

P O L I T E C H N I K A R Z E S Z O W S K A

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

**Aldona Świrad**

Predykcja szeregów czasowych: metody klasyczne i elementy uczenia maszynowego

**Praca dyplomowa inżynierska**

Opiekun pracy:  
dr Ewa Rejwer-Kosińska

Rzeszów, 2024

Spis treści

[Wstęp 5](#_Toc185347427)

[Rozdział 1 Wprowadzenie teoretyczne 6](#_Toc185347428)

[1.1 Definicja i własności szeregów czasowych 6](#_Toc185347429)

[1.2 Model ARIMA 7](#_Toc185347430)

[Rozdział 2. Metody uczenia maszynowego dla szeregów czasowych 9](#_Toc185347431)

[2.1 Model RNN 15](#_Toc185347432)

[2.2 Model LSTM 19](#_Toc185347433)

[Rozdział 3. Analiza wybranych szeregów czasowych metodami klasycznymi 23](#_Toc185347434)

[3.1 Analiza – opis, wykresy, zakres zmienności 24](#_Toc185347435)

[3.2 Dobór modelu 30](#_Toc185347436)

[Rozdział 4. Prognozowanie szeregów czasowych 30](#_Toc185347437)

[4.1 Prognozowanie metodą ARIMA 31](#_Toc185347438)

[4.2 Prognozowanie metodą LSTM 31](#_Toc185347439)

[3.3 Porównanie 48](#_Toc185347440)

Wstęp

1. Wprowadzenie teoretyczne
   1. Definicja szeregów czasowych
   2. Model ARIMA
2. Metody uczenia maszynowego dla szeregów czasowych

2.1 Model RNN

2.2 Model LSTM

1. Analiza wybranych szeregów czasowych metodami klasycznymi
   1. Analiza – opis, wykresy, zakres zmienności
   2. Dobór modelu
2. Prognozowanie szeregów czasowych

4.1 Prognozowanie metodą ARIMA

4.2 Prognozowanie metodą LSTM

4.3 Porównanie

Podsumowanie

Bibliografia

Dodatek

# Wstęp

Tekst akapitu.

Tekst akapitu.

**Cel pracy**

Celem niniejszej pracy inżynierskiej jest zbadanie i porównanie skuteczności metod analizy szeregów czasowych: metod klasycznych oraz metod opartych na uczeniu maszynowym, ze szczególnym uwzględnieniem modeli LSTM. Głównym celem pracy jest zastosowanie tych modeli do prognozowania szeregów czasowych oraz ocena ich przydatności w różnych scenariuszach analizy danych czasowych.

**Zakres pracy**

W pierwszym rozdziale pracy podane zostaną najważniejsze pojęcia z zakresu modelowania i predykcji szeregów czasowych. Rozdział drugi poświęcony zostanie analizie własności i modelowaniu wybranych szeregów czasowych, z zastosowaniem metod klasycznych. W rozdziale trzecim podane zostaną podstawowe pojęcia i metody uczenia maszynowego (w tym modele RNN i LSTM), stosowane w analizie szeregów czasowych. Porównanie wybranych metod prognozowania szeregów czasowych pod kątem ich skuteczności, wraz z wnioskami dotyczącymi przydatności poszczególnych modeli w różnych scenariuszach analizy szeregów czasowych, będą zawarte w ostatnim rozdziale pracy. W podsumowaniu zawarte będą sugestie dotyczące dalszych badań oraz potencjalnych zastosowań praktycznych uzyskanych wyników.

Praca będzie opierać się głównie na analizie literatury naukowej oraz eksperymentach przeprowadzonych na rzeczywistych danych, co umożliwi kompleksowe zbadanie tematu oraz wyciągnięcie trafnych wniosków dotyczących analizy szeregów czasowych za pomocą metod klasycznych i uczenia maszynowego.

# Rozdział 1 Wprowadzenie teoretyczne

Dodać: (trend, sezonowość, proces stochastyczny, biały szum, def. Parametry, szereg czasowy – parametry, wykresy i rysunki obrazujące szereg czasowy, poprawić wzory)

W rozdziale 1 pracy przybliżymy podstawowe pojęcia związane z szeregami czasowymi oraz zaprezentujemy jeden z najczęściej wykorzystywanych modeli do ich analizy – ARIMA **[tutaj proszę podać numer(y) pozycji literatury dot. ARIMA]**. Zrozumienie tych pojęć oraz metodologii jest kluczowe do poprawnej analizy i prognozowania szeregów czasowych.

## Definicja i własności szeregów czasowych

Szereg czasowy (ang. *time series*) **[numer(y) pozycji literatury dot. szeregów czasowych]** to sekwencja danych gromadzonych w równych odstępach czasu, np. dziennie, miesięcznie, rocznie. Każda obserwacja w szeregu czasowym składa się z dwóch elementów: momentu w czasie, do którego się odnosi, oraz wartości zmiennej mierzonej w tym momencie. Przy czym, kolejność obserwacji ma istotne znaczenie. Przykładami szeregów czasowych mogą być kursy walut, wartości indeksów giełdowych, liczba sprzedanych produktów w danym okresie, czy zmiany temperatury w ciągu roku.

W praktyce, analiza szeregów czasowych ma na celu identyfikację struktury danych, wyodrębnienie trendów, sezonowości oraz innych wzorców, a także prognozowanie przyszłych wartości na podstawie dostępnych danych historycznych, czym będziemy się zajmować w tej pracy.

***Stacjonarność szeregu czasowego***

Szeregi czasowe mogą być stacjonarne lub niestacjonarne **[numer(y) pozycji literatury dot. tego typu szeregów]**. Szereg stacjonarny charakteryzuje się stałą średnią, wariancją i autokorelacją w czasie, natomiast szereg niestacjonarny może zawierać zmieniające się średnie, wariancje lub inne nieliniowe zależności. Aby skutecznie analizować szeregi czasowe i budować modele prognostyczne, często stosuje się metody przekształcania niestacjonarnych szeregów na stacjonarne. **Proszę wymienić i bardziej szczegółowo omówić kilka z tych metod**

***Trend i sezonowość***

**Zamiast podrozdziałów w podrozdziale w taki sposób proszę odnosić się do innych charakterystyk szeregów czasowych, które będzie Pani opisywać (czyli biały szum, itp.)**

***Różnicowanie niestacjonarnego szeregu czasowego***

Różnicowanie **[numer(y) pozycji literatury dot. różnicowania szeregów czasowych]** służy do przekształcania niestacjonarnych szeregów czasowych w szeregi stacjonarne. Proces ten eliminuje trend i sezonowość, co ułatwia modelowanie fluktuacji w krótszych okresach.

Proces różnicowania polega na obliczeniu różnicy między kolejnymi obserwacjami:

**(Różnicowanie będzie Pani wykonywać podczas analizy szeregów czasowych, także być może tutaj będzie trzeba coś więcej dopisać)**

## 1.2 Model ARIMA

Model ARIMA (ang. *AutoRegressive Integrated Moving Average*) **[numer(y) pozycji literatury dot. ARIMA]** to jeden z najpopularniejszych modeli stosowanych w analizie szeregów czasowych. Jest to model liniowy, który łączy trzy kluczowe elementy: autoregresję (AR), różnicowanie (I) i średnią ruchomą (MA).

***Model autoregresyjny AR(p)***

Autoregresja polega na przewidywaniu wartości zmiennej w danym momencie na podstawie wcześniejszych obserwacji tej samej zmiennej. Model *AR(p)*, gdzie parametr *p* oznacza liczbę opóźnionych obserwacji (lagów), zakłada, że wartość zmiennej zależy od jej wcześniejszych wartości:

gdzie to wartość zmiennej w czasie *t*, to współczynniki modelu, to błąd losowy w czasie t.

***Model średniej ruchomej MA(q)***

Średnia ruchoma zakłada, że bieżąca wartość zmiennej jest kombinacją poprzednich błędów losowych. Model *MA(q)*, gdzie *q* oznacza liczbę opóźnień błędów losowych, zakłada, że wartość zmiennej w danym momencie zależy od wcześniejszych błędów:

gdzie to to błędy losowe, to współczynniki modelu.

***Model ARIMA(p,d,q)***

Model *ARIMA* łączy w sobie model autoregresji *AR*(*p*), średniej ruchomej *MA(q)* oraz różnicowanie (*d*). Model ten opisuje szereg czasowy za pomocą kombinacji wcześniejszych wartości zmiennej oraz błędów losowych, przy jednoczesnym uwzględnieniu odpowiedniej liczby różnicowań, w celu zapewnienia stacjonarności. Ogólna postać modelu *ARIMA(p, d, q)* wygląda następująco:

gdzie to przewidywana wartość zmiennej w czasie *t*, to współczynniki autoregresji, to współczynniki średniej ruchomej, błąd losowy w czasie *t*.

Model ARIMA jest szeroko stosowany w prognozowaniu szeregów czasowych ze względu na swoją elastyczność oraz zdolność do modelowania różnych wzorców, takich jak trendy i sezonowość. Jednak, dobór odpowiednich wartości parametrów modelu wymaga analizy danych oraz testów diagnostycznych, takich jak autokorelacja reszt czy testy stacjonarności.

Całość materiału potrzebna do zrozumienia zagadnień tej pracy została omówiona na kursie „Szeregi czasowe” w toku studiów, więc przytoczyłam tylko najbardziej kluczowe aspekty tej wiedzy.

# Rozdział 2. Metody uczenia maszynowego dla szeregów czasowych

Uczenie maszynowe (*ang. machine learning*) **[numer(y) pozycji literatury dot. ML]** to gałąź sztucznej inteligencji (*ang. artificial intellligence*) **[numer(y) pozycji literatury dot. AI]** zajmująca się tworzeniem różnorodnych algorytmów i modeli analizy danych, w której systemy samodzielnie się uczą na podstawie wprowadzonych danych. System uczenia maszynowego jest więc trenowany, a nie programowany w rozumieniu klasycznym. Przedstawia się mu wiele przykładów istotnych dla rozważanego problemu, a następnie znajduje wzorzec statystyczny, który umożliwia systemowi „wymyślenie” reguły poszukiwania rozwiązania. Uczenie maszynowe jest ściśle powiązane ze statystyką matematyczną, ale różni się od niej pod kilkoma ważnymi względami. Ze względu na niestandardowy sposób podejścia do rozwiązywania problemów, uczenie maszynowe umożliwia analizę złożonych, dużych zbiorów danych, których badanie nie byłoby możliwe przy pomocy tradycyjnego programowania i metod statystycznych. W rezultacie uczenie maszynowe wykorzystuje niewiele teorii matematycznych. Jest to praktyczna dyscyplina, w której idee są częściej dowodzone empirycznie, niż teoretycznie.

W zbiorze metod uczenia maszynowego można wyróżnić niezwykle popularne sieci neuronowe (*ang. neural networks*) **[numer(y) pozycji literatury dot. NN]** oraz uczenie głębokie (*ang. deep learning*) **[tutaj proszę podać numer(y) pozycji literatury dot. DL]**. Jedną z klas sieci neuronowych, stanowiących metodę uczenia głębokiego, są rekurencyjne sieci neuronowe (RNN, *ang. recurrent neural networks*) **[numer(y) pozycji literatury dot. RNN]**, a także ich wariant LSTM (*ang. long short-term memory*) **[numer(y) pozycji literatury dot. LSTM]**, które zostaną opisane w niniejszym rozdziale.

Obraz zawierający tekst, krąg

Opis wygenerowany automatycznie

**Rysunek 4.1 Proszę dodać tytuł rysunku.**

**W zapisie na rysunku powinno być „RNN – Recurrent Neural Network” oraz „LSTM – Long Short-Term Memory”**

Cechą charakterystyczną odróżniającą klasyczne programowanie od uczenia maszynowego jest to, że w metodzie klasycznej samodzielnie tworzymy algorytm (reguły) i dostarczamy dane wejściowe, które mają być przetwarzane zgodnie z tym algorytmem (regułami). Na ich podstawie otrzymujemy dane wyjściowe, stanowiące rozwiązanie problemu. W przypadku uczenia maszynowego wprowadzamy do systemu dane wejściowe i wyjściowe (przewidywane odpowiedzi), które dzielimy na *dane treningowe i testowe*. Dane treningowe służą do nauki modelu, natomiast dane testowe pozwalają sprawdzić jego skuteczność na nieznanych wcześniej przykładach. Można również wyodrębnić *dane walidacyjne*, które posłużą do wstępnej oceny modelu podczas jego budowy. Dostarczone dane są przez program interpretowane za pomocą neuronów, które są w swej istocie równaniami matematycznymi. Na wyjściu otrzymujemy algorytm (zestaw reguł), który można zastosować do pracy z nowymi danymi, celem uzyskania oryginalnych rozwiązań.

Obraz zawierający tekst, zrzut ekranu, diagram, design

Opis wygenerowany automatycznie

**Rysunek 4.2 Proszę dodać tytuł rysunku.**

***Sieć neuronowa***

Sieć neuronowa **[numer(y) pozycji literatury dot. NN]** inspirowana jest neuronem biologicznym (rysunek 4.3). Nie jest to nowy wynalazek. Pierwszą sieć neuronową opracowali już w 1943 roku Warren McCulloch i Walter Pitts. Jej reprezentantem   
w świecie biologicznym jest mózg, zaś w świecie cyfrowym komputer, który interpretuje dane wejściowe i zwraca odpowiedzi. Pojedynczy neuron (węzeł) otrzymuje dane wejściowe poprzez synapsy, modelowane przez pojedynczą liczbę lub wagę (*ang. weight*), określającą siłę znaczenia danego połączenia w neuronie. Dane wejściowe są mnożone przez wagi i sumowane, celem aktywacji węzła. Wartość funkcji aktywacji jest następnie porównywana z wartością progową (odchyleniem, *ang. bias*), celem umożliwienia modelowi przesunięcia funkcji aktywacji i lepszego dopasowania do danych. Jeżeli wartość funkcji aktywacji przekracza wartość progową, na wyjściu otrzymujemy istotną wartość dodatnią (np. 1), w przeciwnym razie dostajemy wartość 0. Otrzymana wartość jest przenoszona do kolejnych neuronów, jako dana wejściowa. W ten sposób tworzy się cała sieć połączeń (sieć neuronowa). Sztuczny neuron posiada warstwy, które możemy podzielić na warstwę wejścia, warstwy ukryte i warstwę wyjścia (patrz rysunek 4.4).

Obraz zawierający tekst, kwiat

Opis wygenerowany automatycznie

**Rysunek 4.3 Proszę dodać tytuł rysunku.**

Obraz zawierający tekst, diagram, linia, design

Opis wygenerowany automatycznie

**Rysunek 4.4 Proszę dodać tytuł rysunku.**

***Funkcje aktywacji***

*Funkcja aktywacji* **[numer(y) pozycji literatury dot. funkcji aktywacji]** to kluczowy element sztucznych sieci neuronowych, wybierany dla warstw ukrytych i warstwy wyjściowej, który decyduje o tym, czy i w jaki sposób neuron "aktywuje się" (przekazuje sygnał). Funkcja aktywacji wprowadza nieliniowość do modelu. Bez niej operacje na danych wejściowych składałyby się wyłącznie z iloczynu skalarnego danych wejściowych i wag, do których dodana zostałaby wartość progowa (odchylenie). Wówczas, bez względu na ilość warstw, mogłyby one uczyć się tylko liniowych transformacji danych wejściowych, co znacznie ograniczyłoby zakres przestrzeni hipotez sieci. Ponadto, niektóre z funkcji aktywacji dokonują normalizacji wyjścia, co oznacza przekształcenie wartości wyjściowej w określony zakres. Pomaga to w stabilnym trenowaniu sieci nauronowej.

Najpopularniejszą funkcją aktywacji jest *funkcja ReLU* (*ang. Rectified Linear Unit*) **[numer(y) pozycji literatury dot. ReLU]**.

gdzie dla ujemnych argumentów wejściowych funkcja zwraca zawsze 0, a dla dodatnich wartości pozostają bez zmian.

Obraz zawierający linia, diagram, Wykres, tekst

Opis wygenerowany automatycznie

**Rysunek 4.5 Proszę dodać tytuł rysunku.**

**Rysunek był zapewne robiony w Geogebrze ? Proszę mi przesłać kod LaTeX rysunku, odpowiednio go poprawię.**

Użycie funkcji ReLU, ze względu na jej postać, jest wydajne, ułatwia obliczenia   
i pozwala na zwiększenie szybkości obliczeń, jednak może prowadzić do sytuacji gdy neurony nie będą się uczyć - wygenerują wartość wyjściową 0. Wartość 0 zawsze jest problematyczna w uczeniu sieci neuronowych, gdyż może powodować „zapominanie” - utratę informacji, przez co uczenie jest nieefektywne.

Chcąc uniknąć takiej sytuacji, można skorzystać z innego typu funkcji aktywacji - *funkcji sigmoidalnej* **[numer(y) pozycji literatury dot. funkcji sigmoidalnej]**,postaci:

Funkcja ta osiąga wartości w przedziale (0, 1). Jej wyniki można interpretować   
w następujący sposób: wartość funkcji bliżej 0 oznacza odpowiedź negatywną, a bliżej 1 odpowiedź pozytywną. Funkcja sigmoidalna pozwala sieciom neuronowym na modelowanie złożonych zagadnień, jednak wykazuje strome nachylenie, gdy jej argumenty mieszczą się w przedziale (-2, 2). Oznacza to, że niewielkie zmiany w danych wejściowych mogą mieć istotny wpływ na dane wyjściowe.

Obraz zawierający linia, Wykres, diagram, Równolegle

Opis wygenerowany automatycznie

**Rysunek 4.6 Proszę dodać tytuł rysunku.**

**Rysunek był zapewne robiony w Geogebrze ? Proszę mi przesłać kod LaTeX rysunku, odpowiednio go poprawię.**

**Funkcja tanh** (tangens hiperboliczny) jest odmianą funkcji sigmoidalnej. Jej wartości są zawarte w przedziale (-1,1).

Podobnie, jak funkcja sigmoidalna, pozwala na modelowanie złożonych zagadnień. Jednak, ze względu na wartości symetryczne względem punktu (0, 0), ma ona szersze zastosowania w warstwach ukrytych sieci neuronowej, co ułatwia naukę kolejnych warstw.

Obraz zawierający linia, Wykres, diagram, Równolegle

Opis wygenerowany automatycznie

**Rysunek 4.7 Proszę dodać tytuł rysunku.**

**Rysunek był zapewne robiony w Geogebrze ? Proszę mi przesłać kod LaTeX rysunku, odpowiednio go poprawię.**

Wybór funkcji aktywacji jest bardzo ważny, gdyż ma wpływ na wydajność sieci. Każda z tych funkcji ma swoje wady i zalety, a ich dobór zależy od analizowanego problemu.

***Propagacja wsteczna***

*Propagacja wsteczna* (*ang. backpropagation*) **[numer(y) pozycji literatury dot. backpropagation]** to kluczowy algorytm stosowany podczas uczenia sztucznej sieci neuronowej. Polega on na minimalizacji błędu predykcji (*funkcji kosztu*) poprzez aktualizację wag i progów (odchyleń) w każdej epoce (*ang. epoch*) sieci. *Funkcja kosztu* (*ang. cost function*) **[numer(y) pozycji literatury dot. funkcji kosztu]**, nazywana również funkcją straty (*ang.* *loss function*), mierzy jak dobrze model przewiduje oczekiwane wyniki. Inaczej, funkcja ta określa różnicę pomiędzy wartościami przewidywanymi przez model, a wartościami rzeczywistymi z danych treningowych. Jej minimalizacja jest kluczowym celem podczas trenowania modelu. To ona wskazuje, czy model uczy się dobrze, czy też zachodzą trudności. Jednymi z najczęściej stosowanych funkcji kosztu są:

MSE - błąd średniokwadratowy (*ang. mean squared error*).

MAE – średni błąd bezwzględny (*ang. mean absolute error*).

gdzie to wartości rzeczywiste, to wartości przewidywane, to ilość elementów.

Celem minimalizacji funkcji kosztu propagacja wsteczna często wykorzystuje metody optymalizacji. Wagi i odchylenia są aktualizowane zgodnie z regułą *gradientu spadkowego* (*ang. gradient descent*) **[numer(y) pozycji literatury dot. gradient descent]**, zmniejszając błąd modelu poprzez przesunięcie parametrów w kierunku przeciwnym do gradientu. Algorytm oblicza gradient, korzystając z reguły łańcuchowej **[numer(y) pozycji literatury dot. reguły łańcuchowej]**, dla każdego parametru (wag i odchyleń) w każdej warstwie sieci, co pozwala określić w jakim stopniu dany parametr wpływa na błąd.

Algorytm propagacji wstecznej składa się z dwóch etapów:

1. *Przejście do przodu* - dane wejściowe są wprowadzane do warstwy wejściowej. Po połączeniu z odpowiednimi wagami i dodaniu odchyleń, są przekazywane do warstw ukrytych sieci. Każda warstwa ukryta używa funkcji aktywacji. Dane wyjściowe   
z ostatniej warstwy ukrytej są przekazywane do warstwy wyjściowej, gdzie stosowana jest kolejna funkcja aktywacji. Na koniec, dla danych wyjściowych obliczana jest wartość funkcji kosztu.

2. *Przejście do tyłu* – obliczona wartość funkcji kosztu jest propagowana z powrotem przez sieć (od warstwy wyjściowej do wejściowej), aby dostosować wagi i odchylenia, a przez to zminimalizować błąd w kolejnej iteracji.

**(Jeżeli będzie czas, może Pani tutaj podać prosty przykład działania algorytmu propagacji wstecznej)**

Głównymi zaletami algorytmu propagacji wstecznej są m.in.:

- *łatwość stosowania* - nie wymaga szerokiej wiedzy nt. sieci neuronowych,

- *elastyczność* - może być stosowana dla szerokiego zakresu zagadnień,

- *wydajność* - przyspiesza uczenie się poprzez bieżącą aktualizację parametrów.

Do wad algorytmu zalicza się natomiast:

- *problem zanikającego gradientu* - w bardzo głębokich sieciach, szczególnie przy stosowaniu sigmoidalnych funkcji aktywacji, gradienty mogą stawać się na tyle małe, że utrudnią proces uczenia się sieci,

- *problem wybuchającego gradientu* - sytuacja odwrotna do problemu zanikającego gradientu, może powodować rozbieżność sieci w trakcie uczenia się,

- *przeuczenie modelu* – czyli jego nadmierne dopasowanie, pojawia się zwykle wówczas, gdy sieć jest zbytnio złożona lub źle regulowana.

## 2.1 Model RNN

*Rekurencyjne sieci neuronowe* (*RNN,* *ang. Recurrent Neural Networks*) **[numer(y) pozycji literatury dot. RNN]** to sieci neuronowe zaprojektowane do przetwarzania danych sekwencyjnych, m.in. szeregów czasowych. Struktura RNN umożliwia rekurencję, co oznacza, że dane wyjściowe otrzymane z jednego kroku czasowego są przekazywane jako dane wejściowe do następnego kroku. Odróżnia je to od tradycyjnych sieci neuronowych, w których  dane wejściowe i wyjściowe są traktowane niezależnie.

Każdy krok czasowy w RNN można opisać jako:

gdzie to macierz wymiaru zawierająca dane wyjściowe w kroku czasowym *t*, to ilość próbek, liczba neuronów, to funkcja aktywacji, to macierz wymiaru zawierająca dane wejściowe, gdzie to liczba wejść, to macierz wymiaru zawierająca wagi dla wejść w kroku czasowym *t*, to macierz wymiaru zawierająca wagi dla wyjść z poprzedniej iteracji, to wektor   
(o wymiarze ) odchyleń dla każdego neuronu. W kroku czasowym nie istnieją żadne dane wyjściowe, zatem zakłada się, że ich wartość równa jest 0.

Neuron RNN potrafi zachować informacje o wartości stanu w poszczególnych krokach czasowych w *komórkach pamięci* (*ang.* *memory cells*). Pojedynczy neuron RNN może zapamiętywać sekwencje z około 10 lub więcej kroków czasowych, zależnie od rozważanego zagadnienia. Jest to kluczowe w przewidywaniu zdarzeń i podejmowaniu decyzji, np.: podczas przetwarzania języka naturalnego, rozpoznawania mowy, analizy wideo, czy analizy szeregów czasowych. Neuron RNN można sobie wyobrazić jako zapętlony sam w sobie (rysunek 4.8), czyli działający rekurencyjnie. Można go też przedstawić w postaci rozwijającego na kolejne neurony (rysunek 4.9) - taka wizualizacja jest bardziej przejrzysta dla dalszych rozumowań.

Obraz zawierający diagram, Czcionka, linia, tekst

Opis wygenerowany automatycznie

**Rysunek 4.8 Proszę dodać tytuł rysunku.**

Obraz zawierający diagram, tekst, szkic, linia

Opis wygenerowany automatycznie

**Rysunek 4.9 Proszę dodać tytuł rysunku.**

***Rodzaje sieci RNN***

Modele RNN możemy podzielić na różne rodzaje:

- *Sieć sekwencyjna* (*ang. sequence-to-sequence network*) - otrzymuje dane wejściowe   
i dokonuje predykcji na ciąg kolejnych kroków czasowych, np.: podając ceny produktu z *N* poprzednich dni sieć może zostać wytrenowana celem predykcji ceny produktu na 7 kolejnych dni do przodu, czyli np. od *N-7* dnia do przyszłego tygodnia.

- *Sieć sekwencyjno-wektorowa* (*ang. sequence-to-vector network*) - otrzymuje dane wejściowe i generuje wyłącznie ostateczny wynik (lub pojedynczą wartość), np.: zbierając sekwencję zdań sieć określi, czy mają one zabarwienie pozytywne - 1, czy negatywne - 0.

- *Sieć wektorowo-sekwencyjna* (*ang. vector-to-sequence network*) – w każdym kroku czasowym podawana jest taka sama informacja, na postawie której otrzymujemy wynik, np.: przedstawiamy sygnał dźwiękowy i dostajemy jego opis.

- *Koder-dekoder (ang. encoder-decoder)* – połączenie sieci sekwencyjno-wektorowej oraz wektorowo-sekwencyjnej, stosowana przykładowo do przetwarzania języka naturalnego na inny język.

**(Tutaj proszę wstawić wizualizację opisanych rodzajów sieci)**

***Propagacja wsteczna w czasie***

W rekurencyjnych sieciach neuronowych stosowany jest szczególny typ algorytmu propagacji wstecznej, tj. *propagacja wsteczna w czasie* *(BPTT, ang. Backpropagation Through Time)* **[numer(y) pozycji literatury dot. BPTT]***.* Algorytm ten rozszerza propagację wsteczną na dane, które mają strukturę czasową, umożliwiając propagowanie błędów przez wiele kroków czasowych. W sieciach rekurencyjnych, BPTT oblicza gradienty nie tylko dla wag, ale również dla "ukrytych stanów" w każdym kroku czasowym. Błąd jest propagowany zarówno wstecz przez warstwy, jak i przez czas, co oznacza, że gradienty muszą być obliczane na podstawie całej sekwencji danych. Ten rodzaj propagacji wstecznej będzie wykorzystywany w niniejszej pracy, w części praktycznej – predykcji za pomocą LSTM.

Podobnie, jak w przypadku klasycznego algorytmu propagacji wstecznej, na etapie propagacji wstecznej w czasie, może pojawić się kilka istotnych problemów. Dokonując pierwszych obliczeń przy dłuższych sekwencjach można zauważyć, że im dłuższa sekwencja tym mniej dokładne wyniki, co wydaje się sprzeczne z ideą sieci neuronowych, gdyż teoretycznie ilość danych do nauki zwiększa się. Problem ten można powiązać   
z problemem *zanikającego gradientu* (*ang. vanishing gradient*)lub *wybuchającego gradientu* (*ang. exploding gradient*) w algorytmie propagacji wstecznej.

***Problem zanikającego gradientu w algorytmie BPTT***

W algorytmie BPTT na poszczególnych krokach czasowych dochodzi wielokrotnego mnożenia wag. W przypadku, gdy wartości wag (elementy macierzy ) są z przedziału   
(-1, 1), w kolejnych krokach czasowych dochodzi do coraz mniejszej aktualizacji wag   
z poprzedniego kroku czasowego, a zatem zanikania gradientu. Efekt zanikania gradientu powoduje, że początkowy gradient w długim okresie czasu ma minimalny wpływ na końcowy wynik, utrudniając modelowi naukę i przewidywanie długoterminowych zależności.

***Problem wybuchającego gradientu w algorytmie BPTT***

Odwrotne zachowanie wykaże model, gdy wartości wag (elementy macierzy ) są   
z przedziałów . W wyniku wielokrotnego mnożenia wag, w kolejnych krokach czasowych wartości wag staną się bardzo duże, co doprowadzi do błędów numerycznych. Funkcja kosztu zacznie gwałtownie rosnąć, uniemożliwiając dalsze uczenie sieci. A czym dłuższa sekwencja tym wyniki będą bardziej odbiegały od prawdy.

Ze względu na wyżej wymienione ograniczenia, RNN nie jest odpowiednim modelem do zadań wymagających przetwarzania skomplikowanych lub długich zależności czasowych. Dlatego, w niniejszej pracy, do analizy szeregów czasowych zostanie wykorzystany *model LSTM* (*ang. Long Short-Term Memory*) **[numer(y) pozycji literatury dot. LSTM]**, który stanowi ulepszoną wersję RNN. LSTM, dzięki swojej architekturze, opierającej się na komórkach pamięci i mechaniźmie bramek, skutecznie radzi sobie zarówno z problemem zanikającego, jak i wybuchającego gradientu, co pozwala na modelowanie zależności długoterminowych.

LSTM nie tylko eliminuje główne wady klasycznych RNN, ale także cechuje się większą stabilnością i skutecznością w uczeniu sieci neuronowej. Model LSTM będzie zatem kluczowym narzędziem w realizacji celów opisanych w pracy.

## 2.2 Model LSTM

Jedną z bardziej rozbudowanych rekurencyjnych sieci neuronowych jest model LSTM (*ang. long short-term memory*) **[numer(y) pozycji literatury dot. LSTM]**. Jest on zaprojektowany tak, aby efektywnie gospodarować „pamięcią”. Jest też odpowiedzią na problem zanikającego i wybuchającego gradientu. Struktura LSTM jest o wiele bardziej skomplikowana od konwencjonalnej RNN, lecz dzięki temu model ten może swobodnie działać na długich sekwencjach danych i ich długoterminowych zależności.

***Struktura sieci LSTM***

Podstawowym zadaniem sieci LSTM jest uczenie się jakie informacje należy zapamiętywać przez dłuższy okres czasu, a jakie odrzucać. W celu ograniczenia zakresu przekazywanych informacji, w sieci LSTM wprowadzono trzy bramki (zapominającą, wejściową i wyjściową), patrz rysunek 4.10. W odróżnieniu od RNN, składającej się   
z jednej warstwy sieci neuronowej (z funkcją aktywacji sigmoidalną, tanh lub ReLU), sieć LSTM składa się z trzech warstw sigmoidalnych oraz warstwy z funkcją aktywacji­ tanh. Stan pamięci długotrwałej (*ang. cell*) oznaczono na rysunku linią zieloną, zaś stan pamięci krótkotrwałej linią czerwoną. Stan został pozbawiony niewygodnego mnożenia wag, dzięki czemu usunięto problem zanikającego lub wybuchającego gradientu. Stan posiada wagi dla każdej kolejnej bramki, dzięki czemu w wygodny sposób będzie je modyfikował, nie tracąc istotnych informacji.

Obraz zawierający tekst, diagram, zrzut ekranu, Plan

Opis wygenerowany automatycznie

**Rysunek 4.10 Proszę dodać tytuł rysunku.**

Obraz zawierający tekst, zrzut ekranu, Czcionka, diagram

Opis wygenerowany automatycznie

**Rysunek 4.11 Proszę dodać tytuł rysunku.  
(na rysunku, zamiast przy linii czerwonej proszę wpisać Natomiast przy szarej linii poniżej proszę wpisać )**

***Bramka zapominająca* *(ang. forget gate)***pełni głownie rolę zapominania (usuwania informacji, które nie są przydatne), lecz co ciekawe ma ona też wpływ na zapamiętywanie poprzez dodanie nowych „wspomnień”. Do bramki podawane są dane wejściowe oraz , które są mnożone przez macierze wag odpowiednio i . Do sumy iloczynów danych wejściowych i wag dodawane jest odchylenie (bias). Wynik podlega działaniu sigmoidalnej funkcji aktywacji

Jeżeli wyjście z funkcji aktywacji wynosi 0, to część informacji jest zapominana, gdy zaś wynosi 1, to informacja jest przechowywana do dalszego użycia.

Obraz zawierający tekst, zrzut ekranu, diagram, Równolegle

Opis wygenerowany automatycznie

**Rysunek 4.12 Proszę dodać tytuł rysunku.  
(na rysunku, przy linii czerwonej proszę wpisać natomiast przy zielonej )**

***Bramka wejściowa*** ***(ang. input gate)***decyduje o dodawaniu nowych przydatnych informacji do aktualnej pamięci długotrwałej. W tym celu dane wejściowe są najpierw poddawane działaniu sigmoidalnej funkcji aktywacji

Spośród tych danych wybierane są informacje, które mają zostać zapamiętane. Korzystając następnie z funkcji aktywacji tanh, przekazującej wartości z zakresu [-1, 1]

otrzymujemy wektor wartości nowego potencjalnego stanu długotrwałego

Jest to zatem główna warstwa generująca pamięć długotrwałą, a więc tą która w większym stopniu wpływa na wynik końcowy i jest kluczowa w strukturze całego modelu.

Uzyskany nowy stan długotrwały wykorzystamy w ***bramce wyjściowej (ang. output gate).*** Bramka ta odpowiada za wyodrębnienie z aktualnego stanu komórki istotnych informacji, które zostaną wyprowadzone na wyjściu. W pierwszej kolejności nowy wektor stanu długotrwałego poddawany jest działaniu funkcji , po czym dane wejściowe regulowane są za pomocą funkcji sigmoidalnej

Na koniec, wartości regulowane przez funkcję sigmoidalną oraz wartości wektora , są mnożone celem wyodrębnienia danych wyjściowych, a za razem danych wejściowych do kolejnej komórki

Obraz zawierający tekst, zrzut ekranu, diagram, róż

Opis wygenerowany automatycznie

**Rysunek 4.13 Proszę dodać tytuł rysunku.**

**(na rysunku, przy linii czerwonej proszę wpisać natomiast przy szarej )**

Rozdział 3 Analiza wybranych szeregów czasowych metodami klasycznymi

Dane do analizy, która zostanie przeprowadzona w niniejszym rozdziale, pobrano ze strony <https://fred.stlouisfed.org/>. *Federal Reserve Economic Data (FRED),* to platforma internetowa Federal Reserve Bank of St. Louis, która oferuje otwarty dostęp do szerokiej bazy danych ekonomicznych, dotyczących m.in. inflacji, zatrudnienia, produkcji, stóp procentowych, bilansów handlowych. Jest to jedno z najważniejszych źródeł danych gospodarczych w Stanach Zjednoczonych i na świecie.

1. Wybrano trzy typy szeregów czasowych: stacjonarny, z trendem, a także z trendem   
   i sezonowością, zawierające dane rzeczywiste zgromadzone w okresach miesięcznych od 2014.01.01 do 2024.01.01. Obróbkę danych, analizę i predykcję przeprowadzono przy użyciu języka programowania Python w środowisku Microsoft Visual Studio. Do tego celu wykorzystano biblioteki: **numpy**, **pandas** i **matplotlib**.
2. import numpy as np # algebra
3. import pandas as pd # data processing, zarządzanie plikami, ramki danych
4. import matplotlib as mpl # wykresy
5. import matplotlib.pyplot as plt # wizualizacja danych
6. import matplotlib.dates as mdates # wizualizacja danych – dat
7. from statsmodels.tsa.seasonal import seasonal\_decompose # dekompozycja szeregów
8. 3.1 Analiza szeregów czasowych – opis, wykresy, zakres zmienności
9. ***Szereg stacjonarny***
10. Jako pierwszy do analizy wybrano szereg stacjonarny z cechą cykliczności **(Proszę podać link do źródła danych)**. Przedstawia on jednomiesięczną realną stopę procentową (ang. 1-Month Real Interest Rate). Fragment danych przedstawiono w tabeli 3.1, a wizualizację całego szeregu czasowego pokazano na rysunku 3.1. Kod służący do pobrania danych podano poniżej:
11. data\_rate = pd.read\_csv("C:/Users/Aldona/Documents/GitHub/Machine-learning-in-time-series-analysis/data/interest\_rate.csv", sep=',', encoding='utf-8', index\_col = 'DATE', parse\_dates = True) # upewnienie się, że daty będą rozpoznawane jako daty
12. df\_rate = pd.DataFrame(data\_rate)
13. df\_rate.columns.values[0] = 'REAL\_INTEREST\_RATE'
14. df\_rate.index.freq = 'MS' # zaznaczam, że dane są miesięczne (zazwyczaj powinno się i tak ustawić automatycznie)
15. print(df\_rate.head(10))
16. Za pomocą analogicznego kodu zaimportowano także pozostałe dwa szeregi czasowe.
17. **Proszę podać wniosek, czy dane wymagały jakiejś dodatkowej obróbki ?!**

|  |  |
| --- | --- |
| **DATE** | **REAL\_INTEREST\_RATE** |
| 2014-01-01 | -1.547831 |
| 2014-02-01 | -1.563561 |
| 2014-03-01 | -0.425359 |
| 2014-04-01 | -1.576272 |
| 2014-05-01 | -1.945522 |
| 2014-06-01 | -1.248905 |
| 2014-07-01 | -2.327665 |
| 2014-08-01 | -1.696280 |
| 2014-09-01 | -2.775060 |
| 2014-10-01 | -1.153638 |

1. **Obraz zawierający tekst, linia, Wykres, pismo odręczne

   Opis wygenerowany automatycznieTabela 3.1 Proszę dodać tytuł tabeli.**
2. **Rysunek 3.1 Proszę dodać tytuł rysunku.**
3. Można zauważyć, że dane charakteryzuje brak trendu, czy sezonowości. Widoczna jest natomiast cykliczność danych.
4. **Przydałoby się tutaj zastosować testy stacjonarności szeregu (np. rozszerzonego testu Dickeya-Fullera, testu Kwiatkowski-Phillips-Schmidt-Shin), które potwierdziłyby powyższe spostrzeżenia, a następnie podać wnioski wynikające z testów.**
5. Po zastosowaniu dekompozycji szeregu otrzymujemy
6. **Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**
7. Obraz zawierający tekst, linia, Czcionka, Wykres

   Opis wygenerowany automatycznie
8. **Rysunek 3.2 Proszę dodać tytuł rysunku.**
9. **Proszę podać wnioski z dekompozycji (również dotyczące sezonowości i reszt).**
10. ***Szereg z trendem***
11. Kolejnym szeregiem czasowym, wybranym do analizy, jest szereg opisujący indeks średnich cen konsumpcyjnych w miastach **(Proszę podać link do źródła danych)** (ang. Consumer Price Index (CPI) for all urban consumers). Fragment zbioru danych podano   
    w tabeli 3.2, zaś wizualizację szeregu czasowego przedstawiono na rysunku 3.3
12. **(Czy dane wymagały jakiejś dodatkowej obróbki ?!)**

|  |  |
| --- | --- |
| **DATE** | **CPI** |
| 2014-01-01 | 235.288 |
| 2014-02-01 | 235.547 |
| 2014-03-01 | 236.028 |
| 2014-04-01 | 236.468 |
| 2014-05-01 | 236.918 |
| 2014-06-01 | 237.231 |
| 2014-07-01 | 237.498 |
| 2014-08-01 | 237.460 |
| 2014-09-01 | 237.477 |
| 2014-10-01 | 237.430 |

1. **Tabela 3.2 Proszę dodać tytuł tabeli.**
2. Obraz zawierający linia, Wykres, tekst, diagram

   Opis wygenerowany automatycznie**Rysunek 3.3 Proszę dodać tytuł rysunku.**
3. Widzimy wyraźnie zaznaczający się trend. Dekompozycja szeregu (rysunek 3.4) potwierdza istnienie trendu w analizowanym szeregu czasowym.
4. **Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**
5. Obraz zawierający tekst, linia, Czcionka, Wykres

   Opis wygenerowany automatycznie
6. **Rysunek 3.4 Proszę dodać tytuł rysunku.**
7. **Proszę podać wnioski wynikające z dekompozycji szeregu (również dotyczące sezonowości i reszt).**
8. ***Szereg z trendem i sezonowością***
9. Ostatnim szereg czasowy poddany analizie jest szeregiem z widoczną sezonowością oraz delikatnie zaznaczonym trendem. **(Proszę opisać jakie dane zawiera ten zbiór   
   i podać link do źródła danych).** Fragment zbioru danych podano w tabeli 3.3, zaś wizualizację szeregu czasowego przedstawiono na rysunku 3.5.
10. **(Czy dane wymagały jakiejś dodatkowej obróbki ?!)**

|  |  |
| --- | --- |
| **DATE** | **CPI** |
| 2014-01-01 | 235.288 |
| 2014-02-01 | 235.547 |
| 2014-03-01 | 236.028 |
| 2014-04-01 | 236.468 |
| 2014-05-01 | 236.918 |
| 2014-06-01 | 237.231 |
| 2014-07-01 | 237.498 |
| 2014-08-01 | 237.460 |
| 2014-09-01 | 237.477 |
| 2014-10-01 | 237.430 |

1. **Tabela 3.3 Proszę dodać tytuł tabeli.**
2. Obraz zawierający tekst, linia, pismo odręczne, Wykres

   Opis wygenerowany automatycznie**Rysunek 3.5 Proszę dodać tytuł rysunku.**
3. Widać wyraźnie zaznaczoną sezonowość, a także delikatny trend na przestrzeni 10-ciu lat, przyśpieszający na przestrzeni ostatnich 5-ciu lat.
4. **Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**
5. Obraz zawierający tekst, linia, Czcionka, Wykres

   Opis wygenerowany automatycznie
6. **Rysunek 3.6 Proszę dodać tytuł rysunku.**
7. Dekompozycja szeregu (rysunek 3.6) wskazuje na ….
8. **Proszę podać wnioski wynikające z dekompozycji szeregu (również dotyczące sezonowości i reszt).**
9. Dodać informacje o obszarze zmienności

***Testy stacjonarności***

W celu sprawdzenia stacjonarności szeregu, w pierwszej kolejności przeprowadza się testy stacjonarności. Stacjonarność jest kluczowym założeniem wielu modeli ekonometrycznych i analiz szeregów czasowych, ponieważ umożliwia uniknięcie problemów związanych z niestabilnością wariancji i autokorelacją danych.

Do analizy stacjonarności wykorzystujemy następujące testy:

1. **Rozszerzony test Dickeya-Fullera** (ang. *Augmented Dickey-Fuller Test*) – test pierwiastka jednostkowego, który wykrywa obecność trendu w szeregu czasowym. Hipotezy: H₀: Szereg nie jest stacjonarny. H₁: Szereg jest stacjonarny.
2. **Test Phillipsa-Perrona** – test statystyczny bazujący na estymacji autokorelacji reszt. Hipotezy: H₀: Szereg nie jest stacjonarny. H₁: Szereg jest stacjonarny.
3. **Test Kwiatkowskiego-Phillipsa-Schmidta-Shina** (*KPSS Test*) – test oparty na analizie wariancji między próbkami z różnych okresów szeregu czasowego. Hipotezy: H₀: Szereg jest stacjonarny. H₁: Szereg nie jest stacjonarny.

Przyjmujemy poziom istotności α=0,05. Aby ocenić wyniki, sprawdzamy wartość p-value:

* Jeśli p-value jest większe od poziomu istotności, nie ma podstaw do odrzucenia hipotezy zerowej.
* Jeśli p-value jest mniejsze niż 0.05, hipotezę zerową odrzucamy na rzecz hipotezy alternatywnej.

Aby uznać szereg za stacjonarny, wszystkie testy muszą to potwierdzić. Jeśli choć jeden test wskaże, że szereg jest niestacjonarny, konieczne jest jego przekształcenie do formy stacjonarnej przez różnicowanie.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Po wykonaniu testów wynika że pierwszy szereg nie jest stacjonarny. Z tego powodu dokonałam różnicowania, aby doprowadzić go do stacjonarności.

Obraz zawierający tekst, Czcionka, linia, zrzut ekranu

Opis wygenerowany automatycznie

Po zróżnicowaniu szereg jest już stacjonarny. W przypadku różnicowania szeregu czasowego, pierwsza wartość różnicowana zawsze będzie NaN – wartością pustą, ponieważ dla niej nie ma wcześniejszej wartości do obliczenia różnicy. Przed wykonaniem testów, należy usunąć puste wartości.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Analogicznie wygląda sytuacja z szeregiem drugim i trzecim, które podlegają analizie.

Wyniki dla szeregu drugiego przed różnicowaniem. Szereg nie jest stacjonarny.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Po wykonaniu testów wynika, że drugi szereg nie jest stacjonarny. Z tego powodu dokonałam różnicowania, aby doprowadzić go do stacjonarności.

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Niestety po pierwszym różnicowaniu nadal nie wszystkie testy wykazują stacjonarność szeregu, dlatego dokonuje różnicowania drugiego rzędu.

Obraz zawierający tekst, Czcionka, linia, zrzut ekranu

Opis wygenerowany automatycznie

Po kolejnym zróżnicowaniu szereg jest już stacjonarny. Podobnie jak wcześniej usuwam wartości puste przed wykonaniem testów.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Jak pokazują wyniki szereg jest stacjonarny, a więc będzie można go wykorzystywać do dalszych analiz metodami klasycznymi.

Teraz warto przyglądnąć się ostatniemu szeregowi którego analizuję.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Po wykonaniu testów wynika, że drugi szereg nie jest stacjonarny. Z tego powodu dokonałam różnicowania, aby doprowadzić go do stacjonarności.

Obraz zawierający tekst, Czcionka, zrzut ekranu, linia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

Podobnie jak dla szeregu drugiego, niestety po pierwszym różnicowaniu nadal nie wszystkie testy wykazują stacjonarność szeregu, dlatego dokonuje różnicowania drugiego rzędu.

Obraz zawierający tekst, Czcionka, zrzut ekranu, pismo odręczne

Opis wygenerowany automatycznie

Po kolejnym zróżnicowaniu szereg jest już stacjonarny. Podobnie jak wcześniej usuwam wartości puste przed wykonaniem testów.

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

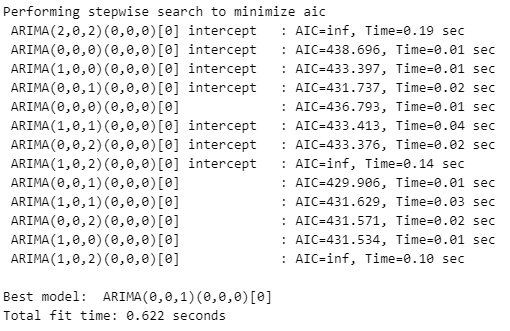
Dzięki temu, że szeregi są stacjonarne można dokonywać na nich analiz klasycznych, takich jak ARIMA. Do metody LSTM różnicowanie nie jest konieczne, lecz przy dużych błędach w predykcjach również można je zastosować.

## Dobór modelu

Funkcja auto\_arima z biblioteki pmdarima automatycznie przeszukuje przestrzeń parametrów ARIMA, wybierając te, które minimalizują kryterium informacyjne, takie jak AIC (Akaike Information Criterion) lub BIC (Bayesian Information Criterion). Ułatwia to optymalizację modelu bez potrzeby ręcznego testowania wielu kombinacji parametrów. Dzięki tym zaletom, zamiast ręcznie dobierać model zrobię to za pomocą narzędzi automatycznych. Do analizy podaję dane po zróżnocowaniu. Usuwamy wszelkie wartości, które są puste.

|  |
| --- |
| from pmdarima import auto\_arima  rate\_fit = auto\_arima(df\_rate['RATE\_diff'].dropna(), trace = True)  cpi\_fit = auto\_arima(df\_cpi['CPI\_diff2'].dropna(), trace = True)  mat\_fit = auto\_arima(df\_materials['MAT\_diff2'].dropna(), trace = True) |

W przypadku dopasowywania modelu ARIMA do danych pierwszego szeregu, najlepszym modelem okazał się ARIMA(0,0,1) z wyrazem wolnym, osiągając wartość AIC równą 429.906, co czyni go najskuteczniejszym wyborem spośród testowanych modeli. Model ten uwzględniał jedynie składnik średniej ruchomej (q=1) i wyraz wolny, sugerując, że dane wymagają uwzględnienia opóźnionych błędów, ale nie mają silnych zależności autoregresyjnych. Chociaż testowano również bardziej złożone modele, takie jak ARIMA(1,0,1) czy ARIMA(0,0,2), ich AIC były wyższe, co oznacza, że dodanie dodatkowych parametrów nie poprawiło znacząco dopasowania. Wyniki sugerują, że dla tych danych pierwszy model ARIMA(0,0,1) był najbardziej odpowiedni, oferując proste, ale efektywne dopasowanie.



W przypadku dopasowywania modelu ARIMA do drugiego szeregu, najlepszym modelem okazał się ARIMA(0,0,2) z wyrazem wolnym, który osiągnął minimalną wartość AIC równą 229.137. Model ten uwzględniał składnik średniej ruchomej (q=2) i wyraz wolny, sugerując, że dane wymagają uwzględnienia opóźnionych błędów, ale nie mają silnych zależności autoregresyjnych (p=0). Testowane były także bardziej złożone modele, takie jak ARIMA(1,0,2) i ARIMA(0,0,3), ale ich AIC były wyższe, co wskazuje na to, że dodanie dodatkowych parametrów nie poprawiło istotnie dopasowania. Wyniki sugerują, że dla danych drugiego najlepszym rozwiązaniem jest model ARIMA(0,0,2), który skutecznie uwzględnia opóźnione błędy, oferując prostsze i bardziej efektywne dopasowanie niż modele z większą liczbą parametrów.

Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne i białe

Opis wygenerowany automatycznie

W przypadku dopasowywania modelu ARIMA do trzeciego szeregu, najlepszym modelem okazał się ARIMA(4,0,0) z wyrazem wolnym, który osiągnął minimalną wartość AIC równą 2279.937. Model ten uwzględniał cztery składniki autoregresyjne (p=4) i brak składnika średniej ruchomej (q=0), co sugeruje, że dane wykazują silne zależności autoregresyjne, które najlepiej modelować za pomocą większej liczby opóźnionych wartości. Testowane były także inne modele, takie jak ARIMA(5,0,0) oraz ARIMA(4,0,1), jednak ich AIC były wyższe, co oznacza, że dodanie dodatkowych parametrów nie poprawiło znacząco dopasowania. Wyniki wskazują, że dla danych trzeciego najlepiej sprawdził się model ARIMA(4,0,0), który skutecznie uchwycił zależności w danych, oferując najbardziej efektywne dopasowanie wśród przetestowanych wariantów.Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne i białe

Opis wygenerowany automatycznie

Automatyczny dobór modeli przy użyciu funkcji auto\_arima umożliwił znalezienie najbardziej optymalnych parametrów dla każdego szeregu czasowego. Wyniki pokazują, że różne szeregi mają odmienne cechy – od prostych struktur po bardziej złożone, wymagające podwójnego różnicowania lub uwzględnienia wpływu błędów. Dzięki temu proces automatyzacji pozwolił na szybkie i skuteczne dopasowanie modeli, minimalizując wartości kryterium informacyjnego AIC i poprawiając jakość predykcji.

# Rozdział 4. Prognozowanie szeregów czasowych

Modele LSTM (a także inne modele RNN) nie wymagają badania stacjonarności szeregu czasowego, ponieważ potrafią efektywnie pracować na danych niestacjonarnych. Jest to jedna z ich kluczowych zalet w porównaniu do klasycznych metod analizy szeregów czasowych, gdzie stacjonarność jest niezbędnym warunkiem. W przypadku stosowania tradycyjnych metod, takich jak ARIMA, konieczne jest zatem zadbanie o stacjonaryzację danych.

Jeśli jednak model LSTM lub RNN nie daje satysfakcjonujących wyników, warto rozważyć stacjonaryzację szeregu czasowego. Proces ten, polegający na usunięciu trendu lub sezonowości, może pomóc modelowi lepiej nauczyć się wzorców w danych, co często przekłada się na poprawę jakości prognoz.

Ważnym krokiem jest również analiza długości szeregu czasowego oraz odpowiedni podział danych na zbiór treningowy, walidacyjny i testowy, aby umożliwić efektywne uczenie modelu oraz jego weryfikację.

## 4.1 Prognozowanie metodą ARIMA

Dlaczego w szeregu pierwszym i drugim nie może znaleźć mi predykcji a w 4 jest ok?

Szereg pierwszy model+predykcje

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Szereg drugi model+predykcje

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Trzeci model działa (daje jakieś wyniki)

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, numer

Opis wygenerowany automatycznie

## 4.2 Prognozowanie metodą LSTM

Można również podejść do prognozowania za pomocą metod uczenia maszynowego. Modelem, którego użyję będzie opisany wcześniej LSTM.

Z racji, że dane są z przedziały 2014-01 do 2024-01, podzielone są one na 10 lat i jeden miesiąc, a więc ogólniej na 121 miesięcy. Dane podzieliłam w stosunku około 2:1, czyli początkowe 97 miesięcy przeznaczone są na część testową, a pozostałe na część treningową.

Ze pomocą biblioteki MinMaxScaler dane zostały zeskalowane do przedziału (0,1), aby łatwiej się na nich pracowało, a proces uczenia przebiegał lepiej.

Tabela 1. Zestawienie danych z pierwszego roku niezeskalowanych i zeskalowanych dla zbioru reprezentującego szereg stacjonarny.

|  |  |  |
| --- | --- | --- |
| **DATE** | **REAL\_INTEREST\_RATE** | **SCALED\_RATE** |
| 2014-01-01 | -1,54783 | 0,367893 |
| 2014-02-01 | -1,56356 | 0,366291 |
| 2014-03-01 | -0,42536 | 0,482203 |
| 2014-04-01 | -1,57627 | 0,364997 |
| 2014-05-01 | -1,94552 | 0,327394 |
| 2014-06-01 | -1,2489 | 0,398335 |
| 2014-07-01 | -2,32766 | 0,288477 |
| 2014-08-01 | -1,69628 | 0,352776 |
| 2014-09-01 | -2,77506 | 0,242916 |
| 2014-10-01 | -1,15364 | 0,408037 |
| 2014-11-01 | -0,0394 | 0,521508 |
| 2014-12-01 | -2,86519 | 0,233737 |

Tabela 2. Zestawienie danych z pierwszego roku niezeskalowanych i zeskalowanych dla zbioru reprezentującego szereg z trendem.

|  |  |  |
| --- | --- | --- |
| **DATE** | **CPI** | **SCALED\_CPI** |
| 2014-01-01 | 235,288 | 0,011745 |
| 2014-02-01 | 235,547 | 0,017368 |
| 2014-03-01 | 236,028 | 0,027811 |
| 2014-04-01 | 236,468 | 0,037363 |
| 2014-05-01 | 236,918 | 0,047133 |
| 2014-06-01 | 237,231 | 0,053928 |
| 2014-07-01 | 237,498 | 0,059725 |
| 2014-08-01 | 237,46 | 0,0589 |
| 2014-09-01 | 237,477 | 0,059269 |
| 2014-10-01 | 237,43 | 0,058249 |
| 2014-11-01 | 236,983 | 0,048544 |
| 2014-12-01 | 236,252 | 0,032674 |

Tabela 3. Zestawienie danych z pierwszego roku niezeskalowanych i zeskalowanych dla zbioru reprezentującego szereg z sezonowością.

|  |  |  |
| --- | --- | --- |
| **DATE** | **MATERIALS** | **SCALED\_MATERIALS** |
| 2014-01-01 | 19688 | 0,031619 |
| 2014-02-01 | 18801 | 0 |
| 2014-03-01 | 24103 | 0,188999 |
| 2014-04-01 | 30137 | 0,404092 |
| 2014-05-01 | 33416 | 0,520978 |
| 2014-06-01 | 30072 | 0,401775 |
| 2014-07-01 | 28642 | 0,3508 |
| 2014-08-01 | 26446 | 0,27252 |
| 2014-09-01 | 26195 | 0,263573 |
| 2014-10-01 | 27329 | 0,303996 |
| 2014-11-01 | 24821 | 0,214594 |
| 2014-12-01 | 24056 | 0,187324 |

***Generator***

TimeseriesGenerator to narzędzie dostępne w bibliotece **Keras** (część TensorFlow) używane do przetwarzania szeregów czasowych podczas uczenia modeli sieci neuronowych. Pomaga automatycznie tworzyć **sekwencje danych wejściowych** i odpowiadające im **dane wyjściowe** na podstawie danych szeregów czasowych, co jest będzie przydatne dla naszego problemu.

Obraz zawierający tekst, Czcionka, linia, diagram

Opis wygenerowany automatycznie

Rysunek 1 Przykład działania generatora dla 3 wejść i jednego wyjścia.

W moim zadaniu potrzebuję podzielić dane na sekwencje o stałej długości - 12 miesięcy oraz przewidywać przyszłe wartości, czyli następny miesiąc. Ręczne tworzenie takich sekwencji może być skomplikowane i czasochłonne, dlatego TimeseriesGenerator zautomatyzuje ten proces.

Jeśli dany szereg ma więcej cech (kolumn) niż jedną można ustawić dla generatora większą liczbę. Przykłady szeregów, które używam mają tylko po jednej cesze.

Wszystkie trze szeregi zostały przetworzone przez generator do dalszych działań. Przykład działania generatora dla szeregu stacjonarnego w pierwszym kroku można zobaczyć poniżej.

|  |
| --- |
| Wektor X (wektor danych wejściowych):  [[[0.36789334]  [0.36629144]  [0.48220301]  [0.36499705]  [0.32739352]  [0.39833529]  [0.28847711]  [0.3527757 ]  [0.24291551]  [0.40803698]  [0.52150835]  [0.23373715]]]  Wyjście y:  [[0.74678252]] |

***Definiowanie modelu LSTM***

Kojelnym krokiem jest zdefiniowanie modelu sieci neuronowej LSTM (Long Short-Term Memory) przy użyciu klasy Sequential z biblioteki Keras. Model składa się z dwóch głównych warstw: warstwy LSTM i warstwy Dense. Warstwa LSTM zawiera 100 neuronów i wykorzystuje funkcję aktywacji ReLU, która jest powszechnie stosowana ze względu na swoją prostotę i skuteczność w eliminowaniu problemu zanikania gradientu. Parametr input\_shape definiuje rozmiar danych wejściowych, gdzie n\_input oznacza liczbę kroków czasowych, a n\_features liczbę cech na każdy krok. Warstwa Dense pełni rolę wyjściową i zawiera tylko jeden neuron, co jest odpowiednie dla zadań regresji, gdzie przewidywana jest pojedyncza wartość, np. następny krok czasowy. Model jest kompilowany przy użyciu optymalizatora Adam, który automatycznie dostosowuje współczynniki uczenia, oraz funkcji straty MSE (Mean Squared Error), powszechnie stosowanej w regresji. Na końcu metoda summary() wyświetla podsumowanie architektury modelu, prezentując liczbę warstw, ich kształty oraz całkowitą liczbę parametrów do nauczenia. Taki model jest idealny do analizy szeregów czasowych, prognozowania wartości przyszłych oraz innych zadań związanych z danymi sekwencyjnymi.

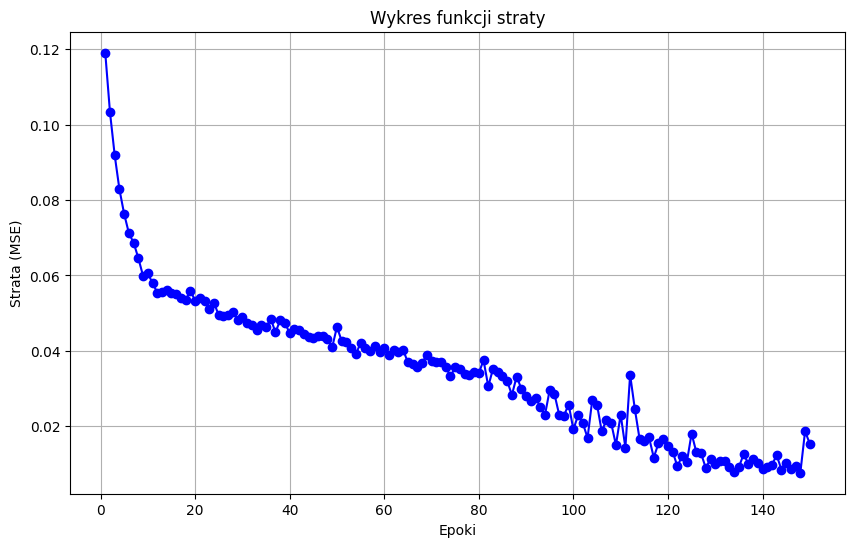
Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

***Predykcja dla szeregu stacjonarnego – trenowanie modelu***

Na tym etapie model LSTM został przystosowany do danych treningowych za pomocą funkcji fit(), która wywołuje proces uczenia modelu. Dane wejściowe zostały przekazane w postaci generatora szeregów czasowych (obiektu typu TimeseriesGenerator), co pozwala na wydajne przetwarzanie danych sekwencyjnych w partiach (batchach). Proces trenowania został przeprowadzony przez **75 epok**, czyli 75 pełnych iteracji na całym zbiorze treningowym. Podczas każdej epoki model dostosowywał swoje wagi i przesunięcia na podstawie gradientów wyliczanych przy użyciu funkcji straty MSE (Mean Squared Error), która mierzy różnicę pomiędzy wartościami przewidywanymi a rzeczywistymi. Po zakończeniu treningu historia wartości funkcji straty została zapisana w atrybucie history.history['loss']. Za pomocą biblioteki matplotlib straty te zostały przedstawione na wykresie w funkcji liczby epok, co pozwala na ocenę efektywności modelu w czasie. Zmniejszanie wartości funkcji straty na wykresie świadczy o tym, że model stopniowo się uczył i poprawiał swoje przewidywania wraz z kolejnymi epokami. Dzięki temu można ocenić, czy model się konwergował, czyli czy proces uczenia doprowadził do stabilnego poziomu błędu.

|  |
| --- |
| # zostosuje model do danych (fit model)  model.fit(generator, epochs=75)  #funkcja straty - badanie efektywności modelu  loss\_per\_epochs = model.history.history['loss']  plt.plot(range(len(loss\_per\_epochs)), loss\_per\_epochs) |



Rysunek 2. Wykres funkcji straty dla szeregu stacjonarnego.

***Predykcja na danych testowych szeregu stacjonarnego przy użyciu wytrenowanego modelu LSTM***

W tym etapie przeprowadzana jest predykcja na danych testowych przy użyciu wytrenowanego modelu LSTM. Proces ten polega na generowaniu przewidywań krok po kroku w oparciu o wcześniejsze prognozy. Najpierw pobierana jest ostatnia sekwencja danych testowych (o długości n\_input) jako **first\_eval\_batch**, która stanowi początkową partię danych wejściowych. Następnie dane te są przekształcane w odpowiedni kształt trójwymiarowy **(1, n\_input, n\_features)**, który jest wymagany przez model LSTM.

W pętli **for**, która iteruje przez całą długość zbioru testowego (len(test\_rate)), model generuje przewidywania dla bieżącej sekwencji za pomocą metody **predict()**. Wartość prognozowana **current\_pred** zostaje zapisana na liście **test\_predictions**, która przechowuje wszystkie przewidywane wartości. Następnie sekwencja wejściowa **current\_batch** jest aktualizowana: pierwsza wartość (najstarsza) zostaje usunięta, a na jej miejsce dodawana jest nowa przewidywana wartość. Aktualizacja jest realizowana za pomocą funkcji **np.append**, która łączy bieżącą sekwencję z nową wartością wzdłuż odpowiedniej osi, zapewniając utrzymanie wymaganego kształtu danych.

W rezultacie model używa własnych prognoz jako danych wejściowych do przewidywania kolejnych kroków. Dzięki temu proces jest autoregresyjny – każda nowa prognoza jest oparta na wcześniejszych prognozach, a nie tylko na danych rzeczywistych. Takie podejście pozwala na symulację przyszłych wartości szeregu czasowego, ale może prowadzić do skumulowania błędów, jeśli prognozy są mniej dokładne.

|  |
| --- |
| test\_predictions = []  first\_eval\_batch = scaled\_test\_rate[-n\_input:]  current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))  for i in range(len(test\_rate)):      current\_pred = model.predict(current\_batch)[0] # wartość przewidywana dla pierwszej serii      test\_predictions.append(current\_pred) # dodaje wartość przewidywaną do serii      current\_batch = np.append(current\_batch[:,1:,:], [[current\_pred]], axis = 1) #uaktualniam serie i usuwam pierwszą wartość |

Po tych działaniach predykcje dla danych testowych wyglądają następująco.

|  |
| --- |
| [array([0.7650406], dtype=float32),  array([0.73754275], dtype=float32),  array([0.646534], dtype=float32),  array([0.58121693], dtype=float32),  array([0.6436824], dtype=float32),  array([0.651811], dtype=float32),  array([0.6896413], dtype=float32),  array([0.6915552], dtype=float32),  array([0.67072856], dtype=float32),  array([0.6268702], dtype=float32),  array([0.5687128], dtype=float32),  array([0.5714391], dtype=float32),  array([0.5759404], dtype=float32),  array([0.59799635], dtype=float32),  array([0.60568196], dtype=float32),  array([0.5945229], dtype=float32),  array([0.5791113], dtype=float32),  array([0.5570304], dtype=float32),  array([0.5465989], dtype=float32),  array([0.54750854], dtype=float32),  array([0.55530447], dtype=float32),  array([0.5604594], dtype=float32),  array([0.5559184], dtype=float32),  array([0.5475482], dtype=float32),  array([0.53620803], dtype=float32)] |

Ponieważ dane są przeskalowane, nie pasowałyby do ostatecznych wyników, dlatego musimy ponownie przeskalować dane to skali pierwotnej.

|  |  |  |
| --- | --- | --- |
| DATE | REAL\_INTEREST\_RATE | PREDICTIONS |
| 2022-01-01 | -3.265321 | 2.924620 |
| 2022-02-01 | -4.126292 | 2.129006 |
| 2022-03-01 | -4.816444 | 1.653075 |
| 2022-04-01 | -5.598164 | 1.235324 |
| 2022-05-01 | -5.857062 | 1.069865 |
| 2022-06-01 | -7.102906 | 1.112894 |
| 2022-07-01 | -3.553523 | 1.251459 |
| 2022-08-01 | -3.056842 | 1.420282 |
| 2022-09-01 | -5.056445 | 1.554928 |
| 2022-10-01 | -0.366113 | 1.596043 |
| 2022-11-01 | -0.450654 | 1.520387 |
| 2022-12-01 | 0.835077 | 1.352219 |
| 2023-01-01 | 1.817899 | 1.183102 |
| 2023-02-01 | 1.982413 | 1.082244 |
| 2023-03-01 | 4.151604 | 1.063427 |
| 2023-04-01 | 1.845097 | 1.108981 |
| 2023-05-01 | 1.607673 | 1.187557 |
| 2023-06-01 | 6.004632 | 1.265255 |
| 2023-07-01 | 2.854389 | 1.312419 |
| 2023-08-01 | 3.100313 | 1.312224 |
| 2023-09-01 | 2.492336 | 1.268672 |
| 2023-10-01 | 3.005341 | 1.205080 |
| 2023-11-01 | 2.907568 | 1.150504 |
| 2023-12-01 | 1.538694 | 1.124136 |
| 2024-01-01 | 3.511612 | 1.129304 |

Porównując dane przewidywane do danych rzeczywistych możemy przeanalizować predykcje, które otrzymaliśmy.

Obraz zawierający diagram, tekst, Wykres, linia

Opis wygenerowany automatycznie

Rysunek 3. Wykres predykcji i wartości rzeczywistych dla szeregu stacjonarnego.

***Analiza wyników predykcji szeregu stacjonarnego***

Do analizy skali błędnych danych wykorzystuję RSME (*ang rooot mean squared erro*), czy pierwiastek błędu średniokwadratowego. Błąd w przybliżeniu wynosi 3, co oznacza, że jest on średni. Może to oznaczać, że ten typ szeregu stacjonarnego z cyklicznościami, poprzez brak paternów, jest dla modelu LSTM trudniejszy do predykcji, jednak prognozowane wyniki nie są wynikami najgorszymi.

Należy pamiętać, aby nie dać zbyt dużej liczby epok gdyż wtedy nasz model może się „przeuczyć” i podawać błędne wyniki. Widać to wyraźnie na wykresie straty, gdzie kolejne iteracje nie zbliżają do celu, a czasem wręcz odwortnie.

Obraz zawierający tekst, Wykres, zrzut ekranu, linia

Opis wygenerowany automatycznie

Rysunek 7. Wykres funkcji straty dla modelu przeuczonego.

***Automatyzacja predykcji***

Analogicznie zostały przeanalizowane pozostały dwa szeregi. Dla zautomatyzowania pracy najpierw tworze trzy modele

|  |
| --- |
| model\_rate = modelLSTM(n\_input, n\_features)  model\_cpi = modelLSTM(n\_input, n\_features)  model\_mat = modelLSTM(n\_input, n\_features) |

Następnie wszystkie kroki, które opisałam powyżej wykonuje w funkcji.

|  |
| --- |
| def gen\_pred(model, train\_data, test\_data, n\_input, n\_features, n\_epochs):        scaler = MinMaxScaler(feature\_range=(-1, 1))      # Skalowanie danych (zakładając, że skalujemy tylko kolumnę z rzeczywistymi wartościami)      scaled\_train\_data = scaler.fit\_transform(train\_data.iloc[:, [0]])      scaled\_test\_data = scaler.transform(test\_data.iloc[:, [0]])      # Generator dla 12 miesięcy      generator = TimeseriesGenerator(scaled\_train\_data, scaled\_train\_data, length=n\_input, batch\_size=1)      # Trenowanie modelu i zbieranie historii      history = model.fit(generator, epochs=n\_epochs)  # liczba epok      # Generowanie predykcji      test\_predictions = []      first\_eval\_batch = scaled\_test\_data[-n\_input:]      current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))      for i in range(len(scaled\_test\_data)):          current\_pred = model.predict(current\_batch)[0]  # wartość przewidywana dla pierwszej serii          test\_predictions.append(current\_pred)  # dodaje wartość przewidywaną do serii          current\_batch = np.append(current\_batch[:, 1:, :], [[current\_pred]], axis=1)  # aktualizacja serii        # Odwrócenie skalowania predykcji      true\_predictions = scaler.inverse\_transform(test\_predictions)      # Zwracanie predykcji i funkcji straty      return true\_predictions, history.history['loss'] |

W kolejnym kroku generuje predykcje za pomocą wywołania funkcji.

|  |
| --- |
| true\_pred\_rate = gen\_pred(model\_rate, train\_rate, test\_rate, n\_input, n\_features, 75)  true\_pred\_rate = gen\_pred(model\_cpi, train\_cpi, test\_cpi, n\_input, n\_features, 75)  true\_pred\_rate = gen\_pred(model\_mat, train\_mat, test\_mat, n\_input, n\_features, 75) |

***Predykcje dla szeregu z trendem***

Po wytrenowaniu modelu, jego efektywność w procesie uczenia można obserwować na wykresie funkcji straty, co pozwala na ocenę, jak szybko model zbiega do optymalnego rozwiązania.Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 4. Wykres funkcji straty dla szeregu z trendem.

W analizie widoczne jest, że już po 100 epokach model osiąga stabilizację, co sugeruje, że proces uczenia skutecznie minimalizuje błąd. Liczba epok ma istotny wpływ na zachowanie predykcji. Przy mniejszej liczbie epok, na przykład 75, przewidywania modelu będą znajdować się w pobliżu krzywej regresji szeregu (?), przy czym ich charakter może przypominać błądzenie losowe (?). Oznacza to, że model jest w stanie wychwycić główny trend danych, jednak wciąż może występować pewna niestabilność w indywidualnych predykcjach.Obraz zawierający linia, Wykres, diagram, stok

Opis wygenerowany automatycznie

Rysunek 5. Wykres predykcji i wartości rzeczywistych dla szeregu z trendem dla 75 epok.

Z kolei dla większej liczby epok, jak 100, model nie tylko zachowuje wykryty trend, ale jego predykcje mogą przesuwać się względem osi strat. Jest to efekt dalszego dopasowywania modelu do danych, które minimalizuje błąd, ale może również prowadzić do nadmiernego skupienia się na konkretnych wzorcach w danych. Warto więc uwzględnić odpowiedni balans między liczbą epok a zachowaniem modelu, aby uniknąć potencjalnych przesunięć i zapewnić, że model wiernie odwzorowuje zarówno trend, jak i zmienność w szeregach czasowych.

Obraz zawierający Wykres, diagram, linia, tekst

Opis wygenerowany automatycznie

Rysunek 6 Wykres predykcji i wartości rzeczywistych dla szeregu z trendem dla 100 epok.

***Analiza wyników predykcji szeregu z trendem***

Do oceny skali błędów predykcji wykorzystano miarę RMSE (ang. *Root Mean Squared Error*), czyli pierwiastek błędu średniokwadratowego. Uzyskana wartość RMSE wynosi około 35, co oznacza, że błąd modelu jest stosunkowo niewielki w porównaniu do skali danych, na których operujemy. Taka wartość błędu wskazuje na zadowalającą dokładność modelu, umożliwiającą jego praktyczne wykorzystanie w przewidywaniu przyszłych wartości szeregu czasowego.

Ciekawym aspektem jest to, że model próbował odnaleźć sezonowość w danych, mimo że jest ona praktycznie nieistniejąca. Może to wynikać z kilku czynników, takich jak błędy przybliżeń wprowadzone podczas wcześniejszego skalowania danych lub naturalna skłonność modelu do poszukiwania wzorców, nawet jeśli są one znikome bądź przypadkowe. W początkowych etapach uczenia model podejmował próby identyfikacji cykliczności w danych, jednak wraz ze wzrostem liczby epok stopniowo odchodził od szukania sezonowości na rzecz dokładniejszego odwzorowania samego trendu. W miarę postępu uczenia model coraz bardziej dostrajał się do głównego trendu danych, co świadczy o jego zdolności do adaptacji i optymalizacji wzorców predykcji.

W związku z tym kluczowe jest odpowiednie dobranie liczby epok — tak, aby umożliwić modelowi nauczenie się głównych wzorców, jednocześnie unikając nadmiernego dopasowania. Monitorowanie procesu uczenia poprzez analizę wykresu funkcji straty oraz porównanie błędów na zbiorze treningowym i testowym pomaga znaleźć optymalny punkt zatrzymania treningu modelu.

***Predykcje dla szeregu z sezonowością***

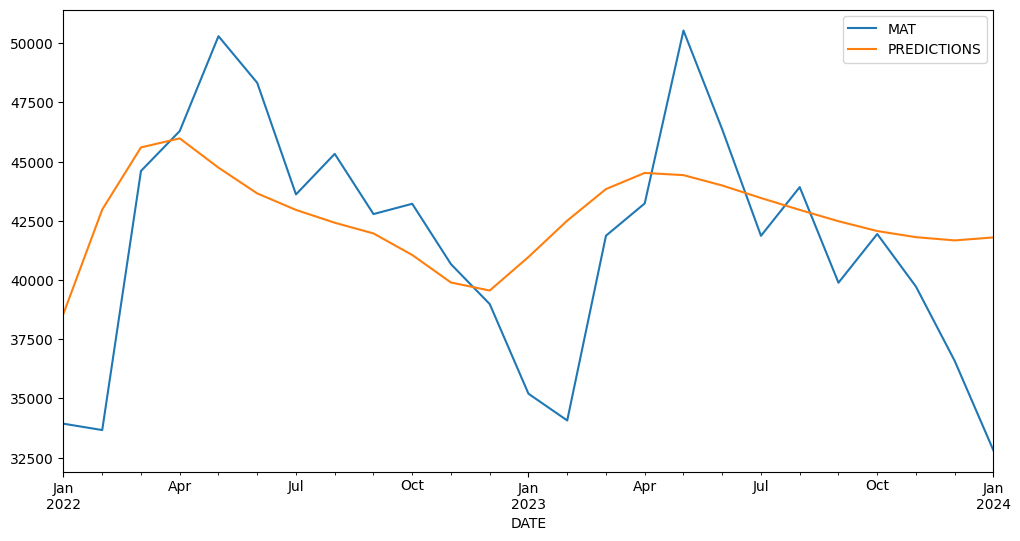
Dla modelu prognozującego szereg czasowy z sezonowością optymalna liczba epok wynosi około 500. Na wykresie funkcji straty można zauważyć, że po osiągnięciu tej liczby funkcja ta zaczyna się stabilizować, zbliżając się do wartości granicznych w pobliżu zera. Stabilizacja ta wskazuje, że model nauczył się odwzorowywać wzorce sezonowe zawarte w danych, minimalizując błąd predykcji.

Obraz zawierający tekst, Wykres, linia, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek 8. Wykres funkcji straty dla szeregu z sezonowością.

Podczas analizy wyników predykcji widoczne jest, że model skutecznie identyfikuje i odtwarza sezonowość w szeregach czasowych. Na wykresie wartości przewidywanych sezonowe zmiany są wyraźnie zaznaczone i pojawiają się w miesiącach, w których rzeczywiście występują w danych rzeczywistych. Oznacza to, że model potrafi odróżnić regularne wahania sezonowe od pozostałych komponentów szeregu, co jest kluczowe w tego typu prognozach.



Rysunek 9. Wykres predykcji i wartości rzeczywistych dla szeregu z sezonowością.

Wyniki te potwierdzają, że model dobrze radzi sobie z uchwyceniem sezonowego charakteru danych, co czyni go użytecznym narzędziem do prognozowania szeregów czasowych o wyraźnych cechach. Dalsze zwiększanie liczby epok może prowadzić do zjawiska nadmiernego dopasowania, dlatego zaleca się zatrzymanie procesu uczenia w momencie stabilizacji funkcji straty.

***Analiza wyników predykcji szeregu z sezonowością***

Wyniki analizy pokazują, że model skutecznie identyfikuje i odwzorowuje sezonowość w danych. Na wykresie wartości przewidywanych sezonowe zmiany są wyraźnie zaznaczone, pojawiając się w miesiącach, w których rzeczywiście występują w danych rzeczywistych. Model odtwarza te cykliczne wahania z dużą dokładnością, co świadczy o jego zdolności do rozpoznawania i przewidywania istotnych regularnych zmian w szeregach czasowych.

Dodatkowo, wskaźnik RMSE (Root Mean Squared Error) dla tego modelu wynosi około 6100. Jest to bardzo dobry wynik, biorąc pod uwagę skalę, w której operują dane. Taka wartość błędu wskazuje na wystarczającą jakość predykcji, co czyni model dobrze dopasowanym do analizy i prognozowania tego typu szeregów czasowych.

Podsumowując, model wykazuje wysoką skuteczność w przewidywaniu danych z sezonowością, co czyni go solidnym narzędziem w zastosowaniach praktycznych. Jednocześnie należy pamiętać o unikaniu zjawiska nadmiernego dopasowania, które mogłoby wystąpić przy dalszym zwiększaniu liczby epok powyżej punktu stabilizacji funkcji straty.

## Porównanie

|  |  |
| --- | --- |
| POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza | Rzeszów, Rok |
| Wydział Elektrotechniki i Informatyki |  |
|  |  |

**STRESZCZENIE (wybierz rodzaj pracy)**

**TYTUŁ PRACY**

Autor: Imię Nazwisko, nr albumu: (wybierz symbol studiów)-123456 Opiekun: (tytuł naukowy przed) Imię Nazwisko(tytuł naukowy po)

Słowa kluczowe: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

(tekst streszczenia - max. 10 wierszy)

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, Rok

Faculty of Electrical and Computer Engineering

**DIPLOMA THESIS (wybierz rodzaj pracy) ABSTRACT**

**TYTUŁ PRACY W WERSJI ANGIELSKIEJ**

Author: Imię Nazwisko, code: (wybierz symbol studiów) -123456 Supervisor: (tytuł naukowy przed) Imię Nazwisko (tytuł naukowy po)

Key words: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

(tekst streszczenia w jęz. angielskim - max. 10 wierszy)