Introduction and Scope

The S&P 500 stock market index was created in 1957, and it's a collection of US equities used to track the performance of the broader American equity market. Today, this index tracks the performance of 505 common stocks issued by 500 of the largest firms in the US by market capitalisation, and the index accounts for roughly 80% of the US equity market in terms of market cap.

The S&P 500 index has changed over time as different public companies have been added to or removed from the index as a result of their growth or decline. Companies have also been removed from the index due to other reasons such as being taken private by private equity firms or because of mergers and acquisitions. For simplicity, the project will only use price data on extant constituents of the S&P 500 and will not consider companies who used to be part of the S&P 500.

The goal of this project is to use a long short-term memory (LSTM) network to generate buy and sell signals for stock that is part of the S&P 500 index. To achieve this, the neural network will be trained on the daily stock price data of the constituents of the S&P500 index from 03/01/2001 to 15/10/2021. In this kernel, the stock that signals will be generated for is Apple.

Import packages and set up file directory

In [1]:

```
import os
import pandas_datareader.data as web # for obtaining stock data from Yahoo Financ
e
import datetime as dt
import numpy as np
import pandas as pd
import random
from collections import deque
from sklearn import preprocessing
import time
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, BatchNormalization
from tensorflow.compat.v1.keras.layers import CuDNNLSTM
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.callbacks import ModelCheckpoint
import pickle

# Configuring chart settings
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import style
style.use('seaborn') ## ggplot
```

Getting the S&P 500 Data

This is the code that I used to obtain the stock price data for all the stocks in the S&P 500 index.

In [2]:

```
def get_sp500_tickers():
    """
    Returns a data frame of the S&P 500 tickers from the
    Wikipedia on the S&P 500 Index

    Also saves a pickle file of the tickers for future use
    """
    tickers = pd.read_html(
        'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[0]['Symbol'
]

    with open('sp500_tickers','wb') as f:
        pickle.dump(tickers,f)
    return tickers
```

In [3]:

```
def get_sp500_data(reload_sp500 = False):
    """
    Creates a directory called "stocks_dfs" and stores the price
    data of all the stocks in the S&P 500 index in that folder

    Price data on each company is from 03/01/2001 - 15/10/2021
    """
    tickers = []
    if reload_sp500==True:
        tickers = get_sp500_tickers()
    else:
        with open('sp500_tickers','rb') as f:
            tickers = pickle.load(f)

    for i in range(len(tickers)):
        tickers[i] = tickers[i].replace('.', '-')

    if not os.path.exists('stock_dfs'):
        os.makedirs('stock_dfs')

    start = dt.datetime(2000,1,1) # 12/31/1999
    end = dt.datetime.now() # Get up to the most recent time

    for ticker in tickers:
        if not os.path.exists(f'stock_dfs/{ticker}.csv'):
            df = web.DataReader(ticker,'yahoo',start,end)
            df.to_csv(f'stock_dfs/{ticker}.csv')
        else:
            print(f'Already have {ticker}')
```

In [4]:

```
def compile_data():
    with open('sp500_tickers','rb') as f:
        tickers = pickle.load(f)

    for i in range(len(tickers)):
        tickers[i] = tickers[i].replace('.', '-')

    main_df = pd.DataFrame()

    for count, ticker in enumerate(tickers):
        df = pd.read_csv(f'stock_dfs/{ticker}.csv')
        df.set_index('Date',inplace=True)
        df.rename(columns = {'Adj Close':f'{ticker}_close',
                             'Open':f'{ticker}_open',
                             'High':f'{ticker}_high',
                             'Low':f'{ticker}_low',
                             'Volume':f'{ticker}_volume'},
                  inplace = True)
        df.drop(['Close'],axis=1,inplace = True)

        if main_df.empty:
            main_df = df
        else:
            main_df = pd.concat([main_df,df],axis =1)

        if count % 100 == 0:
            print(count)

    main_df.to_csv('sp500_data.csv')
```

Although I couldn't get these functions to work on Kaggle (because I couldn't find a way to access the output directory),

```
get_sp500_data(reload_sp500 = True)
compile_data()
```

the functions do work my local device, so please run this code locally to get the data on the S&P 500's constituents' stock prices. Once you download the data, then upload it to this notebook and continue reading!

Exploring the Data

In [5]:

```
target_stock = 'AAPL' # Let's try investing in Apple
```

In [6]:

```
sp500 = pd.read_csv('/kaggle/input/sp500-dataset/sp500_data.csv',index_col = 0)
```

In [7]:

```
sp500.index = pd.to_datetime(sp500.index)
```

In [8]:

```
# daily stock data on sp500 companies from 31/12/1999 - 23/8/2021
sp500.head(1)
```

Out[8]:

	MMM_high	MMM_low	MMM_open	MMM_close	MMM_volume	MMM_adj_close	AE
2000-01-03	48.25	47.03125	48.03125	47.1875	2173400.0	27.179523	16

1 rows × 3030 columns

In [9]:

```
sp500.tail(1)
```

Out[9]:

	MMM_high	MMM_low	MMM_open	MMM_close	MMM_volume	MMM_adj_close
2021-10-15	183.0	180.679993	180.690002	181.940002	2160800.0	181.940002

1 rows × 3030 columns

In [10]:

```
sp500.shape
```

Out[10]:

```
(5483, 3030)
```

Removing the "close" columns from the data and keeping "adjusted close" prices

Remove the "close" columns from the dataframe and keep only the adjusted closes, which account for price adjustments after corporate actions (like stock splits)

In [11]:

```
# remove the close column
def remove_close(df):
    df = df[(col for col in df.columns if 'high' in col or 'low' in col or 'ope
n' in col or 'volume' in col or 'adj_close' in col)]
    return df
```

In [12]:

```
sp500 = remove_close(sp500)

# there should be 505 fewer columns
sp500.shape
```

Out[12]:

```
(5483, 2525)
```

Adding each stock's intraday spread to our data

- intraday spread is defined as: $P_{high} - P_{low}$

In [13]:

```
def add_intraday_spread(df):
    high_cols = sorted([col for col in df.columns if 'high' in col])
    low_cols = sorted([col for col in df.columns if 'low' in col])
    for i in range(len(high_cols)):
        df[f"{high_cols[i].split('_')[0]}_intraday_spread"] = df[high_cols[i]] -
df[low_cols[i]]
    return df
```

In [14]:

```
sp500 = add_intraday_spread(sp500)

# now we'll have 3030 columns again
sp500.shape
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:5: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  """
```

Out[14]:

```
(5483, 3030)
```

Data visualisation

Apple's adjusted closing price and trading volume

In [15]:

```
mpl.rcParams['figure.dpi'] = 300
mpl.rcParams["figure.figsize"] = [20,8]
font = {'family': 'serif',
        'weight': 'normal',
        'size': 18}
mpl.rc('font', **font)
```

In [16]:

```
ax1 = plt.subplot2grid((7,1),(0,0),rowspan = 4,colspan = 1)
ax2 = plt.subplot2grid((7,1),(5,0),rowspan = 3,colspan = 1)

# Plot 1
ax1.plot(sp500.index,sp500['AAPL_adj_close'],color = 'blue',linewidth=1)
ax1.title.set_text("Apple's Adjusted Closing Price and Trading Volume")
ax1.legend(['Apple Stock Adjusted Closing Price'])

# Plot 2
ax2.bar(sp500.index,sp500['AAPL_volume'])
ax2.legend(['Volume'])
```

Out[16]:

```
<matplotlib.legend.Legend at 0x7f51d28e1690>
```

In [17]:

```
sp500.head(2)
```

Out[17]:

	MMM_high	MMM_low	MMM_open	MMM_volume	MMM_adj_close	ABT_high	AE
2000-01-03	48.25000	47.03125	48.03125	2173400.0	27.179523	16.160433	15
2000-01-04	47.40625	45.31250	46.43750	2713800.0	26.099533	15.599306	15

2 rows × 3030 columns

In [18]:

```
sp500.shape
```

Out[18]:

```
(5483, 3030)
```

Plotting the Autocorrelation and Partial Autocorrelation of 3 Stocks' Adjusted Closing Price

In [19]:

```
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf

def tsplot(y,title,lags=None,figsize=(12,8)):
    """
    Examine the patterns of ACF and PACF as well as the time series plot and histo
gram of the variable
    """
    fig = plt.figure(figsize=figsize)
    layout = (2,2)

    ts_ax = plt.subplot2grid(layout, (0,0))
    hist_ax = plt.subplot2grid(layout, (0,1))
    acf_ax = plt.subplot2grid(layout, (1,0))
    pacf_ax = plt.subplot2grid(layout, (1,1))

    y.plot(ax=ts_ax)
    ts_ax.set_title(title,fontsize=14,weight='bold')
    y.plot(ax=hist_ax,kind='hist',bins = 25,)
    hist_ax.set_title('Histogram')
    plot_acf(y,lags = lags, ax=acf_ax)
    plot_pacf(y,lags = lags, ax=pacf_ax)
    [ax.set_xlim(0) for ax in [acf_ax,pacf_ax]]
    sns.despine()
    plt.tight_layout()

    return ts_ax,acf_ax,pacf_ax
```

In [20]:

```
tsplot(sp500['AAPL_adj_close'].dropna(),title = 'Apple Adjusted Closing Price',lags = 48)
```

Out[20]:

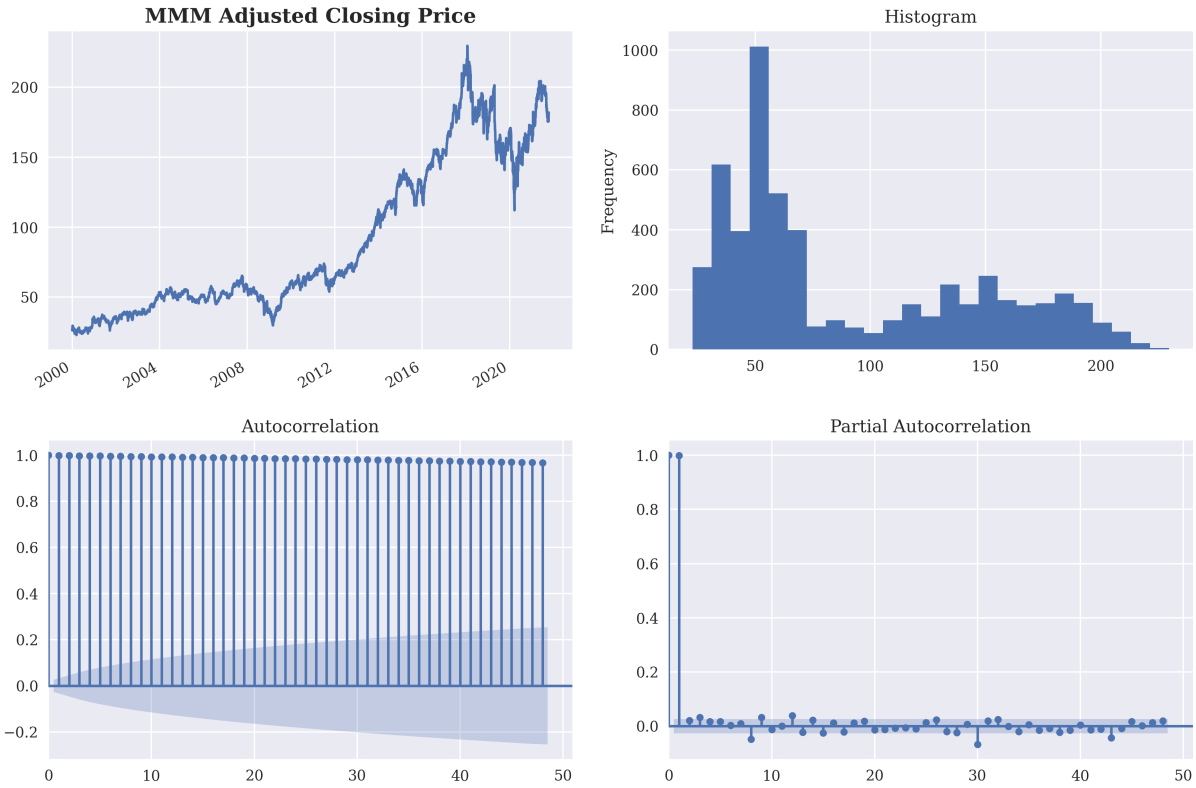
```
(<AxesSubplot:title={'center':'Apple Adjusted Closing Price'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

In [21]:

```
tsplot(sp500['MMM_adj_close'].dropna(),title = 'MMM Adjusted Closing Price',lags = 48)
```

Out[21]:

```
(<AxesSubplot:title={'center':'MMM Adjusted Closing Price'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

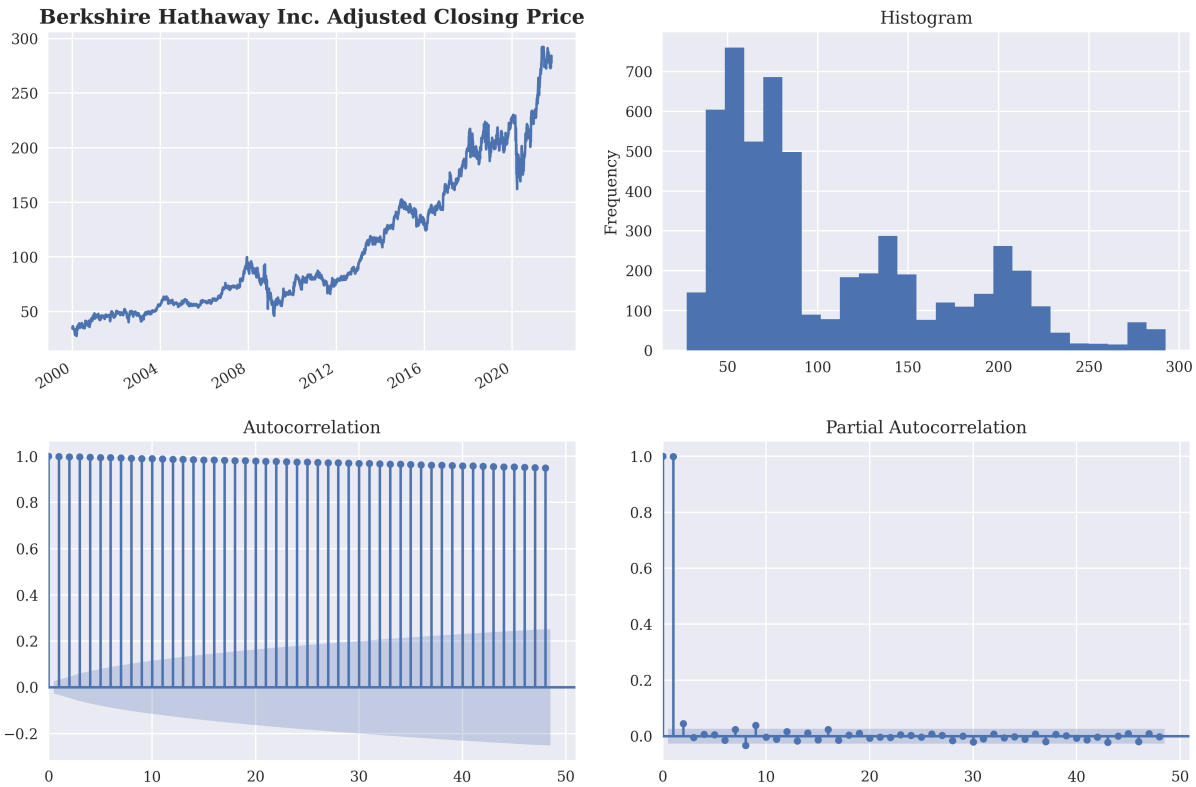


In [22]:

```
tsplot(sp500['BRK-B_adj_close'].dropna(),title = 'Berkshire Hathaway Inc. Adjusted Closing Price',lags = 48)
```

Out[22]:

```
(<AxesSubplot:title={'center':'Berkshire Hathaway Inc. Adjusted Closing Price'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```



The adjusted closing prices of these stocks exhibit autocorrelation and they are non-stationary. To make this data stationary, we can compute the log returns of each company's adjusted close price.

Although stationarity is not required to properly train a LSTM network, it may make the model's predictions more accurate. Therefore, we'll add the returns data to our dataset.

In addition to this motivation, we could compare the LSTM's performance on different datasets. More specifically, we may want to compare the LSTM's performance on 3 different datasets:

1. one that includes the adjusted close data and no returns data
2. one that includes the returns data and no adjusted close data

*For the purpose of this notebook, **stationarity** is defined in terms of its **weak definition**, which implies that a time series has a time independent mean and variance, and that the autocovariance between two observations depends only on the size of the time displacement between the two observations. In other words, for a stationary time series, the autocovariance between observations at time $t = 2$ and $t = 3$ should be the same as the autocovariance between the observations at time $t = 4$ and $t = 5$.*

Making Adjusted Close Stationary: add each stock's log returns to our data

- Log returns computed as: $log(P_{t+1}/P_t)$
  - P_t = adjusted close price at time t

In [23]:

```
# add returns columns for all the stocks
for col in sp500.columns:
    if 'adj_close' in col:
        sp500[f"{col.split('_')[0]}_returns"] = np.log(sp500[col].div(sp500[col]
.shift(1)))
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  after removing the cwd from sys.path.
```

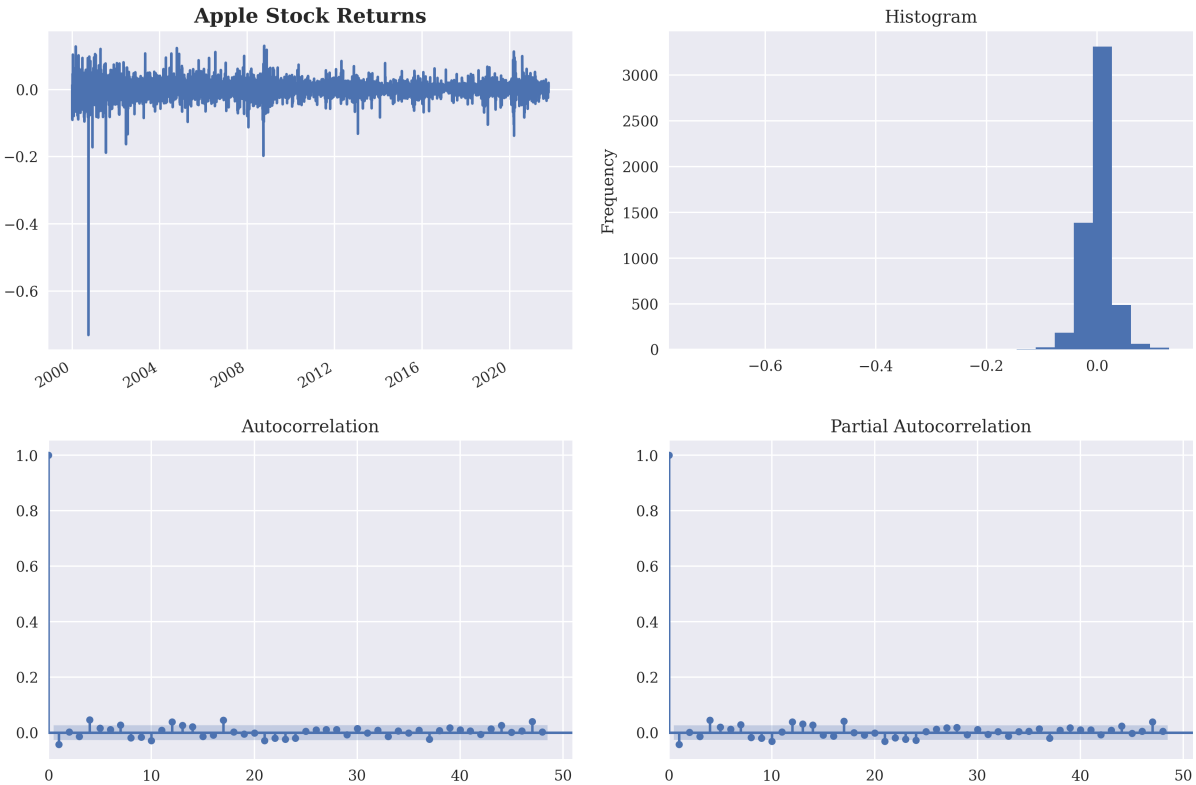
Plotting the Autocorrelation and Partial Autocorrelation of 3 Stocks' Returns

In [24]:

```
# outlier in the returns between 2000 and 2002 due to internet bubble bursting
tsplot(sp500['AAPL_returns'].dropna(),title = "Apple Stock Returns",lags = 48)
```

Out[24]:

```
(<AxesSubplot:title={'center':'Apple Stock Returns'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

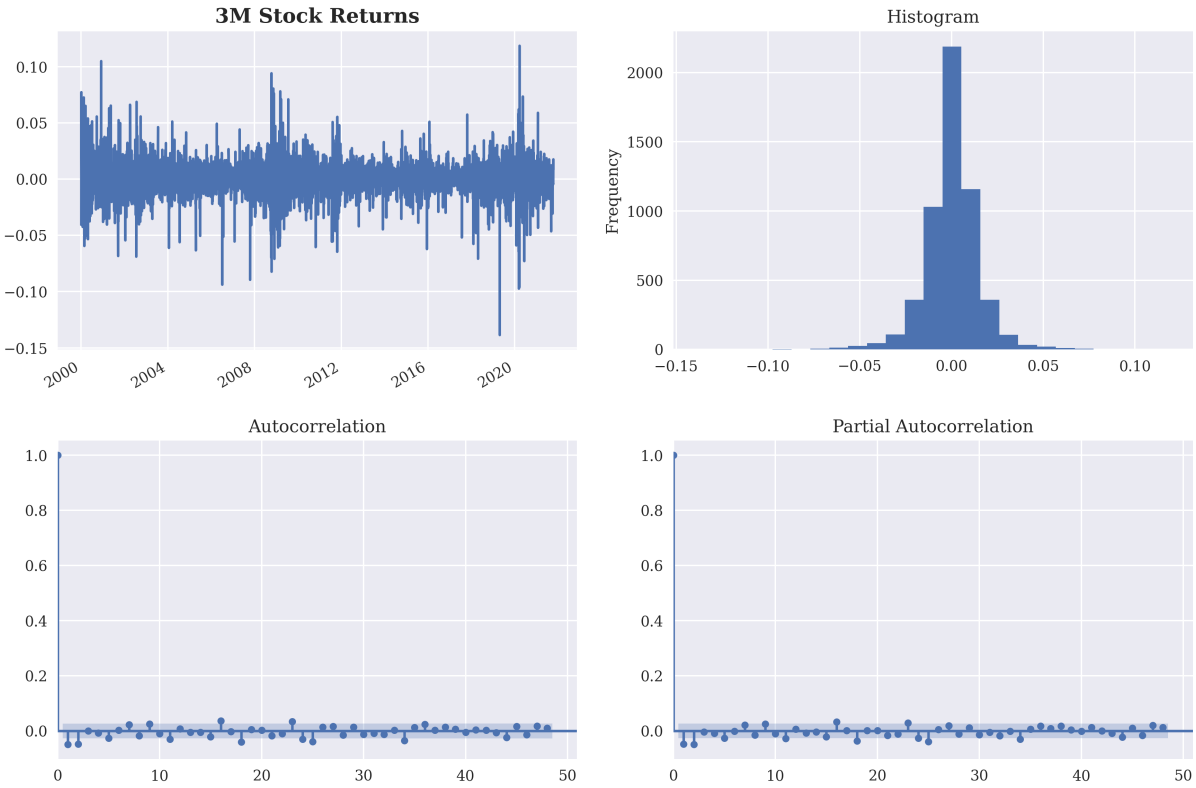


In [25]:

```
# returns look stationary and have a normal distribution
tsplot(sp500['MMM_returns'].dropna(),title = "3M Stock Returns",lags = 48)
```

Out[25]:

```
(<AxesSubplot:title={'center':'3M Stock Returns'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

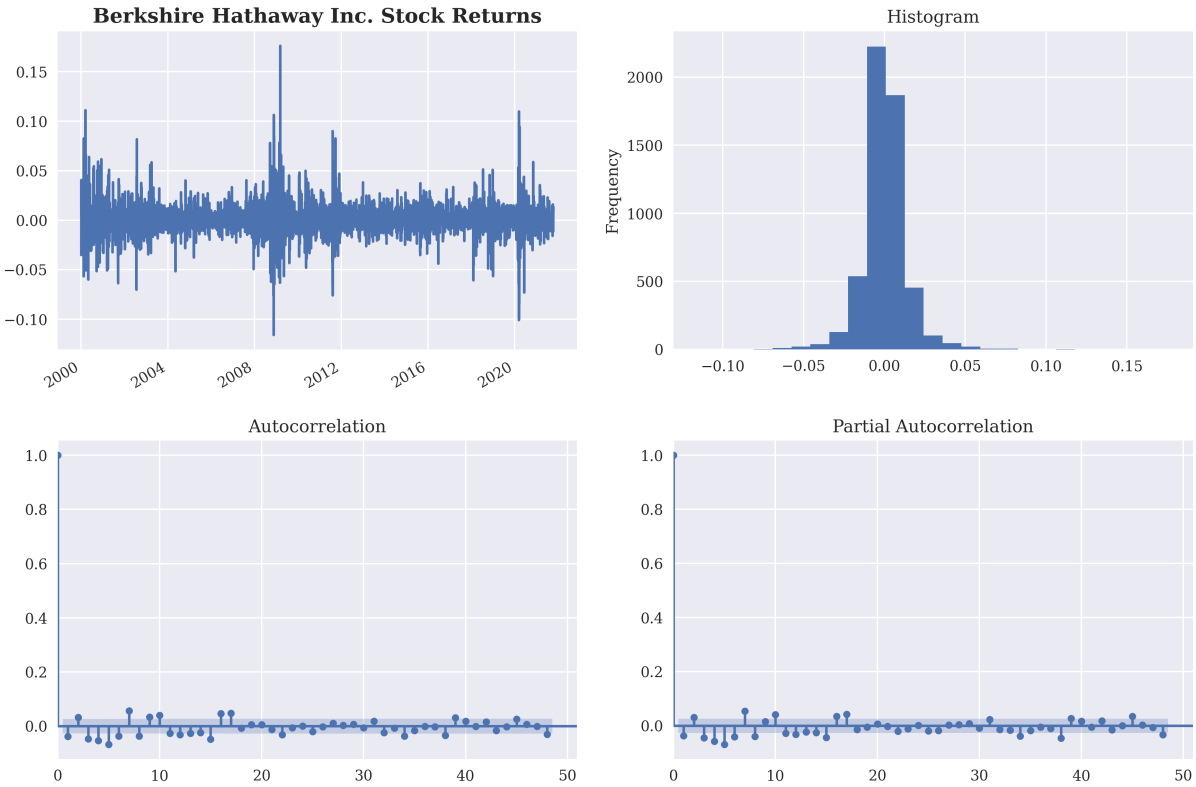


In [26]:

```
# returns look stationary and have a normal distribution
tsplot(sp500['BRK-B_returns'].dropna(),title = "Berkshire Hathaway Inc. Stock Returns",lags = 48)
```

Out[26]:

```
(<AxesSubplot:title={'center':'Berkshire Hathaway Inc. Stock Returns'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```



Plotting Other Variables

It's likely that other variables such as a stock's high price, low price, open price, and volume do not exhibit stationarity. If the variables are non-staionary, we can attempt to make them stationary by taking their log differences as well.

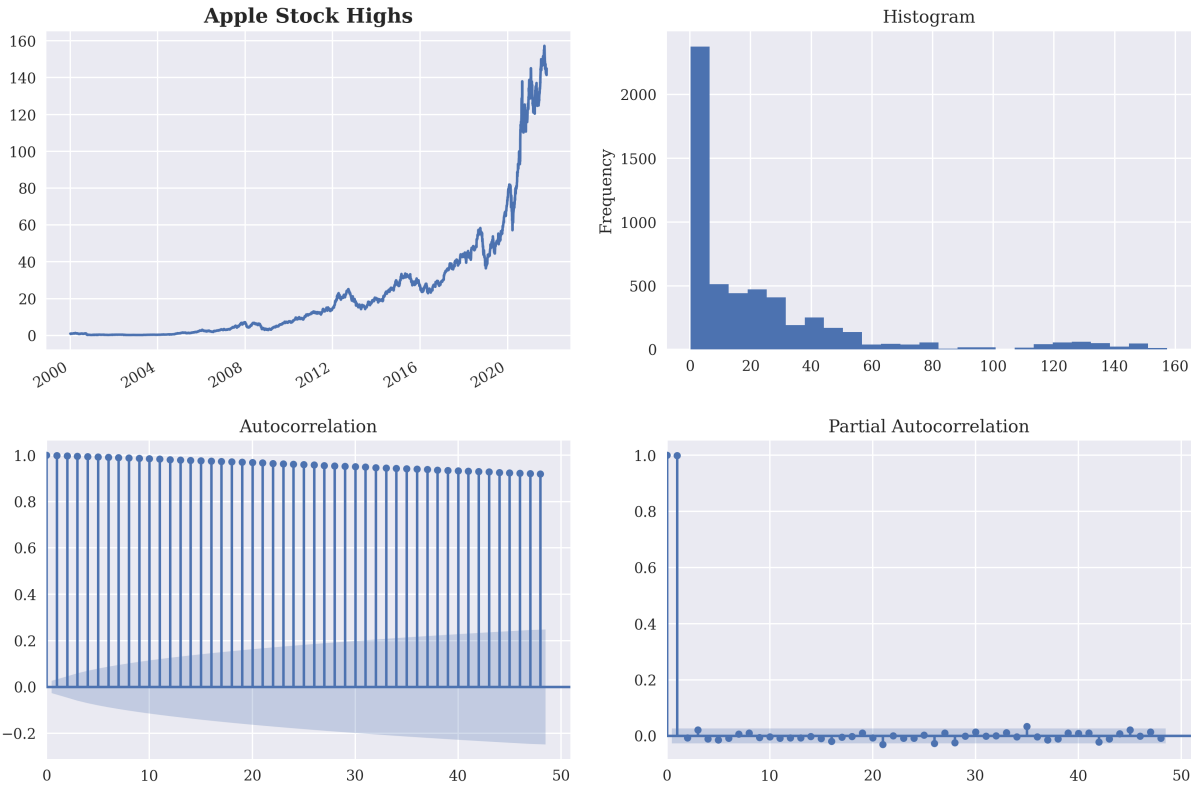
Time Series of Apple Stock's High Price, Low Price, Open Price, Intraday Spread, and Volume

In [27]:

```
# High price
tsplot(sp500['AAPL_high'].dropna(),title = "Apple Stock Highs",lags = 48)
```

Out[27]:

```
(<AxesSubplot:title={'center':'Apple Stock Highs'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

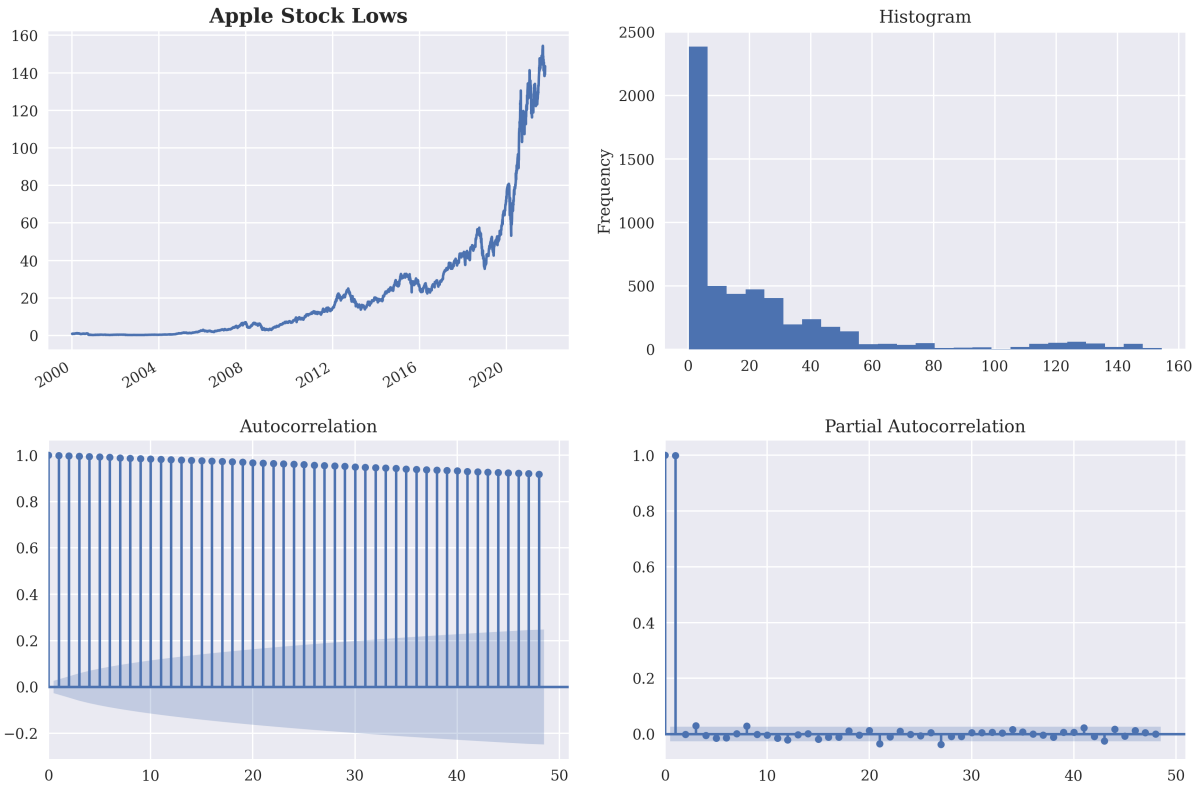


In [28]:

```
# Low price
tsplot(sp500['AAPL_low'].dropna(),title = "Apple Stock Lows",lags = 48)
```

Out[28]:

```
(<AxesSubplot:title={'center':'Apple Stock Lows'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

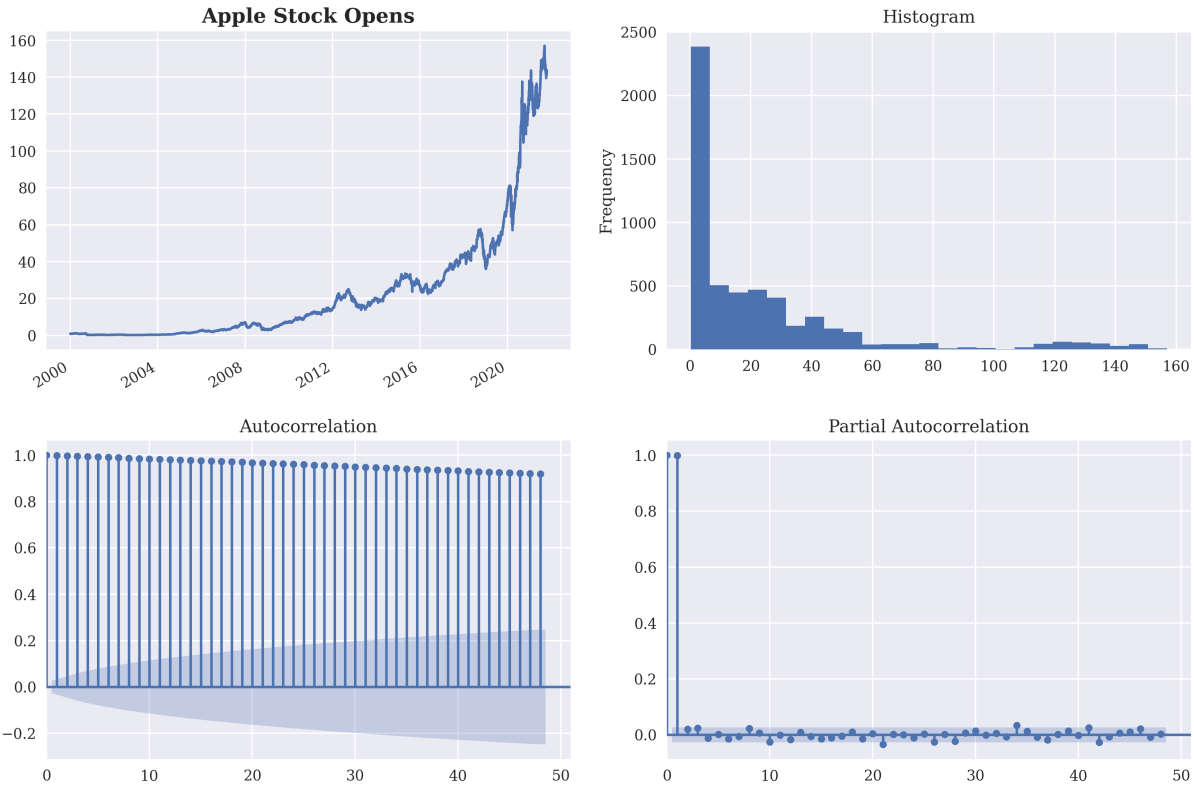


In [29]:

```
# Open price
tsplot(sp500['AAPL_open'].dropna(),title = "Apple Stock Opens",lags = 48)
```

Out[29]:

```
(<AxesSubplot:title={'center':'Apple Stock Opens'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

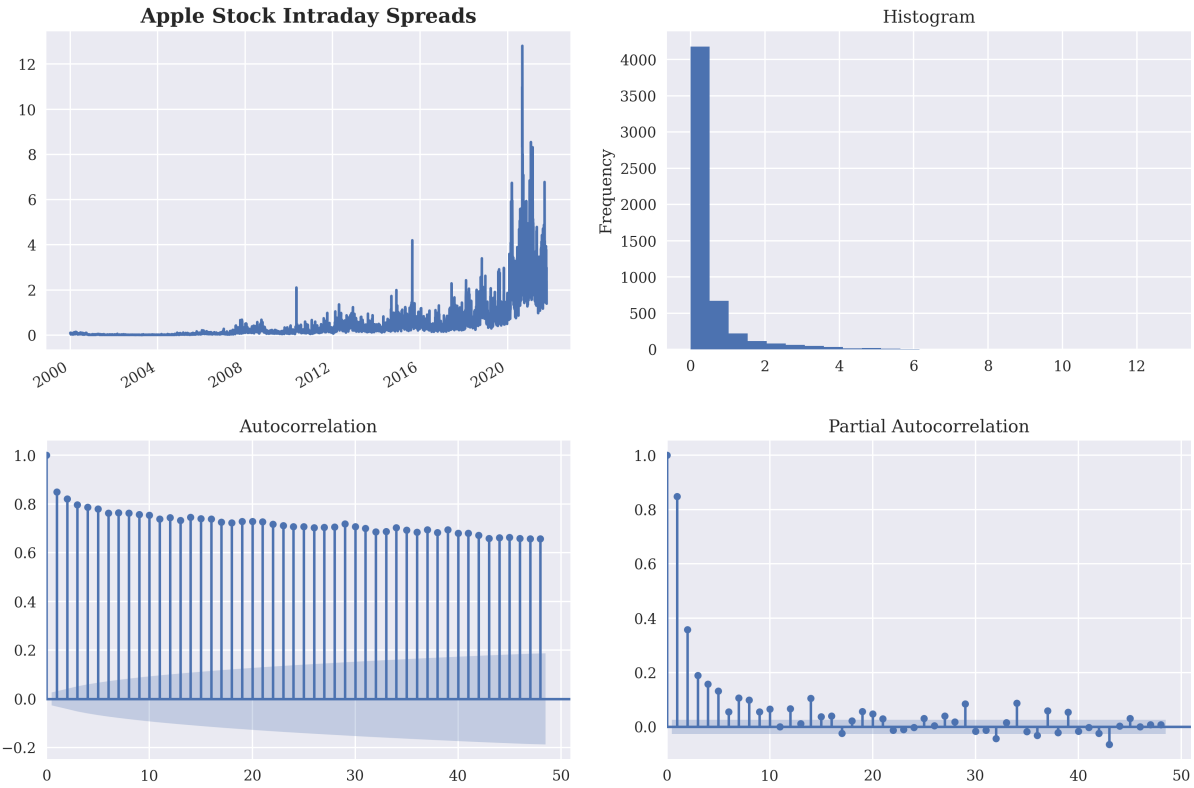


In [30]:

```
tsplot(sp500['AAPL_intraday_spread'].dropna(),title = "Apple Stock Intraday Spreads",lags = 48)
```

Out[30]:

```
(<AxesSubplot:title={'center':'Apple Stock Intraday Spreads'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

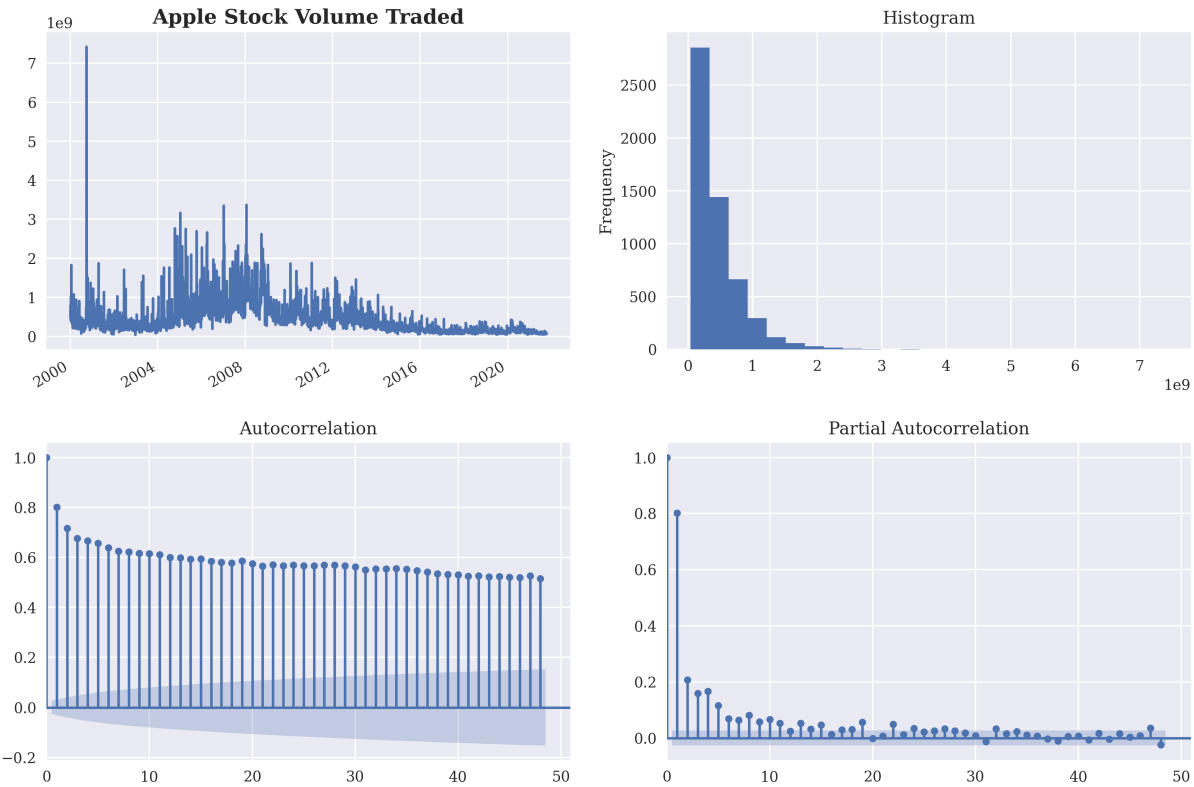


In [31]:

```
# Volume
tsplot(sp500['AAPL_volume'].dropna(),title = "Apple Stock Volume Traded",lags =
48)
```

Out[31]:

```
(<AxesSubplot:title={'center':'Apple Stock Volume Traded'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```



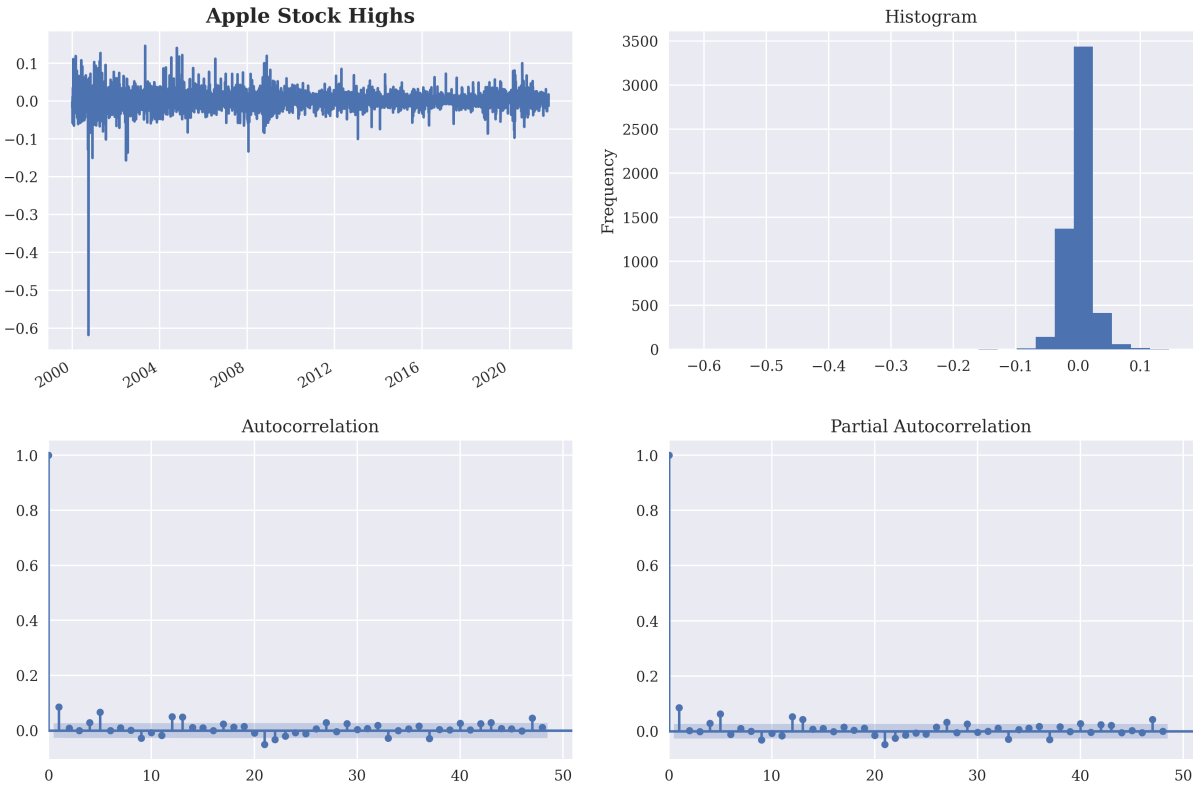
Making those Variables Stationary

In [32]:

```
tsplot(np.log(sp500['AAPL_high'].div(sp500['AAPL_high'].shift(1))).dropna(),titl
e = "Apple Stock Highs",lags = 48)
```

Out[32]:

```
(<AxesSubplot:title={'center':'Apple Stock Highs'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

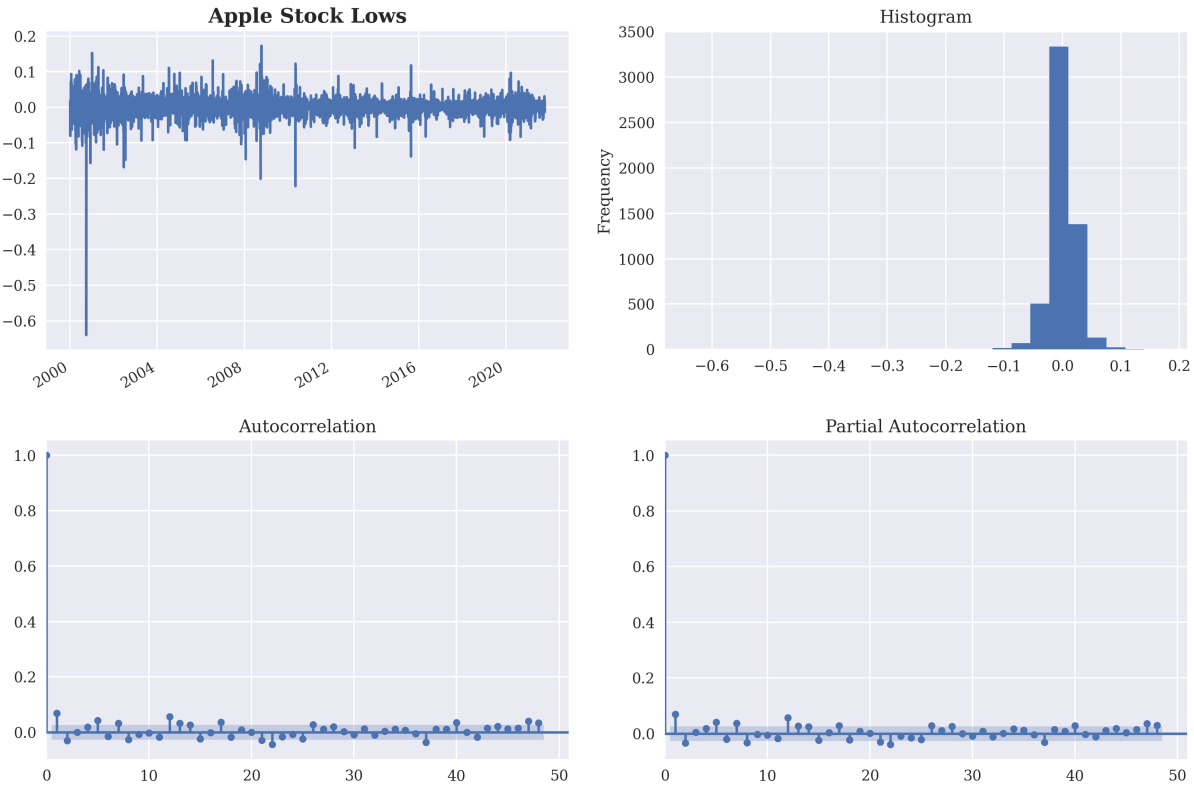


In [33]:

```
tsplot(np.log(sp500['AAPL_low'].div(sp500['AAPL_low'].shift(1))).dropna(),title
= "Apple Stock Lows",lags = 48)
```

Out[33]:

```
(<AxesSubplot:title={'center':'Apple Stock Lows'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

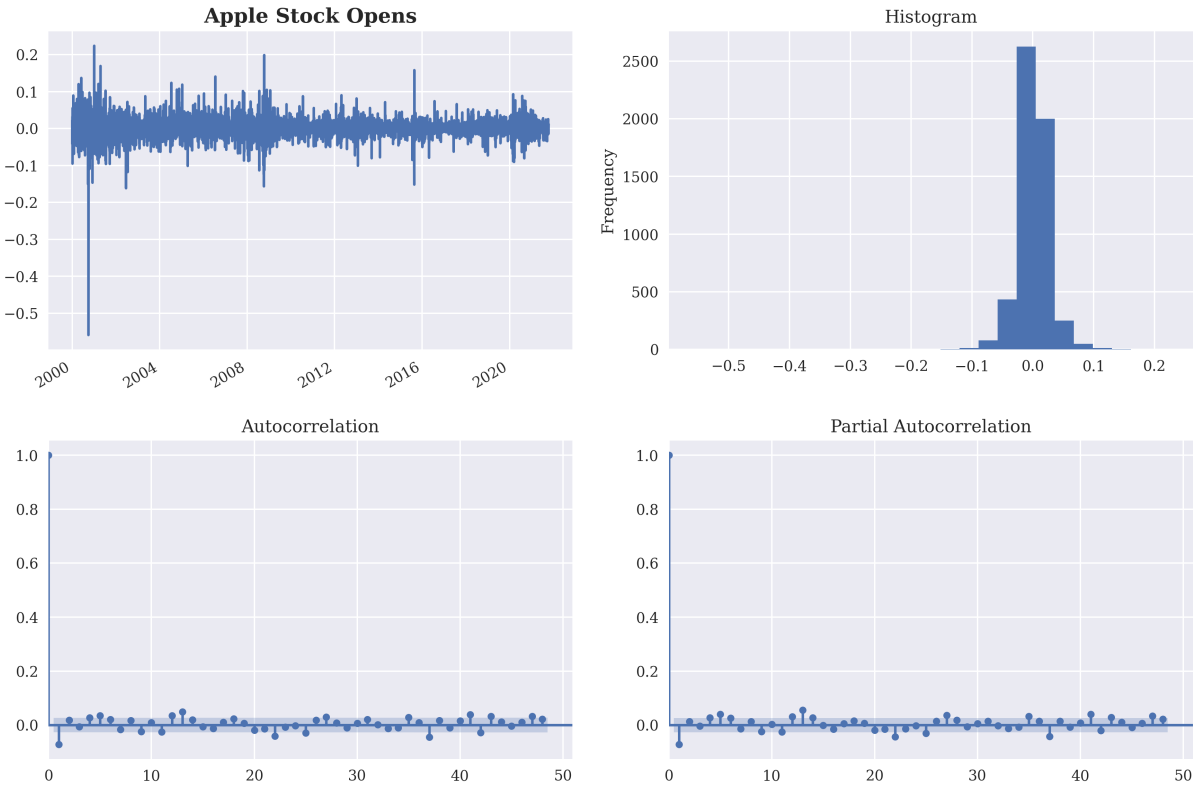


In [34]:

```
tsplot(np.log(sp500['AAPL_open'].div(sp500['AAPL_open'].shift(1))).dropna(),titl
e = "Apple Stock Opens",lags = 48)
```

Out[34]:

```
(<AxesSubplot:title={'center':'Apple Stock Opens'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

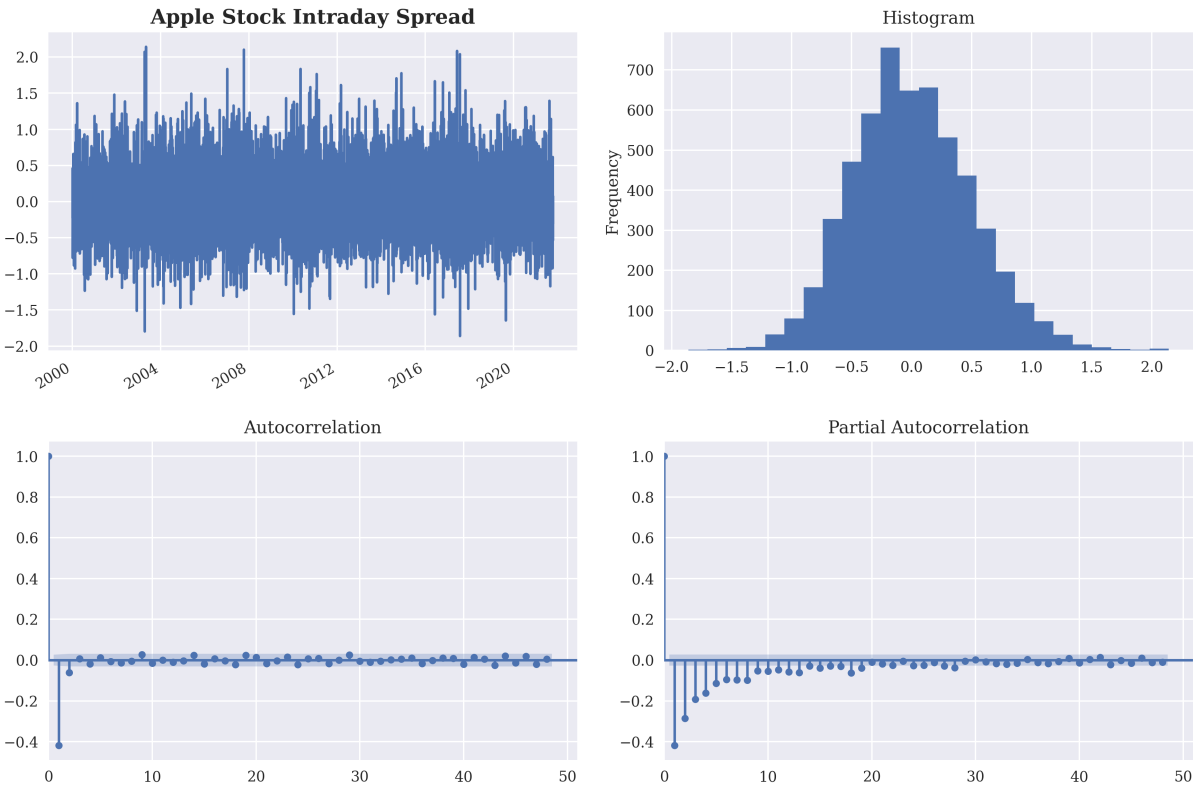


In [35]:

```
tsplot(np.log(sp500['AAPL_intraday_spread'].div(sp500['AAPL_intraday_spread'].sh
ift(1))).dropna(),
       title = "Apple Stock Intraday Spread",lags = 48)
```

Out[35]:

```
(<AxesSubplot:title={'center':'Apple Stock Intraday Spread'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```

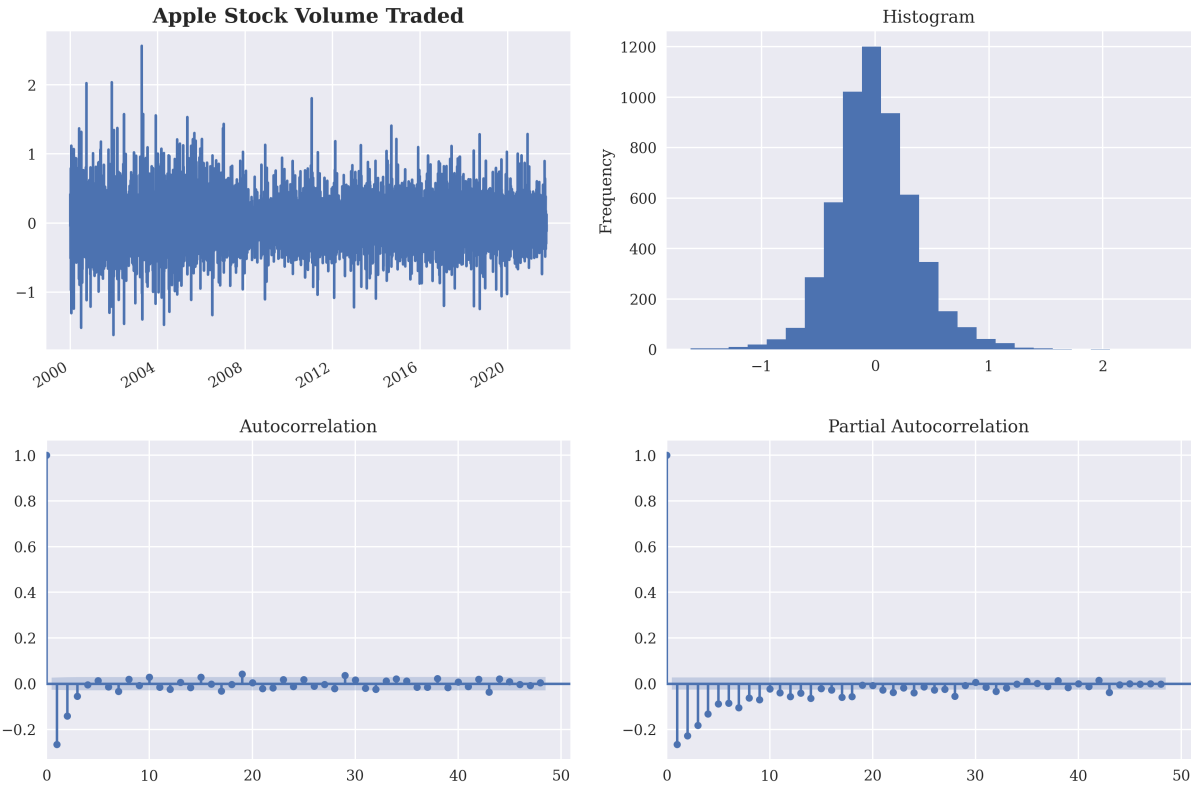


In [36]:

```
tsplot(np.log(sp500['AAPL_volume'].div(sp500['AAPL_volume'].shift(1))).dropna(),
title = "Apple Stock Volume Traded",lags = 48)
```

Out[36]:

```
(<AxesSubplot:title={'center':'Apple Stock Volume Traded'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```



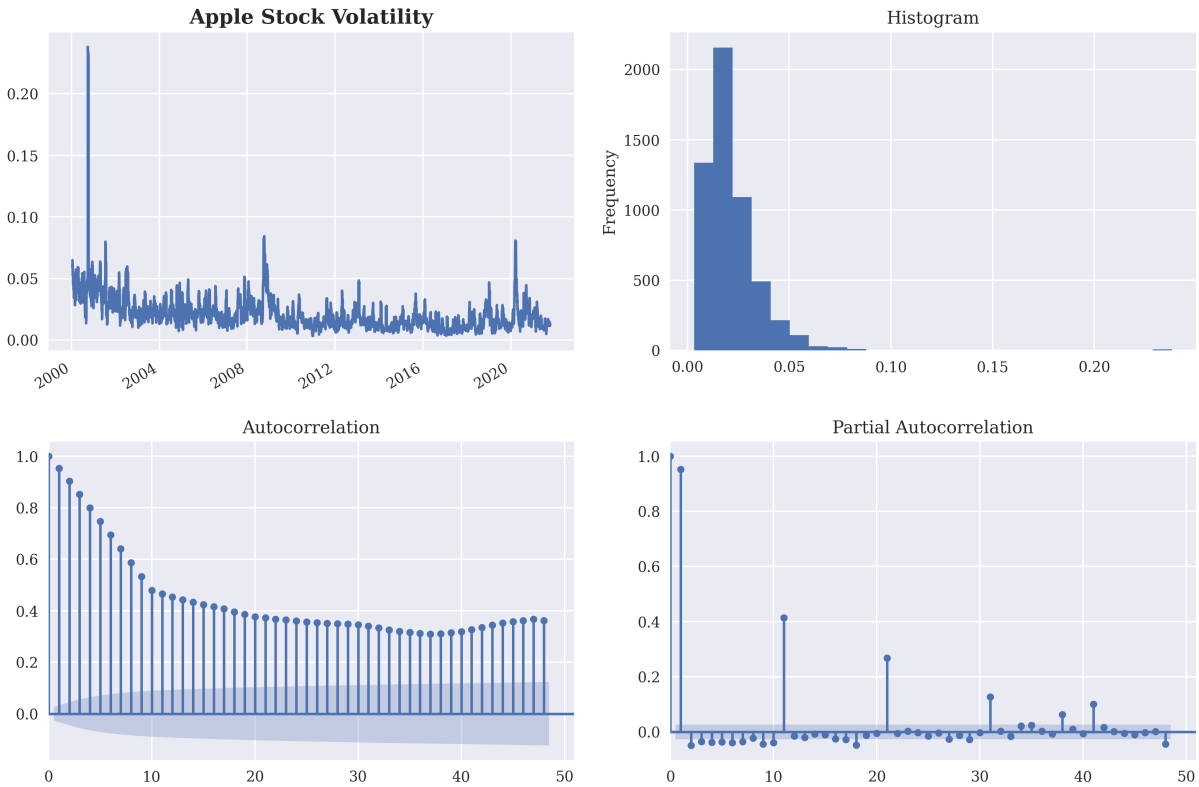
Another Variable to Consider: Stock returns 10 day rolling volatility

In [37]:

```
# looks like an autoregressive process
tsplot(sp500['AAPL_returns'].rolling(window = 10).std().dropna(),title = "Apple
 Stock Volatility",lags = 48)
```

Out[37]:

```
(<AxesSubplot:title={'center':'Apple Stock Volatility'}>,
 <AxesSubplot:title={'center':'Autocorrelation'}>,
 <AxesSubplot:title={'center':'Partial Autocorrelation'}>)
```



In [38]:

```
# add volatility columns for all the stocks
for col in sp500.columns:
    if 'returns' in col:
        sp500[f"{col.split('_')[0]}_volatility"] = sp500[col].rolling(window = 1
0).std()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  after removing the cwd from sys.path.
```

In [39]:

```
# add transformed stationary columns for all the stocks' price info
for col in sp500.columns:
    if 'open' in col or 'volume' in col or 'high' in col or 'low' in col or 'int
raday_spread' in col or 'volatility' in col:
        sp500[f"{col}_transformed"] = np.log(sp500[col].div(sp500[col].shift(1
)))
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  after removing the cwd from sys.path.
/opt/conda/lib/python3.7/site-packages/pandas/core/arraylike.py:364:
RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [40]:

```
sp500.shape
```

Out[40]:

```
(5483, 7070)
```

In [41]:

```
sp500.head(2)
```

Out[41]:

	MMM_high	MMM_low	MMM_open	MMM_volume	MMM_adj_close	ABT_high	AB
2000-01-03	48.25000	47.03125	48.03125	2173400.0	27.179523	16.160433	15
2000-01-04	47.40625	45.31250	46.43750	2713800.0	26.099533	15.599306	15

2 rows × 7070 columns

In [42]:

```
# create a separate data frame with only stationary variables
sp500_stationary = pd.DataFrame()
sp500_copy = sp500.copy(deep = True) #for good measure

for col in sp500_copy.columns:
    if 'returns' in col or 'transformed' in col:
        sp500_stationary[col] = sp500_copy[col]

# only non-stationary variable we'll keep for now is the
# adjusted close price of our target stock
# so that we can use it to create our target variable
sp500_stationary[f'{target_stock}_adj_close'] = sp500_copy[f'{target_stock}_adj_
close']

del(sp500_copy)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:7: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  import sys
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:12: Per
formanceWarning: DataFrame is highly fragmented.  This is usually th
e result of calling `frame.insert` many times, which has poor perfor
mance.  Consider joining all columns at once using pd.concat(axis=1)
instead.  To get a de-fragmented frame, use `newframe = frame.copy()
`
  if sys.path[0] == '':
```

In [43]:

```
sp500_stationary.shape
```

Out[43]:

```
(5483, 3536)
```

In [44]:

```
sp500_stationary.head(2)
```

Out[44]:

	MMM_returns	ABT_returns	ABBV_returns	ABMD_returns	ACN_returns	ATVI_returns
2000-01-03	NaN	NaN	NaN	NaN	NaN	NaN
2000-01-04	-0.040546	-0.028988	NaN	-0.024265	NaN	-0.030891

2 rows × 3536 columns

In [45]:

```
# we'll also create another data frame without any stationary variables
sp500_limited = pd.DataFrame()
sp500_copy = sp500.copy(deep = True) #for good measure
for col in sp500_copy.columns:
    if 'returns' not in col or 'transformed' not in col:
        sp500_limited[col] = sp500_copy[col]
del(sp500_copy)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:6: Perf
ormanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performa
nce.  Consider joining all columns at once using pd.concat(axis=1) i
nstead.  To get a de-fragmented frame, use `newframe = frame.copy()`
```

In [46]:

```
sp500_limited.shape
```

Out[46]:

```
(5483, 7070)
```

In [47]:

```
sp500_limited.head(2)
```

Out[47]:

	MMM_high	MMM_low	MMM_open	MMM_volume	MMM_adj_close	ABT_high	AE
2000-01-03	48.25000	47.03125	48.03125	2173400.0	27.179523	16.160433	15
2000-01-04	47.40625	45.31250	46.43750	2713800.0	26.099533	15.599306	15

2 rows × 7070 columns

Creating a Target Variable and Preparing our Data for the LSTM

Now that we've added some relevant variables to our data, we can begin working on preparing our target variable and our data for the LSTM network.

Our target variable is going to be comprised of buy, sell, and hold signals. We want to buy the stock at time t if there is an increase in the stock's future price of 7% or more; we want to sell the stock at time t if there is a decrease in the stock's future price of 5% or more. Thus we have a 2% margin.

Currently, our data is stored in a Pandas DataFrame object. However, LSTM's do not take Pandas Data Frames as input. Instead, they take arrays of data. Therefore, we have to transform our data into that format to run it through our LSTM network and generate predictions.

In [48]:

```
def classify_target_variable(current, future):
    # if price increases more than or equal to 2%, buy
    if (float(future) - float(current))/float(current) >= 0.02:
        return 1
    # if price drops more than or equal to 1%, sell
    if (float(future) - float(current))/float(current) <= -0.01:
        return -1
    # otherwise, hold
    else:
        return 0
```

In [49]:

```
def add_target_to_df(df,target_stock,periods_ahead):
    df['future'] = df[f'{target_stock}_adj_close'].shift(-periods_ahead)
    df['target'] = list(map(classify_target_variable, df[f'{target_stock}_adj_cl
ose'],
                            df['future']))
    df = df.drop(['future'],axis = 1)
    return df
```

In [50]:

```
from sklearn.preprocessing import StandardScaler

def scale_data(df):
    scaler = StandardScaler()
    df = df.copy(deep = True)
    index = df.index
    columns = df.columns
    target = df['target']
    df = np.asarray(df.drop(['target'],axis = 1))
    scaler = scaler.fit(df)
    df = df.reshape(df.shape[0],df.shape[1])
    df = scaler.transform(df)
    df = pd.DataFrame(df,index=index)
    df = pd.concat([df,target],axis = 1)
    df.columns = columns

    # clean up dataframe just in case
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df = df.ffill()
    df = df.dropna(axis = 0)

    return df
```

In [51]:

```
from sklearn.preprocessing import StandardScaler

def preprocess_data(df):
    """
    Drop columns with too many NA values;
    Scale the data (except for the target column);


    Parameters
    - df: a dataframe
    """
    # fill forward NA values
    df = df.ffill()

    # identify columns with too many NAs
    # drop them from the dataframe
    col_NAs = {col:df[col].isna().sum() for col in df.columns}
    bad_cols = []
    for col, num_NAs in col_NAs.items():
        if col_NAs[col] > 500:
            bad_cols.append(col)
    df = df.drop(bad_cols,axis = 1)

    # replace infinites with NAs
    df.replace([np.inf, -np.inf], np.nan, inplace=True)

    # fill forward again, just in case
    df = df.ffill()
    df = df.dropna(axis = 0) # drop all the remaining rows with NA values from th
e dataframe

    df = scale_data(df)

    return df
```

In [52]:

```
# number of periods into the future that we are trying to predict
periods_ahead = 2

sp500_stationary_new = add_target_to_df(sp500_stationary,target_stock,periods_ahead)
sp500_stationary_new.drop([f'{target_stock}_adj_close'],axis = 1,inplace = True)

sp500_new = add_target_to_df(sp500,target_stock,periods_ahead)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:4: PerformanceWarning: DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  after removing the cwd from sys.path.
```

In [53]:

```
# dataset with only stationary variables
sp500_stationary_clean = preprocess_data(sp500_stationary_new)

# dataset with all variables
sp500_clean = preprocess_data(sp500_new)
```

In [54]:

```
sp500_stationary_clean.head(2)
```

Out[54]:

	MMM_returns	ABT_returns	ABMD_returns	ACN_returns	ATVI_returns	ADBE_returns
2001-12-31	-1.028039	-0.919374	-2.045855	0.718140	-0.811653	-1.674932
2002-01-02	-0.659161	0.082910	0.245253	-1.477806	0.629768	1.068328

2 rows × 2745 columns

In [55]:

```
sp500_clean.head(2)
```

Out[55]:

	MMM_high	MMM_low	MMM_open	MMM_volume	MMM_adj_close	ABT_high	A
2001-12-31	-1.139356	-1.133326	-1.127862	0.066583	-1.118277	-0.585551	-
2002-01-02	-1.158054	-1.153431	-1.156022	0.517418	-1.124254	-0.593275	-

2 rows × 5489 columns

In [56]:

```
sp500_stationary_clean.shape
```

Out[56]:

```
(4984, 2745)
```

In [57]:

```
sp500_clean.shape
```

Out[57]:

```
(4984, 5489)
```

Create Different Data Sets to Test

- sp500_clean is only one of a few variations of data sets that were discussed earlier. In particular, this data set includes ALL variables
- for other data sets, we're going to remove some of the variables and see how that affects model performance

In [58]:

```
"""
This data set will have no high, low, or adjusted close data
because our intraday spread and returns variables are quite similar to them
thus, these variables may be redundant
"""
"""
sp500_clean_noHLC = sp500_clean.copy(deep = True)
for col in sp500_clean_noHLC.columns:
    if 'high' in col or 'low' in col or 'adj_close' in col:
        sp500_clean_noHLC.drop([col],axis = 1,inplace = True)

# We can apply the same logic to our stationary data set
sp500_stationary_clean_noHLC = sp500_stationary_clean.copy(deep = True)
for col in sp500_stationary_clean_noHLC.columns:
    if 'high' in col or 'low' in col or 'adj_close' in col:
        sp500_stationary_clean_noHLC.drop([col],axis = 1,inplace = True)
"""
```

Out[58]:

```
"\nsp500_clean_noHLC = sp500_clean.copy(deep = True)\nfor col in sp5
00_clean_noHLC.columns:\n    if 'high' in col or 'low' in col or 'ad
j_close' in col:\n        sp500_clean_noHLC.drop([col],axis = 1,inpl
ace = True)\n        \n# We can apply the same logic to our stationa
ry data set\nsp500_stationary_clean_noHLC = sp500_stationary_clean.c
opy(deep = True)\nfor col in sp500_stationary_clean_noHLC.columns:\n
if 'high' in col or 'low' in col or 'adj_close' in col:\n        sp5
00_stationary_clean_noHLC.drop([col],axis = 1,inplace = True)\n"
```

In [59]:

```
"""
print('Initial data set shape:',sp500_clean.shape)
print('New data set shape:',sp500_clean_noHLC.shape)
"""
```

Out[59]:

```
"\nprint('Initial data set shape:',sp500_clean.shape)\nprint('New da
ta set shape:',sp500_clean_noHLC.shape)\n"
```

In [60]:

```
"""
print('Initial data set shape:',sp500_stationary_clean.shape)
print('New data set shape:',sp500_stationary_clean_noHLC.shape)
"""
```

Out[60]:

```
"\nprint('Initial data set shape:',sp500_stationary_clean.shape)\npr
int('New data set shape:',sp500_stationary_clean_noHLC.shape)\n"
```

Save the Data Sets

In [61]:

```
"""
filename = "sp500_clean_noHLC.pickle"
with open(filename,'wb') as f:
    pickle.dump(sp500_clean_noHLC, f)

filename = "sp500_clean.pickle"
with open(filename,'wb') as f:
    pickle.dump(sp500_clean, f)

filename = "sp500_stationary_clean_noHLC.pickle"
with open(filename,'wb') as f:
    pickle.dump(sp500_stationary_clean_noHLC, f)

filename = "sp500_stationary_clean.pickle"
with open(filename,'wb') as f:
    pickle.dump(sp500_stationary_clean, f)
"""
```

Out[61]:

```
'\nfilename = "sp500_clean_noHLC.pickle"\nwith open(filename,\'wb\')
as f:\n    pickle.dump(sp500_clean_noHLC, f)\n    \nfilename = "sp50
0_clean.pickle"\nwith open(filename,\'wb\') as f:\n    pickle.dump(s
p500_clean, f)\n\nfilename = "sp500_stationary_clean_noHLC.pickle"\n
with open(filename,\'wb\') as f:\n    pickle.dump(sp500_stationary_c
lean_noHLC, f)\n    \nfilename = "sp500_stationary_clean.pickle"\nwi
th open(filename,\'wb\') as f:\n    pickle.dump(sp500_stationary_cle
an, f)\n'
```

Load the Data Sets

In [62]:

```
"""
filename = "sp500_clean.pickle"
with open(filename,'rb') as f:
    sp500_clean = pickle.load(f)

filename = "sp500_clean_noHLC.pickle"
with open(filename,'rb') as f:
    sp500_clean_noHLC = pickle.load(f)

filename = "sp500_stationary_clean.pickle"
with open(filename,'rb') as f:
    sp500_stationary_clean = pickle.load(f)

filename = "sp500_stationary_clean_noHLC.pickle"
with open(filename,'rb') as f:
    sp500_stationary_clean_noHLC = pickle.load(f)
"""
```

Out[62]:

```
'\nfilename = "sp500_clean.pickle"\nwith open(filename,\'rb\') as
f:\n    sp500_clean = pickle.load(f)\n\nfilename = "sp500_clean_noHL
C.pickle"\nwith open(filename,\'rb\') as f:\n    sp500_clean_noHLC =
pickle.load(f)\n    \nfilename = "sp500_stationary_clean.pickle"\nwi
th open(filename,\'rb\') as f:\n    sp500_stationary_clean = pickle.
load(f)\n\nfilename = "sp500_stationary_clean_noHLC.pickle"\nwith op
en(filename,\'rb\') as f:\n    sp500_stationary_clean_noHLC = pickl
e.load(f)\n'
```

In [63]:

```
#sp500_clean.head(2)
```

In [64]:

```
#sp500_clean_noHLC.head(2)
```

In [65]:

```
#sp500_stationary_clean.head(2)
```

In [66]:

```
#sp500_stationary_clean_noHLC.head(2)
```

Create Train-Test Split & Convert Data into a Readable form for the LSTM

In [67]:

```
def train_test(df):
    dates = sorted(df.index.values)
    split_date = (df.index[-int(len(dates)*0.2)])

    df_val = df[df.index >= split_date]
    df_train = df[df.index < split_date]

    return df_train, df_val
```

In [68]:

```
#sp500_train, sp500_val = train_test(sp500_clean)
#sp500_noHLC_train, sp500_noHLC_val = train_test(sp500_clean_noHLC)
#sp500_stationary_train, sp500_stationary_val = train_test(sp500_stationary_clean)
#sp500_stationary_noHLC_train, sp500_stationary_noHLC_val = train_test(sp500_stati
onary_clean_noHLC)
```

Convert Data into LSTM-Readable Format

When generating the training and testing data for the complete S&P 500 data, the kernel kept restarting, so I had to run part of the code below outside of the notebook. Still, the code works, and I uploaded the relevant data that it produces to train the LSTM network below.

In [69]:

```
# length of preceeding data used in generating predictions
def ts_to_supervised(df,sequence_length):

    """
    df: a data frame of time series data with stock price information
    sequence_length: the length of preceeding data used in generating predictions
    """

    # list containing sequences of data
    sequential_data = []

    # the sequences of data that we're going to gradually
    # add to our sequential_data list
    previous_days = deque(maxlen=sequence_length)

    # iterate over the values of each column
    for i in df.values:
        # store all values except for the target variable
        previous_days.append([n for n in i[:-1]])
        if len(previous_days) == sequence_length:
            sequential_data.append([np.array(previous_days), i[-1]])

    # we shuffle our list of sequences because
    # each sequence is used independently used to make predictions
    random.shuffle(sequential_data)

    buys = []  # list that will store our buy sequences and targets
    sells = []  # list that will store our sell sequences and targets
    # holds = [] # list that will store our hold sequences and targets

    # encode our target data as buy-hold-sell
    for seq, target in sequential_data:
        if target == 1:
            buys.append([seq, 1])
        elif target == -1:
            sells.append([seq, 0])
        #elif target == 0:
        #    holds.append([seq, 1])

    # shuffle again
    random.shuffle(buys)
    random.shuffle(sells)
    # random.shuffle(holds)
```

```
    lower = min(len(buys), len(sells)) # len(holds)

    # make sure both lists are only as long as the shortest length of the other
    # so that we do not introduce bias into our LSTM network
    buys = buys[:lower]
    sells = sells[:lower]
    #holds = holds[:lower]

    # add the buys and sells together
    sequential_data = buys+sells #+holds

    # another shuffle, so the model doesn't get confused with
    # having all one class then the other
    random.shuffle(sequential_data)

    X = []
    y = []

    for seq, target in sequential_data:  # going over our new sequential data
        X.append(seq)  # X is the sequences
        y.append(target)  # y is the targets/labels (buys vs sell/notbuy)

    return np.array(X), y
```

Full S&P 500 Data

In [70]:

```
"""
sequence_length = 50
X_train_sp500, y_train_sp500 = ts_to_supervised(sp500_train,sequence_length)
X_val_sp500, y_val_sp500 = ts_to_supervised(sp500_val,sequence_length)
print(f"TRAIN data: {len(X_train_sp500)} validation: {len(X_val_sp500)}")
print(f"TRAIN Sells: {y_train_sp500.count(0)}, buys: {y_train_sp500.count(1)}")
print(f"VALIDATION Sells: {y_val_sp500.count(0)}, buys: {y_train_sp500.count(1)}")
"""
```

Out[70]:

```
'\nsequence_length = 50\nX_train_sp500, y_train_sp500 = ts_to_superv
ised(sp500_train,sequence_length)\nX_val_sp500, y_val_sp500 = ts_to_
supervised(sp500_val,sequence_length)\nprint(f"TRAIN data: {len(X_tr
ain_sp500)} validation: {len(X_val_sp500)}")\nprint(f"TRAIN Sells:
{y_train_sp500.count(0)}, buys: {y_train_sp500.count(1)}")\nprint
(f"VALIDATION Sells: {y_val_sp500.count(0)}, buys: {y_train_sp500.co
unt(1)}")\n'
```

S&P 500 Data with no High, Low, Close Information

In [71]:

```
"""
sequence_length = 50
X_train_noHLC, y_train_noHLC = ts_to_supervised(sp500_noHLC_train,sequence_length)
X_val_noHLC, y_val_noHLC = ts_to_supervised(sp500_noHLC_val,sequence_length)
print(f"TRAIN data: {len(X_train_noHLC)} validation: {len(X_val_noHLC)}")
print(f"TRAIN Sells: {y_train_noHLC.count(0)}, buys: {y_train_noHLC.count(1)}")
print(f"VALIDATION Sells: {y_val_noHLC.count(0)}, buys: {y_val_noHLC.count(1)}")
"""
```

Out[71]:

```
'\nsequence_length = 50\nX_train_noHLC, y_train_noHLC = ts_to_superv
ised(sp500_noHLC_train,sequence_length) \nX_val_noHLC, y_val_noHLC =
ts_to_supervised(sp500_noHLC_val,sequence_length) \nprint(f"TRAIN da
ta: {len(X_train_noHLC)} validation: {len(X_val_noHLC)}")\nprint(f"T
RAIN Sells: {y_train_noHLC.count(0)}, buys: {y_train_noHLC.count
(1)}")\nprint(f"VALIDATION Sells: {y_val_noHLC.count(0)}, buys: {y_v
al_noHLC.count(1)}")\n'
```

S&P 500 Stationary Data

In [72]:

```
"""
sequence_length = 50
X_train_stationary, y_train_stationary = ts_to_supervised(sp500_stationary_train,s
equence_length)
X_val_stationary, y_val_stationary = ts_to_supervised(sp500_stationary_val,sequenc
e_length)
print(f"TRAIN data: {len(X_train_stationary)} validation: {len(X_val_stationar
y)}")
print(f"TRAIN sells: {y_train_stationary.count(0)}, buys: {y_train_stationary.coun
t(1)}")
print(f"VALIDATION Sells: {y_val_stationary.count(0)}, buys: {y_val_stationary.cou
nt(1)}")
"""
```

Out[72]:

```
'\nsequence_length = 50\nX_train_stationary, y_train_stationary = ts
_to_supervised(sp500_stationary_train,sequence_length) \nX_val_stati
onary, y_val_stationary = ts_to_supervised(sp500_stationary_val,sequ
ence_length) \nprint(f"TRAIN data: {len(X_train_stationary)} validat
ion: {len(X_val_stationary)}")\nprint(f"TRAIN sells: {y_train_statio
nary.count(0)}, buys: {y_train_stationary.count(1)}")\nprint(f"VALID
ATION Sells: {y_val_stationary.count(0)}, buys: {y_val_stationary.co
unt(1)}")\n'
```

S&P 500 Stationary Data with no High, Low, Close Information

In [73]:

```
"""
X_train_stationary_noHLC, y_train_stationary_noHLC = ts_to_supervised(sp500_stationary_noHLC_train,sequence_length)
X_val_stationary_noHLC, y_val_stationary_noHLC = ts_to_supervised(sp500_stationary_noHLC_val,sequence_length)
print(f"TRAIN data: {len(X_train_stationary_noHLC)} validation: {len(X_val_stationary_noHLC)}")
print(f"TRAIN sells: {y_train_stationary_noHLC.count(0)}, buys: {y_train_stationary_noHLC.count(1)}")
print(f"VALIDATION Sells: {y_val_stationary_noHLC.count(0)}, buys: {y_val_stationary_noHLC.count(1)}")
"""
```

Out[73]:

```
'\nX_train_stationary_noHLC, y_train_stationary_noHLC = ts_to_supervised(sp500_stationary_noHLC_train,sequence_length)\nX_val_stationary_noHLC, y_val_stationary_noHLC = ts_to_supervised(sp500_stationary_noHLC_val,sequence_length)\nprint(f"TRAIN data: {len(X_train_stationary_noHLC)} validation: {len(X_val_stationary_noHLC)}")\nprint(f"TRAIN sells: {y_train_stationary_noHLC.count(0)}, buys: {y_train_stationary_noHLC.count(1)}")\nprint(f"VALIDATION Sells: {y_val_stationary_noHLC.count(0)}, buys: {y_val_stationary_noHLC.count(1)}")\n'
```

Save Train and Test Sets

In [74]:

```
"""
# full data set
with open('X_train_sp500.pickle','wb') as f:
    pickle.dump(X_train_sp500,f)
with open('y_train_sp500.pickle','wb') as f:
      pickle.dump(y_train_sp500,f)
with open('X_val_sp500.pickle','wb') as f:
      pickle.dump(X_val_sp500,f)
with open('y_val_sp500.pickle','wb') as f:
      pickle.dump(y_val_sp500,f)

# no high, low, close
with open('X_train_noHLC.pickle','wb') as f:
    pickle.dump(X_train_noHLC,f)
with open('y_train_noHLC.pickle','wb') as f:
      pickle.dump(y_train_noHLC,f)
with open('X_val_noHLC.pickle','wb') as f:
      pickle.dump(X_val_noHLC,f)
with open('y_val_noHLC.pickle','wb') as f:
      pickle.dump(y_val_noHLC,f)

# stationary data
with open('X_train_stationary.pickle','wb') as f:
      pickle.dump(X_train_stationary,f)
with open('y_train_stationary.pickle','wb') as f:
      pickle.dump(y_train_stationary,f)
with open('X_val_stationary.pickle','wb') as f:
      pickle.dump(X_val_stationary,f)
with open('y_val_stationary.pickle','wb') as f:
      pickle.dump(y_val_stationary,f)

# stationary data with no high, low, close
with open('X_train_stationary_noHLC.pickle','wb') as f:
      pickle.dump(X_train_stationary_noHLC,f)
with open('y_train_stationary_noHLC.pickle','wb') as f:
      pickle.dump(y_train_stationary_noHLC,f)
with open('X_val_stationary_noHLC.pickle','wb') as f:
      pickle.dump(X_val_stationary_noHLC,f)
with open('y_val_stationary_noHLC.pickle','wb') as f:
      pickle.dump(y_val_stationary_noHLC,f)
"""
```

Out[74]:

```
"\n# full data set\nwith open('X_train_sp500.pickle','wb') as f:\n
pickle.dump(X_train_sp500,f)  \nwith open('y_train_sp500.pickle','w
b') as f:\n      pickle.dump(y_train_sp500,f)\nwith open('X_val_sp50
0.pickle','wb') as f:\n      pickle.dump(X_val_sp500,f)  \nwith open
('y_val_sp500.pickle','wb') as f:\n      pickle.dump(y_val_sp500,f)
\n\n# no high, low, close     \nwith open('X_train_noHLC.pickle','w
b') as f:\n    pickle.dump(X_train_noHLC,f)  \nwith open('y_train_no
HLC.pickle','wb') as f:\n      pickle.dump(y_train_noHLC,f)\nwith op
en('X_val_noHLC.pickle','wb') as f:\n      pickle.dump(X_val_noHLC,
f)  \nwith open('y_val_noHLC.pickle','wb') as f:\n      pickle.dump
(y_val_noHLC,f)\n\n# stationary data\nwith open('X_train_stationary.
pickle','wb') as f:\n      pickle.dump(X_train_stationary,f)\nwith o
pen('y_train_stationary.pickle','wb') as f:\n      pickle.dump(y_tra
in_stationary,f)\nwith open('X_val_stationary.pickle','wb') as f:\n
pickle.dump(X_val_stationary,f)\nwith open('y_val_stationary.pickl
e','wb') as f:\n      pickle.dump(y_val_stationary,f)\n\n# stationar
y data with no high, low, close\nwith open('X_train_stationary_noHL
C.pickle','wb') as f:\n      pickle.dump(X_train_stationary_noHLC,f)
\nwith open('y_train_stationary_noHLC.pickle','wb') as f:\n      pic
kle.dump(y_train_stationary_noHLC,f)\nwith open('X_val_stationary_no
HLC.pickle','wb') as f:\n      pickle.dump(X_val_stationary_noHLC,f)
\nwith open('y_val_stationary_noHLC.pickle','wb') as f:\n      pickl
e.dump(y_val_stationary_noHLC,f)\n"
```

Load Train-Test Data

Full S&P 500 Train-Test Data

This data set was too large for my computer to handle

In [75]:

```
#with open('../input/train-test-data-sp500/X_train_sp500.pickle','rb') as f:
#    X_train_sp500 = pickle.load(f)
#with open('../input/train-test-data-sp500/y_train_sp500.pickle','rb') as f:
#    y_train_sp500 = pickle.load(f)
```

In [76]:

```
#with open('../input/train-test-data-sp500/X_val_sp500.pickle','rb') as f:
#    X_val_sp500 = pickle.load(f)
#with open('../input/train-test-data-sp500/y_val_sp500.pickle','rb') as f:
#    y_val_sp500 = pickle.load(f)
```

S&P 500 Data with no High, Low, Close Information

This data set was too large for my computer to handle

In [77]:

```
#with open('../input/train-test-data-sp500/X_train_noHLC.pickle','rb') as f:
#    X_train_noHLC = pickle.load(f)
#with open('../input/train-test-data-sp500/y_train_noHLC.pickle','rb') as f:
#    y_train_noHLC = pickle.load(f)
```

In [78]:

```
#with open('../input/train-test-data-sp500/X_val_noHLC.pickle','rb') as f:
#    X_val_noHLC = pickle.load(f)
#with open('../input/train-test-data-sp500/y_val_noHLC.pickle','rb') as f:
#    y_val_noHLC = pickle.load(f)
```

S&P 500 Stationary Data

In [79]:

```
with open('../input/train-test-data-sp500/X_train_stationary.pickle','rb') as f:
    X_train_stationary = pickle.load(f)
with open('../input/train-test-data-sp500/y_train_stationary.pickle','rb') as f:
    y_train_stationary = pickle.load(f)
```

In [80]:

```
with open('../input/train-test-data-sp500/X_val_stationary.pickle','rb') as f:
    X_val_stationary = pickle.load(f)
with open('../input/train-test-data-sp500/y_val_stationary.pickle','rb') as f:
    y_val_stationary = pickle.load(f)
```

S&P 500 Stationary Data with no High, Low, Close Information

In [81]:

```
with open('../input/train-test-data-sp500/X_train_stationary_noHLC.pickle','rb')
as f:
    X_train_stationary_noHLC = pickle.load(f)
with open('../input/train-test-data-sp500/y_train_stationary_noHLC.pickle','rb')
as f:
    y_train_stationary_noHLC = pickle.load(f)
```

In [82]:

```
with open('../input/train-test-data-sp500/X_val_stationary_noHLC.pickle','rb') a
s f:
    X_val_stationary_noHLC = pickle.load(f)
with open('../input/train-test-data-sp500/y_val_stationary_noHLC.pickle','rb') a
s f:
    y_val_stationary_noHLC = pickle.load(f)
```

Building the LSTM network

In [83]:

```
def build_model(X_train):
    model = Sequential()
    model.add(CuDNNLSTM(2048, input_shape=(X_train.shape[1:]), return_sequences=
True))
    model.add(Dropout(0.2))
    # Normalise activation outputs
    model.add(BatchNormalization())

    model.add(CuDNNLSTM(2048, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(CuDNNLSTM(1024))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(3, activation='softmax'))

    # model optimiser settings
    opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay=1e-6)

    # Compile model
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=opt,
        metrics=['accuracy']
    )
    return model
```

In [84]:

```
# model_sp500 = build_model(X_train_sp500)
```

In [85]:

```
#model_noHLC = build_model(X_train_noHLC)
```

In [86]:

```
model_stationary = build_model(X_train_stationary)
```

```
2021-10-22 05:46:52.186970: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:52.281014: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:52.281736: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:52.283891: I tensorflow/core/platform/cpu_feature_g
uard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Ne
ural Network Library (oneDNN) to use the following CPU instructions
in performance-critical operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appr
opriate compiler flags.
2021-10-22 05:46:52.284347: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:52.285428: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:52.286358: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:54.108348: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:54.109439: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:54.110251: I tensorflow/stream_executor/cuda/cuda_g
pu_executor.cc:937] successful NUMA node read from SysFS had negativ
e value (-1), but there must be at least one NUMA node, so returning
NUMA node zero
2021-10-22 05:46:54.111849: I tensorflow/core/common_runtime/gpu/gpu
_device.cc:1510] Created device /job:localhost/replica:0/task:0/devi
```

```
ce:GPU:0 with 15403 MB memory:  -> device: 0, name: Tesla P100-PCIE-
16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
2021-10-22 05:46:54.511168: W tensorflow/core/framework/cpu_allocato
r_impl.cc:80] Allocation of 67108864 exceeds 10% of free system memo
ry.
2021-10-22 05:46:54.567369: W tensorflow/core/framework/cpu_allocato
r_impl.cc:80] Allocation of 67108864 exceeds 10% of free system memo
ry.
2021-10-22 05:47:02.682228: W tensorflow/core/framework/cpu_allocato
r_impl.cc:80] Allocation of 67108864 exceeds 10% of free system memo
ry.
2021-10-22 05:47:02.732823: W tensorflow/core/framework/cpu_allocato
r_impl.cc:80] Allocation of 67108864 exceeds 10% of free system memo
ry.
```

In [87]:

```
model_stationary_noHLC = build_model(X_train_stationary_noHLC)
```

```
2021-10-22 05:47:12.872452: W tensorflow/core/framework/cpu_allocato
r_impl.cc:80] Allocation of 67108864 exceeds 10% of free system memo
ry.
```

Save Models

In [88]:

```
"""
with open('model_sp500.pickle','wb') as f:
    pickle.dump(model_sp500,f)
with open('model_noHLC.pickle','wb') as f:
      pickle.dump(model_noHLC,f)
with open('model_stationary.pickle','wb') as f:
      pickle.dump(model_stationary,f)
with open('model_stationary_noHLC.pickle','wb') as f:
      pickle.dump(model_stationary_noHLC,f)
"""
```

Out[88]:

```
"\nwith open('model_sp500.pickle','wb') as f:\n    pickle.dump(model
_sp500,f)  \nwith open('model_noHLC.pickle','wb') as f:\n      pickl
e.dump(model_noHLC,f)\nwith open('model_stationary.pickle','wb') as
f:\n      pickle.dump(model_stationary,f)  \nwith open('model_statio
nary_noHLC.pickle','wb') as f:\n      pickle.dump(model_stationary_n
oHLC,f)\n"
```

Load Models

In [89]:

```
"""
with open('../output/kaggle/working/model_sp500.pickle','rb') as f:
    model_sp500 = pickle.load(f)
with open('../output/kaggle/working/model_noHLC.pickle','rb') as f:
    model_noHLC = pickle.load(f)
with open('../output/kaggle/working/model_stationary.pickle','rb') as f:
    model_stationary = pickle.load(f)
with open('../output/kaggle/working/model_stationary_noHLC.pickle','rb') as f:
    model_stationary_noHLC = pickle.load(f)
"""
```

Out[89]:

```
"\nwith open('../output/kaggle/working/model_sp500.pickle','rb') as
f:\n    model_sp500 = pickle.load(f)\nwith open('../output/kaggle/wo
rking/model_noHLC.pickle','rb') as f:\n    model_noHLC = pickle.load
(f)\nwith open('../output/kaggle/working/model_stationary.pickle','r
b') as f:\n    model_stationary = pickle.load(f)\nwith open('../outp
ut/kaggle/working/model_stationary_noHLC.pickle','rb') as f:\n    mo
del_stationary_noHLC = pickle.load(f)\n"
```

In [90]:

```
# Train model
def train_model(model,X_train,y_train,X_val,y_val,batch_size,epochs):
    history = model.fit(
        X_train, np.array(y_train),
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(X_val, np.array(y_val))
)
```

In [91]:

```
#epochs = 25
#batch_size = 32
#train_model(model_sp500,X_train_sp500,y_train_sp500,X_val_sp500,y_val_sp500,
#            batch_size,epochs)
```

In [92]:

```
#epochs = 20
#batch_size = 64
#train_model(model_noHLC,X_train_noHLC,y_train_noHLC,X_val_noHLC,y_val_noHLC,
#             batch_size,epochs)
```

In [93]:

```
epochs = 20
batch_size = 48
train_model(model_stationary,X_train_stationary,y_train_stationary,
            X_val_stationary,y_val_stationary,
            batch_size,epochs)
```

```
2021-10-22 05:47:37.820659: I tensorflow/compiler/mlir/mlir_graph_op
timization_pass.cc:185] None of the MLIR Optimization Passes are ena
bled (registered 2)


Epoch 1/20


2021-10-22 05:47:41.509244: I tensorflow/stream_executor/cuda/cuda_d
nn.cc:369] Loaded cuDNN version 8005
```

```
40/40 [==============================] - 17s 284ms/step - loss: 0.84
12 - accuracy: 0.5159 - val_loss: 0.7301 - val_accuracy: 0.4702
Epoch 2/20
40/40 [==============================] - 10s 250ms/step - loss: 0.74
43 - accuracy: 0.5122 - val_loss: 0.7209 - val_accuracy: 0.4541
Epoch 3/20
40/40 [==============================] - 10s 252ms/step - loss: 0.71
01 - accuracy: 0.5181 - val_loss: 0.7170 - val_accuracy: 0.4679
Epoch 4/20
40/40 [==============================] - 10s 249ms/step - loss: 0.71
35 - accuracy: 0.5128 - val_loss: 0.7023 - val_accuracy: 0.4885
Epoch 5/20
40/40 [==============================] - 10s 251ms/step - loss: 0.68
11 - accuracy: 0.5882 - val_loss: 0.7069 - val_accuracy: 0.4954
Epoch 6/20
40/40 [==============================] - 10s 251ms/step - loss: 0.64
77 - accuracy: 0.6265 - val_loss: 0.7853 - val_accuracy: 0.4977
Epoch 7/20
40/40 [==============================] - 10s 251ms/step - loss: 0.57
18 - accuracy: 0.7131 - val_loss: 0.7854 - val_accuracy: 0.5115
Epoch 8/20
40/40 [==============================] - 10s 250ms/step - loss: 0.45
89 - accuracy: 0.7912 - val_loss: 0.8270 - val_accuracy: 0.5161
Epoch 9/20
40/40 [==============================] - 10s 252ms/step - loss: 0.35
83 - accuracy: 0.8475 - val_loss: 1.0964 - val_accuracy: 0.5344
Epoch 10/20
40/40 [==============================] - 10s 251ms/step - loss: 0.18
85 - accuracy: 0.9341 - val_loss: 1.2242 - val_accuracy: 0.5298
Epoch 11/20
40/40 [==============================] - 10s 251ms/step - loss: 0.20
52 - accuracy: 0.9240 - val_loss: 1.2770 - val_accuracy: 0.4954
Epoch 12/20
40/40 [==============================] - 10s 252ms/step - loss: 0.18
51 - accuracy: 0.9293 - val_loss: 1.7641 - val_accuracy: 0.4862
Epoch 13/20
40/40 [==============================] - 10s 252ms/step - loss: 0.16
70 - accuracy: 0.9405 - val_loss: 1.1454 - val_accuracy: 0.5298
Epoch 14/20
40/40 [==============================] - 10s 251ms/step - loss: 0.06
93 - accuracy: 0.9787 - val_loss: 2.0042 - val_accuracy: 0.5482
Epoch 15/20
40/40 [==============================] - 10s 251ms/step - loss: 0.08
13 - accuracy: 0.9756 - val_loss: 1.8721 - val_accuracy: 0.5482
```

```
Epoch 16/20
40/40 [==============================] - 10s 250ms/step - loss: 0.08
12 - accuracy: 0.9745 - val_loss: 1.3471 - val_accuracy: 0.5528
Epoch 17/20
40/40 [==============================] - 10s 253ms/step - loss: 0.07
38 - accuracy: 0.9729 - val_loss: 2.1790 - val_accuracy: 0.5482
Epoch 18/20
40/40 [==============================] - 10s 251ms/step - loss: 0.03
70 - accuracy: 0.9862 - val_loss: 2.5086 - val_accuracy: 0.5252
Epoch 19/20
40/40 [==============================] - 10s 250ms/step - loss: 0.03
72 - accuracy: 0.9899 - val_loss: 2.8060 - val_accuracy: 0.5390
Epoch 20/20
40/40 [==============================] - 10s 251ms/step - loss: 0.02
32 - accuracy: 0.9915 - val_loss: 2.1733 - val_accuracy: 0.5573
```

In [94]:

```
epochs = 20
batch_size = 64
train_model(model_stationary_noHLC,X_train_stationary_noHLC,y_train_stationary_n
oHLC,
            X_val_stationary_noHLC,y_val_stationary_noHLC,
            batch_size,epochs)
```

```
Epoch 1/20
30/30 [==============================] - 12s 312ms/step - loss: 0.88
63 - accuracy: 0.5090 - val_loss: 0.7606 - val_accuracy: 0.4908
Epoch 2/20
30/30 [==============================] - 8s 262ms/step - loss: 0.775
5 - accuracy: 0.5170 - val_loss: 0.7279 - val_accuracy: 0.4817
Epoch 3/20
30/30 [==============================] - 8s 263ms/step - loss: 0.754
9 - accuracy: 0.5074 - val_loss: 0.6943 - val_accuracy: 0.5252
Epoch 4/20
30/30 [==============================] - 8s 263ms/step - loss: 0.738
4 - accuracy: 0.4910 - val_loss: 0.6959 - val_accuracy: 0.4977
Epoch 5/20
30/30 [==============================] - 8s 262ms/step - loss: 0.729
4 - accuracy: 0.4910 - val_loss: 0.6912 - val_accuracy: 0.5275
Epoch 6/20
30/30 [==============================] - 8s 262ms/step - loss: 0.725
4 - accuracy: 0.4952 - val_loss: 0.7139 - val_accuracy: 0.4977
Epoch 7/20
30/30 [==============================] - 8s 262ms/step - loss: 0.705
3 - accuracy: 0.5308 - val_loss: 0.6948 - val_accuracy: 0.5046
Epoch 8/20
30/30 [==============================] - 8s 263ms/step - loss: 0.707
3 - accuracy: 0.5069 - val_loss: 0.6966 - val_accuracy: 0.5000
Epoch 9/20
30/30 [==============================] - 8s 263ms/step - loss: 0.695
0 - accuracy: 0.5250 - val_loss: 0.7087 - val_accuracy: 0.5000
Epoch 10/20
30/30 [==============================] - 8s 262ms/step - loss: 0.699
3 - accuracy: 0.4936 - val_loss: 0.7026 - val_accuracy: 0.5000
Epoch 11/20
30/30 [==============================] - 8s 265ms/step - loss: 0.691
4 - accuracy: 0.5170 - val_loss: 0.6944 - val_accuracy: 0.5000
Epoch 12/20
30/30 [==============================] - 8s 263ms/step - loss: 0.698
9 - accuracy: 0.4878 - val_loss: 0.6973 - val_accuracy: 0.4587
Epoch 13/20
30/30 [==============================] - 8s 263ms/step - loss: 0.699
8 - accuracy: 0.5165 - val_loss: 0.7036 - val_accuracy: 0.5000
Epoch 14/20
30/30 [==============================] - 8s 262ms/step - loss: 0.694
1 - accuracy: 0.5191 - val_loss: 0.6987 - val_accuracy: 0.5000
Epoch 15/20
30/30 [==============================] - 8s 269ms/step - loss: 0.683
```

```
6 - accuracy: 0.5675 - val_loss: 0.7226 - val_accuracy: 0.5023
Epoch 16/20
30/30 [==============================] - 8s 263ms/step - loss: 0.651
6 - accuracy: 0.6440 - val_loss: 0.8035 - val_accuracy: 0.5092
Epoch 17/20
30/30 [==============================] - 8s 261ms/step - loss: 0.533
0 - accuracy: 0.7460 - val_loss: 0.7704 - val_accuracy: 0.5069
Epoch 18/20
30/30 [==============================] - 8s 263ms/step - loss: 0.481
3 - accuracy: 0.7790 - val_loss: 0.7672 - val_accuracy: 0.4977
Epoch 19/20
30/30 [==============================] - 8s 263ms/step - loss: 0.303
8 - accuracy: 0.8810 - val_loss: 1.0124 - val_accuracy: 0.5252
Epoch 20/20
30/30 [==============================] - 8s 264ms/step - loss: 0.257
8 - accuracy: 0.8985 - val_loss: 1.0366 - val_accuracy: 0.4885
```

Score Model Performance

In [95]:

```
# Score model
def score_model(model,X_val,y_val):
    score = model.evaluate(X_val, np.asarray(y_val), verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
```

In [96]:

```
#score_model(model_noHLC,X_val_noHLC,y_val_noHLC)
```

In [97]:

```
score_model(model_stationary,X_val_stationary,y_val_stationary)
```

```
Test loss: 2.1733367443084717
Test accuracy: 0.5573394298553467
```

In [98]:

```
score_model(model_stationary_noHLC,X_val_stationary_noHLC,y_val_stationary_noHLC
)
```

```
Test loss: 1.0365926027297974
Test accuracy: 0.48853209614753723
```

Conclusion and Future Study

From the results, we see that both data sets scored pretty similarly and that they both produced buy and sell predictions only slightly better than a coin toss. This suggests that the data used to predict the target could be filled with a lot of noise that the LSTM is using to make its predictions. Alternatively, this observation may suggest that more data (larger "sequence_length"s) may be needed for the LSTM to pick up reliable patterns.

In the future, dimension reduction could reduce the amount of noise that the LSTM picks up. To perform dimensionality reduction, I would first run a logistic regression of all the stationary variables in the data set onto the target variable and then pick out the statisitically significant variables to move forward with in training and testing. An alternative approach would be to use the Boruta Algorithm, although I suspect it would take much longer and not function on my computer.

Going in the opposite direction, instead of reducing the number of variable inputs, it may be worthwhile to test the LSTM on the larger data sets that my computer couldn't handle to see if the LSTM picks up on other important patterns.

In addition to altering the number of input variables, it might be worthwhile to alter the prediction period. Perhaps the model would show more definite results if it aimed to predict a stock's price changes 1 period into the future instead of 2 periods into the future (i.e. if periods_ahead = 1, not 2).

Finally, moving forward, it may be worthwhile looking into how this model can be applied toward the volatile markets of Cryptocurrencies.

However, as it stands, the data in this notebook is quite versatile, and we can use this notebook to build out models that generate buy/sell predictions on the price of any single S&P 500 equity based on the movement of the other 504 equities in the S&P 500.