O'REILLY®

# Sztuczna inteligencja w finansach

## Używaj języka Python do projektowania i wdrażania algorytmów AI

Helion

Yves Hilpisch

Tytuł oryginału: Artificial Intelligence in Finance: A Python-Based Guide

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-8893-2



Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*https://helion.pl/user/opinie/szinfi*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

# Spis treści

# ROZDZIAŁ 11.

# Zarządzanie ryzykiem

> Poważną przeszkodą do wprowadzenia pojazdów autonomicznych na skalę masową jest konieczność zagwarantowania ich bezpieczeństwa.
>
> — Majid Khonji i in. (2019)

> Lepsze predykcje pozwalają podejmować lepsze decyzje. W końcu wiedza o tym, jak prawdopodobne są opady deszczu, nie będzie zbyt pomocna, jeśli nie wiesz, na ile zależy Ci na pozostaniu suchym lub jak bardzo nie znosisz nosić parasola.
>
> — Ajay Agrawal i in. (2018)

Zwektoryzowane testy historyczne pozwalają ocenić ekonomiczny potencjał bazującej na predykcjach strategii handlu algorytmicznego w jej wyjściowej, czystej postaci. Jednak większość inteligentnych agentów stosowanych w praktyce ma więcej komponentów niż sam model generowania predykcji. Na przykład pojazdy autonomiczne nie są w pełni samodzielne — obowiązuje je wiele reguł i heurystyk ograniczających, jakie działania sztuczna inteligencja podejmuje lub może podejmować. W pojazdach autonomicznych te reguły dotyczą przede wszystkim ograniczania ryzyka związanego z kolizjami lub wypadkami.

W kontekście finansowym inteligentne agenty lub boty handlowe też nie są stosowane w czystej postaci. Zamiast tego zwykle używa się wielu standardowych technik zarządzania ryzykiem, takich jak *zlecenia stop loss*, *trailing stop loss* lub *take profit*. Powody stosowania tego podejścia są proste — gdy otwierane są kierunkowe pozycje na rynkach finansowych, należy unikać zbyt wysokich strat. Podobnie po osiągnięciu jakiegoś poziomu zysku można je zabezpieczyć, zamykając pozycję. Stosowanie takich mechanizmów zależy najczęściej od ludzkiego osądu, czasem wspartego statystykami i formalnymi analizami odpowiednich danych. Jest to ważne zagadnienie omówione w książce Agrawala i in. (2018) — sztuczna inteligencja zapewnia lepsze predykcje, jednak ludzki osąd nadal odgrywa rolę w tworzeniu reguł podejmowania decyzji i w wyznaczaniu ograniczeń działań.

W tym rozdziale przyjąłem trzy cele. Po pierwsze, przeprowadzam testy historyczne w modelu *zwektoryzowanym* i *bazującym na zdarzeniach*, aby zbadać strategie handlu algorytmicznego oparte na agencie bazującym na algorytmie uczenia głębokiego Q-learning. Od tego miejsca takie agenty nazywam *botami handlowymi*. Po drugie, oceniam ryzyko związane z instrumentem finansowym, do którego stosowane są strategie. Po trzecie, przeprowadzam testy historyczne typowych zleceń obronnych

(takich jak zlecenia *stop loss*), używając przedstawionego w tym rozdziale podejścia bazującego na zdarzeniach. Główną zaletą testów historycznych bazujących na zdarzeniach (w porównaniu z podejściem zwektoryzowanym) jest większa swoboda w modelowaniu i analizowaniu reguł podejmowania decyzji oraz mechanizmów zarządzania ryzykiem. Można więc dokładniej przyjrzeć się szczegółom, które w podejściu zwektoryzowanym są mniej widoczne.

W podrozdziale „Bot handlowy" opisuję i uczę bota handlowego bazującego na agencie używającym algorytmu Q-learning z rozdziału 9. W podrozdziale „Zwektoryzowane testy historyczne" używam zwektoryzowanych testów historycznych z rozdziału 10. do oceny wyników ekonomicznych wspomnianego bota. W podrozdziale „Testy historyczne bazujące na zdarzeniach" przedstawiam tytułowe testy. Najpierw omawiam klasę bazową, a następnie z jej użyciem piszę i przeprowadzam testy historyczne bota handlowego (zobacz też Hilpisch, 2020, rozdział 6.). W podrozdziale „Ocena ryzyka" analizuję wybrane wskaźniki statystyczne ważne przy tworzeniu reguł zarządzania ryzykiem, na przykład *maksymalne osunięcie* i *wskaźnik ATR* (ang. *average true range*). W podrozdziale „Testy historyczne zleceń obronnych" przeprowadzam testy historyczne wpływu stosowania głównych zleceń obronnych na wyniki bota handlowego.

# Bot handlowy

W tym podrozdziale przedstawiam bota handlowego bazującego na finansowym agencie używającym algorytmu Q-learning — `FQLAgent` z rozdziału 9. Jest to bot handlowy analizowany w dalszych podrozdziałach. Jak zwykle należy zacząć od instrukcji importu:

```
In [1]: import os
        import numpy as np
        import pandas as pd
        from pylab import plt, mpl
        plt.style.use('seaborn')
        mpl.rcParams['savefig.dpi'] = 300
        mpl.rcParams['font.family'] = 'serif'
        pd.set_option('mode.chained_assignment', None)
        pd.set_option('display.float_format', '{:.4f}'.format)
        np.set_printoptions(suppress=True, precision=4)
        os.environ['PYTHONHASHSEED'] = '0'
```

W podrozdziale „Środowisko finansowe" przedstawiam moduł Pythona z używaną dalej klasą `Finance`. W podrozdziale „Bot handlowy" przedstawiam moduł Pythona z klasą `TradingBot` oraz funkcjami pomocniczymi do wyświetlania wyników uczenia i walidacji. Obie wspomniane klasy są podobne do klas z rozdziału 9., dlatego używam ich bez dodatkowych objaśnień.

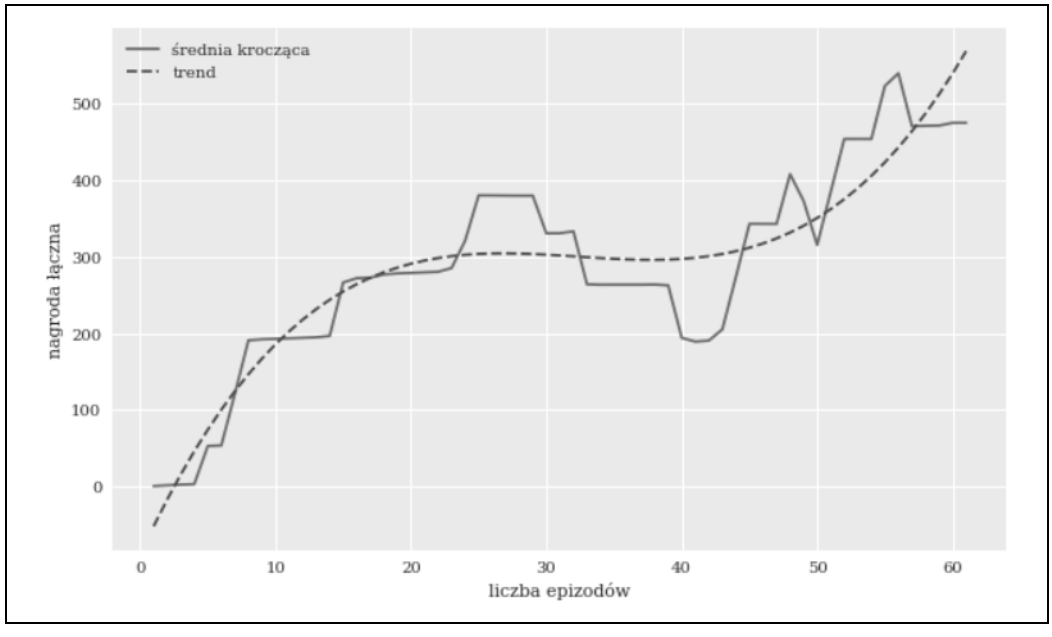
Poniższy kod uczy bota handlowego na podstawie historycznych danych EOD. Uwzględniony jest też podzbiór danych walidacyjnych. Na rysunku 11.1 widać średnią łączną nagrodę uzyskaną w różnych epizodach:

```
In [2]: import finance
        import tradingbot
        Using TensorFlow backend.

In [3]: symbol = 'EUR='
        features = [symbol, 'r', 's', 'm', 'v']
```

```
In [4]: a = 0
        b = 1750
        c = 250

In [5]: learn_env = finance.Finance(symbol, features, window=20, lags=3,
                        leverage=1, min_performance=0.9, min_accuracy=0.475,
                        start=a, end=a + b, mu=None, std=None)

In [6]: learn_env.data.info()
        <class 'pandas.core.frame.DataFrame'>
        DatetimeIndex: 1750 entries, 2010-02-02 to 2017-01-12
        Data columns (total 6 columns):
         #   Column  Non-Null Count  Dtype
        ---  ------  --------------  -----
         0   EUR=    1750 non-null   float64
         1   r       1750 non-null   float64
         2   s       1750 non-null   float64
         3   m       1750 non-null   float64
         4   v       1750 non-null   float64
         5   d       1750 non-null   int64
        dtypes: float64(5), int64(1)
        memory usage: 95.7 KB

In [7]: valid_env = finance.Finance(symbol, features=learn_env.features,
                                    window=learn_env.window,
                                    lags=learn_env.lags,
                                    leverage=learn_env.leverage,
                                    min_performance=0.0, min_accuracy=0.0,
                                    start=a + b, end=a + b + c,
                                    mu=learn_env.mu, std=learn_env.std)

In [8]: valid_env.data.info()
        <class 'pandas.core.frame.DataFrame'>
        DatetimeIndex: 250 entries, 2017-01-13 to 2018-01-10
        Data columns (total 6 columns):
         #   Column  Non-Null Count  Dtype
        ---  ------  --------------  -----
         0   EUR=    250 non-null    float64
         1   r       250 non-null    float64
         2   s       250 non-null    float64
         3   m       250 non-null    float64
         4   v       250 non-null    float64
         5   d       250 non-null    int64
        dtypes: float64(5), int64(1)
        memory usage: 13.7 KB

In [9]: tradingbot.set_seeds(100)
        agent = tradingbot.TradingBot(24, 0.001, learn_env, valid_env)

In [10]: episodes = 61

In [11]: %time agent.learn(episodes)
         ================================================================================
         epizod: 10/61 | WALIDACJA | nag. łącz.: 247 | wynik: 0.936 | eps.: 0.95
         ================================================================================
         ================================================================================
         epizod: 20/61 | WALIDACJA | nag. łącz.: 247 | wynik: 0.897 | eps.: 0.86
         ================================================================================
```

```
================================================================================
epizod: 30/61 | WALIDACJA | nag. łącz.: 247 | wynik: 1.035 | eps.: 0.78
================================================================================
================================================================================
epizod: 40/61 | WALIDACJA | nag. łącz.: 247 | wynik: 0.935 | eps.: 0.70
================================================================================
================================================================================
epizod: 50/61 | WALIDACJA | nag. łącz.: 247 | wynik: 0.890 | eps.: 0.64
================================================================================
================================================================================
epizod: 60/61 | WALIDACJA | nag. łącz.: 247 | wynik: 0.998 | eps.: 0.58
================================================================================
epizod: 61/61 | nag. łącz.: 17 | wynik: 0.979 | średnia: 475.1 | maks.: 1747
CPU times: user 51.4 s, sys: 2.53 s, total: 53.9 s
Wall time: 47 s
```

```
In [12]: tradingbot.plot_treward(agent)
```

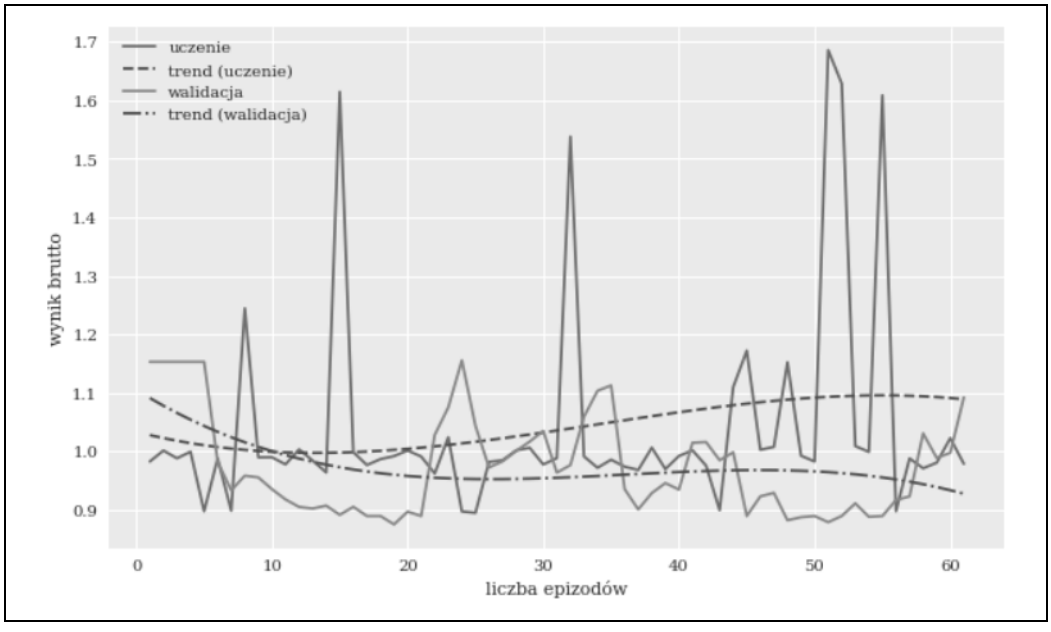



*Rysunek 11.1. Średnia łączna nagroda w poszczególnych epizodach uczenia*

Na rysunku 11.2 porównane są wyniki brutto bota handlowego dla danych treningowych (z dość wysoką zmiennością z powodu stosowania zarówno eksploatacji, jak i eksploracji) i dla danych walidacyjnych (tylko eksploatacja):

```
In [13]: tradingbot.plot_performance(agent)
```

Ten wyuczony bot handlowy jest używany w testach historycznych w dalszych podrozdziałach.




*Rysunek 11.2. Wynik brutto dla treningowego i walidacyjnego zbioru danych*

# Zwektoryzowane testy historyczne

Nie da się przeprowadzić zwektoryzowanych testów historycznych bezpośrednio na bocie handlowym. W rozdziale 10. do zilustrowania takich testów posłużyłem się gęstymi sieciami neuronowymi. W tym ujęciu dane z cechami i etykietami są najpierw przygotowywane, a następnie przekazywane do gęstej sieci neuronowej w celu wygenerowania wszystkich predykcji. W uczeniu przez wzmacnianie dane są generowane i zbierane w wyniku interakcji ze środowiskiem działanie po działaniu i krok po kroku.

W poniższym kodzie w Pythonie zdefiniowałem funkcję `backtest`, która jako dane wejściowe przyjmuje instancje klas `TradingBot` i `Finance`. Funkcja ta generuje w pierwotnych obiektach typu `DataFrame` ze środowiska `Finance` kolumny z pozycjami otwieranymi przez bota handlowego i wynikami strategii:

```
In [14]: def reshape(s):
             return np.reshape(s, [1, learn_env.lags,
                                   learn_env.n_features]) ❶

In [15]: def backtest(agent, env):
             env.min_accuracy = 0.0
             env.min_performance = 0.0
             done = False
             env.data['p'] = 0 ❷
             state = env.reset()
             while not done:
                 action = np.argmax(
                     agent.model.predict(reshape(state))[0, 0]) ❸
```

```
            position = 1 if action == 1 else -1 ❹
            env.data.loc[:, 'p'].iloc[env.bar] = position ❺
            state, reward, done, info = env.step(action)
        env.data['s'] = env.data['p'] * env.data['r'] * learn_env.leverage ❻
```

❶ Przekształca dane na kombinacje cech i etykiet.

❷ Generowanie kolumny z wartościami reprezentującymi otwarte pozycje.

❸ Określanie optymalnego działania (predykcji) za pomocą wyuczonej gęstej sieci neuronowej.

❹ Ustalanie otwartej pozycji (+1 to długa, –1 to krótka)…

❺ …i zapisywanie w odpowiedniej kolumnie pod właściwym indeksem.

❻ Obliczanie logarytmicznych stóp zwrotu dla strategii na podstawie zajmowanych pozycji.

Dzięki funkcji `backtest` zwektoryzowane testy historyczne sprowadzają się do kilku wierszy kodu w Pythonie, tak jak w rozdziale 10.

Na rysunku 11.3 porównany jest wynik brutto pasywnej inwestycji porównawczej z wynikiem brutto strategii:

```
In [16]: env = agent.learn_env ❶

In [17]: backtest(agent, env) ❷

In [18]: env.data['p'].iloc[env.lags:].value_counts() ❸
Out[18]:  1    961
          -1    786
          Name: p, dtype: int64

In [19]: env.data[['r', 's']].iloc[env.lags:].sum().apply(np.exp) ❹
Out[19]: r    0.7725
          s    1.5155
          dtype: float64

In [20]: env.data[['r', 's']].iloc[env.lags:].sum().apply(np.exp) - 1 ❺
Out[20]: r   -0.2275
          s    0.5155
          dtype: float64

In [21]: env.data[['r', 's']].iloc[env.lags:].cumsum(
                 ).apply(np.exp).plot(figsize=(10, 6));
```

❶ Określanie środowiska.

❷ Generowanie dodatkowych potrzebnych danych.

❸ Zliczanie pozycji długich i krótkich.

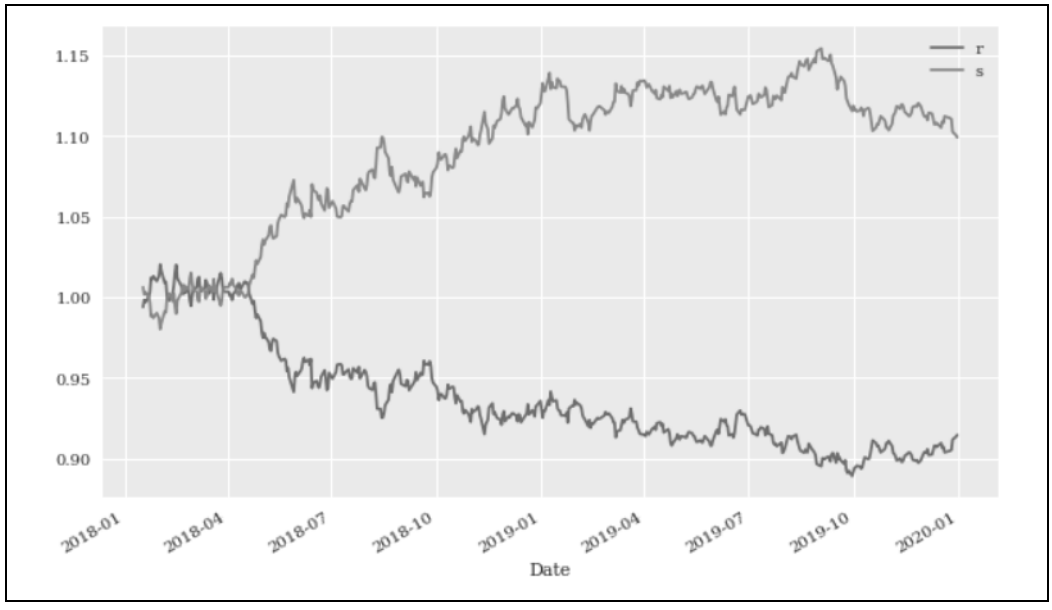
❹ Obliczanie wyniku brutto pasywnej inwestycji porównawczej (`r`) i strategii (`s`)…

❺ …a także odpowiednich wyników netto.


*Rysunek 11.3. Wynik brutto pasywnej inwestycji porównawczej i bota handlowego (dane z próbki)*

Aby uzyskać bardziej realistyczny obraz wyników bota handlowego, w następnym fragmencie kodu w Pythonie tworzę środowisko testowe z danymi, których bot jeszcze nie zna. Na rysunku 11.4 pokazane jest, jak bot radzi sobie w porównaniu z pasywną inwestycją porównawczą:

```
In [22]: test_env = finance.Finance(symbol, features=learn_env.features,
                                    window=learn_env.window,
                                    lags=learn_env.lags,
                                    leverage=learn_env.leverage,
                                    min_performance=0.0, min_accuracy=0.0,
                                    start=a + b + c, end=None,
                                    mu=learn_env.mu, std=learn_env.std)

In [23]: env = test_env

In [24]: backtest(agent, env)

In [25]: env.data['p'].iloc[env.lags:].value_counts()
Out[25]: -1    437
          1     56
         Name: p, dtype: int64

In [26]: env.data[['r', 's']].iloc[env.lags:].sum().apply(np.exp)
Out[26]: r    0.9144
         s    1.0992
         dtype: float64

In [27]: env.data[['r', 's']].iloc[env.lags:].sum().apply(np.exp) - 1
Out[27]: r   -0.0856
         s    0.0992
         dtype: float64

In [28]: env.data[['r', 's']].iloc[env.lags:].cumsum(
                    ).apply(np.exp).plot(figsize=(10, 6));
```




*Rysunek 11.4. Wynik brutto pasywnej inwestycji porównawczej i bota handlowego (dane spoza próbki)*

Wyniki spoza próbki nawet bez stosowania żadnych zleceń obronnych wyglądają obiecująco. Jednak aby móc prawidłowo ocenić rzeczywiste wyniki strategii handlowej, należy dodać takie zlecenia. Do tego przydatne będą testy historyczne bazujące na zdarzeniach.

# Testy historyczne bazujące na zdarzeniach

Opisane w poprzednim podrozdziale wyniki dla danych spoza próbki bez zleceń obronnych są obiecujące. Jednak aby odpowiednio przeanalizować takie mechanizmy, na przykład zlecenia *trailing stop loss*, potrzebne są *testy historyczne bazujące na zdarzeniach*. W tym podrozdziale przedstawiam tę inną metodę oceny skuteczności strategii handlu algorytmicznego.

W podrozdziale „Klasa BacktestingBase" przedstawiam klasę `BacktestingBase`, którą można stosować do testowania różnych rodzajów strategii kierunkowych. W kodzie tej klasy znajdziesz szczegółowe komentarze na temat ważnych wierszy. Oto metody z tej klasy:

`get_date_price()`

: Zwraca datę (`date`) i cenę (`price`) dla danego słupka (`bar`; jest to indeks z obiektu typu `DataFrame` zawierającego dane finansowe).

`print_balance()`

: Wyświetla aktualny stan konta (gotówkę) bota handlowego dla danego słupka.

`calculate_net_wealth()`

: Dla danej ceny (`price`) wyświetla majątek netto obejmujący aktualny stan konta (gotówkę) i wartość pozycji.

`print_net_wealth()`

Dla danego słupka wyświetla majątek netto bota handlowego.

`place_buy_order()`, `place_sell_order()`

Na podstawie danego słupka i określonej liczby jednostek (`units`) lub kwoty (`amount`) te metody składają zlecenie kupna lub sprzedaży i odpowiednio dostosowują właściwe wartości (na przykład z uwzględnieniem kosztów transakcyjnych).

`close_out()`

Dla danego słupka ta metoda zamyka otwarte pozycje oraz oblicza i wyświetla statystyki dotyczące wyników.

Poniższy kod w Pythonie pokazuje w prostych krokach, jak działa instancja klasy `BacktestingBase`:

```
In [29]: import backtesting as bt

In [30]: bb = bt.BacktestingBase(env=agent.learn_env, model=agent.model,
                                 amount=10000, ptc=0.0001, ftc=1.0,
                                 verbose=True) ❶

In [31]: bb.initial_amount ❷
Out[31]: 10000

In [32]: bar = 100 ❸

In [33]: bb.get_date_price(bar) ❹
Out[33]: ('2010-06-25', 1.2374)

In [34]: bb.env.get_state(bar) ❺
Out[34]:               EUR=       r       s       m      v
         Date
         2010-06-22 -0.0242 -0.5622 -0.0916 -0.2022 1.5316
         2010-06-23  0.0176  0.6940 -0.0939 -0.0915 1.5563
         2010-06-24  0.0354  0.3034 -0.0865  0.6391 1.0890

In [35]: bb.place_buy_order(bar, amount=5000) ❻
         2010-06-25 | zakup 4040 jednostek po 1.2374
         2010-06-25 | stan gotówki = 4999.40

In [36]: bb.print_net_wealth(2 * bar) ❼
         2010-11-16 | majątek netto = 10450.17

In [37]: bb.place_sell_order(2 * bar, units=1000) ❽
         2010-11-16 | sprzedaż 1000 jednostek po 1.3492
         2010-11-16 | stan gotówki = 6347.47

In [38]: bb.close_out(3 * bar) ❾
         ================================================
         2011-04-11 | *** ZAMKNIĘCIE POZYCJI ***
         2011-04-11 | sprzedaż 3040 jednostek po 1.4434
         2011-04-11 | stan gotówki = 10733.97
         2011-04-11 | wynik netto [%] = 7.3397
         2011-04-11 | liczba transakcji [#] = 3
         ================================================
```

❶ Tworzenie instancji klasy `BacktestingBase`.

❷ Sprawdzanie wartości atrybutu `initial_amount`.

❸ Podawanie numeru słupka (`bar`).

❹ Pobieranie daty (`date`) i ceny (`price`) dla słupka (`bar`).

❺ Pobieranie stanu środowiska `Finance` dla słupka (`bar`).

❻ Składanie zlecenia kupna na podstawie wartości parametru `amount`.

❼ Wyświetlanie majątku netto w późniejszym momencie (`2 * bar`).

❽ Składanie zlecenia sprzedaży w późniejszym momencie na podstawie wartości parametru `units`.

❾ Zamykanie reszty długiej pozycji jeszcze później (`3 * bar`).

Klasa `TBBacktester` dziedziczy po klasie `BacktestingBase` i obsługuje bazujące na zdarzeniach testy historyczne bota handlowego:

```
In [39]: class TBBacktester(bt.BacktestingBase):
             def _reshape(self, state):
                 ''' Metoda pomocnicza do zmiany kształtu obiektów state.
                 '''
                 return np.reshape(state, [1, self.env.lags, self.env.n_features])
             def backtest_strategy(self):
                 ''' Bazujące na zdarzeniach testy historyczne wyników bota handlowego.
                 '''
                 self.units = 0
                 self.position = 0
                 self.trades = 0
                 self.current_balance = self.initial_amount
                 self.net_wealths = list()
                 for bar in range(self.env.lags, len(self.env.data)):
                     date, price = self.get_date_price(bar)
                     if self.trades == 0:
                         print(50 * '=')
                         print(f'{date} | *** POCZĄTEK TESTU ***')
                         self.print_balance(bar)
                         print(50 * '=')
                     state = self.env.get_state(bar) ❶
                     action = np.argmax(self.model.predict(
                                 self._reshape(state.values))[0, 0]) ❷
                     position = 1 if action == 1 else -1 ❸
                     if self.position in [0, -1] and position == 1: ❹
                         if self.verbose:
                             print(50 * '-')
                             print(f'{date} | *** OTWARCIE DŁUGIEJ POZYCJI ***')
                         if self.position == -1:
                             self.place_buy_order(bar - 1, units=-self.units)
                         self.place_buy_order(bar - 1,
                                              amount=self.current_balance)
                         if self.verbose:
                             self.print_net_wealth(bar)
                         self.position = 1
                     elif self.position in [0, 1] and position == -1: ❺
                         if self.verbose:
                             print(50 * '-')
                             print(f'{date} | *** OTWARCIE KRÓTKIEJ POZYCJI ***')
```

```
                if self.position == 1:
                    self.place_sell_order(bar - 1, units=self.units)
                self.place_sell_order(bar - 1,
                                      amount=self.current_balance)
                if self.verbose:
                    self.print_net_wealth(bar)
                self.position = -1
            self.net_wealths.append((date,
                                     self.calculate_net_wealth(price))) ❻
        self.net_wealths = pd.DataFrame(self.net_wealths,
                                        columns=['date', 'net_wealth']) ❻
        self.net_wealths.set_index('date', inplace=True) ❻
        self.net_wealths.index = pd.DatetimeIndex(
                                        self.net_wealths.index) ❻
        self.close_out(bar)
```

❶ Pobieranie stanu środowiska `Finance`.

❷ Generowanie optymalnych działań (predykcji) na podstawie stanu i obiektu `model`.

❸ Określanie optymalnej pozycji (długa/krótka) na podstawie optymalnych działań (predykcji).

❹ Otwieranie *długiej* pozycji, gdy spełnione są warunki.

❺ Otwieranie *krótkiej* pozycji, gdy spełnione są warunki.

❻ Pobieranie wartości majątku netto w różnych momentach i przekształcanie tych wartości na obiekt typu `DataFrame`.

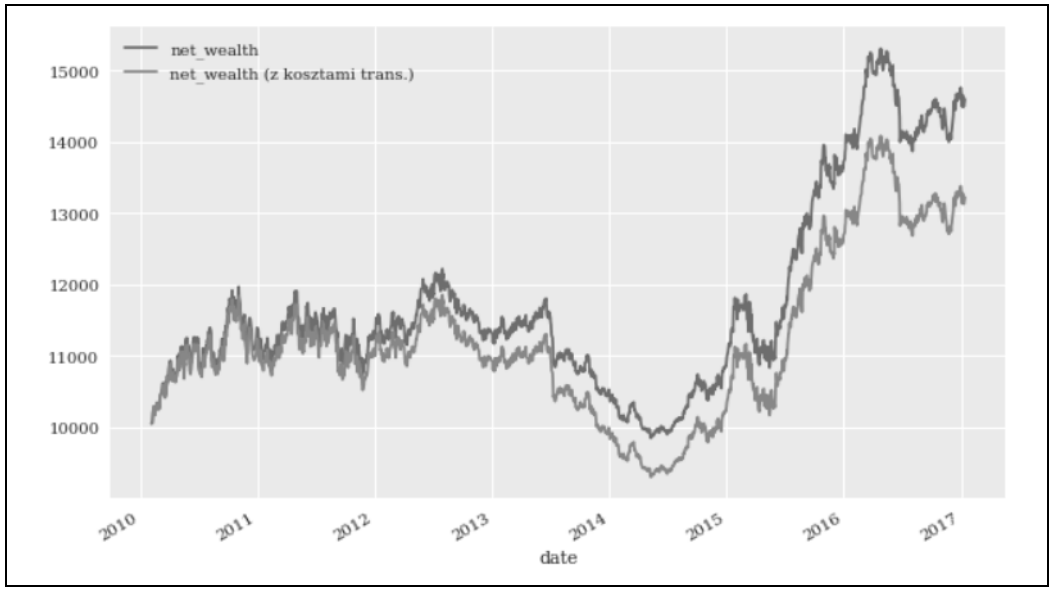
Ponieważ dostępne są już instancje klas `Finance` i `TradingBot`, używanie klasy `TBBacktester` jest proste. Poniższy kod przeprowadza testy historyczne bota handlowego najpierw na danych ze *środowiska, w którym przeprowadzono uczenie* — bez kosztów transakcyjnych i z ich uwzględnieniem. Rysunek 11.5 porównuje wyniki dla tych dwóch możliwości:

```
In [40]: env = learn_env

In [41]: tb = TBBacktester(env, agent.model, 10000,
                           0.0, 0, verbose=False) ❶

In [42]: tb.backtest_strategy() ❶
         ================================================
         2010-02-05 | *** POCZĄTEK TESTU ***
         2010-02-05 | stan gotówki = 10000.00
         ================================================
         ================================================
         2017-01-12 | *** ZAMKNIĘCIE POZYCJI ***
         2017-01-12 | stan gotówki = 14601.85
         2017-01-12 | wynik netto [%] = 46.0185
         2017-01-12 | liczba transakcji [#] = 828
         ================================================

In [43]: tb_ = TBBacktester(env, agent.model, 10000,
                            0.00012, 0.0, verbose=False) ❷

In [44]: tb_.backtest_strategy() ❷
         ================================================
         2010-02-05 | *** POCZĄTEK TESTU ***
         2010-02-05 | stan gotówki = 10000.00
```

```
        ================================================
        ================================================
        2017-01-12 | *** ZAMKNIĘCIE POZYCJI ***
        2017-01-12 | stan gotówki = 13222.08
        2017-01-12 | wynik netto [%] = 32.2208
        2017-01-12 | liczba transakcji [#] = 828
        ================================================

In [45]: ax = tb.net_wealths.plot(figsize=(10, 6))
         tb_.net_wealths.columns = ['net_wealth (z kosztami trans.)']
         tb_.net_wealths.plot(ax=ax);
```

❶ Test bazujący na zdarzeniach dla danych z próbki *bez* kosztów transakcyjnych.

❷ Test bazujący na zdarzeniach dla danych z próbki *z* kosztami transakcyjnymi.




*Rysunek 11.5. Wynik brutto bota handlowego bez kosztów transakcyjnych i z nimi (dane z próbki)*

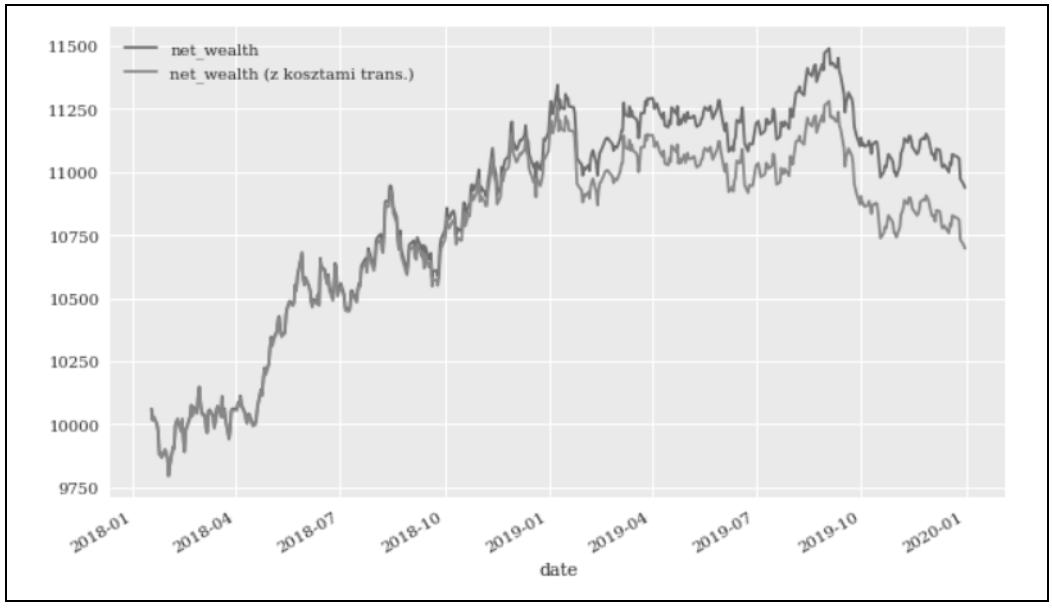
Na rysunku 11.6 porównany jest wynik brutto dla *środowiska testowego*. Ponownie uwzględnione są przypadki bez kosztów transakcyjnych i z kosztami transakcyjnymi:

```
In [46]: env = test_env

In [47]: tb = TBBacktester(env, agent.model, 10000,
                           0.0, 0, verbose=False) ❶

In [48]: tb.backtest_strategy() ❶
         ================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ================================================
         ================================================
         2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
         2019-12-31 | stan gotówki = 10936.79
```

```
        2019-12-31 | wynik netto [%] = 9.3679
        2019-12-31 | liczba transakcji [#] = 186
        ================================================

In [49]: tb_ = TBBacktester(env, agent.model, 10000,
                            0.00012, 0.0, verbose=False) ❷

In [50]: tb_.backtest_strategy() ❷
        ================================================
        2018-01-17 | *** POCZĄTEK TESTU ***
        2018-01-17 | stan gotówki = 10000.00
        ================================================
        ================================================
        2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
        2019-12-31 | stan gotówki = 10695.72
        2019-12-31 | wynik netto [%] = 6.9572
        2019-12-31 | liczba transakcji [#] = 186
        ================================================

In [51]: ax = tb.net_wealths.plot(figsize=(10, 6))
        tb_.net_wealths.columns = ['net_wealth (z kosztami trans.)']
        tb_.net_wealths.plot(ax=ax);
```

❶ Test bazujący na zdarzeniach dla danych spoza próbki *bez* kosztów transakcyjnych.

❷ Test bazujący na zdarzeniach dla danych spoza próbki *z* kosztami transakcyjnymi.

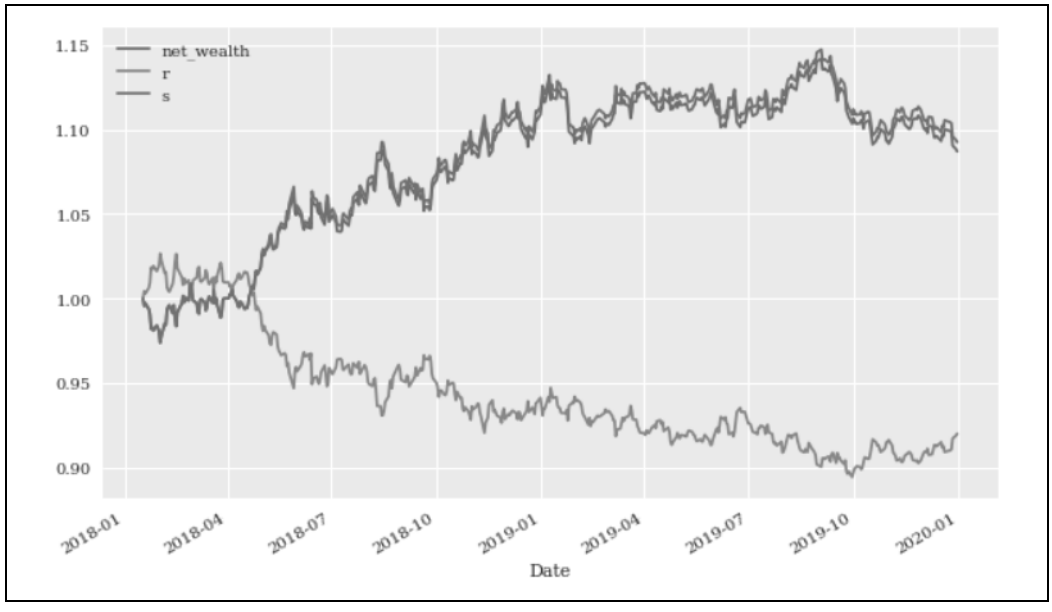



*Rysunek 11.6. Wynik brutto bota handlowego bez kosztów transakcyjnych i z nimi (dane spoza próbki)*

Jak wypada porównanie wyników bez kosztów transakcyjnych z testu bazującego na zdarzeniach i testu zwektoryzowanego? Na rysunku 11.7 pokazany jest zgromadzony znormalizowany majątek brutto z obu metod. Z powodu zastosowania różnych podejść technicznych dwa widoczne szeregi czasowe nie są identyczne, ale są dość zbliżone do siebie. Różnice w wynikach można wyjaśnić przede

wszystkim tym, że w testach bazujących na zdarzeniach zakłada się, że każda pozycja ma tę samą wartość. W testach zwektoryzowanych uwzględniany jest procent składany, co skutkuje nieco wyższym wynikiem:

```
In [52]: ax = (tb.net_wealths / tb.net_wealths.iloc[0]).plot(figsize=(10, 6))
         tp = env.data[['r', 's']].iloc[env.lags:].cumsum().apply(np.exp)
         (tp / tp.iloc[0]).plot(ax=ax);
```




*Rysunek 11.7. Wynik brutto dla pasywnej inwestycji porównawczej i bota handlowego (testy zwektoryzowane i bazujące na zdarzeniach)*

### Różnice w wynikach

Wyniki z testów zwektoryzowanego i bazującego na zdarzeniach są zbliżone, ale nie identyczne. W teście zwektoryzowanym zakładam, że zawsze można kupić instrument finansowymi za całą dostępną kwotę. Ponadto zyski są stale reinwestowane. W teście bazującym na zdarzeniach kupowane są tylko pełne jednostki instrumentu finansowego, co lepiej odzwierciedla rzeczywistość. Do obliczania majątku netto używane są różnice w cenach. Kod testu bazującego na zdarzeniach w obecnej postaci nie sprawdza, czy aktualny stan konta jest wystarczający, by zapewnić pokrycie danej transakcji. Jest to oczywiście uproszczenie, a kupowanie na kredyt nie zawsze jest możliwe. Można jednak łatwo wprowadzić potrzebne poprawki w kodzie klasy `BacktestingBase`.

# Ocena ryzyka

Wprowadzenie zleceń obronnych wymaga zrozumienia ryzyk związanych z handlem wybranym instrumentem. Dlatego aby odpowiednio skonfigurować parametry dla takich zleceń (takich jak zlecenia stop loss), ważna jest ocena ryzyk. Używanych jest wiele technik pomiaru ryzyka związanego

z instrumentem finansowym. Istnieją na przykład *niekierunkowe miary ryzyka* takie jak zmienność lub wskaźnik ATR (ang. *average true return*). Stosuje się też *kierunkowe miary ryzyka* takie jak maksymalne osunięcie lub VaR (ang. *value at risk*).

Częstą praktyką stosowaną przy wyznaczaniu poziomów zleceń stop loss, zleceń trailing stop loss lub zleceń take profit jest łączenie ich z wartością wskaźnika ATR[1]. Poniższy kod w Pythonie oblicza wskaźnik ATR w wartościach względnych i bezwzględnych dla instrumentu finansowego używanego do uczenia i testowania bota handlowego (czyli dla kursu wymiany pary walutowej EUR/USD). W obliczeniach wykorzystywane są dane ze środowiska treningowego i typowe okno o długości 14 dni (słupków). Na rysunku 11.8 pokazane są obliczone wartości (zauważ ich znaczną zmienność w czasie):

```
In [53]: data = pd.DataFrame(learn_env.data[symbol]) ❶

In [54]: data.head() ❶
Out[54]:              EUR=
         Date
         2010-02-02 1.3961
         2010-02-03 1.3898
         2010-02-04 1.3734
         2010-02-05 1.3662
         2010-02-08 1.3652

In [55]: window = 14 ❷

In [56]: data['min'] = data[symbol].rolling(window).min() ❸

In [57]: data['max'] = data[symbol].rolling(window).max() ❹

In [58]: data['mami'] = data['max'] - data['min'] ❺

In [59]: data['mac'] = abs(data['max'] - data[symbol].shift(1)) ❻

In [60]: data['mic'] = abs(data['min'] - data[symbol].shift(1)) ❼

In [61]: data['atr'] = np.maximum(data['mami'], data['mac']) ❽

In [62]: data['atr'] = np.maximum(data['atr'], data['mic']) ❾

In [63]: data['atr%'] = data['atr'] / data[symbol] ❿

In [64]: data[['atr', 'atr%']].plot(subplots=True, figsize=(10, 6));
```

❶ Kolumna z cenami instrumentu z oryginalnego obiektu typu `DataFrame`.

❷ Długość okna używanego w obliczeniach.

❸ Kroczące minimum.

❹ Kroczące maksimum.

❺ Różnica między kroczącym maksimum i minimum.

❻ Bezwzględna różnica między kroczącym maksimum a ceną z poprzedniego dnia.

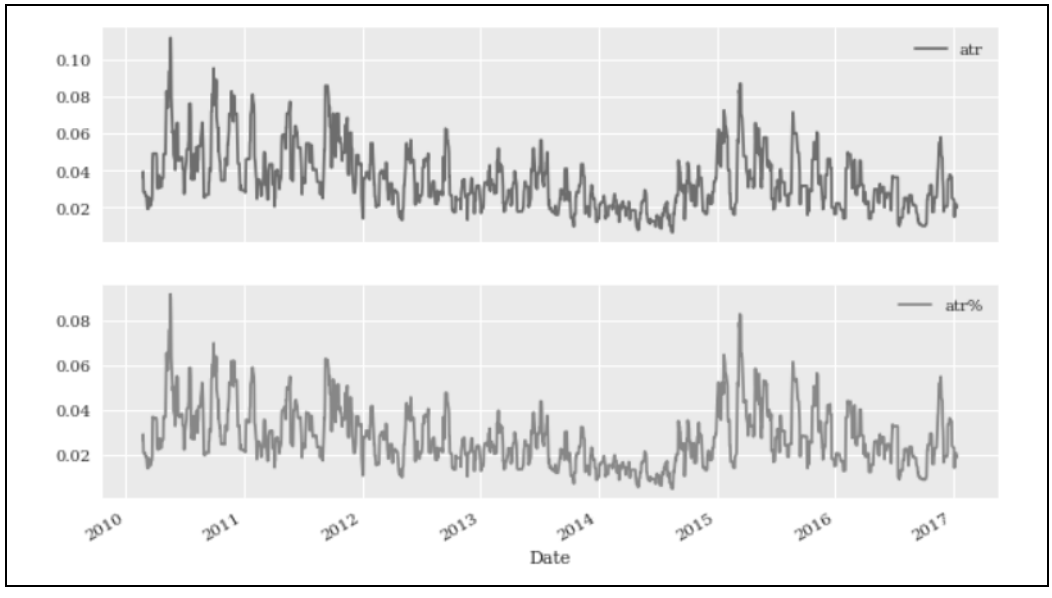
[1] Więcej informacji o wskaźniku ATR znajdziesz w Investopedii (*https://oreil.ly/2sUsg* i *https://oreil.ly/zwrnO*).

❼ Bezwzględna różnica między kroczącym minimum a ceną z poprzedniego dnia.

❽ Maksimum z różnic maksimum-minimum i maksimum-cena.

❾ Maksimum z maksimum z poprzedniego punktu i różnicy minimum-cena (= ATR).

❿ Procentowo ujęty wskaźnik ATR (bezwzględna wartość ATR względem ceny).




*Rysunek 11.8. Wskaźnik ATR wyrażony w wartościach bezwzględnych (cena) i względnych (procent)*

Poniższy kod wyświetla ostateczne wartości wskaźnika ATR w ujęciu bezwzględnym i względnym. Typową regułą jest na przykład ustawianie poziomu zlecenia stop loss na cenę otwarcia pozycji minus *x* razy ATR. W zależności od akceptacji ryzyka przez gracza lub inwestora *x* może być mniejsze lub większe od 1. Na tym poziomie stosowane są osąd człowieka lub formalne strategie zarządzania ryzykiem. Jeśli $x = 1$, poziom zlecenia stop loss to mniej więcej 2% poniżej ceny otwarcia pozycji:

```
In [65]: data[['atr', 'atr%']].tail()

Out[65]:                atr   atr%
         Date
         2017-01-06 0.0218 0.0207
         2017-01-09 0.0218 0.0206
         2017-01-10 0.0218 0.0207
         2017-01-11 0.0199 0.0188
         2017-01-12 0.0206 0.0194
```

Istotna jest tu *dźwignia finansowa*. Jeśli używana jest dźwignia na poziomie 10 (jest to dość niska wartość jak na handel na foreksie), wskaźnik ATR trzeba pomnożyć przez tę wartość. Dlatego gdy mnożnik wskaźnika ATR wynosi 1, wcześniejsze zlecenie stop loss należy ustawić na poziomie 20% (a nie 2%) poniżej ceny otwarcia pozycji. Jeżeli uwzględnić medianę wskaźnika ATR z całego zbioru danych, poziom ten powinien wynosić 25%:

```
In [66]: leverage = 10

In [67]: data[['atr', 'atr%']].tail() * leverage
Out[67]:               atr    atr%
         Date
         2017-01-06 0.2180 0.2070
         2017-01-09 0.2180 0.2062
         2017-01-10 0.2180 0.2066
         2017-01-11 0.1990 0.1881
         2017-01-12 0.2060 0.1942

In [68]: data[['atr', 'atr%']].median() * leverage
Out[68]: atr    0.3180
         atr%   0.2481
         dtype: float64
```

Łączenie poziomów zleceń stop loss i take profit ze wskaźnikiem ATR wynika z tego, że należy unikać zbyt wysokich i zbyt niskich wartości. Rozważ pozycję z 10-krotną dźwignią, dla której wskaźnik ATR wynosi 20%. Ustawienie zlecenia stop loss na poziomie 3% lub 5% może zmniejszyć ryzyko, ale grozi zbyt szybkim aktywowaniem tego zlecenia z powodu ruchów typowych dla danego instrumentu. Takie „typowe ruchy" w określonym zakresie są często nazywane *szumem*. Zlecenia stop loss zwykle stosuje się do ochrony przed niekorzystnymi ruchami rynku, które są większe niż typowe ruchy cen (szum).

To samo dotyczy zleceń take profit. Jeśli ich poziom będzie zbyt wysoki (na przykład trzykrotność wskaźnika ATR), możliwe, że solidne zyski nie zostaną zabezpieczone, pozycja pozostanie otwarta zbyt długo, a wcześniejsze zyski zostaną utracone. Choć można tu zastosować formalne analizy i wzory matematyczne, ustawianie poziomów zleceń obronnych jest bardziej sztuką niż nauką. W finansach mamy dużą swobodę przy ustawianiu takich poziomów i często wykorzystuje się do tego ludzki osąd. W innych dziedzinach, na przykład w pojazdach autonomicznych, sytuacja wygląda inaczej, ponieważ nie jest potrzebny osąd człowieka, aby nakazać sztucznej inteligencji unikania kolizji z ludźmi.

#### Nienormalny rozkład i nieliniowe zależności

Mechanizm *margin stop out* powoduje zamknięcie pozycji w sytuacji, gdy depozyt (zainwestowana kwota) zostanie utracona. Załóżmy, że otwarta zostaje pozycja z dźwignią i mechanizmem margin stop out. Dla 10-krotnej dźwigni depozyt jest równy 10% wartości pozycji. *Niekorzystny* ruch na poziomie 10% lub większym powoduje utratę depozytu i skutkuje zamknięciem pozycji. Oznacza to utratę całego depozytu. *Korzystny* ruch na poziomie 25% prowadzi do stopy zwrotu z depozytu na poziomie 150%. Nawet jeśli stopy zwrotu z instrumentu mają rozkład normalny, dźwignia i mechanizm margin stop out skutkuje nienormalnym rozkładem stóp zwrotu i asymetryczną, nieliniową zależnością między instrumentem a pozycją.

# Testy historyczne zleceń obronnych

Znajomość wartości wskaźnika ATR instrumentu finansowego jest zwykle dobrym punktem wyjścia do zastosowania zleceń obronnych. Aby móc przeprowadzić testy historyczne typowych zleceń obronnych, warto wprowadzić kilka zmian w klasie `BacktestingBase`. W poniższym kodzie w Pythonie znajduje się nowa klasa bazowa, `BacktestBaseRM`, która dziedziczy po klasie `BacktestingBase` i pomaga

śledzić cenę otwarcia ostatniej pozycji oraz ceny maksymalną oraz minimalną od czasu jej otwarcia. Te wartości posłużą do obliczenia wyników strategii w trakcie bazujących na zdarzeniach testów z użyciem zleceń stop loss, trailing stop loss i take profit:

```
#
# Testy historyczne bazujące na zdarzeniach
# -- klasa bazowa (2)
#
# (c) Dr Yves J. Hilpisch
#
from backtesting import *

class BacktestingBaseRM(BacktestingBase):

    def set_prices(self, price):
        ''' Zapisywanie cen na potrzeby śledzenia wyników,
            na przykład w celu sprawdzania aktywowania zlecenia trailing stop loss.
        '''
        self.entry_price = price ❶
        self.min_price = price ❷
        self.max_price = price ❸

    def place_buy_order(self, bar, amount=None, units=None, gprice=None):
        ''' Składa zlecenia kupna dla danego słupka
            z określoną kwotą lub liczbą jednostek.
        '''
        date, price = self.get_date_price(bar)
        if gprice is not None:
            price = gprice
        if units is None:
            units = int(amount / price)
        self.current_balance -= (1 + self.ptc) * units * price + self.ftc
        self.units += units
        self.trades += 1
        self.set_prices(price) ❹
        if self.verbose:
            print(f'{date} | kupno {units} jednostek po {price:.4f}')
            self.print_balance(bar)

    def place_sell_order(self, bar, amount=None, units=None, gprice=None):
        ''' Składa zlecenie sprzedaży dla danego słupka
            z określoną kwotą lub liczbą jednostek.
        '''
        date, price = self.get_date_price(bar)
        if gprice is not None:
            price = gprice
        if units is None:
            units = int(amount / price)
        self.current_balance += (1 - self.ptc) * units * price - self.ftc
        self.units -= units
        self.trades += 1
        self.set_prices(price) ❹
        if self.verbose:
            print(f'{date} | sprzedaż {units} jednostek po {price:.4f}')
            self.print_balance(bar)
```

❶ Zapisywanie ceny *otwarcia* ostatniej pozycji.

❷ Ustawianie początkowej *minimalnej* ceny od momentu otwarcia ostatniej pozycji.

❸ Ustawianie początkowej *maksymalnej* ceny od momentu otwarcia ostatniej pozycji.

❹ Ustawianie odpowiednich cen po otwarciu pozycji.

W podrozdziale „Klasa do przeprowadzania testów historycznych" przedstawiona jest nowa klasa do przeprowadzania testów historycznych, `TBBacktesterRM`. Dziedziczy ona po pokazanej tu nowej klasie bazowej i umożliwia uwzględnianie zleceń stop loss, trailing stop loss i take profit. Ważne fragmenty kodu omawiam w dalszych punktach. W parametrach przykładowych testów historycznych używany jest wskaźnik ATR na poziomie mniej więcej 2%, zgodnie z obliczeniami z poprzedniego podrozdziału.

#### Teoria oczekiwanej użyteczności a zlecenia obronne

W teorii oczekiwanej użyteczności, modelu Markowitza i modelu wyceny dóbr kapitałowych (zobacz rozdziały 3. i 4.) zakłada się, że agent finansowy zna rozkład stóp zwrotu instrumentu finansowego. W nowoczesnej teorii portfelowej i modelu wyceny dóbr kapitałowych przyjmuje się ponadto, że jest to rozkład normalny i że występuje zależność liniowa między stopą zwrotu portfela a stopami zwrotu instrumentów finansowych. Stosowanie zleceń stop loss, trailing stop loss i take profit (podobnie jak korzystanie z dźwigni i mechanizmu margin stop out) skutkuje „gwarancją nienormalności" rozkładu i wysoce asymetrycznymi, nieliniowymi zyskami z pozycji względem zmian cen instrumentu.

## Zlecenia stop loss (SL)

Pierwszym zleceniem obronnym jest stop loss. Wymaga ono określenia stałego poziomu cenowego lub, częściej, stałej wartości procentowej, która powoduje zamknięcie pozycji. Na przykład jeśli cena otwarcia pozycji bez dźwigni wynosi 100, a poziom zlecenia stop loss to 5%, długa pozycja jest zamykana po osiągnięciu ceny 95, a krótka pozycja jest zamykana po osiągnięciu ceny 105.

Poniższy kod w Pythonie to ważna część klasy `TBBacktesterRM` obsługująca zlecenia stop loss. Klasa ta umożliwia określenie, czy cena z poziomu stop loss dla danego zlecenia jest gwarantowana, czy nie[2]. Używanie gwarantowanych cen dla zleceń stop loss może prowadzić do nadmiernie optymistycznych wyników:

```
# Zlecenie stop loss.
if sl is not None and self.position != 0: ❶
    rc = (price - self.entry_price) / self.entry_price ❷
    if self.position == 1 and rc < -self.sl: ❸
        print(50 * '-')
        if guarantee:
            price = self.entry_price * (1 - self.sl)
            print(f'*** STOP LOSS (DŁUGA POZYCJA | {-self.sl:.4f}) ***')
```

[2] *Gwarantowana* cena dla zleceń stop loss jest dostępna tylko w niektórych krajach dla określonych grup klientów domów maklerskich, na przykład dla inwestorów i graczy detalicznych.

```
            else:
                print(f'*** STOP LOSS (DŁUGA POZYCJA | {rc:.4f}) ***')
            self.place_sell_order(bar, units=self.units, gprice=price) ❹
            self.wait = wait ❺
            self.position = 0 ❻
        elif self.position == -1 and rc > self.sl: ❼
            print(50 * '-')
            if guarantee:
                price = self.entry_price * (1 + self.sl)
                print(f'*** STOP LOSS (KRÓTKA POZYCJA | -{self.sl:.4f}) ***')
            else:
                print(f'*** STOP LOSS (KRÓTKA POZYCJA | -{rc:.4f}) ***')
            self.place_buy_order(bar, units=-self.units, gprice=price) ❽
            self.wait = wait ❺
            self.position = 0 ❻
```

❶ Sprawdzanie, czy zlecenie stop loss jest zdefiniowane i czy pozycja nie jest neutralna.

❷ Obliczanie wyniku na podstawie ceny otwarcia ostatniej pozycji.

❸ Sprawdzanie, czy nastąpiło aktywowanie zlecenia stop loss dla *długiej* pozycji.

❹ Zamykanie *długiej* pozycji (albo przy aktualnym poziomie ceny, albo z gwarantowaną ceną).

❺ Ustawianie liczby słupków, jaką należy odczekać przed otwarciem następnej pozycji, na wartość `wait`.

❻ Ustawianie pozycji na neutralną.

❼ Sprawdzanie, czy nastąpiło aktywowanie zlecenia stop loss dla *krótkiej* pozycji.

❽ Zamykanie *krótkiej* pozycji (albo przy aktualnym poziomie ceny, albo z gwarantowaną ceną).

Poniższy kod w Pythonie przeprowadza testy historyczne strategii stosowanej przez bota handlowego ze zleceniami stop loss i bez takich zleceń. Przy użytych parametrach zlecenia stop loss mają negatywny wpływ na wyniki strategii:

```
In [69]: import tbbacktesterrm as tbbrm

In [70]: env = test_env

In [71]: tb = tbbrm.TBBacktesterRM(env, agent.model, 10000,
                                   0.0, 0, verbose=False) ❶

In [72]: tb.backtest_strategy(sl=None, tsl=None, tp=None, wait=5) ❷
         ================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ================================================
         ================================================
         2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
         2019-12-31 | stan gotówki = 10936.79
         2019-12-31 | wynik netto [%] = 9.3679
         2019-12-31 | liczba transakcji [#] = 186
================================================

In [73]: tb.backtest_strategy(sl=0.0175, tsl=None, tp=None,
                              wait=5, guarantee=False) ❸
         ================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
```

```
2018-01-17 | stan gotówki = 10000.00
================================================
------------------------------------------------
*** STOP LOSS (KRÓTKA POZYCJA | -0.0203) ***
================================================
2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
2019-12-31 | stan gotówki = 10717.32
2019-12-31 | wynik netto [%] = 7.1732
2019-12-31 | liczba transakcji [#] = 188
================================================

In [74]: tb.backtest_strategy(sl=0.017, tsl=None, tp=None,
                              wait=5, guarantee=True) ❹
================================================
2018-01-17 | *** POCZĄTEK TESTU ***
2018-01-17 | stan gotówki = 10000.00
================================================
------------------------------------------------
*** STOP LOSS (KRÓTKA POZYCJA | -0.0170) ***
================================================
2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
2019-12-31 | stan gotówki = 10753.52
2019-12-31 | wynik netto [%] = 7.5352
2019-12-31 | liczba transakcji [#] = 188
================================================
```

❶ Tworzenie instancji klasy do testów historycznych z obsługą zarządzania ryzykiem.

❷ Testy historyczne bota handlowego bez zleceń obronnych.

❸ Testy historyczne bota handlowego ze zleceniami stop loss (*bez* gwarancji ceny).

❹ Testy historyczne bota handlowego ze zleceniami stop loss (*z* gwarancją ceny).

## Zlecenia trailing stop loss (TSL)

W odróżnieniu od standardowego zlecenia stop loss zlecenie trailing stop loss jest modyfikowane każdorazowo po osiągnięciu nowego maksimum po otwarciu pozycji. Załóżmy, że cena otwarcia długiej pozycji bez dźwigni to 95, a trailing stop loss wynosi 5%. Jeśli cena instrumentu dojdzie do 100, a następnie spadnie do 95, aktywowane zostanie zlecenie trailing stop loss i pozycja zostanie zamknięta po cenie otwarcia. Także jeżeli cena dojdzie do 110, a następnie spadnie do 104,5, aktywowane zostanie zlecenie trailing stop loss.

Poniższy kod w Pythonie to fragment klasy `TBBacktesterRM` obsługujący zlecenia trailing stop loss. Aby poprawnie obsługiwać takie zlecenia, trzeba śledzić ceny maksymalne i minimalne. Cena maksymalna jest istotna dla pozycji długich, a cena minimalna dla pozycji krótkich:

```
# Zlecenia trailing stop loss.
if tsl is not None and self.position != 0:
    self.max_price = max(self.max_price, price) ❶
    self.min_price = min(self.min_price, price) ❷
    rc_1 = (price - self.max_price) / self.entry_price ❸
    rc_2 = (self.min_price - price) / self.entry_price ❹
    if self.position == 1 and rc_1 < -self.tsl: ❺
        print(50 * '-')
        print(f'*** TRAILING STOP LOSS (DŁUGA POZYCJA | {rc_1:.4f}) ***')
```

```
            self.place_sell_order(bar, units=self.units)
            self.wait = wait
            self.position = 0
        elif self.position == -1 and rc_2 < -self.tsl: ❻
            print(50 * '-')
            print(f'*** TRAILING STOP LOSS (KRÓTKA POZYCJA | {rc_2:.4f}) ***')
            self.place_buy_order(bar, units=-self.units)
            self.wait = wait
            self.position = 0
```

❶ Aktualizowanie ceny *maksymalnej*, gdy jest to konieczne.

❷ Aktualizowanie ceny *minimalnej*, gdy jest to konieczne.

❸ Obliczanie wyników dla *długiej* pozycji.

❹ Obliczanie wyników dla *krótkiej* pozycji.

❺ Sprawdzanie, czy nastąpiło aktywowanie zlecenia trailing stop loss dla *długiej* pozycji.

❻ Sprawdzanie, czy nastąpiło aktywowanie zlecenia trailing stop loss dla *krótkiej* pozycji.

Jak pokazują dalsze testy historyczne, stosowanie zleceń trailing stop loss przy używanych parametrach obniża wynik brutto w porównaniu ze strategią bez takich zleceń:

```
In [75]: tb.backtest_strategy(sl=None, tsl=0.015,
                              tp=None, wait=5) ❶
         ================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ================================================
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0152) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0169) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0164) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0191) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0166) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0194) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0172) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0181) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0153) ***
         ------------------------------------------------
         *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0160) ***
         ================================================
         2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
         2019-12-31 | stan gotówki = 10577.93
         2019-12-31 | wynik netto [%] = 5.7793
         2019-12-31 | liczba transakcji [#] = 201
         ================================================
```

❶ Testy historyczne bota handlowego ze zleceniami trailing stop loss.

## Zlecenia take profit (TP)

Istnieją też zlecenia take profit. Takie zlecenie powoduje zamknięcie pozycji po osiągnięciu określonego poziomu zysku. Załóżmy, że otwierana jest długa pozycja bez dźwigni w cenie 100, a zlecenie take profit wynosi 5%. W tym scenariuszu jeśli cena osiągnie 105, pozycja zostanie zamknięta.

Poniższy kod z klasy `TBBacktesterRM` to fragment odpowiedzialny za zlecenia take profit. Napisanie tego fragmentu jest proste, gdy znany jest już kod do obsługi zleceń stop loss i trailing stop loss. W analizie strategii ze zleceniami take profit można stosować ceny gwarantowane zamiast poziomów maksymalnego i minimalnego, które prawdopodobnie powodują uzyskanie zbyt optymistycznych wyników[3]:

```
# Zlecenia take profit.
if tp is not None and self.position != 0:
    rc = (price - self.entry_price) / self.entry_price
    if self.position == 1 and rc > self.tp:
        print(50 * '-')
        if guarantee:
            price = self.entry_price * (1 + self.tp)
            print(f'*** TAKE PROFIT (DŁUGA POZYCJA | {self.tp:.4f}) ***')
        else:
            print(f'*** TAKE PROFIT (DŁUGA POZYCJA | {rc:.4f}) ***')
        self.place_sell_order(bar, units=self.units, gprice=price)
        self.wait = wait
        self.position = 0
    elif self.position == -1 and rc < -self.tp:
        print(50 * '-')
        if guarantee:
            price = self.entry_price * (1 - self.tp)
            print(f'*** TAKE PROFIT (KRÓTKA POZYCJA | {self.tp:.4f}) ***')
        else:
            print(f'*** TAKE PROFIT (KRÓTKA POZYCJA | {-rc:.4f}) ***')
            self.place_buy_order(bar, units=-self.units, gprice=price)
            self.wait = wait
            self.position = 0
```

Przy zastosowanych parametrach zlecenia take profit (bez ceny gwarantowanej) znacznie poprawiają wyniki bota handlowego względem pasywnej inwestycji porównawczej. Jednak z powodu opisanych wcześniej kwestii rezultaty te mogą być zbyt optymistyczne. W tym scenariuszu bardziej realistyczne wyniki dadzą zlecenia take profit z ceną gwarantowaną:

```
In [76]: tb.backtest_strategy(sl=None, tsl=None, tp=0.015,
                              wait=5, guarantee=False) ❶
         ================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ================================================
         ------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0155) ***
         ------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0155) ***
         ------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0204) ***
```

[3] W zleceniu take profit określony jest stały poziom docelowy. Dlatego używanie ceny maksymalnej z określonego interwału dla długich pozycji lub ceny minimalnej dla krótkich pozycji do obliczania zysku jest nierealistyczne.

```
--------------------------------------------------
*** TAKE PROFIT (KRÓTKA POZYCJA | 0.0240) ***
--------------------------------------------------
*** TAKE PROFIT (KRÓTKA POZYCJA | 0.0168) ***
--------------------------------------------------
*** TAKE PROFIT (KRÓTKA POZYCJA | 0.0156) ***
--------------------------------------------------
*** TAKE PROFIT (KRÓTKA POZYCJA | 0.0183) ***
==================================================
2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
2019-12-31 | stan gotówki = 11210.33
2019-12-31 | wynik netto [%] = 12.1033
2019-12-31 | liczba transakcji [#] = 198
==================================================
```

```
In [77]: tb.backtest_strategy(sl=None, tsl=None, tp=0.015,
                              wait=5, guarantee=True) ❷
         ==================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ==================================================
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         --------------------------------------------------
         *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0150) ***
         ==================================================
         2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
         2019-12-31 | stan gotówki = 10980.86
         2019-12-31 | wynik netto [%] = 9.8086
         2019-12-31 | liczba transakcji [#] = 198
         ==================================================
```

❶ Testy historyczne bota handlowego ze zleceniami take profit (*bez* ceny gwarantowanej).

❷ Testy historyczne bota handlowego ze zleceniami take profit (*z* ceną gwarantowaną).

Oczywiście można łączyć zlecenia stop loss i trailing stop loss ze zleceniami take profit. Rezultaty testów historycznych z poniższego kodu w Pythonie pokazują, że w obu scenariuszach wyniki są gorsze niż dla strategii bez zleceń obronnych. W obszarze zarządzania ryzykiem nie ma nic za darmo:

```
In [78]: tb.backtest_strategy(sl=0.015, tsl=None,
                              tp=0.0185, wait=5) ❶
         ==================================================
         2018-01-17 | *** POCZĄTEK TESTU ***
         2018-01-17 | stan gotówki = 10000.00
         ==================================================
         --------------------------------------------------
         *** STOP LOSS (KRÓTKA POZYCJA | -0.0203) ***
```

```
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0202) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0213) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0240) ***
        -------------------------------------------------
        *** STOP LOSS (KRÓTKA POZYCJA | -0.0171) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0188) ***
        -------------------------------------------------
        *** STOP LOSS (KRÓTKA POZYCJA | -0.0153) ***
        -------------------------------------------------
        *** STOP LOSS (KRÓTKA POZYCJA | -0.0154) ***
        =================================================
        2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
        2019-12-31 | stan gotówki = 10552.00
        2019-12-31 | wynik netto [%] = 5.5200
        2019-12-31 | liczba transakcji [#] = 201
        =================================================

In [79]: tb.backtest_strategy(sl=None, tsl=0.02,
                              tp=0.02, wait=5) ❷
        =================================================
        2018-01-17 | *** POCZĄTEK TESTU ***
        2018-01-17 | stan gotówki = 10000.00
        =================================================
        -------------------------------------------------
        *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0235) ***
        -------------------------------------------------
        *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0202) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0250) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0227) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0240) ***
        -------------------------------------------------
        *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0216) ***
        -------------------------------------------------
        *** TAKE PROFIT (KRÓTKA POZYCJA | 0.0241) ***
        -------------------------------------------------
        *** TRAILING STOP LOSS (KRÓTKA POZYCJA | -0.0206) ***
        =================================================
        2019-12-31 | *** ZAMKNIĘCIE POZYCJI ***
        2019-12-31 | stan gotówki = 10346.38
        2019-12-31 | wynik netto [%] = 3.4638
        2019-12-31 | liczba transakcji [#] = 198
        =================================================
```

❶ Testy historyczne bota handlowego ze zleceniami stop loss i take profit.

❷ Testy historyczne bota handlowego ze zleceniami trailing stop loss i take profit.

**Wpływ na wyniki**

Zlecenia obronne mają swoje uzasadnienie i zalety. Jednak zmniejszenie ryzyka może skutkować niższym ogólnym wynikiem. Z kolei w przykładzie ze zleceniami take profit wyniki były wyższe, co da się wytłumaczyć tym, że przy określonej wartości wskaźnika ATR instrumentu finansowego dany poziom zysku można uznać za wystarczająco wysoki, aby zrealizować zyski. Nadzieja na jeszcze wyższe zyski jest zwykle rozwiewana przez zmianę kierunku rynku.

# Wnioski

W tym rozdziale poruszyłem trzy zagadnienia. Przeprowadziłem testy historyczne bota handlowego (agenta, który przeszedł uczenie głębokie z użyciem algorytmu Q-learning) dla danych spoza próbki w podejściu zwektoryzowanym i bazującym na zdarzeniach. Ponadto oceniłem ryzyko za pomocą wskaźnika ATR, który mierzy *typową* zmienność cen danego instrumentu finansowego. Na koniec omówiłem i przetestowałem (za pomocą podejścia bazującego na zdarzeniach) typowe zlecenia obronne: stop loss, trailing stop loss i take profit.

Boty handlowe, podobnie jak pojazdy autonomiczne, w praktyce prawie nigdy nie działają wyłącznie na podstawie predykcji generowanych przez sztuczną inteligencję. Aby uniknąć ryzyka dużych strat i poprawić wyniki skorygowane o ryzyko, zwykle stosuje się zlecenia obronne. Standardowe zlecenia obronne opisane w tym rozdziale są dostępne dla graczy detalicznych na prawie każdej platformie handlowej. W następnym rozdziale pokazuję, jak korzystać z nich na platformie Oanda (*http://oanda.com*). Testy historyczne bazujące na zdarzeniach zapewniają dużo swobody w stosowaniu algorytmów i umożliwiają poprawne zbadanie wpływu zleceń obronnych. Choć „ograniczenie ryzyka" może wydawać się atrakcyjne, wyniki testów historycznych pokazują, że często związane jest to z kosztami — wyniki mogą się okazać niższe w porównaniu z czystą strategią bez zleceń obronnych. Jednak wyniki pokazują też, że jeśli odpowiednio dostosować ustawienia, zlecenia take profit mogą mieć pozytywny wpływ na zyskowność.

# Literatura cytowana

Oto książki i prace przytaczane w tym rozdziale:

Agrawal A., Gans J. i Goldfarb A., *Prediction Machines: The Simple Economics of Artificial Intelligence*, Harvard Business Review Press, Boston 2018.

Hilpisch Y., *Python for Algorithmic Trading: From Idea to Cloud Deployment*, O'Reilly, Sebastopol 2020.

Khonji M., Dias J. i Seneviratne L., *Risk-Aware Reasoning for Autonomous Vehicles*, „arXiv", 6 października 2019. *https://oreil.ly/2Z6WR*.

# Kod w Pythonie

## Środowisko Finance

Oto moduł Pythona z klasą `Finance` reprezentująca środowisko:

```
#
# Środowisko Finance
#
# (c) Dr Yves J. Hilpisch
# Sztuczna inteligencja w finansach
#
import math
import random
import numpy as np
import pandas as pd

class observation_space:
    def __init__(self, n):
        self.shape = (n,)

class action_space:
    def __init__(self, n):
        self.n = n

    def sample(self):
        return random.randint(0, self.n - 1)

class Finance:
    intraday = False
    if intraday:
        url = 'http://hilpisch.com/aiif_eikon_id_eur_usd.csv'
    else:
        url = 'http://hilpisch.com/aiif_eikon_eod_data.csv'

    def __init__(self, symbol, features, window, lags,
                 leverage=1, min_performance=0.85, min_accuracy=0.5,
                 start=0, end=None, mu=None, std=None):
        self.symbol = symbol
        self.features = features
        self.n_features = len(features)
        self.window = window
        self.lags = lags
        self.leverage = leverage
        self.min_performance = min_performance
        self.min_accuracy = min_accuracy
        self.start = start
        self.end = end
        self.mu = mu
        self.std = std
        self.observation_space = observation_space(self.lags)
        self.action_space = action_space(2)
        self._get_data()
        self._prepare_data()

    def _get_data(self):
        self.raw = pd.read_csv(self.url, index_col=0,
```

```
                            parse_dates=True).dropna()
        if self.intraday:
            self.raw = self.raw.resample('30min', label='right').last()
            self.raw = pd.DataFrame(self.raw['CLOSE'])
            self.raw.columns = [self.symbol]

    def _prepare_data(self):
        self.data = pd.DataFrame(self.raw[self.symbol])
        self.data = self.data.iloc[self.start:]
        self.data['r'] = np.log(self.data / self.data.shift(1))
        self.data.dropna(inplace=True)
        self.data['s'] = self.data[self.symbol].rolling(self.window).mean()
        self.data['m'] = self.data['r'].rolling(self.window).mean()
        self.data['v'] = self.data['r'].rolling(self.window).std()
        self.data.dropna(inplace=True)
        if self.mu is None:
            self.mu = self.data.mean()
            self.std = self.data.std()
        self.data_ = (self.data - self.mu) / self.std
        self.data['d'] = np.where(self.data['r'] > 0, 1, 0)
        self.data['d'] = self.data['d'].astype(int)
        if self.end is not None:
            self.data = self.data.iloc[:self.end - self.start]
            self.data_ = self.data_.iloc[:self.end - self.start]

    def _get_state(self):
        return self.data_[self.features].iloc[self.bar -
                                              self.lags:self.bar]

    def get_state(self, bar):
        return self.data_[self.features].iloc[bar - self.lags:bar]

    def seed(self, seed):
        random.seed(seed)
        np.random.seed(seed)

    def reset(self):
        self.treward = 0
        self.accuracy = 0
        self.performance = 1
        self.bar = self.lags
        state = self.data_[self.features].iloc[self.bar -
                                               self.lags:self.bar]
        return state.values

    def step(self, action):
        correct = action == self.data['d'].iloc[self.bar]
        ret = self.data['r'].iloc[self.bar] * self.leverage
        reward_1 = 1 if correct else 0
        reward_2 = abs(ret) if correct else -abs(ret)
        self.treward += reward_1
        self.bar += 1
        self.accuracy = self.treward / (self.bar - self.lags)
        self.performance *= math.exp(reward_2)
        if self.bar >= len(self.data):
            done = True
        elif reward_1 == 1:
            done = False
```

```
        elif (self.performance < self.min_performance and
              self.bar > self.lags + 15):
            done = True
        elif (self.accuracy < self.min_accuracy and
              self.bar > self.lags + 15):
            done = True
        else:
            done = False
        state = self._get_state()
        info = {}
        return state.values, reward_1 + reward_2 * 5, done, info
```

## Bot handlowy

Tu przedstawiony jest moduł Pythona z klasą `TradingBot` bazującą na agencie finansowym uczonym za pomocą algorytmu Q-learning:

```
#
# Agent uczony za pomocą algorytmu Q-learning
#
# (c) Dr Yves J. Hilpisch
# Sztuczna inteligencja w finansach
#
import os
import random
import numpy as np
from pylab import plt, mpl
from collections import deque
import tensorflow as tf
from keras.layers import Dense, Dropout
from keras.models import Sequential
from keras.optimizers import Adam, RMSprop

os.environ['PYTHONHASHSEED'] = '0'
plt.style.use('seaborn')
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['font.family'] = 'serif'

def set_seeds(seed=100):
    ''' Funkcja konfigurująca ziarno dla
        wszystkich generatorów liczb losowych.
    '''
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

class TradingBot:
    def __init__(self, hidden_units, learning_rate, learn_env,
                 valid_env=None, val=True, dropout=False):
        self.learn_env = learn_env
        self.valid_env = valid_env
        self.val = val
        self.epsilon = 1.0
        self.epsilon_min = 0.1
        self.epsilon_decay = 0.99
        self.learning_rate = learning_rate
        self.gamma = 0.5
```

```
        self.batch_size = 128
        self.max_treward = 0
        self.averages = list()
        self.trewards = []
        self.performances = list()
        self.aperformances = list()
        self.vperformances = list()
        self.memory = deque(maxlen=2000)
        self.model = self._build_model(hidden_units,
                                        learning_rate, dropout)

    def _build_model(self, hu, lr, dropout):
        ''' Metoda tworząca nowy model gęstej sieci neuronowej.
        '''
        model = Sequential()
        model.add(Dense(hu, input_shape=(
            self.learn_env.lags, self.learn_env.n_features),
            activation='relu'))
        if dropout:
            model.add(Dropout(0.3, seed=100))
        model.add(Dense(hu, activation='relu'))
        if dropout:
            model.add(Dropout(0.3, seed=100))
        model.add(Dense(2, activation='linear'))
        model.compile(
            loss='mse',
            optimizer=RMSprop(lr=lr)
        )
        return model

    def act(self, state):
        ''' Metoda do podejmowania działań na podstawie
            a) eksploracji,
            b) eksploatacji.
        '''
        if random.random() <= self.epsilon:
            return self.learn_env.action_space.sample()
        action = self.model.predict(state)[0, 0]
        return np.argmax(action)

    def replay(self):
        ''' Metoda ucząca gęstą sieć neuronową
            na podstawie porcji zapamiętanych doświadczeń.
        '''
        batch = random.sample(self.memory, self.batch_size)
        for state, action, reward, next_state, done in batch:
            if not done:
                reward += self.gamma * np.amax(
                    self.model.predict(next_state)[0, 0])
            target = self.model.predict(state)
            target[0, 0, action] = reward
            self.model.fit(state, target, epochs=1,
                           verbose=False)
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

    def learn(self, episodes):
        ''' Metoda do uczenia agenta DQL.
        '''
```

```
        for e in range(1, episodes + 1):
            state = self.learn_env.reset()
            state = np.reshape(state, [1, self.learn_env.lags,
                                       self.learn_env.n_features])
            for _ in range(10000):
                action = self.act(state)
                next_state, reward, done, info = self.learn_env.step(action)
                next_state = np.reshape(next_state,
                                        [1, self.learn_env.lags,
                                        self.learn_env.n_features])
                self.memory.append([state, action, reward,
                                    next_state, done])
                state = next_state
                if done:
                    treward = _ + 1
                    self.trewards.append(treward)
                    av = sum(self.trewards[-25:]) / 25
                    perf = self.learn_env.performance
                    self.averages.append(av)
                    self.performances.append(perf)
                    self.aperformances.append(
                        sum(self.performances[-25:]) / 25)
                    self.max_treward = max(self.max_treward, treward)
                    templ = 'epizod: {:2d}/{} | nag. łącz.: {:4d} | '
                    templ += 'wynik: {:5.3f} | średnia: {:5.1f} | maks.: {:4d}'
                    print(templ.format(e, episodes, treward, perf,
                                       av, self.max_treward), end='\r')
                    break
            if self.val:
                self.validate(e, episodes)
            if len(self.memory) > self.batch_size:
                self.replay()
        print()

    def validate(self, e, episodes):
        ''' Metoda do sprawdzania skuteczności
            agenta DQL.
        '''
        state = self.valid_env.reset()
        state = np.reshape(state, [1, self.valid_env.lags,
                                   self.valid_env.n_features])
        for _ in range(10000):
            action = np.argmax(self.model.predict(state)[0, 0])
            next_state, reward, done, info = self.valid_env.step(action)
            state = np.reshape(next_state, [1, self.valid_env.lags,
                                            self.valid_env.n_features])
            if done:
                treward = _ + 1
                perf = self.valid_env.performance
                self.vperformances.append(perf)
                if e % int(episodes / 6) == 0:
                    templ = 71 * '='
                    templ += '\nepizod: {:2d}/{} | WALIDACJA | '
                    templ += nag. łącz.: {:4d} | wynik: {:5.3f} | eps.: {:.2f}\n'
                    templ += 71 * '='
                    print(templ.format(e, episodes, treward,
                                       perf, self.epsilon))
                break

def plot_treward(agent):
```

```
    ''' Funkcja wyświetlająca łączną nagrodę
        za epizod w procesie uczenia.
    '''
    plt.figure(figsize=(10, 6))
    x = range(1, len(agent.averages) + 1)
    y = np.polyval(np.polyfit(x, agent.averages, deg=3), x)
    plt.plot(x, agent.averages, label='średnia krocząca')
    plt.plot(x, y, 'r--', label='trend')
    plt.xlabel('liczba epizodów')
    plt.ylabel('nagroda łączna')
    plt.legend()

def plot_performance(agent):
    ''' Funkcja wyświetlająca wynik finansowy brutto
        z epizodu w procesie uczenia.
    '''
    plt.figure(figsize=(10, 6))
    x = range(1, len(agent.performances) + 1)
    y = np.polyval(np.polyfit(x, agent.performances, deg=3), x)
    plt.plot(x, agent.performances[:], label='uczenie')
    plt.plot(x, y, 'r--', label='trend (uczenie)')
    if agent.val:
        y_ = np.polyval(np.polyfit(x, agent.vperformances, deg=3), x)
        plt.plot(x, agent.vperformances[:], label='walidacja')
        plt.plot(x, y_, 'r-.', label='trend (walidacja)')
    plt.xlabel('liczba epizodów')
    plt.ylabel('wynik brutto')
    plt.legend()
```

## Klasa BacktestingBase

Poniżej pokazany jest moduł Pythona z klasą `BacktestingBase` do przeprowadzania testów historycznych bazujących na zdarzeniach:

```
#
# Testy historyczne bazujące na zdarzeniach
# -- klasa bazowa (1)
#
# (c) Dr Yves J. Hilpisch
# Sztuczna inteligencja w finansach
#

class BacktestingBase:
    def __init__(self, env, model, amount, ptc, ftc, verbose=False):
        self.env = env ❶
        self.model = model ❷
        self.initial_amount = amount ❸
        self.current_balance = amount ❸
        self.ptc = ptc ❹
        self.ftc = ftc ❺
        self.verbose = verbose ❻
        self.units = 0 ❼
        self.trades = 0 ❽

    def get_date_price(self, bar):
        ''' Zwraca datę i cenę dla danego słupka.
        '''
```

```
        date = str(self.env.data.index[bar])[:10] ❾
        price = self.env.data[self.env.symbol].iloc[bar] ❿
        return date, price

    def print_balance(self, bar):
        ''' Wyświetla aktualny stan gotówki dla danego słupka.
        '''
        date, price = self.get_date_price(bar)
        print(f'{date} | stan gotówki = {self.current_balance:.2f}') ⓫

    def calculate_net_wealth(self, price):
        return self.current_balance + self.units * price ⓬

    def print_net_wealth(self, bar):
        ''' Wyświetla majątek netto dla danego słupka
            (gotówkę + wartość pozycji).
        '''
        date, price = self.get_date_price(bar)
        net_wealth = self.calculate_net_wealth(price)
        print(f'{date} | majątek netto = {net_wealth:.2f}') ⓭

    def place_buy_order(self, bar, amount=None, units=None):
        ''' Składa zlecenie kupna dla danego słupka
            z określoną kwotą lub liczbą jednostek.
        '''
        date, price = self.get_date_price(bar)
        if units is None:
            units = int(amount / price) ⓮
            # units = amount / price ⓮
        self.current_balance -= (1 + self.ptc) * \
            units * price + self.ftc ⓯
        self.units += units ⓰
        self.trades += 1 ⓱
        if self.verbose:
            print(f'{date} | kupno {units} jednostek po {price:.4f}')
            self.print_balance(bar)

    def place_sell_order(self, bar, amount=None, units=None):
        ''' Składa zlecenie sprzedaży dla danego słupka
            z określoną kwotą lub liczbą jednostek.
        '''
        date, price = self.get_date_price(bar)
        if units is None:
            units = int(amount / price) ⓮
            # units = amount / price ⓮
        self.current_balance += (1 - self.ptc) * \
            units * price - self.ftc ⓯
        self.units -= units ⓰
        self.trades += 1 ⓱
        if self.verbose:
            print(f'{date} | sprzedaż {units} jednostek po {price:.4f}')
            self.print_balance(bar)

    def close_out(self, bar):
        ''' Zamknięcie otwartej pozycji w danym słupku.
        '''
        date, price = self.get_date_price(bar)
        print(50 * '=')
```

```
        print(f'{date} | *** ZAMKNIĘCIE POZYCJI ***')
        if self.units < 0:
            self.place_buy_order(bar, units=-self.units) ⓲
        else:
            self.place_sell_order(bar, units=self.units) ⓳
        if not self.verbose:
            print(f'{date} | stan gotówki = {self.current_balance:.2f}')
        perf = (self.current_balance / self.initial_amount - 1) * 100 ⓴
        print(f'{date} | wynik netto [%] = {perf:.4f}')
        print(f'{date} | liczba transakcji [#] = {self.trades}')
        print(50 * '=')
```

❶ Odpowiednie środowisko `Finance`.

❷ Używany model gęstej sieci neuronowej (bota handlowego).

❸ Początkowy i aktualny stan gotówki.

❹ Proporcjonalnie naliczane koszty transakcyjne.

❺ Stałe koszty transakcyjne.

❻ Czy należy wyświetlać rozbudowane informacje?

❼ Początkowa liczba jednostek instrumentu finansowego.

❽ Początkowa liczba transakcji.

❾ *Data* odpowiadająca danemu słupkowi.

❿ *Cena instrumentu* dla danego słupka.

⓫ Wyświetlanie *daty* i *aktualnego stanu gotówki* dla określonego słupka.

⓬ Obliczanie wartości *majątku netto* na podstawie aktualnego stanu gotówki i wartości pozycji.

⓭ Wyświetlanie *daty* i *majątku netto* dla danego słupka.

⓮ Liczba jednostek kupowanych lub sprzedawanych za daną kwotę.

⓯ Wpływ transakcji i powiązanych kosztów na aktualny stan gotówki.

⓰ Zmiana liczby jednostek w pozycji.

⓱ Zmiana liczby przeprowadzonych transakcji.

⓲ Zamykanie *krótkiej* pozycji…

⓳ …lub *długiej* pozycji.

⓴ Wynik netto obliczany na podstawie początkowej kwoty i końcowego stanu gotówki.

## Klasa do przeprowadzania testów historycznych

Poniżej przedstawiam moduł Pythona z klasą `TBBacktesterRM` do przeprowadzania testów historycznych bazujących na zdarzeniach z uwzględnieniem zleceń obronnych (stop loss, trailing stop loss i take profit):

```
#
# Testy historyczne bazujące na zdarzeniach
# -- klasa do testowania bota handlowego
```

```
# (z uwzględnieniem zarządzania ryzykiem)
#
# (c) Dr Yves J. Hilpisch
#
import numpy as np
import pandas as pd
import backtestingrm as btr

class TBBacktesterRM(btr.BacktestingBaseRM):
    def _reshape(self, state):
        ''' Metoda pomocnicza do zmiany kształtu obiektów ze stanem.
        '''
        return np.reshape(state, [1, self.env.lags, self.env.n_features])

    def backtest_strategy(self, sl=None, tsl=None, tp=None,
                          wait=5, guarantee=False):
        ''' Bazujące na zdarzeniach testy historyczne wyników bota handlowego.
            Uwzględniane są zlecenia stop loss, trailing stop loss i take profit.
        '''
        self.units = 0
        self.position = 0
        self.trades = 0
        self.sl = sl
        self.tsl = tsl
        self.tp = tp
        self.wait = 0
        self.current_balance = self.initial_amount
        self.net_wealths = list()
        for bar in range(self.env.lags, len(self.env.data)):
            self.wait = max(0, self.wait - 1)
            date, price = self.get_date_price(bar)
            if self.trades == 0:
                print(50 * '=')
                print(f'{date} | *** POCZĄTEK TESTU ***')
                self.print_balance(bar)
                print(50 * '=')

            # Zlecenie stop loss.
            if sl is not None and self.position != 0:
                rc = (price - self.entry_price) / self.entry_price
                if self.position == 1 and rc < -self.sl:
                    print(50 * '-')
                    if guarantee:
                        price = self.entry_price * (1 - self.sl)
                        print(f'*** STOP LOSS (DŁUGA POZYCJA | {-self.sl:.4f}) ***')
                    else:
                        print(f'*** STOP LOSS (DŁUGA POZYCJA | {rc:.4f}) ***')
                    self.place_sell_order(bar, units=self.units, gprice=price)
                    self.wait = wait
                    self.position = 0
                elif self.position == -1 and rc > self.sl:
                    print(50 * '-')
                    if guarantee:
                        price = self.entry_price * (1 + self.sl)
                        print(f'*** STOP LOSS (KRÓTKA POZYCJA | -{self.sl:.4f}) ***')
                    else:
                        print(f'*** STOP LOSS (KRÓTKA POZYCJA | -{rc:.4f}) ***')
                    self.place_buy_order(bar, units=-self.units, gprice=price)
```

```
                self.wait = wait
                self.position = 0

            # Zlecenie trailing stop loss.
            if tsl is not None and self.position != 0:
                self.max_price = max(self.max_price, price)
                self.min_price = min(self.min_price, price)
                rc_1 = (price - self.max_price) / self.entry_price
                rc_2 = (self.min_price - price) / self.entry_price
                if self.position == 1 and rc_1 < -self.tsl:
                    print(50 * '-')
                    print(f'*** TRAILING STOP LOSS (DŁUGA POZYCJA | {rc_1:.4f}) ***')
                    self.place_sell_order(bar, units=self.units)
                    self.wait = wait
                    self.position = 0
                elif self.position == -1 and rc_2 < -self.tsl:
                    print(50 * '-')
                    print(f'*** TRAILING STOP LOSS (KRÓTKA POZYCJA | {rc_2:.4f}) ***')
                    self.place_buy_order(bar, units=-self.units)
                    self.wait = wait
                    self.position = 0

            # Zlecenie take profit.
            if tp is not None and self.position != 0:
                rc = (price - self.entry_price) / self.entry_price
                if self.position == 1 and rc > self.tp:
                    print(50 * '-')
                    if guarantee:
                        price = self.entry_price * (1 + self.tp)
                        print(f'*** TAKE PROFIT (DŁUGA POZYCJA | {self.tp:.4f}) ***')
                    else:
                        print(f'*** TAKE PROFIT (DŁUGA POZYCJA | {rc:.4f}) ***')
                        self.place_sell_order(bar, units=self.units, gprice=price)
                        self.wait = wait
                        self.position = 0
                    elif self.position == -1 and rc < -self.tp:
                        print(50 * '-')
                        if guarantee:
                            price = self.entry_price * (1 - self.tp)
                            print(f'*** TAKE PROFIT (KRÓTKA POZYCJA | {self.tp:.4f})\
***')
                        else:
                            print(f'*** TAKE PROFIT (KRÓTKA POZYCJA | {-rc:.4f}) ***')
                        self.place_buy_order(bar, units=-self.units, gprice=price)
                        self.wait = wait
                        self.position = 0

                state = self.env.get_state(bar)
                action = np.argmax(self.model.predict(
                    self._reshape(state.values))[0, 0])
                position = 1 if action == 1 else -1
                if self.position in [0, -1] and position == 1 and self.wait == 0:
                    if self.verbose:
                        print(50 * '-')
                        print(f'{date} | *** DŁUGA POZYCJA ***')
                    if self.position == -1:
                        self.place_buy_order(bar - 1, units=-self.units)
                    self.place_buy_order(bar - 1, amount=self.current_balance)
```

```
            if self.verbose:
                self.print_net_wealth(bar)
            self.position = 1
        elif self.position in [0, 1] and position == -1 and self.wait == 0:
            if self.verbose:
                print(50 * '-')
                print(f'{date} | *** KRÓTKA POZYCJA ***')
            if self.position == 1:
                self.place_sell_order(bar - 1, units=self.units)
            self.place_sell_order(bar - 1, amount=self.current_balance)
            if self.verbose:
                self.print_net_wealth(bar)
            self.position = -1
        self.net_wealths.append((date, self.calculate_net_wealth(price)))
    self.net_wealths = pd.DataFrame(self.net_wealths,
                                    columns=['date', 'net_wealth'])
    self.net_wealths.set_index('date', inplace=True)
    self.net_wealths.index = pd.DatetimeIndex(self.net_wealths.index)
    self.close_out(bar)
```

# Skorowidz

## E



## F



## G



## H



## I



## K



## L



## M

## N



## O



## P

## Q



## R



## S



## Ś

## T



## U



## V



## W



## X



## Z

# Poznaj zastosowania AI w branży finansowej

W świecie finansów sztuczna inteligencja okazała się przełomową technologią — w połączeniu z odpowiednim zastosowaniem algorytmów i dużych zbiorów danych bowiem pozwala na poprawę jakości usług finansowych. Autor tej książki zdaje sobie z tego sprawę — ma wieloletnie doświadczenie i kompleksową wiedzę na temat projektowania i wdrażania zaawansowanych mechanizmów AI w największych podmiotach z branży. Swoją wiedzą dzieli się z Czytelnikami.

Dr Yves Hilpisch szczegółowo opisuje zarówno podstawy teoretyczne, jak i praktyczne aspekty używania algorytmów sztucznej inteligencji w ramach usług i produktów finansowych. Opierając się na przykładach z języka Python, pokazuje metodyki, modele, założenia i techniki wdrażania AI, a także analizuje problemy mogące utrudniać to zadanie i przybliża ich rozwiązania. Znajdziemy tutaj skomplikowane zagadnienia wytłumaczone w logiczny i zrozumiały sposób. Autor z powodzeniem łączy teorię z praktyką, a jego podejście do tematu i prezentowane przypadki bazujące na doświadczeniu są cennym źródłem wiedzy dla każdego, kto chce poznać tajniki dotyczące zastosowania sztucznej inteligencji, uczenia maszynowego, algorytmów i zbiorów danych w szeroko pojętym świecie finansów.

## Dzięki książce dowiesz się:

- na czym polega zastosowanie AI w usługach i produktach finansowych
- dlaczego i w jaki sposób użycie sztucznej inteligencji fundamentalnie zmienia sektor finansowy i jakie ma to skutki dla niego i konsumentów
- jak w języku Python konstruować i wdrażać algorytmy bazujące na rozbudowanych zbiorach danych
- jak dzięki AI i uczeniu maszynowemu usprawniać usługi i produkty finansowe

**Dr Yves Hilpisch** — właściciel firm The Python Quants i The AI Machine specjalizujących się w projektowaniu i we wdrażaniu mechanizmów algorytmicznych, sztucznej inteligencji i uczenia maszynowego przy użyciu języka Python. Autor kilku książek analizujących zastosowanie tego języka w biznesie i profesor kontraktowy finansów obliczeniowych.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl



KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-8893-2
9 788328 388932
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 99,00 zł