```
import pandas as pd
from datetime import datetime
import tradelib
import numpy as np
import peakutils
from matplotlib import style
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn import preprocessing, cross_validation, svm, naive_bayes, linear_model, ensemble
from sklearn.metrics import f1_score, precision_recall_curve
import random
import time
import random
import copy

class MarketSimulator:

    def __init__(self, start_date, end_date, stock_list, cash=10000, pct_per_trade=0.2, broker_fee=0,
    pre_load_date=None, stop_loss_pct=0.9, profit_take_pct=1.8, classifier=None, fe_scale=[],
    feature_set=['DR1'],  expiry_days=30, ban_days=30, bump_expiry=True, every_second=False,
    hard_stop_loss=False,use_probability=False, probability=0.5, scaler=None):
        print("Loading data...")
        if pre_load_date==None:
            pre_load_date = start_date
        self.hist_data = tradelib.HistoricalData(pre_load_date,end_date)
        self.hist_data.set_db_directory('pickles/')
        self.hist_data.pull_list_db(stock_list)
        self.hist_data.clean_data()


        index_list = ['GSPC']

        self.hd_ind = tradelib.HistoricalData(pre_load_date,end_date)
        self.hd_ind.set_db_directory('pickles/')
        self.hd_ind.pull_list_db_yahoo('index', index_list)

        self.analytics_price = self.hist_data.adjusted_close
        self.analytics_high = self.hist_data.adjusted_high
        self.analytics_low = self.hist_data.adjusted_low
        self.analytics_open = self.hist_data.adjusted_open
        self.analytics_volume = self.hist_data.adjusted_volume

        self.simulator_price = self.hist_data.price_close
        self.simulator_split = self.hist_data.split_ratio
        self.simulator_low = self.hist_data.price_low

        self.ban_days = ban_days
        self.bump_expiry = bump_expiry
        self.every_second = every_second
        self.probability = probability
        self.use_probability = use_probability

        self.scaler = scaler

        if fe_scale==None:
            self.feature_ext = tradelib.FeatureExtraction(self.analytics_price, scale=False,
            fe_type='ohlc',df_high=self.analytics_high,df_low=self.analytics_low,df_open=self.analytics_op
            en,df_volume=self.analytics_volume, df_market=hd_ind.adjusted_close, market_index='GSPC')
        else:
            self.feature_ext = tradelib.FeatureExtraction(self.analytics_price, scale=False,
            mean=fe_scale[0], std=fe_scale[1],
            fe_type='ohlc',df_high=self.analytics_high,df_low=self.analytics_low,df_open=self.analytics_op
            en,df_volume=self.analytics_volume, df_market=self.hd_ind.adjusted_close, market_index='GSPC')
        nan_list = self.feature_ext.extract_features()
        for nan_stock in nan_list:
            stock_list.remove(nan_stock)

        self.feature_ext.prescale_features()

        if classifier == None:
```

```
            self.use_classifier = False
        else:
            self.use_classifier = True
            self.classifier = classifier
            self.feature_set = feature_set
            print(self.feature_set)

        self.date_list = self.analytics_price.index
        index = 0
        self.start_index = 0
        self.stop_loss_pct = stop_loss_pct
        for day in self.date_list:
            if day >= start_date:
                self.start_index = index
                break
            index += 1
        self.cash = cash
        self.pct_per_trade = pct_per_trade
        self.broker_fee = broker_fee
        self.open_positions = pd.DataFrame(columns=['Stock','Amount','Purchase Amount', 'Stop
        Loss','Profit Take','Expiry Date', 'Do Not Buy'])
        self.position = cash
        self.stock_list = stock_list
        self.trade_log = pd.DataFrame(columns = ['Date', 'Stock', 'Num. Shares', 'Transaction', 'Price',
        'Amount', 'Reason', 'Profit', 'Cash'])
        self.ban_list = pd.DataFrame(columns = ['Stock', 'Expiry Date'])
        self.trade_counter = 0
        self.position_counter = 0
        self.position_log_counter = 0
        self.position_log = pd.DataFrame(columns=['Position'])
        self.profit_take_pct = profit_take_pct
        self.expiry_days = expiry_days
        self.hard_stop_loss = hard_stop_loss



    def simulate(self):



        for day_index in range(self.start_index, len(self.date_list)):
            self.process_day(day_index)

        self.trade_log.to_csv('trade_log.csv')
        print(self.trade_log)
        self.position_log.to_csv('position_log.csv')
        print(self.position_log)
        print(self.open_positions)
        print(self.cash)

    def process_day(self, day_index):
        print('Day:',self.date_list[day_index])
        self.daily_features = self.feature_ext.get_list_features(self.feature_set,
        day_index,self.stock_list)


        for index, position in self.ban_list.iterrows():
            if position['Expiry Date'] == day_index:
                self.ban_list.drop(index, inplace=True)



        for index, position in self.open_positions.iterrows():
            if self.hard_stop_loss:
                stop_loss = (self.simulator_low[position['Stock']][day_index] < position['Stop Loss'])
            else:
                stop_loss = (self.simulator_price[position['Stock']][day_index] < position['Stop Loss'])

            if stop_loss or position['Expiry Date'] <= day_index or day_index == len(self.date_list)-1 or
            self.simulator_price[position['Stock']][day_index] > position['Profit Take']:
```

```
            print("Selling...")
            if stop_loss and self.hard_stop_loss:
                sale_amount = position['Stop Loss']*position['Amount']
            else:
                sale_amount = self.simulator_price[position['Stock']][day_index]*position['Amount']

            if stop_loss and self.ban_days > 0:
                temp = pd.Series({'Stock': position['Stock'], 'Expiry Date':
                (day_index+self.ban_days)})
                self.ban_list = self.ban_list.append(temp, ignore_index=True)



            self.cash += sale_amount
            self.cash -= self.broker_fee
            if position['Expiry Date'] <= day_index:
                reason = 'Expiry'
            elif stop_loss:
                reason = 'Stop Loss'
            elif self.simulator_price[position['Stock']][day_index] > position['Profit Take']:
                reason = 'Profit Take'
            else:
                reason = 'EOS'


            profit = (sale_amount-2*self.broker_fee)/position['Purchase Amount']
            temp = pd.Series({'Date':self.date_list[day_index], 'Stock':position['Stock'], 'Num.
            Shares':position['Amount'], 'Transaction':'Sell',
            'Price':self.simulator_price[position['Stock']][day_index], 'Amount':sale_amount,
            'Reason':reason, 'Profit':profit, 'Cash':self.cash })
            self.trade_log = self.trade_log.append(temp, ignore_index=True)
            self.trade_counter += 1
            self.open_positions.drop(index, inplace=True)

    if day_index < len(self.date_list)-1:
        df_predict = self.group_buy_signal(self.stock_list, day_index)
        i = 0
        for stock in df_predict['Pre']:
            if stock < self.probability:
                break;
            i += 1
        buy_list = list(df_predict.index[:i])
        print("Number of buy options:",i)

        #for stock in self.stock_list:
        if len(buy_list) > 0:
            if self.use_probability:
                pass
            else:
                random.shuffle(buy_list)
            for index, position in self.open_positions.iterrows():
                if position['Stock'] in buy_list:
                    if position['Do Not Buy']:
                        buy_list.remove(position['Stock'])
                        if self.every_second:
                            position['Do Not Buy'] = False
                        if self.bump_expiry:
                            position['Expiry Date'] = day_index+self.expiry_days
            for index, position in self.ban_list.iterrows():
                if position['Stock'] in buy_list:
                    buy_list.remove(position['Stock'])
            for stock in buy_list:
                if self.cash > self.position*self.pct_per_trade+self.broker_fee:
                    print("Buying...")
                    current_price = self.simulator_price[stock][day_index]
                    amount = int(self.position*self.pct_per_trade/current_price)
                    buy_amount = amount*current_price
                    self.cash -= buy_amount
                    self.cash -= self.broker_fee
                    if self.bump_expiry or self.every_second:
```

```
                        dnb = True
                    else:
                        dnb = False
                    temp = pd.Series({'Date':self.date_list[day_index], 'Stock':stock, 'Num.
Shares':amount, 'Transaction':'Buy', 'Price':self.simulator_price[stock]
[day_index], 'Amount':buy_amount, 'Reason':'N/A', 'Profit': 1.00, 'Cash':self.cash
})
                    self.trade_log = self.trade_log.append(temp, ignore_index=True)
                    self.trade_counter += 1
                    # Open position
                    expiry_date_index = day_index+self.expiry_days
                    stop_loss = self.stop_loss_pct*current_price
                    profit_take = self.profit_take_pct*current_price

                    temp = pd.Series({'Stock':stock, 'Amount':amount, 'Purchase Amount': buy_amount,
'Stop Loss':stop_loss, 'Profit Take':profit_take,'Expiry Date':expiry_date_index,
'Do Not Buy': dnb})
                    self.open_positions = self.open_positions.append(temp, ignore_index=True)
                    self.position_counter += 1


        self.calculate_position(day_index)

    def group_buy_signal(self, stock_list, day_index):
        if day_index+self.expiry_days < len(self.date_list):
            df_split = self.hist_data.split_ratio[stock_list]
            [day_index:day_index+self.expiry_days].cumprod().iloc[[-1]]
            if np.array(df_split).max() > 1.0 or np.array(df_split).min() < 1.0:
                for stock in stock_list:
                    if df_split[stock][-1] != 1.0:
                        print('Split warning', stock, 'on', self.date_list[day_index])
                        stock_list.remove(stock)
        else:
            df_split = self.hist_data.split_ratio[stock_list][day_index:-1].cumprod().iloc[[-1]]
            if np.array(df_split).max() > 1.0 or np.array(df_split).min() < 1.0:
                for stock in stock_list:
                    if df_split[stock][-1] != 1.0:
                        print('Split warning', stock, 'on', self.date_list[day_index])
                        stock_list.remove(stock)
        df_X = copy.deepcopy(self.daily_features[stock_list])

#         X = np.array(self.daily_features[stock])#self.feature_ext.get_features(self.feature_set,
day_index, stock)
#         X_vol =
np.array(self.daily_vol_features[stock])#self.feature_ext_vol.get_features(self.feature_vol_set,
day_index, stock)
#         X = np.concatenate((X, X_vol))

        df_X.replace([np.inf, -np.inf], np.nan, inplace=True)
        df_X.dropna(axis=1, inplace=True)
        stock_list_reduced = df_X.columns
        other_stocks = list(set(stock_list)-set(stock_list_reduced))
        X_pred = df_X.as_matrix().T


        X_pred = self.scaler.transform(X_pred)
        if self.use_probability:
            predict = self.classifier.predict_proba(X_pred)[:,1]
        else:
            predict = self.classifier.predict(X_pred)

        rest = np.zeros((len(other_stocks),))
        predict = np.concatenate([predict, rest])

        #df_predict = pd.DataFrame([predict], columns=(list(stock_list_reduced)+list(other_stocks)))
        df_predict = pd.DataFrame(predict, index=(list(stock_list_reduced)+list(other_stocks)),
        columns=['Pre'])

        df_s = df_predict.sort_values(by=('Pre'),ascending=False)
```

```
        return df_s

    def buy_signal(self, stock, day_index):
        if day_index+self.expiry_days < len(self.date_list):
            if self.hist_data.split_ratio[stock][day_index:day_index+self.expiry_days].cumprod()[-1] != 1:
                #print('Split warning', stock, 'on', self.date_list[day_index])
                return False
        else:
            if self.hist_data.split_ratio[stock][day_index:-1].cumprod()[-1] != 1:
                #print('Split warning', stock, 'on', self.date_list[day_index])
                return False

        if self.use_classifier:
            X = np.array(self.daily_features[stock])#self.feature_ext.get_features(self.feature_set,
day_index, stock)

            X = self.scaler.transform(X)

            meanx = X.mean()
            if meanx < 1000 and meanx > -1000:
                try:
                    predict = self.classifier.predict(X.reshape(1,-1))[0]
                except Exception as e:
                    print(str(e), X)
                    predict = 0
            else:
                predict = 0

            if predict == 1:
                return True
            else:
                return False
        else:
            if np.random.randint(365)*len(self.stock_list) < 100:
                return True
            else:
                return False

    def calculate_position(self,day_index):
        self.position = self.cash
        for position in self.open_positions.itertuples():
            self.position += self.simulator_price[position[1]][day_index]*position[2]

        temp = pd.Series({'Position':self.position}, name=str(self.date_list[day_index]))
        self.position_log = self.position_log.append(temp)

        self.position_log_counter += 1
```