# Technical and Time Series Analysis in KDB+ q (and Python Pandas) Guidelines

By Li Li (June 2018)

In this project, we implement the technical and time series analysis mainly by qsql, but sometimes by Python Pandas.

# Read in csv Files

A comma separated values (csv) file is the most common use file format for data import/export. Data processing packages all have easy-to use functions to read and write csv files.

A csv reading method executes three functions:

1. Locate the right csv file;
2. Import data in columns into internal data table;
3. Format each column.

Table below summarizes these features for Python Pandas and KDB+ q:

|  |  |  |
| --- | --- | --- |
| Method/Feature | Python Pandas | KDB+ q |
| Read in csv file | Pd.read\_csv() | t: (“*informat*”;enlist “,”) 0: `*filename.csv* |
| Path and file name | my\_dir='C:\\Users\\Li Li\\Documents\\data\\antonia2018\\'  filename=my\_dir+’t.csv’ | `:../qdata2018/t.csv |
| First line includes header | Header parameter. For example, Header=None to indicate no header in the csv file. | 0: |
| Informat | Converters parameter | For example, “SDFFFFI” for symbol, date,open,high,low,close,volume for stock price data. S-string,D-date, F-float,I-intger. |
| Missing value | na\_value parameter. |  |

In Python Pandas, a csv file can be read by the pd.read\_csv method:

Import pandas as pd

df=pd.read\_csv(my\_dir+'month\\'+'t\_out2.csv',na\_values=[],converters={'date':pd.to\_datetime})

A converter is like the “informat” in SAS. It can also be a user defined function. Converters for more than one column can be separated by a comma (,) within the brackets {}.

Below is an example:

Anaconda 5.1.0 interactive window [PTVS 3.0.17320.1-15.0]

Type $help for a list of commands.

>>> import pandas as pd

>>> df1=pd.read\_csv('C://Users//Li Li//Documents//data//antonia2018//stocks//test0.csv')

>>> df1

date symbol o h l c v

0 2018-04-13 00:00:00 SP-500 2676.90 2680.26 2645.05 2656.30 7183.0

1 2018-04-16 00:00:00 SP-500 2670.10 2686.49 2665.16 2677.84 7481.0

2 2018-04-17 00:00:00 SP-500 2692.74 2713.34 2692.05 2706.39 7199.0

3 2018-04-18 00:00:00 SP-500 2710.11 2717.49 2703.63 2708.64 7702.0

4 2018-04-19 00:00:00 SP-500 2701.16 2702.84 2681.90 2693.13 7471.0

……

To read in a csv file in q, a sample command line is:

t1:("SDFFFFI";enlist ",") 0: `:../qdata2018/dd\_data.csv

In this case, the csv file “dd\_data.csv” is located in folder c:\qdata2018\. The first line of the data includes a header. The columns are (Symbol, date, open, high, low, close, volume). The data types are string, date, float, float, float, float and integer.

# Data Selection - Pandas

|  |  |  |
| --- | --- | --- |
| Operation | SQL | Python Pandas |
| Data | Table\_a | df |
| Select columns | SELECT TOP 5 Var1, Var2 FROM Table\_a | df[‘Var1’,’Var2’] |
| Select rows | SELECT TOP 5 Var1 FROM Table\_a | df.iloc[brow:erow] #select rows from brow (including row #=brow) to less than erow.  df.head(n) # select the top n rows  df.tail(n) # select the bottom n rows |
| Where | SELECT TOP 5 \* FROM Table\_a WHERE Var1=’x’ | df[df[‘Var1’]==’x’]  or  df[df[‘Symbol’].str.contains(‘x’)] |
| Missing records | SELECT \* FROM Table\_a WHERE Var1 IS NULL | df[df[‘Var1’].isna()] |
| SELECT DISTINCT | SELECT DISTINCT Var1 FROM Table\_a | df.Var1.unique() # returns an array |
| Loop through array | N/A. The SQL table is a set which does not allowed to be looped through. | for index, value in enumerate(df.Var1.unique():  do something |
| Group by | SELECT Var1, COUNT(\*) AS N FROM Table\_a GROUP BY Var1 | df.groupby(‘Var1’).size() |
| Join | SELECT a.Var1,b.Var2 FROM Table\_a a INNER JOIN Table\_b b ON a.Var\_1=b.Var1 | pd.merge(dfa,dfb,ON=’Var1’) |
| Union | SELECT Var1 FROM Table\_a  UNION  SELECT Var1 FROM Table\_b | Pd.concate([dfa,dfb]).drop\_duplicates |

The Python pandas table is a list which allows for duplicates and looping through/ordering. By setting up the appropriate index, the searching job is easy. Indexing the pandas table has no corresponding part in SQL.

To see the index for df, use command:

df.index.values

For example, it returns:

>>> df1.index.values

array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], dtype=int64)

The pandas read\_csv method automatically adds an index as it reads the csv file line by line. To rebuild an index, use the set\_index method. For example:

df.set\_index(np.arrange(0,30,1),’col’)

This sets up a dictionary (a map) between the numerical index and the column ‘col’. The right records can be easily accessed by the index number.

# Data Selection – q

KDB+ q is extremely built for stock price data. The typical historical data table therefore contains information such as Symbol, date, open, high, low, close and volume. Table below shows the comparison between the standard SQL and the qsql. The qsql is similar, but not the same as the standard SQL.

|  |  |  |
| --- | --- | --- |
| Operation | SQL | q |
| Data | Table\_a | t |
| Select columns | SELECT \* FROM Table\_a | q) select from t |
| Select rows (limiting results) | SELECT TOP 5 Var1 FROM Table\_a | q) select [5] from t /select the top 5 rows  q) select [-5] from t /select the bottom 5 rows  q) select [5,10] from t / select the rows between row 5 and row 10  q) select [3;>v] from t where symbol=`AAPL /select the top 3 volumes for AAPL. > means descending, < means ascending. |
| Where | SELECT \* FROM Table\_a WHERE Var1=’x’ | q) select from t where var1=`x  q) select from t where date<=’2018.02.03 |
| Missing records | SELECT \* FROM Table\_a WHERE Var1 IS NULL | q) null t1 /pick out null values in all columns. 0~not null. 1~ null. |
| SELECT DISTINCT | SELECT DISTINCT Var1 FROM Table\_a | q) select distinct from t |
| ORDER (sorting) | SELECT \* FROM Table\_a ORDER BY v ASC | Select first, then  q) `c2 xasc /ascending by c2  q)`c1`c2 xasc /ascending by c1, c2  Use xdesc to replace xasc for descending sorting.  q) select[>v] from t / ~SELECT \* FROM t ORDER BY v ASC  q) select[<v] from t/~SELECT \* FROM t ORDER BY v DESC |
| Loop through array | N/A. The SQL table is a set which does not allowed to be looped through. | N/A. |
| Group by | SELECT Var1, COUNT(\*) AS N FROM Table\_a GROUP BY Var1 | q) select n:count c,av:avg v by symbol from t1 /unlike SQL, the var after the by does not show in the select var list  q) t2: select date,c by symbol from t /output nested list  q) ungroup t2 /convert back to the normal table structure |
| Aggregated function | SELECT Var1, SUM(x) AS Sum\_x FROM Table\_a GROUP BY Var1 | q)f:{[x] *f(x)*} /returns calculation of f(x). for example, 0.5+x\*x  q)t2: select close,f(close) by Symbol from t  q)unqroup t2 |
| Having | SELECT Var1, MAX(x) AS MAX\_x FROM Table\_a GROUP BY Var1 HAVING MAX(x)>=yy | q) select from t where z=(max;x) fby var1 |
| Join | SELECT a.Var1,b.Var2 FROM Table\_a a INNER JOIN Table\_b b ON a.Var\_1=b.Var1 | q) select l.col1,r.col2 from l,r where l.k=r.k  q) t3: t1^t2 /when count of t1 is the same for t2 |
| Union | SELECT Var1 FROM Table\_a  UNION  SELECT Var1 FROM Table\_b | q) t1 insert t2 |
| INSERT | INSERT INTO Table\_a VALUES (…) | q) t1 upsert (…;…) /values are within the () separated by ; |
| Stored procedure |  | Parameterized sql for flexible where conditions and table:  q) mysql:{[s] select from t where Symbol in s}  q) mys{`AAPL}  Functional sql for flexible cloumns:  q) mysql:{c: ()},x;?[`t;();0b;c!c]}  q) mysql `Symbol`date`c /select Symbol, date, close  q) mysql `Symbol`date`h /select Symbol, dte, close |

# Aggregating Functions in q

|  |  |  |
| --- | --- | --- |
| Class | Name | Function |
| Total | average | avg x |
| correlation | x cor y |
| covariance | x cov y |
| standard deviation | dev x |
| maximum | max x |
| median | med x |
| minimum | min x |
| variance | var x |
| weighted average | x wavg y /x is the weight |
| weighted sum | x wsum y /x is the weight |
| sum | sum x |
| product | prd x |
| Cumulative | averages | avgs x |
| maximums | maxs x |
| minimums | mins x |
| sums | sums x |
| products | prds x |
| Rolling (rolling window size=n) | mavg | n mavg x |
| ema | (2%1+n) ema x /the left parameter is the decay factor for ema . There must be the round brackets since the q reads the formula from right to left. |
| msum | n msum x |
| mmax | n mmax x |
| mmin | n mmin x |

# Sample q Programs

We show two daily stock price process examples. The first one is the SPX data. For simplicity, we only take 30 records. The data test01.csv is attached in appendix 1.

We import the data into q, create some technical indicators, and output the processed data to a new csv file. Below are the q script codes:

t0:("DSFFFFI";enlist ",") 0: `:../qin/test0.csv /SPX data

R1:{[x] (x-prev x)%prev x}

M:{[x;n] x- n xprev x}

PM:{[x;n] (x-n xprev x)%n xprev x}

ZZL:{[x;n] n mmin x}

ZZH:{[x;n] n mmax x}

/processing

/t0b: daily return and momentum 21

t0a:select R1:R1[c],M21:M[c;21],PM21:PM[c;21] from t0

t0b: t0^t0a

/t0c: zzh5 and zzl5

t0c:select zzh5:ZZH[h;5],zzl5:ZZL[l;5] from t0b

t0c:t0b^t0c

/output table t0c to t0c.csv

save `:../qdata1/t0c.csv

We created some indicators:

* R1: daily price return. Defined as .
* Momentum M. .
* Percentage Momentum PM: .
* Running highest high and lowest low within past n days ZZH and ZZL.

We save the script in c:/qin/s01.q. To run it, type

q)/l \qin\s01.q

in the q command window. Table below is the final output:



In the second example we took three stock daily price series for AAPL, AMZN and XLE. Appendix 2 is the raw data. We calculate some technical indicators on the price data.

* SMAn – n-day simple moving average.
* EMAn – n-day exponential moving average. The decay factor is linking to n by .
* SlowK – a 3-day simple moving average on the FastK, which is calculated as
* Ongoing Balance Volume (OBV) – Cumulative volume indicator where the cumulation/distribution of volume depends on daily price change. If today’s price is up, then the volume is cumulative, other wise the volume is distributed (deducted from the volume cumulative sum).
* Volume Accumulation (VA) – Volume cumulation/distribution depends on today’s close price to today’s mid-price ((high+low)/2). Volume accumulation is proportional to .
* Volume Percentage – This is an indicator based on the cumulative volume indicator. For example, the OBVPct is calculated by

Below are the scripts:

t1:("DSFFFFI";enlist ",") 0: `:../qin/test1.csv

lag5:{[x] 5 xprev x}

R1:{[x] (x-prev x)%prev x}

M:{[x;n] x- n xprev x}

PM:{[x;n] (x-n xprev x)%n xprev x}

ZZL:{[x;n] n mmin x}

ZZH:{[x;n] n mmax x}

SMA:{[x;n] n mavg x}

EMA:{[x;n] (2%1+n) ema x}

SLOWK:{[c;h;l;n] fastk:(c-n mmin l)%(n mmax h-n mmin l);3 mavg fastk}

OBV:{[c;v] x:?[c>=prev c;v;-1\*v];sums x}

VA:{[c;h;l;v] sums v\*(c-0.5\*h+l)}

VP:{[x;v;n] (x-n xprev x)%n msum v}

/processing

/t1a3: percantage momentum 21 for stocks

t1a:select PM21:PM[c;21] by symbol from t1 /do not select date

t1a2:ungroup t1a

t1a3:t1^t1a2

/t1b3: sma and ema

t1b1:select sma21:SMA[c;21],ema21:EMA[c;21] by symbol from t1a3

t1b2:ungroup t1b1

t1b3:t1a3^t1b2

/t1c3: slowk

t1c1:select SlowK21:SLOWK[c;h;l;21] by symbol from t1b3

t1c2:ungroup t1c1

t1c3:t1b3^t1c2

/t1d3: obv and va

t1d1:select obv:OBV[c;v],va:VA[c;h;l;v] by symbol from t1c3

t1d2:ungroup t1d1

t1d3:t1c3^t1d2

/t1e3: obvp and vap

t1e1:select obvp21:VP[obv;v;21],vap21:VP[va;v;21] by symbol from t1d3

t1e2:ungroup t1e1

t1e3:t1d3^t1e2

/output table t1e3 to t1e3.csv

save `:../qdata1/t1e3.csv

Our third sample simulates a trades tick data table, then we query some summaries on the data table. We use the following features in q:

* N ? M – to generate N uniform distributed random numbers x between zero and M (.
* Col xasc t – to sort table t by column(s) Col.
* W xbar Col – returns bucket divisions for numbers in column Col with width W. This is very useful to bin a continuous variable into equal width buckets.

Below is the list of the codes for s03.q:

/1. define the vectors

dts:2015.01.01+100000?31

tms:100000?24:00:00.000000000

syms:100000?`aapl`amzn`goog

vols:10\*1+100000?1000

pxs:90.0+(100000?2001)%100

/2.create the trades table from the vectors

trades:([] dt:dts;tm:tms;sym:syms;vol:vols;px:pxs)

trades:`dt`tm xasc trades /sort by date and time

/3. rescale the prices to fit the actual price range

trades:update px:1.75\*px from trades where sym=`aapl

trades:update px:12\*px from trades where sym=`amzn

trades:update px:10.5\*px from trades where sym=`goog

/4. queries on trades

y1:select h: max px, l: min px, vwap:vol wavg px by sym,bucket:100000000000 xbar tm from trades /group by bucket the tm every 100000000

y2:select max px-mins px from trades where sym=`aapl /the maximum profit realized over the trading period

# SQL and qsql

The qsql is actually very different from the standard SQL.

## Table

A table in q is a collection of lists. A table is created when we import a csv file, or can be created by:

t:([]date:`date$();v:`float$())

The square brackest [] indicates this is an unkeyed table. The word such as `date indicates the column date type. Table below shows some useful column types:

|  |  |  |  |
| --- | --- | --- | --- |
| type | Type symbol | Type char | Type number |
| boolean | `boolean | B | 1h |
| int | `int | I | 5h |
| long | `long | J | 7h |
| float | `float | F | 9h |
| symbol | ` | S | 11h |
| timestamp | `timestamp | P | 12h |
| month | `month | M | 13h |
| date | `date | D | 14h |
| datetime | `datetime | Z | 15h |
| minute | `minute | U | 17h |
| second | `second | V | 18h |
| time | `time | T | 19h |

There are three ways to express a column type: symbol, char and number. The q returns the type number when you type command type. For example:

q)type 2018.05.28

-14h

Since lists are ordered, it is easy to use command such as t1^t1a (both have the same length) to stick back the lists together. In contrast, a SQL table is a set usually without order defined. A set is easy to do set operation but lacks some features of lists (we shall see this later).

A table may look like this in q command window:

q)t

date v

------------------------

2017.03.02 14.3

2017.03.03 17.2

…

## Keyed Table

If we create a q table by:

kt:([date:`date$()] v:`float$())

This defines a keyed table. A keyed table is a dictionary. The data looks like this:

q)kt

date | v

---------------------------

2017.03.02 | 14.3

2017.03.03 | 17.2

…

If a table t is imported from a csv file, a keyed table can be created by the xkey command:

q)kt:`date xkey `t

A keyed table can be unkeyed by:

q)k: () xkey `kt

## Column and Row

Columns in q table are vectors. Therefore, the operations on q table columns are operations on vectors. If you are hit by the runtime error message “Length”, it often means you misunderstand the vector operatopn. The by clause in qsql has the similar meaning of GROUP BY clause in standard SQL. However, since qsql can perform vector operation, the qsql by clause has more complicated meanings than the SQL GROUP BY.

Consider this SQL:

SELECT symbol,avg(v) AS AV FROM t GROUP BY symbol

The corresponding qsql is:

select av:avg v by symbol from t

You should not use select symbol,av:avg v in qsql since the by clause already implies the symbol in the output.

Note that in the qsql the select variables may be vectors rather than an aggregated scalar. For example, the qsql may output cumulative sum or rolling average, these are the vector outputs. The qsql outputs a nested column, which should be ungrouped to convert back to the normal way in a table. Finally, the qsql by statement only outputs the aggregation relevant columns, you need to use the ^ operation to merge back the other columns.

## Control Flow

Since q is doing vector operations, the control flow is very tricky. For example, consider the if statement. The if or the ? statement seems similar to that in Microsoft EXCEL. They have different use. In Microsoft EXCEL, the if statement returns a value when the condition is true and another value when the condition is false. In q, the ? or the if statement does not return a value. It executes a statement when the condition is true and another statement when the condition is false. See the use in the OBV function definition in s02.q above.

Appendix 1: Data test0.csv

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| date | symbol | o | h | l | c | v |
| 4/13/2018 | SP-500 | 2676.9 | 2680.26 | 2645.05 | 2656.3 | 7183 |
| 4/16/2018 | SP-500 | 2670.1 | 2686.49 | 2665.16 | 2677.84 | 7481 |
| 4/17/2018 | SP-500 | 2692.74 | 2713.34 | 2692.05 | 2706.39 | 7199 |
| 4/18/2018 | SP-500 | 2710.11 | 2717.49 | 2703.63 | 2708.64 | 7702 |
| 4/19/2018 | SP-500 | 2701.16 | 2702.84 | 2681.9 | 2693.13 | 7471 |
| 4/20/2018 | SP-500 | 2692.56 | 2693.94 | 2660.61 | 2670.14 | 9112 |
| 4/23/2018 | SP-500 | 2675.4 | 2682.86 | 2657.99 | 2670.29 | 7297 |
| 4/24/2018 | SP-500 | 2680.8 | 2683.55 | 2617.32 | 2634.56 | 8611 |
| 4/25/2018 | SP-500 | 2634.92 | 2645.3 | 2612.67 | 2639.4 | 7880 |
| 4/26/2018 | SP-500 | 2651.65 | 2676.48 | 2647.16 | 2666.94 | 8266 |
| 4/27/2018 | SP-500 | 2675.47 | 2677.35 | 2659.01 | 2669.91 | 7242 |
| 4/30/2018 | SP-500 | 2675.05 | 2682.92 | 2648.04 | 2648.05 | 10093 |
| 5/1/2018 | SP-500 | 2643.64 | 2655.27 | 2625.41 | 2654.8 | 7892 |
| 5/2/2018 | SP-500 | 2654.24 | 2660.87 | 2631.7 | 2635.67 | 8919 |
| 5/3/2018 | SP-500 | 2628.08 | 2637.14 | 2594.62 | 2629.73 | 8550 |
| 5/4/2018 | SP-500 | 2621.45 | 2670.93 | 2615.32 | 2663.42 | 7657 |
| 5/7/2018 | SP-500 | 2669.36 | 2683.35 | 2664.7 | 2672.63 | 7583 |
| 5/8/2018 | SP-500 | 2670.26 | 2676.34 | 2655.2 | 2671.92 | 8938 |
| 5/9/2018 | SP-500 | 2678.12 | 2701.27 | 2674.14 | 2697.79 | 9308 |
| 5/10/2018 | SP-500 | 2705.02 | 2726.11 | 2704.54 | 2723.07 | 7761 |
| 5/11/2018 | SP-500 | 2722.7 | 2732.86 | 2717.45 | 2727.72 | 7109 |
| 5/14/2018 | SP-500 | 2733.37 | 2742.1 | 2725.47 | 2730.13 | 7357 |
| 5/15/2018 | SP-500 | 2718.59 | 2718.59 | 2701.91 | 2711.45 | 7785 |
| 5/16/2018 | SP-500 | 2712.62 | 2727.76 | 2712.17 | 2722.46 | 7553 |
| 5/17/2018 | SP-500 | 2719.71 | 2731.96 | 2711.36 | 2720.13 | 7419 |
| 5/18/2018 | SP-500 | 2717.35 | 2719.5 | 2709.18 | 2712.97 | 8720 |
| 5/21/2018 | SP-500 | 2725.95 | 2739.19 | 2725.7 | 2733.01 | 7122 |
| 5/22/2018 | SP-500 | 2738.34 | 2742.24 | 2721.88 | 2724.44 | 8208 |
| 5/23/2018 | SP-500 | 2713.98 | 2733.33 | 2709.54 | 2733.29 | 8115 |
| 5/24/2018 | SP-500 | 2730.94 | 2731.97 | 2707.38 | 2727.76 | 8010 |

Appendix 2: test1.csv (part)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| date | symbol | o | h | l | c | v |
| 4/13/2018 | AAPL | 174.88 | 175.84 | 173.85 | 174.73 | 251046 |
| 4/16/2018 | AAPL | 174.98 | 176.19 | 174.83 | 175.82 | 215626 |
| 4/17/2018 | AAPL | 176.51 | 178.94 | 176.41 | 178.24 | 265950 |
| 4/18/2018 | AAPL | 177.78 | 178.82 | 176.88 | 177.84 | 205520 |
| 4/19/2018 | AAPL | 174.83 | 175.39 | 172.66 | 172.8 | 347561 |
| 4/20/2018 | AAPL | 170.45 | 171.22 | 165.43 | 165.72 | 654366 |
| 4/23/2018 | AAPL | 166.86 | 166.92 | 164.09 | 165.24 | 364815 |
| 4/24/2018 | AAPL | 165.75 | 166.33 | 161.22 | 162.94 | 336807 |
| 4/25/2018 | AAPL | 162.62 | 165.42 | 162.41 | 163.65 | 283476 |
| 4/26/2018 | AAPL | 164.06 | 165.73 | 163.37 | 164.22 | 279264 |
| 4/27/2018 | AAPL | 164.07 | 164.33 | 160.63 | 162.32 | 356198 |
| 4/30/2018 | AAPL | 162.19 | 167.26 | 161.84 | 165.26 | 422717 |
| 5/1/2018 | AAPL | 166.35 | 169.2 | 165.27 | 169.1 | 528961 |
| 5/2/2018 | AAPL | 175.25 | 177.75 | 173.8 | 176.57 | 664894 |
| 5/3/2018 | AAPL | 175.68 | 177.5 | 174.44 | 176.89 | 340333 |
| 5/4/2018 | AAPL | 178.17 | 184.25 | 178.17 | 183.83 | 561448 |
| 5/7/2018 | AAPL | 185.11 | 187.67 | 184.75 | 185.16 | 425680 |
| 5/8/2018 | AAPL | 185 | 186.22 | 183.66 | 186.05 | 283642 |
| 5/9/2018 | AAPL | 186.58 | 187.4 | 185.22 | 187.36 | 231869 |
| 5/10/2018 | AAPL | 187.74 | 190.37 | 187.65 | 190.04 | 279634 |
| 5/11/2018 | AAPL | 189.55 | 190.06 | 187.45 | 188.59 | 261744 |
| 5/14/2018 | AAPL | 189.1 | 189.53 | 187.86 | 188.15 | 204012 |
| 5/15/2018 | AAPL | 186.79 | 187.07 | 185.1 | 186.44 | 236436 |
| 5/16/2018 | AAPL | 186.08 | 188.46 | 186 | 188.18 | 190503 |
| 5/17/2018 | AAPL | 188 | 188.91 | 186.36 | 186.99 | 170237 |
| 5/18/2018 | AAPL | 187.2 | 187.81 | 186.13 | 186.31 | 180872 |
| 5/21/2018 | AAPL | 187.92 | 189.27 | 186.91 | 187.63 | 183922 |
| 5/22/2018 | AAPL | 188.36 | 188.88 | 186.78 | 187.16 | 152322 |
| 5/23/2018 | AAPL | 186.31 | 188.5 | 185.76 | 188.36 | 200026 |
| 5/24/2018 | AAPL | 188.75 | 188.84 | 186.21 | 188.15 | 231818 |
| 4/13/2018 | AMZN | 1450 | 1459.78 | 1424.52 | 1430.79 | 36804 |
| 4/16/2018 | AMZN | 1445 | 1447 | 1427.48 | 1441.5 | 27940 |
| 4/17/2018 | AMZN | 1463.1 | 1507.19 | 1457.02 | 1503.83 | 51009 |
| 4/18/2018 | AMZN | 1515.38 | 1533.8 | 1504.11 | 1527.84 | 51282 |
| 4/19/2018 | AMZN | 1544.99 | 1568.52 | 1539.06 | 1556.91 | 65103 |
| 4/20/2018 | AMZN | 1559.7 | 1561.2 | 1516.09 | 1527.49 | 55309 |
| 4/23/2018 | AMZN | 1548 | 1548 | 1503.41 | 1517.86 | 44508 |
| 4/24/2018 | AMZN | 1531.97 | 1539.5 | 1448.45 | 1460.09 | 74720 |
| 4/25/2018 | AMZN | 1458 | 1469.99 | 1415.02 | 1460.17 | 64864 |
| 4/26/2018 | AMZN | 1485.1 | 1529.42 | 1478.5 | 1517.96 | 86031 |
| 4/27/2018 | AMZN | 1633.78 | 1638.1 | 1567.39 | 1572.62 | 130325 |