Hackathon_MidPrice_Prediction

In [39]:

```
#[DONOTCHANGE]
#1st step - Download files from Google drive
#Manual download from https://drive.google.com/open?
id=1lExwruiCpiOwgE_ijiZ1l6sIYbHnt5v_
import os
import urllib.request

def DownloadZipFiles():
  print('Download zip file to local drive')
  !rm -rf intraday.zip
  !rm -rf intraday
  urllib.request.urlretrieve("https://drive.google.com/uc?
authuser=0&id=1lExwruiCpiOwgE_ijiZ1l6sIYbHnt5v_&export=download", "intraday
.zip")
  !unzip intraday.zip -d .

def FilesExists():
  if not os.path.exists('./intraday'):
    return False
  fileList = ['quote_in.csv', 'quote_out.csv', 'trade_in.csv',
'trade_out.csv']
  listDir = os.listdir('./intraday')
  listDir.sort()
  return listDir == fileList

try:
  if not FilesExists():
    DownloadZipFiles()
  else:
    print('Files exist')
except Exception as ex:
  print('File download from google drive failed')
```

```
Download zip file to local drive
```

```
'rm' is not recognized as an internal or external command,
operable program or batch file.
'rm' is not recognized as an internal or external command,
operable program or batch file.
'unzip' is not recognized as an internal or external command,
operable program or batch file.
```

In [40]:

```
#2nd step - Install all required libraries
"""
!pip install pandas
!pip install pytz
"""
```

Out[40]:

```
'\n!pip install pandas\n!pip install pytz\n'
```

In [41]:

```
#[DONOTCHANGE]
#3rd step - all parameters
class Parameters(object):
    pass

param = Parameters()
param.tickSize = 0.5 #tick size is 0.5 GBp i.e. 0.005 GBP

param.fileDirectory = './intraday'

param.trade_InSampleFile = 'trade_in.csv'
param.quote_InSampleFile = 'quote_in.csv'

param.trade_OutSampleFile = 'trade_out.csv'
param.quote_OutSampleFile = 'quote_out.csv'
```

In [42]:

```
#4th step - Model specific parameters
param.imbalanceThreshold = 0.7
param.timeDuration = 30 #30 seconds
```

In [43]:

```
#Initialise libraries and functions
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import datasets, linear_model
from math import sqrt
import os
import numpy as np
import pandas as pd
import pickle

#Disable certain warnings
pd.options.mode.chained_assignment = None
```

In [44]:

```
# INNCOMMING DATAFRAME HAS
# colunms=
['datetime','sym',ask','bid','asize','bsize','mid','midChangeGroup']  #
def IdentifyFutureMidPrices(df,predictionDuration = '30S'):

 # downSampling by 30seconds#
    futureData = df.resample(predictionDuration, on = 'datetime').first()

# shiftup by one index #
    futureData = futureData.shift(periods=-1)

# drop all columns but mid and preMid and inplace
    futureData.drop(columns = ['datetime', 'sym', 'bsize', 'bid', 'ask',
'asize'], inplace = True)

# rename column "mid" as "futMid" #
    futureData.rename(columns = {"mid":"futMid"}, inplace = True)

# reset the index,inplace #
```

```
    futureData.reset_index(inplace = True)

# This is similar to a left-join except that we match on nearest key rather
than equal keys. #
    return pd.merge_asof(df, futureData[['datetime', 'futMid']],
on='datetime')

# OUTGOING DATAFRAME HAS
# colunms=
['datetime','sym_X','sym_Y',ask','bid','asize','bsize',futMid','midChangeGr
']  #
```

In [45]:

```
# This method reads CSV file and returns pandas DataFrame object
def ReadCSV(file):
    print('Loading file - ' + file)
    df = pd.read_csv(file)
    df['datetime'] = pd.to_datetime(df['datetime'], format="%Y-
%m-%dD%H:%M:%S.%f")
    return df
```

In [46]:

```
#Load data
def LoadData(path, tradeFile, quoteFile):
    tradeFile = os.path.join(path, tradeFile)
    quoteFile = os.path.join(path, quoteFile)

# CSV to DataFrame #
# colunms=['datetime','sym',ask','bid','asize','bsize']  #
    quote_df = ReadCSV(quoteFile)

# colunms=['datetime','sym','price','size']  #
    trade_df = ReadCSV(tradeFile)

# adding "mid" column to DataFrame #
    quote_df['mid'] = 0.5*(quote_df['bid'].copy() + quote_df['ask'].copy())

# adding "midChangeGroup" column to DataFrame #
# cumulative sum of changes in "mid" column of DataFrame #
    quote_df['midChangeGroup'] = quote_df['mid'].diff().ne(0).cumsum()

    quote_df = IdentifyFutureMidPrices(quote_df)
    print('Files loaded')

    return trade_df, quote_df
# colunms=
['datetime','sym',ask','bid','asize','bsize','mid,'futMid','midChangeGroup'
#
```

In [47]:

```
# INCOMING DATAFRAME HAS
# colunms=
['datetime','sym',ask','bid','asize','bsize','futMid','midChangeGroup','pre
d']  #
```

```
# this method calculates the root mean square error
def RMS(df):
    df = df.groupby(['midChangeGroup']).first().reset_index()
    tmp = df.dropna(subset=['predMid', 'futMid'])
    rms = sqrt(mean_squared_error(tmp['futMid'], tmp['predMid']))
    predCount = len(tmp['predMid'])
    print('RMS = %.4f. #Predictions = %s' % (rms, predCount))
```

In [48]:

```
# INCOMING DATAFRAME HAS
# colunms=
['datetime','sym',ask','bid','asize','bsize','futMid','midChangeGroup']  #

def InSamplePredictionModel(quote_df, trade_df):
#Load pickle to predict

    pickleModel = open('Model.pkl','rb')
    Model = pickle.load(pickleModel)
    print('Prediction model')

# adding new features "spread","microMid" to the model
    quote_df['spread']=quote_df['bid'].copy()-quote_df['ask'].copy()
    quote_df['microMid']=(quote_df['bid'].copy()*quote_df['bsize'].copy()+q
uote_df['ask'].copy()*quote_df['asize'].copy()) /(quote_df.copy()['bsize']+
quote_df['asize'].copy())

# Now merging quote_df and trade_df
    Z=pd.merge_asof(quote_df,trade_df,on ='datetime')
# dropping NaN values from DataFrame
    Z.dropna(axis=0,how='any',inplace = True)
# resetting the DataFrame index
    Z.reset_index(drop=True,inplace=True)

# selecting features and labels from DataFrame
# X_test -->features,Y_test-->label


X_test=Z.as_matrix(columns=['ask','bid','asize','bsize','midChangeGroup','s
ize','price','mid','microMid','spread'])
    Y_test=Z.as_matrix(columns=['futMid'])

# predicting values as trainned Model
    pred=Model.predict(X_test)
# new column "predMid" added to DataFrame
    Z['predMid']=pred

    return Z
```

In [49]:

```
# Load the CSV if not in memory
dirContents = dir()
if not ('tradeIndf' in dirContents and 'quoteIndf' in dirContents):
    tradeIndf, quoteIndf = LoadData(param.fileDirectory,
param.trade_InSampleFile, param.quote_InSampleFile)
```

Tranning Code

In [50]:

```
def MachineLearningModel(quote_df, trade_df):
# adding new features "spread","microMid" to the model
    quote_df['spread']=quote_df['bid'].copy()-quote_df['ask'].copy()
    quote_df['microMid']=(quote_df['bid'].copy()*quote_df['bsize'].copy()+q
uote_df['ask'].copy()*quote_df['asize'].copy()) /(quote_df.copy()['bsize']+
quote_df['asize'].copy())

# Now merging quote_df and trade_df
    Z=pd.merge_asof(quote_df,trade_df,on ='datetime')
# dropping NaN values from DataFrame
    Z.dropna(axis=0,how='any',inplace = True)
# resetting the DataFrame index
    Z.reset_index(drop=True,inplace=True)

# selecting features and labels from DataFrame
# X_test -->features,Y_test-->label

X_train=Z.as_matrix(columns=['ask','bid','asize','bsize','midChangeGroup','
size','price','mid','microMid','spread'])
    Y_train=Z.as_matrix(columns=['futMid'])

# Using LinearRegression we train model
    Model=linear_model.LinearRegression()
    Model.fit(X_train,Y_train)
    # pickle file
    tempPickle = 'Model.pkl'
    pickleModel= open(tempPickle,'wb')
    pickle.dump(Model,pickleModel)
    pickleModel.close()
```

As model has been trained using InSample Files and saved as "Model.pkl" file.No need to Train Model using InSample file again and again

In [55]:

```
# Tranning the model using InSample Files
# we can comment this line if you don't want to use Insample File again
MachineLearningModel(quoteIndf,tradeIndf)
```

In [52]:

```
# Load the out-sample csv if not in memory
dirContents = dir()
if not ('tradeOutdf' in dirContents and 'quoteOutdf' in dirContents):
    tradeOutdf, quoteOutdf = LoadData(param.fileDirectory,
param.trade_OutSampleFile, param.quote_OutSampleFile)
```

In [53]:

```
# predicts the value for OutSample Files
def OutSamplePrediction(quote_df, trade_df):
    print('Out-sample prediction')
    return InSamplePredictionModel(quote_df, trade_df)
```

In [54]:

```
# calculating Root Mean Square Error
res = OutSamplePrediction(quoteOutdf, tradeOutdf)
RMS(res)
```

```
Out-sample prediction
Prediction model
RMS = 2.2354. #Predictions = 53135
```

Summary

In this problem we have to predict future midPrice of the stock using historical data.So this problem fall in the category of continuous analysis.

As data is given in CSV format so no much of cleaning and formatting of data is required,it is ready to use format.

The data of the stock market given follows the linear relationship. This can be depiced by observing the visual plotting of the data. From the plotted graph, we can conclude that the regressors following the linear model are the best models that can be fitted.

Some new features are created in the data which can be very help in predicting the future values.

Scikit Learn library provides the machine learning models and also various functions for calculation related to it.

It provides a function to check the performance of different regressors that can be fit as the model. So using that Linear Regressor was performing best over the given data.

SGD Regressor can be used as the training set is quite large but it was lacking in performance. So Linear Regressor was the best fit midel for this system.

Prediction model

RMS = 2.2354

Predictions = 53135

# happy Hackathon!!