Introduction to BBOTools

Mindy Mallory

2016-03-08

Table of Contents

Introduction

This vignette is written with graduate students in mind. The intent is to get graduate students and other collaborators who may not be familiar with data analysis with R and RStudio up and running doing basic analysis of the BBO datasets in as little time as possible. This vignette covers the basics of how to use R, RStudio, and the `BBOTools` package to analyse the CME's BBO datasets. First download and install R from https://www.r-project.org/, then download and install RStudio from https://www.rstudio.com/. Before continuing, it is expected the reader will have completed an extensive introductory R tutorial. There are several available.

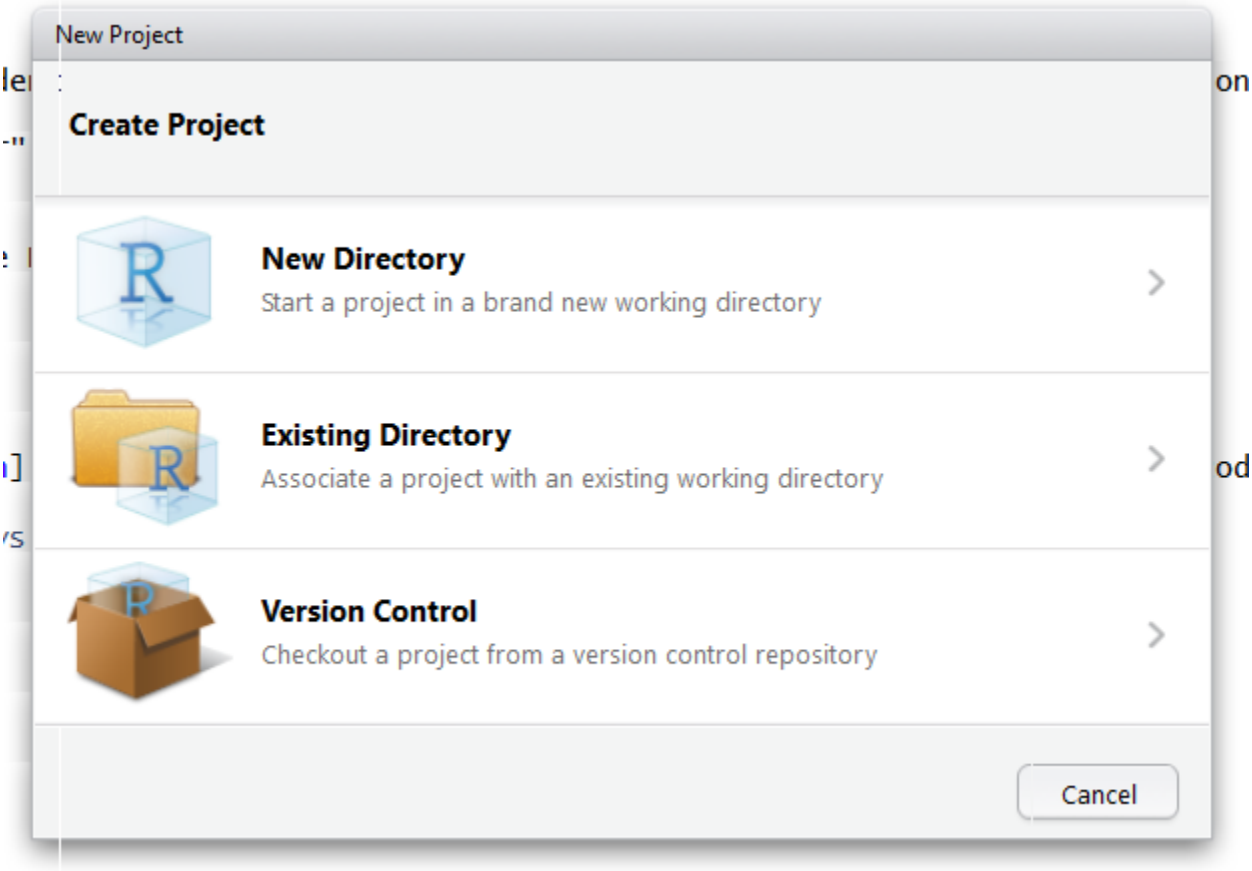
1. http://www.r-tutor.com/r-introduction
2. https://www.datacamp.com/courses/free-introduction-to-r
3. http://www.cyclismo.org/tutorial/R/
4. http://tryr.codeschool.com/

There are other others, but this is a start. For one's own research it is required that you will go much deeper than what is covered in these introductory tutorials, but hopefully you will come out of the tutorials with an ability to Google what you want to do and how to implement it in R. This package makes heavy use of the `data.table` package. The `data.table` package was written with careful memory management in mind. Since we are working with large files, memory management is essential. Read the excellent series of vignettes for the data.table package for details about how `data.table` makes efficient use of memory verses data.frames. The rest of this vignette will assume the reader has at least installed the latest version of R and RStudio. IF you are familiar with R and RStudio, skip ahead to The BBOTools Package.

Table of Contents

Set Up Your First Project

In Rstudio, click "New Project" from the File menu. You will see a menu like the following:

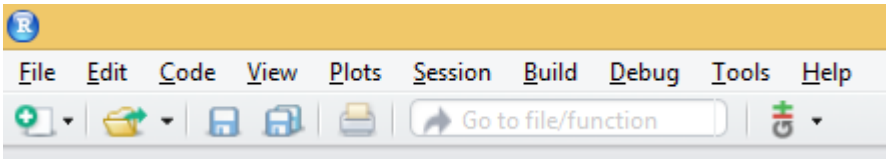


Choose "New Directory", and then "Empty Project". You will be asked to navigate to the place where you want your project folder to reside. This creates a file in the directory you chose named `YourProject.Rproj`. Assuming you named your project 'YourProject'. This should be the first step on any analysis in R. Working within the framework of an R Project tells RStudio where to put things by default. Without it, the default working directory will inevitably end up being something unintuitive or unexpected, and you will have to tell your R code hard file paths every time you want to refer to a data file or code script. This makes working across machines or with collaborators difficult because you have to change the file path every time you are on a different machine.

Table of Contents

Your First R Script

Open a new R script file by clicking the green '+' icon below the File menu in RStudio.



Follow along by copy and pasting the commands from this vignette into your R script and executing the commands. To execute a command, place the cursor on the line you want to execute and click "Run". In the figure below, executing line 2 will assign a value of '1' to 'x'.

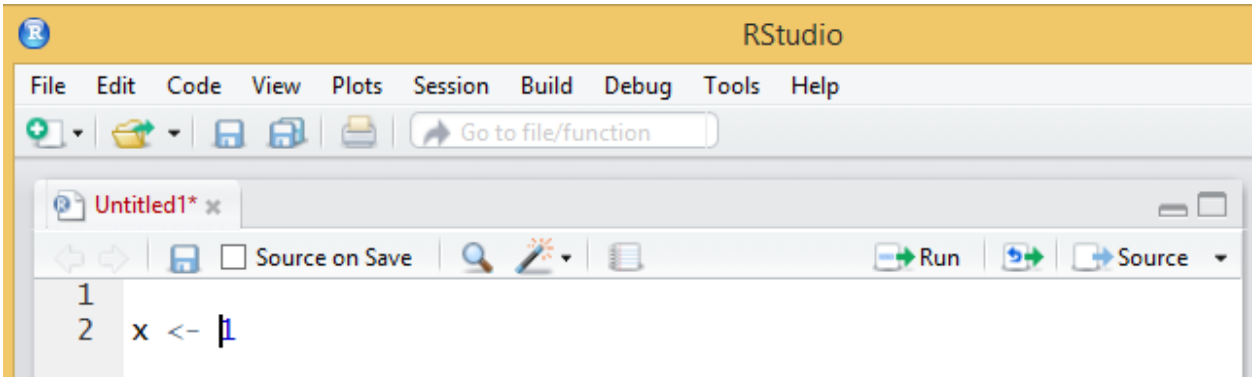


Table of Contents

The BBOTools Package

This package provides tools for working with the Top-of-the-Book (Best Bid-Offer) datasets from the CME Group's Historical Datamine. These data provide record of every revision to the best bid or best offer. These revisions may have resulted from a new limit order arriving to the market, a limit order being cancelled, or a transaction matching a market order with a limit order or a limit order with another limit order.

The raw data provided by the CME Group are organized as large fixed-width files. The files are large enough that if one is not careful about memory use, even simple analyses becomes a problem. An extension of R's base data.frame called data.table was recently developed by Matt Dowle and Arun Srinivasan makes working with large datasets in R much more feasible.

This package utilizes the data.table package extensively. Most of the functions in the BBOTools package are simple wrapper functions that make working with the BBO data more convenient.

Table of Contents

BBOTools as a Private GitHub Repository

Currently, BBOTools is hosted in a private repository in the ProfMalloryResearch GitHub repository. To use this package you will need to have a GitHub account and be added to the ProfMalloryResearch organization. If we plan to collaborate, contact Mindy Mallory to be added.

Even after being added to the organization you will still need to generate a personal access token from GitHub in order to authenticate and install packages from this private organization. Further, you will want to save it in such a way that it will not be publicly visible anywhere but on your local machine.

Steps to get a PAT

1. Go to your account settings on GitHub.com
2. Click 'Personal Access Tokens'
3. Click 'Generate New Token'
4. You will have a long string of numbers that is your token.

You want to save it in a place that will not be accessible to the public in any way. For example, you don't want it to get uploaded to a public (or even private) repository on GitHub.com.

Steps to save your PAT in a safe place.

1. Find out what R's 'home' directory is on your machine.

- execute `normalizePath("~/")` in the R console. The file identified is where you should save a plain text file called .Renvirion

2. Navigate to and open the .Renviron in a plain text editor.
3. Type the following: GITHUB_PAT = "StringofnumbersthatisyourPAT"
4. Save .Renviron

Your PAT should be stored in a safe place and accessible from the R console. To check execute the following r code: `Sys.getenv("GITHUB_PAT")` You should see your own PAT in the output.

Table of Contents

Installing BBOTools from GitHub

Now to install the package `BBOToolkit` from ProfMalloryResearch execute the following r code:

```
install_github("ProfMalloryResearch/BBOToolkit", auth_token = Sys.getenv("GITHUB_PAT"))
library(BBOTools)
```

This installs and loads the BBOTools package into your R environment.

Table of Contents

Importing BBO Data with bboread()

The CME Group offers a sample of the BBO data. Go to Historical Datamine and download the corn futures bbo data sample. Extract the compressed file to your R project folder so that the file `XCBT_C_FUT_110110.TXT` resides in your project folder.

Next, make sure you have installed and loaded the `data.table` and `readr` packages.

```
#install.packages("data.table")
#install.packages("readr")
library(data.table)
library(readr)
library(tidyr)
library(BBOTools)
```

Next, lets load the sample BBO data and take a look at it.

```
DATA <- as.data.table(bboread('XCBT_C_FUT_110110.TXT'))
DATA
```

```
##           TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##       1:  20110110    130604          10         1103          0 0005950
##       2:  20110110    161500          20         1103         12 0005960
##       3:  20110110    161500          20         1103          8 0005960
##       4:  20110110    161554          30         1103          0 0005956
##       5:  20110110    161554          40         1103          8 0005956
##      ---
## 1038066:  20110110    105201          80         1412          2 0005200
## 1038067:  20110110    105201          80         1412          1 0005124
```

```
## 1038068:  20110110    133000          90         1412          2 0005200
## 1038069:  20110110    133000          90         1412         39 0005120
## 1038070:  20110110    142729         100         1412          0 0005200
##           ASKBID EntryDate
##       1:     NA    110109
##       2:      A    110109
##       3:      B    110109
##       4:     NA    110109
##       5:      A    110109
##      ---
## 1038066:      A    110110
## 1038067:      B    110110
## 1038068:      A    110110
## 1038069:      B    110110
## 1038070:     NA    110110
```

This loads the sample BBO data into your R environment as a `data.table` object. The use of the `as.data.table()` function here is important because without it the data get loaded as a `data.frame` object.

```
DATA <- bboread('XCBT_C_FUT_110110.TXT')
DATA
```

Notice the difference in the way the object, `DATA`, displays as a data.table in the first case and a data.frame in the second case. To clean up our memory after that example, remove the `DATA` variable and load it again as a data.table.

```
rm(DATA)
DATA <- as.data.table(bboread('XCBT_C_FUT_110110.TXT'))
DATA
```

Click on the `DATA` variable in the 'Environment' tab in RStudio. This displays the data.table object in a spreadsheet-like display. The `bboread()` function only imported the data columns we need for analyzing this data. The raw data contains many additional columns that are unnecessary for our purposes. If you want to see the full dataset, look at the documentation for using the `read_fwf()` function of the readr package.

We have imported the following data columns:

```
names(DATA)
```

```
## [1] "TradeDate"    "TradeTime"    "TradeSeqNum"  "DeliveryDate"
## [5] "TrQuantity"   "TrPrice"      "ASKBID"       "EntryDate"
```

Table of Contents

Data Description

TradeDate and EntryDate

`EntryDate` refers to the date on which the trade was placed, while `TradeDate` refers to the settlement date associated with those trades. These are character strings representing dates in the format `%YYYY%mm%dd`. `TradeDate` and `EntryDate` are only different for trades that take place in the overnight session before midnight.

For example, if you look at the first few lines of `DATA` you can tell that these are the first trades and quotes from the beginning of the overnight trading session. These trades take place on calendar date 2011-01-09, but are recorded as trades on contract settlement day 2011-01-10 since daily settlement happens at the end of the daytime session (1:15pm CST).

TradeTime

`TradeTime` is a character string that contains the time the trade or quote took place - to the second. It is in format `%H:%m:%s`. For example, the first line reads `130604`, that is 1:06:04pm CST.

TradeSeqNum

`TradeSeqNum` is simply a running index of the trades and quotes as they occur in sequence. For some reason, this variable always increments by 10. Prior to 2012, CME had the practice of copying both the bid and ask when one changes. For example, if someone cancels a limit order to sell, the ask line will change. The quantity offered goes down by the order size that was canceled and the ask price may go up if it was the only limit order to sell at the top of the book. So in the BBO dataset a new ask must be recorded for the top of the book and the `TradeSeqNum` is incremented by 10. In addition, the best bid information is simply copied with the new `TradeSeqNum`. This makes building the top-of-the-book easy for data prior to 2012. Beginning in 2012, the data no longer contain the bid or ask that was unchanged, they only record the bid if the best bid was revised and the only record the ask if the best ask was revised. Also, `TradeSeqNum` no longer increments by 10. An example of building the top-of-the-book for both the pre-2012 and post-2012 case follows in the Useful Scripts section.

DeliveryDate

This variable identifies which maturity futures contract the row is recorded for. Format is `%Y%m` so that `1103` stands for the March 2011 delivery contract.

TrQuantity

This variable contains the size of the bid, ask, or transaction.

TrPrice

This variable contains the price of the bid, ask, or transaction. This variable is a character string with leading zeros before displaying the price in cents per bushel. A quirk of these markets is that prices are quoted to the eighth of a cent and the last character of this price variable represents the fraction of an eighth of a cent. For example a price of `0005956` represents `595 + 6/8 = 595.75` cents per bushel. This package contains a function called `decimalprices()` that converts this variable to a decimal for you. An example will follow in this vignette.

ASKBID

This column identifies if the row represents an ask ('A'), bid ('B'), or transaction ('NA').

Table of Contents

Showcasing the Custom BBOTools Functions

This section provides detailed examples of how to execute tasks common to the analysis of the BBO data.

`decimalprices()`: Converting 1/8th of a cent 'Ticks' to Decimal

For the grains and oilseeds futures contracts prices are quoted in 1/4 of a cent increments and the last character of the `TrPrice` variable is the numerator in of the fraction expressed as 1/8th of a cent. For example, the only numbers that appear as last character in the `TrPrice` column are 0, 2, 4, and 6 representing $0/8 = 0.00$, $2/8 = 0.25$, $4/8 = 0.5$, and $6/8 = 0.75$.

The function `decimalprices()` takes the column `TrPrice` and converts the character string to a floating decimal price and automatically handles the fraction of a cent issue from the first column.

```
DATA[, Price := decimalprices(DATA$TrPrice)]
DATA
```

```
##           TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##       1:  20110110    130604          10         1103          0 0005950
##       2:  20110110    161500          20         1103         12 0005960
##       3:  20110110    161500          20         1103          8 0005960
##       4:  20110110    161554          30         1103          0 0005956
##       5:  20110110    161554          40         1103          8 0005956
##      ---
## 1038066:  20110110    105201          80         1412          2 0005200
## 1038067:  20110110    105201          80         1412          1 0005124
## 1038068:  20110110    133000          90         1412          2 0005200
## 1038069:  20110110    133000          90         1412         39 0005120
## 1038070:  20110110    142729         100         1412          0 0005200
##           ASKBID EntryDate  Price
##       1:     NA    110109 595.00
##       2:      A    110109 596.00
##       3:      B    110109 596.00
##       4:     NA    110109 595.75
##       5:      A    110109 595.75
##      ---
## 1038066:      A    110110 520.00
## 1038067:      B    110110 512.50
## 1038068:      A    110110 520.00
## 1038069:      B    110110 512.00
## 1038070:     NA    110110 520.00
```

`RollContracts()`: Rolling Contracts to Create a Nearby Series

A common task is to string together a series of 'nearby' or 'first deferred' contracts. In this package the function `RollContracts(accum, r)` takes a list of `data.table` objects for different dates and identifies which contracts are the nearby, first deferred, and second deferred. More distant contracts than these are removed. The value `r` can be set to specify the date of the month prior to expiration on which the roll should occur. The 20th is the default. A new column called `Deferreds` is created, allowing the nearby contract across multiple days to be strung together to form a nearby series.

If you are working with data prior to 2012, it is natural and best to load each day's data into elements of a list because they distribute every day's worth of data as separate text files. So for data prior to 2012, supplying the `RollContracts()` function with a list of data.table objects will be natural.

Starting in 2012, they changed the format of how they report quote revisions and also they deliver a few months worth of data in one data file. In this case because of system memory limitations you will want only load one file at a time. If you need to identify a nearby and/or deferred contracts, place the data of this one file as a single element in a list so that you can pass it to the function `RollContracts()`. `RollContracts()` does several data munging processes via calls to the `lapply()` function. Some of these actions can be completed as `data.table` actions on the j column, but some of them could not so supplying the data as a list is required for the `RollContracts()` function.

```
accum <- as.list(NULL)
accum[[1]] <- DATA
accum[[2]] <- DATA                  # accum is a list of two 'days' worth of BBO data
accum <- RollContracts(accum, r = 20)
accum
```

```
## [[1]]
##         TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##      1:  20110110    130604          10         1103          0 0005950
##      2:  20110110    161500          20         1103         12 0005960
##      3:  20110110    161500          20         1103          8 0005960
##      4:  20110110    161554          30         1103          0 0005956
##      5:  20110110    161554          40         1103          8 0005956
##     ---
## 625385:  20110110    143448      944770         1107          1 0006192
## 625386:  20110110    145927      944780         1107          5 0006200
## 625387:  20110110    145927      944780         1107          1 0006192
## 625388:  20110110    154751      944790         1107          8 0006200
## 625389:  20110110    154751      944790         1107          1 0006192
##         ASKBID EntryDate  Price    Deferreds
##      1:     NA    110109 595.00       Nearby
##      2:      A    110109 596.00       Nearby
##      3:      B    110109 596.00       Nearby
##      4:     NA    110109 595.75       Nearby
##      5:      A    110109 595.75       Nearby
##     ---
## 625385:      B    110110 619.25 2st Deferred
## 625386:      A    110110 620.00 2st Deferred
## 625387:      B    110110 619.25 2st Deferred
## 625388:      A    110110 620.00 2st Deferred
## 625389:      B    110110 619.25 2st Deferred
##
## [[2]]
##         TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##      1:  20110110    130604          10         1103          0 0005950
##      2:  20110110    161500          20         1103         12 0005960
##      3:  20110110    161500          20         1103          8 0005960
##      4:  20110110    161554          30         1103          0 0005956
##      5:  20110110    161554          40         1103          8 0005956
##     ---
## 625385:  20110110    143448      944770         1107          1 0006192
## 625386:  20110110    145927      944780         1107          5 0006200
## 625387:  20110110    145927      944780         1107          1 0006192
## 625388:  20110110    154751      944790         1107          8 0006200
## 625389:  20110110    154751      944790         1107          1 0006192
##         ASKBID EntryDate  Price    Deferreds
##      1:     NA    110109 595.00       Nearby
##      2:      A    110109 596.00       Nearby
##      3:      B    110109 596.00       Nearby
##      4:     NA    110109 595.75       Nearby
##      5:      A    110109 595.75       Nearby
##     ---
## 625385:      B    110110 619.25 2st Deferred
```

```
## 625386:      A    110110 620.00 2st Deferred
## 625387:      B    110110 619.25 2st Deferred
## 625388:      A    110110 620.00 2st Deferred
## 625389:      B    110110 619.25 2st Deferred
```

Table of Contents

`minutebins()`: Break Trading Day into Ten Minute Bins

Sometimes you want to isolate market effects at specific times of the day. Often, the market open and market close are of particular interest. The function `minutebins(x , minutes)` takes a `data.table` object and the size of the minute bins, `minutes = 10` is the default. It takes the timestamp and rounds to the nearest minute interval allowing you to easily focus on market activity that happens exclusively in specific time minute intervals. For example, suppose you want to analyze the bid-ask spread on the market open. This function easily allows you to subset for a number of minutes directly after the market open. This is accomplished without much computing time because we manipulate the date and time character strings without converting to a date class. Converting to a date class alone takes a lot of time for R to accomplish, so if you can do subset by date or time without converting to a date class, you save a lot of time.

```
DATA[, bins := minutebins(DATA, 10)]
DATA
```

```
##          TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##       1:  20110110    130604          10         1103          0 0005950
##       2:  20110110    161500          20         1103         12 0005960
##       3:  20110110    161500          20         1103          8 0005960
##       4:  20110110    161554          30         1103          0 0005956
##       5:  20110110    161554          40         1103          8 0005956
##      ---
## 1038066:  20110110    105201          80         1412          2 0005200
## 1038067:  20110110    105201          80         1412          1 0005124
## 1038068:  20110110    133000          90         1412          2 0005200
## 1038069:  20110110    133000          90         1412         39 0005120
## 1038070:  20110110    142729         100         1412          0 0005200
##          ASKBID EntryDate  Price    bins
##       1:     NA    110109 595.00  13 : 0
##       2:      A    110109 596.00 16 : 10
##       3:      B    110109 596.00 16 : 10
##       4:     NA    110109 595.75 16 : 10
##       5:      A    110109 595.75 16 : 10
##      ---
## 1038066:      A    110110 520.00 10 : 50
## 1038067:      B    110110 512.50 10 : 50
## 1038068:      A    110110 520.00 13 : 30
## 1038069:      B    110110 512.00 13 : 30
## 1038070:     NA    110110 520.00 14 : 20
```

`datemanip()` and `timemanip()` for Converting to IDate Class Objects.

As mentioned in the discussion of the `minutebins()` function, converting the date and time variables is costly in terms of computing time; however it is often unavoidable at some point in our analysis. Particularly if you want to include any plots of time series analysis, you need the x-axis to be specified as a variable in one of

the time classes. From a computing time perspective, this is best to be done after your analysis is performed and you want to plot some kind of aggregated results. For example, perhaps I want to plot the number of transactions per day. You can first compute the number of transactions per day before converting the date variable to a date object. Now you have something that has only one observation per day. This will be less time consuming to convert to a date object.

```
DATA[, c("TradeDate", "TradeTime") := .(datemanip(TradeDate), timemanip(TradeTime))]
DATA
```

```
##            TradeDate TradeTime TradeSeqNum DeliveryDate TrQuantity TrPrice
##       1: 2011-01-10  13:06:04          10         1103          0 0005950
##       2: 2011-01-10  16:15:00          20         1103         12 0005960
##       3: 2011-01-10  16:15:00          20         1103          8 0005960
##       4: 2011-01-10  16:15:54          30         1103          0 0005956
##       5: 2011-01-10  16:15:54          40         1103          8 0005956
##      ---
## 1038066: 2011-01-10  10:52:01          80         1412          2 0005200
## 1038067: 2011-01-10  10:52:01          80         1412          1 0005124
## 1038068: 2011-01-10  13:30:00          90         1412          2 0005200
## 1038069: 2011-01-10  13:30:00          90         1412         39 0005120
## 1038070: 2011-01-10  14:27:29         100         1412          0 0005200
##          ASKBID EntryDate  Price    bins
##       1:     NA    110109 595.00  13 : 0
##       2:      A    110109 596.00 16 : 10
##       3:      B    110109 596.00 16 : 10
##       4:     NA    110109 595.75 16 : 10
##       5:      A    110109 595.75 16 : 10
##      ---
## 1038066:      A    110110 520.00 10 : 50
## 1038067:      B    110110 512.50 10 : 50
## 1038068:      A    110110 520.00 13 : 30
## 1038069:      B    110110 512.00 13 : 30
## 1038070:     NA    110110 520.00 14 : 20
```

`getTD()` Assigning Trade Direction to Transactions

This function assigns each transaction as buyer initiated (+1) or seller initiated (-1). Since we need to have built the top-of-the-book and observe the bid-ask-spread for the function, a detailed example of `getTD()`'s implementation follows after we build the top-of-the-book below.

Table of Contents

Useful Scripts

In this section we will discuss some tasks that are commonly required for analysis on BBO data including building the top-of-the-book and computing the bid-ask-spread (BAS), looping over daily data files to build a large `data.table` object with many days worth of data, creating a summary plot with price, number of revisions to the bids, revisions to asks, and transactions.

Build Top-of-the-Book and Bid-Ask Spread

A frequent topic of analysis with the BBO data is to examine the bid-ask-spread. The following code creates the BAS by using the `dcast.data.table()` function from the `data.table` package. This function allows

you to take the ASKBID column and create a new column for each factor that exists in the ASKBID column.

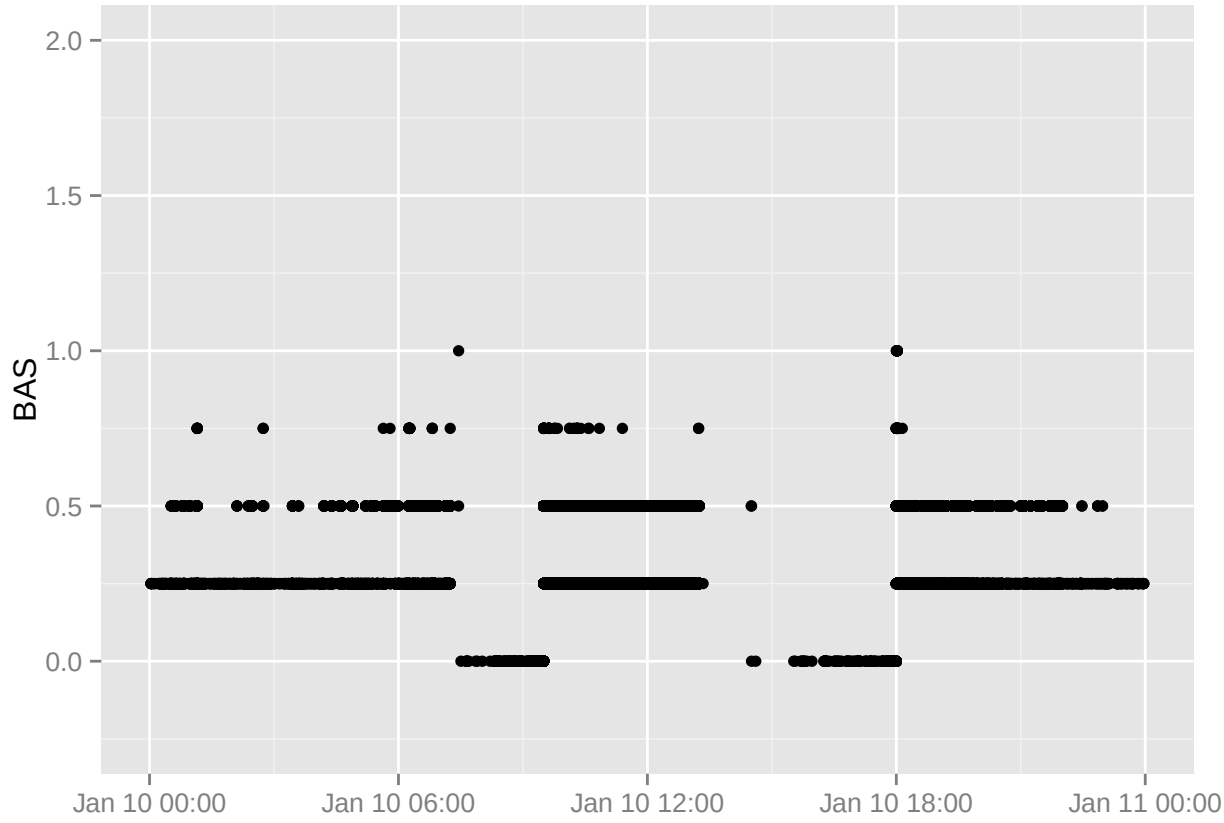
```
library(ggplot2)
DT <- dcast.data.table(DATA, TradeDate + TradeTime + TradeSeqNum + DeliveryDate + EntryDate + bins ~ ASK
                       value.var = c("TrQuantity", "Price"))

DT[, BAS := (Price_A - Price_B)]
DT
```

```
##          TradeDate TradeTime TradeSeqNum DeliveryDate EntryDate    bins
##      1: 2011-01-10  00:02:06       70820         1107    110110  00 : 0
##      2: 2011-01-10  00:02:06      108220         1103    110110  00 : 0
##      3: 2011-01-10  00:02:06      108230         1103    110110  00 : 0
##      4: 2011-01-10  00:02:07       29600         1203    110110  00 : 0
##      5: 2011-01-10  00:02:07       29610         1203    110110  00 : 0
##     ---
## 535881: 2011-01-10  23:57:28       70800         1107    110109 23 : 50
## 535882: 2011-01-10  23:57:28      108190         1103    110109 23 : 50
## 535883: 2011-01-10  23:57:28      108200         1103    110109 23 : 50
## 535884: 2011-01-10  23:58:08       70810         1107    110109 23 : 50
## 535885: 2011-01-10  23:58:08      108210         1103    110109 23 : 50
##         TrQuantity_A TrQuantity_B TrQuantity_NA Price_A Price_B Price_NA
##      1:           44            2            NA  614.75   614.0       NA
##      2:           NA           NA             1      NA      NA      601
##      3:           20            2            NA  601.25   601.0       NA
##      4:           20           11            NA  556.00   554.5       NA
##      5:           21           11            NA  556.00   554.5       NA
##     ---
## 535881:           44            2            NA  614.75   614.0       NA
## 535882:           21            3            NA  601.25   601.0       NA
## 535883:           21            2            NA  601.25   601.0       NA
## 535884:           44            3            NA  614.75   614.0       NA
## 535885:           21            3            NA  601.25   601.0       NA
##          BAS
##      1: 0.75
##      2:   NA
##      3: 0.25
##      4: 1.50
##      5: 1.50
##     ---
## 535881: 0.75
## 535882: 0.25
## 535883: 0.25
## 535884: 0.75
## 535885: 0.25
```

```
BAS_plot <- qplot(DT[DeliveryDate == 1103, as.POSIXct("2011-01-10") + TradeTime], DT[DeliveryDate == 110
                  ylim = c(-0.25, 2), xlab = NULL, ylab = "BAS")

BAS_plot
```



We can see that on this day, January 10th, 2011, the BAS ranged from 0 to 1 cent and spent most of its time at 0.25 cents, which is one tick (minimum allowable price fluctuation) in this market.

This script works for building the top-of-the book prior to 2012, but in 2012 the CME Group changed the format of the BBO data so that it does not record the bid and ask that is unchanged. Let us see what the `DT` looks like if we try to build the top-of-the-book for dates after 2012. Note that you can only follow along with this portion of the vignette if you have access to the proprietary data from CME's Historical Datamine. First load one of the datasets using the `DATA <- as.data.table(bboread('filepath'))` command from earlier in the vignette. For this example I am reading the large dataset in with `readr`'s `read_fwf()` function so I can specify `nmax = 500000` and have a less memory intensive example. Then, similar code is executed as what we did earlier. Notice that now there are many missing values, `NA`, after we cast the data.table.

The `fill()` function from the `tidyr` package takes missing values and fills in the previous value. Since an unchanged bid or ask is what produces the `NA`'s this is exactly what we need to do to build the top of the book. Then we can compute the BAS is a similar way as we did before.

```
rm(DATA)
start <- c(1, 9, 15, 28, 32, 45, 53, 65)
end   <- c(8, 14, 22, 31, 36, 51, 53, 70)

DATA <- readr::read_fwf('C:/Users/mallorym/BBOCORNDATA/2012Jan-2013Nov_txt/BBO_CBT_20120102-20131130_95
                        readr::fwf_positions(start, end, col_names = c("TradeDate", "TradeTime", "Trad
                                                                       "DeliveryDate", "TrQuantity
                                                                       "ASKBID", "EntryDate")),
                         col_types = readr::cols(
                           TradeDate = readr::col_character(),
                           TradeTime = readr::col_character(),
```

```
                            TradeSeqNum = readr::col_integer(),
                            DeliveryDate = readr::col_character(),
                            TrQuantity = readr::col_integer(),
                            TrPrice = readr::col_character(),
                            ASKBID = readr::col_character(),
                            EntryDate = readr::col_character()
                            ## Put this back into function bboread

                          ), n_max = 500000, progress=FALSE)
DATA <- as.data.table(DATA)
DATA[, Price := decimalprices(DATA$TrPrice)]
DATA[, bins := minutebins(DATA, 10)]

DT <- dcast.data.table(DATA,
                       TradeDate + TradeTime + TradeSeqNum + DeliveryDate + EntryDate + bins ~ ASKBID,
                       value.var = c("TrQuantity", "Price"))
rm(DATA)
# data.table syntax does not work properly. tidyr functions must not be callable by data.table objects
# DT[, c("TrQuantity_A", "TrQuantity_B", "Price_A", "Price_B") := .(fill(TrQuantity_A), fill(TqQuantity

# The following is making copies of DT in memory, but I don't know of a solution right now.
DT <- fill(DT, TrQuantity_A:Price_B, -TrQuantity_NA)
DT[, BAS := (Price_A - Price_B)]
```

```
## Warning in `[.data.table`(DT, , `:=`(BAS, (Price_A - Price_B))):
## Invalid .internal.selfref detected and fixed by taking a (shallow) copy
## of the data.table so that := can add this new column by reference. At
## an earlier point, this data.table has been copied by R (or been created
## manually using structure() or similar). Avoid key<-, names<- and attr<-
## which in R currently (and oddly) may copy the whole data.table. Use set*
## syntax instead to avoid copying: ?set, ?setnames and ?setattr. Also, in
## R<=v3.0.2, list(DT1,DT2) copied the entire DT1 and DT2 (R's list() used to
## copy named objects); please upgrade to R>v3.0.2 if that is biting. If this
## message doesn't help, please report to datatable-help so the root cause can
## be fixed.
```

Get Trade Direction

For many microsturcture models, you need to know the trade direction of each transaction. That is, whether a transaction was buyer initiated or seller initiated. The simplest case is that if you observe a transaction that occurs with a price exactly equal to the ask, you assume it was initiated by a market order to buy, and if you observe a transaction that occurs with a price exactly equal to the bid, you assume it was initiated by a market order to sell. Sometimes when the spread is wider than one tick, you observe trades that occur inside the spread and the data gives no indication about how the transaction inside the spread came about. Was is two market orders that god matched and given a random price inside the spread? Are the transactions and quote revisions not 100% in order? It is impossible to tell. But this issue comes up frequently enough that there are several algorithms whose job it is to sign a trade direction to each transaction. Lee and Ready (1991) developed an early algorithm that is still widely used, although there have been criticisms of this algorithm (Panayides, Shohfi, and Smith 2014), and other competing algorithms have been developed (Easley, Prado, and O'Hara 2016).

Currently, the function `getTD()` only supports signing trade directions according to the Lee and Ready algorithm, but I think the research would benefit from developing the program to implement some of the

newer algorithms. Support for such will come sometime in the future.

Below is an example in our context of how to run the `getTD()` function to sign trades according to the Lee and Ready algorithm.

```
# rm(DATA) # Start fresh
# rm(DT)
DATA   <- as.data.table(bboread('XCBT_C_FUT_110110.TXT'))
DATA[, Price := decimalprices(DATA$TrPrice)]
DATA[, bins := minutebins(DATA, 10)]
DT     <- dcast.data.table(DATA, TradeDate + TradeTime + TradeSeqNum + DeliveryDate + EntryDate + bins
                           value.var = c("TrQuantity", "Price"))
DT <- fill(DT, TrQuantity_A:Price_B, -TrQuantity_NA)
DT[, BAS := (Price_A - Price_B)]
```

```
## Warning in `[.data.table`(DT, , `:=`(BAS, (Price_A - Price_B))):
## Invalid .internal.selfref detected and fixed by taking a (shallow) copy
## of the data.table so that := can add this new column by reference. At
## an earlier point, this data.table has been copied by R (or been created
## manually using structure() or similar). Avoid key<-, names<- and attr<-
## which in R currently (and oddly) may copy the whole data.table. Use set*
## syntax instead to avoid copying: ?set, ?setnames and ?setattr. Also, in
## R<=v3.0.2, list(DT1,DT2) copied the entire DT1 and DT2 (R's list() used to
## copy named objects); please upgrade to R>v3.0.2 if that is biting. If this
## message doesn't help, please report to datatable-help so the root cause can
## be fixed.
```

```
tqdata <- DT[!is.na(Price_NA)  ]
tqdata <- tqdata[, c("diff1", "diff2") := .(diff(Price_NA, lag = 1), diff(Price_NA, lag = 2))]
```

```
## Warning in `[.data.table`(tqdata, , `:=`(c("diff1", "diff2"), .
## (diff(Price_NA, : Supplied 31544 items to be assigned to 31545 items of
## column 'diff1' (recycled leaving remainder of 1 items).
```

```
## Warning in `[.data.table`(tqdata, , `:=`(c("diff1", "diff2"), .
## (diff(Price_NA, : Supplied 31543 items to be assigned to 31545 items of
## column 'diff2' (recycled leaving remainder of 2 items).
```

```
tqdata[, td := getTD(tqdata)]
tqdata
```

```
##        TradeDate TradeTime TradeSeqNum DeliveryDate EntryDate    bins
##     1:  20110110    000206      108220         1103    110110  00 : 0
##     2:  20110110    000207       89300         1105    110110  00 : 0
##     3:  20110110    000207      108240         1103    110110  00 : 0
##     4:  20110110    000401      108300         1103    110110  00 : 0
##     5:  20110110    001140       28380         1109    110110 00 : 10
##    ---
## 31541:  20110110    233932       32200         1112    110109 23 : 30
## 31542:  20110110    234011      107970         1103    110109 23 : 40
## 31543:  20110110    234337      108060         1103    110109 23 : 40
## 31544:  20110110    235104      108110         1103    110109 23 : 50
```

```
## 31545:  20110110    235643      108160         1103    110109 23 : 50
##        TrQuantity_A TrQuantity_B TrQuantity_NA Price_A Price_B Price_NA
##     1:           44            2             1  614.75  614.00   601.00
##     2:           44            4             1  614.75  614.00   610.00
##     3:           21            3             9  610.00  609.75   601.25
##     4:           21           12             1  610.00  609.75   601.00
##     5:            3           12             1  601.25  601.00   575.00
##    ---
## 31541:           10           14             5  556.00  554.50   546.75
## 31542:           22            3             1  610.00  609.75   601.00
## 31543:           42            5             2  614.75  614.00   601.25
## 31544:           22            2             1  610.00  609.75   601.00
## 31545:           23            2             1  601.25  601.00   601.25
##         BAS  diff1  diff2 td
##     1: 0.75   9.00   0.25 -1
##     2: 0.75  -8.75  -9.00 -1
##     3: 0.25  -0.25 -26.25 -1
##     4: 0.25 -26.00   0.00 -1
##     5: 0.25  26.00  26.00 -1
##    ---
## 31541: 1.50  54.25  54.50 -1
## 31542: 0.25   0.25   0.00 -1
## 31543: 0.75  -0.25   0.00 -1
## 31544: 0.25   0.25   0.25 -1
## 31545: 0.25   9.00  -9.00  1
```

References

Easley, David, Marcos López de Prado, and Maureen O'Hara. 2016. "Discerning Information from Trade Data." *Journal of Financial Economics*. Elsevier.

Lee, Charles, and Mark J Ready. 1991. "Inferring Trade Direction from Intraday Data." *The Journal of Finance* 46 (2). Wiley Online Library: 733–46.

Panayides, Marios A, Thomas Shohfi, and Jared D Smith. 2014. "Comparing Trade Flow Classification Algorithms in the Electronic Era: The Good, the Bad, and the Uninformative." *Available at SSRN 2503628*.