My Stock Forecasting Project

The Problem:

```
Picking good stocks is the problem. I intend to build an app that'll pick
the best stock from a list of 4 that the    user will input.
Retail investing as grown by leaps and bounds over the past few years; la
regly due to stock trading apps like Robinhood and the recent Wall Street
Bets/Gamestop drama. More and more average folks are getting into investi
ng looking to make a quick buck.
But stock research is HARD. Wherever you look, there are just as many voi
ces saying a stock is a buy as there are telling you it's a dud.
So the idea here is to use some simple time series forecasting to create
 a quick and easy way to decide how to throw away some money.
```

Getting started

Step 1: Import libraries

```
Since this is the big project, we'll be importing everything. And I mean
 everything.
```

In [1]:
```
import pandas as pd
import pandas.tseries
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import statsmodels.api as sm
import matplotlib
import pmdarima as pm
import datetime as dt
import yfinance as yf
import requests
from sklearn.model_selection import TimeSeriesSplit
from sktime.forecasting.model_selection import temporal_train_test_split
from pandas.plotting import lag_plot
from pandas import datetime
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from pmdarima import model_selection
from pmdarima.utils import decomposed_plot
from pmdarima.arima import decompose
from sklearn import metrics
from sklearn.neighbors import KNeighborsRegressor
from fbprophet import Prophet
from prophet.diagnostics import cross_validation
from prophet.plot import plot_cross_validation_metric
from prophet.diagnostics import performance_metrics
from sklearn.linear_model import LinearRegression
from iexfinance.stocks import Stock
import random
from trafalgar import*
```

Step 2: The data

```
Thanks to Yahoo Finance (the library) I can research the stock history fo
r most assets being traded today.
So the first step will be to run through the process on a test stock, the
n create the functions necessary to do it with any stock. Let's use Citig
roup, since its stock symbol is only 1 letter:C.
```

In [2]:
```
c = yf.Ticker("C")
```

In [3]:
```
c.info
```

Out[3]:
```
{'zip': '10013',
 'sector': 'Financial Services',
 'fullTimeEmployees': 214000,
 'longBusinessSummary': 'Citigroup Inc., a diversified financial services hol
ding company, provides various financial products and services to consumers,
corporations, governments, and institutions in North America, Latin America,
Asia, Europe, the Middle East, and Africa. The company operates in two segmen
ts, Global Consumer Banking (GCB) and Institutional Clients Group (ICG). The
GCB segment offers traditional banking services to retail customers through r
etail banking, Citi-branded cards, and Citi retail services. It also provides
various banking, credit card, lending, and investment services through a netw
ork of local branches, offices, and electronic delivery systems. The ICG segm
ent offers wholesale banking products and services, including fixed income an
d equity sales and trading, foreign exchange, prime brokerage, derivative, eq
uity and fixed income research, corporate lending, investment banking and adv
isory, private banking, cash management, trade finance, and securities servic
es to corporate, institutional, public sector, and high-net-worth clients. As
of December 31, 2020, it operated 2,303 branches primarily in the United Stat
es, Mexico, and Asia. Citigroup Inc. was founded in 1812 and is headquartered
```

In [4]:
```
# get historical market data, here max is 5 years.
c.history(period="max")
```

Out[4]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1977-01-03	7.764117	7.814047	7.764117	7.814047	47952	0.00	0.0
1977-01-04	7.814046	7.839010	7.789080	7.839010	34217	0.00	0.0
1977-01-05	7.839010	7.839010	7.764115	7.789080	15422	0.00	0.0
1977-01-06	7.764115	7.764115	7.664254	7.739149	39036	0.00	0.0
1977-01-07	7.739149	7.764114	7.664254	7.764114	20482	0.00	0.0
...	...	...	...	...	...	...	...
2021-07-27	66.582827	67.893043	66.285048	67.476158	17127700	0.00	0.0
2021-07-28	67.813631	68.101483	66.989781	67.595261	20275900	0.00	0.0
2021-07-29	68.200738	69.064293	67.902966	68.299995	22261700	0.00	0.0
2021-07-30	67.919998	68.519997	67.239998	67.620003	18074300	0.51	0.0
2021-08-02	67.949997	69.120003	67.690002	67.800003	14211483	0.00	0.0

11240 rows × 7 columns

Well, no problem getting enough data for this one, unless my math is wrong, this is 44 years of data. Ok, let's do a train-test split and start predicting.....just kidding. One of the lessons learned from my last project is using too much data in my training set. What we'll need to do is determine the period for our predictions: how far in advance do we intend to predict? I'd say no more than a month.

In [5]:
```
df=c.history(period="max")
```

In [6]:
```
df.tail()
```

Out[6]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2021-07-27	66.582827	67.893043	66.285048	67.476158	17127700	0.00	0.0
2021-07-28	67.813631	68.101483	66.989781	67.595261	20275900	0.00	0.0
2021-07-29	68.200738	69.064293	67.902966	68.299995	22261700	0.00	0.0
2021-07-30	67.919998	68.519997	67.239998	67.620003	18074300	0.51	0.0
2021-08-02	67.949997	69.120003	67.690002	67.805000	14212532	0.00	0.0

What I want to figure out here is how we're going to split the data for training and testing; different splits give different results (better or worse). So let's make some loops and test a number of different training and testing values, and see what works best.

In [7]:
```
df1=df['Close']
```

In [8]:
```
trains=[14,30,60,180,360,720,900]
tests=[7,14,21,28,56]
```

In [9]:
```
df1.isna().sum()
```

Out[9]: 0

As the output I'd like to make a dataframe with some performance metrics, and see what split gives us the lowest errors.

In [10]:
```
cols=['Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
```

In [11]:
```
outs = pd.DataFrame(columns=cols)
```

In [12]:
```
for test_val in tests:
    for train_val in trains:
        val_a=test_val+train_val
        df_mod=df1.tail(val_a)
        train_data, test_data = temporal_train_test_split(df_mod, test_size=test_
        test_sq=test_data.squeeze()
        train_sq=train_data.squeeze()
        arima = pm.auto_arima(train_sq,error_action='ignore', trace=True,
                       suppress_warnings=True, maxiter=100,seasonal=True, m=1)
        y_pred = arima.predict(n_periods=test_data.shape[0])
        y_true=test_data
        ev_score= metrics.explained_variance_score(y_true, y_pred)
        mae= metrics.mean_absolute_error(y_true, y_pred)
        rmse = metrics.mean_squared_error(y_true, y_pred, squared=False)
        r2 = metrics.r2_score(y_true, y_pred)
        outs.loc[len(outs.index)] = [train_val,test_val,ev_score,mae,rmse,r2]
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.29 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=50.709, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=49.937, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=48.933, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=159.592, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=50.899, Time=0.03 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=50.809, Time=0.03 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.14 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=inf, Time=0.02 sec

Best model:  ARIMA(0,0,1)(0,0,0)[0] intercept
Total fit time: 0.623 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.16 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=98.996, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=98.645, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=97.995, Time=0.01 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=99.093, Time=0.01 sec
```

In [13]:
```
outs.head()
```

Out[13]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	14.0	7.0	0.558740	0.351103	0.439928	0.447912
1	30.0	7.0	0.000000	1.497639	1.610427	-6.398242
2	60.0	7.0	0.000000	1.414457	1.533376	-5.707236
3	180.0	7.0	-0.604902	1.457131	1.608086	-6.376750
4	360.0	7.0	-0.156665	0.894126	0.955222	-1.602887

In [14]:
```
outs[outs.MAE == outs.MAE.min()]
```

Out[14]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	14.0	7.0	0.55874	0.351103	0.439928	0.447912

In [15]:
```
outs[outs.RMSE == outs.RMSE.min()]
```

Out[15]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	14.0	7.0	0.55874	0.351103	0.439928	0.447912

In [16]:
```
outs[outs.R2 == outs.R2.min()]
```

Out[16]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
21	14.0	28.0	-10.085726	10.803809	12.078671	-54.441656

STONKS!!!

Now that we have a nice little bit of code to test various train/test splits, let's test it out on some more stocks. Finding files for the S&P 500, NASDAQ and Dow were very easy. So we can read them, and go through them all and see what we come up with.

In [17]:
```
sp_500=pd.read_csv('Data/constituents_csv.csv')
```

In [18]:
```
nsdq=pd.read_csv('Data/nasdaq.csv')
```

In [19]:
```
dow_30=pd.read_excel('Data/dow-jones-industrial-average-components.xls')
```

In [20]:
```
sp_500.head()
```

Out[20]:

	Unnamed: 0	Symbol	Name	Sector
0	0	MMM	3M	Industrials
1	1	ABT	Abbott Laboratories	Health Care
2	2	ABBV	AbbVie	Health Care
3	3	ABMD	Abiomed	Health Care
4	4	ACN	Accenture	Information Technology

In [21]:
```
sp_500.isna().sum()
```

Out[21]:
```
Unnamed: 0    0
Symbol        0
Name          0
Sector        0
dtype: int64
```

In [22]:
```
nsdq.head()
```

Out[22]:

	Unnamed: 0	Symbol	Company Name
0	1	AAL	American Airlines Group, Inc.
1	2	AAME	Atlantic American Corporation
2	3	AAOI	Applied Optoelectronics, Inc.
3	4	AAON	AAON, Inc.
4	5	AAPL	Apple Inc.

In [23]:
```
nsdq.isna().sum()
```

Out[23]:
```
Unnamed: 0      0
Symbol          0
Company Name    0
dtype: int64
```

In [24]:
```
dow_30.head()
```

Out[24]:

	Company Name	Ticker Symbol	Weighting %
0	3M Company	MMM	0.038022
1	American Express Company	AXP	0.025567
2	Amgen Inc.	AMGN	0.048569
3	Apple Inc.	AAPL	0.028752
4	Caterpillar Inc.	CAT	0.039120

In [25]:
```
dow_30.isna().sum()
```

Out[25]:
```
Company Name     0
Ticker Symbol    0
Weighting %      0
dtype: int64
```

In [26]:
```
dow_30.head()
```

Out[26]:

	Company Name	Ticker Symbol	Weighting %
0	3M Company	MMM	0.038022
1	American Express Company	AXP	0.025567
2	Amgen Inc.	AMGN	0.048569
3	Apple Inc.	AAPL	0.028752
4	Caterpillar Inc.	CAT	0.039120

In [27]:
```
new_cols=['Name','Symbol','Weight%']
dow_30.columns=new_cols
```

In [28]:
```
dow_30.head()
```

Out[28]:

	Name	Symbol	Weight%
0	3M Company	MMM	0.038022
1	American Express Company	AXP	0.025567
2	Amgen Inc.	AMGN	0.048569
3	Apple Inc.	AAPL	0.028752
4	Caterpillar Inc.	CAT	0.039120

What I want to do now is iterate through a variety of stocks, testing the various train/test splits, and coming up with a dataframe containing the stock symbol, the best train/test split, and the metrics.

In [29]:
```
stock = yf.Ticker(sp_500['Symbol'][0])
```

In [30]:
```
stock_df=stock.history(period='max')
```

In [31]:
```
stock_df.iloc[::-1]
```

Out[31]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-08-02	199.070007	200.699997	197.600006	197.639999	911953	0.0	0.0
2021-07-30	198.000000	199.240005	197.199997	197.940002	1910300	0.0	0.0
2021-07-29	200.000000	200.000000	197.940002	198.169998	2120200	0.0	0.0
2021-07-28	199.320007	200.369995	198.100006	198.279999	2139100	0.0	0.0
2021-07-27	197.210007	201.059998	194.910004	200.470001	2999900	0.0	0.0
...	...	...	...	...	...	...	...
1970-01-08	1.492125	1.515440	1.488795	1.512109	304000	0.0	0.0
1970-01-07	1.483799	1.495456	1.480468	1.492126	164800	0.0	0.0
1970-01-06	1.468811	1.483799	1.467146	1.483799	176000	0.0	0.0
1970-01-05	1.462150	1.470477	1.462150	1.468811	446400	0.0	0.0
1970-01-02	1.460485	1.468811	1.458819	1.460485	72000	0.0	0.0

13012 rows × 7 columns

In [32]:
```
stock_df.isna().sum()
```

Out[32]:
```
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

Let's try this out through the first 5 stocks in the dow dataframe

In [33]:
```
dow_5=dow_30.head()
```

In [34]:
```
dow_5
```

Out[34]:

	Name	Symbol	Weight%
0	3M Company	MMM	0.038022
1	American Express Company	AXP	0.025567
2	Amgen Inc.	AMGN	0.048569
3	Apple Inc.	AAPL	0.028752
4	Caterpillar Inc.	CAT	0.039120

In [35]:
```
cols2=['Symbol','Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
```

Out[35]:

index	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2

In [36]:
```
def tt_test (asset, train_list, test_list):
    """This function will take in a financial asset (stock, etf) as well as 2 lis
    Then the asset will be looked up through yahoo finance and gather the price h
    of the training and testing lists and run auto arima models on all of them. I
    dataframe with all the results."""

    stock = yf.Ticker(asset)
    df1=stock.history(period='5y')
    df=df1['Close']
    print("Processing: ",stock)
    if len(df)<(train_list[0]+test_list[0]):
        print ('Not enough historical data to model.')
        return None
    else:


        for test_val in test_list:
            for train_val in train_list:
                val_a=test_val+train_val
                df_mod=df.tail(val_a)
                train_data, test_data = temporal_train_test_split(df_mod, test_si
                test_sq=test_data.squeeze()
                train_sq=train_data.squeeze()
                arima = pm.auto_arima(train_sq,error_action='ignore', trace=True,
                        suppress_warnings=True, maxiter=100,seasonal=True, m=1)
                y_pred = arima.predict(n_periods=test_data.shape[0])
                y_true=test_data
                ev_score= metrics.explained_variance_score(y_true, y_pred)
                mae= metrics.mean_absolute_error(y_true, y_pred)
                rmse = metrics.mean_squared_error(y_true, y_pred, squared=False)
                r2 = metrics.r2_score(y_true, y_pred)
                reslts.loc[len(reslts.index)] = [stock,train_val,test_val,ev_scor

        return reslts
```

In [37]:
```
cols2=['Symbol','Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in dow_5['Symbol']:
    stock_res_d5 = tt_test(each,trains,tests)
    print (stock_res_d5[(stock_res_d5.MAE == stock_res_d5.MAE.min())])
```

```
Processing:  yfinance.Ticker object <MMM>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=64.758, Time=0.28 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=57.296, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=59.194, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=59.131, Time=0.01 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=190.069, Time=0.00 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=61.022, Time=0.06 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.395 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=140.015, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=157.135, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=134.130, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=144.359, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.551, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=136.226, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=136.028, Time=0.10 sec
```

In [38]:
```
stock_res_d5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 175 entries, 0 to 174
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Symbol     175 non-null    object
 1   Train_Len  175 non-null    object
 2   Test_Len   175 non-null    object
 3   Exp_var    175 non-null    float64
 4   MAE        175 non-null    float64
 5   RMSE       175 non-null    float64
 6   R2         175 non-null    float64
dtypes: float64(4), object(3)
memory usage: 10.9+ KB
```

In [39]:
```
stock_res_d5.head()
```

Out[39]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	yfinance.Ticker object <MMM>	14	7	2.220446e-16	1.498981	1.617162	-0.196756
1	yfinance.Ticker object <MMM>	30	7	6.480774e-02	1.306134	1.471526	0.009088
2	yfinance.Ticker object <MMM>	60	7	-1.245108e-01	1.511385	1.572529	-0.131608
3	yfinance.Ticker object <MMM>	180	7	-7.025763e-01	2.016206	2.304249	-1.429728
4	yfinance.Ticker object <MMM>	360	7	-8.272976e-02	1.479489	1.541166	-0.086920

In [40]:
```
stock_res_d5.tail()
```

Out[40]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
170	yfinance.Ticker object <CAT>	60	56	2.220446e-16	16.555398	19.465362	-1.255766
171	yfinance.Ticker object <CAT>	180	56	-1.662267e+00	30.867880	37.101112	-7.194887
172	yfinance.Ticker object <CAT>	360	56	-7.233721e-01	23.397647	28.284696	-3.762908
173	yfinance.Ticker object <CAT>	720	56	-3.804227e-01	20.193412	24.243187	-2.499037
174	yfinance.Ticker object <CAT>	900	56	-3.126438e-01	19.642331	23.528238	-2.295702

I don't like that 'yfinance Ticker object' at the beginning of each entry. So let's get rid of that.

In [41]:

```
for each in stock_res_d5.index:
    stock_res_d5['Symbol'][each]=str(stock_res_d5['Symbol'][each])
    stock_res_d5['Symbol'][each]=stock_res_d5['Symbol'][each].replace('>','')
    stock_res_d5['Symbol'][each]=stock_res_d5['Symbol'][each].split('<', 1)[-1]
```

```
D:\anaconda3\envs\learn-env\lib\site-packages\ipykernel_launcher.py:2: SettingW
ithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)

D:\anaconda3\envs\learn-env\lib\site-packages\ipykernel_launcher.py:3: SettingW
ithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)

D:\anaconda3\envs\learn-env\lib\site-packages\ipykernel_launcher.py:4: SettingW
ithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

We get some warnings with that, but it works.

In [42]:
```
stock_res_d5
```

Out[42]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	MMM	14	7	2.220446e-16	1.498981	1.617162	-0.196756
1	MMM	30	7	6.480774e-02	1.306134	1.471526	0.009088
2	MMM	60	7	-1.245108e-01	1.511385	1.572529	-0.131608
3	MMM	180	7	-7.025763e-01	2.016206	2.304249	-1.429728
4	MMM	360	7	-8.272976e-02	1.479489	1.541166	-0.086920
...	...	...	...	...	...	...	...
170	CAT	60	56	2.220446e-16	16.555398	19.465362	-1.255766
171	CAT	180	56	-1.662267e+00	30.867880	37.101112	-7.194887
172	CAT	360	56	-7.233721e-01	23.397647	28.284696	-3.762908
173	CAT	720	56	-3.804227e-01	20.193412	24.243187	-2.499037
174	CAT	900	56	-3.126438e-01	19.642331	23.528238	-2.295702

Now let's look at the first stock in the dataframe, 3M. I'll pull it out and make it a generic dataframe.

In [43]:
```
df_x=pd.DataFrame(columns=stock_res_d5.columns)
```

Since the results dataframe is 175 entries long, and 175/5 is 35, taking the first 35 entries will get us all of 3M.

In [44]:
```
df_x=stock_res_d5.loc[stock_res_d5.index[0:35]]
```

In [45]:
```
df_x.sort_values(by=['MAE']).head()
```

Out[45]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	MMM	30	7	6.480774e-02	1.306134	1.471526	0.009088
5	MMM	720	7	2.220446e-16	1.381430	1.487678	-0.012783
6	MMM	900	7	2.220446e-16	1.381430	1.487678	-0.012783
20	MMM	900	21	-4.440892e-16	1.431906	1.709450	-0.132281
19	MMM	720	21	-4.440892e-16	1.431906	1.709450	-0.132281

In [46]:
```
df_y=df_x.sort_values(by=['MAE']).head(2)
```

In [47]:
```
df_y
```

Out[47]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	MMM	30	7	6.480774e-02	1.306134	1.471526	0.009088
5	MMM	720	7	2.220446e-16	1.381430	1.487678	-0.012783

In [48]:
```
df_y=pd.DataFrame(columns=stock_res_d5.columns)
```

In [49]:
```
a=0
while a<=(len(stock_res_d5)-35):
    df_x=stock_res_d5.loc[stock_res_d5.index[a:(a+35)]]
    df_ph=df_x.sort_values(by=['MAE']).head(2)
    df_y=pd.concat([df_y,df_ph])
    a+=35
```

In [50]:
```
df_y
```

Out[50]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	MMM	30	7	6.480774e-02	1.306134	1.471526	0.009088
5	MMM	720	7	2.220446e-16	1.381430	1.487678	-0.012783
41	AXP	900	7	1.077875e-01	1.296784	1.483945	-0.006632
40	AXP	720	7	1.059213e-01	1.299171	1.489844	-0.014650
85	AMGN	30	21	8.035365e-02	1.651802	2.095074	0.044105
84	AMGN	14	21	9.815283e-02	1.677383	2.035643	0.097568
108	AAPL	180	7	0.000000e+00	1.306431	1.463181	-0.014998
115	AAPL	180	14	0.000000e+00	1.468930	1.969493	-0.244176
146	CAT	900	7	-7.358317e-02	2.074228	2.572308	-0.076399
144	CAT	360	7	-5.799273e-03	2.077785	2.489329	-0.008073

Ok, so I managed to get some data on train/test splits, but I still need more. So far it looks like 30 and 60 are leading on the training set. Clearly 7 days for the test set is winning. But let's run through the entire dow 30, and see what that'll get us.

In [51]:
```
df_y['Train_Len'].value_counts()
```

Out[51]:
```
30     2
900    2
180    2
720    2
14     1
360    1
Name: Train_Len, dtype: int64
```

In [52]:
```
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in dow_30['Symbol']:
    stock_res_d30 = tt_test(each,trains,tests)
    print (stock_res_d30[(stock_res_d30.MAE == stock_res_d30.MAE.min())])
for each in stock_res_d30.index:
    stock_res_d30['Symbol'][each]=str(stock_res_d30['Symbol'][each])
    stock_res_d30['Symbol'][each]=stock_res_d30['Symbol'][each].replace('>','')
    stock_res_d30['Symbol'][each]=stock_res_d30['Symbol'][each].split('<', 1)[-1]
```

```
Processing:  yfinance.Ticker object <MMM>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=64.758, Time=0.27 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=57.296, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=59.194, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=59.131, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=190.069, Time=0.00 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=61.022, Time=0.06 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.393 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=140.015, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=157.135, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=134.130, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=144.359, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.551, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=136.226, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=136.028, Time=0.10 sec
```

In [53]:
```
stock_res_d30.info
```

Out[53]:
```
<bound method DataFrame.info of      Symbol Train_Len Test_Len       Exp_var
MAE      RMSE        R2
0       MMM        14        7  0.000000e+00  1.514695  1.639434 -0.201534
1       MMM        30        7  6.422228e-02  1.321849  1.484691  0.014582
2       MMM        60        7 -1.233653e-01  1.527099  1.591401 -0.132159
3       MMM       180        7 -6.962104e-01  2.031920  2.328771 -1.424385
4       MMM       360        7 -8.185265e-02  1.495203  1.559628 -0.087401
...     ...       ...      ...           ...       ...       ...       ...
1045    WMT        60       56 -2.220446e-16  4.445892  4.810491 -5.394898
1046    WMT       180       56 -2.220446e-16  4.445892  4.810491 -5.394898
1047    WMT       360       56 -2.194376e-03  3.917759  4.291901 -4.090428
1048    WMT       720       56 -2.705722e-01  2.246594  2.702353 -1.018083
1049    WMT       900       56 -1.512597e-03  3.997541  4.371827 -4.281787

[1050 rows x 7 columns]>
```

In [54]:
```
len(stock_res_d30)
```

Out[54]:
```
1050
```

In [55]:
```
df_y_d30=pd.DataFrame(columns=stock_res_d30.columns)
```

In [56]:
```
df_y_d30
```

Out[56]:

Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2

In [57]:
```
a=0
b=len(stock_res_d30)
c=int(b/len(dow_30))
while a<=(b-c):
    df_x=stock_res_d30.loc[stock_res_d30.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(2)
    df_y_d30=pd.concat([df_y,df_ph])
    a+=c
```

In [58]:
```
df_y_d30
```

Out[58]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	MMM	30	7	6.480774e-02	1.306134	1.471526	0.009088
5	MMM	720	7	2.220446e-16	1.381430	1.487678	-0.012783
41	AXP	900	7	1.077875e-01	1.296784	1.483945	-0.006632
40	AXP	720	7	1.059213e-01	1.299171	1.489844	-0.014650
85	AMGN	30	21	8.035365e-02	1.651802	2.095074	0.044105
84	AMGN	14	21	9.815283e-02	1.677383	2.035643	0.097568
108	AAPL	180	7	0.000000e+00	1.306431	1.463181	-0.014998
115	AAPL	180	14	0.000000e+00	1.468930	1.969493	-0.244176
146	CAT	900	7	-7.358317e-02	2.074228	2.572308	-0.076399
144	CAT	360	7	-5.799273e-03	2.077785	2.489329	-0.008073
1016	WMT	30	7	1.110223e-16	1.125711	1.145048	-28.859270
1018	WMT	180	7	1.110223e-16	1.125711	1.145048	-28.859270

This MAE is pretty low, but so is the Explained Variance (in fact, it's almost non-existent).

In [59]:
```
df_y_d30['Train_Len'].value_counts()
```

Out[59]:
```
30     3
180    3
900    2
720    2
14     1
360    1
Name: Train_Len, dtype: int64
```

In [60]:
```
df_y_d30['Test_Len'].value_counts()
```

Out[60]:
```
7     9
21    2
14    1
Name: Test_Len, dtype: int64
```

So we have a clear winner in the Test Length. The Training Length is still a little close.

Next on the agenda is to run through this modeling again, but with 50 stocks from the S&P 500. But to shorten the run time of this notebook, I'm going to cut the test list down to 7 and 14 days. 7 seems to be the clear winner, but I'd really like to use 14 days. I wish I had a good reason why, but I don't.

In [61]:
```
picks_sp50=[]
for i in range (0,50):
    x = random.randint(0,(len(sp_500)-1))
    picks_sp50.append(x)
```

In [62]:
```
tests=[7, 14]
```

In [63]:
```
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in sp_500['Symbol'][picks_sp50]:
    stock_res_sp50 = tt_test(each,trains,tests)
    print (stock_res_sp50[(stock_res_sp50.MAE == stock_res_sp50.MAE.min())])
for each in stock_res_sp50.index:
    stock_res_sp50['Symbol'][each]=str(stock_res_sp50['Symbol'][each])
    stock_res_sp50['Symbol'][each]=stock_res_sp50['Symbol'][each].replace('>','')
    stock_res_sp50['Symbol'][each]=stock_res_sp50['Symbol'][each].split('<', 1)[-
df_y_sp50=pd.DataFrame(columns=stock_res_sp50.columns)
```

```
Processing:  yfinance.Ticker object <VIAC>
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.21 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=35.868, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=37.863, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=37.861, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=35.039, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=39.840, Time=0.04 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.306 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=87.895, Time=0.35 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=117.425, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=85.641, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=99.701, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=311.759, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=87.070, Time=0.08 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=87.269, Time=0.05 sec
```

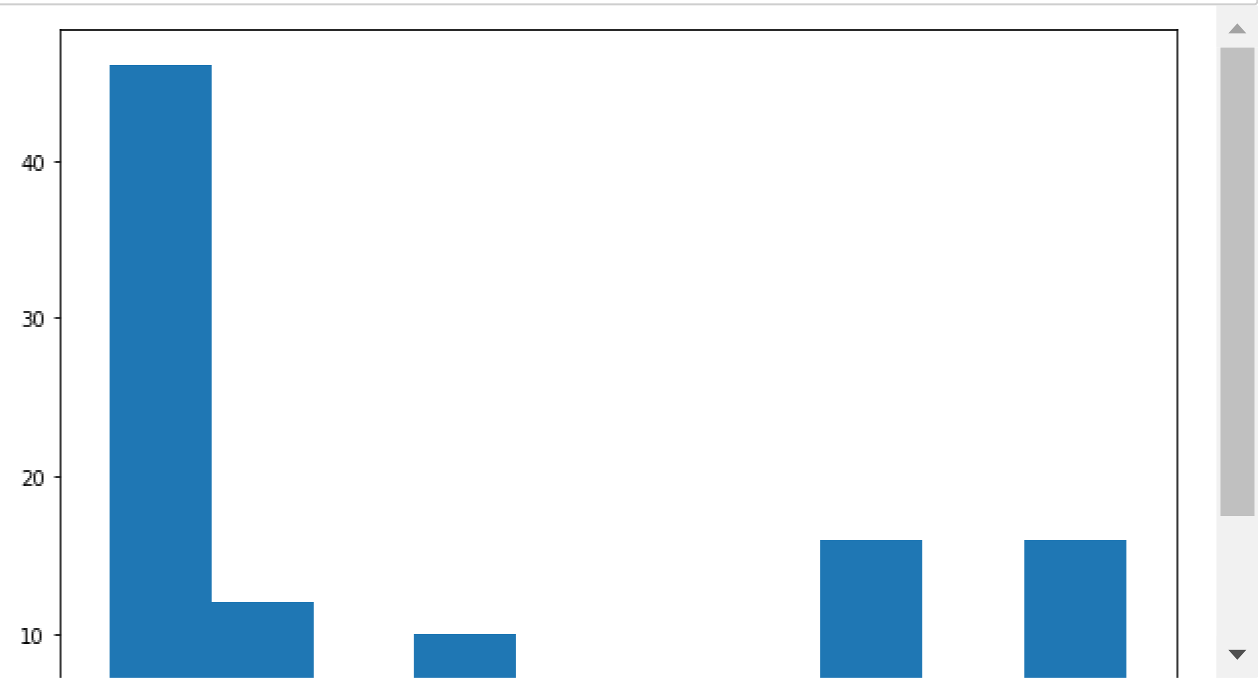
In [64]:
```
df_y_sp50=pd.DataFrame(columns=stock_res_sp50.columns)
```

In [65]:
```
a=0
b=len(stock_res_sp50)
c=int((len(stock_res_sp50))/(len(picks_sp50)))
while a<=(b-c):
    df_x=stock_res_sp50.loc[stock_res_sp50.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(2)
    df_y_sp50=pd.concat([df_y_sp50,df_ph])
    a+=c
```

In [66]:
```
df_y_sp50['Train_Len'].value_counts()
```

Out[66]:
```
14     18
900    16
720    16
30     15
60     13
180    12
360    10
Name: Train_Len, dtype: int64
```

In [67]:
```
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(df_y_sp50['Train_Len'])
plt.show()
```



In [68]:
```
df_y_sp50['Test_Len'].value_counts()
```

Out[68]:
```
7     66
14    34
Name: Test_Len, dtype: int64
```

So, a pretty clear winner in the test value.

Again, no clear winner for training, but clearly 7 days is the winner in testing. Let's do this one more time with the Nasdaq.

In [69]:
```
len(nsdq)
```

Out[69]: 1701

In [70]:
```
nsdq.head()
```

Out[70]:

	Unnamed: 0	Symbol	Company Name
0	1	AAL	American Airlines Group, Inc.
1	2	AAME	Atlantic American Corporation
2	3	AAOI	Applied Optoelectronics, Inc.
3	4	AAON	AAON, Inc.
4	5	AAPL	Apple Inc.

In [71]:
```
nsdq.drop(columns='Unnamed: 0',inplace=True)
```

This dataframe is WAY too big to look through, so let's just do another 50, like we did with the S&P.

I had found a few stocks were missing from my NASDAQ list, so I've written the following code to get rid of all the symbols that had been delisted since I got this data. I've hashed it out since I saved the file, and don't need this code at the moment; but I may in the future, so I want to keep it around.

In [72]:
```
#no_data=[]
#for each in nsdq['Symbol']:
#    stock = yf.Ticker(each)
#    df1=stock.history(period='1d')
#    df=df1['Close']
#    if len(df)==0:
#        no_data.append(each)
```

In [73]:
```
#len(no_data)
```

In [74]:
```
#no_data[4:8]
```

In [75]:
```
#y=nsdq.loc[nsdq['Symbol']==no_data[4]].index
```

In [76]:
```
#y[0]
```

So here's this loop will create a list of index numbers for all the delisted/bad stock symbols

In [77]:
```
#nd_index=[]
#for each in no_data:
#    y=nsdq.loc[nsdq['Symbol']==each].index
#    nd_index.append(y[0])
```

In [78]:
```
#len(nd_index)
```

In [79]:
```
#nsdq = nsdq.drop(labels=nd_index, axis=0)
```

In [80]:
```
nsdq.reset_index(drop=True,inplace=True)
```

Like I mentioned above, saving it so it saves time running this notebook, and it's readily useable.

In [81]:
```
#nsdq.to_csv('Data/nasdaq.csv',index=True)
```

Just to be safe, let's do the same thing with the S&P data.

In [82]:
```
#no_data_sp=[]
#for each in sp_500['Symbol']:
#    stock = yf.Ticker(each)
#    df1=stock.history(period='1d')
#    df=df1['Close']
#    if len(df)==0:
#        no_data_sp.append(each)
```

In [83]:
```
#no_data_sp
```

In [84]:
```
#sp_500.loc[sp_500['Symbol']==no_data_sp[1]].index
```

In [85]:
```
#sp_500.loc[sp_500['Symbol']==no_data_sp[0]].index
```

In [86]:
```
#sp_500.loc[sp_500['Symbol']==no_data_sp[2]].index
```

In [87]:
```
#sp_500 = sp_500.drop(labels=[18,64,78], axis=0)
```

In [88]:
```
#sp_500.to_csv('Data/constituents_csv.csv',index=True)
```

In [89]:
```
#sp_500.reset_index(drop=True,inplace=True)
```

In [90]:
```
sp_500.head()
```

Out[90]:

	Unnamed: 0	Symbol	Name	Sector
0	0	MMM	3M	Industrials
1	1	ABT	Abbott Laboratories	Health Care
2	2	ABBV	AbbVie	Health Care
3	3	ABMD	Abiomed	Health Care
4	4	ACN	Accenture	Information Technology

In [91]:
```
sp_500.drop(columns='Unnamed: 0',inplace=True)
```

We'll make another list of random numbers for us to test. We'll just use 25 for this test.

In [92]:
```
picks=[]
for i in range (0,25):
    x = random.randint(0,(len(nsdq)-1))
    picks.append(x)
```

In [93]:
```
nsdq.head()
```

Out[93]:

	Symbol	Company Name
0	AAL	American Airlines Group, Inc.
1	AAME	Atlantic American Corporation
2	AAOI	Applied Optoelectronics, Inc.
3	AAON	AAON, Inc.
4	AAPL	Apple Inc.

In [94]:
```
cols2=['Symbol','Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in nsdq['Symbol'][picks]:
    stock_res_nd50 = tt_test(each,trains,tests)
    print (stock_res_nd50[(stock_res_nd50.MAE == stock_res_nd50.MAE.min())])
for each in stock_res_nd50.index:
    stock_res_nd50['Symbol'][each]=str(stock_res_nd50['Symbol'][each])
    stock_res_nd50['Symbol'][each]=stock_res_nd50['Symbol'][each].replace('>','')
    stock_res_nd50['Symbol'][each]=stock_res_nd50['Symbol'][each].split('<', 1)[-
df_x=pd.DataFrame(columns=stock_res_nd50.columns)
df_y_nd50=pd.DataFrame(columns=stock_res_nd50.columns)
```

```
Processing:  yfinance.Ticker object <FLDM>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=-6.605, Time=0.22 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=-8.650, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-11.619, Time=0.10 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=-10.972, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=90.995, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-10.038, Time=0.05 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-9.890, Time=0.11 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-10.089, Time=0.26 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.03 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.813 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-19.746, Time=0.17 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-27.737, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-25.742, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-25.742, Time=0.03 sec
```

In [95]:
```
a=0
b=len(stock_res_nd50)
c=int(b/len(picks))
while a<=(b-c):
    df_x=stock_res_nd50.loc[stock_res_nd50.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(2)
    df_y_nd50=pd.concat([df_y_nd50,df_ph])
    a+=c
```

In [96]:
```
stock_res_nd50.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 350 entries, 0 to 349
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Symbol     350 non-null    object 
 1   Train_Len  350 non-null    object 
 2   Test_Len   350 non-null    object 
 3   Exp_var    350 non-null    float64
 4   MAE        350 non-null    float64
 5   RMSE       350 non-null    float64
 6   R2         350 non-null    float64
dtypes: float64(4), object(3)
memory usage: 31.9+ KB
```

In [97]:
```
df_y_nd50['Test_Len'].value_counts()
```

Out[97]:
```
7     34
14    16
Name: Test_Len, dtype: int64
```

In [98]:
```
df_y_nd50['Train_Len'].value_counts()
```

Out[98]:
```
180    11
360    10
14      7
30      7
60      5
900     5
720     5
Name: Train_Len, dtype: int64
```

Still no clear winner in the training length. But test length is definitely going to be 7 days. What if there is some correlation between the test length and training length? Let's run through these indices again, but this time with just 7 days as our test length.

In [99]:
```
tests=[7]
```

In [100]:
```
cols2=['Symbol','Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in dow_30['Symbol']:
    stock_res_d30 = tt_test(each,trains,tests)
    print (stock_res_d30[(stock_res_d30.MAE == stock_res_d30.MAE.min())])
for each in stock_res_d30.index:
    stock_res_d30['Symbol'][each]=str(stock_res_d30['Symbol'][each])
    stock_res_d30['Symbol'][each]=stock_res_d30['Symbol'][each].replace('>','')
    stock_res_d30['Symbol'][each]=stock_res_d30['Symbol'][each].split('<', 1)[-1]
df_x=pd.DataFrame(columns=stock_res_d30.columns)
df_y_d30=df_x.sort_values(by=['MAE']).head(3)
```

```
Processing:  yfinance.Ticker object <MMM>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=64.758, Time=0.26 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=57.296, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=59.194, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=59.131, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=190.069, Time=0.02 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=61.022, Time=0.05 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.374 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=140.015, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=157.135, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=134.130, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=144.359, Time=0.01 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.551, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=136.226, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=136.028, Time=0.09 sec
```

In [101]:
```
stock_res_d30.head(14)
```

Out[101]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	MMM	14	7	0.000000	1.516123	1.641495	-0.201955
1	MMM	30	7	0.064166	1.323276	1.485939	0.015057
2	MMM	60	7	-0.123256	1.528527	1.593159	-0.132210
3	MMM	180	7	-0.695601	2.033348	2.331023	-1.423829
4	MMM	360	7	-0.081770	1.496631	1.561349	-0.087449
5	MMM	720	7	0.000000	1.398573	1.504749	-0.010036
6	MMM	900	7	0.000000	1.398573	1.504749	-0.010036
7	AXP	14	7	0.002374	1.555829	1.843971	-0.648910
8	AXP	30	7	0.000000	1.422856	1.556314	-0.174582
9	AXP	60	7	0.000000	1.422856	1.556314	-0.174582
10	AXP	180	7	-1.331191	2.086724	2.447084	-1.903933

In [102]:
```
df_y_d30=pd.DataFrame(columns=stock_res_d30.columns)
```

In [103]:
```
a=0
b=len(stock_res_d30)
c=int(b/len(dow_30))
while a<=(b-c):
    df_x=stock_res_d30.loc[stock_res_d30.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_d30=pd.concat([df_y_d30,df_ph])
    a+=c
```

In [104]:
```
df_y_d30.head(15)
```

Out[104]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	MMM	30	7	6.416622e-02	1.323276	1.485939	0.015057
5	MMM	720	7	0.000000e+00	1.398573	1.504749	-0.010036
6	MMM	900	7	0.000000e+00	1.398573	1.504749	-0.010036
13	AXP	900	7	1.171686e-01	1.269641	1.448686	-0.017741
12	AXP	720	7	1.152928e-01	1.272027	1.454902	-0.026493
11	AXP	360	7	1.752703e-01	1.289380	1.484851	-0.069189
16	AMGN	60	7	2.517400e-01	1.726488	2.105256	0.239561
19	AMGN	720	7	-4.671574e-03	2.020674	2.631600	-0.188213
15	AMGN	30	7	0.000000e+00	2.042862	2.650627	-0.205457
24	AAPL	180	7	2.220446e-16	1.311430	1.467487	-0.015772
21	AAPL	14	7	2.852306e-01	1.565541	1.991554	-0.870818

In [105]:
```
df_y_d30['Train_Len'].value_counts()
```

Out[105]:
```
900    17
720    14
360    13
14     12
30     12
180    12
60     10
Name: Train_Len, dtype: int64
```

Again, unfortunately there is no clear leader in training length. Let's try again with the S&P

In [106]:
```
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in sp_500['Symbol'][picks_sp50]:
    stock_res_sp50 = tt_test(each,trains,tests)
    print (stock_res_sp50[(stock_res_sp50.MAE == stock_res_sp50.MAE.min())])
for each in stock_res_sp50.index:
    stock_res_sp50['Symbol'][each]=str(stock_res_sp50['Symbol'][each])
    stock_res_sp50['Symbol'][each]=stock_res_sp50['Symbol'][each].replace('>','')
    stock_res_sp50['Symbol'][each]=stock_res_sp50['Symbol'][each].split('<', 1)[-
df_x_sp50=pd.DataFrame(columns=stock_res_sp50.columns)
df_y_sp50=df_x_sp50.sort_values(by=['MAE']).head(3)
```

```
Processing:  yfinance.Ticker object <VIAC>
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.20 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=35.868, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=37.863, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=37.861, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=35.039, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=39.840, Time=0.03 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.296 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=87.895, Time=0.35 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=117.425, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=85.641, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=99.701, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=311.759, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=87.070, Time=0.08 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=87.269, Time=0.06 sec
```

In [107]:
```
len(stock_res_sp50)
```

Out[107]: 350

In [108]:
```
c=int((len(stock_res_sp50))/(len(picks_sp50)))
```

In [109]:
```
a=0
b=len(stock_res_sp50)
while a<=(b-c):
    df_x=stock_res_sp50.loc[stock_res_sp50.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_sp50=pd.concat([df_y_sp50,df_ph])
    a+=c
```

In [110]:
```
df_y_sp50.head(15)
```

Out[110]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	VIAC	30	7	-4.851410e-01	0.413629	0.553222	-0.503760
5	VIAC	720	7	2.089355e-01	0.487941	0.591491	-0.719001
0	VIAC	14	7	0.000000e+00	0.510001	0.674506	-1.235379
13	GILD	900	7	-3.543124e-03	0.546585	0.717810	-0.288433
11	GILD	360	7	-5.816927e-03	0.546769	0.720180	-0.296952
12	GILD	720	7	-3.260923e-02	0.569731	0.750903	-0.409970
20	GIS	900	7	-7.666760e-01	0.367147	0.389013	-0.940960
16	GIS	60	7	-2.220446e-16	0.375715	0.468112	-1.810526
17	GIS	180	7	-2.220446e-16	0.375715	0.468112	-1.810526
26	ETN	720	7	1.672066e-01	0.678099	0.799427	0.135654
27	ETN	900	7	1.294324e-01	0.680861	0.845992	0.032030
25	ETN	360	7	2.012794e-01	0.700435	0.770921	0.196198
30	RHI	60	7	3.454420e-01	8.104862	8.252624	-17.443951
31	RHI	180	7	3.489703e-01	8.494306	8.634652	-19.191080
28	RHI	14	7	-2.220446e-16	8.842142	9.048539	-21.173117

In [111]:
```
df_y_sp50['Train_Len'].value_counts()
```

Out[111]:
```
14     23
30     23
360    23
180    22
60     20
720    20
900    19
Name: Train_Len, dtype: int64
```

Still no clear winner with the training length. Let's try this one more time, with the NASDAQ stocks.

In [112]:
```
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
for each in nsdq['Symbol'][picks]:
    print (each)
    stock_res_nd50 = tt_test(each,trains,tests)
    print (stock_res_nd50[(stock_res_nd50.MAE == stock_res_nd50.MAE.min())])
for each in stock_res_nd50.index:
    stock_res_nd50['Symbol'][each]=str(stock_res_nd50['Symbol'][each])
    stock_res_nd50['Symbol'][each]=stock_res_nd50['Symbol'][each].replace('>','')
    stock_res_nd50['Symbol'][each]=stock_res_nd50['Symbol'][each].split('<', 1)[-
df_x=pd.DataFrame(columns=stock_res_nd50.columns)
df_y_nd50=df_x.sort_values(by=['MAE']).head(3)
```

```
FLDM
Processing:  yfinance.Ticker object <FLDM>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=-6.605, Time=0.20 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=-8.650, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=-11.619, Time=0.08 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=-10.972, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=90.995, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=-10.038, Time=0.05 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=-9.890, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=-10.089, Time=0.27 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.02 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.770 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-19.746, Time=0.17 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-27.737, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-25.742, Time=0.05 sec
```

In [113]:
```
a=0
b=len(stock_res_nd50)
c=int((len(stock_res_nd50))/(len(picks)))
while a<=(b-c):
    df_x=stock_res_nd50.loc[stock_res_nd50.index[a:(a+c)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_nd50=pd.concat([df_y_nd50,df_ph])
    a+=c
```

In [114]:
```
df_y_nd50['Train_Len'].value_counts()
```

Out[114]:
```
720    15
900    14
180    13
360    12
30      8
60      7
14      6
Name: Train_Len, dtype: int64
```

Results

So from the earlier 2 runs it looks like 30 days will be our best training length, and 7 our best test. Let's actually take a look at how these predictions look.

In [115]:
```
dow_5
```

Out[115]:

	Name	Symbol	Weight%
0	3M Company	MMM	0.038022
1	American Express Company	AXP	0.025567
2	Amgen Inc.	AMGN	0.048569
3	Apple Inc.	AAPL	0.028752
4	Caterpillar Inc.	CAT	0.039120

In [116]:
```
df1=yf.Ticker('MMM')
df=df1.history(period="max")
MMM=df['Close']
```

In [117]:
```
df1=yf.Ticker('AXP')
df=df1.history(period="max")
AXP=df['Close']
```

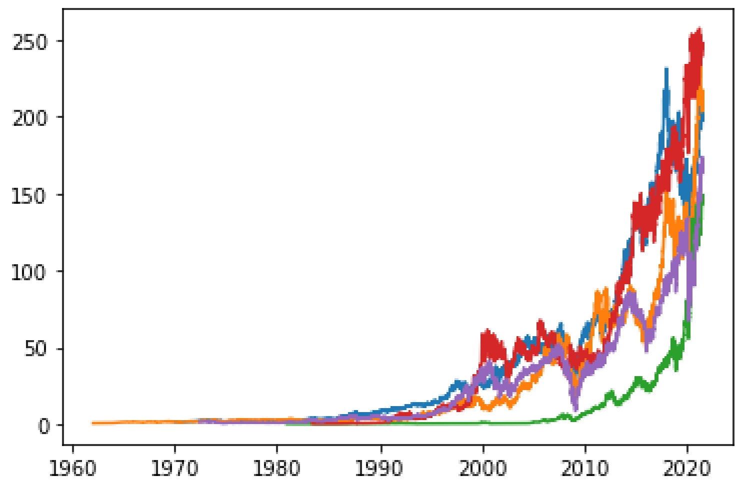
In [118]:
```
df1=yf.Ticker('AMGN')
df=df1.history(period="max")
AMGN=df['Close']
```

In [119]:
```
df1=yf.Ticker('AAPL')
df=df1.history(period="max")
AAPL=df['Close']
```

In [120]:
```
df1=yf.Ticker('CAT')
df=df1.history(period="max")
CAT=df['Close']
```

In [121]:
```
plt.plot(MMM)
plt.plot(CAT)
plt.plot(AAPL)
plt.plot(AMGN)
plt.plot(AXP)
```

Out[121]: [<matplotlib.lines.Line2D at 0x14c44135eb8>]



In [122]:
```
MMM=MMM.tail(37)
AXP=AXP.tail(37)
AAPL=AAPL.tail(37)
AMGN=AMGN.tail(37)
CAT=CAT.tail(37)
```

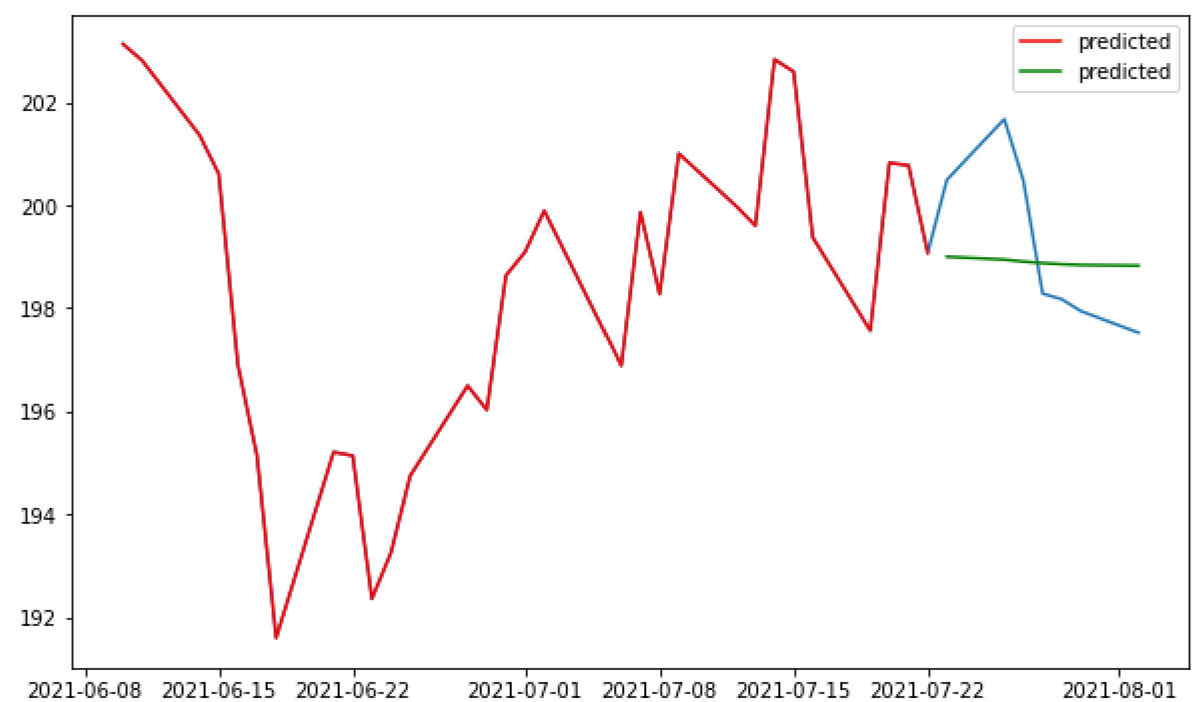
In [123]:
```
stonks=[MMM,AXP,AAPL,AMGN,CAT]
```

In [124]:
```
train, test = temporal_train_test_split(MMM, test_size=7)
test_sq=test.squeeze()
train_sq=train.squeeze()
arima = pm.auto_arima(train_sq,error_action='ignore', trace=True,
       suppress_warnings=True, maxiter=100,seasonal=True, m=1)
y_pred = arima.predict(n_periods=test.shape[0])
y_true=test
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=140.015, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=157.135, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=134.130, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=144.359, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.551, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=136.226, Time=0.05 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=136.028, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=138.005, Time=0.09 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.02 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.480 seconds
```

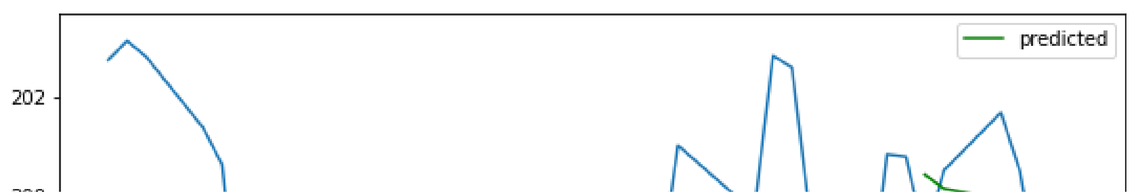
In [127]:
```
plt.figure(figsize=(10,6))
plt.plot(MMM)
plt.plot(train.index,train_sq.values , color='red', label = 'predicted')
plt.plot(test.index, y_pred, color='green', label = 'predicted')
plt.legend()
plt.show()
```



In [131]:
```
for each in stonks:
    train, test = temporal_train_test_split(each, test_size=7)
    test_sq=test.squeeze()
    train_sq=train.squeeze()
    arima = pm.auto_arima(train_sq,error_action='ignore', trace=True,
           suppress_warnings=True, maxiter=100,seasonal=True, m=1)
    y_pred = arima.predict(n_periods=test.shape[0])
    y_true=test
    plt.figure(figsize=(10,6))
    plt.plot(each)
    plt.plot(test.index, y_pred, color='green', label = 'predicted')
    plt.legend()
    plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=139.404, Time=0.12 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=158.962, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=133.698, Time=0.05 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=145.659, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.588, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=135.804, Time=0.09 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=135.657, Time=0.11 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=137.505, Time=0.09 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.02 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.504 seconds
```



The results don't "look" great, even though some of the MAE numbers weren't terrible; the R2 and Explained Variance were pretty bad. But that's not surprising; ARIMA isn't ideal for forecasting something like a stock price. Since it uses (among other things) the previous entry to make its predictions. And there is just too much randomness in the data to really get good predictions.

In the next notebook, we'll try again with Prophet, Facebook's time forecasting library.

In [ ]: