# Appendix

## Python Code:

1. # Author: C. Lee Fanzilli
2. # Date: March 2015
3. # CS/ECO ID Senior Thesis
4. # Advisors: Tomas Dvorak and Nick Webb
5. # Functions in this python script are taken from the walkthrough examples in Mining the Social Web 2nd Edition
6. #   and modeled for use in C. Lee Fanzilli's Senior Thesis
7. # Link to their open source code: https://github.com/ptwobrussell/Mining-the-Social-Web-2nd-Edition
8. **import** twitter
9. **import** sys
10. **import** time
11. **from** urllib2 **import** URLError
12. **from** httplib **import** BadStatusLine
13. **import** io, json
14. **import** json
15. **import** couchdb
16. **import** csv
17. **from** TweetStore **import** TweetStore
19. # Method for logging in to the Twitter API
20. # Local variables have the value # for security reasons
21. **def** oauth\_login():
22. CONSUMER\_KEY = '#'
23. CONSUMER\_SECRET = '#'
24. OAUTH\_TOKEN = '#'
25. OAUTH\_TOKEN\_SECRET = '#'
27. auth = twitter.oauth.OAuth(OAUTH\_TOKEN, OAUTH\_TOKEN\_SECRET,
28. CONSUMER\_KEY, CONSUMER\_SECRET)
30. twitter\_api = twitter.Twitter(auth=auth)
32. **return** twitter\_api

35. # Methods for saving and loading the twitter data as a .json format
36. **def** save\_json(filename, data):
37. with io.open("C:/Users/Lee/Desktop/Senior Thesis work/" + filename + ".json".format(filename),
38. 'w', encoding='utf-8') as f:
39. f.write(unicode(json.dumps(data, ensure\_ascii=False)))
41. **def** load\_json(filename):
42. with io.open("C:/Users/Lee/Desktop/Senior Thesis work/" + filename + ".json".format(filename),
43. encoding='utf-8') as f:
44. **return** f.read()
46. # Method for making twitter api calls
47. **def** make\_twitter\_request(twitter\_api\_func, max\_errors=10, \*args, \*\*kw):
49. # A nested helper function that handles common HTTPErrors. Return an updated
50. # value for wait\_period if the problem is a 500 level error. Block until the
51. # rate limit is reset if it's a rate limiting issue (429 error). Returns None
52. # for 401 and 404 errors, which requires special handling by the caller.
53. **def** handle\_twitter\_http\_error(e, wait\_period=2, sleep\_when\_rate\_limited=True):
55. **if** wait\_period > 3600: # Seconds
56. **print** >> sys.stderr, 'Too many retries. Quitting.'
57. **raise** e
59. # See https://dev.twitter.com/docs/error-codes-responses for common codes
61. **if** e.e.code == 401:
62. **print** >> sys.stderr, 'Encountered 401 Error (Not Authorized)'
63. **return** None
64. **elif** e.e.code == 404:
65. **print** >> sys.stderr, 'Encountered 404 Error (Not Found)'
66. **return** None
67. **elif** e.e.code == 429:
68. **print** >> sys.stderr, 'Encountered 429 Error (Rate Limit Exceeded)'
69. **if** sleep\_when\_rate\_limited:
70. **print** >> sys.stderr, "Retrying in 15 minutes...ZzZ..."
71. sys.stderr.flush()
72. time.sleep(60\*15 + 5)
73. **print** >> sys.stderr, '...ZzZ...Awake now and trying again.'
74. **return** 2
75. **else**:
76. **raise** e # Caller must handle the rate limiting issue
77. **elif** e.e.code **in** (500, 502, 503, 504):
78. **print** >> sys.stderr, 'Encountered %i Error. Retrying in %i seconds' % \
79. (e.e.code, wait\_period)
80. time.sleep(wait\_period)
81. wait\_period \*= 1.5
82. **return** wait\_period
83. **else**:
84. **raise** e
86. # End of nested helper function
88. wait\_period = 2
89. error\_count = 0
91. **while** True:
92. **try**:
93. **return** twitter\_api\_func(\*args, \*\*kw)
94. **except** twitter.api.TwitterHTTPError, e:
95. error\_count = 0
96. wait\_period = handle\_twitter\_http\_error(e, wait\_period)
97. **if** wait\_period **is** None:
98. **return**
99. **except** URLError, e:
100. error\_count += 1
101. **print** >> sys.stderr, "URLError encountered. Continuing."
102. **if** error\_count > max\_errors:
103. **print** >> sys.stderr, "Too many consecutive errors...bailing out."
104. **raise**
105. **except** BadStatusLine, e:
106. error\_count += 1
107. **print** >> sys.stderr, "BadStatusLine encountered. Continuing."
108. **if** error\_count > max\_errors:
109. **print** >> sys.stderr, "Too many consecutive errors...bailing out."
110. **raise**
112. # Method for harvesting tweets from a given user
113. **def** harvest\_user\_timeline(twitter\_api, screen\_name="@levie", user\_id=None, max\_results=1000):
115. **assert** (screen\_name != None) != (user\_id != None), \
116. "Must have screen\_name or user\_id, but not both"
118. kw = {  # Keyword args for the Twitter API call
119. 'count': 200,
120. 'trim\_user': 'true',
121. 'include\_rts' : 'true',
122. 'since\_id' : 1
123. }
125. **if** screen\_name:
126. kw['screen\_name'] = screen\_name
127. **else**:
128. kw['user\_id'] = user\_id
130. max\_pages = 16
131. results = []
133. tweets = make\_twitter\_request(twitter\_api.statuses.user\_timeline, \*\*kw)
135. **if** tweets **is** None: # 401 (Not Authorized) - Need to bail out on loop entry
136. tweets = []
138. results += tweets
140. **print** >> sys.stderr, 'Fetched %i tweets' % len(tweets)
142. page\_num = 1
144. # Many Twitter accounts have fewer than 200 tweets so you don't want to enter
145. # the loop and waste a precious request if max\_results = 200.
147. # Note: Analogous optimizations could be applied inside the loop to try and
148. # save requests. e.g. Don't make a third request if you have 287 tweets out of
149. # a possible 400 tweets after your second request. Twitter does do some
150. # post-filtering on censored and deleted tweets out of batches of 'count', though,
151. # so you can't strictly check for the number of results being 200. You might get
152. # back 198, for example, and still have many more tweets to go. If you have the
153. # total number of tweets for an account (by GET /users/lookup/), then you could
154. # simply use this value as a guide.
156. ##    print page\_num
157. ##    print max\_pages
158. ##    print kw['count']
159. ##    print kw['max\_id']
160. ##    print max\_results
162. **if** max\_results == kw['count']:
163. page\_num = max\_pages # Prevent loop entry
165. **while** page\_num < max\_pages **and** len(tweets) > 0 **and** len(results) < max\_results:
167. # Necessary for traversing the timeline in Twitter's v1.1 API:
168. # get the next query's max-id parameter to pass in.
169. # See https://dev.twitter.com/docs/working-with-timelines.
170. **print** 'got here'
172. kw['max\_id'] = min([ tweet['id'] **for** tweet **in** tweets]) - 1
174. **print** kw['max\_id']
175. **print** 'got out'
177. tweets = make\_twitter\_request(twitter\_api.statuses.user\_timeline, \*\*kw)
178. results += tweets
180. **print** >> sys.stderr, 'Fetched %i tweets' % (len(tweets),)
182. page\_num += 1
184. **print** >> sys.stderr, 'Done fetching tweets'
186. **return** results[:max\_results]
188. twitter\_api = oauth\_login()
190. ##datafile ='C:\\Python27\\tweeterHandles.csv'
191. ##storage = TweetStore('twitter\_data\_db-test')
192. ##with open(datafile, 'rb') as f:
193. ##    reader = csv.reader(f)
194. ##    ls = []
195. ##    for row in reader:
196. ##        q = row[0]
197. ##        ls.append(row[0])
198. ##
199. ##for item in ls:
200. ##    tweets = harvest\_user\_timeline(twitter\_api, screen\_name=item, \
201. ##                               max\_results=3200)
202. ##    for tweet in tweets:
203. ##        storage.save\_tweet(tweet)
204. ##
205. ##    print storage.count\_tweets()
206. ##
207. ####q = '@levie tweets'



212. tweets = harvest\_user\_timeline(twitter\_api, screen\_name="@reidhoffman", \
213. max\_results=3200)
215. storage = TweetStore('twitter\_data\_db-test')
217. #Method for storing tweets in CouchDB
218. **for** item **in** tweets:
219. **print** 'save tweet'
220. storage.save\_tweet(item)
221. **print** 'saved'
223. storage.count\_tweets()
224. # Author: C. Lee Fanzilli
225. # Date: March 2015
226. # CS/ECO ID Senior Thesis
227. # Advisors: Tomas Dvorak and Nick Webb
228. # Methods in this script are derived from NLTK Experiments
229. # by Doug Williams, open source code is available at http://nbviewer.ipython.org/github/williamsdoug/NLTK\_Experiments/blob/master/MPQA.ipynb
231. **import** couchdb
232. **import** sys
233. **import** nltk
234. **import** csv
236. # Logs user in to the CouchDB Server
237. couch = couchdb.Server()
238. db = couch['twitter\_data\_db-test']
239. map\_fun = '''''function(doc) {
240. var text
241. text = doc.text;
242. key = doc.created\_at;
243. emit(key,text)
244. }'''
246. trueNeutral = 1
247. results = db.query(map\_fun)
249. # Reads in the MPQA Subjectivity Lexicon
250. fswn=open('C:\Users\Lee\Desktop\subjectivity\_clues\_hltemnlp05\subjclueslen1-HLTEMNLP05.tff')
251. mpqaLines = [line **for** line **in** fswn.readlines()]
252. fswn.close


256. #ArrayList that will hold all dictionaries of word, POS & rating
257. mpqaArrList = []
259. #Cycle through each of the physical lines of the MPQA File
260. **for** line **in** mpqaLines:
261. #Split on the space to create the Key=Value with the = sign
262. data = line.strip().split(" ") #Result Ex: len=1
264. #Initialize Dictionary for each line that will hold the key=value
265. word\_dict = {}
266. #Split on the = to get the Key and Value split up
267. **for** d **in** data:
268. **try**:
269. keyValue = d.strip().split("=")
270. key = keyValue[0]
271. value = keyValue[1]
272. #Store the key value pair into a dictionary
273. word\_dict[key] = value
274. **except** IndexError:
275. **print** "Error"
277. #Add To The MPQA ArrayList
278. mpqaArrList.append(word\_dict)

281. # Method for calculating sentiment of a tweet about a certain stock/company
282. # then writes it to a CSV including Date, Text, and Sentiment, the TrueNeutral
283. # variable is only for analysis purposes and has no effect on the overall data.
284. **def** writeCSV():
285. polarDict = {}
286. **for** dict **in** mpqaArrList:
287. polarDict[dict['word1']] = dict['priorpolarity']
289. trueNeutral = True
290. tweetSentiment = 0
291. AMDList = ['AMD','amd','#amd','#AMD','@amd', '@AMD']
292. aaplList = ['aapl', 'AAPL', 'APPLE', 'apple', '#aapl', '#AAPL', 'Apple']
293. file = open('BBRYSentiment.csv', 'w')
294. fields = ('DatePosted', 'Tweet', 'TweetSentiment', 'TrueNeutral')
295. wr = csv.DictWriter(file, fieldnames=fields, lineterminator = '\n')
296. wr.writeheader()
297. **for** row **in** results:
298. sentence = row.value
299. sentence = sentence.encode('ascii','ignore')
300. #if any(word in sentence.split() for word in aaplList):
301. #print sentence.split()
302. **if**  'BBRY' **in** sentence **or** 'bbry' **in** sentence **or**  '#bbry' **in** sentence **or**  '#BBRY' **in** sentence **or**  '@BlackBerry' **in** sentence **or** '#BlackBerry' **in** sentence **or** '#blackberry' **in** sentence:
303. **for** word **in** sentence.split():
304. **for** key, value **in** polarDict.items():
305. #If a match is found
306. **if** word.upper() == key.upper():
307. **if** value == 'positive':
308. tweetSentiment = tweetSentiment + 1
309. trueNeutral = False
310. #print dict['priorpolarity'] + " "+ word
311. **if** value == 'negative':
312. tweetSentiment = tweetSentiment - 1
313. trueNeutral = False
314. #print dict['priorpolarity'] + " "+ word
315. date = row.key
316. date = date.encode('ascii','ignore')
317. #print tweetSentiment
318. wr.writerow({'DatePosted':date, 'Tweet':sentence, 'TweetSentiment':tweetSentiment, 'TrueNeutral':trueNeutral})
319. tweetSentiment = 0
320. #print (row.key, row.value)
322. file.close()


326. **def** main():
328. writeCSV()

## RStudio Code:

# Authors: C. Lee Fanzilli and T. Dvorak

# Date: March 2015

# CS/ECO ID Senior Thesis

# Advisors: Tomas Dvorak and Nick Webb

# Script for calculating future returns

library(dplyr) #new package for manipulating datasets

library(tidyr) #new package for reshaping datasets, requires R3.1

library(lubridate) #needed for rounding dates

library(stringr) #for manupulating strings

library(ggplot2)

library(reshape2)

library(scales)

library(erer)

library(PerformanceAnalytics)

library(stargazer)

library(gridExtra)

# read in sentiment scores for a stock based on our python script output

t <- read.csv("BBRYSentiment.csv")

t <- select(t, DatePosted, Tweet, TweetSentiment)

t <- t %>%

separate(DatePosted, into = c("dow", "month", "day", "time", "x", "year"), sep = " ")

# extract different parts of the date and time so we can organize the data uniformly

t$datetime <- as.datesubstr(t$DatePosted, start=2,stop=100) #remove the first character in logdate because it is "["

t$datestring <- paste(t$day,t$month,t$year,t$time)

t$datetime <- as.POSIXct(t$datestring, format="%d %b %Y %H:%M:%S", tz="UTC") #turn logdate into time variable

t$datetime2 <-as.Date(t$datetime)

#t$correctdate <- format(t$datetime, tz="America/Atikokan")

# converts date time to be in the correct time zone

t$datetime <- as.POSIXct(format(t$datetime, tz="America/Atikokan"))

t$hour <- hour(t$datetime)

t$minutes <-minute(t$datetime)

# normalize tweets posted before open to be posted at open and before close to be at close

t$addDay <- ifelse(t$hour > 16, 1, 0)

t$datetime1.5 <- t$datetime + t$addDay \* 60 \* 60 \* 24

# we round the time of tweets here to open or close

t$timeRounded <- ifelse(t$hour > 16 | (t$hour < 10 & t$minutes < 30), "09:30:00", "16:00:00")

t$datetime2 <- ifelse(t$) #paste it together and convert it to posixct format,

# extract the year month and day from our datetime variable

t$year <- year(t$datetime1.5)

t$month <- month(t$datetime1.5)

t$day <- day(t$datetime1.5)

t$datetime2 <- paste(t$year, t$month, t$day, t$timeRounded)

t$datetime3 <- as.POSIXct(t$datetime2, format = "%Y %m %d %H:%M:%S")

# finish formatting date time to be what we want

t$datetime <- t$datetime3

# load in daily stock price information and split it to be time series of open and close times

p<-read.csv("C:/Users/BBRY Daily.csv")

p$Open2 <- p$Open\*(p$Adj.Close/p$Close)

#do the same for close

p$Close2 <- p$Close\*(p$Adj.Close/p$Open)

p$time <-"09:30:00"

p$timeC <- "16:00:00"

#paste it with date and create datetime variable

p$datetimeC <- paste(p$Date, p$timeC)

p$datetime <- paste(p$Date, p$time)

p$datetimeC <- as.POSIXct(p$datetime, format="%m/%d/%Y %H:%M:%S") #turn logdate into time variable

p$datetime <- as.POSIXct(p$datetime, format="%m/%d/%Y %H:%M:%S") #turn logdate into time variable

# select the information we want from price data

# separate open price and close price on a given day into two separate instances

p2 <- select(p, datetime, Open2)

p2$price <- p2$Open2

p2 <- select(p2, datetime, price)

p3 <-select(p, datetimeC, Close2)

p3$price <- p3$Close2

p3$datetime <- p3$datetimeC

p3 <- select(p3, datetime, price)

# combine open and close prices and arrange by date and time

prices <- rbind(p2, p3)

prices <- arrange(prices, datetime)

# merge prices and sentiment datasets into one dataframe on date and time

data <- merge(prices, t, by="datetime")

data <- select(data, datetime3, price, TweetSentiment, Tweet)

#probably the best way is to create two dataframes and them stack them on top so that this dataframe has these variables

# calculate future returns

price1 <- data$price[3:length(data$price)]

# shift the price data

#b <- a[-nrow(a),]

data2 <- head(data, -2)

data2$price1 <- price1

price2 <- tail(price1, -1)

data2$returns1 <- ((data2$price1 - data2$price) / data2$price)

data2$Returns <- data2$returns1

# not used in this project but would be when calculating abnormal returns

s <- read.csv("C:/Users/SnP 500.csv")

s$time <-"09:30:00"

s$timeC <- "16:00:00"

#paste it with date and create datetime variable

#s$Date <- as.POSIXct(s$Date, format="%d/%m/%Y %H:%M:%S") #turn logdate into time variable

s$datetimeC <- paste(s$Date, s$timeC)

s$datetime <- paste(s$Date, s$time)

s$datetimeC <- as.POSIXct(as.character(s$datetimeC), format="%Y-%m-%d %H:%M:%S") #turn logdate into time variable

s$datetime <- as.POSIXct(s$datetime, format="%Y-%m-%d %H:%M:%S") #turn logdate into time variable

# better looking scatterplot for regression

out <- ggplot(data2, aes(x=TweetSentiment, y=Returns)) +

geom\_point(shape=1) + # Use hollow circles

geom\_smooth(method=lm) +

xlab("Tweet Sentiment") +

ylab("Returns") +

ggtitle("Google")

# linear regression model of our dataset and scatterplot

datamodel = lm(data2$Returns ~ data2$TweetSentiment)

plot(data2$returnsx ~ data2$TweetSentiment, data=data2)

abline(lm(data2$returnsx ~ data2$TweetSentiment, data=data2))

# Authors: C. Lee Fanzilli and T. Dvorak

# Date: March 2015

# CS/ECO ID Senior Thesis

# Advisors: Tomas Dvorak and Nick Webb

# Script for calculating contemporaneous returns and past stock returns

library(dplyr) #new package for manipulating datasets

library(tidyr) #new package for reshaping datasets, requires R3.1

library(lubridate) #needed for rounding dates

library(stringr) #for manupulating strings

library(ggplot2)

library(reshape2)

library(scales)

library(erer)

library(PerformanceAnalytics)

library(stargazer)

# read in sentiment scores for a stock based on our python script output

t <- read.csv("AMDSentiment.csv")

t <- select(t, DatePosted, Tweet, TweetSentiment)

t <- t %>%

separate(DatePosted, into = c("dow", "month", "day", "time", "x", "year"), sep = " ")

# extract different parts of the date and time so we can organize the data uniformly

t$datetime <- as.datesubstr(t$DatePosted, start=2,stop=100) #remove the first character in logdate because it is "["

t$datestring <- paste(t$day,t$month,t$year,t$time)

t$datetime <- as.POSIXct(t$datestring, format="%d %b %Y %H:%M:%S", tz="UTC") #turn logdate into time variable

t$datetime2 <-as.Date(t$datetime)

#t$correctdate <- format(t$datetime, tz="America/Atikokan")

# converts date time to be in the correct time zone

t$datetime <- as.POSIXct(format(t$datetime, tz="America/Atikokan"))

t$hour <- hour(t$datetime)

t$minutes <-minute(t$datetime)

# normalize tweets posted before open to be posted at open and before close to be at close

t$addDay <- ifelse(t$hour > 16, 1, 0)

t$datetime1.5 <- t$datetime + t$addDay \* 60 \* 60 \* 24

# we round the time when a tweet is posted to open or close

t$timeRounded <- ifelse(t$hour > 16 | (t$hour < 10 & t$minutes < 30), "09:30:00", "16:00:00")

t$datetime2 <- ifelse(t$) #paste it together and convert it to posixct format,

#and add date to somemthing. look up lubridate for adding days

# extract the year month and day from our datetime variable

t$year <- year(t$datetime1.5)

t$month <- month(t$datetime1.5)

t$day <- day(t$datetime1.5)

t$datetime2 <- paste(t$year, t$month, t$day, t$timeRounded)

t$datetime3 <- as.POSIXct(t$datetime2, format = "%Y %m %d %H:%M:%S")

# finish formatting date time to be what we want

t$datetime <- t$datetime3

p<-read.csv("C:/Users/Lee/Desktop/Senior Thesis work/NYSE Datasets/AMD Daily.csv")

p$Open2 <- p$Open\*(p$Adj.Close/p$Close)

# calculate adjusted close

p$Close2 <- p$Close\*(p$Adj.Close/p$Open)

p$time <-"09:30:00"

p$timeC <- "16:00:00"

#paste it with date and create datetime variable

p$datetimeC <- paste(p$Date, p$timeC)

p$datetime <- paste(p$Date, p$time)

p$datetimeC <- as.POSIXct(p$datetimeC, format="%m/%d/%Y %H:%M:%S") #turn logdate into time variable

p$datetime <- as.POSIXct(p$datetime, format="%m/%d/%Y %H:%M:%S") #turn logdate into time variable

# select open prices

p2 <- select(p, datetime, Open2)

p2$price <- p2$Open2

p2 <- select(p2, datetime, price)

# select closing prices

p3 <-select(p, datetimeC, Close2)

p3$price <- p3$Close2

p3$datetime <- p3$datetimeC

p3 <- select(p3, datetime, price)

# merge open and close prices and arrange by date

prices <- rbind(p2, p3)

prices <- arrange(prices, datetime)

# merge price and sentiment data by date

data <- merge(prices, t, by="datetime")

data <- select(data, datetime3, price, TweetSentiment, Tweet)

# organize the data to calculate contemporaneous returns

price1 <- data$price[2:length(data$price)]

#b <- a[-nrow(a),]

data <- data[-nrow(data),]

data$price1 <- price1

#price2 <- tail(price1, -1)

data$returns <- ((data$price1 - data$price) / data$price)

# better looking scatterplot for regression

out3 <- ggplot(data, aes(x=TweetSentiment, y=returns)) +

geom\_point(shape=1) + # Use hollow circles

geom\_smooth(method=lm) +

xlab("Tweet Sentiment") +

ylab("Returns") +

ggtitle("Google")

# linear regression model of our dataset and scatterplot

datamodel = lm(data$returns ~ data$TweetSentiment)

plot(data$returns ~ data$TweetSentiment, data=data)

abline(lm(data$returns ~ data$TweetSentiment, data=data))

newdata <- data

newReturn <- newdata$returns

#b <- a[-nrow(a),]

#data <- data[-nrow(data),]

# organize the data to calculate past returns

newReturn <- head(newReturn, -2)

newdata <- head(newdata, -2)

newdata$TweetSentiment <- newSenti

newdata$Return <- newReturn

#data$price1 <- price1

#price2 <- tail(price1, -1)

# linear regression model of our dataset and scatterplot

datamodel1 = lm(newdata$Return ~ newdata$TweetSentiment)

plot(newdata$Return ~ newdata$TweetSentiment, data=newdata)

abline(lm(newdata$Return ~ newdata$TweetSentiment, data=newdata))

# better looking scatterplot for regression

out2 <-ggplot(newdata, aes(x=TweetSentiment, y=Return)) +

geom\_point(shape=1) + # Use hollow circles

geom\_smooth(method=lm) +

xlab("Tweet Sentiment") +

ylab("Returns") +

ggtitle("Microsoft")

# function for organizing multiple scatter plots in one window

# open source code for this function is available at

# http://www.cookbook-r.com/Graphs/Multiple\_graphs\_on\_one\_page\_(ggplot2)/

multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

require(grid)

# Make a list from the ... arguments and plotlist

plots <- c(list(...), plotlist)

numPlots = length(plots)

# If layout is NULL, then use 'cols' to determine layout

if (is.null(layout)) {

# Make the panel

# ncol: Number of columns of plots

# nrow: Number of rows needed, calculated from # of cols

layout <- matrix(seq(1, cols \* ceiling(numPlots/cols)),

ncol = cols, nrow = ceiling(numPlots/cols))

}

if (numPlots==1) {

print(plots[[1]])

} else {

# Set up the page

grid.newpage()

pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

# Make each plot, in the correct location

for (i in 1:numPlots) {

# Get the i,j matrix positions of the regions that contain this subplot

matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,

layout.pos.col = matchidx$col))

}

} }