

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.

***Walidacja***

Walidacja w kontekście uczenia maszynowego to proces oceny wydajności modelu na zbiorze danych, który nie był używany podczas jego trenowania. Celem walidacji jest sprawdzenie, jak dobrze model generalizuje na nowych, niewidzianych wcześniej danych. Jest to kluczowy etap, ponieważ pomaga w identyfikacji problemów, takich jak przeuczenie (overfitting), gdzie model może doskonale dopasować się do danych treningowych, ale słabo radzić sobie z danymi testowymi lub walidacyjnymi.

Zwykle proces walidacji polega na podziale dostępnych danych na dwie części: jedną do trenowania modelu, a drugą do walidacji (oceny wydajności modelu). W trakcie trenowania, model jest regularnie oceniany na danych walidacyjnych, co pozwala monitorować jego zdolność do generalizowania oraz dostosowywać hiperparametry, aby poprawić jego ogólną wydajność.

W praktyce, walidacja jest istotnym krokiem w budowie modelu, który pomaga upewnić się, że model nie tylko "uczy się" danych treningowych, ale także potrafi skutecznie przewidywać na nowych, nieznanych danych.

Strata treningowa i strata walidacyjna to kluczowe miary oceny jakości modelu uczenia maszynowego. **Strata treningowa** mierzy błąd, który model popełnia podczas dopasowywania się do danych treningowych, wskazując, jak dobrze model "uczy się" na podstawie tych danych. Celem trenowania jest minimalizowanie tej straty, aby model jak najlepiej dopasował się do danych treningowych. **Strata walidacyjna** natomiast mierzy błąd modelu na danych walidacyjnych, które nie były używane w trakcie trenowania. Pokazuje to, jak dobrze model generalizuje i jak skutecznie potrafi przewidywać na nowych, niewidzianych wcześniej danych. Strata treningowa zazwyczaj maleje z każdą epoką treningową, ponieważ model stopniowo dopasowuje się do danych treningowych. Natomiast strata walidacyjna może początkowo maleć, ale jeśli model zaczyna przeuczać się na danych treningowych (staje się zbyt dopasowany do tych danych), strata walidacyjna może zacząć rosnąć, podczas gdy strata treningowa nadal będzie maleć. Monitorowanie obu tych miar jest istotne, aby upewnić się, że model nie przeucza się i potrafi skutecznie przewidywać na nowych danych.

***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.

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**.

import numpy as np # algebra

import pandas as pd # data processing, zarządzanie plikami, ramki danych

import matplotlib as mpl # wykresy

import matplotlib.pyplot as plt # wizualizacja danych

import matplotlib.dates as mdates # wizualizacja danych – dat

from statsmodels.tsa.seasonal import seasonal\_decompose # dekompozycja szeregów

3.1 Analiza szeregów czasowych – opis, wykresy, zakres zmienności

***Szereg stacjonarny***

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:

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

df\_rate = pd.DataFrame(data\_rate)

df\_rate.columns.values[0] = 'REAL\_INTEREST\_RATE'

df\_rate.index.freq = 'MS' # zaznaczam, że dane są miesięczne (zazwyczaj powinno się i tak ustawić automatycznie)

print(df\_rate.head(10))

Za pomocą analogicznego kodu zaimportowano także pozostałe dwa szeregi czasowe.

**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.**

Można zauważyć, że dane charakteryzuje brak trendu, czy sezonowości. Widoczna jest natomiast cykliczność danych.

**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.**

Po zastosowaniu dekompozycji szeregu otrzymujemy

**Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**

1. Obraz zawierający tekst, linia, Czcionka, Wykres

   Opis wygenerowany automatycznie
2. **Rysunek 3.2 Proszę dodać tytuł rysunku.**

**Proszę podać wnioski z dekompozycji (również dotyczące sezonowości i reszt).**

***Szereg z trendem***

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

**(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.**

Widzimy wyraźnie zaznaczający się trend. Dekompozycja szeregu (rysunek 3.4) potwierdza istnienie trendu w analizowanym szeregu czasowym.

**Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**

Obraz zawierający tekst, linia, Czcionka, Wykres

Opis wygenerowany automatycznie

1. **Rysunek 3.4 Proszę dodać tytuł rysunku.**

**Proszę podać wnioski wynikające z dekompozycji szeregu (również dotyczące sezonowości i reszt).**

***Szereg z trendem i sezonowością***

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.

**(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.**

Obraz zawierający tekst, linia, pismo odręczne, Wykres

Opis wygenerowany automatycznie**Rysunek 3.5 Proszę dodać tytuł rysunku.**

1. Widać wyraźnie zaznaczoną sezonowość, a także delikatny trend na przestrzeni 10-ciu lat, przyśpieszający na przestrzeni ostatnich 5-ciu lat.
2. **Proszę podać fragment kodu, za pomocą którego przeprowadziła Pani dekompozycję.**
3. Obraz zawierający tekst, linia, Czcionka, Wykres

   Opis wygenerowany automatycznie

**Rysunek 3.6 Proszę dodać tytuł rysunku.**

1. Dekompozycja szeregu (rysunek 3.6) wskazuje na ….
2. **Proszę podać wnioski wynikające z dekompozycji szeregu (również dotyczące sezonowości i reszt).**
3. Dodać informacje o obszarze zmienności

***Testy stacjonarności***

W celu sprawdzenia stacjonarności szeregu, w pierwszej kolejności przeprowadza się testy stacjonarności. Jest to kluczowe do stosowania matod klasycznych analizy szeregów czasowych. 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. Nie trzeba ręcznie różnicować danych przed użyciem auto\_arima, ponieważ model ten zautomatyzuje proces wyboru najlepszego poziomu różnicowania. Jednak jeśli masz już przetworzone dane, takie jak dane stacjonarne, możesz wprowadzić je bez dalszego różnicowania.

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

W przypadku dopasowywania modelu ARIMA do danych pierwszego szeregu, najlepszym modelem okazał się ARIMA(1,0,1) z wyrazem wolnym, osiągając wartość AIC równą 429.732, co czyni go najskuteczniejszym wyborem spośród testowanych modeli. Model ten uwzględniał jeden składnik autoregresyjny (p=1) oraz jeden składnik średniej ruchomej (q=1), z wyrazem wolnym. Sugeruje to, że dane mają pewne zależności autoregresyjne oraz że obecność opóźnionych błędów w modelu jest istotna, co najlepiej tłumaczy zmienność w danych.

Chociaż testowano również inne modele, takie jak ARIMA(0,0,1) czy ARIMA(2,0,1), ich AIC były wyższe, co wskazuje, że dodanie dodatkowych parametrów lub zmian w strukturze modelu nie poprawiło znacząco dopasowania. Modele o wyższych wartościach AIC, takie jak ARIMA(2,0,2), uzyskały wartość AIC wynoszącą "inf", co oznacza, że nie były w stanie dopasować się do danych, co może sugerować problemy z konwergencją lub nadmierną złożonością modelu.

Wyniki sugerują, że dla tych danych model ARIMA(1,0,1) był najbardziej odpowiedni, oferując dobre dopasowanie przy jednoczesnym zachowaniu prostoty i efektywności w modelowaniu zmienności danych.

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

Opis wygenerowany automatycznie

W przypadku dopasowywania modelu ARIMA do danych drugiego szeregu czasowego, najlepszym modelem okazał się ARIMA(0,2,2) bez wyrazu wolnego, osiągając wartość AIC równą 229.137, co czyni go najskuteczniejszym wyborem spośród testowanych modeli. Model ten uwzględniał dwa składniki w modelu średniej ruchomej (q=2) oraz dwa różnicowania (d=2), co sugeruje, że dane wymagają podwójnego różnicowania, aby uzyskać stacjonarność, a także uwzględnienia opóźnionych błędów w modelowaniu zmienności.

Testowano również inne modele, takie jak ARIMA(1,2,2), ARIMA(0,2,1) oraz ARIMA(0,2,3), ale ich AIC były wyższe, co oznacza, że dodanie dodatkowych parametrów lub zmian w strukturze modelu nie poprawiło znacząco dopasowania. Model ARIMA(2,2,2) również uzyskał dość dobrą wartość AIC (231.555), ale nie był lepszy od ARIMA(0,2,2), który osiągnął najniższy AIC spośród wszystkich testowanych modeli.

Wyniki sugerują, że dla tych danych model ARIMA(0,2,2) był najbardziej odpowiedni, oferując prostą, ale efektywną strukturę z minimalną wartością AIC, co wskazuje na odpowiednie dopasowanie do danych.

Obraz zawierający tekst, Czcionka, zrzut ekranu, dokument

Opis wygenerowany automatycznie

W przypadku dopasowywania modelu ARIMA do danych trzeciego szeregu czasowego, najlepszym modelem okazał się ARIMA(2,1,2) bez wyrazu wolnego, osiągając wartość AIC równą 2277.976, co czyni go najskuteczniejszym wyborem spośród testowanych modeli. Model ten uwzględniał dwa składniki autoregresyjne (p=2), jedno różnicowanie (d=1) oraz dwa składniki średniej ruchomej (q=2). Sugeruje to, że dane wymagają zarówno zależności autoregresyjnych, jak i uwzględnienia opóźnionych błędów w modelu, co pozwala na uchwycenie bardziej skomplikowanej struktury czasowej.

Wyniki sugerują, że dla tych danych model ARIMA(2,1,2) był najbardziej odpowiedni, oferując dobre dopasowanie przy jednoczesnym zachowaniu prostoty i efektywności. Proces dopasowywania modelu trwał 0.360 sekundy, co świadczy o jego wydajności.

Obraz zawierający tekst, Czcionka, zrzut ekranu, papier

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

Metodą klasyczną, którą będę wykorzystywała w tej pracy będzie model ARIMA, opisany już wcześniej. Wszystkie trzy szeregi będą przewidywane tą metodą. Dobór modelu oraz testy stacjonarności zostały już przeprowadzone w wcześniejszym rozdziale. Dokładne działanie kodu krok po kroku opiszę dla szeregu jednomiesięcznej rzeczywistej stopy procentowej. Dla pozostałych szeregów przedstawię tylko wyniki, gdyż działanie kodu będzie analogiczne.

***Tworzenie modelu dla szeregu pierwszego***

Do stworzenia modelu wykorzystuje funkcje ARIMA z biblioteki statsmodels. Podaje jej dane treningowe pierwszego szeregu oraz parametry, które dobrałam wcześniej.

|  |
| --- |
| from statsmodels.tsa.arima.model import ARIMA  model\_arima\_rate=ARIMA(train\_rate, order=(1,0,1))  model\_arima\_rate = model\_arima\_rate.fit()  model\_arima\_rate.summary() |

Następnie model jest dopasowywany do danych treningowych za pomocą metody .fit(), która optymalizuje parametry modelu, aby jak najlepiej odwzorowywały dane.

***Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie***

Tak przygotowany model prezentuje się w ten sposób.

***Prognozowanie dla szeregu pierwszego***

W tym kroku generowane są prognozy na danych testowych za pomocą modelu ARIMA. Najpierw prognozy dla zbioru testowego są obliczane, a następnie obliczane są przedziały ufności dla tych prognoz, ustawiając poziom ufności na 95%.. Ostatecznie, wyodrębniane są prognozy, które są gotowe do dalszej analizy i porównania z rzeczywistymi danymi.

|  |
| --- |
| rate\_pred = model\_arima\_rate.get\_forecast(len(test\_rate.index))  rate\_pred\_df = rate\_pred.conf\_int(alpha = 0.05)  rate\_pred\_df["Predictions"] = model\_arima\_rate.predict(start = rate\_pred\_df.index[0], end = rate\_pred\_df.index[-1])  rate\_pred\_df.index = test\_rate.index  rate\_pred\_out = rate\_pred\_df["Predictions"]  rate\_pred\_out |

Wyniki predykcji wyglądaj następująco.

Obraz zawierający tekst, Czcionka, czarne i białe, dokument

Opis wygenerowany automatycznie

Zgodnie z wynikami można zauważyć na wykresie, że model przewidział w pewnym zakresie wzrost danych testowych. Jednak poprzez cykliczno-szumowy charakter szeregu pierwszego

Obraz zawierający tekst, Wykres, diagram, linia

Opis wygenerowany automatycznie

Jednak poprzez cykliczno-szumowy charakter szeregu pierwszego model w dalszym przedziale czasowym szuka średniej i do niej dąży, gdyż nie jest w stanie uchwycić żadnych wzorców.

Warto zauważyć, że przedział ufności większość z danych testowych dobrze uchwycił. Dane testowe znajdują się w przedziale ufności, czyli model trafnie przewidział gdzie kolejne dane mogą się znajdować. Można również zauważyć, że przewidział ufności się nie rozszerza, co może oznaczać, że model nie przewiduje żadnego trendu. Prognoza nie wskazuje na to, że wartość stopy procentowej będzie rosła lub malała w większym zakresie.

***Tworzenie modelu dla szeregu drugiego***

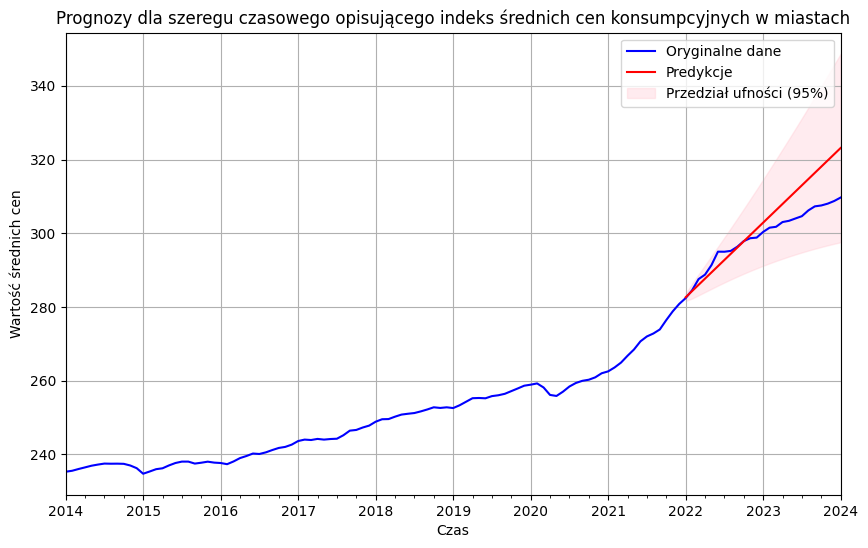
Analogicznie dla szeregu pierwszego tworzymy model ARIMA dla szeregu drugiego.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Opis najważniejszych wartości z prof. Rybarską-Rusinek

***Prognozowanie dla szeregu drugiego***



Model przewidział występujący trend w danych. Ponieważ dane nie są jednolite i występują w nich różne fluktuacje trend ten nigdy nie będzie przewidziany idealnie, jednak prognoza jest dokonana w sposób wystarczający.

Dane testowe w pełni znajdują się w obrębie obszaru zmienności, co oznacza, że predykcje są prawidłowe.

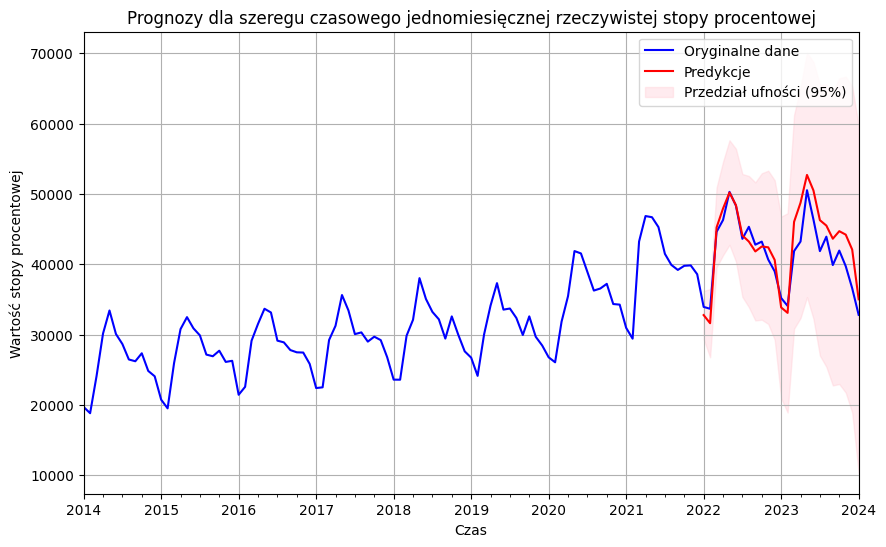
***Tworzenie modelu dla szeregu trzeciego***

Obraz zawierający tekst, menu, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Opis najważniejszych wartości z prof. Rybarską-Rusinek

***Prognozowanie dla szeregu trzeciego***



Model przewidział występujący trend i sezonowość w danych. Ponieważ dane nie są jednolite i występują w nich różne fluktuacje trend ten nigdy nie będzie przewidziany idealnie, jednak prognoza jest dokonana w sposób bardzo dobry.

Dane testowe w pełni znajdują się w obrębie obszaru zmienności, co oznacza, że predykcje są prawidłowe.

Obszar zmienności również zachowuje się sezonowo i rozszerza się co oznacza że model zauważa i tren i sezonowość. Model przewidział, że przyszłe dane mogą się przesunąć względem wcześniejszych. Model dokonał wystarczająco precyzyjnych prognoz.

## 4.2 Prognozowanie metodą LSTM

Można również podejść do prognozowania za pomocą metod uczenia maszynowego. Modelem, którego użyję będzie opisany wcześniej LSTM. Dokładne działanie kodu krok po kroku opiszę dla szeregu jednomiesięcznej rzeczywistej stopy procentowej. Dla pozostałych szeregów przedstawię tylko wyniki, gdyż działanie kodu będzie analogiczne.

***Podział na dane treningowe i testowe***

Z racji, że dane są z przedziału 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 96 miesięcy przeznaczone są na część testową, a pozostałe na część treningową.

|  |
| --- |
| # Podział danych na zbiór treningowy i testowy  train\_size\_rate = 96  # Pierwsze 96 miesięcy to zbiór treningowy  train\_data\_rate = scaled\_data\_rate[:train\_size\_rate]  test\_data\_rate = scaled\_data\_rate[train\_size\_rate - sequence\_length:] |

***Skalowanie danych***

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

|  |
| --- |
| # Normalizacja danych  scaler\_rate = MinMaxScaler(feature\_range=(0, 1))  scaled\_data\_rate = scaler\_rate.fit\_transform(df\_rate) |

Z racji na szczególność tego kroku przedstawiam dane dla trzech szeregów.

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 |

***Sekwencje dla LSTM***

Funkcja create\_sequences służy do przygotowania danych wejściowych i wyjściowych do modelu LSTM, który jest wykorzystywany w analizie szeregów czasowych. Funkcja ta dzieli dane na sekwencje o zadanej długości (seq\_length), które będą używane do trenowania modelu. Dla każdej sekwencji wejściowej (tworzonej na podstawie kolejnych punktów danych) funkcja przypisuje wartość wyjściową, która jest punktem czasowym bezpośrednio następującym po tej sekwencji. W ten sposób tworzony jest zbiór danych, w którym model LSTM będzie mógł uczyć się zależności czasowych i na tej podstawie prognozować przyszłe wartości. Parametr sequence\_length określa, ile punktów danych ma zawierać każda sekwencja wejściowa. Dzięki tej funkcji, dane są odpowiednio przygotowane do trenowania modelu LSTM, który analizuje wzorce w danych szeregów czasowych.

|  |
| --- |
| # Funkcja tworząca sekwencje dla LSTM  def create\_sequences(data, seq\_length):      X, y = [], []      for i in range(len(data) - seq\_length):          X.append(data[i:i+seq\_length, 0])          y.append(data[i+seq\_length, 0])      return np.array(X), np.array(y)  # Parametry  sequence\_length = 12  # Długość sekwencji wejściowych  # Generowanie sekwencji  X\_train\_rate, y\_train\_rate = create\_sequences(train\_data\_rate, sequence\_length)  X\_test\_rate, y\_test\_rate = create\_sequences(test\_data\_rate, sequence\_length)  # Zmiana kształtu danych dla LSTM: (samples, timesteps, features)  X\_train\_rate = X\_train\_rate.reshape((X\_train\_rate.shape[0], X\_train\_rate.shape[1], 1))  X\_test\_rate = X\_test\_rate.reshape((X\_test\_rate.shape[0], X\_test\_rate.shape[1], 1)) |

Ten fragment kodu służy do przygotowania danych do trenowania modelu LSTM, który jest wykorzystywany w analizie szeregów czasowych. W pierwszej linii ustawiany jest parametr sequence\_length na 12, co oznacza, że każda sekwencja wejściowa będzie składała się z 12 punktów czasowych (np. 12 miesięcy). Następnie, za pomocą funkcji create\_sequences, generowane są sekwencje dla zbiorów treningowego i testowego. Funkcja ta tworzy dane wejściowe (X\_train\_rate, X\_test\_rate) oraz odpowiadające im wartości wyjściowe (y\_train\_rate, y\_test\_rate) na podstawie zadanego rozmiaru sekwencji. Każda sekwencja wejściowa składa się z 12 punktów czasowych, a odpowiadająca jej wartość wyjściowa to wartość w kolejnym punkcie czasowym. Ostatecznie, dane wejściowe (X\_train\_rate, X\_test\_rate) są przekształcane do odpowiedniego kształtu, który jest wymagany przez model LSTM, czyli do formatu (samples, timesteps, features), gdzie samples to liczba próbek, timesteps to długość sekwencji, a features to liczba cech (w tym przypadku 1, ponieważ mamy jedną zmienną wejściową).

***Definiowanie modelu LSTM***

Kojelnym krokiem jest zdefiniowanie modelu sieci neuronowej LSTM (Long Short-Term Memory) przy użyciu klasy Sequential z biblioteki Keras. Model jest używany do przewidywania wartości na podstawie szeregów czasowych. Pierwsza linia model\_rate = Sequential() inicjalizuje model sekwencyjny, co oznacza, że warstwy są dodawane jedna po drugiej. Następnie, za pomocą model\_rate.add(LSTM(100, activation='tanh', return\_sequences=False, input\_shape=(sequence\_length, 1))), dodana jest warstwa LSTM z 100 jednostkami (neurony), które analizują dane wejściowe, a funkcja aktywacji tanh wprowadza nieliniowość. Parametr return\_sequences=False oznacza, że model zwróci tylko ostatnią wartość z sekwencji, co jest typowe w zadaniach regresyjnych. input\_shape=(sequence\_length, 1) wskazuje, że wejście ma postać sekwencji o długości sequence\_length i jednej cesze (np. stopa procentowa). Kolejna warstwa, model\_rate.add(Dense(1)), to warstwa wyjściowa, która składa się z pojedynczego neuronu, co oznacza, że model ma przewidywać jedną wartość na podstawie przetworzonych danych. Na końcu, model jest kompilowany za pomocą optymalizatora adam oraz funkcji straty mse (średni błąd kwadratowy), co pozwala na efektywne trenowanie modelu w zadaniu regresji.

|  |
| --- |
| # Budowa modelu LSTM  model\_rate = Sequential()  model\_rate.add(LSTM(100, activation='tanh', return\_sequences=False, input\_shape=(sequence\_length, 1)))  model\_rate.add(Dense(1))  # Warstwa wyjściowa  model\_rate.compile(optimizer='adam', loss='mse')  model\_rate.summary() |

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

***Trenowanie modelu***

Na tym etapie kodu model LSTM jest trenowany na danych treningowych, aby nauczyć się wzorców i zależności w szeregach czasowych. Funkcja fit jest używana do przeprowadzenia procesu treningowego, który trwa przez określoną liczbę epok. W tym przypadku model będzie trenowany przez 250 epok, co oznacza, że dane treningowe będą wielokrotnie przetwarzane, a wagi modelu będą dostosowywane w celu minimalizacji funkcji straty.

W trakcie treningu model aktualizuje swoje wagi po każdej partii danych (mini-batch) o rozmiarze 12 próbek, co pomaga w efektywnym uczeniu się. Dodatkowo, dane walidacyjne (zbiór testowy) są używane do oceny, jak dobrze model generalizuje, czyli jak dobrze przewiduje wyniki na nowych, niewidzianych wcześniej danych. Dzięki temu możliwe jest monitorowanie zarówno postępów w nauce na danych treningowych, jak i oceny modelu na danych, które nie były używane do jego trenowania. Wartości funkcji straty są wyświetlane po każdej epoce, co umożliwia śledzenie procesu uczenia.

|  |
| --- |
| # Trenowanie modelu  history\_rate = model\_rate.fit(X\_train\_rate, y\_train\_rate,                                epochs=250,                                batch\_size=12,                                validation\_data=(X\_test\_rate, y\_test\_rate),                                verbose=1) |

***Tworzenie prognozy***

Następnie wykonywana jest prognoza za pomocą modelu model\_rate na dwóch różnych zbiorach danych: treningowym (X\_train\_rate) i testowym (X\_test\_rate). Pierwsza linia kodu, train\_predict\_rate = model\_rate.predict(X\_train\_rate), generuje prognozy na danych treningowych, na których model był wcześniej uczony. Prognozy te odzwierciedlają, jak dobrze model dopasował się do danych, na których był trenowany. Druga linia, test\_predict\_rate = model\_rate.predict(X\_test\_rate), wykonuje prognozę na danych testowych, które nie były używane w procesie trenowania modelu. Dzięki temu możemy ocenić, jak dobrze model potrafi generalizować swoje przewidywania na nieznanych danych. Ostatecznie, uzyskane prognozy z obu zbiorów mogą być analizowane w celu oceny wydajności i jakości modelu.

|  |
| --- |
| # Prognoza  train\_predict\_rate = model\_rate.predict(X\_train\_rate)  test\_predict\_rate = model\_rate.predict(X\_test\_rate) |

***Odskalowanie danych***

W tym kroku odbywa się odskalowanie (ang. *inverse scaling*) danych, które były wcześniej przekształcone do skali innej niż oryginalna, aby model mógł je łatwiej przetwarzać. Skaler scaler\_rate jest używany do odtworzenia pierwotnych wartości, czyli przekształcenia prognoz i rzeczywistych wyników z powrotem do oryginalnej skali.

Pierwsza część kodu dotyczy prognoz modelu. Dla danych treningowych (train\_predict\_rate) i testowych (test\_predict\_rate) najpierw do prognoz dodawana jest kolumna wypełniona zerami, aby dostosować dane do formatu oczekiwanego przez funkcję inverse\_transform. Następnie funkcja inverse\_transform skaluje te dane z powrotem do ich oryginalnej skali, a [:, 0] służy do wyodrębnienia pierwszej kolumny, czyli odwróconych prognoz.

Podobnie, dla rzeczywistych wartości (y\_train\_rate i y\_test\_rate), po przekształceniu danych do formatu zgodnego z funkcją inverse\_transform, również dokonuje się odskalowania, przywracając te wartości do oryginalnej skali. Dzięki temu, zarówno prognozy, jak i rzeczywiste dane, są teraz w tej samej skali, co pozwala na ich porównanie w kontekście rzeczywistych wartości. Ten krok ma na celu przywrócenie danych do ich pierwotnej skali, co jest kluczowe dla interpretacji wyników i oceny skuteczności modelu.

|  |
| --- |
| # Odskalowanie danych  train\_predict\_rate = scaler\_rate.inverse\_transform(np.concatenate((train\_predict\_rate, np.zeros((train\_predict\_rate.shape[0], 1))), axis=1))[:, 0]  test\_predict\_rate = scaler\_rate.inverse\_transform(np.concatenate((test\_predict\_rate, np.zeros((test\_predict\_rate.shape[0], 1))), axis=1))[:, 0]  y\_train\_rescaled\_rate = scaler\_rate.inverse\_transform(np.concatenate((y\_train\_rate.reshape(-1, 1), np.zeros((y\_train\_rate.shape[0], 1))), axis=1))[:, 0]  y\_test\_rescaled\_rate = scaler\_rate.inverse\_transform(np.concatenate((y\_test\_rate.reshape(-1, 1), np.zeros((y\_test\_rate.shape[0], 1))), axis=1))[:, 0] |

***Funkcja straty dla szeregu pierwszego***

Na wykresie funkcji straty dla 250 epok widać wyraźną różnicę między stratą treningową a walidacyjną. Strata treningowa spada bardzo szybko, osiągając znacznie niższy poziom w porównaniu do straty walidacyjnej. W pierwszych epokach procesu treningowego straty są duże, ale w miarę postępu treningu, straty te maleją w tempie znacznie szybszym niż w przypadku straty walidacyjnej. Z kolei strata walidacyjna, choć również spada, utrzymuje się na wyższym poziomie, co sugeruje, że model dobrze dopasowuje się do danych treningowych, ale jego zdolność generalizacji do danych walidacyjnych rozwija się wolniej. Można zauważyć, że funkcja jeszcze lekko dąży do minimum, ale wyraźnie hamuje. Tego typu zachowania mogą wskazywać, że nie ma sensu wprowadzać większej liczby epok bo może dość do przeuczenia modelu.

Obraz zawierający tekst, Wykres, linia, diagram

Opis wygenerowany automatycznie

Kolejny raz ucząc ten sam model, który teraz będzie już po 500 epokach, można zauważyć, że się przeuczył (*ang. overfitting*). Pomimo, że strata treningowa maleje to strata walidacyjna zamiast dążyć do minimum w okolicach 350 (250+100) epoki zaczęła rosnąć. Oznacza to że model po 500 epokach za bardzo dopasowuje się do danych treningowych i jego predykcje będą odbiegać od rzeczywistości dla danych testowych.

Obraz zawierający tekst, Wykres, linia, diagram

Opis wygenerowany automatycznie

Analizując te wykresy uznałam, że liczba 250 epok dla tego szeregu będzie odpowiednia, gdyż funkcja straty jest blisko minimum, a zjawisko overfittingu nie będzie bardzo wpływało na wyniki.

***Wyniki prognoz dla szeregu pierwszego***

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Jak można zauważyć predykcja przewidziała dane testowe w dobry sposób. Szereg ten jest cykliczny przypominający błądzenie losowe, nie ma typowych wzorców, dlatego jego prognozowanie jest trudne, ale model LSTM bardzo dobrze poradził sobie z tym zadaniem. Może na to również wskazywać wartość błędu RMSE.

RMSE (Root Mean Squared Error) to miara jakości modelu, która ocenia, jak dobrze model prognozuje wartości na podstawie rzeczywistych danych. Jest to pierwiastek kwadratowy z średniej kwadratów błędów, gdzie błąd to różnica między rzeczywistymi a przewidywanymi wartościami.



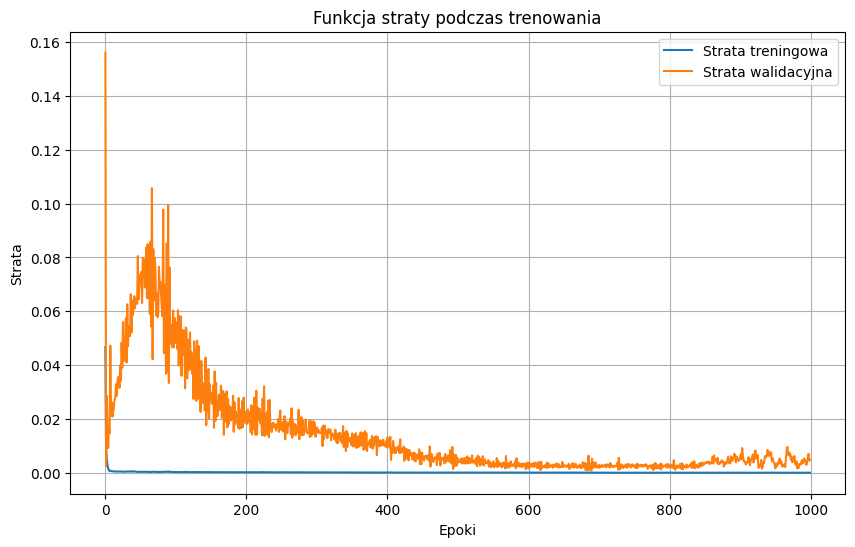
***Funkcja straty dla szeregu drugiego***

Dla 250 epok można zauważyć wyraźny wzrost straty walidacyjnej około 75 epok. Może to oznaczać, że w tym momencie model zaczyna się uczyć pewnych wzorców i daje dość błędne wyniki. Dlatego uznałam, że warto dać więcej epok aby znaleźć możliwie najmniejsze straty.

Obraz zawierający tekst, Wykres, zrzut ekranu, linia

Opis wygenerowany automatycznie

Dla empirycznego sprawdzenia, gdzie może znajdować się minimum podałam modelowi 1000 epok. Można zauważyć że około 900 epoki zaczęło występować zjawisko przeuczenia się modelu.



Na podstawie analizy wykresów uznałam, że najbardziej optymalną liczbą epok dla tego modelu będzie 650.

***Wyniki prognoz dla szeregu drugiego***

Obraz zawierający tekst, Wykres, linia, diagram

Opis wygenerowany automatycznie

Jak można zauważyć predykcja przewidziała wartości danych testowych w dobry sposób. Jest to szereg z wyraźnym trendem, i został on dobrze uchwycony. Model LSTM poradziła sobie z wychwyceniem trendu i przewidział dane, które mogły być rzeczywiste. Może na to również wskazywać wartość błędu RMSE.



***Funkcja straty dla szeregu trzeciego***

Dla 250 epok można zauważyć stagnacje funkcji straty. Może to oznaczać, że większa ilość epok daje dość błędne wyniki. Dlatego uznałam, że warto dać więcej epok aby sprawdzić tą tezę.

Obraz zawierający tekst, Wykres, linia, diagram

Opis wygenerowany automatycznie

Około epoki 500 można zauważyć, że strata walidacyjna znajduje się na podobnym poziomie, choć jest niestabilna, a strata treningowa maleje. Takie zachowanie wskazuje na zjawisko przeuczenia się modelu. Około epoki 350 można zauważyć wyraźny wzrost straty walidacyjnej.

Obraz zawierający tekst, Wykres, linia, zrzut ekranu

Opis wygenerowany automatycznie

Analizując wyniki uznałam, że najbardziej optymalną liczbą epok dla tego szeregu będzie 300 epok.

***Wyniki prognoz dla szeregu trzeciego***

Obraz zawierający tekst, linia, Wykres, Czcionka

Opis wygenerowany automatycznie

Jak można zauważyć predykcja przewidziała wartości danych testowych w dobry sposób. Jest to szereg z trendem i wyraźną sezonowością. Parametry te został on dobrze uchwycone. Model LSTM poradziła sobie z wychwyceniem trendu i sezonowości, dzięki czemu przewidział dane, które mogły być rzeczywiste. Warto też zwrócić uwagę na to że w danych testowych po głównym skoku sezonowym znajdują się dwa mniejsze, które nie są tak popularne w wcześniejszych sezonach, a jednak model je przewidział, co tylko potwierdza niezwykłe działanie modeli LSTM. Zjawisko, o którym mowa zaznaczyłam w fioletowych okręgach.

Obraz zawierający tekst, linia, Wykres, Czcionka

Opis wygenerowany automatycznie

Na dobre prognozy może również wskazywać wartość błędu RMSE.



## 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)