My Capstone Project:

```
Welcome to my final project for the Flatiron School. This is it, the big
 one, the one that defines my talent as a data       scientist.
```

The Problem:

```
Picking good stocks is the problem. I intend to build a website that'll p
ick 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	7.822093	7.872396	7.822093	7.872396	47952	0.0	0.0
1977-01-04	7.872394	7.897545	7.847242	7.897545	34217	0.0	0.0
1977-01-05	7.897544	7.897544	7.822090	7.847241	15422	0.0	0.0
1977-01-06	7.822091	7.822091	7.721485	7.796939	39036	0.0	0.0
1977-01-07	7.796939	7.822091	7.721485	7.822091	20482	0.0	0.0
...	...	...	...	...	...	...	...
2021-07-16	68.709999	68.760002	66.419998	66.900002	19278900	0.0	0.0
2021-07-19	65.459999	66.070000	64.360001	65.080002	33318600	0.0	0.0
2021-07-20	65.180000	66.779999	64.779999	66.290001	20568400	0.0	0.0
2021-07-21	67.010002	68.250000	66.930000	67.889999	23387300	0.0	0.0
2021-07-22	67.750000	67.790001	66.410004	66.930000	16519500	0.0	0.0

11233 rows × 7 columns

Well, no problem getting enough data for this one, unless my birthday 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]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Date							
2021-07-16	68.709999	68.760002	66.419998	66.900002	19278900	0.0	0.0
2021-07-19	65.459999	66.070000	64.360001	65.080002	33318600	0.0	0.0
2021-07-20	65.180000	66.779999	64.779999	66.290001	20568400	0.0	0.0
2021-07-21	67.010002	68.250000	66.930000	67.889999	23387300	0.0	0.0
2021-07-22	67.750000	67.790001	66.410004	66.930000	16519500	0.0	0.0

After some more thought, I've decided to predict 2 weeks out.

Another important decision to make 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]:
```
def report_metrics(y_true, y_pred):
    print("Explained Variance:\n\t", metrics.explained_variance_score(y_true, y_p
    print("MAE:\n\t", metrics.mean_absolute_error(y_true, y_pred))
    print("RMSE:\n\t", metrics.mean_squared_error(y_true, y_pred, squared=False))
    print("r^2:\n\t", metrics.r2_score(y_true, y_pred))
```

In [10]:
```
df1.isna().sum()
```

Out[10]: 0

In [11]:
```
cols=['Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
```

In [12]:
```
outs = pd.DataFrame(columns=cols)
```

In [13]:
```
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=54.163, Time=0.16 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=54.483, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=48.589, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=50.707, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=160.551, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=50.356, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=50.356, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=52.382, Time=0.25 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: 0.695 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=92.879, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=91.013, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=91.588, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=94.242, Time=0.01 sec
```

In [14]:
```
outs.head()
```

Out[14]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	14.0	7.0	-0.263137	2.128251	2.458587	-4.039090
1	30.0	7.0	0.100019	0.934656	1.189247	-0.179032
2	60.0	7.0	-0.628039	1.573249	1.815575	-1.747954
3	180.0	7.0	-0.145053	2.002649	2.320378	-3.488471
4	360.0	7.0	0.040634	1.441446	1.796824	-1.691485

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

Out[15]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	30.0	7.0	0.100019	0.934656	1.189247	-0.179032

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

Out[16]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
1	30.0	7.0	0.100019	0.934656	1.189247	-0.179032

In [17]:
```
outs[outs.R2 == outs.R2.min()]
```

Out[17]:

	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
15	30.0	21.0	-7.07047	7.947557	9.274031	-25.278265

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 [18]:
```
sp_500=pd.read_csv('Data/constituents_csv.csv')
```

In [19]:
```
nsdq=pd.read_csv('Data/nasdaq.csv')
```

In [20]:
```
dow_30=pd.read_excel('Data/dow-jones-industrial-average-components.xls')
```

In [21]:
```
sp_500.head()
```

Out[21]:

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

In [22]:
```
sp_500.isna().sum()
```

Out[22]:
```
Symbol    0
Name      0
Sector    0
dtype: int64
```

In [23]:
```
nsdq.head()
```

Out[23]:

	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 [24]:
```
nsdq.isna().sum()
```

Out[24]:
```
Unnamed: 0      0
Symbol          0
Company Name    0
dtype: int64
```

In [25]:
```
dow_30.head()
```

Out[25]:

	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 [26]:
```
dow_30.isna().sum()
```

Out[26]:
```
Company Name     0
Ticker Symbol    0
Weighting %      0
dtype: int64
```

In [27]:
```
dow_30.head()
```

Out[27]:

	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 [28]:
```
new_cols=['Name','Symbol','Weight%']
dow_30.columns=new_cols
```

In [29]:
```
dow_30.head()
```

Out[29]:

	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 [30]:
```
stock = yf.Ticker(sp_500['Symbol'][0])
```

In [31]:
```
stock_df=stock.history(period='max')
```

In [32]:
```
stock_df.iloc[::-1]
```

Out[32]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2021-07-22	201.509995	201.649994	198.679993	199.070007	1700500	0.0	0.0
2021-07-21	201.130005	202.880005	199.960007	200.770004	1999100	0.0	0.0
2021-07-20	198.339996	202.220001	198.070007	200.820007	2783300	0.0	0.0
2021-07-19	197.789993	198.539993	195.110001	197.559998	3146300	0.0	0.0
2021-07-16	203.119995	203.210007	198.910004	199.369995	2474100	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

13005 rows × 7 columns

In [33]:
```
stock_df.isna().sum()
```

Out[33]:
```
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

In [34]:
```
dow_5=dow_30.head()
```

In [35]:
```
dow_5
```

Out[35]:

	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 [36]:
```
cols2=['Symbol','Train_Len','Test_Len','Exp_var','MAE','RMSE','R2']
reslts = pd.DataFrame(columns=cols2)
reslts.reset_index()
```

Out[36]:

index	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2

In [37]:
```
fin_results=pd.DataFrame()
```

In [38]:
```
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 [39]:
```
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']:
    #print(each)
    stock_res_d5 = tt_test(each,trains,tests)
#    print(stock_res[(stock_res.MAE == stock_res.MAE.min()) & (stock_res.Exp_var 
    print (stock_res_d5[(stock_res_d5.MAE == stock_res_d5.MAE.min())])
#    if len(placeh)==1:
 #       fin_results.loc[len(fin_results.index)]=placeh
  #  else:
   #     placeh1=stock_res[(stock_res.MAE == stock_res.MAE.min()) & (stock_res.Ex
    #                     & (stock_res.Train_Len == stock_res.Train_Len.min())
     #                    & (stock_res.Test_Len == stock_res.Test_Len.min())]
      #  fin_results.loc[len(fin_results.index)]=placeh1
```

```
Processing:  yfinance.Ticker object <MMM>
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.27 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=55.134, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=51.552, Time=0.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=52.976, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=54.417, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=53.426, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=53.317, Time=0.05 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=55.277, Time=0.13 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=53.622, Time=0.03 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 0.637 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=134.931, Time=0.66 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=168.982, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=129.450, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=152.510, Time=0.04 sec
```

In [40]:
```
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 [41]:
```
stock_res_d5.head()
```

Out[41]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	yfinance.Ticker object <MMM>	14	7	-1.085215	2.896388	3.145175	-2.161375
1	yfinance.Ticker object <MMM>	30	7	-0.022183	1.628433	1.938849	-0.201363
2	yfinance.Ticker object <MMM>	60	7	0.019558	1.631949	1.972687	-0.243663
3	yfinance.Ticker object <MMM>	180	7	-0.287820	1.754506	2.047308	-0.339532
4	yfinance.Ticker object <MMM>	360	7	0.022251	1.635361	1.987119	-0.261927

In [42]:
```
stock_res_d5.tail()
```

Out[42]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
170	yfinance.Ticker object <CAT>	60	56	0.000000	12.510378	13.121852	-0.023020
171	yfinance.Ticker object <CAT>	180	56	-1.486945	20.851492	26.219280	-3.084467
172	yfinance.Ticker object <CAT>	360	56	-0.621362	16.436480	18.936444	-1.130545
173	yfinance.Ticker object <CAT>	720	56	-0.326448	14.719769	16.158049	-0.551214
174	yfinance.Ticker object <CAT>	900	56	0.000082	12.525372	13.155593	-0.028288

In [43]:
```
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)
```

In [44]:
```
stock_res_d5
```

Out[44]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	MMM	14	7	-1.085215	2.896388	3.145175	-2.161375
1	MMM	30	7	-0.022183	1.628433	1.938849	-0.201363
2	MMM	60	7	0.019558	1.631949	1.972687	-0.243663
3	MMM	180	7	-0.287820	1.754506	2.047308	-0.339532
4	MMM	360	7	0.022251	1.635361	1.987119	-0.261927
...	...	...	...	...	...	...	...
170	CAT	60	56	0.000000	12.510378	13.121852	-0.023020
171	CAT	180	56	-1.486945	20.851492	26.219280	-3.084467
172	CAT	360	56	-0.621362	16.436480	18.936444	-1.130545
173	CAT	720	56	-0.326448	14.719769	16.158049	-0.551214
174	CAT	900	56	0.000082	12.525372	13.155593	-0.028288

In [45]:
```
df_x=pd.DataFrame(columns=stock_res_d5.columns)
```

In [46]:
```
df_x=stock_res_d5.loc[stock_res_d5.index[0:35]]
```

In [47]:
```
df_x.sort_values(by=['MAE']).head()
```

Out[47]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598
13	MMM	900	14	-2.220446e-16	1.455716	1.811044	-0.244598
15	MMM	30	21	5.903764e-01	1.495587	1.899614	0.507038
9	MMM	60	14	-2.193911e-02	1.557796	1.904678	-0.376621

In [48]:
```
df_y=df_x.sort_values(by=['MAE']).head(2)
```

In [49]:
```
df_y
```

Out[49]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598

In [50]:
```
a=0
while a<=141:
    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 [51]:
```
df_y
```

Out[51]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598
66	AXP	180	56	7.995575e-01	2.147050	2.644699	0.799417
35	AXP	14	7	9.636264e-02	2.312994	3.118099	0.086774
71	AMGN	30	7	0.000000e+00	1.304286	1.728579	-1.065180
72	AMGN	60	7	0.000000e+00	1.304286	1.728579	-1.065180
111	AAPL	900	7	-6.982642e-02	1.573754	2.113508	-0.088963
110	AAPL	720	7	-7.617848e-02	1.581981	2.127865	-0.103808
141	CAT	30	7	-4.717259e-01	3.064428	3.609392	-1.030076
159	CAT	720	21	-8.264442e-04	3.197051	3.811401	-0.024560

Ok, so I managed to get some data on train/test splits, but I still need more. So far it looks like 720 and 360 are leading. But let's run through the entire dow 30, and see what that'll get us.

In [52]:
```
df_y['Train_Len'].value_counts()
```

Out[52]:
```
30     4
720    4
14     1
900    1
180    1
60     1
Name: Train_Len, dtype: int64
```

In [53]:
```
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(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=55.134, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=51.552, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=52.976, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=54.417, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=53.426, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=53.317, Time=0.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=55.277, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=53.622, Time=0.01 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 0.259 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=134.931, Time=0.26 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=168.982, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=129.450, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=152.510, Time=0.02 sec
```

In [54]:
```
stock_res_d30.info
```

Out[54]:
```
<bound method DataFrame.info of      Symbol Train_Len Test_Len       Exp_var
MAE      RMSE        R2
0       MMM        14        7 -1.085215e+00  2.896388  3.145175 -2.161375
1       MMM        30        7 -2.218283e-02  1.628433  1.938849 -0.201363
2       MMM        60        7  1.955766e-02  1.631949  1.972687 -0.243663
3       MMM       180        7 -2.878204e-01  1.754506  2.047308 -0.339532
4       MMM       360        7  2.225113e-02  1.635361  1.987119 -0.261927
...     ...       ...      ...           ...       ...       ...       ...
1045    WMT        60       56 -1.021074e-01  1.859599  2.093813 -0.306830
1046    WMT       180       56  1.110223e-16  1.713821  2.396052 -0.711337
1047    WMT       360       56 -1.500608e-04  1.569951  2.186984 -0.425720
1048    WMT       720       56 -6.651733e-01  3.652480  4.348661 -4.637079
1049    WMT       900       56 -5.375263e-01  3.416385  4.097732 -4.005302

[1050 rows x 7 columns]>
```

In [55]:
```
len(stock_res_d30)
```

Out[55]:
```
1050
```

In [56]:
```
df_y_d30=pd.DataFrame(columns=stock_res_d30.columns)
```

In [57]:
```
df_y_d30
```

Out[57]:

Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2

In [58]:
```
a=2
b=len(stock_res_d30)
while a<=(b-35):
    df_x=stock_res_d30.loc[stock_res_d30.index[a:(a+35)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_d30=pd.concat([df_y,df_ph])
    a+=35
```

In [59]:
```
df_y_d30
```

Out[59]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598
8	MMM	30	14	-2.220446e-16	1.455716	1.811044	-0.244598
12	MMM	720	14	-2.220446e-16	1.455716	1.811044	-0.244598
66	AXP	180	56	7.995575e-01	2.147050	2.644699	0.799417
35	AXP	14	7	9.636264e-02	2.312994	3.118099	0.086774
71	AMGN	30	7	0.000000e+00	1.304286	1.728579	-1.065180
72	AMGN	60	7	0.000000e+00	1.304286	1.728579	-1.065180
111	AAPL	900	7	-6.982642e-02	1.573754	2.113508	-0.088963
110	AAPL	720	7	-7.617848e-02	1.581981	2.127865	-0.103808
141	CAT	30	7	-4.717259e-01	3.064428	3.609392	-1.030076
159	CAT	720	21	-8.264442e-04	3.197051	3.811401	-0.024560
1015	WMT	14	7	-7.970009e+00	0.759850	0.959125	-15.173741
982	WBA	60	7	0.000000e+00	1.041428	1.118359	-6.527480
983	WBA	180	7	0.000000e+00	1.041428	1.118359	-6.527480

In [60]:
```
df_y['Train_Len'].value_counts()
```

Out[60]:
```
30     4
720    4
14     1
900    1
180    1
60     1
Name: Train_Len, dtype: int64
```

In [61]:
```
df_y['Test_Len'].value_counts()
```

Out[61]:
```
7     6
14    4
21    1
56    1
Name: Test_Len, dtype: int64
```

So we have a clear winner in the Test Length. The Training Length is still a little close, as far as the best; although the worst seems pretty clear. 30 days seems to be a bad number to use.

Next on the agenda is to run through this modeling again, but with 50 stocks from the S&P 500

In [62]:
```
picks_sp50=[]
for i in range (0,50):
    x = random.randint(0,len(sp_500))
    picks_sp50.append(x)
```

In [63]:
```
tests=[7, 14, 21, 28]
```

In [64]:
```
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)
```

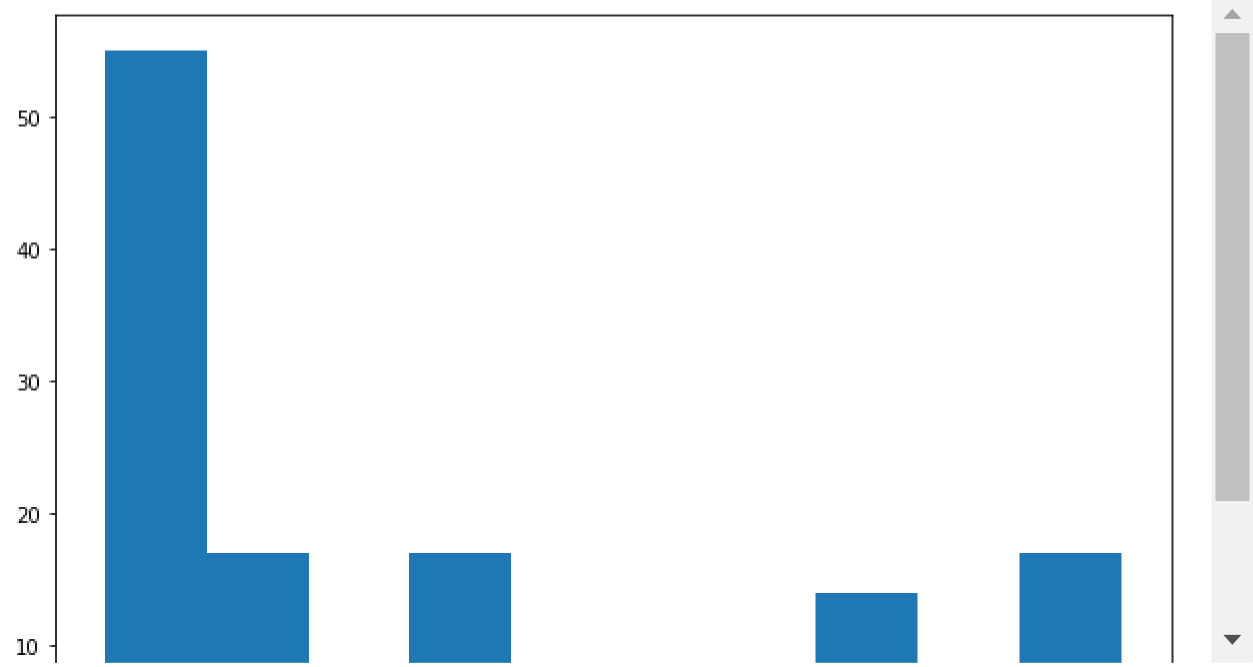
```
Processing:  yfinance.Ticker object <PENN>
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.25 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=46.062, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=48.005, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.08 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=45.823, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.09 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.455 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.22 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=112.002, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=113.937, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=113.859, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=111.506, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=115.539, Time=0.03 sec
```

In [65]:
```
a=0
b=len(stock_res_sp50)
while a<=(b-35):
    df_x=stock_res_sp50.loc[stock_res_sp50.index[a:(a+35)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_sp50=pd.concat([df_y_sp50,df_ph])
    a+=35
```

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

Out[66]:
```
30     20
60     19
360    17
900    17
180    17
14     16
720    14
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     54
14    26
28    20
21    20
Name: Test_Len, dtype: int64
```

So, a pretty clear winner in the test value.

Again, slight edge to 60 days 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 [72]:
```
nsdq.loc[nsdq['Symbol']=='AFH']
```

Out[72]:

Unnamed: 0	Symbol	Company Name

In [86]:
```
#nsdq = nsdq.drop(labels=[78], axis=0)
```

2970 stocks is WAY too many 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.

In [113]:
```
#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)
```

```
- AAIT: No data found for this date range, symbol may be delisted
- AAVL: No data found for this date range, symbol may be delisted
- ABAC: No data found for this date range, symbol may be delisted
- ABAX: No data found for this date range, symbol may be delisted
- ABCD: No data found for this date range, symbol may be delisted
- ABCO: No data found for this date range, symbol may be delisted
- ABCW: No data found for this date range, symbol may be delisted
- ABDC: No data found, symbol may be delisted
- ABGB: No data found for this date range, symbol may be delisted
- ABTL: No data found for this date range, symbol may be delisted
- ABY: No data found for this date range, symbol may be delisted
- ACAS: No data found for this date range, symbol may be delisted
- ACAT: No data found for this date range, symbol may be delisted
- ACFC: No data found for this date range, symbol may be delisted
- ACHN: No data found, symbol may be delisted
- ACPW: No data found for this date range, symbol may be delisted
- ACSF: No data found, symbol may be delisted
- ACTA: No data found for this date range, symbol may be delisted
- ACTS: No data found for this date range, symbol may be delisted
```

In [114]:
```
#len(no_data)
```

Out[114]: 1265

In [115]:
```
#no_data[4:8]
```

Out[115]: ['ABCD', 'ABCO', 'ABCW', 'ABDC']

In [116]:
```
#y=nsdq.loc[nsdq['Symbol']==no_data[4]].index
```

In [117]:
```
#y[0]
```

Out[117]: 12

In [118]:
```
#nd_index=[]
#for each in no_data:
#    y=nsdq.loc[nsdq['Symbol']==each].index
#    nd_index.append(y[0])
```

In [119]:
```
#len(nd_index)
```

Out[119]: 1265

In [120]:
```
#nsdq = nsdq.drop(labels=nd_index, axis=0)
```

In [73]:
```
nsdq.reset_index(drop=True,inplace=True)
```

In [127]:
```
#nsdq.to_csv('Data/nasdaq.csv',index=True)
```

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

In [74]:
```
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)
```

```
- ALXN: No data found for this date range, symbol may be delisted
- BRK.B: No data found, symbol may be delisted
- BF.B: No data found for this date range, symbol may be delisted
```

In [75]:
```
no_data_sp
```

Out[75]:
```
['ALXN', 'BRK.B', 'BF.B']
```

In [76]:
```
sp_500.loc[sp_500['Symbol']==no_data_sp[1]].index
```

Out[76]:
```
Int64Index([64], dtype='int64')
```

In [77]:
```
sp_500.loc[sp_500['Symbol']==no_data_sp[0]].index
```

Out[77]:
```
Int64Index([18], dtype='int64')
```

In [78]:
```
sp_500.loc[sp_500['Symbol']==no_data_sp[2]].index
```

Out[78]:
```
Int64Index([78], dtype='int64')
```

In [79]:
```
sp_500 = sp_500.drop(labels=[18,64,78], axis=0)
```

In [80]:
```
sp_500.to_csv('Data/constituents_csv.csv',index=True)
```

In [81]:
```
sp_500.reset_index(drop=True,inplace=True)
```

In [82]:
```
sp_500.head()
```

Out[82]:

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

In [83]:
```
picks=[]
for i in range (0,25):
    x = random.randint(0,(len(nsdq)-1))
    picks.append(x)
```

In [84]:
```
nsdq.reset_index(drop=True,inplace=True)
```

In [85]:
```
nsdq.head()
```

Out[85]:

	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 [87]:
```
nsdq.drop(columns='Unnamed: 0',inplace=True)
```

In [88]:
```
len(nsdq)
```

Out[88]: 1701

In [89]:
```
tests=[7,14,21,28]
```

In [90]:
```
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=df_x.sort_values(by=['MAE']).head(3)
```

```
Processing:  yfinance.Ticker object <MCBK>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=7.788, Time=0.23 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=11.190, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=6.017, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=6.392, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=139.321, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=6.104, Time=0.11 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=7.038, Time=0.20 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=9.345, Time=0.18 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: 0.804 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.19 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=33.118, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=34.847, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=34.567, Time=0.02 sec
```

In [91]:
```
1400/25
```

Out[91]: 28.0

In [92]:
```
a=0
b=len(stock_res_nd50)
while a<=(b-28):
    df_x=stock_res_nd50.loc[stock_res_nd50.index[a:(a+28)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_nd50=pd.concat([df_y_nd50,df_ph])
    a+=28
```

In [93]:
```
stock_res_nd50.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 700 entries, 0 to 699
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Symbol     700 non-null    object 
 1   Train_Len  700 non-null    object 
 2   Test_Len   700 non-null    object 
 3   Exp_var    700 non-null    float64
 4   MAE        700 non-null    float64
 5   RMSE       700 non-null    float64
 6   R2         700 non-null    float64
dtypes: float64(4), object(3)
memory usage: 63.8+ KB
```

In [94]:
```
df_y_nd50['Test_Len'].value_counts()
```

Out[94]:
```
7     34
14    21
21    14
28     6
Name: Test_Len, dtype: int64
```

In [95]:
```
df_y_nd50['Train_Len'].value_counts()
```

Out[95]:
```
180    20
720    12
14     11
900    11
360     9
60      8
30      4
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 [96]:
```
tests=[7]
```

In [97]:
```
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(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.16 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=57.175, Time=0.00 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=53.265, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=57.047, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=54.973, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=54.915, Time=0.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.12 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=55.996, Time=0.01 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 0.429 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=136.682, Time=0.12 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=168.781, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=132.206, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=153.099, Time=0.02 sec
```

In [98]:
```
stock_res_d30.head(14)
```

Out[98]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
0	MMM	14	7	-1.527278	4.522738	4.903086	-10.054422
1	MMM	30	7	0.109295	1.742471	2.200984	-1.227567
2	MMM	60	7	0.122077	1.268467	1.807821	-0.502823
3	MMM	180	7	-0.016990	3.115064	3.451853	-4.479005
4	MMM	360	7	0.108944	2.044693	2.433342	-1.722723
5	MMM	720	7	0.000000	2.830715	3.191812	-3.684591
6	MMM	900	7	0.000000	2.830715	3.191812	-3.684591
7	AXP	14	7	-0.077585	2.716503	3.525871	-0.077609
8	AXP	30	7	0.000000	2.700006	4.102008	-0.458550
9	AXP	60	7	0.082892	4.184591	5.300090	-1.434977
10	AXP	180	7	-0.153000	4.722948	5.967223	-2.086548
11	AXP	360	7	0.000000	2.700006	4.102008	-0.458550
12	AXP	720	7	0.016514	2.775771	3.966898	-0.364051
13	AXP	900	7	0.018703	2.752636	3.931237	-0.339637

In [99]:
```
len(stock_res_d30)
```

Out[99]: 210

In [100]:
```
df_y_d30=pd.DataFrame(columns=stock_res_d30.columns)
```

In [101]:
```
a=0
b=len(stock_res_d30)
while a<=(b-7):
    df_x=stock_res_d30.loc[stock_res_d30.index[a:(a+7)]]
    df_ph=df_x.sort_values(by=['MAE']).head(3)
    df_y_d30=pd.concat([df_y_d30,df_ph])
    a+=7
```

In [102]:
```
df_y_d30
```

Out[102]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
2	MMM	60	7	1.220771e-01	1.268467	1.807821	-0.502823
1	MMM	30	7	1.092953e-01	1.742471	2.200984	-1.227567
4	MMM	360	7	1.089438e-01	2.044693	2.433342	-1.722723
8	AXP	30	7	0.000000e+00	2.700006	4.102008	-0.458550
11	AXP	360	7	0.000000e+00	2.700006	4.102008	-0.458550
...	...	...	...	...	...	...	...
197	WBA	30	7	0.000000e+00	0.645244	0.735737	-3.331504
198	WBA	60	7	0.000000e+00	0.645244	0.735737	-3.331504
206	WMT	180	7	3.330669e-16	0.323571	0.405996	-0.011650
209	WMT	900	7	-2.427866e-03	0.344178	0.440091	-0.188697
208	WMT	720	7	-2.960771e-03	0.346331	0.445349	-0.217269

In [103]:
```
df_y_d30['Train_Len'].value_counts()
```

Out[103]:
```
360    18
720    18
30     14
14     12
900    12
180    11
60      5
Name: Train_Len, dtype: int64
```

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

In [104]:
```
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 <PEP>
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.26 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=39.801, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=41.655, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=41.604, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=43.127, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=43.570, Time=0.03 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.353 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.22 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=96.344, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=97.702, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=97.686, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=95.647, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=99.685, Time=0.03 sec
```

In [105]:
```
len(stock_res_sp50)
```

Out[105]: 350

In [106]:
```
c=(len(stock_res_sp50))/(len(picks_sp50))
```

In [107]:
```
c=int(c)
```

In [108]:
```
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 [109]:
```
df_y_sp50.head()
```

Out[109]:

	Symbol	Train_Len	Test_Len	Exp_var	MAE	RMSE	R2
4	PEP	360	7	-0.035109	1.217324	1.431944	-2.732871
6	PEP	900	7	-0.017788	1.237815	1.446118	-2.807136
5	PEP	720	7	-0.015867	1.244177	1.451203	-2.833958
11	GL	360	7	0.080291	0.847538	1.152617	0.028960
12	GL	720	7	0.075792	0.853535	1.168060	0.002767

In [110]:
```
df_y_sp50['Train_Len'].value_counts()
```

Out[110]:
```
360    28
720    26
30     24
900    23
180    22
14     14
60     13
Name: Train_Len, dtype: int64
```

In [111]:
```
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)
```

```
MCBK
Processing:  yfinance.Ticker object <MCBK>
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=7.788, Time=0.24 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=11.190, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=6.017, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=6.392, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=139.321, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=6.104, Time=0.12 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=7.038, Time=0.19 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=9.345, Time=0.19 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=inf, Time=0.04 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.824 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=33.118, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=34.847, Time=0.02 sec
```

In [112]:
```
c=int((len(stock_res_nd50))/(len(picks)))
```

In [113]:
```
a=0
b=len(stock_res_nd50)
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]:
```
180    14
900    13
720    13
360    12
60      9
14      8
30      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']
```

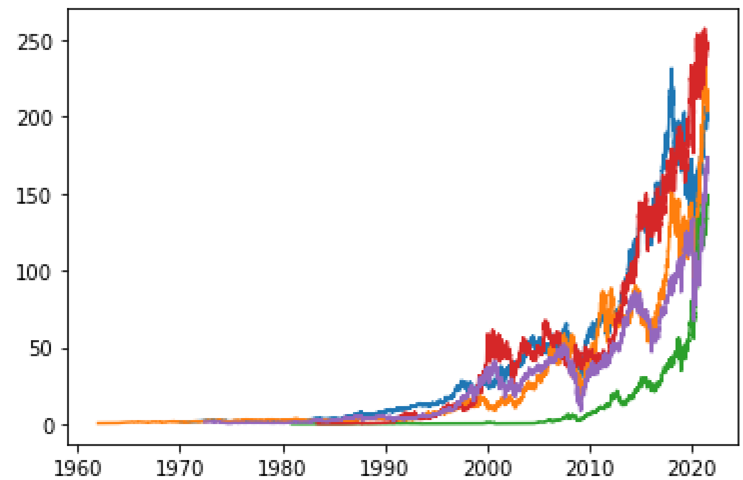
In [118]:
```
df1=yf.Ticker('AMGN')
df=df1.history(period="max")
AMGN=df['Close']
```

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 0x26630faab00>]
```



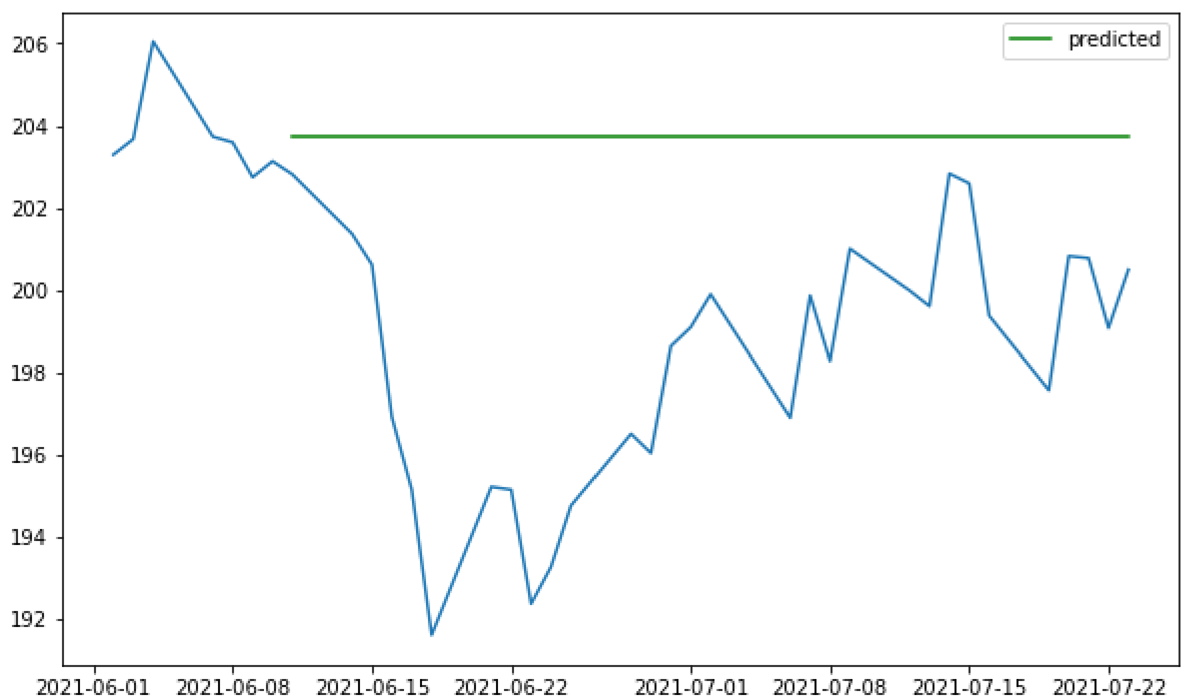
In [122]:
```
MMM=MMM.tail(37)
AXP=AXP.tail(37)
AAPL=AAPL.tail(37)
AMGN=AMGN.tail(37)
CAT=CAT.tail(37)
```

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

In [124]:
```
train, test = temporal_train_test_split(MMM, test_size=30)
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(1,0,1)(0,0,0)[0] intercept   : AIC=27.799, Time=0.08 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=23.803, Time=0.06 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=25.751, Time=0.06 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=25.732, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=96.301, Time=0.00 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.229 seconds
```

In [125]:
```
plt.figure(figsize=(10,6))
plt.plot(MMM)
plt.plot(test.index, y_pred, color='green', label = 'predicted')
plt.legend()
plt.show()
```



In [126]:
```
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=136.682, Time=0.11 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=168.781, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=132.206, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=153.099, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=404.785, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=133.502, Time=0.12 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=133.728, Time=0.14 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=134.969, Time=0.14 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: 0.596 seconds
```

These results are all pretty terrible. In the next notebook, we'll try again with Prophet, Facebook's time forecasting library.

In [ ]: