

Data Generation

```
# !pip install fredapi
# !pip install quandl
# !pip install hmmlearn
# !pip install yfinance
# !pip install openpyxl

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from IPython.display import Image
from fredapi import Fred
import quandl
import yfinance as yf
import plotly.express as px

%matplotlib inline
```

First we combine all the data we received from the excel sheet.

```
basepath = '/data/workspace_files/'
freq = 'W-MON'
```

```

# Squeeze_data
filename = 'Squeeze_Metrics.csv'
df_squeeze = pd.read_csv(basepath+filename)
df_squeeze.index = pd.to_datetime(df_squeeze['date'])
df_squeeze = df_squeeze.drop(columns={'date'})

# Sector_Valuation
filename = 'Sector_Valuation.csv'
df_sector_val = pd.read_csv(basepath+filename)
df_sector_val.index = pd.to_datetime(df_sector_val['date'])
df_sector_val = df_sector_val.drop(columns={'date', 'Real Estate'})

# Implied Correlation
filename = 'IC.csv'
df_IC = pd.read_csv(basepath+filename)
df_IC.index = pd.to_datetime(df_IC['date'])
df_IC = df_IC.drop(columns={'date'})

# Equity_Risk_Premium
filename = 'Equity_Risk_Premium.csv'
df_ER = pd.read_csv(basepath+filename)
df_ER.index = pd.to_datetime(df_ER['date'])
df_ER = df_ER.drop(columns={'date'})
df_ER['Real_Bond_Yield'] = pd.to_numeric(df_ER['Real_Bond_Yield'])

# Average Investor Holding
filename = 'Avg_Investor_Holding.csv'
df_Avg_Inv_Hol = pd.read_csv(basepath+filename)
df_Avg_Inv_Hol.index = pd.to_datetime(df_Avg_Inv_Hol['date'])
df_Avg_Inv_Hol = df_Avg_Inv_Hol.drop(columns={'date'})

# Skew
filename = 'SKEW.csv'
df_skew = pd.read_csv(basepath+filename)
df_skew.index = pd.to_datetime(df_skew['date'])
df_skew = df_skew.drop(columns={'date'})

# Liquidity
filename = 'IAQF_Liquidity.csv'
df_liquidity = pd.read_csv(basepath+filename)
df_liquidity.index = pd.to_datetime(df_liquidity['date'])
df_liquidity = df_liquidity.drop(columns={'date'})

```

Fred data

```
fred = Fred(api_key='d19731bfe189a749611dfbace3924358')
```

```

rates = {'DFF': 'Fed_Fund_Rate', 'DGS1M0': 'Rate_1M', 'DGS3M0': 'Rate_3M', 'DGS6M0': 'Rate_6M', 'DGS1': 'Rate_1Y', 'DGS2': 'Rate_2Y',
        'DGS3': 'Rate_3Y', 'DGS5': 'Rate_5Y', 'DGS7': 'Rate_7Y', 'DGS10': 'Rate_10Y'}

VIX = {'VIXCLS': 'VIX', 'VXVCLS': 'VIX_3M'}

money_credit = {'T10Y2Y': 'Term_Spread',
                'BAMLH0A0HYM2': 'HY_OAS', 'BAMLC0A0CM': 'IG_OAS', 'BAMLC0A4CBBB': 'BBB_OAS',
                'BAMLEMCBP1OAS': 'EM_OAS'}

all_vars = {**rates, **VIX, **money_credit}
final_D = pd.DataFrame()

for var in all_vars:
    var_freq = fred.get_series_info(var)
    series_tmp = fred.get_series(var, aggregation_method='eop')
    series_df_tmp = pd.DataFrame(series_tmp, columns=[all_vars[var]])
    if len(final_D)==0:
        final_D = series_df_tmp
    else:
        final_D = pd.concat([final_D, series_df_tmp], axis=1)

```

```
final_D = final_D.loc[pd.to_datetime('1990-01-02'),:]
```

```
index = pd.bdate_range(start='2/1/1990', end='2/14/2022')
final_D = final_D.loc[index.values]
```

```
consumer_sentiment = {'UMCSENT': 'Cons_Sent', 'CPIAUCNS': 'CPI'}
industrial_data = {'AMTMNO': 'Manufacturing_New_Order'}
labor_market = {'UNRATE': 'Unemployment_rate'}
case_shiller = {'CSUSHPINSA': 'Case Shiller'}

all_vars = {**consumer_sentiment, **industrial_data, **labor_market, **case_shiller}
final_M = pd.DataFrame()

for var in all_vars:
    var_freq = fred.get_series_info(var)
    series_tmp = fred.get_series(var, aggregation_method='eop')
    series_df_tmp = pd.DataFrame(series_tmp, columns=[all_vars[var]])
    if len(final_M)==0:
        final_M = series_df_tmp
    else:
        final_M = pd.concat([final_M, series_df_tmp], axis=1).dropna()
```

```
GDP_based_indicators = {'JHDUSRGPBR': 'GDP_based_Recess_Ind',
                        'NA000334Q': 'GDP'}
all_vars = {**GDP_based_indicators}
final_Q = pd.DataFrame()

for var in all_vars:
    var_freq = fred.get_series_info(var)
    series_tmp = fred.get_series(var, aggregation_method='eop')
    series_df_tmp = pd.DataFrame(series_tmp, columns=[all_vars[var]])
    if len(final_Q)==0:
        final_Q = series_df_tmp
    else:
        final_Q = pd.concat([final_Q, series_df_tmp], axis=1).dropna()
```

```
asset = '^VIX'
VIX_data_df = yf.download(asset,
                          start='1990-01-01',
                          end='2021-12-31',
                          progress=False)[['Close']].rename(columns={'Close': 'VIX'})
```

```
final = pd.concat([final_D, final_M, final_Q, df_squeeze, df_sector_val,
                  df_IC, df_ER, df_Avg_Inv_Hol, VIX_data_df, df_skew, df_liquidity], axis=1)
final = final.loc[pd.to_datetime('1990-01-01'),:]
freq = 'W-MON'
final = final.resample(freq).first()
final = final.ffill()
```

```
filter = final.isna().sum()/final.shape[0]
filter_new = filter[filter<0.70]
filter_new.index

final_new_70 = final[filter_new.index].dropna()
```

```
filter = final.isna().sum()/final.shape[0]
filter_new = filter[filter<0.1]
filter_new.index

final_new_10 = final[filter_new.index].dropna()
```

Data from Yahoo Fin

```
filter = final.isna().sum()/final.shape[0]
filter_new = filter[filter<0.30]
filter_new.index

final_new_30 = final[filter_new.index].dropna()
```

```
def preprocess_spy_vix_data(filename='MFE+230K+HW1.xlsx', basepath = '/data/workspace_files/'):
    """
    Read SPY Index, VIX Index, and Risk free rate from the xlsx file.
    preprocess the data.

    :param filename (str): name of the data file
    :param basepath (str): name of the base path
    :return data (DataFrame): all the data
    """

    data = pd.read_excel(basepath+filename,
                        sheet_name='SP',
                        header=1,
                        index_col=0)
    data.index = pd.to_datetime(data.index)
    data = data.dropna()
    data = data.rename(columns={'S&P Return Index': 'SP500',
                              'Overnight Risk Free Rate': 'RF'})
    data = data.drop(columns={'VIX'})

    data['RF'] = data['RF'] / 100
    return data

spy_vix_data = preprocess_spy_vix_data()
```

```
asset = '^RUA'
russe1_data_df = yf.download(asset,
                             start='1990-01-01',
                             end='2021-12-31',
                             progress=False)
```

```
asset = 'IWM'
russe1_etf_data_df = yf.download(asset,
                                  start='1990-01-01',
                                  end='2021-12-31',
                                  progress=False)
```

```
spx_pe = quandl.get("MULTPL/SP500_PE_RATIO_MONTH", authtoken="Go75ZA9T38nzzWtKMhZ7", start_date= '1990-01-01',
                    end_date= '2021-12-31')
```

```
spx_price = yf.download('^GSPC',
                        start='1990-01-01',
                        end='2021-12-31',
                        progress=False)
spx_data = pd.merge(spx_price['Close'], spx_pe, left_index = True, right_index = True, how = 'outer')
spx_data['Value'] = spx_data['Value'].shift(1)
spx_data = spx_data.dropna(how='all')
spx_data['Close'].fillna(method = 'ffill', inplace = True)
spx_data['earnings'] = spx_data['Close']/spx_data['Value']
spx_data['earnings'].fillna(method = 'ffill', inplace = True)
spx_data['pe'] = spx_data['Close']/spx_data['earnings']
```

Fama Fench factors

```
ff_data = pd.read_csv('/data/workspace_files/F-F_Research_Data_5_Factors_2x3_daily.csv', skiprows = 3, index_col = 0, parse_dates = True)
ff_data = ff_data.loc['1990-01-01':]
ff_data.drop('RF', axis = 1, inplace = True)
```

```
# cumulative return
ff_data_w = ((ff_data/100.+1).apply(np.log).resample(freq).sum().apply(np.exp) - 1)*100
```

Volume

```
data_df = pd.concat([russef_data_df, spy_vix_data[['RF']], spx_data['pe']],axis=1).dropna()
#data_df['Volume'] = russef_data_df['Volume']
data_df = data_df.drop(columns=['Volume'])
data_df_W = data_df.resample(freq).first()
```

```
# add ff factors
data_df = pd.concat([data_df, ff_data], axis=1).dropna()
data_df_W = pd.concat([data_df_W, ff_data_w], axis=1).dropna()
```

```
# Feature params
# corresponds to biweekly frequency
ma_period = 10
column_price = 'Close'

#computed daily
data_df['close_open_return'] = (data_df['Open'] - data_df['Close'].shift(1)) / data_df['Open']

close_weekly = data_df[column_price].resample(freq).first()
data_df_W['last_return'] = close_weekly.pct_change() #weekly return
data_df_W['last_return1'] = close_weekly.pct_change(periods=4) #monthly
data_df_W['last_return2'] = data_df[column_price].pct_change(periods=12) #quarterly
data_df_W['close_open_return'] = (data_df[['close_open_return']]+1.).apply(np.log).resample(freq).sum().apply(np.exp) - 1.

#forward looking return
return_period = 1 # future reutrnr
data_df_W["future_return"] = close_weekly.pct_change(return_period).shift(-return_period)

def values_deviation(vals):
    """ z-score for volumes and price """
    return (vals[-1] - np.mean(vals)) / np.std(vals)

price_deviation_period = 20
data_df['price_deviation'] = data_df[column_price].rolling(price_deviation_period).apply(values_deviation)
#data_df['volume_deviation'] = data_df['Volume'].rolling(5).apply(values_deviation) #weekly

data_df_W["price_deviation"] = data_df["price_deviation"].resample(freq).first()
#data_df_W["volume_deviation"] = data_df["volume_deviation"].resample(freq).first()

data_df = data_df.replace([np.inf, -np.inf], np.nan)
data_df_W = data_df_W.replace([np.inf, -np.inf], np.nan)
```

```
orig_data_df = data_df.copy()
orig_data_df_W = data_df_W.copy()
```

```
data_df_W = orig_data_df_W[['Close',
                             'RF',
                             'last_return',
                             'last_return1',
                             'last_return2',
                             'close_open_return',
                             'future_return',
                             "pe",
                             "Mkt-RF", 'SMB', 'HML', 'RMW', 'CMA',
                             'price_deviation',
                             ]]
```

```
data_df_W = data_df_W.ffill().dropna()
```

```
# Transform the data
def transform(data):
    #data['Close'] = data['Close'].diff()
    data['pe'] = data['pe'].apply(np.log).diff()
    data['Fed_Fund_Rate'] = data['Fed_Fund_Rate'].diff()
    data['VIX'] = data['VIX'].apply(np.log).diff()
    data['Cons_Sent'] = data['Cons_Sent'].apply(np.log).diff()
    data['CPI'] = data['CPI'].diff()
    data['Manufacturing_New_Order'] = data['Manufacturing_New_Order'].diff()

    for c in ['Avg. Stock Holding', 'SKEW', 'Consumer Staples',
              'Equity Risk Premium', 'Long_Term_Growth_Forecast',
              'Real_Bond_Yield', 'CAPE', 'GDP', 'Case Shiller',
              'Unemployment_rate', 'Energy', 'Financials', 'Industrials', 'Consumer Discretionary',
              'Information Technology', 'Materials', 'Utilities', 'Communication Services', 'Health Care',
              'VVIX', 'HY_OAS', 'IG_OAS', 'BBB_OAS', 'price', 'gex', 'IC', 'Liquidity (Scaled by GDP)',
              'Rate_1M', 'Rate_3M', 'Rate_6M', 'Rate_1Y', 'Rate_2Y',
              'Rate_3Y', 'Rate_5Y', 'Rate_7Y', 'Rate_10Y', 'Term_Spread',]:
        if c in data.columns:
            data[c] = data[c].diff()

    return data
```

```
data_temp = pd.concat([data_df_W, final_new_10],axis=1).dropna()
data_fut_return = data_temp[['future_return']]
data_df_92 = data_temp.drop(columns={'future_return'})

data_vix = data_temp[['VIX']]
data_df_92 = data_temp.drop(columns={'VIX'})

data_rf = data_temp[['RF']]
data_df_92 = data_temp.drop(columns={'RF'})
data_df_W = data_df_W.drop(columns={'RF'})

data_df_92 = transform(data_df_92)
```

```
data_df_92.columns
```

```
Index(['Close', 'last_return', 'last_return1', 'last_return2',
       'close_open_return', 'future_return', 'pe', 'Mkt-RF', 'SMB', 'HML',
       'RMW', 'CMA', 'price_deviation', 'Fed_Fund_Rate', 'Rate_3M', 'Rate_6M',
       'Rate_1Y', 'Rate_2Y', 'Rate_3Y', 'Rate_5Y', 'Rate_7Y', 'Rate_10Y',
       'VIX', 'Term_Spread', 'Cons_Sent', 'CPI', 'Manufacturing_New_Order',
       'Unemployment_rate', 'Case Shiller', 'GDP_based_Recess_Ind', 'GDP',
       'CAPE', 'Real_Bond_Yield', 'Long_Term_Growth_Forecast',
       'Equity Risk Premium', 'Avg. Stock Holding', 'SKEW',
       'Liquidity (Scaled by GDP)'],
      dtype='object')
```

```
data_df_99 = pd.concat([data_df_W, final_new_30],axis=1).dropna()
data_df_99 = data_df_99.drop(columns={'future_return'})
data_df_99 = transform(data_df_99)
```

```
data_df_11 = pd.concat([data_df_W, final_new_70],axis=1).dropna()
data_df_11 = data_df_11.drop(columns={'future_return'})
data_df_11 = transform(data_df_11)
```

```
from pandas import read_csv
from statsmodels.tsa.stattools import adfuller
def is_stationary(X):
    # Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
    result = adfuller(X)
    return result[1] < 0.05

for col in data_df_92.columns:
    if not is_stationary(data_df_92[col].dropna()):
        print(col)

print("99:")
for col in data_df_99.columns:

    if not is_stationary(data_df_99[col].dropna()):
        print(col)

print("11:")
for col in data_df_11.columns:
    if not is_stationary(data_df_11[col].dropna()):
        print(col)
```

```
Close
99:
Close
11:
Close
```

```
data_close = data_df_92[['Close']]
```

Dump to pickle file

```
pd.to_pickle(data_df_92.dropna(), '/data/workspace_files/data_df92.pickle') #37 columns

pd.to_pickle(data_fut_return.dropna(), '/data/workspace_files/data_fut_return.pickle')
pd.to_pickle(data_vix.dropna(), '/data/workspace_files/data_vix.pickle')
pd.to_pickle(data_rf.dropna(), '/data/workspace_files/data_rf.pickle')
pd.to_pickle(data_close.dropna(), '/data/workspace_files/data_close.pickle')

pd.to_pickle(data_df_99.dropna(), '/data/workspace_files/data_df99.pickle') #51 columns
pd.to_pickle(data_df_11.dropna(), '/data/workspace_files/data_df11.pickle') #58 columns
```

```
data_df_92.columns
```

```
Index(['Close', 'last_return', 'last_return1', 'last_return2',
      'close_open_return', 'future_return', 'pe', 'Mkt-RF', 'SMB', 'HML',
      'RMW', 'CMA', 'price_deviation', 'Fed_Fund_Rate', 'Rate_3M', 'Rate_6M',
      'Rate_1Y', 'Rate_2Y', 'Rate_3Y', 'Rate_5Y', 'Rate_7Y', 'Rate_10Y',
      'VIX', 'Term_Spread', 'Cons_Sent', 'CPI', 'Manufacturing_New_Order',
      'Unemployment_rate', 'Case_Shiller', 'GDP_based_Recess_Ind', 'GDP',
      'CAPE', 'Real_Bond_Yield', 'Long_Term_Growth_Forecast',
      'Equity_Risk_Premium', 'Avg. Stock Holding', 'SKEW',
      'Liquidity (Scaled by GDP)'],
      dtype='object')
```

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. Observed data is our market features, hidden states are our market behavior.

It is simple enough. But also its rich enough.

We can use it as a baseline model.

<https://www.youtube.com/watch?v=TPRoLreU9IA>

```
import quandl
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from matplotlib import cm, pyplot as plt
from hmmlearn.hmm import GaussianHMM
import scipy
import datetime
import json
import seaborn as sns
import joblib
import quandl
from dateutil.relativedelta import relativedelta

import yfinance as yf
import plotly.express as px
from statistics import mode

import numpy as np
from sklearn.decomposition import SparsePCA, PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings
warnings.filterwarnings("ignore")
```

load data

```
vix_df = pd.read_pickle('/data/workspace_files/data_vix.pickle')
rf_df = pd.read_pickle('/data/workspace_files/data_rf.pickle')
fut_return = pd.read_pickle('/data/workspace_files/data_fut_return.pickle')
```

PCA

```
def pca_algo(data_df):  
    x92 = StandardScaler().fit_transform(data_df) #standardize  
    pca_transformer = PCA()  
    pca_transformer.fit(x92)  
    px.line(pd.DataFrame(pca_transformer.explained_variance_ratio_).cumsum())  
  
    pca_model = PCA(n_components=20).fit(x92)  
    pca_data = pca_model.transform(x92)  
    pca_data = pd.DataFrame(pca_data, index=data_df.index)  
    return pca_data
```

SPARSE PCA

```
def spca_algo(data_df):  
    x92 = StandardScaler().fit_transform(data_df) #standardize  
    spca_transformer = SparsePCA(n_components=12, alpha=1., random_state=0)  
    spca_transformer.fit(x92)  
    spca_data = spca_transformer.transform(x92)  
    spca_data = pd.DataFrame(spca_data, index=data_df.index)  
    return spca_data
```

HMM Model

```

def plot_hidden_states(model, data, X, close_series, fut_ret_series):
    plt.figure(figsize=(15, 15))
    fig, axs = plt.subplots(model.n_components, 3, figsize = (15, 15))
    colours = cm.prism(np.linspace(0, 1, model.n_components))
    hidden_states = model.predict(X)

    print(data.shape, X.shape, close_series.shape, fut_ret_series.shape)

    ret_list = []
    for i, (ax, colour) in enumerate(zip(axs, colours)):
        mask = hidden_states == i
        ax[0].plot(data.index, close_series, c = 'grey')
        ax[0].plot(data.index[mask], close_series[mask], '.', c = colour)
        ax[0].set_title("{0}th hidden state".format(i))
        ax[0].grid(True)

        ax[1].hist(fut_ret_series[mask], bins = 30)
        ax[1].set_xlim([-0.1, 0.1])
        ax[1].set_title("future return distribution at {0}th hidden state".format(i))
        ax[1].grid(True)

        cum_return = (fut_ret_series[mask] + 1).cumprod()
        ax[2].plot(cum_return, c = colour)
        ax[2].set_title("cumulative future return at {0}th hidden state".format(i))
        ax[2].grid(True)
        ret_list.append(cum_return.iloc[-1].squeeze())
    plt.tight_layout()
    return ret_list

```

```

# Brute force modelling
def hmm_model(data, state, max_iter = 10000):
    best_model = GaussianHMM(n_components = state,
                              random_state = 100,
                              covariance_type = "full", n_iter = max_iter).fit(data)

    return best_model

```

```

def get_data(mtype, data_df):
    if mtype == "spca":
        spca_data = spca_algo(data_df)
        return spca_data.copy()
    elif mtype == "pca":
        pca_data = pca_algo(data_df)
        return pca_data.copy()
    elif type(mtype)==list:
        return data_df[mtype].copy()

def train_hmm_model(train_data, data_df, close_data, fut_return, filen):
    train_ind = int(np.where(train_data.index >= pd.to_datetime('2018-01-01 00:00:00'))[0][0])
    train_set = train_data.iloc[:train_ind]
    test_set = train_data.iloc[train_ind:]
    test_set_orig = data_df.loc[test_set.index]
    print("train set size is ", train_set.shape[0], ", test set size is ", test_set.shape[0])

    model = hmm_model(train_set, state=3, max_iter=1000000)
    print("Best model with {0} states ".format(str(model.n_components)))

    rets = plot_hidden_states(model,
                              data_df.iloc[:train_ind],
                              train_set,
                              close_data[:train_ind],
                              fut_return.iloc[:train_ind])

    test_hidden_states = model.predict(test_set)

    bear_id = rets.index(min(rets))
    bull_id = rets.index(max(rets))
    neutral_id = 3 - bear_id - bull_id
    print("bull", bull_id, " bear", bear_id, " neutral", neutral_id)

    states = {"bear": bear_id,
              "bull": bull_id,
              "static": neutral_id}

    signal_map = {v: k for k, v in states.items()}
    test_set_res = train_data.iloc[train_ind:]
    test_orig_data = data_df.loc[train_data.index][train_ind:]

    test_orig_data['signal'] = test_hidden_states
    test_set_res['signal'] = test_hidden_states
    actions = np.select([
        (test_orig_data[['signal']] == states['bull']),
        (test_orig_data[['signal']] == states['bear']),
        (test_orig_data[['signal']] == states['static'])],
        [1, -1, 0])
    test_set_res['weight'] = actions

    train_h = pd.DataFrame(model.predict(train_set), columns=['signal'], index=train_set.index)
    train_out = pd.DataFrame(np.select([
        (train_h[['signal']] == states['bull']),
        (train_h[['signal']] == states['bear']),
        (train_h[['signal']] == states['static'])],
        [1, -1, 0])

```

```

    ),
    index=train_set.index)
train_out.to_pickle('/data/workspace_files/output/hmm_%s_train.pickle'%filen)

out = pd.DataFrame(test_set_res['weight'], index=test_set.index)
out.to_pickle('/data/workspace_files/output/hmm_%s_test.pickle'%filen)

plot_hidden_states(model,
                    data_df.iloc[train_ind:],
                    test_set,
                    close_data.iloc[train_ind:],
                    fut_return.iloc[train_ind:])

return rets, model, test_set_res, test_set_orig

```

Baseline

```

data_df92 = pd.read_pickle('/data/workspace_files/data_df92.pickle')
close_data92 = data_df92[['Close']]
data_df92 = data_df92[['last_return']]
fut_return92 = fut_return.loc[data_df92.index]

rets, model, test_set_res, test_set_orig = train_hmm_model(data_df92,
                                                            data_df92,
                                                            close_data92,
                                                            fut_return92,
                                                            "baseline")

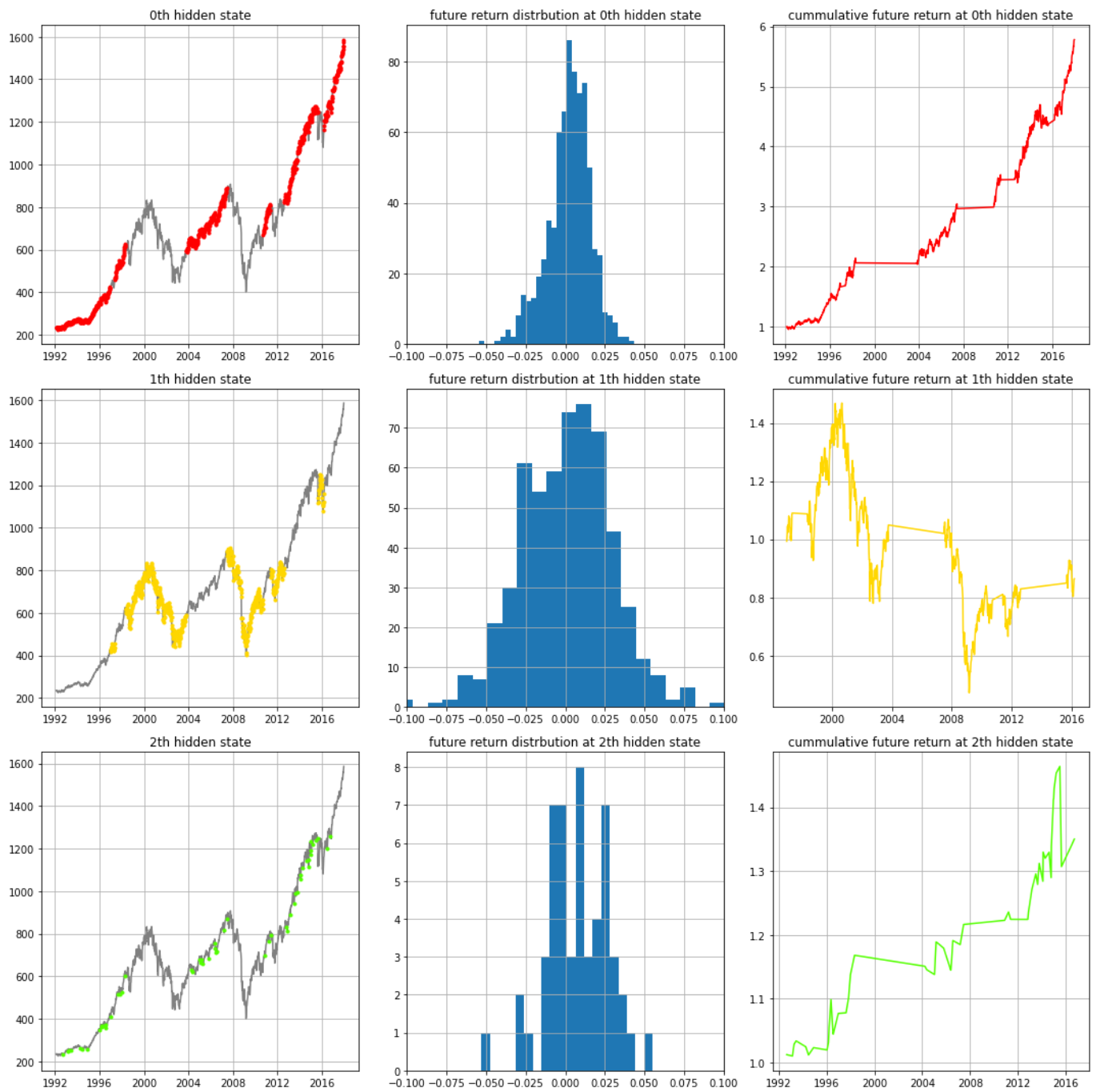
```

```

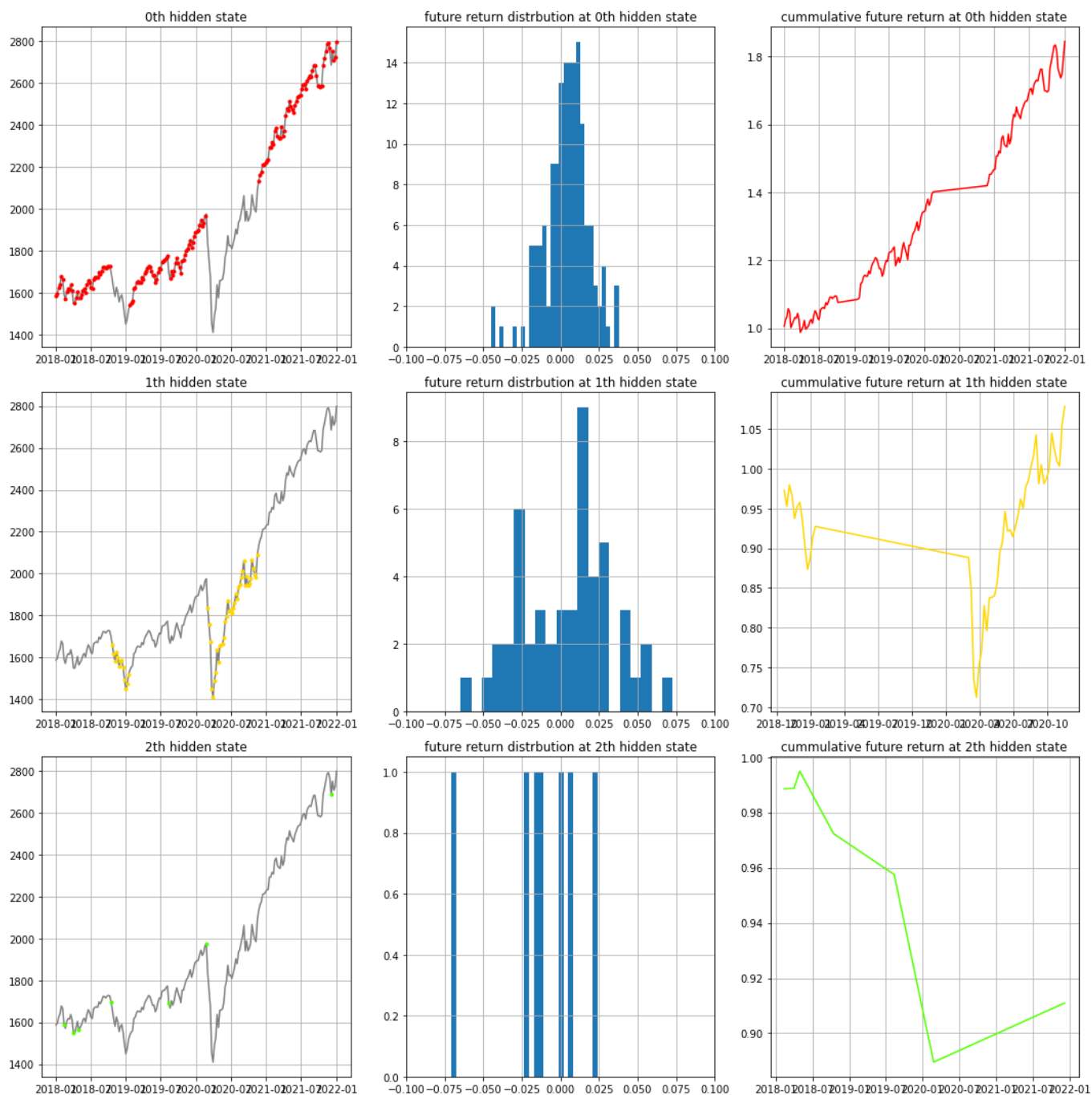
train set size is 1351 , test set size is 210
Best model with 3 states
(1351, 1) (1351, 1) (1351, 1) (1351, 1)
bull 0 bear 1 neutral 2
(210, 1) (210, 1) (210, 1) (210, 1)

```

<Figure size 1080x1080 with 0 Axes>



<Figure size 1080x1080 with 0 Axes>



Trading Strategy

[illegible]

Markov-Switching Model (MSM)

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from matplotlib import cm, pyplot as plt
from datetime import datetime
from io import BytesIO
from dateutil.relativedelta import relativedelta
import quandl
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import requests
import statsmodels.api as sm
import seaborn as sns
import joblib
```

```
import yfinance as yf
import plotly.express as px
```

```
data_df = pd.read_pickle('/data/workspace_files/data_df92.pickle')
rf_df = pd.read_pickle('/data/workspace_files/data_rf.pickle')
```

```
data_df = pd.merge_asof(data_df, rf_df, right_index=True, left_index=True)
```

```
# Split the data on sets
train_set = data_df.iloc[data_df.index < pd.to_datetime('2018-01-01 00:00:00')]
test_set = data_df.iloc[data_df.index >= pd.to_datetime('2017-12-22 00:00:00')]
train_ind = list(data_df.index).index(test_set.index[0]) + 1
print("train set size is ", train_set.shape[0], ", test set size is ", test_set.shape[0])
```

```
train set size is 1351 , test set size is 211
```

```
basepath = '/data/workspace_files/'

filename = 'Equity_Risk_Premium.csv'
df_ER = pd.read_csv(basepath+filename)
df_ER.index = pd.to_datetime(df_ER['date'])
df_ER = df_ER.drop(columns={'date'})
df_ER['Real_Bond_Yield'] = pd.to_numeric(df_ER['Real_Bond_Yield'])

filename = 'Avg_Investor_Holding.csv'
df_IH = pd.read_csv(basepath+filename)
df_IH.set_index('date', inplace=True)
df_IH.index = pd.to_datetime(df_IH.index)
```

Model

```

mod_hamilton = sm.tsa.MarkovAutoregression(
    train_set.future_return,
    k_regimes = 3,
    order = 1, switching_ar=True,
    exog_tvtp=train_set.VIX, #switching_exog=True,
    # trend='ct',
    # switching_trend=True,
    switching_variance=True
)
mod_hamilton_predicton = sm.tsa.MarkovAutoregression(
    test_set.future_return,
    k_regimes = 3,
    order = 1, switching_ar=True,
    exog_tvtp=test_set.VIX, #switching_exog=True,
    switching_variance=True
)
res_hamilton = mod_hamilton.fit()

```

```

/opt/python/envs/default/lib/python3.8/site-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization
warnings.warn("Maximum Likelihood optimization failed to "

```

```
res_hamilton.summary()
```

Markov Switching Model Results

```
res_hamilton.params
```

```
mod_hamilton.predict_conditional(res_hamilton.params)[:,:0]
```

```

fig, axes = plt.subplots(3, figsize=(7, 7))

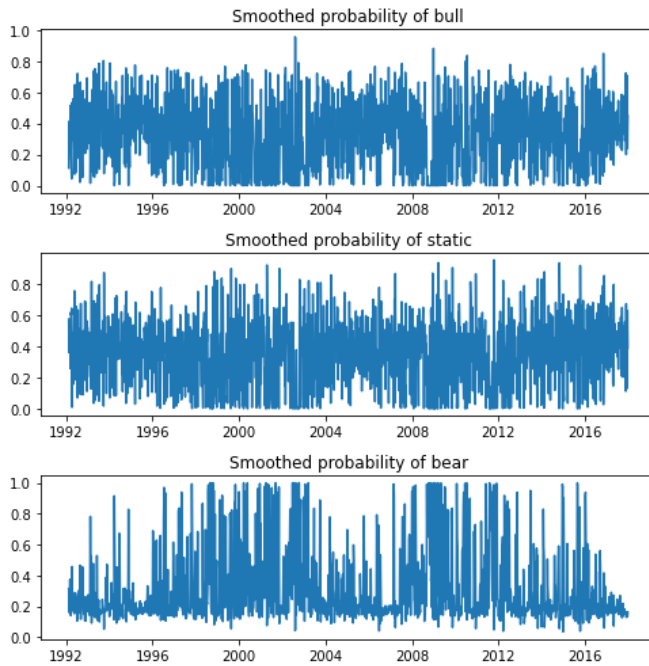
ax = axes[0]
ax.plot(res_hamilton.smoothed_marginal_probabilities[0])
ax.set(title="Smoothed probability of bull")

ax = axes[1]
ax.plot(res_hamilton.smoothed_marginal_probabilities[1])
ax.set(title="Smoothed probability of static")

ax = axes[2]
ax.plot(res_hamilton.smoothed_marginal_probabilities[2])
ax.set(title="Smoothed probability of bear")

fig.tight_layout()

```



```
res_hamilton.smoothed_marginal_probabilities.idxmax(axis=1).value_counts()
```

```
def plot_hidden_states(res_hamilton, data, column_price):
    plt.figure(figsize=(15, 15))
    fig, axs = plt.subplots(3, 3, figsize = (15, 15))
    colours = cm.prism(np.linspace(0, 1, 3))
    hidden_states = res_hamilton.smoothed_marginal_probabilities.idxmax(axis=1)
    print(hidden_states.value_counts())

    for i, (ax, colour) in enumerate(zip(axs, colours)):
        mask = hidden_states == i
        ax[0].plot(data.index, data[column_price], c = 'grey')
        ax[0].plot(data.index[mask], data[column_price][mask], '.', c = colour)
        ax[0].set_title("{0}th hidden state".format(i))
        ax[0].grid(True)

        ax[1].hist(data["future_return"][mask], bins = 30)
        ax[1].set_xlim([-0.1, 0.1])
        ax[1].set_title("future return distribution at {0}th hidden state".format(i))
        ax[1].grid(True)

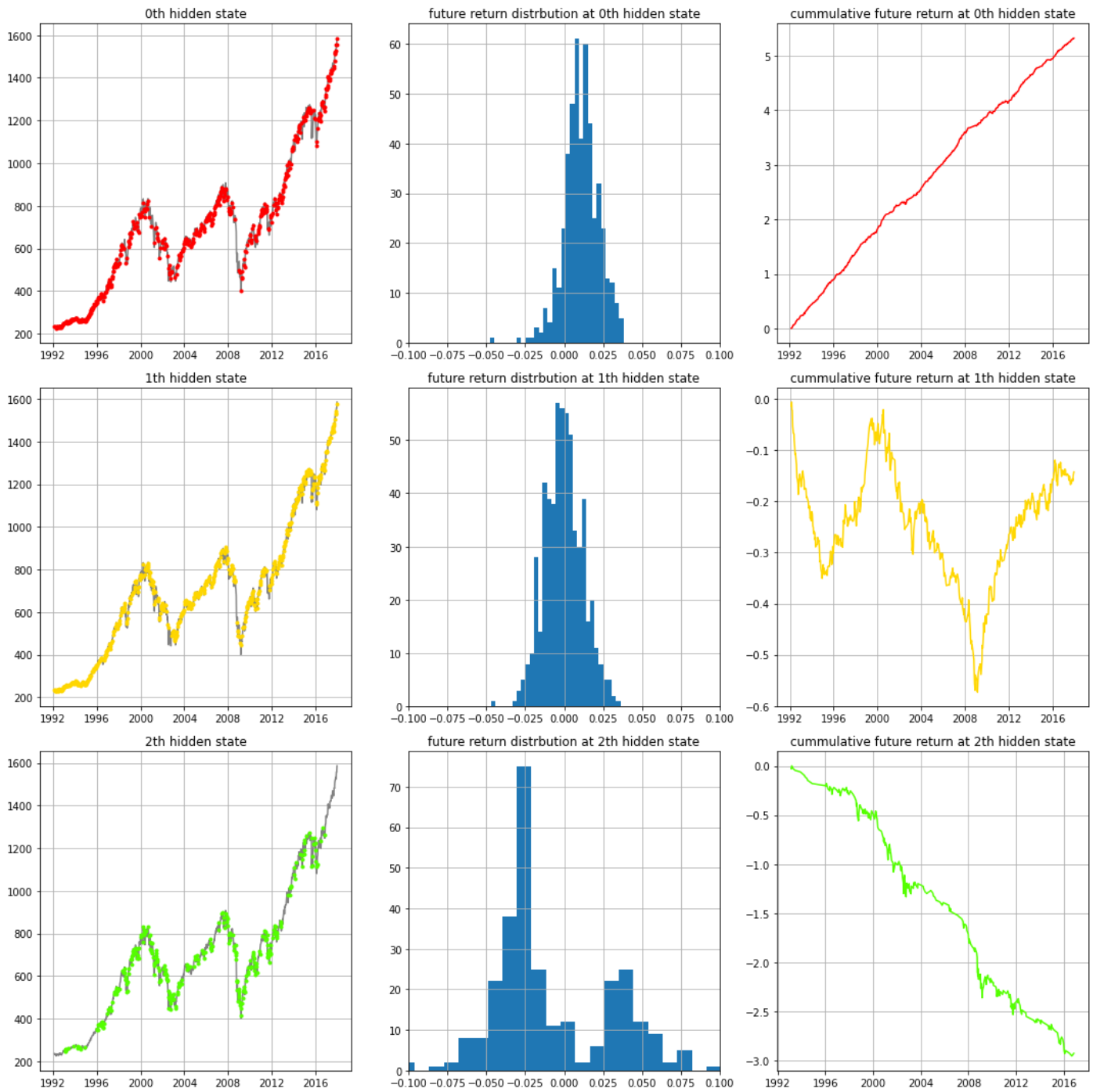
        ax[2].plot(data["future_return"][mask].cumsum(), c = colour)
        ax[2].set_title("cumulative future return at {0}th hidden state".format(i))
        ax[2].grid(True)

    plt.tight_layout()
```

```
plot_hidden_states(res_hamilton, data_df[:train_ind][1:], 'Close')
```

```
1    578
0    479
2    293
dtype: int64
```

<Figure size 1080x1080 with 0 Axes>



```
states_map = {"bear": 2,
              "bull": 0,
              "static": 1}

signal_map = {v: k for k, v in states_map.items()}
```

```
def state_predictor(params, model):
    conditional_predicition = model.predict(params=params, conditional=True).T
    model_prediction = model.predict(params=params)
    s = np.abs(conditional_predicition.T - [model_prediction] * 3)
    return s.argmax(axis=0)

test_set_res = test_set.copy()[1:]
test_set_res['signal'] = state_predictor(res_hamilton.params, mod_hamilton_predicition)
test_set_res.signal.value_counts()
```

```

train_set.drop(columns=['VIX'], inplace=True)
vix_df = pd.read_pickle('/data/workspace_files/data_vix.pickle')
train_set = pd.merge_asof(train_set, vix_df / 100, right_index=True, left_index=True)
test_set_res.drop(columns=['VIX'], inplace=True)
test_set_res = pd.merge_asof(test_set_res, vix_df / 100, right_index=True, left_index=True)

```

/opt/python/envs/default/lib/python3.8/site-packages/pandas/core/frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
return super().drop(

```

train_set_res = pd.merge_asof(train_set.copy()[1:], df_ER[['Equity Risk Premium']], left_index=True, right_index=True)
train_set_res = pd.merge_asof(train_set_res, df_IH[['Avg. Stock Holding']], left_index=True, right_index=True)
train_set_res['signal'] = np.array(res_hamilton.smoothed_marginal_probabilities).argmax(axis=1)

```

```

risk_aversion = dict()
risk_premium = dict()
for regime in ['bull', 'bear', 'static']:
    risk_aversion[regime] = 3 / train_set_res.VIX.mean() * train_set_res[train_set_res.signal==states_map[regime]].VIX.mean()
    # risk_aversion[regime] = 1 / train_set_res[train_set_res.signal==states_map[regime]]['Avg. Stock Holding'].mean()
    # risk_premium[regime] = train_set_res[train_set_res.signal==states_map[regime]]['Equity Risk Premium'].mean() - train_set_res[tra
    risk_premium[regime] = train_set_res[train_set_res.signal==states_map[regime]]['future_return'].mean() * 52 - train_set_res[train_

```

```
(risk_aversion, risk_premium)
```

```

({'bull': 2.835931188138735,
 'bear': 3.7231633660694152,
 'static': 2.769380786545341},
 {'bull': 0.5504972425280706,
 'bear': -0.5455112454439829,
 'static': -0.038761033165902264})

```

```

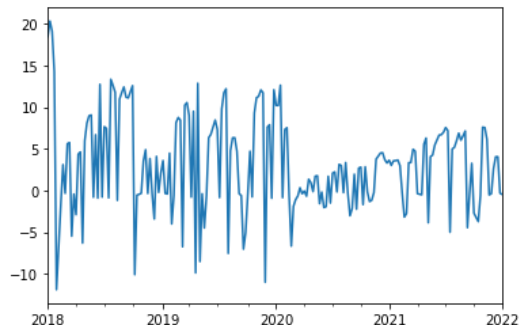
def mean_variance_weight(vol, signal):
    variance = vol**2
    weight= np.divide((1/risk_aversion[signal_map[signal]]) * (risk_premium[signal_map[signal]]), variance)
    return weight

```

```
test_set_res['weight'] = [mean_variance_weight(vol, int(signal)) for i, (vol, signal) in test_set_res[['VIX','signal']].iterrows()]
```

```
test_set_res['weight'].plot()
```

<AxesSubplot:>



```
test_set_res['weight'] = np.minimum(np.maximum(test_set_res['weight'], -1),1)
```

```
initial_capital = 100
test_set_res['strategy_1'] = 0.
test_set_res['strategy_2'] = 0.
test_set_res['strategy_1'].iloc[0] = initial_capital
test_set_res['strategy_2'].iloc[0] = initial_capital

v_t = initial_capital
### Strategy 1
for i in range(test_set_res.shape[0]-1):
    if test_set_res['signal'].iloc[i] == states_map['bull']:
        v_t = v_t * (1 + test_set_res['future_return'].iloc[i])
    if test_set_res['signal'].iloc[i] == states_map['bear']:
        v_t = v_t * (1 - test_set_res['future_return'].iloc[i])
    test_set_res['strategy_1'].iloc[i+1] = v_t

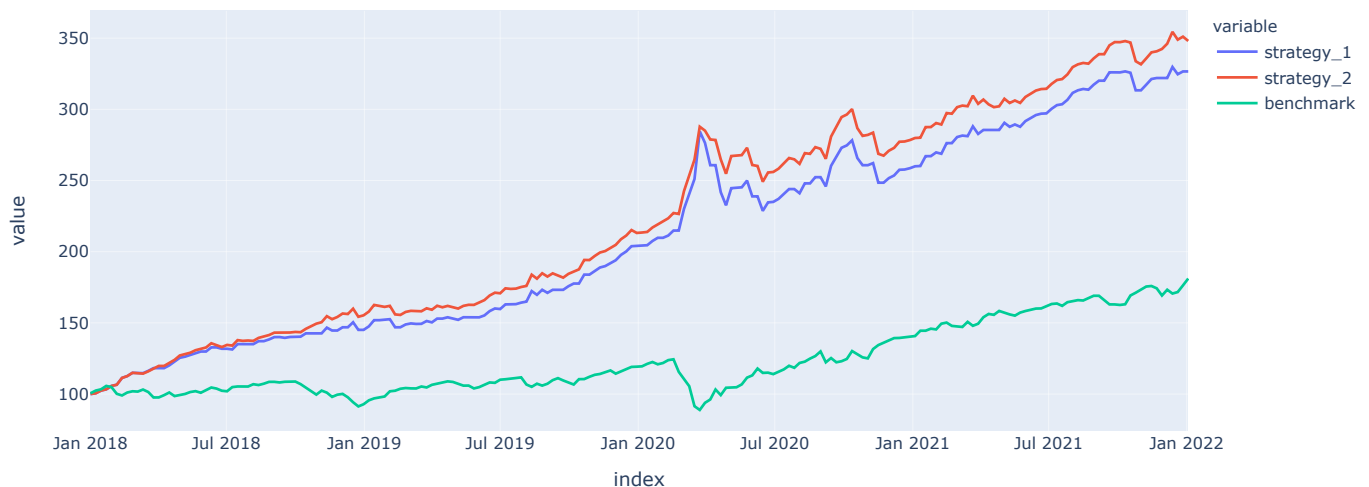
v_t = initial_capital
### Strategy 2
for i in range(0, test_set_res.shape[0]-1):
    v_t = v_t * (1 + test_set_res['future_return'].iloc[i] * test_set_res['weight'].iloc[i])
    test_set_res['strategy_2'].iloc[i+1] = v_t
```

/opt/python/envs/default/lib/python3.8/site-packages/pandas/core/indexing.py:1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
self._setitem_single_block(indexer, value, name)

```
test_set_res['benchmark'] = 100 * (test_set_res['future_return'] + 1).cumprod()
```

```
px.line(test_set_res[['strategy_1', 'strategy_2', 'benchmark']])
```



```
# test_set_new = pd.concat([train_set_res, spy_vix_data[['VIX', 'RF']].resample('W').last()], axis=1).dropna()
```

```
test_set_new = test_set_res.copy()
```

```

columns = ['strategy_1', 'strategy_2', 'benchmark']

df_summary = pd.DataFrame()
final = {}

for column in columns:
    df_returns = test_set_new[column].pct_change()
    s = {}

    yearly_multiplier = 52 #weekly
    year_count = relativedelta(test_set_new['RF'].index[-1], test_set_new['RF'].index[0]).years
    rf = (1 + test_set_new['RF'] / yearly_multiplier).prod() ** (1 / year_count) - 1
    s['Mean Return'] = np.exp(np.log(test_set_new[column]/test_set_new[column].shift(1)).mean() * yearly_multiplier) - 1
    s['Volatility'] = df_returns.std() * np.sqrt(yearly_multiplier)
    s['Skewness'] = df_returns.skew()
    s['Kurtosis'] = df_returns.kurtosis()
    s['Maximum Drawdown'] = (test_set_new[column] / test_set_new[column].cummax()-1).min()
    s['Sharpe Ratio'] = (s['Mean Return'] - rf) / s['Volatility']
    s['Semi-Deviation'] = df_returns[df_returns < df_returns.mean()].std()*np.sqrt(yearly_multiplier)
    s['1-month Autocorrelation'] = df_returns.autocorr(lag=1)
    s['3-month Autocorrelation'] = df_returns.autocorr(lag=3)
    s['1-year Autocorrelation'] = df_returns.autocorr(lag=12)
    s
    final[column] = s

df_summary = pd.DataFrame.from_dict(final, orient='index').round(4).T
# df_summary.to_csv('res.csv')
df_summary

```

	strategy_1	strategy_2	benchmark
--	------------	------------	-----------

```

## writing output of models
rename_map = {
    'bull' : 1,
    'bear' : -1,
    'static' : 0
}
replace_map = {v : rename_map[k] for k, v in states_map.items()}
train_set_res.signal.replace(replace_map, inplace=True)
test_set_res.signal.replace(replace_map, inplace=True)

train_set_res[['signal']].to_pickle('/data/workspace_files/output/MSM_train.pickle')
test_set_res[['signal']].to_pickle('/data/workspace_files/output/MSM_test.pickle')

```

```
# !pip install quandl
# !pip install hmmlearn
# !pip install yfinance

import quandl
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from matplotlib import cm, pyplot as plt
from hmmlearn.hmm import GaussianHMM
import scipy
import datetime
import json
import seaborn as sns
import joblib

import yfinance as yf
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')
```

```
from scipy.stats import mode

asset = '^RUA'
data_df = yf.download(asset,
                      start='1990-01-01',
                      end='2021-12-31',
                      progress=False)

freq = 'W-MON'
df_W = data_df.resample(freq).first()[['Close']]

df_W['return'] = df_W['Close']/df_W['Close'].shift(1) - 1
df_W['ret_avg'] = df_W['return'].rolling(8).mean()

conditions = [ df_W['ret_avg'] <= -0.004, (df_W['ret_avg'] > -0.004) & (df_W['ret_avg'] <= 0), df_W['ret_avg'] > 0 ]
choices = [ -1, 0, 1 ]

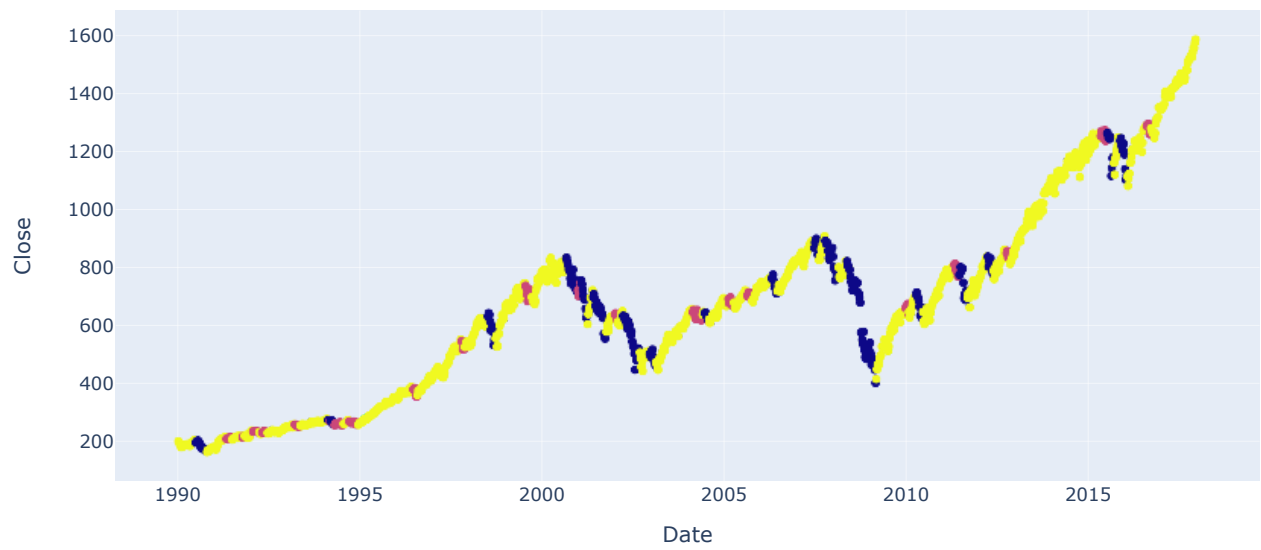
df_W['Indicator'] = np.select(conditions, choices, default=np.nan)
df_W['final_state'] = df_W['Indicator'].rolling(window=10).apply(lambda x: mode(x)[0]).shift(-9)
```

```
df_W_new = df_W[['final_state']]
df_W_new = df_W_new.ffill().bfill()
df_W_new
```

final_state

```
df_W_new[['final_state']].to_pickle('/data/workspace_files/rule_based.pickle')
df_W_new[['final_state']].to_pickle('/data/workspace_files/output/rule_based.pickle')
```

```
px.scatter(df_W[df_W.index<pd.to_datetime('2018-01-01')]['Close'],
          color=df_W_new[df_W_new.index<pd.to_datetime('2018-01-01')]['final_state'],
          ).update_layout(yaxis_title='Close').update_coloraxes(showscale=False)
```



```
# !pip install keras_tuner
# !pip install pydot
# !pip install graphviz
```

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.utils import np_utils
import tensorflow as tf
import keras_tuner as kt
```

```
np.random.seed(7)
```

```
x_data = pd.read_pickle('/data/workspace_files/data_df92.pickle')
y_data = pd.read_pickle('/data/workspace_files/rule_based.pickle').loc[x_data.index]
y_modified = pd.DataFrame(np_utils.to_categorical(y_data - y_data.min()), index=y_data.index)
```

```
from sklearn.preprocessing import StandardScaler
x_data = pd.DataFrame(StandardScaler().fit_transform(x_data),
                      columns=x_data.columns,
                      index=x_data.index)
```

```
y_modified.sum()
```

```
# load the dataset but only keep the top n words, zero the rest
train_ind = int(np.where(x_data.index >= pd.to_datetime('2018-01-01 00:00:00'))[0][0])
val_ind = int(train_ind*0.8)
X_train = x_data.iloc[:val_ind]
X_val = x_data.iloc[val_ind:train_ind]
X_test = x_data.iloc[train_ind:]

y_train = y_modified.iloc[:val_ind]
y_val = y_modified.iloc[val_ind:train_ind]
y_test = y_modified.iloc[train_ind:]
print("size of train data: ", X_train.shape[0],
      ", size of val data: ", X_val.shape[0],
      ", size of test data: ", X_test.shape[0])

X_train = tf.expand_dims(X_train, axis=1)
X_val = tf.expand_dims(X_val, axis=1)
X_test = tf.expand_dims(X_test, axis=1)
print(X_train.shape)
```

```
size of train data: 1080 , size of val data: 271 , size of test data: 210
(1080, 1, 38)
```

Hyperparameter Search

```
def build_model(hp):
    model = Sequential()
    model.add(LSTM(hp.Choice('units', [8, 16, 32, 64, 128]),
                      input_shape=(X_train.shape[1], X_train.shape[2]),
                      return_sequences=False))
    model.add(Dropout(hp.Choice('dropout', [0., 0.1, 0.2, 0.3, 0.4])))
    model.add(Dense(3, activation=hp.Choice(
        'dense_activation',
        values=['tanh', 'sigmoid'],
        default='sigmoid'),))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["CategoricalAccuracy"])
    return model
```

```
%capture
tuner = kt.RandomSearch(
    build_model,
    objective='categorical_accuracy',
    max_trials=100, overwrite=True)

tuner.search(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
best_model = tuner.get_best_models()[0]
```

INFO:tensorflow:Oracle triggered exit

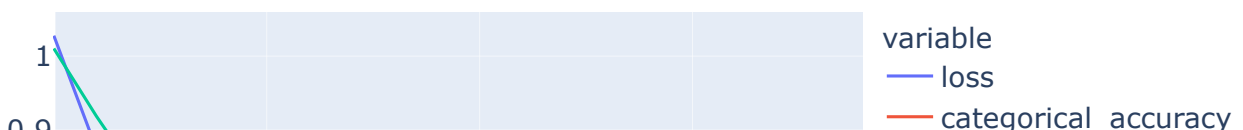
From the hyper parameter tuning, we found out the best architecture for our model is 128 LSTM nodes with 0.1 dropout and sigmoid activation function for the last dense layer.

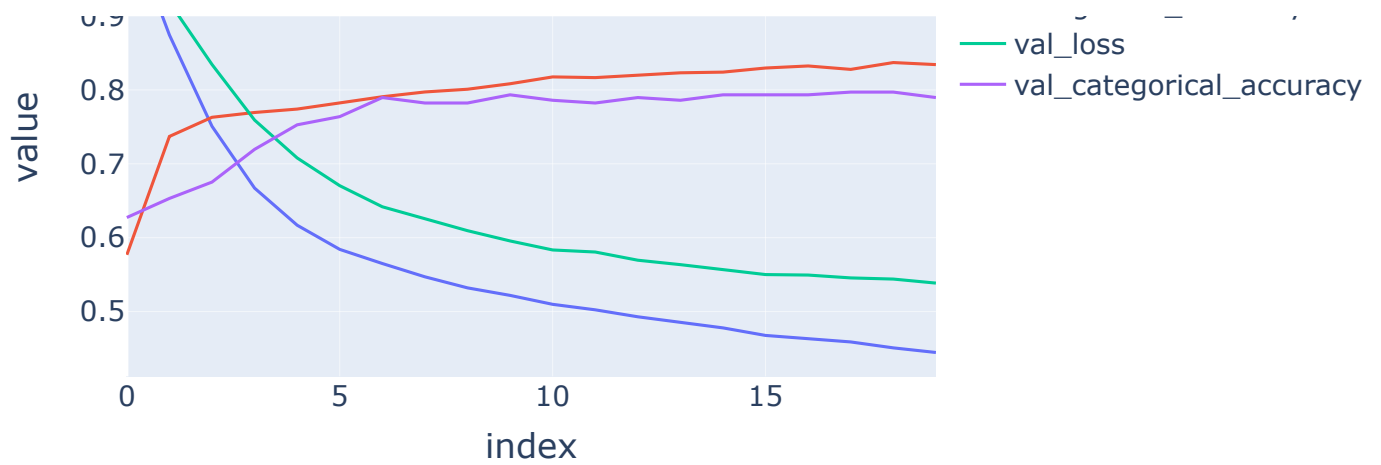
So, lets create the model, train and predict our output.

```
%capture
model = Sequential()

model.add(LSTM(128,
              input_shape=(X_train.shape[1], X_train.shape[2]),
              return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(3, activation='sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["CategoricalAccuracy"])
print(model.summary())
model_res = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=20,
                    batch_size=60)
```

```
import plotly.express as px
fig = px.line(pd.DataFrame.from_dict(model_res.history))
fig.update_layout(font=dict(size=18))
# fig.update_layout(legend=dict(y = 1.0, x = 0.8))
```





```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Accuracy: 64.76%

```
y_lstm = pd.DataFrame(
    pd.DataFrame(model.predict(X_test)).apply(lambda x:x.idxmax()-1, axis=1),
    columns=['lstm']
)
y_lstm.index = y_test.index
```

dump the output to the output folder, which will be used in the ensemble method!

```
pd.to_pickle(y_lstm, '/data/workspace_files/output/lstm_test.pickle')
pd.to_pickle(y_data, '/data/workspace_files/output/lstm_train.pickle')
```

Ensemble learning

```
import quandl
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats
from matplotlib import cm, pyplot as plt
from hmmlearn.hmm import GaussianHMM, GMMHMM
import scipy
import datetime
import json
import seaborn as sns
import joblib
import quandl
from dateutil.relativedelta import relativedelta

import yfinance as yf
import plotly.express as px
from statistics import mode
import pickle
import glob

import warnings
warnings.filterwarnings("ignore")
```

```

# Majority Voting

def majorityEnsemble(files=[], train_test_split_date='2018-01-01 00:00:00', states={'bull': 1, 'bear': -1, 'static': 0}, baseline='hmm',
    """
    files (list): list of pickle files to consider in 'output' folder;
    if this list is empty, read all pickle files in folder
    train_test_split_date (str): start date of test set
    """

    ##### SETUP FILE PATHS #####

    if len(files) == 0:
        raise Exception('No files provided')
    else:
        files.append(baseline)
        raw_files_train = ['/data/workspace_files/output/{}_train.pickle'.format(f.replace('.pickle', '')) for f in files]
        raw_files_test = ['/data/workspace_files/output/{}_test.pickle'.format(f.replace('.pickle', '')) for f in files]

    model_names = [f.split('/')[~1].replace('.pickle', '') for f in files]

    if len(raw_files_train) == 0 or len(raw_files_test) == 0:
        raise Exception('Could not find required files in output folder')

    ##### READ FILES INTO COMMON DF #####

    df_close = pd.read_pickle('/data/workspace_files/data_close.pickle')
    df_ret = pd.read_pickle('/data/workspace_files/data_fut_return.pickle')
    df_train, df_test = df_ret[df_ret.index < train_test_split_date], df_ret[df_ret.index >= train_test_split_date]

    for name, train_path, test_path in zip(model_names, raw_files_train, raw_files_test):
        print('Reading model file: ' + name)
        df_train_temp = pd.read_pickle(train_path)
        df_test_temp = pd.read_pickle(test_path)

        if len(df_train_temp.columns) != 1:
            raise Exception('Error reading train file: expected 1 column but loaded {} columns'.format(len(df_train_temp.columns)))
        if len(df_test_temp.columns) != 1:
            raise Exception('Error reading test file: expected 1 column but loaded {} columns'.format(len(df_test_temp.columns)))
        df_train_temp.columns = [name]
        df_test_temp.columns = [name]
        df_train = df_train.join(df_train_temp, how='left')
        df_test = df_test.join(df_test_temp, how='left')

    df_train = df_train.fillna(0)
    if df_test.isnull().values.any():
        raise Exception('Null values detected in test joined dataframe')

    def _ensemble(models, baseline):
        model_names = [m for m in models if m != baseline]
        def f(df):
            arr = [df[name] for name in model_names]
            # count = 0
            # for x in arr:
            #     if x == states['static']:
            #         count = count + 1
            # if np.sum(arr) == 0 or count==2:
            #     return 0

            return mode([df[name] for name in model_names])

        return f

    ##### SETUP RETURN ACTIONS; calculate returns using future_return column #####
    ret = []

    for df, lab in zip([df_train, df_test], ['train', 'test']):
        df['benchmark'] = states['bull']
        df['ensemble'] = df.apply(_ensemble(model_names, baseline), axis=1)
        df[['ensemble']].to_pickle('/data/workspace_files/output/ensemble_{}.pickle'.format(lab))
        model_names_full = ['benchmark', *model_names, 'ensemble']
        return_labels = ['benchmark' if name == 'benchmark' \
            else 'model_{}'.format(name) for name in model_names_full]

        df[model_names_full] = df[model_names_full].astype(int)

    for model, label in zip(model_names_full, return_labels):
        actions = np.select([
            (df[model] == states['bull']),
            (df[model] == states['bear']),

```

```

        (df[model] == states['static'])),
        [1, -1, 0]
    )
    df[label] = actions
    df[label] = 100 * ((df[label] * df['future_return']) + 1).cumprod()
    ret = ret + [df, return_labels]
    df_train, train_labels, df_test, test_labels = ret
    return df_train, train_labels, df_test, test_labels

df_train, train_labels, df, labels = majorityEnsemble(['lstm',
    'MSM',
    #'hmm_spca92.pickle',
    'hmm_pca92',
    #'hmm_baseline',
    #'hmm_last_return.pickle',
    #'hmm_spca99_test.pickle',
    #'hmm_spca11_test.pickle',
    ])

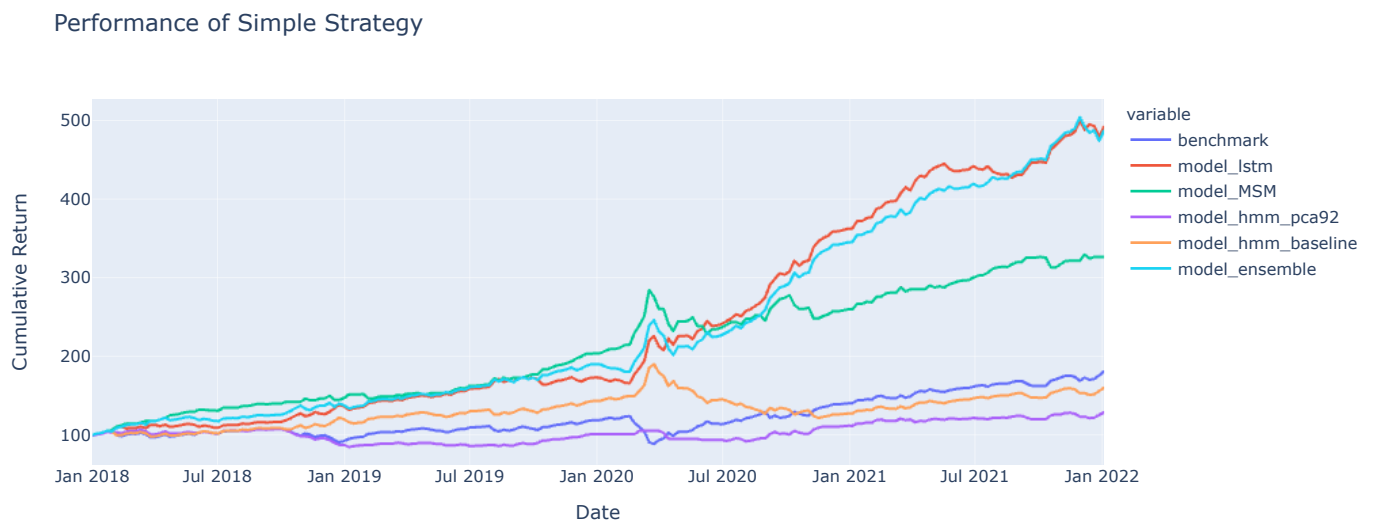
```

Reading model file: lstm
 Reading model file: MSM
 Reading model file: hmm_pca92
 Reading model file: hmm_baseline

```

px.line(df[labels], title='Performance of Simple Strategy', labels={'index': 'Date', 'value': 'Cumulative Return'})

```



```

rf_df = pd.read_pickle('/data/workspace_files/data_rf.pickle')
test_set_res = pd.merge_asof(df, rf_df, right_index=True, left_index=True)

df_summary = pd.DataFrame()
final = {}

for column in labels:
    df_returns = test_set_res[column].pct_change()
    s = {}
    year_count = relativedelta(test_set_res['RF'].index[-1], test_set_res['RF'].index[0]).years
    rf = (1 + test_set_res['RF']/365).prod() ** (1/ year_count) - 1
    s['Mean Return'] = np.exp(np.log(test_set_res[column]/test_set_res[column].shift(1)).mean() *52) - 1
    s['Volatility'] = df_returns.std() * np.sqrt(52)
    s['Skewness'] = df_returns.skew()
    s['Kurtosis'] = df_returns.kurtosis()
    s['Maximum Drawdown'] = (test_set_res[column]/test_set_res[column].cummax()-1).min()
    s['Sharpe Ratio'] = (s['Mean Return'] - rf) / s['Volatility']
    # s['Adjusted Sharpe Ratio'] = s['Sharpe Ratio']*(1+s['Skewness']/6*s['Sharpe Ratio']-(s['Kurtosis']-3)/24*s['Sharpe Ratio']**2)
    s['Semi-Deviation'] = df_returns[df_returns < df_returns.mean()].std()*np.sqrt(52)
    s['1-month Autocorrelation'] = df_returns.autocorr(lag=1)
    s['3-month Autocorrelation'] = df_returns.autocorr(lag=3)
    s['1-year Autocorrelation'] = df_returns.autocorr(lag=12)
    final[column] = s

df_summary = pd.DataFrame.from_dict(final, orient='index').round(4).T
# df_summary.to_csv('res.csv')

#cols = ['benchmark_return', 'hmm_baseline_return', 'lstm_return', 'MSM_return', 'hmm_pca92_return', 'ensemble_return']
#labels = ['Benchmark', 'Baseline', 'LSTM', 'MSM', 'HMM PCA', 'Ensemble']

cols = ['model_benchmark', 'model_lstm', 'model_MSM', 'model_hmm_pca92', 'model_hmm_baseline', 'model_ensemble']
labels = ['Benchmark', 'LSTM', 'MSM', 'HMM PCA', 'Baseline', 'Ensemble']

df_summary = df_summary.rename(columns={k:v for k,v in zip(cols, labels)})[labels]
df_summary

```

	Benchmark	LSTM	MSM	HMM PCA	Baseline	Ensemble
Mean Return	0.1577	0.4845	0.3406	0.0676	0.1236	0.4807
Volatility	0.1580	0.1491	0.1419	0.1139	0.1533	0.1490
Skewness	-1.3324	1.2587	0.7900	-0.1766	0.5905	0.8621
Kurtosis	7.5491	6.6545	10.4265	3.1473	7.1940	7.3056
Maximum Drawdown	-0.2860	-0.0799	-0.1962	-0.2124	-0.3600	-0.1795
Sharpe Ratio	0.9878	3.2386	2.3881	0.5792	0.7960	3.2145
Semi-Deviation	0.1393	0.0830	0.1032	0.0841	0.1092	0.0959
1-month Autocorrelation	0.1642	0.0243	0.0476	0.1467	0.1396	0.0995
3-month Autocorrelation	-0.0511	0.0355	0.0960	-0.0830	-0.1590	-0.0092
1-year Autocorrelation	-0.0287	0.0234	-0.0595	0.0863	0.0406	0.0600

```
print(df_summary.round(2).to_latex())
```

```

\begin{tabular}{lrrrrrr}
\toprule
{} & Benchmark & LSTM & MSM & HMM PCA & Baseline & Ensemble \\
\midrule
Mean Return & & 0.16 & 0.48 & 0.34 & 0.07 & 0.12 & 0.48 \\
Volatility & & 0.16 & 0.15 & 0.14 & 0.11 & 0.15 & 0.15 \\
Skewness & & -1.33 & 1.26 & 0.79 & -0.18 & 0.59 & 0.86 \\
Kurtosis & & 7.55 & 6.65 & 10.43 & 3.15 & 7.19 & 7.31 \\
Maximum Drawdown & & -0.29 & -0.08 & -0.20 & -0.21 & -0.36 & -0.18 \\
Sharpe Ratio & & 0.99 & 3.24 & 2.39 & 0.58 & 0.80 & 3.21 \\
Semi-Deviation & & 0.14 & 0.08 & 0.10 & 0.08 & 0.11 & 0.10 \\
1-month Autocorrelation & & 0.16 & 0.02 & 0.05 & 0.15 & 0.14 & 0.10 \\
3-month Autocorrelation & & -0.05 & 0.04 & 0.10 & -0.08 & -0.16 & -0.01 \\
1-year Autocorrelation & & -0.03 & 0.02 & -0.06 & 0.09 & 0.04 & 0.06 \\
\bottomrule
\end{tabular}

```

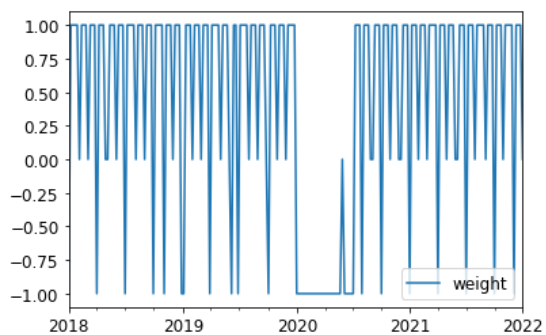
```
pd.read_pickle('/data/workspace_files/output/lstm_test.pickle')
```

	lstm
Date	
2018-01-01	1
2018-01-08	1
2018-01-15	1
2018-01-22	1
2018-01-29	1
...	...
2021-12-06	-1
2021-12-13	-1
2021-12-20	-1
2021-12-27	-1
2022-01-03	1

210 rows × 1 columns

```
pd.read_pickle('/data/workspace_files/output/hmm_scca99_test.pickle').plot()
```

<AxesSubplot:>



Trading Strategy with vol timing

```
vix_df = pd.read_pickle('/data/workspace_files/data_vix.pickle')
rf_df = pd.read_pickle('/data/workspace_files/data_rf.pickle')
close_df = pd.read_pickle('/data/workspace_files/data_close.pickle')
fut_return = pd.read_pickle('/data/workspace_files/data_fut_return.pickle')
```

```

def get_risk_params(train_df_w_signal, states_map, regime_list=['buy', 'neutral', 'sell']):
    """
    Function to get the risk aversion and risk premium in different regimes.

    Inputs:
        train_df_w_signal : DataFrame - Training period with columns ['signal', 'future_return'],
                                index as datetime
        states_map : Dict<{str : int}>> - Mapping of regimes to numerical signals
        regime_list : List<str> - List of regimes

    Output:
        risk_aversion : Dict<{str : float}>> - Value of absolute risk aversion coefficient
                                based on current regime
        risk_premium : Dict<{str : float}>> - Value of equity risk premium based on
                                current regime
    """
    df = pd.merge_asof(train_df_w_signal, vix_df / 100, right_index=True, left_index=True)
    df = pd.merge_asof(df, rf_df, right_index=True, left_index=True)
    df = pd.merge_asof(df, fut_return, right_index=True, left_index=True)
    risk_aversion = dict()
    risk_premium = dict()

    for regime in regime_list:
        regime_vix_mean = df[df.signal == states_map[regime]].VIX.mean()
        risk_aversion[regime] = 3 / df.VIX.mean() * regime_vix_mean

        # return mean has to be geometric mean!
        regime_return_mean = df[df.signal == states_map[regime]].future_return.mean()
        risk_premium[regime] = regime_return_mean*52 - df.RF.mean()
    return risk_aversion, risk_premium

def mean_variance_weight(vol, signal, risk_aversion, risk_premium, signal_map):
    """
    Function to get weight at a timestamp based on the regime.

    Inputs:
        vol : <float> - value of current volatility (annualized)
        signal : <int> - current signal value
        risk_aversion : Dict<str> - Value of absolute risk aversion coefficient based on
                                current regime
        risk_premium : Dict<{str : float}>> - Value of equity risk premium based on
                                current regime
        signal_map : Dict<{int : str}>> - Mapping of numerical signals to regimes

    Output:
        weight : <float> capped weight of asset allocation for the strategy
    """
    var = vol**2
    rpremia = (risk_premium[signal_map[signal]])
    r_av = 1 / risk_aversion[signal_map[signal]]
    weight = np.divide(r_av * rpremia, var)
    return weight

def get_strategy_weights(test_df_w_signal,
                        signal_map,
                        risk_aversion,
                        risk_premium,
                        strategy='vol-timing'):
    """
    Function to get the weight allocation series for the strategy.

    Inputs:
        test_df_w_signal : DataFrame - Test period with columns ['signal'], index as datetime
        signal_map : Dict<{int : str}>> - Mapping of numerical signals to regimes
        risk_aversion : Dict<str> - Value of absolute risk aversion coefficient based on
                                current regime
        risk_premium : Dict<{str : float}>> - Value of equity risk premium based on current regime
        strategy : <str> - Trading strategy, must be one out of ['vol-timing', 'full-allocation']

    Output:
        risk_aversion : Dict<{str : float}>> - Value of absolute risk aversion coefficient based on
                                current regime
        risk_premium : Dict<{str : float}>> - Value of equity risk premium based on current regime
    """
    df = pd.merge_asof(test_df_w_signal, vix_df / 100, right_index=True, left_index=True)
    weights = None
    if strategy=='full-allocation':
        weights = df.signal
    elif strategy=='vol-timing':

```

```

weights = [mean_variance_weight(vol, int(signal), risk_aversion, risk_premium, signal_map)
            for i, (vol, signal) in df[['VIX', 'signal']].iterrows()]

return np.minimum(np.maximum(weights, -1), 1)

```

```

### use this to convert states_map to signal_map
states_map = {
    'buy' : 1,
    'neutral' : 0,
    'sell' : -1
}
signal_map = {v: k for k, v in states_map.items()}
initial_capital = 100

# models = ['lstm', 'MSM', 'hmm_last_return', 'hmm_pca92', 'hmm_spc11', 'hmm_spc92', 'hmm_spc99', 'ensemble'] # Placeholder
models = ['lstm', 'MSM', 'hmm_pca92', 'ensemble', 'hmm_baseline'] # Placeholder

## ANY MODEL JUST TO GET TIME INDEX
test_df_w_signal = pd.read_pickle('/data/workspace_files/output/lstm_test.pickle')
test_set_res = pd.DataFrame(index=test_df_w_signal.index)
test_set_res = pd.merge_asof(test_set_res, fut_return, right_index=True, left_index=True)

for model in models:
    print('processing model {}'.format(model))
    train_df_w_signal = pd.read_pickle('/data/workspace_files/output/{}_train.pickle'.format(model))
    test_df_w_signal = pd.read_pickle('/data/workspace_files/output/{}_test.pickle'.format(model))
    # if model in ['MSM']:
    #     train_df_w_signal.drop(columns=['future_return'], inplace=True)
    #     test_df_w_signal.drop(columns=['future_return'], inplace=True)

    train_df_w_signal.columns=['signal']
    test_df_w_signal.columns=['signal']

    risk_aversion, risk_premium = get_risk_params(train_df_w_signal, states_map)
    #print(risk_aversion, risk_premium)
    test_df_w_signal['weight'] = get_strategy_weights(test_df_w_signal,
                                                    signal_map, risk_aversion, risk_premium,
                                                    )
    # strategy='full-allocation')
    # test_df_w_signal['weight'] = get_strategy_weights(test_df_w_signal,
    # signal_map, risk_aversion, risk_premium, strategy='full-allocation')

    # test_set_res = pd.DataFrame(index=test_df_w_signal.index)
    # test_set_res = pd.merge_asof(test_set_res, fut_return, right_index=True, left_index=True)
    test_set_res['model_{}'.format(model)] = 0
    test_set_res['model_{}'.format(model)].iloc[0] = initial_capital

    v_t = initial_capital
    ### Model Out-of-Sample PnL
    for i in range(test_set_res.shape[0]-1):
        v_t = v_t * (1 + test_set_res['future_return'].iloc[i] * test_df_w_signal['weight'].iloc[i])
        test_set_res['model_{}'.format(model)].iloc[i+1] = v_t

test_set_res['benchmark'] = 100 * (test_set_res['future_return'] + 1).cumprod()
test_set_res = test_set_res.drop(columns=['future_return'])
#px.line(test_set_res)
px.line(test_set_res, title='Performance of Vol-timing Strategy', labels={'value': 'Cumulative Return'})

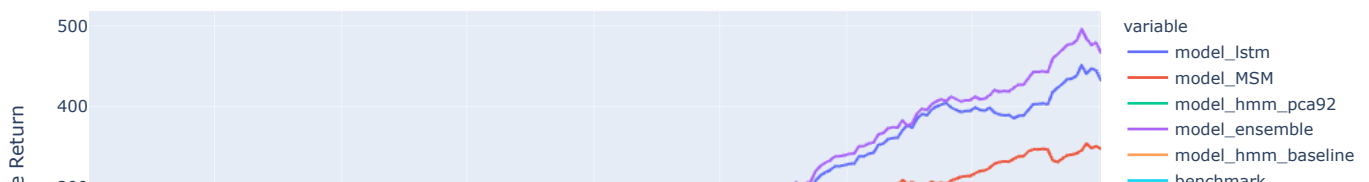
```

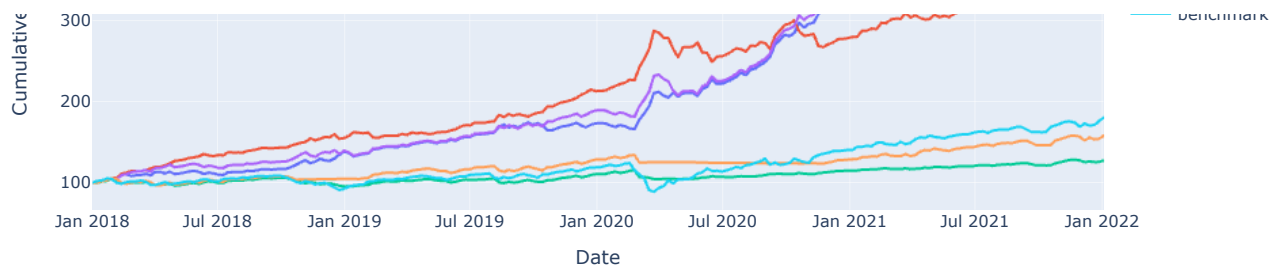
```

processing model lstm
processing model MSM
processing model hmm_pca92
processing model ensemble
processing model hmm_baseline

```

Performance of Vol-timing Strategy





```
test_set_res = pd.merge_asof(test_set_res, rf_df, right_index=True, left_index=True)
columns = ['model_{}'.format(model) for model in models] + ['benchmark']

df_summary = pd.DataFrame()
final = {}

for column in columns:
    df_returns = test_set_res[column].pct_change()
    s = {}
    year_count = relativedelta(test_set_res['RF'].index[-1], test_set_res['RF'].index[0]).years
    rf = (1 + test_set_res['RF']/365).prod() ** (1/ year_count) - 1
    s['Mean Return'] = np.exp(np.log(test_set_res[column]/test_set_res[column].shift(1)).mean() *52) - 1
    s['Volatility'] = df_returns.std() * np.sqrt(52)
    s['Skewness'] = df_returns.skew()
    s['Kurtosis'] = df_returns.kurtosis()
    s['Maximum Drawdown'] = (test_set_res[column]/test_set_res[column].cummax()-1).min()
    s['Sharpe Ratio'] = (s['Mean Return'] - rf) / s['Volatility']
    s['Semi-Deviation'] = df_returns[df_returns < df_returns.mean()].std()*np.sqrt(52)
    s['1-month Autocorrelation'] = df_returns.autocorr(lag=1)
    s['3-month Autocorrelation'] = df_returns.autocorr(lag=3)
    s['1-year Autocorrelation'] = df_returns.autocorr(lag=12)
    s
    final[column] = s

df_summary = pd.DataFrame.from_dict(final, orient='index').round(4).T
# df_summary.to_csv('res.csv')
# models = ['lstm', 'MSM', 'hmm_pca92', 'ensemble', 'hmm_baseline'] # Placeholder
cols = ['benchmark', 'model_hmm_baseline', 'model_lstm', 'model_MSM', 'model_hmm_pca92', 'model_ensemble']
labels = ['Benchmark', 'Baseline', 'LSTM', 'MSM', 'HMM PCA', 'Ensemble']

df_summary = df_summary.rename(columns={k:v for k,v in zip(cols, labels)})[labels]
df_summary
```

	Benchmark	Baseline	LSTM	MSM	HMM PCA	Ensemble
Mean Return	0.1577	0.1221	0.4396	0.3640	0.0630	0.4667
Volatility	0.1580	0.0957	0.1284	0.1280	0.0819	0.1319
Skewness	-1.3324	-1.1279	0.6635	0.1646	-1.9465	0.5757
Kurtosis	7.5491	5.4747	1.8391	4.0882	9.0980	2.6945
Maximum Drawdown	-0.2860	-0.0817	-0.0542	-0.1344	-0.1015	-0.1104
Sharpe Ratio	0.9878	1.2586	3.4105	2.8306	0.7489	3.5253
Semi-Deviation	0.1393	0.0799	0.0720	0.0940	0.0772	0.0802
1-month Autocorrelation	0.1642	0.0677	0.0566	0.0443	0.1117	0.0440
3-month Autocorrelation	-0.0511	-0.0555	0.0369	0.0443	-0.0147	0.0194
1-year Autocorrelation	-0.0287	0.0627	0.0097	-0.0616	0.0069	0.0454

```
print(df_summary.round(2).to_latex())
```

```
\begin{tabular}{lrrrrrrr}
\toprule
{} & Benchmark & Baseline & LSTM & MSM & HMM PCA & Ensemble \\
\midrule
Mean Return & & 0.16 & & 0.12 & 0.44 & 0.36 & 0.06 & 0.47 \\
Volatility & & 0.16 & & 0.10 & 0.13 & 0.13 & 0.08 & 0.13 \\
Skewness & & -1.33 & & -1.13 & 0.66 & 0.16 & -1.95 & 0.58 \\
Kurtosis & & 7.55 & & 5.47 & 1.84 & 4.09 & 9.10 & 2.69 \\
Maximum Drawdown & & -0.29 & & -0.08 & -0.05 & -0.13 & -0.10 & -0.11
\end{tabular}
```

Sharpe Ratio	&	0.99	&	1.26	&	3.41	&	2.83	&	0.75	&	3.53	\\
Semi-Deviation	&	0.14	&	0.08	&	0.07	&	0.09	&	0.08	&	0.08	\\
1-month Autocorrelation	&	0.16	&	0.07	&	0.06	&	0.04	&	0.11	&	0.04	\\
3-month Autocorrelation	&	-0.05	&	-0.06	&	0.04	&	0.04	&	-0.01	&	0.02	\\
1-year Autocorrelation	&	-0.03	&	0.06	&	0.01	&	-0.06	&	0.01	&	0.05	\\

\bottomrule
\end{tabular}

PLOT

```
def plot_hidden_states(hidden_states, close_series, fut_ret_series):
    matplotlib.rcParams.update({'font.size': 12})
    plt.figure(figsize=(15, 15))
    fig, axs = plt.subplots(3, 2, figsize = (15, 15))
    colours = cm.prism(np.linspace(0, 0.5, 3))

    ret_list = []
    for i, (ax, colour) in enumerate(zip(axs, ['red', 'blue', 'green'])):
        mask = hidden_states[hidden_states.iloc[:, 0] == i-1].index
        ax[0].plot(close_series.index, close_series, c = 'grey')
        ax[0].plot(close_series.loc[mask].index, close_series.loc[mask], '.', c = colour)
        ax[0].set_title("{}th hidden state".format(i), fontsize = 18)
        ax[0].grid(True)

        cum_return = (fut_ret_series.loc[mask] + 1).cumprod()
        ax[1].plot(cum_return, c = colour)
        ax[1].set_title("cummulative future return at {}th hidden state".format(i), fontsize = 18)
        ax[1].grid(True)

    ret_list.append(cum_return.iloc[-1].squeeze())
    plt.tight_layout()

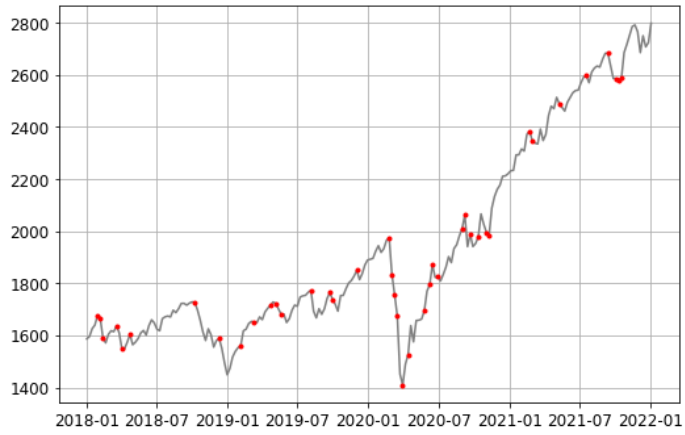
    fig.savefig('/data/workspace_files/classification.png')
    return ret_list
```

```
hidden_states = pd.read_pickle('/data/workspace_files/output/MSM_test.pickle')
df_ret = pd.read_pickle('/data/workspace_files/data_fut_return.pickle').loc[hidden_states.index]
df_close = pd.read_pickle('/data/workspace_files/data_close.pickle').loc[hidden_states.index]
plot_hidden_states(hidden_states, df_close, df_ret)
```

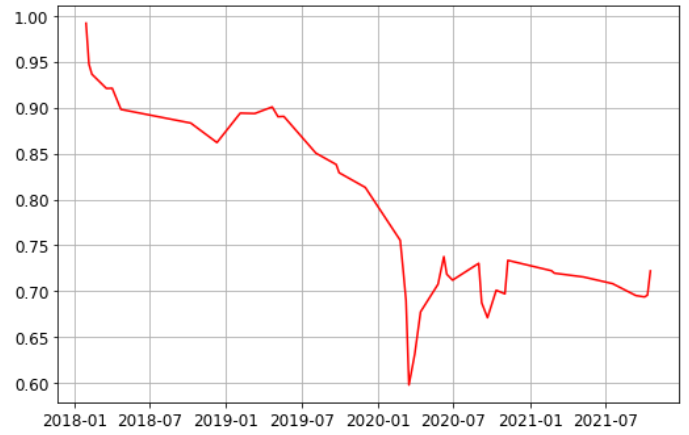
```
[0.7220731035365416, 1.0017437592385006, 2.50423033420589]
```

```
<Figure size 1080x1080 with 0 Axes>
```

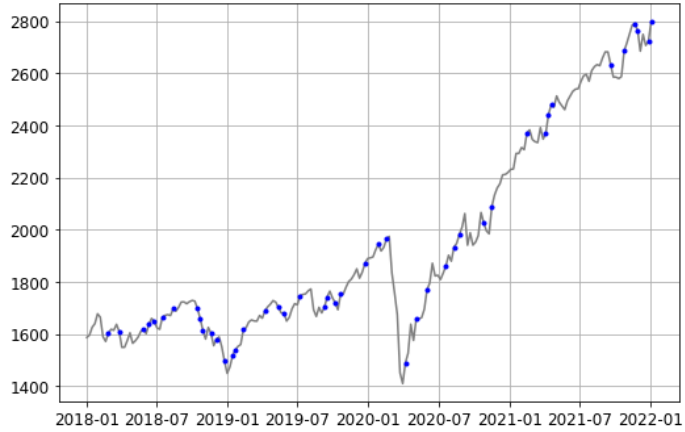
0th hidden state



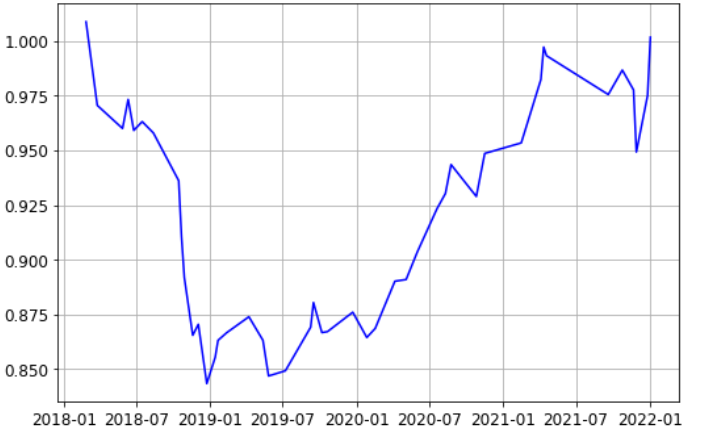
cumulative future return at 0th hidden state



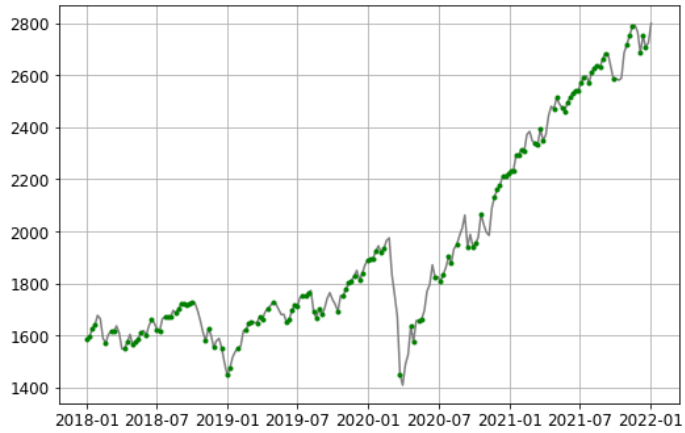
1th hidden state



cumulative future return at 1th hidden state



2th hidden state



cumulative future return at 2th hidden state

