Data Table

Max Lang

10/29/2022

dt_intersect

This function returns all rows in `a` that also occur in `b` if column order is ignored.

Inputs - a, b: `data.table` with atomic (non-list) columns and unique column names Output - `data.table`

Code

```
dt_intersect <- function(a, b) {
  assertDataTable(a, types = "atomic", col.names = "unique")
  assertDataTable(b, types = "atomic", col.names = "unique")
  if (!setequal(colnames(a), colnames(b)) ||
    !isTRUE(all.equal(a[FALSE, ], b[FALSE, colnames(a), with = FALSE]))) {
    return(a[FALSE, ])
  }
  b <- unique(b)[, colnames(a), with = FALSE]
  a[b, on = colnames(a)]
}
```

Case Study Shop

You are working for a small electronics reseller located in Germany who sells electronic parts online on the widely used "Yangtze" e-commerce platform. When a customer buys a part from the platform, the reseller orders that part from various sources, which could be domestic or international. The reseller has contracts with different sources, which offer parts for a fixed price, but because of varying exchange rates, the actual cost incurred by the reseller may vary. For every part, there is a possible price in EUR (for if the part is ordered domestically), one in USD (when bought in USA), and one in CNY (when sourced from the PRC). The costs include shipping, customs, and any other overhead created. The reseller has an internal database that lists the costs in all currencies, in which the part is available. The database has the columns `article` (`character`), a unique article identifier made up of 6 digits; `description` (`character`), a short description of the article, as well as `cost.eur`, `cost.usd`, `cost.cny` – the prices of each part if bought from a source in the respective currency. The `cost`-columns are `numeric`. `cost.usd` and `cost.cny` may have missing values if the part is not available in the given currency (the part is always available in EUR, so `cost.eur` never contains missing values). An example exerpt from the database looks like this:

```
costs.example <- rbindlist(list(
    list(article = NULL, description = NULL,       cost.eur = NULL,  cost.usd = NULL,  cost.cny = NULL)
    list("026434",       "Resistor, 66k0",         0.13,             0.20,             1.10),
    list("100733",       "Wire, copper, 1mm x 1m", 2.10,             NA,               37.17),
    list("005235",       "LED, green",             0.30,             0.10,             0.65),
    list("005236",       "LED, blue",              0.31,             0.33,             NA)
))
```

The reseller also keeps a record of exchange rates for various currencies. It is a table that lists how many EUR the resller has to spend to get one unit of each currency, including exchange commission etc. An example exerpt of this table for the week of 2021-07-26 to 2021-08-01 looks like the following:

```
exchange.rates.example <- rbindlist(list(
    list(date = NULL,  xr.eur = NULL, xr.usd = NULL, xr.cny = NULL),
    list("2021-07-26", 1.000000,      0.848372,      0.130817),
    list("2021-07-27", 1.000000,      0.846682,      0.130218),
    list("2021-07-28", 1.000000,      0.846721,      0.130340),
    list("2021-07-29", 1.000000,      0.847325,      0.130433),
    list("2021-07-30", 1.000000,      0.842195,      0.130127),
    list("2021-07-31", 1.000000,      0.841314,      0.130227),
    list("2021-08-01", 1.000000,      0.842673,      0.130414)
))
```

Finally, the reseller has a record of sales made, which records a transaction id (`txnid`, a `character`), the article bought (`article`, `character`) as found in the costs database, the quantity of items bought (`quantity`, always an integer numeric greater or equal to 1), and the date of the transaction (`date`). An example is the following:

```
transactions.example <- rbindlist(list(
    list(txnid = NULL, article = NULL, quantity = NULL, date = NULL),
    list("000012443", "005236",       1000,             "2021-07-31"),
    list("000012449", "005235",       100,              "2021-07-27"),
    list("000012547", "100733",       20,               "2021-07-28"),
    list("000012541", "100733",       5,                "2021-07-27"),
    list("000012440", "005235",       500,              "2021-08-01"),
    list("000012442", "005236",       300,              "2021-07-26")
))
```

The reseller is selling the parts at the same price at which the parts could be bought in Germany, so the price at which the parts get sold is always the cost listed in the `cost.eur` column of the costs table. This is so that the customers do not feel like they could have bought the parts cheaper from a different vendor in Germany.

Write a function that is given a table of transactions and costs, and returns the transactions table with a `price` column added, indicating the amount of money that the customer paid for the given item.

Input

- `transactions`: a `data.table` with the columns and format as given in the examples above.
- `costs`: a `data.table` with the columns and format as given in the examples above.

Return

A `data.table` with the columns `txnid`, `article`, `quantity`, `date`, and `price`, in that order. The first few columns should have the identical content as the content given to the `transactions` argument, in the same order. The `price` column should be a `numeric` column giving the price that the customer paid for the given item in the given quantity. This price is given as the EUR value of the article in the `costs` table, multiplied by the quantity of items bought.

The `transactions` table may be empty (i.e. contain 0 rows), make sure the return value is then also a 0-row `data.table` with the right columns, and where the columns have the right type.

You can rely on the fact that every `article` listed in the given `transactions` is also present in the `costs` table.

```r
ex01TransactionPrice <- function(transactions, costs) {
  assertDataTable(transactions)
  assertDataTable(costs)
  if (!nrow(transactions)) {
    return(data.table(txnid = character(0), article = character(0),
                      quantity = numeric(0), date = character(0), price = numeric(0)))
  }
  costs[transactions, .(txnid, article, quantity, date, price = cost.eur * quantity), on = "article"]
}
```

The reseller always buys the parts from the cheapest offering whenever parts are ordered, given the exchange rates on that day. The sourcing cost that a transaction incurs may therefore be different on different days for the same item, because exchange rates vary.

Write a function that is given a table of costs, exchange rates, and transactions, and returns the transactions table with a `cost` column added.

Input

- `transactions`: a `data.table` with the columns and format as given in the examples above.
- `costs`: a `data.table` with the columns and format as given in the examples above.
- `exchange.rates`: a `data.table` with the columns and format as given in the examples above.

Return

A `data.table` with the columns `txnid`, `article`, `quantity`, `date`, and `cost`, in that order. The cost should be in EUR and give the money spent on the items by the reseller, assuming that the reseller bought the parts from the cheapest offering, given the exchange rate. The cost is therefore calculated as the quantity of items bought in that transaction, multiplied with the minimum from either (1) the article's cost in EUR, (2) the cost in USD, multiplied with the USD exchange rate for the day of the transaction (unless the cost in USD is NA), or (3) the cost in CNY, multiplied with the CNY exchange rate for that day (unless the cost in CNY is NA). Cost should *not* be rounded to cents.

The `transactions` table may be empty (i.e. contain 0 rows), make sure the return value is then also a 0-row `data.table` with the right columns, and where the columns have the right type.

You can rely on the fact that every `article` listed in the given `transactions` is also present in the `costs` table, and that the `date` listed in the `transactions` is also present in the `exchange.rates` table.

```r
ex02TransactionCost <- function(transactions, costs, exchange.rates) {
  assertDataTable(transactions)
  assertDataTable(costs)
  assertDataTable(exchange.rates)
  if (!nrow(transactions)) {
    return(data.table(txnid = character(0), article = character(0),
                      quantity = numeric(0), date = character(0), price = numeric(0)))
  }
  costs[exchange.rates[transactions, on = "date"],
    .(txnid, article, quantity, date,
      cost = pmin(cost.eur, cost.usd * xr.usd, cost.cny * xr.cny, na.rm = TRUE) * quantity),
    on = "article"]
}
```

The revenue is calculated from the price paid by the customer, minus the commission paid to the sales platform, minus the cost of buying the parts. The Yangtze e-commerce platform takes a commission of 10%

of each sale, paid on the price paid by the customer. Write a function that, given the transactions table, the table of prices, and of exchange rates, calculates the revenue for each transaction (which may be negative if the cost is more than 90% of the price paid by the customer).

Input

- `transactions`: a `data.table` with the columns and format as given in the examples above.
- `costs`: a `data.table` with the columns and format as given in the examples above.
- `exchange.rates`: a `data.table` with the columns and format as given in the examples above.

Return

A `data.table` with the columns `txnid`, `article`, `quantity`, `date`, and `revenue`, in that order. The revenue is the transaction's "price" (e.g. as calculated by `ex01TransactionPrice()`), minus the transaction's "cost" (e.g. as calculated by `ex02TransactionCost()`), minus the commission (which is 10% of the price). Revenue should *not* be rounded to cents.

You are encouraged to call `ex01TransactionPrice()` and `ex02TransactionCost()` in your solution, although you don't have to. Make sure that the return value of your function *only* contains the columns listed above, and in the correct order; should you use `ex01TransactionPrice()` and `ex02TransactionCost()`, you may have to remove the `price` and `cost` columns.

The `transactions` table may be empty (i.e. contain 0 rows), make sure the return value is then also a 0-row `data.table` with the right columns, and where the columns have the right type.

You can rely on the fact that every `article` listed in the given `transactions` is also present in the `costs` table, and that the `date` listed in the `transactions` is also present in the `exchange.rates` table.

```r
ex03TransactionRevenue <- function(transactions, costs, exchange.rates) {
  assertDataTable(transactions)
  assertDataTable(costs)
  assertDataTable(exchange.rates)
  if (!nrow(transactions)) {
    return(data.table(txnid = character(0), article = character(0),
                      quantity = numeric(0), date = character(0), revenue = numeric(0)))
  }
  copy(transactions)[, revenue := (
      ex01TransactionPrice(transactions, costs)$price * 0.9 -
      ex02TransactionCost(transactions, costs, exchange.rates)$cost
    )][]
}
```

The reseller suspects that some of the parts being sold are more profitable than others and wants to analyse this further. The overhead incurred by selling parts is approximately constant for each transaction, independent of whether customers buy 1 unit or 1000 units, so the reseller cares about the average revenue generated per transaction, for each of its items on sale. Write a function that calculates the average revenue per transaction generated by each article.

Input

- `transactions`: a `data.table` with the columns and format as given in the examples above.
- `costs`: a `data.table` with the columns and format as given in the examples above.
- `exchange.rates`: a `data.table` with the columns and format as given in the examples above.

Return

A `data.table` with the columns `article`, `description`, `revenue.avg`, in that order. The table should contain one row for each row in the `costs` input. The revenue is the average revenue, calculated as in ex03TransactionRevenue, of all transactions where the article is sold; articles that are not sold in any transaction have average revenue of 0. The returned `data.table` should be sorted by `revenue.avg` descending, with highest average revenue coming first (ordering among ties may be arbitrary). Average revenue should *not* be rounded to cents.

You are encouraged to call `ex03TransactionRevenue()` in your solution, although you don't have to.

The `transactions` table may be empty (i.e. contain 0 rows); the `price` table is never empty.

You can rely on the fact that every `article` listed in the given `transactions` is also present in the `costs` table, and that the `date` listed in the `transactions` is also present in the `exchange.rates` table.

```r
ex04AverageRevenue <- function(transactions, costs, exchange.rates) {
  assertDataTable(transactions)
  assertDataTable(costs)
  assertDataTable(exchange.rates)

  revenue <- ex03TransactionRevenue(transactions, costs, exchange.rates)
  articlerevenue <- revenue[costs, .(article, description, revenue), on = "article"]
  meanrevenue <- articlerevenue[, .(revenue.avg = mean(revenue)), by = c("article", "description")]
  setnafill(meanrevenue, fill = 0, cols = "revenue.avg")[order(-revenue.avg)]
}
```

The reseller considers a strategy where a part is listed as "not available" whenever selling that part would incur a loss (i.e. negative revenue) given exchange rates on that day. Because all costs and prices scale linearly with number of units sold, whether a part is profitable or not on a given day is independent of the units. Write a function that calculates the average revenue for each article, counting only the transactions with positive revenue.

Input

- `transactions`: a `data.table` with the columns and format as given in the examples above.
- `costs`: a `data.table` with the columns and format as given in the examples above.
- `exchange.rates`: a `data.table` with the columns and format as given in the examples above.

Return

A `data.table` with the columns `article`, `description`, `revenue.positive.avg`, in that order. The table should contain one row for each row in the `price` input. The revenue is the average revenue, calculated as in ex03TransactionRevenue, of all transactions where the article is sold and has a positive revenue; articles that are not sold in any transaction or only incur negative revenue have average positive revenue of 0. The returned `data.table` should be sorted by `revenue.positive.avg` descending, with highest average positive revenue coming first (ordering among ties may be arbitrary). Avg positive revenue should *not* be rounded to cents. You are encouraged to call `ex03TransactionRevenue()` in your solution, although you don't have to.

The `transactions` table may be empty (i.e. contain 0 rows); the `price` table is never empty.

You can rely on the fact that every `article` listed in the given `transactions` is also present in the `costs` table, and that the `date` listed in the `transactions` is also present in the `exchange.rates` table.

```r
ex05AveragePositiveRevenue <- function(transactions, costs, exchange.rates) {
  assertDataTable(transactions)
  assertDataTable(costs)
  assertDataTable(exchange.rates)

  revenue <- ex03TransactionRevenue(transactions, costs, exchange.rates)[revenue > 0]
  articlerevenue <- revenue[costs, .(article, description, revenue), on = "article"]
  meanrevenue <- articlerevenue[, .(revenue.avg = mean(revenue)), by = c("article", "description")]
  setnafill(meanrevenue, fill = 0, cols = "revenue.avg")[order(-revenue.avg)]
}
```