In [131]:
```
import numpy as np
import pandas as pd
import os
from dateutil.parser import parse
import matplotlib.pyplot as plt
from math import radians, cos, sin, asin, sqrt
from scipy.stats import pearsonr
from sklearn.linear_model import LinearRegression as LR
import re
import random
from  datetime import datetime
from datetime import timedelta
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import logging
from sklearn.ensemble import RandomForestRegressor as RF
import warnings
warnings.filterwarnings('ignore')
```

In [111]:
```
data_path = 'data/Year1/'
home_paths = ['Home1_yr1.csv', 'Home2_yr1.csv', 'Home3_yr1.csv', 'Home4_yr1.csv', 'Home5_yr1.csv', \
              'Home6_yr1.csv', 'Home7_yr1.csv', 'Home8_yr1.csv', 'Home9_yr1.csv', 'Home10_yr1.csv']

TIME_BEGIN = datetime(2014,12,1)
TIME_END = datetime(2015,12,1)
TIME_DELTA = timedelta(0,0,0,0,15) #15 minutes delta

MIN_INPUT = 96 #entries in a day
MAX_INPUT = 34944 #max_minutes - 1 day
DAY_MINUTES = 96 #entry count in a day
```

In [134]:
```
def get_data(path, count=None):
    train_df =  pd.read_csv(path, nrows=count, header=None)
    return train_df
```

In [135]:
```
home_dfs = []
for home_path in home_paths:
    home_dfs.append (get_data(data_path + home_path))
```

In [136]:
```
#building time features
curtime = TIME_BEGIN


month_list = []
day_list = [] #day of month
weekday_list = []
minute_list = []


while curtime < TIME_END:
    month_list.append(curtime.month)
    day_list.append(curtime.day)
    weekday_list.append(curtime.isoweekday())
    minute_list.append(curtime.hour*60 + curtime.minute)

    curtime += TIME_DELTA
```

In [101]:
```
#enrich data with time features
#features considered : month, day_of_month, day_of_week, minute
date_df = pd.DataFrame()
date_df['month'] = month_list
date_df['day'] = day_list
date_df['weekday'] = weekday_list
date_df['minute'] = minute_list
```

In [102]:
```
X = date_df
```

In [112]:
```
timeslot = int(input('Input Timeslot in entry index: '))

if timeslot > MAX_INPUT:
    timeslot = MAX_INPUT

if timeslot < MIN_INPUT:
    timeslot = MIN_INPUT

print ('Timeslot considered is: ' + str(timeslot))
```

```
Input Timeslot in entry index: 1232354235
Timeslot considered is: 34944
```

In [ ]:
```
X_train = X[:timeslot]
X_test = X[timeslot:timeslot+DAY_MINUTES]
```

In [129]:
```
# Linear Regression
print ('MODEL : LINEAR REGRESSION')
linReg = LR (normalize=True, n_jobs=-1) #n_jobs=-1 uses all the cpus

i = 1
for home in home_dfs:
    Y_train = home[0][:timeslot]
    Y_test = home[0][timeslot:timeslot+DAY_MINUTES]

    linReg.fit(X_train,Y_train)
    Y_pred_linear = linReg.predict(X_test)
    print ('MAE: House[' + str(i) + '], ' + str(mean_absolute_error(Y_pred_linear, Y_test)))
    i += 1
```

```
MODEL : LINEAR REGRESSION
MAE: House[1], 1.3842771912426108
MAE: House[2], 1.6575772741759216
MAE: House[3], 1.757195698276184
MAE: House[4], 0.8508289271089308
MAE: House[5], 1.6159036257161234
MAE: House[6], 1.0995928273446747
MAE: House[7], 1.7165909204053094
MAE: House[8], 1.6850882502435782
MAE: House[9], 1.264611838218349
MAE: House[10], 1.3748836072516915
```

In [132]:
```
print ('MODEL : RANDOM FOREST')
rand_forest_reg = RF(random_state=0, n_jobs = -1, n_estimators = 10, oob_score = True)

i = 1
for home in home_dfs:
    Y_train = home[0][:timeslot]
    Y_test = home[0][timeslot:timeslot+DAY_MINUTES]
    rand_forest_reg.fit (X_train, Y_train)
    Y_pred_rf = rand_forest_reg.predict(X_test)
    print ('MAE: House[' + str(i) + '], ' + str(mean_absolute_error(Y_pred_rf, Y_test)))
    i += 1
```

```
MODEL : RANDOM FOREST
MAE: House[1], 1.1018142677083334
MAE: House[2], 1.4142444208333333
MAE: House[3], 1.0788536979166667
MAE: House[4], 0.8294385572916667
MAE: House[5], 1.0180408385416666
MAE: House[6], 1.0075472694444445
MAE: House[7], 1.0564943020833333
MAE: House[8], 1.2987328552083335
MAE: House[9], 0.7396080083333333
MAE: House[10], 1.1589714322916664
```