portfolio_optimizer

September 17, 2017

0.0.1 Import all required packages

```
In [1]: from pandas_datareader import data
        import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        import pickle
        import copy
        import json
        import os
        import glob
        import time
        import datetime
        from IPython.display import display                # Allows the use of display() for 

        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score, fbeta_score, make_scorer, f1_score
        from sklearn.naive_bayes import GaussianNB
        from sklearn import cross_validation
        from sklearn.svm import SVC
        from sklearn.ensemble import AdaBoostClassifier
        from sklearn.grid_search import GridSearchCV

        from keras.models import Sequential
        from keras.layers import Dense
        from keras.models import model_from_json
        from keras.utils import np_utils
        from sklearn.preprocessing import LabelEncoder
        from keras import optimizers
        from keras import regularizers

        import multiprocessing as mp

        # Pretty display for notebooks
        %matplotlib inline
```

```
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/cross_validation.py:44: DeprecationWar
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/grid_search.py:43: DeprecationWarning:
  DeprecationWarning)
Using TensorFlow backend.
```

0.0.2 Updateable Parameters

```
In [2]: ## Start time.  Used to measure execution time.
        START = time.clock()

        ## Yahoo's API has changed, so we'll use google as our source
        #DATA_SOURCE = 'google'
        DATA_SOURCE = 'yahoo'

        ## Date range used for training data
        #TRAINING_START_DATE = '2010-01-01'
        #TRAINING_END_DATE = '2017-7-31'
        TRAINING_START_DATE = datetime.datetime(2014, 4, 1)
        TRAINING_END_DATE = datetime.datetime(2016, 5, 31)

        ## Date range used for testing data
        #TEST_START_DATE = '2016-06-01'
        #TEST_END_DATE = '2017-7-31'
        TEST_START_DATE = datetime.datetime(2016, 6, 1)
        TEST_END_DATE = datetime.datetime(2017, 7, 31)

        # ## Stock tickers for training data
        TRAINING_TICKERS = ['AAPL','GOOG','T','IMAX','IBM','NFLX','SIRI','S','PLUG','C',\
                            'ZNGA','WMS','BAC','AMZN','FB','P','WM','NOK','DDD','XME','XONE','SI
                            'TSLA','SSYS','TXN','F','GS','LQMT','HTZ','BAH','GLW','SPWR',\
                            'BIDU','SRPT','YGE','CNX','URRE','VJET','RAD','NQ',\
                            'KORS','TWTR','HLF','ORCL','WLL','BLDP','PEG','MJNA','CBIS',\
                            'TM','SBUX','MBLY','MRK','DBO','PFE','CAMP','TRXC',\
                            'BMY','FE','VTR','UHT','MVO','KF','RACE','STOR','MU','RTN']
        # TRAINING_TICKERS = ['AAPL', 'GOOG', 'YHOO', 'T', 'IMAX', 'IBM', 'NFLX', 'SIRI', 'S',
        #             'C', 'BAC', 'P', 'NOK', 'XONE', 'SSYS', 'TSLA', 'AMZN', 'SDRL', 'DDD', \
        #             'DBO', 'SRPT', 'SPWR', 'SCTY', 'FB', 'URRE', 'NQ', 'TWTR', 'F', 'BAH', \
        #             'MZDAY','FSYS', 'BIDU', 'KORS', 'HLF', 'ORCL', 'MBLY']
                   ## SOME NETWORK CAUSES MZDAY AND FSYS TO HAVE SOME NaN FOR SOME REASON

        ## Stock tickers for testing data
        ## unable to read SHOP,HEMP,LMT
        # TESTING_TICKERS = ['BA','OLED', 'HON','MA','TPLM', 'SD', 'FCEL', 'CHK', 'CMG','UHT',
        #                      'UHT','BMY','FE','VTR','UHT','MVO','KF','RACE','MU','RTN']
        #TESTING_TICKERS = ['BA','OLED', 'HON','MA','TPLM', 'SD', 'FCEL', 'CHK', 'CMG']
        TESTING_TICKERS = ['BA', 'HON','MA','TPLM', 'SD', 'FCEL', 'CHK', 'CMG']
        #TESTING_TICKERS = ['BA','OLED', 'HON','MA']
        #TESTING_TICKERS = ['SD', 'FCEL', 'CHK', 'CMG']
```

```
## Initial money to be invested
MONEY = 10000

## Commision rate when buying/selling stocks
COMM_RATE = 4.95

## Long term capital gain tax rate (percentage)
GAIN_LONG = 0.15

## Short term capital gain tax rate (percentage).
## Also used for losses, assuming its the individuals tax bracket
GAIN_SHORT = 0.25

## Models predict the ratio base on these targets
#TARGET_RATIOS = ['vr15','vr25','vr40']
TARGET_RATIOS = ['vClose','vr2','vr3','vr5','vr10','vr15','vr25','vr40']
## New target ratio, will not just be relative to close
#TARGET_RATIOS = ['Close_pc','r2_p2','r3_p3','r5_p5','r10_p10','r15_p15','r25_p25','r4

## Used to determine when predictions will be a buy/sell
#SELL_BUY_VALUES = [(1,1),(0.99,1.01),(0.995,1.005)]
SELL_BUY_VALUES = [(0.985,0.985),(0.99,0.99),(0.995,0.995), (1,1),(1.005,1.005),\
                   (1.01,1.01),(1.015,1.015), \
                   (0.985,1.015),(0.99,1.01),(0.995,1.005),\
                   (0.995,1.00),(1.00,1.005),(1.005,1.01), \
                   (1.00,1.015),(1.00,1.01)]

## Set to True if we are reading existing models
## Set to False to generate new models
READ_EXISTING_MODELS = True

## Set True if using multiprocessing
MULTIPROCESSOR = True

## Number of processes for multiprocessing pool
NUM_PROCESSES = 8

## Set True to read existing training stocks
## If training tickers have been updated, set this to False
READ_EXISTING_TRAINING_STOCKS = True

## Set True to read existing testing stocks
## If testing tickers have been updated, set this to False
READ_EXISTING_TESTING_STOCKS = True
```

0.0.3 Get all the stocks data. Save data as pickle files

```
In [3]: def gather_training_data():
            all_weekdays = pd.date_range(start=TRAINING_START_DATE,end=TRAINING_END_DATE,freq=
            ## panel type
            panel_data = data.DataReader(TRAINING_TICKERS,DATA_SOURCE,TRAINING_START_DATE,\
                                         TRAINING_END_DATE)

            if DATA_SOURCE == 'yahoo':
                ## Yahoo has extra column
                panel_data.drop('Adj Close', inplace=True)

            panel_data.drop('Volume', inplace=True)

            ## save to pickle
            panel_data.to_pickle('training_stocks.pkl')

        def gather_testing_data():
            all_weekdays = pd.date_range(start=TEST_START_DATE,end=TEST_END_DATE,freq='B')
            panel_data = data.DataReader(TESTING_TICKERS,DATA_SOURCE,TEST_START_DATE,TEST_END_I

            if DATA_SOURCE == 'yahoo':
                ## Yahoo has extra column
                panel_data.drop('Adj Close', inplace=True)

            panel_data.drop('Volume', inplace=True)

            panel_data.to_pickle('testing_stocks.pkl')

        ## Read existing data
        if READ_EXISTING_TRAINING_STOCKS == False:
            gather_training_data()

        if READ_EXISTING_TESTING_STOCKS == False:
            gather_testing_data()
```

0.0.4 Helper function to get the rolling averages. 2, 3, 5, 10, 15, 25 and 40-day moving averages.

```
In [4]: def get_rolling(df):
            stock = df['Close']
            r2 = stock.rolling(window=2).mean()
            r3 = stock.rolling(window=3).mean()
            r5 = stock.rolling(window=5).mean()
            r10 = stock.rolling(window=10).mean()
            r15 = stock.rolling(window=15).mean()
            r25 = stock.rolling(window=25).mean()
            r40 = stock.rolling(window=40).mean()
```

```
        return r2, r3, r5, r10, r15, r25, r40
```

0.0.5 Helper function to plot stocks data, with rolling averages

```
In [5]: def plot_stock(tick, df):
            print("Plotting: ", tick)
            close = df['vClose']
            r2 = df['vr2']
            r3 = df['vr3']
            r5 = df['vr5']
            r10 = df['vr10']
            r15 = df['vr15']
            r25 = df['vr25']
            r40 = df['vr40']

            fig = plt.figure()
            ax = fig.add_subplot(1,1,1)
            ax.plot(close.index,close,label=tick)
            ax.plot(r2.index, r2, label='2 days rolling')
            ax.plot(r3.index, r3, label='3 days rolling')
            ax.plot(r5.index, r5, label='5 days rolling')
            ax.plot(r10.index, r10, label='10 days rolling')
            ax.plot(r15.index, r15, label='15 days rolling')
            ax.plot(r25.index, r25, label='25 days rolling')
            ax.plot(r40.index, r40, label='40 days rolling')
            ax.set_xlabel('Date')
            ax.set_ylabel('Closing prices ($)')
            ax.legend()

            #plt.show()
            #fig.savefig("figures/"+tick+'.png')
            fig.savefig("figures/"+tick+'.svg', format='svg', dpi=1200)
            plt.close(fig)
```

0.0.6 Helper function to statistics of the training data

```
In [6]: def describe_data():
            df = pd.DataFrame.from_csv("training_data/all_raw_data.csv")
            df = df.describe()
            df.to_csv("training_data/all_raw_data_description.csv")
            print("\nRaw training data description:\n",df)

            df = pd.DataFrame.from_csv("training_data/all_processed_data.csv")
            df = df.describe()
            df.to_csv("training_data/all_processed_data_description.csv")
            print("\nProcessed training data description:\n",df)
```

0.0.7 Get each DataFrame of the entire list of stocks for training, from the pickle file

```
In [7]: def get_training_stocks_df(filename):
            processed_df = pd.DataFrame()
            raw_inputs_df = pd.DataFrame()
            raws = []
            processed = []
            panel_data = pd.read_pickle(filename)                          ## read saved stocks d

            for tick in TRAINING_TICKERS:
                ## Extract single stock from panel_data
                df = panel_data[:,:,tick]                                  ## becomes df type, fr
                df.to_csv("training_data/"+tick+".csv")                    ## raw input

                raws.append(df)                                            ## gather all the raw

                ## Get full df with 109 columns
                df = get_stock_df(df,tick)

                ## Save df
                df.to_csv("training_data/"+tick+"_processed.csv")

                ## Plot stock
                plot_stock(tick,df)

                #processed_df = processed_dfmain_df.append(df)
                processed.append(df)                               ## faster to append once, with

            ## append all raw input to one df, then save
            raw_inputs_df = raw_inputs_df.append(raws)
            raw_inputs_df.to_csv("training_data/all_raw_data.csv")

            ## append all processed input to on df, then save
            processed_df = processed_df.append(processed)          ## faster to append once, with
            processed_df.to_csv("training_data/all_processed_data.csv")

            return processed_df
```

0.0.8 Helper function to get DataFrame of individual stocks. Generates 108 columns, 20 columns will be removed later

```
In [8]: """
        Columns generated:

        #predict, prev_close, prev_r2, prev_r3, prev_r5, prev_r10, prev_r15, prev_r25, prev_r4
        predict, prev_close, prev_r2, prev_r3, prev_r5, prev_r10, prev_r15, prev_r25, prev_r40
        vOpen, vHigh, vLow, vClose, vr2, vr3, vr5, vr10, vr15, vr25, vr40,
        Open_pc, High_pc, Low_pc, Close_pc, r2_pc, r3_pc, r5_pc, r10_pc, r15_pc, r25_pc, r40_p
```

```
Open_p2, High_p2, Low_p2, Close_p2, r2_p2, r3_p2, r5_p2, r10_p2, r15_p2, r25_p2, r40_p
Open_p3, High_p3, Low_p3, Close_p3, r2_p3, r3_p3, r5_p3, r10_p3, r15_p3, r25_p3, r40_p
Open_p5, High_p5, Low_p5, Close_p5, r2_p5, r3_p5, r5_p5, r10_p5, r15_p5, r25_p5, r40_p
Open_p10, High_p10, Low_p10, Close_p10, r2_p10, r3_p10, r5_p10, r10_p10, r15_p10, r25_
Open_p15, High_p15, Low_p15, Close_p15, r2_p15, r3_p15, r5_p15, r10_p15, r15_p15, r25_
Open_p25, High_p25, Low_p25, Close_p25, r2_p25, r3_p25, r5_p25, r10_p25, r15_p25, r25_
Open_p40, High_p40, Low_p40, Close_p40, r2_p40, r3_p40, r5_p40, r10_p40, r15_p40, r25_

"""
def get_stock_df(df,tick):
    ## Yahoo and Google data source returns different orders, ## make sure its this or
    #df = df.reindex_axis(['Open','High','Low','Close','Volume'], axis=1)
    df = df.reindex_axis(['Open','High','Low','Close'], axis=1)

    r2, r3, r5, r10, r15, r25, r40 = get_rolling(df)        ## get moving averages

    ## Rename Volume column then remove it
#     vol = df['Volume']
#     vol = vol.to_frame()
#     vol.columns = ['Vol']
#     df = df.drop('Volume', 1)                              ## 1 for axis 1, which i

    ## Rename columns
    df_r2= r2.to_frame()                                    ## from series to df
    df_r2.columns = ['r2']                                  ## change column title

    df_r3= r3.to_frame()
    df_r3.columns = ['r3']

    df_r5= r5.to_frame()
    df_r5.columns = ['r5']

    df_r10= r10.to_frame()
    df_r10.columns = ['r10']

    df_r15= r15.to_frame()
    df_r15.columns = ['r15']

    df_r25= r25.to_frame()
    df_r25.columns = ['r25']

    df_r40= r40.to_frame()
    df_r40.columns = ['r40']

    ## Shift rows to generate next/previous values
    #predict = df['Close'].copy()
    predict = df['Close'].shift(-1)                                          ## a seri
    predict = predict.to_frame()
```

```
    predict.columns = ['predict']

    prev_close = df['Close'].shift(1)
    prev_close = prev_close.to_frame()
    prev_close.columns = ['prev_close']

    prev_r2 = df_r2['r2'].shift(1)
    prev_r2 = prev_r2.to_frame()
    prev_r2.columns = ['prev_r2']

    prev_r3 = df_r3['r3'].shift(1)
    prev_r3 = prev_r3.to_frame()
    prev_r3.columns = ['prev_r3']

    prev_r5 = df_r5['r5'].shift(1)
    prev_r5 = prev_r5.to_frame()
    prev_r5.columns = ['prev_r5']

    prev_r10 = df_r10['r10'].shift(1)
    prev_r10 = prev_r10.to_frame()
    prev_r10.columns = ['prev_r10']

    prev_r15 = df_r15['r15'].shift(1)
    prev_r15 = prev_r15.to_frame()
    prev_r15.columns = ['prev_r15']

    prev_r25 = df_r25['r25'].shift(1)
    prev_r25 = prev_r25.to_frame()
    prev_r25.columns = ['prev_r25']

    prev_r40 = df_r40['r40'].shift(1)
    prev_r40 = prev_r40.to_frame()
    prev_r40.columns = ['prev_r40']

    ## Generate entire dataframe
    ## encapsulate in a list for multiple df
#     df1 = predict.join([prev_close,prev_r2,prev_r3,prev_r5,prev_r10,prev_r15,\
#                           prev_r25,prev_r40,vol])
    df1 = predict.join([prev_close,prev_r2,prev_r3,prev_r5,prev_r10,prev_r15,\
                        prev_r25,prev_r40])
    df2 = df.join([df_r2,df_r3,df_r5,df_r10,df_r15, df_r25, df_r40])
    df3 = df2.copy()
    df4 = df2.copy()
    df5 = df2.copy()
    df6 = df2.copy()
    df6 = df2.copy()
    df7 = df2.copy()
    df8 = df2.copy()
```

```
df9 = df2.copy()
df10 = df2.copy()
## will have original value (not percentage)
df10.columns = ['vOpen','vHigh','vLow','vClose','vr2','vr3',\
                'vr5','vr10','vr15','vr25','vr40']
## will be with respect to prev_close
df2.columns = ['Open_pc','High_pc','Low_pc','Close_pc','r2_pc','r3_pc','r5_pc',\
               'r10_pc','r15_pc','r25_pc','r40_pc']
## will be with respect to prev_r2
df3.columns = ['Open_p2','High_p2','Low_p2','Close_p2','r2_p2','r3_p2','r5_p2',\
               'r10_p2','r15_p2','r25_p2','r40_p2']
## will be with respect to prev_r3
df4.columns = ['Open_p3','High_p3','Low_p3','Close_p3','r2_p3','r3_p3','r5_p3',\
               'r10_p3','r15_p3','r25_p3','r40_p3']
## will be with respect to prev_r5
df5.columns = ['Open_p5','High_p5','Low_p5','Close_p5','r2_p5','r3_p5','r5_p5',\
               'r10_p5','r15_p5','r25_p5','r40_p5']
## will be with respect to prev_r10
df6.columns = ['Open_p10','High_p10','Low_p10','Close_p10','r2_p10','r3_p10',\
               'r5_p10','r10_p10','r15_p10','r25_p10','r40_p10']
## will be with respect to prev_r15
df7.columns = ['Open_p15','High_p15','Low_p15','Close_p15','r2_p15','r3_p15',\
               'r5_p15','r10_p15','r15_p15','r25_p15','r40_p15']
## will be with respect to prev_r25
df8.columns = ['Open_p25','High_p25','Low_p25','Close_p25','r2_p25','r3_p25',\
               'r5_p25','r10_p25','r15_p25','r25_p25','r40_p25']
## will be with respect to prev_r40
df9.columns = ['Open_p40','High_p40','Low_p40','Close_p40','r2_p40','r3_p40',\
               'r5_p40','r10_p40','r15_p40','r25_p40','r40_p40']

## Combine all to one dataframe
df = df1.join([df10,df2,df3,df4,df5,df6,df7,df8,df9])

## Drop N/A
df = df.dropna(axis=0,how='any')                    ## drop rows containing at lea

## Normalize columns, base on previous values.  Get ratios/percentage
#df[columns_to_divide] = df[columns_to_divide] / df['prev_close']   ## having size

cols_to_divide = ['Open_pc','High_pc','Low_pc','Close_pc','r2_pc','r3_pc','r5_pc',
                  'r10_pc','r15_pc','r25_pc','r40_pc']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_close'].values,axis=0)

cols_to_divide = ['Open_p2','High_p2','Low_p2','Close_p2','r2_p2','r3_p2','r5_p2',
                  'r10_p2','r15_p2','r25_p2','r40_p2']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r2'].values,axis=0)

cols_to_divide = ['Open_p3','High_p3','Low_p3','Close_p3','r2_p3','r3_p3','r5_p3',
```

```
                     'r10_p3','r15_p3','r25_p3','r40_p3']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r3'].values,axis=0)

cols_to_divide = ['Open_p5','High_p5','Low_p5','Close_p5','r2_p5','r3_p5','r5_p5',\
                     'r10_p5','r15_p5','r25_p5','r40_p5']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r5'].values,axis=0)

cols_to_divide = ['Open_p10','High_p10','Low_p10','Close_p10','r2_p10','r3_p10',\
                     'r5_p10','r10_p10','r15_p10','r25_p10','r40_p10']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r10'].values,axis=0)

cols_to_divide = ['Open_p15','High_p15','Low_p15','Close_p15','r2_p15','r3_p15',\
                     'r5_p15','r10_p15','r15_p15','r25_p15','r40_p15']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r15'].values,axis=0)

cols_to_divide = ['Open_p25','High_p25','Low_p25','Close_p25','r2_p25','r3_p25',\
                     'r5_p25','r10_p25','r15_p25','r25_p25','r40_p25']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r25'].values,axis=0)

cols_to_divide = ['Open_p40','High_p40','Low_p40','Close_p40','r2_p40','r3_p40',\
                     'r5_p40','r10_p40','r15_p40','r25_p40','r40_p40']
df[cols_to_divide] = df[cols_to_divide].div(df['prev_r40'].values,axis=0)

return df
```

0.0.9 Update target, base on target ratio

```
In [9]: def get_target(X,target_ratio):
            ## update predict base on target_ratio
            ## All these are base on predict (relative to close only)
            cols_to_divide = ['predict']
            X[cols_to_divide] = X[cols_to_divide].div(X[target_ratio].values,axis=0)

            ## New targets: prev target_ratio/current target ratio
            ## Shift rows to generate next values
            #target = X[target_ratio].shift(-1)
            ##target = X[target_ratio].copy()
            #X['predict'] = target
            ### Drop N/A
            #X = X.dropna(axis=0,how='any')

            ## Convert target base on sell/buy prices thats provided
            y = convert_target_value(X['predict'],sell_below,buy_above)

            ## Delete unneccesary columns
            del X['predict'],X['prev_close'],X['prev_r2'],X['prev_r3'],X['prev_r5']
            del X['prev_r10'],X['prev_r15'],X['prev_r25'],X['prev_r40']
            ## Delete columns with actual price
```

```
        del X['vOpen'],X['vHigh'],X['vLow'],X['vClose'],X['vr2']
        del X['vr3'],X['vr5'],X['vr10'],X['vr15'],X['vr25'],X['vr40']

        return X, y
```

0.0.10 Helper function to convert target. Buy (1), Sell (-1), or Neutral (0). Base on sell/buy prices

```
In [10]: def convert_target_value(arr,sell_below,buy_above):
             ans = []

             for x in arr:
                 if x <= sell_below:
                     ans.append(-1)
                 elif x >= buy_above:
                     ans.append(1)
                 else:
                     ans.append(0)

             return ans
```

0.0.11 Create Neural Network Model

```
In [11]: def neural_network_model():
             ## create model
             model = Sequential()
             model.add(Dense(300,input_dim=88,activation='tanh',kernel_regularizer=regularizers
             model.add(Dense(150,activation='tanh'))
             model.add(Dense(3,activation='softmax'))

             ## for binary classifier
             #model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
             #model.compile(loss='mean_squared_error',optimizer='adam',metrics=['accuracy'])
             #model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accurac
             #model.compile(loss='categorical_crossentropy',optimizer='adamax',metrics=['accur
             #model.compile(loss='categorical_crossentropy',optimizer='nadam',metrics=['accura
             #model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accu
             #model.compile(loss='categorical_crossentropy',optimizer='adagrad',metrics=['accu
             #model.compile(loss='categorical_crossentropy',optimizer='adadelta',metrics=['acc
             #model.compile(loss='categorical_crossentropy',optimizer='tfoptimizer',metrics=['

             opt = optimizers.SGD(lr=0.001,momentum=0.9,decay=1e-6,nesterov=True)
             model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy'])

             return model
```

0.0.12 Helper function to get encoding, for 'target' in Neural Network

In [12]:
```
"""
Returns a (XXX,3) from (XXX,1), for the input of the neural network
Didnt use np_utils.to_categorical() since output could be -1,1 or
-1,0,1 depending on sell/buy values
"""
def myEncoder(arr):
    s = (len(arr),3)
    numpy_arr = np.zeros(s)

    for i,num in enumerate(arr):
        if num == -1:
            numpy_arr[i] = np.array([1,0,0])
        elif num == 1:
            numpy_arr[i] = np.array([0,0,1])
        else:
            numpy_arr[i] = np.array([0,1,0])

    return numpy_arr
```

0.0.13 Helper function to decode, as 1 dimensional target

In [13]:
```
"""
Returns a (XXX,1) from (XXX,3), from predict of the neural network
"""
def myDecoder(numpy_arr):
    arr = []
    ## Models predictions are ndarray, percentages per category
    for i,a in enumerate(numpy_arr):
        if np.argmax(a) == 0:
            arr.append(-1)
        elif np.argmax(a) == 2:
            arr.append(1)
        else:
            arr.append(0)

    ## Convert the arr list to an ndarray.
    return np.array(arr)
```

0.0.14 Helpter function to scale input data for neural network

In [14]:
```
def rescale_input(arr):
    """
    Scales data to be between a - b
    """
    ## Function below is not used.  Initially thought arr was a df
    def scaler(x):
```

```
        new_x = (((highest_scale-lowest_scale)*(x-min_input))/(max_input-min_input))+l
        return new_x

    ## Tried -1 to 1 before
    lowest_scale = -1
    highest_scale = 1
    max_input = 1.15
    min_input = 0.85

    #result = arr.applymap(scaler)    ## if its a df, use applymap

    ## arr is an ndarray.  Use vectorize instead of applymap
    scaler = lambda x: (((highest_scale-lowest_scale)*(x-min_input))/(max_input-min_in
                       + lowest_scale
    func = np.vectorize(scaler)                              ## vectorize scaler function
    result = func(arr)                                       ## pass arr to vectorized fun

    return result
```

0.0.15 Get the metrics of the models

```
In [15]: def get_model_metrics(target_ratio,sell_below,buy_above,X):
             temp_result = {}
             temp_result['target_ratio'] = target_ratio
             temp_result['sell_below'] = sell_below
             temp_result['buy_above'] = buy_above

             X,y = get_target(X,target_ratio)
             #print("Input shape: {} Target shape: {}".format(X.shape,len(y)))
             #print("Target found are: {}".format(set(y)))

             ## Perform cross validation.  95% to have as much training data as possible.
             ## Also, performance will be base on totally different set of stocks
             try: X_train, X_test, y_train, y_test \
                 = cross_validation.train_test_split(X,y,train_size=0.90,stratify=y)
             ## error sometimes: The least populated class in y has only 1 member, which is to
             ## The minimum number of labels for any class cannot be less than 2.
             except: X_train, X_test, y_train, y_test \
                 = cross_validation.train_test_split(X,y,train_size=0.90)

             #print("Sample of training data:")
             #print("Number of rows: {}. Number of columns: {}.".format(len(X_train),len(X_tra
             #print(X_train.head())

             beta = 0.5

             ## Initialize Models
             ## entropy for exploratory analysis, gini (default) to minimize misclassification
```

```
## max_features default None
#clf1 = DecisionTreeClassifier(criterion="entropy",random_state=0,max_features=No
clf1 = DecisionTreeClassifier()
clf2 = GaussianNB()
## kernel 'rbf' default, others are linear, poly, sigmoid, C is penalty parameter
#clf3 = SVC()                     ## <-- makes execution time 20x longer
## defaults are 1 for learning rate and 50 for n_estimators
#clf4 = AdaBoostClassifier(random_state=0,learning_rate=0.7,n_estimators=50)
clf4 = AdaBoostClassifier()
clf5 = neural_network_model()

if READ_EXISTING_MODELS == True:
    ## Read existing models

    ## For DecisionTree
    with open("models/DecisionTree_"+target_ratio+"_"+str(sell_below)\
              +"_"+str(buy_above)+".pkl", 'rb') as f:
        clf1 = pickle.load(f)

    ## For GaussianNB
    with open("models/GaussianNB_"+target_ratio+"_"+str(sell_below)\
              +"_"+str(buy_above)+".pkl", 'rb') as f:
        clf2 = pickle.load(f)

    ## For SVC model
    #with open("models/SVC_"+target_ratio+"_"+str(sell_below)\
    #          +"_"+str(buy_above)+".pkl", 'rb') as f:
    #    clf3 = pickle.load(f)

    ## For Adaboost
    with open("models/Adaboost_"+target_ratio+"_"+str(sell_below)\
              +"_"+str(buy_above)+".pkl", 'rb') as f:
        clf4 = pickle.load(f)

    ## For Neural Network
    ## Load json and create model
    json_file = open("models/NN_"+target_ratio+"_"+str(sell_below)\
                     +"_"+str(buy_above)+".json","r")
    loaded_model_json = json_file.read()
    json_file.close()
    clf5 = model_from_json(loaded_model_json)
    ## Load weights into new model
    clf5.load_weights("models/NN_"+target_ratio+"_"+str(sell_below)+"_"+str(buy_al
    ## Compile, make sure its the same as above
    opt = optimizers.SGD(lr=0.001,momentum=0.9,decay=1e-6,nesterov=True)
    clf5.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy

else:
```

```python
## Generate new models

## Fit Data to DecisionTree Model
clf1.fit(X_train,y_train)
## Save model to a file
with open("models/DecisionTree_"+target_ratio+"_"+str(sell_below)\
          +"_"+str(buy_above)+".pkl", 'wb') as f:
    pickle.dump(clf1, f)

# Fit Data to GaussianNB Model
clf2.fit(X_train,y_train)
## Save model to a file
with open("models/GaussianNB_"+target_ratio+"_"+str(sell_below)\
          +"_"+str(buy_above)+".pkl", 'wb') as f:
    pickle.dump(clf2, f)

# Fit Data to SVC Model
#clf3.fit(X_train,y_train)
## Save model to a file
#with open("models/SVC_"+target_ratio+"_"+str(sell_below)\
#          +"_"+str(buy_above)+".pkl", 'wb') as f:
#    pickle.dump(clf3, f)

# Fit Data to Adaboost Model
clf4.fit(X_train,y_train)
## Save model to a file
with open("models/Adaboost_"+target_ratio+"_"+str(sell_below)\
          +"_"+str(buy_above)+".pkl", 'wb') as f:
    pickle.dump(clf4, f)

## Fit Data to Neural Model
input_train = X_train.as_matrix(columns=None)               ## convert df to
#np.savetxt("test1.csv",input_train,delimiter=",")
input_train = rescale_input(input_train)
#print("Input_train:",input_train[np.r_[0:5]])              ## print first 5

### encode class values as integers
#encoder = LabelEncoder()
#encoder.fit(y_train)
#encoded_y = encoder.transform(y_train)
### convert integers to dummy variables (i.e one hot encoded)
#dummy_y = np_utils.to_categorical(encoded_y)
#print(dummy_y)
dummy_y = myEncoder(y_train)
clf5.fit(input_train,dummy_y,epochs=20,batch_size=100)
## Serialize model to JSON
model_json = clf5.to_json()
with open("models/NN_"+target_ratio+"_"+str(sell_below)\
```

```python
                +"_"+str(buy_above)+".json","w") as json_file:
            json_file.write(model_json)
        ## Serialize weights to HDF5
        clf5.save_weights("models/NN_"+target_ratio+"_"+str(sell_below)+"_"+str(buy_al

    ## Get accuracy and fscore of the models
    temp_result['DecisionTree Accuracy'] = accuracy_score(y_test, clf1.predict(X_test
    temp_result['DecisionTree Fscore'] = fbeta_score(y_test, clf1.predict(X_test),\
                                                      beta,average='weighted')

    temp_result['GaussianNB Accuracy'] = accuracy_score(y_test,clf2.predict(X_test))
    temp_result['GaussianNB Fscore'] = fbeta_score(y_test, clf2.predict(X_test),\
                                                    beta,average='weighted')

    #temp_result['SVC Accuracy'] = accuracy_score(y_test,clf3.predict(X_test))
    #temp_result['SVC Fscore'] = fbeta_score(y_test, clf3.predict(X_test),\
    #                                         beta,average='weighted')

    temp_result['Adaboost Accuracy'] = accuracy_score(y_test,clf4.predict(X_test))
    temp_result['Adaboost Fscore'] = fbeta_score(y_test,clf4.predict(X_test),\
                                                  beta,average='weighted')

    input_test = X_test.as_matrix(columns=None)
    input_test = rescale_input(input_test)
    dummy_y = myEncoder(y_test)
    #scores = clf5.evaluate(input_test, dummy_y)
    #temp_result['NN Accuracy'] = scores[1]*100
    temp_result['NN Accuracy'] = accuracy_score(y_test,myDecoder(clf5.predict(input_te
    temp_result['NN Fscore'] = fbeta_score(y_test, \
                               myDecoder(clf5.predict(input_test)),beta,average='we

    ## Determine performance of the portfolio on each model
    #models = [('DecisionTree',clf1),('GaussianNB',clf2),('Adaboost',clf4),\
    #             ('SVC',clf3),('NN',clf5)]
    ## No SVC model
    models = [('DecisionTree',clf1),('GaussianNB',clf2),('Adaboost',clf4),('NN',clf5)]
    test_model_performance(models,temp_result,target_ratio,sell_below,buy_above)

    ## Check if multiprocessor is set
    if MULTIPROCESSOR == False:
        ## Result will contain all the result from each combination of the models
        result.append(temp_result)
    else:
        ## To support multiprocessing, will have to save for later
        ## temp_result is a dict, not df
        #temp_result.to_csv("temp_results/"+str(time.clock())+".csv")
        with open("temp_results/"+str(time.clock())+".json", 'w') as fp:
            json.dump(temp_result, fp)
```

0.0.16 Calculate model performance

```
In [16]: def test_model_performance(models,temp_result,target_ratio,sell_below,buy_above):
             panel_data = pd.read_pickle('testing_stocks.pkl')           ## read saved stocks

             ## Initialize total to be 0 across all models
             total = {'benchmark':0}
             for model_name,_ in models:
                 total[model_name] = 0                          ## will hold total money for that
                 total[model_name+"_transactions"] = 0          ## will hold total transactions f

             ## Go through each stock
             for tick in TESTING_TICKERS:
                 ## Extract single stock from panel_data
                 #df = panel_data[:,:,tick]
                 #X = get_stock_df(df,tick)
                 ## We will now just read the test stocks, generated before
                 X = pd.DataFrame.from_csv("testing_data/"+tick+'_processed.csv')

                 ## Delete unnecessary columns
                 del X['predict'],X['prev_close'],X['prev_r2'],X['prev_r3'],X['prev_r5']
                 del X['prev_r10'],X['prev_r15'],X['prev_r25'],X['prev_r40']
                 ## Delete columns with actual price
                 del X['vOpen'],X['vHigh'],X['vLow'],X['vClose'],X['vr2']
                 del X['vr3'],X['vr5'],X['vr10'],X['vr15'],X['vr25'],X['vr40']

                 ## Get predictions base on each models
                 for model_name, model in models:
                     ## predictions will be an ndarray
                     if model_name == 'NN':
                         input_test = X.as_matrix(columns=None)
                         input_test = rescale_input(input_test)
                         predictions = model.predict(input_test)
                         predictions = myDecoder(predictions)
                     else:
                         predictions = model.predict(X)

                     ## Add predictions to the dataframe
                     pred = pd.DataFrame(predictions.flatten(),index=X.index,columns=['Predicti
                     S = X.join(pred)
                     S['Transactions'] = 0                          ## will contain number of tra
                     S['Money'] = 0                                 ## will contain total current

                     ## Calculate transactions (NOT COMPLETE)
                     #temp_df = pd.DataFrame(X['Close_pc'].values,columns=['Close_pc'])
                     #temp_df = temp_df.join(pd.DataFrame(predictions.flatten(),columns=['Pred
                     #temp_df['playing'] = temp_df['Predictions'].shift().eq(1)
                     ### cumprod of 'Close_pc' where 'playing' is True.  Then multiple with in
```

```
        #temp_df['Money'] = \
        #     temp_df['Close_pc'].where(temp_df['playing'],1).cumprod().mul(MONEY
        ### get just last value from 'Money'
        #temp_result[tick+'_'+model_name] = float(format(temp_df['Money'].iloc[-1
        #total[model_name] += float(format(temp_df['Money'].iloc[-1], '.2f'))

        ## Calculate transactions. SLOWER? BUT COMPLETE
        playing = False                     ## used to determine if currently in
        money = copy.copy(MONEY)
        transactions = 0
        i = 0
        for index, row in S.iterrows():
            ## Update money
            if i > 0 and playing == True:
                money = float(format(money*row['Close_pc'],'.2f'))

            ## Buy/Sell
            if row['Predictions'] == 1:
                if playing == False:
                    playing = True          ## Buy, playing after this
                    transactions += 1       ## increment transaction number for t
            elif row['Predictions'] == -1:
                if playing == True:
                    playing = False         ## Sell, not playing after this
                    transactions += 1       ## increment transaction number for t
            i += 1

            ## Change value in sample data S in index and column provided,
            ## with value/data provided
            S.set_value(index,'Money',money)
            S.set_value(index,'Transactions',transactions)

        ## Save dataframe for testing purposes
        S.to_csv("model_predictions/"+tick+"_"+model_name+"_"\
                 +target_ratio+"_"+str(sell_below)+"_"+str(buy_above)+".csv")

        ## If still playing at the end, we'll sell, thus increment number of tran
        transactions = transactions + 1 if playing == True else transactions
        ## Will contain total money for this stock and model
        temp_result[tick+'_'+model_name] = money
        ## Will contain total number of transactions for this stock and model
        temp_result[tick+'_'+model_name+"_transactions"] = transactions
        ## Will contain total money for this model, including all stocks
        total[model_name] += money
        ## total transactions for the model, including all stocks
        total[model_name+"_transactions"] += transactions

    ## Calculate benchmark portfolio for current stock (NOT COMPLETE)
```

```
        #temp_df = X['Close_pc'].to_frame()
        #temp_df['Close_pc'].iloc[0] = 1       ## since first one is not played
        #temp_df['Money'] = temp_df['Close_pc'].cumprod().mul(MONEY)
        ### total money in benchmark, for that stock
        #temp_result[tick+'_benchmark'] = float(format(temp_df['Money'].iloc[-1], '.2
        #total['benchmark'] += float(format(temp_df['Money'].iloc[-1], '.2f'))

        ## Calculate benchmark portfolio for current stock. SLOWER? BUT COMPLETE
        money = copy.copy(MONEY)
        for i,r in enumerate(X['Close_pc']):
            if i > 0:
                money = float(format(money*r,'.2f'))
        temp_result[tick+'_benchmark'] = money      ## total money in benchmark, for
        total['benchmark'] += money          ## will contain total money in benchmark
        total['benchmark_transcations'] = 2         ## Initial buy and the sell at th

    ## Determine Total values of each portfolio
    temp_result['total_benchmark'] = total['benchmark']   ## total money in benchmark

    ## Get total money and transactions per each model.  Each including all stocks
    for model_name,_ in models:
        temp_result['total_'+model_name] = total[model_name]
        temp_result['total_'+model_name+"_transactions"] = total[model_name+"_transac
```

0.0.17 Helper function to save each of the testing stocks as csv

```
In [17]: def save_testing_stocks(filename):
             panel_data = pd.read_pickle(filename)                   ## read saved stocks data

             ## Go through each stock
             for tick in TESTING_TICKERS:
                 ## Extract single stock from panel_data
                 df = panel_data[:,:,tick]

                 ## save raw stock data
                 df.to_csv("testing_data/"+tick+".csv")

                 ## process data
                 df = get_stock_df(df,tick)

                 ## Plot stock
                 plot_stock(tick,df)

                 ## Save to csv
                 df.to_csv("testing_data/"+tick+"_processed.csv")
```

0.0.18 Helper function to delete all the old data from previous runs

In [18]:
```
def cleanup_contents():
    filelist = glob.glob("temp_results/*.json")
    for f in filelist:
        os.remove(f)

    filelist = glob.glob("model_predictions/*.csv")
    for f in filelist:
        os.remove(f)

    filelist = glob.glob("figures/*")
    for f in filelist:
        if "README.md" not in f:
            os.remove(f)

    filelist = glob.glob("training_data/*.csv")
    for f in filelist:
        os.remove(f)

    filelist = glob.glob("testing_data/*.csv")
    for f in filelist:
        os.remove(f)

    if READ_EXISTING_MODELS == False:
        filelist = glob.glob("models/*")
        for f in filelist:
            if "README.md" not in f:
                os.remove(f)
```

0.0.19 Main()

In [19]:
```
%%time


result = []

## Remove all existing .json files in temp folder and remove old csv files in data fol
cleanup_contents()

filename = 'training_stocks.pkl'
## Generate data to be inputted to the models
X = get_training_stocks_df(filename)

## Describe training data
describe_data()
```

```
    filename = 'testing_stocks.pkl'
    ## Generate data of test stocks and save to csv files
    save_testing_stocks(filename)

    ## Check if multiprocessor is set
    if MULTIPROCESSOR == False:
        ## For single processing
        # Get performance of different types of models
        for sell_below, buy_above in SELL_BUY_VALUES:
            for target_ratio in TARGET_RATIOS:
                ## copy to prevent updating
                get_model_metrics(target_ratio,sell_below,buy_above,X.copy())
                print("Completed target ratio: {} with sell: {} and buy: {}"\
                      .format(target_ratio,sell_below,buy_above))
    else:
        ## For multiprocessing
        ## Get the arguments for the pool
        args = []
        for sell_below, buy_above in SELL_BUY_VALUES:
            for target_ratio in TARGET_RATIOS:
                arg = (target_ratio,sell_below,buy_above,X.copy())
                args.append(arg)

        ## With multiprocessing
        pool = mp.Pool(processes=NUM_PROCESSES)
        pool.starmap(get_model_metrics,args)
        pool.close()
        pool.join()
```

```
Plotting:  AAPL
Plotting:  GOOG
Plotting:  T
Plotting:  IMAX
Plotting:  IBM
Plotting:  NFLX
Plotting:  SIRI
Plotting:  S
Plotting:  PLUG
Plotting:  C
Plotting:  ZNGA
Plotting:  WMS
Plotting:  BAC
Plotting:  AMZN
Plotting:  FB
Plotting:  P
Plotting:  WM
Plotting:  NOK
Plotting:  DDD
```

```
Plotting:  XME
Plotting:  XONE
Plotting:  SDRL
Plotting:  TSLA
Plotting:  SSYS
Plotting:  TXN
Plotting:  F
Plotting:  GS
Plotting:  LQMT
Plotting:  HTZ
Plotting:  BAH
Plotting:  GLW
Plotting:  SPWR
Plotting:  BIDU
Plotting:  SRPT
Plotting:  YGE
Plotting:  CNX
Plotting:  URRE
Plotting:  VJET
Plotting:  RAD
Plotting:  NQ
Plotting:  KORS
Plotting:  TWTR
Plotting:  HLF
Plotting:  ORCL
Plotting:  WLL
Plotting:  BLDP
Plotting:  PEG
Plotting:  MJNA
Plotting:  CBIS
Plotting:  TM
Plotting:  SBUX
Plotting:  MBLY
Plotting:  MRK
Plotting:  DBO
Plotting:  PFE
Plotting:  CAMP
Plotting:  TRXC
Plotting:  BMY
Plotting:  FE
Plotting:  VTR
Plotting:  UHT
Plotting:  MVO
Plotting:  KF
Plotting:  RACE
Plotting:  STOR
Plotting:  MU
Plotting:  RTN
```

```
Raw training data description:
              Close          High           Low          Open
count  35863.000000  35863.000000  35863.000000  35863.000000
mean      57.492626     58.181523     56.792067     57.509487
std       98.580690     99.558195     97.533216     98.600096
min        0.011000      0.011000      0.010000      0.011000
25%       11.600000     11.960000     11.300000     11.670000
50%       30.750000     31.250000     30.270000     30.790001
75%       54.279999     54.880001     53.740002     54.340000
max      776.599976    789.869995    766.900024    784.500000

Processed training data description:
            predict    prev_close       prev_r2       prev_r3       prev_r5  \
count  33116.000000  33116.000000  33116.000000  33116.000000  33116.000000
mean      57.650932     57.637520     57.634570     57.631384     57.625151
std      100.020293     99.834763     99.784070     99.734307     99.636152
min        0.011000      0.011000      0.011000      0.011000      0.011000
25%       10.937500     11.007500     11.045000     11.040000     11.111000
50%       30.270000     30.320000     30.320000     30.313334     30.319000
75%       54.340000     54.340000     54.335000     54.336666     54.328000
max      776.599976    776.599976    773.799988    770.036662    763.478003

           prev_r10      prev_r15      prev_r25      prev_r40         vOpen  \
count  33116.000000  33116.000000  33116.000000  33116.000000  33116.000000
mean      57.608467     57.590755     57.557355     57.505788     57.655950
std       99.392219     99.143964     98.668996     97.983877     99.939926
min        0.011200      0.011467      0.012200      0.012275      0.011000
25%       11.131500     11.207667     11.433800     11.601563     11.000000
50%       30.374500     30.423333     30.618000     30.815375     30.299999
75%       54.333500     54.259167     54.133000     53.965500     54.369999
max      756.831995    754.413330    754.352402    747.968002    784.500000

           ...          High_p40       Low_p40     Close_p40        r2_p40  \
count      ...      33116.000000  33116.000000  33116.000000  33116.000000
mean       ...          1.003903      0.969174      0.986053      0.986247
std        ...          0.114326      0.110338      0.113037      0.109736
min        ...          0.375742      0.295510      0.343568      0.350964
25%        ...          0.954380      0.921143      0.936566      0.938189
50%        ...          1.006872      0.983364      0.994662      0.995022
75%        ...          1.051578      1.026764      1.039941      1.038682
max        ...          3.582875      2.254823      2.855776      2.779254

             r3_p40        r5_p40       r10_p40       r15_p40       r25_p40  \
count  33116.000000  33116.000000  33116.000000  33116.000000  33116.000000
mean       0.986453      0.986874      0.988081      0.989481      0.992811
std        0.106759      0.101113      0.087536      0.073997      0.046771
min        0.357857      0.371710      0.399461      0.440318      0.566140
```

```
25%        0.939784      0.942944      0.950446      0.958205      0.973870
50%        0.995284      0.995688      0.996419      0.996939      0.998050
75%        1.037749      1.035906      1.030854      1.026081      1.016421
max        2.504456      2.095218      1.732265      1.502319      1.271795

            r40_p40
count  33116.000000
mean       0.999160
std        0.004857
min        0.965386
25%        0.997213
50%        0.999780
75%        1.001645
max        1.036849

[8 rows x 108 columns]
Plotting:  BA
Plotting:  HON
Plotting:  MA
Plotting:  TPLM
Plotting:  SD
Plotting:  FCEL
Plotting:  CHK
Plotting:  CMG


/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)
/Users/Gio/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:1113: Undefin
  'precision', 'predicted', average, warn_for)


CPU times: user 46.4 s, sys: 5.75 s, total: 52.1 s
Wall time: 2min 42s
```

0.0.20 Read all json in temp folder and append to result (used for multiprocessing)

```
In [20]: if MULTIPROCESSOR == True:
             result = []

             directory = "temp_results/"
             for filename in os.listdir(directory):
                 if filename.endswith(".json"):
                     js = open(directory+filename).read()
                     temp_dict = json.loads(js)
                     result.append(temp_dict)
```

0.0.21 Print Results of the models

```
In [21]: result_df = pd.DataFrame(result)
         print(result_df)
```

```
     Adaboost Accuracy  Adaboost Fscore  BA_Adaboost  \
0             0.687802         0.647832     14010.03
1             0.853563         0.805866     16438.43
2             0.687198         0.650150     13386.15
3             0.889191         0.847062     16960.51
4             0.891002         0.856818     16657.76
5             0.687198         0.650150     13386.15
6             0.890700         0.860514     16466.70
7             0.808575         0.761511     15652.32
8             0.564614         0.533326     10819.24
9             0.621075         0.597002     13027.33
10            0.808575         0.761511     15652.32
11            0.808575         0.761511     15652.32
12            0.564614         0.533326     10819.24
13            0.853865         0.804029     16455.14
14            0.621075         0.597002     13027.33
15            0.853865         0.804029     16455.14
16            0.621075         0.597002     13027.33
17            0.687198         0.658004     13928.51
18            0.853865         0.804029     16455.14
19            0.853865         0.804029     16455.14
20            0.621075         0.597002     13027.33
21            0.887077         0.850144     17090.62
22            0.687198         0.658004     13928.51
23            0.687198         0.658004     13928.51
24            0.887077         0.850144     17090.62
25            0.887077         0.850144     17090.62
26            0.887077         0.850144     17090.62
27            0.687198         0.658004     13928.51
28            0.525060         0.347277     10000.00
29            0.766304         0.715252     14708.86
```

..	...	...	...
90	0.846316	0.797931	16455.14
91	0.615942	0.589623	12839.41
92	0.884662	0.844267	17090.62
93	0.678744	0.641656	13550.79
94	0.678744	0.641656	13550.79
95	0.564614	0.533326	10819.24
96	0.523551	0.340272	10000.00
97	0.884964	0.843269	16721.16
98	0.884360	0.842980	16656.12
99	0.884964	0.843269	16721.16
100	0.677536	0.637333	13429.85
101	0.528684	0.343793	10000.00
102	0.763285	0.724312	14582.51
103	0.527174	0.356807	10000.00
104	0.528684	0.343793	10000.00
105	0.575785	0.548450	10966.00
106	0.808575	0.761511	15652.32
107	0.564614	0.533326	10819.24
108	0.767210	0.727584	14458.21
109	0.762983	0.709144	14834.16
110	0.807669	0.765459	15726.71
111	0.567331	0.536495	10970.40
112	0.807971	0.767944	15728.18
113	0.575785	0.548450	10966.00
114	0.628019	0.600945	13396.57
115	0.811594	0.773247	15990.51
116	0.631341	0.605814	12993.82
117	0.854167	0.808356	16438.43
118	0.851449	0.803986	16544.40
119	0.628019	0.600945	13396.57

	BA_Adaboost_transactions	BA_DecisionTree	BA_DecisionTree_transactions	\
0	58	12436.22	72	
1	22	16191.35	32	
2	52	12909.32	66	
3	18	16417.43	28	
4	16	16902.40	18	
5	52	12909.32	66	
6	18	16860.40	14	
7	34	15130.06	50	
8	28	11790.71	60	
9	46	12133.34	58	
10	34	15130.06	50	
11	34	15130.06	50	
12	28	11790.71	60	
13	20	16124.31	30	
14	46	12133.34	58	

15	20	16124.31	30
16	46	12133.34	58
17	52	14435.33	64
18	20	16124.31	30
19	20	16124.31	30
20	46	12133.34	58
21	16	16830.42	20
22	52	14435.33	64
23	52	14435.33	64
24	16	16830.42	20
25	16	16830.42	20
26	16	16830.42	20
27	52	14435.33	64
28	0	12788.26	64
29	40	15974.41	48
..	...	...	...
90	20	15291.05	34
91	48	11512.71	70
92	16	16442.51	30
93	60	11749.76	68
94	60	11749.76	68
95	28	11790.71	60
96	0	10802.09	50
97	20	17435.12	20
98	20	16511.07	24
99	20	17435.12	20
100	56	14880.55	54
101	0	10714.08	74
102	34	13752.63	52
103	0	10373.58	66
104	0	10714.08	74
105	30	11769.59	72
106	34	15130.06	50
107	28	11790.71	60
108	38	13298.70	52
109	40	13417.17	56
110	28	15760.05	44
111	30	11807.68	80
112	28	14731.20	42
113	30	11769.59	72
114	58	12951.93	72
115	32	14669.48	42
116	58	11891.82	68
117	22	15971.00	32
118	18	16684.27	30
119	58	12951.93	72

```
     BA_GaussianNB  BA_GaussianNB_transactions     BA_NN  BA_NN_transactions  \
```

0	16577.61	2	14193.64	58
1	16577.61	2	16793.74	20
2	16577.61	2	12316.68	62
3	16577.61	2	17037.26	14
4	16577.61	2	17123.13	14
5	16577.61	2	12316.68	62
6	16577.61	2	17123.13	14
7	16577.61	2	15691.99	34
8	10334.98	2	10326.56	10
9	16577.61	2	11038.13	56
10	16577.61	2	15691.99	34
11	16577.61	2	15691.99	34
12	10334.98	2	10326.56	10
13	16577.61	2	16444.62	20
14	16577.61	2	11038.13	56
15	16577.61	2	16444.62	20
16	16577.61	2	11038.13	56
17	16577.61	2	12829.09	62
18	16577.61	2	16444.62	20
19	16577.61	2	16444.62	20
20	16577.61	2	11038.13	56
21	16577.61	2	17020.63	14
22	16577.61	2	12829.09	62
23	16577.61	2	12829.09	62
24	16577.61	2	17020.63	14
25	16577.61	2	17020.63	14
26	16577.61	2	17020.63	14
27	16577.61	2	12829.09	62
28	10334.98	2	10000.00	0
29	16577.61	2	14355.48	42
..	...	...	...	...
90	16577.61	2	16624.50	16
91	16577.61	2	11019.15	60
92	16577.61	2	17020.63	14
93	16577.61	2	14101.76	58
94	16577.61	2	14101.76	58
95	10334.98	2	10326.56	10
96	10334.98	2	10000.00	0
97	16577.61	2	17037.26	14
98	16577.61	2	17123.13	14
99	16577.61	2	17037.26	14
100	16577.61	2	12310.20	60
101	10334.98	2	10000.00	0
102	16577.61	2	14648.65	42
103	10334.98	2	10000.00	0
104	10334.98	2	10000.00	0
105	10334.98	2	10517.79	14
106	16577.61	2	15691.99	34

```
107      10334.98                                    2  10326.56                 10
108      16577.61                                    2  14833.46                 40
109      16577.61                                    2  14648.65                 42
110      16577.61                                    2  15617.80                 36
111      10334.98                                    2  10517.79                 14
112      16577.61                                    2  15707.85                 38
113      10334.98                                    2  10517.79                 14
114      16577.61                                    2  11172.91                 60
115      16577.61                                    2  16077.98                 28
116      16577.61                                    2  11244.13                 58
117      16577.61                                    2  16793.74                 20
118      16577.61                                    2  16884.75                 18
119      16577.61                                    2  11172.91                 60

         ...          target_ratio  total_Adaboost  \
0        ...                   vr5        72031.89
1        ...                  vr25        75496.54
2        ...                   vr5        70617.86
3        ...                  vr40        76612.10
4        ...                  vr40        77254.12
5        ...                   vr5        70617.86
6        ...                  vr40        76761.01
7        ...                  vr15        81043.92
8        ...                   vr2        59765.96
9        ...                   vr3        66142.60
10       ...                  vr15        81043.92
11       ...                  vr15        81043.92
12       ...                   vr2        59765.96
13       ...                  vr25        75812.47
14       ...                   vr3        66142.60
15       ...                  vr25        75812.47
16       ...                   vr3        66142.60
17       ...                   vr5        71664.07
18       ...                  vr25        75812.47
19       ...                  vr25        75812.47
20       ...                   vr3        66142.60
21       ...                  vr40        77857.79
22       ...                   vr5        71664.07
23       ...                   vr5        71664.07
24       ...                  vr40        77857.79
25       ...                  vr40        77857.79
26       ...                  vr40        77857.79
27       ...                   vr5        71664.07
28       ...                vClose        83363.26
29       ...                  vr10        72114.89
..       ...                   ...             ...
90       ...                  vr25        75490.07
91       ...                   vr3        66550.10
```

```
92          ...                  vr40       77293.08
93          ...                   vr5       69477.77
94          ...                   vr5       69477.77
95          ...                   vr2       59765.96
96          ...                vClose       83892.83
97          ...                  vr40       76747.22
98          ...                  vr40       77169.35
99          ...                  vr40       76747.22
100         ...                   vr5       71971.39
101         ...                vClose       86962.55
102         ...                  vr10       71362.06
103         ...                vClose       80525.35
104         ...                vClose       86962.55
105         ...                   vr2       60653.77
106         ...                  vr15       81043.92
107         ...                   vr2       59765.96
108         ...                  vr10       71935.62
109         ...                  vr10       73162.54
110         ...                  vr15       76210.15
111         ...                   vr2       60577.61
112         ...                  vr15       79766.38
113         ...                   vr2       60653.77
114         ...                   vr3       65545.02
115         ...                  vr15       79227.53
116         ...                   vr3       66566.11
117         ...                  vr25       75468.25
118         ...                  vr25       76107.41
119         ...                   vr3       65545.02

     total_Adaboost_transactions  total_DecisionTree  \
0                            436            73496.24
1                            218            77506.96
2                            408            74806.42
3                            166            76424.04
4                            158            74208.49
5                            408            74806.42
6                            160            79157.69
7                            264            78234.12
8                            550            68061.95
9                            524            79271.89
10                           264            78234.12
11                           264            78234.12
12                           550            68061.95
13                           214            75745.18
14                           524            79271.89
15                           214            75745.18
16                           524            79271.89
17                           406            72209.33
```

```
18                           214           75745.18
19                           214           75745.18
20                           524           79271.89
21                           168           79960.76
22                           406           72209.33
23                           406           72209.33
24                           168           79960.76
25                           168           79960.76
26                           168           79960.76
27                           406           72209.33
28                           128           77643.18
29                           326           72970.45
..                           ...                ...
90                           208           75482.47
91                           526           72318.04
92                           168           76291.39
93                           440           69016.94
94                           440           69016.94
95                           550           68061.95
96                            80           74426.96
97                           168           82063.68
98                           168           74038.06
99                           168           82063.68
100                          424           77211.57
101                           78           88674.42
102                          304           66784.54
103                          106           76610.97
104                           78           88674.42
105                          582           78112.62
106                          264           78234.12
107                          550           68061.95
108                          322           77931.76
109                          316           73779.01
110                          258           73905.73
111                          580           87892.70
112                          258           76554.09
113                          582           78112.62
114                          536           67972.31
115                          266           70404.53
116                          512           78424.13
117                          214           72801.83
118                          212           78454.71
119                          536           67972.31

     total_DecisionTree_transactions  total_GaussianNB  \
0                                540          69375.30
1                                272          73744.83
2                                538          69942.73
```

3	270	76606.62
4	230	76606.62
5	538	69942.73
6	230	76606.62
7	378	71651.71
8	642	62234.79
9	574	67110.88
10	378	71651.71
11	378	71651.71
12	642	62234.79
13	302	73744.83
14	574	67110.88
15	302	73744.83
16	574	67110.88
17	560	69893.21
18	302	73744.83
19	302	73744.83
20	574	67110.88
21	232	76386.58
22	560	69893.21
23	560	69893.21
24	232	76386.58
25	232	76386.58
26	232	76386.58
27	560	69893.21
28	618	71583.00
29	454	71372.60
..	...	...
90	314	74506.69
91	616	67954.58
92	260	73259.61
93	572	69301.98
94	572	69301.98
95	642	62234.79
96	592	71687.30
97	220	73645.78
98	218	76323.88
99	220	73645.78
100	560	69228.59
101	636	71762.17
102	482	71024.09
103	628	71422.87
104	636	71762.17
105	638	62438.38
106	378	71651.71
107	642	62234.79
108	428	71502.80
109	446	72286.02

110	374	71761.09
111	634	61737.31
112	346	72141.38
113	638	62438.38
114	626	66564.88
115	384	72030.06
116	612	67219.68
117	286	73744.83
118	288	73848.41
119	626	66564.88

	total_GaussianNB_transactions	total_NN	total_NN_transactions	\
0	124	70913.04	428	
1	68	78494.64	208	
2	124	70437.85	436	
3	60	76987.90	162	
4	62	75660.41	162	
5	124	70437.85	436	
6	62	75532.20	168	
7	88	74075.27	276	
8	138	67412.39	426	
9	146	66547.47	482	
10	88	74075.27	276	
11	88	74075.27	276	
12	138	67412.39	426	
13	68	77505.77	208	
14	146	66547.47	482	
15	68	77505.77	208	
16	146	66547.47	482	
17	124	70046.18	438	
18	68	77505.77	208	
19	68	77505.77	208	
20	146	66547.47	482	
21	60	76505.30	160	
22	124	70046.18	438	
23	124	70046.18	438	
24	60	76505.30	160	
25	60	76505.30	160	
26	60	76505.30	160	
27	124	70046.18	438	
28	90	76350.15	52	
29	94	73366.95	324	
..	...	...	...	
90	68	78255.87	196	
91	136	67871.86	482	
92	62	76505.30	162	
93	122	71519.51	446	
94	122	71519.51	446	

95	138	67412.39	426
96	86	93475.52	40
97	64	76498.13	162
98	62	75732.33	166
99	64	76498.13	162
100	122	69339.79	438
101	90	91098.91	38
102	96	73231.91	334
103	86	90704.68	44
104	90	91098.91	38
105	138	67131.81	450
106	88	74075.27	276
107	138	67412.39	426
108	96	73232.51	318
109	96	72871.88	330
110	88	74208.37	278
111	142	69437.54	436
112	86	74127.82	284
113	138	67131.81	450
114	136	67803.96	488
115	88	78448.76	258
116	144	68393.86	488
117	68	77681.49	210
118	70	78530.02	204
119	136	67803.96	488

	total_benchmark
0	74037.88
1	74037.88
2	74037.88
3	74037.88
4	74037.88
5	74037.88
6	74037.88
7	74037.88
8	74037.88
9	74037.88
10	74037.88
11	74037.88
12	74037.88
13	74037.88
14	74037.88
15	74037.88
16	74037.88
17	74037.88
18	74037.88
19	74037.88
20	74037.88

```
21          74037.88
22          74037.88
23          74037.88
24          74037.88
25          74037.88
26          74037.88
27          74037.88
28          74037.88
29          74037.88
..               ...
90          74037.88
91          74037.88
92          74037.88
93          74037.88
94          74037.88
95          74037.88
96          74037.88
97          74037.88
98          74037.88
99          74037.88
100         74037.88
101         74037.88
102         74037.88
103         74037.88
104         74037.88
105         74037.88
106         74037.88
107         74037.88
108         74037.88
109         74037.88
110         74037.88
111         74037.88
112         74037.88
113         74037.88
114         74037.88
115         74037.88
116         74037.88
117         74037.88
118         74037.88
119         74037.88

[120 rows x 92 columns]
```

0.0.22 Add a column FinalValue per model. Which takes taxes and commisions into account

```
In [22]: ## Go through each row
         for index, row in result_df.iterrows():
```

```
## Initialize params
benchmark = {'value':0,'loss':0,'gain':0,'transactions':len(TESTING_TICKERS)*2.0}
decisiontree = {'value':0,'loss':0,'gain':0,'transactions':0}
gaussiannb = {'value':0,'loss':0,'gain':0,'transactions':0}
adaboost = {'value':0,'loss':0,'gain':0,'transactions':0}
svc = {'value':0,'loss':0,'gain':0,'transactions':0}
nn = {'value':0,'loss':0,'gain':0,'transactions':0}

#models_dict = {'benchmark':benchmark,'DecisionTree':decisiontree,\
#                'GaussianNB':gaussiannb,'Adaboost':adaboost, 'SVC':svc, 'NN':nn}
## No SVC model
models_dict = {'benchmark':benchmark,'DecisionTree':decisiontree,\
               'GaussianNB':gaussiannb,'Adaboost':adaboost, 'NN':nn}

## Gather each stock information
for tick in TESTING_TICKERS:
    ## update each models
    for key, model in models_dict.items():
        stock_value = row[tick+"_"+key]
        if stock_value > MONEY:                                 ## capital gain
            model['gain'] += stock_value-MONEY
        else:                                                   ## loss
            model['loss'] += MONEY-stock_value

        if key != 'benchmark':
            model['transactions'] += row[tick+"_"+key+"_transactions"]


## Get Final Values
for key, model in models_dict.items():
    ## more gains than loss
    if model['gain'] > model['loss']:
        if key == 'benchmark':
            model['value'] = float(format(row['total_'+key] \
                                    - COMM_RATE*model['transactions'] \
                                    - GAIN_LONG*(model['gain']-model['loss']),'.2f')
        else:
            model['value'] = float(format(row['total_'+key] \
                                    - COMM_RATE*model['transactions'] \
                                    - GAIN_SHORT*(model['gain']-model['loss']),'.2f
    ## more loss than gain
    else:
        ## All model gains GAIN_LONG (assuming its on 25% tax bracket)
        model['value'] = float(format(row['total_'+key] - COMM_RATE*model['transac
                         + GAIN_LONG*(model['loss']-model['gain']),'.2f')) ## add


    result_df.set_value(index,key+"_FinalValue",model['value'])
```

```
In [23]: print(result_df)
         result_df.to_csv('Results.csv')  ## can use parameters: mode='a', header=False, if me
```

```
    Adaboost Accuracy  Adaboost Fscore  BA_Adaboost  \
0            0.687802         0.647832     14010.03
1            0.853563         0.805866     16438.43
2            0.687198         0.650150     13386.15
3            0.889191         0.847062     16960.51
4            0.891002         0.856818     16657.76
5            0.687198         0.650150     13386.15
6            0.890700         0.860514     16466.70
7            0.808575         0.761511     15652.32
8            0.564614         0.533326     10819.24
9            0.621075         0.597002     13027.33
10           0.808575         0.761511     15652.32
11           0.808575         0.761511     15652.32
12           0.564614         0.533326     10819.24
13           0.853865         0.804029     16455.14
14           0.621075         0.597002     13027.33
15           0.853865         0.804029     16455.14
16           0.621075         0.597002     13027.33
17           0.687198         0.658004     13928.51
18           0.853865         0.804029     16455.14
19           0.853865         0.804029     16455.14
20           0.621075         0.597002     13027.33
21           0.887077         0.850144     17090.62
22           0.687198         0.658004     13928.51
23           0.687198         0.658004     13928.51
24           0.887077         0.850144     17090.62
25           0.887077         0.850144     17090.62
26           0.887077         0.850144     17090.62
27           0.687198         0.658004     13928.51
28           0.525060         0.347277     10000.00
29           0.766304         0.715252     14708.86
..                ...              ...          ...
90           0.846316         0.797931     16455.14
91           0.615942         0.589623     12839.41
92           0.884662         0.844267     17090.62
93           0.678744         0.641656     13550.79
94           0.678744         0.641656     13550.79
95           0.564614         0.533326     10819.24
96           0.523551         0.340272     10000.00
97           0.884964         0.843269     16721.16
98           0.884360         0.842980     16656.12
99           0.884964         0.843269     16721.16
```

100	0.677536	0.637333	13429.85
101	0.528684	0.343793	10000.00
102	0.763285	0.724312	14582.51
103	0.527174	0.356807	10000.00
104	0.528684	0.343793	10000.00
105	0.575785	0.548450	10966.00
106	0.808575	0.761511	15652.32
107	0.564614	0.533326	10819.24
108	0.767210	0.727584	14458.21
109	0.762983	0.709144	14834.16
110	0.807669	0.765459	15726.71
111	0.567331	0.536495	10970.40
112	0.807971	0.767944	15728.18
113	0.575785	0.548450	10966.00
114	0.628019	0.600945	13396.57
115	0.811594	0.773247	15990.51
116	0.631341	0.605814	12993.82
117	0.854167	0.808356	16438.43
118	0.851449	0.803986	16544.40
119	0.628019	0.600945	13396.57

	BA_Adaboost_transactions	BA_DecisionTree	BA_DecisionTree_transactions \
0	58	12436.22	72
1	22	16191.35	32
2	52	12909.32	66
3	18	16417.43	28
4	16	16902.40	18
5	52	12909.32	66
6	18	16860.40	14
7	34	15130.06	50
8	28	11790.71	60
9	46	12133.34	58
10	34	15130.06	50
11	34	15130.06	50
12	28	11790.71	60
13	20	16124.31	30
14	46	12133.34	58
15	20	16124.31	30
16	46	12133.34	58
17	52	14435.33	64
18	20	16124.31	30
19	20	16124.31	30
20	46	12133.34	58
21	16	16830.42	20
22	52	14435.33	64
23	52	14435.33	64
24	16	16830.42	20
25	16	16830.42	20

26	16	16830.42	20
27	52	14435.33	64
28	0	12788.26	64
29	40	15974.41	48
..	...	...	...
90	20	15291.05	34
91	48	11512.71	70
92	16	16442.51	30
93	60	11749.76	68
94	60	11749.76	68
95	28	11790.71	60
96	0	10802.09	50
97	20	17435.12	20
98	20	16511.07	24
99	20	17435.12	20
100	56	14880.55	54
101	0	10714.08	74
102	34	13752.63	52
103	0	10373.58	66
104	0	10714.08	74
105	30	11769.59	72
106	34	15130.06	50
107	28	11790.71	60
108	38	13298.70	52
109	40	13417.17	56
110	28	15760.05	44
111	30	11807.68	80
112	28	14731.20	42
113	30	11769.59	72
114	58	12951.93	72
115	32	14669.48	42
116	58	11891.82	68
117	22	15971.00	32
118	18	16684.27	30
119	58	12951.93	72

	BA_GaussianNB	BA_GaussianNB_transactions	BA_NN	BA_NN_transactions	\
0	16577.61	2	14193.64	58	
1	16577.61	2	16793.74	20	
2	16577.61	2	12316.68	62	
3	16577.61	2	17037.26	14	
4	16577.61	2	17123.13	14	
5	16577.61	2	12316.68	62	
6	16577.61	2	17123.13	14	
7	16577.61	2	15691.99	34	
8	10334.98	2	10326.56	10	
9	16577.61	2	11038.13	56	
10	16577.61	2	15691.99	34	

11	16577.61	2	15691.99	34
12	10334.98	2	10326.56	10
13	16577.61	2	16444.62	20
14	16577.61	2	11038.13	56
15	16577.61	2	16444.62	20
16	16577.61	2	11038.13	56
17	16577.61	2	12829.09	62
18	16577.61	2	16444.62	20
19	16577.61	2	16444.62	20
20	16577.61	2	11038.13	56
21	16577.61	2	17020.63	14
22	16577.61	2	12829.09	62
23	16577.61	2	12829.09	62
24	16577.61	2	17020.63	14
25	16577.61	2	17020.63	14
26	16577.61	2	17020.63	14
27	16577.61	2	12829.09	62
28	10334.98	2	10000.00	0
29	16577.61	2	14355.48	42
..	...	...	...	...
90	16577.61	2	16624.50	16
91	16577.61	2	11019.15	60
92	16577.61	2	17020.63	14
93	16577.61	2	14101.76	58
94	16577.61	2	14101.76	58
95	10334.98	2	10326.56	10
96	10334.98	2	10000.00	0
97	16577.61	2	17037.26	14
98	16577.61	2	17123.13	14
99	16577.61	2	17037.26	14
100	16577.61	2	12310.20	60
101	10334.98	2	10000.00	0
102	16577.61	2	14648.65	42
103	10334.98	2	10000.00	0
104	10334.98	2	10000.00	0
105	10334.98	2	10517.79	14
106	16577.61	2	15691.99	34
107	10334.98	2	10326.56	10
108	16577.61	2	14833.46	40
109	16577.61	2	14648.65	42
110	16577.61	2	15617.80	36
111	10334.98	2	10517.79	14
112	16577.61	2	15707.85	38
113	10334.98	2	10517.79	14
114	16577.61	2	11172.91	60
115	16577.61	2	16077.98	28
116	16577.61	2	11244.13	58
117	16577.61	2	16793.74	20

118	16577.61	2	16884.75	18
119	16577.61	2	11172.91	60

	...	total_GaussianNB	total_GaussianNB_transactions	\
0	...	69375.30	124	
1	...	73744.83	68	
2	...	69942.73	124	
3	...	76606.62	60	
4	...	76606.62	62	
5	...	69942.73	124	
6	...	76606.62	62	
7	...	71651.71	88	
8	...	62234.79	138	
9	...	67110.88	146	
10	...	71651.71	88	
11	...	71651.71	88	
12	...	62234.79	138	
13	...	73744.83	68	
14	...	67110.88	146	
15	...	73744.83	68	
16	...	67110.88	146	
17	...	69893.21	124	
18	...	73744.83	68	
19	...	73744.83	68	
20	...	67110.88	146	
21	...	76386.58	60	
22	...	69893.21	124	
23	...	69893.21	124	
24	...	76386.58	60	
25	...	76386.58	60	
26	...	76386.58	60	
27	...	69893.21	124	
28	...	71583.00	90	
29	...	71372.60	94	
..	...	...	...	
90	...	74506.69	68	
91	...	67954.58	136	
92	...	73259.61	62	
93	...	69301.98	122	
94	...	69301.98	122	
95	...	62234.79	138	
96	...	71687.30	86	
97	...	73645.78	64	
98	...	76323.88	62	
99	...	73645.78	64	
100	...	69228.59	122	
101	...	71762.17	90	
102	...	71024.09	96	

103	...	71422.87	86
104	...	71762.17	90
105	...	62438.38	138
106	...	71651.71	88
107	...	62234.79	138
108	...	71502.80	96
109	...	72286.02	96
110	...	71761.09	88
111	...	61737.31	142
112	...	72141.38	86
113	...	62438.38	138
114	...	66564.88	136
115	...	72030.06	88
116	...	67219.68	144
117	...	73744.83	68
118	...	73848.41	70
119	...	66564.88	136

	total_NN	total_NN_transactions	total_benchmark	GaussianNB_FinalValue	\
0	70913.04	428	74037.88	70355.21	
1	78494.64	208	74037.88	74346.51	
2	70437.85	436	74037.88	70837.52	
3	76987.90	162	74037.88	76818.63	
4	75660.41	162	74037.88	76808.73	
5	70437.85	436	74037.88	70837.52	
6	75532.20	168	74037.88	76808.73	
7	74075.27	276	74037.88	72468.35	
8	67412.39	426	74037.88	64216.47	
9	66547.47	482	74037.88	68321.55	
10	74075.27	276	74037.88	72468.35	
11	74075.27	276	74037.88	72468.35	
12	67412.39	426	74037.88	64216.47	
13	77505.77	208	74037.88	74346.51	
14	66547.47	482	74037.88	68321.55	
15	77505.77	208	74037.88	74346.51	
16	66547.47	482	74037.88	68321.55	
17	70046.18	438	74037.88	70795.43	
18	77505.77	208	74037.88	74346.51	
19	77505.77	208	74037.88	74346.51	
20	66547.47	482	74037.88	68321.55	
21	76505.30	160	74037.88	76631.59	
22	70046.18	438	74037.88	70795.43	
23	70046.18	438	74037.88	70795.43	
24	76505.30	160	74037.88	76631.59	
25	76505.30	160	74037.88	76631.59	
26	76505.30	160	74037.88	76631.59	
27	70046.18	438	74037.88	70795.43	
28	76350.15	52	74037.88	72400.05	

29	73366.95	324	74037.88	72201.41
..	...	...	...	...
90	78255.87	196	74037.88	74994.09
91	67871.86	482	74037.88	69088.19
92	76505.30	162	74037.88	73963.77
93	71519.51	446	74037.88	70302.78
94	71519.51	446	74037.88	70302.78
95	67412.39	426	74037.88	64216.47
96	93475.52	40	74037.88	72508.51
97	76498.13	162	74037.88	74282.11
98	75732.33	166	74037.88	76568.40
99	76498.13	162	74037.88	74282.11
100	69339.79	438	74037.88	70240.40
101	91098.91	38	74037.88	72552.34
102	73231.91	334	74037.88	71895.28
103	90704.68	44	74037.88	72283.74
104	91098.91	38	74037.88	72552.34
105	67131.81	450	74037.88	64389.52
106	74075.27	276	74037.88	72468.35
107	67412.39	426	74037.88	64216.47
108	73232.51	318	74037.88	72302.18
109	72871.88	330	74037.88	72967.92
110	74208.37	278	74037.88	72561.33
111	69437.54	436	74037.88	63773.81
112	74127.82	284	74037.88	72894.47
113	67131.81	450	74037.88	64389.52
114	67803.96	488	74037.88	67906.95
115	78448.76	258	74037.88	72789.95
116	68393.86	488	74037.88	68423.93
117	77681.49	210	74037.88	74346.51
118	78530.02	204	74037.88	74424.65
119	67803.96	488	74037.88	67906.95

	benchmark_FinalValue	NN_FinalValue	Adaboost_FinalValue	\
0	74853.0	70157.48	71068.91	
1	74853.0	77690.84	75092.96	
2	74853.0	69713.97	70005.58	
3	74853.0	76637.82	76298.59	
4	74853.0	75509.45	76883.90	
5	74853.0	69713.97	70005.58	
6	74853.0	75370.77	76454.86	
7	74853.0	73597.78	79476.14	
8	74853.0	67191.83	60078.57	
9	74853.0	66179.45	65627.41	
10	74853.0	73597.78	79476.14	
11	74853.0	73597.78	79476.14	
12	74853.0	67191.83	60078.57	
13	74853.0	76850.30	75381.30	

14	74853.0	66179.45	65627.41
15	74853.0	76850.30	75381.30
16	74853.0	66179.45	65627.41
17	74853.0	69371.15	70904.76
18	74853.0	76850.30	75381.30
19	74853.0	76850.30	75381.30
20	74853.0	66179.45	65627.41
21	74853.0	76237.51	77347.52
22	74853.0	69371.15	70904.76
23	74853.0	69371.15	70904.76
24	74853.0	76237.51	77347.52
25	74853.0	76237.51	77347.52
26	74853.0	76237.51	77347.52
27	74853.0	69371.15	70904.76
28	74853.0	76640.23	81888.85
29	74853.0	72758.11	71683.96
..	...	...	...
90	74853.0	77547.29	75136.96
91	74853.0	67305.18	65963.89
92	74853.0	76227.61	76867.52
93	74853.0	70583.88	68878.10
94	74853.0	70583.88	68878.10
95	74853.0	67191.83	60078.57
96	74853.0	89908.64	82523.62
97	74853.0	76221.51	76403.54
98	74853.0	75550.78	76762.35
99	74853.0	76221.51	76403.54
100	74853.0	68770.72	71076.88
101	74853.0	88136.08	84835.81
102	74853.0	72593.82	71152.95
103	74853.0	87810.71	79869.31
104	74853.0	88136.08	84835.81
105	74853.0	66834.54	60674.80
106	74853.0	73597.78	79476.14
107	74853.0	67191.83	60078.57
108	74853.0	72673.53	71551.38
109	74853.0	72307.60	72623.96
110	74853.0	73701.01	75501.53
111	74853.0	68863.71	60619.97
112	74853.0	73602.85	78524.32
113	74853.0	66834.54	60674.80
114	74853.0	67217.77	65060.07
115	74853.0	77404.35	78026.70
116	74853.0	67719.18	66046.79
117	74853.0	76989.77	75088.71
118	74853.0	77740.72	75641.90
119	74853.0	67217.77	65060.07

	DecisionTree_FinalValue
0	71798.80
1	76534.52
2	72922.36
3	75623.93
4	73938.72
5	72922.36
6	78145.54
7	76627.90
8	66674.76
9	76539.81
10	76627.90
11	76627.90
12	66674.76
13	74888.50
14	76539.81
15	74888.50
16	76539.81
17	70605.93
18	74888.50
19	74888.50
20	76539.81
21	78818.25
22	70605.93
23	70605.93
24	78818.25
25	78818.25
26	78818.25
27	70605.93
28	74937.60
29	71777.58
..	...
90	74605.80
91	70421.13
92	75560.68
93	67833.00
94	67833.00
95	66674.76
96	72332.52
97	80458.76
98	73853.25
99	80458.76
100	74857.83
101	83357.62
102	66380.96
103	74010.72
104	83357.62
105	75237.63

```
106               76627.90
107               66674.76
108               76123.40
109               72504.46
110               72968.57
111               82781.22
112               75358.28
113               75237.63
114               66677.76
115               69943.05
116               75631.11
117               72465.86
118               77260.90
119               66677.76

[120 rows x 97 columns]
```

0.0.23 Get the top performing model of each model

```
In [24]: max_benchmark_FinalValue = result_df['benchmark_FinalValue'].iloc[0]
         max_NN_FinalValue = 0
         max_GaussianNB_FinalValue = 0
         max_Adaboost_FinalValue = 0
         max_DecisionTree_FinalValue = 0
         max_SVC_FinalValue = 0

         row_benchmark = result_df.iloc[0]
         row_NN = None
         row_GaussianNB = None
         row_Adaboost = None
         row_DecisionTree = None
         row_SVC = None

         for index, row in result_df.iterrows():
             if row['NN_FinalValue'] > max_NN_FinalValue:
                 row_NN = copy.deepcopy(row)
                 max_NN_FinalValue = row['NN_FinalValue']

             if row['GaussianNB_FinalValue'] > max_GaussianNB_FinalValue:
                 row_GaussianNB = copy.deepcopy(row)
                 max_GaussianNB_FinalValue = row['GaussianNB_FinalValue']

             if row['Adaboost_FinalValue'] > max_Adaboost_FinalValue:
                 row_Adaboost = copy.deepcopy(row)
                 max_Adaboost_FinalValue = row['Adaboost_FinalValue']

             if row['DecisionTree_FinalValue'] > max_DecisionTree_FinalValue:
```

```
                row_DecisionTree = copy.deepcopy(row)
                max_DecisionTree_FinalValue = row['DecisionTree_FinalValue']

            #if row['SVC_FinalValue'] > max_SVC_FinalValue:
            #    row_SVC = copy.deepcopy(row)
            #    max_SVC_FinalValue = row['SVC_FinalValue']

        print("benchmark FinalValue: {}".format(max_benchmark_FinalValue))
        print("Max NN FinalValue: {}".format(max_NN_FinalValue))
        print("Max GaussianNB FinalValue: {}".format(max_GaussianNB_FinalValue))
        print("Max Adaboost FinalValue: {}".format(max_Adaboost_FinalValue))
        print("Max DecisionTree FinalValue: {}".format(max_DecisionTree_FinalValue))
        #print("Max SVC FinalValue: {}".format(max_SVC_FinalValue))
```

```
benchmark FinalValue: 74853.0
Max NN FinalValue: 90873.81
Max GaussianNB FinalValue: 76818.63
Max Adaboost FinalValue: 84835.81
Max DecisionTree FinalValue: 83357.62
```

0.0.24 Get summary of the top models

```
In [25]: #model_list = [('benchmark',row_benchmark),('NN',row_NN),('GaussianNB',row_GaussianNB
         #               ('SVC',row_SVC),('Adaboost',row_Adaboost),('DecisionTree',row_Decisio
         ## No SVC
         model_list = [('benchmark',row_benchmark),('NN',row_NN),('GaussianNB',row_GaussianNB)
                       ('Adaboost',row_Adaboost),('DecisionTree',row_DecisionTree)]

         best_model = None
         best_model_val = 0

         for name, model in model_list:
             ## Get all data relevant to each model
             for index_name in model.index:
                 if name not in index_name and index_name not in ["sell_below","buy_above","ta
                 #if name not in index_name:
                     del model[index_name]

             print("\n{} Data: \n{}".format(name,model))

             if model[name+"_FinalValue"] > best_model_val:
                 best_model_val = model[name+"_FinalValue"]
                 best_model = name + "_" + model["target_ratio"] + "_" + str(model["sell_below"
                                  + "_" + str(model["buy_above"])
```

```
benchmark Data:
```

```
BA_benchmark            18139.2
CHK_benchmark            9749.5
CMG_benchmark           8140.79
FCEL_benchmark          2896.68
HON_benchmark           11879.7
MA_benchmark            13322.6
SD_benchmark            8545.85
TPLM_benchmark          1363.63
buy_above                  1.01
sell_below                    1
target_ratio                vr5
total_benchmark         74037.9
benchmark_FinalValue      74853
Name: 0, dtype: object

NN Data:
BA_NN                     10000
BA_NN_transactions            0
CHK_NN                    10000
CHK_NN_transactions           0
CMG_NN                    10000
CMG_NN_transactions           0
FCEL_NN                 9465.17
FCEL_NN_transactions         10
HON_NN                    10000
HON_NN_transactions           0
MA_NN                     10000
MA_NN_transactions            0
NN Accuracy             0.52506
NN Fscore               0.31867
SD_NN                     10000
SD_NN_transactions            0
TPLM_NN                 25402.8
TPLM_NN_transactions         46
buy_above                  1.01
sell_below                 1.01
target_ratio             vClose
total_NN                  94868
total_NN_transactions        56
NN_FinalValue           90873.8
Name: 30, dtype: object

GaussianNB Data:
BA_GaussianNB                     16577.6
BA_GaussianNB_transactions              2
CHK_GaussianNB                    10001.8
CHK_GaussianNB_transactions            14
CMG_GaussianNB                     8169.4
```

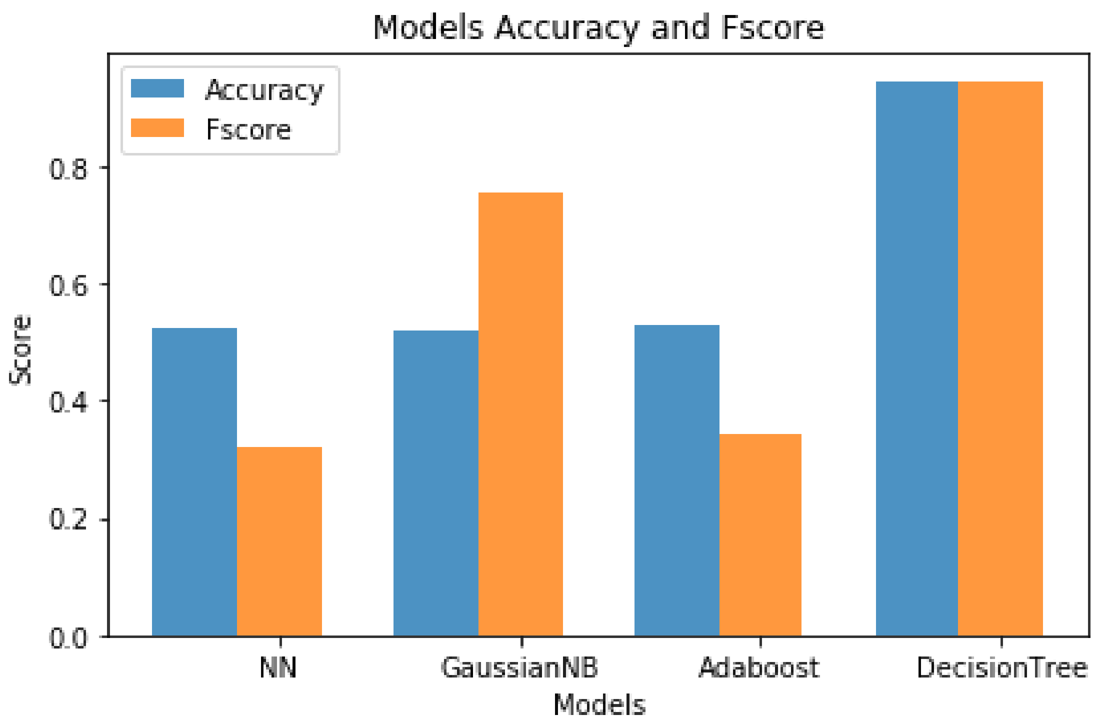
```
CMG_GaussianNB_transactions              6
FCEL_GaussianNB                    9371.85
FCEL_GaussianNB_transactions             8
GaussianNB Accuracy               0.521135
GaussianNB Fscore                 0.754801
HON_GaussianNB                     10430.8
HON_GaussianNB_transactions              2
MA_GaussianNB                      10216.5
MA_GaussianNB_transactions               2
SD_GaussianNB                      8904.41
SD_GaussianNB_transactions               6
TPLM_GaussianNB                    2934.24
TPLM_GaussianNB_transactions            20
buy_above                             1.01
sell_below                               1
target_ratio                          vr40
total_GaussianNB                   76606.6
total_GaussianNB_transactions           60
GaussianNB_FinalValue              76818.6
Name: 3, dtype: object

Adaboost Data:
Adaboost Accuracy               0.528684
Adaboost Fscore                 0.343793
BA_Adaboost                        10000
BA_Adaboost_transactions               0
CHK_Adaboost                     9833.75
CHK_Adaboost_transactions             14
CMG_Adaboost                       10000
CMG_Adaboost_transactions              0
FCEL_Adaboost                    14118.2
FCEL_Adaboost_transactions            36
HON_Adaboost                       10000
HON_Adaboost_transactions              0
MA_Adaboost                        10000
MA_Adaboost_transactions               0
SD_Adaboost                      9588.21
SD_Adaboost_transactions               2
TPLM_Adaboost                    13422.4
TPLM_Adaboost_transactions            26
buy_above                           1.01
sell_below                         1.005
target_ratio                      vClose
total_Adaboost                   86962.6
total_Adaboost_transactions           78
Adaboost_FinalValue              84835.8
Name: 101, dtype: object
```

```
DecisionTree Data:
BA_DecisionTree                      10714.1
BA_DecisionTree_transactions              74
CHK_DecisionTree                     10049.8
CHK_DecisionTree_transactions            114
CMG_DecisionTree                     9551.77
CMG_DecisionTree_transactions             86
DecisionTree Accuracy               0.944746
DecisionTree Fscore                 0.944735
FCEL_DecisionTree                    5306.21
FCEL_DecisionTree_transactions           102
HON_DecisionTree                     10574.8
HON_DecisionTree_transactions             54
MA_DecisionTree                        11038
MA_DecisionTree_transactions              44
SD_DecisionTree                      9046.95
SD_DecisionTree_transactions              58
TPLM_DecisionTree                    22392.7
TPLM_DecisionTree_transactions           104
buy_above                               1.01
sell_below                             1.005
target_ratio                          vClose
total_DecisionTree                   88674.4
total_DecisionTree_transactions          636
DecisionTree_FinalValue              83357.6
Name: 101, dtype: object
```

0.0.25 Graph model's score

In [26]:
```
accuracies = []
fscores = []
names = []

for name, model in model_list:
    if name != "benchmark":
        accuracies.append(model[name+" Accuracy"])
        fscores.append(model[name+" Fscore"])
        names.append(name)


n_groups = len(model_list) - 1 ## subtract benchmark


## create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
```

```
opacity = 0.8

# Bar features
rects1 = plt.bar(index, accuracies, bar_width,alpha=opacity,label="Accuracy")
rects2 = plt.bar(index+bar_width, fscores, bar_width,alpha=opacity,label="Fscore")

## Labels
plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Models Accuracy and Fscore')
plt.xticks(index + bar_width, names)
plt.legend()

plt.tight_layout()
plt.show()
fig.savefig("figures/Scores.svg", format='svg', dpi=1200)
```



0.0.26 Graph each stocks performance of the best optimal model

```
In [27]: print(best_model)
```

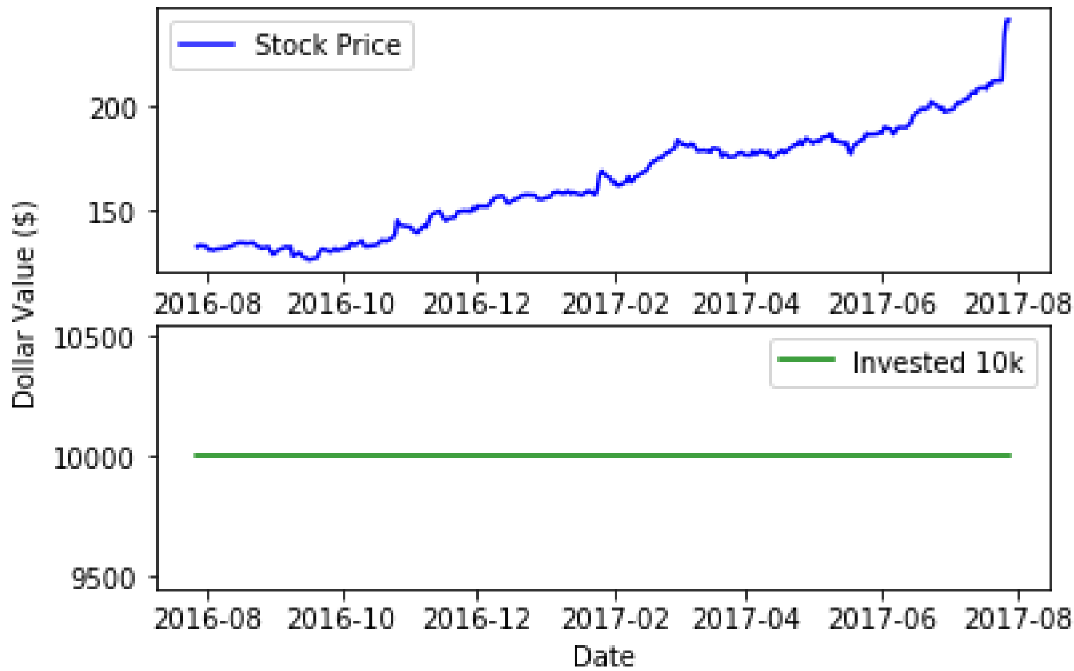
```
NN_vClose_1.01_1.01
```

```
In [28]: graph_data = []

         ## Get test data that used best model
         filelist = glob.glob("model_predictions/*")
         for f in filelist:
             if best_model+".csv" in f:
                 graph_data.append(f)

         print(graph_data)
```

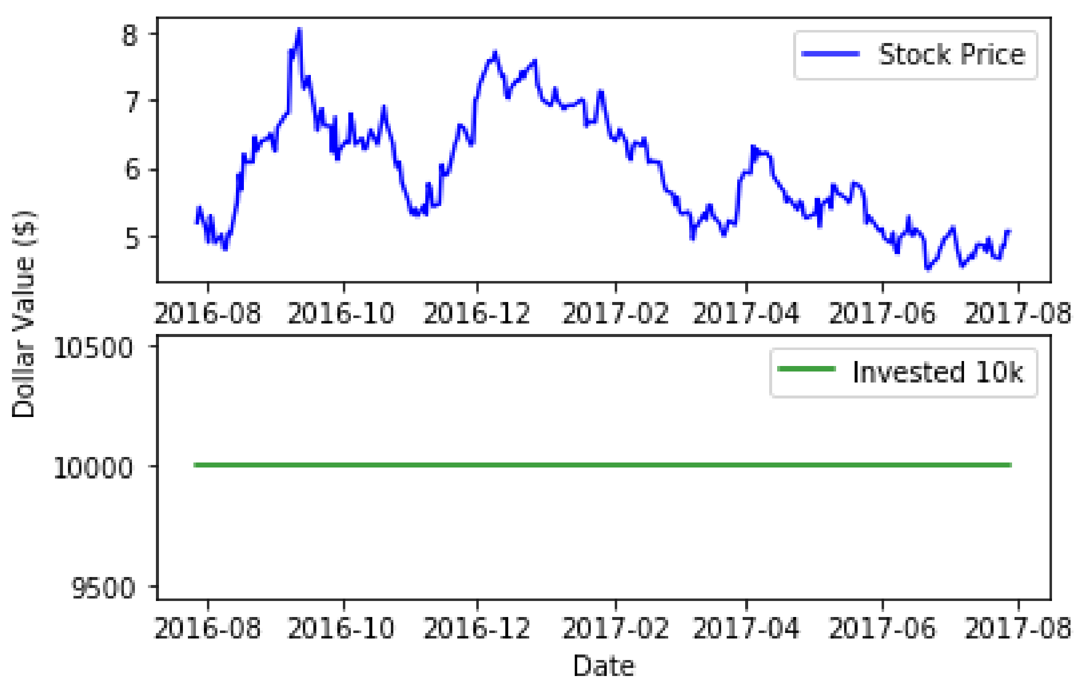
```
['model_predictions/BA_NN_vClose_1.01_1.01.csv', 'model_predictions/CHK_NN_vClose_1.01_1.01.cs
```

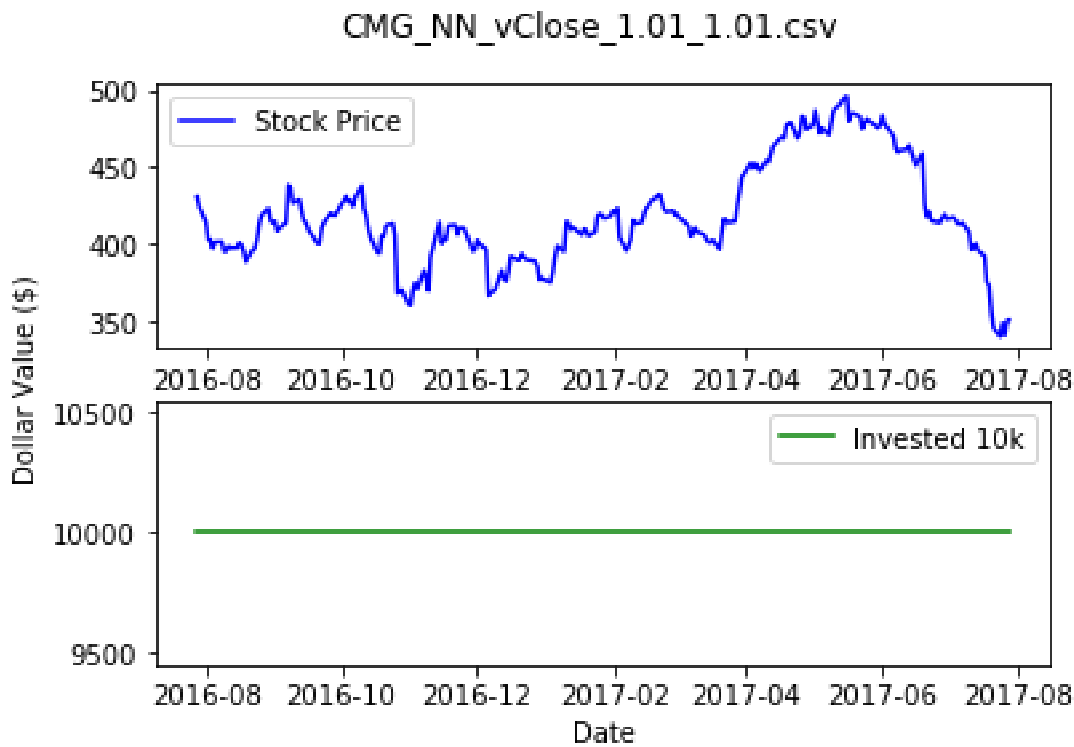
```
In [29]: def plot_data(f,df):
             remove = len("model_predictions/")
             file = f[remove:]
             tick = file.split('_')[0]

             fig = plt.figure()
             ## Top plot
             df0 = pd.DataFrame.from_csv("testing_data/"+tick+"_processed.csv")
             price = df0['vClose']
             ax1 = fig.add_subplot(2,1,1)
             ax1.plot(price.index,price,label='Stock Price',color='b')
             ax1.legend()

             ## Bottom plot
             money = df['Money']
             ax2 = fig.add_subplot(2,1,2)
             ax2.plot(money.index,money,label='Invested 10k',color='g')
             ax2.set_xlabel('Date')
             ax2.legend()

             fig.text(0.00, 0.5, 'Dollar Value ($)', va='center', rotation='vertical')

             plt.suptitle(file)
             plt.show()
             #fig.savefig("figures/"+file+'.png')
             fig.savefig("figures/"+file+'.svg', format='svg', dpi=1200)
             plt.close(fig)


         for f in graph_data:
             df = pd.DataFrame.from_csv(f)

             plot_data(f,df)
```

BA_NN_vClose_1.01_1.01.csv

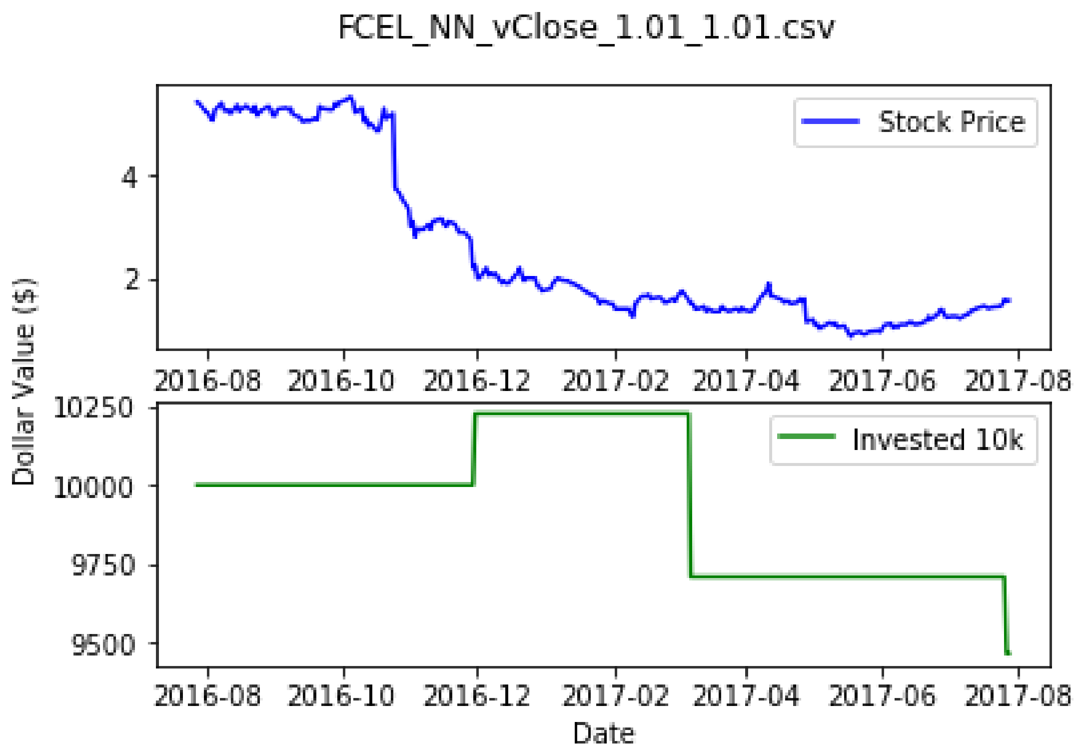


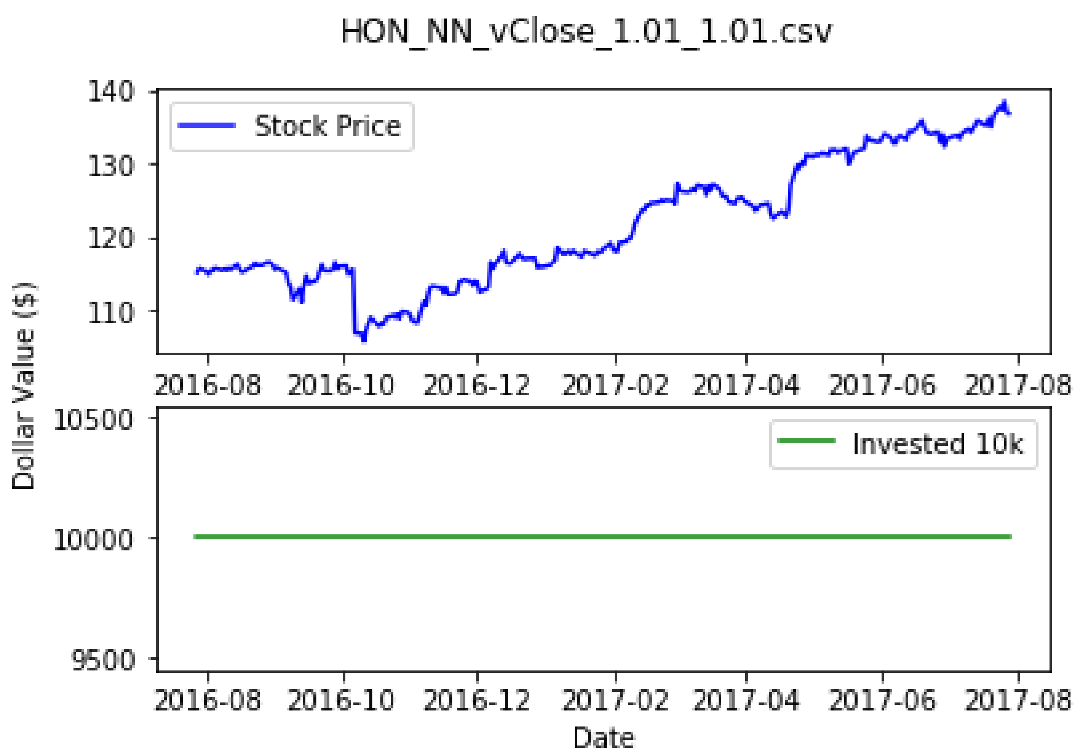
CHK_NN_vClose_1.01_1.01.csv

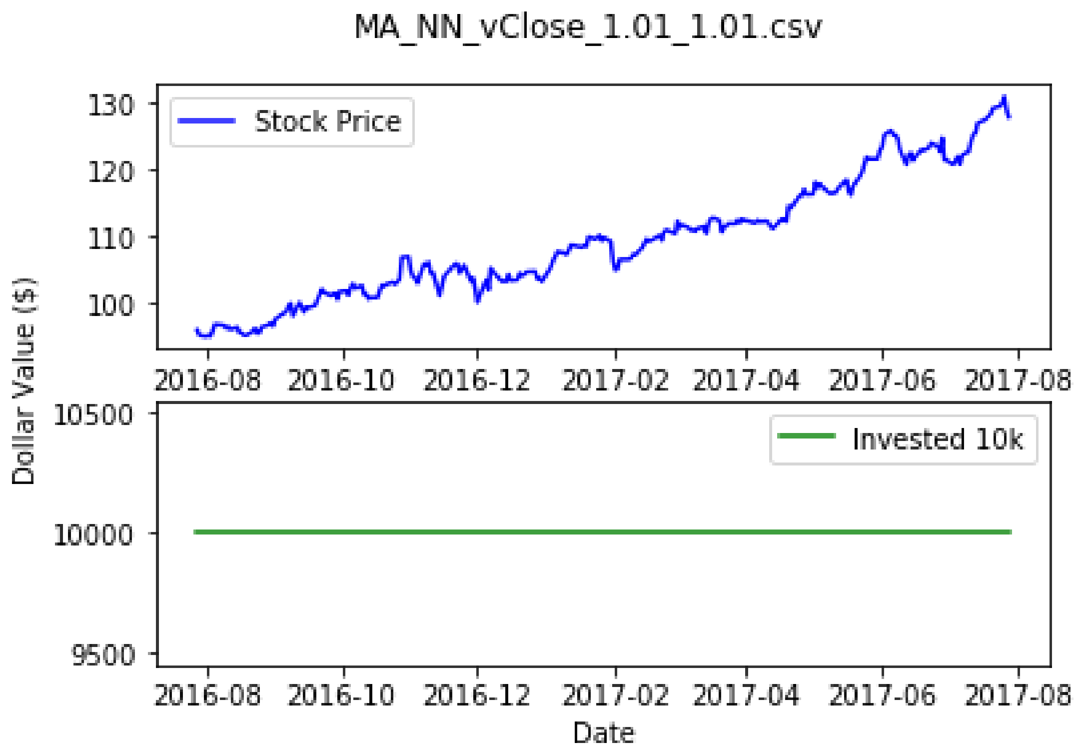


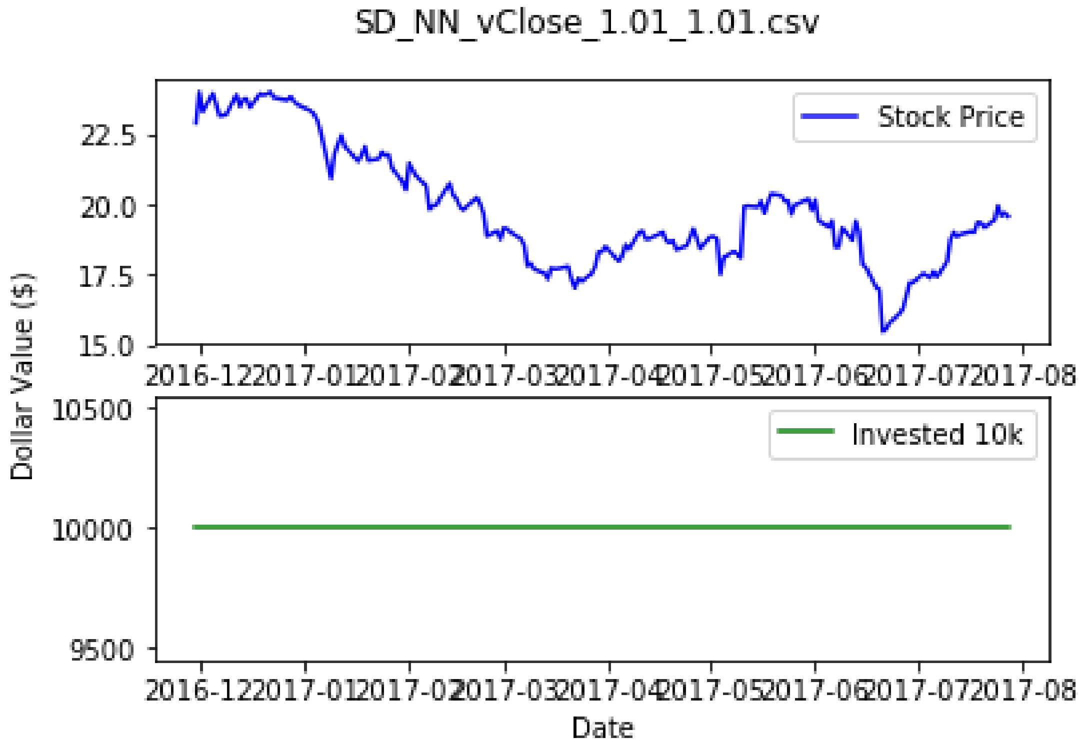


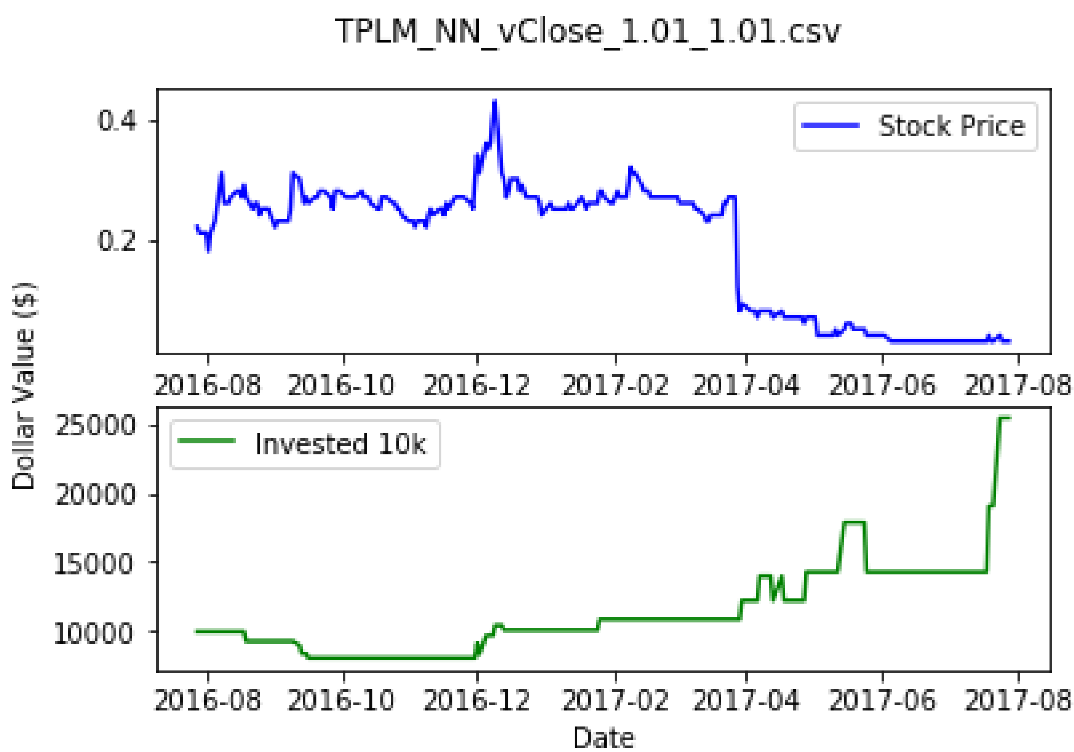
CMG_NN_vClose_1.01_1.01.csv
Stock Price
Invested 10k
Dollar Value ($)
Date
500
450
400
350
10500
10000
9500
2016-08
2016-10
2016-12
2017-02
2017-04
2017-06
2017-08


FCEL_NN_vClose_1.01_1.01.csv
Stock Price
Invested 10k
Dollar Value ($)
Date
4
2
10250
10000
9750
9500
2016-08
2016-10
2016-12
2017-02
2017-04
2017-06
2017-08


HON_NN_vClose_1.01_1.01.csv
Stock Price
Invested 10k
Dollar Value ($)
Date
140
130
120
110
10500
10000
9500
2016-08
2016-10
2016-12
2017-02
2017-04
2017-06
2017-08


MA_NN_vClose_1.01_1.01.csv
Stock Price
130
120
110
100
2016-08
2016-10
2016-12
2017-02
2017-04
2017-06
2017-08
Dollar Value ($)
Invested 10k
10500
10000
9500
Date


SD_NN_vClose_1.01_1.01.csv
Stock Price
Invested 10k
Dollar Value ($)
Date
22.5
20.0
17.5
15.0
10500
10000
9500
2016-12
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08


TPLM_NN_vClose_1.01_1.01.csv
Stock Price
Invested 10k
Dollar Value ($)
Date
0.4
0.2
25000
20000
15000
10000
2016-08
2016-10
2016-12
2017-02
2017-04
2017-06
2017-08

0.0.27 Helper function for create bar graph

```
In [30]: def bar_graph(graph_data,number_of_groups, filename, xlabel, title, xticks):
             n_groups = number_of_groups

             ## create plot
             fig, ax = plt.subplots()
             index = np.arange(n_groups)
             bar_width = 0.167   ## each bar need 100%.  There are 5 models plus the space, so
             opacity = 0.8

             # Bar features
             rects = {}
             pos = 0
             for k, v in graph_data.items():
                 ## pos moves the bar through x-axis
                 rects[k] = plt.bar(index+pos, v, bar_width,alpha=opacity,label=k)
                 pos += bar_width

             ## Labels
             plt.xlabel(xlabel)
             plt.ylabel('Value')
             plt.title(title)
             plt.xticks(index + bar_width, xticks)
             plt.legend()

             plt.tight_layout()
             plt.show()
             fig.savefig("figures/"+filename+".svg", format='svg', dpi=1200)
```

0.0.28 Gather data of each stock for each optimal models

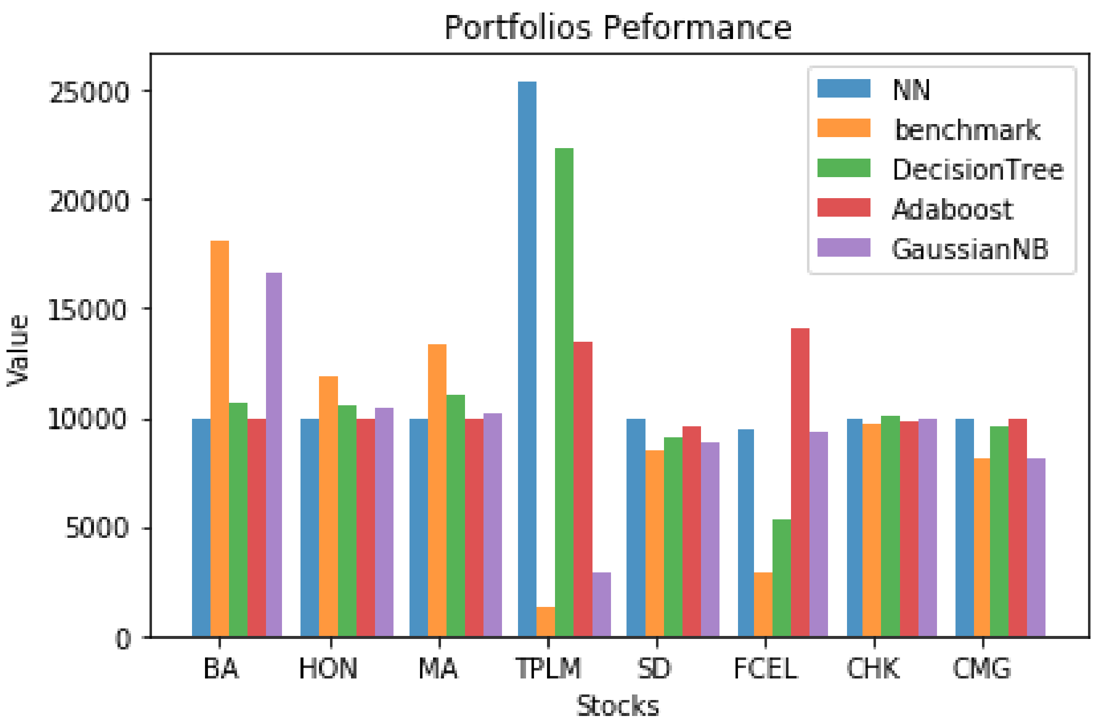
```
In [31]: graph_data = {}

         for name, model in model_list:
             graph_data[name] = []

         for tick in TESTING_TICKERS:
             for name, model in model_list:
                 for index_name in model.index:
                     ## Get data about that ticker
                     if tick in index_name and 'transactions' not in index_name:
                         graph_data[name].append(model[index_name])
```

```
print(graph_data)
```

```
{'NN': [10000.0, 10000.0, 10000.0, 25402.84, 10000.0, 9465.17, 10000.0, 10000.0], 'benchmark':
```

0.0.29 Graph Models

```
In [32]: bar_graph(graph_data = graph_data, \
                   number_of_groups = len(TESTING_TICKERS),\
                   filename = 'Portfolio',\
                   xlabel = 'Stocks', \
                   title = 'Portfolios Peformance',\
                   xticks = TESTING_TICKERS)
```



0.0.30 Gather data of each optimal models

```
In [33]: graph_data = {}

         for name, model in model_list:
             graph_data[name] = []

         tick = 'FinalValue'
         for name, model in model_list:
             for index_name in model.index:
```

```
        ## Get data about that ticker
        if tick in index_name and 'transactions' not in index_name:
            graph_data[name].append(model[index_name])

print(graph_data)
```

```
{'NN': [90873.81], 'benchmark': [74853.0], 'DecisionTree': [83357.62], 'Adaboost': [84835.81],
```

0.0.31 Graph Total Final Value of the optimal models

```
In [34]: bar_graph(graph_data = graph_data, \
                   number_of_groups = 1,\
                   filename = 'Portfolios_Total',\
                   xlabel = 'Models',\
                   title = 'Portfolios Peformance',\
                   xticks = "")
```