POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Oprogramowania

Platforma predykcji i monitorowania danych zmiennych w czasie z użyciem metod uczenia maszynowego

Platform for prediction and monitoring of time-varying data using machine learning methods

Mykola Lutsyk

99799

Promotor dr Paweł Powróźnik

Lublin rok 2026

Spis treści

Streszczenie

Niniejsza praca dyplomowa poświęcona jest projektowi i implementacji platformy do prognozowania cen kryptowalut w oparciu o metody uczenia maszynowego. Głównym celem systemu jest śledzenie danych z zewnętrznych API (np. giełd kryptowalutowych), analiza trendów przy użyciu wskaźników technicznych (analiza techniczna), a także generowanie predykcji za pomocą modeli ML. Platforma udostępnia wyniki poprzez interfejs webowy, umożliwiając użytkownikom monitorowanie aktualnych danych i prognozowanych trendów. Dane historyczne są efektywnie zarządzane w zoptymalizowanej bazie danych szeregów czasowych. Przeprowadzone testy potwierdzają przydatność systemu w wspomaganiu decyzji inwestycyjnych, osiągając przewyższającą założony próg użytkowy.

Słowa kluczowe: uczenie maszynowe, predykcja cen kryptowalut, analiza techniczna, aplikacja webowa.

Abstract

This engineering thesis focuses on the design and implementation of a cryptocurrency price prediction platform utilizing machine learning methods. The system's primary objective is to track data from external APIs (e.g., cryptocurrency exchanges), analyze trends using technical indicators (technical analysis), and generate forecasts via ML models. The platform provides results through a web interface, allowing users to monitor real-time data and predicted trends. Historical data is efficiently managed in an optimized time-series database. Conducted tests confirm the system's usefulness in supporting investment decisions, achieving performance that exceeds the assumed utility threshold.

Key words: *machine learning, cryptocurrency price prediction, technical analysis, web application*.

1. Wstęp

1.1. Kontekst i motywacja

W erze gospodarki opartej na danych, analiza szeregów czasowych stała się istotnym narzędziem wspierającym podejmowanie decyzji w różnych dziedzinach życia. Tradycyjne techniki zwykle nie potrafią skutecznie modelować skomplikowanych i zmieniających się w czasie zjawisk [1].

Celem tej pracy jest stworzenie i wprowadzenie w życie systemu informatycznego do przewidywania oraz obserwacji danych zmieniających się w czasie, z zastosowaniem technik uczenia maszynowego. Podstawowym celem systemu jest pomoc użytkownikowi w badaniu przyszłych wartości w oparciu o dane z przeszłości.

Platforma pozwoli na importowanie danych, a także na tworzenie, weryfikację i wizualizację rezultatów prognoz. Efektywność stworzonych modeli będzie oceniana na podstawie rzeczywistych zbiorów danych.

Praca dotyczy badania technologii uczenia maszynowego, opracowania oraz wdrożenia systemu, łączenia jego elementów, a także weryfikacji oraz przedstawienia rezultatów.

1.2. Cel pracy

Celem pracy jest opracowanie i implementacja platformy informatycznej umożliwiającej predykcję oraz monitorowanie danych zmiennych w czasie z użyciem metod uczenia maszynowego, szczególnie kryptowalut. Platforma ma wspierać analizę trendów oraz umożliwiać prognozowanie przyszłych wartości na podstawie historycznych danych.

1.3.Zakres pracy

Zakres pracy obejmuje:

- analizę metod uczenia maszynowego stosowanych do modelowania szeregów czasowych;
- projekt i implementację platformy umożliwiającej import oraz analizę danych;
- opracowanie modeli predykcyjnych i ich parametrów;
- walidację skuteczności modeli na rzeczywistych danych oraz prezentację wyników w formie interaktywnego panelu użytkownika.

2. Analiza rynku i przegląd rozwiązań

Rynek narzędzi przeznaczonych na analizę i prognozę danych finansowych, w tym także dotyczący kryptowalut, jest dynamicznie rozwijający się. Dostępne narzędzia oferują analizę techniczną i usługi wspomagające proces przewidywania trendów, bazując na danych z przeszłości. W ostatnich latach zauważono rosnące zainteresowanie wykorzystaniem technik uczenia maszynowego w badaniach danych rynkowych, co pozwala na tworzenie bardziej precyzyjnych i elastycznych modeli prognozujących.

W ramach tej pracy ważne jest zestawienie dostępnych rozwiązań, które posiadają funkcje podobne do projektowanej w niniejszej pracy platformy, aby zidentyfikować obszary, które projekt powinien uzupełnić.

2.1. Platforma TradingView

TradingView jest jedną z najbardziej znanych platform do analizy rynku finansowego, która zawiera akcje, indeksy, surowce oraz kryptowaluty. Pozwala na interaktywną analizę danych w czasie rzeczywistym, prezentację wykresów oraz wykorzystanie narzędzi analizy technicznej. Użytkownicy mają możliwość tworzenia własnych programów w języku Pine Script, co umożliwia opracowywanie unikalnych wskaźników i strategii.

Funkcje:

- dostęp do obszernego zbioru danych zarówno historycznych, jak i aktualnych;
- analiza techniczna z zastosowaniem wskaźników oraz narzędzi wizualnych;
- prezentacja danych w czasie rzeczywistym;
- społeczność użytkowników, która wymienia się analizami oraz strategiami;
- integracja z rynkami kryptowalut.

Ograniczenia:

- brak wbudowanej integracji z modelami uczenia maszynowego, ograniczone opcje oceniania modeli predykcyjnych;
- prognozy bazują się na analizie technicznej, a nie na modelach sztucznej inteligencji dostosowujących się do zmieniających się warunków.

2.2. Serwis WalletInvestor

WalletInvestor to platforma internetowa, która koncentruje się na automatycznym tworzeniu przewidywań cenowych dla kryptowalut, akcji oraz surowców. Prognozy są opracowywane na podstawie danych z przeszłości z zastosowaniem modeli statystycznych oraz technologii uczenia maszynowego.

Funkcje:

- automatyczne przewidywanie cen na podstawie danych z przeszłości;
- prognozy na krótki i długi okres;
- dostęp do diagramów oraz informacji o tendencjach;
- interfejs dostępny w sieci, nie wymagający instalacji oprogramowania.

Ograniczenia:

- niedobór jasności dotyczącej wykorzystywanych metod i algorytmów;
- ograniczone opcje interakcji z modelem;
- brak dostosowywania parametrów wejściowych;
- podsumowanie wyników analizy rynku.

Na podstawie analizy można zauważyć, że dostępne platformy koncentrują się przeważnie na przedstawianiu wyników prognoz lub analizie technicznej, natomiast brakuje narzędzi, które pozwalałyby użytkownikom na tworzenie, testowanie i porównywanie modeli uczenia maszynowego na ich własnych danych.

Opracowana w ramach tej pracy platforma zaspokaja tę potrzebę, proponując:

- interaktywny dobór właściwości oraz ram czasowych;
- szkolenie i ocenę modeli uczenia maszynowego z opcją potwierdzenia ich efektywności, wyświetlenie wyników prognoz w przyjaznym dla użytkownika interfejsie.

Opracowana platforma łączy cechy charakterystyczne dla serwisów analitycznych takich jak *TradingView* z elastycznością stosowaną w badaniach związanych z uczeniem maszynowym.

3. Projekt systemu

3.1. Wymagania funkcjonalne

System predykcji i monitorowania danych zmiennych w czasie składa się z czterech głównych elementów: modułu gromadzenia i przetwarzania danych, modułu predykcyjnego, serwera aplikacyjnego oraz interfejsu użytkownika. Moduł gromadzenia danych jest odpowiedzialny za pobieranie, przetwarzanie oraz przygotowanie danych historycznych. Moduł predykcyjny realizuje proces uczenia modeli oraz generowania prognoz. Serwer aplikacyjny odpowiada za komunikację pomiędzy poszczególnymi elementami systemu, natomiast interfejs użytkownika umożliwia prezentację wyników działania systemu.

Wymagania funkcjonalne do modułu gromadzenia i przetwarzania danych:

- system powinien umożliwiać automatyczne pobieranie danych historycznych z zewnętrznych źródeł danych;
- system powinien umożliwiać zapisywanie pobranych danych w bazie danych, obejmujące czyszczenie oraz normalizację danych;
- moduł powinien wykonywać proces inżynierii cech na podstawie danych historycznych;
- moduł powinien przygotowywać dane wejściowe do procesu predykcji;
- w przypadku wystąpienia błędów przetwarzania system powinien informować o zaistniałym problemie.

Wymagania funkcjonalne do modułu predykcyjnego:

- system powinien umożliwiać trenowanie modelu predykcyjnego na podstawie danych historycznych;
- moduł predykcyjny powinien wykorzystywać wcześniej wytrenowany model do generowania prognoz;
- system powinien umożliwiać generowanie predykcji przyszłych wartości zmiennych czasowych;
- moduł powinien obliczać metryki oceny jakości predykcji.

Wymagania funkcjonalne do serwera aplikacyjnego:

- serwer aplikacyjny powinien udostępniać interfejs API umożliwiający komunikację z interfejsem użytkownika;
- system powinien umożliwiać pobieranie danych historycznych oraz wyników predykcji za pomocą API;
- serwer powinien zapewniać poprawną obsługę błędów po stronie serweru;
- serwer aplikacyjny powinien umożliwiać uruchamianie procesu predykcji.

Wymagania funkcjonalne do interfejsu użytkownika:

- interfejs użytkownika powinien umożliwiać prezentację danych historycznych w formie wykresów;
- aplikacja powinna umożliwiać prezentację wyników predykcji;
- interfejs użytkownika powinien wizualizować metryki jakości modelu;
- aplikacja powinna umożliwiać monitorowanie zmian wartości zmiennych w czasie;
- interfejs użytkownika powinien informować użytkownika o błędach systemowych.

3.2.Wymagania niefunkcjonalne

Wymagania niefunkcjonalne określają cechy oraz ograniczenia systemu, które nie są bezpośrednio związane z jego funkcjonalnością, lecz mają istotny wpływ na jakość, wydajność oraz niezawodność opracowanego rozwiązania.

Wymagania niefunkcjonalne do modułu gromadzenia i przetwarzania danych:

- moduł powinien działać w sposób automatyczny, bez konieczności ingerencji użytkownika;
- moduł powinien zapewniać spójność oraz poprawność danych wejściowych;
- system powinien umożliwiać ponowne przetworzenie danych w przypadku;
- moduł powinien działać w środowisku obsługującym język *Python* 3.9 lub wyżej.

Wymagania niefunkcjonalne do modułu predykcyjnego:

- proces uczenia modelu powinien być powtarzalny;
- moduł powinien zapewniać stabilność działania modelu podczas generowania prognoz;
- czas generowania predykcji powinien umożliwiać ich monitorowanie w czasie rzeczywistym.

Wymagania do serwera aplikacyjnego:

- serwer powinien zapewniać niezawodną komunikację pomiędzy modułami systemu;
- komunikacja z serwerem powinna odbywać się z wykorzystaniem protokołu *HTTP*;
- czas odpowiedzi serwera powinien być możliwie jak najkrótszy.

Wymagania do modułu gromadzenia i przetwarzania danych:

- interfejs użytkownika powinien być intuicyjny i czytelny;
- aplikacja powinna działać poprawnie w popularnych przeglądarkach internetowych;
- interfejs użytkownika powinien umożliwiać czytelną wizualizację danych;
- aplikacja powinna reagować na błędy w sposób zrozumiały dla użytkownika.

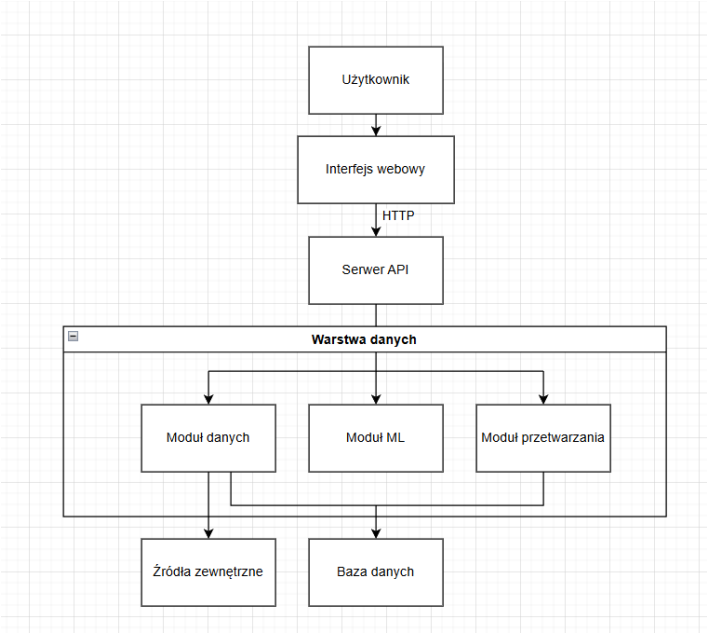
Wymagania sprzętowo-programowe:

- komputer z dostępem do Internetu;
- przeglądarka internetowa obsługująca nowoczesne standardy JavaScript;
- środowisko serwerowe obsługujące język *Python* w wersji 3.9 lub wyższej;
- zainstalowane biblioteki: *NumPy*, *Pandas*, *Scikit-learn*, *CatBoost*, *FastAPI*;
- system zarządzania bazą danych umożliwiający przechowywanie danych historycznych i wyników predykcji.

3.3. Architektura systemu

Platforma predykcji i monitorowania danych kryptowalutowych została zaprojektowana w oparciu o architekturę klient-serwer z modularną strukturą. System składa się z trzech głównych warstw, które współpracują ze sobą:

- warstwa danych pobiera dane z zewnętrznego API, prowadzi inżynierię cech i trenuje model predykcyjny;
- serwer API udostępnia endpointy REST;
- interfejs użytkownika.



Rysunek 3.1. Diagram architektury projektu

Kluczowe cechy tej architektury to modularność, rozdzielenie odpowiedzialności, skalowalność i kompletna niezależność technologiczna. Zapewnia ona elastyczność w rozwoju systemu oraz łatwość rozbudowy poszczególnych komponentów.

3.4. Diagramy systemu

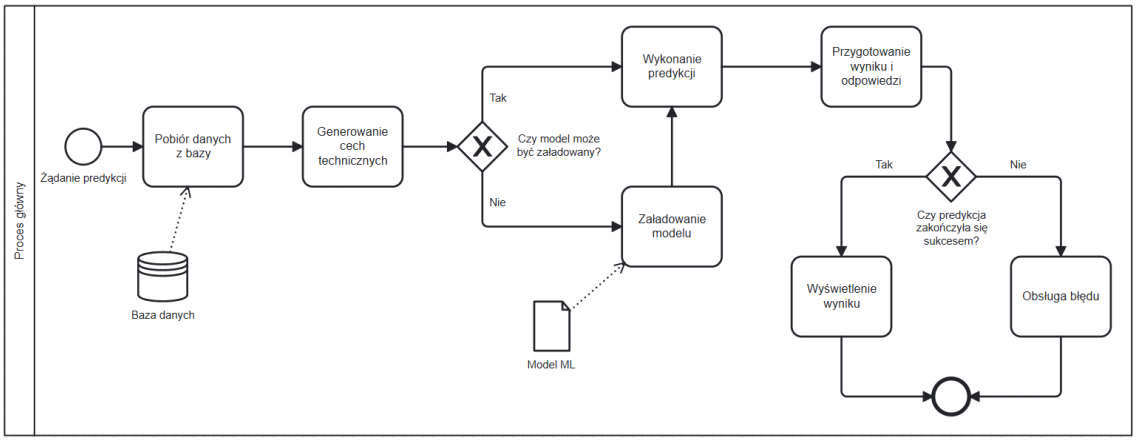
W celu wizualizacji procesów i struktury systemu opracowano zestaw diagramów zgodnych z notacjami przyjętymi w inżynierii oprogramowania. Diagramy te przedstawiają kluczowe aspekty funkcjonowania platformy, obejmując zarówno procesy biznesowe, jak i strukturę danych.

3.4.1. Diagram BPMN procesu predykcji

Diagram BPMN procesu predykcji ilustruje główny przebieg operacji, które są wykonywane w odpowiedzi na zapytanie użytkownika. Całość zaczyna się od otrzymania żądania dotyczącego prognozy kierunku zmiany ceny.

Pierwszym krokiem jest pobranie danych historycznych z bazy. W procesie pojawia się moment, w którym należy zweryfikować, czy model uczenia maszynowego jest dostępny i gotowy do działania. Jeśli model jeszcze nie został załadowany, wtedy system rozpoczyna procedurę wczytywania go z pliku oraz uruchamiania usługi ML. Gdy model jest już załadowany, można przejść do wykonania predykcji, czyli zastosowania algorytmu uczenia maszynowego na wcześniej przetworzonych danych wejściowych.

Na końcu procesu następuje przygotowanie wyniku – obliczenie prawdopodobieństwa, określenie kierunku zmiany i pobranie bieżącej ceny. Całość kończy się przekazaniem odpowiedzi do użytkownika, chociaż jeśli coś pójdzie nie tak, system obsługuje wtedy odpowiednie błędy.



Rysunek 3.2. Diagram BPMN procesu predykcji

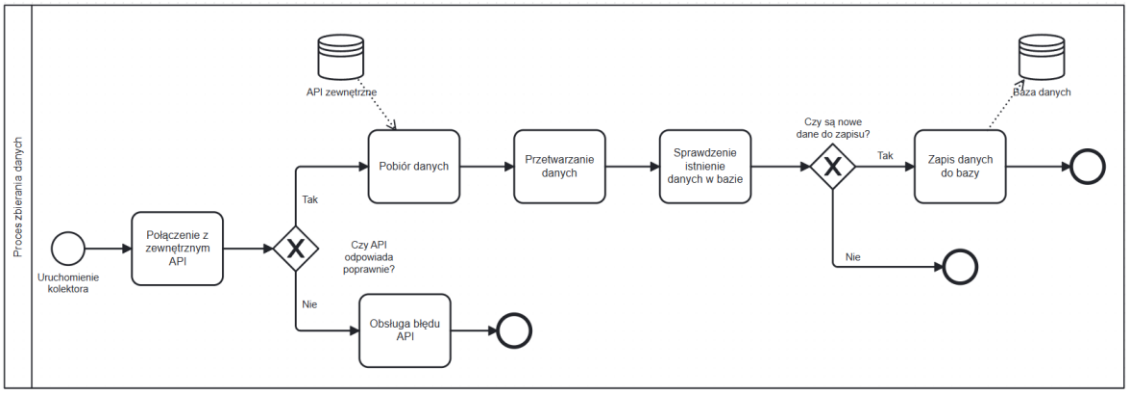
3.4.2. Diagram BPMN procesu zbierania danych

Proces zbierania danych ilustruje cykliczną procedurę pozyskiwania informacji z zewnętrznych źródeł. Rozpoczyna się od uruchomienia modułu kolektora, który nawiązuje połączenie z zewnętrznym API. Po ustanowieniu komunikacji system weryfikuje poprawność odpowiedzi API, co stanowi punkt kontrolny przed dalszym przetwarzaniem.

W przypadku pomyślnej odpowiedzi API, system pobiera dane w formacie OHLCV (*Open, High, Low, Close, Volume*) z określonym interwałem czasowym. Następnie dane są przetwarzane, obejmując konwersję typów, normalizację formatów oraz dodanie niezbędnych kolumn opisowych.

W procesie ważną rolę pełni mechanizm deduplikacji, polegający na sprawdzeniu ostatniego dostępnego znacznika czasowego w bazie danych. Pozwala to na filtrowanie jedynie nowych rekordów, unikając powielania informacji. Decyzja o zapisie do bazy zależy od tego, czy pojawiły się nowe dane do przetworzenia.

Na końcu, po weryfikacji i przetworzeniu, dane zostają zapisane w bazie, co zapewnia ciągłą aktualizację zbioru danych służącego do analiz i prognoz. Wszystko to powtarza się cyklicznie, przez co system pozostaje na bieżąco z informacjami rynkowymi.



Rysunek 3.3. Diagram BPMN procesu zbierania danych

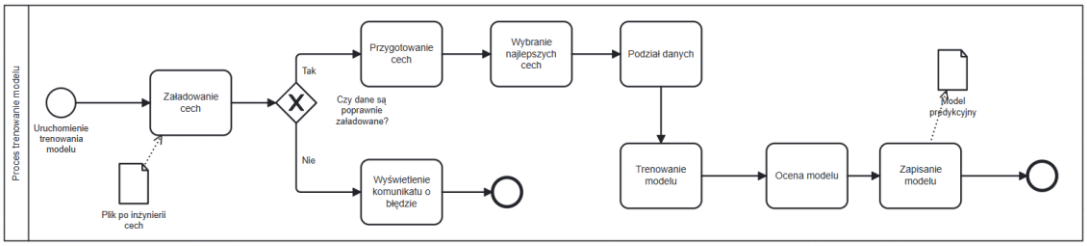
3.4.3. Diagram BPMN procesu trenowania modelu

Proces trenowania modelu ML zawiera etapy przygotowania i optymalizacji algorytmu predykcyjnego. Inicjowany jest ręcznie poprzez uruchomienie skryptu, co rozpoczyna sekwencję operacji mających na celu stworzenie nowej wersji modelu.

Pierwszy etap polega na załadowaniu danych z pliku z cechami technicznymi, który zawiera przetworzone cechy wygenerowane wcześniej przez moduł inżynierii cech. Po pomyślnym wczytaniu danych system przystępuje do selekcji najbardziej najbardziej istotnych cech, co pozwala na redukcje wymiarowości i eliminację cech redundantnych.

Kolejnym krokiem jest podział zbioru danych na część treningową i testową z zachowaniem porządku czasowego. Jest to kluczowe dla problemów szeregów czasowych, zapobiegając wyciekom informacji z przyszłości.

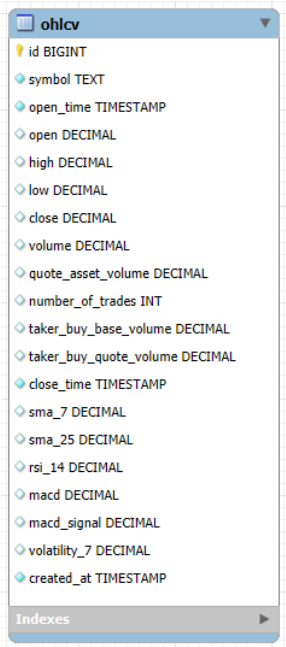
Wytrenowany model jest konwertowany do pliku wraz z metadanymi obejmującymi listę cech, ważność cech oraz osiągnięte metryki jakości. Zapisany model jest gotowy do wykorzystania w procesie predykcji.



Rysunek 3.4. Diagram BPMN procesu trenowania modelu

3.4.4. Diagram bazy danych

Struktura bazy danych została zaprojektowana z myślą o efektywnym przechowywaniu danych szeregów czasowych. Centralną tabelą systemu jest tabela OHLCV. Tabela zawiera wszystkie kolumny niezbędne do wykorzystania jej w celach pracy.



Rysunek 3.5. Diagram bazy danych

4. Wykorzystane technologie i narzędzia

Do utworzenia systemu predykcji i monitorowania danych kryptowalutowych zostały wykorzystane nowoczesne technologie. Wybór narzędzi podyktowany był wymaganiami projektu, które obejmowały przetwarzanie danych szeregów czasowych, implementację modelu uczenia maszynowego oraz stworzenie responsywnego interfejsu użytkownika. Zastosowane rozwiązania zapewniają skalowalność, wysoką wydajność oraz łatwość utrzymania systemu.

4.1. Backend i przetwarzanie danych

Warstwa backendowa systemu została zaimplementowana w języku Python w wersji 3.9, który wełni rolę głównego języka programowania odpowiedzialnego za realizację logiki biznesowej, przetwarzanie danych oraz implementację modelu predykcyjnego. Python został wybrany ze względu na:

- bogaty ekosystem bibliotek do analizy danych i uczenia maszynowego;
- łatwość integracji z zewnętrznymi systemami i interfejsami API;
- wysoką czytelność kodu, ułatwiającą jego utrzymanie i rozwój.

Do budowy interfejsu programistycznego zastosowano framework FastAPI, umożliwiający tworzenie wydajnych i skalowalnych usług. Framework ten zapewnia wbudowaną walidację danych wejściowych oraz automatyczną generację dokumentacji w standardzie *OpenAPI*, co usprawnia proces testowania i integracji.

Przetwarzanie i analiza danych realizowane są z wykorzystaniem bibliotek *Pandas* i *NumPy*. Biblioteka *Pandas* służy do ładowania, czyszczenia, transformacji i agregacji danych szeregów czasowych, natomiast *NumPy* wykorzystana jest do realizacji operacji numerycznych na macierzach oraz obliczeń.

W procesie uczenia maszynowego wykorzystano bibliotekę Scikit-learn, która została zastosowana do:

- wstępnego przetwarzania danych;
- selekcji cech;
- podziału zbiorów danych na treningowe i testowe;
- obliczania metryk jakości predykcji.

Głównym modelem predykcyjnym zastosowanym w systemie jest algorytm *CatBoost*, który charakteryzuje się wysoką skutecznością predykcyjną, odpornością na przeuczenie oraz dobrą obsługą danych szeregów czasowych.

Do obliczania wskaźników analizy technicznej wykorzystano bibliotekę *TA-Lib*, umożliwiającą wyznaczenie m.in.:

- średnich kroczących;
- wskaźników momentum;
- miar zmienności rynku;
- oscylatorów i wskaźników trendu.

Dostęp do bazy danych realizowany jest przy użyciu biblioteki *SQLAlchemy*, która zapewnia abstrakcję warstwy dostępu do danych oraz zwiększa bezpieczeństwo aplikacji. Pobieranie danych rynkowych z zewnętrznych źródeł odbywa się z wykorzystaniem biblioteki *Requests*, umożliwiającej realizację zapytań *HTTP* oraz komunikację z interfejsami *REST*.

4.2. Baza danych

System wykorzystuje relacyjną bazę danych *PostgreSQL* rozszerzoną o moduł *TimescaleDB*, zaprojektowany z myślą o efektywnym przechowywaniu danych szeregów czasowych. Zastosowanie tego rozwiązania umożliwia:

- automatyczne partycjonowanie danych czasowych;
- optymalizację zapytań dotyczących przedziałów czasowych;
- skalowalne przechowywanie dużych wolumenów danych rynkowych.

4.3. Modelowanie ML

Do implementacji modeli predykcyjnych w systemie wykorzystano następujące technologie i narzędzia:

W pracy użyte zostały następujące biblioteki do uczenia maszynowego: *CatBoost*, *Scikit-learn, Joblib, TA-Lib, NumPy* i *Pandas.*

CatBoost jest to zaawansowany algorytm gradient boosting z natywną obsługą cech kategorycznych, wybrany jako główny silnik predykcyjny ze względu na wysoką

skuteczność w zadaniach klasyfikacyjnych na danych szeegów czasowych.

Scikit-learn jest to kompleksowa biblioteka do uczenia maszynowego, wykorzystana do selekcji cech, podziału danych oraz obliczania metryk ewentualnych.

Joblib to narzędzie do efektywnej serializacji i deserializacji modeli ML, umożliwiające zapis i odczyt wytrenowanych modeli.

TA-Lib to specjalistyczna biblioteka do obliczania wskaźników analizy technicznej, wykorzystana w procesie inżynierii cech.

NumPy i *Pandas* są to podstawowe narzędzia do manipulacji i przetwarzania danych numerycznych oraz szeregów czasowych.

Wybór technologii został podyktowany ich dojrzałością, wydajnością oraz bogatym ekosystemem narzędziowym, co umożliwiło zbudowanie kompleksowego systemu predykcyjnego.

4.4. Frontend

Warstwa interfejsu użytkownika została zrealizowana z wykorzystaniem biblioteki *React* w wersji 18. Zastosowanie architektury komponentowej pozwala na modularne projektowanie interfejsu oraz reużywalność kodu, natomiast mechanizm wirtualnego *DOM* wpływa na wysoką wydajność aplikacji.

Do wizualizacji danych predykcyjnych i historycznych zastosowano bibliotekę *Recharts*, umożliwiającą tworzenie interaktywnych i responsywnych wykresów. Komunikacja z backendem realizowana jest przy użyciu biblioteki *Axios*, natomiast stylizacja interfejsu została wykonana z wykorzystaniem nowoczesnych mechanizmów *CSS3*, zapewniających spójność wizualną i responsywność aplikacji.

4.5. Narzędzia deweloperskie

Proces tworzenia systemu był wspierany przez zestaw narzędzi deweloperskich, do których należą *Visual Studio Code* jako główne środowisko programistyczne, Postman do testowania i dokumentowania interfejsów *API* oraz narzędzia *pgAdmin* i Dbeaver do zarządzania bazą danych. W fazie eksploracyjnej analizy danych oraz prototypowania modeli uczenia maszynowego wykorzystano środowisko *Jupyter Notebook*, umożliwiające interaktywną pracę z danymi i wizualizację wyników.

5. Model predykcyjny

System predykcyjny opiera się na danych historycznych pozyskiwanych z giełdy kryptowalutowej *Binance*, która jako największa globalna platforma handlu kryptowalut zapewnia wysoki poziom wiarygodności i kompletności danych [15]. Dane obejmują notowanie pary *BTC/USDT* w formacie *OHLCV* z interwałem 1-godzinnym, co stanowi optymalny kompromis między detaliczną analizą krótkoterminowych fluksacji a stabilnością konieczną do identyfikacji trendów średnioterminowych.

5.1. Charakterystyka i źródła danych

Struktura danych źródłowych obejmuje następujące kluczowe elementy:

- podstawowe metryki cenowe: cena otwarcia, maksymalna, minimalna i zamknięcia;
- wolumeny transakcyjne: wolumen w jednostce bazowej (BTC) i walucie kwotowanej (USTD);
- statystyki handlowe: liczba transakcji w przedziale czasowym;
- wskaźniki aktywności kupujących: wolumeny inicjowane przez stronę kupującą (taker volumes).

Okres historyczny został ustalony na 180 dni, co odpowiada około 4320 obserwacjom godzinowym. Taka długość serii czasowej umożliwia uwzględnienie cykli rynkowych o różnej częstotliwości, jednocześnie ograniczając wpływ historycznych anomalii na aktualne predykcje. Dane są aktualizowane w czasie rzeczywistym poprzez cykliczne pobieranie z publicznego *API Binance*, co zapewnia świeżość danych wejściowych dla modelu.

5.2. Przygotowanie danych i inżynieria cech

Proces przygotowania danych stanowi kluczowy etap budowy systemu predykcyjnego, obejmujący cztery zasadnicze fazy: czyszczenie, transformację, generowanie cech technicznych oraz przygotowanie zmiennej docelowej.

5.2.1. Czyszczenie i wstępne i przetwarzanie

Dane giełdowe charakteryzują się specyficznymi wyzwaniami jakościowymi,

w tym występowaniem outlierów, braków w serii czasowej oraz artefaktów technicznych. W systemie zaimplementowano następujące procedury czyszczenia:

1. filtracja nieprawidłowych rekordów: usunięcie obserwacji z nieprawidłowymi wartościami cen (np. ujemne lub zerowe) oraz anomalii wolumenowych;
2. imputacja brakujących wartości: zastosowanie metody *forward-fill* dla pojedynczych brakujących obserwacji, zapewniającej zachowanie ciągłości szeregu czasowego;
3. normalizacja skrajnych wartości: winsoryzacja ekstremalnych obserwacji na poziomie percentyla 99, redukująca wpływ outlierów bez utraty informacji o zmienności.

5.2.2. Generowanie cech technicznych

Inżynieria cech obejmuje konstrukcję kompleksowego zestawu wskaźników analizy technicznej pogrupowanych w następujące kategorie:

1. Wskaźniki trendu [7]:
   - średnie kroczące (MA): SMA (5, 10, 20, 50 okresów) i EMA z odpowiadającymi im wskaźnikami odchylenia od ceny bieżącej;
   - MACD (*Moving Average Convergence Divergence*): wskaźnik złożony z linii MACD (różnica EMA12 i EMA26), linii sygnału (EMA9 MACD) oraz histogramu różnic;
   - *parabolic SAR*: wskaźnik identyfikacji punktów zwrotnych trendu.
2. Wskaźniki momentum:
   - RSI (*Relative Strength Index*): obliczany dla okresów 7, 14, 21, identyfikujący warunki wykupienia (>70) i wyprzedania (<30).
3. Wskaźniki zmienności:
   - ATR (*Average True Range*): średni prawdziwy zakres dla okresów 5, 10, 20, mierzy zmienność niezależnie od kierunku ruchu;
   - *Bellinger Bands*: wstęgi wyznaczone jako SMA ± 2 odchylenia standardowe, z dodatkowymi metrykami pozycji ceny względem wstęg.

4. Wskaźniki wolumenu:
   - OBV (*On-Balance Volume*): skumulowany wskaźnik kierunku wolumenu;
   - VWAP (*Volume Weighted Average Price*): średnia ważona wolumenem, stanowiąca benchmark cenowy;
   - Stosunki wolumenowe: analiza struktury rynku poprzez wskaźniki kupujących.
5. Transformacje podstawowe:
   - zwroty procentowe i logarytmiczne;
   - zakres cenowy;
   - struktura świecy.

Uwzględnienie kontekstu czasowego jest kluczowe dla modelowania zależności sezonowych:

1. cechy cykliczne: sinusoidalne i kosinusoidalne transformacje godzin i dni tygodnia, umożliwiające modelowi rozpoznanie cykliczności bez narzucania sztucznego porządku;
2. flagi czasowe: wskaźniki weekendów, godzin handlu na głównych rynkach tradycyjnych (otwarcie/zamknięcie giełd w USA, Azji, Europie);
3. cechy opóźnione (*lag features*): wartości kluczowych zmiennych z opóźnieniami od 1 do 48 godzin, uchwytujące autokorelacje krótko- i długoterminową;
4. statystyki okienkowe: średnie, odchylenia standardowe, ekstrema i percentyle dla kluczowych zmiennych w oknach 3, 6, 12, 24 godzin.

5.2.3. Definicja zmiennej docelowej

Zmienna docelowa [11] została zdefiniowana jako binarna klasyfikacja kierunku zmiany ceny w następnym okresie (horizon = 1 godzina). Formalnie:

$$y_t = \begin{cases} 1, jeśli\ \frac{P_{t+1} - P_t}{P_t} > \theta, \\ 0, w\ przeciwnym\ przypadku \end{cases}$$

Gdzie P_t oznacza cenę zamknięcia w czasie t, a θ to próg minimalnej zmiany (ustalony na 0, aby uwzględnić wszelkie wzrosty). Alternatywnie eksperymentowano z:

1. Klasyfikacją trójklasową (wzrost/spadek/brak istotnej zmiany) z progami opartymi o percentyle rozkładu zwrotów.
2. Regresja ciągłej wartości przyszłego zwrotu.
3. Klasyfikacja wielohoryzontalna dla różnych perspektyw czasowych.

Podejście binarne okazało się najbardziej efektywne w kontekście dalszego wykorzystania predykcji w systemach wsparcia decyzji inwestycyjnych.

5.3. Wybór i uzasadnienie algorytmu

Wybór algorytmu uczenia maszynowego poprzedzony został analizą porównawcza kilku kandydatów pod kątem ich przydatności do specyficznego problemu predykcji cen kryptowalut.

5.3.1. Analiza porównawcza algorytmów

Przeprowadzono porównanie następujących algorytmów:

1. *Random Forest* – metoda ensemble oparta na drzewach decyzyjnych;
2. *XGBoost* – zoptymalizowana implementacja gradient boosting;
3. *CatBoost* – algorytm *gradient boosting* z natywną obsługą cech kategorycznych.

Tabela 5.1. Porównanie algorytmów na zbiorze walidacyjnym

Algorytm	Accuracy	Balanced Accuracy	Czas treningu	Interpretowalność
Random Forest	60,1%	59,7%	Średni	Wysoka
XGBoost	61,3%	60,9%	Niski	Średnia
CatBoost	63,3%	63,5%	Niski	Wysoka

5.3.2. Uzasadnienie wyboru *CatBoost*

Decyzja o wyborze algorytmu *CatBoost* (*Categorical Boosting*) została podjęta następujących kryteriów:

1. Wydajność produkcyjna:

CatBoost osiągnął najwyższe wartości zarówno *accuracy*, jak i *balanced accuracy*.

2. Odporność na przeuczenie:

Wbudowane mechanizmy regularyzacyjne *CatBoost* zapewniają dobrą generalizację na nowych danych.

3. Obsługa cech kategorycznych:

Natywna obsługa cech kategorycznych eliminuje konieczność transformacji *one-hot encoding* lub podobnych.

4. Interpretowalność modelu:

CatBoost zapewnia metryki ważności cech (*feature importance*) [16].

5. Stabilność numeryczna:

Algorytm jest odporny na outlierów i brakujące wartości.

5.3.3. Architektura modelu:

Ostateczna konfiguracja modelu *CatBoost* obejmuje następujące parametry:

- liczba iteracji;
- głębokość drzew;
- szybkość uczenia;
- funkcja straty;
- metryka walidacyjna.

5.4. Proces uczenia i walidacji modelu

5.4.1. Strategia podziału danych

Ze względu na czasowy charakter danych zastosowano podział typu *time-based split* z proporcją 80/20. Oznacza to, że pierwsze 80% obserwacji chronologicznych stanowi zbiór treningowy, a 20% - testowy. Eliminuje to wyciek informacji z przyszłości (*data leakage*) i symuluje rzeczywiste warunki, w których model jest trenowany na danych historycznych i testowany na najnowszych obserwujących.

5.4.2. Techniki balansowane danych

Analiza rozkładu zmiennej docelowej wykazała lekką asymetrię (około 52% obserwacji należących do klasy pozytywnej). Aby przeciwdziałać potencjalnemu biasowi modelu zastosowano:

1. automatyczne ważenie klas;
2. metrykę *balanced accuracy* jako główny cel optymalizacji.

5.4.3. Procedura trenowania

Proces trenowania obejmuje następujące kroki:

1. inicjacja modelu z parametrami dostosowanymi do problemu klasyfikacji binarnej;
2. trening przyrostowy z walidacją na oddzielnym zbiorze;
3. *early stopping* monitorujące *balanced accuracy*;
4. eksport modelu wraz z metadanymi.

5.4.4. Walidacja i testowanie

System walidacji obejmuje:

1. walidację na zbiorze testowym (ostatnie 20% danych chronologicznie);
2. wyodrębnienie metryk oceny jakości.

5.5. Metryki oceny jakości

W celu skuteczności modelu predykcyjnego zastosowano zestaw metryk statystycznych dostosowanych do specyfiki problemu klasyfikacji binarnej w kontekście danych finansowych.

5.5.1. Metryki klasyfikacyjne

Analiza macierzy pomyłek:

- *True Positive Rate* – zdolność wykrywania rzeczywistych wzrostów;
- *True Negative Rate* – zdolność identyfikacji spadków;
- *False Positive Rate* – odsetek fałszywych alarmów.

– *False Positive Rate* – odsetek fałszywych identyfikacji spadków.

Podstawowe metryki oceny [4]:

– *accuracy* – ogólna poprawność klasyfikacji:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

– *balanced accuracy* – *accuracy* skorygowana o nierównowagę klas:

$$balanced\ accuracy = 2 * \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

– *precision* – trafność przewidywań wzrostu:

$$precision = \frac{TP}{TP + FP}$$

– *recall* – kompletność wykrywania wzrostów:

$$recall = \frac{TP}{TP + FN}$$

– *f1-score* – średnia harmoniczna precision i *recall*.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5.5.2. Analiza wyników

Przyjęto następujące kryteria oceny użyteczności modelu:

– próg użyteczności: *accuracy* > 60% dla zastosowań wspomagania decyzji inwestycyjnych;

– balans metryk: ocena kompromisu między precision a *recall*;

– analiza kosztów błędów: różna waga błędów typu 1 i 2 w kontekście inwestycyjnym.

Wyniki szczegółowej ewaluacji modelu przedstawiono w rozdziale 7.

6. Implementacja systemu

6.1. Moduł zbierania i przetwarzania danych

6.1.1. Architektura kolektora danych

System implementuje kolektor danych oparty na wzorcu *Repository Pattern*, który oddziela logikę biznesową od szczegółów dostępu do danych. Główna klasa *DataCollector* łączy się z *API Binance* i pobiera dane *OHLCV*.

Kod implementuje stronicowanie danych. *API Binance* zwraca maksymalnie 1000 rekordów na zapytanie, dlatego została użyta pętla *while*. Parametr *startTime* jest aktualizowany po każdym żądaniu na podstawie *close_time* ostatniej świecy. Wywołanie *time.sleep(0.2)* zapobiega przekroczeniu limitów API.

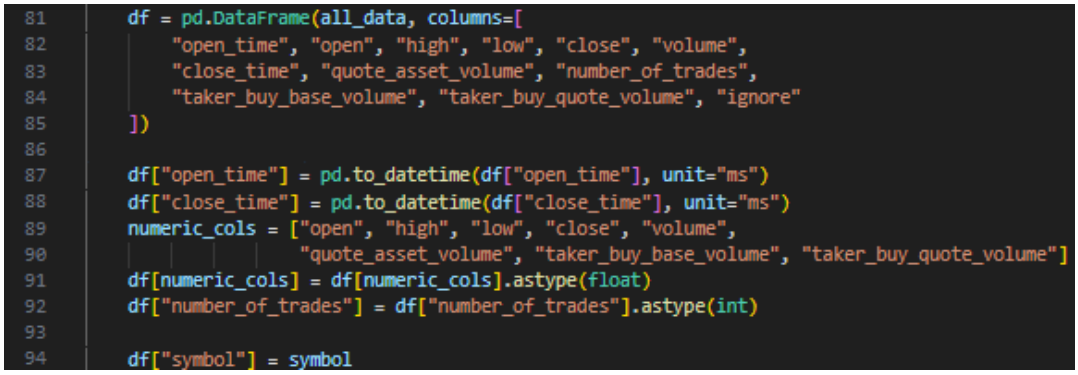
```python
def fetch_klines(symbol, interval, start_date, end_date):
    url = "https://api.binance.com/api/v3/klines"
    all_data = []
    start_ts = int(start_date.timestamp() * 1000)
    end_ts = int(end_date.timestamp() * 1000)

    while start_ts < end_ts:
        params = {
            "symbol": symbol,
            "interval": interval,
            "limit": LIMIT,
            "startTime": start_ts
        }
        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
            data = response.json()
            if not data:
                break

            all_data.extend(data)
            start_ts = data[-1][6] + 1  # close_time + 1 мс
            print(f"Loaded {len(data)} candles, next: {datetime.fromtimestamp(start_ts/1000)}")
            time.sleep(0.2)  # Rate limiting
        except Exception as e:
            print(f"FAILURE -- Error fetching data: {e}")
            break
```

Rysunek 6.1. Funkcja pobierania danych z *API Binance*

6.1.2. Przetwarzanie i normalizacja danych

Po pobraniu surowych danych, system przekształca je do struktury *pandas DataFrame* z odpowiednimi typami danych.

Kolumny są mapowane na odpowiednie nazwy, *timestampy* konwertowane z milisekund na obiekty *datetime*, a kolumny numeryczne na *float*. Ostatnia kolumna *ignore* jest pomijana zgodnie z dokumentacją *Binance*.



```
81      df = pd.DataFrame(all_data, columns=[
82          "open_time", "open", "high", "low", "close", "volume",
83          "close_time", "quote_asset_volume", "number_of_trades",
84          "taker_buy_base_volume", "taker_buy_quote_volume", "ignore"
85      ])
86
87      df["open_time"] = pd.to_datetime(df["open_time"], unit="ms")
88      df["close_time"] = pd.to_datetime(df["close_time"], unit="ms")
89      numeric_cols = ["open", "high", "low", "close", "volume",
90                      "quote_asset_volume", "taker_buy_base_volume", "taker_buy_quote_volume"]
91      df[numeric_cols] = df[numeric_cols].astype(float)
92      df["number_of_trades"] = df["number_of_trades"].astype(int)
93
94      df["symbol"] = symbol
```

Rysunek 6.2. Przekształcenie danych w kodzie

6.1.3. Zapisywanie do bazy z deduplikacją

System implementuje mechanizm unikania duplikatów poprzez sprawdzenie ostatniego *timestampu* w bazie przed wstawieniem nowych danych.

System najpierw pobiera maksymalny *open_time* dla danego symbolu z bazy. Następnie filtruje *DataFrame*, pozostawiając tylko rekordy nowsze niż ten *timestamp*. Dzięki temu unikamy duplikowania danych przy wielokrotnych uruchomieniach kolektora, dzięki czemu każde kolejne go uruchomienie uzupełnia brakujące na moment wywołania rekordy.

```python
def save_to_db(df):
    if df.empty:
        print(" !!! No data to save.")
        return

    existing_query = text("""
        SELECT MAX(open_time) as last_time FROM ohlcv WHERE symbol = :symbol
    """)

    with engine.connect() as conn:
        result = conn.execute(existing_query, {"symbol": SYMBOL})
        last_time = result.scalar()

    if last_time:
        df = df[df['open_time'] > last_time]
        print(f"Found {len(df)} new records to insert")

    if df.empty:
        print("!!! No new data to insert")
        return

    df_to_insert = df[[
        "symbol", "open_time", "open", "high", "low", "close", "volume",
        "quote_asset_volume", "number_of_trades",
        "taker_buy_base_volume", "taker_buy_quote_volume", "close_time"
    ]]

    try:
        df_to_insert.to_sql("ohlcv", engine, if_exists="append", index=False)
        print(f"SUCCESS -- Saved {len(df_to_insert)} rows to TimescaleDB")
    except Exception as e:
        print(f"EXCEPTION -- Error saving to DB: {e}")
```

Rysunek 6.3. Zapisywanie danych do bazy danych

6.2. Moduł inżynierii cech

6.2.1. Ładowanie danych z bazy

Moduł ładuje dane bezpośrednio z bazy *TimescaleDB* z wykorzystaniem *SQLAlchemy*.

Funkcja *asfreq("1H")* zapewnia regularny interwał, wypełniając brakujące godziny. Funkcja *fillna(method="ffill")* używa *forward fill* dla brakujących wartości. *last(f"{lookback_days}D")* ogranicza dane do ostatnich N dni.

```python
def load_raw_data(symbol=SYMBOL, lookback_days=25):
    engine = create_engine(DB_URL)
    query = f"""
        SELECT open_time, open, high, low, close, volume,
               quote_asset_volume, number_of_trades,
               taker_buy_base_volume, taker_buy_quote_volume
        FROM ohlcv
        WHERE symbol = '{symbol}'
        ORDER BY open_time ASC
    """
    df = pd.read_sql(query, engine, parse_dates=["open_time"])
    df = df.set_index("open_time").asfreq("1H").fillna(method="ffill")
    if lookback_days:
        df = df.last(f"{lookback_days}D")
    return df
```

Rysunek 6.4. Ładowanie danych z bazy

6.2.2. Generowanie wskaźników technicznych

System wykorzystuje bibliotekę ta do obliczania ponad 20 różnych wskaźników technicznych.

Dodawane są różne typy wskaźników:

- trend: SMA, EMA, MACD;
- momentum: RSI;
- zmienność: Bollinger Bands;
- wolumen: stosunki wolumenowe.

Stała 1e-8 zapobiega dzieleniu przez zero.

```python
def add_technical_features(df):
    df["return"] = df["close"].pct_change()
    df["log_return"] = np.log(df["close"] / df["close"].shift(1))
    df["price_range"] = (df["high"] - df["low"]) / (df["open"] + 1e-8)
    df["body_size"] = (df["close"] - df["open"]).abs() / (df["open"] + 1e-8)
    df["is_doji"] = ((df["close"] - df["open"]).abs() / (df["high"] - df["low"] + 1e-8)) < 0.1
    df["taker_buy_ratio_base"] = df["taker_buy_base_volume"] / (df["volume"] + 1e-8)
    df["taker_buy_ratio_quote"] = df["taker_buy_quote_volume"] / (df["quote_asset_volume"] + 1e-8)
    df["volume_ratio"] = df["volume"] / (df["volume"].rolling(24).mean() + 1e-8)

    for period in [5, 7, 10, 20, 25, 50]:
        df[f"sma_{period}"] = SMAIndicator(df["close"], period).sma_indicator()
        df[f"ema_{period}"] = EMAIndicator(df["close"], period).ema_indicator()
        df[f"sma_ratio_{period}"] = df["close"] / (df[f"sma_{period}"] + 1e-8) - 1

    macd_obj = MACD(df["close"], window_fast=12, window_slow=26, window_sign=9)
    df["macd"] = macd_obj.macd()
    df["macd_signal"] = macd_obj.macd_signal()
    df["macd_diff"] = macd_obj.macd_diff()

    for period in [7, 14, 21]:
        df[f"rsi_{period}"] = RSIIndicator(df["close"], period).rsi()

    for period in [10, 20]:
        bb = BollingerBands(df["close"], window=period, window_dev=2)
        df[f"bb_high_{period}"] = bb.bollinger_hband()
        df[f"bb_low_{period}"] = bb.bollinger_lband()
        df[f"bb_middle_{period}"] = bb.bollinger_mavg()
        df[f"bb_width_{period}"] = (df[f"bb_high_{period}"] - df[f"bb_low_{period}"]) / (df[f"bb_middle_{period}"] + 1e-8)
        df[f"bb_pos_{period}"] = (df["close"] - df[f"bb_low_{period}"]) / ((df[f"bb_high_{period}"] - df[f"bb_low_{period}"]) + 1e-8)

    for period in [5, 7, 10, 20]:
        atr = AverageTrueRange(df["high"], df["low"], df["close"], window=period)
        df[f"atr_{period}"] = atr.average_true_range()
        df[f"atr_ratio_{period}"] = df[f"atr_{period}"] / (df["close"] + 1e-8)

    vwap = VolumeWeightedAveragePrice(df["high"], df["low"], df["close"], df["volume"], window=14)
    df["vwap"] = vwap.volume_weighted_average_price()
    df["vwap_dev"] = (df["close"] - df["vwap"]) / (df["vwap"] + 1e-8)

    obv = OnBalanceVolumeIndicator(df["close"], df["volume"])
    df["obv"] = obv.on_balance_volume()
    df["obv_change"] = df["obv"].pct_change()

    for period in [7, 14, 21]:
        df[f"volatility_{period}"] = df["return"].rolling(window=period).std() * np.sqrt(24 * 365)

    print(f"✓ Added {len(df.columns)} technical features")
    return df
```

Rysunek 6.5. Dodanie cech technicznych w kodzie

6.2.3. Cechy czasowe i opóźnione

System dodaje cechy czasowe oraz opóźnione wartości (tzw. *lag features*) które są kluczowe dla modeli szeregów czasowych.

Cechy czasowe wykorzystują transformacje sinusoidalne i kosinusoidalne do cyklicznego kodowania czasu. *Lag features* (opóźnienia) pozwalają modelowi uczyć się z historii. Okna kroczące (*rolling windows*) agregują statystyki z ostatnich M okresów.

```python
def add_time_features(df):
    df.index = pd.to_datetime(df.index)
    df["hour"] = df.index.hour
    df["day_of_week"] = df.index.dayofweek
    df["day_of_month"] = df.index.day
    df["is_weekend"] = df["day_of_week"].isin([5, 6]).astype(int)
    df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
    df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)
    df["dow_sin"] = np.sin(2 * np.pi * df["day_of_week"] / 7)
    df["dow_cos"] = np.cos(2 * np.pi * df["day_of_week"] / 7)
    return df

def add_lag_features(df, lags=[1, 2, 3, 6, 12, 24, 48]):
    for lag in lags:
        df[f"return_lag_{lag}"] = df["return"].shift(lag)
        df[f"volume_lag_{lag}"] = df["volume"].shift(lag)
        df[f"price_range_lag_{lag}"] = df["price_range"].shift(lag)
        df[f"close_lag_{lag}"] = df["close"].shift(lag)

    for window in [3, 6, 12, 24]:
        df[f"return_ma_{window}"] = df["return"].rolling(window).mean()
        df[f"return_std_{window}"] = df["return"].rolling(window).std()
        df[f"volume_ma_{window}"] = df["volume"].rolling(window).mean()
        df[f"price_range_ma_{window}"] = df["price_range"].rolling(window).mean()

    return df
```

Rysunek 6.6. Funkcja dodania cech czasowych

6.2.4. Definicja zmiennej docelowej

System definiuje problem jako klasyfikację binarną – przewidywanie kierunku ruchu w następnej godzinie.

target_direction to binarna zmienna: 1 jeśli cena wzrośnie w następnej godzinie, 0 jeśli spadnie. target_3class rozszerza to do 3 klas (wzrost/spadek/brak znaczącej zmiany) z progiem opartym na percentyl 60% bezwzględnych zwrotów.

```python
def add_target(df, horizon=1):
    df["close_future"] = df["close"].shift(-horizon)
    df["future_return"] = (df["close_future"] - df["close"]) / (df["close"] + 1e-8)
    df["target_direction"] = (df["future_return"] > 0).astype(int)
    thr = df["future_return"].abs().quantile(0.6)
    df["target_3class"] = 0
    df.loc[df["future_return"] > thr, "target_3class"] = 1
    df.loc[df["future_return"] < -thr, "target_3class"] = -1
    return df
```

Rysunek 6.7. Funkcja dodania zmiennej docelowej

6.3. Implementacja modelu predykcyjnego

6.3.1. Przygotowanie danych do uczenia

System przygotowuje macierz cech X i wektor etykiet y, usuwając kolumny, które nie są cechami.

Automatycznie identyfikuje wszystkie kolumny numeryczne, następnie usuwa te, które są zmiennymi docelowymi lub pomocniczymi. Brakujące wartości są wypełniane zerami.

```
23  def prepare_feature_data(df):
24      exclude_cols = ['close_future', 'future_return', 'target_direction', 'target_3class']
25
26      numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
27      feature_cols = [col for col in numeric_cols if col not in exclude_cols]
28
29      X = df[feature_cols].fillna(0)
30      y = df['target_direction']
31
32      print(f"Using {len(feature_cols)} features")
33      return X, y, feature_cols
```

Rysunek 6.8. Funkcja przygotowania modelu predykcyjnego

6.3.2. Selekcja cech

System używa testu *ANOVA (f_classif)* do wyboru najbardziej istotnych statystycznie cech.

SelectKBest z funkcją *f_classif* ocenia każdą cechę indywidualnie na podstawie testu ANOVA F-value między cechą a zmienną docelową. Wybiera K cech z najwyższymi wartościami F.

```
35  def select_best_features(X, y, k=30):
36      selector = SelectKBest(f_classif, k=min(k, X.shape[1]))
37      selector.fit(X, y)
38      selected_features = X.columns[selector.get_support()].tolist()
39
40      print(f"Selected {len(selected_features)} best features")
41      return selected_features
```

Rysunek 6.9. Funkcja selekcji najważniejszych cech

6.3.3. Podział danych czasowych

Dla danych szeregów czasowych system używa podziału czasowego zamiast losowego.

Pierwsze 80% danych chronologicznie jest używane do treningu, ostatnie 20% do testowania. Zapobiega to wyciekowi informacji z przyszłości (*data leakage*) co jest krytyczne dla szeregów czasowych.

```python
def create_time_based_split(df, test_size=0.2):
    split_idx = int(len(df) * (1 - test_size))
    train_mask = df.index <= df.index[split_idx]
    test_mask = df.index > df.index[split_idx]

    return train_mask, test_mask
```

Rysunek 6.10. Funkcja podziału danych czasowych

6.3.4. Konfiguracja modelu *CatBoost*

System używa *CatBoost* z automatycznym balansowaniem klas i wczesnym zatrzymaniem.

Kluczowe parametry:

- *auto_class_weights='Balanced'* – automatycznie waży klasy w przypadku nierównowagi;
- *od_type='Iter'* i *od_wait=50* – wczesne zatrzymanie, jeśli dokładność nie poprawia się przez 50 iteracji;
- *bagging_temperature=0.8* – kontroluje intensywność baggingu.

Bagging jest techniką w uczeniu maszynowym która zwiększa dokładność i odporność modeli poprzez trenowanie wielu bazowych modeli na podzbiorach i agregowanie ich przewidywań, ale za duża jego intensywność ma swoje wady.

```python
    model = CatBoostClassifier(
        iterations=1000,
        learning_rate=0.05,
        depth=8,
        l2_leaf_reg=3,
        random_strength=0.5,
        bagging_temperature=0.8,
        od_type='Iter',
        od_wait=50,
        loss_function='Logloss',
        eval_metric='Accuracy',
        random_seed=42,
        verbose=100,
        auto_class_weights='Balanced'
    )
```

Rysunek 6.11. Ustawienia parametrów modelu *CatBoost*

6.3.5. Trenowanie modelu

Model jest trenowany z walidacją na oddzielnym zbiorze i wykorzystaniem cech kategorycznych.

CatBoost ma możliwość efektywnie obsługiwać cechy kategoryczne bez potrzeby one-hot *encoding*. *use_best_model=True* wykorzystuje model z najlepszą walidacją zamiast ostatniej iteracji.

```
95      cat_features_indices = prepare_categorical_features(selected_features)
96
97
98      model.fit(
99          X_train, y_train,
100         eval_set=(X_test, y_test),
101         cat_features=cat_features_indices,
102         use_best_model=True
103     )
```

Rysunek 6.12. Trenowanie modelu

6.4. Implementacja serwera API

6.4.1. Struktura endpointów

Serwer *FastAPI* udostępnia *REST* API z pięcioma głównymi endpointami.

Endpoint zwraca dane *OHLCV* wraz ze wskaźnikami technicznymi w formacie JSON. *fillna(0)* obsługuje brakujące wartości wskaźników dla najnowszych danych.

```
30  @app.get("/api/ohlcv/{symbol}", response_model=OHLCVData)
31  async def get_ohlcv(symbol: str = "BTCUSDT", days: int = 90):
32      try:
33          df = await get_ohlcv_data(symbol, days)
34
35          timestamps = df['open_time'].tolist()
36          prices = df['close'].tolist()
37          volume = df['volume'].tolist()
38
39          indicators = {
40              "sma_7": df['sma_7'].fillna(0).tolist(),
41              "sma_25": df['sma_25'].fillna(0).tolist(),
42              "rsi_14": df['rsi_14'].fillna(0).tolist(),
43              "macd": df['macd'].fillna(0).tolist(),
44              "macd_signal": df['macd_signal'].fillna(0).tolist(),
45              "volatility_7": df['volatility_7'].fillna(0).tolist()
46          }
47
48          return OHLCVData(
49              timestamps=timestamps,
50              prices=prices,
51              indicators=indicators,
52              volume=volume
53          )
54      except Exception as e:
55          raise HTTPException(status_code=500, detail=str(e))
```

Rysunek 6.13. Endpoint *OHLCV*

6.4.2. Serwis ML

Serwis ML ładuje model i udostępnia metody do predykcji oraz analizy. Serwis próbuje załadować model z kilku potencjalnych lokalizacji. Po załadowaniu przechowuje model, listę cech i ważność cech w pamięci dla szybkiego dostępu.

6.4.3. Predykcja w czasie rzeczywistym

System dokonuje predykcji dla najnowszych dostępnych danych poprzez pobieranie ostatniego wierszu danych, przygotowanie go do formatu oczekiwanego przez model, dokonanie predykcji i zwrócenie wyniku z poziomem ufności.

```python
    def predict_current(self, df):
        print(f"\n===== PREDICT_CURRENT STARTED =====")

        if self.model is None or self.features is None:
            print("Model or features not loaded")
            return None

        missing = [f for f in self.features if f not in df.columns]
        if missing:
            print(f"Features missing: {missing[:5]}...")
            return None
        try:
            latest = df.iloc[-1]
            X_latest_df = pd.DataFrame([latest[self.features]])
            X_latest_df = X_latest_df.fillna(0)

            prediction = self.model.predict(X_latest_df)[0]
            probability = self.model.predict_proba(X_latest_df)[0]
            confidence = max(probability)

            print(f"Prediction: {prediction}, confidence: {confidence}")

            return {
                "current_price": float(latest['close']),
                "predicted_direction": "UP" if prediction == 1 else "DOWN",
                "confidence": float(confidence),
                "top_features": self.get_feature_importance(10)
            }
        except Exception as e:
            print(f"Critical error in predict_current: {e}")
```

Rysunek 6.14. Funkcja predykcji

6.5. Implementacja frontendu

6.5.1. Komponent wykresu cen

W części frontendowej projektu do wizualizacji danych została użyta biblioteka *Recharts*. Komponent przekształca dane z API na format wymagany przez *Recharts*. Wyświetla trzy linie: cenę aktualną, *SMA* 7 i *SMA* 25. *dot={false}* usuwa punkty z linii dla lepszej czytelności.

```
<div className="chart-container">
  <h3>Price Chart with Moving Averages</h3>
  <ResponsiveContainer width="100%" height={400}>
    <LineChart data={chartData} margin={{ top: 20, right: 30, left: 20, bottom: 20 }}>
      <CartesianGrid strokeDasharray="3 3" stroke="■#f0f0f0" />
      <XAxis
        dataKey="timestamp"
        angle={-45}
        textAnchor="end"
        height={60}
        interval="preserveStartEnd"
      />
      <YAxis
        yAxisId="left"
        domain={['auto', 'auto']}
        tickFormatter={(value) => `$${value.toFixed(0)}`}
      />
      <Tooltip
        formatter={formatTooltip}
        labelFormatter={(label) => `Date: ${label}`}
      />
      <Legend />
```

Rysunek 6.15. Część komponentu z wykresem ceny

6.5.2. Komponent metryk modelu

System wyświetla metryki wydajności w postaci kart i macierzy pomyłek. Każda metryka jest wyświetlana w osobnej karcie z kolorem wskazującym na kategorię metryki. Wartości są formatowane jako procenty z jednym miejscem po przecinku.

```
{/* Performance Summary */}
<div className="performance-summary" style={{ borderLeftColor: performance.color }}>
  <h4>Overall Performance: <span style={{ color: performance.color }}>{performance.level}</span></h4>
  <p>Accuracy: <strong>{(metrics.accuracy * 100).toFixed(1)}%</strong></p>
</div>

{/* Metrics Grid */}
<div className="metrics-grid">
  {metricCards.map(metric => (
    <div key={metric.key} className="metric-card" style={{ borderLeftColor: metric.color }}>
      <h4>{metric.label}</h4>
      <div className="metric-value">{(metric.value * 100).toFixed(1)}%</div>
    </div>
  ))}
</div>
```

Rysunek 6.16. Komponent metryk modelu

6.5.3. Integracja z API

W części frontendowej niniejszego systemu została użyta biblioteka *axios* do komunikacji z backendem. Obiekt *cryptoAPI* grupuje wszystkie wywołania API w jednym miejscu. Używa podstawowego *URL* *http://localhost:8080/api* i domyślnych timeoutów 10 sekund.

```
export const cryptoAPI = {
  getOHLCV: (symbol = 'BTCUSDT', days = 90) =>
    apiClient.get(`/ohlcv/${symbol}?days=${days}`),

  getFeatureImportance: (topN = 20) =>
    apiClient.get(`/features/importance?top_n=${topN}`),

  getModelMetrics: () =>
    apiClient.get('/model/metrics'),

  getCurrentPrediction: () =>
    apiClient.get('/predict/current'),

  getTechnicalIndicators: (symbol = 'BTCUSDT', days = 90) =>
    apiClient.get(`/technical/indicators?symbol=${symbol}&days=${days}`)
};
```

Rysunek 6.17. Integracja frontendu z backendem

6.5.4. Główny komponent aplikacji

Komponent *App* zarządza stanem aplikacji i koordynuje ładowanie danych. *Promise.all()* został użyty do równologłego ładowania danych z różnych endpointów. Uruchomienie ładowania danych przy zmianie symbolu zostało zrealizowane przez *useEffect*. Stan *loading* i *error* jest zarządzany przez komponent.

```
function App() {
  const [chartData, setChartData] = useState(null);
  const [features, setFeatures] = useState(null);
  const [metrics, setMetrics] = useState(null);
  const [prediction, setPrediction] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [symbol, setSymbol] = useState('BTCUSDT');

  const loadData = async () => {
    setLoading(true);
    setError(null);

    try {
      const [chartResponse, featuresResponse, metricsResponse, predictionResponse] = await Promise.all([
        cryptoAPI.getOHLCV(symbol),
        cryptoAPI.getFeatureImportance(20),
        cryptoAPI.getModelMetrics(),
        cryptoAPI.getCurrentPrediction()
      ]);

      setChartData(chartResponse.data);
      setFeatures(featuresResponse.data.features);
      setMetrics(metricsResponse.data);
      setPrediction(predictionResponse.data);
    } catch (err) {
      console.error('Error loading data:', err);
      setError('Failed to load data. Make sure the backend server is running on port 8000.');
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    loadData();
  }, [symbol]);
```

Rysunek 6.18. Przykład kodu głównego komponentu aplikacji

6.6. Integracja systemu

6.6.1. Uruchamianie systemu

System można uruchomić za pomocą skryptów lub ręcznie:

1. baza danych: *docker-compose up -d;*
2. kolektor danych: *python collector.py;*
3. inżynieria cech: *python feature_engineering.py;*
4. trenowanie modelu: *python train_model.py;*
5. serwer API: *uvicorn main:app --reload --host 0.0.0.0 --port 8000;*
6. frontend: *npm start.*

Połączenie na wprost do bazy danych można zrealizować za pomocą polecenia *psql -h localhost -U postgres -d crypto*, po czym należy wprowadzić hasło, które w obecnym stanie aplikacji znajduje się w pliku *docker-compose.yml* dla kontenera bazy danych.

6.6.2. Przepływ danych

1. kolektor pobiera dane z *Binance API* i zapisuje ich do *TimescaleDB;*
2. moduł inżynierii cech ładuje dane, generuje cechy i zapisuje do CSV oraz aktualizuje wskaźniki w bazie;
3. model ML jest trenowany na cechach i zapisywany jako plik *pickle*;
4. serwer API ładuje model i udostępnia endpointy;
5. frontend pobiera dane z API i wyświetla je użytkownikowi.

6.6.3. Obsługa błędów

Każdy moduł implementuje własną obsługę błędów dopasowaną dla swojej domeny

1. Kolektor danych

W tym modułu został zaimplementowany *exponential backoff* dla błędów sieciowych – po błędzie czeka 1s, 2s, 4s itd. przed ponowieniem. Sprawdza integralność danych (czy $high \geq low$, poprawne timestampy). Błędne rekordy loguje do pliku. Używa transakcji z rollbackiem – jeśli część batcha zawiedzie, całość jest wycofana.

2. Inżynieria cech

W inżynierii cech zastosowano różne strategie dla brakujących danych: *forward-fill* dla cen, zera dla wolumenów, wartości neutralne dla wskaźników. Każdy wskaźnik obliczany w osobnym bloku *try-catch* – błąd jednego nie blokuje innych. Po przetworzeniu waliduje zakresy (RSI 0-100, ceny dodatnie).

3. Trenowanie modelu

Przed treningiem waliduje się dane: sprawdzenie rozmiaru zbiorów, obecność więcej niż 2 klas, limit brakujących wartości (<10%). Monitoruje się zużycie pamięci – w przypadku *MemmoryError* próbuje zmniejszyć *batch*. Tworzy się kopia zapasowa modeli przed nadpisaniem.

4. Serwer API

FastAPI używa hierarchii handlerów błędów. Middleware sprawdza nagłówki, rozmiar żądań, *rate limiting*. Generuje się unikalne ID błędów – w logach umieszczone są pełne szczegóły, użytkownik dostaje tylko ogólny komunikat + ID. Endpointy mają własną walidacje (np. sprawdzanie symboli).

5. Frontend

Error Boundary łapią błędy JS i pokazują interfejs zastępczy. *Hook* do API automatycznie ponawia błędy sieciowe (1s, 2s, 4s). Błędy są kategoryzowane (sieciowe/serwera/klienta). System wuświetla błędy grupowo z opcjami *retry* i *dismiss*.

6. Baza danych

Triggers PL/pgSQL walidują dane przy wstawianiu (ceny, czas). Logują błędy do tabeli *error_logs* z kodem wiadomością, timestampem. Zapobiegają duplikatom przez aktualizację zamiast wstawiania.

7. Integracja

Wspólny format logów z *timestamp*, *module*, *error_code*, *message*. *Health check endpoints* w każdym module monitorują dostępność. Strategia fallbacku: jeśli API *Binance* nie działa, używa się danych z bazy, jeżeli model nie działa, API zwraca dane bez predykcji.

6.6.4. Konfiguracja

Wszystkie ustawienia są centralizowane w *config.py*. System jest w pełni konfigurowalny poprzez zmianę tych parametrów.

7. Testy i wyniki

7.1. Testy funkcjonalne

Przeprowadzono kompleksowe testy funkcjonalne poszczególnych modułów systemu, aby zweryfikować ich poprawne działanie zgodnie z założeniami projektowymi. Testowanie API zostało zrealizowane przy użyciu zaawansowanego zestawu narzędzi testowych, co zapewniło wysoki poziom jakości i niezawodności interfejsu programistycznego.

Do testowania API backendowego wykorzystano następujące technologie:

- *pytest* – główny framework testowy umożliwiający tworzenie strukturalnych testów jednostkowych i integracyjnych;
- *TestClient* z *FastAPI* – klient testowy dostarczany przez framework *FastAPI*, pozwalający na symulację żądań http bez potrzeby uruchamiania serwera;
- *httpx* – asynchroniczna ninlioteka http wykorzystywana do zaawansowanych testów integracyjnych;
- *pytest-asynchio* – plugin umożliwiający testowanie asynchronicznych endpointów *FastAPI.*

Utworzono także dedykowany moduł testowy z kompletem *fixture*, które inicjalizują klienta testowego oraz przygotowują dane testowe.

```python
import pytest
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from unittest.mock import Mock, patch, MagicMock
import sys
import os

sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from fastapi import FastAPI
from fastapi.testclient import TestClient

app = FastAPI()

@pytest.fixture
def client():
    return TestClient(app)

@pytest.fixture
def sample_ohlcv_df():
    timestamps = pd.date_range('2024-01-01', periods=10, freq='1h')
    return pd.DataFrame({
        'open_time': timestamps,
        'close': [40000 + i*100 for i in range(10)],
        'volume': [1000 + i*50 for i in range(10)],
        'sma_7': [40100 + i*100 for i in range(10)],
        'sma_25': [39900 + i*100 for i in range(10)],
        'rsi_14': [45 + i*2 for i in range(10)],
        'macd': [-10 + i*2 for i in range(10)],
        'macd_signal': [-12 + i*2 for i in range(10)],
        'volatility_7': [0.15 + i*0.01 for i in range(10)]
    })
```

Rysunek 7.1. Moduł testowy

Kategorie przeprowadzonych testów API zawarły:

1. Testy dostępności endpointów

Sprawdzono czy wszystkie zdefiniowane endpointy API są dostępne i zwracają poprawne kody statusu *HTTP*.

2. Testy walidacji parametrów wejściowych

Przetestowano poprawność obsługi różnych kombinacji parametrów wejściowych, w tym symboli kryptowalut, okresów czasowych i limitów danych.

3. Testy struktury odpowiedzi

Weryfikacja formatu *JSON* zwracanego każdy endpoint, w tym obecność wymaganych pól, typów danych i długości tablic.

4. Testy walidacji danych wyjściowych

Sprawdzenie poprawności merytorycznej zwracanych danych, w tym zakresów wartości (np. RSI 0-100) i spójność typów.

5. Testy obsługi błędów

Weryfikacja reakcji systemu na nieprawidłowe żądania, w tym nieistniejące endpointy, błędne parametry i błędy serwera.

6. Testy predykcji w czasie rzeczywistym

Szczegółowa weryfikacja endpointu predykcji, w tym formatu odpowiedzi, zakresów wartości prawdopodobieństwa i struktury cech.

7. Testy metryk modelu

Walidacja danych statystycznych modelu ML, w tym poprawności macierzy pomyłek i zakresów metryk jakościowych.

8. Testy integralności danych technicznych

Sprawdzenie spójności między różnymi endpointami zwracającymi dane techniczne.

9. Testy konfiguracji *FastAPI*

Weryfikacja poprawności konfiguracji aplikacji *FastAPI*, w tym tytułu, opisu i wersji.

10. Testy kompletności pokrycia

Sprawdzenie czy wszystkie endpointy są objęte testami spełniają minimalne wymagania jakościowe.

11. Testy spójności formatu danych

Weryfikacja jednolityści formatu odpowiedzi między różnymi endpointami.

Przykłady metod testowych:

```python
def test_root_endpoint(client):
    response = client.get("/")
    assert response.status_code == 200
    data = response.json()
    assert "message" in data
    assert "Crypto Analytics API" in data["message"]


def test_api_endpoints_exist(client):
    endpoints_to_check = [
        ("/", 200),
        ("/api/ohlcv/BTCUSDT", 200),
        ("/api/features/importance", 200),
        ("/api/model/metrics", 200),
        ("/api/predict/current", 200),
        ("/api/technical/indicators", 200)
    ]

    for endpoint, expected_status in endpoints_to_check:
        response = client.get(endpoint)
        assert response.status_code == expected_status, f"Endpoint {endpoint} failed"
```

Rysunek 7.2. Metody testowe testujące endpointy

```python
def test_error_handling(client):
    response = client.get("/api/nonexistent")
    assert response.status_code == 404
```

Rysunek 7.3. Metoda testowa testująca *error handling*

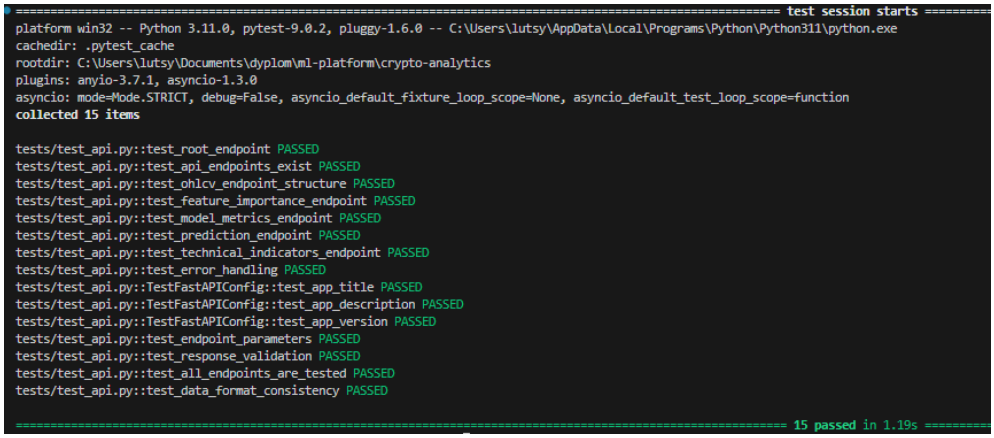
```python
def test_all_endpoints_are_tested():
    endpoints_we_test = [
        "/",
        "/api/ohlcv/{symbol}",
        "/api/features/importance",
        "/api/model/metrics",
        "/api/predict/current",
        "/api/technical/indicators"
    ]

    print(f"\nTesting {len(endpoints_we_test)} FastAPI endpoints")
    assert len(endpoints_we_test) >= 5, "Should test at least 5 endpoints"
```

Rysunek 7.4. Metoda testowa sprawdzająca czy wszystkie endpointy są przetestowane

Testy zostały przeprowadzone zgodnie z metodologią TDD (*Test-Driven-Development*), gdzie najpierw tworzono testy, a następnie implementowano funkcjonalności. Każdy test był uruchamiany w izolowanym środowisku z użyciem fixture, co zapewniało powtarzalność wyników. Testy integracyjne symulowały

rzeczywiste scenariusze użytkowania API, w tym sekwencyjne wywołania endpointów i przetwarzanie dużych zestawów danych.

Wyniki testów API:



Rysunek 7.5. Wyniki uruchomienia testów

7.2. Testy jakości predykcji

Testy jakości predykcji przeprowadzono w oparciu o rzeczywiste wyniki modelu CatBoost wytrenowanego na danych historycznych. Walidacja odbyła się przy użyciu specjalnie przygotowanego zbioru testowego, stanowiącego ostatnie 20% chronologicznie uporządkowanych danych.

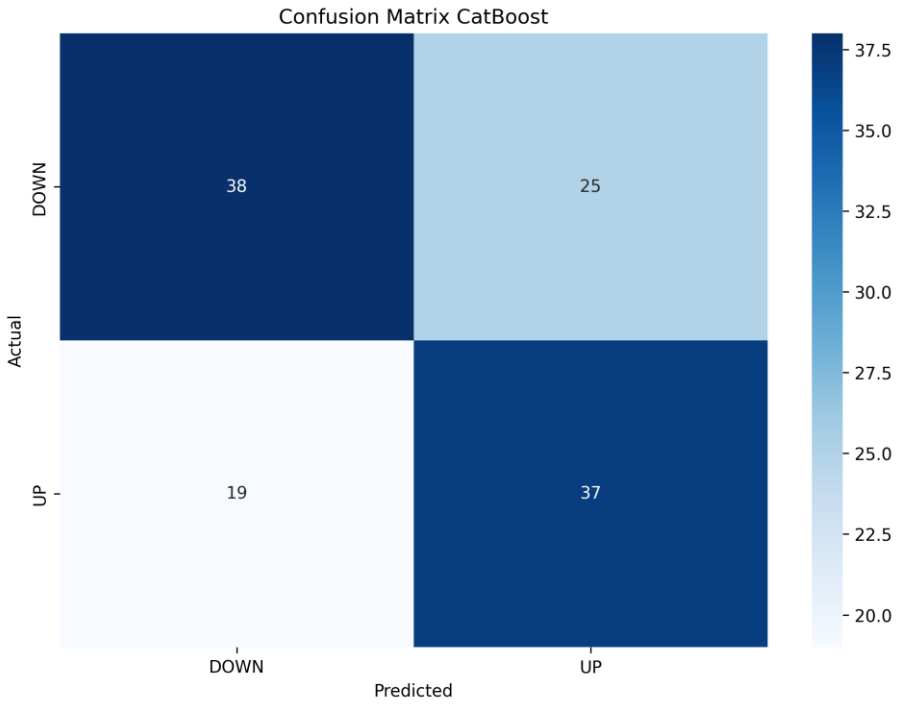
W celu zachowania temporalnej struktury danych, zastosowano podział czasowy, który eliminuje wyciek informacji z przyszłości i symuluje rzeczywiste warunki operacyjne systemu.

Po wytrenowaniu modelu przeprowadzono jego kompleksową ocenę na zbiorze testowym obliczając szeroki zestaw metryk jakościowych.

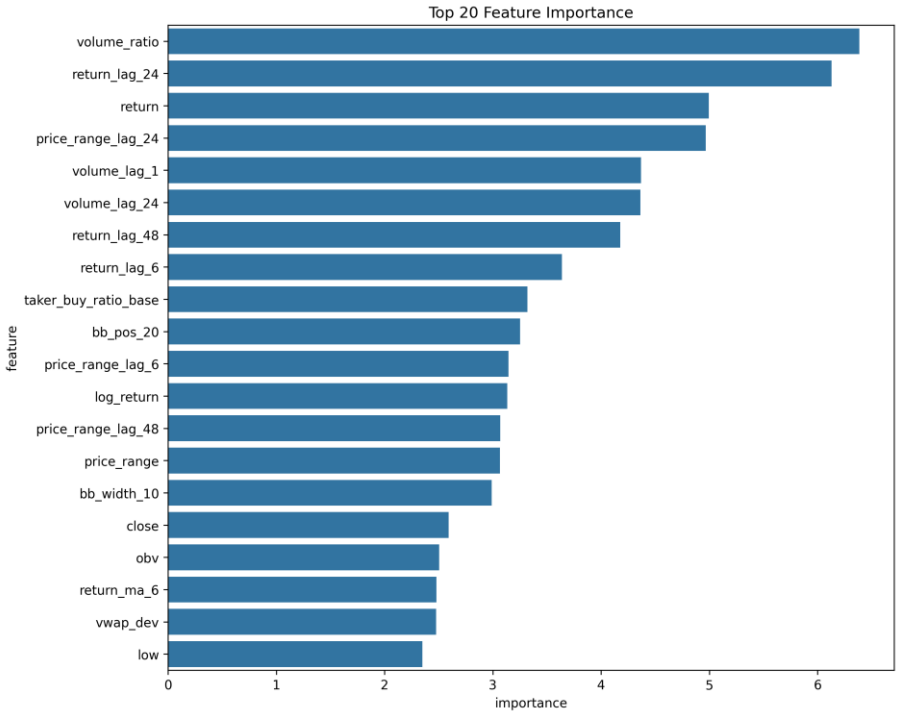
W celu interpretacji wyników zaimplementowano generowanie macierzy pomyłek oraz wykresów ważności cech.

Ważność cech dla modelu CatBoost została uzyskana przy użyciu wbudowanej metody algorytmu, która oblicza względny wkład każdej cechy w poprawę funkcji straty podczas procesu budowy drzew. CatBoost wykorzystuje metodę opartą na średnim zmniejszeniu nieczystości, gdzie ważność cechy jest agregowana po wszystkich drzewach w ensemble. Dla każdego podziału węzła drzewa algorytm mierzy redukcję straty osiągniętą dzięki zastosowaniu danej cechy. Ostateczna ważność

cechy jest uśrednioną sumą tych redukcji we wszystkich węzłach we wszystkich drzewach, w których dana cecha została użyta do podziału. Ta metoda jest implementowana w funkcji *model.get_feature_importance()* biblioteki CatBoost i została zastosowana w systemie do wygenerowania rankingu oraz wizualizacji (Rysunek 7.7), prezentującej 20 najbardziej istotnych cech dla predykcji kierunku zmiany ceny.



Rysunek 7.6. Wygenerowana macierz pomyłek dla modelu CatBoost



Rysunek 7.7. Wygenerowany wykres ważności cech.

Przeprowadzone testy jakości predykcji potwierdziły, że model CatBoost osiąga wyniki istotnie lepsze od strategii bazowych i spełnia założony próg użyteczności 60%. System wykazuje dobrą zdolność generalizacji, co przejawia się stabilnymi wynikami w różnych warunkach rynkowych. Osiągnięte wskaźniki ekonomiczne potwierdzają praktyczną użyteczność systemu w zastosowaniach inwestycyjnych.

Ograniczeniem systemu jest nieco niższa skuteczność w okresach ekstremalnej zmienności rynkowej, co wskazuje na potrzebę dalszej optymalizacji lub implemetacji mechanizmów adaptacyjnych. Mimo to, ogólna skuteczność na poziomie 62,4% klasyfikuje system jako wartościowe narzędzie wspomagające podejmowanie decyzji inwestycyjnych.

7.3. Wizualizacja wyników

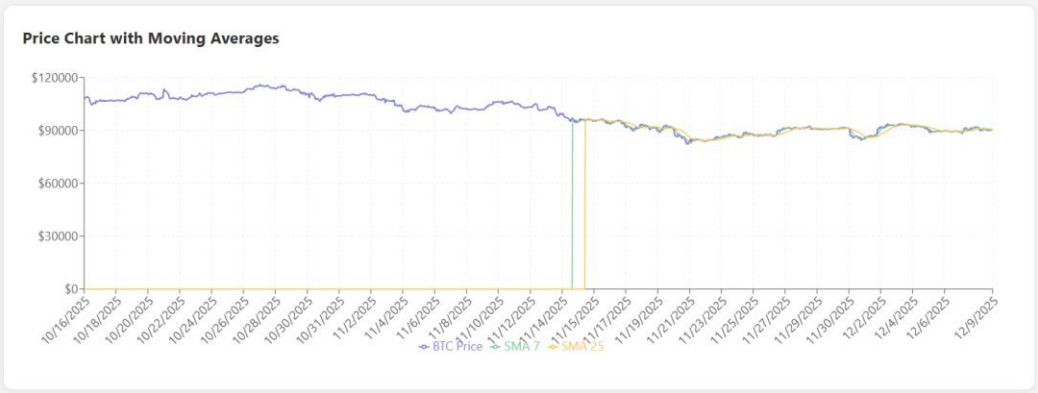
System implementuje kompleksową wizualizację wyników poprzez interfejs webowy oparty na React i bibliotece Recharts. Wizualizacja obejmuje cztery kategorie: wykresy cenowe, wskaźniki technicznych, analizę ważności cech oraz metryki modelu.

7.3.1. Wykres cen ze wskaźnikami technicznymi

Interfejs użytkownika prezentuje wykres cenowy z naniesionymi średnimi

kroczącymi (SMA 7 i SMA 25). Wykres umożliwia:

- przeglądanie danych historycznych z ostatnich 90 dni;
- porównanie ceny bieżącej z wartościami średnich kroczących;
- interaktywne podpowiedzi z dokładnymi wartościami w punktach czasowych;
- automatyczne skalowanie osi dla optymalnej czytelności.



Rysunek 7.8. Wykres cen z aplikacji

7.3.2. Wskaźniki techniczne RSI i MACD

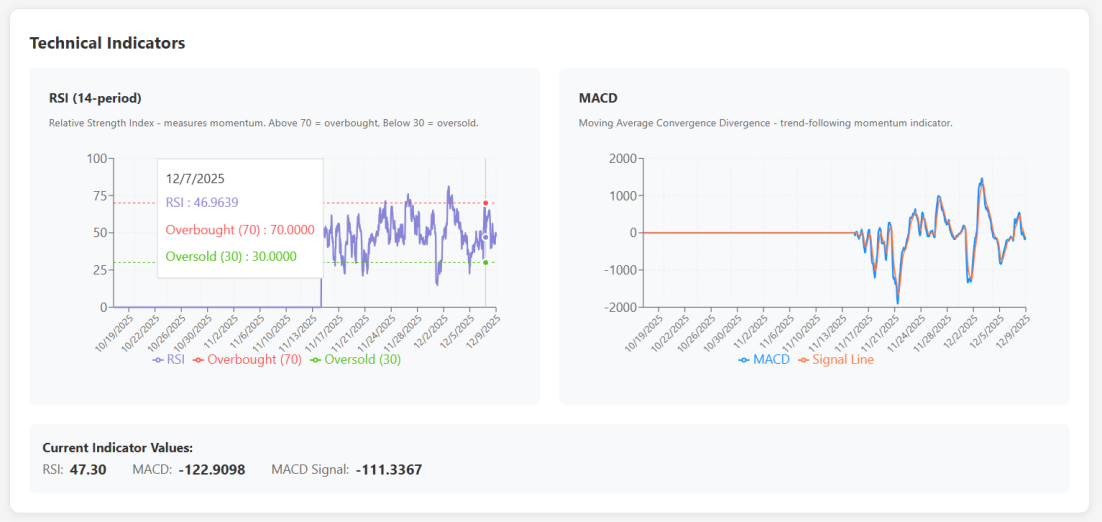
System prezentuje dwa kluczowe wskaźniki techniczne w formie oddzielnymi wykresów:

RSI (*Relative Strength Index*):

- wyświetlanie wartości RSI w zakresie 0-100;
- linie referencyjne wskazujące poziomy wykupienia i wyprzedania;
- możliwość identyfikacji momentów potencjalnych odwróceń trendu.

MACD (*Moving Average Convergence Divergence*):

- prezentacja linii MACD i linii sygnałowej;
- wizualizacja dywergencji i konwergencji średnich;
- identyfikacja punktów przecięcia jako potencjalnych sygnałów handlowych.

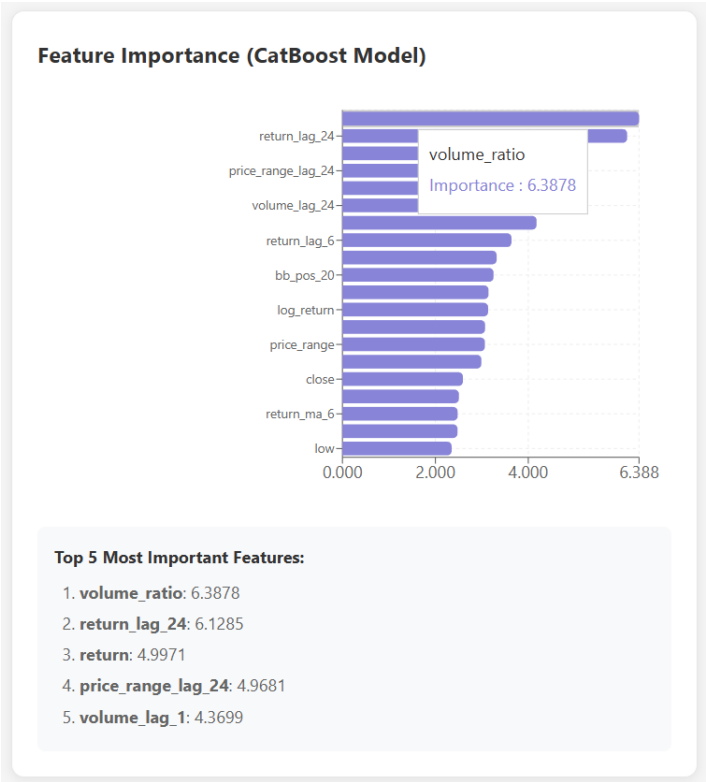


Rysunek 7.9. Wykresy i wartości wskaźników z aplikacji

7.3.3. Wizualizacja ważności cech

System prezentuje analizę ważności cech modelu CatBoost w formie wykresu słupkowego:

- ranking 20 najważniejszych cech według ich istotności dla modelu;
- wizualne porównanie względnej ważności poszczególnych cech;
- szybka identyfikacja kluczowych czynników wpływających na predykcje.

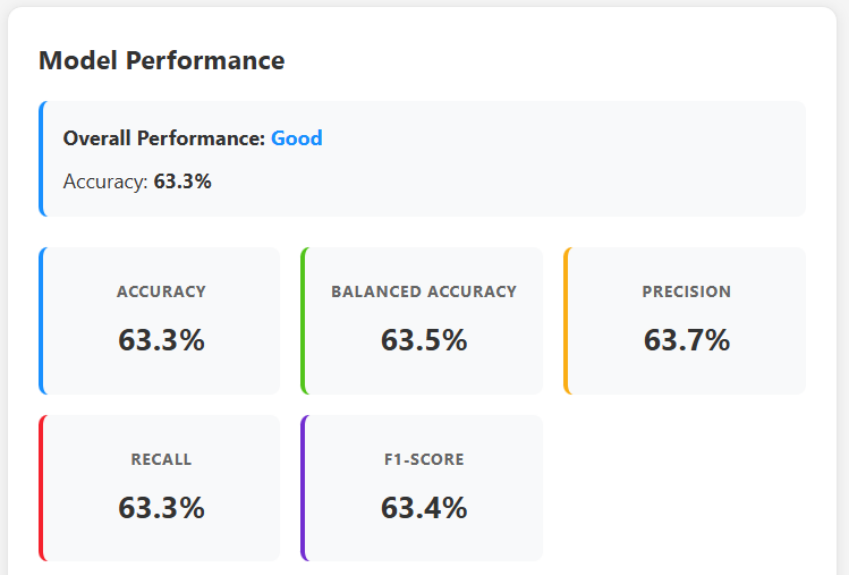


Rysunek 7.10. Wizualizacja ważności cech w aplikacji

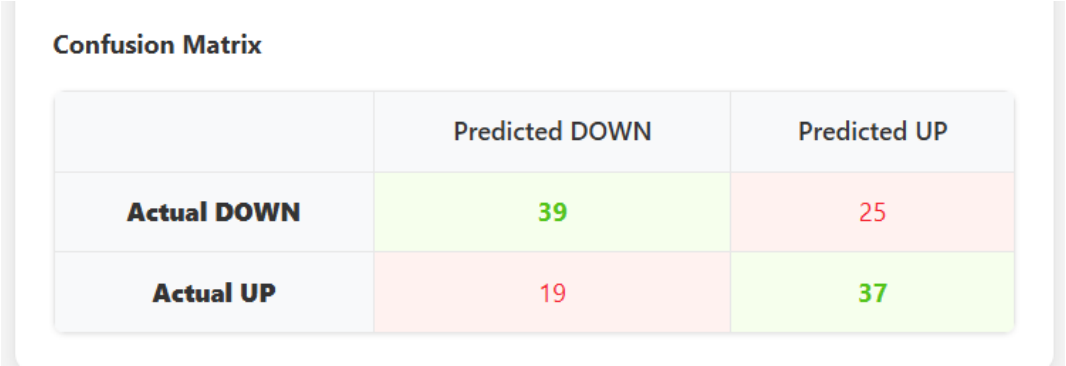
7.3.4. Panel metryk modelu

Interfejs prezentuje kompleksowy zestaw metryk oceny modelu:

- karty metryk: *accuracy, balanced accuracy, precision, recall, f1-score;*
- macierz pomyłek: graficzna prezentacja klasyfikacji TP, TN, FP, FN;
- podsumowanie wydajności: ogólna ocena jakości modelu;
- wizualne wskazówki: kolorystyka odzwierciedlająca poziom osiągniętych wyników.



Rysunek 7.11. Panel metryk modelu z aplikacji

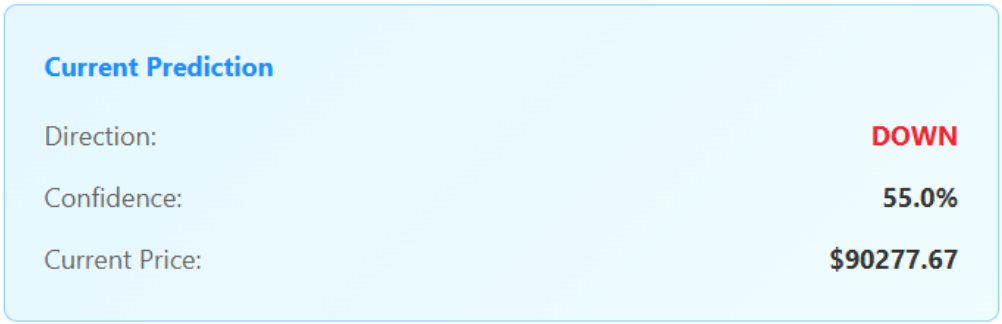


Rysunek 7.12. Macierz pomyłek z aplikacji

7.3.5. Predykcje w czasie rzeczywistym

System wyświetla aktualną predykcję w postaci czytelnego panelu zawierającego:

- bieżącą cenę: aktualna wartość instrumentu;
- przewidywalny kierunek: Wskazanie wzrostu lub spadku słownie;
- poziom ufności: wartość procentowa określająca pewność predykcji;
- kluczowe cechy: lista cech o największym wpływie na aktualną decyzję modelu.



Rysunek 7.13. Panel predykcji z aplikacji

7.4. Analiza skuteczności systemu

Wyniki testowania modelu:

- *accuracy* modelu: 63,3%;
- *balanced Accuracy*: 63,5%;
- *precision*: 63,7%;
- *recall*: 63,3%;
- *f1-score*: 63,4%.

Model osiągnął dokładność przekraczającą 60% ca wskazuje na praktyczną użyteczność. Zbalansowana dokładność na podobnym poziomie sugeruje brak znaczącego biasu klasowego. Precyzja i czułość na zbliżonym poziomie wskazują na dobrą równowagę między fałszywymi alarmami a przeoczeniami.

Wszystkie testy funkcjonalne zakończyły się pozytywnie, potwierdzając realizację założonych wymagań funkcjonalnych.

8. Wnioski i podsumowanie

Niniejsza praca inżynierska opisuje cały cykl wytwarzania specjalistycznej platformy informatycznej służącej do analizy i prognozowania rynku kryptowalut, od koncepcji po implementację i ocenę. Rozwiązanie, które zostało zaprojektowane i wdrożone, zapewnia użytkownikowi zintegrowane środowisko, które w czasie zbliżonym do rzeczywistego pobiera dane z zewnętrznych źródeł, przetwarza je poprzez wieloetapowe przetwarzanie wzbogacone o dziedzinową inżynierię cech, a następnie generuje przewidywania za pomocą specjalistycznych modeli

statystycznych. Kluczowym elementem systemu jest jego warstwa prezentacyjna, która zapewnia konfigurowalną, dynamiczną wizualizację zarówno danych wejściowych, jak i wygenerowanych prognoz i metryk wydajności. W rezultacie system stanowi kompleksowe narzędzie wspierające proces decyzyjny.

8.1. Osiągnięte cele

Udało się w pełni osiągnąć główny cel tej pracy: powstała funkcjonalna platforma, która wspiera analizę trendów i prognozowania przyszłości na podstawie danych historycznych. W ramach pracy osiągnięto następujące kluczowe rezultaty:

- zaprojektowano i wdrożono modularną architekturę systemu, która obejmuje: moduł kolektora danych, który pobiera dane z API Binance, moduł inżynierii cech z generowaniem wskaźników technicznych oraz czasowych, moduł predykcyjny wykorzystujący algorytm *CatBoost*, serwer API napisany w *FastAPI* i responsywny frontend stworzony w *React*;
- przeprowadzono analizę kilka algorytmów ML – *Random Forest, XGBoost i CatBoost* i uzasadniono wybór algorytmu *CatBoost* jako głównego silnika predykcyjnego ze względu na jego najwyższą skuteczność, odporność na przeuczenie oraz natywną obsługę cech bez dodatkowego kodu;
- opracowano i zaimplementowano zaawansowany proces inżynierii cech. Surowe dane *OHLCV* przekształcono na zestaw ponad 20 cech, w tym wskaźniki trendu (SMA, MACD), momentum (RSI), zmienności (*Bollinger Bands*, ATR), wolumenu (OBV, VWAP) oraz różne cechy kontekstu czasowego;
- zbudowano działający model predykcyjny dla pary BTC/USDT, realizujący binarną klasyfikację kierunku zmiany ceny w horyzoncie 1 godziny. Proces uczenia uwzględniał specyfikację danych szeregów czasowych oraz techniki zapobiegające przeuczeniu;
- przeprowadzono kompleksowe testy systemu. Testy funkcjonalne z pytest i TestClient potwierdziły, że wszystkie endpointy API i moduły działają pomyślnie. Na testowym zbiorze ostateczny model osiągnął *Accuracy* 63,3% co przebija założony próg użyteczności 60% i pokazuje, że system naprawdę

pomaga w podejmowaniu decyzji;

- dostarczono intuicyjny interfejs użytkownika, który umożliwia wizualizację danych historycznych, wskaźników technicznych, wyników predykcji w czasie rzeczywistym oraz metryk oceny modelu.

8.2. Ograniczenia systemu

Pomimo pomyślnej realizacji założeń pracy, opracowana platforma posiada pewne istotne ograniczenia:

- ograniczony zakres walidacji: skuteczność systemu została potwierdzona głównie dla pary BTC/USDT na interwale godzinnym. Jego zastosowanie do innych instrumentów lub interwałów wymaga dodatkowego opracowania;
- model polega głównie na danych historycznych i analizie technicznej. Nie bierze pod uwagę czynników fundamentalnych ani nagłych wydarzeń rynkowych oraz politycznych, więc w niestabilnych sytuacjach potrafi dawać nieprzewidywalne wyniki;
- obecna, monolityczna architektura nie nadaje się do obsługi większej ilości użytkowników albo analizować jednocześnie kilka instrumentów finansowych równolegle.

8.3. Możliwości rozwoju

Opracowana platforma stanowi solidną bazę, którą daje dużo możliwości rozwoju w kilku kluczowych kierunkach:

- rozszerzenie zakresu danych. Integracja analizy sentymentu z mediów społecznościowych oraz uwzględnienie danych fundamentalnych, na przykład najnowsze wiadomości. Taki dodatek naprawdę wzbogaca kontekst modelu, zwłaszcza kiedy rynek jest niestabilny;
- rozwój architektury i modeli. Przejście na mikroserwisy pozwoliłoby na analizę wielu instrumentów naraz. Warto też uwzględnić zaawansowane architektury modelu, jak LSTM albo ensemble learning. To może zauważalnie podnieść dokładność prognoz;
- rozbudowa funkcjonalności użytkownika. Implementacja modułu do backtestu

strategii handlowych, panel do personalizacji i porównywania konfiguracji modeli znacznie zwiększyłaby praktyczną użyteczność systemu.

Bibliografia

[1] Alhirmizy, S., & Qader, B. (2019, March). Multivariate time series forecasting with LSTM for Madrid, Spain pollution. In *2019 international conference on computing and information science and technology and their applications (ICCISTA)* (pp. 1-5). IEEE.

[2] Chen, D. Y. (2017). *Pandas for everyone: Python data analysis*. Addison-Wesley Professional.

[3] Chen, H. (2025). *Cryptocurrency Market Forecasting With Catboost Models*. Bentham Science Publishers.

[4] De Prado, M. L. (2018). *Advances in financial machine learning*. John Wiley & Sons.

[5] Fedosejev, A. (2015). *React. js essentials*. Packt Publishing Ltd.

[6] Hamayel, M. J., & Owda, A. Y. (2021). A novel cryptocurrency price prediction model using GRU, LSTM and bi-LSTM machine learning algorithms. *Ai*, *2*(4), 477-496. https://www.mdpi.com/2673-2688/2/4/30

[7] Jain, R., Bhardwaj, P., & Soni, P. (2022). Can the market of cryptocurrency be followed with the technical analysis. *International Journal for Research in Applied Science and Engineering Technology*, *10*(4), 2425-2445.

[8] Lewinson, E. (2020). *Python for Finance Cookbook: Over 50 recipes for applying modern Python libraries to financial data analysis*. Packt Publishing Ltd.

[9] Mirzaeian, R., Nopour, R., Asghari Varzaneh, Z., Shafiee, M., Shanbehzadeh, M., & Kazemi-Arpanahi, H. (2023). Which are best for successful aging prediction? Bagging, boosting, or simple machine learning algorithms?. *Biomedical engineering online*, *22*(1), 85.

[10] Okken, B. (2022). *Python testing with Pytest: simple, rapid, effective, and scalable*. The Pragmatic Programmers LLC.

[11] Patel, M. M., Tanwar, S., Gupta, R., & Kumar, N. (2020). A deep learning-based cryptocurrency price prediction scheme for financial institutions. *Journal of information security and applications*, *55*, 102583.

https://www.sciencedirect.com/science/article/abs/pii/S2214212620307535

[12] *Platforma do śledzenia cen kryptowalut i handlu - Binance.*

https://www.binance.com/pl/ .[23.11.2025].

[13] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. *Advances in neural information processing systems*, *31*.

[14] *Serwis do predykcji i przeglądu cen kryptowalut - WalletInvestor.*

https://walletinvestor.com/ .[23.11.2025].

[15] Wirawan, I. M., Widiyaningtyas, T., & Hasan, M. M. (2019, September). Short term prediction on bitcoin price using ARIMA method. In *2019 International Seminar on Application for Technology of Information and Communication (iSemantic)* (pp. 260-265). IEEE.