

In [1]: `##Price Prediction`

```

In [1]: import warnings
import time
import sys
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn import hmm

warnings.filterwarnings("ignore", category=DeprecationWarning)

PLOT_SHOW=True
PLOT_TYPE = False

NUM_TEST = 100
K = 50
NUM_ITERS=10000

STOCKS=['apple.csv', 'cmcst.csv', 'google.csv']
#NUM_STATES=12
#FILE_NAME='HistoricalQuotes.csv'
#TRAIN_CHUNK_SIZE=100

#dirichlet_params = np.array([1., 20., 20., 20.])
#dirichlet_params = np.random.randint(1,50,NUM_STATES)
labels = ['Close', 'Open', 'High', 'Low']
likelihood_vect = np.empty([0,1])
aic_vect = np.empty([0,1])
bic_vect = np.empty([0,1])

# Possible number of states in Markov Model
STATE_SPACE = range(2,15)

# Calculating Mean Absolute Percentage Error of predictions
def calc_mape(predicted_data, true_data):
    return np.divide(np.sum(np.divide(np.absolute(predicted_data - true_data), tr

for stock in STOCKS:
    dataset = np.genfromtxt(stock, delimiter=',')
    predicted_stock_data = np.empty([0,dataset.shape[1]])
    likelihood_vect = np.empty([0,1])
    aic_vect = np.empty([0,1])
    bic_vect = np.empty([0,1])
    for states in STATE_SPACE:
        num_params = states**2 + states
        dirichlet_params_states = np.random.randint(1,50,states)
        #model = hmm.GaussianHMM(n_components=states, covariance_type='full', sto
        model = hmm.GaussianHMM(n_components=states, covariance_type='full', tol=
        model.fit(dataset[NUM_TEST:, :])
        if model.monitor_iter == NUM_ITERS:
            print('Increase number of iterations')
            sys.exit(1)
        likelihood_vect = np.vstack((likelihood_vect, model.score(dataset)))
        aic_vect = np.vstack((aic_vect, -2 * model.score(dataset) + 2 * num_param
        bic_vect = np.vstack((bic_vect, -2 * model.score(dataset) + num_params *

    opt_states = np.argmin(bic_vect) + 2

```

```

print('Optimum number of states are {}'.format(opt_states))

for idx in reversed(range(NUM_TEST)):
    train_dataset = dataset[idx + 1,: ]
    test_data = dataset[idx,: ]
    num_examples = train_dataset.shape[0]
    #model = hmm.GaussianHMM(n_components=opt_states, covariance_type='full',
    if idx == NUM_TEST - 1:
        model = hmm.GaussianHMM(n_components=opt_states, covariance_type='full')
    else:
        # Retune the model by using the HMM paramters from the previous iteration
        model = hmm.GaussianHMM(n_components=opt_states, covariance_type='full')
        model.transmat_ = transmat_retune_prior
        model.startprob_ = startprob_retune_prior
        model.means_ = means_retune_prior
        model.covars_ = covars_retune_prior

    model.fit(np.flipud(train_dataset))

    transmat_retune_prior = model.transmat_
    startprob_retune_prior = model.startprob_
    means_retune_prior = model.means_
    covars_retune_prior = model.covars_

    if model.monitor_.iter == NUM_ITERS:
        print('Increase number of iterations')
        sys.exit(1)
    #print('Model score : ', model.score(dataset))
    #print('Dirichlet parameters : ',dirichlet_params)

    iters = 1;
    past_likelihood = []
    curr_likelihood = model.score(np.flipud(train_dataset[0:K - 1, :]))
    while iters < num_examples / K - 1:
        past_likelihood = np.append(past_likelihood, model.score(np.flipud(train_dataset[iters*K: (iters+1)*K, :]))
        iters = iters + 1
        likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
        predicted_change = train_dataset[likelihood_diff_idx,: ] - train_dataset[likelihood_diff_idx-1,: ]
        predicted_stock_data = np.vstack((predicted_stock_data, dataset[idx + 1, : ] + predicted_change))
    np.savetxt('{}forecast.csv'.format(stock),predicted_stock_data,delimiter=',')

    mape = calc_mape(predicted_stock_data, np.flipud(dataset[range(100),:]))
    print('MAPE for the stock {} is {}'.format(stock,mape))

    if PLOT_TYPE:
        hdl_p = plt.plot(range(100), predicted_stock_data);
        plt.title('Predicted stock prices')
        plt.legend(iter(hdl_p), ('Close', 'Open', 'High', 'Low'))
        plt.xlabel('Time steps')
        plt.ylabel('Price')
        plt.figure()
        hdl_a = plt.plot(range(100),np.flipud(dataset[range(100),:]))
        plt.title('Actual stock prices')
        plt.legend(iter(hdl_p), ('Close', 'Open', 'High', 'Low'))
        plt.xlabel('Time steps')
        plt.ylabel('Price')
    else:

```

```

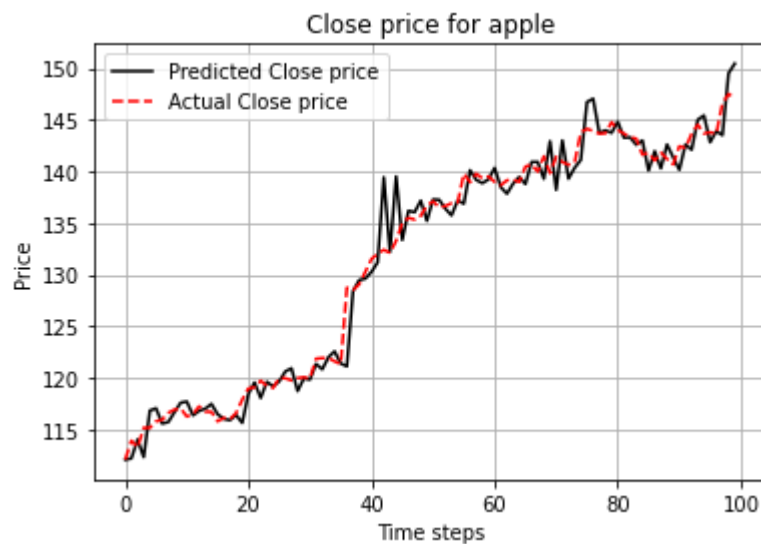
for i in range(4):
    plt.figure()
    plt.plot(range(100), predicted_stock_data[:,i], 'k-', label = 'Predicted')
    plt.plot(range(100), np.flipud(dataset[range(100),i]), 'r--', label = 'Actual')
    plt.xlabel('Time steps')
    plt.ylabel('Price')
    plt.title(labels[i]+' price' + ' for '+stock[:-4])
    plt.grid(True)
    plt.legend(loc = 'upper left')

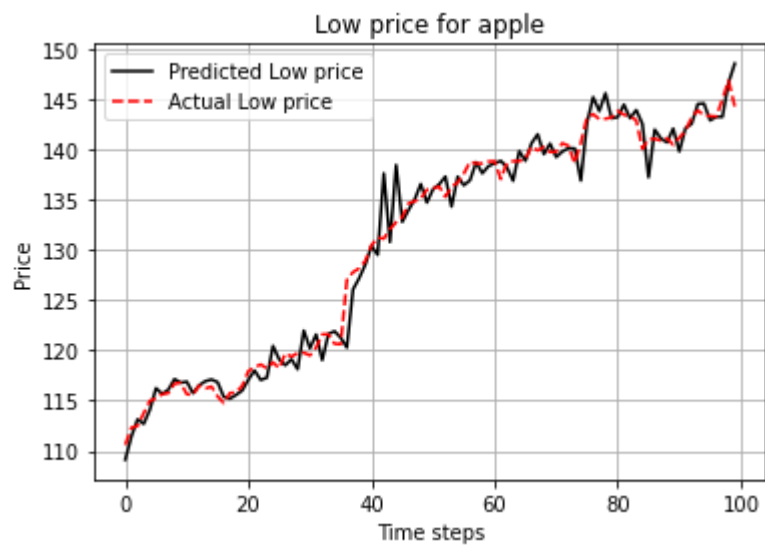
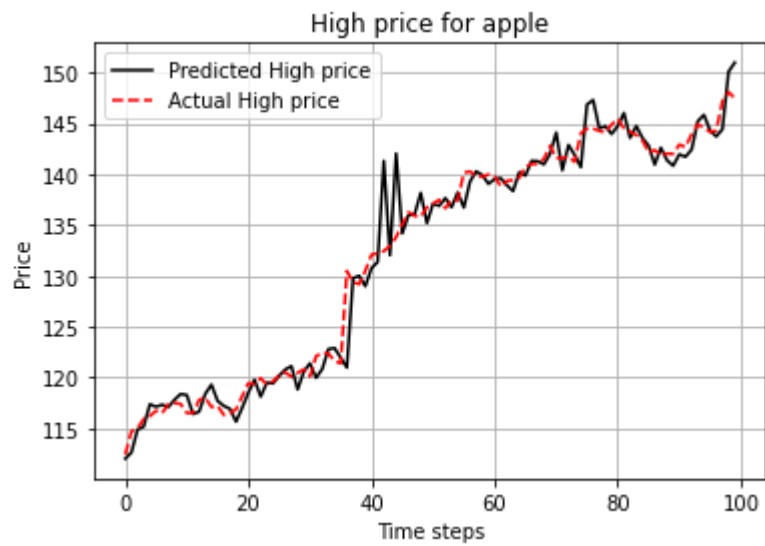
if PLOT_SHOW:
    plt.show(block=False)

```

Optimum number of states are 13

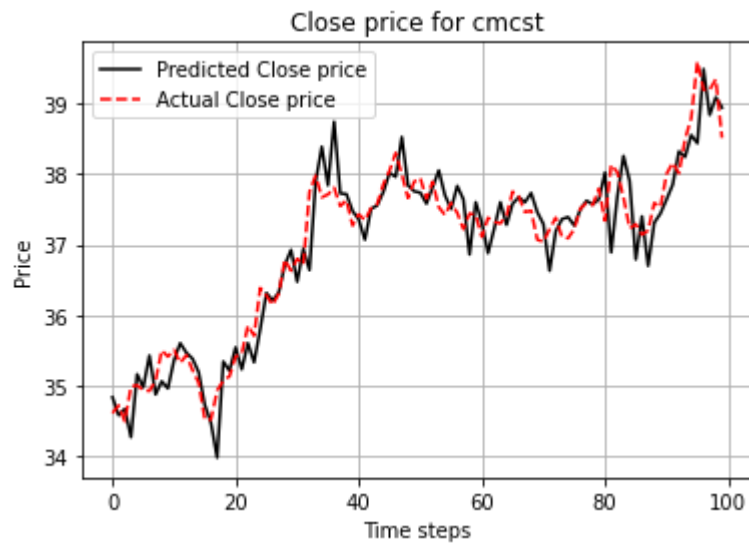
MAPE for the stock apple.csv is [0.00888693 0.00980546 0.00920139 0.00866747]

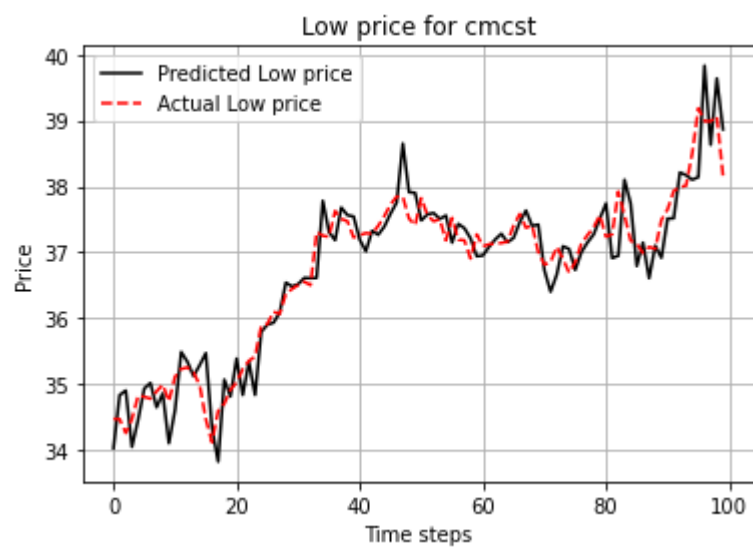
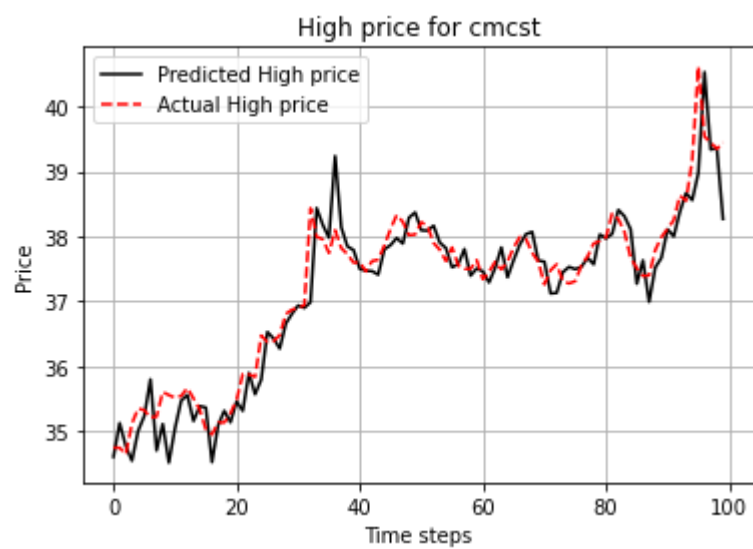
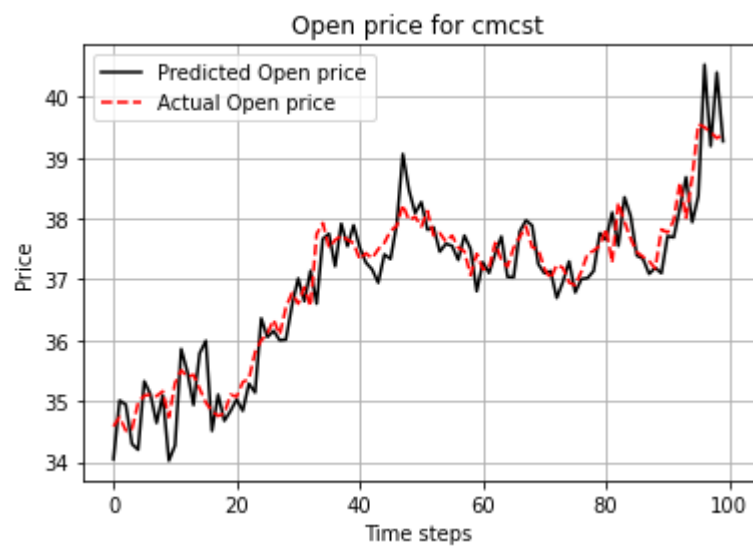




Optimum number of states are 14

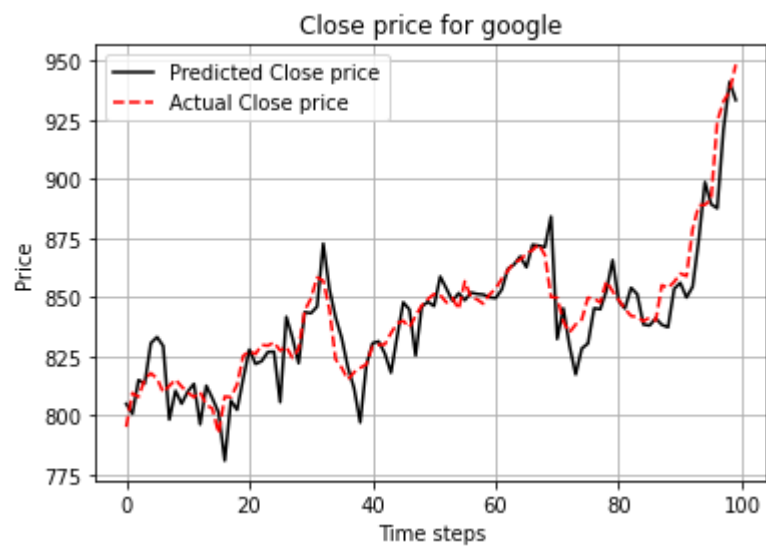
MAPE for the stock cmcst.csv is [0.00781854 0.00942235 0.0075632 0.00799607]





Optimum number of states are 14

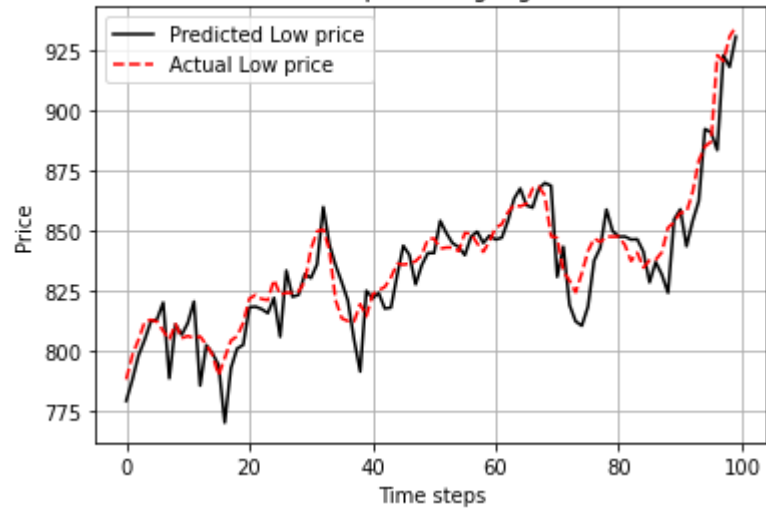
MAPE for the stock google.csv is [0.00989672 0.00972749 0.00963705 0.0097885]



High price for google



Low price for google




```
In [2]: print(predicted_stock_data)
```

```
[[804.93  777.95  803.44  778.95 ]
 [800.73  788.69  815.84  787.815 ]
 [815.01  795.04  826.79  797.96  ]
 [813.46  800.56  828.19  804.4411]
 [830.63  812.15  831.02  812.15  ]
 [833.18  815.68  830.98  812.99  ]
 [829.41  825.46  832.5   819.9686]
 [798.29  807.76  806.29  788.335 ]
 [810.46  819.47  821.5   810.8514]
 [805.01  818.04  818.88  806.5   ]
 [810.16  825.91  821.    811.4514]
 [813.38  822.1   818.07  820.405 ]
 [796.25  797.46  798.0596 785.325 ]
 [812.59  799.65  813.02  802.18  ]
 [807.23  804.3   810.35  798.82  ]
 [801.19  791.33  798.17  793.848 ]
 [780.9   792.66  790.375 769.835 ]
 [806.32  789.62  803.855 792.594 ]
 [802.41  814.54  810.76  800.75  ]
 [815.68  798.47  810.76  802.3   ]
 [827.91  819.08  829.23  818.    ]
 [821.82  831.02  827.76  818.26  ]
 [823.01  829.42  828.64  817.24  ]
 [826.86  828.97  829.13  815.57  ]
 [827.01  821.76  825.73  821.94  ]
 [805.64  839.17  833.    805.715 ]
 [841.74  836.35  841.13  833.3451]
 [832.87  829.35  830.3   822.41  ]
 [822.13  830.44  831.7399 823.18  ]
 [843.73  826.5   837.39  831.87  ]
 [843.26  832.31  844.5228 830.22  ]
 [846.05  845.98  847.05  835.9601]
 [872.73  859.9   869.7436 859.885 ]
 [853.5   858.05  856.53  844.2002]
 [841.55  858.    862.53  835.5801]
 [832.75  843.63  844.5036 828.49  ]
 [820.39  830.39  830.29  820.84  ]
 [811.6   806.44  809.84  804.62  ]
 [797.06  793.06  794.79  791.181 ]
 [821.36  816.22  819.51  824.89  ]
 [830.54  827.49  829.6636 821.75  ]
 [831.36  826.17  836.9504 823.9826]
 [826.24  812.97  820.09  817.48  ]
 [818.11  831.68  837.98  817.8799]
 [833.38  838.45  839.3564 831.2901]
 [847.88  844.27  849.0086 843.706  ]
 [844.82  840.99  847.17  839.84  ]
 [825.37  838.76  847.7725 827.5999]
 [846.02  838.05  843.18  835.59  ]
 [848.11  841.11  846.57  840.6599]
 [846.27  850.34  851.43  840.65  ]
 [858.8   855.57  866.74  854.119 ]
 [853.7   855.17  852.89  849.    ]
 [848.46  852.68  850.69  844.7775]
 [851.76  844.96  852.26  843.1701]
```

```

[848.78 846.9 849.32 839.7722]
[852.01 853.78 856.16 847.4522]
[851.41 856.11 856.12 849.5999]
[851.17 848.95 852.41 844.8651]
[849.99 853.54 854.2 847.94 ]
[849.68 852.76 855.5364 846.3001]
[853.28 856.2 855.76 847.0399]
[862.04 854.26 864.49 854.09 ]
[863.895 868.56 867.829 863.34 ]
[867.3 867.51 872.39 867.59 ]
[862.72 860.32 863.32 860.5901]
[872.24 867.49 870.37 859.625 ]
[871.86 867.83 875.02 867.5699]
[871.2 874.38 873.4 869.89 ]
[884.17 872. 886.6428 868.765 ]
[832.37 870.64 876.6 830.709 ]
[845.34 845.28 851.27 843.295 ]
[829.5 833.3 828.03 819. ]
[817.37 842.58 847.13 812.122 ]
[828.36 820. 827.72 810.3 ]
[830.48 831.6 831.74 818.265 ]
[845.36 843.36 853.895 837.48 ]
[844.97 852.59 854.31 842.87 ]
[852.9 862.2 855.5372 858.8245]
[865.7 850.67 868.44 849.8151]
[848.39 847.25 847. 847.4802]
[845.25 861.42 868.18 847.535 ]
[854.045 851.42 863.03 846.2853]
[851.34 848.06 852.065 846.415 ]
[838.51 838.11 842.4772 841.2501]
[838.06 841.86 842.5228 828.41 ]
[841.12 817.88 825.6 836.903 ]
[838.36 841.2 841.6228 831.6603]
[837.36 841.96 858.77 824.052 ]
[853.59 849.08 858.2472 854.74 ]
[856.12 866.615 860.615 858.9175]
[849.93 851.65 850.27 843.5 ]
[854.49 856.42 858.36 854.025 ]
[874.47 864.24 875.88 862.405 ]
[898.75 896.08 904.54 892.4502]
[889.44 900.52 893.73 891.0199]
[887.625 884.79 886.3774 883.66 ]
[920.86 935.71 943.49 923.235 ]
[941.12 919.3 935.74 918.38 ]
[933.43 939.98 950.575 931.015 ]]

```

In []: