Data Analysis

Get data from NYC MTA Turnstile dataset according to Objectives

1.Total number of entries & exits accross the Subway System for August 1st, 2017

```
import pandas as pd

data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index =
'2017-08-01'")
print(data.shape)
print(data.columns)
print(data.index)

totals1 = {'total entries':data['entries'].sum().astype(int), 'total
exits':data['exits'].sum().astype(int)}
print (totals1)

totals2 = {'total entries':data['entries'].agg(['sum']).astype(int),
'total exits':data['exits'].agg(['sum']).astype(int)}
print(totals2)

print(data.head(2))
```

```
(2413, 9)
Index(['ca', 'unit', 'scp', 'station', 'linename', 'division', 'desc',
       'entries', 'exits'],
      dtype='object')
DatetimeIndex(['2017-08-01', '2017-08-01', '2017-08-01', '2017-08-01',
               '2017-08-01', '2017-08-01', '2017-08-01', '2017-08-01',
               '2017-08-01', '2017-08-01',
               ...
               '2017-08-01', '2017-08-01', '2017-08-01', '2017-08-01',
               '2017-08-01', '2017-08-01', '2017-08-01', '2017-08-01',
               '2017-08-01', '2017-08-01'],
              dtype='datetime64[ns]', name='time', length=2413,
freq=None)
{'total entries': 108664471552, 'total exits': 89470230528}
{'total entries': sum    108664471552
Name: entries, dtype: int64, 'total exits': sum    89470230528
Name: exits, dtype: int64}
              ca  unit       scp station linename division     desc  \
time
2017-08-01  A002  R051  02-00-00   59 ST  NQR456W      BMT  REGULAR
2017-08-01  A002  R051  02-00-01   59 ST  NQR456W      BMT  REGULAR

              entries      exits
time
```

```
2017-08-01  6273623.0  2125396.0
2017-08-01  5665973.0  1260028.0
```

Remarks: 1) We had two approaches to calculate the total of entries / exits: by using sum() function and by using agg() function (which allows more enumerations of aggregate functions inside like count(), mean(), min(), max() etc). We received the same results in both cases: Total entries accross the subway system = 108664471552; Total exits accross the subway system = 89470230528

2. Busiest station; Busiest turnstile

```
import pandas as pd
data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index = 
'2017-08-01'")

#add new column to data as 'traffic'
data['traffic']=data['entries']+data['exits']
#print(data.head())

#group data per station
#grouped_data = data.groupby(['station', 'ca'])
#print(grouped_data.head())

#group data by station with total of traffic per station
traffic_per_station = 
data.groupby(['station']).sum().astype(int).sort_values(by=['traffic']
, ascending = False)

traffic_per_ca = data.groupby(['station', 'ca', 
'linename']).sum().astype(int).sort_values(by=['traffic'], ascending = 
False)

traffic_per_scp = data.groupby(['station', 
'scp']).sum().astype(int).sort_values(by=['traffic'], ascending = 
False)

print(traffic_per_station.head())
print(traffic_per_ca.head())
print(traffic_per_scp.head())

                    entries       exits      traffic
station                                             
42 ST-PORT AUTH  7230184448  6060483584  13290668032
34 ST-HERALD SQ  5700049920  7240227328  12940277760
TIMES SQ-42 ST   5812783104  4528741376  10341524480
CHAMBERS ST      5084269056  4453331968   9537600512
104 ST           4844778496  3522390528   8367169024
                                       entries       exits     traffic
station         ca    linename                                        
42 ST-PORT AUTH N063A ACENQRS1237W  4979162112  4261638400  9240800256
```

```
57 ST-7 AV      A011  NQRW         4127833344  3061122048  7188955648
34 ST-HERALD SQ A025  BDFMNQRW     2940512768  3930140672  6870653440
HIGH ST         N100  AC           2747675648  3696782592  6444458496
23 ST           N508  FM           3083830528  3243232512  6327063552
                             entries       exits     traffic
station         scp
47-50 STS ROCK  01-03-02  1922993408  2036462720  3959456256
CHAMBERS ST     00-00-02  2115170944  1712247296  3827418112
23 ST           00-00-02  1854337152  1946778624  3801115904
42 ST-PORT AUTH 01-00-01  2032318208  1691701632  3724019712
HIGH ST         00-00-02  1913238272  1778705024  3691943424
```

3.The busiest and least busy stations in the system over all of July 2017

```
import pandas as pd

data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index >=
'2017-07-01' and index <= '2017-07-31'")

#add new column to data as 'traffic'
data['traffic']=data['entries']+data['exits']
#print(data.head())

#group data by station with total of traffic per station
traffic_per_station_desc =
data.groupby(['station']).sum().astype(int).sort_values(by=['traffic']
, ascending = False)
traffic_per_station_asc =
data.groupby(['station']).sum().astype(int).sort_values(by=['traffic']
, ascending = True)

print(traffic_per_station_desc.head())
print(traffic_per_station_asc.head())

                       entries          exits        traffic
station
23 ST             1234557861888  1348212359168  2582770089984
42 ST-PORT AUTH   1394389286912  1142871031808  2537260318720
34 ST-HERALD SQ   1054386487296  1341189390336  2395575943168
CANAL ST          1086386470912  1083935817728  2170322288640
125 ST            1246636146688   712432156672  1959068434432
                   entries      exits    traffic
station
ORCHARD BEACH     88400984    5087615   93488592
PATH WTC 2        34859812   94173312  129033128
SUTTER AV        179548960   68088320  247637280
BROAD CHANNEL    221241808   29585608  250827424
JFK JAMAICA CT1  223608368  197500784  421109152
```

4. Station that had the highest average number of entries between midnight & 4am on Fridays in July 2017

```
import pandas as pd

# Create a pandas DataFrame for July 2017
data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index >=
'2017-07-01' and index <= '2017-07-31'")  # (1)

# filter by the first 4 hours of the day and for Fridays
data = data.iloc[data.index.indexer_between_time('00:00', '04:00')]
data_shape1 = data.shape # for quantitative check

data = data.iloc[data.index.weekday ==4] # final data filtered by
weekday = Friday
data_shape2 = data.shape # for quantitative check

print(data_shape1, data_shape2)

# Group filtered data by stations and sort it descendently
average_per_station_desc =
data.groupby(['station']).mean().astype(int).sort_values(by=['entries'
], ascending = False)

print(average_per_station_desc.head())


# TODO1:print(data_shape1, data_shape2, 'Ratio
week/day:'data_shape1/data_shape2) # we should have roughly
data_shape1 = 7*data_shape2 due to the ratio week/day

# mask for Fridays
#mask = pd.DatetimeIndex.dayofweek # couldn't use it with a logical
condition to filter the dataframe

# create a mask for Fridays for the entire month of July 2017
#start = dt.date(2017, 7, 1)
#end = dt.date(2017, 7, 31)
#d = pd.date_range(start, end)

#mask = (d.weekday == 4)
#data = data[mask]

#data = data.resample('W-FRI')  # resampling = bad idea, normal
filtering of data should be done with iloc + conditions.

#data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index
>= '2017-07-01' and index <= '2017-07-31' and
index.indexer_between_time(start='00:00', end='04:00')")  # (1)
```

```
# data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", index =
pd.date_range('2017-07-01', '2017-07-02', freq='1H'))  # (2)

#rng = pd.date_range(start="2017-7-01",periods=4,freq='1H')
#data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", index = rng)
# (3)-simple data partition, not working

#add new column to data as 'traffic'
#data['traffic']=data['entries']+data['exits']

#group data by station with total of traffic per station
#traffic_per_station_desc =
data.groupby(['station']).mean().astype(int).sort_values(by=['traffic'
], ascending = False)
#traffic_per_station_asc =
data.groupby(['station']).sum().astype(int).sort_values(by=['traffic']
, ascending = True)


# Resample data to 4H, from 00:00 AM to 04:00 AM
#data = data.resample(data, freq ='1H')

#print(data.head(25))
#print(data.shape)
```

```
(214600, 9) (28335, 9)
                   entries      exits
station
104 ST           538305408  391375680
HIGH ST          459306080  618256704
183 ST           448343808  467319648
EASTCHSTER/DYRE  421811264  455807328
57 ST-7 AV       337862464  248386976
```

5.Stations that have seen the highest monthly usage growth/decline over the period from July 2016 to July 2017

5.1 The problem

We need to define "Growth / Deline". If we'd plot the various values of traffic per each station, between July 2016 and June 2017, we'd get a plot of points (one point / month), therefore a curve with 12 points. On this graph we have growths and declines. Thus, "Growth" means that for two consecutive months in the range of research (not necesarily contiguous, e.g. months[2] and months[5] in a list called months[]) the difference in traffic (usage) for a specific station is positive (>0). Our goal will be to identify the maximum of this difference(maximum growth), associated with a station. Similarly, "Decline" means that for two consecutive months in the range of research (not necesarily contiguous, e.g. months[2] and months[5] in a list called months[]) the difference in traffic (usage) for a specific station is negative (<0). Our goal will be to identify the maximum (in absolute value) of this difference, associated with a station.

5.2 Implementation

```
import pandas as pd

# Create a pandas DataFrame between July 1st 2016 and July 1st 2017
(12 months)
data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index >=
'2016-07-01' and index <= '2017-07-1'")

# Create a list of stations
df_ref = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index
>= '2017-07-01' and index <= '2017-07-2'")
station_lst = df_ref.station.unique() # We defined a smaller dataframe
i.e. "df" to minimize the computation overhead,
                                      # in order to get a list of the
existing stations

print(station_lst.shape) # (1)
# print(station_lst)

# Create new columns 'year', 'month', 'traffic' for the entire
dataframe
data['year']= data.index.year
data['month']= data.index.month
data['traffic']=data['entries']+data['exits']

# Test query to understand better the data
# data = data.loc[(data['month']==8) & (data['station']=='104 ST')]
# print(data.head(25))

print(data.shape)     # (2)

# We'll create two functions that calculate the monthly usage max
```

```
growth / decline for the period in question
# (1)and (2) show that we have a total of 376 stations and  10,151,873
records, concerning entries and exits per each
station/turnstiles/month
# between July 2016 and July 2017.

# Model and algorithms:

months_lst = [7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6] # Contains the
months in question, starting with _
                                                     # July(2016) and
finishing with June(2017)
def calc_max_growth():
    max_growth=0
    stations_with_max_growth =[] # Empty list of stations that will be
progressively populated _
                                 # as we identify stations according
to our criteria
    for st in station_lst:  # 376 stations => 376 iterations
        for i in range(0,len(months_lst)-1): # Indices by which we are
going through the months_lst stored values;
                                             # for i = 10 we'll access
the last stored value (i+1) on position '11'
            for j in range (i+1, len(months_lst)): # Complexity of
this algorithm: the number of loops that results here and the
                                                   # previous
iterations is 66;
                                                   # the total number
of loops considering the number of stations is 376 x 66 = 24816
                                                   # in each loop we
have adds and subtracts, therefore this is a O(n)problem
                                                   # in our case, n =
24816 and we can safely assume t=1s to make the computations
                                                   # in most inner
loop, therefore a total time of T=24816s, which means aprox.7h.
                                                   # this algorithm
was tested in a simplified form for about 4h and didn't finish...
                                                   # Of course, a
deeper analysis to optimize this algorithm is necessary...
                #print("station: %s" % st)

                df2 = data.loc[(data['month']==months_lst[j]) &
(data['station']==st)]
                st_traffic2 = df2['traffic'].sum().astype(int)
                #print("st_traffic2: %i" %(st_traffic2))

                df1 = data.loc[(data['month']==months_lst[i]) &
(data['station']==st)]
                st_traffic1 = df1['traffic'].sum().astype(int)
                #print("st_traffic1: %i" %(st_traffic1))
```

```
                st_diff_traffic = st_traffic2 - st_traffic1 # the
difference in traffic is the actual growth/decline between
                                                            # two
consecutive months
                #print("st_diff_traffic: %i" %(st_diff_traffic))

                if st_diff_traffic > 0 and st_diff_traffic >
max_growth: # "growth" means that difference of traffic _

# in consecutive months is a positive value(>0)
                    stations_with_max_growth.append(st)
                    max_growth = st_diff_traffic
                    print("Station '%s"  %st+"' was appended to the
list;Month less traffic= %i" %months_lst[i]+", Month more traffic= %i"
%months_lst[j]+";Growth: %i" %max_growth)

    # the criterion to populate the list is max_growth(st(i)) <
max_growth(st(i+1)) which means "growth" is increasing _
    # therefore the last elemenent of the list will have the maximum
growth value for a certain station and is the one
    # we need
    station_with_max_growth = stations_with_max_growth[-1]
    print(stations_with_max_growth) # we print the whole list of
appended stations that meet our criterion
    print ("Final result: Station with maximum growth = '%s "
%station_with_max_growth+"'; Maximum growth: %i" %max_growth)

    return

def calc_max_decline():
    max_decline=0
    stations_with_max_decline =[] # Empty list of stations that will
be progressively populated _
                                  # as we identify stations according
to our criteria
    for st in station_lst:
        for i in range(0,len(months_lst)-1): # Indices by which we are
going through the months_lst stored values;
                                             # for i = 10 we'll access
the last stored value (i+1) on position '11'
            for j in range (i+1, len(months_lst)):
                #print("station: %s" % st)

                df2 = data.loc[(data['month']==months_lst[j]) &
(data['station']==st)]
                st_traffic2 = df2['traffic'].sum().astype(int)
                #print("st_traffic2: %i" %(st_traffic2))
```

```
                df1 = data.loc[(data['month']==months_lst[i]) &
(data['station']==st)]
                st_traffic1 = df1['traffic'].sum().astype(int)
                #print("st_traffic1: %i" %(st_traffic1))

                st_diff_traffic = st_traffic2 - st_traffic1 # the
difference in traffic is the actual growth / decline
                                                            # between
two consecutive months
                #print("st_diff_traffic: %i" %(st_diff_traffic))

                if st_diff_traffic < 0 and st_diff_traffic <
max_decline: # "decline" means that difference of traffic _

# in consecutive months is a negative value(<0)
                    stations_with_max_decline.append(st)
                    max_decline = st_diff_traffic
                    print("Station '%s"  %st+"' was appended to the
list;Month more traffic= %i" %months_lst[i]+", Month less traffic= %i"
%months_lst[j]+";Decline: %i" %max_decline)

    # the criterion to populate the list is max_decline(st(i+1)) <
max_growth(st(i)) which means "decline" is increasing _
    # (in absolute value)therefore the last elemenent of the list will
have the maximum decline value for a certain station
    # and is the one we need
    station_with_max_decline= stations_with_max_decline[-1]
    print(stations_with_max_decline) # we print the whole list of
appended stations that meet our criterion
    print ("Final result: Station with maximum decline= '%s"
%station_with_max_decline+"'; Maximum decline: %i" %max_decline)

    return

#calc_max_growth()
calc_max_decline()
```

```
(376,)
(10151873, 12)
Station '59 ST' was appended to the list;Month more traffic= 7, Month
less traffic= 8;Decline: -7846756352
Station '59 ST' was appended to the list;Month more traffic= 7, Month
less traffic= 9;Decline: -16444129280
Station '59 ST' was appended to the list;Month more traffic= 11, Month
less traffic= 12;Decline: -26381385728
Station '59 ST' was appended to the list;Month more traffic= 11, Month
less traffic= 1;Decline: -42292609024
```

```
Station '59 ST' was appended to the list;Month more traffic= 11, Month
less traffic= 2;Decline: -102772965376
Station '57 ST-7 AV' was appended to the list;Month more traffic= 7,
Month less traffic= 9;Decline: -361620307968
Station '57 ST-7 AV' was appended to the list;Month more traffic= 7,
Month less traffic= 10;Decline: -764178333696
Station '57 ST-7 AV' was appended to the list;Month more traffic= 7,
Month less traffic= 11;Decline: -1284927389696
Station '57 ST-7 AV' was appended to the list;Month more traffic= 7,
Month less traffic= 2;Decline: -1350282772480
Station '57 ST-7 AV' was appended to the list;Month more traffic= 8,
Month less traffic= 11;Decline: -1382017138688
Station '57 ST-7 AV' was appended to the list;Month more traffic= 8,
Month less traffic= 2;Decline: -1447372521472

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-6-da590cd862da> in <module>()
    106
    107 #calc_max_growth()
--> 108 calc_max_decline()
    109
    110

<ipython-input-6-da590cd862da> in calc_max_decline()
     78                 #print("station: %s" % st)
     79
---> 80                 df2 = data.loc[(data['month']==months_lst[j])
& (data['station']==st)]
     81                 st_traffic2 = df2['traffic'].sum().astype(int)
     82                 #print("st_traffic2: %i" %(st_traffic2))

/usr/local/lib/python3.5/dist-packages/pandas/core/ops.py in
wrapper(self, other, axis)
    877
    878             with np.errstate(all='ignore'):
--> 879                 res = na_op(values, other)
    880             if is_scalar(res):
    881                 raise TypeError('Could not compare {typ} type
with Series'

/usr/local/lib/python3.5/dist-packages/pandas/core/ops.py in na_op(x,
y)
    781
    782         if is_object_dtype(x.dtype):
--> 783             result = _comp_method_OBJECT_ARRAY(op, x, y)
    784         else:
    785
```

```
/usr/local/lib/python3.5/dist-packages/pandas/core/ops.py in
_comp_method_OBJECT_ARRAY(op, x, y)
    761         result = lib.vec_compare(x, y, op)
    762     else:
--> 763         result = lib.scalar_compare(x, y, op)
    764     return result
    765

KeyboardInterrupt:
```

5.3 Conclusions

The time needed to run entirely the two functions turned out to be quite big (aprox 4 hours from initial tests when the algorithm didn't finish). Inside the code for the function "calc_max_growth" we included a short analysis of complexity of the algorithm, that shows an order of complexity of O(n)...As n = 376(stations) x 66(2nd, 3rd inner loops) = 28416...This shows that for one algorithm would take about 7h to produce its final results...

Therefore, we ran separately each function (and disabled the other) and after some time when enough data was gathered, we interupted the kernel. The results we could produce are listed below:

List with stations with max_growth(as appended in stations_with_max_growth[]):

(376,) (10151873, 12) Station '59 ST' was appended to the list;Month less traffic= 7, Month more traffic= 10;Growth: 100171907072 Station '59 ST' was appended to the list;Month less traffic= 7, Month more traffic= 11;Growth: 293825216512 Station '59 ST' was appended to the list;Month less traffic= 7, Month more traffic= 4;Growth: 400098656256 Station '59 ST' was appended to the list;Month less traffic= 7, Month more traffic= 5;Growth: 808745107456 Station '59 ST' was appended to the list;Month less traffic= 8, Month more traffic= 5;Growth: 816591863808 Station '59 ST' was appended to the list;Month less traffic= 9, Month more traffic= 5;Growth: 825189236736 Station '34 ST-HERALD SQ' was appended to the list;Month less traffic= 7, Month more traffic= 6;Growth: 879962357760 Station '34 ST-HERALD SQ' was appended to the list;Month less traffic= 9, Month more traffic= 6;Growth: 921625690112 Station '34 ST-HERALD SQ' was appended to the list;Month less traffic= 11, Month more traffic= 6;Growth: 938787733504

Station with max growth between July 2016 and July 2017: '34 ST-HERALD SQ'; Final growth between month 11(Nov 2016) and month 6(June 2017) : 938787733504

List with stations with max_decline(as appended in stations_with_max_decline[]):

(376,) (10151873, 12) Station '59 ST' was appended to the list;Month more traffic= 7, Month less traffic= 8;Decline: -7846756352 Station '59 ST' was appended to the list;Month more traffic= 7, Month less traffic= 9;Decline: -16444129280 Station '59 ST' was appended to the list;Month more traffic= 11, Month less traffic= 12;Decline: -26381385728 Station '59 ST' was appended to the list;Month more traffic= 11, Month less traffic= 1;Decline: -

42292609024 Station '59 ST' was appended to the list;Month more traffic= 11, Month less traffic= 2;Decline: -102772965376 Station '57 ST-7 AV' was appended to the list;Month more traffic= 7, Month less traffic= 9;Decline: -361620307968 Station '57 ST-7 AV' was appended to the list;Month more traffic= 7, Month less traffic= 10;Decline: -764178333696 Station '57 ST-7 AV' was appended to the list;Month more traffic= 7, Month less traffic= 11;Decline: -1284927389696 Station '57 ST-7 AV' was appended to the list;Month more traffic= 7, Month less traffic= 2;Decline: -1350282772480 Station '57 ST-7 AV' was appended to the list;Month more traffic= 8, Month less traffic= 11;Decline: -1382017138688 Station '57 ST-7 AV' was appended to the list;Month more traffic= 8, Month less traffic= 2;Decline: -1447372521472

Station with max decline between July 2016 and July 2017: '57 ST-7 AV'; Final decline between month 8(Aug 2016) and month 2(Feb 2017) :-1447372521472

Modeling

1. Model development, fit and evaluation

1.1 Identification of a model

Hypothesis in predicting 'Exit' values:

H1. 'Exits' vs 'Time' is a time series => should show : trend, periodicity, seasonality, white noise… H2. 'Exits' depends on 'Station' and 'Line' and 'Turnstile' due to concentration of population in specific areas (Station) and specific daily destinations (Station, Line, Turnstile) inside the city(most of the traffic is generated by people going to work in the morning and returning from work in the afternoon)

H1 =>Time series model

H2=>Regression model

Both hypothesis are valid….However, we are going to prove that with the data provided (Station(ST), Line(L), Turnstile(T)) the only possible way to make predictions is by using the Time Series model. In the following lines we'll show why a regression model is not relevant with the set of attributes provided.

Considerations on the Regression model

We identify here the dependencies between Station – Line – Turnstiles (ST, L, T) and 'Exits' and we'll make the analysis of predictions means of the volume of 'Exits' based on the study of these 3 attributes…Since the response variable is 'Exits' and is numerical, is clear that in order to make predictions we need to employ a type of regression. But is not the usual type (simple or multivariate linear regression) where we use the independent variables also of numerical type…Here, our independent variables are of categorical type (Station – Line – Turnstile) and from this combination it turns out that the candidate models for predictions would be KNN(we could create a classifier with multiple classes based on an Euclidian distance to the center of the city, for instance) or an algorithm based on decision trees(e.g. CART, C4.5 etc).

Why we can exclude KNN algorithm:

Reason1.The main problem when choosing KNN is that we need to keep the entire volume of observations in order to make predictions. Thus, only for one year of observations (July 2016-July 2017), we saw that the number of records was about 10.000.000. Only by taking the number of stations of 376, one line and one turnstile per station and 365 x 24 hours/day of observations, would give us a volume of data Vdata = 376 x 365 x 24 = 3,293,760 records… Assuming that we have Tr_data (volume of data for training) = 8000000 with Test_data =2000000, this would give us a very long time to train the algorithm for only one year of observations… So from the standpoint of the volume of data to train/test, KNN would require in our case a very long time to process and make predictions and therefore is not a good choice…

Reason 2. It can also be shown that only taking into account the existing categorical attributes (ST, L, T), the vectors of observations we could form would be irrelevant in terms of using Euclidian distance to form proper clusters…Without going into details, we would also need information about the distances between stations to understand their distribution and based on THIS information to calculate Euclidian distance (and perhaps also using ST and L information) in order to form coherent clusters in terms of position and clusters of stations inside the city... For these 2 reasons, we won't be using KNN algorithm to model this data.

Algorithm based on decision trees:

The other candidate for regression would be one of the algorithms based on decision trees (like CART, C4.5 etc). We will show that using a tree model in this case is equivalent to using a simple filter (ST, L) applied to the Dataframe and thus, the regression actually takes into account only the (time, 'Exits') tuple and thus, we finally reach the H1 hypothesis of using a Time Series to make specific predictions for a given tuple (ST, L).

Starting with the decision tree, let's assume that we set the root to the pool of stations…A sketch of the decision tree would look like this: ROOT--> ST1—(Y)L1-->L1—(Y)T1-->T1-->Exit111; ROOT--> ST1—(N)L1-->L2—(Y)T1-->T1-->Exit121; ROOT--> ST2—(Y)L1-->L1—(Y)T2-->T2-->Exit212; ROOT--> ST2—(N)L1-->L3—(N)T2-->T3-->Exit233; ....

We use the notation of ((Y)Lx/(N)Lx)/ ((Y)Tx/(N)Tx) to depict the following situations: when we move down on the branches of the tree and the next node is confirmed to be line Lx (i.e. (Y)Lx) we reach Lx ; if the next node is not Lx (i.e. (N)Lx) we reach another line, say Ly... We see that at the leaf level we have the observed numerical values for exits, e.g Exit111, Exit121, Exit212 and so on…The format of "Exit" value is "Exit xyz" corresponding with indices of STx, Ly, Tz and therefore, any path in the decision tree will depend on these three indices, which thus uniquely identifies a turnstile(Tz) based on the station (STx) and a line (Ly) inside the station…Therefore, any path in the tree reaching a final leaf, is a combination like this tuple (STx, Ly, Tz)…This tuple can be associated to a filter in the dataset having the same coordinates for Station(STx), Line(Ly) and Turnstile(Tz). We are actually interested in filtering by the pair (STx, Ly) and we will take the average of "Exits" for all the possible Turnstiles (Tz), as a first level of prediction concerning the turnstiles.

Remark1: We should note that there cannot be given turnstiles alone, as entry data, they are always related to a pair (STx, Ly), so our approach above makes sense.

Conclusion on choosing the model to develop

Since this approach of applying a filter (STx, Ly) is equivalent to the one of following a specific path in a decision tree, it follows that our forecasts means can focus only to forecasts based on the time series corresponding a (STx, Ly) filter, for which we take only the observables Avg(Tz, Exits xyz) From the considerations written above, we can state that the final model we are going to use is a Time Series model. This model will be applied to a set of data filtered by a (STx, Ly) peer and for a period of time which we are going to choose.

1.2 Development of a Time Series model to forecast 'Exits' values for given turnstiles

```
import pandas as pd
from pandas import Series
from pandas import DataFrame
#from pandas.core import datetools
from pandas.plotting import scatter_matrix


from matplotlib import pyplot

from sklearn.metrics import mean_squared_error
from math import sqrt

from statsmodels.tsa.stattools import adfuller

# DATA PREPROCESSING
# Create a pandas Time Series between July 1st 2016 and July 1st 2017
(12 months) based on a filter (STx, Ly)
data = pd.read_hdf("/data/dscmta-e6/mta_turnstile.h5", where="index >=
'2016-01-01' and index <= '2017-07-31'") # (1)

# Filter data by a specific Station, we chose '34 ST-HERALD SQ'
data = data.loc[data['station']=='34 ST-HERALD SQ']
data = data.loc[data['linename']=='BDFMNQR']
#data = data.loc[data['ca']=='A025']

# Data inspection by using different final filters;

#data_00 = data.loc[data['scp']=='01-00-00']  # filter by a specific
turnstile
#data_01 = data.loc[data['scp']=='01-03-01']
#data_02 = data.loc[data['scp']=='01-03-02']
```

```
#data_03 = data.loc[data['scp']=='01-03-03']

# Make one single record based on (STx, Ly, Tz) tuple, by using
groupby() and mean() functions
#data_per_ca_00 = data_00.groupby(['station', 'linename', 'ca',
'scp']).mean().astype(int)#.sort_values(by=['traffic'], ascending =
False)
#data_per_ca_00 = data_00.groupby(['station', 'linename', 'ca',
'scp']).mean().astype(int)
#data_per_ca_01 = data_01.groupby(['station', 'linename', 'ca',
'scp'])#.mean().astype(int)
#data_per_ca_01 = data_01.groupby(['station', 'linename', 'ca',
'scp']).sum().astype(int)
#data_per_ca_02 = data_02.groupby(['station', 'linename', 'ca',
'scp']).mean().astype(int)
#data_per_ca_02 = data_02.groupby(['station', 'linename', 'ca',
'scp']).sum().astype(int)
#data_per_ca_03 = data_03.groupby(['station', 'linename', 'ca',
'scp']).mean().astype(int)
#data = data.groupby(['station', 'linename', 'ca',
'scp']).mean().astype(int)

data = data['exits'] # our targeted data;
# data is a Dataframe. We need to convert it to a Time Series
series = pd.Series(data)

# Resample data:
# We are going to take an average of all samples taken every 4h (6
samples) on every day, because
# we are going to make daily predictions (see Objectives / Modeling)
series = series.resample('D').mean()
print(series.head())

# Number of records of the series
print(series.shape)


# Is not clear to me whether the 'scp' is actually the
# information recorded by a turnstile... If we display the data
filtered by a 'scp' we get an almost strait line, like this:


# From what we can see by looking at the data plot, we can assume a
additive model of type y(t) = X(t) + T(t) + S(t)
# It appears that our data is trend stationary, meaning it doesn't
have a trend component

# DATA ANALISYS
# Print a summary statistics of data
```

```
print(series.describe())

# Discussion:
# -The number of observations (count) matches our expectation, meaning
we are handling the data correctly.
# -The mean is about 60,000,000,  which we might consider our level in
this series.
# -The standard deviation (average spread from the mean) is relatively
large at 3,400,000 exits/day.
# -The percentiles along with the standard deviation do suggest a
large spread to the data.

# Line Plot
# A line plot of a time series can provide a lot of insight into the
problem at hand; This is the plot of our series for
# the studied period; We can visually tell that we don't have any
trend component, but we may have a small seasonal component
series.plot()
pyplot.show()


# Density Plot TODO: Discussion
pyplot.figure(1)
pyplot.subplot(211)
series.hist()
pyplot.subplot(212)
series.plot(kind='kde')
pyplot.show()


# ESTABLISH A BASELINE (PERSISTENCE)
# It is very important to establish a baseline that will allow us to
make comparissons between
# assessment results of different models that we are going to test
# The baseline prediction for time series forecasting is called the
naive forecast, or persistence.
# This is where the observation from the previous time step is used as
the prediction for the observation at the next time step.

# split data in train data and test data
X = series.values
X = X.astype('float32')
train_size = int(len(X) * 0.70)
train, test = X[0:train_size], X[train_size:]

# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
```

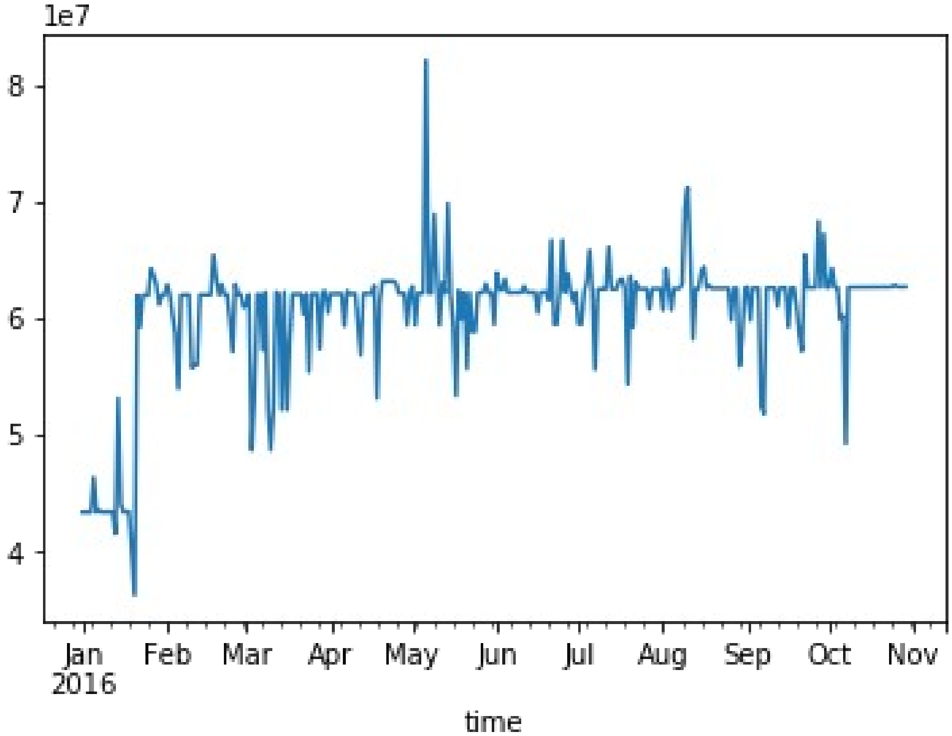
```
# predictions
    yhat = history[-1]
    predictions.append(yhat)
# observations
    obs = test[i]
    history.append(obs)
# print comparisson results
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)

# Result: we get a RMSE = 3376608.131 (2).
# This is a reference value, the worst we can get from a naive
prediction test, where we only used a Moving Average (MA) with
# a lag of 1, the minimum we can set in order to have observed data on
column 't' and predicted data on column 't+1'.

# Our objective is that by using an ARIMA model (AR-Autoreggressive I-
Integrated MA-Moving Average,
# which is a much major improvement compared with a simple MA) to get
a better value (smaller) for RMSE value

# ARIMA MODELS
# In this section, we will develop an Autoregressive Integrated Moving
Average, or ARIMA model,
# for the problem. We will approach this in two steps:
#1. Developing a manually configured ARIMA model.
#2. Analysis of forecast residual errors to evaluate any bias in the
model.

# Manually Configured ARIMA

# Nonseasonal ARIMA(p,d,q) requires 3 parameters and is traditionally
configured manually.
# parameters p, d, and q are non-negative integers, p is the order
(number of time lags) of the autoregressive model,
# d is the degree of differencing (the number of times the data have
had past values subtracted),
# and q is the order of the moving-average(MA)model.
# Analysis of the time series data assumes that we are working with a
stationary time series. The
# time series is in most cases non-stationary. We can make it
stationary by first differencing
# the series (use a MA-Moving Average with a lag of minimum 1 to
detrend the observed data) and using a statistical test to
# confirm that the result is stationary.

# The test used to confirm stationarity of the series will be the
augmented Dickey-Fuller test
```

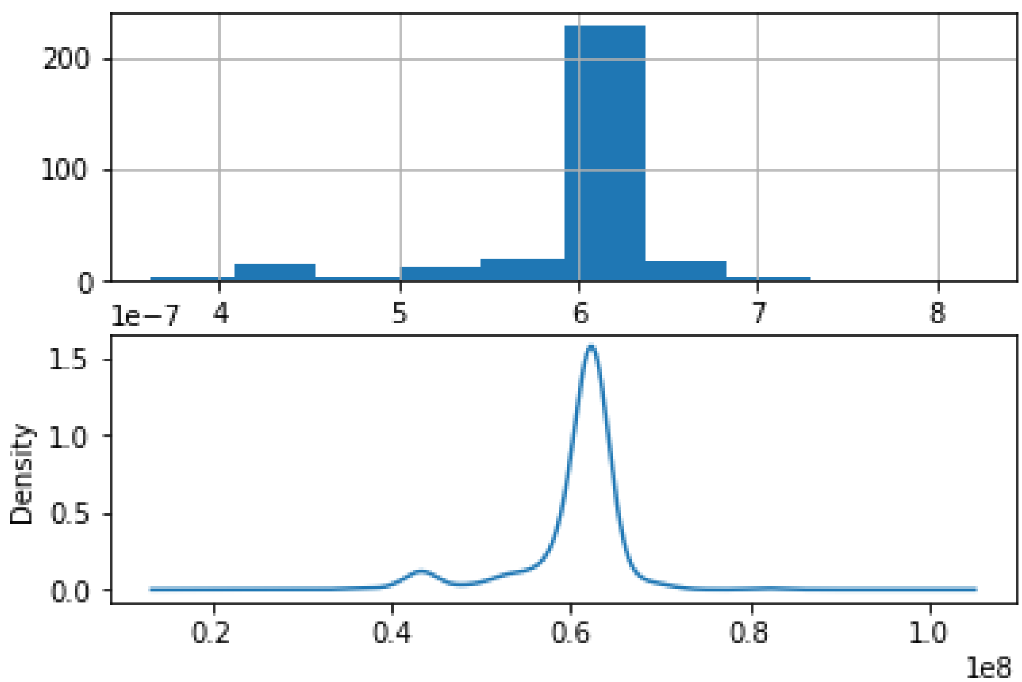
```python
# create a differenced time series
def difference(dataset):
    diff = list()
    for i in range(1, len(dataset)):
        value = dataset[i] - dataset[i - 1] # this is where we form
the MA using a window with width=2 and with lag = 1
        diff.append(value)
    return Series(diff)

X = series.values
# difference data
stationary = difference(X)
stationary.index = series.index[1:]
# check if stationary
result = adfuller(stationary)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.3f' % (key, value))


# Analysis of results:

# We get the following results:
# ADF Statistic: -10.896470
# p-value: 0.000000
# Critical Values:
#    5%: -2.871
#    1%: -3.453
#    10%: -2.572

#The results show that the test statistic value -10.896470 is much
smaller than the critical value at 1% of -3.453. This
#suggests that we can reject the null hypothesis with a significance
level of less than 1% (i.e. a
#very low probability that the result is a statistical fluke).
Rejecting the null hypothesis means that
#the process has no unit root, and in turn, that the 1-lag differenced
time series is stationary or
#does not have any time-dependent structure.

# This suggests that at least one level of differencing is required.
The d parameter in our
# ARMA model should at least be a value of 1. The next step is to
select the lag values for the Autoregression (AR)
# and Moving Average (MA) parameters, p and q respectively. We can do
this by reviewing Autocorrelation Function (ACF)
```

```
# and Partial Autocorrelation Function(PACF) plots.

# ACF and PACF plots of time series
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

pyplot.figure()
pyplot.subplot(211)
plot_acf(series, ax=pyplot.gca())
pyplot.subplot(212)
plot_pacf(series, ax=pyplot.gca())
pyplot.show()

# ARIMA Manual Configuration results for baseline model
# Some observations from the plots:
# The ACF shows a significant lag for 4 months (the part bigger that
0.5 in the black area,before data crosses the light blue area)
# Similarly, the PACF shows a significant lag for 2 months.
# Both the ACF and PACF show a drop-off at almost the same point,
perhaps suggesting a mix of
# AR and MA.
# Therefore, we can state that we could configure a baseline ARIMA
model with the following parameters: ARIMA(4, 1, 2)


# MODEL IMPROVEMENT
# ARIMA final parameters
# This quick analysis suggests an ARIMA(4,1,2) on the raw data may be
a starting point, a baseline for our models.

# We keep the level of diffencing of 1(d = 1) because this is the way
of making predictions using MA, we need the observed data on column
't-1'
# and the predicted values on column 't', therefore we need to always
make at least of diffrence, d = 1 (and we cannot have
# d = 0, otherwise we don't have predictions, we cannot have columns
't' and 't+1' and we don't use a MA for predictions)
# We also want the best (AR)-closer to 0, we will try exactly p = 0
from the graph of (ACF), instead of 4.
# Thus, the model can be simplified to ARIMA(0,1,2), where q =2 means
that we use a Moving Average window with width of 2,
# therefore a lag of 1,a bare minimum to make predictions using MAs.
What is improved here compared to the naive model is
# the fact we use AR = 0 (best Autoregression coefficient)...This
model is even better than the baseline model for ARIMA that we
# stated initially, ARIMA(4,1,2)

# The example below demonstrates the performance of this ARIMA model
using the test walk-forward.
```

```
# Evaluate ARIMA model
from statsmodels.tsa.arima_model import ARIMA

# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # predictions
    model = ARIMA(history, order=(0,1,2))
    model_fit = model.fit(disp=0)
    yhat = model_fit.forecast()[0]
    predictions.append(yhat)
    # observations
    obs = test[i]
    history.append(obs)
    print('>Predicted=%.3f, Expected=%3.f' % (yhat, obs))
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)

# Result analysis:
# We get an RMSE = 2935009.264 which is clearly a much better value
than the previous one of RMSE = 3376608.131
# obtained at (2)

# REVIEW RESIDUAL ERRORS
# A good final check of a model is to review residual forecast errors.
Ideally, the distribution
# of residual errors should be a Gaussian with a zero mean. We can
check this by plotting the
# residuals with a histogram and density plots. The example below
calculates the residual errors
# for predictions on the test set and creates these density plots.

# errors plot
residuals = [test[i]-predictions[i] for i in range(len(test))]
residuals = DataFrame(residuals)
pyplot.figure()
pyplot.subplot(211)
residuals.hist(ax=pyplot.gca())
pyplot.subplot(212)
residuals.plot(kind='kde', ax=pyplot.gca())
pyplot.show()

# Result analysis:
# As we can see from results, we have a very nit Gaussian distribution
of errors, with a mean of 0, which proves that our
# ARIMA(0,1,2) model is much better that the baseline model set up at
(2).
```

```
time
2016-01-01    43350976.0
2016-01-02    43352116.0
2016-01-03    43353412.0
2016-01-04    43354596.0
2016-01-05    46331280.0
Freq: D, Name: exits, dtype: float32
(303,)
count         303.0
mean     60410736.0
std       5556117.0
min      36241712.0
25%      60465052.0
50%      62180684.0
75%      62615474.0
max      82105128.0
Name: exits, dtype: float64
```

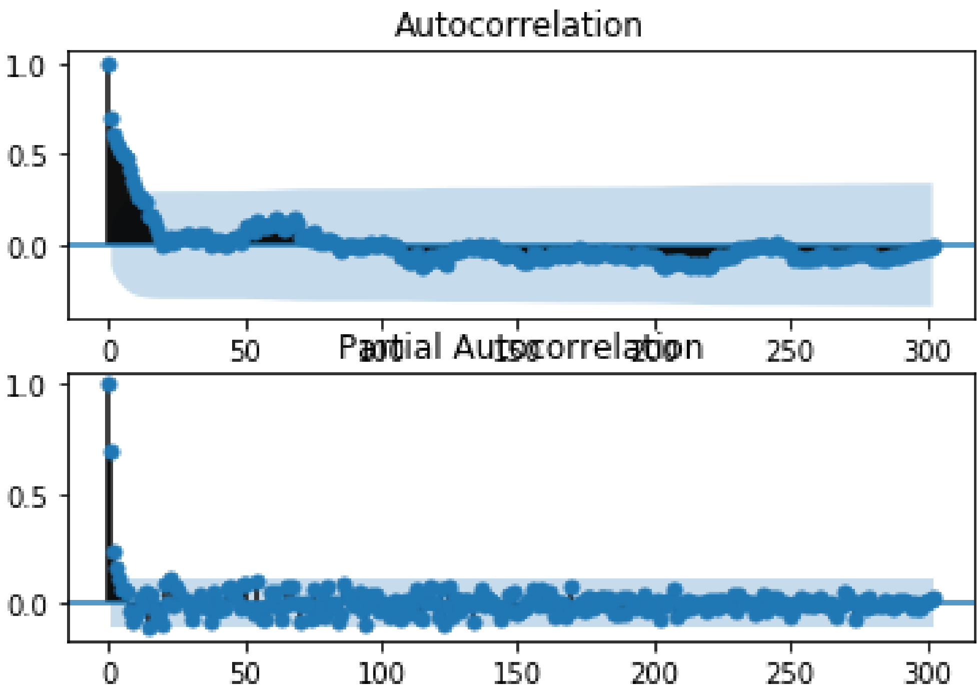





```
>Predicted=62515120.000, Expected=62516320
>Predicted=62516320.000, Expected=60693520
>Predicted=60693520.000, Expected=64227428
>Predicted=64227428.000, Expected=62646696
>Predicted=62646696.000, Expected=60699480
```

```
>Predicted=60699480.000, Expected=62525704
>Predicted=62525704.000, Expected=62527672
>Predicted=62527672.000, Expected=62528876
>Predicted=62528876.000, Expected=62892344
>Predicted=62892344.000, Expected=69548856
>Predicted=69548856.000, Expected=71207192
>Predicted=71207192.000, Expected=65174296
>Predicted=65174296.000, Expected=58205200
>Predicted=58205200.000, Expected=62540116
>Predicted=62540116.000, Expected=62541088
>Predicted=62541088.000, Expected=63913520
>Predicted=63913520.000, Expected=64402220
>Predicted=64402220.000, Expected=62546032
>Predicted=62546032.000, Expected=62777424
>Predicted=62777424.000, Expected=62550052
>Predicted=62550052.000, Expected=62551860
>Predicted=62551860.000, Expected=62552980
>Predicted=62552980.000, Expected=62554104
>Predicted=62554104.000, Expected=62556024
>Predicted=62556024.000, Expected=62558012
>Predicted=62558012.000, Expected=62559992
>Predicted=62559992.000, Expected=59799308
>Predicted=59799308.000, Expected=62563824
>Predicted=62563824.000, Expected=62564920
>Predicted=62564920.000, Expected=55862552
>Predicted=55862552.000, Expected=58371808
>Predicted=58371808.000, Expected=62569828
>Predicted=62569828.000, Expected=62656936
>Predicted=62656936.000, Expected=59810992
>Predicted=59810992.000, Expected=62575488
>Predicted=62575488.000, Expected=62576560
>Predicted=62576560.000, Expected=62577496
>Predicted=62577496.000, Expected=52474472
>Predicted=52474472.000, Expected=51761836
>Predicted=51761836.000, Expected=62582712
>Predicted=62582712.000, Expected=62584772
>Predicted=62584772.000, Expected=62586676
>Predicted=62586676.000, Expected=62587848
>Predicted=62587848.000, Expected=60984124
>Predicted=60984124.000, Expected=62590972
>Predicted=62590972.000, Expected=62592976
>Predicted=62592976.000, Expected=62595080
>Predicted=62595080.000, Expected=59235880
>Predicted=59235880.000, Expected=62599060
>Predicted=62599060.000, Expected=62600144
>Predicted=62600144.000, Expected=60865356
>Predicted=60865356.000, Expected=58526144
>Predicted=58526144.000, Expected=57156364
>Predicted=57156364.000, Expected=65441992
>Predicted=65441992.000, Expected=62609300
```

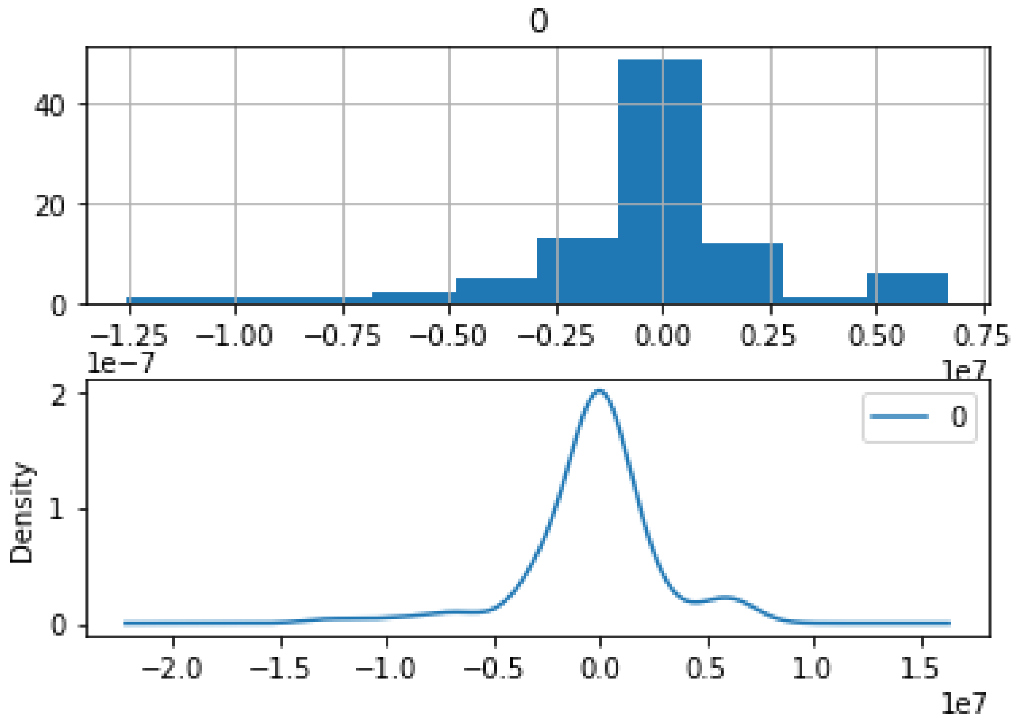
```
>Predicted=62609300.000, Expected=62611212
>Predicted=62611212.000, Expected=62612336
>Predicted=62612336.000, Expected=62613484
>Predicted=62613484.000, Expected=68294680
>Predicted=68294680.000, Expected=62617464
>Predicted=62617464.000, Expected=67240496
>Predicted=67240496.000, Expected=62621568
>Predicted=62621568.000, Expected=62623516
>Predicted=62623516.000, Expected=64277072
>Predicted=64277072.000, Expected=62625836
>Predicted=62625836.000, Expected=62627668
>Predicted=62627668.000, Expected=59866600
>Predicted=59866600.000, Expected=60192660
>Predicted=60192660.000, Expected=49213344
>Predicted=49213344.000, Expected=62635808
>Predicted=62635808.000, Expected=62637028
>Predicted=62637028.000, Expected=62638144
>Predicted=62638144.000, Expected=62640000
>Predicted=62640000.000, Expected=62641964
>Predicted=62641964.000, Expected=62643896
>Predicted=62643896.000, Expected=62646000
>Predicted=62646000.000, Expected=62647956
>Predicted=62647956.000, Expected=62649240
>Predicted=62649240.000, Expected=62650408
>Predicted=62650408.000, Expected=62652352
>Predicted=62652352.000, Expected=62654364
>Predicted=62654364.000, Expected=62656428
>Predicted=62656428.000, Expected=62658488
>Predicted=62658488.000, Expected=62660460
>Predicted=62660460.000, Expected=62661676
>Predicted=62661676.000, Expected=62662872
>Predicted=62662872.000, Expected=62835528
>Predicted=62835528.000, Expected=62666952
>Predicted=62666952.000, Expected=62669112
>Predicted=62669112.000, Expected=62671244
>Predicted=62671244.000, Expected=62672888
RMSE: 3376608.131
ADF Statistic: -10.896470
p-value: 0.000000
Critical Values:
	5%: -2.871
	1%: -3.453
	10%: -2.572
```




```
>Predicted=62577195.937, Expected=62516320
>Predicted=62638768.334, Expected=60693520
>Predicted=61906817.138, Expected=64227428
>Predicted=63165212.636, Expected=62646696
>Predicted=62820451.782, Expected=60699480
>Predicted=62056051.785, Expected=62525704
>Predicted=62544500.153, Expected=62527672
>Predicted=62584375.193, Expected=62528876
>Predicted=62649795.175, Expected=62892344
>Predicted=62845562.960, Expected=69548856
>Predicted=65756487.672, Expected=71207192
>Predicted=67585923.490, Expected=65174296
>Predicted=66055479.261, Expected=58205200
>Predicted=62915628.070, Expected=62540116
>Predicted=63769858.485, Expected=62541088
>Predicted=63381710.932, Expected=63913520
>Predicted=63852649.286, Expected=64402220
>Predicted=64122638.646, Expected=62546032
>Predicted=63456898.667, Expected=62777424
>Predicted=63434088.184, Expected=62550052
>Predicted=63217940.723, Expected=62551860
>Predicted=63118741.828, Expected=62552980
>Predicted=63036805.704, Expected=62554104
>Predicted=62978142.966, Expected=62556024
>Predicted=62935277.639, Expected=62558012
>Predicted=62904143.545, Expected=62559992
>Predicted=62881580.895, Expected=59799308
>Predicted=61653396.810, Expected=62563824
>Predicted=62493631.010, Expected=62564920
```

```
>Predicted=62503968.738, Expected=55862552
>Predicted=59662013.016, Expected=58371808
>Predicted=59932288.282, Expected=62569828
>Predicted=61342685.055, Expected=62656936
>Predicted=61686130.665, Expected=59810992
>Predicted=60771747.636, Expected=62575488
>Predicted=61858486.921, Expected=62576560
>Predicted=62033950.931, Expected=62577496
>Predicted=62257693.580, Expected=52474472
>Predicted=57978156.871, Expected=51761836
>Predicted=56413926.186, Expected=62582712
>Predicted=60153403.561, Expected=62584772
>Predicted=60507956.587, Expected=62586676
>Predicted=61161531.571, Expected=62587848
>Predicted=61568088.650, Expected=60984124
>Predicted=61163448.453, Expected=62590972
>Predicted=61957616.034, Expected=62592976
>Predicted=62112091.953, Expected=62595080
>Predicted=62308224.398, Expected=59235880
>Predicted=60923744.918, Expected=62599060
>Predicted=62185658.488, Expected=62600144
>Predicted=62211221.442, Expected=60865356
>Predicted=61615972.285, Expected=58526144
>Predicted=60486666.992, Expected=57156364
>Predicted=59492460.381, Expected=65441992
>Predicted=62728126.412, Expected=62609300
>Predicted=61891759.870, Expected=62611212
>Predicted=62287514.082, Expected=62612336
>Predicted=62397539.032, Expected=62613484
>Predicted=62516062.508, Expected=68294680
>Predicted=65146162.179, Expected=62617464
>Predicted=63264145.725, Expected=67240496
>Predicted=65438888.130, Expected=62621568
>Predicted=63748356.072, Expected=62623516
>Predicted=63700892.049, Expected=64277072
>Predicted=64185274.083, Expected=62625836
>Predicted=63502166.125, Expected=62627668
>Predicted=63403916.133, Expected=59866600
>Predicted=62045235.754, Expected=60192660
>Predicted=61780071.822, Expected=49213344
>Predicted=56435845.207, Expected=62635808
>Predicted=60861695.575, Expected=62637028
>Predicted=60988331.841, Expected=62638144
>Predicted=61532858.964, Expected=62640000
>Predicted=61852782.755, Expected=62641964
>Predicted=62107796.159, Expected=62643896
>Predicted=62297381.402, Expected=62646000
>Predicted=62440887.324, Expected=62647956
>Predicted=62549064.160, Expected=62649240
>Predicted=62630479.226, Expected=62650408
```

```
>Predicted=62691832.576, Expected=62652352
>Predicted=62738436.778, Expected=62654364
>Predicted=62773815.243, Expected=62656428
>Predicted=62800785.226, Expected=62658488
>Predicted=62821413.209, Expected=62660460
>Predicted=62837233.916, Expected=62661676
>Predicted=62849129.057, Expected=62662872
>Predicted=62858216.057, Expected=62835528
>Predicted=62938846.382, Expected=62666952
>Predicted=62890925.691, Expected=62669112
>Predicted=62896663.641, Expected=62671244
>Predicted=62894504.583, Expected=62672888
RMSE: 2935009.264
```



2. Conclussions on the model developed

We can answer now to the second question of the Modeling section of the challenge. Thus, we developed an ARIMA model which uses a regressed Moving Average with a lag of 1, in a Moving Average with a window widh of 2, to make predictions of future values of 'exits' for a certain station. line and turnstile. The ARIMA model was tested against a baseline model created only based on a MA with lag =1. The features used in the model were a Moving Average based on which the predictions were made and a RSME metric to compare the errors (predicted, observed) values between the ARIMA model and the MA model. The RMSE for the ARIMA model was clearly smaller that the RSME for the MA model, which shows an improvement in the way of making predictions using the ARIMA model.