Amihud (2002) Replication

Chase DeHan

August 6, 2016

Replication of: Amihud, Yakov (2002) "Illiquidity and Stock Returns: Cross-Section and Time-Series Effects" Journal of Financial Markets

This replication is of an influential paper hypothesizing that excess stock returns partly represent an illiquidity premium set for publication in Critical Finance Review. Contained below is the code as an appendix which will be made freely available at Finance Recipes. You are free to use and modify any of the code found below in whole or in part. In fact, you are encouraged to take this code and extend it into something new and insightful. You are under no obligation to cite the code that you do use, although it would be appreciated.

Notes Before Starting

While the code below confirms the results found by Amihud (2002), It is possible that the code below does contain an error. If you spot an error, please send me an email letting me know what you see and possibly some ways to make it better.

You will notice that the replication is relatively long; most of this length is the explanation behind how the code works and displaying the data as we progress. You will see that I show what the data looks like after almost every step with the head() function, which shows the first six observations of a dataframe. In this way, you are able to see how particular lines of code change the data. One of the other reasons for the length is that the code was written in a way that should be more intuitive for those learning. As with all code writing, there is a tradeoff between efficient code writing (defined as less typing) versus readability.

As far as coding ability is concerned, R is a beautiful language, but can be challenging to learn. There are a ton of other resources out there to get started with using R. Some that helped me immensely were Norman Matloff's Art of R Programming and the DplyR package. Data Camp is GREAT and you should definitely check it out; they also have a student/faculty discount for $9 per month.

The organization of this document is as follows:

1. Short high-level description of what we are accomplishing
2. This description expanded to include the code chunks

So Let's Get Started!

Section 2. Cross-section relationship between illiquidity and stock return

Step 1: Estimating the illiquidity variable

1. Construct $ILLIQ_{iy}$, eq (1)

The construction of the illiquidity measure $ILLIQ_{iy}$, defined as:

$$ILLIQ_{iy} = 1/D_{iy} \sum_{t=1}^{D_{iy}} |R_{iyd}|/VOLD_{ivyd} \qquad (1)$$

This measure is the illiquidity measure for each stock i for year y based off daily data d.

2. Construct AILLIQ_y, eq (3)

We then average the ILLIQ_iy figures across all companies for each year as follows:

$$AILLIQ_y = 1/N_y \sum_{t=1}^{N_y} ILLIQ_{iy} \qquad (3)$$

Which gives us an annual illiquidity value.

3. Construct ILLIQMA_iy, eq (4)

The next step is to construct the mean-adjusted value for use in the estimations. We take the ILLIQ_iy and adjust it to the average annual value AILLIQ_y as follows:

$$ILLIQMA_{iy} = ILLIQ_{iy}/AILLIQ_y \qquad (4)$$

Step 2: Create additional variables

This step involves a little bit of coding work to get the following values into the dataframe.
###Risk Variables
$BETA_{py}$ = Computed on size based portfolios? * $SDRET_{iy}$ = standard deviation of daily returns * $Ln(SIZE)$ is described as instrumenting the Beta, but I'm not sure how it is being handled.

Additional Variables

- $DIVYLD_{iy}$ = sum of dividends during year y divided by end-of-year price
- $R100_{iy}$ = return on stock i during last 100 days of year y
- $R100YR_{iy}$ = return on stock i from beginning of year and 100 days before end.

Step 4: Filter out stocks that don't meet criteria

There are a number of conditions that must hold in order for a stock to remain in the dataframe for the empiric portion. The way I interpreted these restrictions was to perform all the data manipulations above before removal. The actual order here matters because if we remove a stock at (t-1) because it does not meet the criteria, it will necessarily mean that the stock will also be removed at time t because we rely on observations from (t-1) in the empirical section. Not having (t-1) means that the lagged values will turn up as an NA, which then gets removed. Therefore, we are removing these observations in this step. If this is incorrect, changing it to come before the data manipulations is easy as we can just copy/paste it in the appropriate spot.

In order to remain in the dataframe, a stock must meet the following characteristics:

- Return data for more than 200 days during the previous year
- Stock price greater than \$5 or less than \$1000 at the end of the previous year
- Has a market capitalization greater than \$0 at the end of the previous year

Section 3. The effect over time of market illiquidity on expected stock excess return

Does illiqudity impact returns?

The idea in this section is that expected market illiquidity positively impacts exected excess return. Basically, if investors expect there to be future illiquidity, it will be priced into the stocks at the current time, thus generating that future excess return.

"Investors are assumed to predict illiquidity for year y based on information available in year y-1, and then use this prediction to set prices that will generate the desired expected return in year y." (p. 43). It is assumed that expected market illiquidity follows the AR(1) model of:

$$lnAILLIQ_y = c_0 + c_1 lnAILLIQ_{y-1} + u_y \qquad (7)$$

where c_1 is expected to be positive and the residual u_y containing the unexpected portion of liquidity.

The two hypotheses are:
1. ex ante stock excess return is an increasing function of expected liquidity. - below: $g_1 > 0$
2. unexpected illiquidity has a negative effect on contemporaneous unexpected stock return. - below: $g_2 < 0$

g_1 and g_2 are estimated from the following equation:

$$(RM - Rf)_y = g_0 + g_1 lnAILLIQ_{y-1} + g_2 lnAILLIQ_y^U + w_y, \qquad (10)$$

where $lnAILLIQ_y^U$ is the unexpected illiquidity from year y, and $lnAILLIQ_y^U$ is the unexpected illiquidity as the residual from eq (7).

section 3.2: Illiquidity and excess returns on size portfolios

They hypothesize that, yes, with increased illiquidity, there is a decline in stock prices. More interesting, I think, is that when there is a market wide flight to liquidity, the price impact on the relatively more liquid stocks is lesser as investors dump the relatively less liquid.
The "flight to liquidity" is going to be highly correlated with the size of the company. So, to test for this impact the above specification is tested using five different size portfolios. We estimate the results in this section from equation 10:

$$(RM - Rf)_y = g_0 + g_1 lnAILLIQ_{y-1} + g_2 lnAILLIQ_y^U + w_y, \qquad (10)$$

section 3.3: Monthly data - Illiquidity and Stock Prices

Compute an autoregressive model on monthly data similar to what we did above

$$lnMILLIQ_m = c_0 + c_1 lnMILLIQ_{m-1} + u_y$$

Where MILLIQ is the average monthly illiquidity versus the annual data of AILLIQ.

Because of the use of monthly data in this section, it is necessary to control for the known "January Effect" by including a January Dummy. This estimation is similar to eq. (10), specified as:

$$(RM - Rf)_m = g_0 + g_1 lnMILLIQ_{m-1} + g_2 lnMILLIQ_m^U + g_3 JANDUM_m + w_y, \qquad (10m)$$

Section 3.4: Controlling for the effects of bond yield premiums

In this section we perform essentially the same estimations as we did in Section 3.3 with the one exception being that we control for the default yield premium and the term yield premium.

The default yield premium is defined as the difference in yield between long-term BAA-rated and AAA-rated bonds,

$$DEF_m = YBAA_m - YAAA_m$$

The term yield premium is defined as the difference in yield between long-term Treasuries and three-month T-Bills,

$$TERM_m = YLONG_m - YTB3_m$$

Once we have calculated the values for $\mathrm{DEF_m}$ and $\mathrm{TERM_m}$, we can empirically test it as:

$$(RM-Rf)_m = g_0+g_1 lnMILLIQ_{m-1}+g_2 lnMILLIQ_m^U+g_3 JANDUM_m+a_1 DEF_{m-1}+a_2 TERM_{m-1}+u_y, \quad (11)$$

In this section we also compute the effects of illiquidity on the size portfolios like we did in Sections 3.2 and 3.3.

Writing the code to implement this:

Before we get started, there is a lot of code written using the R package "dplyr" and if you have never seen it before the code does look a little confusing. I promise that once you get the hang of it, the power is incredible - I don't know how I got anything done before I found it. Therefore, if you are unfamiliar or need a refresher I recommend you check out the official introduction in CRAN.

Step 1: Estimating the illiquidity variable

Load the necessary packages

We have to load the packages into R that will allow us to use the appropriate functions. The wide availability of packages for nearly every programming task is one of the great benefits to the use of R. These packages are constantly evolving and new packages appear every day. When attempting to do something new, there is a good chance that someone else has done the same thing and they will often wrap their code into a package for you to use. A web search is helpful for pulling up where others have done similar things.

```
library(dplyr)
library(tidyr)
library(lubridate)
library(data.table)
library(Quandl)
library(PerformanceAnalytics)
library(lmtest)
library(sandwich)
library(plm)
```

So why do we need these packages in particular?

- dplyr
  - The most amazing package that allows us to manipulate data very easily
  - Used extensively for this replication as it makes certain tasks incredibly easy
- tidyr
  - From the same developer as dplyr - allows for making "tidy data"
  - Used only in limited amounts here, most CRSP data is downloaded as "tidy"
- lubridate
  - A date class that allows for easy movements of dates
  - Dates are really important in financial data and this package allows us to make sure we are merging the appropriate dates/times.
- data.table
  - Used for similar purposes to dplyr, but code is not as intuitive. However, it is much faster because it is written in C++ behind the scenes.
  - We use it here because it is able to upload data with fread() much faster than other methods.
- Quandl
  - A really neat data API that allows us to download a multitude of data feeds directly in R
- PerformanceAnalytics
  - Use the skewness function for Table 1
- lmtest
  - short for "linear model test", provides many diagnostic tools for the built in lm() function
  - use for Durbin-Watson and testing estimated coefficients - dwtest() and coeftest() respectively
- sandwich
  - Used to estimate alternative standard errors tests
  - Newey-West and White's Heteroskedastic-consistent standard errors - NeweyWest() and vcovHC() respectively
- plm
  - For the Fama-Macbeth fixed effects regressions

Input the data

We have to input the data into our workspace that was downloaded from CRSP/Compustat. I like to input data with the fread() function out of the data.table library. There are a ton of other ways we can input the data, but I like this function because it is substantially faster than any others; notice how long it takes to upload a file thats nearly 1GB. names() is changing the names of the variables because the way I pull the data from CRSP does not insert a name into the first two variables. The ymd() function converts the date variable into a different class from the lubridate library, which makes working with dates a lot easier.

```
dailyData <- fread("Amihud_2002_Daily.txt")
```

```
##
Read 9.6% of 14117207 rows
Read 27.3% of 14117207 rows
Read 44.9% of 14117207 rows
Read 62.6% of 14117207 rows
Read 80.2% of 14117207 rows
Read 97.8% of 14117207 rows
Read 14117207 rows and 6 (of 6) columns from 0.933 GB file in 00:00:08
```

```r
names(dailyData)[1:2]<- c("PERMNO", "Date") #renaming columns
head(dailyData)
```

```
##    PERMNO     Date   Tprc    Totret     TCap Tvol
## 1:  10006 19640102 61.250 -0.012097 180565.0 1100
## 2:  10006 19640103 61.750  0.008163 182039.0 1100
## 3:  10006 19640106 62.500  0.012146 184250.0 3400
## 4:  10006 19640107 63.875  0.022000 188303.5 4600
## 5:  10006 19640108 64.750  0.013699 190883.0 4200
## 6:  10006 19640109 65.000  0.003861 191620.0 3100
```

```r
dailyData$Date <- ymd(dailyData$Date) #Converting the dates to lubridate class

#Importing Monthly Returns for same time period/PERMNO
monthlyData <- fread("Amihud_2002_Monthly.txt")
names(monthlyData)[1:2] <- c("PERMNO", "Date")
head(monthlyData)
```

```
##    PERMNO     Date       Ret
## 1:  10006 19640131  0.068548
## 2:  10006 19640228  0.043774
## 3:  10006 19640331  0.000000
## 4:  10006 19640430  0.003636
## 5:  10006 19640528 -0.010870
## 6:  10006 19640630  0.075646
```

```r
monthlyData$Date <- ymd(monthlyData$Date)

#Importing annual data for the same time period/PERMNOs
annualData <- fread("Amihud_2002_Annual.txt")
names(annualData)[1:2] <- c("PERMNO", "Date")
head(annualData)
```

```
##    PERMNO     Date   Tprc     TCap TDivamt
## 1:  10006 19631231 62.000 182776.0   2.825
## 2:  10006 19641231 79.500 235956.0   2.250
## 3:  10006 19651231 47.250 279720.0   3.525
## 4:  10006 19661230 38.250 219861.0   2.100
## 5:  10006 19671229 44.875 253229.6   2.200
## 6:  10006 19681231 62.625 353267.6   2.250
```

```r
annualData$Date <- ymd(annualData$Date)
```

1. Construct $\mathbf{ILLIQ_{iy}}$, eq (1)

The construction of the illiquidity measure ILLIQ_{iy}, defined as:

$$ILLIQ_{iy} = 1/D_{iy} \sum_{t=1}^{D_{iy}} |R_{iyd}|/VOLD_{ivyd} \qquad (1)$$

This measure is the illiquidity measure for each stock i for year y based off daily data d.

The first step in creating ILLIQ_{iy} is to eliminate the observations in CRSP. We are using the total return (Totret) and CRSP frustratingly encodes -66,-77,-88, and -99 into the returns. We therefore have to filter out the observations equal to that. Using mutate() we can create the necessary variables in eq (1). We are also creating the Year so that we can group the variables by year in later sections. We assign this to a new dataframe called "daily".

```
daily <- dailyData %>%
  filter(Totret != -66, Totret != -77, Totret != -88, Totret != -99) %>%
  filter(Tvol != -99) %>%
  mutate(Year = year(Date),
         VOLD = Tprc*Tvol,
         Riyd = abs(Totret),
         ILLIQiyd = Riyd / VOLD) %>%
  select(-Tvol) %>%  #removes the Tvol variable
  na.omit()
head(daily)
```

```
##    PERMNO       Date   Tprc    Totret     TCap Year   VOLD     Riyd
## 1:  10006 1964-01-02 61.250 -0.012097 180565.0 1964  67375 0.012097
## 2:  10006 1964-01-03 61.750  0.008163 182039.0 1964  67925 0.008163
## 3:  10006 1964-01-06 62.500  0.012146 184250.0 1964 212500 0.012146
## 4:  10006 1964-01-07 63.875  0.022000 188303.5 1964 293825 0.022000
## 5:  10006 1964-01-08 64.750  0.013699 190883.0 1964 271950 0.013699
## 6:  10006 1964-01-09 65.000  0.003861 191620.0 1964 201500 0.003861
##        ILLIQiyd
## 1: 1.795473e-07
## 2: 1.201767e-07
## 3: 5.715765e-08
## 4: 7.487450e-08
## 5: 5.037323e-08
## 6: 1.916129e-08
```

Now that we have the components of ILLIQ_{iy}, we can calculate the mean value for each stock, for each each year. We do this first using group_by() which breaks into these annual groups. Then, summarise() computes the mean, standard deviation(sd()), and number of observations (n()).

```
illiqTable <- daily %>%
  group_by(PERMNO, Year) %>%
  summarise(ILLIQiy = mean(ILLIQiyd) * 1000000, #And multiplied by 10^6
            SDRETiy = sd(Totret) * 100, #And multiplied by 10^2
            count = n()) #Counting the N for dropping in the later section
head(illiqTable)
```

```
##   PERMNO Year    ILLIQiy  SDRETiy count
## 1  10006 1964 0.06519902 1.371656   253
## 2  10006 1965 0.04204054 1.329256   252
## 3  10006 1966 0.06042057 1.847645   252
## 4  10006 1967 0.05512413 1.617939   251
## 5  10006 1968 0.03325193 2.596928   226
## 6  10006 1969 0.05805737 1.594501   250
```

Now that we have ILLIQ_{iy}, assigned to "illiqTable", we can move on to the next section.

2. Construct AILLIQ_y, eq (3)

We then average the ILLIQ_iy figures across all companies for each year as follows:

$$AILLIQ_y = 1/N_y \sum_{t=1}^{N_y} ILLIQ_{iy} \qquad (3)$$

Which gives us an annual illiquidity value. The code is relatively straightforward in that we group everything by year (group_by()) and then compute the mean value inside summarise() with mean(). This is then assigned to the "ailliqTable" dataframe as follows.

```
ailliqTable <- illiqTable %>%
  group_by(Year) %>%
  summarise(AILLIQy = mean(ILLIQiy))
head(ailliqTable)
```

```
##   Year   AILLIQy
## 1 1964 0.7938818
## 2 1965 0.6086327
## 3 1966 0.6188801
## 4 1967 0.3520700
## 5 1968 0.2171579
## 6 1969 0.3078331
```

3. Construct ILLIQMA_iy, eq (4)

The next step is to construct the mean-adjusted value for use in the estimations. We take the ILLIQ_iy and adjust it to the average annual value AILLIQ_y as follows:

$$ILLIQMA_{iy} = ILLIQ_{iy}/AILLIQ_y \qquad (4)$$

This is a simple calculation to take one variable and divide by another. However, astute readers will notice that we created two dataframes above, "illiqTable" and "ailliqTable", which we have to merge together to get the two variables to match up for eq (4). We do this below by merging these two dataframes together and using the pipe operator (%>%) we then use mutate() to create the new variable ILLIQMA_iy for each stock for each year.

```
illiqmaTable <- merge(illiqTable, ailliqTable, by="Year") %>%
  mutate(ILLIQMAiy = ILLIQiy / AILLIQy) #1964 - 1997
head(illiqmaTable)
```

```
##   Year PERMNO    ILLIQiy   SDRETiy count   AILLIQy  ILLIQMAiy
## 1 1964  10006 0.06519902 1.3716559   253 0.7938818 0.08212686
## 2 1964  10014 7.34452952 3.4626001   253 0.7938818 9.25141411
## 3 1964  10030 0.14499817 0.8731568   253 0.7938818 0.18264453
## 4 1964  10057 0.32441210 1.1739814   241 0.7938818 0.40864029
## 5 1964  10065 0.29378034 0.8887408   253 0.7938818 0.37005550
## 6 1964  10102 0.03060658 0.9711845   253 0.7938818 0.03855306
```

Memory Management

At this point it is probably relevant to acknowledge the issues with memory (as in computer RAM) and R. R holds everything being worked on in memory and with the daily stock data being a few GB of data, this is a constraint on many older home computers in that some commands may take a long time or crash the computer. As such, when we are done with a dataframe I have gotten in the habit of clearing it out of memory with rm(). You include all the objects in your workspace that you wish to remove, separated by commas.

```
rm(illiqTable, dailyData)
```

Step 2: Create the risk variables and additional variables

In computing BETA and the additional variables it makes sense to combine these into one step to limit the repetitive code. "DRY" if you aren't familiar

Bringing in market data

The first step in these calculations is to bring in the appropriate data. I will be using the market return from Ken French's collection downloaded through the Quandl package in R. This makes things so much easier in that we don't have to go to the website, find the data we want, download the data, import it, then start to use it. We can download the Fama-French daily data and have it ready to start using it with one line of code:

```
marketData <- Quandl("KFRENCH/FACTORS_D", start_date = "1964-01-01", end_date = "1997-12-31")
head(marketData)
```

```
##         Date Mkt-RF   SMB   HML    RF
## 1 1997-12-31   0.21  0.88 -0.30 0.022
## 2 1997-12-30   1.76 -0.17 -0.64 0.022
## 3 1997-12-29   1.60 -0.77 -0.47 0.022
## 4 1997-12-26   0.33 -0.27 -0.05 0.022
## 5 1997-12-24  -0.58  0.42  0.31 0.022
## 6 1997-12-23  -1.22  1.07  0.86 0.022
```

After the data is downloaded, we compute the market return (RMty) through mutate() and convert the Date to lubridate class. Dplyr also provides select() to select only the columns/variables we want.

```
marketData <- marketData %>%
  mutate(RMty = (`Mkt-RF` + RF)/100,  #Adding the risk-free rate back in
         Date = ymd(Date)) %>% #Converting to lubridate class
  select(Date, RMty)
head(marketData)
```

```
##         Date     RMty
## 1 1997-12-31  0.00232
## 2 1997-12-30  0.01782
## 3 1997-12-29  0.01622
## 4 1997-12-26  0.00352
## 5 1997-12-24 -0.00558
## 6 1997-12-23 -0.01198
```

Creating dfBeta

We are going to be creating a number of new variables and do not want to be modifying "daily" (or working with the full dataset's complete variables because of memory management). dfBeta is created to mitigate this issue by restricting it to only the variables we need in this section using select().

```
dfBeta <- select(daily, PERMNO, Date, Year, Riyd)
head(dfBeta)
```

```
##    PERMNO       Date Year     Riyd
## 1:  10006 1964-01-02 1964 0.012097
## 2:  10006 1964-01-03 1964 0.008163
## 3:  10006 1964-01-06 1964 0.012146
## 4:  10006 1964-01-07 1964 0.022000
## 5:  10006 1964-01-08 1964 0.013699
## 6:  10006 1964-01-09 1964 0.003861
```

R100

We create the variables a little out of order from the paper in that we are going to compute $\text{R100YR}_{\text{iy}}$ and R100_{iy} first. This calculation was a little challenging in that the 275th day of the year was not guaranteed to fall on a trading day and it was questioned whether rolling back or forward made more sense. The way I am calculating these returns was to find the observation that fell on trading day (275/365), extract the price from that day, and then compute the returns from that point. We do this through creating a new dataframe "R100df":

```
R100df <- daily %>%
  group_by(Year, PERMNO) %>%
  arrange(Date) %>%
  summarise(open = first(Tprc),
            close = last(Tprc),
            P100 = nth(Tprc, floor((265/365)*length(Tprc))) ) %>%
  mutate(R100iy = (close-P100)/P100,
         R100YRiy = (P100-open)/open ) %>%
  select(Year, PERMNO, R100iy, R100YRiy)%>%
  na.omit()
head(R100df)
```

```
##   Year PERMNO       R100iy    R100YRiy
## 1 1964  10006 -0.026033691  0.33265306
## 2 1964  10014 -0.047619048 -0.12500000
## 3 1964  10030  0.008492569  0.16009852
## 4 1964  10057 -0.067567568  0.24369748
## 5 1964  10065  0.004464286  0.05164319
## 6 1964  10102 -0.031460674 -0.02197802
```

What we are doing is:

1. Grouping by stock and year
2. Ordering the stocks by date using arrange() because we need the appropriate dates/prices to compute R100_{iy}

3. We grab the first/lase observation (opening/closing price of the year) using first() and last() - the functions do exactly as the names suggest.
4. P100 looks complicated, but is grabbing the 265/365th observation in the time series.
    - nth() is a neat dplyr function that takes as arguments a vector of data and N, returning the Nth value of that vector
    - "floor((265/365)*length(Tprc))" finds the length of the price vector, multiplying that length by 265/365
    - P100 is assigned the price from the proportional price in the vector
5. R100iy and R100YRiy returns can now be computed using mutate()
6. We keep only the variables we need (select()) and remove any NA values (na.omit())

With having the R100 returns we need to merge it back into the dfBeta dataframe we created for the additional variables, ensuring that the appropriate returns are put back into place for each stock. This is done with the command "by=c("Year","PERMNO")" and will have the same values for each PERMNO for the entire year.

```
dfBeta<- merge(dfBeta, R100df, by=c("Year", "PERMNO"))
head(dfBeta)
```

```
##   Year PERMNO       Date     Riyd      R100iy  R100YRiy
## 1 1964  10006 1964-01-02 0.012097 -0.02603369 0.3326531
## 2 1964  10006 1964-01-03 0.008163 -0.02603369 0.3326531
## 3 1964  10006 1964-01-06 0.012146 -0.02603369 0.3326531
## 4 1964  10006 1964-01-07 0.022000 -0.02603369 0.3326531
## 5 1964  10006 1964-01-08 0.013699 -0.02603369 0.3326531
## 6 1964  10006 1964-01-09 0.003861 -0.02603369 0.3326531
```

```
rm(R100df) #remove from memory
```

Compute Dividend Yield $\mathbf{DIVYLD_{iy}}$

With the annual data that we downloaded from CRSP and read into R on the second step, we can compute the dividend yield (DIVYLDiy) and assign to annualAddDF. This simple calculation takes the total dividend amount over the year and divides by the last price recorded on CRSP in each particular year.

```
annualAddDF <- annualData %>%
  mutate(Year = year(Date),
         DIVYLDiy = (TDivamt / Tprc)*100 ) %>%
  select(PERMNO, Year, TCap, DIVYLDiy, Tprc)
head(annualAddDF)
```

```
##    PERMNO Year     TCap DIVYLDiy   Tprc
## 1:  10006 1963 182776.0 4.556452 62.000
## 2:  10006 1964 235956.0 2.830189 79.500
## 3:  10006 1965 279720.0 7.460317 47.250
## 4:  10006 1966 219861.0 5.490196 38.250
## 5:  10006 1967 253229.6 4.902507 44.875
## 6:  10006 1968 353267.6 3.592814 62.625
```

Just like we did with "R100df", we need to merge this DIVYLDiy back into "dfBeta", and purge "annualAddDF" from memory.

```
dfBeta <- merge(dfBeta, annualAddDF, by = c("PERMNO", "Year"))
head(dfBeta)
```

```
##    PERMNO Year       Date     Riyd      R100iy  R100YRiy   TCap DIVYLDiy
## 1:  10006 1964 1964-01-02 0.012097 -0.02603369 0.3326531 235956 2.830189
## 2:  10006 1964 1964-01-03 0.008163 -0.02603369 0.3326531 235956 2.830189
## 3:  10006 1964 1964-01-06 0.012146 -0.02603369 0.3326531 235956 2.830189
## 4:  10006 1964 1964-01-07 0.022000 -0.02603369 0.3326531 235956 2.830189
## 5:  10006 1964 1964-01-08 0.013699 -0.02603369 0.3326531 235956 2.830189
## 6:  10006 1964 1964-01-09 0.003861 -0.02603369 0.3326531 235956 2.830189
##    Tprc
## 1: 79.5
## 2: 79.5
## 3: 79.5
## 4: 79.5
## 5: 79.5
## 6: 79.5
```

```
rm(annualAddDF) #remove from memory
```

Computing Beta - Breaking into Portfolios

The Beta calculations in Amihud (2002) are computed by breaking the individual stocks into ten size portfolios and then using the portfolio Beta as a proxy for each stock contained in the portfolio. So the first step is to assign each company to the appropriate decile according to market capitalization (TCap) for each year. This is actually pretty easy in that we group_by() the year and then assign the appropriate decile to a new variable called "Decile" with the ntile() function.

```
dfBeta <- dfBeta %>%
  group_by(Year) %>%
  mutate(Decile=ntile(TCap, 10))
head(dfBeta)
```

```
##   PERMNO Year       Date     Riyd      R100iy  R100YRiy   TCap DIVYLDiy
## 1  10006 1964 1964-01-02 0.012097 -0.02603369 0.3326531 235956 2.830189
## 2  10006 1964 1964-01-03 0.008163 -0.02603369 0.3326531 235956 2.830189
## 3  10006 1964 1964-01-06 0.012146 -0.02603369 0.3326531 235956 2.830189
## 4  10006 1964 1964-01-07 0.022000 -0.02603369 0.3326531 235956 2.830189
## 5  10006 1964 1964-01-08 0.013699 -0.02603369 0.3326531 235956 2.830189
## 6  10006 1964 1964-01-09 0.003861 -0.02603369 0.3326531 235956 2.830189
##   Tprc Decile
## 1 79.5      8
## 2 79.5      8
## 3 79.5      8
## 4 79.5      8
## 5 79.5      8
## 6 79.5      8
```

ntile() is another of the revolutionary dplyr functions that takes as a first observation a vector of values (this case is TCap), with the second argument the number of groups (this case is 10 for the deciles). We could change the second argument to 4 if we wanted quartiles, 100 for percentiles, or any other number we wished

to divide into. This straightforward portfolio assignment is lightyears ahead of other approaches that have been used in the past; anyone who has had to do these calculations otherwise should be able to immediately identify the time-saving aspects.

Computing Beta

Since we have grouped the stocks each into portfolios, we have to compute the equal-weighted return by day for each stock in the portfolio. So, we will group each portfolio (by day and portfolio) and compute the mean value of the daily returns as follows:

```
dfRpty <- dfBeta %>%
  group_by(Date, Decile) %>%
  summarise(Rpty = mean(Riyd))
head(dfRpty)
```

```
##         Date Decile        Rpty
## 1 1964-01-02      8 0.011925974
## 2 1964-01-03      8 0.008335897
## 3 1964-01-06      8 0.010245293
## 4 1964-01-07      8 0.009652474
## 5 1964-01-08      8 0.008971202
## 6 1964-01-09      8 0.007883500
```

Then we merge the new values in this dataframe, "dfRpty", with the market data calculated above in "marketData" for each day in the time-series.

```
dfRpty <- merge(dfRpty, marketData, by = "Date")  %>%
  mutate(Year = year(Date))
head(dfRpty)
```

```
##          Date Decile       Rpty    RMty Year
## 1: 1964-01-02      8 0.01192597 0.00613 1964
## 2: 1964-01-02      1 0.02342089 0.00613 1964
## 3: 1964-01-02      6 0.01414477 0.00613 1964
## 4: 1964-01-02      9 0.01100236 0.00613 1964
## 5: 1964-01-02     10 0.01006818 0.00613 1964
## 6: 1964-01-02      7 0.01235712 0.00613 1964
```

The actual beta calculations follow. The process is to take each portfolio for each year, obtaining the coefficient attached to the market return as follows:

$$R_{pty} = a_{py} + BETA_{py} * RM_{ty} + e_{pty} \qquad (5)$$

This process could be extremely complicated because we have 10 portfolios x 34 years of data = 340 regressions of daily return data; dplyr makes this easy.

1. We group the regressions (group_by()) by the portfolio(Decile) and Year.

2. We create a new variable called "BETApy" and assign to it the BETA coefficient assigned through a linear model estimation.

3. The regression is estimated by "lm(Rpty ~ RMty)"
4. Extract the coefficients with "$coefficients$" + $We take the second coefficient$"coefficients[2]" because the first coefficient is the intercept
    - If unfamiliar with how this works see TODO (discussion of object attributes)
5. Assign this BETA coefficient to every observation in the group.

```
dfRpty <- dfRpty %>%
  group_by(Decile, Year) %>%
  mutate(BETApy = lm(Rpty ~ RMty)$coefficients[2])
```

Summarising the BETA estimates

```
summary(dfRpty$BETApy)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.37860 -0.03846  0.01652  0.02302  0.10080  0.27570
```

And our dataframe now has the BETA coefficients as an additional variable.

```
head(dfRpty)
```

```
##         Date Decile       Rpty    RMty Year       BETApy
## 1 1964-01-02      8 0.01192597 0.00613 1964 -0.006384071
## 2 1964-01-02      1 0.02342089 0.00613 1964  0.105820632
## 3 1964-01-02      6 0.01414477 0.00613 1964 -0.044375271
## 4 1964-01-02      9 0.01100236 0.00613 1964 -0.011093251
## 5 1964-01-02     10 0.01006818 0.00613 1964 -0.035460283
## 6 1964-01-02      7 0.01235712 0.00613 1964 -0.003725529
```

With those Beta coefficients now in hand, we will merge the coefficients back in with the "dfBeta" dataframe. Notice how the second dataframe we are merging is " select(dfRpty, Date, Decile, BETApy)" which extracts only the variables we wish to be merged. This could be done on multiple lines, but we combined them here to be more concise.

```
dfBeta <- merge(dfBeta, select(dfRpty, Date, Decile, BETApy), by = c("Date", "Decile"))
head(dfBeta)
```

```
##         Date Decile PERMNO Year     Riyd      R100iy   R100YRiy     TCap
## 1 1964-01-02      1  10014 1964 0.043478 -0.04761905 -0.1250000  8422.50
## 2 1964-01-02      1  10532 1964 0.052632 -0.02985075 -0.1625000  3241.88
## 3 1964-01-02      1  10743 1964 0.022901 -0.03896104  0.1492537 13912.00
## 4 1964-01-02      1  10807 1964 0.032787 -0.04545455 -0.3015873  5664.75
## 5 1964-01-02      1  10938 1964 0.000000 -0.04301075  0.2567568 11770.25
## 6 1964-01-02      1  11084 1964 0.028986 -0.13829787  0.3239437  3229.88
##   DIVYLDiy   Tprc    BETApy
## 1 0.000000  2.500 0.1058206
## 2 3.076923  8.125 0.1058206
## 3 3.783784 18.500 0.1058206
## 4 0.000000  5.250 0.1058206
## 5 4.494382 11.125 0.1058206
## 6 0.000000 10.125 0.1058206
```

Clearing "dfRpty" out of memory.

```
rm(dfRpty)
```

Convert from Daily to Annual

For the "dfBeta" dataframe we had to use daily data in order to compute the necessary variables. However, we need the annual observations we computed for the following variables:

- TCap
- Tprc
- DIVYLDiy
- BETApy
- R100iy
- R100YRiy

These variables remain constant across every daily observation for each PERMNO and Year giving us a couple hundred identical observations. In the next step we can collapse these identical observations into one. The dplyr function distinct() makes quick work of this by returning zero duplicate observations. We take "dfBeta" and assign to a new dataframe "addVars". You should notice the use of select(), which leaves out "Date", the only variable(column) that differs in "dfBeta".

```
addVars<- dfBeta %>%
  select(PERMNO, Year, TCap, Tprc, DIVYLDiy, BETApy, R100iy, R100YRiy) %>%
  distinct()
head(addVars)
```

```
##   PERMNO Year     TCap   Tprc DIVYLDiy    BETApy      R100iy   R100YRiy
## 1  10014 1964  8422.50  2.500 0.000000 0.1058206 -0.04761905 -0.1250000
## 2  10532 1964  3241.88  8.125 3.076923 0.1058206 -0.02985075 -0.1625000
## 3  10743 1964 13912.00 18.500 3.783784 0.1058206 -0.03896104  0.1492537
## 4  10807 1964  5664.75  5.250 0.000000 0.1058206 -0.04545455 -0.3015873
## 5  10938 1964 11770.25 11.125 4.494382 0.1058206 -0.04301075  0.2567568
## 6  11084 1964  3229.88 10.125 0.000000 0.1058206 -0.13829787  0.3239437
```

We also have to merge back in the data from "illiqmaTable" for the date manipulations in the next step.

```
addVars <- merge(addVars, illiqmaTable, by=c("PERMNO", "Year"), all = TRUE) %>%
  select(-ILLIQiy, -AILLIQy)
head(addVars)
```

```
##    PERMNO Year     TCap   Tprc DIVYLDiy       BETApy      R100iy
## 1:  10006 1964 235956.0 79.500 2.830189 -0.006384071 -0.02603369
## 2:  10006 1965 279720.0 47.250 7.460317 -0.027614909  0.17027864
## 3:  10006 1966 219861.0 38.250 5.490196 -0.038458947 -0.12068966
## 4:  10006 1967 253229.6 44.875 4.902507  0.051868759 -0.12652068
## 5:  10006 1968 353267.6 62.625 3.592814  0.007366095  0.06144068
## 6:  10006 1969 276409.0 49.000 4.897959  0.024974187  0.06521739
##       R100YRiy  SDRETiy count  ILLIQMAiy
## 1:  0.33265306 1.371656   253 0.08212686
## 2: -0.48566879 1.329256   252 0.06907375
```

```
## 3: -0.05691057 1.847645   252 0.09762887
## 4:  0.32580645 1.617939   251 0.15657148
## 5:  0.34090909 2.596928   226 0.15312331
## 6: -0.27128713 1.594501   250 0.18860015
```

Lag correct variables to get (t-1)

We now lag the annual variables because they are called as t=(y-1) when inserted into the estimations. We can do this simply with the lag() function inside mutate() for the new variables. It is also important that we grou_by() the individual stocks so that we don't lag one stock into the adjacent one. Once that is complete, we select() out the variables we no longer need and remove NAs with na.omit().

```
addVars <- addVars %>%
  group_by(PERMNO) %>%
  mutate(SIZEiy1 = lag(TCap),
         CountY1 = lag(count),
         PriceY1 = lag(Tprc),
         DIVYLDiy1 = lag(DIVYLDiy),
         BETApy1 = lag(BETApy),
         SDRETiy1 = lag(SDRETiy),
         ILLIQiy1 = lag(ILLIQMAiy),
         R100y1 = lag(R100iy),
         R100YRy1 = lag(R100YRiy)) %>%
  select(-TCap, -count, -Tprc, -DIVYLDiy, -BETApy, -SDRETiy, -ILLIQMAiy, -R100iy, -R100YRiy) %>%
  na.omit() %>%
  arrange(Year)
head(addVars)
```

```
##   PERMNO Year  SIZEiy1 CountY1 PriceY1 DIVYLDiy1      BETApy1 SDRETiy1
## 1  13661 1964 769224.5     252   104.5  2.392344 -0.011093251 0.684858
## 2  14605 1964 126126.0     253    33.0  4.545455 -0.044375271 1.090915
## 3  16328 1964  17434.5     248    29.5  4.745763  0.105820632 1.361771
## 4  18673 1964  87875.0     251    37.0  9.189189 -0.008051318 1.192849
## 5  18702 1964  28443.0     253    19.0  5.263158 -0.008132323 1.505309
## 6  20052 1964  41000.0     239    40.0  6.250000  0.072840645 1.131495
##     ILLIQiy1      R100y1    R100YRy1
## 1 0.03881242  0.00000000  0.21688501
## 2 0.35809835 -0.10508475  0.08455882
## 3 0.68954977 -0.05976096  0.24257426
## 4 0.27844159 -0.11377246 -0.34251969
## 5 0.60072190 -0.06172840 -0.02409639
## 6 0.50608511 -0.12087912  0.05813953
```

Finalize Dataframe for Section 2

For Section 2 in the estimations we are looking at using the monthly returns. We will take these out of the "monthlyData" that we uploaded at the top and then adding back in the created variables.

```
DF2 <- monthlyData %>%
  mutate(Rimy = Ret,  #Renaming
         Year = year(Date),
         Month = month(Date)) %>%
```

```
  select(-Date, -Ret) %>%
  arrange(Year)
```

We are going to have to merge this new data frame with the additional variables so we will look at "DF2" and "addVars" side by side.

```
head(DF2); head(addVars)
```

```
##    PERMNO      Rimy Year Month
## 1:  10006  0.068548 1964     1
## 2:  10006  0.043774 1964     2
## 3:  10006  0.000000 1964     3
## 4:  10006  0.003636 1964     4
## 5:  10006 -0.010870 1964     5
## 6:  10006  0.075646 1964     6
```

```
##   PERMNO Year  SIZEiy1 CountY1 PriceY1 DIVYLDiy1      BETApy1 SDRETiy1
## 1  13661 1964 769224.5     252   104.5  2.392344 -0.011093251 0.684858
## 2  14605 1964 126126.0     253    33.0  4.545455 -0.044375271 1.090915
## 3  16328 1964  17434.5     248    29.5  4.745763  0.105820632 1.361771
## 4  18673 1964  87875.0     251    37.0  9.189189 -0.008051318 1.192849
## 5  18702 1964  28443.0     253    19.0  5.263158 -0.008132323 1.505309
## 6  20052 1964  41000.0     239    40.0  6.250000  0.072840645 1.131495
##     ILLIQiy1      R100y1    R100YRy1
## 1 0.03881242  0.00000000  0.21688501
## 2 0.35809835 -0.10508475  0.08455882
## 3 0.68954977 -0.05976096  0.24257426
## 4 0.27844159 -0.11377246 -0.34251969
## 5 0.60072190 -0.06172840 -0.02409639
## 6 0.50608511 -0.12087912  0.05813953
```

Then, merge them with each other:

```
DF2<- merge(DF2, addVars, by=c("PERMNO", "Year"))
head(DF2)
```

```
##   PERMNO Year      Rimy Month SIZEiy1 CountY1 PriceY1 DIVYLDiy1
## 1  10006 1965  0.053459     1  235956     253    79.5  2.830189
## 2  10006 1965 -0.025970     2  235956     253    79.5  2.830189
## 3  10006 1965  0.040248     3  235956     253    79.5  2.830189
## 4  10006 1965  0.084821     4  235956     253    79.5  2.830189
## 5  10006 1965 -0.043621     5  235956     253    79.5  2.830189
## 6  10006 1965 -0.104348     6  235956     253    79.5  2.830189
##        BETApy1 SDRETiy1   ILLIQiy1      R100y1  R100YRy1
## 1 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
## 2 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
## 3 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
## 4 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
## 5 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
## 6 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531
```

Step 4: Filter out stocks that don't meet criteria

"Stocks are admitted to the cross-sectional estimation procedure in month m of year y if thy have a return for that month and they satisfy the following criteria: . . . " (p. 36). We restrict the dataframe with use of the filter() function; in order to stay in the dataframe the condition must be TRUE; if FALSE, the observation drops out.

```
#(i) Return data for more than 200 days during y-1
DF2 <- DF2 %>% filter(CountY1 > 200)
#(ii) Stock price greater than $5 or less than $1000 at end  of y-1
DF2 <- DF2 %>% filter(PriceY1 > 5)
DF2 <- DF2 %>% filter(PriceY1 < 1000)
#(iii) Has market cap at end of y-1
DF2 <- DF2 %>% filter(SIZEiy1 > 0)
```

The next exclusion criteria is "(iv) Eliminate ILLIQiy in y-1 is at highest or lowest 1% of distribution. (after satisfying i-iii)" The way this is done is by assigning each ILLIQiy1 to the percentile by year, then filtering out the 1 and 99 deciles. To do this, we once again use the ntile() function on ILLIQiy1 and then using filter()

```
DF2 <- DF2 %>%
  group_by(Year) %>%
  mutate(Percentile=ntile(ILLIQiy1, 100)) %>%
  filter(Percentile != 1 & Percentile != 100) #Removes exactly 2% of the observations
head(DF2)
```

```
##   PERMNO Year      Rimy Month SIZEiy1 CountY1 PriceY1 DIVYLDiy1
## 1  10006 1965  0.053459     1  235956     253    79.5  2.830189
## 2  10006 1965 -0.025970     2  235956     253    79.5  2.830189
## 3  10006 1965  0.040248     3  235956     253    79.5  2.830189
## 4  10006 1965  0.084821     4  235956     253    79.5  2.830189
## 5  10006 1965 -0.043621     5  235956     253    79.5  2.830189
## 6  10006 1965 -0.104348     6  235956     253    79.5  2.830189
##         BETApy1 SDRETiy1   ILLIQiy1      R100y1  R100YRy1 Percentile
## 1 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
## 2 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
## 3 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
## 4 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
## 5 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
## 6 -0.006384071 1.371656 0.08212686 -0.02603369 0.3326531         22
```

We can look at the summary of observations and see that those in the 1 and 99 percentiles have been removed. One of the great things about the pipe operator (%>%) inside dplyr is that we can combine the filter() lines of code into just a few. Below is the above code written in a much more concise manner, but executes in exactly the same manner.

```
DF2 <- DF2 %>%
  group_by(Year) %>%
  mutate(Percentile=ntile(ILLIQiy1, 100)) %>%
  filter(Percentile != 1,
         Percentile != 100,
         CountY1 > 200,
         PriceY1 > 5,
         PriceY1 < 1000,
```

```
         SIZEiy1 > 0)  %>%
  select(-CountY1, -PriceY1, -Percentile)
```

Step 5: Generate the summary statistics in Table 1

For the summary statistics in the table we need to remove the monthly return observations and look at the distinct() observations.

```
summaryStatistics <- DF2 %>%
  select(-Rimy) %>%
  distinct()
```

We then compute the values from the table. The table reports the "Mean of Annual Mean" which require first using summarise() for the mean() of values for each year, then computing the mean() of those previously computed means. That is why we use summarise twice. (I'm sure there is a more elegant solution than I propose here, but this works)

```
Table1 <- summaryStatistics %>%
  group_by(Year) %>%
  summarise(ILLIQmean = mean(ILLIQiy1),
            SIZEmean = mean(SIZEiy1)/1000,
            DIVYLDmean = mean(DIVYLDiy1),
            SDRETmean = mean(SDRETiy1),
            ILLIQsd = sd(ILLIQiy1),
            SIZEsd = sd(SIZEiy1)/1000,
            DIVYLDsd = sd(DIVYLDiy1),
            SDRETsd = sd(SDRETiy1),
            ILLIQskew = skewness(ILLIQiy1),
            SIZEskew = skewness(SIZEiy1),
            DIVYLDskew = skewness(DIVYLDiy1),
            SDRETskew = skewness(SDRETiy1))
Table1 <- Table1 %>%
  summarise(ILLIQ = mean(ILLIQmean),
            SIZE = mean(SIZEmean),
            DIVYLD = mean(DIVYLDmean),
            SDRET = mean(SDRETmean),
            ILLIQsd = mean(ILLIQsd),
            SIZEsd = mean(SIZEsd),
            DIVYLDsd = mean(DIVYLDsd),
            SDRETsd = mean(SDRETsd),
            ILLIQmed = median(ILLIQmean),
            SIZEmed = median(SIZEmean),
            DIVYLDmed = median(DIVYLDmean),
            SDRETmed = median(SDRETmean),
            ILLIQskew = mean(ILLIQskew),
            SIZEskew = mean(SIZEskew),
            DIVYLDskew = mean(DIVYLDskew),
            SDRETskew = mean(SDRETskew),
            ILLIQmin = min(ILLIQmean),
            SIZEmin = min(SIZEmean),
            DIVYLDmin = min(DIVYLDmean),
            SDRETmin = min(SDRETmean),
```

```
            ILLIQmax = max(ILLIQmean),
            SIZEmax = max(SIZEmean),
            DIVYLDmax = max(DIVYLDmean),
            SDRETmax = max(SDRETmean))
```

After we computed those as variables (which is what summarise() does), we want it in a table form for display. This requires a little bit of manipulation: first, we coerce "Table1" into a matrix so that we can do manipulations on it, then we extract the values as vectors we want and then stack next to each other (with cbind()). This is inside as.data.frame() which coerces this matrix back into the standard dataframe. We then name the columns with names() and print the results.

```
Table1 <- as.matrix(Table1)
Table1<- as.data.frame(cbind(Table1[1,1:4],
                             Table1[1,5:8],
                             Table1[1,9:12],
                             Table1[1,13:16],
                             Table1[1,17:20],
                             Table1[1,21:24]))
names(Table1) <- c("meanOfMeans",
                   "meanOfsd",
                   "medianOfMeans",
                   "meanOfSkew",
                   "minOfMean",
                   "maxOfMean")
knitr::kable(Table1)
```

	meanOfMeans	meanOfsd	medianOfMeans	meanOfSkew	minOfMean	maxOfMean
ILLIQ	0.3939067	0.5613193	0.3831945	2.927381	0.0117451	0.8965867
SIZE	888.3680887	1749.6641325	572.8139099	5.128550	189.5933287	2234.4646871
DIVYLD	4.3870708	6.7164878	4.1845602	8.472172	2.7904171	7.0419095
SDRET	1.9980149	0.7126479	1.9894153	1.044987	1.2802381	2.7529578

Step 6: Estimate Eq (2) and Table 2

Section 3. The effect over time of market illiquidity on expected stock excess return

Does illiqudity impact returns?

The idea in this section is that expected market illiquidity positively impacts exected excess return. Basically, if investors expect there to be future illiquidity, it will be priced into the stocks at the current time, thus generating that future excess return.

Predicting Market Illiquidity (and unexpected portion)

"Investors are assumed to predict illiquidity for year y based on information available in year y-1, and then use this prediction to set prices that will generate the desired expected return in year y." (p. 43). It is assumed that expected market illiquidity follows the AR(1) model of:

$$lnAILLIQ_y = c_0 + c_1 lnAILLIQ_{y-1} + u_y \qquad (7)$$

where c_1 is expected to be positive and the residual u_y containing the unexpected portion of liquidity.

We are looking for the residual from eq (7) so that we can model how the unexpected portion impacts returns on those assets. We can do this by creating a new dataframe called "ailliqTable". The first step is to take the natural log(log()) and lag the observations for (t-1) using mutate().

```
ailliqTable <- mutate(ailliqTable,
                      lnAILLIQy = log(AILLIQy),
                      lnAILLIQy1 = lag(lnAILLIQy))
```

Then, we want to estimate the AR model and we can use the built-in R function ar() which we are able to extract values from in a similar fashion to what we did with lm(). We assign the coefficients for c_0 and c_1 to their respective values.

```
ailliqAR <- ar(ailliqTable$lnAILLIQy)
c0 <- ailliqAR$x.mean  #c0 = -0.2117
c1 <- ailliqAR$ar      #c1 = 0.7328
```

We need to take the residual values from "ailliqAR" and assign to a new variable called "ResUnAdj" with a process similar to extracting lm() coefficients.

```
table3 <- ailliqTable %>%
  mutate(ResUnAdj = ailliqAR$resid)
head(table3)
```

```
##   Year   AILLIQy  lnAILLIQy lnAILLIQy1     ResUnAdj
## 1 1964 0.7938818 -0.2308207         NA           NA
## 2 1965 0.6086327 -0.4965403 -0.2308207 -0.270826142
## 3 1966 0.6188801 -0.4798437 -0.4965403 -0.059404835
## 4 1967 0.3520700 -1.0439251 -0.4798437 -0.635721875
## 5 1968 0.2171579 -1.5271307 -1.0439251 -0.705557356
## 6 1969 0.3078331 -1.1781974 -1.5271307 -0.002521365
```

section 3.2: Illiquidity and excess returns on size portfolios

Amihud hypothesize that, yes, with increased illiquidity, there is a decline in stock prices. More interesting, I think, is that when there is a market wide flight to liquidity, the price impact on the relatively more liquid stocks is lesser as an investor dumping the relatively less liquid stocks should have a larger price impact. The "flight to liquidity" is going to be highly correlated with the size of the company. So, to test for this impact the above specification is tested using five different size portfolios.

The two hypotheses are:
1. ex ante stock excess return is an increasing function of expected liquidity. - below: $g_1 > 0$
2. unexpected illiquidity has a negative effect on contemporaneous unexpected stock return. - below: $g_2 < 0$

g_1 and g_2 are estimated from the following equation:

$$(RM - Rf)_y = g_0 + g_1 lnAILLIQ_{y-1} + g_2 lnAILLIQ_y^U + w_y, \qquad (10)$$

where $lnAILLIQ_y^U$ is the unexpected illiquidity from year y, and $lnAILLIQ_y^U$ is the unexpected illiquidity as the residual from eq (7).

Input Market Return Data and Merge

Because we are looking at the excess return on the market in each period, we need to bring in the market data from CRSP.

```
marketData <- as.data.frame(fread("Amihud_MarketReturn.txt"))
head(marketData)
```

```
##    Series     Date        TRet
## 1 1000000 19630131  0.05129926
## 2 1000000 19630228 -0.02253303
## 3 1000000 19630329  0.03421342
## 4 1000000 19630430  0.04790276
## 5 1000000 19630531  0.02039786
## 6 1000000 19630628 -0.01799958
```

You should notice that the market data comes in a different form than the CRSP data we have been working with from the start. The form this data came in does not play nicely with observations as rows and variables as columns. (see Wickham(2014) "Tidy Data") The "tidyr" package makes manipulating the data quite easy and is pretty intuitive if familiar with dplyr. (Its written by the same people) For more info, see this cheat sheet on data manipulation. What we are doing is converting the dataframe into "tidy data" where we have each row as a singular date with each CRSP series being recorded as a variable.

```
marketData <- spread(marketData, Series, TRet)
head(marketData)
```

```
##       Date     1000000     1000001      1000002      1000003      1000004
## 1 19630131  0.05129926  0.07818522  0.116501700  0.111511600  0.102516000
## 2 19630228 -0.02253303 -0.01477975 -0.002443536 -0.010540440 -0.008185182
## 3 19630329  0.03421342  0.02087015  0.011834350  0.007485228  0.027356970
## 4 19630430  0.04790276  0.03850007  0.022506870  0.034139730  0.049507190
## 5 19630531  0.02039786  0.03267867  0.053883120  0.048420480  0.037697120
## 6 19630628 -0.01799958 -0.01577538 -0.023357840 -0.014410450 -0.011982400
##       1000005     1000006     1000007     1000008     1000009      1000010
## 1  0.07648660  0.08328042  0.06637781  0.06700648  0.05999529  0.056445090
## 2 -0.02185773 -0.01479508 -0.01435569 -0.01925426 -0.01733441 -0.017571970
## 3  0.01363800  0.01745298  0.01846576  0.01541369  0.02491851  0.033138420
## 4  0.03843699  0.03067376  0.03118780  0.04687088  0.04986782  0.038639060
## 5  0.03357137  0.03918702  0.03272865  0.02793653  0.02173318  0.020775540
## 6 -0.01840536 -0.01637947 -0.01800028 -0.01783885 -0.02058187 -0.006966115
##       1000011  1000706
## 1  0.04508766 0.002809
## 2 -0.02523980 0.002577
## 3  0.03852723 0.002439
## 4  0.05120860 0.002847
## 5  0.01785470 0.001777
## 6 -0.02013321 0.002935
```

As you saw before we tidy'd up our data, the deciles of NYSE returns was stacked and now is spread() out in a horizontal fashion. What you don't see is appropriate names for the variables so we should probably change them to reflect what they really are. The next step generates a vector of strings with the necessary deciles.

```
deciles <- paste("Decile_", c(1:10), sep="")
deciles
```

```
##  [1] "Decile_1"  "Decile_2"  "Decile_3"  "Decile_4"  "Decile_5"
##  [6] "Decile_6"  "Decile_7"  "Decile_8"  "Decile_9"  "Decile_10"
```

Of course, you could have done this by typing out the names of each Decile, but using paste() and a vector of numbers "c(1:10)" was able to create the vector with a lot less typing (remember DRY?). We can then change the names of "marketData" and convert the dates to lubridate class.

```
names(marketData) <- c("Date", "NYSE_Value", "NYSE_Equal", deciles, "OneYrT")
marketData$Date <- ymd(marketData$Date)
head(marketData)
```

```
##         Date  NYSE_Value  NYSE_Equal     Decile_1     Decile_2
## 1 1963-01-31  0.05129926  0.07818522  0.116501700  0.111511600
## 2 1963-02-28 -0.02253303 -0.01477975 -0.002443536 -0.010540440
## 3 1963-03-29  0.03421342  0.02087015  0.011834350  0.007485228
## 4 1963-04-30  0.04790276  0.03850007  0.022506870  0.034139730
## 5 1963-05-31  0.02039786  0.03267867  0.053883120  0.048420480
## 6 1963-06-28 -0.01799958 -0.01577538 -0.023357840 -0.014410450
##       Decile_3    Decile_4    Decile_5    Decile_6    Decile_7    Decile_8
## 1  0.102516000  0.07648660  0.08328042  0.06637781  0.06700648  0.05999529
## 2 -0.008185182 -0.02185773 -0.01479508 -0.01435569 -0.01925426 -0.01733441
## 3  0.027356970  0.01363800  0.01745298  0.01846576  0.01541369  0.02491851
## 4  0.049507190  0.03843699  0.03067376  0.03118780  0.04687088  0.04986782
## 5  0.037697120  0.03357137  0.03918702  0.03272865  0.02793653  0.02173318
## 6 -0.011982400 -0.01840536 -0.01637947 -0.01800028 -0.01783885 -0.02058187
##       Decile_9   Decile_10   OneYrT
## 1  0.056445090  0.04508766 0.002809
## 2 -0.017571970 -0.02523980 0.002577
## 3  0.033138420  0.03852723 0.002439
## 4  0.038639060  0.05120860 0.002847
## 5  0.020775540  0.01785470 0.001777
## 6 -0.006966115 -0.02013321 0.002935
```

We have to get $(RM - Rf)_y$ which entails subtracting the 1 year Treasury from each of the return series. We convert the returns in "marketData" to be the risk premium on each return series as follows.

```
marketData <- marketData %>%
  mutate(Year = year(Date),
         Month = month(Date)) %>%
  select(-Date)
marketData <- cbind(marketData$Year,
                    marketData$Month,
                    marketData$OneYrT,
                    marketData[,1:12] - marketData$OneYrT)
names(marketData)[1:3] <- c("Year", "Month", "OneYrT")
head(marketData)
```

```
##   Year Month   OneYrT  NYSE_Value  NYSE_Equal     Decile_1     Decile_2
## 1 1963     1 0.002809  0.04849026  0.07537622  0.113692700  0.108702600
```

```
## 2 1963     2 0.002577 -0.02511003 -0.01735675 -0.005020536 -0.013117440
## 3 1963     3 0.002439  0.03177442  0.01843115  0.009395350  0.005046228
## 4 1963     4 0.002847  0.04505576  0.03565307  0.019659870  0.031292730
## 5 1963     5 0.001777  0.01862086  0.03090167  0.052106120  0.046643480
## 6 1963     6 0.002935 -0.02093458 -0.01871038 -0.026292840 -0.017345450
##      Decile_3    Decile_4    Decile_5    Decile_6    Decile_7    Decile_8
## 1  0.09970700  0.07367760  0.08047142  0.06356881  0.06419748  0.05718629
## 2 -0.01076218 -0.02443473 -0.01737208 -0.01693269 -0.02183126 -0.01991141
## 3  0.02491797  0.01119900  0.01501398  0.01602676  0.01297469  0.02247951
## 4  0.04666019  0.03558999  0.02782676  0.02834080  0.04402388  0.04702082
## 5  0.03592012  0.03179437  0.03741002  0.03095165  0.02615953  0.01995618
## 6 -0.01491740 -0.02134036 -0.01931447 -0.02093528 -0.02077385 -0.02351687
##       Decile_9   Decile_10
## 1  0.053636090  0.04227866
## 2 -0.020148970 -0.02781680
## 3  0.030699420  0.03608823
## 4  0.035792060  0.04836160
## 5  0.018998540  0.01607770
## 6 -0.009901115 -0.02306821
```

What is going on here? There are 12 return series and of course we could subtract "OneYrT" from each of the columns manually, but that would be a little bit cumbersome. R is vectorised so we can just say "marketData[,1:12] - marketData$OneYrT" to do calculate all 12 columns. You should notice that it is inside cbind() which "column binds" the vectors inside - we want "Year" and "OneYrT" to stay with the data. We also had to change the names of the first two variables because when we cbind(), the names from the first two vectors come through as whatever is written within the function.

The other thing we have to do here is to convert the monthly returns in "marketData" to annual. Unfortunately, the only way I am able to download the indices data from CRSP is monthly so we do have to convert it all. To convert into annual as (1+r1)(1+r2)...(1+r12) - 1. We group_by() the Year for the total return that year, then summarise() for the single value, using cumprod() as the cumulative product of the returns + 1. cumprod() returns a vector of values the same length as the input vector - meaning, our 12 monthly returns give a vector of length 12. We need the 12th value of the vector as it is the cumulative product of all 12 values "cumprod((1+returnVector))[12]". We then select() only the variables we want to keep and mutate() the values by subtracting 1 from each.

```
marketDataAnnual <- marketData %>%
  group_by(Year) %>%
  summarise(returns = cumprod((1+NYSE_Equal))[12],
            d2 = cumprod((1+Decile_2))[12],
            d4 = cumprod((1+Decile_4))[12],
            d6 = cumprod((1+Decile_6))[12],
            d8 = cumprod((1+Decile_8))[12],
            d10 = cumprod((1+Decile_10))[12]) %>%
  select(Year, returns, d2, d4, d6, d8, d10) %>%
  mutate(returns = returns - 1,
         d2 = d2-1,
         d4 = d4-1,
         d6 = d6-1,
         d8 = d8-1,
         d10= d10-1)
head(marketDataAnnual)
```

```
## # A tibble: 6 x 7
```

```
##    Year    returns          d2         d4         d6         d8
##   <int>      <dbl>       <dbl>      <dbl>      <dbl>      <dbl>
## 1  1963  0.1532375  0.18909955  0.1397022  0.1229985  0.1468506
## 2  1964  0.1352795  0.14304380  0.1529537  0.1178633  0.1384502
## 3  1965  0.2425724  0.29357654  0.2985012  0.2448992  0.2076231
## 4  1966 -0.1161764 -0.09712026 -0.1097583 -0.1157538 -0.1128259
## 5  1967  0.4407390  0.67859935  0.6714891  0.4650922  0.3136685
## 6  1968  0.2376389  0.34249083  0.2458077  0.2207294  0.1193215
## # ... with 1 more variables: d10 <dbl>
```

Now that we have the annual market data we can merge it back in with the data from "table3".

```
table3 <- na.omit(merge(table3, marketDataAnnual, by = "Year"))
head(table3)
```

```
##    Year   AILLIQy  lnAILLIQy lnAILLIQy1     ResUnAdj    returns
## 1: 1965 0.6086327 -0.4965403 -0.2308207 -0.270826142  0.2425724
## 2: 1966 0.6188801 -0.4798437 -0.4965403 -0.059404835 -0.1161764
## 3: 1967 0.3520700 -1.0439251 -0.4798437 -0.635721875  0.4407390
## 4: 1968 0.2171579 -1.5271307 -1.0439251 -0.705557356  0.2376389
## 5: 1969 0.3078331 -1.1781974 -1.5271307 -0.002521365 -0.2482673
## 6: 1970 0.8645492 -0.1455471 -1.1781974  0.774423678 -0.1193455
##             d2         d4         d6         d8         d10
## 1:  0.29357654  0.2985012  0.2448992  0.2076231  0.05040290
## 2: -0.09712026 -0.1097583 -0.1157538 -0.1128259 -0.14701742
## 3:  0.67859935  0.6714891  0.4650922  0.3136685  0.16206468
## 4:  0.34249083  0.2458077  0.2207294  0.1193215  0.01908614
## 5: -0.33785182 -0.2606326 -0.2262578 -0.1509551 -0.11148517
## 6: -0.15472881 -0.1250840 -0.1285127 -0.0700246 -0.06650791
```

Estimate Table 3 results

The dataframe for eq (10) estimations is complete and now need to estimate g_1 and g_2 from:

$$(RM - Rf)_y = g_0 + g_1 lnAILLIQ_{y-1} + g_2 lnAILLIQ_y^U + w_y, \qquad (10)$$

Because the calculations for the entire market and the portfolios are the same and is just rewriting the same code, all I show here is the results from the returns on the entire market. Generating the linear model results is straightforward with lm() and summary().

```
table3Results <- lm((returns*100) ~ lnAILLIQy1 + ResUnAdj, data = table3)
summary(table3Results)
```

```
##
## Call:
## lm(formula = (returns * 100) ~ lnAILLIQy1 + ResUnAdj, data = table3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -25.24 -15.76  -1.11  12.11  35.85
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    8.693      2.976   2.921  0.00656 **
## lnAILLIQy1     7.569      3.166   2.391  0.02330 *
## ResUnAdj     -15.390      4.659  -3.303  0.00248 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.73 on 30 degrees of freedom
## Multiple R-squared:  0.3531, Adjusted R-squared:  0.3099
## F-statistic: 8.187 on 2 and 30 DF,  p-value: 0.001455
```

As can be seen, my results are slightly different than those reported in Amihud(2002), but they are relatively close on all measures. Amihud also conducts a Durbin-Watson test and Newey-West correction to heteroskedasticy and autocorrelation. The functions dwtest() comes from the "lmtest" package and does what it's name suggests - tests Durbin-Watson.

```
dwtest(table3Results)
```

```
##
##  Durbin-Watson test
##
## data:  table3Results
## DW = 2.3465, p-value = 0.7957
## alternative hypothesis: true autocorrelation is greater than 0
```

For Newey-West we use coeftest() from the "lmtest" package. coeftest() takes as its first argument the lm() object we are using, and the second argument specifying which covariance matrix we wish to use on the estimated coefficients. In this case we use the NeweyWest() function from the "sandwich" library as the covariance matrix within coeftest().

```
coeftest(table3Results, vcov = NeweyWest(table3Results))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   8.6929     1.4082  6.1730 8.605e-07 ***
## lnAILLIQy1    7.5686     3.1425  2.4084  0.022371 *
## ResUnAdj    -15.3897     5.2746 -2.9177  0.006622 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Building Table 3

The estimates above now need to be moved into a table. Naturally, one way to do this is to copy the values into a spreadsheet and generate a pretty table. Since this is showing how to write code, we are going to create the table by extracting the values out of the lm() objects. The concept of building the table is relatively straightforward and I will explain it step-by-step. Basically, what we are attempting to accomplish is to not copy-paste any code and have the repetitive tasks automated for us. We are essentially doing the same task six times (for the market and 5 size portfolios). This lends itself to writing a for loop.

A high level overview is that we are creating a vector of values that are extracted from regression results and then stacking them side by side.

```
depVars <- c("returns", "d2", "d4", "d6", "d8", "d10")
table3DF <- c("Constant", "", "LnAILLIQ(y-1)", "", "LnAILLIQ(U,y)", "", "R^2", "D-W")

for (i in 1:6) {
  results <- lm((get(depVars[i])*100) ~ lnAILLIQy1 + ResUnAdj, data = table3)
  nw <- coeftest(results, vcov = NeweyWest(results))
  coef<-results$coefficients
  vector <- round(as.vector(c(coef[1],
                              nw[1,3],
                              coef[2],
                              nw[2,3],
                              coef[3],
                              nw[3,3],
                              summary(results)$r.squared,
                              dwtest(results)$statistic) ),
                  digits=3)
  table3DF <- cbind(table3DF, vector)
}

table3DF <- as.data.frame(table3DF)
table3Names <- c("", "RM-Rf","RSZ(2)-Rf", "RSZ(4)-Rf", "RSZ(6)-Rf", "RSZ(8)-Rf", "RSZ(10)-Rf")
names(table3DF) <- table3Names

knitr::kable(table3DF)
```

	RM-Rf	RSZ(2)-Rf	RSZ(4)-Rf	RSZ(6)-Rf	RSZ(8)-Rf	RSZ(10)-Rf
Constant	8.693	10.675	10.488	8.904	7.627	4.384
	6.173	3.355	4.689	7.49	10.417	2.051
LnAILLIQ(y-1)	7.569	8.958	8.094	7.023	6.219	2.905
	2.408	2.286	2.476	2.409	2.604	1.568
LnAILLIQ(U,y)	-15.39	-19.147	-15.099	-13.008	-11.644	-8.493
	-2.918	-3.553	-2.723	-2.23	-2.592	-1.914
R^2	0.353	0.334	0.277	0.281	0.301	0.17
D-W	2.346	2.136	2.352	2.298	2.298	1.996

What we did is:

1. Create a vector of strings that are the names of the dependent variables.
2. Create vector of the row names - this will be the first column of the dataframe.
   - We have to do this to initialize the new dataframe before entering the loop.
   - Notice "", we want a blank in these spots and have to place a blank space in the vector
3. Start the for loop, we specify it as running 6 times because there are 6 regressions
   - Every time it goes through the loop, i gets larger by 1.
4. Run the regression and assign to "results". Notice "get(depVars[1])"?
   - depVars[i] gets the value from whatever iteration i is on. First time through, i->1, and depVars[1]-> "returns"
   - get() converts that string to a literal value that can be executed inside lm()
5. Create "nw" and "coef" to make for less typing

- Assign Newey-West values to "nw", which is generated as a dataframe that we can extract the appropriate values from.
- Create a vector of coefficient values

6. Create a vector of the table values for each regression
    - This is wrapped in round() because we only want three decimal points
7. Attach this vector to "table3DF" with cbind()
8. Repeat until all 6 regressions and results have been completed
9. Convert to a dataframe (it is currently as a matrix and cannot have column names)
10. Rename the dataframe to appropriate titles

11. Display table with knitr::kable()

section 3.3: Monthly data - Illiquidity and Stock Prices

The above section used annual illiquidity data and this section is evaluating the impact of monthly estimated liquidity. There are only a couple of small changes to this section from Section 3.2.

Generating monthly illiquidity data

We compute MILLIQ like $\text{AILLIQ}_{\text{iy}}$ from the first section with the only change being that we are adding an additional grouping of month into group_by(). Notice how month(Date) will be able to extract the month out of the Date variable? This is a really neat feature of the lubridate package in that it provides these capabilities. month() will extract a number (as in 1 for January), whereas months() will extract a string(such as "January").

```
milliqTable<- daily %>%
  group_by(PERMNO, Year, month(Date)) %>%
  summarise(ILLIQiy = mean(ILLIQiyd) * 1000000) %>%
  group_by(Year, `month(Date)`) %>%
  summarise(MILLIQm = mean(ILLIQiy))
names(milliqTable)[2] <- "Month"
milliqTable <- ungroup(milliqTable)
head(milliqTable)
```

```
##   Year Month   MILLIQm
## 1 1964     1 0.9387919
## 2 1964     2 0.9271245
## 3 1964     3 0.7927149
## 4 1964     4 0.7597969
## 5 1964     5 0.8234511
## 6 1964     6 0.9204541
```

What we just did above is to create a new dataframe called "milliqTable", which contains the monthly illiquidity data rather than the annual figures as calculated in all of the above calculations. The difference is that in the beginning sections we were grouping by Year, whereas here we group by Month. "milliqTable" contains the data we need to move to the next section.

Generate AR(1) estimates

Compute an autoregressive model on monthly data similar to what we did in Section 3.2.

$$lnMILLIQ_m = c_0 + c_1 lnMILLIQ_{m-1} + u_y \qquad (7m)$$

Where MILLIQ is the average monthly illiquidity. We are obtaining the residuals from eq. (7m), which then is estimated similar to eq. (10) as follows:

The first step is to take the natural log of the monthly illiquidities and lag the values. We create a new dataframe called "milliqTable."

```
milliqTable <- mutate(milliqTable, lnMILLIQm = log(MILLIQm), lnMILLIQm1 = lag(lnMILLIQm))
head(milliqTable)
```

```
##   Year Month   MILLIQm   lnMILLIQm  lnMILLIQm1
## 1 1964     1 0.9387919 -0.06316146          NA
## 2 1964     2 0.9271245 -0.07566744 -0.06316146
## 3 1964     3 0.7927149 -0.23229170 -0.07566744
## 4 1964     4 0.7597969 -0.27470416 -0.23229170
## 5 1964     5 0.8234511 -0.19425112 -0.27470416
## 6 1964     6 0.9204541 -0.08288813 -0.19425112
```

With the data in the right form we generate the AR(1) specificationwith ar(). You should notice that we inserted an argument into the function (order.max=1) to limit it to the AR(1). The ar() function by default selects the autoregressive time series to the data by complexity accoring to the AIC; without the argument the function selected an AR(2). By assigning to "milliqAR" we can extract the values we want, such as c0 and c1. These values are pretty close to those reported in the paper, with the exception of c0 in which my estimates are -0.329, versus the paper showed 0.313. I am assuming that this was an oversight in the original paper and would be relatively easy to miss.

```
milliqAR <- ar(milliqTable$lnMILLIQm, order.max=1)
c0 <- milliqAR$x.mean
c0
```

```
## [1] -0.3293527
```

```
c1 <- milliqAR$ar
c1
```

```
## [1] 0.9198247
```

With the AR results in hand we take "milliqTable", then add a new variable called "MonthlyRes" and assign to it the vector of residuals (milliqAR$resid). This is assigned to a new dataframe called "table4" and NAs removed.

```
table4 <- milliqTable %>%
  mutate(MonthlyRes = milliqAR$resid) %>% #Assigns the vector of residuals into the dataset
  na.omit()
head(table4)
```

```
##   Year Month   MILLIQm   lnMILLIQm  lnMILLIQm1   MonthlyRes
## 1 1964     2 0.9271245 -0.07566744 -0.06316146  0.008835976
## 2 1964     3 0.7927149 -0.23229170 -0.07566744 -0.136284975
## 3 1964     4 0.7597969 -0.27470416 -0.23229170 -0.034630561
## 4 1964     5 0.8234511 -0.19425112 -0.27470416  0.084834504
## 5 1964     6 0.9204541 -0.08288813 -0.19425112  0.122194797
## 6 1964     7 0.7899440 -0.23579319 -0.08288813 -0.133144690
```

In Section 3.2 we created a dataframe called "marketData" which included the monthly return data. We merge this with "table4" and select() only the variables we wish to keep to keep the dataframe manageable.

```
table4 <- merge(table4, marketData, by = c("Year", "Month")) %>%
  select(Year, Month, lnMILLIQm1, MonthlyRes, OneYrT, NYSE_Equal,
         Decile_2, Decile_4, Decile_6, Decile_8, Decile_10)
```

In the estimation we also need to generate the January Dummy, which is equal to 1 if January, 0 otherwise. We create this variable inside mutate() with the ifelse() function. The first argument of ifelse() is the TRUE/FALSE test which in this case is whether "Month" is equal to 1 (remember that "==" is the conditional, not "="). The second argument is the value it returns if TRUE, and the last is the value if FALSE.

```
table4 <- mutate(table4, JANDUMm = ifelse(Month==1,1,0))
head(table4)
```

```
##    Year Month  lnMILLIQm1   MonthlyRes   OneYrT   NYSE_Equal     Decile_2
## 1: 1964     2 -0.06316146  0.008835976 0.001689  0.024848850  0.019857280
## 2: 1964     3 -0.07566744 -0.136284975 0.004160  0.027595680  0.035725870
## 3: 1964     4 -0.23229170 -0.034630561 0.005247 -0.008723141 -0.019013200
## 4: 1964     5 -0.27470416  0.084834504 0.002527  0.009727150 -0.003582725
## 5: 1964     6 -0.19425112  0.122194797 0.003922  0.011529230  0.009424270
## 6: 1964     7 -0.08288813 -0.133144690 0.004481  0.023339330  0.034044410
##       Decile_4     Decile_6     Decile_8   Decile_10 JANDUMm
## 1:  0.03108484  0.024518220  0.026030650 0.013071270       0
## 2:  0.05307744  0.023688990  0.024941020 0.008366830       0
## 3: -0.02262504 -0.019265620 -0.009272046 0.003130518       0
## 4:  0.02281576  0.005647104  0.012968090 0.015149810       0
## 5:  0.01525510  0.017083530  0.011402320 0.012256000       0
## 6:  0.00900031  0.028578720  0.019373730 0.011804690       0
```

We now have a complete dataframe and can run our estimations for the section.

Estimate Equation (10m) and Table 4

We are estimating the values of g_1 amd g_2 from the following equation.

$$(RM - Rf)_m = g_0 + g_1 lnMILLIQ_{m-1} + g_2 lnMILLIQ_m^U + g_3 JANDUM_m + w_y, \qquad (10m)$$

```
table4Results <- lm((NYSE_Equal*100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm, data = table4)
summary(table4Results)
```

```
##
## Call:
## lm(formula = (NYSE_Equal * 100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm,
##     data = table4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.7902  -2.3615   0.1446   2.9562  24.1536
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.2684     0.2553   1.051   0.2937
## lnMILLIQm1    0.4644     0.2551   1.821   0.0694 .
## MonthlyRes   -4.4072     0.6778  -6.502 2.35e-10 ***
## JANDUMm       5.6514     0.8867   6.373 5.07e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.694 on 403 degrees of freedom
## Multiple R-squared:  0.1427, Adjusted R-squared:  0.1363
## F-statistic: 22.35 on 3 and 403 DF,  p-value: 2.082e-13
```

The coefficients and statistical significance are close to those from the paper for the entire market. We also need to conduct the Durbin-Watson test and White's heteroskedastic-consistent test. White's 1981 test is "HC1" and is tested similar to how we did Newey-West in Section 3.2. We use coeftest() from the "lmtest" package and specify the covariance matrix using vcovHC() from the "sandwich" package. vcovHC() gives us a number of "type" choices (check the sandwich documentation for other choices), but in this case we specify "HC1."

```
dwtest(table4Results)
```

```
##
##  Durbin-Watson test
##
## data:  table4Results
## DW = 1.9862, p-value = 0.4305
## alternative hypothesis: true autocorrelation is greater than 0
```

```
coeftest(table4Results, vcov = vcovHC(table4Results, type = "HC1"))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.26843    0.25305  1.0608    0.2894
## lnMILLIQm1   0.46444    0.29605  1.5688    0.1175
## MonthlyRes  -4.40719    0.95427 -4.6184 5.210e-06 ***
## JANDUMm      5.65140    1.25398  4.5068 8.638e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can set up a table just like we did in Section 3.2 to display the results.

```
depVars <- c("NYSE_Equal",
             "Decile_2",
             "Decile_4",
             "Decile_6",
             "Decile_8",
             "Decile_10")
table4DF <- c("Constant", "",
              "LnMILLIQ(y-1)", "",
              "LnMILLIQ(U,y)", "",
              "JANDUMm", "",
              "R^2",
              "D-W")
for (i in 1:6) {
  results <- lm((get(depVars[i])*100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm, data = table4)
  hc <- coeftest(results, vcov = vcovHC(results, type = "HC1"))
  coef<-results$coefficients
  vector <- round(as.vector(c(coef[1],
                              hc[1,3],
                              coef[2],
                              hc[2,3],
                              coef[3],
                              hc[3,3],
                              coef[4],
                              hc[4,3],
                              summary(results)$r.squared,
                              dwtest(results)$statistic) ),
                  digits=3)
  table4DF <- cbind(table4DF, vector)
}
table4Names <- c("", "RM-Rf","RSZ(2)-Rf", "RSZ(4)-Rf", "RSZ(6)-Rf", "RSZ(8)-Rf", "RSZ(10)-Rf")
table4DF <- as.data.frame(table4DF)
names(table4DF) <- table4Names
knitr::kable(table4DF)
```

	RM-Rf	RSZ(2)-Rf	RSZ(4)-Rf	RSZ(6)-Rf	RSZ(8)-Rf	RSZ(10)-Rf
Constant	0.268	0.151	0.389	0.4	0.433	0.251
	1.061	0.539	1.49	1.566	1.746	1.123
LnMILLIQ(y-1)	0.464	0.508	0.538	0.519	0.465	0.133
	1.569	1.453	1.775	1.75	1.676	0.538
LnMILLIQ(U,y)	-4.407	-5.297	-4.637	-4.271	-3.456	-2.017
	-4.618	-4.732	-4.631	-4.492	-4.032	-2.908
JANDUMm	5.651	8.544	5.821	4.609	3.231	1.454
	4.507	5.225	4.319	3.753	2.866	1.551
R^2	0.143	0.187	0.139	0.119	0.082	0.033
D-W	1.986	2.001	1.952	1.998	2.06	2.182

Section 3.4: Illiquidity effect, controlling for the effects of bond yield premiums

In this section we perform essentially the same estimations as we did in Section 3.3 with the one exception being that we control for the default yield premium and the term yield premium. We are going to append the new market data from the bond yield premiums to the data we used for Table4. The results from this

section are not as close as the previous sections, but do confirm the results found by Amihud (2002). I am assuming that this is largely due to the bond yield data not being exactly the same as what Amihud uses.

We first need to bring in the appropriate bond-yield data. Amihud(2002) cites the bond premiums are from Basic Economics, which sources from the Federal Reserve. However, there seemed to be issues with being able to access this data directly from Basic Economics. This is pretty widespread data and we can download it through Quandl's API. We are looking for AAA, BAA, Long-Term Treasuries, and 3-Month T-Bills to determine the Default Yield Premium and Term Yield Premium.

$$DEF_m = YBAA_m - YAAA_m$$

$$TERM_m = YLONG_m - YTB3_m$$

Downloading the data through Quandl is pretty straightforward by inserting the code for the data series we want, start date, end date, and the frequency. The Quandl codes are defined as:

- MOODY/AAAYLD = Aaa Corporate Bond Yield from Moody's
- MOODY/BAAYLD = Baa Corporate Bond Yield from Moody's
- FRED/DTB3 = 3-Month Treasury Bill: Secondary Market Rate
- FRED/LTGOVTBD = Long-Term US Goverment Securities (Discontinued, but covers the time period)
  - Defined as: Percent Not Seasonally Adjusted, Unweighted average on all outstanding bonds neither due nor callable in less than 10 years.

We download into a new dataframe "marketData" and convert the names into something more easily workable.

```
marketData <- Quandl(c("MOODY/AAAYLD","MOODY/BAAYLD","FRED/DTB3","FRED/LTGOVTBD"), start_date = "1964-0
names(marketData) <- c("Date", "YAAAm", "YBAAm", "YTB3m", "YLONGm")
head(marketData)
```

```
##         Date YAAAm YBAAm YTB3m YLONGm
## 1 1964-01-31  4.39  4.83  3.50   4.15
## 2 1964-02-29  4.36  4.83  3.60   4.14
## 3 1964-03-31  4.38  4.83  3.51   4.18
## 4 1964-04-30  4.40  4.85  3.45   4.20
## 5 1964-05-31  4.41  4.85  3.47   4.16
## 6 1964-06-30  4.41  4.85  3.47   4.13
```

Because all our other data has the dates in Lubridate form, we have to convert the dates with ymd(), get "Year" and "Month" so we can merge with the "table4" dataframe. We also create "DEFm" and "TERMm" for our regressions.

```
marketData <- marketData %>%
  mutate(Date = ymd(Date),
         Year = year(Date),
         Month = month(Date),
         DEFm = YBAAm - YAAAm,
         TERMm = YLONGm - YTB3m) %>%
  select(Year, Month, DEFm, TERMm)
head(marketData)
```

```
##   Year Month DEFm TERMm
## 1 1964     1 0.44  0.65
## 2 1964     2 0.47  0.54
## 3 1964     3 0.45  0.67
## 4 1964     4 0.45  0.75
## 5 1964     5 0.44  0.69
## 6 1964     6 0.44  0.66
```

We merge this "marketData" into "table4".

```
table5 <- merge(marketData, table4, by = c("Year", "Month"))
head(table5)
```

```
##   Year Month DEFm TERMm  lnMILLIQm1   MonthlyRes   OneYrT    NYSE_Equal
## 1 1964    10 0.39  0.61 -0.31338961 -0.102095628 0.002283  0.0147222000
## 2 1964    11 0.38  0.28 -0.41676508 -0.093261520 0.001492 -0.0007512737
## 3 1964    12 0.37  0.32 -0.50301829 -0.023078265 0.005349 -0.0122826420
## 4 1964     2 0.47  0.54 -0.06316146  0.008835976 0.001689  0.0248488500
## 5 1964     3 0.45  0.67 -0.07566744 -0.136284975 0.004160  0.0275956800
## 6 1964     4 0.45  0.75 -0.23229170 -0.034630561 0.005247 -0.0087231410
##       Decile_2    Decile_4      Decile_6     Decile_8   Decile_10 JANDUMm
## 1  0.017326140  0.02558672  0.0282632700  0.012413080 0.005373024       0
## 2 -0.001821667 -0.01131895 -0.0007208139 -0.003880280 0.001022723       0
## 3 -0.024023810 -0.01293574 -0.0145500540 -0.006373341 0.001999679       0
## 4  0.019857280  0.03108484  0.0245182200  0.026030650 0.013071270       0
## 5  0.035725870  0.05307744  0.0236889900  0.024941020 0.008366830       0
## 6 -0.019013200 -0.02262504 -0.0192656200 -0.009272046 0.003130518       0
```

The correlations between the variables are low and similar to the what Amihud (2002) reports:

```
cor(table5$lnMILLIQm, table5$DEFm)
```

```
## [1] -0.05126374
```

```
cor(table5$lnMILLIQm, table5$TERMm)
```

```
## [1] 0.1933702
```

```
cor(table5$DEFm, table5$TERMm)
```

```
## [1] 0.08921768
```

Once we have calculated the values for $\mathrm{DEF_m}$ and $\mathrm{TERM_m}$, we can empirically test it as:

$$(RM-Rf)_m = g_0+g_1 lnMILLIQ_{m-1}+g_2 lnMILLIQ^U_m+g_3 JANDUM_m+a_1 DEF_{m-1}+a_2 TERM_{m-1}+u_y, \quad (11)$$

In this section we also compute the effects of illiquidity on the size portfolios like we did in Sections 3.2 and 3.3. We can code the model as:

```
table5Results <- lm((NYSE_Equal*100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm + lag(DEFm) + lag(TERMm), data
```

Notice that we lag() "DEFm" and "TERMm" within lm() because we want t=(m-1). Summarizing the regression results and using White's HC1 standard error estimator for the t-stats:

```
summary(table5Results)
```

```
##
## Call:
## lm(formula = (NYSE_Equal * 100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm +
##     lag(DEFm) + lag(TERMm), data = table5)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.1753  -2.4857   0.0466   2.9698  24.3117
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.4924     0.5980  -0.823   0.4107
## lnMILLIQm1    0.4015     0.2599   1.545   0.1232
## MonthlyRes   -4.3100     0.6790  -6.347 5.95e-10 ***
## JANDUMm       5.5807     0.8865   6.295 8.09e-10 ***
## lag(DEFm)     0.3917     0.5073   0.772   0.4405
## lag(TERMm)    0.2707     0.1546   1.751   0.0807 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.688 on 400 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.1511, Adjusted R-squared:  0.1405
## F-statistic: 14.24 on 5 and 400 DF,  p-value: 7.707e-13
```

```
coeftest(table5Results, vcov = vcovHC(table5Results, type = "HC1"))
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -0.49240    0.56141 -0.8771   0.38097
## lnMILLIQm1   0.40148    0.30828  1.3023   0.19355
## MonthlyRes  -4.30998    0.94034 -4.5834 6.123e-06 ***
## JANDUMm      5.58069    1.26235  4.4209 1.268e-05 ***
## lag(DEFm)    0.39171    0.45253  0.8656   0.38723
## lag(TERMm)   0.27075    0.15628  1.7324   0.08397 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
dwtest(table5Results)
```

```
##
##  Durbin-Watson test
```

```
##
## data:  table5Results
## DW = 1.9939, p-value = 0.4222
## alternative hypothesis: true autocorrelation is greater than 0
```

And building the appropriate table for the values of these estimations - using basically the same code as above.

```
depVars <- c("NYSE_Equal",
             "Decile_2",
             "Decile_4",
             "Decile_6",
             "Decile_8",
             "Decile_10")
table5DF <- c("Constant", "",
              "LnMILLIQ(y-1)", "",
              "LnMILLIQ(U,y)", "",
              "JANDUMm", "",
              "DEFm1", "",
              "TERMm1", "",
              "R^2",
              "D-W")
for (i in 1:6) {
  results <- lm((get(depVars[i])*100) ~ lnMILLIQm1 + MonthlyRes + JANDUMm + lag(DEFm) + lag(TERMm), data
  hc <- coeftest(results, vcov = vcovHC(results, type = "HC1"))
  coef<-results$coefficients
  vector <- round(as.vector(c(coef[1],
                              hc[1,3],
                              coef[2],
                              hc[2,3],
                              coef[3],
                              hc[3,3],
                              coef[4],
                              hc[4,3],
                              coef[5],
                              hc[5,3],
                              coef[6],
                              hc[6,3],
                              summary(results)$r.squared,
                              dwtest(results)$statistic) ),
                  digits=3)
  table5DF <- cbind(table5DF, vector)
}
table5Names <- c("", "RM-Rf","RSZ(2)-Rf", "RSZ(4)-Rf", "RSZ(6)-Rf", "RSZ(8)-Rf", "RSZ(10)-Rf")
table5DF <- as.data.frame(table5DF)
names(table5DF) <- table5Names
knitr::kable(table5DF)
```

	RM-Rf	RSZ(2)-Rf	RSZ(4)-Rf	RSZ(6)-Rf	RSZ(8)-Rf	RSZ(10)-Rf
Constant	-0.492	-0.594	-0.29	-0.475	-0.189	-0.252
	-0.877	-0.956	-0.489	-0.859	-0.344	-0.507
LnMILLIQ(y-1)	0.401	0.474	0.492	0.468	0.403	0.037
	1.302	1.304	1.56	1.533	1.396	0.142

	RM-Rf	RSZ(2)-Rf	RSZ(4)-Rf	RSZ(6)-Rf	RSZ(8)-Rf	RSZ(10)-Rf
LnMILLIQ(U,y)	-4.31	-5.212	-4.553	-4.166	-3.373	-1.933
	-4.583	-4.696	-4.594	-4.443	-3.992	-2.845
JANDUMm	5.581	8.495	5.767	4.545	3.166	1.368
	4.421	5.152	4.244	3.688	2.798	1.473
DEFm1	0.392	0.487	0.388	0.528	0.284	0.056
	0.866	0.934	0.79	1.184	0.64	0.135
TERMm1	0.271	0.184	0.21	0.248	0.25	0.339
	1.732	1.035	1.295	1.604	1.613	2.367
R^2	0.151	0.191	0.144	0.128	0.09	0.049
D-W	1.994	1.963	1.973	2.075	2.092	2.233

Conclusion

Contained above was the code for the replication of Amihud (2002) "Illiquidity and stock returns: cross-section and time-series effects." The code above generates results that confirm the original findings that"expected market illiquidity positively affects ex ante stock excess return, suggesting that expected stock excess return partly represents an illiquidity premium."