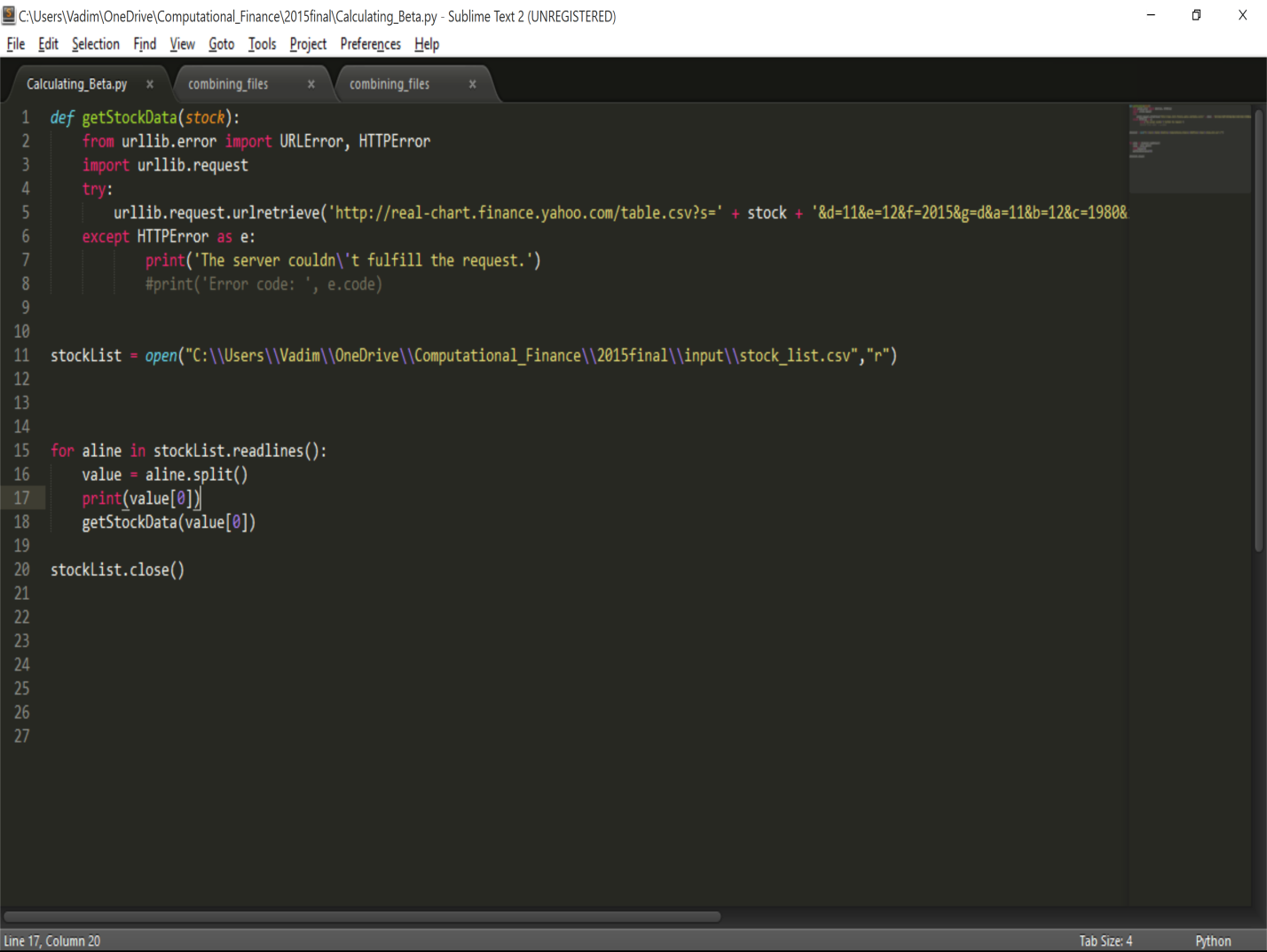
Vadim Serebrinskiy

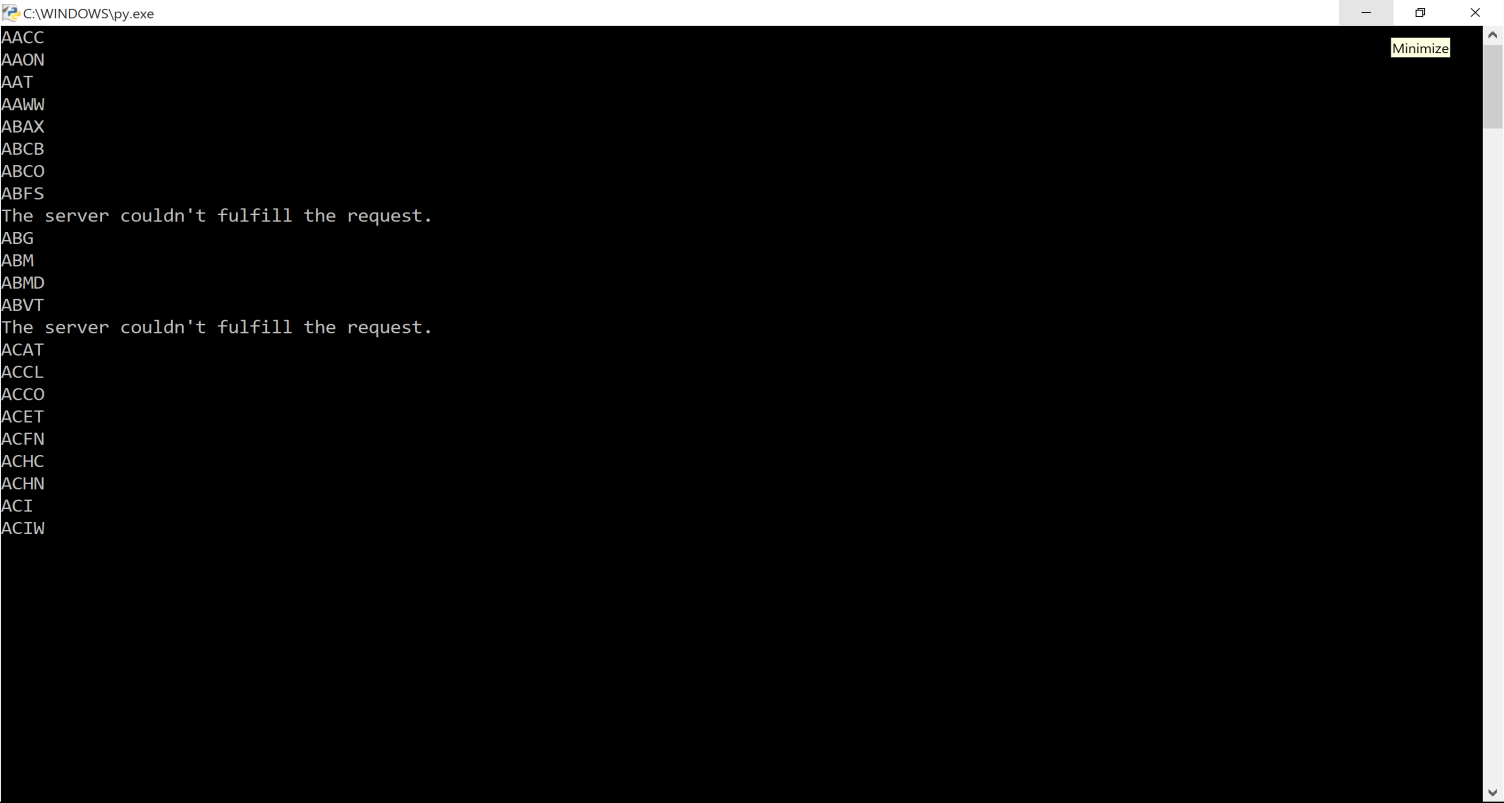
SQL Final Project

December 17, 2015

This SQL project has certainly been a great learning experience mainly about the possible errors that may occur, especially with volatile Microsoft software. I started off with a motivation to learn a new language while performing this project, so I chose to incorporate Python as my data extraction tool. My first step was acquiring the list of stock symbols. I went to the yahoo finance website and noticed that they have an export data functionality that also has a URL link to it. I used Python to run a URL retrieve function that first obtained the list of stock symbols from one excel file and appended those stock symbols to the URL string in order to generate 2000 data files in a location that I selected on my disk. This python function was in a separate file and can be seen here (I took a screen shot to make it easier to read)

So as we can see I created a function that takes in a stock (symbol) value and makes a URL request from yahoo. I also noticed on my command prompt that some symbols aren’t traded anymore.



After I obtained 2000 files in a specified folder location I would need to write another python script to basically have two for-loops in order to first traverse the files then read each line in each file and build them on top of each other to create one general csv file of all stock symbols and their values. This code is a little long so I can’t print screen the whole thing.

import glob

import os

from datetime import datetime

f = glob.glob("C:\\Users\\Vadim\\OneDrive\\Computational\_Finance\\2015final\\data\\\*.csv")

totallinecount=0

filelinecount=0

with open("C:\\Users\\Vadim\\OneDrive\\Computational\_Finance\\2015final\\combinedquotes\\combinedquotes.csv", 'w') as outfile:

for fname in f:

print("working on file " + fname)

with open(fname) as infile:

stock = os.path.basename(fname).split(".")

filelinecount=0

for line in infile:

totallinecount+=1

filelinecount+=1

line = line.replace(',','\t')

elementsInLine = line.split("\t")

# verify that data contains 7 elements, if not it is a bad line from Yahoo

if len(elementsInLine)==7:

# if this the very first line, use for the header line and never again

if totallinecount == 1:

line = 'Stock' +'\t' + line

outfile.write(line)

# if this is a first line in a file, skip it, we already have header line from the very first file

if filelinecount > 1:

# replace GSPC with SNP500 since Yahoo uses GSPC for SP

if 'GSPC' in stock[0]:

#print ("found line " + line)

quote='SNP500'

# Yahoo is using diff date format for SP500 mm/dd/yyyy and we need to make it yyyy-mm-dd for bcp to work in sql

dateField = elementsInLine[0]

#dateField = dateField[-4:] + '-' + dateField[:2] + '-' + dateField[3:-5]

str=dateField

fmt="%M/%d/%Y"

date\_object = datetime.strptime(str,fmt)

line = date\_object.strftime("%Y-%M-%d") + line[len(elementsInLine[0]):]

else:

quote=stock[0]

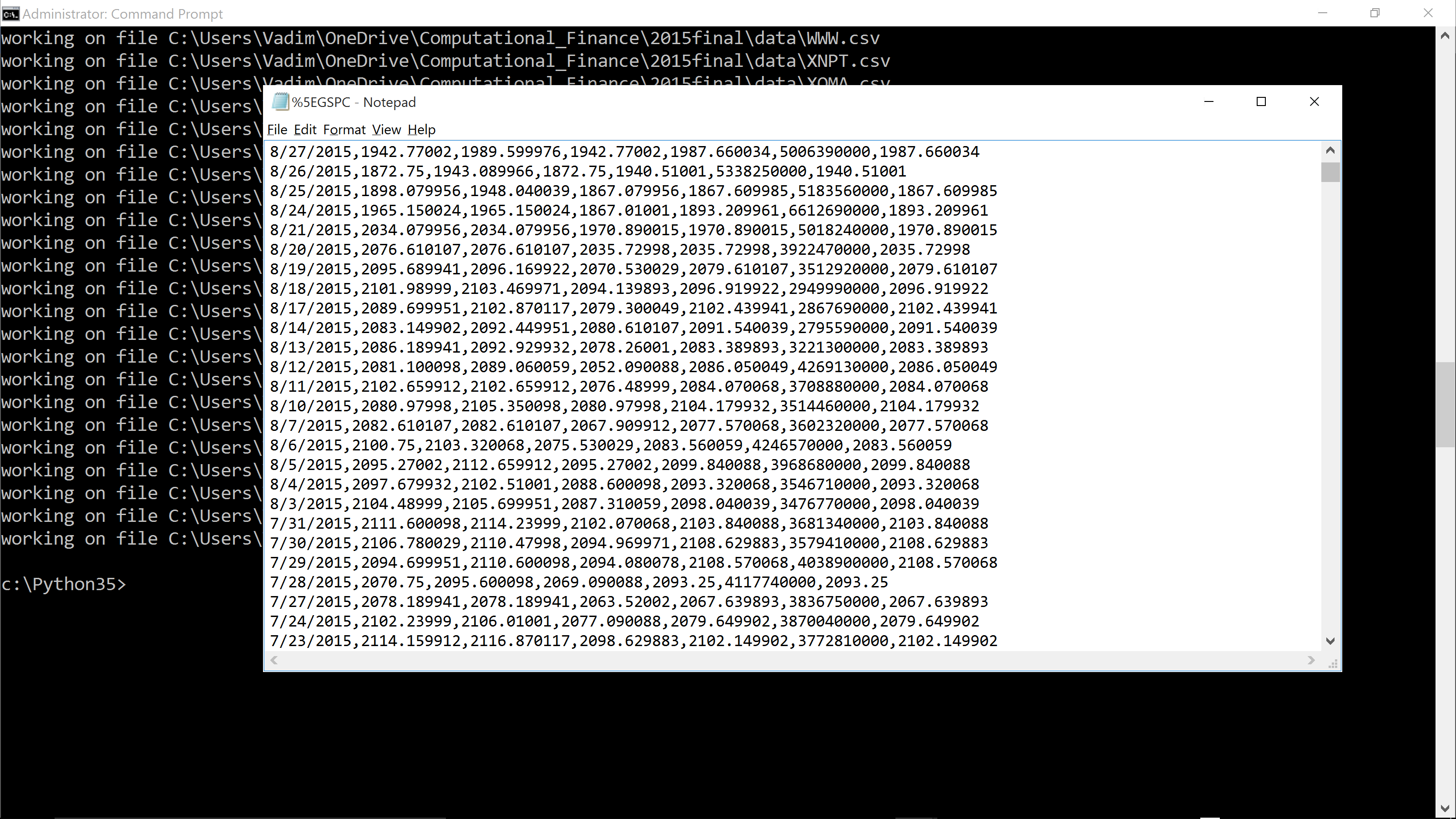
#print ("wrtiting line " + line)

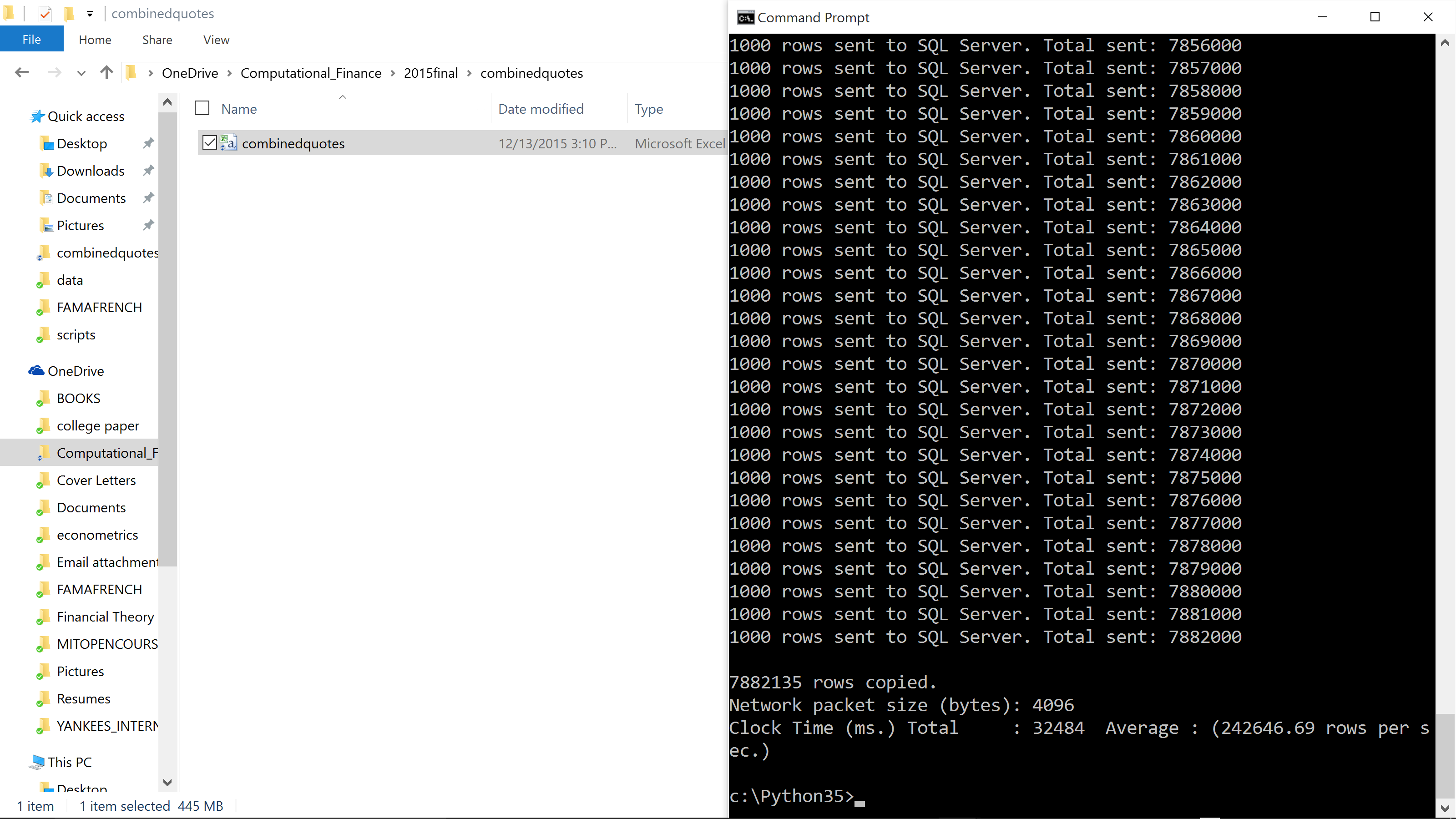
outfile.write(quote + '\t' + line)

else:

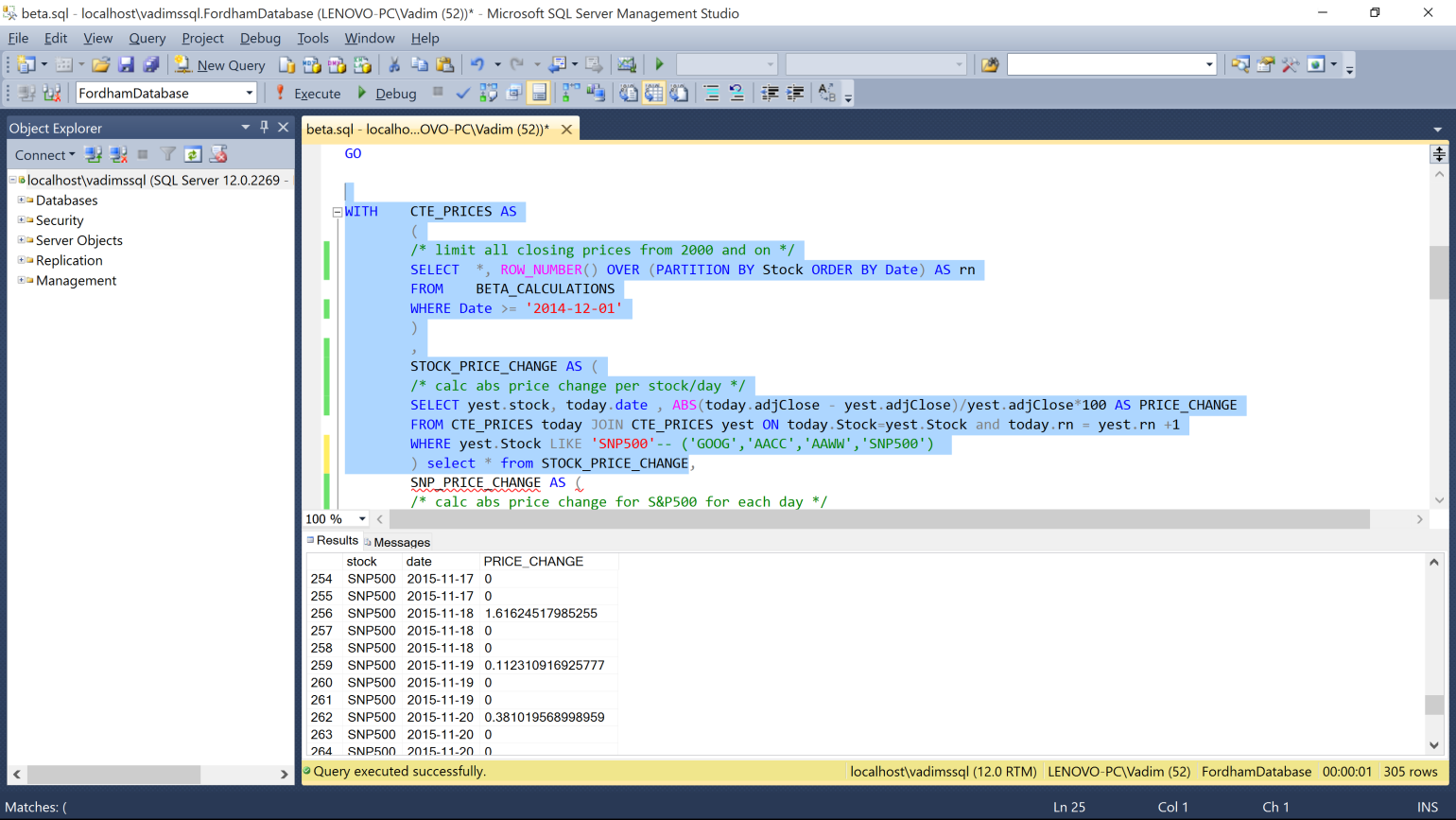
print("Bad data in this row, less than required 7 elements, row position") #+ str(filelinecount) + ", " + line)

This code ran well in command line but the problems really began and sort of kept going back to the BCP procedure which really doesn’t seem like the best way to import data at this point. For some reason it doesn’t like the commas and prefers tab delimited entries, which you can see I was replacing at the top of the above code. I did this to avoid having to do a format file for the BCP. But before I went back to the BCP method I figured I would give the Import Wizard in SQL a chance but it also gave me issues. After digging further I noticed that some stocks have missing values! But it not only has missing values, SQL tries to help but ends up worsening the issue by appending the next line starting from the position where the missing entry is located. So I would end up having a stock symbol ABX in the open price column for ABVC (for example). So I went back to the python file combining code to basically split each line into an array and say that if it doesn’t have 7 elements (not 8 yet) then we throw it out. Notice I didn’t say 8 elements because the files don’t originally come with the stock name on their left hand column; I had to append that in the end. But unfortunately the problems didn’t end there. For some reason the S&P 500 values (dates) didn’t match the proper SQL format, while all of the other stocks did. So BCP kept telling me to reformat it. I first tried to do basic string concatenation but I realized some dates have less elements (i.e April = 4 aka 1 digit and December = 12 aka 2 digits). So I knew there had to be an easier way and found a string formatter function online for python after doing some research. I used this function to finally get the proper date format which was ultimately accepted by the BCP command. Here is a screen shot of the improper format that caused the issue:

The SNP500 not only had improper date format but also used slashes instead of dashes.

After the BCP import I started doing my SQL query work on the newly populated BETA\_CALCULATIONS table. I basically calculated the entire covariance using SQL but something still wasn’t quite right… I tried to check the code by testing market beta against itself (which should come out to 1), but I got an answer like 0.97. I investigated further and found that on my final join, the count was not right because it contained duplicates. The reason it contained duplicates is because my initial BCP attempts created copies in S&P500 rows. In order to get rid of these duplicates I had to resort to creating a temporary table of dupes and deleting them from the S&P 500 data set. I also used unique indexing to make sure I deleted the right rows (since the temporary table had duplicates). I checked for these duplicates in general by using the row number property and noticing which rows had an rn>1 so that would tell me where the rows had unnecessary and empty copies.

As we can see on the next screen shot, we have dates on the SNP500 table with multiple rows of the same dates. Of course the duplicates have no price change because they aren’t supposed to exist in the first place.

So after resolving this final issue I ran my overall query :

USE [FordhamDatabase]

GO

/\*\*\*\*\*\* Object: Table [dbo].[BETA\_CALCULATIONS] Script Date: 12/13/2015 12:19:51 AM \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

DROP TABLE [dbo].[BETA\_CALCULATIONS]

CREATE TABLE [dbo].[BETA\_CALCULATIONS](

[Stock] [varchar](50) NULL,

[Date] [date] NULL,

[Open] [real] NULL,

[High] [real] NULL,

[Low] [real] NULL,

[Close] [real] NULL,

[Volume] bigint NULL,

[adjClose] [real] NULL

) ON [PRIMARY]

GO

create unique index idx1 on BETA\_CALCULATIONS(Stock,[Date])

select \* into #tmp from (

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY Stock, Date ORDER BY Stock, Date) AS rn

FROM BETA\_CALCULATIONS

) a

where rn > 1

select distinct Stock, Date from #tmp

delete b from BETA\_CALCULATIONS b join #tmp on b.Stock=#tmp.Stock and b.Date=#tmp.Date

insert into BETA\_CALCULATIONS

select

distinct

[Stock],

[Date],

[Open],

[High],

[Low] ,

[Close],

[Volume],

[adjClose]

from #tmp

;

WITH CTE\_PRICES AS

(

/\* limit all closing prices from 2000 and on \*/

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY Stock ORDER BY Date) AS rn

FROM BETA\_CALCULATIONS

WHERE Date >= '2014-12-01'

)

,

STOCK\_PRICE\_CHANGE AS (

/\* calc abs price change per stock/day \*/

SELECT yest.stock, today.date , ABS(today.adjClose - yest.adjClose)/yest.adjClose\*100 AS PRICE\_CHANGE

FROM CTE\_PRICES today JOIN CTE\_PRICES yest ON today.Stock=yest.Stock and today.rn = yest.rn +1

WHERE yest.Stock IN ('AACC', 'AAON','AAT', 'AAWW','ABAX','ABCB','ABCO','ABFS', 'ABG', 'ABM', 'ABMD','ABVT','ACAT','ACCL','ACCO' 'ACET','ACFN','ACHC' 'ACHN','ACI', 'ACIW','ACLS','ACO', 'ACOM','ACOR' 'ACRE' 'ACTG' 'ACUR','ACW', 'ACXM' 'ADC', 'ADES', 'ADGE','ADNC','GOOG','AACC','AAWW','SNP500', 'AMZN','MSFT')

) ,

SNP\_PRICE\_CHANGE AS (

/\* calc abs price change for S&P500 for each day \*/

SELECT yest.stock, today.date , ABS(today.adjClose - yest.adjClose)/yest.adjClose\*100 AS PRICE\_CHANGE

FROM CTE\_PRICES today JOIN CTE\_PRICES yest ON today.Stock=yest.Stock and today.rn = yest.rn +1

WHERE yest.Stock='SNP500'

)

,

AVG\_SP AS (SELECT AVG(PRICE\_CHANGE) as avg\_chg FROM SNP\_PRICE\_CHANGE) ,

SNP\_VAR AS (

/\* calc var for S&P500 for the whole date range \*/

SELECT SUM((snp.PRICE\_CHANGE - AVG\_SP.avg\_chg)\*(snp.PRICE\_CHANGE - avg\_sp.avg\_chg))/(COUNT(\*)-1)

AS variance, COUNT(\*) AS SNP\_DAY\_COUNT FROM SNP\_PRICE\_CHANGE snp , AVG\_SP

)

,

STOCK\_COVAR AS (

/\* for each stock calc covar as

SUM of

(index price change - avg of index price change)

times (stock price change - avg of stock price change)

divided by stock price number of observations

\*/

SELECT st.Stock,

SUM

(

(snp.PRICE\_CHANGE - avg\_sp.avg\_chg ) \*

(st.PRICE\_CHANGE - avg\_st.avg\_chg)

)

/

(MAX(avg\_st.STOCK\_COUNT)-1) as covariance,

MAX(avg\_st.STOCK\_COUNT) AS DAY\_COUNT

--avg\_sp.avg\_chg as avg\_snp\_price\_change, snp.PRICE\_CHANGE as snp\_price\_change, snp.Date as snp\_date, st.date, st.PRICE\_CHANGE, avg\_st.avg\_chg

FROM AVG\_SP,

STOCK\_PRICE\_CHANGE st JOIN

SNP\_PRICE\_CHANGE snp

ON snp.Date=st.Date

JOIN

( SELECT Stock, AVG(PRICE\_CHANGE) as avg\_chg, COUNT(\*) AS STOCK\_COUNT FROM STOCK\_PRICE\_CHANGE GROUP BY Stock) avg\_st

ON st.Stock=avg\_st.Stock

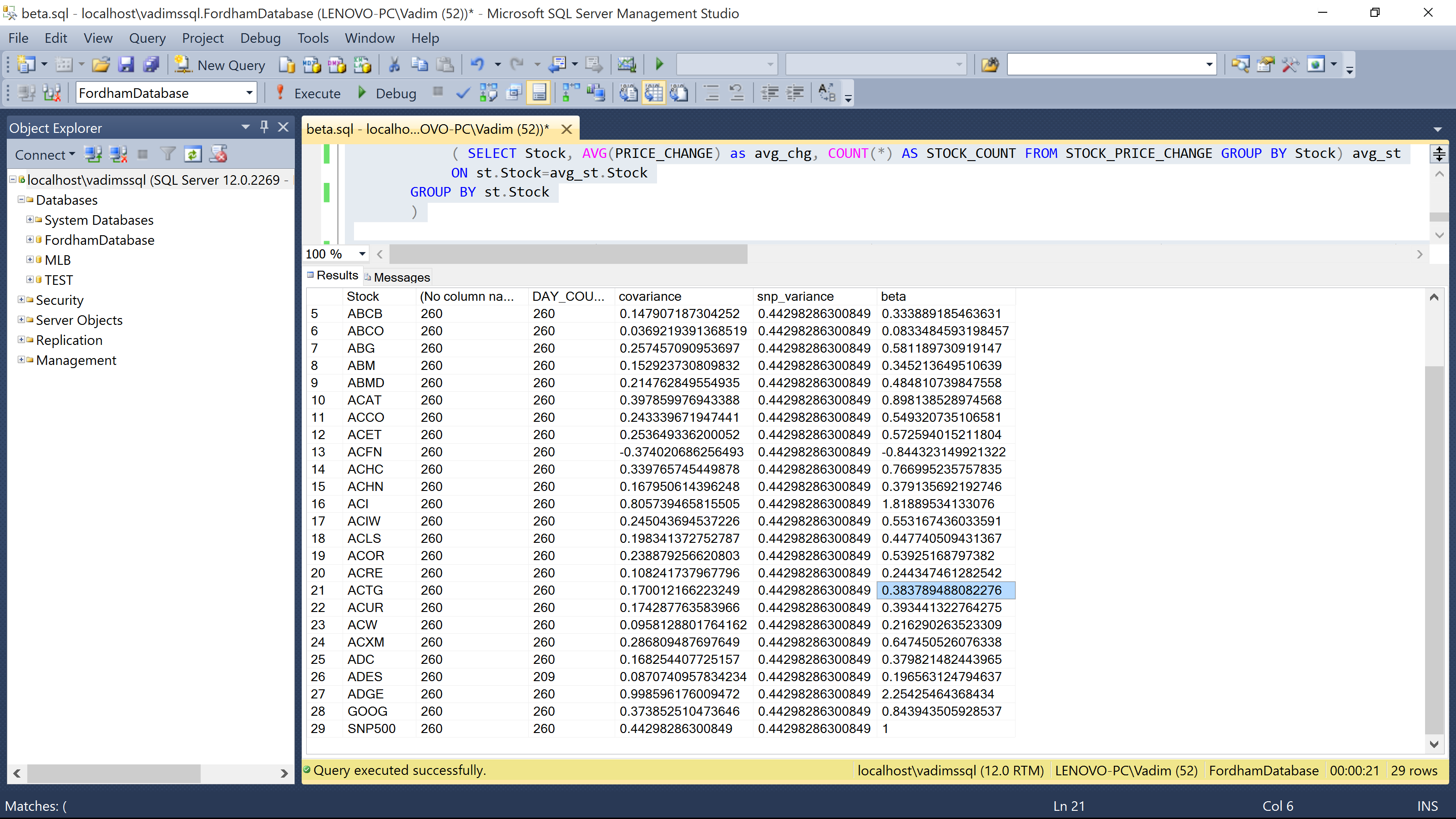
GROUP BY st.Stock

)

SELECT stcovar.Stock, (SELECT SNP\_DAY\_COUNT from SNP\_VAR), DAY\_COUNT, stcovar.covariance , (SELECT variance from SNP\_VAR) as snp\_variance, stcovar.covariance / (SELECT variance from SNP\_VAR) as beta

FROM STOCK\_COVAR stcovar

For the sake of saving time I ran a simple test for a handful of stocks and the SNP500 to make sure my results are rational using the LIKE clause. I think my resulting betas are reasonable because they aren’t huge (i.e between -1 and 1.5) and the SNP500 beta is 1 which makes sense because the market should be correlated 100 percent with itself. Finding the beta on the S&P is like dividing market variance by the market variance so it should be 1. The resulting output set appeared as :



In conclusion I would definitely say that I learned a great deal about coding and table manipulation during this project. Looking back I think I would’ve chosen to pull the data back out of SQL and manipulate it in a different environment and then finally send it back into the database as a new table.