In this competition the challenge is to predict the return of a stock, given the history of the past few days.

We provide 5-day windows of time, days D-2, D-1, D, D+1, and D+2. You are given returns in days D-2, D-1, and part of day D, and you are asked to predict the returns in the rest of day D, and in days D+1 and D+2.

During day D, there is intraday return data, which are the returns at different points in the day. We provide 180 minutes of data, from t=1 to t=180. In the training set you are given the full 180 minutes, in the test set just the first 120 minutes are provided.

For each 5-day window, we also provide 25 features, Feature_1 to Feature_25. These may or may not be useful in your prediction.

Each row in the dataset is an arbitrary stock at an arbitrary 5 day time window.

File descriptions

train.csv - the training set, including the columns of:

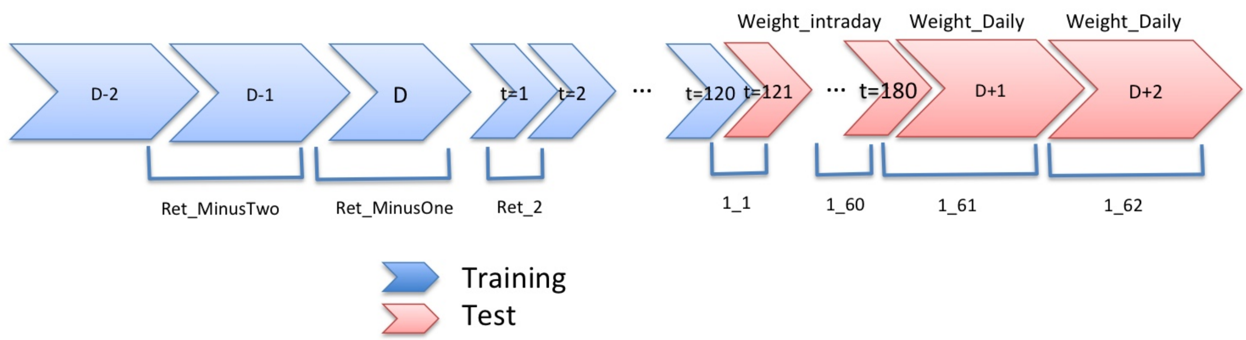
Feature_1 - Feature_25
Ret_MinusTwo, Ret_MinusOne
Ret_2 - Ret_120
Ret_121 - Ret_180: target variables
Ret_PlusOne, Ret_PlusTwo: target variables
Weight_Intraday, Weight_Daily

test.csv - the test set, including the columns of:

Feature_1 - Feature_25
Ret_MinusTwo, Ret_MinusOne
Ret_2 - Ret_120

sample_submission.csv - a sample submission file in the correct format

Graphical illusion of Data



ref: https://storage.googleapis.com/kaggle-competitions/kaggle/4504/media/Presentation1%20(1).jpg (https://storage.googleapis.com/kaggle-competitions/kaggle/4504/media/Presentation1%20(1).jpg)

Data fields

Feature_1 to Feature_25: different features relevant to prediction

Ret_MinusTwo: this is the return from the close of trading on day D-2 to the close of trading on day D-1 (i.e. 1 day)

Ret_MinusOne: this is the return from the close of trading on day D-1 to the point at which the intraday returns start on day D (approximately 1/2 day)

Ret_2 to Ret_120: these are returns over approximately one minute on day D. Ret_2 is the return between t=1 and t=2.

Ret_121 to Ret_180: intraday returns over approximately one minute on day D. These are the target variables you need to predict as {id}_{1-60}.

Ret_PlusOne: this is the return from the time Ret_180 is measured on day D to the close of trading on day D+1. (approximately 1 day). This is a target variable you need to predict as {id}_61.

Ret_PlusTwo: this is the return from the close of trading on day D+1 to the close of trading on day D+2 (i.e. 1 day) This is a target variable you need to predict as {id}_62.

Weight_Intraday: weight used to evaluate intraday return predictions Ret 121 to 180

Weight_Daily: weight used to evaluate daily return predictions (Ret_PlusOne and Ret_PlusTwo).

Metric

Submissions to this competition are judged on weighted mean absolute error, where each return being predicted is compared with the actual return, scaled by its corresponding weight:

$WMAE = 1/n \cdot \Sigma wi \cdot |yi - \hat{y}i\ |$

where wi is the weight associated with the return, Weight_Intraday, Weight_Daily for intraday and daily returns, i, yi is the predicted return, $\hat{y}i$ is the actual return, n is the number of predictions.

The weights for the training set are given in the training data. The weights for the test set are unknown.

In [0]:
```
!pip install transform
```

```
Collecting transform
  Downloading https://files.pythonhosted.org/packages/94/43/34b101ebf41dc26eb
5e4dd2dce0692db367c09b05519c9bb17eb122a454e/Transform-0.0.1.tar.gz
Building wheels for collected packages: transform
  Building wheel for transform (setup.py) ... done
  Created wheel for transform: filename=Transform-0.0.1-cp36-none-any.whl siz
e=2657 sha256=ac6069f1856d464ab4c6d4506777b6bb022465d7da482b8527cd7233d5a312a
c
  Stored in directory: /root/.cache/pip/wheels/b2/3c/4c/54869afb2ddfcf16134da
5e71a27070234029bc999816c3683
Successfully built transform
Installing collected packages: transform
Successfully installed transform-0.0.1
```

In [6]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import Imputer,StandardScaler, MinMaxScaler
import matplotlib.pyplot as plt
from progressbar import ProgressBar
import xgboost as xgb

from sklearn import ensemble
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression

from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, LSTM, Masking
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from keras.callbacks import EarlyStopping, ModelCheckpoint,ReduceLROnPlateau

import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

Colab session time check

In [7]:
```
import time, psutil
uptime = time.time() - psutil.boot_time()
remain = 12*60*60 - uptime
print(remain/3600)
```

```
11.81664834075504
```

In [8]:
```
from google.colab import files
files.upload()
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

In [9]:
```
# downloading the kaggle data
!kaggle competitions download -c the-winton-stock-market-challenge
```

```
Warning: Looks like you're using an outdated API Version, please consider upd
ating (server 1.5.6 / client 1.5.4)
Downloading train.csv.zip to /content
 95% 69.0M/72.7M [00:01<00:00, 28.8MB/s]
100% 72.7M/72.7M [00:01<00:00, 41.9MB/s]
Downloading sample_submission_2.csv.zip to /content
 56% 9.00M/16.1M [00:01<00:00, 11.7MB/s]
100% 16.1M/16.1M [00:01<00:00, 15.1MB/s]
Downloading test_2.csv.zip to /content
 92% 105M/114M [00:01<00:00, 57.3MB/s]
100% 114M/114M [00:01<00:00, 60.8MB/s]
```

In [10]:
```
!unzip train.csv.zip
!unzip test_2.csv.zip
!unzip sample_submission_2.csv.zip
```

```
Archive:  train.csv.zip
  inflating: train.csv
   creating: __MACOSX/
  inflating: __MACOSX/._train.csv
Archive:  test_2.csv.zip
  inflating: test_2.csv
  inflating: __MACOSX/._test_2.csv
Archive:  sample_submission_2.csv.zip
  inflating: sample_submission_2.csv
  inflating: __MACOSX/._sample_submission_2.csv
```

Exploratory data analysis and Data preprocessing

In [0]:
```
from prettytable import PrettyTable
conc = PrettyTable(["Model Name", "MSE for daily returns", "MSE for intra day
 returns"])
```

In [11]:
```
files_path=''
df=pd.read_csv(files_path+"train.csv")
df.head()
```

Out[11]:

	Id	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	F
0	1	NaN	NaN	NaN	NaN	8.0	NaN	75751	0.2254	
1	2	NaN	NaN	NaN	NaN	3.0	0.388896	17369	0.0166	
2	3	NaN	-0.696727	0.739591	-0.167928	9.0	0.471947	8277	0.3650	
3	4	NaN	-0.694350	1.568248	0.479073	5.0	0.120653	22508	0.2654	
4	5	6.0	-1.736489	2.765531	1.245280	7.0	4.866985	22423	0.2138	

5 rows × 211 columns

In [0]:
```
# the shape of train data
df.shape
```

Out[0]: (40000, 211)

In [0]:
```
# https://stackoverflow.com/questions/26266362/how-to-count-the-nan-values-in-a-column-in-pandas-dataframe
count_nan = len(df) - df.count()
print(count_nan[:5])
print(df.count()[:5])
```

```
Id               0
Feature_1    33313
Feature_2     9146
Feature_3     1237
Feature_4     7721
dtype: int64
Id           40000
Feature_1     6687
Feature_2    30854
Feature_3    38763
Feature_4    32279
dtype: int64
```

In [0]:
```
# as we can see there are many nan values so replacing the values with mean
len(count_nan)
```

Out[0]: 211

In [0]:
```
# https://medium.com/better-programming/handling-missing-data-in-python-using-
scikit-imputer-7607c8957740

#  replacing NaN values

imputer = Imputer(missing_values="NaN", strategy="mean", axis = 0)
imputer = imputer.fit(df.values)
df_ar = imputer.transform(df)

# imputer will return the numpy arrays
df.fillna(df.mean(axis=0), inplace=True)
# we are using the above line because it reatains the datatype as Dataframe an
d in future code
#    it will be easy to access through column names
```

In [0]:
```
df_ar[:5]
```

Out[0]:
```
array([[ 1.00000000e+00,  3.59024974e+00, -1.17557876e-01, ...,
         2.88463645e-02,  1.25150797e+06,  1.56438496e+06],
       [ 2.00000000e+00,  3.59024974e+00, -1.17557876e-01, ...,
        -1.02532062e-02,  1.73395035e+06,  2.16743794e+06],
       [ 3.00000000e+00,  3.59024974e+00, -6.96726938e-01, ...,
         1.57107386e-02,  1.52919738e+06,  1.91149673e+06],
       [ 4.00000000e+00,  3.59024974e+00, -6.94349689e-01, ...,
        -2.19045788e-03,  1.71156942e+06,  2.13946178e+06],
       [ 5.00000000e+00,  6.00000000e+00, -1.73648913e+00, ...,
        -2.65516002e-02,  1.26727026e+06,  1.58408783e+06]])
```

In [0]:
```
df.head()
```

Out[0]:

	Id	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	F
0	1	3.59025	-0.117558	0.558392	0.405572	8.0	0.430972	75751	0.2254	
1	2	3.59025	-0.117558	0.558392	0.405572	3.0	0.388896	17369	0.0166	
2	3	3.59025	-0.696727	0.739591	-0.167928	9.0	0.471947	8277	0.3650	
3	4	3.59025	-0.694350	1.568248	0.479073	5.0	0.120653	22508	0.2654	
4	5	6.00000	-1.736489	2.765531	1.245280	7.0	4.866985	22423	0.2138	

5 rows × 211 columns

In [0]:
```
df.describe()
```

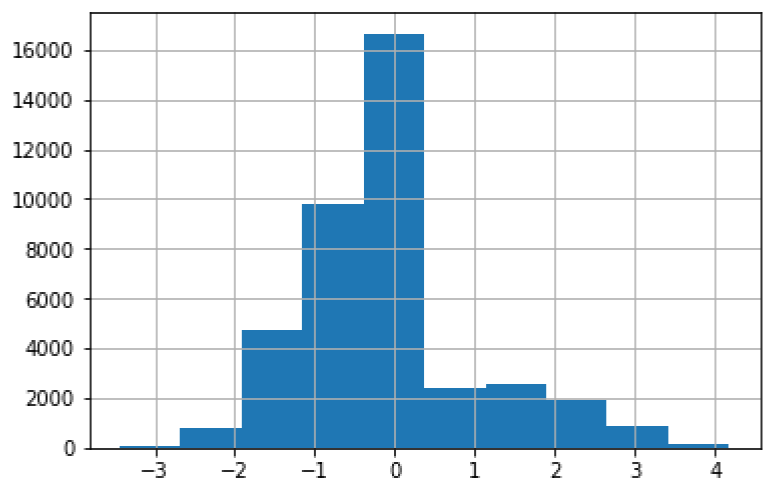
Out[0]:

	Id	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	F
count	40000.00000	40000.000000	40000.000000	40000.000000	40000.000000	40000.000000	4000
mean	20000.50000	3.590250	-0.117558	0.558392	0.405572	5.482775	
std	11547.14972	1.144166	1.085751	0.888172	0.717828	2.942324	
min	1.00000	1.000000	-3.440521	-4.643526	-5.440596	1.000000	-
25%	10000.75000	3.590250	-0.791707	-0.092559	0.014992	2.000000	-
50%	20000.50000	3.590250	-0.117558	0.475221	0.405572	6.000000	
75%	30000.25000	3.590250	0.091542	1.037897	0.804371	8.000000	
max	40000.00000	10.000000	4.175150	4.530405	2.953163	10.000000	1

8 rows × 211 columns

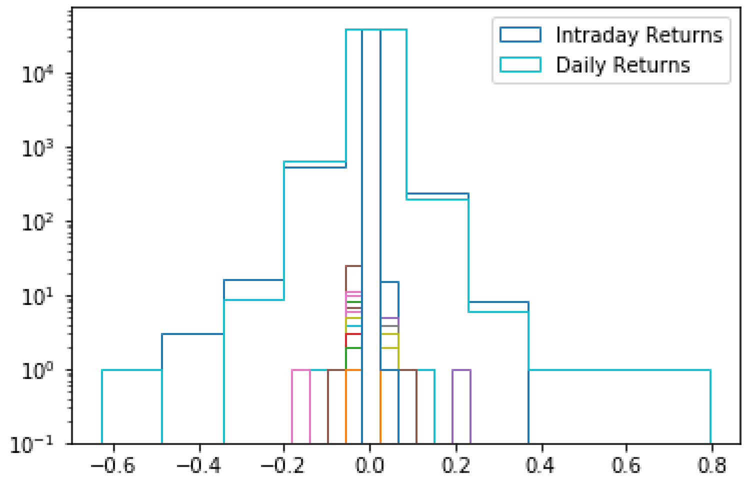
In [0]:
```
df['Feature_2'].hist()
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f28e1db2cf8>

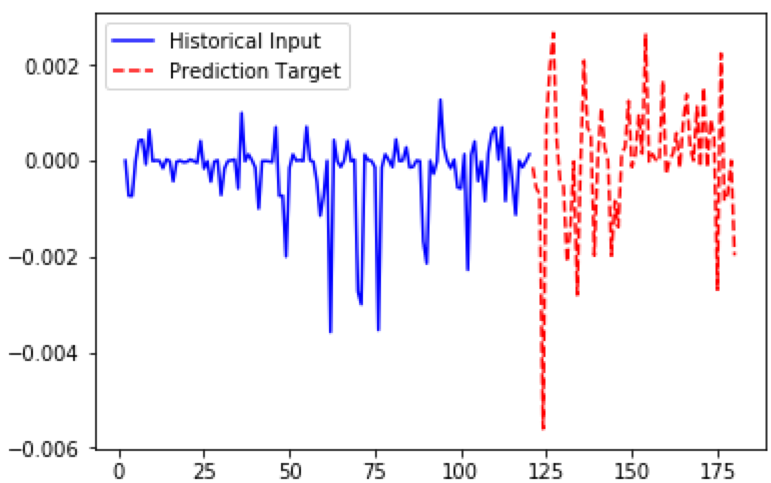


PLotting Intra day returns and Faily Returns

In [0]:
```
# https://github.com/higgsb0/winton-stock/blob/master/stock-prediction.html
plt.hist(df.loc[:, 'Ret_2':'Ret_180'].values, log=True,
        histtype='step', label='Intraday Returns')
plt.hist(df.loc[:,'Ret_PlusOne':'Ret_PlusTwo'].values, log=True,
        histtype='step', label='Daily Returns')
plt.legend()
plt.show()
```



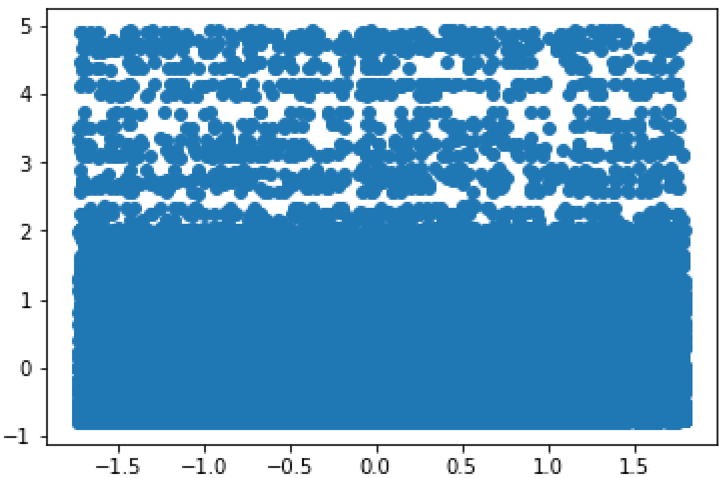
In [0]:
```
# https://github.com/higgsb0/winton-stock/blob/master/stock-prediction.html
plt.plot(range(2,121),df.iloc[0]['Ret_2':'Ret_120'], 'b-', label='Historical I
nput')
plt.plot(range(121,181),df.iloc[0]['Ret_121':'Ret_180'], 'r--', label='Predict
ion Target')
plt.legend()
plt.show()
```



Applying PCA

In [0]:
```
%%time
%matplotlib inline

from sklearn.decomposition import PCA
pca = PCA(n_components=5,whiten=True)
X = df.loc[:,'Feature_1':'Feature_25'].values
X_r = pca.fit_transform(X)
plt.scatter(X_r[:,0], X_r[:,1])
```

```
CPU times: user 320 ms, sys: 211 ms, total: 531 ms
Wall time: 331 ms
```



In [0]:
```
pca.explained_variance_
```

Out[0]:
```
array([7.97633707e+08, 2.39965424e+01, 1.43961516e+01, 7.75598275e+00,
       6.71316629e+00])
```

In [0]:
```
submission_df=pd.read_csv(files_path+"sample_submission_2.csv")
submission_df.head()
```

Out[0]:

	Id	Predicted
0	1_1	0
1	1_2	0
2	1_3	0
3	1_4	0
4	1_5	0

In [0]:
```
submission_df['Id'].values[0:63]
```

Out[0]:
```
array(['1_1', '1_2', '1_3', '1_4', '1_5', '1_6', '1_7', '1_8', '1_9',
       '1_10', '1_11', '1_12', '1_13', '1_14', '1_15', '1_16', '1_17',
       '1_18', '1_19', '1_20', '1_21', '1_22', '1_23', '1_24', '1_25',
       '1_26', '1_27', '1_28', '1_29', '1_30', '1_31', '1_32', '1_33',
       '1_34', '1_35', '1_36', '1_37', '1_38', '1_39', '1_40', '1_41',
       '1_42', '1_43', '1_44', '1_45', '1_46', '1_47', '1_48', '1_49',
       '1_50', '1_51', '1_52', '1_53', '1_54', '1_55', '1_56', '1_57',
       '1_58', '1_59', '1_60', '1_61', '1_62', '2_1'], dtype=object)
```

In [0]:
```
submission_df.shape
# there are 120000 id's and  62 return values for each id 62 *120000 = 7440000
```

Out[0]:
```
(7440000, 2)
```

Models

Data Preprocessing

In [0]:
```
first = ['Id']
allcol = ['Ret_121','Ret_122','Ret_123','Ret_124','Ret_125','Ret_126','Ret_12
7','Ret_128','Ret_129','Ret_130',
'Ret_131','Ret_132','Ret_133','Ret_134','Ret_135','Ret_136','Ret_137','Ret_13
8','Ret_139','Ret_140','Ret_141',
'Ret_142','Ret_143','Ret_144','Ret_145','Ret_146','Ret_147','Ret_148','Ret_14
9','Ret_150','Ret_151','Ret_152',
'Ret_153','Ret_154','Ret_155','Ret_156','Ret_157','Ret_158','Ret_159','Ret_16
0','Ret_161','Ret_162','Ret_163',
'Ret_164','Ret_165','Ret_166','Ret_167','Ret_168','Ret_169','Ret_170','Ret_17
1','Ret_172','Ret_173','Ret_174',
'Ret_175','Ret_176','Ret_177','Ret_178','Ret_179','Ret_180','Ret_PlusOne','Ret
_PlusTwo']

total = first + allcol
feature_cols = [col for col in df.columns if col not in total]

X = df[feature_cols] # .loc[:, 'Feature_1':'Ret_120']
y = df[allcol] #.loc[:, 'Ret_121':'Weight_Daily']

X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(X, y, test
_size=0.20, random_state=42)
```

In [0]:
```
intraDay_cols = ['Ret_PlusOne','Ret_PlusTwo']

dailyReturn_cols = ['Ret_121','Ret_122','Ret_123','Ret_124','Ret_125','Ret_12
6','Ret_127','Ret_128','Ret_129','Ret_130',
'Ret_131','Ret_132','Ret_133','Ret_134','Ret_135','Ret_136','Ret_137','Ret_13
8','Ret_139','Ret_140','Ret_141',
'Ret_142','Ret_143','Ret_144','Ret_145','Ret_146','Ret_147','Ret_148','Ret_14
9','Ret_150','Ret_151','Ret_152',
'Ret_153','Ret_154','Ret_155','Ret_156','Ret_157','Ret_158','Ret_159','Ret_16
0','Ret_161','Ret_162','Ret_163',
'Ret_164','Ret_165','Ret_166','Ret_167','Ret_168','Ret_169','Ret_170','Ret_17
1','Ret_172','Ret_173','Ret_174',
'Ret_175','Ret_176','Ret_177','Ret_178','Ret_179','Ret_180']
```

In [0]:
```
y_train_raw.shape
```

Out[0]:
```
(32000, 62)
```

In [0]:
```
def preprocess_data(x_raw, y_raw):
    # top 25 features
    features = x_raw.loc[:, 'Feature_1':'Feature_25'].values[:, None, :]

    # 120 return values
    returns_intraday = x_raw.loc[:, 'Ret_2':'Ret_120'].values[:, :, None]
    print("No of repeatitions : ",returns_intraday.shape[1])

    features_repeated = np.repeat(features, returns_intraday.shape[1], axis=1)
    X_intraday = np.dstack((features_repeated, returns_intraday))

    returns_daily = x_raw.loc[:, 'Ret_MinusTwo':'Ret_MinusOne'].values[:, :, N
one]
    print("No of repeatitions : ",returns_daily.shape[1])
    features_repeated = np.repeat(features, returns_daily.shape[1], axis=1)
    X_daily = np.dstack((features_repeated, returns_daily))

    # targets should consist of returns only
    y_intraday = y_raw.loc[:, 'Ret_121':'Ret_180']
    y_daily = y_raw.loc[:, 'Ret_PlusOne':'Ret_PlusTwo']

    return X_intraday, X_daily, y_intraday, y_daily
```

In [0]:
```
X_train_intraday, X_train_daily, y_train_intraday, y_train_daily = preprocess_
data(X_train_raw, y_train_raw)
X_test_intraday, X_test_daily, y_test_intraday, y_test_daily= preprocess_data(
X_test_raw, y_test_raw)
```

```
No of repeatitions :  119
No of repeatitions :  2
No of repeatitions :  119
No of repeatitions :  2
```

In [0]:
```
print("X_train_intraday.shape :",X_train_intraday.shape)
print("X_train_daily.shape :",X_train_daily.shape)
```

```
X_train_intraday.shape : (32000, 119, 26)
X_train_daily.shape : (32000, 2, 26)
```

In [0]:
```
X_train_intraday[:2][:2][:2]
```

Out[0]:
```
array([[[            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01,  8.63062304e-04],
        [            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01, -2.64819597e-04],
        [            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01,  2.92963178e-04],
        ...,
        [            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01, -1.14146922e-03],
        [            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01, -2.55115231e-03],
        [            nan, -1.09807731e+00,  1.43122925e-01, ...,
         -1.33675658e+00, -1.31101549e-01, -3.03053183e-06]],

       [[            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01, -7.00989192e-06],
        [            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01, -1.36719518e-03],
        [            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01,  7.78957121e-05],
        ...,
        [            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01,  2.53330212e-04],
        [            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01, -4.43259597e-04],
        [            nan, -5.48323039e-01,  2.65133173e-02, ...,
                     nan, -5.30744054e-01,  4.46983030e-04]]])
```

1. LSTM with masking layers

In [0]:
```
def build_lstm_model(input_data, output_size, neurons=20, activ_func='relu',
                     dropout=.4, loss='mae', loss_weights=None, sample_weight_
mode=None,
                     optimizer='adam'):
    model = Sequential()
    # https://stackoverflow.com/questions/49670832/keras-lstm-with-masking-lay
er-for-variable-length-inputs
    # https://www.quora.com/What-is-masking-in-a-recurrent-neural-network-RNN
    model.add(Masking(mask_value=0., input_shape=(input_data.shape[1], input_d
ata.shape[2])))
    model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape
[2])))

    model.add(Dropout(dropout))
    model.add(Dense(units=2 * neurons))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))
    model.compile(loss=loss, loss_weights=loss_weights, sample_weight_mode=sam
ple_weight_mode, optimizer=optimizer)
    print(model.summary())
    return model
```

a. Model for daily returns

In [0]:
```
lstm_neurons = 100
epochs = 3
batch_size = 500
loss = 'mse'
dropout = 0.20

checkpointer_daily = ModelCheckpoint(filepath='best_weights_daily.hdf5',
                               verbose=1, save_best_only=True)
model_daily = build_lstm_model(X_train_daily, y_train_daily.shape[1], neurons=
lstm_neurons,
                               activ_func='relu', optimizer='adam',
                               dropout=0.3)
history_daily = model_daily.fit(X_train_daily, y_train_daily, validation_split
=0.25, batch_size=batch_size,
                           verbose=1, shuffle=True,
                          epochs=epochs, callbacks=[checkpointer_daily])
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
masking_2 (Masking)          (None, 2, 26)             0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               50800     
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 200)               20200     
_________________________________________________________________
dropout_4 (Dropout)          (None, 200)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 402       
_________________________________________________________________
activation_2 (Activation)    (None, 2)                 0         
=================================================================
Total params: 71,402
Trainable params: 71,402
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use
tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf.c
ompat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use tf.
compat.v1.Session instead.

Train on 24000 samples, validate on 8000 samples
Epoch 1/3
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:190: The name tf.get_default_session is deprecated. Ple
ase use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please use
tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:207: The name tf.global_variables is deprecated. Please
use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:216: The name tf.is_variable_initialized is deprecated.
Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:223: The name tf.variables_initializer is deprecated. P
lease use tf.compat.v1.variables_initializer instead.
```

```
24000/24000 [==============================] - 8s 323us/step - loss: 0.0156 -
val_loss: 0.0154

Epoch 00001: val_loss improved from inf to 0.01537, saving model to best_weig
hts_daily.hdf5
Epoch 2/3
24000/24000 [==============================] - 1s 36us/step - loss: 0.0155 -
val_loss: 0.0154

Epoch 00002: val_loss did not improve from 0.01537
Epoch 3/3
24000/24000 [==============================] - 1s 35us/step - loss: 0.0155 -
val_loss: 0.0154

Epoch 00003: val_loss did not improve from 0.01537
```

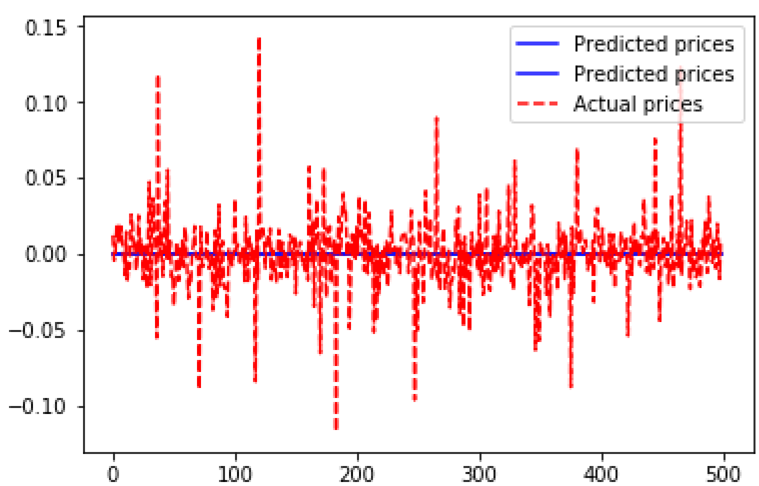
In [0]:
```
m1= mean_squared_error(model_daily.predict(X_test_daily), y_test_daily)
print("MSE error :"m1)
```

```
0.000650198712438325
```

In [0]:
```
model_daily.predict(X_test_daily)
```

Out[0]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       ...,
       [0., 0.],
       [0., 0.],
       [0., 0.]], dtype=float32)
```

In [0]:
```
plt.plot(range(0, len(y_test_daily[:500])), model_daily.predict(X_test_daily)
[:500], 'b-', label='Predicted prices')
plt.plot(range(0, len(y_test_daily[:500])),y_test_daily['Ret_PlusOne'][:500],
'r--', label='Actual prices')
plt.legend()
plt.show()
```



b. Model for intra day returns

In [0]:
```
lstm_neurons = 250
epochs = 3
batch_size = 500
loss = 'mae'
dropout = 0.20
optimizer = 'adam'
activ_func = 'relu'

checkpointer_intraday = ModelCheckpoint(filepath='best_weights_intraday.hdf5',
                                verbose=1, save_best_only=True)
model_intraday = build_lstm_model(X_train_intraday, y_train_intraday.shape[1],
neurons=lstm_neurons,
                                activ_func=activ_func, optimizer=optimizer,
                                dropout=dropout)
history_intraday = model_intraday.fit(X_train_intraday, y_train_intraday, vali
dation_split=0.25, batch_size=batch_size,
                            verbose=1, shuffle=True,
                           epochs=epochs, callbacks=[checkpointer_intraday])
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
masking_3 (Masking)          (None, 119, 26)           0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 250)               277000    
_________________________________________________________________
dropout_5 (Dropout)          (None, 250)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 500)               125500    
_________________________________________________________________
dropout_6 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 60)                30060     
_________________________________________________________________
activation_3 (Activation)    (None, 60)                0         
=================================================================
Total params: 432,560
Trainable params: 432,560
Non-trainable params: 0
_________________________________________________________________
None
Train on 24000 samples, validate on 8000 samples
Epoch 1/3
24000/24000 [==============================] - 23s 954us/step - loss: 7.1501e
-04 - val_loss: 6.3264e-04

Epoch 00001: val_loss improved from inf to 0.00063, saving model to best_weig
hts_intraday.hdf5
Epoch 2/3
24000/24000 [==============================] - 22s 930us/step - loss: 6.3405e
-04 - val_loss: 6.3264e-04

Epoch 00002: val_loss did not improve from 0.00063
Epoch 3/3
24000/24000 [==============================] - 22s 918us/step - loss: 6.3405e
-04 - val_loss: 6.3264e-04

Epoch 00003: val_loss did not improve from 0.00063
```

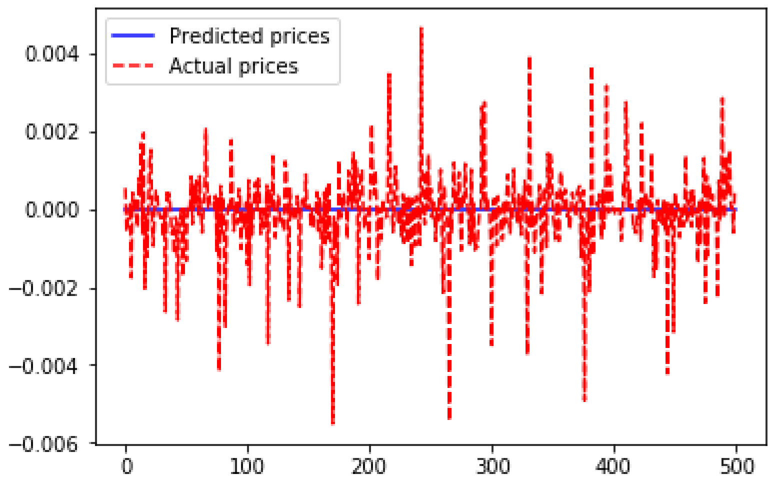
In [0]:

```
y_pred=model_intraday.predict(X_test_intraday)
m2= mean_squared_error(y_pred, y_test_intraday)
print("MSE error :"m2)
```

```
1.2613788683900336e-06
```

In [0]:
```
plt.plot(range(0, 500), y_pred[:,0][:500], 'b-', label='Predicted prices')
plt.plot(range(0, 500),y_test_intraday['Ret_121'][:500], 'r--', label='Actual
 prices')
plt.legend()
```

Out[0]: <matplotlib.legend.Legend at 0x7f13845a7908>



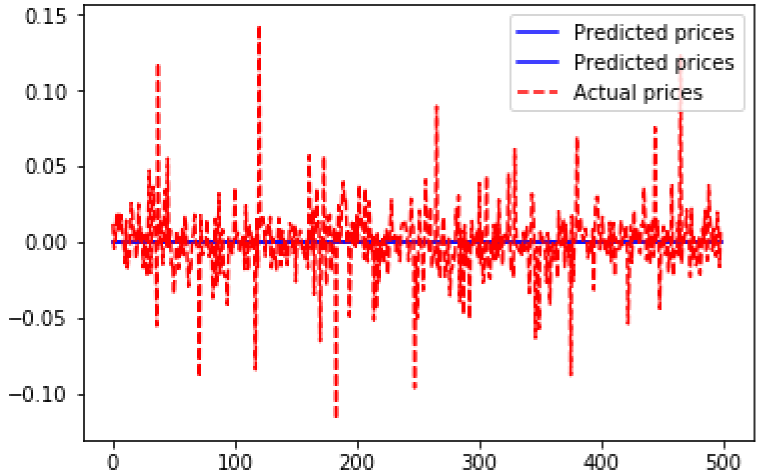
In [0]:
```
conc.add_row(["LSTM With masking layer", m1, m2])
print(conc)
```

```
+-------------------------+-------------------------+-----------------------
--+
|        Model Name       | MSE for intra day returns | MSE for daily returns
|
+-------------------------+-------------------------+-----------------------
--+
| LSTM With masking layer |    0.000650198712438325   | 1.2613788683900336e-0
6 |
+-------------------------+-------------------------+-----------------------
--+
```

In [0]:
```
# feature_cols = [col for col in df.columns if col not in total]
# X = df[feature_cols]
# y = df[allcol]
# X.fillna(X.median(axis=0), inplace=True)
# y.fillna(y.mean(axis=0), inplace=True)

# X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(X, y, te
st_size=0.20, random_state=42)
# X_train_intraday, X_train_daily, y_train_intraday, y_train_daily = preproces
s_data(X_train_raw, y_train_raw)
# X_test_intraday, X_test_daily, y_test_intraday, y_test_daily= preprocess_dat
a(X_test_raw, y_test_raw)

# lstm_neurons = 100
# epochs = 3
# batch_size = 500
# loss = 'mse'
# dropout = 0.20

# checkpointer_daily = ModelCheckpoint(filepath='best_weights_daily.hdf5',
#                                 verbose=1, save_best_only=True)
# model_daily = build_lstm_model(X_train_daily, y_train_daily.shape[1], neuron
s=lstm_neurons,
#                                 activ_func='relu', optimizer='adam',
#                                 dropout=0.3)
# history_daily = model_daily.fit(X_train_daily, y_train_daily, validation_spl
it=0.25, batch_size=batch_size,
#                             verbose=1, shuffle=True,
#                            epochs=epochs, callbacks=[checkpointer_daily])

# m1= mean_squared_error(model_daily.predict(X_test_daily), y_test_daily)
# print(m1)
# model_daily.predict(X_test_daily)
# plt.plot(range(0, len(y_test_daily[:500])), model_daily.predict(X_test_dail
y)[:500], 'b-', label='Predicted prices')
# plt.plot(range(0, len(y_test_daily[:500])),y_test_daily['Ret_PlusOne'][:50
0], 'r--', label='Actual prices')
# plt.legend()
# plt.show()

# lstm_neurons = 250
# epochs = 3
# batch_size = 500
# loss = 'mae'
# dropout = 0.20
# optimizer = 'adam'
# activ_func = 'relu'

# checkpointer_intraday = ModelCheckpoint(filepath='best_weights_intraday.hdf
5',
#                                 verbose=1, save_best_only=True)
# model_intraday = build_lstm_model(X_train_intraday, y_train_intraday.shape
[1], neurons=lstm_neurons,
#                                 activ_func=activ_func, optimizer=optimizer,
#                                 dropout=dropout)
# history_intraday = model_intraday.fit(X_train_intraday, y_train_intraday, va
```

```
lidation_split=0.25, batch_size=batch_size,
#                             verbose=1, shuffle=True,
#                            epochs=epochs, callbacks=[checkpointer_intraday])

# y_pred=model_intraday.predict(X_test_intraday)
# m2= mean_squared_error(y_pred, y_test_intraday)
# print(m2)

# plt.plot(range(0, 500), y_pred[:,0][:500], 'b-', label='Predicted prices')
# plt.plot(range(0, 500),y_test_intraday['Ret_121'][:500], 'r--', label='Actua
l prices')
# plt.legend()

# conc.add_row(["LSTM With masking layer and replacing NaN with median", m1, m
2])
# print(conc)
```

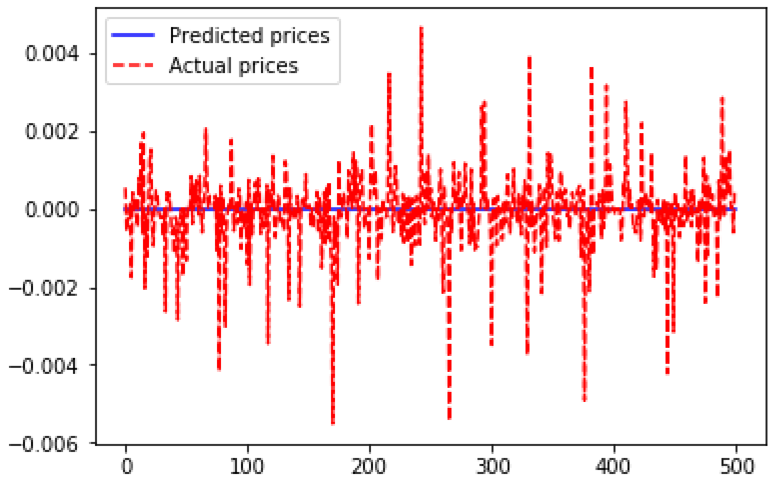
```
No of repeatitions :  119
No of repeatitions :  2
No of repeatitions :  119
No of repeatitions :  2
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
masking_10 (Masking)         (None, 2, 26)             0         
_________________________________________________________________
lstm_10 (LSTM)               (None, 100)               50800     
_________________________________________________________________
dropout_19 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 200)               20200     
_________________________________________________________________
dropout_20 (Dropout)         (None, 200)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 2)                 402       
_________________________________________________________________
activation_10 (Activation)   (None, 2)                 0         
=================================================================
Total params: 71,402
Trainable params: 71,402
Non-trainable params: 0
_________________________________________________________________
None
Train on 24000 samples, validate on 8000 samples
Epoch 1/3
24000/24000 [==============================] - 4s 168us/step - loss: 0.0312 - 
val_loss: 0.0154

Epoch 00001: val_loss improved from inf to 0.01537, saving model to best_weig
hts_daily.hdf5
Epoch 2/3
24000/24000 [==============================] - 1s 37us/step - loss: 0.0155 - 
val_loss: 0.0154

Epoch 00002: val_loss did not improve from 0.01537
Epoch 3/3
24000/24000 [==============================] - 1s 38us/step - loss: 0.0155 - 
val_loss: 0.0154

Epoch 00003: val_loss did not improve from 0.01537
0.000650198712438325
```



Predicted prices
Predicted prices
Actual prices
0.15
0.10
0.05
0.00
−0.05
−0.10
0
100
200
300
400
500

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
masking_11 (Masking)         (None, 119, 26)           0         
_________________________________________________________________
lstm_11 (LSTM)               (None, 250)               277000    
_________________________________________________________________
dropout_21 (Dropout)         (None, 250)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 500)               125500    
_________________________________________________________________
dropout_22 (Dropout)         (None, 500)               0         
_________________________________________________________________
dense_22 (Dense)             (None, 60)                30060     
_________________________________________________________________
activation_11 (Activation)   (None, 60)                0         
=================================================================
Total params: 432,560
Trainable params: 432,560
Non-trainable params: 0
_________________________________________________________________
None
Train on 24000 samples, validate on 8000 samples
Epoch 1/3
24000/24000 [==============================] - 26s 1ms/step - loss: 0.0074 -
val_loss: 6.3264e-04

Epoch 00001: val_loss improved from inf to 0.00063, saving model to best_weig
hts_intraday.hdf5
Epoch 2/3
24000/24000 [==============================] - 22s 936us/step - loss: 6.3435e
-04 - val_loss: 6.3264e-04

Epoch 00002: val_loss did not improve from 0.00063
Epoch 3/3
24000/24000 [==============================] - 22s 936us/step - loss: 6.3446e
-04 - val_loss: 6.3264e-04

Epoch 00003: val_loss did not improve from 0.00063
1.2613788683900336e-06
+-----------------------------------------------------+-------------------
-------+-----------------------+
|                      Model Name                     | MSE for intra day r
eturns | MSE for daily returns  |
+-----------------------------------------------------+-------------------
-------+-----------------------+
|                LSTM With masking layer              |    0.00065019871243
8325   | 1.2613788683900336e-06 |
|  LSTM With masking layer and replacing NaN with mean  |    0.00065019871243
8325   | 1.2613788683900336e-06 |
|  LSTM With masking layer and replacing NaN with mean  |    0.00065019871243
8325   | 1.2613788683900336e-06 |
|  LSTM With masking layer and replacing NaN with mean  |    0.00065019871243
8325   | 1.2613788683900336e-06 |
| LSTM With masking layer and replacing NaN with median |    0.00065019871243
8325   | 1.2613788683900336e-06 |
```

```
+-------------------------------------------------------+-------------------
-------+-----------------------+
```



The above model doesn't get improve if we replace the NaN values with median and mean

2. XGBoost

In [0]:
```
# https://stackoverflow.com/questions/24147278/how-do-i-create-test-and-train-
samples-from-one-dataframe-with-pandas
msk = np.random.rand(len(training_csv)) < 0.8
print(msk)
training_csv = df[msk]
testing_csv = df[~msk]

# training_data = training_csv.drop(training_csv.columns[range(146, 210)], axi
s=1)
# testing_csv = testing_csv.drop(testing_csv.columns[range(146, 210)], axis=1)
```

```
[False  True  True ...  True  True  True]
```

a. Daily returns

In [1]:
```
%%time

model_parameters = {'max_depth': 10,
                    'eta': 0.1,
                    'silent': 1,
                    'gamma': 0,
                    'lambda': 500,
                    'alpha': 400,
                    'verbose':0}
number_of_rounds = 500
# https://stackoverflow.com/questions/24147278/how-do-i-create-test-and-train-
samples-from-one-dataframe-with-pandas
msk = np.random.rand(len(df)) < 0.8
print(msk)
training_csv = df[msk]
testing_csv = df[~msk]
res =  pd.DataFrame()

mse_list=[]
major_list=[]
pro =ProgressBar()

training_data = training_csv.drop(training_csv.columns[range(146, 210)], axis=
1)
testing_data = testing_csv.drop(testing_csv.columns[range(146, 210)], axis=1)

training_data = training_data.values
testing_data = testing_data.values

# as we plot only Ret_1 for every model to compare test prediction and actual
 value
plot_res =  pd.DataFrame()
for Number in pro(range(1,61)): # From 1 to 62

    name_of_column = 'Ret_'+str(Number+120)
    name_of_weight = 'Weight_Intraday'

    train_targets = training_csv[name_of_column].values
    test_targets = testing_csv[name_of_column].values

    train_weights = training_csv[name_of_weight].values

    data_train = xgb.DMatrix(training_data, label=train_targets, missing=np.Na
N, weight=train_weights)
    data_test = xgb.DMatrix(testing_data, missing=np.NaN)

    watchlist = [(data_train, 'train')]
    bst = xgb.train(model_parameters,
                    data_train,
                    number_of_rounds,
                    watchlist,
                    early_stopping_rounds=10,
                    verbose_eval=0)
    #
    predictions = bst.predict(data_test)
    if Number is 1:
```

```
        plot_res=plot_res.append(pd.DataFrame({'x':bst.predict(data_test).toli
st(), 'y':test_targets.tolist()}))

    res=res.append(pd.DataFrame({'x':bst.predict(data_test).tolist(), 'y':test
_targets.tolist()}))

    # print('c'*10)
    for ID, P in enumerate(predictions):
        major_list.append({'Id': str(ID+1)+'_'+str(Number), 'Predicted': P })

output = pd.DataFrame(data=major_list)
output.sort_values(by='Id', inplace=True)
print(output.head())
```

```
N/A% (0 of 60) |                         | Elapsed Time: 0:00:00 ETA:  --:--:
--[ True  True  True ...  True  True False]
100% (60 of 60) |#######################| Elapsed Time: 11:51:44 Time: 11:51:
44
            Id  Predicted
999     1000_1  -0.000205
72234  1000_10   0.000284
80149  1000_11   0.000042
88064  1000_12  -0.000153
95979  1000_13  -0.000092
CPU times: user 23h 39min 19s, sys: 19.6 s, total: 23h 39min 39s
Wall time: 11h 51min 45s
```

In [0]:

```
res.to_csv("res.csv")
output.to_csv("output.csv")
plot_res.to_csv("plot_res.csv")
```

In [89]:

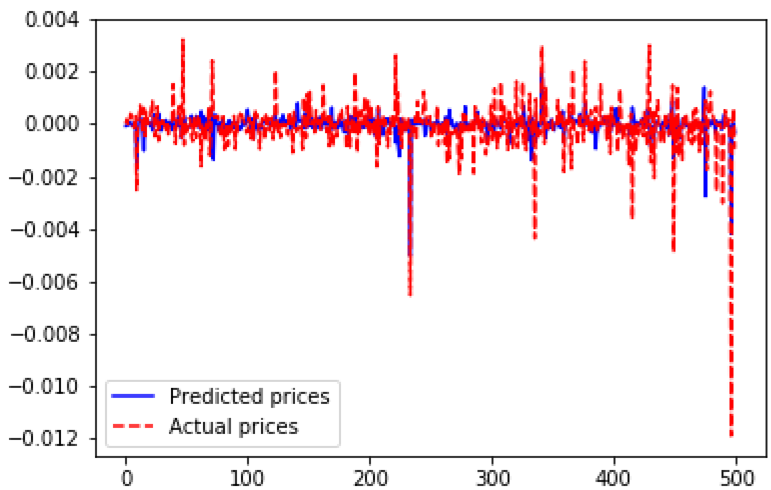
```
res=pd.read_csv("res.csv",index_col=0)
output=pd.read_csv("output.csv",index_col=0)
plot_res=pd.read_csv("plot_res.csv",index_col=0)
res.head()
```

Out[89]:

	x	y
0	-0.000078	0.000011
1	-0.000076	0.000185
2	0.000007	-0.000003
3	-0.000016	0.000065
4	0.000074	0.000394

In [90]:
```
# val=plot_res.shape[0]
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
MSE error: 1.2405071522992503e-06
```



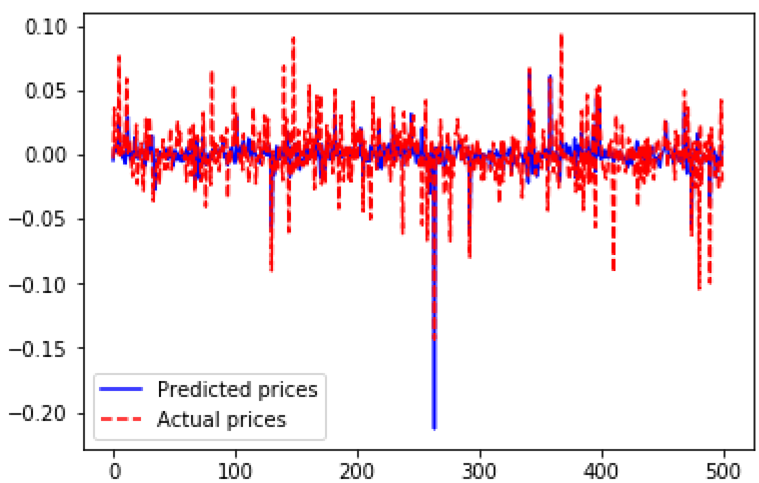
b. Intra day returns

In [0]:
```
%%time
pro =ProgressBar()
# as we plot only Ret_1 for every model to compare test prediction and actual
 value
plot_res =  pd.DataFrame()
res =  pd.DataFrame()
for Number in pro(range(61,63)): # From 1 to 62
    if Number == 61:
        name_of_column = 'Ret_PlusOne'
        name_of_weight = 'Weight_Daily'
    else:
        name_of_column = 'Ret_PlusTwo'
        name_of_weight = 'Weight_Daily'

    train_targets = training_csv[name_of_column].values
    test_targets = testing_csv[name_of_column].values

    train_weights = training_csv[name_of_weight].values

    data_train = xgb.DMatrix(training_data, label=train_targets, missing=np.Na
N, weight=train_weights)
    data_test = xgb.DMatrix(testing_data, missing=np.NaN)

    watchlist = [(data_train, 'train')]
    bst = xgb.train(model_parameters,
                    data_train,
                    number_of_rounds,
                    watchlist,
                    early_stopping_rounds=10,
                    verbose_eval=0)

    predictions = bst.predict(data_test)
    if Number is 61:
        plot_res=plot_res.append(pd.DataFrame({'x':bst.predict(data_test).toli
st(), 'y':test_targets.tolist()}))

    res=res.append(pd.DataFrame({'x':bst.predict(data_test).tolist(), 'y':test
_targets.tolist()}))
    for ID, P in enumerate(predictions):
        major_list.append({'Id': str(ID+1)+'_'+str(Number), 'Predicted': P })

output = pd.DataFrame(data=major_list)
output.sort_values(by='Id', inplace=True)
print(output.head())
```

```
100% (2 of 2) |##########################| Elapsed Time: 0:31:10 Time:  0:31:
10

            Id  Predicted
999     1000_1   0.000069
8974   1000_61  -0.010980
16949  1000_62  -0.001672
1000    1001_1  -0.000213
8975   1001_61  -0.004094
CPU times: user 1h 1min 21s, sys: 2.22 s, total: 1h 1min 23s
Wall time: 31min 10s
```

In [0]:
```
output = output.sort_index(axis = 0)
output.to_csv("xgboost_Intraday_returns_sol_submission.csv")
print(res.head())
# print(output.head())

# val=plot_res.shape[0]
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'], res['y'])
print("MSE error:",m2)
# plot_res.shape
```

```
          x         y
0 -0.004701 -0.002939
1  0.004895  0.036104
2  0.006895  0.004728
3  0.002062  0.003166
4  0.003386  0.000229
    Id  Predicted
0  1_1   0.000184
1  2_1  -0.000291
2  3_1   0.000126
3  4_1   0.000080
4  5_1  -0.000064
MSE error: 0.0005016972798379116
```



In [165]:
```
conc.add_row(["XGboost", m1, m2])
print(conc)
```

```
+-------------------------+-------------------------+-------------------------
--+
|        Model Name       | MSE for daily returns  | MSE for intra day return
s |
+-------------------------+-------------------------+-------------------------
--+
| LSTM With masking layer |  0.000650198712438325  |   1.2613788683900336e-06
|
|         XGboost         | 1.2405071522992503e-06 |   0.0005016972798379116
|
+-------------------------+-------------------------+-------------------------
--+
```

3. Ensemble Technique

a. Daily returns

In [16]:
```
df_train, df_test= train_test_split( df, test_size=0.20, random_state=42)
len(df_train.columns)
```

Out[16]:
```
211
```

In [0]:

```
# https://www.programcreek.com/python/example/102434/sklearn.ensemble.ExtraTre
esRegressor
res =  pd.DataFrame()
plot_res =  pd.DataFrame()

feature_cols = [col for col in df_train.columns if col not in total]
pro=ProgressBar()
for i in pro(dailyReturn_cols):
    X_train = df_train[feature_cols]
    X_train = X_train.fillna(0)
    X_test = df_test[feature_cols]
    X_test = X_test.fillna(0)

    y = df_train[i].astype('float32')
    y = y.fillna(0)

    y_p1 = df_test[i].astype('float32')
    y_p1 = y_p1.fillna(0)

    classification = ExtraTreesRegressor(n_estimators=10,max_depth=None, min_s
amples_split=2)   #rf = Pipeline([("scale", StandardScaler()),("rf", RandomFor
estClassifier(n_estimators=10, 2))])
    X = Imputer().fit_transform(X_train)
    y=y.values.reshape(-1,1)
    y = Imputer().fit_transform(y)
    classification.fit(X_train, y)
    y_pred = classification.predict(X_test)

    if i is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':y_pred.tolist(), 'y':y_p1.t
olist()}))
    res=res.append(pd.DataFrame({'x':y_pred.tolist(), 'y':y_p1.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 0:19:44 Time:  0:19:
44
```

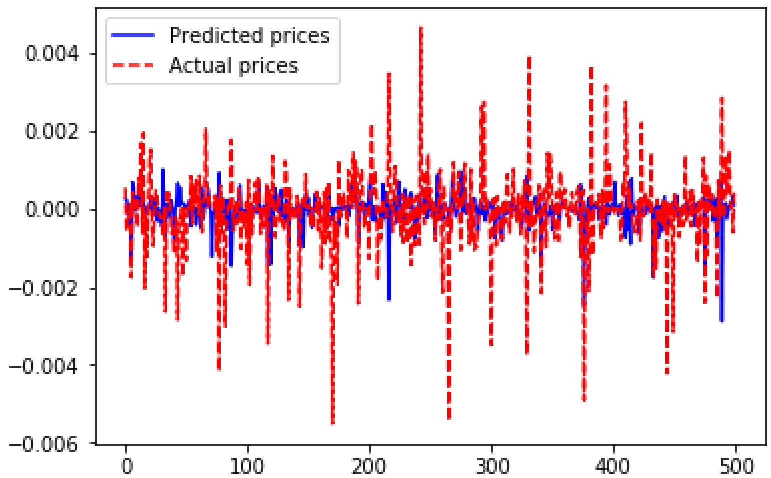
In [0]:

```
print(plot_res.head())
```

```
          x         y
0  0.000245  0.000531
1  0.000221 -0.000597
2  0.000052 -0.000173
3 -0.000186 -0.000201
4 -0.000088  0.000116
```

In [0]:
```
# val=plot_res.shape[0]
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
MSE error: 1.2636426776403615e-06
```



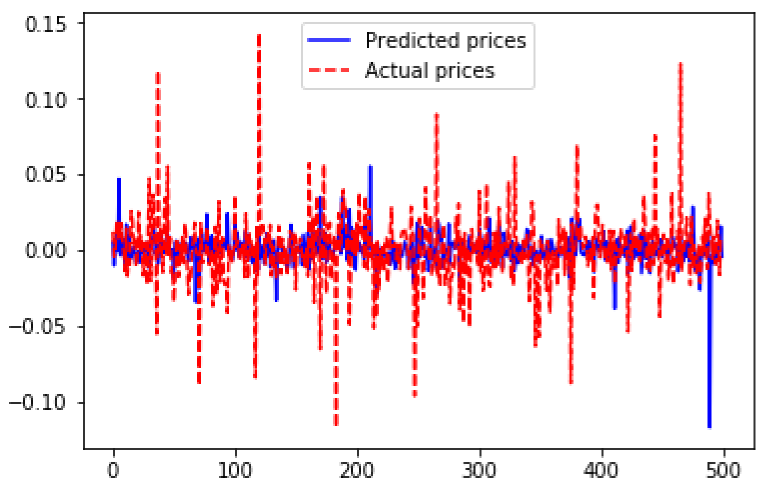
b. Intra day returns

In [166]:
```
# https://www.programcreek.com/python/example/102434/sklearn.ensemble.ExtraTre
esRegressor
res =  pd.DataFrame()
plot_res =  pd.DataFrame()

feature_cols = [col for col in df_train.columns if col not in total]
pro=ProgressBar()
for i in pro(intraDay_cols):
    X_train = df_train[feature_cols]
    X_train = X_train.fillna(0)
    X_test = df_test[feature_cols]
    X_test = X_test.fillna(0)

    y = df_train[i].astype('float32')
    y = y.fillna(0)

    y_p1 = df_test[i].astype('float32')
    y_p1 = y_p1.fillna(0)

    classification = ExtraTreesRegressor(n_estimators=10,max_depth=None, min_s
amples_split=2)   #rf = Pipeline([("scale", StandardScaler()),("rf", RandomFor
estClassifier(n_estimators=10, 2))])
    X = Imputer().fit_transform(X_train)
    y=y.values.reshape(-1,1)
    y = Imputer().fit_transform(y)
    classification.fit(X_train, y)
    y_pred = classification.predict(X_test)

    if i is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':y_pred.tolist(), 'y':y_p1.t
olist()}))
    res=res.append(pd.DataFrame({'x':y_pred.tolist(), 'y':y_p1.tolist()}))
```

```
100% (2 of 2) |##########################| Elapsed Time: 0:00:38 Time:  0:00:
38
```

In [167]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'], res['y'])
print("MSE error:",m2)
```

```
          x         y
0  0.005029  0.011966
1 -0.010219 -0.004402
2  0.001798 -0.001938
3  0.010047  0.017789
4 -0.000822  0.007983
MSE error: 0.0006274016286506527
```



In [168]:
```
conc.add_row(["ExtraTreesRegressor ", m1, m2])
print(conc)
```

```
+-----------------------+------------------------+-------------------------
--+
|       Model Name      | MSE for daily returns  | MSE for intra day return
s |
+-----------------------+------------------------+-------------------------
--+
| LSTM With masking layer |  0.000650198712438325  |   1.2613788683900336e-06
|
|        XGboost        | 1.2405071522992503e-06 |   0.0005016972798379116
|
|   ExtraTreesRegressor   | 1.2636426776403615e-06 |   0.0006274016286506527
|
+-----------------------+------------------------+-------------------------
--+
```

4. LSTM with 2 hidden layers

In [169]:
```
model = Sequential()
model.add(Dense(12, input_dim=147, kernel_initializer='normal'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))
print(model.summary())

model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])

earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode
='min')
mcp_save = ModelCheckpoint('mdl_wts.hdf5', save_best_only=True, monitor='val_l
oss', mode='min')
# reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=
7, verbose=1, epsilon=1e-4, mode='min')
# https://stackoverflow.com/questions/48285129/saving-best-model-in-keras
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_24 (Dense)             (None, 12)                1776      
_________________________________________________________________
dense_25 (Dense)             (None, 8)                 104       
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 9         
=================================================================
Total params: 1,889
Trainable params: 1,889
Non-trainable params: 0
_________________________________________________________________
None
```

In [170]:
```
df = pd.read_csv(files_path+'train.csv')
df.fillna(df.mean(axis=0), inplace=True)
X = df.iloc[:,0:147]
msk = np.random.rand(len(X)) < 0.8
print(msk)
x_train=X[msk]
x_test=X[~msk]
```

```
[ True  True  True ...  True  True False]
```

In [0]:
```
scaler_x = MinMaxScaler()
scaler_x.fit(x_train)
x_train=scaler_x.transform(x_train)
x_test=scaler_x.transform(x_test)
```

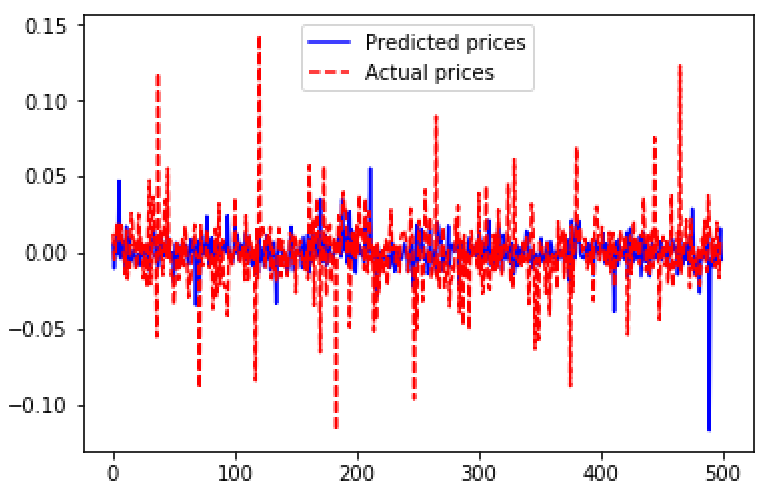
a. Daily returns

In [172]:
```
# Y = df.iloc[:,147]
pro=ProgressBar()
for col in pro(dailyReturn_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    history = model.fit(x_train, y_train,epochs=10, batch_size=50,callbacks=[e
arlyStopping, mcp_save],verbose=0)

    if col is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 0:28:24 Time:  0:28:
24
```

In [173]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
          x         y
0  0.005029  0.011966
1 -0.010219 -0.004402
2  0.001798 -0.001938
3  0.010047  0.017789
4 -0.000822  0.007983
MSE error: 3.4314954120847986e-05
```

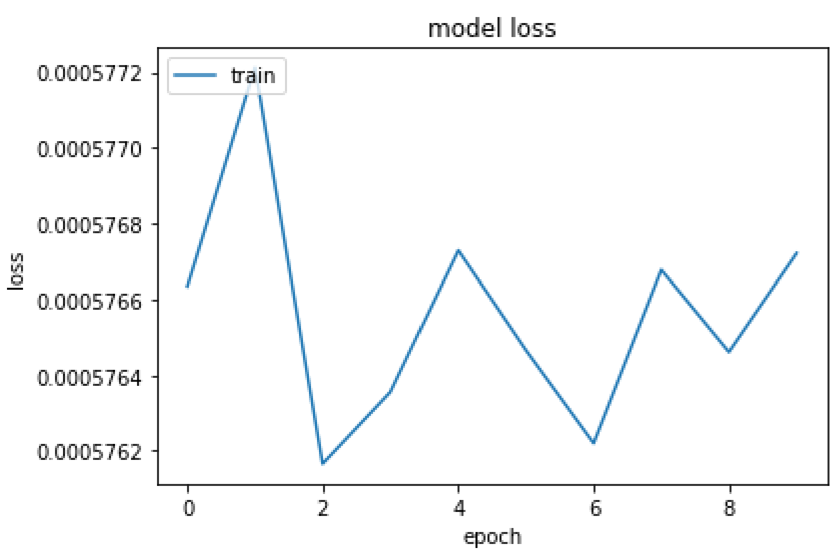


b. Intra day returns

In [174]:
```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(intraDay_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    history = model.fit(x_train, y_train,epochs=10, batch_size=50,callbacks=[e
arlyStopping, mcp_save],verbose=0)

    if col is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

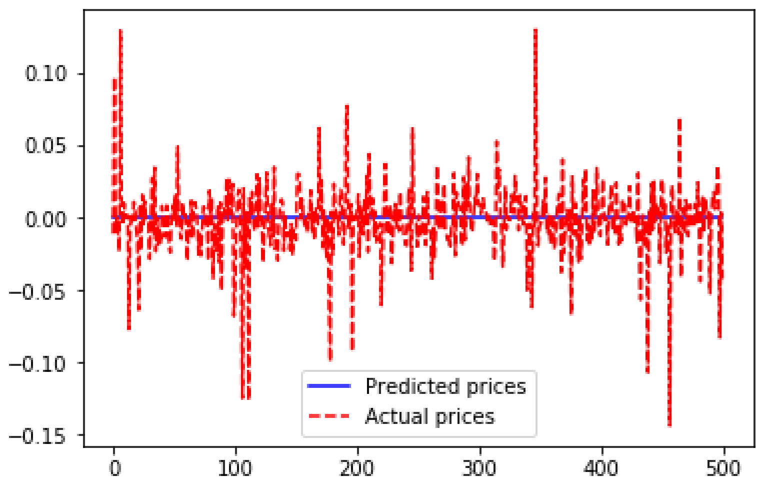
```
100% (2 of 2) |##########################| Elapsed Time: 0:00:56 Time:  0:00:
56
```

In [175]:
```
# this last plot is only for last column which is Ret_180
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



In [176]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'].values, res['y'].values)
print("MSE error:",m2)
```

```
          x         y
0 -0.000393 -0.011105
1 -0.000393  0.097741
2 -0.000393 -0.010067
3 -0.000393  0.000229
4 -0.000393 -0.007368
MSE error: 0.0006371389497042122
```



In [180]:
```
conc.add_row(["LSTM with 2 hidden layer ", m1, m2])
print(conc)
```

```
+---------------------------+------------------------+----------------------
----+
|         Model Name        | MSE for daily returns  | MSE for intra day retu
rns |
+---------------------------+------------------------+----------------------
----+
|  LSTM With masking layer  |  0.000650198712438325  |   1.2613788683900336e-
06  |
|          XGboost          | 1.2405071522992503e-06 |   0.000501697279837911
6   |
|    ExtraTreesRegressor    | 1.2636426776403615e-06 |   0.000627401628650652
7   |
| LSTM with 2 hidden layer  | 3.4314954120847986e-05 |   0.000637138949704212
2   |
+---------------------------+------------------------+----------------------
----+
```

5. LSTM with 4 hidden layers

In [115]:
```
model = Sequential()
model.add(Dense(128, input_dim=147, kernel_initializer='normal'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(8, activation='linear'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
print(model.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 128)               18944     
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_16 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_17 (Dense)             (None, 8)                 264       
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 9         
=================================================================
Total params: 29,553
Trainable params: 29,553
Non-trainable params: 0
_________________________________________________________________
None
```

In [116]:
```
df = pd.read_csv(files_path+'train.csv')
df.fillna(df.mean(axis=0), inplace=True)
X = df.iloc[:,0:147]
msk = np.random.rand(len(X)) < 0.8
print(msk)
x_train=X[msk]
x_test=X[~msk]
```

```
[ True  True  True ...  True  True False]
```

In [0]:
```
scaler_x = MinMaxScaler()
scaler_x.fit(x_train)
x_train=scaler_x.transform(x_train)
x_test=scaler_x.transform(x_test)
```

a. Daily returns

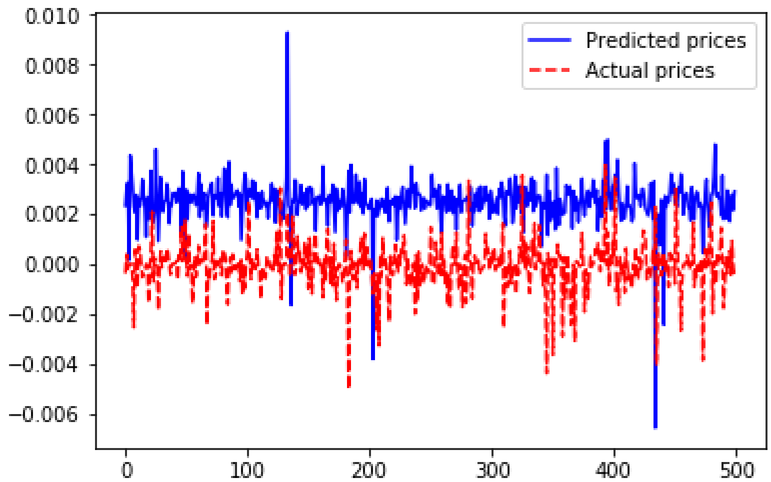
In [118]:

```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(dailyReturn_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)

    history = model.fit(x_train, y_train,epochs=10, batch_size=50,verbose=0)

    if col is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 0:32:53 Time:  0:32:
53
```

In [119]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
          x         y
0  0.002336 -0.000367
1  0.003032  0.000394
2  0.003263 -0.000191
3  0.000163  0.000010
4  0.004363  0.000006
MSE error: 1.5848400656118064e-06
```

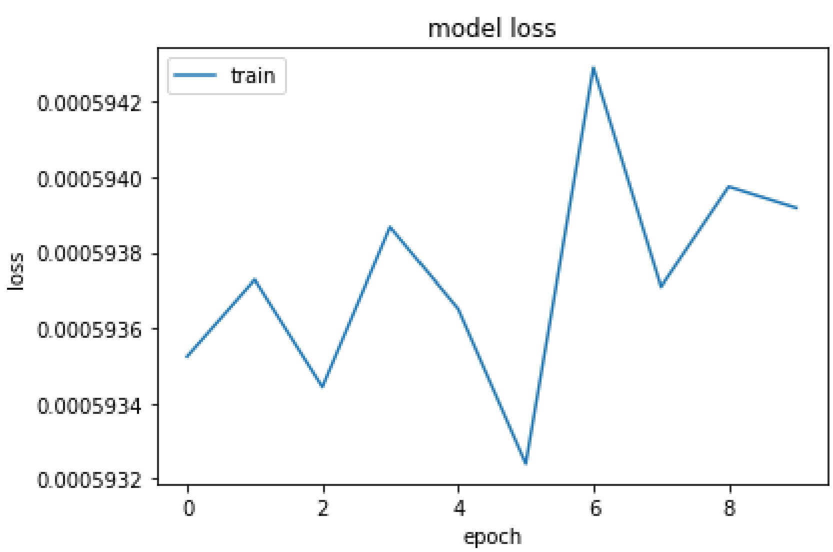


b. Intra day returns

In [120]:
```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(intraDay_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    history = model.fit(x_train, y_train,epochs=10, batch_size=50,callbacks=[e
arlyStopping, mcp_save],verbose=0)

    if col is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

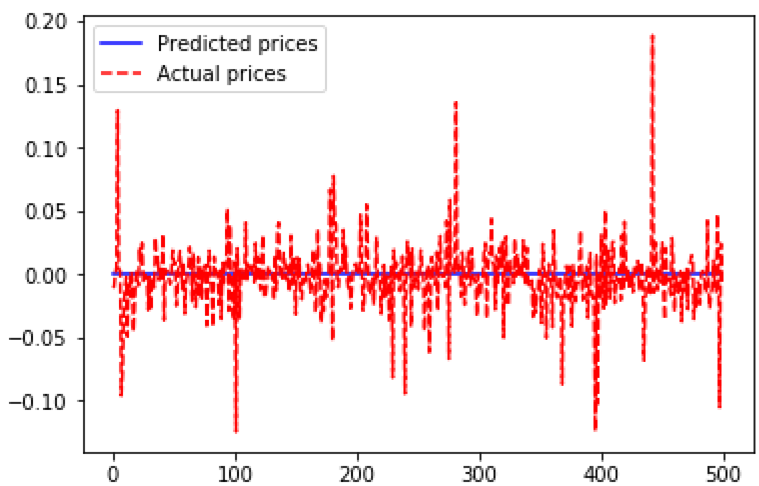
```
100% (2 of 2) |##########################| Elapsed Time: 0:01:06 Time:  0:01:
06
```

In [121]:
```
# this last plot is only for last column which is Ret_180
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



In [122]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'].values, res['y'].values)
print("MSE error:",m2)
```

```
          x         y
0 -0.000196 -0.009348
1 -0.000196 -0.010067
2 -0.000196  0.000229
3 -0.000196  0.005781
4 -0.000196  0.129496
MSE error: 0.0005776596093536332
```



In [181]:
```
conc.add_row(["LSTM with 4 hidden layer ", m1, m2])
print(conc)
```

```
+---------------------------+-----------------------+----------------------
----+
|         Model Name        | MSE for daily returns  | MSE for intra day retu
rns |
+---------------------------+-----------------------+----------------------
----+
|  LSTM With masking layer  |  0.000650198712438325  |   1.2613788683900336e-
06  |
|          XGboost          | 1.2405071522992503e-06 |   0.000501697279837911
6   |
|    ExtraTreesRegressor    | 1.2636426776403615e-06 |   0.000627401628650652
7   |
| LSTM with 2 hidden layer  | 3.4314954120847986e-05 |   0.000637138949704212
2   |
| LSTM with 4 hidden layer  | 1.5848400656118064e-06 |   0.000577659609353633
2   |
+---------------------------+-----------------------+----------------------
----+
```

6. LSTM with 4 layers and batch normalization layer

In [182]:
```
from keras.layers.normalization import BatchNormalization
model = Sequential()
model.add(Dense(128, input_dim=147, kernel_initializer='normal'))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())

model.add(Dense(32, activation='relu'))
model.add(Dense(8, activation='linear'))
model.add(Dense(1, activation='linear'))
print(model.summary())

model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 128)               18944
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dense_28 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dense_29 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_30 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 9
=================================================================
Total params: 30,321
Trainable params: 29,937
Non-trainable params: 384
_________________________________________________________________
None
```

In [183]:
```
df = pd.read_csv(files_path+'train.csv')
df.fillna(df.mean(axis=0), inplace=True)
X = df.iloc[:,0:147]
msk = np.random.rand(len(X)) < 0.8
print(msk)
x_train=X[msk]
x_test=X[~msk]
```

```
[ True  True  True ...  True False False]
```

In [0]:
```
scaler_x = MinMaxScaler()
scaler_x.fit(x_train)
x_train=scaler_x.transform(x_train)
x_test=scaler_x.transform(x_test)
```

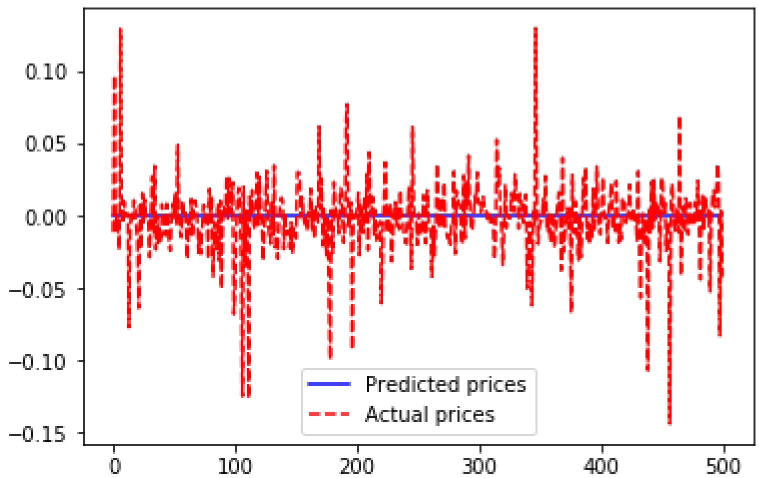
a. Daily returns

In [185]:
```
# Y = df.iloc[:,147]
pro=ProgressBar()
for col in pro(dailyReturn_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    history = model.fit(x_train, y_train,epochs=10, batch_size=50,callbacks=[e
arlyStopping, mcp_save],verbose=0)

    if col is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 1:01:02 Time:  1:01:
02
```

In [186]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
          x         y
0 -0.000393 -0.011105
1 -0.000393  0.097741
2 -0.000393 -0.010067
3 -0.000393  0.000229
4 -0.000393 -0.007368
MSE error: 2.1893272052289176e-05
```

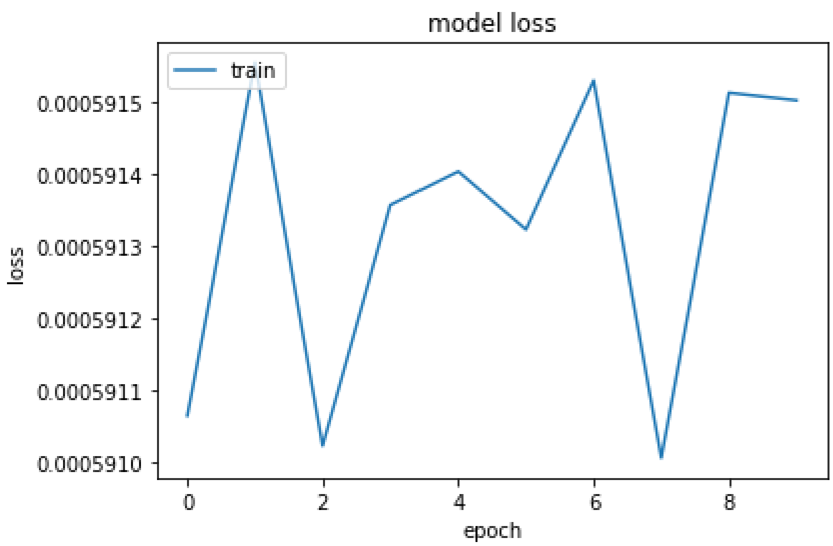


b. Intra day returns

In [187]:
```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(intraDay_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    history = model.fit(x_train, y_train,epochs=10, batch_size=50,callbacks=[e
arlyStopping, mcp_save],verbose=0)

    if col is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

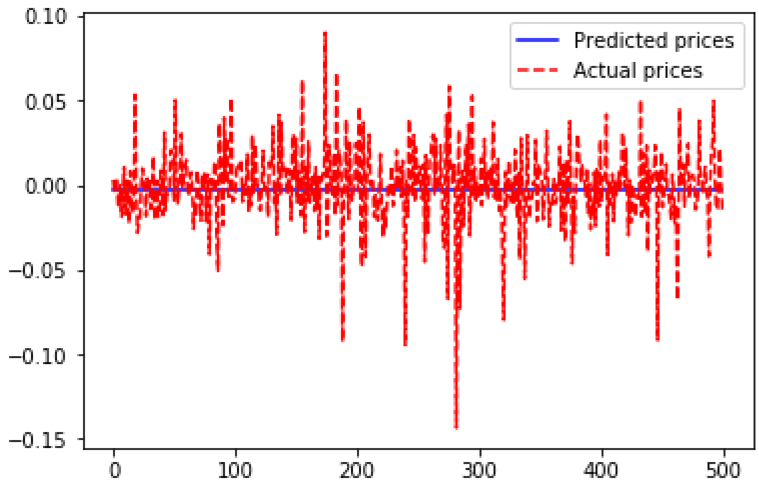
```
100% (2 of 2) |##########################| Elapsed Time: 0:02:05 Time:  0:02:
05
```

In [188]:
```
# this last plot is only for last column which is Ret_180
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



In [189]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'].values, res['y'].values)
print("MSE error:",m2)
```

```
          x         y
0 -0.002599  0.003166
1 -0.002599 -0.005204
2 -0.002599  0.003828
3 -0.002599  0.004051
4 -0.002599 -0.011652
MSE error: 0.0006294481172375907
```

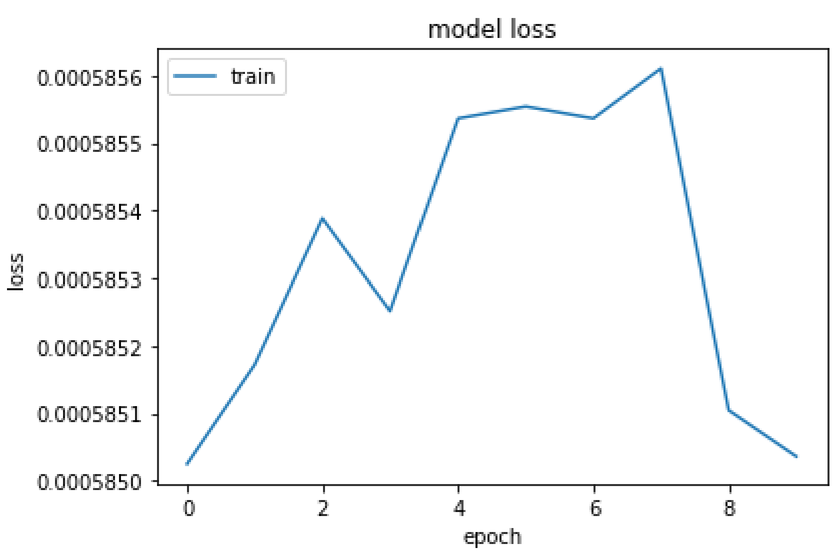


In [190]:
```
conc.add_row(["LSTM with 4 hidden layer and batchnormal layers ", m1, m2])
print(conc)
```

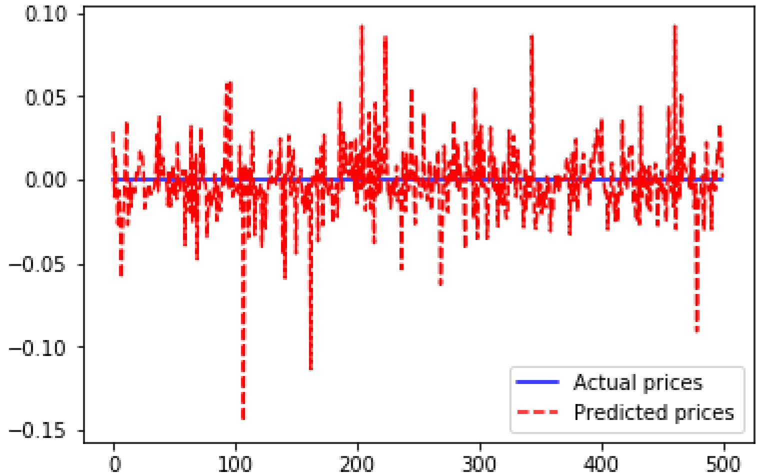
```
+--------------------------------------------------+-----------------------+---------------------------+
|                    Model Name                    | MSE for daily returns  | MSE for intra day returns |
+--------------------------------------------------+-----------------------+---------------------------+
|              LSTM With masking layer             |  0.000650198712438325  |   1.2613788683900336e-06  |
|                      XGboost                     | 1.2405071522992503e-06 |   0.0005016972798379116   |
|               ExtraTreesRegressor                | 1.2636426776403615e-06 |   0.0006274016286506527   |
|              LSTM with 2 hidden layer            | 3.4314954120847986e-05 |   0.0006371389497042122   |
|              LSTM with 4 hidden layer            | 1.5848400656118064e-06 |   0.0005776596093536332   |
| LSTM with 4 hidden layer and batchnormal layers  | 2.1893272052289176e-05 |   0.0006294481172375907   |
+--------------------------------------------------+-----------------------+---------------------------+
```

In [131]:
```
# scaler_x = MinMaxScaler()
# scaler_x.fit(X)
# xscale=scaler_x.transform(X)

# history = model.fit(xscale, Y,epochs=10, batch_size=50)
# plt.plot(history.history['loss'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train'], loc='upper left')
# plt.show()

# plt.plot(range(0, len(Y[:500])), model.predict(xscale[:500]), 'b-', label='A
ctual prices')
# plt.plot(range(0, len(Y[:500])), Y[:500], 'r--', label='Predicted prices')
# plt.legend()
```

```
Epoch 1/10
40000/40000 [==============================] - 7s 184us/step - loss: 5.8502e-
04 - mean_squared_error: 5.8502e-04 - mean_absolute_error: 0.0153
Epoch 2/10
40000/40000 [==============================] - 8s 193us/step - loss: 5.8517e-
04 - mean_squared_error: 5.8517e-04 - mean_absolute_error: 0.0153
Epoch 3/10
40000/40000 [==============================] - 8s 194us/step - loss: 5.8539e-
04 - mean_squared_error: 5.8539e-04 - mean_absolute_error: 0.0153
Epoch 4/10
40000/40000 [==============================] - 8s 193us/step - loss: 5.8525e-
04 - mean_squared_error: 5.8525e-04 - mean_absolute_error: 0.0153
Epoch 5/10
40000/40000 [==============================] - 8s 189us/step - loss: 5.8554e-
04 - mean_squared_error: 5.8554e-04 - mean_absolute_error: 0.0153
Epoch 6/10
40000/40000 [==============================] - 8s 193us/step - loss: 5.8555e-
04 - mean_squared_error: 5.8555e-04 - mean_absolute_error: 0.0154
Epoch 7/10
40000/40000 [==============================] - 8s 199us/step - loss: 5.8554e-
04 - mean_squared_error: 5.8554e-04 - mean_absolute_error: 0.0153
Epoch 8/10
40000/40000 [==============================] - 8s 194us/step - loss: 5.8561e-
04 - mean_squared_error: 5.8561e-04 - mean_absolute_error: 0.0153
Epoch 9/10
40000/40000 [==============================] - 8s 194us/step - loss: 5.8510e-
04 - mean_squared_error: 5.8510e-04 - mean_absolute_error: 0.0154
Epoch 10/10
40000/40000 [==============================] - 8s 191us/step - loss: 5.8504e-
04 - mean_squared_error: 5.8504e-04 - mean_absolute_error: 0.0153
```



Out[131]: <matplotlib.legend.Legend at 0x7fd849b71ef0>



7. RBF kernel SVR

In [191]:
```
df = pd.read_csv(files_path+'train.csv')
df.fillna(df.mean(axis=0), inplace=True)
X = df.iloc[:,0:147]
msk = np.random.rand(len(X)) < 0.8
print(msk)
x_train=X[msk]
x_test=X[~msk]
```

```
[False  True  True ... False  True  True]
```

a. Daily returns

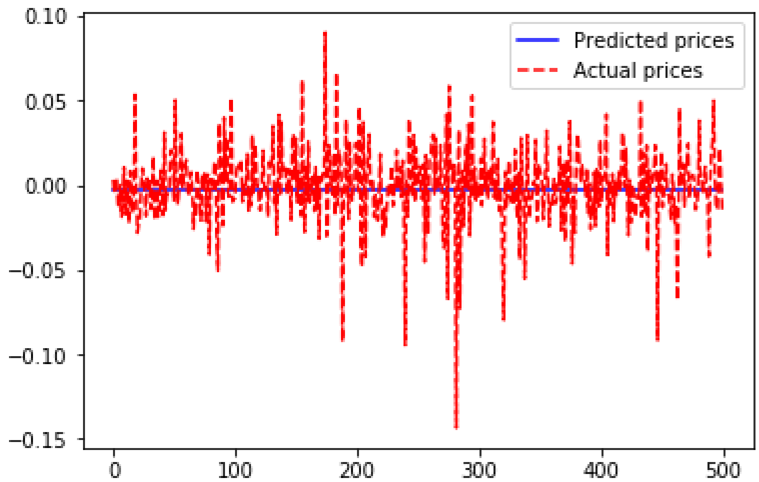
In [192]:

```
# Y = df.iloc[:,147]
pro=ProgressBar()
for col in pro(dailyReturn_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    model = SVR(kernel='rbf', C=1e3, gamma=0.1)
    #svr_lin = SVR(kernel='linear', C=1e3)
    #svr_poly = SVR(kernel='poly', C=1e3, degree=2)

    y_rbf = model.fit(x_train, y_train).predict(x_test)
    if col is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 0:00:06 Time:  0:00:
06
```

In [193]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
          x         y
0 -0.002599  0.003166
1 -0.002599 -0.005204
2 -0.002599  0.003828
3 -0.002599  0.004051
4 -0.002599 -0.011652
MSE error: 0.00011203147433272193
```



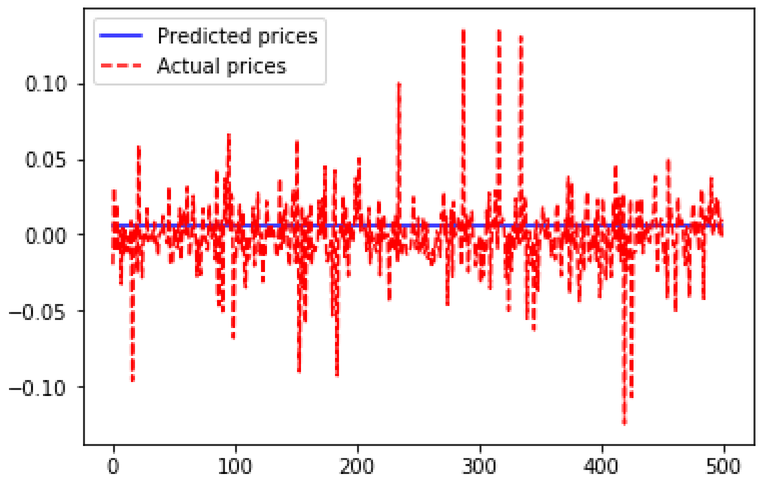
b. Intra day returns

In [194]:
```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(intraDay_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)

    model = SVR(kernel='rbf', C=1e3, gamma=0.1)
    #svr_lin = SVR(kernel='linear', C=1e3)
    #svr_poly = SVR(kernel='poly', C=1e3, degree=2)

    model.fit(x_train, y_train)

    if col is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (2 of 2) |##########################| Elapsed Time: 0:00:05 Time:  0:00:
05
```

In [195]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'].values, res['y'].values)
print("MSE error:",m2)
```

```
          x         y
0  0.006286 -0.019512
1  0.006286  0.031098
2  0.006286 -0.011105
3  0.006286  0.020268
4  0.006286 -0.010936
MSE error: 0.0006476464760422494
```



In [196]:
```
conc.add_row(["RBF kernel SVR ", m1, m2])
print(conc)
```

```
+--------------------------------------------------+-----------------------+
---------------------------+
|                    Model Name                    | MSE for daily returns  |
MSE for intra day returns |
+--------------------------------------------------+-----------------------+
---------------------------+
|             LSTM With masking layer              |  0.000650198712438325  |
1.2613788683900336e-06  |
|                     XGboost                      | 1.2405071522992503e-06 |
0.0005016972798379116   |
|               ExtraTreesRegressor                | 1.2636426776403615e-06 |
0.0006274016286506527   |
|             LSTM with 2 hidden layer             | 3.4314954120847986e-05 |
0.0006371389497042122   |
|             LSTM with 4 hidden layer             | 1.5848400656118064e-06 |
0.0005776596093536332   |
| LSTM with 4 hidden layer and batchnormal layers  | 2.1893272052289176e-05 |
0.0006294481172375907   |
|                  RBF kernel SVR                  | 0.00011203147433272193 |
0.0006476464760422494   |
+--------------------------------------------------+-----------------------+
---------------------------+
```

Linear regression

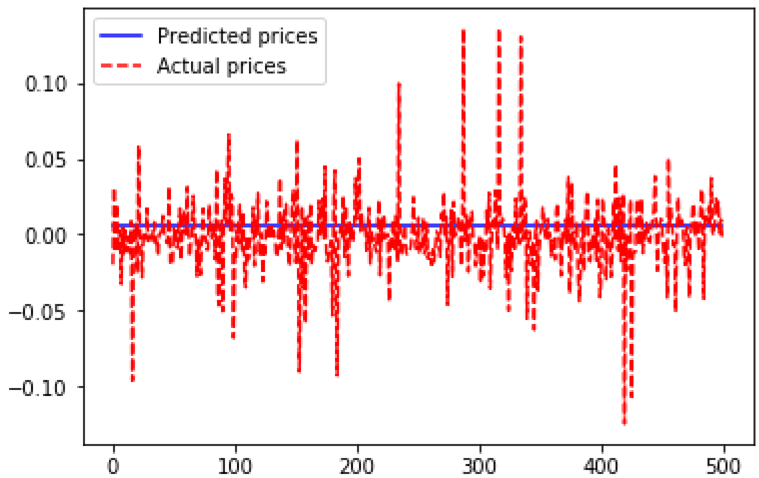
a. Daily returns

In [197]:
```
# Y = df.iloc[:,147]
pro=ProgressBar()
for col in pro(dailyReturn_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)
    model = LinearRegression()

    y_rbf = model.fit(x_train, y_train).predict(x_test)
    if col is 'Ret_121':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (60 of 60) |########################| Elapsed Time: 0:00:27 Time:  0:00:
27
```

In [198]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m1= mean_squared_error(res['x'], res['y'])
print("MSE error:",m1)
```

```
          x         y
0  0.006286 -0.019512
1  0.006286  0.031098
2  0.006286 -0.011105
3  0.006286  0.020268
4  0.006286 -0.010936
MSE error: 2.227150463237327e-05
```



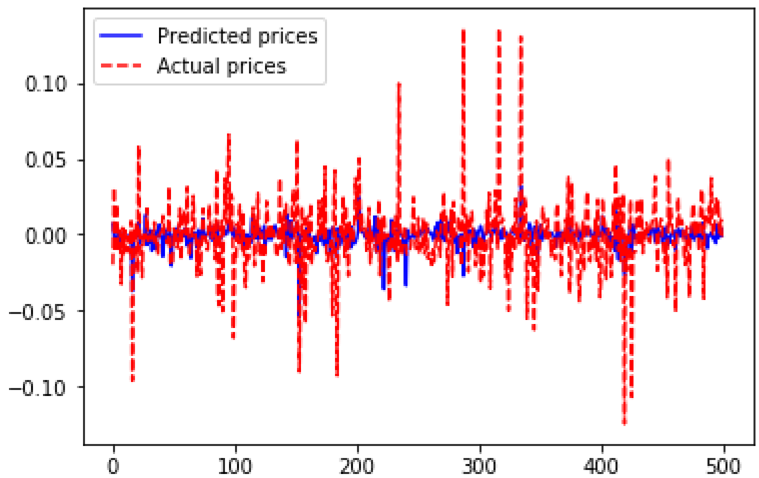
b. Intra day returns

In [199]:
```
# Y = df.iloc[:,147]
plot_res=pd.DataFrame()
res=pd.DataFrame()
pro=ProgressBar()
for col in pro(intraDay_cols):
    Y = df[col]
    y_train=Y[msk]
    y_test=Y[~msk]

    y_train = y_train.astype('float32')
    y_train = y_train.fillna(0)

    y_test = y_test.astype('float32')
    y_test = y_test.fillna(0)

    model = LinearRegression()
    #svr_lin = SVR(kernel='linear', C=1e3)
    #svr_poly = SVR(kernel='poly', C=1e3, degree=2)

    model.fit(x_train, y_train)

    if col is 'Ret_PlusOne':
        plot_res=plot_res.append(pd.DataFrame({'x':model.predict(x_test).flatt
en().tolist(), 'y':y_test.tolist()}))
    res=res.append(pd.DataFrame({'x':model.predict(x_test).flatten().tolist(),
'y':y_test.tolist()}))
```

```
100% (2 of 2) |##########################| Elapsed Time: 0:00:00 Time:  0:00:
00
```

In [200]:
```
print(plot_res.head())
val=500
plt.plot(range(0, val), plot_res['x'][:val], 'b-', label='Predicted prices')
plt.plot(range(0, val), plot_res['y'][:val], 'r--', label='Actual prices')
plt.legend()

m2= mean_squared_error(res['x'].values, res['y'].values)
print("MSE error:",m2)
```

```
          x         y
0  0.006958 -0.019512
1 -0.001696  0.031098
2 -0.004265 -0.011105
3 -0.004253  0.020268
4  0.002089 -0.010936
MSE error: 0.0005815655752938785
```



In [201]:
```
conc.add_row(["Linear regression ", m1, m2])
print(conc)
```

```
+-----------------------------------------------------+------------------------+
--------------------------+
|                     Model Name                      | MSE for daily returns  |
MSE for intra day returns |
+-----------------------------------------------------+------------------------+
--------------------------+
|              LSTM With masking layer                |  0.000650198712438325  |
1.2613788683900336e-06  |
|                      XGboost                        | 1.2405071522992503e-06 |
0.0005016972798379116   |
|                ExtraTreesRegressor                  | 1.2636426776403615e-06 |
0.0006274016286506527   |
|             LSTM with 2 hidden layer                | 3.4314954120847986e-05 |
0.0006371389497042122   |
|             LSTM with 4 hidden layer                | 1.5848400656118064e-06 |
0.0005776596093536332   |
| LSTM with 4 hidden layer and batchnormal layers   | 2.1893272052289176e-05 |
0.0006294481172375907   |
|                  RBF kernel SVR                     | 0.00011203147433272193 |
0.0006476464760422494   |
|                Linear regression                    | 2.227150463237327e-05  |
0.0005815655752938785   |
+-----------------------------------------------------+------------------------+
--------------------------+
```

Conclusion

In [202]:
```
print(conc)
```

```
+-----------------------------------------------+-----------------------+
--------------------------+
|                   Model Name                  | MSE for daily returns  |
MSE for intra day returns |
+-----------------------------------------------+-----------------------+
--------------------------+
|             LSTM With masking layer           |  0.000650198712438325  |
1.2613788683900336e-06   |
|                     XGboost                   | 1.2405071522992503e-06 |
0.0005016972798379116    |
|               ExtraTreesRegressor             | 1.2636426776403615e-06 |
0.0006274016286506527    |
|             LSTM with 2 hidden layer          | 3.4314954120847986e-05 |
0.0006371389497042122    |
|             LSTM with 4 hidden layer          | 1.5848400656118064e-06 |
0.0005776596093536332    |
| LSTM with 4 hidden layer and batchnormal layers | 2.1893272052289176e-05 |
0.0006294481172375907    |
|                  RBF kernel SVR               | 0.00011203147433272193 |
0.0006476464760422494    |
|                 Linear regression             | 2.227150463237327e-05  |
0.0005815655752938785    |
+-----------------------------------------------+-----------------------+
--------------------------+
```

1. Out of 852 teams that used this data, only 370 (43%) did better than predicting 0 for every stock price.
2. The Model 1 (LSTM with masking) doesn't get improve if we replace the NaN values with median and mean in train and test data.
3. For Daily Returns XGboost has less MSE and LSTM's worked well for sequence data.
4. For Intra day Returns LSTM with masking layer has less MSE.

References

http://blog.kaggle.com/2016/02/12/winton-stock-market-challenge-winners-interview-3rd-place-mendrika-ramarlina/ (http://blog.kaggle.com/2016/02/12/winton-stock-market-challenge-winners-interview-3rd-place-mendrika-ramarlina/)

https://medium.com/better-programming/handling-missing-data-in-python-using-scikit-imputer-7607c8957740 (https://medium.com/better-programming/handling-missing-data-in-python-using-scikit-imputer-7607c8957740)

https://www.kaggle.com/c/the-winton-stock-market-challenge/discussion (https://www.kaggle.com/c/the-winton-stock-market-challenge/discussion)

https://navinjain.blog/2018/12/02/the-winton-stock-market-challenge/ (https://navinjain.blog/2018/12/02/the-winton-stock-market-challenge/)

https://github.com/KhaledSharif/winton-stock-market/blob/master/submission.py (https://github.com/KhaledSharif/winton-stock-market/blob/master/submission.py)