3.1-Calendar-Test.R

frmhelp

2021-01-12

# 3.1 Calendar Test.R  
# Insight: Illustrate various day counting with jrvFinance package  
# jrvFinance: yearFraction()  
# date: mdy.date()  
# Exploring the package jrvFinance  
# rmarkdown::render("3.1 Calendar Test.R", "word\_document")  
rm(list = ls()) # Take out the Environment "trash"  
cat("\014") # Clear Console, making error checking easier.

while (!is.null(dev.list())) dev.off() # Clear old plots  
par(family = 'Times New Roman') # Globally set fonts for graphs  
# Libraries  
# date - functions for handling dates  
# jrvFinance - basic financial analysis functions (similar to spreadsheets)   
#?jrvFinance # Remember the help command (only works if installed)  
Packages <- c("jrvFinance", "date", "tis")   
if(length(setdiff(Packages, rownames(installed.packages()))) > 0) {  
 install.packages(setdiff(Packages, rownames(installed.packages())))  
} # Make sure libraries are installed on this computer  
lapply(Packages, library, character.only=TRUE) # Load and attach libraries

## [[1]]  
## [1] "tis" "jrvFinance" "date" "stats" "graphics"   
## [6] "grDevices" "utils" "datasets" "methods" "base"   
##   
## [[2]]  
## [1] "tis" "jrvFinance" "date" "stats" "graphics"   
## [6] "grDevices" "utils" "datasets" "methods" "base"   
##   
## [[3]]  
## [1] "tis" "jrvFinance" "date" "stats" "graphics"   
## [6] "grDevices" "utils" "datasets" "methods" "base"

rm(Packages)  
# Computing fraction of year  
InputStartMonth <- 6  
InputStartDay <- 1  
InputStartYear <- 2020  
JulianStartDate = mdy.date(InputStartMonth,InputStartDay,InputStartYear,nineteen = FALSE)  
InputEndMonth <- 6  
InputEndDay <- 1  
InputEndYear <- 2021  
JulianEndDate = mdy.date(InputEndMonth, InputEndDay, InputEndYear, nineteen = FALSE)  
d1 <- JulianStartDate  
d2 <- JulianEndDate  
r1 <- mdy.date(1, 1, InputStartYear, nineteen = FALSE) # Beginning of year, not June  
r2 <- mdy.date(1, 1, InputEndYear, nineteen = FALSE)  
# 2020 is a leap year, see jrvFinance package help

F1 <- yearFraction(d1, d2, r1, r2, freq = 2, convention = "30/360") # 360/360  
F2 <- yearFraction(d1, d2, r1, r2, freq = 1, convention = "ACT/ACT") # 366/365  
F3 <- yearFraction(d1, d2, r1, r2, freq = 2, convention = "ACT/360") # 366/360  
F4 <- yearFraction(d1, d2, r1, r2, freq = 2, convention = "30/360E") # 360/360 (semi ?)  
F5 <- yearFraction(d1, d2, r1, r2, freq = 12, convention = "ACT/ACT") # Monthly  
F6 <- daycount.actual(d1, d2, variant = "bond")  
F7 <- daycount.30.360(d1, d2, variant = "US")  
F1; F2; F3; F4; F5; F6; F7

# Unless convention=“ACT/ACT”, you can ignore freq (by default, freq=2)

# F4 <- yearFraction(d1, d2, r1, r2, convention = "30/360E")

## [1] 1

## [1] 0.9972678

## [1] 1.013889

## [1] 1

## [1] 0.08310565

## [1] 365

## [1] 360

# Find today on the system  
?'Sys.Date'  
TodaysDate = Sys.Date() # But in unusable format  
TodaysYear <- as.integer(format(TodaysDate, "%Y")) # year -- upper case  
TodaysMonth <- as.integer(format(TodaysDate, "%m")) # month -- note case sensitive  
TodaysDay <- as.integer(format(TodaysDate, "%d")) # day -- note case sensitive  
JulianTodaysDate = mdy.date(TodaysMonth, TodaysDay, TodaysYear, nineteen = FALSE)  
TodaysDate; TodaysYear; TodaysMonth; TodaysDay

## [1] "2021-01-12"

## [1] 2021

## [1] 1

## [1] 12

# Difference in days  
JulianTodaysDate = mdy.date(TodaysMonth, TodaysDay, TodaysYear, nineteen = FALSE)  
InputStartMonth <- 6  
InputStartDay <- 14  
InputStartYear <- 1986  
JulianStartDate = mdy.date(InputStartMonth,InputStartDay,InputStartYear,nineteen = FALSE)  
ActualDays = abs(JulianTodaysDate - JulianStartDate)  
YearsIn365 = ActualDays/365.0  
YearsIn365

## [1] 34.60548

# Just checking various date functions  
SDate = date.mdy(JulianStartDate)  
SMonth = SDate$month  
SDay = SDate$day  
SYear = SDate$year  
SDate; SMonth; SDay; SYear

## $month  
## [1] 6  
##   
## $day  
## [1] 14  
##   
## $year  
## [1] 1986

## [1] 6

## [1] 14

## [1] 1986

EDate = date.mdy(JulianTodaysDate)  
EMonth = EDate$month  
EDay = EDate$day  
EYear = EDate$year  
EDate; EMonth; EDay; EYear

## $month  
## [1] 1  
##   
## $day  
## [1] 12  
##   
## $year  
## [1] 2021

## [1] 1

## [1] 12

## [1] 2021

x <- 20220101  
x <- mdy.date(1, 1, 2021, nineteen = FALSE)  
# See package tis: Time Indexes and Time Indexed Series  
board <- FALSE # Presidential inauguration is not a holiday  
nextBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)

## [1] 20210104  
## class: ti

previousBusinessDay(x, holidays = NULL, goodFriday = F, board = F, inaug = board)

## [1] 20201231  
## class: ti

isHoliday(x, goodFriday = TRUE, board = FALSE, inaug = board, businessOnly = TRUE)

## [1] TRUE

isBusinessDay(x)

## [1] FALSE

isGoodFriday(x)

## [1] FALSE

isEaster(x)

## [1] FALSE

years <- 2021  
holidays(years, goodFriday = F, board = F, inaug = board, businessOnly = T)

## NewYears MLKing GWBirthday Memorial Independence   
## 20210101 20210118 20210215 20210531 20210705   
## Labor Columbus Veterans Thanksgiving   
## 20210906 20211011 20211111 20211125

federalHolidays(years, board = F, businessOnly = T)

## NewYears MLKing GWBirthday Memorial Independence   
## 20210101 20210118 20210215 20210531 20210705   
## Labor Columbus Veterans Thanksgiving   
## 20210906 20211011 20211111 20211125

goodFriday(years)

## GoodFriday   
## 20210402

easter(years)

## [1] 20210404

inaugurationDay(years)

## Inauguration   
## 20210120

startTi <- 20190101  
endTi <- 20210101  
holidaysBetween(startTi, endTi, goodFriday = F, board = F, inaug = board,  
 businessOnly = T)

## NewYears MLKing GWBirthday Memorial Independence   
## 20190101 20190121 20190218 20190527 20190704   
## Labor Columbus Veterans Thanksgiving Christmas   
## 20190902 20191014 20191111 20191128 20191225   
## NewYears MLKing GWBirthday Memorial Labor   
## 20200101 20200120 20200217 20200525 20200907   
## Columbus Veterans Thanksgiving Christmas NewYears   
## 20201012 20201111 20201126 20201225 20210101

#  
# Illustration of building your own date function  
# Test whether a date is valid and either move it to preceeding or following  
#  
source('Adjust Date.R')  
TestMonth <- 9  
TestDay <- 26 # Normal Tuesday  
TestYear <- 2019  
Convention <- "MBP" # Modified Business Following or MBP (Preceeding)  
AdjustDate(TestMonth, TestDay, TestYear, Convention)

## [1] 26Sep2019

TestDay <- 29 # Sunday  
AdjustDate(TestMonth, TestDay, TestYear, Convention)

## [1] 27Sep2019

Convention <- "MBF"  
AdjustDate(TestMonth, TestDay, TestYear, Convention)

## [1] 30Sep2019