The code defines a Fuzzy Predictor class which takes as an argument ticker of an asset and uses fuzzy logic to predict the price of an asset.

The logic assumes that the prices of the asset depends of the previous set of intervals represented by Fuzzy_data.

Matrix R contains transition probobilities from one Fuzzy state to another.

In [1]:

```
import yfinance as yahooFinance
import datetime
import requests
from bs4 import BeautifulSoup as bs
import re
import numpy as np
import pandas as pd
from sklearn.neighbors import KernelDensity
import matplotlib.pylab as plt
from scipy.optimize import minimize
from scipy.linalg import block_diag
from sklearn.covariance import LedoitWolf

import pandas as pd
import numpy as np
from tqdm.notebook import tqdm
from sklearn.metrics import mean_squared_error as mse
import matplotlib.pyplot as plt
import seaborn as sns

from math import pi
import bokeh
from bokeh.plotting import figure, show

from bokeh.io import output_notebook
from bokeh.resources import INLINE
output_notebook(INLINE)




%matplotlib inline

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

BokehJS 3.0.2 successfully loaded.

In [2]:

```
class FuzzyPredictor:

    def __init__(self, ticker, start_date, train_size_days = 1600, shape_n = 20, n = 10):
        self.name = ticker
        self.ticker = yahooFinance.Ticker(ticker)
        self.startDate = start_date
        self.all_data = pd.DataFrame(self.ticker.history(start=self.startDate))
        self.all_data.index =  pd.to_datetime(self.all_data.index, format='%d %b %Y')
        self.data_to_train = self.all_data[:train_size_days]

        self.shape_n = shape_n
        self.train_size_days = train_size_days
        self.R = np.zeros((shape_n, shape_n))
        self.mu = []
        self.n = n

        for j in range(0, shape_n):
            self.mu.append(self.n*j + self.n/2)

        self.actual_predict_data = pd.DataFrame({'actual_price': [self.data_to_train.Close[-1]] , 'predicted_price': [self.data_to_train.Close[-1]] , "Fuzzy_data": ['A'
        self.actual_predict_data = self.actual_predict_data.set_index("Date")
        self.last_date_data = self.actual_predict_data.iloc[-1]

    def create_df_predict(self):
        for i in range(self.all_data.shape[0] - self.data_to_train.shape[0]-1):
            self.update_actual_predict_data()
            self.last_date_data = self.actual_predict_data.iloc[-1]
            self.update_R()

        return self.actual_predict_data

    def update_actual_predict_data(self):
        predicted_price = self.predict_calc()

        real_price = self.all_data.iloc[self.all_data.index.get_loc(self.actual_predict_data.index[-1]) + 1].Close

        delta_days = self.all_data.index[self.all_data.index.get_loc(self.actual_predict_data.index[-1])
                               + 1]- self.all_data.index[self.all_data.index.get_loc(self.actual_predict_data.index[-1])]

        self.actual_predict_data.loc[self.actual_predict_data.index[-1] + delta_days] = {
            "actual_price": real_price,
            "predicted_price": predicted_price,
            "Fuzzy_data": 'A' + str(int(real_price//10)),
            "Fuzzy_data_change": self.actual_predict_data.Fuzzy_data[-1] + 'A' + str(int(real_price//10))}


    def predict_calc(self):
        s = 0
        k = int(self.last_date_data.Fuzzy_data[1:])
        for i in range(len(self.mu)):
            if i == k:
                s += self.R[k][i] * self.last_date_data.actual_price
            else:
                s += self.R[k][i] * self.mu[i]
        return s/self.R[k].sum()

    def get_for_matrix_data(self):
        bins = np.arange(0, self.shape_n*(self.n), self.n)
        self.data_to_train['Fuzzy_data'] = np.nan
        self.data_to_train['volume_of_interval'] = np.nan
        self.data_to_train['medium_of_interval'] = np.nan

        for i in range(self.data_to_train.shape[0]):
            for j in range(2,len(bins)):
```

```python
                if self.data_to_train['Close'][i] <= self.n*(j+1) and self.data_to_train['Close'][i] > self.n*j:
                    self.data_to_train['Fuzzy_data'][i] = 'A'+str(j)
                    self.data_to_train['medium_of_interval'][i] = self.n*j + self.n/2

        self.data_to_train['Fuzzy_data_change'] = self.data_to_train['Fuzzy_data'].shift(1) + '_' + self.data_to_train['Fuzzy_data']
        for_matrix_data = self.data_to_train.Close.groupby(self.data_to_train.Fuzzy_data_change).count()

        return self.data_to_train, for_matrix_data


    def get_R_matrix(self, for_matrix_data):
        for ind in for_matrix_data.index:
            for i in range(1, self.shape_n):
                for j in range(1, self.shape_n):
                    if ind.startswith('A'+str(i)+'_') and ind.endswith('A'+str(j)):
                        self.R[i][j] = for_matrix_data[ind]
        return self.R

    def train_get_R(self):
        df_data_to_look, for_matrix_data = self.get_for_matrix_data()
        self.R = self.get_R_matrix(for_matrix_data)
        return self.R

    def normalize_matrix(self):
        for i in range(self.shape_n):
            if self.R[i].sum() != 0:
                self.R[i] = self.R[i]/self.R[i].sum()
        return self.R

    def update_R(self, prnt = False):
        last_date_data = self.last_date_data

        k = int(self.last_date_data.Fuzzy_data_change[1:3])
        j = int(self.last_date_data.Fuzzy_data[1:])

        if prnt == True:
            print('R[',k,'][',j,'] was: ', self.R[k][j])

        self.R[k][j] += 1

        if prnt == True:
            print('R[',k,'][',j,'] become: ', self.R[k][j])
            print("Fuzzy state was: A", k , ' -  become: A', j)


    def get_data(self):
        return self.all_data

    def plot_price(self, grid=True, figsize=(14, 9)):
        self.all_data.Open.plot(grid=grid, figsize=figsize)
        plt.title(self.name)
        plt.xlabel("Date")
        plt.ylabel("Price, US$")

    def plot_bokeh(self, name_bokeh, start_date = None, days_long = 100, vol=True, pred_on=False, pted_df=[0]):
        from math import pi
        if start_date == None:
            start_date = self.startDate

        result_date = start_date + datetime.timedelta(days=days_long)

        inc = self.all_data[start_date : result_date]['Close'] > self.all_data[start_date : result_date]['Open']
        dec = self.all_data[start_date : result_date]['Close'] < self.all_data[start_date : result_date]['Open']

        w = 12 * 60 * 60 * 1000  # half day in ms

        TOOLS = "pan,wheel_zoom,box_zoom,reset,save"

        p = figure(x_axis_type="datetime", tools=TOOLS, width=950, title=name_bokeh)
        p.xaxis.major_label_orientation = pi/4
        p.grid.grid_line_alpha = 0.3

        p.segment(self.all_data[start_date : result_date].index, self.all_data[start_date : result_date]['High'], self.all_data[start_date : result_date].index, self.al
        p.vbar(self.all_data[start_date : result_date].index[inc], w, self.all_data[start_date : result_date]['Open'][inc], self.all_data[start_date : result_date]['Clo
               line_color="black")
        p.vbar(self.all_data[start_date : result_date].index[dec], w, self.all_data[start_date : result_date]['Open'][dec], self.all_data[start_date : result_date]['Clo
               line_color="black")
        p.title.text_font_size = "25px"
        p.title.align = "center"
        p.yaxis.axis_label = 'price, US$'

        if pred_on == True:
            p.line(self.actual_predict_data[start_date : result_date].index, self.actual_predict_data.predicted_price[start_date : result_date], line_width=2, color='or


        if vol == True:
            p2 = figure(x_axis_type="datetime", tools="", toolbar_location=None, width=950, height=200)
            p2.xaxis.major_label_orientation = pi/4
            p2.grid.grid_line_alpha=0.3
            p2.vbar(self.all_data[start_date : result_date].index, w, self.all_data[start_date : result_date].Volume, [0]*self.all_data[start_date : result_date].shape[
            p2.xaxis.axis_label = 'Date'
            p2.yaxis.axis_label = 'Volume, US$'

            output_notebook()
            show(p)
            show(p2)
        else:
            show(p)
```

Создаем обьект класса FuzzyPredictor

In [3]:

```
apple = FuzzyPredictor("AAPL", datetime.datetime(2016, 1, 1))
```

Тренируем матрицу переходов R

Создаем датасет предсказанных значений

In [4]:

```
apple.train_get_R()
APP_pred = apple.create_df_predict()
APP_pred
```

Out[4]:

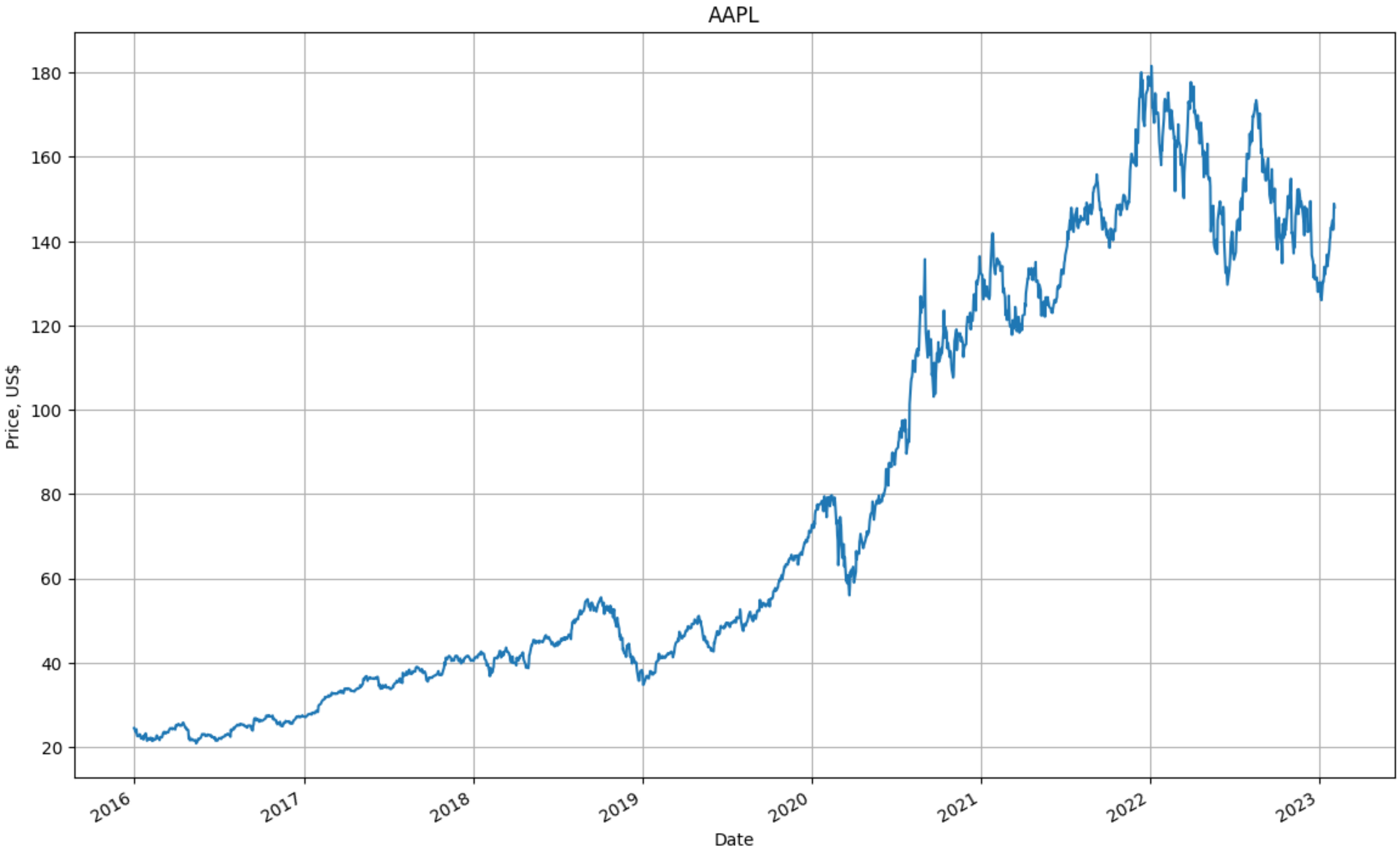
Date	actual_price	predicted_price	Fuzzy_data	Fuzzy_data_change
2022-05-09	151.597595	151.597595	A15	A15A14
2022-05-10	154.040131	153.476308	A15	A15A15
2022-05-11	146.054504	154.943865	A14	A15A14
2022-05-12	142.126480	146.405159	A14	A14A14
2022-05-13	146.662643	142.935130	A14	A14A14
...	...	...	...	...
2023-01-27	145.929993	144.210532	A14	A14A14
2023-01-30	143.000000	145.831236	A14	A14A14
2023-01-31	144.289993	143.415584	A14	A14A14
2023-02-01	145.429993	144.478188	A14	A14A14
2023-02-02	150.820007	145.419673	A15	A14A15

186 rows × 4 columns

У класса есть метод построения графиков

In [5]:

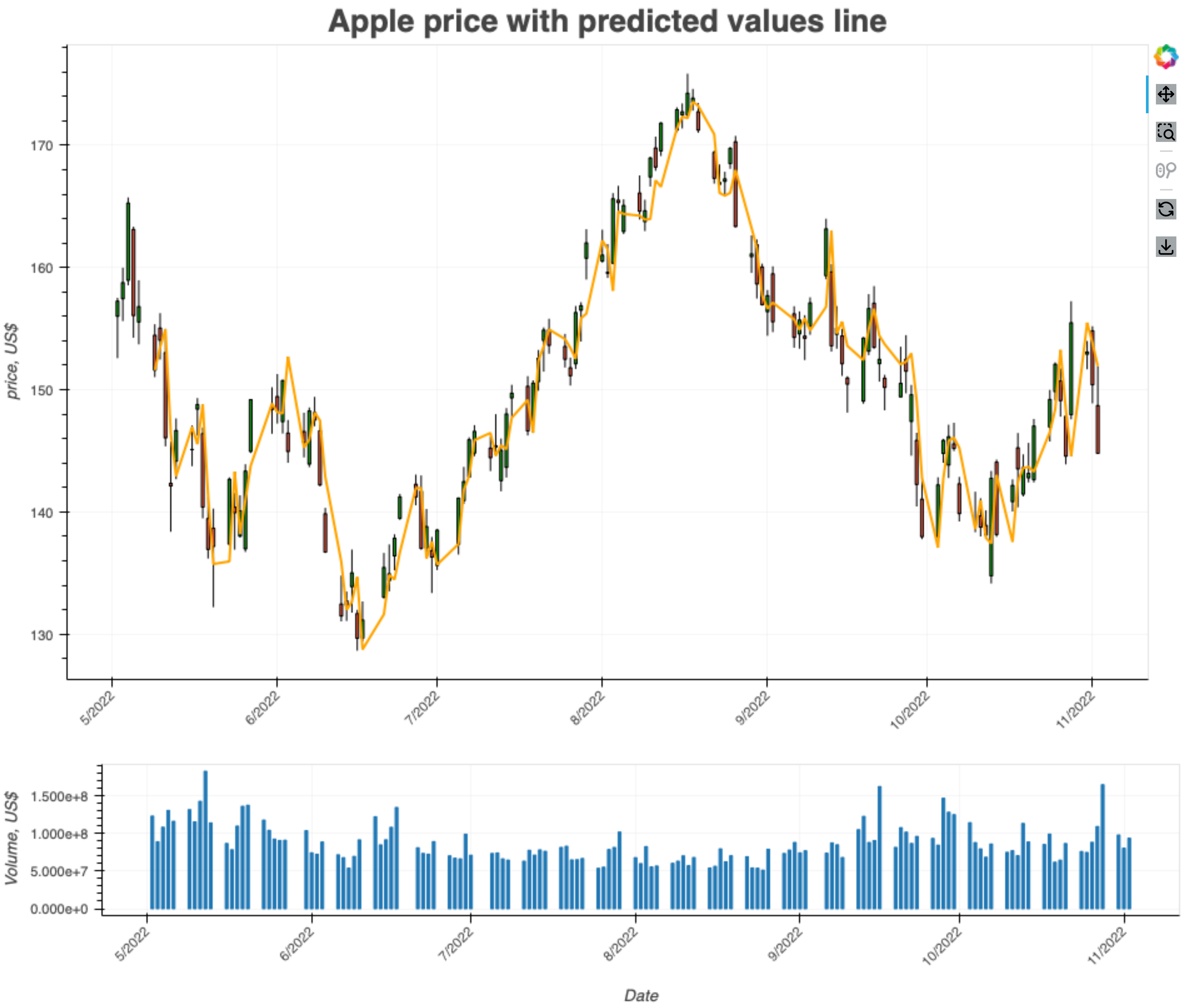
```
apple.plot_price()
```



И метод построения графиков свечей; линия - предсказанные значения - убирается/вставляется опционально с помощью параметра pred_on=True/False

```
In [6]: apple.plot_bokeh(name_bokeh = "Apple price with predicted values line", pred_on=True, start_date = datetime.datetime(2022, 5, 1), days_long = 185)
```

Loading BokehJS ...

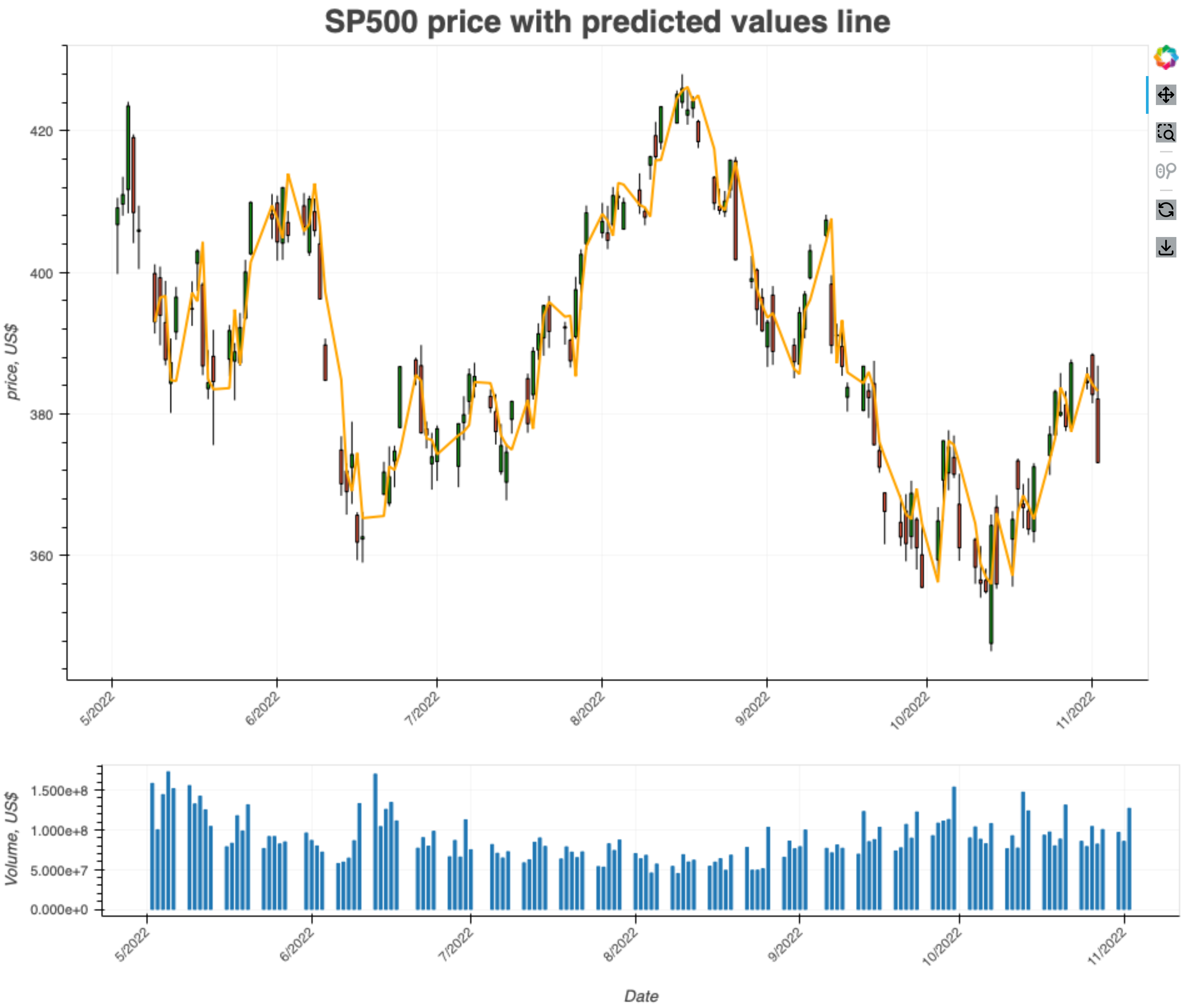


Аналогичные расчеты проделаны для индекса SP500

```
In [7]: SP500 =  FuzzyPredictor("SPY", datetime.datetime(2016, 1, 1), shape_n = 55)
In [8]: SP500.train_get_R()
        SP500_pred = SP500.create_df_predict()
```

```
In [9]: SP500.plot_bokeh(name_bokeh = "SP500 price with predicted values line", pred_on=True, start_date = datetime.datetime(2022, 5, 1), days_long = 185)
```

BokehJS 3.0.2 successfully loaded.



Стратегия на основе fuzzy predictions

Класс StockStrategy принимает датасет предсказаний класса FuzzyPredictor

In [10]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import quantstats as qs
import warnings
import numpy as np

warnings.filterwarnings("ignore")

class StockStrategy:

    def __init__(self, df, threshold_buy=0, threshold_sell=0, portfl=1000, ticker_num=0):
        self.df = df
        self.df = self.df.reset_index(drop=False)
        self.threshold_buy = threshold_buy
        self.threshold_sell = threshold_sell
        self.portfl = portfl
        self.ticker_num = ticker_num
        self.percent_guess = 0
        self.strategy_profit = None

        self.df["predicted_price_change"] = self.df.predicted_price - self.df.actual_price.shift(1)
        self.df["real_price_change"] = self.df.actual_price - self.df.actual_price.shift(1)
        self.df["GUESS_true_false"] = np.sign(self.df["real_price_change"]) == np.sign(self.df["predicted_price_change"])

        self.df["Market (Buy and Hold Strategy)"] = self.portfl * self.df.actual_price / self.df.actual_price[0]
        self.df["predicted_price_change"] = self.df.predicted_price - self.df.actual_price.shift(1)
        self.df["real_price_change"] = self.df.actual_price - self.df.actual_price.shift(1)
        self.df["GUESS_true_false"] = np.sign(self.df["real_price_change"]) == np.sign(self.df["predicted_price_change"])


    def strategy_test(self):

        self.percent_guess = round(len(self.df[self.df.GUESS_true_false == True]) / len(self.df) * 100, 1)

        for i in range(1, self.df.shape[0]):
            # buy signal
            if self.df.predicted_price_change.iloc[i] > self.threshold_buy and self.portfl != 0:
                self.ticker_num = self.portfl / self.df.iloc[i].actual_price
                self.portfl = 0

            # sell signal
            if self.df.predicted_price_change.iloc[i] < -self.threshold_sell and self.ticker_num != 0:
                self.portfl = self.ticker_num * self.df.iloc[i].actual_price
                self.ticker_num = 0

            self.df.loc[i, 'ticker_num'] = self.ticker_num
            self.df.loc[i, 'portfl'] = self.portfl
        self.df["Strategy"] = self.df.portfl + self.df.ticker_num * self.df.actual_price
        self.df["Strategy"].iloc[0] = self.df["Strategy"].iloc[1]

        self.strategy_profit = self.df[['Date', "Market (Buy and Hold Strategy)", 'Strategy']].set_index(['Date'])

        return self.strategy_profit

    def plot_strategy(self, name = "Fuzzy strategy"):
        self.strategy_profit.plot(grid=True, figsize=(14, 9))
        plt.title(name)
        plt.xlabel("Date")
        plt.ylabel("Profit, US$")


    def basic_metrics(self):
        profit = self.strategy_profit
        print(f'Cumulative return:\n{round(((profit.iloc[-1] - profit.iloc[0])/profit.iloc[0])*100,2).to_string()}\n')
        print(f'Sharpe ratio:\n{qs.stats.sharpe(profit).to_string()}\n')
        print(f'Max markdown:\n{round(qs.stats.max_drawdown(profit)*100,2).to_string()}\n')
```

Класс StockStrategy имеет следующие параметры для тюнинга стратегии: threshold_buy=0, threshold_sell=0, portfl=1000, ticker_num=0 - имеющиезначения по умолчанию

Пример использования класса на основе предсказания цен акции Apple

In [11]:

```
# Create an instance of the StrategyTester class
strategy_tester = StockStrategy(APP_pred)
```

In [12]:

```
# Test the strategy using the default parameters (threshold_buy=0, threshold_sell=0, portfl=1000, ticker_num=0)
strategy_tester.strategy_test()
```

Out[12]:

	Market (Buy and Hold Strategy)	Strategy
Date		
2022-05-09	1000.000000	1000.000000
2022-05-10	1016.111967	1000.000000
2022-05-11	963.435496	948.158793
2022-05-12	937.524635	922.658787
2022-05-13	967.447031	952.106720
...	...	...
2023-01-27	962.614166	957.132256
2023-01-30	943.286731	937.914887
2023-01-31	951.796056	937.914887
2023-02-01	959.315961	945.325120
2023-02-02	994.870711	980.361333

186 rows × 2 columns

Класс имеет 2 метода метрики качества стратегии: plot_strategy() и basic_metrics(), по которым можно оценить насколько стратегия удачна для данного инструмента.

Параметры стратегии threshold_buy, threshold_sell - показывают порог сигнала при котором происходит покупка/продажа: threshold_buy = 2, если предсказанная цена выше нынешней на 2 % - происходит покупка.

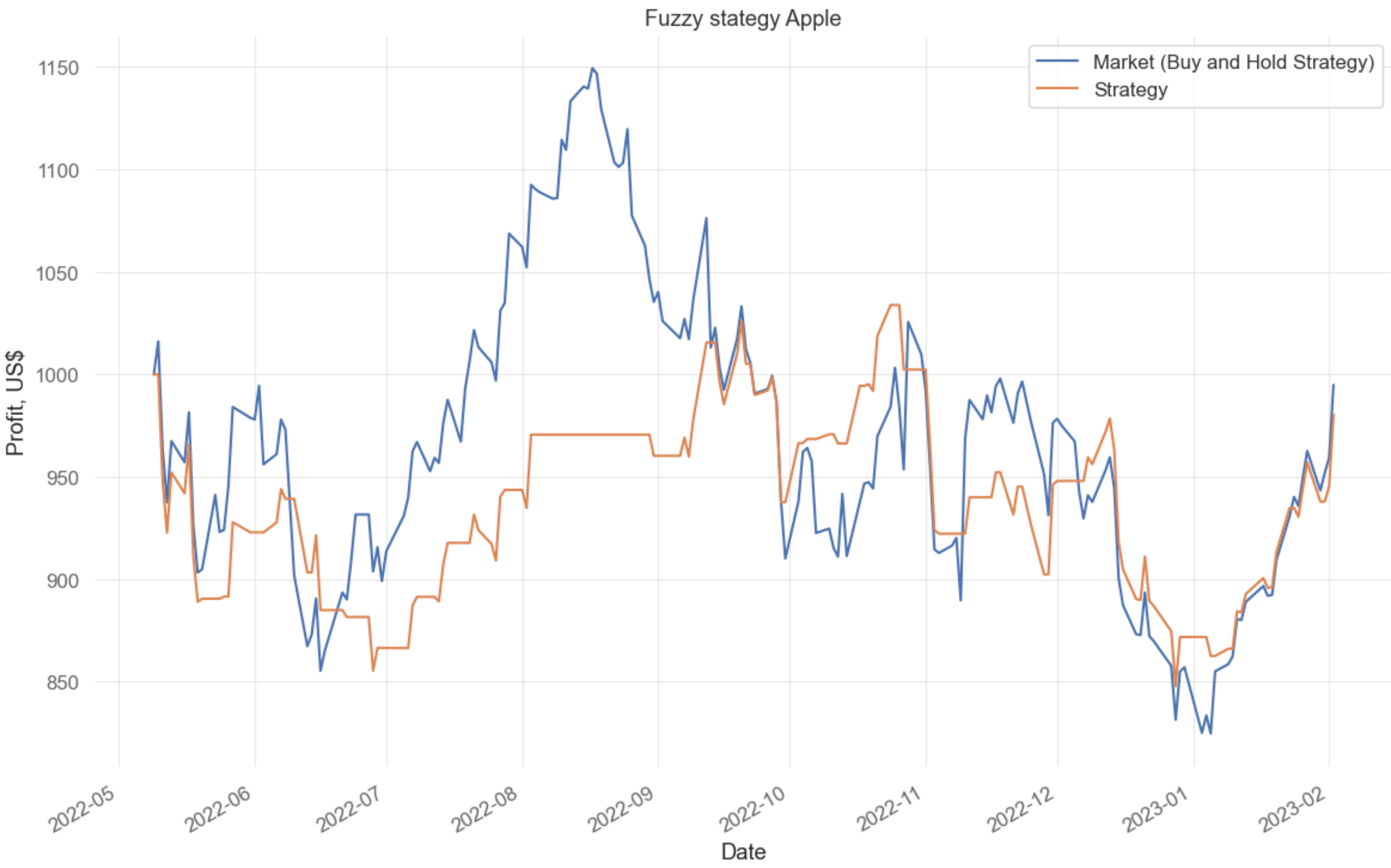
In [13]:

```
strategy_tester.plot_strategy('Fuzzy stategy Apple')
strategy_tester.basic_metrics()
```

```
Cumulative return:
Market (Buy and Hold Strategy)   -0.51
Strategy                         -1.96

Sharpe ratio:
Market (Buy and Hold Strategy)    0.160322
Strategy                          0.024197

Max markdown:
Market (Buy and Hold Strategy)   -28.26
Strategy                         -18.00
```



Прогоним стратегию для индкекса SP500:

In [14]:
```
# Create an instance of the StrategyTester class
# Test the strategy using the parameters (threshold_buy = 2.5, threshold_sell=1.5, portfl=1000, ticker_num=0)
strategy_tester_SP500 = StockStrategy(SP500_pred, threshold_buy = 2.5, threshold_sell = 1.5)
```

In [15]:
```
strategy_tester_SP500.strategy_test()
```

Out[15]:

	Market (Buy and Hold Strategy)	Strategy
Date		
2022-05-09	1000.000000	1000.000000
2022-05-10	1002.310536	1000.000000
2022-05-11	986.387662	984.113832
2022-05-12	985.357988	983.086531
2022-05-13	1008.915716	983.086531
...	...	...
2023-01-27	1032.183899	1143.791670
2023-01-30	1019.233267	1143.791670
2023-01-31	1034.219410	1143.791670
2023-02-01	1045.210858	1143.791670
2023-02-02	1060.425981	1143.791670

186 rows × 2 columns

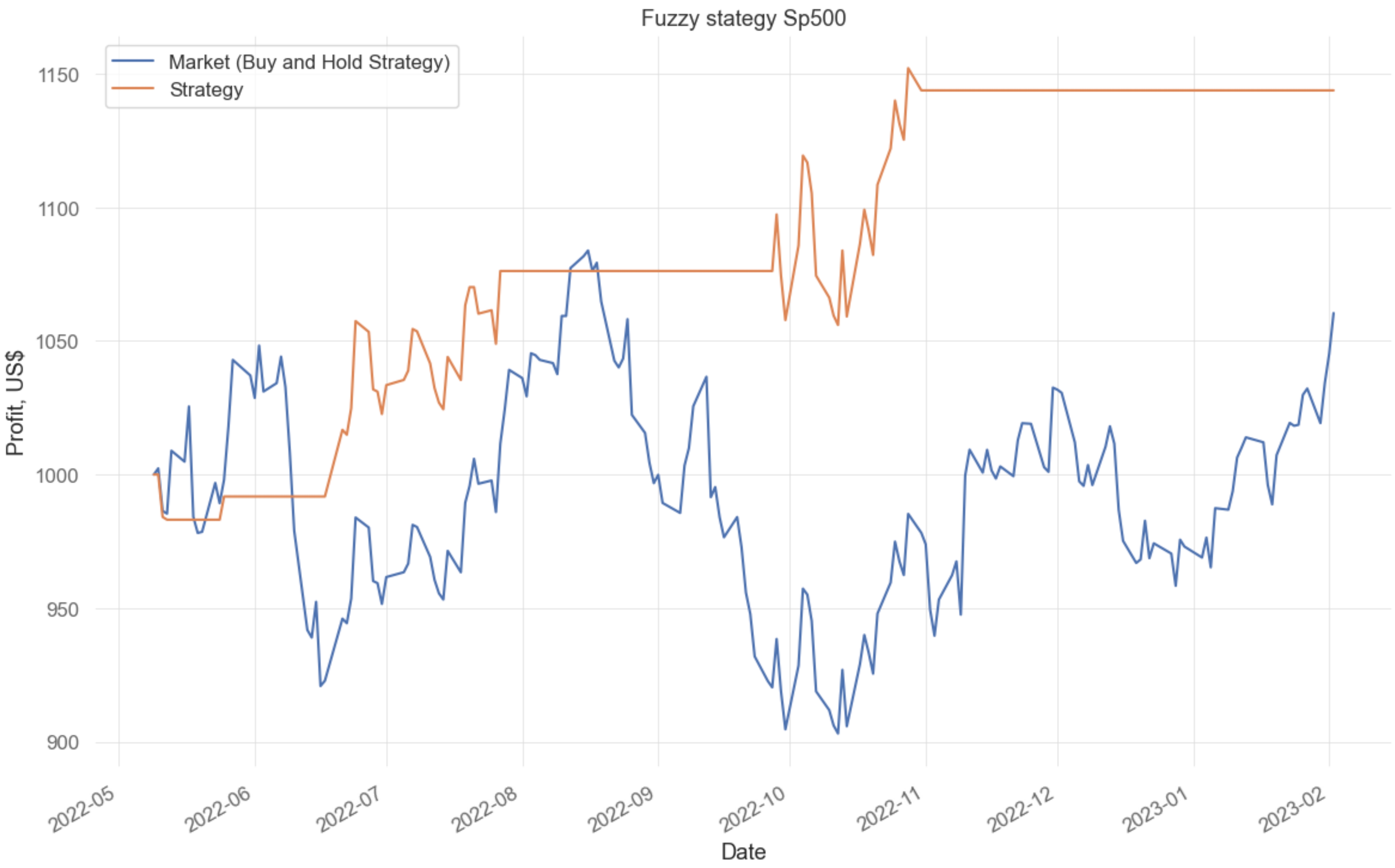
In [16]:

```
strategy_tester_SP500.plot_strategy('Fuzzy stategy Sp500')
strategy_tester_SP500.basic_metrics()
```

```
Cumulative return:
Market (Buy and Hold Strategy)     6.04
Strategy                          14.38

Sharpe ratio:
Market (Buy and Hold Strategy)    0.452934
Strategy                          1.448303

Max markdown:
Market (Buy and Hold Strategy)   -16.68
Strategy                          -5.66
```


Fuzzy stategy Sp500
Market (Buy and Hold Strategy)
Strategy
1150
1100
1050
1000
950
900
Profit, US$
2022-05
2022-06
2022-07
2022-08
2022-09
2022-10
2022-11
2022-12
2023-01
2023-02
Date

In [ ]:

In [ ]:

In [ ]:

In [ ]: