboard
st_gt_dashboard(
  top_rpgs,
  title = "Top Role-Playing Games Q4 2024",
  ranking_metric = "revenue_30d_ww",
  show_retention = FALSE,
  save_path = "dashboard.png"
)

# Change color scheme
st_gt_dashboard(
  top_rpgs,
  color_scheme = list(
    revenue = "#E74C3C",
    downloads = "#3498DB",
    engagement = "#9B59B6"
  )
)

## End(Not run)
```

st_is_valid_filter_id *Validate Custom Field Filter ID Format*

Description

Checks if a filter ID matches the expected 24-character hexadecimal format used by Sensor Tower.

Usage

```
st_is_valid_filter_id(filter_id)
```

Arguments

filter_id	Character string. The filter ID to validate

Value

Logical. TRUE if valid format, FALSE otherwise

Examples

```
## Not run:
# Valid filter ID
st_is_valid_filter_id("687df26ac5a19ebcfe817d7f")  # TRUE

# Invalid filter IDs
st_is_valid_filter_id("invalid")  # FALSE
st_is_valid_filter_id("687df26ac5a19ebcfe817d7")  # FALSE (too short)

## End(Not run)
```

st_metrics *Fetch Sensor Tower Metrics for Apps*

Description

Retrieves metrics for apps. The OS parameter controls which platform's data is returned, regardless of which app IDs are provided. The function will automatically look up the appropriate IDs if needed.

Usage

```
st_metrics(
  os,
  app_id = NULL,
  ios_app_id = NULL,
  android_app_id = NULL,
  unified_app_id = NULL,
  start_date = NULL,
  end_date = NULL,
  countries,
  date_granularity,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE
)
```

Arguments

os	Character. Required. Operating system: "ios", "android", or "unified". This determines which platform's data is returned.
app_id	Character string. Can be a unified app ID, iOS app ID, or Android package name.
ios_app_id	Character string. iOS app ID (optional).
android_app_id	Character string. Android package name (optional).
unified_app_id	Character string. Sensor Tower unified ID (24-char hex).
start_date	Date object or character string (YYYY-MM-DD). Start date.
end_date	Date object or character string (YYYY-MM-DD). End date.
countries	Character vector. Country codes (e.g., "US", "GB", "JP", or "WW" for worldwide). Required.
date_granularity	Character. One of "daily", "weekly", "monthly", "quarterly". Required.
auth_token	Character string. Sensor Tower API token.
verbose	Logical. Print progress messages.

Details

The OS parameter controls what data is returned:

- os = "ios": Returns iOS data only - os = "android": Returns Android data only - os = "unified": Returns combined iOS + Android data (as separate rows)

The function will automatically look up the appropriate IDs based on the OS parameter. For example, if you provide a unified_app_id but set os = "ios", it will look up the iOS app ID and return iOS-only data.

Value

A tibble with columns: app_id, app_id_type, date, country, revenue, downloads

Examples

```
## Not run:
# Get iOS data only
ios_metrics <- st_metrics(
  os = "ios",
  ios_app_id = "1195621598", # Homescapes iOS
  countries = "US",
  date_granularity = "daily",
  start_date = Sys.Date() - 30,
  end_date = Sys.Date() - 1
)

# Get unified data from a unified ID
unified_metrics <- st_metrics(
  os = "unified",
  unified_app_id = "5ba4585f539ce75b97db6bcb",
  countries = "US",
  date_granularity = "daily"
)

# Get iOS data from Android ID (automatic lookup)
ios_from_android <- st_metrics(
  os = "ios",
  android_app_id = "com.king.candycrushsaga",
  countries = "WW",
  date_granularity = "monthly"
)

# Get unified data from platform IDs
unified_from_platforms <- st_metrics(
  os = "unified",
  ios_app_id = "1195621598",
  android_app_id = "com.playrix.homescapes",
  countries = "US",
  date_granularity = "daily"
)

## End(Not run)
```

`st_parse_web_url` *Parse Sensor Tower Web URL to API Parameters*

Description

Converts a Sensor Tower web interface URL into API-compatible parameters that can be used with sensortowerR functions. This is helpful when you want to replicate a web query in R.

Usage

```
st_parse_web_url(url, verbose = TRUE)
```

Arguments

url	Character string. A Sensor Tower web interface URL
verbose	Logical. Whether to print parameter mapping details. Defaults to TRUE.

Value

List of API-compatible parameters suitable for use with st_top_charts() and other sensortowerR functions

Examples

```
## Not run:
# Parse a web URL
url <- "https://app.sensortower.com/market-analysis/top-apps?os=unified&measure=DAU"
params <- st_parse_web_url(url)

# Use the parameters in an API call
data <- do.call(st_top_charts, params)

# Or modify parameters before using
params$limit <- 50
data <- do.call(st_top_charts, params)

## End(Not run)
```

---

st_publisher_apps *Get All Apps from a Publisher*

Description

Retrieves a list of apps associated with a specified unified publisher ID from the Sensor Tower API. Targets the '/v1/unified/publishers/apps' endpoint.

Usage

```
st_publisher_apps(
  unified_id = NULL,
  publisher_id = NULL,
  aggregate_related = FALSE,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE
)
```

Arguments

unified_id	Character. Unified ID to resolve apps for. May be either: - Unified Publisher ID (24-char hex) - Unified App ID (24-char hex) belonging to a publisher The API returns the unified publisher and all associated apps in both cases.
publisher_id	Deprecated alias for 'unified_id'.
aggregate_related	Logical. If TRUE, ensures each app's unified_app_id is the canonical ID that aggregates ALL regional SKUs. This solves the problem where games like "Watcher of Realms" are published under multiple regional publishers (Moonton, Vizta Games, Skystone Games, etc.) and may return different unified_app_ids. When TRUE, the function looks up each app by name to find the true unified_app_id that combines all regional versions. Defaults to FALSE for backwards compatibility.
auth_token	Character. Your Sensor Tower API authentication token. Defaults to the value stored in the 'SENSORTOWER_AUTH_TOKEN' environment variable.
verbose	Logical. If TRUE, prints progress messages during aggregation. Defaults to TRUE.

Value

A [tibble][tibble::tibble] containing details of the apps associated with the publisher. The exact columns depend on the API response but often include app IDs, names, platform, etc. Returns an empty tibble if the publisher ID is invalid, has no apps, or an error occurs.

Solving Regional Publisher Issues

Many publishers have regional subsidiaries or partners that publish the same game under different app IDs in different regions. For example, Moonton's "Watcher of Realms" is published by Moonton in some regions, Vizta Games in others, and Skystone Games in others.

When 'aggregate_related = TRUE', this function ensures you get the unified_app_id that represents the FULL game across all regional publishers, which is required for accurate revenue/download aggregation via 'st_unified_sales_report()'.

API Endpoint Used

- 'GET /v1/unified/publishers/apps'

Examples

```
## Not run:
# Ensure SENSORTOWER_AUTH_TOKEN is set in your environment
# Sys.setenv(SENSORTOWER_AUTH_TOKEN = "your_secure_auth_token_here")

# Basic usage - get publisher's apps
apps_list <- st_publisher_apps(unified_id = "647eb849d9d91f31a54f1792")

# With regional SKU aggregation - ensures canonical unified_app_ids
apps_list <- st_publisher_apps(
```

```
  unified_id = "647eb849d9d91f31a54f1792",
  aggregate_related = TRUE
)

# Then use with st_unified_sales_report() for accurate data
sales <- st_unified_sales_report(
  unified_app_id = apps_list$unified_app_id,
  countries = "WW",
  start_date = "2024-01-01",
  end_date = "2024-12-31",
  date_granularity = "monthly"
)

## End(Not run)
```

`st_publisher_portfolio` *Publisher Portfolio Analysis*

Description

Fetches comprehensive portfolio data for a publisher including revenue, downloads, MAU, and rankings. Returns a tidy data frame ready for visualization or GT table creation.

Usage

```
st_publisher_portfolio(
  publisher = NULL,
  publisher_id = NULL,
  start_date = "2023-01-01",
  end_date = NULL,
  countries = "WW",
  metrics = c("revenue", "downloads", "mau"),
  include_rankings = TRUE,
  include_portfolio_total = TRUE,
  granularity = "yearly",
  min_revenue = 1e+05,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE,
  use_cache = FALSE,
  cache_dir = NULL
)
```

Arguments

`publisher`	Character. Publisher name to search for (e.g., "Lilith Games", "Supercell", "King"). The function will search for the publisher and use the first match.

publisher_id	Character. Optional. If provided, skips the publisher search and uses this unified_publisher_id directly.
start_date	Date or character. Start date for metrics (default: "2023-01-01").
end_date	Date or character. End date for metrics (default: last day of previous month).
countries	Character. Countries for metrics (default: "WW" for worldwide).
metrics	Character vector. Which metrics to fetch. Options: "revenue", "downloads", "mau". Default: all three.
include_rankings	Logical. Whether to fetch subgenre rankings from top charts. Default: TRUE.
include_portfolio_total	Logical. Whether to add a portfolio total row. Default: TRUE.
granularity	Character. How to aggregate the data: "yearly" (default), "quarterly", or "monthly".
min_revenue	Numeric. Minimum revenue threshold to include an app. Default: 100000 (apps with at least $100K in any year).
auth_token	Character. Sensor Tower API token. Defaults to SENSORTOWER_AUTH_TOKEN environment variable
verbose	Logical. Print progress messages. Default: TRUE.
use_cache	Logical. Use cached data if available. Default: FALSE. When TRUE, requires cache_dir to be specified.
cache_dir	Character. Directory for cached data. Default: NULL (no caching). Must be explicitly set to enable caching. Use tempdir() for temporary caching.

Value

A tibble with portfolio data including: - app_name: Game name - subgenre: Game sub-genre - subgenre_rank: Rank within sub-genre - revenue_{year}: Revenue by year - downloads_{year}: Downloads by year - mau_{year}: Average MAU by year (if requested) - revenue_yoy, downloads_yoy, mau_yoy: Year-over-year growth percentages

Examples

```
## Not run:
# Simple usage - just provide publisher name
lilith_portfolio <- st_publisher_portfolio("Lilith Games")

# Piped workflow
library(dplyr)

"Supercell" %>%
  st_publisher_portfolio(
    start_date = "2023-01-01",
    metrics = c("revenue", "downloads")
  ) %>%
  filter(revenue_2024 > 1000000) %>%
  arrange(desc(revenue_2024))

# Custom date range and countries
```

```
portfolio <- st_publisher_portfolio(
  publisher = "King",
  start_date = "2022-01-01",
  end_date = "2024-12-31",
  countries = c("US", "GB", "DE"),
  metrics = c("revenue", "downloads", "mau"),
  include_rankings = TRUE
)

## End(Not run)
```

---

st_retention *Fetch Retention Data for Apps*

Description

Retrieves retention metrics (D1-D90) for specific apps from the Sensor Tower Usage Intelligence API. This function queries the retention endpoint directly using platform-specific app IDs.

Usage

```
st_retention(
  unified_app_id = NULL,
  ios_app_id = NULL,
  android_app_id = NULL,
  os = NULL,
  country = "US",
  date_granularity = "all_time",
  start_date = NULL,
  end_date = NULL,
  auth_token = NULL,
  verbose = TRUE
)
```

Arguments

unified_app_id	Character string. Sensor Tower unified app ID (24-character hex). Will be resolved to platform-specific IDs automatically.
ios_app_id	Character string. iOS app ID (numeric, e.g., "1234567890").
android_app_id	Character string. Android package name (e.g., "com.example.app").
os	Character string. Operating system: "ios" or "android". Required if using platform-specific IDs. When using unified_app_id, defaults to "ios" but will try both platforms.
country	Character string. Country code (e.g., "US", "GB"). Default is "US". Only single country supported per request.

`date_granularity`
: Character string. Either "all_time" (default) or "quarterly". All-time data goes back to Q4 2015. Quarterly data begins Q1 2021.

`start_date`
: Date or character string. Start date for quarterly data in "YYYY-MM-DD" format. Ignored for all_time granularity.

`end_date`
: Date or character string. End date for quarterly data in "YYYY-MM-DD" format. Ignored for all_time granularity.

`auth_token`
: Optional. Character string. Your Sensor Tower API token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

`verbose`
: Logical. If TRUE, prints progress messages.

Value

A [tibble][tibble::tibble] with retention metrics including: - **app_id**: The platform-specific app ID - **os**: Platform (ios or android) - **country**: Country code - **retention_d1** through **retention_d90**: Retention percentages (0-1 scale) - **confidence**: Data confidence level (red=low, yellow=medium, green=high) - **baseline_downloads**: Total downloads in baseline period - **baseline_start_date**, **baseline_end_date**: Dates for baseline period

Data Availability

- Quarterly data begins Q1 2021 - All-time data goes back to Q4 2015 - Data is only available for apps with sufficient user base - Confidence levels: red (<=3), yellow (4-6), green (>=7)

Recommended Workflow

“‘ # Step 1: Search for app by name app <- st_app_info("Royal Match")

Step 2: Get retention data using unified ID retention <- st_retention(unified_app_id = app$unified_app_id[1]) “‘

See Also

[st_app_info()] for searching apps by name, [st_app_lookup()] for resolving app IDs, [st_demographics()] for user demographics data

Examples

```
## Not run:
# Get retention for an app using unified ID
retention <- st_retention(unified_app_id = "5f16a8019f7b275235017614")

# Get retention for iOS app directly
retention <- st_retention(
  ios_app_id = "553834731",
  os = "ios",
  country = "US"
)

# Get quarterly retention data
```

```
retention <- st_retention(
  unified_app_id = "5f16a8019f7b275235017614",
  date_granularity = "quarterly",
  start_date = "2024-01-01",
  end_date = "2024-09-30"
)

## End(Not run)
```

`st_sales_report` *Fetch Sales Report Estimates*

Description

Retrieves download and revenue estimates of apps by country and date. Note: All revenues are returned in cents and need to be divided by 100 for dollar amounts.

Usage

```
st_sales_report(
  os,
  countries,
  start_date,
  end_date,
  date_granularity,
  ios_app_id = NULL,
  android_app_id = NULL,
  unified_app_id = NULL,
  publisher_ids = NULL,
  custom_fields_filter_id = NULL,
  custom_tags_mode = NULL,
  limit = 100,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  auto_segment = TRUE,
  verbose = TRUE
)
```

Arguments

`os`	Character string. Required. Operating system: "ios", "android", or "unified".
`countries`	Character vector. Country codes (e.g., c("US", "GB", "JP"), or "WW" for worldwide). Required.
`start_date`	Date or character string. Start date in "YYYY-MM-DD" format. Required.
`end_date`	Date or character string. End date in "YYYY-MM-DD" format. Required.
`date_granularity`	Character string. One of "daily", "weekly", "monthly", "quarterly". Required.

ios_app_id	Character string. iOS app ID (numeric, e.g., "1234567890").
android_app_id	Character string. Android package name (e.g., "com.example.app").
unified_app_id	Character string. Sensor Tower unified app ID (24-character hex).
publisher_ids	Character vector. Publisher IDs to query. Some Android publisher IDs contain commas.
custom_fields_filter_id	Optional. Character string. ID of a Sensor Tower custom field filter to apply. Use filter IDs from the web interface at app.sensortower.com. When provided, this filter will be used instead of app_ids or publisher_ids.
custom_tags_mode	Optional. Character string. Required if ‘os‘ is ’unified’ and ‘custom_fields_filter_id‘ is provided. Specifies how the custom filter applies to unified apps. Options: "include", "exclude", "include_unified_apps". The "include_unified_apps" option includes all platform versions when any version matches the filter.
limit	Numeric. Number of results to return when using custom_fields_filter_id. Ignored when using specific app ID parameters or publisher_ids. Defaults to 100.
auth_token	Optional. Character string. Your Sensor Tower API token.
auto_segment	Logical. If TRUE, automatically segments date ranges to avoid timeouts.
verbose	Logical. If TRUE, prints progress messages.

Details

App ID Parameters: Provide one of the following: - ‘ios_app_id‘: Specifically for iOS app IDs (numeric) - ‘android_app_id‘: Specifically for Android package names - ‘unified_app_id‘: Specifically for Sensor Tower unified IDs

The function will automatically resolve IDs if needed. For example, if you provide a ‘unified_app_id‘ but set ‘os="ios"‘, it will look up the iOS app ID.

The API has timeout limitations based on date granularity: - daily: limit to 1 week segments - weekly: limit to 3 month segments - monthly: limit to 1 year segments - quarterly: limit to 2 year segments

When auto_segment = TRUE, the function automatically breaks up the date range into appropriate segments and combines the results.

Value

A tibble with download and revenue estimates.

Examples

```
## Not run:
# Get daily sales for a single app using specific parameter
sales <- st_sales_report(
  os = "ios",
  ios_app_id = "553834731",  # Candy Crush iOS
  countries = c("US", "GB"),
  start_date = "2024-01-01",
```

```
  end_date = "2024-01-07",
  date_granularity = "daily"
)

# Get Android data using specific parameter
android_sales <- st_sales_report(
  os = "android",
  android_app_id = "com.king.candycrushsaga",
  countries = "US",
  start_date = "2024-01-01",
  end_date = "2024-01-07",
  date_granularity = "daily"
)

# Get iOS data from unified ID (automatic lookup)
unified_sales <- st_sales_report(
  os = "ios",
  unified_app_id = "5ba4585f539ce75b97db6bcb",
  countries = "US",
  start_date = "2024-01-01",
  end_date = "2024-01-07",
  date_granularity = "daily"
)


## End(Not run)
```

`st_session_metrics` *Fetch Session Metrics Time Series Data*

Description

Retrieves session metrics time series data (session count, session duration, time spent) for apps from the Sensor Tower Usage Intelligence API.

Usage

```
st_session_metrics(
  unified_app_id = NULL,
  ios_app_id = NULL,
  android_app_id = NULL,
  start_date,
  end_date,
  metrics = c("session_count", "session_duration", "time_spent"),
  regions = "US",
  time_period = "week",
  date_granularity = "monthly",
  os = NULL,
```

```
  breakdown = "unified_app_id",
  auth_token = NULL,
  verbose = TRUE
)
```

Arguments

unified_app_id	Character string or vector. Sensor Tower unified app ID(s) (24-character hex). Maximum 100 apps per request.
ios_app_id	Character string or vector. iOS app ID(s) for non-unified queries.
android_app_id	Character string or vector. Android package name(s) for non-unified queries.
start_date	Date or character string. Start date in "YYYY-MM-DD" format. Data is available from 2021-01-01 onward.
end_date	Date or character string. End date in "YYYY-MM-DD" format.
metrics	Character vector. Metrics to retrieve. Options include: - "time_spent" (average seconds per user per day) - "total_time_spent" (total seconds across all users) - "session_duration" (average session length in seconds) - "session_count" (average sessions per user per day) - "total_session_count" (total sessions across all users) Default is c("session_count", "session_duration", "time_spent").
regions	Character vector. Region/country codes (e.g., "US", "GB"). Default is "US". Use NULL for all regions.
time_period	Character string. Session metrics time period. Options: "day", "week". Default is "week". Returns averaged session metrics for each period within a month.
date_granularity	Character string. Aggregate data by granularity. Options: "daily", "weekly", "monthly". Default is "monthly". Note: "daily" granularity may not be supported by the API for all apps; use "weekly" or "monthly" if you receive empty results with "daily".
os	Character string. Filter by platform for unified apps. Options: "ios", "android", or NULL for both. Default is NULL.
breakdown	Character string. Fields for data aggregation. Options: "unified_app_id", "app_id", "region". Default is "unified_app_id".
auth_token	Character string. Your Sensor Tower API token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.
verbose	Logical. If TRUE, prints progress messages.

Value

A [tibble][tibble::tibble] with session metrics including: - **unified_app_id** or **app_id**: The app identifier - **date**: Date of the data point - **time_spent**: Average seconds spent per user per day - **total_time_spent**: Total seconds across all users - **session_duration**: Average session length in seconds - **session_count**: Average sessions per user per day - **total_session_count**: Total session count across all users

Data Availability

- Data is available from 2021-01-01 onward - Session metrics require Usage Intelligence subscription - Maximum 100 apps per request

See Also

[st_retention()] for retention metrics, [st_demographics()] for user demographics, [st_batch_metrics()] for MAU/DAU/WAU metrics

Examples

```
## Not run:
# Get session metrics for a unified app
sessions <- st_session_metrics(
  unified_app_id = "5fbc3849d0b8414136857afc",
  start_date = "2024-01-01",
  end_date = "2024-12-01"
)

# Get specific metrics with weekly granularity
sessions <- st_session_metrics(
  unified_app_id = "5fbc3849d0b8414136857afc",
  start_date = "2024-01-01",
  end_date = "2024-03-01",
  metrics = c("session_count", "session_duration"),
  date_granularity = "weekly"
)

# Get session data for Android app directly
sessions <- st_session_metrics(
  android_app_id = "com.example.app",
  start_date = "2024-01-01",
  end_date = "2024-06-01"
)

## End(Not run)
```

---

st_smart_metrics *Smart Metrics Fetching with Automatic ID Resolution*

Description

Enhanced metrics fetching that automatically handles ID resolution, caching, and fallbacks to minimize API calls.

Usage

```
st_smart_metrics(
  app_ids,
  metrics = c("revenue", "downloads"),
  start_date = Sys.Date() - 30,
  end_date = Sys.Date() - 1,
  countries = "WW",
  granularity = "daily",
  auto_resolve = TRUE,
  use_cache = TRUE,
  parallel = TRUE,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE
)
```

Arguments

app_ids	Character vector. Can be any mix of iOS IDs, Android IDs, or Sensor Tower unified IDs.
metrics	Character vector. Metrics to fetch (e.g., "revenue", "downloads", "dau").
start_date	Date or character string. Start date for metrics.
end_date	Date or character string. End date for metrics.
countries	Character vector. Country codes (default "WW").
granularity	Character. Date granularity ("daily", "weekly", "monthly").
auto_resolve	Logical. Automatically resolve IDs using cache/API (default TRUE).
use_cache	Logical. Use ID cache to minimize lookups (default TRUE).
parallel	Logical. Use parallel processing (default TRUE).
auth_token	Character string. Your Sensor Tower API authentication token.
verbose	Logical. Print progress messages.

Value

A tibble with metrics in long format

Examples

```
## Not run:
# Mixed ID types - automatically resolved
metrics <- st_smart_metrics(
  app_ids = c(
    "553834731", # Candy Crush iOS
    "com.king.candycrushsaga", # Candy Crush Android
    "5ba4585f539ce75b97db6bcb" # Star Trek unified ID
  ),
  metrics = c("revenue", "downloads", "dau"),
  start_date = "2024-01-01",
```

```
  end_date = "2024-12-31"
)

## End(Not run)
```

`st_test_filter` *Test a Custom Filter ID*

Description

Tests whether a custom filter ID works with the Sensor Tower API by making a minimal test request. This helps verify that the filter exists and is accessible with your authentication.

Usage

```
st_test_filter(filter_id, os = "ios", verbose = TRUE, auth_token = NULL)
```

Arguments

`filter_id`	Character string. The filter ID to test
`os`	Character string. Operating system to test with. One of "ios", "android", or "unified". Defaults to "ios".
`verbose`	Logical. Whether to print detailed test results. Defaults to TRUE.
`auth_token`	Optional. Character string. Your Sensor Tower API token. Defaults to environment variable SENSORTOWER_AUTH_TOKEN.

Value

List with test results including success status and any error messages

Examples

```
## Not run:
# Test a filter ID
result <- st_test_filter("687df26ac5a19ebcfe817d7f")

# Test silently
result <- st_test_filter("687df26ac5a19ebcfe817d7f", verbose = FALSE)

# Test with different OS
result <- st_test_filter("687df26ac5a19ebcfe817d7f", os = "unified")

## End(Not run)
```

st_top_charts *Fetch Top Apps by Various Metrics*

Description

Retrieves top apps from Sensor Tower based on revenue, downloads ("units"), or active user metrics (DAU, WAU, MAU). This unified function automatically selects the appropriate API endpoint based on the measure specified.

Usage

```
st_top_charts(
  measure = "revenue",
  os,
  comparison_attribute = "absolute",
  time_range = "month",
  date = NULL,
  category = NULL,
  regions,
  end_date = NULL,
  limit = 20,
  offset = NULL,
  device_type = NULL,
  custom_fields_filter_id = NULL,
  custom_tags_mode = NULL,
  data_model = "DM_2025_Q2",
  auth_token = NULL,
  base_url = "https://api.sensortower.com",
  enrich_response = TRUE,
  deduplicate_apps = TRUE
)
```

Arguments

measure	Character string. Metric to measure. Must be one of: - **Revenue/Downloads**: "revenue" (default), "units" - **Active Users**: "DAU", "WAU", "MAU"
os	Character string. Operating System. Must be one of "ios", "android", or "unified". Required.
comparison_attribute	Character string. Comparison attribute type. Must be one of "absolute", "delta", or "transformed_delta". Defaults to "absolute".
time_range	Character string. Time granularity. Must be one of "day", "week", "month", or "quarter". Defaults to "month". Note: "week" is not available when ‘measure‘ is "MAU".
date	Character string or Date object. Start date for the query in "YYYY-MM-DD" format. Defaults to the start of the current month.

`category` Character string or numeric. The ID of the category to filter by. **Required unless ‘custom_fields_filter_id‘ is provided**. Use ‘st_categories()‘ to find valid IDs.

`regions` Character vector or comma-separated string. Region codes (e.g., ‘"US"‘, ‘c("US", "GB")‘, ‘"WW"‘ for worldwide) to filter results. Required.

`end_date` Optional. Character string or Date object. End date for the query in "YYYY-MM-DD" format, inclusive. Only used for revenue/downloads.

`limit` Optional. Integer. Maximum number of apps to return per call. Defaults to 20.

`offset` Optional. Integer. Number of apps to skip for pagination.

`device_type` Optional. Character string. For ‘os = "ios"‘ or ‘os = "unified"‘: "iphone", "ipad", or "total". Defaults to ‘"total"‘.

`custom_fields_filter_id` Optional. Character string. ID of a Sensor Tower custom field filter to apply.

`custom_tags_mode` Optional. Character string. Required if ‘os‘ is ’unified’ and ‘custom_fields_filter_id‘ is provided.

`data_model` Optional. Character string. The data model to use. Defaults to "DM_2025_Q2". Only used for active user metrics.

`auth_token` Optional. Character string. Your Sensor Tower API token.

`base_url` Optional. Character string. The base URL for the API.

`enrich_response` Optional. Logical. If ‘TRUE‘ (default), enriches the response with app metadata and custom metrics.

`deduplicate_apps` Optional. Logical. If ‘TRUE‘ (default), consolidates apps with the same name but different platform/regional SKUs into single rows with aggregated metrics. If ‘FALSE‘, returns separate rows for each SKU.

Value

A [tibble][tibble::tibble] with top app data including enhanced custom metrics like downloads, revenue, retention rates, and more. For sales data (revenue/downloads), app names are automatically looked up using the app IDs since the sales endpoint doesn't provide app names natively.

Revenue Units: Revenue values are standardized to base currency units (dollars, euros, etc.) for consistency across all sensortowerR functions. The function provides a ‘revenue‘ column in base units alongside the original ‘revenue_absolute‘ (in cents).

Data Cleaning: Numeric metric values are automatically cleaned of special characters (

App Deduplication: By default, apps with the same name but different platform/regional SKUs are consolidated into single rows with aggregated metrics (downloads/revenue summed, rates/percentages averaged).

API Endpoints Used

- **All Measures**: ‘GET /v1/{os}/sales_report_estimates_comparison_attributes‘ - Note: DAU/WAU/MAU measures now use the sales endpoint with custom filters for correct sorting

Enhanced Custom Metrics

The function extracts comprehensive custom metrics including: - Downloads: 'downloads_180d_ww', 'downloads_90d_us' - Revenue: 'revenue_180d_ww', 'revenue_90d_us' - Retention: 'retention_1d_us', 'retention_7d_us', 'retention_30d_us' - Monetization: 'rpd_alltime_us', 'arpu_90d_us' - Demographics: 'male_share_us', 'female_share_us' - Platform: 'ios_share_ww', 'android_share_ww'

Examples

```
## Not run:
# Top apps by revenue (default)
top_revenue <- st_top_charts(
  os = "ios",
  category = 6000,  # iOS Games
  regions = "WW"
)

# Top apps by downloads
top_downloads <- st_top_charts(
  os = "android",
  measure = "units",
  category = 6000,
  regions = "US"
)

# Top apps by Daily Active Users with custom filter
# Custom filter URLs from Sensor Tower web interface can be used directly
# Extract the custom_fields_filter_id from the URL parameter 'uai'
top_word_puzzles <- st_top_charts(
  os = "unified",
  measure = "revenue",  # Use revenue but custom filter handles DAU sorting
  custom_fields_filter_id = "5a39e9681454d22f5a5e75ca",  # Word puzzle filter
  custom_tags_mode = "include_unified_apps",
  category = 7019,  # Puzzle category
  regions = "US",
  date = "2025-07-20",
  end_date = "2025-08-18"
)

# Custom time range and region
top_quarter <- st_top_charts(
  os = "ios",
  measure = "revenue",
  time_range = "quarter",
  regions = "US",
  category = 6000
)

## End(Not run)
```

st_top_publishers *Get Top Publishers by Revenue or Downloads*

Description

Retrieves top app publishers ranked by revenue or downloads for a specified category, time period, and country. This function uses the ‘/v1/{os}/top_and_trending/publishers‘ endpoint.

Usage

```
st_top_publishers(
  measure = "revenue",
  os,
  category = 0,
  time_range = "month",
  comparison_attribute = "absolute",
  date,
  end_date = NULL,
  country,
  limit = 25,
  offset = 0,
  device_type = "total",
  include_apps = TRUE,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN")
)
```

Arguments

measure	Character. Metric to rank by: "revenue" or "units" (downloads). Defaults to "revenue".
os	Character. Operating system: "ios", "android", or "unified". Required.
category	Integer or character. Category ID to filter publishers. For iOS use numeric IDs (e.g., 6014 for Games), for Android use strings (e.g., "game"). Use 0 or "all" for all categories.
time_range	Character. Time period: "day", "week", "month", "quarter", or "year". Defaults to "month".
comparison_attribute	Character. Data type to return: "absolute" (total values), "delta" (growth), or "transformed_delta" (growth rate). Defaults to "absolute".
date	Date or character. Start date in "YYYY-MM-DD" format. Required. **Important**: Must align with time_range boundaries: - ‘month‘: Must be first day of month (e.g., 2025-06-01) - ‘week‘: Must be Monday - ‘quarter‘: Must be quarter start (Jan 1, Apr 1, Jul 1, Oct 1) - ‘year‘: Must be January 1st - ‘day‘: Any date allowed Function will error if date doesn’t align. Defaults to 30 days ago.

end_date	Date or character. Optional end date for aggregating multiple periods. If not provided with ‘time_range = "month"‘, "quarter", or "year", it will be automatically set to the last day of the period. **Important**: If provided, must align with time_range boundaries: - ‘month‘: Must be last day of month (e.g., 2025-06-30, 2025-07-31) - ‘week‘: Must be Sunday - ‘quarter‘: Must be quarter end (Mar 31, Jun 30, Sep 30, Dec 31) - ‘year‘: Must be December 31st - ‘day‘: Any date allowed Function will error if date doesn’t align. Use ‘time_range = "day"‘ for custom date ranges.
country	Character. Country or region code (e.g., "US", "GB", "WW" for worldwide). Required.
limit	Integer. Number of publishers to return (1-100). Defaults to 25.
offset	Integer. Number of publishers to skip for pagination. Defaults to 0.
device_type	Character. For iOS: "iphone", "ipad", or "total". For unified: "total". Ignored for Android. Defaults to "total".
include_apps	Logical. Whether to include each publisher’s top apps in the response. Defaults to TRUE.
auth_token	Character. Your Sensor Tower API authentication token. Defaults to the value stored in the ‘SENSORTOWER_AUTH_TOKEN‘ environment variable.

Value

A [tibble][tibble::tibble] containing top publishers with columns: - ‘publisher_id‘: Unique publisher identifier - ‘publisher_name‘: Publisher display name - ‘date‘: Date of the metrics (as provided by API) - ‘date_start‘: Actual start date of the period covered - ‘date_end‘: Actual end date of the period covered - ‘units_absolute‘: Total downloads for the period - ‘units_delta‘: Download growth vs previous period - ‘units_transformed_delta‘: Download growth rate - ‘revenue_absolute‘: Total revenue in cents for the period - ‘revenue_delta‘: Revenue growth vs previous period - ‘revenue_transformed_delta‘: Revenue growth rate - ‘revenue_usd‘: Total revenue in USD (revenue_absolute / 100) - ‘rank‘: Publisher rank based on selected measure - ‘apps‘: Nested tibble with top apps (if include_apps = TRUE)

API Notes

- All revenue values are returned in cents from the API - The function adds a ‘revenue_usd‘ column for convenience - Growth metrics compare to the previous equivalent period - Worldwide data may have a 2-3 day lag vs country-specific data

Date Handling

When using ‘time_range = "month"‘, "quarter", or "year": - Dates MUST align with period boundaries or the function will error - Example: For ‘time_range = "month"‘, use ‘date = "2025-06-01"‘, not ‘"2025-06-27"‘ - This prevents unexpected data ranges from the API - For custom date ranges (e.g., June 27 - July 26), use ‘time_range = "day"‘ and aggregate

Examples

```
## Not run:
# Get top 10 game publishers by revenue for last month
```

```
top_publishers <- st_top_publishers(
  measure = "revenue",
  category = 6014,  # Games category for iOS
  limit = 10
)

# Get top publishers by downloads with growth metrics
growth_publishers <- st_top_publishers(
  measure = "units",
  comparison_attribute = "delta",
  time_range = "week",
  limit = 20
)

# This will ERROR - dates don't align with month boundaries:
# publishers_custom <- st_top_publishers(
#   date = "2025-06-27",        # ERROR: Not start of month!
#   end_date = "2025-07-26",    # ERROR: Not end of month!
#   time_range = "month"
# )

# Correct way for full months (end_date auto-calculated):
publishers_month <- st_top_publishers(
  date = "2025-06-01",      # First day of June
  time_range = "month"      # end_date auto-set to 2025-06-30
)

# Or specify multiple months:
publishers_months <- st_top_publishers(
  date = "2025-06-01",      # First day of June
  end_date = "2025-07-31",  # Last day of July
  time_range = "month"
)

# For custom date ranges (e.g., June 27 - July 26), use daily:
daily_publishers <- purrr::map_df(
  seq(as.Date("2025-06-27"), as.Date("2025-07-26"), by = "day"),
  ~ st_top_publishers(date = .x, time_range = "day", limit = 50)
) %>%
  group_by(publisher_id, publisher_name) %>%
  summarise(total_revenue = sum(revenue_usd))

## End(Not run)
```

---

st_unified_sales_report

Fetch Unified Sales Report Estimates (All Regional SKUs Aggregated)

Description

Retrieves download and revenue estimates using the unified API endpoint, which properly aggregates ALL regional SKUs (app variants from different publishers/regions) within a unified_app_id.

Usage

```
st_unified_sales_report(
  unified_app_id,
  countries,
  start_date,
  end_date,
  date_granularity,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE
)
```

Arguments

unified_app_id	Character string or vector. Sensor Tower unified app ID(s) (24-character hex format, e.g., "67ec0bf3e540b65904256cc4").
countries	Character vector. Country codes (e.g., c("US", "GB", "JP"), or "WW" for worldwide). Required.
start_date	Date or character string. Start date in "YYYY-MM-DD" format. Required.
end_date	Date or character string. End date in "YYYY-MM-DD" format. Required.
date_granularity	Character string. One of "daily", "weekly", "monthly", "quarterly". Required.
auth_token	Optional. Character string. Your Sensor Tower API token.
verbose	Logical. If TRUE, prints progress messages.

Details

This function solves the problem where apps like "Watcher of Realms" have multiple regional versions (e.g., Moonton, Shanghai Moonton, Vizta Games, Skystone Games publishers) that need to be combined for accurate totals.

Why use this instead of st_sales_report()?

When a game has multiple regional SKUs (same game published under different publishers or app IDs in different regions), the standard ‘st_sales_report()‘ function with ID resolution only fetches data for the FIRST iOS and Android app ID. This can result in significantly undercounted revenue/downloads.

Example: "Watcher of Realms" has 4 iOS apps and 3 Android apps across different publishers (Moonton, Vizta Games, Skystone Games, etc.). Using ‘st_sales_report()‘ with the unified_app_id might only fetch data for 2 of these 7 apps.

This function uses the ‘/v1/unified/sales_report_estimates‘ endpoint which automatically aggregates ALL app IDs within the unified entity.

API Response Fields: The unified API returns ‘unified_revenue‘ and ‘unified_units‘ which are automatically converted to ‘revenue‘ (dollars) and ‘downloads‘.

Value

A tibble with columns: - ‘date‘: Date of the data point - ‘country‘: Country code - ‘unified_app_id‘: The unified app ID - ‘revenue‘: Revenue in dollars (converted from cents) - ‘downloads‘: Number of downloads

Examples

```
## Not run:
# Get unified sales data for Watcher of Realms
sales <- st_unified_sales_report(
  unified_app_id = "67ec0bf3e540b65904256cc4",
  countries = "WW",
  start_date = "2024-01-01",
  end_date = "2024-12-31",
  date_granularity = "monthly"
)

# Multiple apps at once
sales <- st_unified_sales_report(
  unified_app_id = c("67ec0bf3e540b65904256cc4", "5ba4585f539ce75b97db6bcb"),
  countries = c("US", "GB", "JP"),
  start_date = "2024-01-01",
  end_date = "2024-12-31",
  date_granularity = "monthly"
)

## End(Not run)
```

---

st_yoy_metrics *Year-over-Year Metrics Comparison*

Description

Fetches metrics for the same date range across multiple years for year-over-year comparisons. Allows flexible date ranges and supports all available metrics including revenue, downloads, and active users.

Usage

```
st_yoy_metrics(
  os,
  unified_app_id = NULL,
  ios_app_id = NULL,
  android_app_id = NULL,
  publisher_id = NULL,
  years = NULL,
  period_start,
```

```
  period_end,
  metrics = c("revenue", "downloads"),
  countries,
  cache_dir = NULL,
  auth_token = Sys.getenv("SENSORTOWER_AUTH_TOKEN"),
  verbose = TRUE,
  granularity,
  use_single_fetch = TRUE
)
```

Arguments

os	Character. Required. Operating system: "ios", "android", or "unified". This determines which platform's data is returned.
unified_app_id	Character vector. Sensor Tower unified app ID(s). Must be 24-character hex format (e.g., "5ba4585f539ce75b97db6bcb").
ios_app_id	Character vector. iOS app ID(s) (e.g., "1234567890").
android_app_id	Character vector. Android package name(s) (e.g., "com.example.app").
publisher_id	Character vector. Publisher ID(s) (alternative to app IDs).
years	Integer vector. Years to compare (e.g., c(2022, 2023, 2024)). If NULL, uses current year and previous year.
period_start	Character string or Date. Start of the comparison period. Can be "MM-DD" format (e.g., "01-01" for Jan 1) or a full date. If a full date is provided, only the month and day are used.
period_end	Character string or Date. End of the comparison period. Can be "MM-DD" format (e.g., "03-31" for Mar 31) or a full date. If a full date is provided, only the month and day are used.
metrics	Character vector. Metrics to fetch. Supports "revenue", "downloads", "dau", "wau", and "mau". Default is both revenue and downloads.
countries	Character vector. Country codes (e.g., "US", "GB", "JP"). Required.
cache_dir	Character. Directory for caching API responses (optional).
auth_token	Character string. Sensor Tower API token.
verbose	Logical. Print progress messages.
granularity	Character. Date granularity for the data (e.g., "daily", "monthly").
use_single_fetch	Logical. If TRUE, uses a single API call to fetch all data. Defaults to TRUE for efficiency.

Details

This function is designed for year-over-year comparisons:

- **Flexible date ranges**: Compare any period (e.g., Q1, specific months, custom ranges) - **Multiple years**: Compare across 2+ years in a single call - **Smart date handling**: Automatically handles leap years and invalid dates - **YoY calculations**: Includes both percentage and absolute change - **Caching**: Reuses cached data to minimize API calls

The function will apply the same calendar period (month/day range) to each specified year, making it easy to compare seasonal trends, campaign periods, or any custom date range across years.

Value

A tibble in tidy/long format with columns: - ‘app_id‘: The app ID used for fetching data - ‘app_id_type‘: Type of app ID ("ios", "android", or "unified") - ‘entity_id‘: App or publisher ID - ‘entity_name‘: App or publisher name - ‘entity_type‘: "app" or "publisher" - ‘year‘: Year of the data - ‘date_start‘: Start date of the period (YYYY-MM-DD) - ‘date_end‘: End date of the period (YYYY-MM-DD) - ‘country‘: Country code - ‘metric‘: The metric name (e.g., "revenue", "downloads", "dau") - ‘value‘: Metric value (units depend on metric type) - ‘yoy_change‘: Year-over-year change (percentage) - ‘yoy_change_absolute‘: Year-over-year change (absolute value)

Examples

```
## Not run:
# Compare Q1 performance across years
q1_comparison <- st_yoy_metrics(
  os = "ios",
  ios_app_id = "553834731",  # Candy Crush iOS
  years = c(2022, 2023, 2024),
  period_start = "01-01",
  period_end = "03-31",
  countries = "US",
  metrics = c("revenue", "downloads")
)

# Compare holiday season (Nov-Dec) across years
holiday_comparison <- st_yoy_metrics(
  os = "unified",
  unified_app_id = "5ba4585f539ce75b97db6bcb",
  years = c(2021, 2022, 2023),
  period_start = "11-01",
  period_end = "12-31",
  countries = c("US", "GB", "JP"),
  metrics = c("revenue", "downloads", "dau")
)

# Compare specific campaign period using full dates
campaign_comparison <- st_yoy_metrics(
  os = "android",
  android_app_id = "com.king.candycrushsaga",
  years = NULL,  # Uses current and previous year
  period_start = as.Date("2024-02-14"),  # Valentine's campaign
  period_end = as.Date("2024-02-28"),
  countries = c("US", "GB", "JP"),
  metrics = c("revenue", "downloads", "dau", "wau")
)

## End(Not run)
```

try_column_operation *Handle column mapping errors gracefully*

Description

Handle column mapping errors gracefully

Usage

```
try_column_operation(expr, data, context = "operation")
```

Arguments

expr	Expression to evaluate
data	Data frame being processed
context	Context for error message

Value

Result of expression or NULL with warning

validate_columns *Validate required columns exist*

Description

Validate required columns exist

Usage

```
validate_columns(data, required, context = "data")
```

Arguments

data	Data frame to check
required	Character vector of required column names
context	Context for error message

Value

TRUE if all columns exist, otherwise stops with informative error

validate_top_charts_data *Data validation functions for Sensor Tower API responses*

Description

Functions to validate and clean data from API responses Validate top charts data

Usage

```
validate_top_charts_data(data, measure, regions)
```

Arguments

data	Data frame from st_top_charts
measure	The measure used (revenue, units, DAU, etc.)
regions	The regions requested

Value

Validated and potentially corrected data frame

Index