

# 1 Linear Regression (OLS / Ridge / Lasso) - dokumentacja modelu

## 1.1 Nagranie wideo (YouTube)

Wideo prezentujące projekt oraz sposób uruchomienia i demonstrację działania:

- <https://www.youtube.com/watch?v=gqYQ3Df0xIU>
- 

## 1.2 Cel modelu

Model liniowy służy do prognozy **zwrotu BTC na następny dzień** (1-step ahead):

- `ret_btc_next = pct_change(close_btc).shift(-1)`

Czyli w dniu  $t$  używamy cech z dnia  $t$ , aby przewidzieć zwrot w dniu  $t+1$ .

---

## 1.3 Dane i przygotowanie datasetu

### 1.3.1 Źródło danych

- `backend/data/historical_data.parquet`

Po wczytaniu dane są sortowane chronologicznie po `open_time` (jeśli kolumna istnieje).

### 1.3.2 Budowa targetu

W kodzie target jest tworzony następująco:

- `ret_btc = close_btc.pct_change()`
- `ret_btc_next = ret_btc.shift(-1)`

Następnie usuwane są wiersze z brakami w cechach lub w target (`dropna`).

---

## 1.4 Cechy wejściowe (FEATURE\_COLUMNS)

W tym modelu używamy **surowych poziomów** (OHLC, wolumen, liczba transakcji, indeks SPX oraz makro).

Ceny (OHLC) dla krypto:

- `open_btc, high_btc, low_btc, close_btc`
- `open_eth, high_eth, low_eth, close_eth`
- `open_bnb, high_bnb, low_bnb, close_bnb`
- `open_xrp, high_xrp, low_xrp, close_xrp`

Wolumen i liczba transakcji:

- `volume_btc, volume_eth, volume_bnb, volume_xrp`
- `num_trades_btc, num_trades_eth, num_trades_bnb, num_trades_xrp`

Indeks SPX:

- `open_spx, high_spx, low_spx, close_spx, volume_spx`

Makro:

- `gdp, unrate`

Target:

- `ret_btc_next`
- 

## 1.5 Split danych (train/test)

Podział jest chronologiczny, bez losowego mieszania:

- `test_size = 0.2`
- `shuffle = False`

Dla uruchomionego runu:

- `n_train = 1680`
- `n_test = 420`

To utrzymuje realistyczny scenariusz predykcji w szeregach czasowych (trenujemy na przeszłości i testujemy na przyszłości).

---

## 1.6 Pipeline i skalowanie

Model jest trenowany jako `sklearn.pipeline.Pipeline`:

1. `StandardScaler()`
2. regresor liniowy: `LinearRegression` / `Ridge` / `Lasso`

Skalowanie jest szczególnie istotne dla modeli z regularyzacją (`Ridge`/`Lasso`), bo współczynniki kary zależą od skali cech.

---

## 1.7 Warianty modelu

Skrypt obsługuje 3 tryby (`-model-type`):

### 1.7.1 1) OLS (`LinearRegression`)

- klasyczna regresja najmniejszych kwadratów bez regularyzacji

### 1.7.2 2) Ridge

- regresja z karą L2: `Ridge(alpha=alpha)`
- stabilizuje współczynniki i zmniejsza wariancję, zwykle pomaga przy współliniowości cech

### 1.7.3 3) Lasso

- regresja z karą L1: `Lasso(alpha=alpha)`
  - może zerować część wag (selekcja cech), co bywa korzystne przy dużej liczbie skorelowanych sygnałów
-

## 1.8 Strojenie hiperparametrów (Lasso)

Jeśli `-model-type lasso`, wykonywane jest proste strojenie `alpha` na zbiorze testowym (chronologicznym):

Testowane wartości:

- [0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1.0]

Wybór “best”:

- minimalne MAE na zbiorze testowym

Z logów:

- **Best alpha:** 0.005
  - **MAE** = 0.016855
  - **RMSE** = 0.023352
- 

## 1.9 Metryki

Raportowane metryki na zbiorze testowym:

- `MAE = mean_absolute_error(y_test, y_pred)`
  - `RMSE = sqrt(mean_squared_error(y_test, y_pred))`
- 

## 1.10 Wyniki eksperymentów (z logów)

### 1.10.1 OLS (LinearRegression)

- **MAE:** 0.043481953958960734
- **RMSE:** 0.0649079753374987
- `n_train=1680, n_test=420`

W tej konfiguracji model liniowy bez regularyzacji nie generalizuje dobrze na test-set.

### 1.10.2 Lasso (z tuningiem alpha)

- **MAE:** 0.01685521013312673
- **RMSE:** 0.0233521064909458
- najlepsze `alpha` = 0.005

Widać, że regularyzacja L1 znacząco poprawia wynik względem OLS.

---

## 1.11 Artefakty i zapis modelu

Model jest zapisywany do:

- `backend/data/linear_regression_btc.pkl`

Format zapisu (`joblib.dump`) zawiera słownik:

- `model`: pipeline (scaler + regressor)
- `features`: lista `FEATURE_COLUMNS`
- `target`: `ret_btc_next`
- `target_column_in_df`: `ret_btc_next`

To umożliwia spójne odtworzenie inferencji (te same cechy w tej samej kolejności + ten sam scaler).

## 2 Random Forest

### 2.1 Zbiór danych

Rozpoczynając pracę nad random forestem korzystaliśmy z danych historycznych, które zawierały następujące wartości:

#### 2.1.1 Ceny otwarte, najwyższe, najniższe, zamknięte

- `open_btc`, `open_eth`, `open_bnb`, `open_xrp`, `open_spx`
- `high_btc`, `high_eth`, `high_bnb`, `high_xrp`, `high_spx`
- `low_btc`, `low_eth`, `low_bnb`, `low_xrp`, `low_spx`
- `close_btc`, `close_eth`, `close_bnb`, `close_xrp`, `close_spx`

#### 2.1.2 Wolumen i liczba transakcji

- `volume_btc`, `volume_eth`, `volume_bnb`, `volume_xrp`, `volume_spx`
- `num_trades_btc`, `num_trades_eth`, `num_trades_bnb`, `num_trades_xrp`, `num_trades_spx`

#### 2.1.3 Indeksy globalne

- `gdp`
- `unrate`

### 2.2 Budowa modelu

Rozpoczynając pracę nad modelem, zdecydowaliśmy, że najważniejszymi parametrami naszego modelu będą:

- `max_depth` - ogranicza głębokość drzewa
- `min_samples_leaf` - minimalna liczba próbek w liściu
- `max_features` - liczba cech, które bierze pod uwagę pojedyncze drzewo
- `n_estimators` - liczba drzew w lesie

- `max_samples` - liczba próbek, na których uczy się każde drzewo
- `criterion` - funkcja oceniająca jakość podziału

Dlatego właśnie to optymalizacją tych parametrów zajęliśmy się w pierwszej kolejności.

## 2.3 Pierwsze wnioski i problemy

Wstępna optymalizacja metodą **Grid Search** wykazała niepokojące tendencje. Model dążył do tworzenia skrajnie płytkich lasów, co sugerowało wysoki poziom szumu w danych.

=== BEST MODEL FROM GRID SEARCH ===

Best hyperparameters:

```
max_depth: 3
min_samples_leaf: 300
max_features: log2
n_estimators: 50
max_samples: 0.5
```

Zaniepokoiły nas następujące rzeczy:

- **Płytke drzewa** — Drzewa o maksymalnej głębokości 3 (`max_depth: 3`), nie pozwalają na wykrycie złożonych zależności i reguł między danymi.
- **Mało drzew** — Większa liczba drzew (`n_estimators: 50`) nie przynosi poprawy, co sugeruje, że dane są mocno zaszumione.
- **Duża liczba próbek na liściach** — Wysoka liczba próbek (`min_samples_leaf: 300`) sprawia, że las mocno uśrednia predykcje i nie wykrywa nowych, krótkoterminowych trendów.

W związku z tym, zdecydowaliśmy, że problem stanowią wykorzystywane przez nas dane, które wprowadzają szum do naszego modelu.

## 2.4 Przetworzenie i dodanie nowych cech

Aby wyeliminować szum i dostarczyć modelowi bardziej czytelne sygnały, dokonaliśmy transformacji danych, wprowadzając nowe grupy cech:

### 2.4.1 Zwroty z cen `ret_`

- `ret_close_*` - dzienna zmiana ceny zamknięcia danego rynku (np. BTC, ETH, SPX) wyrażona jako różnica między dniem bieżącym, a poprzednim.
- `ret_hl_*` - analogiczny wskaźnik liczony dla ceny uśrednionej z danego dnia (średnia z wartości `high` i `low`).

### 2.4.2 Aktywność na rynku `dlog`

- `dlog_volume_sum` - zmiana łącznego wolumenu obrotu na wszystkich rynkach w skali logarytmicznej (dzień do dnia).
- `dlog_num_trades_sum` - odpowiednik powyższej miary dla łącznej liczby transakcji na rynku kryptowalut.

### 2.4.3 Wartości wygładzone ewm

Czyli trend w aktywności rynku zamiast dziennego szumu.

- `ewm_ret_close_*` i `ewm_ret_hl2_*` - wykładniczo ważona średnia zwrotów cen, gdzie nowsze obserwacje mają większą wagę niż starsze (lepsze odwzorowanie aktualnego trendu).
- `ewm_dlog_volume_sum_*` i `ewm_dlog_num_trades_sum_*` - identyczna metoda wygładzania zastosowana dla zmian wolumenu i liczby transakcji.

### 2.4.4 Zmienność `roll_std`

- `roll_std_ret_close_btc_*` - miara zmienności rynku, liczona jako odchylenie standardowe zwrotów BTC w ruchomym oknie czasowym (np. 7, 21 dni).

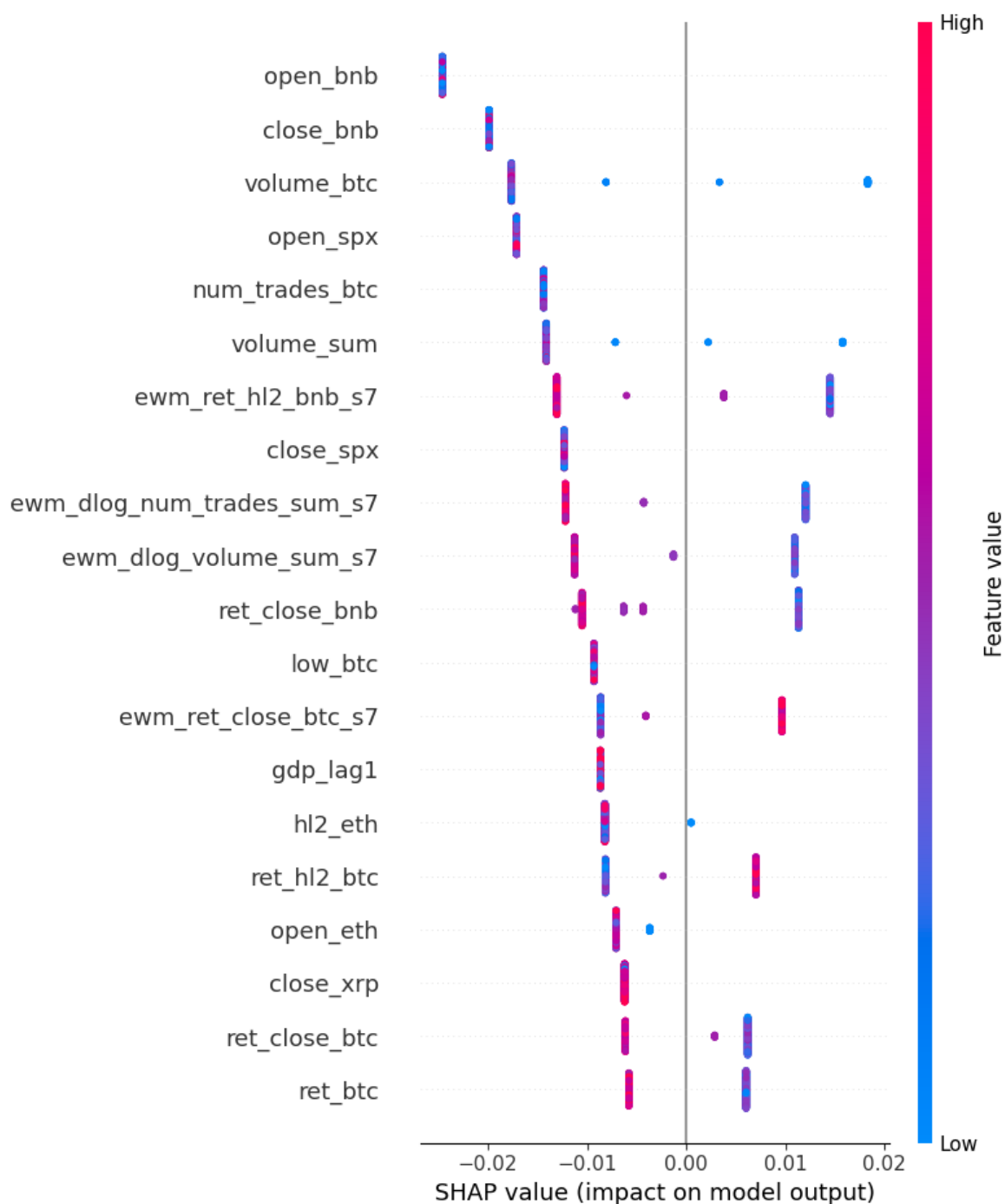
### 2.4.5 Makro `gdp`, `unrate`

- `gdp_lag1`, `gdp_growth`, `unrate_lag1`, `unrate_change` - poziomy i zmiany PKB oraz bezrobocia z poprzednich okresów. Używamy wartości opóźnionych (`lag`), ponieważ dane te publikowane są rzadziej. Dzięki temu chronimy model przed wyciekami danych z przyszłości (`data leakage`).

## 2.5 Analiza `shap`

Za pomocą narzędzia SHAP przyjrzeliśmy się poszczególnym cechom naszego modelu.

Na poniższym wykresie przedstawiono 20 cech, które mają największy wpływ na predykcję drzew. Wykres składa się głównie z pionowych kresek - jest to kwestia działania drzew decyzyjnych w modelu Random Forest, które grupują dane i każda grupa dostaje tę samą wartość SHAP. Kilka zmiennych pojawia się **jako pojedyncza prosta pionowa linia** (np. `open_bnb` i `close_bnb` na samej górze wykresu). To wskazuje, że te niezależnie od tego, czy ich wartości są niskie (niebieski) czy wysokie (czerwony), model nie wie co z nimi zrobić i traktuje je tak samo. Wprowadzają szum i powinny zostać wyeliminowane.

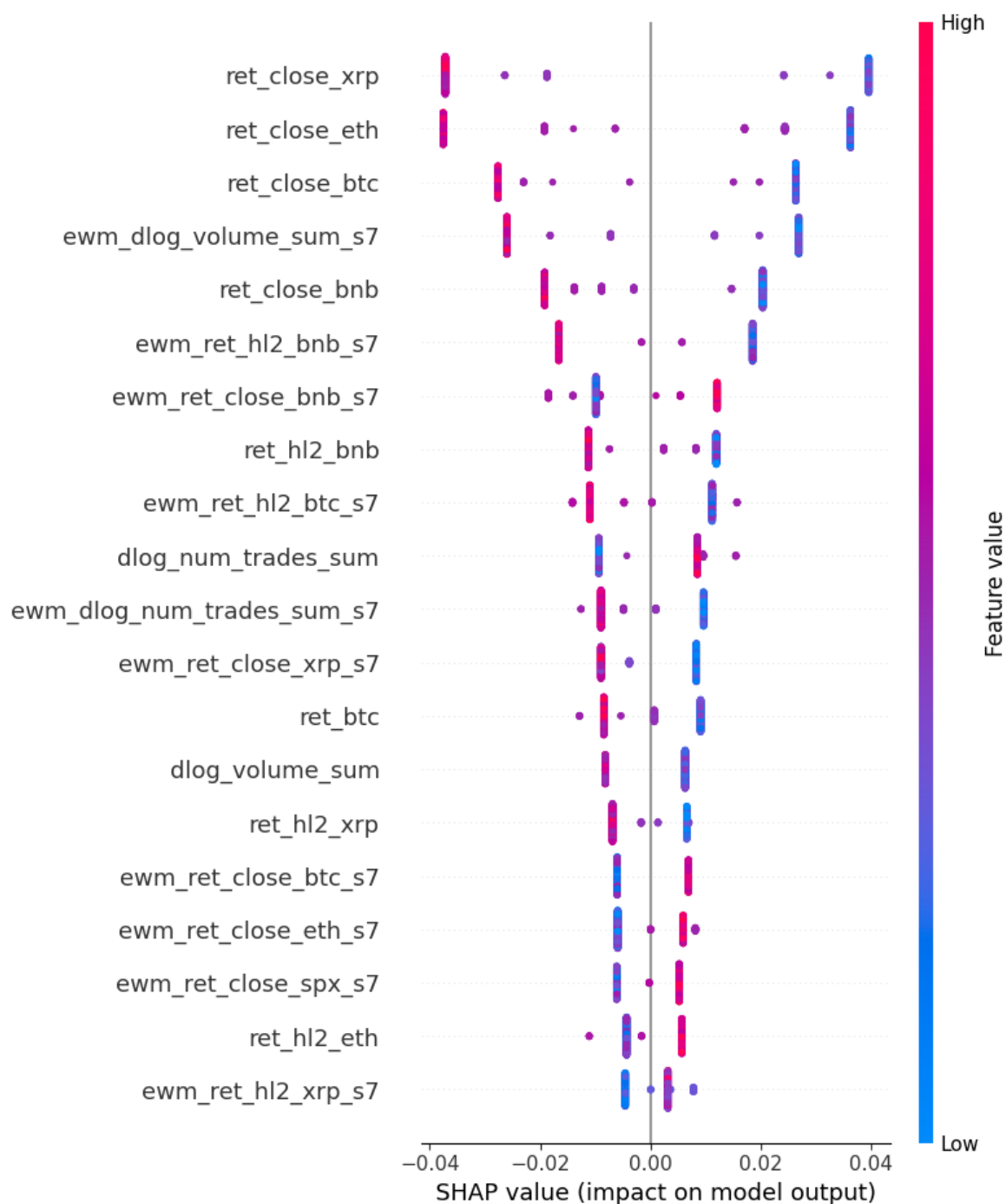


Rysunek 2: \*  
All features shap summary

W pierwszej kolejności rzucała się w oczy konieczność eliminacji surowych danych giełdowych, takich jak `open_{asset}`, `high_{asset}`, `low_{asset}`, `close_{asset}`, `volume_{asset}`, `num_trades_{asset}`. Zostały one zamienione lepszymi cechami, obrazującymi zmianę wartości, a nie surowe poziomy.

Kolejnymi cechami, które nie okazały się produktywnie są dane makroekonomiczne, takie jak `gdp`, `unrate`, `unrate_lag1`.

Po eliminacji tych cech, tak prezentuje się wykres podsumowania SHAP:



Rysunek 4: \*  
Third shap summary

## 2.6 Ostateczna wersja modelu

Finalnie postanowiliśmy skupić się na danych opisujących **zmianę** cen (`ret_close_` czy `ret_hl2_`), zamiast surowych wartości. Wykorzystaliśmy także cechy wygładzone (`ewm_ret_close_` czy `ewm_ret_hl2_`) i zsumowane dla różnych walut (`dlog_num_trades_sum`, `dlog_volume_sum`). Ostateczny model użył na zbiorze testowym wyniku  $MAE \approx 0.01679$  oraz  $RMSE \approx 0.02330$ .

Lista cech:

- `ret_close_bnb`,
- `ret_close_btc`,



- ret\_close\_eth,
- ret\_close\_spx,
- ret\_close\_xrp,
- ret\_hl2\_bnb,
- ret\_hl2\_btc,
- ret\_hl2\_eth,
- ret\_hl2\_spx,
- ret\_hl2\_xrp,
- ewm\_dlog\_num\_trades\_sum\_s7,
- ewm\_dlog\_volume\_sum\_s7,
- ewm\_ret\_close\_bnb\_s7,
- ewm\_ret\_close\_btc\_s7,
- ewm\_ret\_close\_eth\_s7,
- ewm\_ret\_close\_spx\_s7,
- ewm\_ret\_close\_xrp\_s7,
- ewm\_ret\_hl2\_bnb\_s7,
- ewm\_ret\_hl2\_btc\_s7,
- ewm\_ret\_hl2\_eth\_s7,
- ewm\_ret\_hl2\_spx\_s7,
- ewm\_ret\_hl2\_xrp\_s7,
- dlog\_num\_trades\_sum,
- dlog\_volume\_sum,
- gdp\_growth,
- unrate\_change,
- ret\_btc

A także parametrów:

```
n_estimators=100,
max_depth=5,
min_samples_leaf=200,
max_features="log2",
max_samples=0.3,
```

## 3 LSTM (Long Short-Term Memory) - dokumentacja modelu

### 3.1 Cel modelu

Model LSTM prognozuje **1-step ahead log-return BTC** na następny dzień:

- `ret_btc_next = log(close_btc).diff().shift(-1)`

Nie przewidujemy poziomu ceny (niestacjonarnego), tylko **zwrot**. Na etapie inferencji zwrot przeliczamy na poziom ceny:

- `pred_close = last_close * exp(pred_ret)`
- 

### 3.2 Dane i cechy wejściowe

#### 3.2.1 Źródło danych

- `backend/data/historical_data.parquet`

Zawiera szeregi dla:

- krypto: BTC, ETH, BNB, XRP
- indeks: SPX
- makro: GDP, UNRATE

#### 3.2.2 Feature engineering (ten sam kierunek co RF/TFT)

Celem FE jest dostarczenie sygnałów opisujących **zmiany i trend**, zamiast surowych poziomów.

##### 1. Zwroty cen (`ret_`)

- `ret_close_*` - log-return close
- `ret_hl2_*` - log-return hl2 =  $(high + low)/2$

##### 2. Aktywność rynku (`dlog`)

- `volume_sum` (suma wolumenów dla BTC/ETH/BNB/XRP/SPX)
- `num_trades_sum` (suma liczby transakcji dla BTC/ETH/BNB/XRP)
- `dlog_volume_sum = log1p(volume_sum).diff()`
- `dlog_num_trades_sum = log1p(num_trades_sum).diff()`

##### 3. Wygładzanie trendu (`ewm`, `span=7`)

- `ewm_ret_close*_s7`, `ewm_ret_hl2*_s7`
- `ewm_dlog_volume_sum_s7`, `ewm_dlog_num_trades_sum_s7`

EWM liczone na serii opóźnionej (`shift=1`), żeby cechy w dniu  $t$  korzystały wyłącznie z historii do  $t-1$ .

##### 4. Makro (bez wycieku danych)

- `gdp_growth = (gdp.shift(1)).pct_change()`
  - `unrate_change = unrate.shift(1) - unrate.shift(2)`
-

### 3.3 Zestaw cech używany w modelu (26)

Model używa 26 kolumn (FEATURE\_COLUMNS):

- `ret_close_{bnb, btc, eth, spx, xrp}`
- `ret_hl2_{bnb, btc, eth, spx, xrp}`
- `ewm_ret_close_{bnb, btc, eth, spx, xrp}_s7`
- `ewm_ret_hl2_{bnb, btc, eth, spx, xrp}_s7`
- `dlog_volume_sum, dlog_num_trades_sum`
- `ewm_dlog_volume_sum_s7, ewm_dlog_num_trades_sum_s7`
- `gdp_growth, unrate_change`

Target:

- `ret_btc_next`
- 

### 3.4 Budowa sekwencji (lookback window)

LSTM dostaje sekwencje o długości `lookback = L`:

- `X[i] = df[i : i+L, feature_cols] -> shape (L, n_features)`
- `y[i] = df[i+L-1, ret_btc_next]` (target przypisany do końca okna)

Interpretacja:

- “ostatnie **L dni** cech” -> “zwrot **jutro**”.
- 

### 3.5 Split i preprocessing (bez leakage)

1. Sortowanie i czyszczenie

- dane sortowane po `open_time`
- usuwane wiersze z brakami po FE + budowie targetu

2. Podział train/val

- split chronologiczny: `train = [:split_idx], val = [split_idx:]`
- brak shuffla (zachowujemy realizm szeregów czasowych)

3. Skalowanie cech (StandardScaler)

- `mean/std` liczone **wyłącznie na train**
- transformacja stosowana na train i val

4. Skalowanie targetu (włączone w finalnym runie)

- `ret_btc_next` standaryzowany na train (**z-score**)
  - predykcja jest odskalowywana do skali zwrotu przed przeliczeniem na cenę
-

## 3.6 Architektura

### 3.6.1 Warstwa sekwencyjna

- `nn.LSTM(input_size=n_features, hidden_size=hidden_size, num_layers=num_layers, batch_first=True)`
- dropout w LSTM aktywny tylko dla `num_layers > 1`

### 3.6.2 Głowica regresyjna

- bierzemy wektor z ostatniego kroku: `out[:, -1, :]`
  - MLP: `Linear -> ReLU -> Dropout -> Linear(..., 1)`
  - wynik: pojedyncza wartość `pred_ret_btc_next`
- 

## 3.7 Trening (najważniejsze elementy)

- loss: `L1Loss` (raportujemy MAE)
  - optymalizator: `AdamW(lr, weight_decay)`
  - scheduler: `ReduceLROnPlateau` na `val_mae`
  - gradient clipping: `grad_clip = 1.0`
  - early stopping: zatrzymanie po `patience` epokach bez poprawy `val_mae`
  - zapisywany jest najlepszy checkpoint (wg `val_mae`)
- 

## 3.8 Ostateczna wersja modelu (MAE, hiperparametry, wnioski)

### 3.8.1 Konfiguracja finalnego runu

- `lookback = 96, seed = 314`
- `hidden_size = 64, num_layers = 2, dropout = 0.15`
- `batch_size = 64`
- `lr = 8e-4, weight_decay = 5e-4`
- `epochs = 100, patience = 15`
- `target_scaling = True`
- `n_features = 26`

### 3.8.2 Wynik na walidacji (MAE)

- **Best val MAE: 0.015704**

### 3.8.3 Interpretacja przebiegu uczenia

- walidacja poprawia się głównie na początku treningu, a najlepszy wynik pojawia się wcześnie
- później widoczna jest stabilizacja/pogorszenie `val_mae`, mimo dalszego spadku `train_mae`

W praktyce oznacza to, że model szybko “wyciąga” dostępny sygnał, a dalsze uczenie zwiększa dopasowanie do danych treningowych bez poprawy uogólniania.

---

### 3.9 Uzasadnienie doboru hiperparametrów

- `lookback=96`: kompromis między zbyt krótkim kontekstem (szum) a zbyt długim (wyższe ryzyko overfitu i niestabilności).
- `hidden_size=64`: umiarkowana pojemność; wystarczająca do modelowania relacji między wieloma cechami, bez nadmiernego “rozrostu” modelu.
- `num_layers=2`: daje większą ekspresję niż 1 warstwa, a jednocześnie jest bezpieczniejsze niż głębsze RNN w noisy danych.
- `dropout=0.15` + `weight_decay=5e-4`: regularizacja dobrana pod obserwację, że najlepsze `val_mae` pojawia się wcześniej (czyli potrzebujemy kontroli nad overfitem).
- `lr=8e-4` + `AdamW`: szybkie zejście z błędem w pierwszych epokach przy stabilnym treningu; scheduler dodatkowo “hamuje”, gdy walidacja przestaje się poprawiać.
- `target_scaling=True` + `grad_clip=1.0`: stabilizacja treningu (skalowanie amplitudy celu + kontrola gradientów w LSTM).

## 4 TFT (Temporal Fusion Transformer) - eksperymenty i uzasadnienie hiperparametrów

### 4.1 Cel modelu

Model TFT trenujemy do prognozy ceny close bitcoina 1 krok do przodu. Pipeline jest przygotowany pod rolling backtest na zbiorze testowym, z metrykami:

- MAE
- RMSE

Dodatkowo porównujemy go do naiwnego baseline.

---

### 4.2 Dane i cechy

#### 4.2.1 Źródło danych

Dataset zawiera:

- kolumnę czasu: `open_time`
- target: `TARGET_COLUMN`
- cechy bazowe: `FEATURE_COLUMNS`

#### 4.2.2 Feature engineering: cechy kalendarzowe cykliczne

Do danych dodawane są cechy czasowe (cykliczne, sin/cos):

- dzień tygodnia (`dow_sin`, `dow_cos`)
- tydzień roku (`week_sin`, `week_cos`)

To jest kluczowe, bo:

- model dostaje sygnał sezonowości bez numerków (np. `week=1..52`),
- sin/cos zachowuje cykliczność (np. tydzień 52 jest blisko tygodnia 1).

### 4.2.3 Time index w Darts

Zamiast timestampów, budowany jest indeks kroków:

- `_t = 0..N-1`

I dopiero na tym powstają szeregi czasowe Darts:

- `target: TimeSeries(value_cols=[TARGET_COLUMN])`
- `past_covariates: TimeSeries(value_cols=FEATURE_COLUMNS + time_features)`

To jest w porządku, bo model i tak działa sekwencyjnie, a prawdziwy czas jest zakodowany w cechach kalendarzowych + względnym indeksie.

---

## 4.3 Podział danych

### 4.3.1 Train/Test

Split jest prosty, chronologiczny:

- `split_idx = int(n * (1 - test_size))`
- `train = [:split_idx]`
- `test = [split_idx:]`

### 4.3.2 Train/Val wewnątrz train

Walidacja jest wycinana z końcówki traina (`val_ratio`), z zabezpieczeniami na minimalne długości:

- val ma minimum `max(10, output_chunk_length + 2)`
- train musi mieć minimum `input_chunk_length + output_chunk_length + 5`

To jest ważne, bo TFT wymaga sensownego kontekstu sekwencji.

---

## 4.4 Skalowanie

Skalowanie jest robione tylko na train (to poprawnie unika data leakage):

- osobny scaler dla `target`
- osobny scaler dla `past_covariates`

W praktyce:

- model trenuje na znormalizowanych seriach,
  - metryki liczone są po `inverse_transform`, więc MAE/RMSE są w skali oryginalnej.
-

## 4.5 Model

### 4.5.1 Konstrukcja

Używany jest `darts.models.TFTModel` z:

- `add_relative_index=True`
- `loss_fn = nn.MSELoss()`
- `likelihood=None` (czyli deterministycznie, bez probabilistyki)
- trener Lightning na CPU (`accelerator="cpu", devices=1`)
- early stopping: monitor `val_loss`

### 4.5.2 Dlaczego `add_relative_index=True` pomaga?

Bo przy indeksie `_t=0..N-1` model dostaje dodatkową informację o pozycji w oknie wejściowym. To poprawia stabilność uczenia, szczególnie gdy:

- sygnał w cechach jest podobny w różnych fragmentach historii,
- chcemy, żeby model odróżniał dawne vs świeższe punkty w kontekście.

---

## 4.6 Ewaluacja: rolling 1-step ahead (historyczne prognozy)

Test liczony jest metodą zbliżoną do rzeczywistego przewidywania:

- start od `max(split_idx, input_chunk_length, len(target) - max_points)`
- `forecast_horizon=1`
- `stride = eval_stride`
- `retrain=False`
- `last_points_only=True`

To daje realistyczny obraz jakości w symulacji działania produkcyjnego.

---

## 4.7 Tryby eksperymentów

### 4.7.1 Single run

Trening jednego zestawu hiperparametrów i zapis artefaktów:

- `tft_model` (Darts save)
- `scalers.joblib`
- `metadata.json` z configiem i listą covariatów

### 4.7.2 Grid search

Dwa tryby:

- `GRID_MODE="full"`: pełna siatka kombinacji (z constraintem `hidden_size % num_attention_heads == 0`)
- `GRID_MODE="random"`: losowe próby unikalnych kombinacji

Kryterium wyboru best:

- najmniejsze MAE na rolling teście
- 

## 4.8 Przestrzeń przeszukiwania (SearchSpace)

Testowane były:

- `input_chunk_length`: (30, 60, 90)
- `hidden_size`: (16, 96)
- `lstm_layers`: (2,)
- `num_attention_heads`: (1, 4)
- `dropout`: (0.0, 0.2)
- `lr`: (3e-4, 1e-3)
- `batch_size`: (64,)

Constraint:

- `hidden_size % num_attention_heads == 0` czyli (16, 4) i (96, 4) są ok, ale np. (16, 3) byłoby odrzucone.
- 

## 4.9 Uzasadnienie: dlaczego wybrane hiperparametry zadziałały najlepiej

### 4.9.1 1) `output_chunk_length = 1` (stałe)

To ma sens, bo cała ewaluacja i pipeline są zrobione pod jednopunktową predykcję. Dla returnów często najlepsza jakość jest właśnie dla 1 kroku, bo błędy nie kumulują się jak w multi-step.

### 4.9.2 2) `input_chunk_length` w zakresie 30-90

To jest balans między:

- za krótko (np. < 20-30): model nie widzi kontekstu, a attention/LSTM nie mają z czego wyciągać wzorców
- za długo (np. dużo powyżej 90): dla returnów zwykle nie ma stabilnych zależności dalekiego zasięgu, a długi kontekst podnosi wariancję i ułatwia overfit

### 4.9.3 3) `hidden_size = 16` vs `96` (mały vs większy model)

To wprost kontroluje pojemność. Im mniejszy `hidden_size`, tym większa odporność na szum i mniejsze dopasowanie.



#### 4.9.4 4) `lstm_layers = 2`

Dwie warstwy LSTM w TFT powinny dawać rozsądną reprezentację sekwencji, z niewielkim ryzykiem overfitu.

#### 4.9.5 5) `num_attention_heads = 1 lub 4`

Heads kontrolują, ile różnych perspektyw attention może utrzymać.

- 1 head: prostszy model, mniejsze ryzyko overfitu, często lepszy gdy dane są noisy.
- 4 heads: model może równolegle patrzeć na różne fragmenty kontekstu (np. inne cechy/okresy).

#### 4.9.6 6) `dropout = 0.0 albo 0.2`

Im bardziej skomplikowany model, tym większy dropout ma sens.

#### 4.9.7 7) `lr = 3e-4 albo 1e-3`

#### 4.9.8 8) `batch_size = 64`

### 4.10 Dlaczego te ustawienia najlepiej zadziałały?

W tym problemie największym wrogiem jest overfit i udawana przewidywalność. Sprawdziłem w zasadzie model mniejszy i większy. Liczyłem na to, że mniejszy, prostszy model lepiej zgeneralizuje i nie będzie miał zbyt dużego overfitu i chyba się to nawet udało - w końcu tft miał najlepszy wynik ze wszystkich naszych modeli.