In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab

%matplotlib inline
```

In [2]:

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

In [3]:

```
# Read stock dataset
# Download dataset from "https://drive.google.com/uc?id=1pP0Rr83ri0voscgr95-YnVCBv6BYV22w&e
stock_data = pd.read_csv("data_stocks.csv")
```

In [4]:

```
# view header of dataframe
stock_data.head()
```

Out[4]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDA
0	1491226200	2363.6101	42.3300	143.6800	129.6300	82.040	1(
1	1491226260	2364.1001	42.3600	143.7000	130.3200	82.080	1(
2	1491226320	2362.6799	42.3100	143.6901	130.2250	82.030	1(
3	1491226380	2364.3101	42.3700	143.6400	130.0729	82.000	1(
4	1491226440	2364.8501	42.5378	143.6600	129.8800	82.035	1(

5 rows × 502 columns

In [5]:

```
stock_data.shape
print("shape of the DataFrame is : {}".format(stock_data.shape))
print("Rows : {0} \nColumns : {1}".format(stock_data.shape[0],stock_data.shape[1]))
```

```
shape of the DataFrame is : (41266, 502)
Rows : 41266
Columns : 502
```

In [6]:

```
stock_data.dtypes[0:5]
```

Out[6]:

```
DATE              int64
SP500           float64
NASDAQ.AAL      float64
NASDAQ.AAPL     float64
NASDAQ.ADBE     float64
dtype: object
```

In [7]:

```
# check type of data in the stock dataset
stock_data.dtypes.unique()
```

Out[7]:

```
array([dtype('int64'), dtype('float64')], dtype=object)
```

In [8]:

```
# check columns of the stock_data
stock_data.columns
```

Out[8]:

```
Index(['DATE', 'SP500', 'NASDAQ.AAL', 'NASDAQ.AAPL', 'NASDAQ.ADBE',
       'NASDAQ.ADI', 'NASDAQ.ADP', 'NASDAQ.ADSK', 'NASDAQ.AKAM', 'NASDAQ.ALX
N',
       ...
       'NYSE.WYN', 'NYSE.XEC', 'NYSE.XEL', 'NYSE.XL', 'NYSE.XOM', 'NYSE.XR
X',
       'NYSE.XYL', 'NYSE.YUM', 'NYSE.ZBH', 'NYSE.ZTS'],
      dtype='object', length=502)
```

In [9]:

```
# summary of stock_data dataframe
stock_data.describe()
```

Out[9]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.AD
count	4.126600e+04	41266.000000	41266.000000	41266.000000	41266.00000	41266.00000
mean	1.497749e+09	2421.537882	47.708346	150.453566	141.31793	79.44687
std	3.822211e+06	39.557135	3.259377	6.236826	6.91674	2.00028
min	1.491226e+09	2329.139900	40.830000	140.160000	128.24000	74.80000
25%	1.494432e+09	2390.860100	44.945400	144.640000	135.19500	78.03000
50%	1.497638e+09	2430.149900	48.360000	149.945000	142.26000	79.41000
75%	1.501090e+09	2448.820100	50.180000	155.065000	147.10000	80.58000
max	1.504210e+09	2490.649900	54.475000	164.510000	155.33000	90.44000

8 rows × 502 columns

In [10]:

```
# As we observed from the stock_data header,
# Date is in UNIX format which need to converted to make this dataset in readable format
stock_data['DATE'] = pd.to_datetime(stock_data['DATE'],unit='s')
```

In [11]:

```
# Check the header after changing the Date format
stock_data.head()
```

Out[11]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP	NASDAQ
0	2017-04-03 13:30:00	2363.6101	42.3300	143.6800	129.6300	82.040	102.2300	8
1	2017-04-03 13:31:00	2364.1001	42.3600	143.7000	130.3200	82.080	102.1400	8
2	2017-04-03 13:32:00	2362.6799	42.3100	143.6901	130.2250	82.030	102.2125	8
3	2017-04-03 13:33:00	2364.3101	42.3700	143.6400	130.0729	82.000	102.1400	8
4	2017-04-03	2364.8501	42.5378	143.6600	129.8800	82.035	102.0600	8

In [12]:

```
# As observed from the above header,
# All the stocks are having multiple entries for each day So there is need to convert stock
# Open, Closse, High and Low price for any perticular day
```

In [13]:

```
# Create a copy of stock_data DataFrame
stock_data_copy = stock_data.copy()
```

In [14]:

```
# Create DF_list -> Dictionary which contains all the stocks data items with the values as
# Create DF_list1 -> Dictionary which contains all the stocks data items with the Closing v
DF_list = {}
DF_list1 = {}
df2 = pd.DataFrame()
for column in stock_data_copy.columns[1:]:
    try :
        df_col = column.split('.')[1]
        #print(df_col)
    except :
        df_col = column

    Open = stock_data_copy.groupby([stock_data_copy['DATE'].dt.date])[column].first()
    Low = stock_data_copy.groupby([stock_data_copy['DATE'].dt.date])[column].min()
    High = stock_data_copy.groupby([stock_data_copy['DATE'].dt.date])[column].max()
    Close = stock_data_copy.groupby([stock_data_copy['DATE'].dt.date])[column].last()

    df = pd.DataFrame([Open,Close,Low,High])
    #df1 = pd.concat([df1,Close])
    df1 = pd.DataFrame([Close])

    df = df.transpose()
    df1 = df1.transpose()

    df.columns = ['Open','Low','High','Close']
    df1.columns = [df_col]

    DF_list[df_col] = df
    DF_list1[df_col] = df1
    #df2.join(df1)
    #print(df.head())
```

In [15]:

```
# Create a DataFrame df5 (Merge all the stock data present in the dictionary DF_list1)
df5 = DF_list1['AAL'].reset_index()
for item in DF_list1.keys():
    df5 = pd.merge(df5,DF_list1[item].reset_index())
df5.index = df5['DATE']
df5 = df5.drop('DATE', axis=1)
```

In [16]:

```
# View header of the merged DataFrame df5
df5.head()
```

Out[16]:

	AAL	SP500	AAPL	ADBE	ADI	ADP	ADSK	AKAM	ALXN	AMAT	...	WYN
DATE												
2017-04-03	42.48	2358.9600	143.700	129.62	81.22	101.27	85.39	58.10	119.28	38.89	...	84.0
2017-04-04	40.90	2359.9600	144.770	130.04	81.24	101.45	84.95	59.30	118.30	39.00	...	84.1
2017-04-05	41.31	2352.8401	144.020	129.89	80.04	101.93	83.54	58.80	116.21	38.38	...	84.3
2017-04-06	41.72	2357.6699	143.685	130.15	80.32	101.75	84.07	58.73	115.49	38.59	...	85.1
2017-04-07	41.81	2355.6899	143.340	130.22	80.01	102.19	84.79	58.10	115.62	38.93	...	84.6

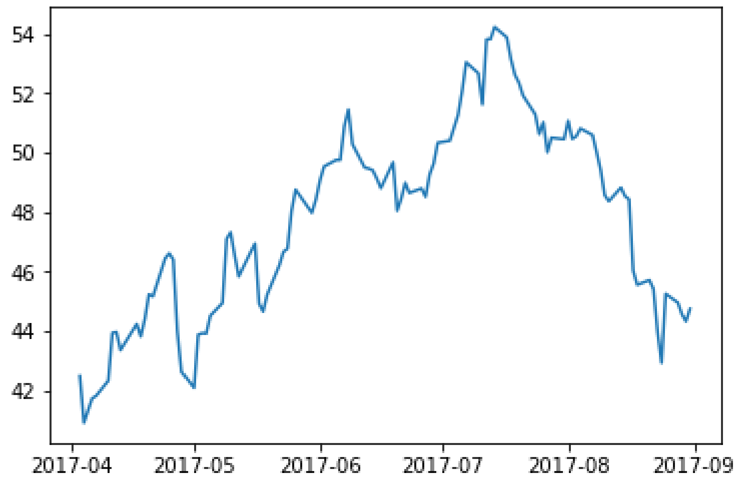
5 rows × 501 columns

In [17]:

```
# Plot stock AAL
plt.plot(df5['AAL'])
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x8fac828>]
```



In [18]:

```
# Plot stocks present in the DataFrame df5
df5.plot(figsize=(75,20))
plt.ylabel('Price')
```

Out[18]:

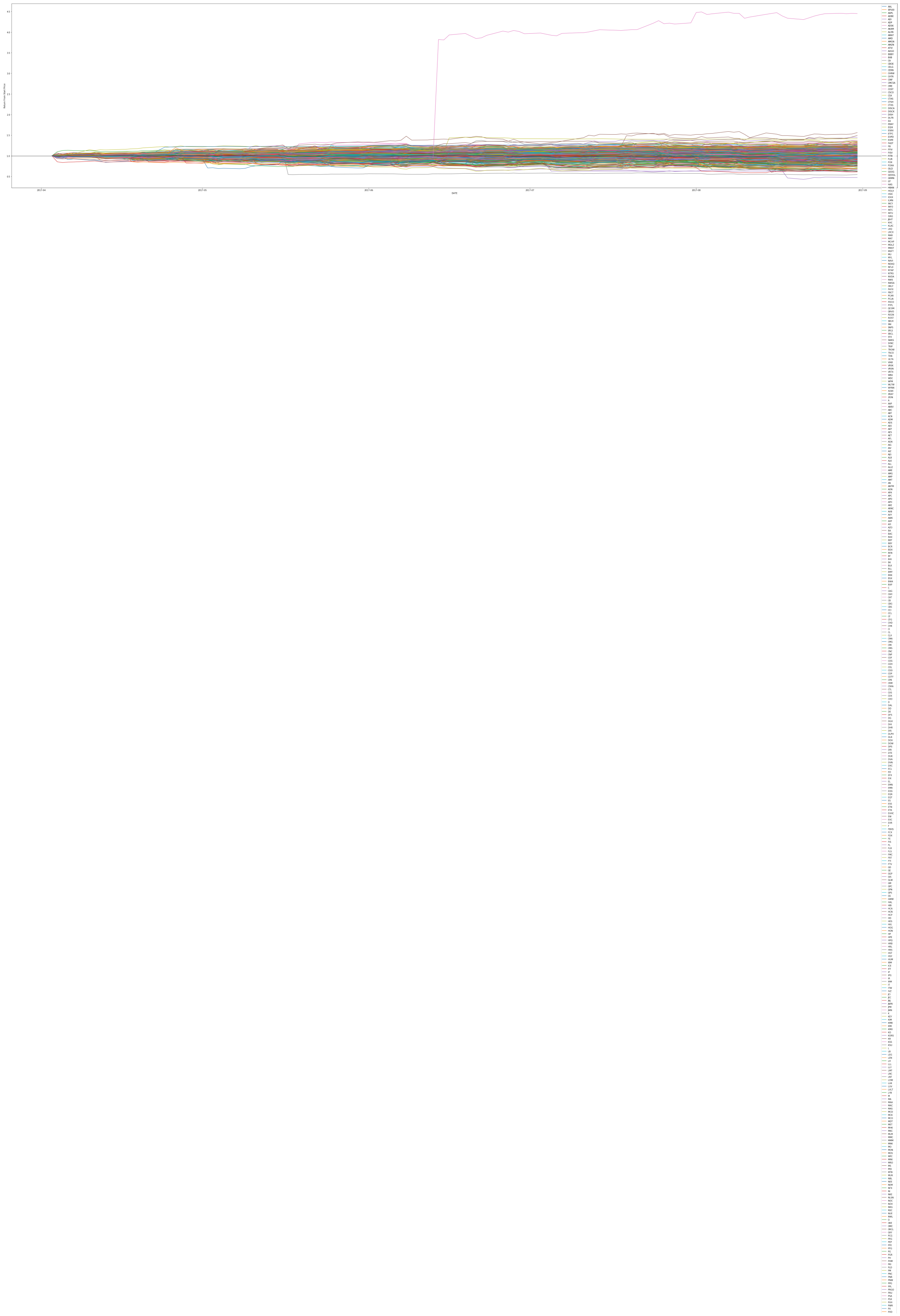
```
Text(0,0.5,'Price')
```

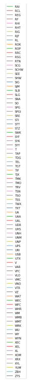
In [19]:

```
# Check return from start price for all the stocks
returnfstart = df5.apply(lambda x: x / x[0])
returnfstart.plot(figsize=(75,16)).axhline(1, lw=1, color='black')
plt.ylabel('Return From Start Price')
```

Out[19]:

```
Text(0,0.5,'Return From Start Price')
```





In [20]:

```
df6=df5.pct_change()

df5.plot(figsize=(30,12))
plt.axhline(0, color='black', lw=1)
plt.ylabel('Daily Percentage Return')
```

Out[20]:

```
Text(0,0.5,'Daily Percentage Return')
```

In [21]:

```
df5.describe()
```

Out[21]:

	AAL	SP500	AAPL	ADBE	ADI	ADP	ADSK	
count	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	1
mean	47.719717	2421.181520	150.372602	141.345755	79.393632	103.468726	102.977642	
std	3.272090	40.031479	6.271409	6.950568	1.984853	4.424074	9.402722	
min	40.900000	2329.139900	140.680000	129.050000	75.490000	96.460000	83.540000	
25%	44.942500	2391.602550	144.597500	135.237500	77.995000	101.316250	94.902500	
50%	48.410000	2429.670050	150.065000	142.055000	79.230000	102.440000	106.730000	
75%	50.307500	2447.609975	155.237500	147.302500	80.547500	104.587500	110.640000	
max	54.220000	2480.600100	163.980000	155.160000	86.130000	118.910000	114.970000	

8 rows × 501 columns

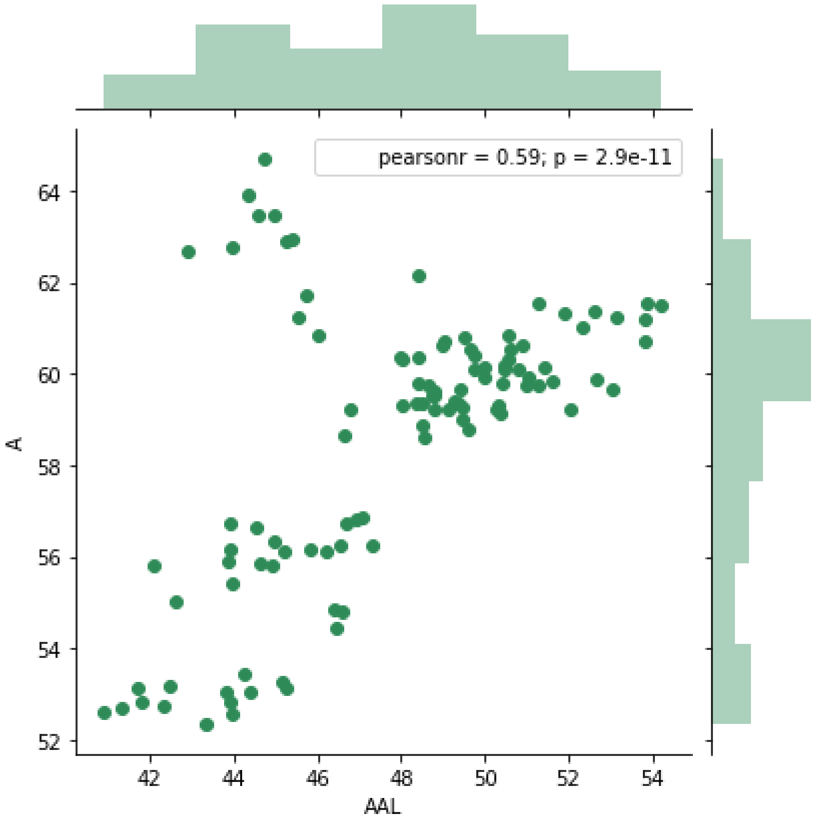
In [22]:

```
# Check Co-relation between 2 stocks using seaborn library
import seaborn as sns
sns.jointplot('AAL', 'A', df5, kind='scatter', color='seagreen')
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[22]:

```
<seaborn.axisgrid.JointGrid at 0x1837c7b8>
```

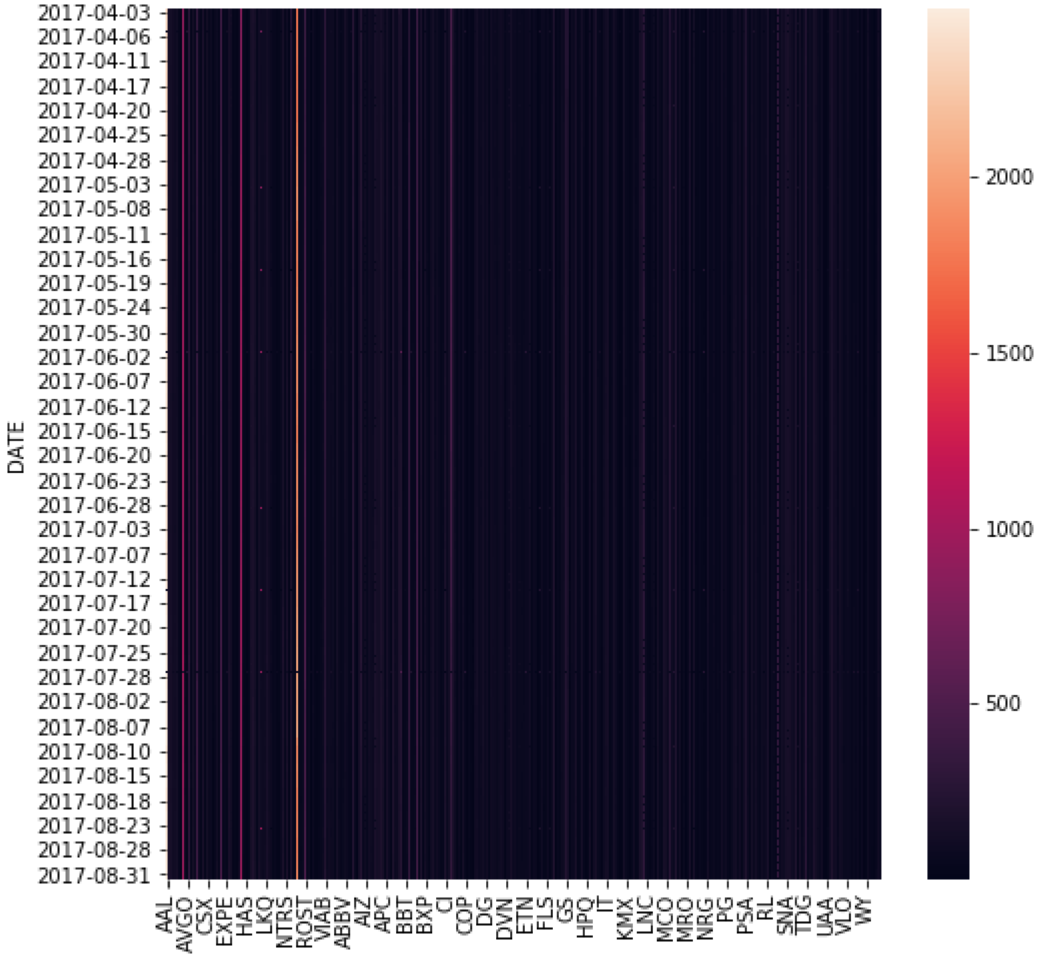


In [23]:

```
# Use Seaborn to plot Co-relation between all the stocks
plt.figure(figsize=(8,8))
sns.heatmap(df5.dropna())
```

Out[23]:

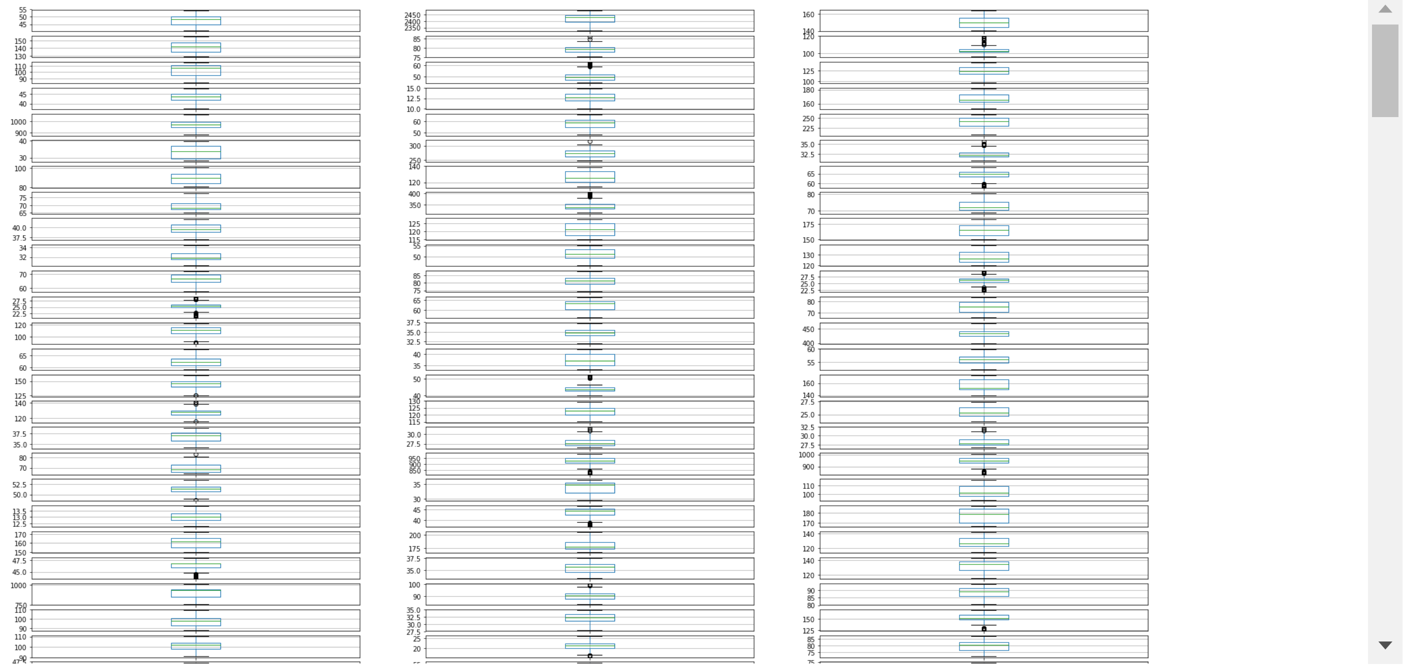
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c09c5c0>
```



In [24]:

```
# Outliers visualization using boxplot
fig, axes = plt.subplots(len(df5.columns)//3, 3, figsize=(30, 120))

i = 0
for triaxis in axes:
    for axis in triaxis:
        df5.boxplot(column = df5.columns[i], ax=axis)
        i = i+1
plt.show()
```



In [25]:

```
# Function to perform scaling on thr stock dataset
def standard_scaler(X_train, X_test):
    train_samples, train_nx, train_ny = X_train.shape
    test_samples, test_nx, test_ny = X_test.shape

    X_train = X_train.reshape((train_samples, train_nx * train_ny))
    X_test = X_test.reshape((test_samples, test_nx * test_ny))

    preprocessor = prep.StandardScaler().fit(X_train)
    X_train = preprocessor.transform(X_train)
    X_test = preprocessor.transform(X_test)

    X_train = X_train.reshape((train_samples, train_nx, train_ny))
    X_test = X_test.reshape((test_samples, test_nx, test_ny))

    return X_train, X_test
```

In [26]:

```
# Function to perfrom pre-processing on the stock dataset
def preprocess_data(stock, seq_len):
    amount_of_features = len(stock.columns)
    data = stock.as_matrix()

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index : index + sequence_length])

    result = np.array(result)
    row = round(0.9 * result.shape[0])
    train = result[: int(row), :]

    train, result = standard_scaler(train, result)

    X_train = train[:, : -1]
    y_train = train[:, -1][: ,-1]
    X_test = result[int(row) :, : -1]
    y_test = result[int(row) :, -1][ : ,-1]

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], amount_of_features))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], amount_of_features))

    return [X_train, y_train, X_test, y_test]
```

In [27]:

```
import time
import math
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import numpy as np
import pandas as pd
import sklearn.preprocessing as prep
```

```
Using TensorFlow backend.
```

In [28]:

```
# Function to build LSTM model -> Apply Dropout -> Create Dense Layer -> Apply Activation f
# Use MSE as loss function -> Calculate compilation time
def build_model(layers):
    model = Sequential()

    model.add(LSTM(
        input_dim=layers[0],
        output_dim=layers[1],
        return_sequences=True))
    model.add(Dropout(0.4))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.3))

    model.add(Dense(
        output_dim=layers[3]))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop", metrics=['accuracy'])
    print("Compilation Time : ", time.time() - start)
    return model
```

In [29]:

```
# Split stock dataset into train-test dataset
window = 20
X_train, y_train, X_test, y_test = preprocess_data(df5[:: -1], window)
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train (76, 20, 501)
y_train (76,)
X_test (9, 20, 501)
y_test (9,)
```

In [30]:

```
# Call function to build Model
model = build_model([X_train.shape[2], window, 100, 1])
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: UserWa
rning: The `input_dim` and `input_length` arguments in recurrent layers are
deprecated. Use `input_shape` instead.
  # Remove the CWD from sys.path while we load stuff.
C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: UserWa
rning: Update your `LSTM` call to the Keras 2 API: `LSTM(return_sequences=Tr
ue, input_shape=(None, 501..., units=20)`
  # Remove the CWD from sys.path while we load stuff.

Compilation Time :  0.09200525283813477

C:\Users\santhu\Anaconda3\lib\site-packages\ipykernel_launcher.py:19: UserWa
rning: Update your `Dense` call to the Keras 2 API: `Dense(units=1)`
```

In [31]:

```
# Fit the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[31]:

```
<keras.callbacks.History at 0x49d7d630>
```

In [32]:

```
# Calculate test/train score
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])))

testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
```

```
Train Score: 0.08 MSE (0.28 RMSE)
Test Score: 2.82 MSE (1.68 RMSE)
```

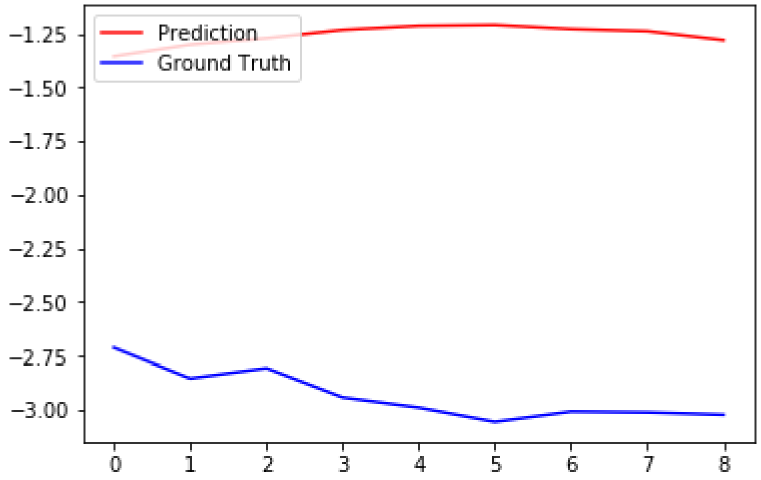
In [33]:

```
diff = []
ratio = []
pred = model.predict(X_test)
for u in range(len(y_test)):
    pr = pred[u][0]
    ratio.append((y_test[u] / pr) - 1)
    diff.append(abs(y_test[u] - pr))
```

In [34]:

```
import matplotlib.pyplot as plt2

plt2.plot(pred, color='red', label='Prediction')
plt2.plot(y_test, color='blue', label='Ground Truth')
plt2.legend(loc=2)
plt2.show()
```


Prediction
Ground Truth