Projekt indywidualny-
przewidywanie cen akcji na giełdzie

Konrad Wołkiewicz
nr. albumu: 331758

16 maja 2025

1 Wstęp

1.1 Skąd pomysł na taki projekt

Do stworzenia tego projektu zainspirowała mnie gra na giełdzie. Stwierdziłem, że to może być dobry sposób na połączenie obowiązku jakim jest zaliczenie projektu indywidualnego wraz z użytecznością. Wybierając ten projekt miałem nadzieje na to, że lepiej zrozumiem jak działa giełda oraz aplikacja będzie wyznaczać przyszłe ceny akcji poprawnie, w wyniku czego będę mógł efektywniej stawiać pieniądze na odpowiednie spółki. Dodatkowo sztuczna inteligencja jest dość ciekawą dziedziną i w przyszłości jest to jedna z kilku ścieżek kariery IT, który rozważam.

1.2 Ogólny opis projektu

Projekt jest podzielony na kilka modułów:

1. app.py
2. main.py
3. get_data.py
4. model_LinearRegression.py
5. model_XGBRegressor.py
6. model_LSTM.py
7. model_LSTM2.py
8. visualization.py

Aplikacja opiera się na GUI w którym użytkownik może wprowadzić parametry takie jak:

- Ticker spółki
- Ilość dni do przewidzenia
- Dzień od którego model pobiera dane spółki
- Ilość dni do trenowania
- Funkcja straty(do porównywania wyników testowania między modelami)

Użytkownik ma możliwość wprowadzenia dla XGBRegressor i LSTM dodatkowe parametry.
Do predykcji są wykorzystywane 4 modele:

- Regresja liniowa
- XGBRegressor

- LSTM, w którym model od razu przewiduje wszystkie dni
- LSTM, w którym model przewiduje jeden dzień i ten dzień wraz z poprzednimi oprócz pierwszego jest na wejściu do przewidzenia kolejnego dnia itd.

Wynikiem działania jest 5 wykresów, w których na osi Y jest wartość akcji w chwili zamknięcia sesji w walucie z takiego kraju, na jakiej giełdzie się znajduje, na osi X jest czas. Te wykresy to:

1. Wykres historyczny danej akcji od daty podanej przez użytkownika
2. Wykres Regresji liniowej
3. Wykres XGBRegressor
4. Wykres LSTM1
5. Wykres LSTM2

Na wykresie są zaznaczone odpowiednimi kolorami wartości: prawdziwe, przewidywane w przyszłości, przewidywane na danych testowych.
Dodatkowo w GUI jest sekcja Status, w której jest informacja zwrotna od modułów. Np. czy zadziałało early stopping, albo informacja że użytkownik podał złe dane

2 Opis problemu do rozwiązania

Ceny akcji są dynamiczne, jednego dnia mogą rosnąć następnego spadać. Jako ludzie możemy znajdywać różne wzorce, które nam pozwolą przewidywać czy akcja spadnie czy wzrośnie. Przykładowo możemy przypuszczać, że skoro przedwczoraj, wczoraj i dzisiaj rośnie to jutro też urośnie. Modele mogą znajdować wzorce, które dla nas nie są widoczne i w ten sposób dokładniej przewidywać co będzie się dziać w przyszłości. Projekt korzystając z 4 modeli przewiduje jaka będzie przyszłość ceny akcji. Aby ocenić jak dobrze sobie radzi należy porównywać przewidywanie testowe z ceną realną i dobrać najlepsze parametry aby uzyskać najlepszą dokładność.

3 Wykorzystane narzędzia

3.1 Język programowania

Python wersja 3.11

3.1.1 Biblioteki

- sklearn (mae, mse, MinMaxScaller, LinearRegression)
- gradio (GUI)
- numpy
- pandas
- yfinance
- xgboost (XGBRegressor)
- tensorflow.keras (Sequential, LSTM, Dense, Dropout, Adam, SGD, RMSprop)
- matplotlib

3.1.2 Narzędzia dodatkowe

- Visual Studio Code wersja 1.100.0
- Github

4 Prezentacja projektu

Projekt podzieliłem na kilka modułów:

4.1 app.py

Jest to moduł który odpowiada za GUI. Został stworzony z wykorzystaniem gradio. Ekran jest podzielony na kilka części:

- Informacje ogólne
- Parametry do XGBRegressor
- Parametry ogólne dla LSTM
- Parametry do poszczególnych warstw LSTM
- Status programu - informacje od modeli
- Wykresy

Są dwie implementacje LSTM, ale z inną logiką działania, oba dostają te same parametry.
Znajdują się 3 przyciski:

1. Predict
   Włącza predykcje, pobiera wszystkie parametry i przesyła dalej, a następnie wyświetla wykresy
2. Stop training
   Zatrzymuje trenowanie i wyświetla aktualny wynik
3. Update number of Layers
   Jest to przycisk do wyświetlania ilości bloków z warstwami LSTM. W programie można maksymalnie utworzyć 5 warstw. Aby GUI było czytelne zbędne warstwy są ukryte. W celu ich odsłonięcia należy wpisać ilość warstw i nacisnąć ten przycisk.

Jest wiele wartości zmiennych potrzebnych do działania tego programu. Wszystkie pola są wypełnione domyślnymi parametrami. Jeżeli użytkownik naciśnie przycisk Predict to wszystkie parametry przechodzą do modułu main.py. Dodatkowo są przekazywane dwie funkcje. Pierwsza, która sprawdza flagę should_stop. Jeżeli użytkownik naciśnie przycisk Stop training to flaga jest aktywowana. Druga funkcja update_progress, jest odpowiedzialna za aktualizowanie postępu i wyświetlaniu go w GUI. Trenowanie często długo trwa i warto, aby użytkownik widział na jakim etapie się znajduje i ile mniej więcej czasu zostało. Wykresy pojawiają się dopiero po uzyskaniu przewidywania z wszystkich modeli. Po zakończeniu działania pojawiają się również informacja z tych modeli, czy uczenie przebiegło pomyślnie lub na którym etapie doszło do przerwania treningu. Jeżeli w trakcie działania wystąpi sytuacja, której program nie wie jak rozwiązać, np. użytkownik poda zły ticker to jest jedynie wyświetlany bład w sekcji status.

4.2 main.py

Po naciśnięciu przycisku Predict przechodzimy do moduły main.py, który zarządza całym backendem. Na początku korzystając z modułu get_stock_data pobiera dane dla danego tickera i początkowej daty. Następnie sprawdza poprawność podanych argumentów days_to_train i days_in_ future_to_predict, ponieważ nie może być sytuacji że przykładowo ramka danych ma 10 rekordów, a użytkownik chce aby do trenowania było użyte 15. Z tego powodu jeżeli te argumenty nie pasują to dni do uczenia stanowią 0.4 wszystkich rekordów, a dni do przewidywania stanowią 0.1 wszystkich rekordów. Jeżeli nastąpiła taka zmiana to powiadomienie będzie dostępne w sekcji Status w GUI. Kolejnym aspektem do sprawdzenia jest ilość rekordów. Minimalna ilość to 3 ponieważ pierwszy rekord do uczenia, drugi rekord jako wynik uczenia i do testowania, 3 rekord jako wynik testowania. Następnie są wywoływane 4 funkcje, każdy do innego modelu, w wyniku których otrzymuję, wynik przewidywania, wynik testowania, ogólna ocena modelu(jest to wynik funkcji straty między faktycznymi cenami akcji, a wynikiem przewidywania testowego), status czyli informacja zwrotna od modelu

Te 4 wartości zawsze są zwracane przez te 4 funkcje, ale dodatkowo:
Model regresji liniowej zwraca najlepszą ilość dni do uczenia aby osiągnąć najlepszy wynik, ponieważ ta wartość jest wyznaczana automatycznie w tym modelu.
Następnie korzysta z modułu visualization, aby uzyskać wykresy:

- historycznych cen akcji
- modelu Regresji liniowej
- modelu XGBRegressor
- modelu LSTM1
- modelu LSTM2

Następnie jest tworzony spójny status, który zawiera informacje z main(jeżeli nastąpiła zmiana ilości dni do uczenia i przewidywania) oraz z wszystkich modeli. Wszystkie wykresy i status aplikacji są wynikiem działania main.py

4.3 get_data.py

Ten moduł służy do wydobycia danych spółki potrzebnych do przewidywania. Najpierw sprawdzane jest czy data startowa i ticker są poprawne. Jeżeli ticker jest źle podany to zwracany jest błąd, który jest w sekcji status w GUI. Następnie sprawdzana jest data początkowa. Jeżeli została błędnie wpisana to występuje błąd i informacja dla użytkownika, aby wprowadził poprawną date w formacie YYYY-MM-DD. Jeżeli zostanie podana data z przyszłości to również jest błąd i informacja zwrotna dla użytkownika, aby podał datę z przeszłości. Jeżeli zostanie podana data wcześniejsza niż dzień od, którego spółka jest notowana na giełdzie to jest ustawiana data, kiedy spółka weszła na giełdę. Po wyznaczeniu poprawnej daty pobierane są z yahoo finance api od tej dany takie kolumny:

1. Open
2. High
3. Low
4. Close
5. Volume
6. Dividends
7. Stock Splits

Następnie pobierane są wyniki finansowe za kwartał. Łączy się je z poprzednio pobranymi danymi i usuwa, nie potrzebne kolumny. To czy kolumna jest wartościowa czy nie jest sprawdzane przez korelacje z kolumną Close. Jeżeli korelacja jest z zakresu <-0.55;0.55> to taka kolumna jest usuwana. Kolejnym etapem jest usunięcie kolumn, które mają same wartości nan lub suma ilości nan z ilością zer stanowi więcej niż 20% całej kolumny. Po tych przetwarzaniach ramka danych trafia z powrotem do main.py, gdzie jest dalej przetwarzana.

4.4 model_LinearRegression.py

4.4.1 Informacje ogólne

Ten moduł odpowiada za regresje liniową. Jest wywoływany z main.py z parametrami: ramka danych, ilość dni do przewidzenia, funkcja straty. Zwraca wynik predykcji na okresie testowym, wynik predykcji w przyszłości, najlepsza ilość dni do uczenia, dokładność modelu. Wykorzystuje tylko kolumnę Close.

4.4.2 Krótki opis

Na początku korzystam z ramki danych od [0 do -ilość dni do przewidzenia]. Następnie znajduje najlepsza ilość dni do uczenia, aby uzyskać najlepszy wynik testowania. Jak już uzyska ten najlepszy wynik to ta funkcja regresji liniowej jest wpasowywana do danych [-ilość dni do przewidzenia do końca]. Analogicznie postępuję aby uzyskać wynik predykcji.

4.4.3 Opis działania

Stwierdziłem, że model regresji liniowej jest na tyle szybki, że można zautomatyzować ilość dni do uczenia. Szukane są dwie ilości dni do trenowania. Pierwszy do przetestowania drugi do przewidywania. Aby wyznaczyć te ilości korzystam z funkcji get_best_num_of_days_to_train, do której przekazuje za pierwszym razem ramkę danych od początku do -(ilość dni do przewidzenia), która znajduje najlepszą ilość dni do uczenia aby przetestować model i analogicznie całą ramkę danych, aby dostać najlepszą ilość dni do przewidywania przyszłości. Aby to uzyskać to w tej funkcji tworzę słownik, w którym ilość dni do uczenia jest kluczem a wartością jest dokładność modelu(początkowo nieskończoność). Ilość dni jest wybierana z pewnym krokiem, jeżeli ilość dni od daty startowej do dnia dzisiejszego odjęta od ilości dni do przewidzenia jest większa od 1000 to od tej liczby do 1000 z krokiem 1000 są wybierane dni do uczenia. Jeżeli ta różnica jest większa od 200 ale mniejsza od 1000 to do 200 z krokiem 50. Analogicznie to wygląda dla pozostałych zakresów. Do 50 z krokiem 10, do 10 z krokiem 5, do 1 z krokiem 1. Następnie dla każdej ilości dni jest dokonywane testowane i porównywanie z faktycznymi cenami za pomocą funkcji straty podanej przez użytkownika. To porównanie jest zapisywane w słowniku. Jak już mamy najlepszą ilość dni do uczenia to wyznaczam predykcje testową i przewidywanie w przyszłość. Do main.py wracają: wynik testowania, wynik przewidywania przyszłości, najlepsza ilość dni do uczenia aby przewidzieć przyszłość, dokładność modelu i status.

4.5 model_XGBRegressor.py

4.5.1 Informacje ogólne

Ten moduł odpowiada ze XGBRegression. Jako argumenty dostaje ramkę danych, ilość dni do przewidzenia, ilość dni do trenowania, parametry modelu, funkcja straty, funkcję sprawdzającą czy został naciśnięty przycisk Stop Training, funkcję, która wyświetla progress w GUI.

4.5.2 Krótki opis

XGBRegressor jest stworzone w ten sposób, że na wejście dostaje tyle dni ile podał użytkownik i wszystkie kolumny, a na wyjście zwraca jedną wartość. Aby przewidzieć kilka dni do przodu tworzę tyle modeli ile jest dni do przewidzenia.

4.5.3 Opis działania

Na początek skaluje dane do zakresu <0,1> i dzielę ramkę danych na dane uczące i testowe i te potrzebne do przewidywania przyszłości. Dane testowe i uczące są podzielone w stosunku 0.8(0.2 stanowią testowe, 0.8 stanowią uczące).
Następnie jest pętla, która się wykona tyle razy ile jest dni do przewidzenia. Na początku wysyłany jest komunikat do progress bar w GUI, który informuje na, którym dniu się znajdujemy oraz sprawdzana jest flaga czy użytkownik nacisnął przycisk Stop Training. Jeżeli flaga jest uruchomiona to kończymy trenowanie i zwracamy wartości None. W przeciwnym razie tworzymy model, który uczymy. Każdy jest uczony tym samym X_train, ale y jest inne, ponieważ w zależności, w której iteracji pętli jesteśmy to model będzie uczony do przewidywania innego dnia w przyszłości. Następnie testuję, przewiduję przyszłość, przewiduję na danych uczących i obliczam dokładność uczenia na danych uczących i testowych, aby można było wyświetlić w progress bar. Gdy już zakończy się pętla to złączam wyniki testowania i przewidywania przyszłości, odwracam skalowanie i obliczam ostateczną dokładność modelu.
Do main.py wracają: wynik testowania, wynik przewidywania przyszłości, dokładność modelu i status

4.6 model_LSTM1.py i model_LSTM2.py

4.6.1 Informacje ogólne

Te moduły odpowiadają za przewidywanie za pomocą modelu z tensorflow Long Short Term Memory. Jako argumenty dostaje ramkę danych, ilość dni do przewidzenia, ilość dni do trenowania, parametry modelu, funkcja straty, funkcję sprawdzającą czy został naciśnięty przycisk Stop Training, funkcję, która wyświetla progress w GUI.

4.6.2 Krótki opis

Są dwie implementacje LSTM, ale różnią się logiką. Model pierwszy od razu przewiduje wszystkie wartości, model drugi przewiduje jedną wartość i ta wartość wraz z poprzednimi oprócz pierwszej są wejściem ponownie do modelu. W ten sposób jest przewidywana druga wartość itd.

4.6.3 Opis działania

Na początku dane są skalowane do zakresu <0;1> i dzielone są dane na uczące, testowe i do przewidywania przyszłości. Dane są dzielone w stosunku 0.8 w modelu pierwszym, a w modelu drugim dane testowe to ostatnia seria danych, dane uczące to reszta. Kolejnym etapem jest stworzenie modelu. Model jest sekwencyjny i przekazane przez użytkownika parametry są tutaj przypisywane do odpowiednich warstw. W tym przypadku dropout jest dodawany po warstwie LSTM.Model pierwszy w modelu ma wyjść tyle ile dni do przewidzenia, model drugi ma jedno wyjście. Kolejnym etapem jest uczenie. W tym przypadku korzystam z pętli, która wykona się maksymalnie tyle razy ile użytkownik wprowadził epok. Na początku każdej iteracji przesyłana jest informacja, do GUI w której epoce się znajduje i jaki jest błąd na danych treningowych i testowych. Następnie sprawdzany jest przycisk Stop Training czy nie został wciśnięty. Potem można przejść do trenowania, po której następuje obliczanie straty na danych testowych z najlepszą jak dotąd uzyskaną wartością. Jeżeli jest lepsza to zapisywana jest ta wartość i wagi w warstwach aby mogła być odtworzona najlepsza sekwencja. Jeżeli jest gorsza to sprawdzamy ile sekwencji z rzędu się nie polepszyła dokładność i jeżeli jest taka jak zostało wskazane w GUI w Early stopping patience to model kończy uczenie. Ostatnimi etapami jest predykcja na danych testowych i w przyszłość, odwrócenie skalowania do realistycznych wartości, wyliczenie błędu zgodnie z funkcją straty. Moduł zwraca przewidywanie na danych testowych, wynik przewidywania przyszłości, dokładność modelu i status.

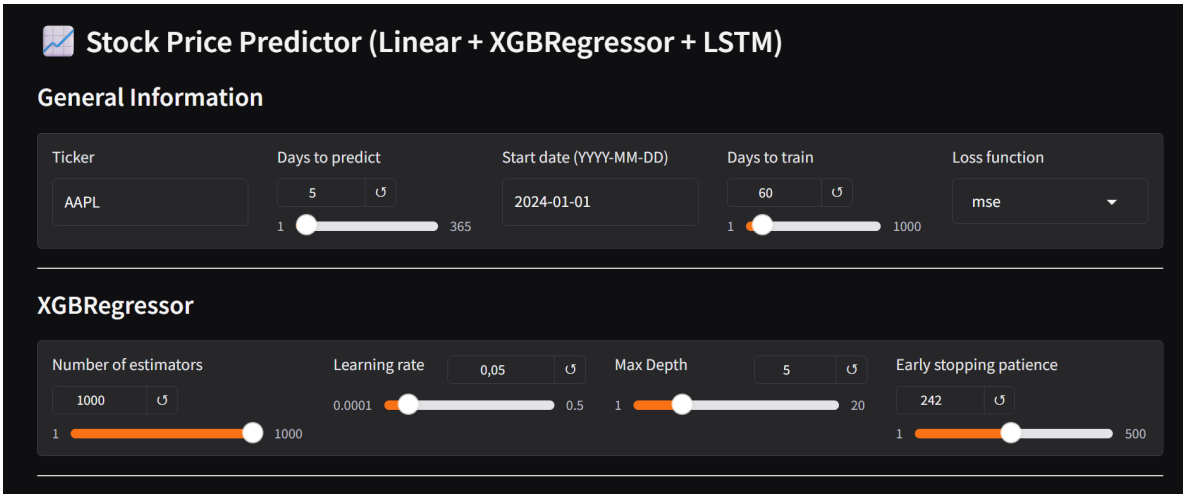
4.7 visualization.py

Jest to moduł, który odpowiada za tworzenie wykresów. Są dwie funkcje:

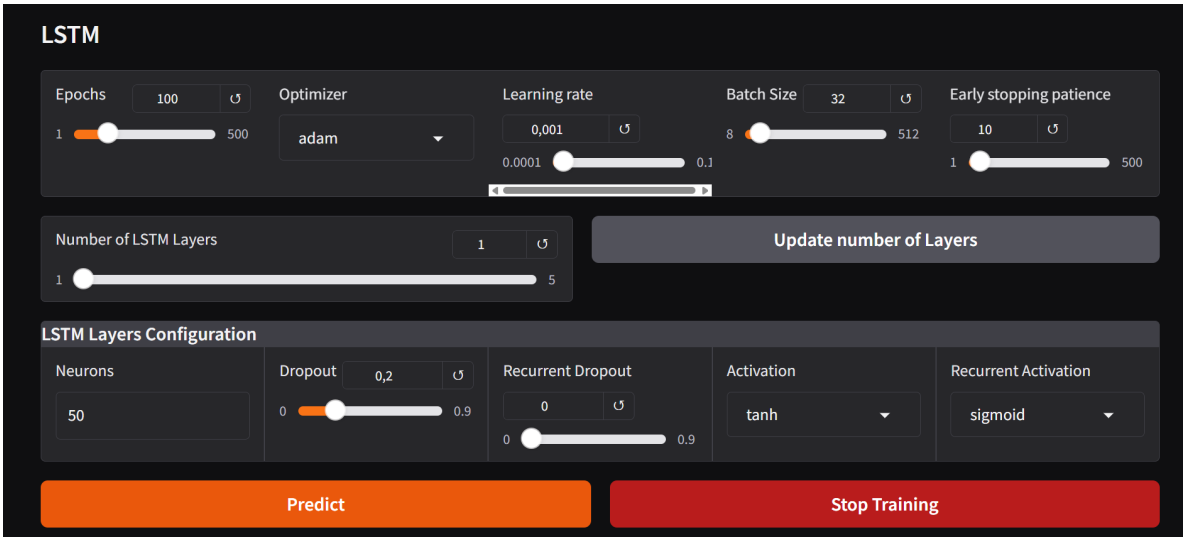
1. Pierwsza, która odpowiada ze stworzenie wykresu z prawdziwymi cenami akcji od daty startowej do dnia dzisiejszego.

2. Druga, która bierze zakres dat: 2* ilość dni do przewidzenia + 10(jeżeli to wykracza poza zakres dostępnych dat to brany jest maksymalny dostępny zakres), aby wyświetlić przeszłość i ilość dni do przewidzenia aby wyświetlić przyszłość. Ta funkcja rysuje prawdziwą cenę, wynik testowania i wynik przewidywania. Dodatkowo w legendzie umieszczona jest informacja o ilości dni do uczenia, do przewidywania i dokładność modelu.

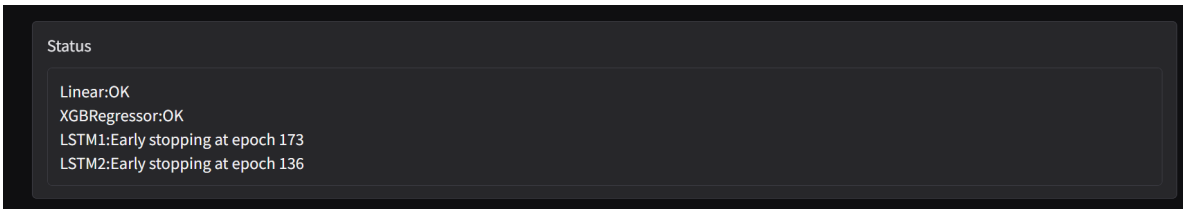
5 Prezentacja



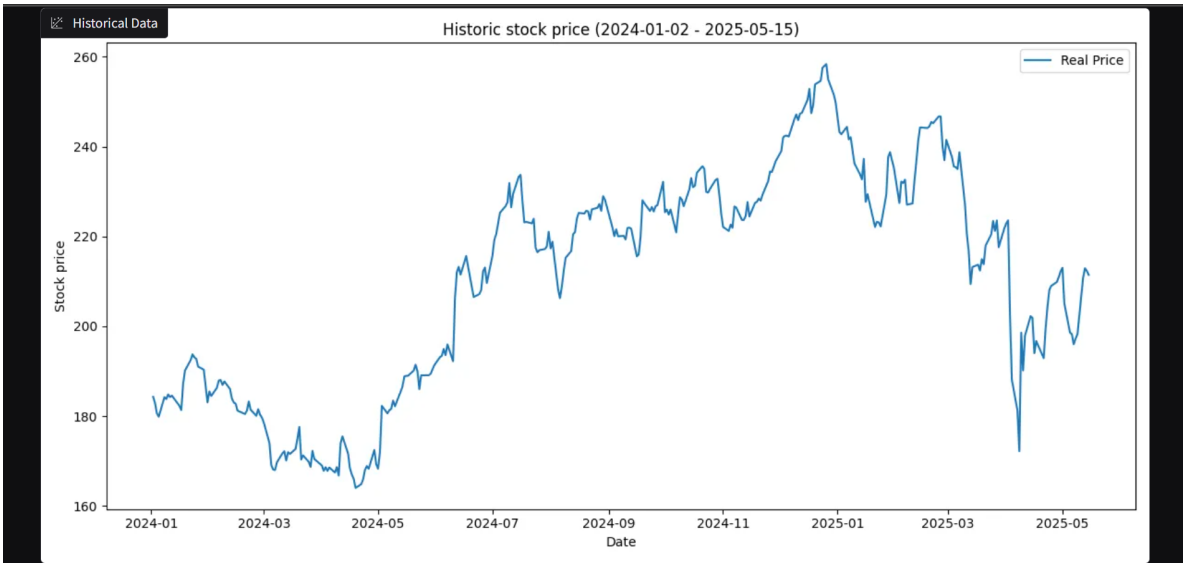
Rysunek 1: GUI - Miejsce do wprowadzenia informacji ogólnych i parametrów XGBRegressor



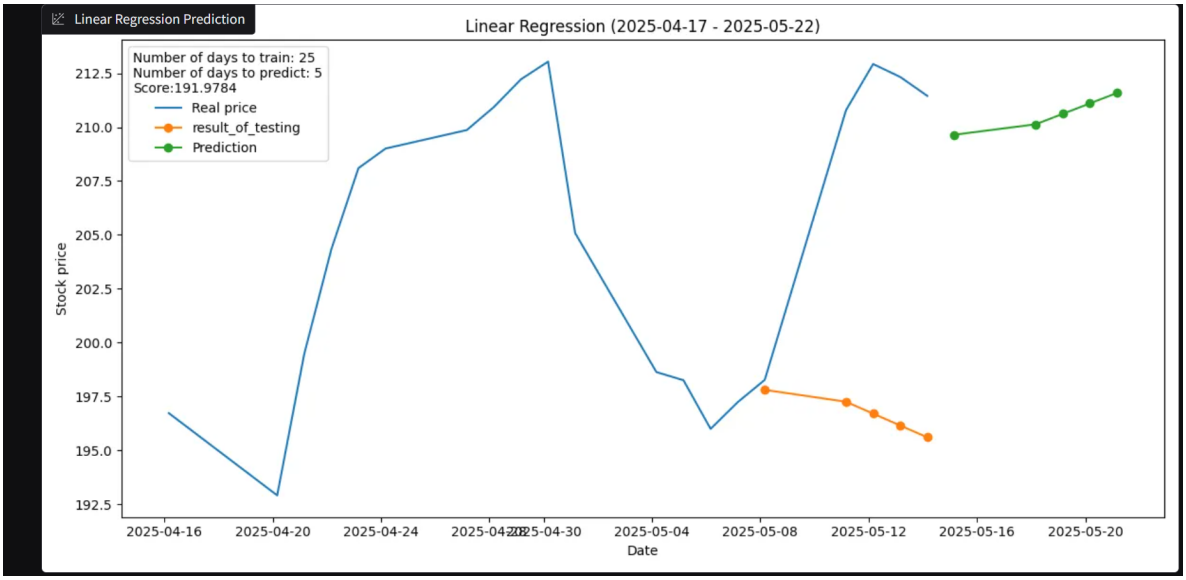
Rysunek 2: GUI - Miejsce do wprowadzenia parametrów LSTM i do poszczególnych warstw oraz przyciski Predict i Stop Training



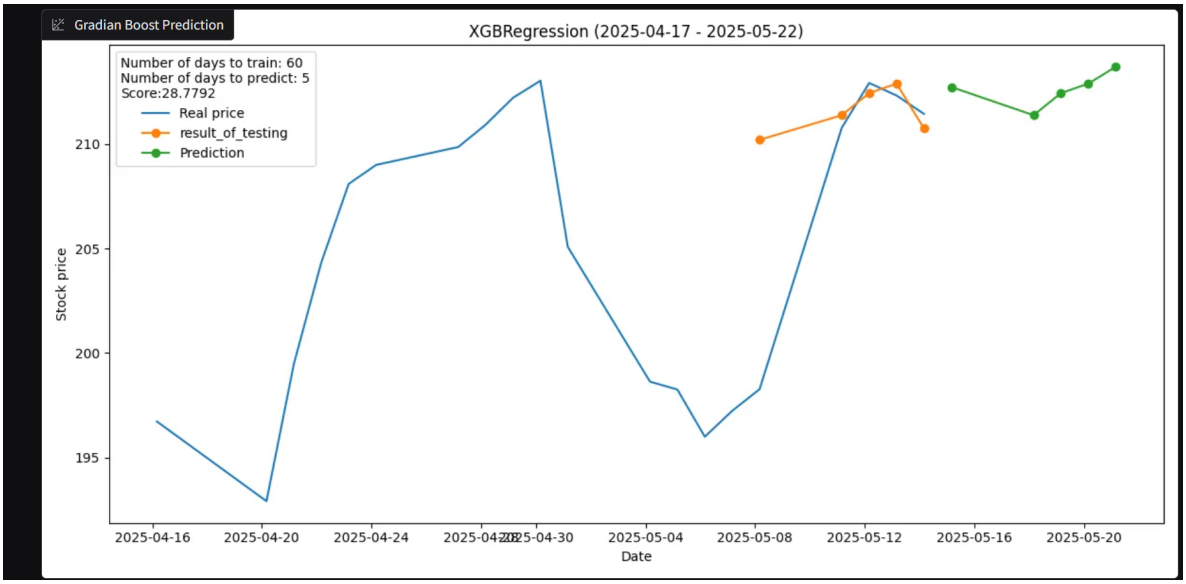
Rysunek 3: GUI - Status aplikacji informacje od modeli



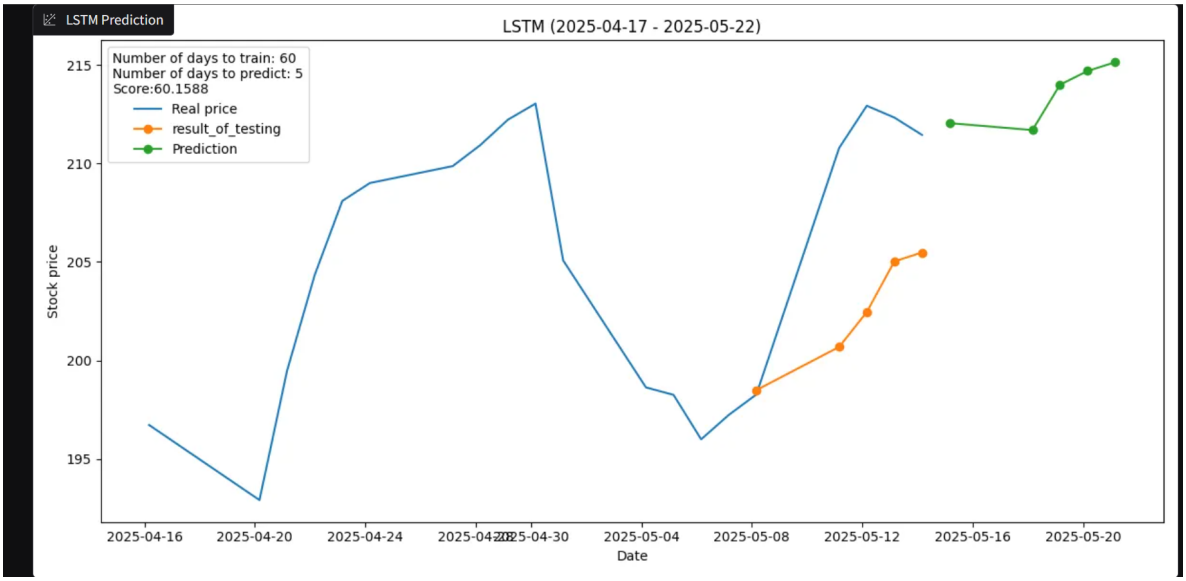
Rysunek 4: GUI - Wykres historycznych cen akcji od dnia jaki podał użytkownik



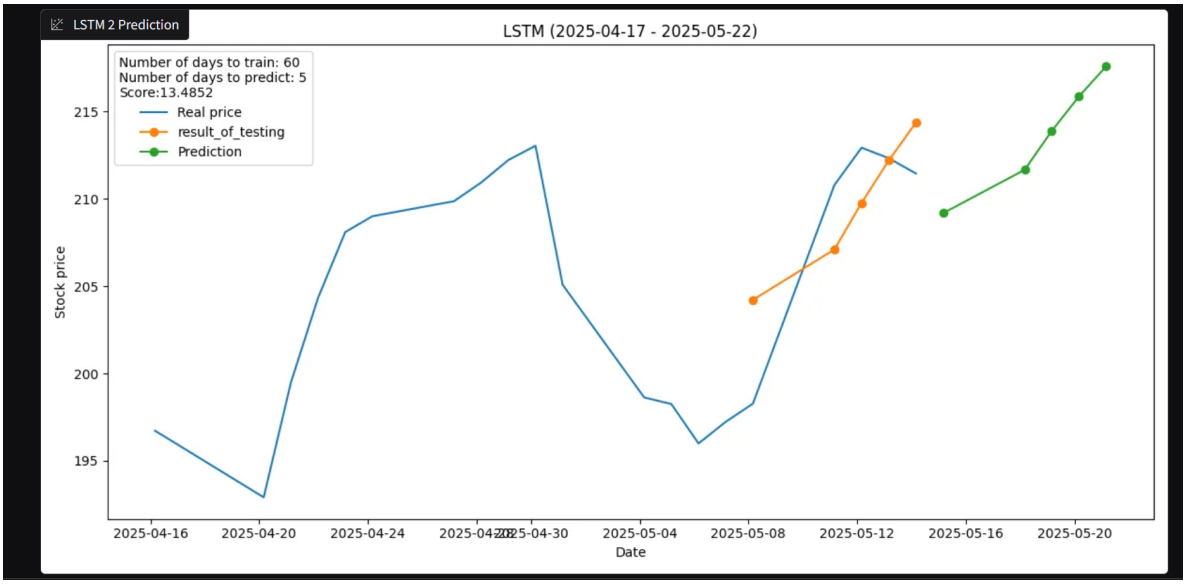
Rysunek 5: GUI - Wykres regresjii liniowej wraz z ilością dni uczenia w legendzie



Rysunek 6: GUI - Wykres XGBRegression



Rysunek 7: GUI - Wykres LSTM pierwszej implementacji



Rysunek 8: GUI - Wykres LSTM drugiej implementacji

6 Wnioski

6.1 własne obserwacje

1. Program nie gwarantuje, że przewidywania się sprawdzą nawet jeżeli testowanie jest bardzo dobre. Testując program zauważyłem, że duży wpływ na ceny akcji mają wydarzenia na świecie, wpisy prezydentów na platformie X, konferencje prasowe przedstawicieli firm itd.
2. Zauważyłem, że warto dostosowywać datę początkową taką aby obejmowała i wartość poniżej i powyżej aktualnej ceny akcji na giełdzie.
3. Warto jest ustawić datę początkową nie za wcześnie. Przykładowo nvidia weszła na giełdę w 1999 roku i do 2023 roku, akcje były warte mniej więcej 10 razy mniej niż obecnie. Dopiero w 2023 akcje wystrzeliły. Do przewidywania warto obcinać takie daty. One i wydłużają działanie i nie koniecznie zwiększają dokładność modelu
4. Modele zdecydowanie lepiej działają przewidując krótko terminowo niż długoterminowo

6.2 możliwości rozwoju tego co powstało

Mam kilka pomysłów jak można rozbudować ten projekt:

1. Dodanie więcej modeli
2. Śledzenie wydarzeń giełdowych, wpisów na X ważnych osób.
3. Powiązanie aplikacji z brokerem, aby program mógł inwestować prawdziwe pieniądze.
4. Wyznaczenie najlepszych parametrów dla każdego modelu automatycznie.
5. Zapisywanie dla danej spółki dla każdego modelu najlepsze wagi i parametry.
6. Ulepszenia GUI