**Projekt NS**

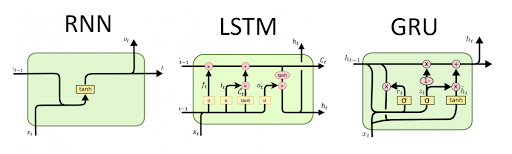


Vypracoval Bc. David Michalica

Bc. Adam Pech

# O projektu

Architektura modelu: Vytváří model hlubokého učení pomocí vrstev LSTM nebo GRU s dávkovou normalizací, regularizací výpadku a hustými výstupními vrstvami. Podporuje konfiguraci modelu pro různé délky sekvencí, horizonty predikce a sady funkcí.



# Soubory projektu

* data\_cleaner.py
* data\_collector.py
* data\_preprocesor.py
* main.py
* model\_evaluator.py
* prediction\_model.py
* training\_pipeline.py

# Soubor main.py

## Globální konstanty

Definuje parametry projektu, např.:

* **Časový rozsah dat:** START\_DATE, END\_DATE.
* **Cesty k souborům:** RAW\_DATA\_PATH, LOG\_FILE\_PATH.
* **Parametry dat:** SEQUENCE\_LENGTH, PREDICTION\_HORIZON, TRAIN\_RATIO, VAL\_RATIO.
* **Ticker akcie:** STOCK\_TICKER (např. "KO" pro Coca-Cola).

## Funkce

### Organizační funkce

1. **setup\_directories()** - Vytvoří potřebné složky pro ukládání modelů, dat a logů, pokud.
2. **setup\_logging() -** Nastaví logování do konzole i do souboru.

### Hlavní pipeline funkce

* **collect\_stock\_data()**
  + Používá StockDataCollector ke sběru denních dat akcií mezi START\_DATE a END\_DATE
  + Data ukládá do RAW\_DATA\_PATH.
* **clean\_stock\_data(stock\_data)**
  + Používá StockDataCleaner k čištění dat.
  + Vypisuje zprávu o čištění do logu.
* **preprocess\_stock\_data(cleaned\_stock\_data)**
  + Používá StockDataPreprocessor k přípravě dat na trénování:
    - Vytváří časové sekvence.
    - Rozděluje data na trénovací, validační a testovací sady.
* **setup\_prediction\_model(X\_train)**
  + Inicializuje model StockPredictionModel s definovanou délkou sekvencí (SEQUENCE\_LENGTH) a počtem vstupních atributů (n\_features).
* **acess\_data\_splits(prepared\_data)**
  + Získá trénovací, validační a testovací data z připravených dat.
  + Vypíše tvary dat (např. X\_train.shape).
* **training\_pipeline(model, X\_train, y\_train, X\_val, y\_val)**
  + Trénuje model pomocí StockTrainingPipeline.
  + Definuje počet epoch (epochs=150), velikost batch (batch\_size=64) a místo pro ukládání checkpointů.
* **evaluate\_model(model, X\_test, y\_test, preprocessor)**
  + Používá StockModelEvaluator pro:
    - Vyhodnocení predikcí.
    - Vizualizaci predikcí a distribuce chyb.
    - Generování evaluační zprávy.

# Soubor data\_cleaner.py

Zpracovává chybějící hodnoty, odstraňuje sloupce s nedostatečnými údaji, čistí základní sloupce tržních dat a přizpůsobuje čtvrtletní finanční údaje denní frekvenci. Poskytuje podrobnou zprávu o čištění dat.

Třída StockDataCleaner slouží k:

* Odstranění sloupců s příliš velkým množstvím chybějících hodnot.
* Vyplňování chybějících hodnot (forward fill a backward fill).
* Čištění technických indikátorů, finančních metrik a dalších dat.
* Generování statistik a reportů o procesu čištění.

## Konstruktor \_\_init\_\_

### Argumenty:

* **min\_non\_null\_ratio**: Určuje minimální poměr nenulových hodnot ve sloupci, aby byl ponechán.

### Proměnné:

* **self.min\_non\_null\_ratio**: Nastavení tohoto parametru (default je 0.7).
* **self.cleaning\_stats**: Slovník, kde jsou ukládány statistiky o procesu čištění pro každý ticker.

## Metody

### clean\_stock\_data(stock\_data)

* **Vstup**: Slovník, kde klíče jsou tickery (např. "KO") a hodnoty jsou DataFrame s daty.
* **Výstup**: Slovník s vyčištěnými DataFrame pro každý ticker.
* **Průběh**:
  1. Pro každý ticker inicializuje statistiky o původních sloupcích a řádcích.
  2. Zavolá metodu \_clean\_single\_stock pro zpracování jednotlivého DataFrame.
  3. Odstraňuje tickery, jejichž data jsou po čištění prázdná.

### \_clean\_single\_stock(df, ticker)

* **Vstup**: DataFrame s daty o jednom tickeru.
* **Výstup**: Vyčištěný DataFrame.
* **Průběh**:
  + **Odstranění sloupců**: Sloupce s chybějících hodnot jsou odstraněny.
  + **Vyplňování důležitých sloupců**:
    - Důležité tržní sloupce: Open, High, Low, Close, Volume.
    - Technické indikátory: Daily\_Return, Volatility, MA50, MA200.
    - Finanční metriky: Sloupce začínající Financial\_.
* **Vyplňování zbývajících null hodnot**: Nejprve forward fill, poté backward fill.
* **Odstranění nekonečných hodnot**: Nekonečna se nahradí null hodnotami a následně vyplní.

## get\_cleaning\_report()

* Výstup: DataFrame s přehledem čištění:
  + Počet původních a konečných sloupců a řádků.
  + Počet odstraněných sloupců.
  + Seznam odstraněných sloupců.

## plot\_null\_percentages(ticker)

* **Vstup**: Ticker akcie.
* **Funkce**: Vizualizuje procento chybějících hodnot ve sloupcích.
* **Poznámka**: Používá matplotlib k vytvoření grafu. Pokud není dostupná, vrací chybu.

## Jak funguje proces čištění

### Odstranění sloupců:

* Pokud je více než 30 % hodnot ve sloupci chybějících (min\_non\_null\_ratio = 0.7), sloupec je odstraněn.
* Statistiky o chybějících hodnotách jsou ukládány.

### Vyplňování hodnot:

* Důležité sloupce (Open, Close, apod.) a technické indikátory (Daily\_Return, Volatility) jsou vyplněny metodou forward fill a backward fill.
* Ostatní sloupce jsou nejprve vyplněny dopředu (forward fill) a pak zpětně (backward fill).

### Odstranění nekonečných hodnot:

* Nekonečné hodnoty jsou nahrazeny NaN a poté znovu vyplněny.

### Konečný výstup:

* Vyčištěný DataFrame, který obsahuje pouze sloupce a řádky s dostatečnými daty.

## Statistiky čištění

* Každý ticker má statistiky o:
* Počtu původních a konečných sloupců/řádků.
* Odstraněných sloupcích a jejich důvodech.

# Soubor data\_collector.py

Sběr dat: Získává historická denní cenová data a čtvrtletní finanční výkazy pro seznam burzovních indexů pomocí rozhraní Yahoo Finance API. Vypočítá další technické ukazatele a finanční poměry.

Třída StockDataCollector slouží k:

* Sbírání denních dat o akciích z Yahoo Finance API pomocí knihovny yfinance.
* Výpočtu technických indikátorů (např. klouzavých průměrů, volatility).
* Převodu čtvrtletních finančních dat na denní frekvenci.
* Ukládání a načítání dat na/z disku.

## Konstruktor \_\_init\_\_

### Parametry:

* **start\_date**: Počáteční datum pro sběr dat (formát YYYY-MM-DD).
* **end\_date**: Koncové datum pro sběr dat (formát YYYY-MM-DD).
* **default\_tickers**: Seznam výchozích tickerů, pokud není zadán jiný seznam.

### Funkce:

* Validuje formát datumu.
* Nastaví výchozí tickery (např. AAPL, MSFT, KO).

## Metody

### collect\_daily\_data(tickers)

* **Vstup**: Seznam tickerů (pokud není zadán, použije výchozí seznam).
* **Výstup**: Slovník s tickery jako klíči a zpracovanými DataFrame jako hodnotami.
* **Průběh**:
* Pro každý ticker:
  + Stáhne denní tržní data (Open, High, Low, Close, Volume) z yfinance.
  + Přepočítá:
    - Market\_Cap: Tržní kapitalizaci jako Close \* Volume.
    - Daily\_Return: Denní návratnost jako procentní změnu Close.
    - Volatility: 30denní klouzavý standardní odklon Daily\_Return.
    - MA50, MA200: Klouzavé průměry za 50 a 200 dnů.
  + Stáhne čtvrtletní finanční a bilanční data.
  + Převede finanční data na denní frekvenci pomocí \_process\_financial\_data.
  + Spočítá finanční ukazatele, např. ROA.
* Uloží zpracovaná data do slovníku.
* Přidá zpoždění (time.sleep) mezi dotazy, aby se předešlo API limitům.

### \_process\_financial\_data(data, daily\_index)

* **Vstup**:
  + data: Čtvrtletní finanční data z yfinance.
  + daily\_index: Denní časová osa (DatetimeIndex).
* **Výstup**: DataFrame s finančními daty převedenými na denní frekvenci.
* **Průběh**:
  + Sloupce s daty převede na datetime.
  + Použije reindex a metodu ffill k rozšíření čtvrtletních dat na denní frekvenci.
  + Vrátí zpracovaný DataFrame.

### save\_data(stock\_data, path)

* **Vstup**:
  + stock\_data: Slovník s daty o akciích.
  + path: Cesta k adresáři, kde budou soubory uloženy.
* **Výstup**: CSV soubory pro každý ticker.
* **Průběh**:
  + Vytvoří adresář, pokud neexistuje.
  + Pro každý ticker:
  + Uloží odpovídající DataFrame jako CSV.

### load\_data(path)

* **Vstup**: Cesta k adresáři s uloženými CSV soubory.
* **Výstup**: Slovník s načtenými DataFrame pro každý ticker.
* **Průběh**:
  + Prohledá adresář pro soubory končící \_daily\_data.csv.
  + Načte každý soubor a uloží DataFrame do slovníku.

## Funkce pro analýzu a technické indikátory

* **Daily\_Return**: Denní návratnost jako procentní změna uzavírací ceny.
* **Volatility**: Standardní odklon denní návratnosti za 30 dnů.
* **MA50**, **MA200**: Klouzavé průměry za 50 a 200 dnů.
* **ROA**: Návratnost aktiv, vypočítaná jako Net\_Income / Total\_Assets (jsou-li dostupné).

# Soubor data\_preprocesor.py

Předzpracování dat: Seskupuje a škáluje prvky podle typu (cena, objem, výnosy, technické ukazatele, finanční údaje) a připravuje je na trénování modelu. Vytváří sekvence historických dat pro předpověď budoucích cen. Rozdělí data na trénovací, ověřovací a testovací množiny.

Třída StockDataPreprocessor slouží k:

* Převodu časových řad akciových dat do vhodného formátu pro trénování modelů.
* Škálování různých skupin vlastností pomocí standardizace.
* Vytváření sekvencí (např. 60 dnů historických dat) a cílových hodnot (např. predikce na 5 dnů dopředu).
* Rozdělení dat na trénovací, validační a testovací sady.

## Konstruktor \_\_init\_\_

### Parametry:

* **sequence\_length**: Počet dnů historických dat používaných pro predikci.
* **prediction\_horizon**: Počet dnů dopředu, které model predikuje.
* **train\_ratio**: Poměr dat vyhrazených pro trénování.
* **val\_ratio**: Poměr dat vyhrazených pro validaci (zbytek pro testování).

### Interní atributy:

* **self.scalers**: Slovník pro ukládání škálovačů (např. pro cenu, objem atd.).

## Metody

### prepare\_data(stock\_data)

* **Vstup**: stock\_data = Slovník obsahující DataFrames s daty o akciích.
* **Výstup**: Slovník obsahující připravené sady pro každý ticker (train, val, test) a názvy vlastností.
* **Průběh**:
  + Rozdělení vlastností na skupiny (price, volume, returns, technical).
  + Škálování dat pro každou skupinu.
  + Tvorba sekvencí (X) a cílových hodnot (y).
  + Rozdělení na trénovací, validační a testovací sady.

### \_group\_features(df)

* **Účel**: Rozdělit vlastnosti na logické skupiny pro škálování:
  + **price**: Sloupce Open, High, Low, Close.
  + **volume**: Sloupec Volume.
  + **returns**: Sloupec Daily\_Return.
  + **technical**: Např. Volatility, MA50, MA200.

### \_scale\_features(feature\_groups, ticker)

* **Vstup**:
  + **feature**\_**groups**: Slovník se skupinami vlastností.
  + **ticker**: Název tickeru (pro identifikaci škálovače).
* **Výstup**: DataFrame se škálovanými hodnotami.
* **Průběh**:
  + Používá StandardScaler pro každou skupinu vlastností.
  + Škálovač uloží do self.scalers pro budoucí použití (např. při inverzním transformování predikcí).

### \_create\_sequences(data)

* **Vstup**: DataFrame s připravenými a škálovanými daty.
* **Výstup**:
  + X: 3D pole (vzorky, délka sekvence, počet vlastností).
  + y: 2D pole (vzorky, predikční horizont).
* **Průběh**:
  + Generuje sekvence o délce sequence\_length pro každou vlastnost.
  + Nastavuje cílové hodnoty na hodnoty Close v následujících dnech podle prediction\_horizon.

### \_split\_data(X, y)

* **Vstup**: Pole X (sekvence) a y (cíle).
* **Výstup**: Rozdělené sady (train, val, test) s příslušnými vstupy a cíli.
* **Průběh**:
  + Rozdělí data podle poměrů train\_ratio a val\_ratio.
  + Zachovává časovou posloupnost dat při rozdělení.

### inverse\_transform\_predictions(predictions, ticker)

* **Účel**: Transformovat predikce ze škálovaných hodnot zpět na původní měřítko.
* **Průběh**:
  + Používá uložený škálovač pro skupinu price.
  + Vytvoří dočasné pole pro rekonstrukci všech hodnot (Open, High, Low, Close).
  + Vybírá původní měřítko pouze pro sloupec Close.

## Tok přípravy dat

1. **Čištění a rozdělení:** Rozdělí vlastnosti na logické skupiny.
2. **Škálování:** Standardizuje data skupinově.
3. **Tvorba sekvencí:** Připraví vstupy (historická data) a cíle (budoucí hodnoty).
4. **Rozdělení:** Vytvoří trénovací, validační a testovací sady.
5. **Predikce:** Model predikuje škálované hodnoty, které jsou transformovány zpět na původní měřítko.

# Soubor model\_evaluator.py

Vypočítá komplexní metriky hodnocení na testovací sadě včetně MSE, RMSE, MAE, R-squared, MAPE a směrové přesnosti. Vytváří vizualizace skutečných a předpovídaných cen a rozdělení chyb předpovědi. Generuje hodnotící zprávu.

## Konstruktor \_\_init\_\_

**Vstupy**:

* **model**: Trénovaný model pro predikci akcií.
* **preprocessor**: Instance předzpracovatele dat (např. škálování).
* **prediction\_horizon (int)**: Počet dní, na které model předpovídá dopředu.

## Metody

### Metoda evaluate\_predictions

**Vstupy:**

* **X\_test *(np.ndarray)***: Testovací vstupy (sekvence dat pro predikci).
* **y\_test *(np.ndarray)***: Odpovídající skutečné hodnoty pro testovací vstupy.
* **ticker *(str)***: Symbol akcie (pro výběr správného škálovače).

**Výstupy:**

* Slovník metrik hodnocení:
  + mse *(float)*: Střední kvadratická chyba.
  + rmse *(float)*: Kvadratická chyba v původní škále.
  + mae *(float)*: Střední absolutní chyba.
  + r2 *(float)*: Koeficient determinace.
  + mape *(float)*: Střední absolutní procentní chyba.
  + directional\_accuracy *(float)*: Přesnost směru pohybu ceny.

**Průběh:**

* Model předpoví hodnoty pomocí self.model.predict(X\_test).
* Predikce i skutečné hodnoty se škálují zpět na původní měřítko pomocí metody self.preprocessor.inverse\_transform\_predictions.
* Vypočítají se regresní metriky, např.:
  + **MAPE**: Vynechá se dělení nulami.
  + **Directional Accuracy**: Pomocí np.sign() porovná směr změny ceny mezi skutečností a predikcí.
* Uloží metriky a predikce do atributů třídy.

### Metoda plot\_predictions

**Vstupy:**

* X\_test *(np.ndarray)*: Testovací vstupy.
* y\_test *(np.ndarray)*: Skutečné testovací hodnoty.
* ticker *(str)*: Symbol akcie.
* samples *(int)*: Počet vzorků k vykreslení (výchozí: 100).

**Výstupy:** Vizualizace grafů (ukazuje predikce vs. skutečnost).

**Průběh:**

* Pokud nebyly predikce dříve vypočítány, zavolá evaluate\_predictions.
* Pro každý den predikce vykreslí časovou řadu:
  + **Skutečné hodnoty**: Modrá linie.
  + **Predikce**: Červená linie.
  + **Interval spolehlivosti**: Poloprůhledná oblast okolo predikce.
* Přidá popisky (např. MSE a MAE) a legendu.
* Výstupem je řada grafů na jedné obrazovce.

### Metoda plot\_error\_distribution

**Vstupy:**

* X\_test *(np.ndarray)*: Testovací vstupy.
* y\_test *(np.ndarray)*: Skutečné testovací hodnoty.
* ticker *(str)*: Symbol akcie.

**Výstupy:**

* Vizualizace grafů ukazujících rozdělení chyb:
  + Histogram chyb.
  + Q-Q graf normality chyb.
  + Krabicové grafy chyb podle predikčního horizontu.

**Průběh:**

* Pokud nebyly predikce dříve vypočítány, zavolá evaluate\_predictions.
* Vypočítají se chyby jako rozdíl mezi skutečnými a predikovanými hodnotami.
* Vykreslí se:
  + Histogram a KDE pro distribuci chyb.
  + Q-Q graf na ověření normality chyb.
  + Krabicové grafy podle dne predikce.

### Metoda generate\_evaluation\_report

**Vstupy:** Řetězec**/**Symbol akcie (např. „AAPL“, „GOOGL“).

**Výstupy:** Tabulka obsahující metriky hodnocení.

**Průběh:**

* Načte uložené metriky z self.evaluation\_metrics pro daný ticker.
* Sestaví přehled s metrikami:
  + Hodnoty (např. 0.1234 pro MSE).
  + Popis metriky (např. „Střední kvadratická chyba“).
* Vrátí tabulku, kterou lze snadno exportovat nebo vykreslit.

# Soubor prediction\_model.py

Predikce: Provádí předpovědi budoucích cen pomocí natrénovaného modelu na nových datech. Provádí inverzní škálování pro převod předpovědí zpět na původní cenovou stupnici.

## Konstruktor \_\_init\_\_

* sequence\_length (int): Počet časových kroků ve vstupní sekvenci.
* n\_features (int): Počet vlastností na časový krok.
* prediction\_horizon (int): Počet časových kroků, na které model předpovídá.
* use\_gru (bool): Určuje, zda použít vrstvu GRU namísto LSTM (výchozí: True).

## Metody

### Metoda \_build\_model

**Vstupy:** Žádné explicitní vstupy (využívá atributy třídy).

**Výstup:** Objekt modelu TensorFlow/Keras.

**Průběh:**

* Sestaví model pomocí Sequential API:
  + **Vstupní vrstva**: Očekává data tvaru (sequence\_length, n\_features).
  + **První rekurentní vrstva**:
    - Používá GRU nebo LSTM s 256 jednotkami.
    - Aplikuje regulární L2 penalizaci (0.0001).
  + **BatchNormalization** a **Dropout**: Stabilizují trénink a redukují přeučení.
  + **Druhá a třetí rekurentní vrstva**: Menší počet jednotek (128, 64) pro hierarchické extrahování rysů.
  + **Husté vrstvy**: Dvě husté vrstvy s aktivací relu (128, 64 jednotek).
  + **Výstupní vrstva**: Lineární aktivace pro regresní výstup o velikosti prediction\_horizon.
* Kompiluje model s optimalizátorem Adam a nízkou rychlostí učení (learning\_rate=0.00005).

### Metoda \_create\_callbacks

**Vstupy:** checkpoint\_path *(str)*: Cesta k souboru, kam se ukládají nejlepší váhy modelu.

**Výstup:** Seznam callbacků pro trénink.

**Průběh:**

* EarlyStopping:
  + Monitoruje validační ztrátu.
  + Zastaví trénink, pokud se ztráta nezlepší po 15 epochách.
* ModelCheckpoint: Ukládá nejlepší váhy modelu na základě validační ztráty.
* ReduceLROnPlateau: Snižuje rychlost učení, pokud se ztráta nezlepší po 5 epochách.

### Metoda train

**Vstupy**:

* X\_train, y\_train *(np.ndarray)*: Trénovací data (vstupy a cíle).
* X\_val, y\_val *(np.ndarray)*: Validační data.
* epochs *(int)*: Maximální počet epoch (výchozí: 100).
* batch\_size *(int)*: Velikost dávky (výchozí: 32).
* checkpoint\_path *(str)*: Cesta k souboru pro ukládání modelu.

**Výstup:** Slovník obsahující tréninkovou historii.

**Průběh:**

1. Připraví callbacky voláním \_create\_callbacks.
2. Spustí trénink modelu pomocí model.fit s předanými daty, epochami a callbacky.
3. Uloží historii trénování do self.history a vrátí ji.

### Metoda evaluate

**Vstupy:** X\_test, y\_test *(np.ndarray)*: Testovací vstupy a skutečné hodnoty.

**Výstup:** Dvojice (loss, mae): (Ztráta vypočtená metrikou mse, Střední absolutní chyba (MAE))

**Průběh:**

1. Spustí metodu model.evaluate pro testovací data.
2. Vrací ztrátu a metriky.

### Metoda predict

**Vstupy:** X *(np.ndarray)*: Nové vstupní sekvence pro predikci.

**Výstup:** Predikované hodnoty *(np.ndarray)*.

**Průběh:**

1. Použije model.predict k výpočtu predikcí na zadaných datech.
2. Vrací predikované hodnoty.

### Metoda get\_model\_summary

**Vstupy**: Žádné.

**Výstup:** Textový přehled modelu.

**Průběh:** Volá metodu model.summary a vrací výstup.

### Metoda save\_model

**Vstupy:** filepath *(str)*: Cesta k souboru pro uložení modelu.

Výstup: Žádný.

**Průběh:** Použije model.save k uložení celého modelu (včetně váh i architektury).

### Metoda load\_model (tovární metoda)

**Vstupy:** filepath *(str)*: Cesta k uloženému modelu.

**Výstup:** Instance třídy StockPredictionModel.

**Průběh**:

1. Načte model pomocí tf.keras.models.load\_model.
2. Vrací instanci třídy s načteným modelem.

# Soubor training\_pipeline.py

Tréninkový kanál: Trénuje model pomocí zpracovaných dat se zpětnými voláními pro včasné zastavení, kontrolní bodování modelu, snížení míry učení a protokolování TensorBoard. Ukládá nejlepší váhy modelu a protokoly trénování.

## Konstruktor \_\_init\_\_

* model: Objekt modelu (instance obsahující Keras model).
* log\_dir (str): Adresář pro ukládání logů (výchozí: "training\_logs").

## Metody

### Metoda setup\_logging

Vstupy: Žádné.

**Výstup:** Žádný přímý výstup. Nastaví konfiguraci logování.

**Průběh:**

* Konfiguruje logovací modul:
  + Formát logů obsahuje čas, úroveň logu a zprávu.
  + Zprávy jsou zapisovány jak do souboru (training.log), tak do konzole.

### Metoda create\_callbacks

**Vstupy:** checkpoint\_dir *(str)*: Adresář pro ukládání modelových checkpointů.

**Výstup**:

* callbacks *(list)*: Seznam callbacků pro trénink.
* model\_path *(str)*: Cesta k uloženému modelu s časovou značkou.

**Průběh:**

1. Zajistí, že adresáře pro checkpointy a TensorBoard logy existují (pokud ne, vytvoří je).
2. Generuje jedinečný název souboru modelu obsahující časovou značku.
3. Vytvoří seznam callbacků:
   * **EarlyStopping**: Zastaví trénink, pokud se validační ztráta nezlepšuje po 10 epochách.
   * **ModelCheckpoint**: Uloží nejlepší váhy modelu (ve formátu .keras) na základě validační ztráty.
   * **ReduceLROnPlateau**: Sníží rychlost učení, pokud se validační ztráta nezlepšuje po 5 epochách.
   * **TensorBoard**: Loguje metriky a histogramy do TensorBoard adresáře.
4. Vrací seznam callbacků a cestu k uloženému modelu.

### Metoda train

**Vstupy:**

* X\_train, y\_train *(np.ndarray)*: Trénovací data (vstupy a cíle).
* X\_val, y\_val *(np.ndarray)*: Validační data.
* epochs *(int)*: Maximální počet epoch (výchozí: 100).
* batch\_size *(int)*: Velikost dávky (výchozí: 32).
* checkpoint\_dir *(str)*: Adresář pro ukládání modelových checkpointů (výchozí: "models").

**Výstup:** history: Tréninková historie vrácená Keras metodou fit.

**Průběh:**

1. Loguje začátek tréninku a informace o velikosti dat.
2. Vytvoří callbacky pomocí create\_callbacks.
3. Spustí trénink modelu voláním model.fit, předá mu:
   * Tréninková a validační data.
   * Počet epoch a velikost dávky.
   * Callbacky.
4. Po úspěšném tréninku:
   * Loguje, že trénink byl úspěšný, a cestu k uloženému modelu.
   * Volá \_log\_training\_results pro logování výsledků tréninku.
5. Pokud dojde k chybě, loguje chybu a vyvolá výjimku.

### Metoda \_log\_training\_results

**Vstupy:** history: Objekt tréninkové historie (obsahuje ztrátu a metriky po každé epoše).

**Výstup:** Žádný přímý výstup. Loguje souhrnné výsledky tréninku.

**Průběh**:

1. Zjistí počet epoch, na kterých trénink proběhl.
2. Načte konečnou tréninkovou a validační ztrátu.
3. Loguje počet epoch, konečnou ztrátu a validační ztrátu.

# Průběh programu

Obsah obrázku text, snímek obrazovky, číslo, Písmo

Popis byl vytvořen automaticky

## Popis diagramu:

1. **User** vytvoří instanci **StockTrainingPipeline**.
2. **StockTrainingPipeline** vytvoří instanci **StockPredictionModel**.
3. **StockTrainingPipeline** zavolá **setup\_logging()** pro nastavení logování.
4. **StockTrainingPipeline** zavolá **create\_callbacks()** pro vytvoření všech potřebných callbacků (EarlyStopping, ModelCheckpoint atd.).
5. **StockTrainingPipeline** zavolá **train()** na modelu pro zahájení trénování.
6. **StockPredictionModel** zavolá metodu **fit()** pro trénování modelu.
7. **Callbacks** monitorují validaci a vykonávají akce na základě podmínek (EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, TensorBoard).
8. Po každé epoše jsou volány zpětné volání, která upravují trénovací parametry (např. EarlyStopping nebo změnu tempa u ReduceLROnPlateau).
9. Po dokončení trénování **StockTrainingPipeline** zavolá metodu **\_log\_training\_results()**, aby zaznamenal výsledky trénování.

# Metody

**. model\_evaluator.py**

* **\_\_init\_\_** Inicializuje hodnoty pro model, preprocesor a horizont předpovědi.
* **evaluate\_predictions** Vyhodnocuje model na základě testovacích dat a vrací metriky jako MSE, RMSE, MAE, R2 a MAPE.
* **plot\_predictions** Vykresluje grafy pro skutečné a predikované hodnoty pro každý den v horizontu předpovědi.
* **plot\_error\_distribution** - histogramy a Q-Q ploty pro distribuci chyb a vykresluje boxplot pro chyby podle jednotlivých dnů v horizontu předpovědi.
* **generate\_evaluation\_report(self, ticker: str) -> pd.DataFrame**  
  Generuje podrobný report s metrikami vyhodnocení modelu včetně MSE, MAE, R2, MAPE a Directional Accuracy.

**2. stock\_prediction\_model.py**

* **\_\_init\_\_(self, sequence\_length: int, n\_features: int, prediction\_horizon: int, use\_gru: bool = True)**  
  Inicializuje parametry pro délku sekvencí, počet vlastností, horizont předpovědi a volbu mezi GRU a LSTM.
* **\_build\_model(self) -> keras.Model**  
  Staví model s GRU nebo LSTM vrstvami, dropoutem, normalizací a hustými vrstvami pro predikce akciového trhu.
* **\_create\_callbacks(self, checkpoint\_path: str) -> list**  
  Vytváří callbacky pro trénování, včetně early stopping, model checkpointu a redukce learning rate.
* **train(self, X\_train: np.ndarray, y\_train: np.ndarray, X\_val: np.ndarray, y\_val: np.ndarray, epochs: int = 100, batch\_size: int = 32, checkpoint\_path: str = 'best\_model.h5') -> Dict**  
  Trénuje model na trénovacích datech s použitím callbacků pro kontrolu učení a nejlepší model.
* **evaluate(self, X\_test: np.ndarray, y\_test: np.ndarray) -> Tuple[float, float]**  
  Vyhodnocuje model na testovacích datech a vrací hodnoty loss a MAE.
* **predict(self, X: np.ndarray) -> np.ndarray**  
  Provádí predikce na nových datech pomocí modelu.
* **get\_model\_summary(self)**  
  Vrací souhrn architektury modelu.
* **save\_model(self, filepath: str)**  
  Ukládá model na zadanou cestu.
* **load\_model(cls, filepath: str) -> 'StockPredictionModel'**  
  Načítá model ze zadané cesty.

**3. stock\_training\_pipeline.py**

* **\_\_init\_\_(self, model, log\_dir="training\_logs")**  
  Inicializuje tréninkovou pipeline s modelem a adresářem pro logování.
* **setup\_logging(self)**  
  Konfiguruje logování pro tréninkový proces, včetně ukládání logů do souboru a výpisu do konzole.
* **create\_callbacks(self, checkpoint\_dir)**  
  Vytváří a vrací callbacky pro trénování modelu včetně EarlyStopping, ModelCheckpoint, ReduceLROnPlateau a TensorBoard.
* **train(self, X\_train, y\_train, X\_val, y\_val, epochs=100, batch\_size=32, checkpoint\_dir='models')**  
  Spouští tréninkový proces modelu, využívající callbacky pro monitorování výkonu a úpravu trénování.
* **\_log\_training\_results(self, history)**  
  Loguje výsledky trénování, včetně konečných hodnot loss a val\_loss.

# Výsledky projektu