Virtual bidding in NYISO's markets

In this lab, we will implement a simple virtual trading strategy in New York ISO's electricity markets. The goal is to maximize profits. We shall train our model on price data from one year, and implement the strategy on the data from the next year. How much can you earn with a certain daily budget? Say $250K?

We will present a trading strategy. You are welcome to try other strategies and compare the gains over multiple runs.

Let's start with customary imports.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import choice
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import pickle

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
```

Load the day-ahead and real-time prices from 11 zones in New York.

The day-ahead prices are defined every hour. The real-time market runs every 5 minutes. For each zone, an average of these prices over an hour is published.

Store the list of zones in the variable 'listOfZones'. Also, store the number of options as the number of zones times the 24 hours available for trading. Finally create another list containing the option names (zone + hour).

In [2]:
```
listOfZones = ['CAPITL', 'CENTRL', 'DUNWOD', 'GENESE', 'HUD VL', 'LONG
               'MILLWD', 'N.Y.C.', 'NORTH', 'WEST']

nOptions = len(listOfZones) * 24
optionNames = [zone + "_Hour_" + str(t) for zone in listOfZones for t
```

Parse the files with DA and RT prices along with DA load forecast.

Define a function that parses three files containing DA and RT prices, along with DA load predictions from a year from all different zones in the list defined before. This function will be used for to both load the data for training the classifiers and testing them. This function has 3 outputs: they are DA prices, difference between DA and RT prices, and finally DA load predictions. The outputs are pandas data frames whose columns are the options, and rows are the days in the year.

In [3]:
```
def loadNYISOData(year):

    # Open the relevant files for DA prices, RT prices, DA load.

    dfPriceDA = pd.read_csv("DAM_NYISO_Zonal_LBMP_" + str(year) + ".cs
    dfPriceRT = pd.read_csv("RTM_NYISO_Zonal_LBMP_" + str(year) + ".cs
    dfLoadDA = pd.read_csv("DAM_NYISO_LoadForecast_" + str(year) + ".c

    # Collect the DA and RT prices from each zone from each hour and c
    # The data should have prices and loads from all days of a year, w
    # contributes 24 rows, corresponding to each hour.

    priceDA = pd.DataFrame({zone: (dfPriceDA.loc[dfPriceDA['Zone Name'
                                                'DAM Zonal LBMP']).va
                            for zone in listOfZones})
    priceRT = pd.DataFrame({zone: (dfPriceRT.loc[dfPriceRT['Zone Name'
                                                'TWI Zonal LBMP']).va
                            for zone in listOfZones})
    loadDA = pd.DataFrame({zone: (dfLoadDA.loc[dfLoadDA['Zone Name'] =
                                              'DAM Forecast Load']).v
                           for zone in listOfZones})

    numberOfDays = int(len(priceDA.index)/24)

    # Compute the price differences between DA and RT prices for all c
    # all days of the year. Store it as a pandas data frame where the 
    # each day is flattened into one row. This operation essentially a
    # independently think of each zone in each hour as a separate opti
    # reshape the prices for the DA market in the same manner.

    priceDART = pd.DataFrame([priceRT.sub(priceDA).loc[day * 24:
                                                      (day + 1
                              listOfZones].values.flatten()
                              for day in range(numberOfDays)],
                             columns=optionNames)

    priceDA = pd.DataFrame([priceDA.loc[day * 24: (day + 1) * 24 - 1,
                            listOfZones].values.flatten()
                            for day in range(numberOfDays)],
                           columns=optionNames)

    return priceDA, priceDART, loadDA
```

Create a function that creates the inputs for training a classifier

Create a function that takes the price and load data and creates two arrays 'X' and 'Y'. Essentially, the rows of 'X' contains all information relevant to predicting the sign of the price difference on the various options on the next day. It takes as an input, three pandas frames corresponding to the DA prices, price differences, and the DA load predictions, and produces three outputs: the arrays 'X', 'Y', and the range of days from the year that were used to create the data 'X' and 'Y'. This function will be used to both train and test classifiers.

In [4]:
```
def createClassifierIO(priceDA, priceDART, loadDA):

    # Define how many past days of prices to use for classification.

    pastPrices = range(1, 3)

    # Define how many past days of load predictions to use for classif

    pastLoad = range(1, 3)

    # Define a date range within the year to create the arrays 'X' and
    # that past price and load data for the first day is within the da
    # pandas frames passed as inputs.

    rangeOfDays = range(3, len(priceDA.index))

    # 'X' will contain three sets of variables:
    #   1. the DA prices from past days in the list 'pastDays',
    #   2. the differences between DA and RT prices from the same past
    #   3. the load predictions from past days in the list 'pastLoad'

    X = [np.concatenate((
        priceDA.loc[[(day - h) for h in pastPrices]].values.flatten(),
        priceDART.loc[[(day - h) for h in pastPrices]].values.flatten(
        loadDA.loc[[(day - h) for h in pastLoad]].values.flatten()
    )) for day in rangeOfDays]

    # Scale the array 'X' to make its data zero mean and unit variance
    X = StandardScaler().fit_transform(X)

    # 'Y' will contain zeros and ones, where a one indicates that the
    # higher than in RT for a particular option. Recall that an option
    # a zone at a particular hour of the day.

    Y = np.array([(priceDART.loc[day].values > 0).astype(int)
                  for day in rangeOfDays])

    # Return the arrays 'X' and 'Y', and finally the range of days fro
    # will be utilized for training or testing the classifier.
    return X, Y, rangeOfDays
```

Design the training module.

The training module utilizes a year's worth of data to determine the following for each option, i.e., for each zone for each hour of the day:

1. Classifiers that predict the sign of the difference between DA and RT prices.
2. Statistics of the mean of the price difference.
3. A quantile of the day-ahead prices that we will use as our bid for each option. You will either train the classifiers here or load them from the folder './Classifiers'. Storing the classifiers from time to time allows you to only vary the bidding strategy and observe the annual reward rather than having to train the classifiers every time.

Define and train the classifiers or load pre-trained classifiers.

In [5]:
```
classifiers = []

# We have two options here. Use previous training experience, or learn
useSavedClassifiers = False

if not useSavedClassifiers:

    print("Starting training module...\n")
    trainPriceDA, trainPriceDART, trainLoadDA = loadNYISOData(2015)

    numberOfDays =  int(len(trainPriceDA.index))
    print("Loaded hourly prices from 2015 for %d days." % numberOfDays

    # We will implement a trading strategy, where we bid a particular
    # DA prices for an option. If you do not know what a quantile mean
    # article on it. Essentially, a 95% quantile of the DA prices equa
    # 95% of the DA prices are below it. Store all quantiles starting
    # 5% in a dictionary. Store them in a pickle file.

    quantilesToStore = [0.70, 0.75, 0.80, 0.85, 0.90, 0.95]
    offerPrices = trainPriceDA.quantile(q=quantilesToStore).transpose(
    pickle.dump(offerPrices, open("./Training/OfferPrices", 'wb'))

    # Calculate the average price spread for each option over the enti
    # us in choosing our portfolio. Store it as a dictionary. Our bid
    # options that our classifier indicates that they will be profitab
    # have higher average price differences, indicating that they have
    # Store them using pickle.

    averagePriceSpread = trainPriceDART.mean(axis=0).transpose().to_di
    pickle.dump(averagePriceSpread, open("./Training/AveragePriceSprea
    # Create the training dataset using the function 'createClassifier
    # loads, and store them in 'trainX', and 'trainY'.

    trainX, trainY, _ = createClassifierIO(trainPriceDA, trainPriceDAR

    # Define a collection of classifiers, one for each option. You can
    # as that based on an SVM, logistic regression, multilayer percept
    # measure training accuracy to indicate how well the classifier wo
    # However, good training accuracy does not always indicate good te
    # Avoid over-fitting.


    classifiers = [MLPClassifier(hidden_layer_sizes=(20, 10), max_iter
                   for _ in range(nOptions)]

    trainingAccuracy = 0

    for ii in range(nOptions):
        classifiers[ii].fit(trainX, trainY[:, ii])
        print("Classifier trained for option " + optionNames[ii])
        trainingAccuracy += classifiers[ii].score(trainX, trainY[:, ii

        # Store the classifier.
        pickle.dump(classifiers[ii], open("./Training/Classifier_" + o

    print("\nOverall training accuracy = %1.2f percent." % (100 * trai

    del numberOfDays, trainPriceDA, trainLoadDA, trainPriceDART, trair
else:

    # Load the classifiers, the offer prices at various quantiles, and

    print("Loading previously trained variables...\n")
    classifiers = [pickle.load(open("./Training/Classifier_" + optionN
                   for ii in range(nOptions)]
    offerPrices = pickle.load(open("./Training/OfferPrices", 'rb'))
    averagePriceSpread = pickle.load(open("./Training/AveragePriceSpre

    print("All training variables were loaded successfully...\n")
```

```
Starting training module...

Loaded hourly prices from 2015 for 365 days.
Classifier trained for option CAPITL_Hour_0
Classifier trained for option CAPITL_Hour_1
Classifier trained for option CAPITL_Hour_2
Classifier trained for option CAPITL_Hour_3
Classifier trained for option CAPITL_Hour_4
Classifier trained for option CAPITL_Hour_5
Classifier trained for option CAPITL_Hour_6
Classifier trained for option CAPITL_Hour_7
Classifier trained for option CAPITL_Hour_8
Classifier trained for option CAPITL_Hour_9
Classifier trained for option CAPITL_Hour_10
Classifier trained for option CAPITL_Hour_11
Classifier trained for option CAPITL_Hour_12
Classifier trained for option CAPITL_Hour_13
Classifier trained for option CAPITL_Hour_14
Classifier trained for option CAPITL_Hour_15
Classifier trained for option CAPITL_Hour_16

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option CAPITL_Hour_17
Classifier trained for option CAPITL_Hour_18
Classifier trained for option CAPITL_Hour_19
Classifier trained for option CAPITL_Hour_20
Classifier trained for option CAPITL_Hour_21
Classifier trained for option CAPITL_Hour_22
Classifier trained for option CAPITL_Hour_23
Classifier trained for option CENTRL_Hour_0
Classifier trained for option CENTRL_Hour_1
Classifier trained for option CENTRL_Hour_2
Classifier trained for option CENTRL_Hour_3
Classifier trained for option CENTRL_Hour_4
Classifier trained for option CENTRL_Hour_5
Classifier trained for option CENTRL_Hour_6
Classifier trained for option CENTRL_Hour_7
Classifier trained for option CENTRL_Hour_8
Classifier trained for option CENTRL_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option CENTRL_Hour_10
Classifier trained for option CENTRL_Hour_11
Classifier trained for option CENTRL_Hour_12
Classifier trained for option CENTRL_Hour_13
Classifier trained for option CENTRL_Hour_14
Classifier trained for option CENTRL_Hour_15
Classifier trained for option CENTRL_Hour_16
Classifier trained for option CENTRL_Hour_17
Classifier trained for option CENTRL_Hour_18
Classifier trained for option CENTRL_Hour_19
Classifier trained for option CENTRL_Hour_20
Classifier trained for option CENTRL_Hour_21
Classifier trained for option CENTRL_Hour_22

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option CENTRL_Hour_23
Classifier trained for option DUNWOD_Hour_0
Classifier trained for option DUNWOD_Hour_1
Classifier trained for option DUNWOD_Hour_2

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option DUNWOD_Hour_3
Classifier trained for option DUNWOD_Hour_4
Classifier trained for option DUNWOD_Hour_5
Classifier trained for option DUNWOD_Hour_6
Classifier trained for option DUNWOD_Hour_7
Classifier trained for option DUNWOD_Hour_8
Classifier trained for option DUNWOD_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option DUNWOD_Hour_10
Classifier trained for option DUNWOD_Hour_11

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option DUNWOD_Hour_12
Classifier trained for option DUNWOD_Hour_13
Classifier trained for option DUNWOD_Hour_14
Classifier trained for option DUNWOD_Hour_15
Classifier trained for option DUNWOD_Hour_16
Classifier trained for option DUNWOD_Hour_17

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
```

```
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option DUNWOD_Hour_18
Classifier trained for option DUNWOD_Hour_19
Classifier trained for option DUNWOD_Hour_20

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option DUNWOD_Hour_21
Classifier trained for option DUNWOD_Hour_22
Classifier trained for option DUNWOD_Hour_23

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_0
Classifier trained for option GENESE_Hour_1
Classifier trained for option GENESE_Hour_2
Classifier trained for option GENESE_Hour_3
Classifier trained for option GENESE_Hour_4

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_5
Classifier trained for option GENESE_Hour_6
Classifier trained for option GENESE_Hour_7
Classifier trained for option GENESE_Hour_8
Classifier trained for option GENESE_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_10
Classifier trained for option GENESE_Hour_11
Classifier trained for option GENESE_Hour_12
Classifier trained for option GENESE_Hour_13

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_14
Classifier trained for option GENESE_Hour_15

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_16
Classifier trained for option GENESE_Hour_17
Classifier trained for option GENESE_Hour_18

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_19
Classifier trained for option GENESE_Hour_20
Classifier trained for option GENESE_Hour_21

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option GENESE_Hour_22
Classifier trained for option GENESE_Hour_23
Classifier trained for option HUD VL_Hour_0
Classifier trained for option HUD VL_Hour_1
Classifier trained for option HUD VL_Hour_2
Classifier trained for option HUD VL_Hour_3
Classifier trained for option HUD VL_Hour_4
Classifier trained for option HUD VL_Hour_5
Classifier trained for option HUD VL_Hour_6
Classifier trained for option HUD VL_Hour_7
Classifier trained for option HUD VL_Hour_8

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
```

```
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_10

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_11
Classifier trained for option HUD VL_Hour_12

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_13
Classifier trained for option HUD VL_Hour_14

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_15
Classifier trained for option HUD VL_Hour_16
Classifier trained for option HUD VL_Hour_17
Classifier trained for option HUD VL_Hour_18
Classifier trained for option HUD VL_Hour_19
Classifier trained for option HUD VL_Hour_20
Classifier trained for option HUD VL_Hour_21

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option HUD VL_Hour_22
Classifier trained for option HUD VL_Hour_23
Classifier trained for option LONGIL_Hour_0
Classifier trained for option LONGIL_Hour_1

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option LONGIL_Hour_2
Classifier trained for option LONGIL_Hour_3
Classifier trained for option LONGIL_Hour_4
Classifier trained for option LONGIL_Hour_5
Classifier trained for option LONGIL_Hour_6
Classifier trained for option LONGIL_Hour_7
Classifier trained for option LONGIL_Hour_8
Classifier trained for option LONGIL_Hour_9
Classifier trained for option LONGIL_Hour_10
Classifier trained for option LONGIL_Hour_11
Classifier trained for option LONGIL_Hour_12
Classifier trained for option LONGIL_Hour_13
Classifier trained for option LONGIL_Hour_14
Classifier trained for option LONGIL_Hour_15
Classifier trained for option LONGIL_Hour_16
Classifier trained for option LONGIL_Hour_17
Classifier trained for option LONGIL_Hour_18
Classifier trained for option LONGIL_Hour_19
Classifier trained for option LONGIL_Hour_20
Classifier trained for option LONGIL_Hour_21

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option LONGIL_Hour_22
Classifier trained for option LONGIL_Hour_23
Classifier trained for option MHK VL_Hour_0
Classifier trained for option MHK VL_Hour_1
Classifier trained for option MHK VL_Hour_2
Classifier trained for option MHK VL_Hour_3

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_4
Classifier trained for option MHK VL_Hour_5
```

```
/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_6

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_7
Classifier trained for option MHK VL_Hour_8
Classifier trained for option MHK VL_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_10
Classifier trained for option MHK VL_Hour_11
Classifier trained for option MHK VL_Hour_12
Classifier trained for option MHK VL_Hour_13
Classifier trained for option MHK VL_Hour_14
Classifier trained for option MHK VL_Hour_15

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_16
Classifier trained for option MHK VL_Hour_17
Classifier trained for option MHK VL_Hour_18
Classifier trained for option MHK VL_Hour_19
Classifier trained for option MHK VL_Hour_20

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_21

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural
```

```
/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)


Classifier trained for option MHK VL_Hour_22

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MHK VL_Hour_23

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MILLWD_Hour_0
Classifier trained for option MILLWD_Hour_1
Classifier trained for option MILLWD_Hour_2
Classifier trained for option MILLWD_Hour_3

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MILLWD_Hour_4
Classifier trained for option MILLWD_Hour_5
Classifier trained for option MILLWD_Hour_6
Classifier trained for option MILLWD_Hour_7
Classifier trained for option MILLWD_Hour_8

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MILLWD_Hour_9
Classifier trained for option MILLWD_Hour_10
Classifier trained for option MILLWD_Hour_11
Classifier trained for option MILLWD_Hour_12
Classifier trained for option MILLWD_Hour_13
Classifier trained for option MILLWD_Hour_14
Classifier trained for option MILLWD_Hour_15
Classifier trained for option MILLWD_Hour_16
Classifier trained for option MILLWD_Hour_16
Classifier trained for option MILLWD_Hour_17

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MILLWD_Hour_18
Classifier trained for option MILLWD_Hour_19
Classifier trained for option MILLWD_Hour_20

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option MILLWD_Hour_21
Classifier trained for option MILLWD_Hour_22
Classifier trained for option MILLWD_Hour_23

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_0

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_1
Classifier trained for option N.Y.C._Hour_2
Classifier trained for option N.Y.C._Hour_3
Classifier trained for option N.Y.C._Hour_4

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_5
Classifier trained for option N.Y.C._Hour_6

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_7

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_8

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_9
Classifier trained for option N.Y.C._Hour_10
Classifier trained for option N.Y.C._Hour_11

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_12
Classifier trained for option N.Y.C._Hour_13

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_14
Classifier trained for option N.Y.C._Hour_15

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_16
Classifier trained for option N.Y.C._Hour_17

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neura
l_network/_multilayer_perceptron.py:585: ConvergenceWarning: Stocha
```

```
stic Optimizer: Maximum iterations (200) reached and the optimizati
on hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option N.Y.C._Hour_18
Classifier trained for option N.Y.C._Hour_19
Classifier trained for option N.Y.C._Hour_20
Classifier trained for option N.Y.C._Hour_21
Classifier trained for option N.Y.C._Hour_22
Classifier trained for option N.Y.C._Hour_23

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option NORTH_Hour_0
Classifier trained for option NORTH_Hour_1
Classifier trained for option NORTH_Hour_2
Classifier trained for option NORTH_Hour_3
Classifier trained for option NORTH_Hour_4
Classifier trained for option NORTH_Hour_5
Classifier trained for option NORTH_Hour_6
Classifier trained for option NORTH_Hour_7
Classifier trained for option NORTH_Hour_8
Classifier trained for option NORTH_Hour_9
Classifier trained for option NORTH_Hour_10

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option NORTH_Hour_11
Classifier trained for option NORTH_Hour_12
Classifier trained for option NORTH_Hour_13

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option NORTH_Hour_14
Classifier trained for option NORTH_Hour_15
Classifier trained for option NORTH_Hour_16

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
```

```
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option NORTH_Hour_17
Classifier trained for option NORTH_Hour_18

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option NORTH_Hour_19
Classifier trained for option NORTH_Hour_20
Classifier trained for option NORTH_Hour_21
Classifier trained for option NORTH_Hour_22
Classifier trained for option NORTH_Hour_23
Classifier trained for option WEST_Hour_0

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_1
Classifier trained for option WEST_Hour_2
Classifier trained for option WEST_Hour_3
Classifier trained for option WEST_Hour_4
Classifier trained for option WEST_Hour_5
Classifier trained for option WEST_Hour_6

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_7
Classifier trained for option WEST_Hour_8
Classifier trained for option WEST_Hour_9

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_10
Classifier trained for option WEST_Hour_11

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
```

```
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_12
Classifier trained for option WEST_Hour_13
Classifier trained for option WEST_Hour_14
Classifier trained for option WEST_Hour_15

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_16
Classifier trained for option WEST_Hour_17

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_18
Classifier trained for option WEST_Hour_19

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/sklearn/neural_
network/_multilayer_perceptron.py:585: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn
't converged yet.
  % self.max_iter, ConvergenceWarning)

Classifier trained for option WEST_Hour_20
Classifier trained for option WEST_Hour_21
Classifier trained for option WEST_Hour_22
Classifier trained for option WEST_Hour_23

Overall training accuracy = 99.94 percent.
```

Test the classifier's accuracy on test data.

In [6]:
```
# First, load the test data from NYISO for the year 2016. Again, utili
# named 'loadNYISOData'.

print("Starting the testing module...\n")
testPriceDA, testPriceDART, testLoadDA = loadNYISOData(2016)

# Create the data for the classifier using the function 'createClassif
testX, testY, rangeOfDays = createClassifierIO(testPriceDA, testPriceD

# The next step is not useful for implementing the trading strategy, b
# your trained classifiers are for the test data. Training accuracy is
# test accuracy.

testingAccuracy = [classifiers[ii].score(testX, testY[:, ii]) for ii i
print("Test Accuracy Stats: Min = %.2f%%, Avg = %.2f%%, Max = %.2f%%"
      (100 * np.min(testingAccuracy),
       100 * np.mean(testingAccuracy),
       100 * np.max(testingAccuracy)))

# Utilize the classifiers to predict the sign of DA - RT prices for ea
# the test data. Store the result in a pandas data frame with columns
# day in year as index.
predictedY = pd.DataFrame(np.column_stack([classifiers[ii].predict(tes
                          columns=optionNames, index=rangeOfDays)
```

```
Starting the testing module...

Test Accuracy Stats: Min = 48.76%, Avg = 59.02%, Max = 70.80%
```

Design and implement the trading strategy.

We define a fairly simple trading strategy. Define a total budget that you are willing to spend in the DA market. Recall that we only invest in options where we buy at the DA market and sell at the RT market. When your bid for one unit of an option clears in the DA market, you have to pay the DA price for that option. There are two possibilities:

1. Your bid clears: Therefore, your offer price was higher than the DA price.
2. Your bid does not clear: Then, the DA price was higher than your bid. In both these cases, the maximum you have to pay is your bid. Therefore, we will enforce that your bids across all options in a day does not exceed your total budget. Keep track of how rewards grow (or fall) through the year as you utilize your strategy. Also, keep track of how much rewards you get from each option. We shall visualize these results after implementing the trading strategy over the NYISO data for 2016.

Choose the bid prices as a suitable quantile of the historical DA prices. The higher the quantile, the better your chances are that your bid will be cleared. However, a higher quantile also indicates that you are budgeting more money for each option, and hence, you will buy fewer options.

In [7]:
```
dailyBudget = 250000
quantileOffer =0.8

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 3: Reward (in $) = -84,518
Day 4: Reward (in $) = 1,537,656
Day 5: Reward (in $) = 4,594
Day 6: Reward (in $) = -24,414
Day 7: Reward (in $) = -22,944
Day 8: Reward (in $) = 23,172
Day 9: Reward (in $) = -15,484
Day 10: Reward (in $) = -112,020
Day 11: Reward (in $) = -78,105
Day 12: Reward (in $) = -53,219
Day 13: Reward (in $) = -18,144
Day 14: Reward (in $) = -19,660
Day 15: Reward (in $) = -608
Day 16: Reward (in $) = 37,676
Day 17: Reward (in $) = -64,691
Day 18: Reward (in $) = -26,029
Day 19: Reward (in $) = 22,098
Day 20: Reward (in $) = -12,769
Day 21: Reward (in $) = -20,516
Day 22: Reward (in $) = 85,685
Day 23: Reward (in $) = 32,472
Day 24: Reward (in $) = -75,394
Day 25: Reward (in $) = -75,704
Day 26: Reward (in $) = -20,113
Day 27: Reward (in $) = -7,776
Day 28: Reward (in $) = -12,362
Day 29: Reward (in $) = -32,296
Day 30: Reward (in $) = -75,683
Day 31: Reward (in $) = -23,200
Day 32: Reward (in $) = -8,336
Day 33: Reward (in $) = -198,109
Day 34: Reward (in $) = -15,657
Day 35: Reward (in $) = 6,369
Day 36: Reward (in $) = -36,010
Day 37: Reward (in $) = -40,425
Day 38: Reward (in $) = 6,721
Day 39: Reward (in $) = 4,217
Day 40: Reward (in $) = -37,159
Day 41: Reward (in $) = -42,013
Day 42: Reward (in $) = -10,080
Day 43: Reward (in $) = -51,450
Day 44: Reward (in $) = -33,948
Day 45: Reward (in $) = -13,137
Day 46: Reward (in $) = 93,883
Day 47: Reward (in $) = -236
Day 48: Reward (in $) = -59,250
Day 49: Reward (in $) = -59,673
Day 50: Reward (in $) = -48,283
Day 51: Reward (in $) = 9,172
Day 52: Reward (in $) = -19,618
Day 53: Reward (in $) = -15,448
Day 54: Reward (in $) = 4,241
Day 55: Reward (in $) = -3,106
Day 56: Reward (in $) = -43,698
Day 57: Reward (in $) = -28,229
Day 58: Reward (in $) = -14,574
Day 59: Reward (in $) = -14,874
Day 60: Reward (in $) = -69,175
Day 61: Reward (in $) = -1,059
Day 62: Reward (in $) = -7,812
Day 63: Reward (in $) = -26,612
Day 64: Reward (in $) = 17,781
Day 65: Reward (in $) = 8,518
Day 66: Reward (in $) = 83,988
Day 67: Reward (in $) = -105,188
Day 68: Reward (in $) = -30,592
Day 69: Reward (in $) = -32,191
Day 70: Reward (in $) = 18,821
Day 71: Reward (in $) = -6,768
Day 72: Reward (in $) = -13,222
Day 73: Reward (in $) = -110,430
Day 74: Reward (in $) = 752,105
Day 75: Reward (in $) = -15,466
Day 76: Reward (in $) = -108,154
Day 77: Reward (in $) = -5,220
Day 78: Reward (in $) = -27,806
Day 79: Reward (in $) = -26,850
Day 80: Reward (in $) = -66,829
Day 81: Reward (in $) = -17,634
Day 82: Reward (in $) = 34,991
Day 83: Reward (in $) = -16,332
Day 84: Reward (in $) = -8,170
Day 85: Reward (in $) = 24,612
Day 86: Reward (in $) = -27,301
Day 87: Reward (in $) = 2,805,209
Day 88: Reward (in $) = 690,801
Day 89: Reward (in $) = -24,141
Day 90: Reward (in $) = -30,640
Day 91: Reward (in $) = -7,080
Day 92: Reward (in $) = -22,158
Day 93: Reward (in $) = 7,887
Day 94: Reward (in $) = 59,731
Day 95: Reward (in $) = -30,711
Day 96: Reward (in $) = 74,107
Day 97: Reward (in $) = 15,178
Day 98: Reward (in $) = 52
Day 99: Reward (in $) = -6,734
Day 100: Reward (in $) = -5,329
Day 101: Reward (in $) = -71,457
Day 102: Reward (in $) = -30,429
Day 103: Reward (in $) = 127,719
Day 104: Reward (in $) = -66,340
Day 105: Reward (in $) = -39,391
Day 106: Reward (in $) = 11,539
Day 107: Reward (in $) = -18,892
Day 108: Reward (in $) = 15,868
Day 109: Reward (in $) = -29,991
Day 110: Reward (in $) = -310
Day 111: Reward (in $) = -51,148
Day 112: Reward (in $) = 137,062
Day 113: Reward (in $) = -44,717
Day 114: Reward (in $) = -56,533
Day 115: Reward (in $) = -26,446
Day 116: Reward (in $) = 25,908
Day 117: Reward (in $) = -16,908
Day 118: Reward (in $) = 8,118
Day 119: Reward (in $) = -744
Day 120: Reward (in $) = 1,253
Day 121: Reward (in $) = 53,363
Day 122: Reward (in $) = -31,897
Day 123: Reward (in $) = -490
Day 124: Reward (in $) = 5,492
Day 125: Reward (in $) = -9,547
Day 126: Reward (in $) = -74,916
Day 127: Reward (in $) = -39,118
Day 128: Reward (in $) = -54,223
Day 129: Reward (in $) = -73,149
Day 130: Reward (in $) = -36,797
Day 131: Reward (in $) = 18,199
Day 132: Reward (in $) = 169,711
Day 133: Reward (in $) = 27,588
Day 134: Reward (in $) = -49,399
Day 135: Reward (in $) = -20,031
Day 136: Reward (in $) = -34,947
Day 137: Reward (in $) = -39,617
Day 138: Reward (in $) = -99,480
Day 139: Reward (in $) = 1,202
Day 140: Reward (in $) = -45,577
Day 141: Reward (in $) = 21,733
Day 142: Reward (in $) = 24,257
Day 143: Reward (in $) = 5,152
Day 144: Reward (in $) = -22,612
Day 145: Reward (in $) = -53,950
Day 146: Reward (in $) = -20,361
Day 147: Reward (in $) = -69,187
Day 148: Reward (in $) = 374,847
Day 149: Reward (in $) = 91,086
Day 150: Reward (in $) = 48,147
Day 151: Reward (in $) = 532,593
Day 152: Reward (in $) = 515,125
Day 153: Reward (in $) = 196,146
Day 154: Reward (in $) = 28,632
Day 155: Reward (in $) = -43,253
Day 156: Reward (in $) = -39,732
Day 157: Reward (in $) = -130,296
Day 158: Reward (in $) = -12,037
Day 159: Reward (in $) = -51,877
Day 160: Reward (in $) = -130,430
Day 161: Reward (in $) = -1,811
Day 162: Reward (in $) = 171,788
Day 163: Reward (in $) = -13,609
Day 164: Reward (in $) = -66,580
Day 165: Reward (in $) = 481,445
Day 166: Reward (in $) = -27,130
Day 167: Reward (in $) = 22,717
Day 168: Reward (in $) = -1,380
Day 169: Reward (in $) = 6,706
Day 170: Reward (in $) = -41,473
Day 171: Reward (in $) = -3,630
Day 172: Reward (in $) = -6,947
Day 173: Reward (in $) = 179,532
Day 174: Reward (in $) = -9,650
Day 175: Reward (in $) = 37,417
Day 176: Reward (in $) = -26,715
Day 177: Reward (in $) = -39,599
Day 178: Reward (in $) = -13,757
Day 179: Reward (in $) = -42,759
Day 180: Reward (in $) = -19,663
Day 181: Reward (in $) = -22,004
Day 182: Reward (in $) = -52,058
Day 183: Reward (in $) = -83,325
Day 184: Reward (in $) = -79,410
Day 185: Reward (in $) = -32,373
Day 186: Reward (in $) = 98,414
Day 187: Reward (in $) = 6,848
Day 188: Reward (in $) = 3,048
Day 189: Reward (in $) = -1,054
Day 190: Reward (in $) = -102,290
Day 191: Reward (in $) = -17,770
Day 192: Reward (in $) = -30,737
Day 193: Reward (in $) = -26,940
Day 194: Reward (in $) = -2,089
Day 195: Reward (in $) = 2,576
Day 196: Reward (in $) = 6,900
Day 197: Reward (in $) = 7,583
Day 198: Reward (in $) = -11,861
Day 199: Reward (in $) = -1,556
Day 200: Reward (in $) = -99,558
Day 201: Reward (in $) = -15,709
Day 202: Reward (in $) = 1,665
Day 203: Reward (in $) = 1,003
Day 204: Reward (in $) = -11,819
Day 205: Reward (in $) = -31,416
Day 206: Reward (in $) = -2,556
Day 207: Reward (in $) = -4,528
Day 208: Reward (in $) = 1,755
Day 209: Reward (in $) = 5,982
Day 210: Reward (in $) = 111,332
Day 211: Reward (in $) = 44,714
Day 212: Reward (in $) = 100,221
Day 213: Reward (in $) = -380
Day 214: Reward (in $) = -87,746
Day 215: Reward (in $) = -47,019
Day 216: Reward (in $) = 6,135
Day 217: Reward (in $) = 3,541
Day 218: Reward (in $) = 26,910
Day 219: Reward (in $) = -24,080
Day 220: Reward (in $) = -2,686
Day 221: Reward (in $) = -30,047
Day 222: Reward (in $) = 18,266
Day 223: Reward (in $) = 9,699
Day 224: Reward (in $) = -49
Day 225: Reward (in $) = 3,453
Day 226: Reward (in $) = -21,829
Day 227: Reward (in $) = -1,552
Day 228: Reward (in $) = 171,689
Day 229: Reward (in $) = -1,651
Day 230: Reward (in $) = -2,979
Day 231: Reward (in $) = 1,217
Day 232: Reward (in $) = -45,532
Day 233: Reward (in $) = -91,494
Day 234: Reward (in $) = -94,926
Day 235: Reward (in $) = -129,937
Day 236: Reward (in $) = 26,061
Day 237: Reward (in $) = 51,989
Day 238: Reward (in $) = -1,207
Day 239: Reward (in $) = -5,287
Day 240: Reward (in $) = -6,227
Day 241: Reward (in $) = -661
Day 242: Reward (in $) = 53,153
Day 243: Reward (in $) = -3,063
Day 244: Reward (in $) = -3,511
Day 245: Reward (in $) = -81,039
Day 246: Reward (in $) = -51,434
Day 247: Reward (in $) = -15,071
Day 248: Reward (in $) = -32,558
Day 249: Reward (in $) = -4,222
Day 250: Reward (in $) = -20,518
Day 251: Reward (in $) = 1,040
Day 252: Reward (in $) = 4,368
Day 253: Reward (in $) = 207,496
Day 254: Reward (in $) = -12,343
Day 255: Reward (in $) = -6,654
Day 256: Reward (in $) = -4,298
Day 257: Reward (in $) = -3,966
Day 258: Reward (in $) = -23,444

Day 259: Reward (in $) = -43,437
Day 260: Reward (in $) = -19,560
Day 261: Reward (in $) = 25,967
Day 262: Reward (in $) = 151
Day 263: Reward (in $) = -60,669
Day 264: Reward (in $) = -40,241
Day 265: Reward (in $) = -46,293
Day 266: Reward (in $) = -68,855
Day 267: Reward (in $) = -8,547
Day 268: Reward (in $) = -24,275
Day 269: Reward (in $) = -83,722
Day 270: Reward (in $) = -77,886
Day 271: Reward (in $) = -7,054
Day 272: Reward (in $) = -34,080
Day 273: Reward (in $) = -32,408
Day 274: Reward (in $) = 18,692
Day 275: Reward (in $) = 23,389
Day 276: Reward (in $) = -7,202
Day 277: Reward (in $) = 59,755
Day 278: Reward (in $) = 165,880
Day 279: Reward (in $) = -37,154
Day 280: Reward (in $) = 11,175
Day 281: Reward (in $) = -11,687
Day 282: Reward (in $) = -201
Day 283: Reward (in $) = -2,470
Day 284: Reward (in $) = 20,967
Day 285: Reward (in $) = -59,346
Day 286: Reward (in $) = -62,160
Day 287: Reward (in $) = -43,753
Day 288: Reward (in $) = 12,635
Day 289: Reward (in $) = 22,185
Day 290: Reward (in $) = 6,360
Day 291: Reward (in $) = -10,047
Day 292: Reward (in $) = -2,683
Day 293: Reward (in $) = -33,990
Day 294: Reward (in $) = -24,786
Day 295: Reward (in $) = -12,298
Day 296: Reward (in $) = -63,668
Day 297: Reward (in $) = -39,688
Day 298: Reward (in $) = -15,372
Day 299: Reward (in $) = -55,716
Day 300: Reward (in $) = -28,555
Day 301: Reward (in $) = -8,520
Day 302: Reward (in $) = -6,155
Day 303: Reward (in $) = -18
Day 304: Reward (in $) = -53,475
Day 305: Reward (in $) = -15,812
Day 306: Reward (in $) = 149,050
Day 307: Reward (in $) = 177,636
Day 308: Reward (in $) = -49,006
Day 309: Reward (in $) = 46,669
Day 310: Reward (in $) = -5,354
Day 311: Reward (in $) = 8,834
Day 312: Reward (in $) = 42,238
Day 313: Reward (in $) = -16,226
Day 314: Reward (in $) = -32,142
Day 315: Reward (in $) = -58,402
Day 316: Reward (in $) = -25,312
Day 317: Reward (in $) = -30,702
Day 318: Reward (in $) = -11,681
Day 319: Reward (in $) = 2,893
Day 320: Reward (in $) = 12,663
Day 321: Reward (in $) = 11,918
Day 322: Reward (in $) = 1,549
Day 323: Reward (in $) = 7,493
Day 324: Reward (in $) = 15,640
Day 325: Reward (in $) = -26,043
Day 326: Reward (in $) = -11,116
Day 327: Reward (in $) = 27,482
Day 328: Reward (in $) = 138,208
Day 329: Reward (in $) = 34,671
Day 330: Reward (in $) = 5,608
Day 331: Reward (in $) = -8,525
Day 332: Reward (in $) = -61,952
Day 333: Reward (in $) = -69,155
Day 334: Reward (in $) = -8,803
Day 335: Reward (in $) = -21,234
Day 336: Reward (in $) = -16,808
Day 337: Reward (in $) = 30,933
Day 338: Reward (in $) = -10,424
Day 339: Reward (in $) = -30,681
Day 340: Reward (in $) = 11
Day 341: Reward (in $) = -36,420
Day 342: Reward (in $) = -21,283
Day 343: Reward (in $) = -18,304
Day 344: Reward (in $) = 278,188
Day 345: Reward (in $) = 95,421
Day 346: Reward (in $) = 1,987
Day 347: Reward (in $) = 27,154
Day 348: Reward (in $) = -57,760
Day 349: Reward (in $) = 41,116
Day 350: Reward (in $) = -3,636
Day 351: Reward (in $) = 269,016
Day 352: Reward (in $) = -13,726
Day 353: Reward (in $) = -11,705
Day 354: Reward (in $) = -41,076
Day 355: Reward (in $) = 2,946
Day 356: Reward (in $) = -37,991
Day 357: Reward (in $) = -53,083
Day 358: Reward (in $) = -29,087
Day 359: Reward (in $) = -14,417
Day 360: Reward (in $) = -55,329
Day 361: Reward (in $) = -82,214
Day 362: Reward (in $) = 17,084
Day 363: Reward (in $) = -51,075
Day 364: Reward (in $) = -33,671
Day 365: Reward (in $) = -26,542
Total money earned over the year (in $) = 4,414,066
```

Task1: Visualize the rewards (25 points)

We would like to plot the cumulative reward over the year 2016. By cumulative reward on a particular date, we mean the total reward from the start of the year till that date.

Also, plot a heat map of the returns from each option.

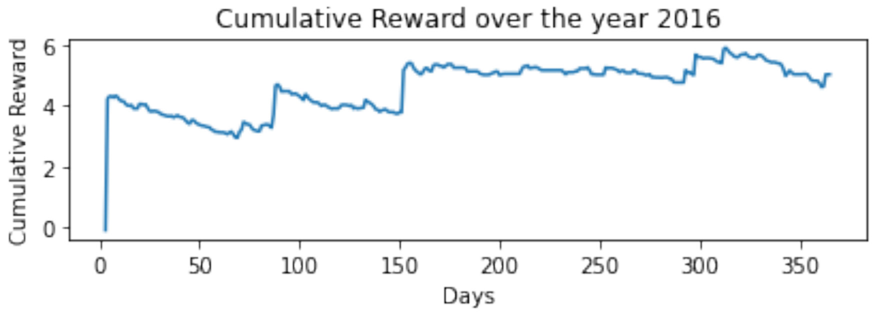
Fill in the missing lines below.

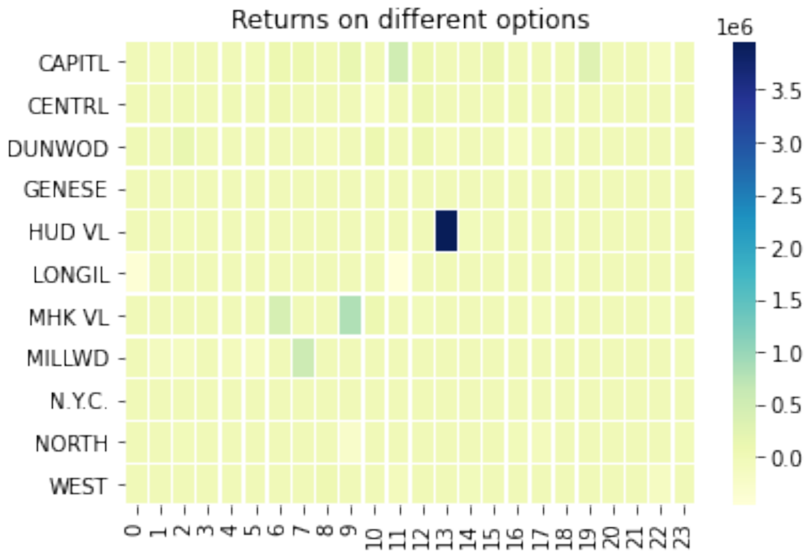
In [52]:
```
# Plot the cumulative reward over the year 2016. Also, plot a heat map
# each option.

fig, axs = plt.subplots(2, 1, tight_layout=True)
axs = axs.ravel()


## Enter code here where you plot in axs[0].
axs[0] = plt.plot(cumulativeReward.keys(), cumulativeReward.values())
plt.title('Cumulative Reward over the year 2016')
plt.xlabel('Days')
plt.ylabel('Cumulative Reward')
plt.show()


axs[1] = sns.heatmap(np.reshape(list(optionReturn.values()), (len(list
                     linewidth=0.5,
                     cmap="YlGnBu")
axs[1].set_yticklabels(listOfZones, rotation=0)
axs[1].set_xticklabels(range(24), rotation=90)
axs[1].set_title('Returns on different options')
plt.show()
```





Task 2: Choosing a classifier (25 points)

We used a multilayer perceptron classifier. Your task is to try out SVM and logistic regression classifiers, and explore which one leads to more profits. Use the relevant functions from 'sklearn'.

(comments here, add a new code cell below)

NN Profits: $4,414,066

SVM Profits: $1,567,256

Logistic Regression Profit: $3,103,216

I have used the same codes above and manipulate the classifier to figure out which one gives more profit and found out that Logistic Regression profits yields more profit.

In [21]:
```
classifiers = []
```

```
# We have two options here. Use previous training experience, or learn
useSavedClassifiers = False

if not useSavedClassifiers:

    print("Starting training module...\n")
    trainPriceDA, trainPriceDART, trainLoadDA = loadNYISOData(2015)

    numberOfDays =  int(len(trainPriceDA.index))
    print("Loaded hourly prices from 2015 for %d days." % numberOfDays

    # We will implement a trading strategy, where we bid a particular
    # DA prices for an option. If you do not know what a quantile mean
    # article on it. Essentially, a 95% quantile of the DA prices equa
    # 95% of the DA prices are below it. Store all quantiles starting
    # 5% in a dictionary. Store them in a pickle file.

    quantilesToStore = [0.70, 0.75, 0.80, 0.85, 0.90, 0.95]
    offerPrices = trainPriceDA.quantile(q=quantilesToStore).transpose(
    pickle.dump(offerPrices, open("./Training/OfferPrices", 'wb'))

    # Calculate the average price spread for each option over the enti
    # us in choosing our portfolio. Store it as a dictionary. Our bid
    # options that our classifier indicates that they will be profitab
    # have higher average price differences, indicating that they have
    # Store them using pickle.

    averagePriceSpread = trainPriceDART.mean(axis=0).transpose().to_di
    pickle.dump(averagePriceSpread, open("./Training/AveragePriceSprea

    # Create the training dataset using the function 'createClassifier
    # loads, and store them in 'trainX', and 'trainY'.

    trainX, trainY, _ = createClassifierIO(trainPriceDA, trainPriceDAR

    # Define a collection of classifiers, one for each option. You can
    # as that based on an SVM, logistic regression, multilayer percept
    # measure training accuracy to indicate how well the classifier wo
    # However, good training accuracy does not always indicate good te
    # Avoid over-fitting.


    classifiers = [LogisticRegression(C=0.9, max_iter=400, solver='lib
                   for _ in range(nOptions)]

    trainingAccuracy = 0

    for ii in range(nOptions):
        classifiers[ii].fit(trainX, trainY[:, ii])
        print("Classifier trained for option " + optionNames[ii])
        trainingAccuracy += classifiers[ii].score(trainX, trainY[:, ii

        # Store the classifier.
        pickle.dump(classifiers[ii], open("./Training/Classifier_" + c

    print("\nOverall training accuracy = %1.2f percent." % (100 * trai

    del numberOfDays, trainPriceDA, trainLoadDA, trainPriceDART, trair
else:

    # Load the classifiers, the offer prices at various quantiles, and

    print("Loading previously trained variables...\n")
    classifiers = [pickle.load(open("./Training/Classifier_" + optionN
                   for ii in range(nOptions)]
    offerPrices = pickle.load(open("./Training/OfferPrices", 'rb'))
    averagePriceSpread = pickle.load(open("./Training/AveragePriceSpre

    print("All training variables were loaded successfully...\n")
```

```
Classifier trained for option WEST_Hour_6
Classifier trained for option WEST_Hour_7
Classifier trained for option WEST_Hour_8
Classifier trained for option WEST_Hour_9
Classifier trained for option WEST_Hour_10
Classifier trained for option WEST_Hour_11
Classifier trained for option WEST_Hour_12
Classifier trained for option WEST_Hour_13
Classifier trained for option WEST_Hour_14
Classifier trained for option WEST_Hour_15
Classifier trained for option WEST_Hour_16
Classifier trained for option WEST_Hour_17
Classifier trained for option WEST_Hour_18
Classifier trained for option WEST_Hour_19
Classifier trained for option WEST_Hour_20
Classifier trained for option WEST_Hour_21
Classifier trained for option WEST_Hour_22
Classifier trained for option WEST_Hour_23

Overall training accuracy = 98.71 percent.
```

In [22]:
```
# First, load the test data from NYISO for the year 2016. Again, utili
# named 'loadNYISOData'.

print("Starting the testing module...\n")
testPriceDA, testPriceDART, testLoadDA = loadNYISOData(2016)

# Create the data for the classifier using the function 'createClassif
testX, testY, rangeOfDays = createClassifierIO(testPriceDA, testPriceD

# The next step is not useful for implementing the trading strategy, b
# your trained classifiers are for the test data. Training accuracy is
# test accuracy.

testingAccuracy = [classifiers[ii].score(testX, testY[:, ii]) for ii i
print("Test Accuracy Stats: Min = %.2f%%, Avg = %.2f%%, Max = %.2f%%"
      (100 * np.min(testingAccuracy),
       100 * np.mean(testingAccuracy),
       100 * np.max(testingAccuracy)))

# Utilize the classifiers to predict the sign of DA - RT prices for ea
# the test data. Store the result in a pandas data frame with columns
# day in year as index.
predictedY = pd.DataFrame(np.column_stack([classifiers[ii].predict(tes
                          columns=optionNames, index=rangeOfDays)
```

```
Starting the testing module...

Test Accuracy Stats: Min = 46.83%, Avg = 57.17%, Max = 67.22%
```

In [23]:
```
dailyBudget = 250000
quantileOffer =0.8

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 317: Reward (in $) = -34,070
Day 317: Reward (in $) = -34,070
Day 318: Reward (in $) = -17,426
Day 319: Reward (in $) = 2,487
Day 320: Reward (in $) = 15,996
Day 321: Reward (in $) = 10,207
Day 322: Reward (in $) = -5,399
Day 323: Reward (in $) = 7,547
Day 324: Reward (in $) = 15,614
Day 325: Reward (in $) = -26,348
Day 326: Reward (in $) = -10,118
Day 327: Reward (in $) = -12,111
Day 328: Reward (in $) = 58,502
Day 329: Reward (in $) = 37,809
Day 330: Reward (in $) = 5,893
Day 331: Reward (in $) = -11,863
Day 332: Reward (in $) = -45,423
Day 333: Reward (in $) = -41,601
Day 334: Reward (in $) = -2,201
Day 335: Reward (in $) = -20,624
Day 336: Reward (in $) = -17,332
```

Task 3: Quantile (25 points)

For the best classifier, try different quanltile choices from the following list: 0.70, 0.75, 0.80, 0.85, 0.90, 0.95. Rank them in terms of profits (low to high).

How do you expect the quantile to affect the portfolio?

(comments here, add a new code cell below)

0.70: $3,608,945

0.75: $3,453,180

0.80: $3,103,216

0.85: $2,435,304

0.90: $1,658,829

0.95: $1,825,928

0.9 < 0.95 < 0.85 < 0.80 < 0.75 < 0.70 in terms of profits from low to high and as we can see above, 0.7 gives the best maximum profit. From this observation, lower quantile value yields better profit using linear regresssion classifier.

In [31]:
```
classifiers = []

# We have two options here. Use previous training experience, or lear
```

```
useSavedClassifiers = False

if not useSavedClassifiers:

    print("Starting training module...\n")
    trainPriceDA, trainPriceDART, trainLoadDA = loadNYISOData(2015)

    numberOfDays =  int(len(trainPriceDA.index))
    print("Loaded hourly prices from 2015 for %d days." % numberOfDays

    # We will implement a trading strategy, where we bid a particular
    # DA prices for an option. If you do not know what a quantile mean
    # article on it. Essentially, a 95% quantile of the DA prices equa
    # 95% of the DA prices are below it. Store all quantiles starting
    # 5% in a dictionary. Store them in a pickle file.

    quantilesToStore = [0.70, 0.75, 0.80, 0.85, 0.90, 0.95]
    offerPrices = trainPriceDA.quantile(q=quantilesToStore).transpose(
    pickle.dump(offerPrices, open("./Training/OfferPrices", 'wb'))

    # Calculate the average price spread for each option over the enti
    # us in choosing our portfolio. Store it as a dictionary. Our bid
    # options that our classifier indicates that they will be profitab
    # have higher average price differences, indicating that they have
    # Store them using pickle.

    averagePriceSpread = trainPriceDART.mean(axis=0).transpose().to_di
    pickle.dump(averagePriceSpread, open("./Training/AveragePriceSprea

    # Create the training dataset using the function 'createClassifier
    # loads, and store them in 'trainX', and 'trainY'.

    trainX, trainY, _ = createClassifierIO(trainPriceDA, trainPriceDAR

    # Define a collection of classifiers, one for each option. You can
    # as that based on an SVM, logistic regression, multilayer percept
    # measure training accuracy to indicate how well the classifier wo
    # However, good training accuracy does not always indicate good te
    # Avoid over-fitting.


    classifiers = [LogisticRegression(C=0.9, max_iter=400, solver='lib
                   for _ in range(nOptions)]

    trainingAccuracy = 0

    for ii in range(nOptions):
        classifiers[ii].fit(trainX, trainY[:, ii])
        print("Classifier trained for option " + optionNames[ii])
        trainingAccuracy += classifiers[ii].score(trainX, trainY[:, ii
        # Store the classifier.
        pickle.dump(classifiers[ii], open("./Training/Classifier_" + o

    print("\nOverall training accuracy = %1.2f percent." % (100 * trai

    del numberOfDays, trainPriceDA, trainLoadDA, trainPriceDART, trair
else:

    # Load the classifiers, the offer prices at various quantiles, and

    print("Loading previously trained variables...\n")
    classifiers = [pickle.load(open("./Training/Classifier_" + optionN
                   for ii in range(nOptions)]
    offerPrices = pickle.load(open("./Training/OfferPrices", 'rb'))
    averagePriceSpread = pickle.load(open("./Training/AveragePriceSpre

    print("All training variables were loaded successfully...\n")
```

```
Classifier trained for option WEST_Hour_7
Classifier trained for option WEST_Hour_8
Classifier trained for option WEST_Hour_9
Classifier trained for option WEST_Hour_10
Classifier trained for option WEST_Hour_11
Classifier trained for option WEST_Hour_12
Classifier trained for option WEST_Hour_13
Classifier trained for option WEST_Hour_14
Classifier trained for option WEST_Hour_15
Classifier trained for option WEST_Hour_16
Classifier trained for option WEST_Hour_17
Classifier trained for option WEST_Hour_18
Classifier trained for option WEST_Hour_19
Classifier trained for option WEST_Hour_20
Classifier trained for option WEST_Hour_21
Classifier trained for option WEST_Hour_22
Classifier trained for option WEST_Hour_23

Overall training accuracy = 98.71 percent.
```

In [32]:
```
# First, load the test data from NYISO for the year 2016. Again, utili
# named 'loadNYISOData'.

print("Starting the testing module...\n")
testPriceDA, testPriceDART, testLoadDA = loadNYISOData(2016)

# Create the data for the classifier using the function 'createClassif
testX, testY, rangeOfDays = createClassifierIO(testPriceDA, testPriceD

# The next step is not useful for implementing the trading strategy, b
# your trained classifiers are for the test data. Training accuracy is
# test accuracy.

testingAccuracy = [classifiers[ii].score(testX, testY[:, ii]) for ii i
print("Test Accuracy Stats: Min = %.2f%%, Avg = %.2f%%, Max = %.2f%%"
      (100 * np.min(testingAccuracy),
       100 * np.mean(testingAccuracy),
       100 * np.max(testingAccuracy)))

# Utilize the classifiers to predict the sign of DA - RT prices for ea
# the test data. Store the result in a pandas data frame with columns
# day in year as index.
predictedY = pd.DataFrame(np.column_stack([classifiers[ii].predict(tes
                          columns=optionNames, index=rangeOfDays)
```

```
Starting the testing module...

Test Accuracy Stats: Min = 46.83%, Avg = 57.17%, Max = 67.22%
```

In [30]:
```
dailyBudget = 250000
quantileOffer =0.95

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 347: Reward (in $) = 2,365
Day 348: Reward (in $) = -24,864
Day 349: Reward (in $) = 48,796
Day 350: Reward (in $) = 4,928
Day 351: Reward (in $) = 47,792
Day 352: Reward (in $) = 33,206
Day 353: Reward (in $) = -11,651
Day 354: Reward (in $) = -28,810
Day 355: Reward (in $) = -30,939
Day 356: Reward (in $) = -15,118
Day 357: Reward (in $) = -22,578
Day 358: Reward (in $) = -3,039
Day 359: Reward (in $) = -6,743
Day 360: Reward (in $) = -20,203
Day 361: Reward (in $) = -41,238
Day 362: Reward (in $) = 11,986
Day 363: Reward (in $) = 162,475
Day 364: Reward (in $) = -430
Day 365: Reward (in $) = -9,646
Total money earned over the year (in $) = 1,825,928
```

Task 4: Daily budget (25 points)

We used a daily budget of 250,000. Try different budgets, let's say, 100,000, 150,000, 200,000, and 300,000. Rank them in terms of profits (low to high). Fix the quantile to the one that led to maximum profits in the previous task.

(comments here, add a new code cell below)

100,000: $1,431,493

150,000: $2,127,473

200,000: $2,809,937

300,000: $4,276,427

100,000 < 150,000 < 200,000 < 300,000 in terms of profits from low to high. As we have observed, dailyBudget increases, corresponding profit also follows its trend and becomes larger.

In [33]:
```
dailyBudget = 100000
quantileOffer =0.70

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
```

```
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
```

```
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 348: Reward (in $) = -13,906
Day 349: Reward (in $) = 34,720
Day 350: Reward (in $) = -2,015
Day 351: Reward (in $) = 28,023
Day 352: Reward (in $) = 17,570
Day 353: Reward (in $) = 0
Day 354: Reward (in $) = -20,161
Day 355: Reward (in $) = -27,929
Day 356: Reward (in $) = -8,719
Day 357: Reward (in $) = -13,543
Day 358: Reward (in $) = 653
Day 359: Reward (in $) = -5,866
Day 360: Reward (in $) = -14,497
Day 361: Reward (in $) = -33,307
Day 362: Reward (in $) = 10,189
Day 363: Reward (in $) = 123,806
Day 364: Reward (in $) = 1,642
Day 365: Reward (in $) = -6,684
Total money earned over the year (in $) = 1,431,493
```

In [34]:

```
dailyBudget = 150000
quantileOffer =0.70

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0
    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format
    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 347: Reward (in $) = 5,258
Day 348: Reward (in $) = -20,484
Day 349: Reward (in $) = 50,543
Day 350: Reward (in $) = -2,856
Day 351: Reward (in $) = 41,821
Day 352: Reward (in $) = 25,593
Day 353: Reward (in $) = 0
Day 354: Reward (in $) = -30,030
Day 355: Reward (in $) = -41,925
Day 356: Reward (in $) = -13,097
Day 357: Reward (in $) = -20,166
Day 358: Reward (in $) = -1
Day 359: Reward (in $) = -8,842
Day 360: Reward (in $) = -21,983
Day 361: Reward (in $) = -49,386
Day 362: Reward (in $) = 15,129
Day 363: Reward (in $) = 182,045
Day 364: Reward (in $) = 3,273
Day 365: Reward (in $) = -10,747
Total money earned over the year (in $) = 2,127,473
```

In [35]:

```
dailyBudget = 200000
quantileOffer =0.70

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for o

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 348: Reward (in $) = -27,597
Day 349: Reward (in $) = 68,113
Day 350: Reward (in $) = -3,950
Day 351: Reward (in $) = 55,735
Day 352: Reward (in $) = 34,782
Day 353: Reward (in $) = 0
Day 354: Reward (in $) = -39,930
Day 355: Reward (in $) = -56,037
Day 356: Reward (in $) = -17,442
Day 357: Reward (in $) = -26,666
Day 358: Reward (in $) = 1,120
Day 359: Reward (in $) = -11,877
Day 360: Reward (in $) = -29,704
Day 361: Reward (in $) = -66,990
Day 362: Reward (in $) = 19,537
Day 363: Reward (in $) = 248,136
Day 364: Reward (in $) = 3,973
Day 365: Reward (in $) = -13,480
Total money earned over the year (in $) = 2,809,937
```

In [36]:
```
dailyBudget = 300000
quantileOffer =0.70

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than c
    # decreasing your budget with each bid.

    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    while budget > np.median([offerPrices[quantileOffer][option] for c

        optionToBuy = choice(chosenOptions, p=probabilityOptions)

        if budget >= offerPrices[quantileOffer][optionToBuy]:
            portfolio[optionToBuy] += 1
            budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 335: Reward (in $) = -27,575
Day 336: Reward (in $) = -23,676
Day 337: Reward (in $) = 24,068
Day 338: Reward (in $) = -12,156
Day 339: Reward (in $) = -20,216
Day 340: Reward (in $) = -8,538
Day 341: Reward (in $) = -70,439
Day 342: Reward (in $) = -150,571
Day 343: Reward (in $) = -165,770
Day 344: Reward (in $) = 229,472
Day 345: Reward (in $) = 105,895
Day 346: Reward (in $) = -40,600
Day 347: Reward (in $) = 9,921
Day 348: Reward (in $) = -40,523
Day 349: Reward (in $) = 103,355
Day 350: Reward (in $) = -5,766
Day 351: Reward (in $) = 83,717
Day 352: Reward (in $) = 51,240
Day 353: Reward (in $) = 0
Day 354: Reward (in $) = 59,647
```

Task 6: Beat the algorithm! (15 points, bonus)

Make changes to increase the profits further! The people with the top 15 total profits will get additional points. Exaplain what did for improvement and how much profits you made. Do not exceed the 250,000 budget!

(comments here, add a new code cell below)

I really do not think of any other modification methods except changing modification of optionToBuy. I simply remove the while loop condition to eliminate the diversity and count everyday. Because of removing while loop, I simply changed next line with if to while loop. This results me $5,030,990 profit which is much higher than previous exercises

In [41]:
```
dailyBudget = 250000
quantileOffer =0.70

# Keep track of your rewards in each day, a cumulative reward over the
# year. Also, keep track of the total reward from each option. Store t
# indexed by day of the year.
reward = {}
cumulativeReward = {}
totalReward = 0
optionReturn = dict((option, 0) for option in optionNames)

# Implement the trading strategy on each day!

for day in rangeOfDays:

    reward[day] = 0

    # Find the options that your classifier says that should be profit
    # names in chosenOptions.
    chosenOptionNumbers = np.ravel(list(np.nonzero(predictedY.loc[day]
    if np.size(chosenOptionNumbers) == 0:
        continue
    chosenOptions = [optionNames[i] for i in chosenOptionNumbers]

    # Design the portfolio based on average price spreads. Our strateg
    # exceeded your daily budget, pick an option from the list of 'cho
    # the probability of choosing it is proportional to exponential(hi
    # is, a historically profitable option is chosen more often than o
    # decreasing your budget with each bid.
    # Start with an empty portfolio.

    portfolio = dict((option, 0) for option in chosenOptions)

    # Calculate the probabilities of choosing each option among the li
    # 'chosenOptions' contains the options that your classifier indica
    priceSpreads = [1.0 * averagePriceSpread[option] for option in cho
    probabilityOptions = [np.exp(p) for p in priceSpreads]
    probabilityOptions /= np.sum(probabilityOptions)

    # Start with your daily budget.
    budget = dailyBudget

    # Sampling among the profitable options and bid based on them.
    optionToBuy = choice(chosenOptions, p=probabilityOptions)

    while budget > offerPrices[quantileOffer][optionToBuy]:
        portfolio[optionToBuy] += 1
        budget -= offerPrices[quantileOffer][optionToBuy]

    # Compute the reward from the day. Go through each of the options
    # If the DA price is lower than the bid price, then your bid is cl
    # have bought, you get a reward equal to the DA - RT price.

    for option in chosenOptions:
        if testPriceDA.at[day, option] < offerPrices[quantileOffer][op
            rewardOptionDay = testPriceDART.at[day, option] * portfoli
            optionReturn[option] += rewardOptionDay
            reward[day] += rewardOptionDay

    totalReward += reward[day]

    # Calculate the cumulative reward in millions of dollars.
    cumulativeReward[day] = totalReward/1000000

    print("Day " + str(day) + ": Reward (in $) = " + "{0:,.0f}".format

print("Total money earned over the year (in $) = " + "{0:,.0f}".format
```

```
Day 348: Reward (in $) = 0
Day 349: Reward (in $) = 0
Day 350: Reward (in $) = 0
Day 351: Reward (in $) = 0
Day 352: Reward (in $) = 10,580
Day 353: Reward (in $) = 0
Day 354: Reward (in $) = -22,786
Day 355: Reward (in $) = -106,897
Day 356: Reward (in $) = -82,624
Day 357: Reward (in $) = -13,292
Day 358: Reward (in $) = 0
Day 359: Reward (in $) = -502
Day 360: Reward (in $) = -82,731
Day 361: Reward (in $) = -117,380
Day 362: Reward (in $) = 23,638
Day 363: Reward (in $) = 383,821
Day 364: Reward (in $) = 726
Day 365: Reward (in $) = 4,439
Total money earned over the year (in $) = 5,030,990
```

Task 7: Implement a different trading strategy (10 points, bonus)

Implement a different trading strategy, explain it and report its outcomes. If the rewards are comparable to the one discussed in class, and you can clearly explain the logic behind you strategy, you get additional 25 points. If your strategy leads to having profits among the top 20, then you automatically completed Task 6 too!

(comments here, add a new code cell below)

```
In [ ]:
```