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

Prognozowanie szeregów czasowych: metody klasyczne i elementy uczenia maszynowego

Praca dyplomowa inżynierska

Opiekun pracy:
dr hab. Liliana Rybarska-Rusinek, prof. PRz

Rzeszów, 2025

Spis treści

Wstęp

Prognozowanie szeregów czasowych stanowi jeden z najważniejszych aspektów analizy danych w kontekście przewidywania przyszłych wartości na podstawie obserwacji historycznych. Szeregi czasowe, będące sekwencją danych mierzonych w regularnych odstępach czasu, występują w wielu dziedzinach, takich jak finanse, ekonomia, czy energetyka. Ich analiza pozwala na identyfikację wzorców, trendów i sezonowości, które są kluczowe dla podejmowania decyzji strategicznych oraz operacyjnych.

Klasyczne podejścia do analizy szeregów czasowych, takie jak modele ARIMA, oferują solidne i matematycznie uzasadnione narzędzia do modelowania danych stacjonarnych. Modele te umożliwiają identyfikację i kwantyfikację zależności pomiędzy wartościami zmiennej w czasie, co czyni je skutecznymi w prognozowaniu wielu zjawisk o regularnym charakterze. Jednak w przypadku danych niestacjonarnych, charakteryzujących się złożonymi wzorcami i długoterminowymi zależnościami, klasyczne metody mogą nie być wystarczająco elastyczne.

W ostatnich latach dynamiczny rozwój uczenia maszynowego, a szczególnie głębokiego uczenia, otworzył nowe możliwości w analizie szeregów czasowych. Modele takie jak LSTM, będące rozszerzeniem rekurencyjnych sieci neuronowych (RNN), zostały zaprojektowane z myślą o radzeniu sobie z niestandardowymi problemami. Dzięki swojej architekturze i mechanizmowi bramek, LSTM są w stanie efektywnie przechowywać informacje o długoterminowych zależnościach w danych, co czyni je szczególnie użytecznymi w przypadku szeregów niestacjonarnych i charakteryzujących się skomplikowaną strukturą.

Cel pracy

Celem niniejszej pracy jest zbadanie skuteczności metod klasycznych oraz nowoczesnych podejść opartych na uczeniu maszynowym w analizie i prognozowaniu szeregów czasowych. Analizie poddano różne typy szeregów czasowych, a wyniki uzyskane za pomocą modeli ARIMA i LSTM zostały porównane pod kątem ich dokładności, zdolności do uchwycenia wzorców oraz zastosowania w praktycznych scenariuszach. Praca ma na celu wskazanie mocnych i słabych stron każdej z metod oraz przedstawienie rekomendacji dotyczących ich wykorzystania w różnych kontekstach.

Zakres pracy

Zakres niniejszej pracy obejmuje przegląd teoretyczny oraz praktyczne zastosowanie klasycznych i nowoczesnych metod analizy i prognozowania szeregów czasowych. Struktura pracy jest podzielona na cztery główne rozdziały, które kolejno wprowadzają w tematykę, prezentują zastosowane metody, przeprowadzają analizę wybranych danych oraz dokonują prognoz przy użyciu wybranych modeli.

W rozdziale pierwszym przedstawiono wprowadzenie teoretyczne, które stanowi fundament dla dalszej analizy. Zawiera ono definicje i podstawowe własności szeregów czasowych, takie jak trend, sezonowość czy stacjonarność, a także omówienie jednego z najpopularniejszych modeli klasycznych – ARIMA. Opisano jego strukturę, zastosowania oraz wymagania związane z analizą danych czasowych.

Rozdział drugi poświęcono metodom uczenia maszynowego w kontekście analizy szeregów czasowych. W pierwszej części omówiono model RNN, który stanowi podstawę dla bardziej zaawansowanego modelu LSTM, przedstawionego w kolejnej sekcji. Skupiono się na architekturze, działaniu oraz możliwościach obu modeli w kontekście danych czasowych.

W trzecim rozdziale zaprezentowano analizę wybranych szeregów czasowych przy użyciu metod klasycznych. Przeprowadzono szczegółową analizę trzech typów szeregów czasowych, uwzględniając ich opis, wizualizacje oraz zakres zmienności. Następnie dokonano doboru odpowiednich modeli, takich jak ARIMA, w oparciu o charakterystykę danych oraz testy diagnostyczne.

Rozdział czwarty koncentruje się na prognozowaniu szeregów czasowych za pomocą wybranych modeli. Prognozy wykonano z zastosowaniem metody ARIMA oraz LSTM, a następnie porównano wyniki obu podejść. Wykorzystano kryteria oceny, takie jak dokładność prognoz, zdolność uchwycenia wzorców w danych oraz przydatność w różnych kontekstach aplikacyjnych.

Na końcu pracy znajduje się podsumowanie, w którym zebrano wnioski płynące z przeprowadzonej analizy oraz prognoz. Zawarto także sugestie dotyczące dalszych badań oraz możliwości rozszerzenia zastosowanych metod na inne obszary. Pracę kończy bibliografia, zawierająca wykaz literatury i źródeł, które stanowiły podstawę teoretyczną i praktyczną dla omawianych zagadnień.

Rozdział 1 Wprowadzenie teoretyczne

W rozdziale 1 pracy przybliżymy podstawowe pojęcia związane z szeregami czasowymi oraz zaprezentujemy ważną grupę modeli – ARIMA [1]. Zrozumienie tych pojęć oraz metodologii jest kluczowe do poprawnej analizy i prognozowania szeregów czasowych. Przedstawiona poniżej teoria ma charakter poglądowy mający na celu ułatwienie zrozumienia analizy praktycznej. Ścisłe definicje matematyczne, własności i twierdzenia stanowiły treść wykładów kursowych z modułu „Szeregi czasowe" i „Procesy stochatyczne" i nie zostały przytoczone w tej pracy.

1.1 Definicja i własności szeregów czasowych

Szereg czasowy (ang. *time series*) [2] 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.

Z teoretycznego punktu widzenia szereg czasowy to pojedyncza realizacja procesu stochastycznego.

Stacjonarność szeregu czasowego

Ważną grupę procesów stochastycznych stanowią procesy stacjonarne dla których średnia i wariancja są stałe, natomiast funkcje autokorelacji zależy wyłącznie od opóźnienia.

Szereg jest stacjonarny jeżeli jest realizacją procesu stacjonarnego. Sprawdzenie, czy szereg czasowy jest stacjonarny, umożliwiają testy m.in.:

- *Rozszerzony test Dickeya-Fullera* – test pierwiastka jednostkowego, który wykrywa obecność trendu w szeregu czasowym.

Hipotezy: H0 - szereg nie jest stacjonarny, H1 - szereg jest stacjonarny.

- *Test Phillipsa-Perrona* – test statystyczny bazujący na estymacji autokorelacji reszt. *Hipotezy*: H0 - szereg nie jest stacjonarny, H1 - szereg jest stacjonarny.
- *Test Kwiatkowskiego-Phillipsa-Schmidta-Shina (KPSS)* – test oparty na analizie wariancji między próbkami z różnych okresów szeregu czasowego.
Hipotezy: H0 - szereg jest stacjonarny, H1 - szereg nie jest stacjonarny.

Aby ocenić wyniki testów, sprawdzamy wartość *p-value*. Jeżeli wartość ta jest większa od poziomu istotności np. $\alpha = 0{,}05$, to nie ma podstaw do odrzucenia hipotezy zerowej.
Teoria dotycząca procesów stacjonarnych jest szeroko rozwinięta, żeby skorzystać z tej teorii do analizy rzeczywistych szeregów czasowych należy wcześniej, poprzez odpowiednie przekształcenia (m.in. transformacja Boxa-Coxa, różnicowanie, korekty kalendarzowe) doprowadzić je do postaci stacjonarnej.

Trend

Trend odnosi się do długoterminowego kierunku, w jakim zmieniają się wartości danych w szeregu czasowym [1]. Jest to ogólny wzorzec wzrostu, spadku lub stagnacji, który utrzymuje się przez dłuższy okres. Trend może być wynikiem różnych czynników, takich jak zmiany demograficzne, technologiczne, gospodarcze lub społeczne. Przykłady trendu w szeregach czasowych to: wzrost liczby wskaźnika sprzedaży na przestrzeni lat, spadek cen określonych towarów w wyniku ulepszeń technologicznych, czy stopniowy wzrost średniej temperatury w wyniku zmian klimatycznych. Trend może być liniowy (stałe tempo zmiany) lub nieliniowy (zmienne tempo wzrostu lub spadku).

Sezonowość

Sezonowość, to powtarzający się wzorzec w danych, który występuje w stałych, regularnych odstępach czasu, najczęściej w ramach jednego roku, miesiąca, tygodnia lub innego okresu. Jest ona spowodowana zjawiskami kalendarzowymi lub sezonowymi, takimi jak pory roku, święta, dni tygodnia, czy godziny w ciągu dnia. Przykładowo, sprzedaż lodów wzrasta latem (sezonowość roczna), a ruch w restauracjach typu fast-food jest wyższy w weekendy (sezonowość tygodniowa).

Różnicowanie niestacjonarnego szeregu czasowego

Różnicowanie [3] służy do przekształcania niestacjonarnych szeregów czasowych w szeregi stacjonarne. Proces ten eliminuje trend i sezonowość.

Proces różnicowania polega na obliczeniu różnicy Y'_t między kolejnymi obserwacjami szeregu Y_t:

$$Y'_t = Y_t - Y_{t-1}.$$

Oprócz różnicowania z opóźnieniem 1 możemy stosować kilkukrotne różnicowanie oraz różnicowanie z opóźnieniem sezonowym s

$$Y'_t = Y_t - Y_{t-s}.$$

p-value

p-value (prawdopodobieństwo testowe), to miara wykorzystywana we wnioskowaniu statystycznym do określenia, czy wyniki testu statystycznego są istotne. Pozwala na weryfikację hipotez statystycznych. Inaczej, *p-value* określa prawdopodobieństwo uzyskania obserwacji ekstremalnych, zakładając prawdziwość hipotezy zerowej. Niska *p-value* (zwykle poniżej poziomu istotności 0,05) sugeruje, że można odrzucić hipotezę zerową na rzecz hipotezy alternatywnej.

Autokorelacja

Autokorelacja określa istnienie zależności (korelacji) pomiędzy wartościami danej zmiennej w różnych okresach czasu. Oznacza to przykładowo, że wcześniejsze obserwacje mogą wpływać na przyszłe obserwacje. Obecność autokorelacji ma wpływ na wybór modelu statystycznego wykorzystywanego w analizie.

Test Ljung-Boxa

Test Ljung-Boxa to test statystyczny używany do oceny, czy dane są skorelowane w czasie, czyli czy występuje autokorelacja w opóźnieniach (lagach). Test ten jest szczególnie użyteczny w analizie szeregów czasowych do sprawdzenia dopasowania modelu do danych (czy reszty są losowe). Hipotezami testu są: H0 – dane nie są skorelowane (teoretyczne autokorelacje są równe 0), H1 – istnieją autokorelacje danych (autokorelacje są niezerowe). Niska *p-value* oznacza, że należy odrzucić hipotezę H0 na rzecz H1, co z kolei wskazuje na autokorelację danych.

Normalność reszt

Polega na sprawdzeniu, czy rozkład reszt modelu jest zgodny z rozkładem normalnym. Normalność reszt oznacza, że model jest w stanie uchwycić wzorce i zmienność w danych. Sprawdzenie normalności reszt można przeprowadzić za pomocą testu Jarque-Bera lub analizy graficznej.

Test Jarque-Bera

Test Jarque-Bera jest testem statystycznym sprawdzającym, czy reszty modelu mają rozkład normalny. Test ten opiera się na analizie skośności (asymetrii) i kurtozy (spłaszczenia) danych. Hipotezami testu są: H0 – reszty mają rozkład normalny, H1 – reszty nie mają rozkładu normalnego. Niska *p-value* oznacza, że należy odrzucić hipotezę H0 na rzecz H1, co wskazuje na brak rozkładu normalnego reszt.

Heteroskedastyczność reszt

Heteroskedastyczność reszt odnosi się do sytuacji, w której wariancja reszt modelu nie jest stała w całym zakresie danych. Jest to problematyczne w modelach regresji, ponieważ może prowadzić do nieefektywnych estymacji parametrów i błędnych wniosków. W wykryciu heteroskedastyczności reszt może pomóc test Breuscha-Pagana.

1.2 Modele ARIMA

Model ARIMA (ang. *AutoRegressive Integrated Moving Average*) [4] 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 Y_t w danym momencie na podstawie wcześniejszych obserwacji tej samej zmiennej [5]. Model *AR(p)*, gdzie parametr *p* oznacza liczbę opóźnionych obserwacji (lagów), zakłada, że wartość zmiennej w chwili t zależy od jej wcześniejszych wartości:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t$$

gdzie Y_t to wartość zmiennej w czasie *t*,

$\phi_1, \phi_2, \ldots, \phi_p$ to współczynniki modelu,

ϵ_t to biały szum.

Model średniej ruchomej MA(q)

Model średniej ruchomej zakłada, że bieżąca wartość zmiennej Y_t jest kombinacją poprzednich losowych fluktuacji [5]. Model *MA(q)*, gdzie q oznacza liczbę opóźnień błędów losowych, zakłada, że wartość zmiennej w danym momencie zależy od wcześniejszych błędów:

$$Y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_p \epsilon_{t-p}$$

gdzie $\epsilon_t, \epsilon_{t-1}, \ldots, \epsilon_{t-p}$ to biały szum,

$\theta_1, \theta_2, \ldots, \theta_p$ to współczynniki modelu.

Model ARIMA(p,d,q)

Model *ARIMA* łączy w sobie model autoregresji *AR*(*p*), średniej ruchomej *MA(q)* oraz d-krotne różnicowanie (*d*) [5]. Model ten opisuje d-krotnie zróżnicowany szereg czasowy za pomocą kombinacji wcześniejszych wartości zmiennej oraz białego szumu.

Ogólna postać modelu *ARIMA(p, d, q)* wygląda następująco: po d-krotnym zróżnicowaniu szeregu Y_t otrzymujemy $\tilde{Y}_t$ oraz

$$\tilde{Y}_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

gdzie

$\phi_1, \phi_2, \ldots, \phi_p$ to współczynniki autoregresji,

$\theta_1, \theta_2, \ldots, \theta_q$ to współczynniki średniej ruchomej,

ϵ_t to szum biały.

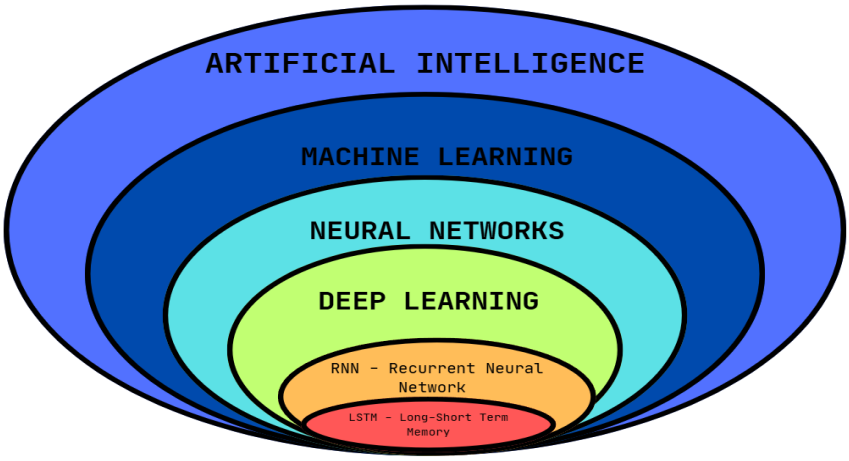
Model ARIMA jest szeroko stosowany w prognozowaniu szeregów czasowych ze względu na swoją elastyczność oraz zdolność do modelowania różnych wzorców, takich jak trendy i sezonowość [6]. Jednak, dobór odpowiednich wartości parametrów modelu wymaga analizy danych oraz testów diagnostycznych, takich jak autokorelacja reszt, czy testy stacjonarności.

Rozdział 2 Metody uczenia maszynowego dla szeregów czasowych

Podobnie jak w rozdziale 1., przedstawiona poniżej teoria ma charakter poglądowy mający na celu ułatwienie zrozumienia analizy praktycznej. Ścisłe definicje matematyczne, własności i twierdzenia stanowiły treść wykładów kursowych z modułu „Uczenie maszynowe”

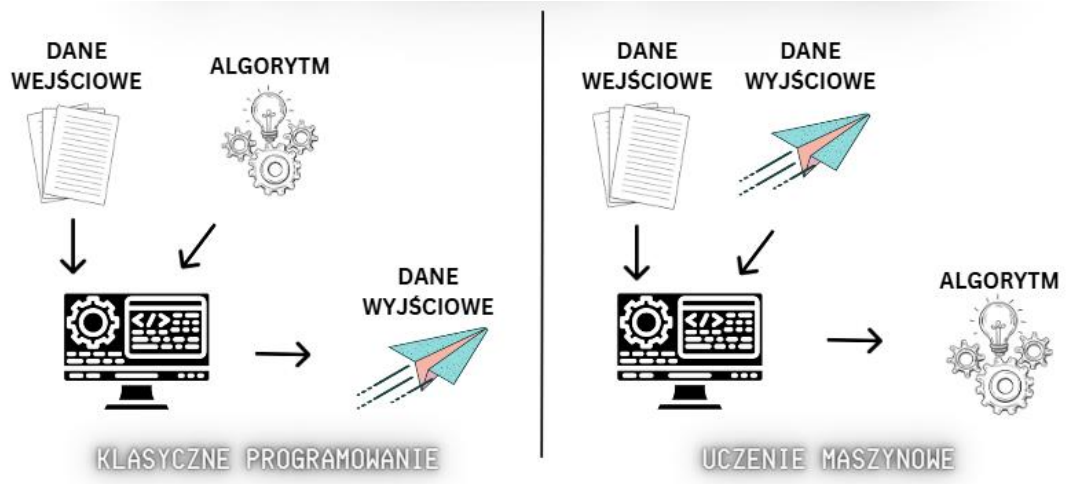
Uczenie maszynowe (*ang. machine learning*) [7] to gałąź sztucznej inteligencji (*ang. artificial intellligence*) [8] zajmująca się tworzeniem różnorodnych algorytmów i modeli analizy danych, w której systemy samodzielnie się uczą na podstawie wprowadzonych danych. System uczenia maszynowego jest więc 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*) [9] oraz uczenie głębokie (*ang. deep learning*) [10]. Jedną z klas sieci neuronowych, stanowiących metodę uczenia głębokiego, są rekurencyjne sieci neuronowe (RNN, *ang. recurrent neural networks*) [11], a także ich wariant LSTM (*ang. long short-term memory*) [12, 13], które zostaną opisane w niniejszym rozdziale.



Rysunek 2.1 Schemat hierarchii metod sztucznej inteligencji.

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.

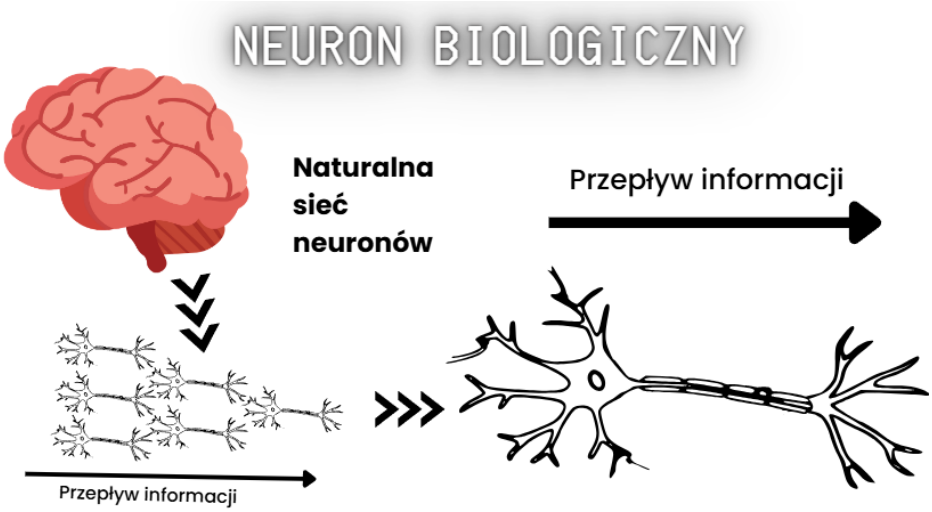


Rysunek 2.2 Różnice pomiędzy klasycznym programowaniem, a uczeniem maszynowym.

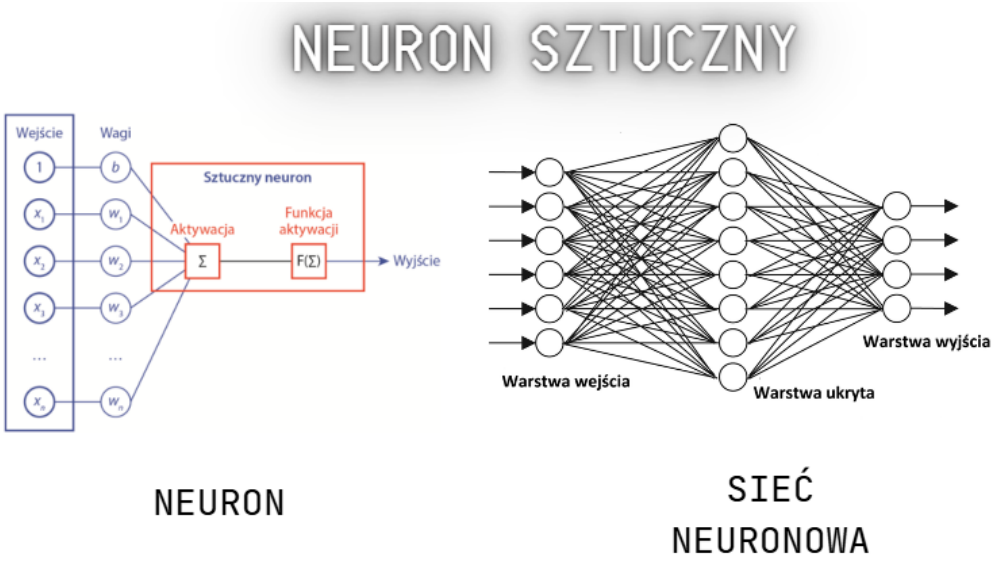
Sieć neuronowa

Sieć neuronowa **[**11**]** inspirowana jest neuronem biologicznym (rysunek 2.3). 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 2.4).



Rysunek 2.3 Przepływ informacji w neuronie naturalnym.



Rysunek 2.4 Schemat neuronu i sieci neuronów.

Funkcje aktywacji

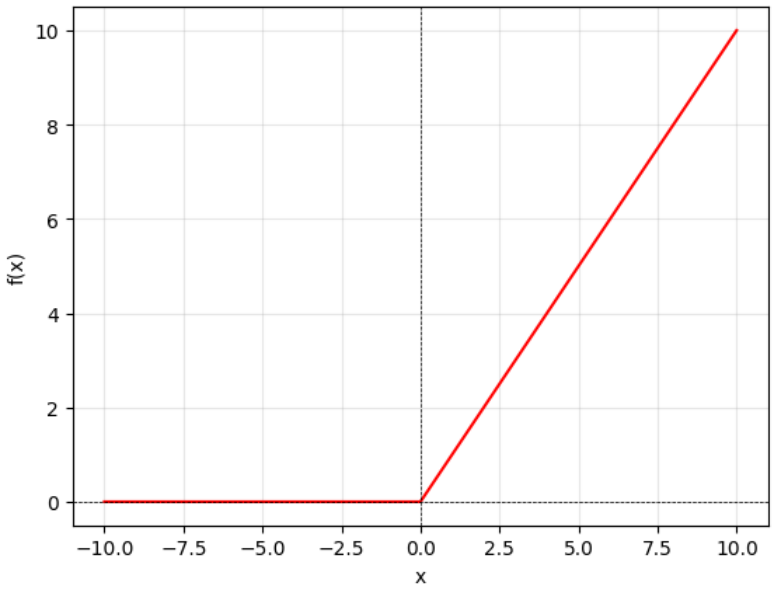
Funkcja aktywacji [14] to kluczowy element sztucznych sieci neuronowych, wybierany dla warstw ukrytych i warstwy wyjściowej, który decyduje o tym, czy i w jaki sposób neuron "aktywuje się" (przekazuje sygnał). Funkcja aktywacji wprowadza

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 neuronowej.

Najpopularniejszą funkcją aktywacji jest *funkcja ReLU* (*ang. Rectified Linear Unit*) [15]

$$f(x) = \max(0, x),$$

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

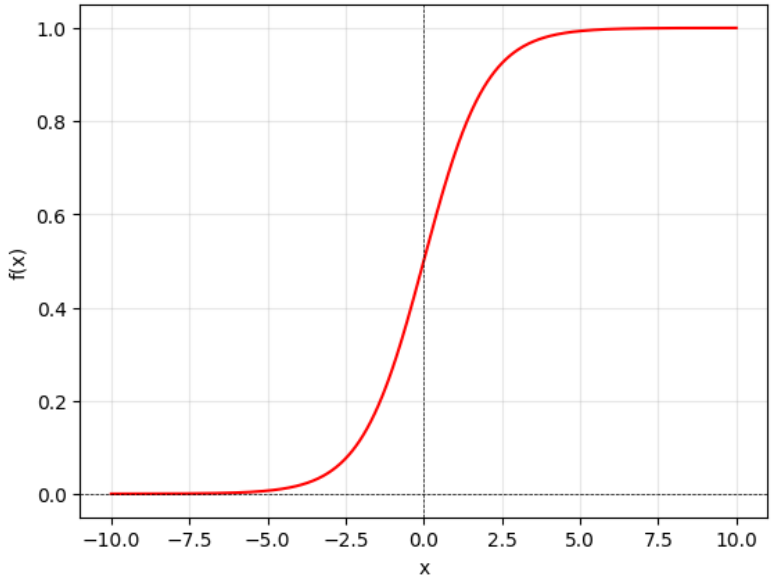


Rysunek 2.5 Wykres funkcji aktywacji ReLU.

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* [15], patrz rysunek 2.6, postaci:

$$f(x) = \frac{1}{1 + e^{-x}}$$

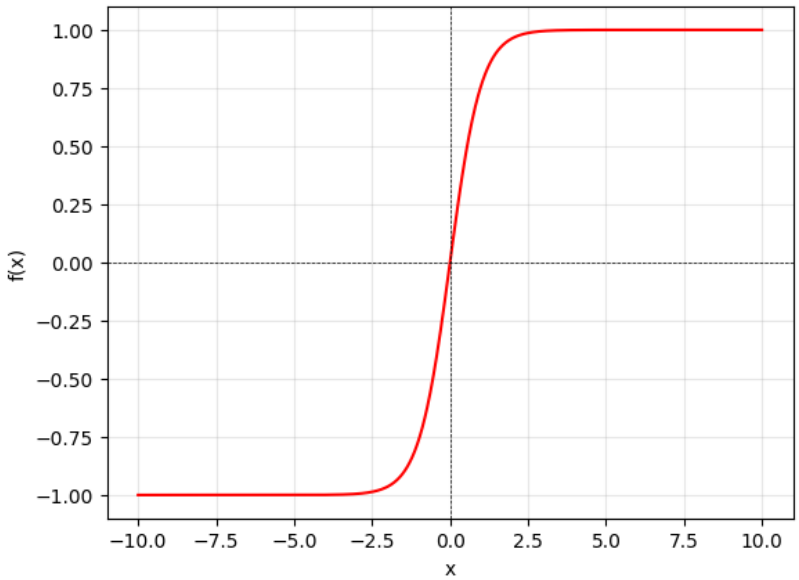


Rysunek 2.6 Wykres funkcji sigmoidalnej.

Funkcja tangensoidalna (tangens hiperboliczny, rysunek 2.7*)* jest odmianą funkcji sigmoidalnej. Jej wartości są zawarte w przedziale (-1,1).

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$

Podobnie, jak funkcja sigmoidalna, funkcja tangensoidalna 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.



Rysunek 2.7 Wykres funkcji tangensoidalnej.

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 krzyżowa

Proces walidacji krzyżowej polega na podziale dostępnych danych na dwa podzbiory: pierwszy służy do trenowania modelu (zwykle zawiera 80%-90% dostępnych danych), zaś drugi do jego testowania. W trakcie trenowania, model jest regularnie oceniany przy użyciu danych testowych (walidacyjnych). Pozwala to monitorować jego zdolność do generalizowania na nowych, niewidzianych wcześniej danych oraz dostosowywać parametry, aby poprawić jego ogólną wydajność. Jest to kluczowy etap umożliwiający identyfikację problemów takich jak przeuczenie (overfitting), gdzie model może doskonale dopasować się do danych treningowych, ale słabo radzić sobie z danymi testowymi.

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. Zatem strata treningowa zazwyczaj maleje z każdą epoką, gdyż model stopniowo dopasowuje się do danych treningowych. **Strata walidacyjna** mierzy błąd modelu na danych testowych, które nie były używane w trakcie trenowania. Strata walidacyjna może początkowo maleć, ale jeśli model zaczyna przeuczać się na danych treningowych (staje się zbyt dopasowany do tych danych), strata walidacyjna zaczyna rosnąć. Monitorowanie obu strat jest zatem niezwykle istotne.

Propagacja wsteczna

Propagacja wsteczna (*ang. backpropagation*) [10] 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*) [12], 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:

- *błąd średniokwadratowy (MSE, ang. mean squared error):* $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$,

- *średni błąd bezwzględny (MAE, ang. mean absolute error)*: $MSE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$,

gdzie y_i to wartości rzeczywiste, $\hat{y}_i$ to wartości przewidywane, n to ilość elementów.

Celem minimalizacji funkcji kosztu propagacja wsteczna często wykorzystuje metody optymalizacji. Wagi i odchylenia są aktualizowane zgodnie z regułą *gradientu prostego* (*ang. gradient descent*) [4], zmniejszając błąd modelu poprzez przesunięcie parametrów w kierunku przeciwnym do gradientu. Algorytm oblicza gradient, korzystając z reguły łańcuchowej [16], 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.

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*) [11] to sieci neuronowe zaprojektowane do przetwarzania danych sekwencyjnych, m.in. szeregów czasowych. Struktura RNN umożliwia rekurencję, co oznacza, że dane wyjściowe

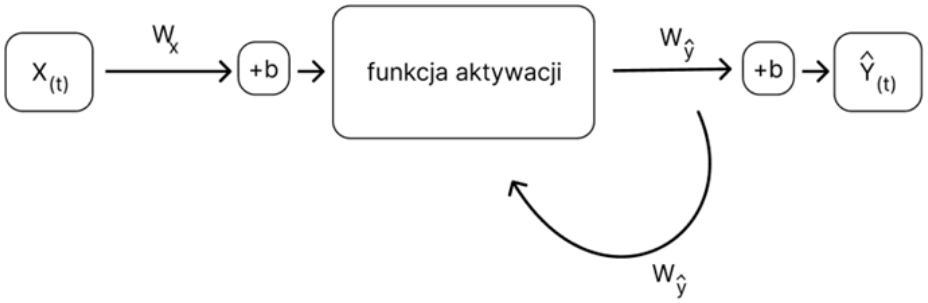
otrzymane z jednego kroku czasowego są przekazywane jako dane wejściowe do następnego kroku. Odróżnia je to od tradycyjnych sieci neuronowych, w których dane wejściowe i wyjściowe są traktowane niezależnie.

Każdy krok czasowy w RNN można opisać jako:

$$\hat{Y}_t = \varphi\left(X_t W_x + \hat{Y}_{t-1} W_{\hat{y}} + b\right),$$

gdzie $\hat{Y}_t$ to macierz wymiaru $(m \times n_{neur})$ zawierająca dane wyjściowe w kroku czasowym t, m to ilość próbek, n_{neur} liczba neuronów, φ to funkcja aktywacji, X_t to macierz wymiaru $(m \times n_{in})$ zawierająca dane wejściowe, gdzie n_{in} to liczba wejść, W_x to macierz wymiaru $(n_{in} \times n_{neur})$ zawierająca wagi dla wejść w kroku czasowym t, $W_{\hat{y}}$ to macierz wymiaru $(n_{neur} \times n_{neur})$ zawierająca wagi dla wyjść z poprzedniej iteracji, b to wektor (o wymiarze n_{neur}) zawierający człony odchyleń dla każdego neuronu. W kroku czasowym $t = 0$ 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 2.8), czyli działający rekurencyjnie. Można go też przedstawić w postaci rozwijającego na kolejne neurony (rysunek 2.9) - taka wizualizacja jest bardziej przejrzysta dla dalszych rozumowań.

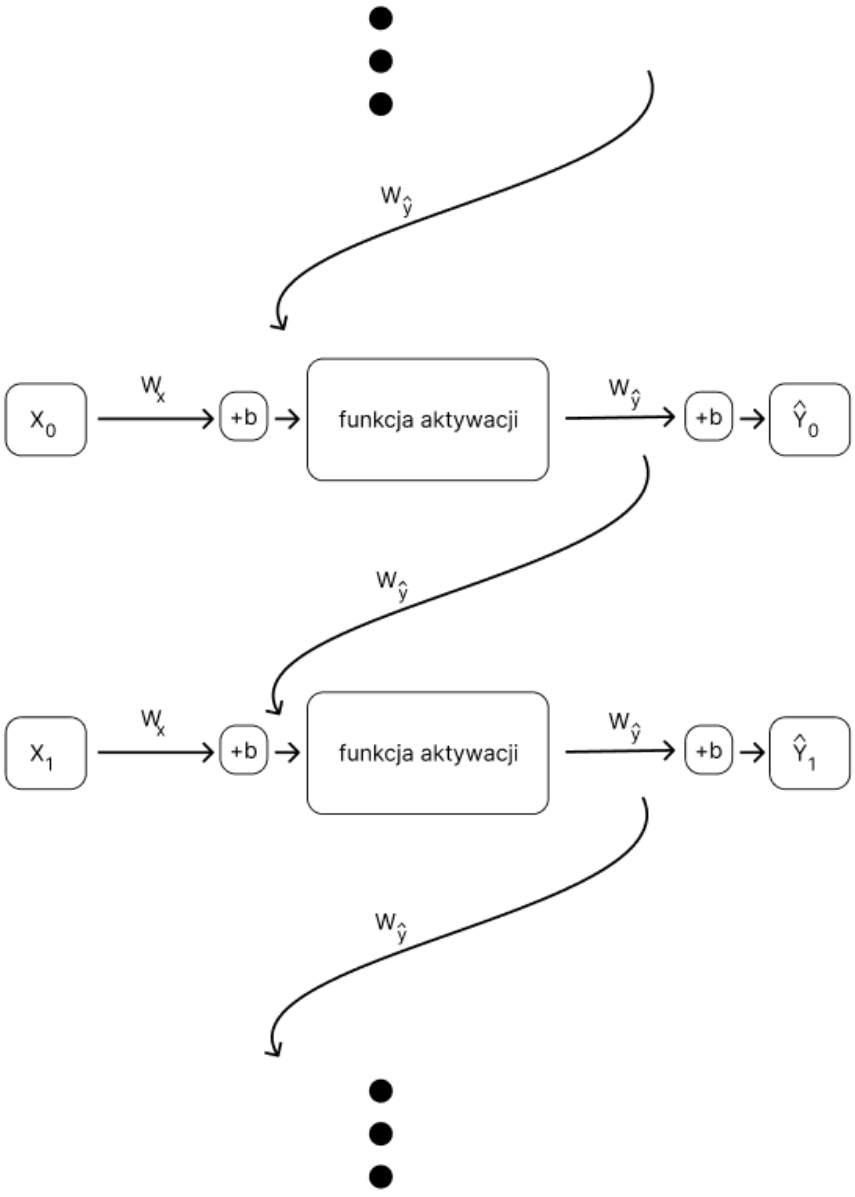


Rysunek 2.8 Pojedynczy neuron RNN.

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.



Rysunek 2.9 Rozwinięty neuron RNN.

- *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.

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)* [17]. 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 do wielokrotnego mnożenia wag. W przypadku, gdy wartości wag (elementy macierzy $W_{\hat{y}}$) 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 $W_{\hat{y}}$) są z przedziałów $(-\infty, -1),\ (1, +\infty)$. 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*) [12, 13], 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.

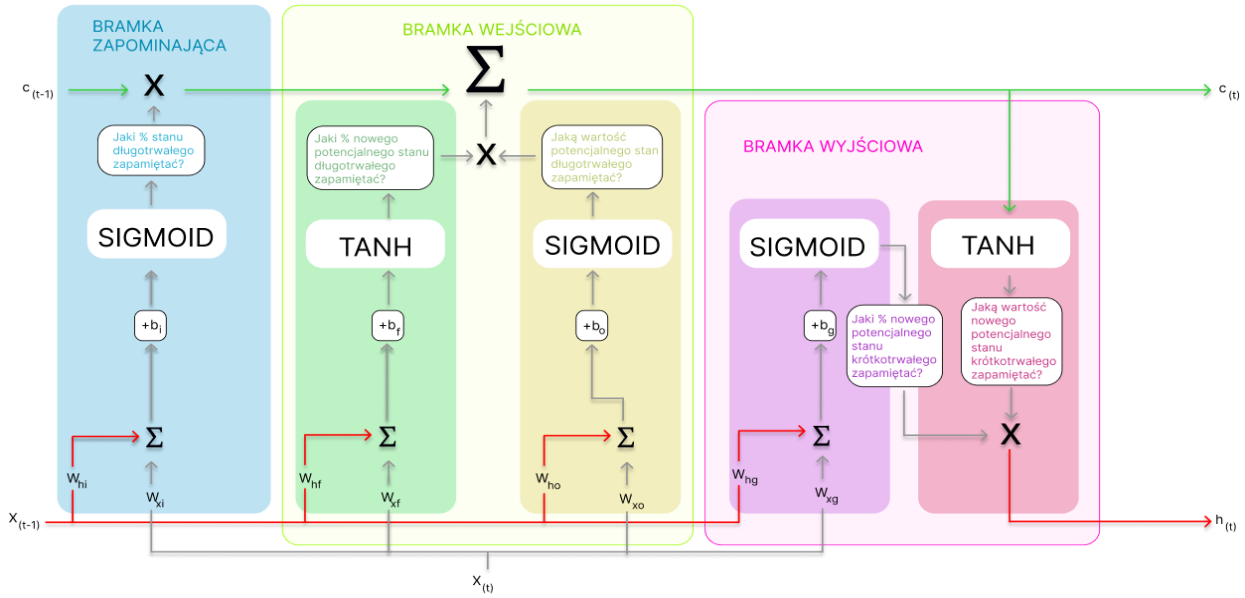
2.2 Model LSTM

Jedną z bardziej rozbudowanych rekurencyjnych sieci neuronowych jest model LSTM (*ang. long short-term memory*) [4]. 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.

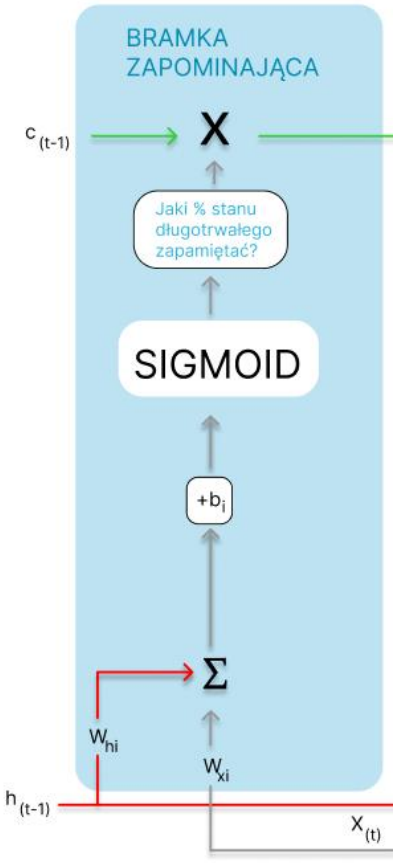
Struktura sieci LSTM

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 2.10. 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 $c_{(t)}$ (*ang. cell*) oznaczono na rysunku linią zieloną, zaś stan pamięci krótkotrwałej $h_{(t)}$ linią czerwoną. Stan $h_{(t)}$ został pozbawiony niewygodnego

mnożenia wag, dzięki czemu usunięto problem zanikającego lub wybuchającego gradientu. Stan $c_{(t)}$ posiada wagi dla każdej kolejnej bramki, dzięki czemu w wygodny sposób będzie je modyfikował, nie tracąc istotnych informacji.



Rysunek 2.10 Schemat budowy modelu LSTM.

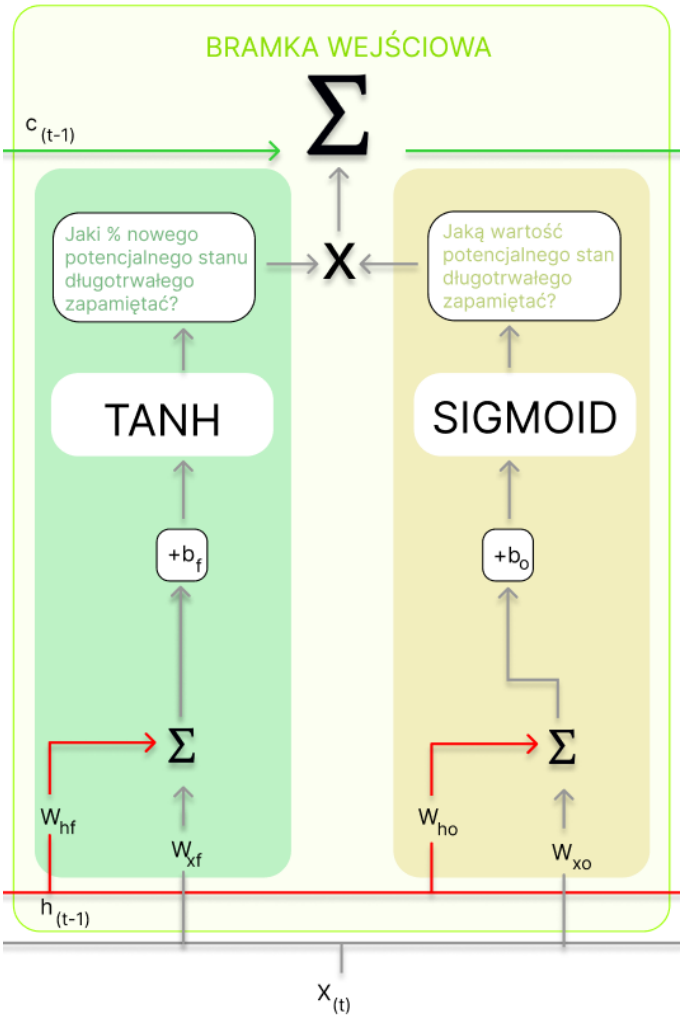


Rysunek 2.11 Schemat budowy bramki zapominającej modelu LSTM.

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 $h_{(t-1)}$ oraz $x_{(t)}$, które są mnożone przez macierze wag odpowiednio W_h i W_x. Do sumy iloczynów danych wejściowych i wag dodawane jest odchylenie (bias). Wynik podlega działaniu sigmoidalnej funkcji aktywacji

$$i_{(t)} = \sigma\left(W_{xi}^T x_{(t)} + W_{hi}^T h_{(t-1)} + b_i\right)$$

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



Rysunek 2.12 Schemat budowy bramki wejściowej modelu LSTM.

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

$$o_{(t)} = \sigma\left(W_{xo}^T x_{(t)} + W_{ho}^T h_{(t-1)} + b_o\right).$$

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

$$f_{(t)} = \tanh\left(W_{xf}^T x_{(t)} + W_{hf}^T h_{(t-1)} + b_f\right)$$

otrzymujemy wektor $c_{(t)}$ wartości nowego potencjalnego stanu długotrwałego

$$c_{(t)} = i_{(t)} \otimes c_{(t-1)} + o_{(t)} \otimes f_{(t)}.$$

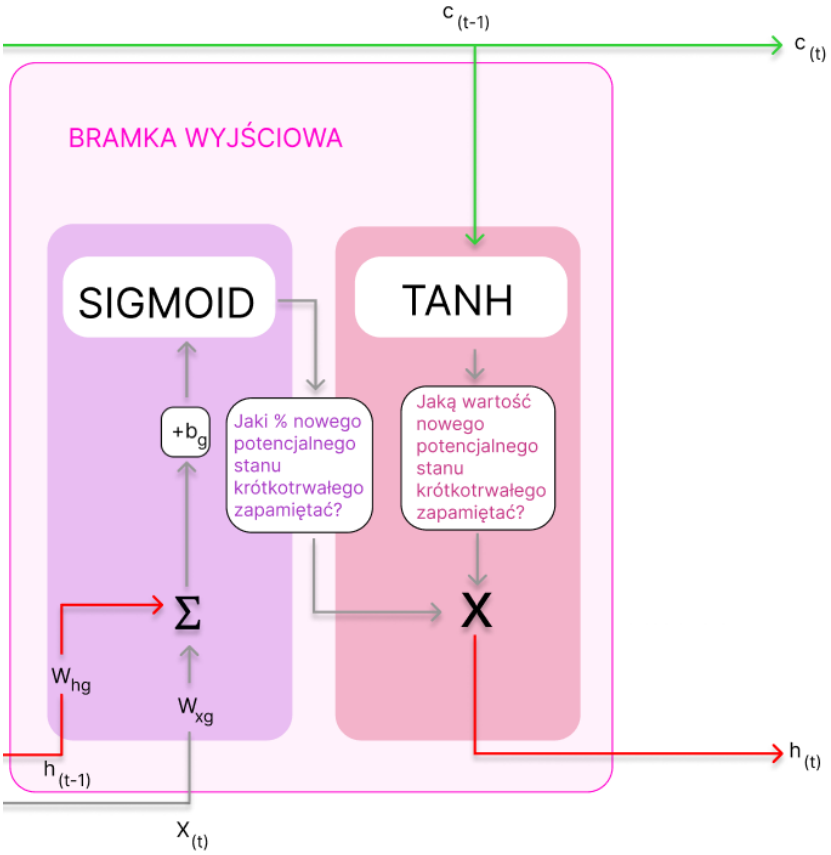
Jest to zatem główna warstwa generująca pamięć długotrwałą, a więc tą która w większym stopniu wpływa na wynik końcowy i jest kluczowa w strukturze całego modelu. Symbol $\otimes$ oznacza iloczyn Kroneckera.

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 $\tanh(c_{(t)})$, po czym dane wejściowe regulowane są za pomocą funkcji sigmoidalnej

$$g_{(t)} = \sigma\left(W_{xg}^T x_{(t)} + W_{hg}^T h_{(t-1)} + b_g\right).$$

Na koniec, wartości regulowane przez funkcję sigmoidalną oraz wartości wektora $\tanh(c_{(t)})$, są mnożone celem wyodrębnienia danych wyjściowych, a za razem danych wejściowych do kolejnej komórki

$$y_{(t)} = h_{(t)} = g_{(t)} \otimes \tanh(c_{(t)})$$



Rysunek 2.13 Schemat budowy bramki wejściowej modelu LSTM.

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: bez trendu i sezonowości, 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

Szereg czasowy z cyklami

Jako pierwszy do analizy wybrano szereg czasowy bez widocznego trendu https://fred.stlouisfed.org/series/REAINTRATREARAT1MO. Przedstawia on jednomiesięczną realną stopę procentową (ang. 1-Month Real Interest Rate). Na tym etapie dane nie wymagały obróbki, nie zawierały wartości pustych (null), braków danych (NA), ani innych komplikacji. Kod służący do pobrania danych podano poniżej:

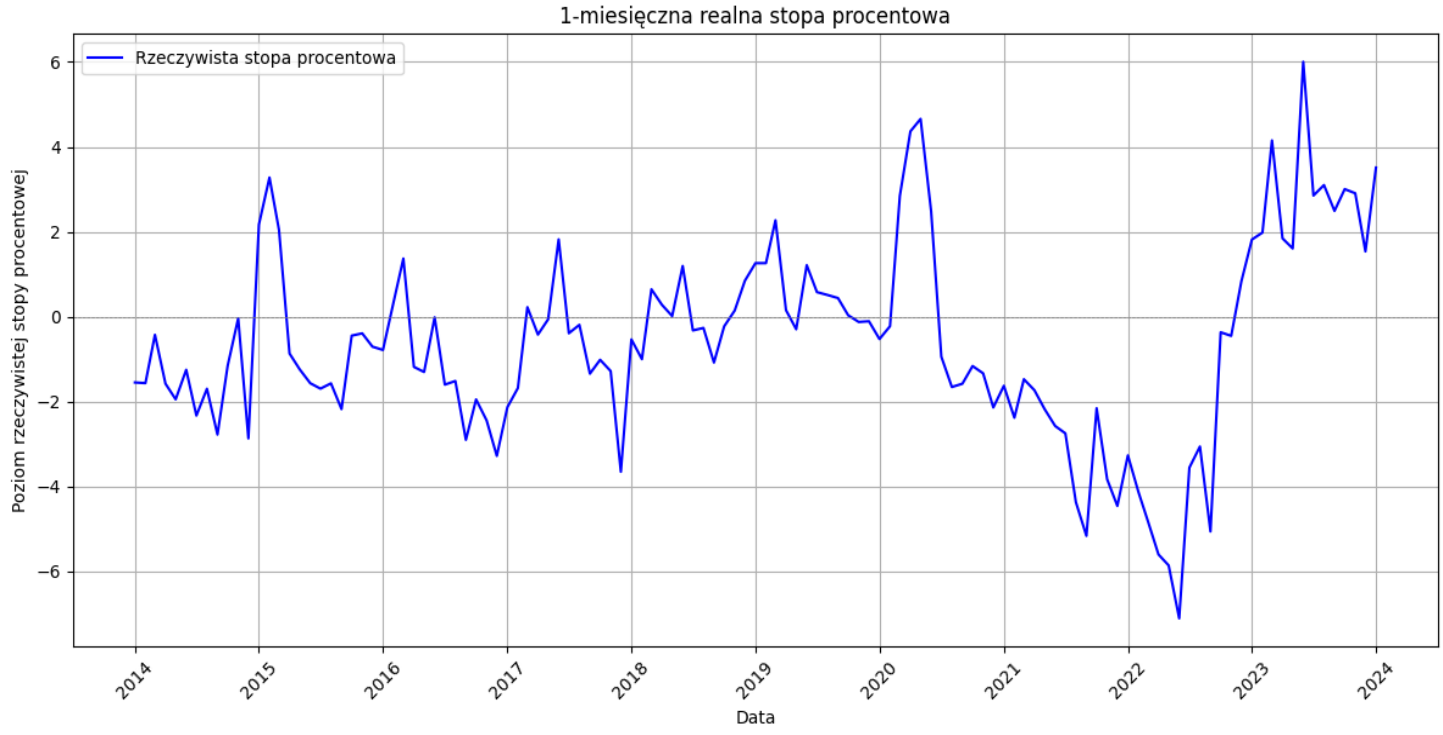
```
data_rate = pd.read_csv("C:/Users/Aldona/Documents/GitHub/Machine-learning-
in-time-series-analysis/data/interest_rate.csv", sep=',', encoding='utf-8',
index_col = 'DATE', parse_dates = True) # upewnienie się, że daty będą
rozpoznawane jako daty
df_rate = pd.DataFrame(data_rate)
df_rate.columns.values[0] = '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.

Fragment danych dla pierwszego szeregu czasowego przedstawiono w tabeli 3.1, a wizualizację całego szeregu czasowego pokazano na rysunku 3.1.

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

Tabela 3.1 Początkowe 10 wierszy danych pierwszego szeregu czasowego.



Rysunek 3.1 Wykres szeregu czasowego 1-miesięcznej realna stopy procentowej.

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

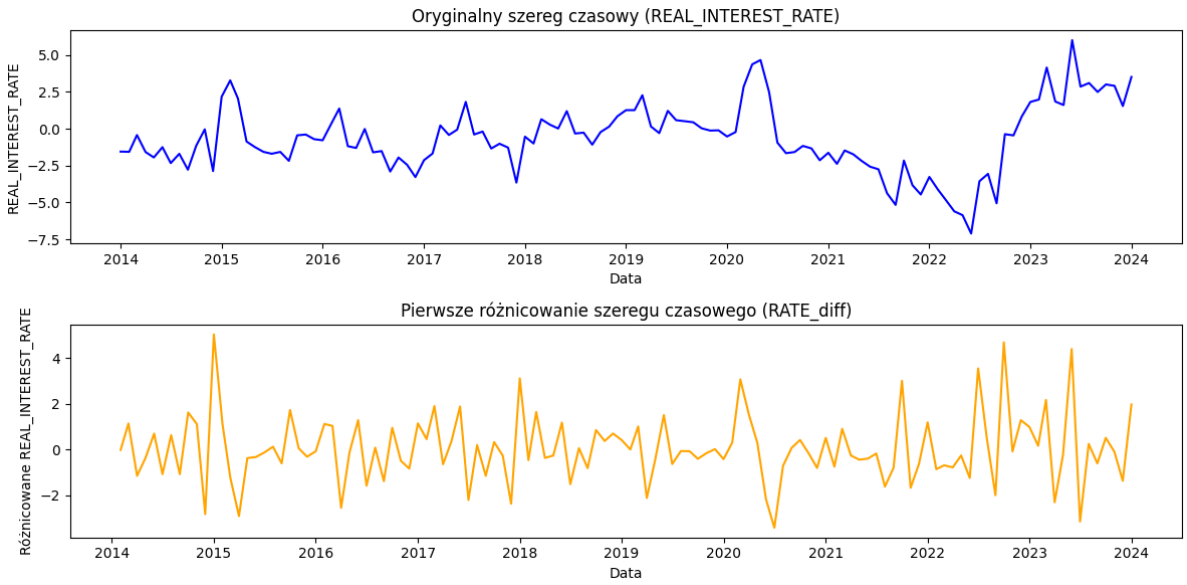
Testy stacjonarności pierwszego szeregu czasowego

Sprawdzimy, czy powyższy szereg jest stacjonarny. W tym celu przeprowadzimy testy stacjonarności. Należy pamiętać, aby przed wykonaniem testów usunąć z danych puste wartości i ewentualne braki danych.

```
Wyniki testów stacjonarności dla szeregu pierwszego (df_rate)
Test Dickeya-Fullera:
Statystyka testowa: -1.800831403563697
P-value: 0.3800490030086589
Liczba opóźnień: 6
Liczba obserwacji: 114
Wartości krytyczne: {'1%': -3.489057523907491, '5%': -2.887246327182993, '10%': -2.5804808802708528}
Brak podstaw do odrzucenia hipotezy zerowej - szereg nie jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: -3.691711098279078
P-value: 0.004231267175176973
Odrzucamy hipotezę zerową - szereg jest stacjonarny.

Test KPSS:
Statystyka testowa: 0.11083851876425525
P-value: 0.1
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Brak podstaw do odrzucenia hipotezy zerowej - szereg jest stacjonarny.
```

Rysunek 3.2 Testy stacjonarności dla pierwszego szeregu czasowego przed różnicowaniem.

Testy Phillipsa-Perrona i KPSS wskazują na stacjonarność szeregu. Natomiast test Dickeya-Fullera zaprzecza stacjonarności (rysunek 3.2). Szereg czasowy nie jest więc stacjonarny. Na rysunku 3.3 podano wyniki różnicowania szeregu.



Rysunek 3.3 Wykresy przed i po różnicowaniu pierwszego szeregu czasowego.

Doprowadzenie szeregu do postaci stacjonarnej wymagało jednokrotnego różnicowania. Po zróżnicowaniu szereg jest już stacjonarny (rysunek 3.4).

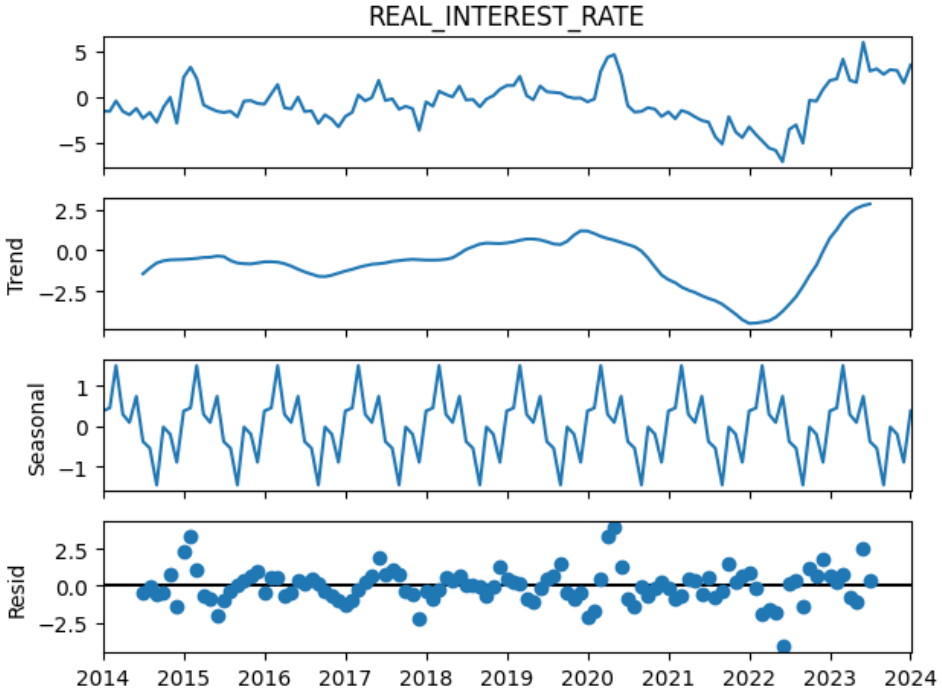
```
Wyniki testów stacjonarności dla szeregu pierwszego (df_rate)
Test Dickeya-Fullera:
Statystyka testowa: -6.254334886115214
P-value: 4.3761293906436355e-08
Liczba opóźnień: 5
Liczba obserwacji: 114
Wartości krytyczne: {'1%': -3.489057523907491, '5%': -2.887246327182993, '10%': -2.5804808802708528}
Odrzucamy hipotezę zerową - szereg jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: -15.449138331931053
P-value: 2.7991672395906277e-28
Odrzucamy hipotezę zerową - szereg jest stacjonarny.

Test KPSS:
Statystyka testowa: 0.09207132932132651
P-value: 0.1
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Brak podstaw do odrzucenia hipotezy zerowej - szereg jest stacjonarny.
```

Rysunek 3.4 Testy stacjonarności dla pierwszego szeregu czasowego po różnicowaniu.

Dekompozycja pierwszego szeregu czasowego

```
results = seasonal_decompose(df_rate['REAL_INTEREST_RATE'])
```

Po zastosowaniu dekompozycji dla oryginalnego szeregu czasowego otrzymujemy



Rysunek 3.5 Wykresy po dekompozycji szeregu czasowego dla 1-miesięcznej realnej stopy procentowej.

Na podstawie dekompozycji szeregu czasowego dla 1-miesięcznej realnej stopy procentowej, można wysnuć kilka istotnych wniosków:

- brak wyraźnego trendu. Po wielu latach stabilnych wahań, w latach 2020–2022 rzeczywista stopa procentowa osiągnęła znaczny spadek, po czym nastąpił jej wyraźny wzrost, który nadal się utrzymuje,
- widoczne są regularne, cykliczne wahania, prawdopodobnie związanych z cyklami gospodarki,
- reszty oscylują wokół zera. W niektórych okresach widać znaczne odchylenia, które mogą być efektem niespodziewanych zdarzeń.

Szereg czasowy z trendem

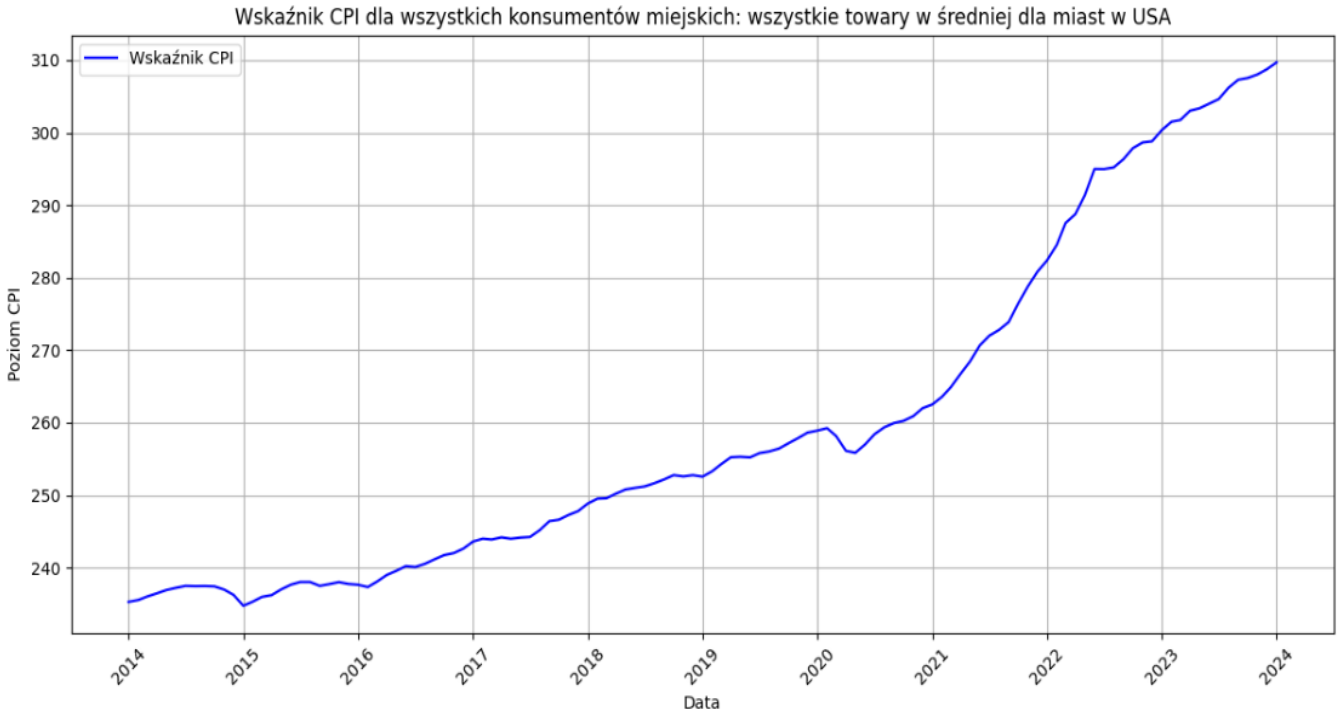
Kolejnym szeregiem czasowym, wybranym do analizy, jest szereg opisujący indeks średnich cen konsumpcyjnych w miastach w USA (ang. Consumer Price Index (CPI) for all urban consumers) https://fred.stlouisfed.org/series/CPIAUCSL. Dane nie wymagały obróbki, nie zawierały wartości pustych (null), braków danych (NA), ani innych komplikacji.

Fragment zbioru danych podano w tabeli 3.2, zaś wizualizację szeregu czasowego przedstawiono na rysunku 3.6**.** Widzimy wyraźnie zaznaczający się trend. Nie zauważamy natomiast sezonowości, ani cykliczności w danych. Przeprowadzamy testy stacjonarności szeregu czasowego.

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

[3]

Tabela 3.2 Początkowe 10 wierszy danych drugiego szeregu czasowego.



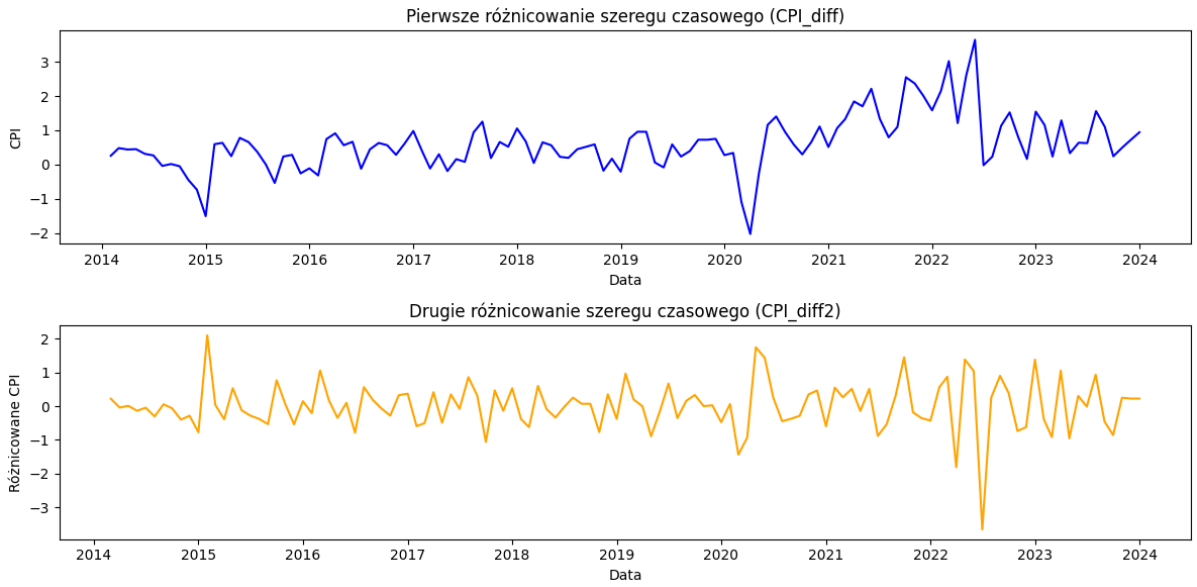
Rysunek 3.6 Wykres szeregu czasowego wskaźnika CPI dla wszystkich konsumentów miejskich.

Testy stacjonarności drugiego szeregu czasowego

```
Wyniki testów stacjonarności dla szeregu drugiego (df_cpi)
Test Dickeya-Fullera:
Statystyka testowa: 0.4649588398766165
P-value: 0.983768221407472
Liczba opóźnień: 7
Liczba obserwacji: 113
Wartości krytyczne: {'1%': -3.489589552580676, '5%': -2.887477210140433, '10%': -2.580604145195395}
Brak podstaw do odrzucenia hipotezy zerowej - szereg nie jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: 2.4650773481189323
P-value: 0.9990377163380877
Brak podstaw do odrzucenia hipotezy zerowej - szereg nie jest stacjonarny.

Test KPSS:
Statystyka testowa: 1.6359283085089014
P-value: 0.01
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Odrzucamy hipotezę zerową - szereg nie jest stacjonarny.
```

Rysunek 3.7 Testy stacjonarności dla drugiego szeregu czasowego przed różnicowaniem.

Zgodnie z oczekiwaniami wszystkie testy wskazały na niestacjonarność analizowanego szeregu czasowego (rysunek 3.7). Po pierwszym różnicowaniu testy nadal wykazywały niestacjonarność szeregu. Dopiero drugie różnicowanie doprowadziło szereg do postaci stacjonarnej (patrz rysunek 3.8 i 3.9).



Rysunek 3.8 Wykresy przed i po różnicowaniu drugiego szeregu czasowego.

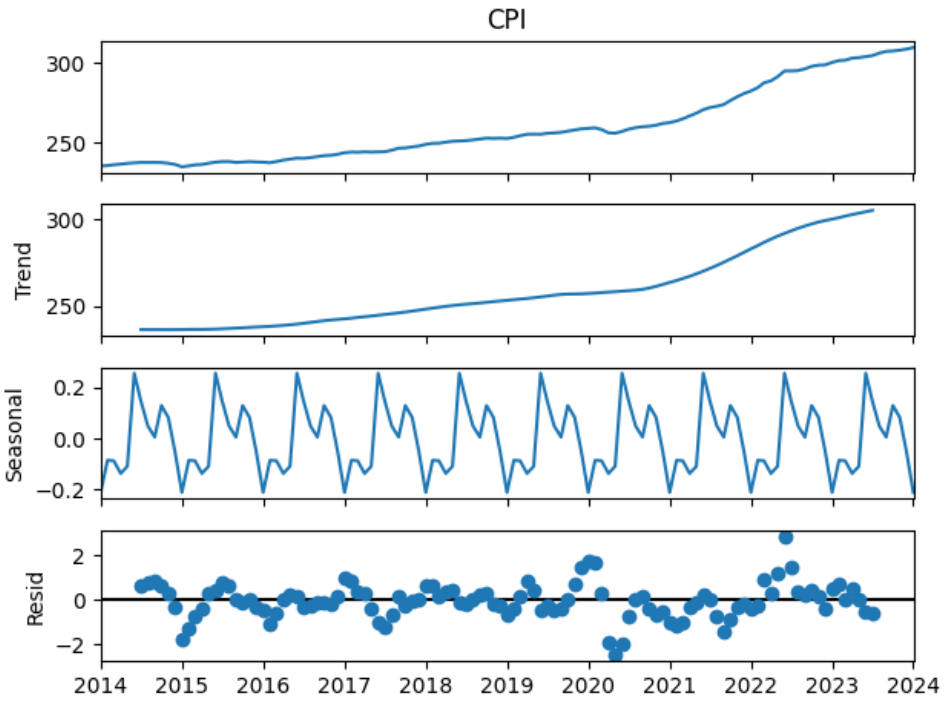
```
Wyniki testów stacjonarności dla szeregu drugiego (df_cpi)
Test Dickeya-Fullera:
Statystyka testowa: -7.821415727482808
P-value: 6.6546163646195385e-12
Liczba opóźnień: 5
Liczba obserwacji: 113
Wartości krytyczne: {'1%': -3.489589552580676, '5%': -2.887477210140433, '10%': -2.580604145195395}
Odrzucamy hipotezę zerową – szereg jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: -19.9853136645572
P-value: 0.0
Odrzucamy hipotezę zerową – szereg jest stacjonarny.

Test KPSS:
Statystyka testowa: 0.19715900759356586
P-value: 0.1
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Brak podstaw do odrzucenia hipotezy zerowej – szereg jest stacjonarny.
```

Rysunek 3.9 Testy stacjonarności dla drugiego szeregu czasowego po różnicowaniu.

Dekompozycja drugiego szeregu czasowego

Dekompozycja analizowanego szeregu czasowego dla indeksu cen konsumpcyjnych (rysunek 3.10) wskazuje na:

- wyraźny trend wzrostowy na przestrzeni całego badanego okresu, odzwierciedlający długoterminowy wzrost poziomu cen i potencjalny wpływ inflacji,
- znikome wahania sezonowe, wskazujące jednoznacznie na brak sezonowości danych,
- nieznaczne reszty oscylujące wokół zera, co świadczy o ich losowym charakterze.



Rysunek 3.10 Wykresy po dekompozycji szeregu czasowego dla wskaźnika CPI.

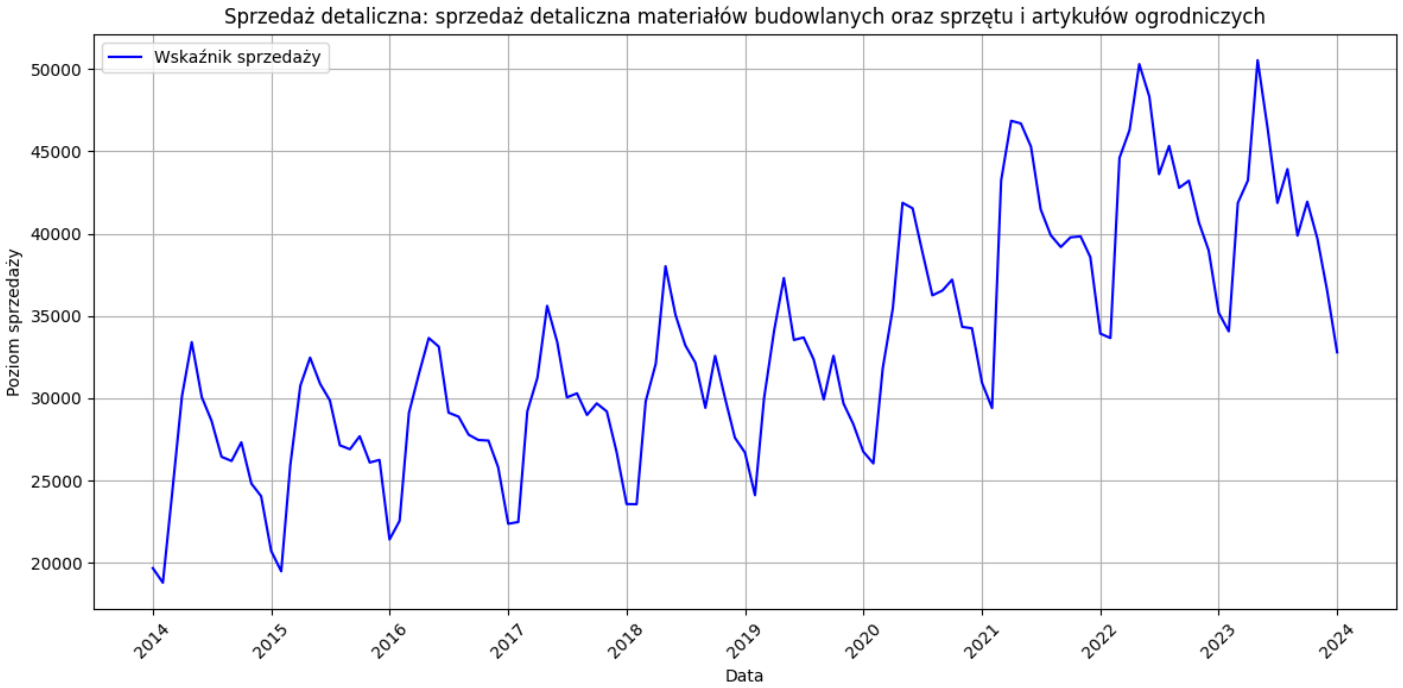
Szereg czasowy z trendem i sezonowością

Ostatni szereg czasowy poddany analizie dotyczy sprzedaży detalicznej materiałów budowlanych i sprzętu ogrodniczego (retail sales: building materials and garden equipment and supplies dealers) https://fred.stlouisfed.org/series/ MRTSSM444USN. Jest to szereg z widoczną sezonowością oraz delikatnie zaznaczonym trendem. Dane nie wymagały obróbki, nie zawierały wartości pustych (null) i braków danych (NA).

Fragment zbioru danych podano w tabeli 3.3, zaś wizualizację szeregu czasowego przedstawiono na rysunku 3.11.

DATE	MATERIALS
2014-01-01	19688
2014-02-01	18801
2014-03-01	24103
2014-04-01	30137
2014-05-01	33416
2014-06-01	30072
2014-07-01	28642
2014-08-01	26446
2014-09-01	26195
2014-10-01	27329

Tabela 3.3 Początkowe 10 wierszy danych trzeciego szeregu czasowego.



Rysunek 3.11 Wykres szeregu czasowego sprzedaży detalicznej materiałów oraz sprzętu i artykułów ogrodniczych.

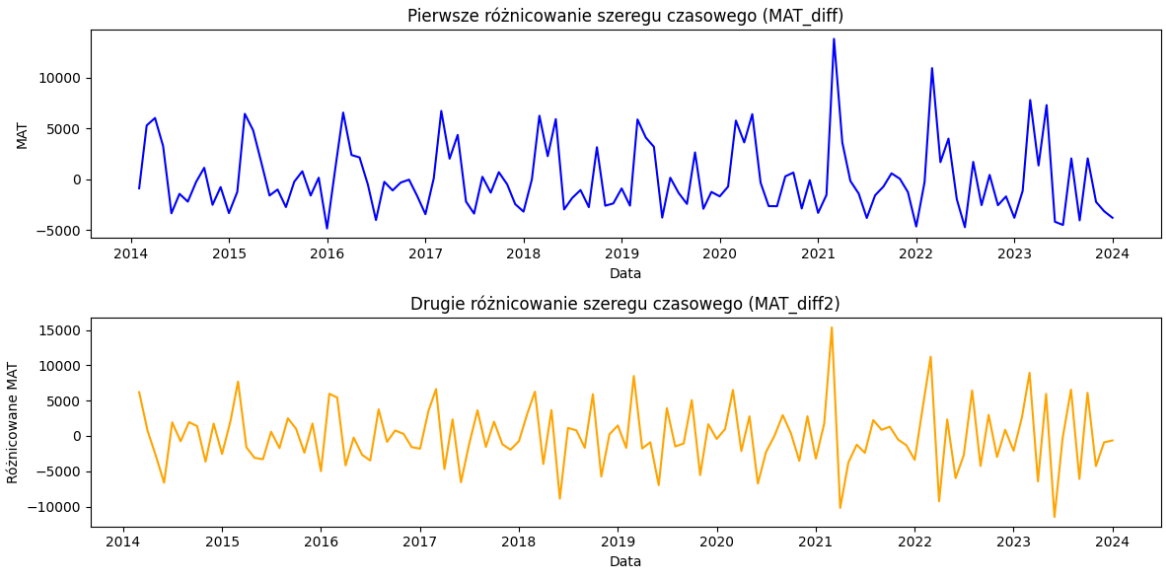
Widać wyraźnie zaznaczoną sezonowość, a także delikatny trend na przestrzeni 10-ciu lat, przyśpieszający na przestrzeni ostatnich 5-ciu lat. Przeprowadzamy testy stacjonarności szeregu czasowego.

Testy stacjonarności trzeciego szeregu czasowego

```
Wyniki testów stacjonarności dla szeregu trzeciego (df_materials)
Test Dickeya-Fullera:
Statystyka testowa: -0.5560137905254044
P-value: 0.8806146270444792
Liczba opóźnień: 12
Liczba obserwacji: 108
Wartości krytyczne: {'1%': -3.4924012594942333, '5%': -2.8886968193364835, '10%': -2.5812552709190673}
Brak podstaw do odrzucenia hipotezy zerowej - szereg nie jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: -2.8016882734502504
P-value: 0.058033014609137186
Brak podstaw do odrzucenia hipotezy zerowej - szereg nie jest stacjonarny.

Test KPSS:
Statystyka testowa: 1.5384020504071987
P-value: 0.01
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Odrzucamy hipotezę zerową - szereg nie jest stacjonarny.
```

Rysunek 3.12 Testy stacjonarności dla trzeciego szeregu czasowego przed różnicowaniem.

Podobnie, jak w przypadku drugiego szeregu czasowego, wszystkie testy wskazały na niestacjonarność analizowanego szeregu (rysunek 3.12). Po pierwszym różnicowaniu test Dickeya-Fullera wykazał niestacjonarność szeregu. Drugie różnicowanie doprowadziło szereg do postaci stacjonarnej (patrz rysunek 3.13 i 3.14).



Rysunek 3.13 Wykresy przed i po różnicowaniu trzeciego szeregu czasowego.

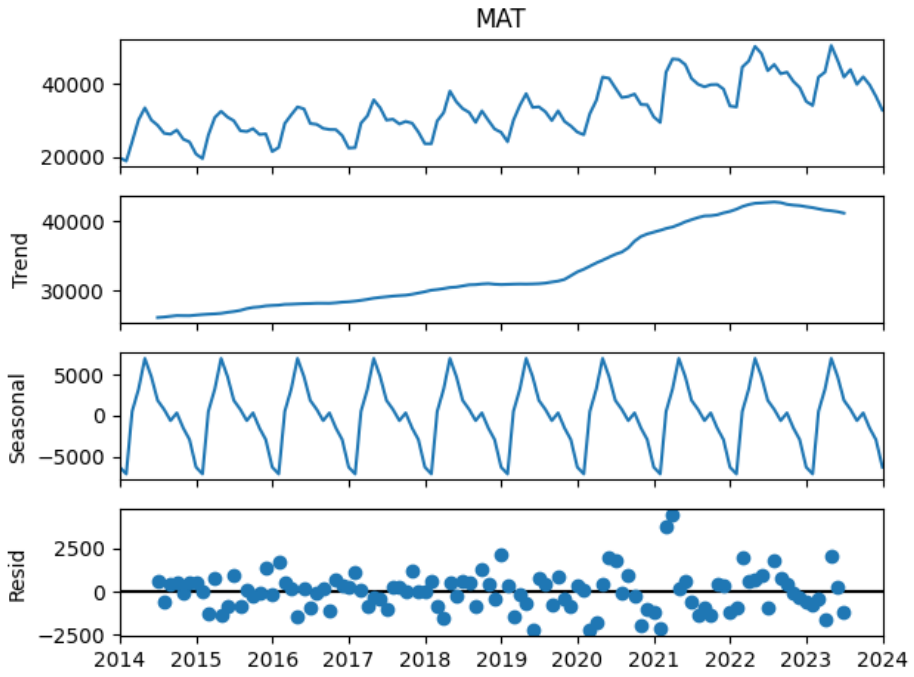
```
Wyniki testów stacjonarności dla szeregu drugiego (df_cpi)
Test Dickeya-Fullera:
Statystyka testowa: -11.926666421331172
P-value: 4.888862846495087e-22
Liczba opóźnień: 12
Liczba obserwacji: 106
Wartości krytyczne: {'1%': -3.4936021509366793, '5%': -2.8892174239808703, '10%': -2.58153320754717}
Odrzucamy hipotezę zerową - szereg jest stacjonarny.

Test Phillipsa-Perrona:
Statystyka testowa: -30.404921222700487
P-value: 0.0
Odrzucamy hipotezę zerową - szereg jest stacjonarny.

Test KPSS:
Statystyka testowa: 0.0894943196136539
P-value: 0.1
Wartości krytyczne: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Brak podstaw do odrzucenia hipotezy zerowej - szereg jest stacjonarny.
```

Rysunek 3.14 Testy stacjonarności dla trzeciego szeregu czasowego po różnicowaniu.

Dekompozycja trzeciego szeregu czasowego

Dekompozycja trzeciego szeregu czasowego (rysunek 3.15) dla sprzedaży detalicznej materiałów oraz sprzętu i artykułów ogrodniczych, wskazuje na:

- trend wzrostowy na przestrzeni niemalże całego analizowanego okresu, z wyraźnym przyspieszeniem w latach 2020-2022 i delikatnym spowolnieniem od końca 2022 roku,
- widoczną, istotną sezonowość roczną, wynikającą z cyklicznych czynników, takich jak zmiany popytu, czy inne zdarzenia specyficzne dla danej branży,
- reszty oscylują wokół zera, ale praktycznie w całym analizowanym okresie widoczne są znaczne odchylenia.



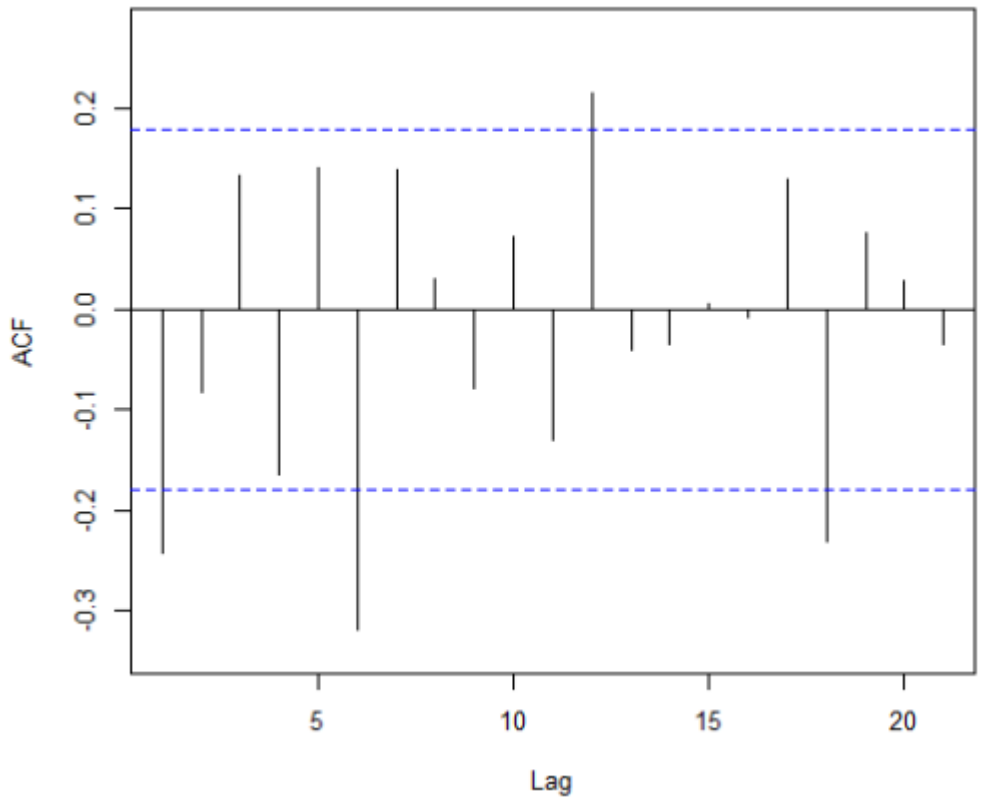
Rysunek 3.15 Wykresy po dekompozycji trzeciego szeregu czasowego.

Ogólnie rzecz biorąc, szereg czasowy sprzedaży detalicznej materiałów oraz sprzętu ogrodniczego cechuje się trendem wzrostowym i wyraźnym wpływem sezonowości, co może być przydatne przy analizie danych i przewidywaniu przyszłych obserwacji.

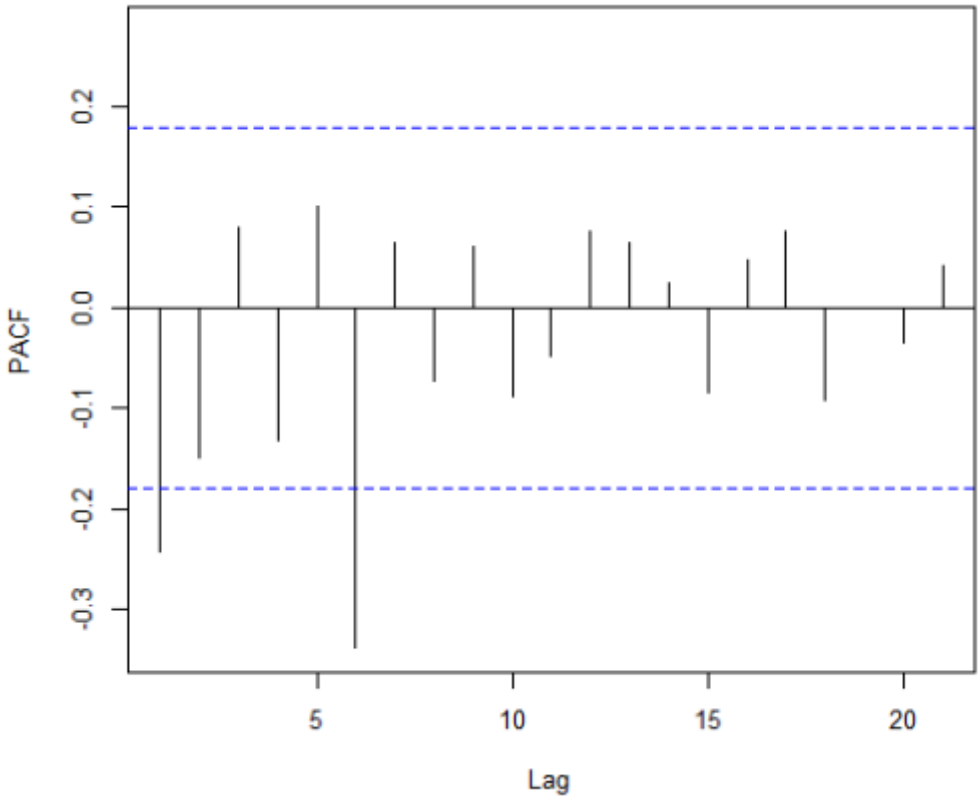
3.2 Dobór modelu ARIMA

Zanim przejdziemy do doboru modelu ARIMA, przyjrzyjmy się wykresom funkcji autokorelacji (ACF) i autokorelacji cząstkowej (PACF) po sprowadzeniu szeregu do postaci stacjonarnej. Wykresy te są kluczowe przy doborze modelu. Na ich podstawie można wstępnie określić wartość parametrów *p* (bazując na postaci funkcji PACF) i *q* (na podstawie funkcji ACF), które reprezentują odpowiednio rząd autoregresji (AR) i rząd średniej ruchomej (MA) w modelu ARIMA.

Dla pierwszego szeregu czasowego funkcje ACF i PACF przedstawiono odpowiednio na rysunku 3.16 i 3.17. Zarówno na wykresie ACF, jak i PACF, wartości dla większości lagów znajdują się w obrębie przedziałów ufności. Wykres funkcji PACF (rysunek 3.17) wskazuje na rząd autoregresji $p = 6$,a więc sugeruje dobór modelu ARIMA(6, 1, 0). W oparciu o wykres funkcji ACF (rysunek 3.16), wnioskujemy natomiast, że rząd średniej ruchomej $q = 18$, co sugeruje dobór modelu ARIMA(0, 1, 18). Modele te mają wysokie rzędy, dlatego szukamy modelu mieszanego korzystając z funkcji auto_arima. Otrzymujemy model mieszany ARIMA(1,0,1).



Rysunek 3.16 Wykres ACF po sprowadzeniu pierwszego szeregu czasowego do postaci stacjonarnej.



Rysunek 3.17 Wykres PACF po sprowadzeniu pierwszego szeregu czasowego do postaci stacjonarnej.

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. Nie ma też potrzeby ręcznego różnicowania danych przed użyciem **auto_arima**, ponieważ model ten automatyzuje proces wyboru najlepszego poziomu różnicowania. Poniżej przedstawiono fragment kodu oraz wyniki działania kodu (rysunek 3.18), w którym zastosowano **auto_arima** celem doboru modelu dla analizowanych szeregów czasowych. Z racji na występującą znaczną sezonowość w trzecim szeregu, dodajemy informacje, o występowaniu sezonowości co 12 miesięcy.

```
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,
seasonal=True, m=12)
```

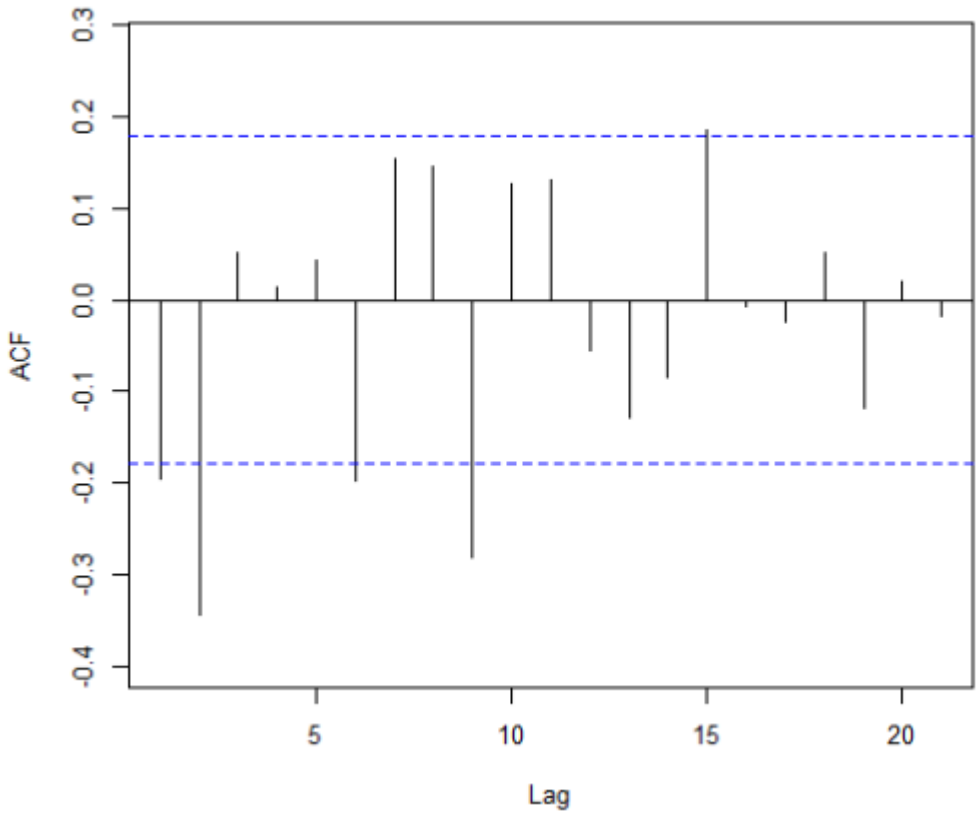
```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.14 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=547.926, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=432.640, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=475.087, Time=0.02 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=554.751, Time=0.01 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=431.613, Time=0.03 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=432.770, Time=0.04 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=433.313, Time=0.05 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=431.318, Time=0.03 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=433.304, Time=0.04 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=465.956, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=429.732, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=478.052, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=431.365, Time=0.01 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=431.731, Time=0.04 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=431.728, Time=0.03 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=468.006, Time=0.02 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=430.094, Time=0.02 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=inf, Time=0.10 sec

Best model:  ARIMA(1,0,1)(0,0,0)[0]
Total fit time: 0.680 seconds
```

Rysunek 3.18 Dobór parametrów modelu ARIMA dla pierwszego szeregu czasowego.

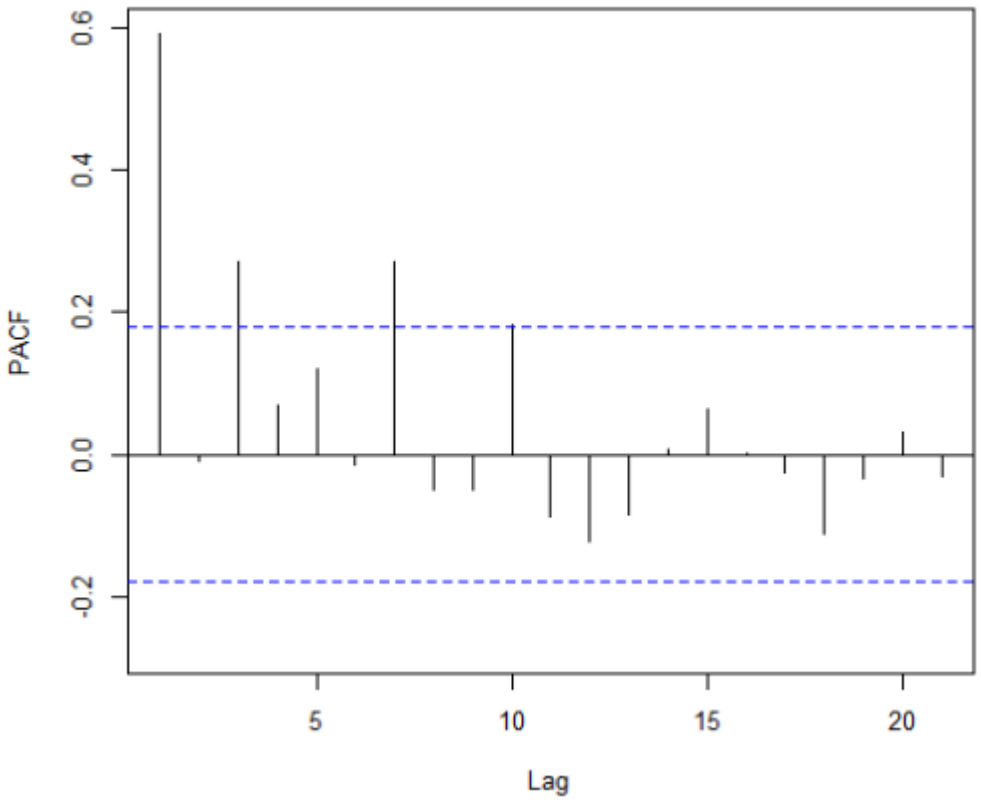
Automatyczne dopasowanie modelu ARIMA do pierwszego szeregu czasowego wskazało, że najlepszym modelem jest ARIMA(1,0,1), osiągając wartość AIC = 429.732, Model ten uwzględnia proces autoregresji rzędu pierwszego ($p = 1$) oraz proces średniej

ruchomej rzędu pierwszego ($q = 1$). Inne modele, takie jak ARIMA(0,0,1), ARIMA(2,0,1), miały wyższe błędy dopasowania, co oznacza, że zmiana w strukturze modelu, czy zwiększenie wartości parametrów nie poprawiło znacząco dopasowania. Ponadto, model ARIMA(2,0,2) uzyskał wartość AIC = inf (rysunek 3.18), co oznacza, że nie był w stanie dopasować się do danych. Może to sugerować problemy z konwergencją lub nadmierną złożonością modelu. Zatem model ARIMA(1,0,1) oferuje dobre dopasowanie przy jednoczesnym zachowaniu prostoty i efektywności w modelowaniu pierwszego szeregu.

W przypadku drugiego szeregu czasowego wykresy funkcji autokorelacji (ACF) i autokorelacji cząstkowej (PACF) przedstawiono na rysunkach 3.19 i 3.20. Na wykresie funkcji PACF (rysunek 3.20) ostatnie opóźnienie, które przekracza przedział ufności wskazuje na rząd autoregresji $p = 7$, co sugeruje dobór modelu ARIMA(7, 2, 0). W oparciu o wykres funkcji ACF (rysunek 3.19), wnioskujemy natomiast, że rząd średniej ruchomej $q = 2$, co sugeruje dobór modelu ARIMA(0, 2, 2). Skorzystamy z funkcji auto_arima w celu znalezienia modelu mieszanego. Otrzymaliśmy model mieszany ARIMA(0,2,2).



Rysunek 3.19 Wykres ACF po sprowadzeniu drugiego szeregu czasowego do postaci stacjonarnej.



Rysunek 3.20 Wykres PACF po sprowadzeniu drugiego szeregu czasowego do postaci stacjonarnej.

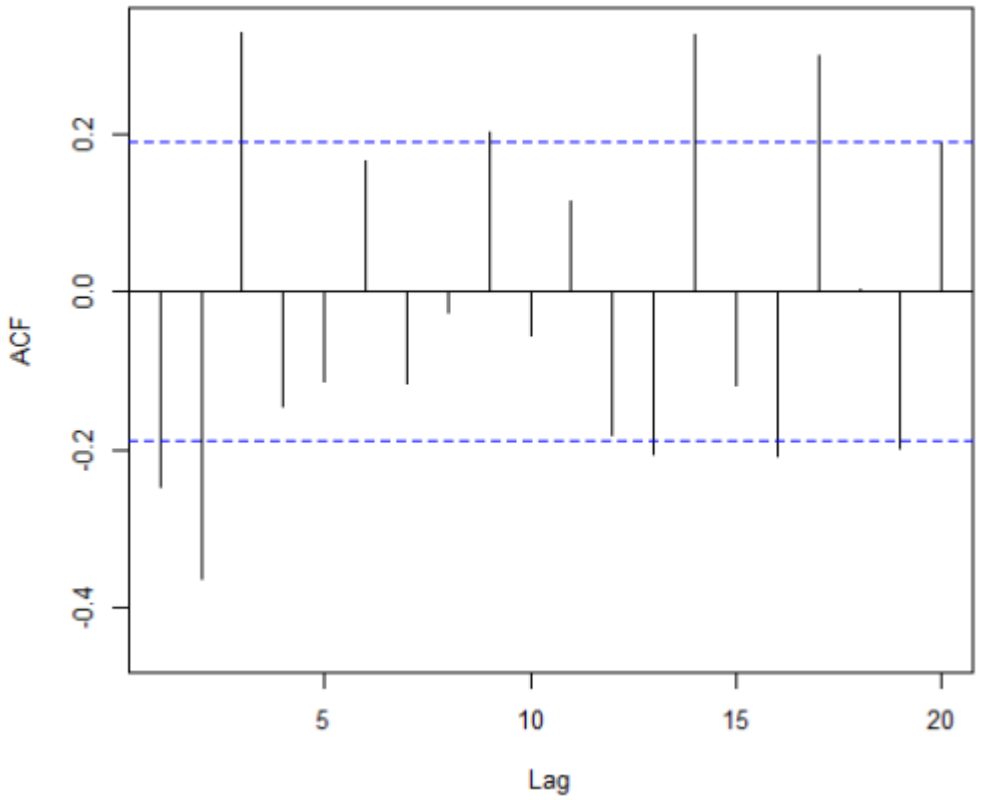
Automatyczne dopasowanie modelu ARIMA do drugiego szeregu czasowego wskazało, że najlepszym modelem jest ARIMA(0,2,2), osiągając wartość AIC = 229.137 (rysunek 3.21). Model ten uwzględnia proces średniej ruchomej rzędu drugiego ($q = 2$) oraz dwa różnicowania ($d = 2$). Jest to zgodne ze spostrzeżeniem, że dane wymagają podwójnego różnicowania, aby sprowadzić szereg do postaci stacjonarnej.

```
Performing stepwise search to minimize aic
 ARIMA(2,2,2)(0,0,0)[0]             : AIC=231.555, Time=0.05 sec
 ARIMA(0,2,0)(0,0,0)[0]             : AIC=261.355, Time=0.02 sec
 ARIMA(1,2,0)(0,0,0)[0]             : AIC=258.764, Time=0.01 sec
 ARIMA(0,2,1)(0,0,0)[0]             : AIC=243.043, Time=0.02 sec
 ARIMA(1,2,2)(0,0,0)[0]             : AIC=229.572, Time=0.04 sec
 ARIMA(0,2,2)(0,0,0)[0]             : AIC=229.137, Time=0.03 sec
 ARIMA(0,2,3)(0,0,0)[0]             : AIC=229.730, Time=0.03 sec
 ARIMA(1,2,1)(0,0,0)[0]             : AIC=235.057, Time=0.03 sec
 ARIMA(1,2,3)(0,0,0)[0]             : AIC=231.547, Time=0.06 sec
 ARIMA(0,2,2)(0,0,0)[0] intercept   : AIC=230.883, Time=0.07 sec

Best model:  ARIMA(0,2,2)(0,0,0)[0]
Total fit time: 0.360 seconds
```

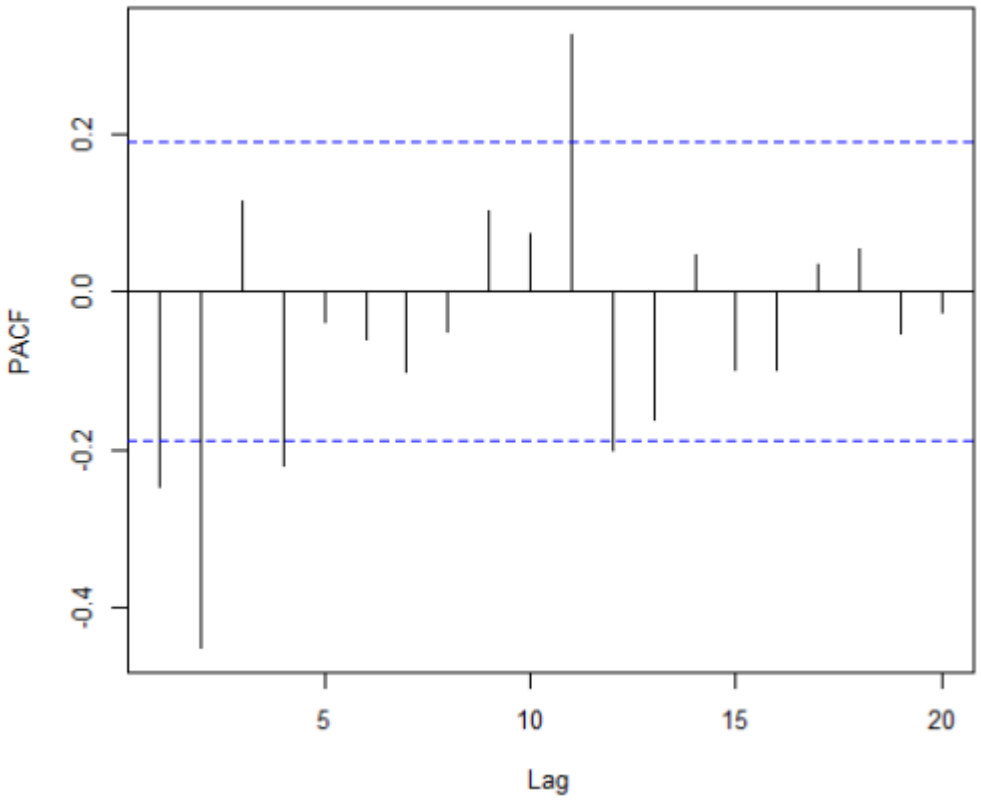
Rysunek 3.21 Dobór parametrów modelu ARIMA dla drugiego szeregu czasowego.

Modele takie jak ARIMA(1,2,2), ARIMA(0,2,3), czy ARIMA(0,2,1) dawały wyższe wartości AIC. Zatem zwiększenie wartości parametrów lub wprowadzenie zmian w strukturze modelu nie poprawiło dopasowania.

Dla trzeciego szeregu czasowego wykresy funkcji ACF i PACF przedstawiono na rysunkach 3.22 i 3.23. Na wykresie funkcji PACF (rysunek 3.23) pierwsze dwa opóźnienia przekraczają przedział ufności, co wskazuje na rząd autoregresji $p = 11$. Na podstawie wykresu funkcji ACF (rysunek 3.22) rząd średniej ruchomej to $q = 12$. Ponadto zauważalne są roczne wahania sezonowe. Sprowadzenie trzeciego szeregu czasowego do postaci stacjonarnej wymagało jednokrotnego różnicowania ($d = 1$). Zatem przewidywany model to ARIMA(2,1,1) z dodatkowymi czynnikami sezonowymi.



Rysunek 3.22 Wykres ACF po sprowadzeniu trzeciego szeregu czasowego do postaci stacjonarnej.



Rysunek 3.23 Wykres PACF po sprowadzeniu trzeciego szeregu czasowego do postaci stacjonarnej.

Automatyczne dopasowanie modelu ARIMA do trzeciego szeregu czasowego (rysunek 3.24) wskazało, że najlepszym modelem jest ARIMA(2,1,2) Rozważane są różne kombinacje parametrów (p, d, q) dla części nie-sezonowej oraz (P, D, Q) dla komponentu sezonowego z okresem m=12. Po analizie różnych konfiguracji najlepszym modelem okazuje się ARIMA(0,0,3)(1,1,1)[12] z uwzględnieniem sezonowości i trendu, co wskazuje na model z trzema parametrami autoregresji w części nie-sezonowej oraz parametrami (1,1,1) w części sezonowej. Model ten osiąga najniższe AIC, co sugeruje jego najlepsze dopasowanie do danych.

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=0.81 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2014.386, Time=0.01 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=1985.480, Time=0.10 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=1974.418, Time=0.21 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2014.386, Time=0.01 sec
 ARIMA(0,0,1)(0,1,0)[12] intercept   : AIC=1974.895, Time=0.07 sec
 ARIMA(0,0,1)(1,1,1)[12] intercept   : AIC=1979.571, Time=0.14 sec
 ARIMA(0,0,1)(0,1,2)[12] intercept   : AIC=1981.052, Time=0.26 sec
 ARIMA(0,0,1)(1,1,0)[12] intercept   : AIC=1981.302, Time=0.08 sec
 ARIMA(0,0,1)(1,1,2)[12] intercept   : AIC=1978.095, Time=0.43 sec
 ARIMA(0,0,0)(0,1,1)[12] intercept   : AIC=2014.833, Time=0.05 sec
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=1978.116, Time=0.35 sec
 ARIMA(0,0,2)(0,1,1)[12] intercept   : AIC=1981.824, Time=0.11 sec
 ARIMA(1,0,0)(0,1,1)[12] intercept   : AIC=1984.478, Time=0.09 sec
 ARIMA(1,0,2)(0,1,1)[12] intercept   : AIC=1972.741, Time=0.42 sec
 ARIMA(1,0,2)(0,1,0)[12] intercept   : AIC=1978.785, Time=0.17 sec
 ARIMA(1,0,2)(1,1,1)[12] intercept   : AIC=1968.581, Time=0.31 sec
 ARIMA(1,0,2)(1,1,0)[12] intercept   : AIC=1978.753, Time=0.18 sec
 ARIMA(1,0,2)(2,1,1)[12] intercept   : AIC=1969.722, Time=0.71 sec
 ARIMA(1,0,2)(1,1,2)[12] intercept   : AIC=1969.814, Time=0.87 sec
 ARIMA(1,0,2)(0,1,2)[12] intercept   : AIC=1969.659, Time=0.48 sec
 ARIMA(1,0,2)(2,1,0)[12] intercept   : AIC=1975.640, Time=0.59 sec
 ARIMA(1,0,2)(2,1,2)[12] intercept   : AIC=inf, Time=1.68 sec
 ARIMA(0,0,2)(1,1,1)[12] intercept   : AIC=1972.017, Time=0.44 sec
...
 ARIMA(0,0,3)(1,1,1)[12]             : AIC=1968.328, Time=0.50 sec

Best model:  ARIMA(0,0,3)(1,1,1)[12]
Total fit time: 15.483 seconds
```

Rysunek 3.24 Dobór parametrów modelu ARIMA dla trzeciego szeregu czasowego.

Użycie funkcji **auto_arima** umożliwiło szybkie i skuteczne znalezienie najbardziej dostosowanych parametrów modelu ARIMA dla każdego spośród analizowanych szeregów czasowych, minimalizując wartości kryterium informacyjnego AIC. Współczynniki modelu dla poszczególnych szeregów zostaną zbadane pod względem ich istotności statystycznej w następnym rozdziale.

Rozdział 4 Prognozowanie szeregów czasowych

Modele LSTM (a także inne modele RNN) nie wymagają badania stacjonarności szeregu czasowego, czy sprowadzania szeregu czasowego do postaci stacjonarnej, gdyż potrafią efektywnie pracować na danych niestacjonarnych. Jest to jedna z ich kluczowych zalet, w porównaniu do klasycznych metod analizy szeregów czasowych, w których niejednokrotnie konieczna jest stacjonaryzacja danych. Jeżeli jednak model LSTM lub RNN nie dawałby 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. Aby umożliwić efektywne uczenie modelu oraz jego weryfikację, ważnym elementem obróbki danych jest również analiza długości szeregu czasowego oraz odpowiedni podział danych na zbiór treningowy i testowy. W poniższej analizie zastosowano podział w stosunku 8:2.

4.1 Prognozowanie metodą ARIMA

Metodą klasyczną, wykorzystywaną w tej pracy do prognozowania wszystkich trzech analizowanych szeregów czasowych, jest model ARIMA. Dobór modelu oraz testy stacjonarności zostały przeprowadzone we wcześniejszym rozdziale. Dokładne działanie kodu, krok po kroku, zostanie opisane dla pierwszego szeregu (jednomiesięcznej realnej stopy procentowej). Dla pozostałych dwóch szeregów przedstawione zostaną tylko wyniki, gdyż działanie kodu jest analogiczne.

Analiza dokładności dopasowania modelu ARIMA dla pierwszego szeregu

Do stworzenia modelu korzystamy z funkcji **ARIMA** z biblioteki **statsmodels**. Na wejściu wymaga ona podania danych treningowych szeregu czasowego oraz dobranych wcześniej parametrów modelu.

```
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()
```

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

SARIMAX Results

Dep. Variable:	MAT	No. Observations:	96
Model:	ARIMA(0, 0, 3)x(1, 1, [1], 12)	Log Likelihood	-760.585
Date:	Fri, 10 Jan 2025	AIC	1533.169
Time:	13:18:16	BIC	1547.754
Sample:	01-01-2014	HQIC	1539.032
	- 12-01-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.4641	0.099	4.692	0.000	0.270	0.658
ma.L2	0.0164	0.098	0.166	0.868	-0.176	0.209
ma.L3	0.1117	0.076	1.462	0.144	-0.038	0.262
ar.S.L12	0.9999	0.006	179.765	0.000	0.989	1.011
ma.S.L12	-0.9967	0.134	-7.434	0.000	-1.259	-0.734
sigma2	4.934e+06	2.45e-08	2.02e+14	0.000	4.93e+06	4.93e+06

Ljung-Box (L1) (Q):	4.74	Jarque-Bera (JB):	124.04
Prob(Q):	0.03	Prob(JB):	0.00
Heteroskedasticity (H):	1.80	Skew:	0.68
Prob(H) (two-sided):	0.13	Kurtosis:	8.79

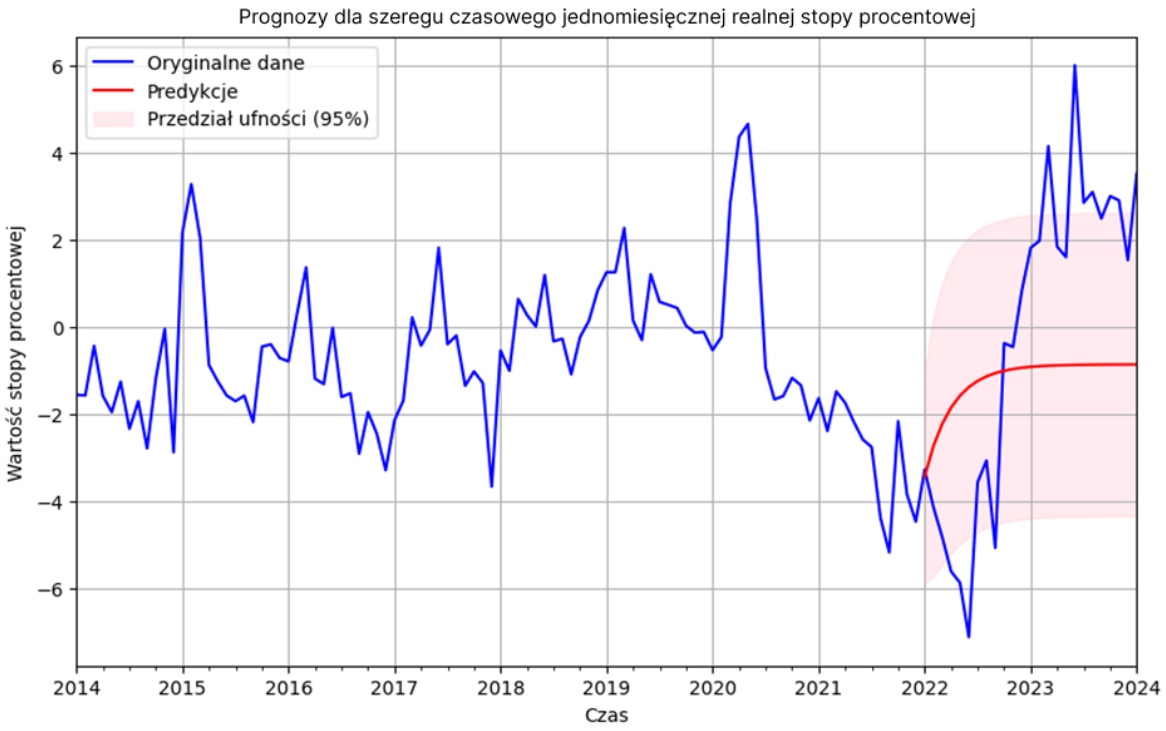
Rysunek 4.1 Model ARIMA dla pierwszego szeregu czasowego.

W modelu uwzględniono trzy składniki średniej ruchomej (MA) oraz jeden sezonowy składnik autoregresyjny (AR) i jeden sezonowy składnik średniej ruchomej (MA). Wyniki wskazują, że składniki MA.L1 oraz sezonowy MA.S.L12 są istotne statystycznie ($p = 0.000 < 0.05$), co oznacza ich silny wpływ na zmienną zależną (współczynniki odpowiednio 0.4641 i -0.9967). Pozostałe składniki (MA.L2, MA.L3 oraz AR.S.L12) są nieistotne ($p > 0.05$), co sugeruje, że ich wpływ na model jest przypadkowy. Kryteria informacyjne AIC (1533.169) i BIC (1547.754) wskazują na dobre dopasowanie modelu. Testy diagnostyczne pokazują brak istotnej autokorelacji reszt (Ljung-Box, $p = 0.47 > 0.05$), jednak test Jarque-Bera ($p = 0.000 < 0.05$) sugeruje znaczące odchylenia od normalności reszt. Ogólnie model opisuje dane wystarczająco dobrze.

Prognozowanie dla pierwszego szeregu czasowego

W tym kroku generowana jest prognoza na danych testowych za pomocą modelu ARIMA na poziomie ufności 95%. Prognoza jest gotowa do dalszej analizy i porównania z danymi rzeczywistymi.

```
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
```



Rysunek 4.2 Prognoza dla pierwszego szeregu czasowego jednomiesięcznej realnej stopy procentowej, generowana za pomocą modelu ARIMA.

Zgodnie z otrzymanymi rezultatami (rysunek 4.2) widzimy, że model przewidział początkową tendencję wzrostową danych. Jednak, ze względu na cykliczno-szumowy charakter szeregu, model w dalszym przedziale czasowym nie był w stanie uchwycić wzorców. W zamian dąży do wartości średniej obserwacji. Warto zauważyć, że przedział ufności zawiera większość danych testowych, nie rozszerza się, a zatem nie przewiduje dalszego trendu wzrostowego (lub spadkowego) stóp procentowych.

Analiza dokładności dopasowania modelu ARIMA dla drugiego szeregu

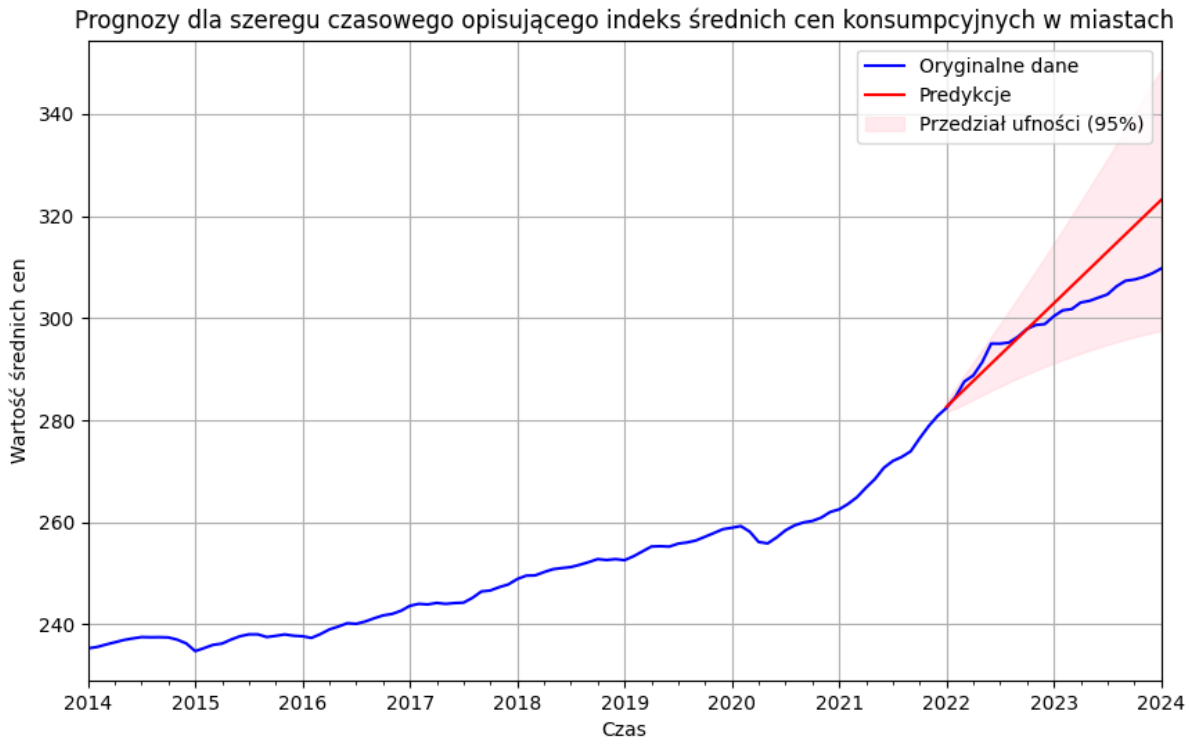
Analogicznie do pierwszego szeregu czasowego tworzymy model ARIMA dla szeregu drugiego (rysunek 4.3). Model uwzględnia dwa składniki średniej ruchomej (MA) i różnicowanie rzędu drugiego, wskazujące na potrzebę eliminacji trendu w danych. Oba składniki MA (ma.L1 = -0.3181 i ma.L2 = -0.4176) są istotne statystycznie ($p = 0.000 < 0.05$), co oznacza, że wpływają na zmienność CPI. Wartość sigma2 = 0.2764 jest również istotna ($p = 0.000 < 0.05$) i wskazuje na wariancję reszt. Kryteria AIC = 152.614, BIC = 160.244 sugerują dobre dopasowanie modelu do danych.

SARIMAX Results

Dep. Variable:		CPI		No. Observations:		96
Model:		ARIMA(0, 2, 2)		Log Likelihood		-73.307
Date:		Fri, 20 Dec 2024		AIC		152.614
Time:		17:01:59		BIC		160.244
Sample:		01-01-2014		HQIC		155.696
		- 12-01-2021				
Covariance Type:		opg				
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.3181	0.076	-4.181	0.000	-0.467	-0.169
ma.L2	-0.4176	0.090	-4.654	0.000	-0.594	-0.242
sigma2	0.2764	0.033	8.303	0.000	0.211	0.342

Ljung-Box (L1) (Q):	0.12	Jarque-Bera (JB):	4.29
Prob(Q):	0.73	Prob(JB):	0.12
Heteroskedasticity (H):	1.65	Skew:	-0.13
Prob(H) (two-sided):	0.17	Kurtosis:	4.01

Rysunek 4.3 Model ARIMA dla drugiego szeregu czasowego.



Rysunek 4.4 Prognoza dla drugiego szeregu czasowego opisującego indeks średnich cen konsumpcyjnych w miastach, generowana za pomocą modelu ARIMA.

Testy diagnostyczne wykazały brak autokorelacji reszt (Ljung-Box, $p = 0.73 > 0.05$) oraz istotnej heteroskedastyczności reszt ($p = 0.17 > 0.05$). Test Jarque-Bera ($p = 0.12 > 0.05$) pokazuje, że reszty mogą być zgodne z rozkładem normalnym. Model jest dobrze dopasowany do danych.

Prognozowanie dla drugiego szeregu czasowego

Na rysunku 4.4 widzimy, że model przewidział istniejący trend w danych. Dane testowe w pełni zawierają się w obszarze ufności, co oznacza, że predykcje są prawidłowe.

Analiza dokładności dopasowania modelu ARIMA dla trzeciego szeregu czasowego

Model ARIMA w tym szeregu, z racji na jego sezonowość o okresie 12 miesięcy, przyjmuje postać SARIMA$(2,1,2)\times(2,2,2)_{12}$.

```
model_arima_mat=ARIMA(train_mat, order=(2,1,2), seasonal_order=(2,2,2,12))
```

SARIMAX Results

Dep. Variable:	y	No. Observations:	121
Model:	SARIMAX(0, 0, 3)x(1, 1, [1], 12)	Log Likelihood	-977.164
Date:	Tue, 07 Jan 2025	AIC	1968.328
Time:	13:34:58	BIC	1987.167
Sample:	01-01-2014	HQIC	1975.968
	- 01-01-2024		

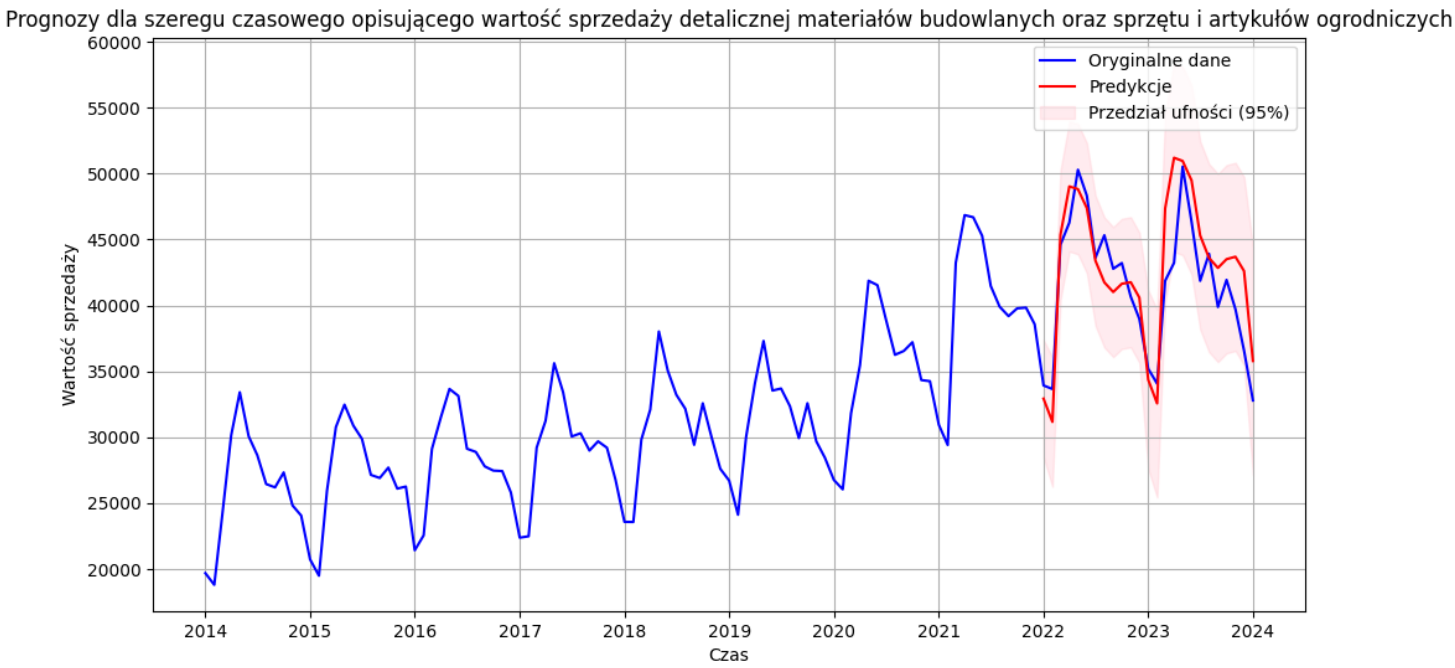
Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	1481.9971	483.950	3.062	0.002	533.473	2430.521
ma.L1	0.5328	0.066	8.091	0.000	0.404	0.662
ma.L2	0.0975	0.095	1.029	0.304	-0.088	0.283
ma.L3	0.2060	0.058	3.538	0.000	0.092	0.320
ar.S.L12	0.4044	0.226	1.787	0.074	-0.039	0.848
ma.S.L12	-0.6205	0.240	-2.583	0.010	-1.091	-0.150
sigma2	3.225e+06	3.56e+05	9.049	0.000	2.53e+06	3.92e+06

Ljung-Box (L1) (Q):	5.62	Jarque-Bera (JB):	58.40
Prob(Q):	0.02	Prob(JB):	0.00
Heteroskedasticity (H):	1.54	Skew:	0.39
Prob(H) (two-sided):	0.20	Kurtosis:	6.50

Rysunek 4.5 Model ARIMA dla trzeciego szeregu czasowego.

Składniki niesezonowe modelu AR (ar.L1 = -1.0727, ar.L2 = -0.9067) i MA (ma.L1 = 1.0665, ma.L2 = 0.7610) są istotne statystycznie ($p = 0.000 < 0.05$), co oznacza, że wpływają znacząco na modelowanie zmiennej zależnej. W części sezonowej tylko składnik ma.S.L12 (-1.5492) jest istotny ($p = 0.035 < 0.05$). Wartości kryteriów AIC (1292.761) i BIC (1313.125) wskazują na relatywnie skomplikowaną strukturę modelu. Test Ljung-Boxa ($p = 0.51 > 0.05$) sugeruje brak autokorelacji reszt, test Jarque-Bera = ($p = 0.00 < 0.05$) wskazuje, że reszty nie mają rozkładu normalnego, zaś test na heteroskedastyczność ($p = 0.00 < 0.05$) ujawnia zmienność wariancji reszt. Pomimo problemów z resztami, model jest dobrze dopasowany.



Prognozowanie dla trzeciego szeregu czasowego

Rysunek 4.6 Prognoza dla trzeciego szeregu czasowego opisującego wartość sprzedaży detalicznej materiałów budowlanych oraz sprzętu i artykułów ogrodniczych, generowana za pomocą modelu ARIMA.

Model prognozuje trend i sezonowość występujące w danych. Dane testowe znajdują się w obrębie obszaru zmienności, co oznacza, że predykcje są prawidłowe. Kształt przedziału ufności wskazuje na dalszą sezonowość i trend wzrostowy danych. Możemy stwierdzić, że model dokonał wystarczająco precyzyjnej prognozy.

4.2 Prognozowanie metodą LSTM

Prognozowanie szeregów czasowych można również przeprowadzić za pomocą metod uczenia maszynowego. Modelem, który zostanie użyty do wykonania prognoz, będzie opisany wcześniej LSTM. Dokładne działanie kodu, krok po kroku, zostanie opisane dla pierwszego szeregu czasowego, zawierającego dane o jednomiesięcznej realnejstopie procentowej. Dla pozostałych dwóch szeregów czasowych przedstawione zostaną jedynie wyniki, gdyż działanie kodu jest w ich przypadku analogiczne.

Podział danych na dane treningowe i testowe. Skalowanie danych

Z racji tego, że dane obejmują okres od 2014-01 do 2024-01, czyli 121 miesięcy, dane podzielono mniej więcej w stosunku 2:1. Dane z początkowych 96 miesięcy potraktowano jako zbiór treningowy, a pozostałe jako zbiór testowy.

```
# 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:]
```

Aby proces uczenia przebiegał sprawniej, za pomocą biblioteki **MinMaxScaler** dane zostały następnie zeskalowane do przedziału (0, 1).

```
# Normalizacja danych
scaler_rate = MinMaxScaler(feature_range=(0, 1))
scaled_data_rate = scaler_rate.fit_transform(df_rate)
```

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 4.1 Zestawienie fragmentu niezeskalowanych i zeskalowanych danych dla zbioru reprezentującego pierwszy szereg czasowy.

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	237.460	0.058900
2014-09-01	237.477	0.059269
2014-10-01	237.430	0.058249
2014-11-01	236.983	0.048544
2014-12-01	236.252	0.032674

Tabela 4.2 Zestawienie fragmentu niezeskalowanych i zeskalowanych danych dla zbioru reprezentującego drugi szereg czasowy.

DATE	MATERIALS	SCALED_MATERIALS
2014-01-01	19688	0.031619
2014-02-01	18801	0.000000
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	0.350800
2014-08-01	26446	0.272520
2014-09-01	26195	0.263573
2014-10-01	27329	0.303996
2014-11-01	24821	0.214594
2014-12-01	24056	0.187324

Tabela 4.3 Zestawienie fragmentu niezeskalowanych i zeskalowanych danych dla zbioru reprezentującego trzeci szereg czasowy.

Sekwencje dla modelu LSTM

Za pomocą funkcji **create_sequences** dane treningowe zostały podzielone na sekwencje o zadanej długości (**seq_length**). Każdej sekwencji wejściowej (tworzonej na podstawie kolejnych zestawów danych) funkcja przypisuje wartość wyjściową, która jest obserwacją w czasie bezpośrednio następującą po tej sekwencji. W ten sposób tworzony jest zbiór danych, na którym model LSTM będzie mógł uczyć się zależności czasowych i na tej podstawie prognozować przyszłe wartości szeregu czasowego. Parametr **sequence_length** określa, ile obserwacji ma zawierać każda sekwencja wejściowa. Dzięki tej funkcji, dane są odpowiednio przygotowywane do trenowania modelu LSTM, który analizuje wzorce w szeregach 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 formatu 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))
```

Powyższy fragment kodu służy do przygotowania danych do trenowania modelu LSTM, który jest wykorzystywany w analizie szeregów czasowych. Parametr *sequence_length = 12* oznacza, że każda sekwencja wejściowa będzie składała się z 12 obserwacji (np. wartości szeregu czasowego z 12 kolejnych miesięcy). Następnie, za pomocą funkcji **create_sequences**, generowane są sekwencje dla zbiorów treningowego i testowego. Funkcja ta, na podstawie zadanego rozmiaru sekwencji, tworzy dane wejściowe *X_train_rate*, *X_test_rate* oraz odpowiadające im wartości wyjściowe *y_train_rate*, *y_test_rate*. Każda sekwencja wejściowa składa się z 12 obserwacji, a odpowiadająca jej wartość wyjściowa to wartość obserwacji w kolejnym punkcie czasowym. Ostatecznie, dane wejściowe są przekształcane do formatu (*samples, timesteps, features*) wymaganego przez model LSTM, 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

Kolejnym krokiem jest zdefiniowanie modelu sieci neuronowej LSTM (Long Short-Term Memory) przy użyciu klasy **Sequential** z biblioteki **Keras**. Model jest używany do przewidywania wartości szeregów czasowych.

```
# 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()
```

Pierwsza linia kodu *model_rate = Sequential()* inicjalizuje model sekwencyjny, co oznacza, że warstwy sieci neuronowej są dodawane jedna po drugiej. Następnie, za pomocą linii kodu *model_rate.add(LSTM(100, activation = 'tanh', return_sequences = False, input_shape = (sequence_length, 1)))*, dodawana jest warstwa LSTM ze 100 jednostkami (neuronami), które analizują dane wejściowe oraz funkcją aktywacji *tanh* wprowadzającą 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 składająca się z pojedynczego neuronu. Oznacza to, że model ma przewidywać jedną wartość na podstawie przetworzonych danych. Na koniec, model jest kompilowany za pomocą optymalizatora Adam oraz funkcji straty MSE (średni błąd kwadratowy), co pozwala na jego efektywne trenowanie w zadaniu regresji.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100)	40,800
dense (Dense)	(None, 1)	101

```
Total params: 40,901 (159.77 KB)

Trainable params: 40,901 (159.77 KB)

Non-trainable params: 0 (0.00 B)
```

Rysunek 4.7 Struktura modelu LSTM dla pierwszego szeregu czasowego.

Model LSTM składa się z dwóch warstw: warstwy LSTM o 100 neuronach, która jest odpowiedzialna za uchwycenie zależności sekwencyjnych, oraz warstwy Dense z jednym neuronem do predykcji wyjściowej. Model zawiera łącznie 40901 trenowalnych parametrów, co oznacza, że wszystkie parametry mogą być optymalizowane podczas treningu. Brak parametrów nietrenowalnych wskazuje, że nie użyto np. zamrożonych wag lub warstw o stałych wartościach. Model jest relatywnie prosty, co może być korzystne dla wydajności, ale jego efektywność w predykcji będzie zależeć od jakości danych i poprawności hiperparametrów, takich jak liczba neuronów LSTM, czy funkcje aktywacji.

Trenowanie modelu

Na tym etapie kodu model LSTM jest trenowany na danych treningowych, aby nauczyć się wzorców i zależności w szeregach czasowych.

```
# 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)
```

Funkcja **model_rate.fit** jest używana do przeprowadzenia procesu treningowego, trwającego przez określoną liczbę epok (iteracji). W tym przypadku model będzie trenowany przez 250 epok. W epoce dane treningowe są przetwarzane, a model aktualizuje swoje wagi po każdej partii danych (mini-batch) o rozmiarze 12 próbek, co pomaga w efektywnym uczeniu się i minimalizacji funkcji straty. Dodatkowo, przy pomocy danych walidacyjnych (testowych) dokonywana jest ocena, 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 ocena modelu na danych, które nie były używane do jego trenowania. Wartości funkcji straty są wyświetlane po każdej epoce, co umożliwia śledzenie procesu uczenia.

Tworzenie prognozy

Za pomocą funkcji **model_rate.predict** wykonywana jest prognoza 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 prognozę na danych treningowych, na których model był wcześniej uczony. Prognoza te odzwierciedla, jak dobrze model dopasował się do danych, na których był trenowany.

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. Uzyskane prognozy są 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

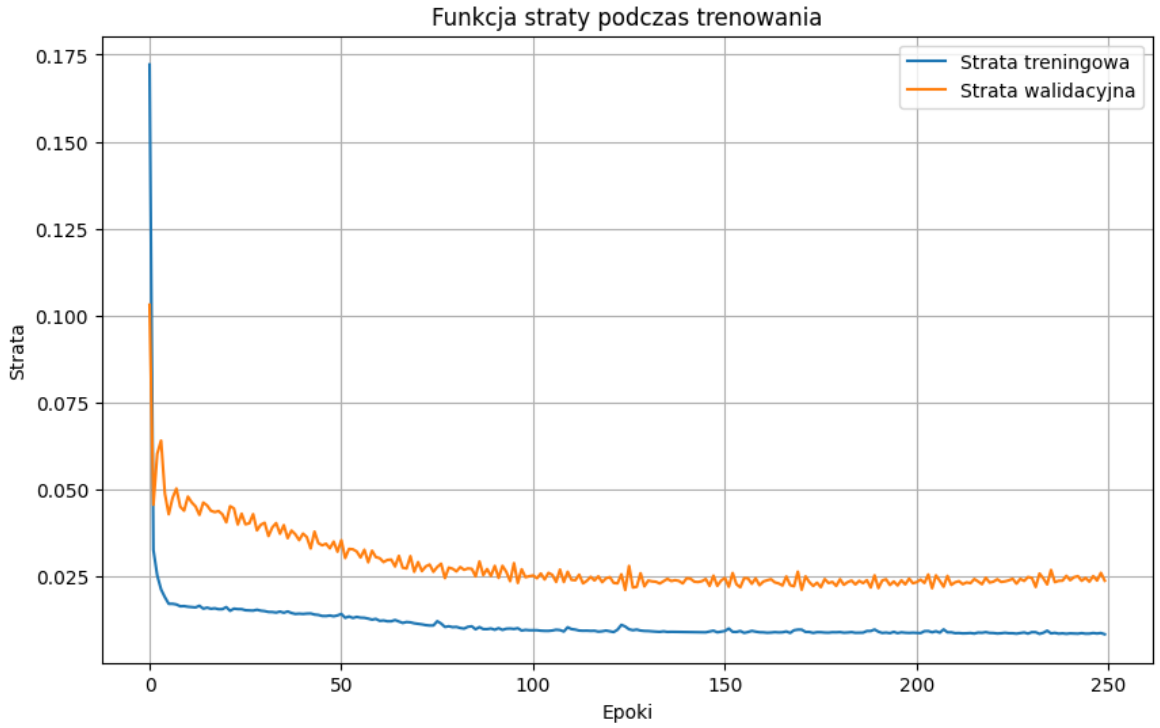
Na tym etapie dokonujemy odskalowania (ang. *inverse scaling*) danych, wcześniej przekształconych do skali innej niż oryginalna, aby model mógł je łatwiej przetworzyć. Do odtworzenia pierwotnych wartości, czyli przekształcenia prognoz i rzeczywistych wyników z powrotem do oryginalnej postaci, używamy skalera **scaler_rate**.

```
# 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]
```

Pierwsza część kodu dotyczy prognoz modelu. Do prognoz dla danych treningowych (*train_predict_rate*) i testowych (*test_predict_rate*) dodawana jest kolumna wypełniona zerami, aby dostosować dane do formatu oczekiwanego przez funkcję **inverse_transform**. Następnie funkcja ta przekształca dane z powrotem do ich oryginalnej skali, a [:, 0] służy do wyodrębnienia pierwszej kolumny, czyli odwróconych prognoz. Analogicznie postępujemy z wartościami rzeczywistymi *y_train_rate* i *y_test_rate*. Dzięki temu, zarówno prognozy, jak i dane rzeczywiste, są teraz w tej samej skali, co pozwala na ich porównanie w kontekście wartości rzeczywistych. Ten krok jest kluczowy dla interpretacji wyników i oceny skuteczności modelu.

Funkcja straty dla pierwszego szeregu czasowego

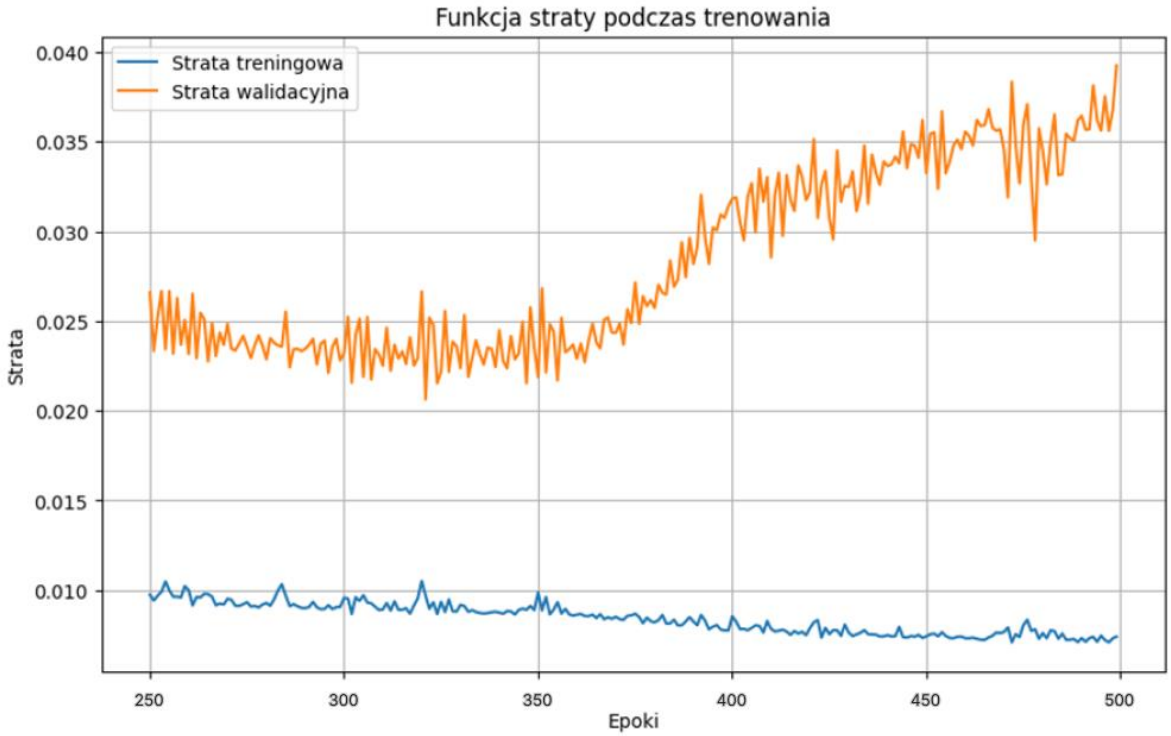
Na rysunku 4.8, przedstawiającym funkcję straty dla 250 epok, widać wyraźną różnicę pomiędzy stratą treningową, a walidacyjną. W pierwszych epokach procesu treningowego obydwie straty są duże. Po czym, strata treningowa spada w tempie znacznie szybszym od straty walidacyjnej. Z kolei strata walidacyjna, choć również spada, utrzymuje się na wyższym poziomie. Oznacza to, że model dobrze dopasowuje się do danych treningowych, ale jego zdolność generalizacji do danych testowych rozwija się wolniej. Można także zauważyć, że obydwie funkcje straty szybko dążą do konkretnej, minimalnej wartości. Zatem, aby uniknąć przeuczenia modelu, nie ma sensu ustalać zbyt dużej liczby epok.



Rysunek 4.8 Funkcja straty podczas trenowania pierwszego szeregu czasowego.

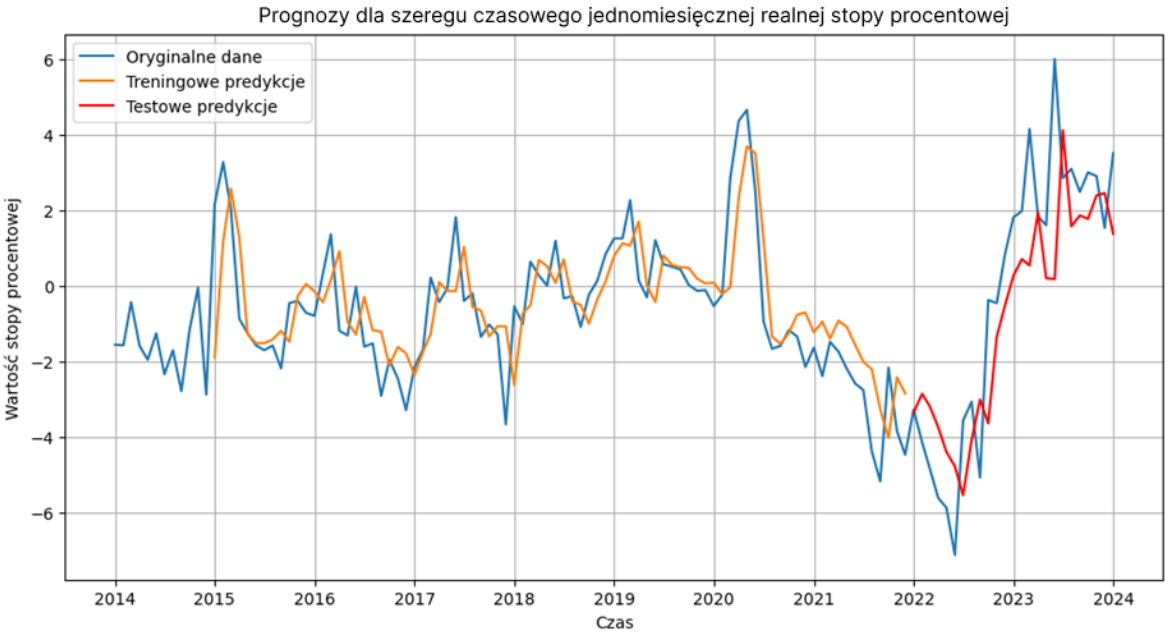
Analizując dane na rysunkach 4.8 i 4.9 można przyjąć, że liczba 250 epok dla pierwszego szeregu czasowego jest odpowiednia. Funkcja straty jest wówczas bliska minimum ale jednocześnie nie dochodzi do zjawiska przeuczenia modelu.

Moment rozpoczynającego się przeuczenia (*ang. overfitting*) modelu zilustrowano na rysunku 4.9. Widzimy, że w okolicach 350 epoki, pomimo malejącej straty treningowej, strata walidacyjna zaczyna rosnąć. Oznacza to, że po około 400 epokach, model zbytnio dopasowuje się do danych treningowych, a jego predykcje będą odbiegać od rzeczywistości dla danych testowych.



Rysunek 4.9 Funkcja straty podczas trenowania pierwszego szeregu czasowego (epoki od 250 do 500).

Wyniki prognoz dla pierwszego szeregu czasowego



Rysunek 4.10 Prognozy dla szeregu czasowego jednomiesięcznej realnej stopy procentowej generowane za pomocą modelu LSTM.

Jak łatwo zauważyć na rysunku 4.10 predykcja odwzorowała dane testowe w miarę dokładnie. Szereg ten jest cykliczny, przypomina błądzenie losowe i nie ma typowych wzorców, dlatego też jego prognozowanie jest utrudnione. Niemniej jednak, model LSTM

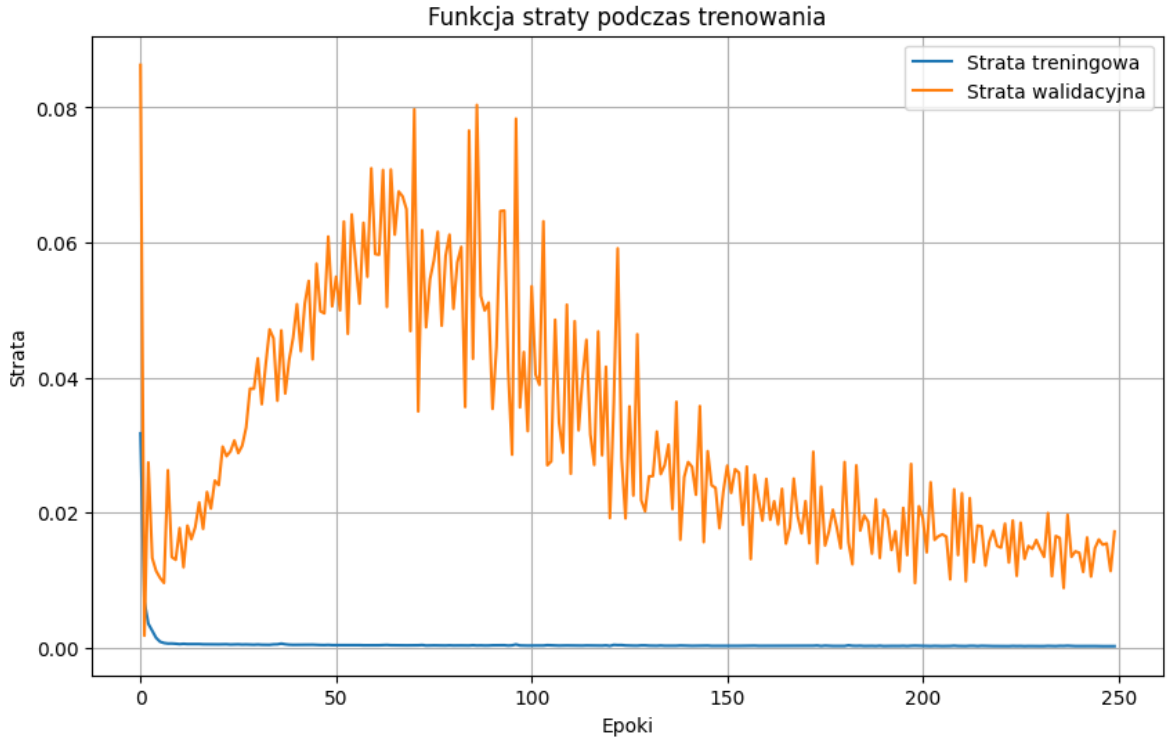
dosyć dobrze poradził sobie z tym zadaniem. Jednak, wartość błędu średniokwadratowego RMSE (ang. Root Mean Squared Error), będącego pierwiastkiem ze średniej kwadratów błędów (różnic pomiędzy rzeczywistymi, a przewidywanymi wartościami szeregu), to:

```
RMSE dla zbioru treningowego: 1.2059454785104926
RMSE dla zbioru testowego: 2.0212022373530893
```

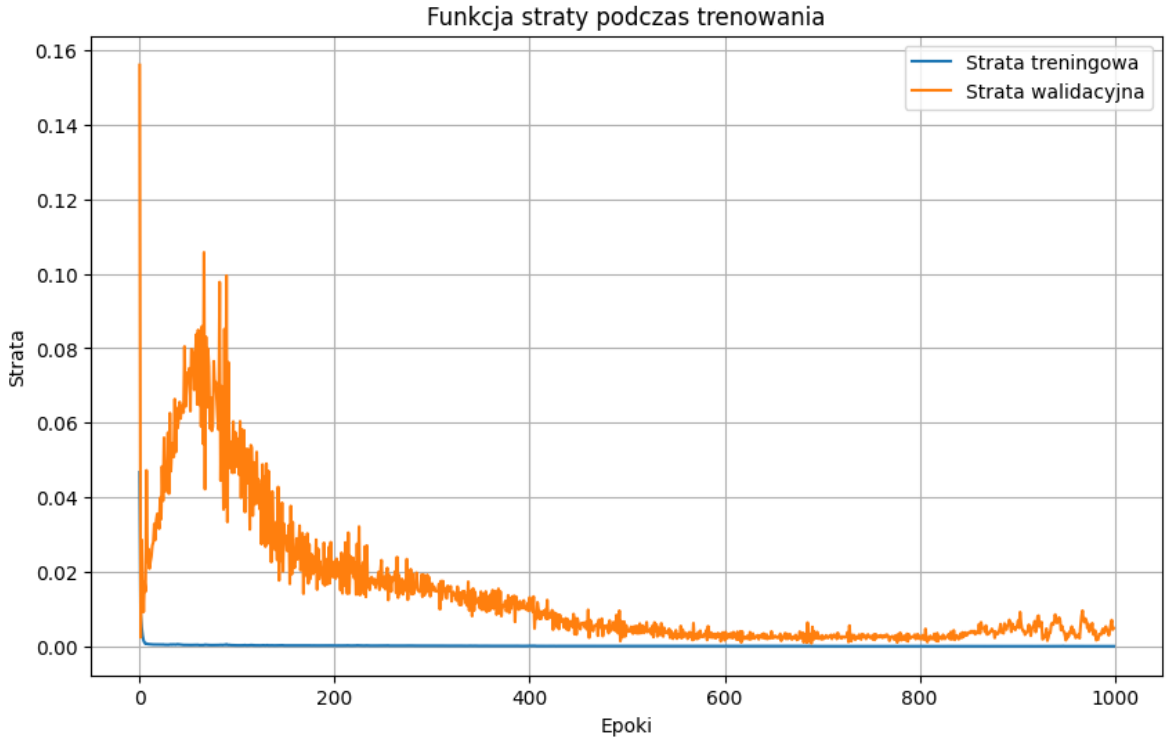
RMSE to miara określająca, jak dobrze model prognozuje wartości na podstawie danych rzeczywistych. Miara ta jest bardzo wrażliwa na wartości odstające, znajdujące się w analizowanym szeregu. Stąd wartość RMSE, szczególnie dla zbioru testowego, jest stosunkowo duża w porównaniu ze skalą zagadnienia (wartości stopy procentowej wahają się od około -8 do 6). W takim przypadku RMSE może mylnie wskazywać na słabą wydajności modelu.

Funkcja straty dla drugiego szeregu czasowego

Na rysunku 4.11 niemalże od 1 do około 80 epoki widoczny jest wyraźny wzrost straty walidacyjnej. Może zachodzić zjawisko niedouczenia (*ang. underfitting*), co oznacza, że model może być zbyt prosty, przez co uczy się niewłaściwych wzorców. W takim przypadku, aby zbadać, czy jesteśmy w stanie uzyskać możliwie najmniejsze straty, zwiększamy liczbę epok (rysunek 4.12).



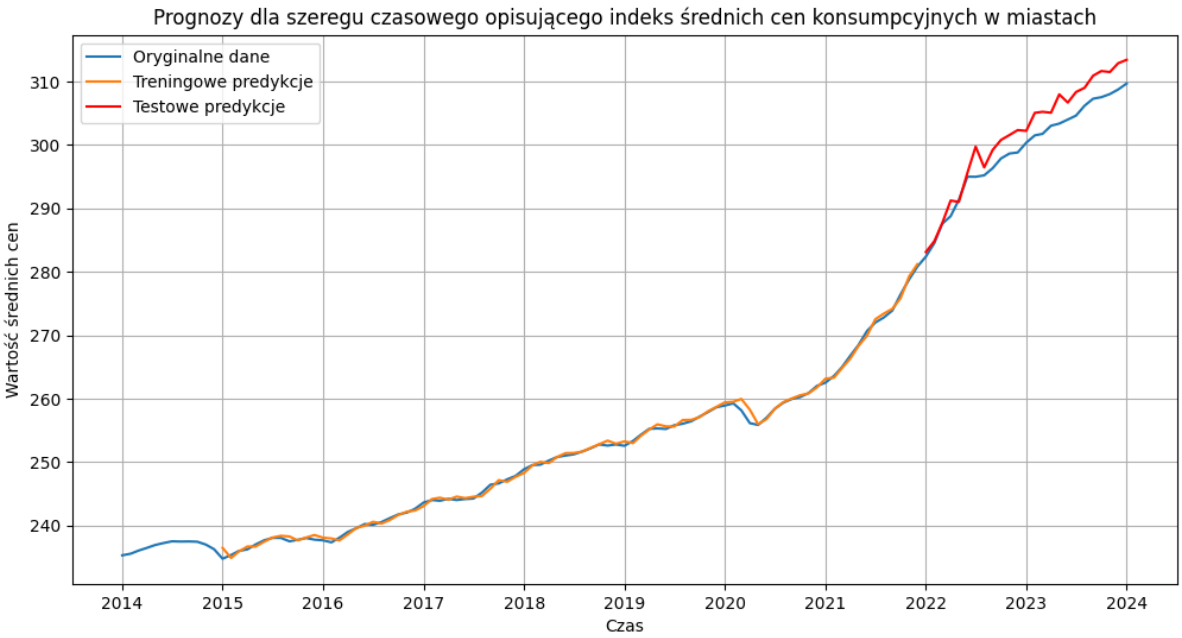
Rysunek 4.11 Funkcja straty podczas trenowania drugiego szeregu czasowego.



Rysunek 4.12 Funkcja straty podczas trenowania drugiego szeregu do 1000 epoki.

Widzimy, że około 900 epoki zaczęło występować zjawisko przeuczenia modelu. Na podstawie analizy wykresów umieszczonych na rysunkach 4.11 i 4.12 można wnioskować, że najbardziej optymalną liczbą epok dla tego modelu będzie 650-700.

Wyniki prognoz dla drugiego szeregu czasowego



Rysunek 4.13 Prognozy dla szeregu czasowego opisującego indeks średnich cen konsumpcyjnych w miastach, generowane za pomocą modelu LSTM.

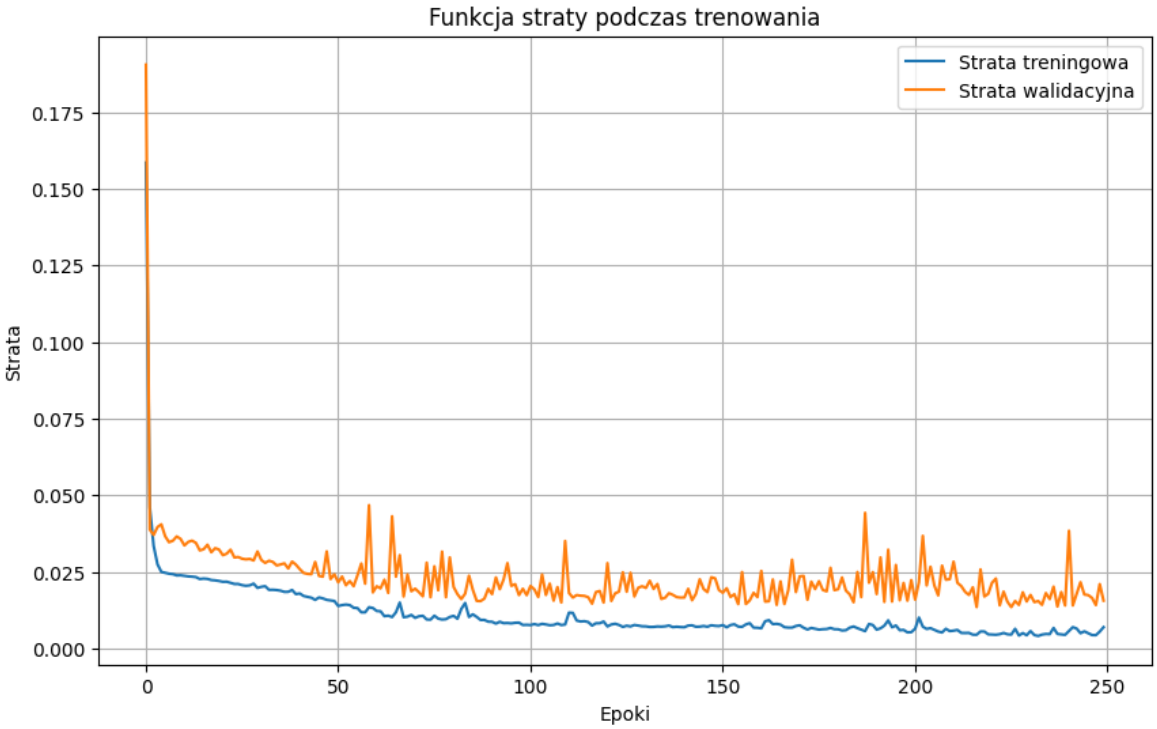
Jak można zauważyć, model LSTM dosyć dokładnie przewidział wartości danych treningowych i testowych. Jest to szereg z wyraźnym trendem, który został dobrze uchwycony. Obserwacje te potwierdzają wartości błędu RMSE:

```
RMSE dla zbioru treningowego: 0.5195647656760103
RMSE dla zbioru testowego: 3.001274198812061
```

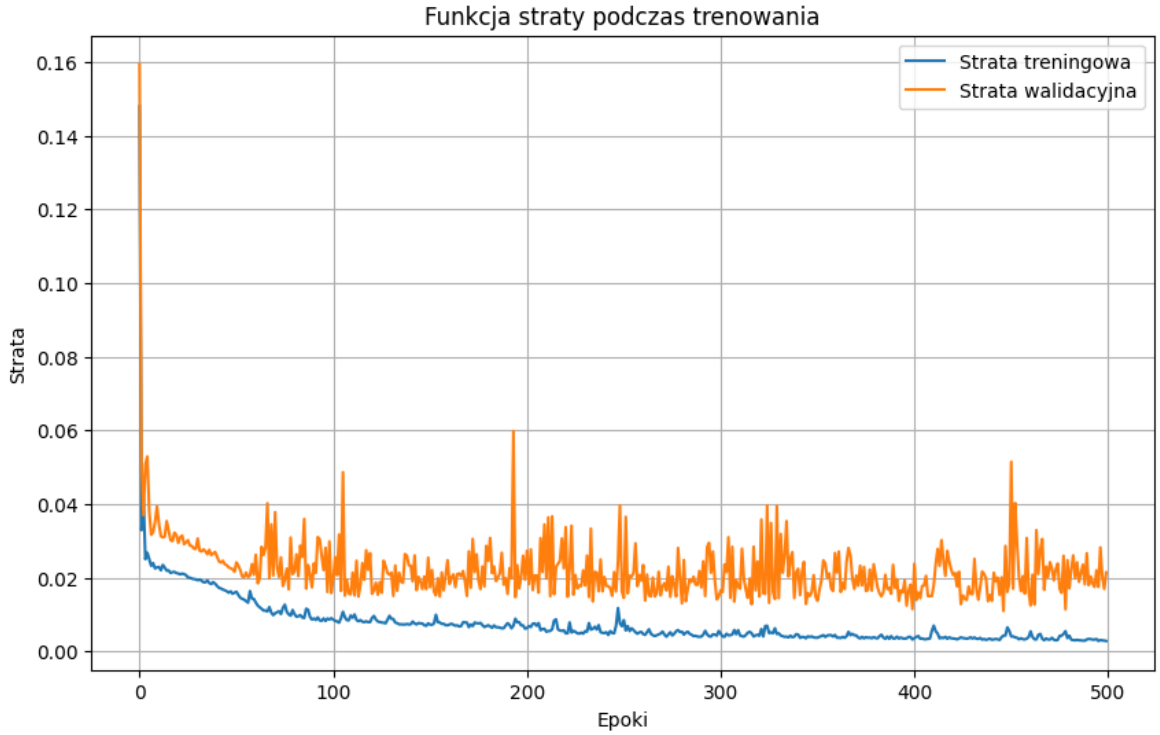
W drugim szeregu czasowym nie ma istotnych wartości odstających. Stąd błąd RMSE dla zbioru testowego na poziomie 3.0, w porównaniu ze skalą zagadnienia - wartości indeksu średnich cen konsumpcyjnych wahają się od około 230 do 320, jest niewielki. W ten sposób można potwierdzić dobrą wydajność modelu.

Funkcja straty dla trzeciego szeregu czasowego

W przypadku trzeciego szeregu czasowego funkcje straty dla zbiorów treningowego i testowego mają trend malejący (rysunek 4.14). Około 230 epoki można zauważyć stagnację funkcji straty. Zwiększając liczbę epok (rysunek 4.15), około 350 epoki zauważamy wzrost straty walidacyjnej, przy nieznacznie malejącej stracie treningowej. Takie zachowanie wskazuje na zjawisko przeuczenia się modelu. Analizując wyniki wnioskujemy, że najbardziej optymalną liczbą epok dla trzeciego szeregu czasowego jest 300 epok.

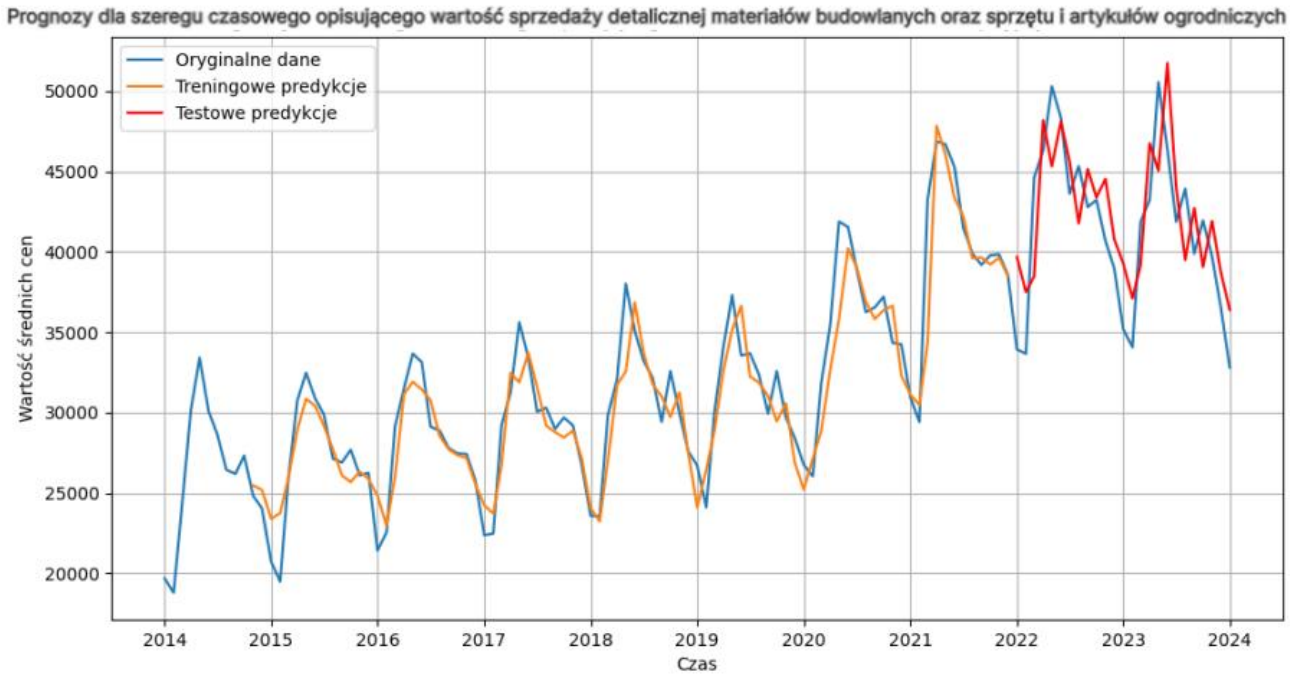


Rysunek 4.14 Funkcja straty podczas trenowania trzeciego szeregu czasowego.



Rysunek 4.15 Funkcja straty podczas trenowania trzeciego szeregu do 500 epoki.

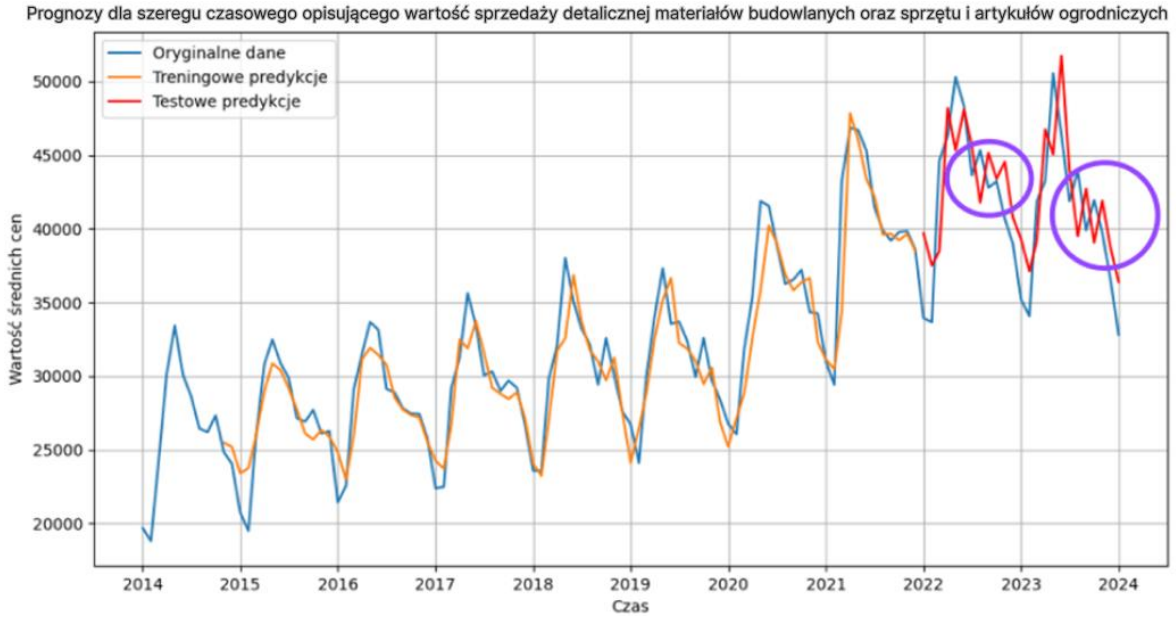
Wyniki prognoz dla szeregu trzeciego



Rysunek 4.16 Prognozy dla szeregu czasowego opisującego wartość sprzedaży detalicznej materiałów budowlanych oraz sprzętu i artykułów ogrodniczych, generowane za pomocą modelu LSTM.

Jak można zauważyć na rysunku 4.16 predykcja obrazuje przyszłe wartości danych testowych w dobry sposób. Model LSTM poradził sobie z wychwyceniem trendu

i sezonowości, charakteryzujących trzeci szereg czasowy. Warto też zwrócić uwagę na to, że w predykcji danych testowych po głównym skoku sezonowym uwidoczniono poprawnie dwa mniejsze skoki (rysunek 4.17). Skoki te nie były wyraźnie widoczne we wcześniejszych sezonach, a jednak model je przewidział. To potwierdza dokładne i wydajne działanie modeli LSTM.



Rysunek 4.17 Prognozy dla szeregu czasowego opisującego wartość sprzedaży detalicznej materiałów budowlanych oraz sprzętu i artykułów ogrodniczych, generowane za pomocą modelu LSTM z zaznaczonymi charakterystycznymi elementami.

Na dobrą jakość prognozy dla trzeciego szeregu czasowego i wydajność modelu LSTM wskazuje wartość błędu RMSE dla zbioru treningowego i testowego.

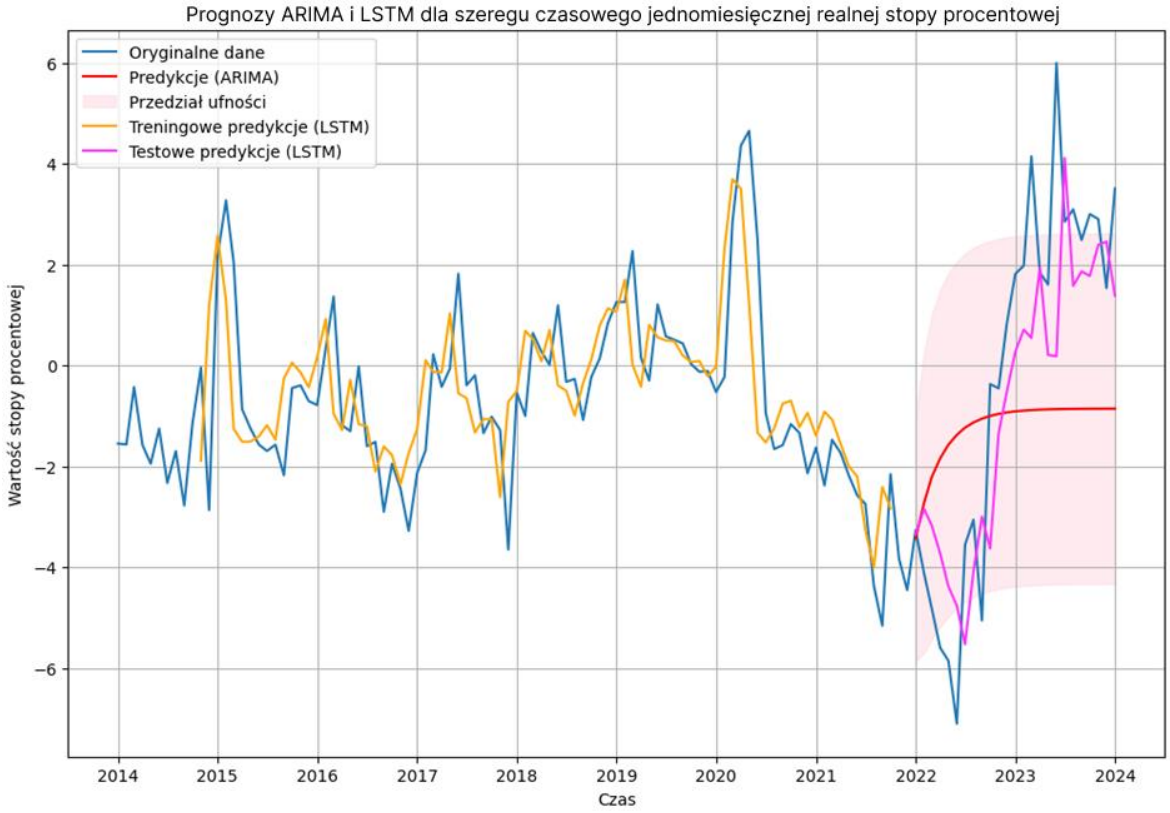
```
RMSE dla zbioru treningowego: 1816.3375664548153
RMSE dla zbioru testowego: 4430.803260037261
```

Błąd RMSE dla zbioru testowego na poziomie około 4430, w porównaniu ze skalą zagadnienia - wartości sprzedaży detalicznej materiałów budowlanych wahają się od około 20000 do 50000, nie jest wielki. Można zatem potwierdzić wystarczająco dobrą wydajność modelu.

4.3 Porównanie prognoz uzyskanych przy pomocy modeli ARIMA i LSTM

W przypadku pierwszego szeregu czasowego model ARIMA dobrze radzi sobie z uśrednieniem szumów oraz wyodrębnieniem trendu. Jednakże, prognozy są zbytnio wygładzone, co oznacza, że model nie reaguje na nagłe zmiany w danych. W przeciwieństwie do ARIMA, model LSTM jest w stanie uchwycić złożone, nieliniowe

wzorce i zależności w danych czasowych. Predykcje są bardziej dynamiczne i lepiej odwzorowują krótkoterminowe fluktuacje, co jest widoczne szczególnie na danych treningowych. Model LSTM trafniej prognozuje dane.



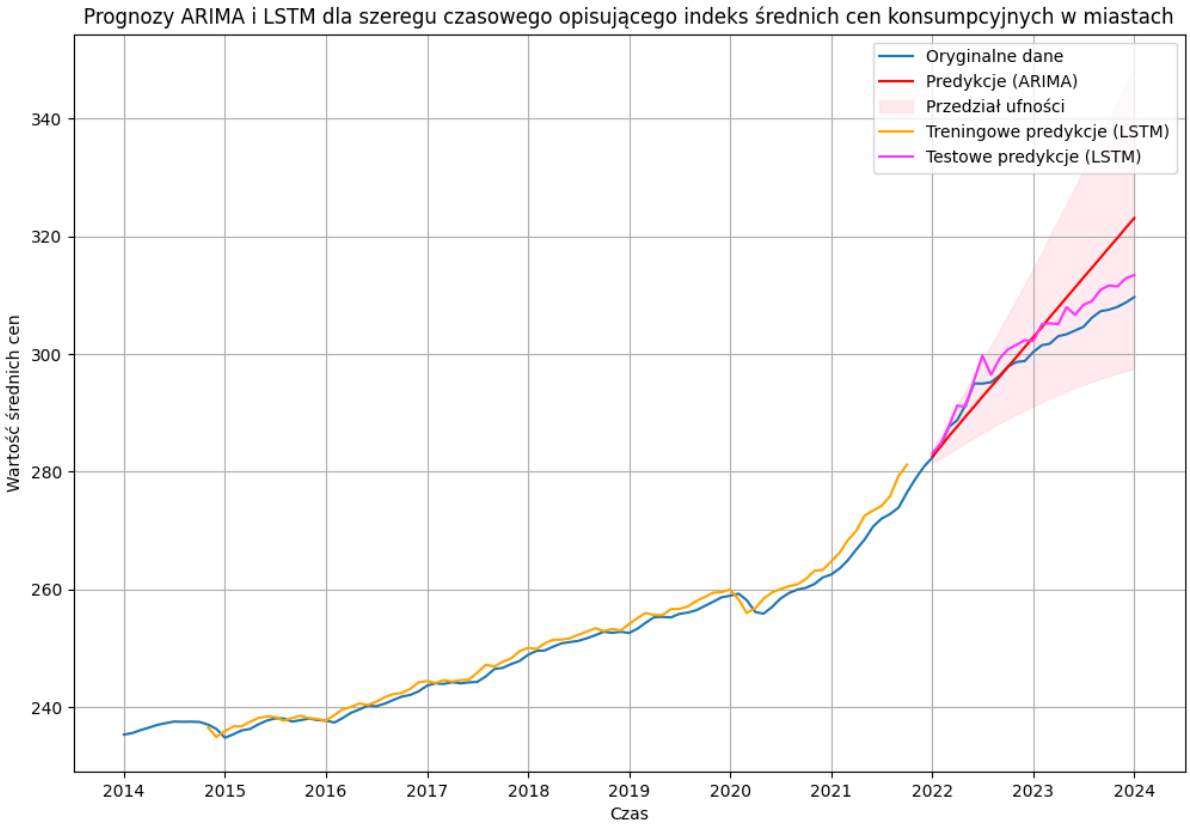
Rysunek 4.18 Porównanie prognoz dla szeregu czasowego jednomiesięcznej realnej stopy procentowej, generowanych za pomocą modeli ARIMA i LSTM.

Porównując obie metody (rysunek 4.18), możemy wnioskować, że ARIMA jest bardziej odpowiednia dla danych z wyraźnym trendem i umiarkowaną zmiennością. Natomiast LSTM dobrze sprawdza się w analizie skomplikowanych i nieregularnych danych. Wybór pomiędzy tymi metodami zależy od specyfiki danych.

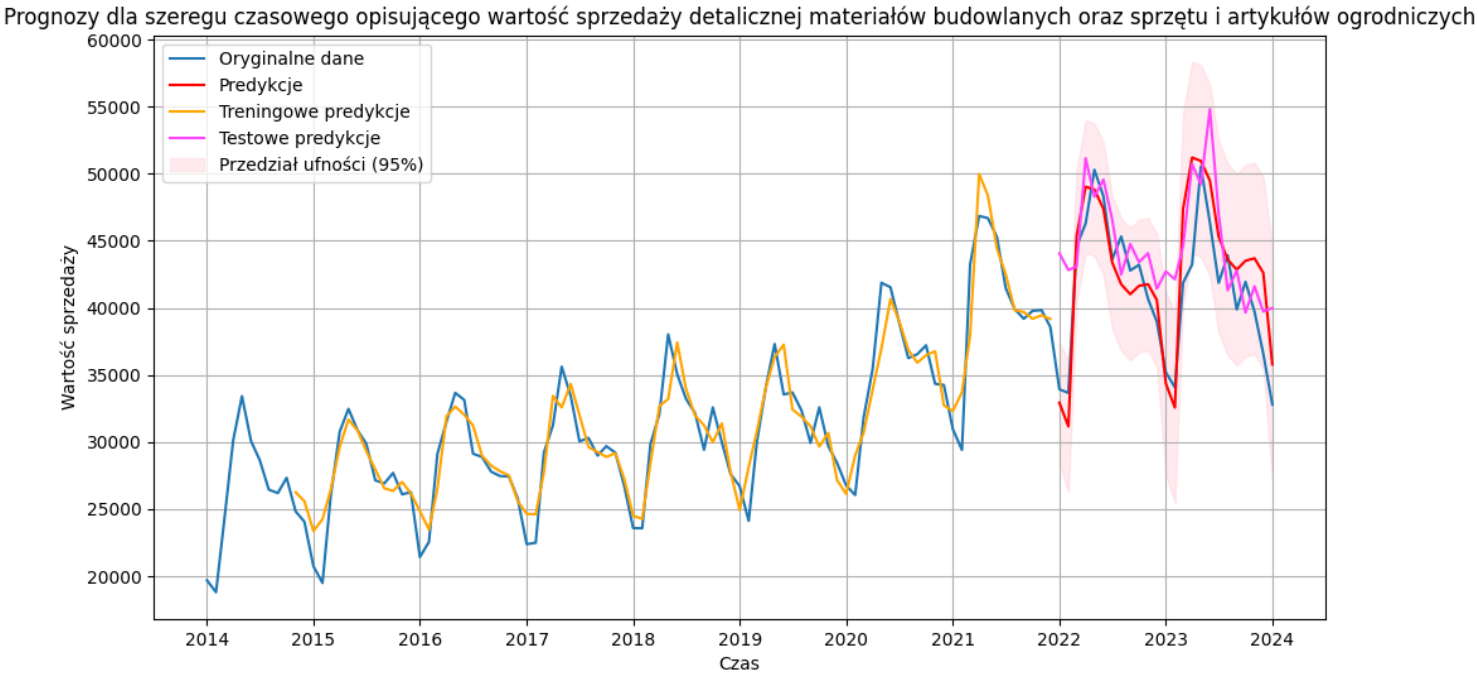
Dla drugiego szeregu czasowego (rysunek 4.19) model ARIMA dobrze odwzorowuje trend wzrostowy w danych. Jego predykcje są jednak mocno wygładzone, przez co nie uwzględniają niewielkich fluktuacji. Przedział ufności wyraźnie poszerza się w czasie, co odzwierciedla rosnącą niepewność prognozy. Model LSTM bardzo dobrze dopasowuje się do danych treningowych i w miarę skutecznie odwzorowuje dane testowe. Brak wizualizacji przedziału ufności oznacza, że trudniej ocenić wiarygodność predykcji w kontekście niepewności. Niemniej jednak, model LSTM trafniej prognozuje dane.

Model ARIMA wygenerował prognozę skupiając się na trendzie długoterminowym w danych i uwzględniając niepewność w formie przedziału ufności. Z kolei LSTM lepiej odwzorowuje szczegóły, co pozwala na dokładniejsze prognozy krótkoterminowe. Jednak,

bez wizualizacji niepewności wyniki mogą być mniej przejrzyste. Wybór odpowiedniego modelu zależy od potrzeby.



Rysunek 4.19 Porównanie prognoz dla szeregu czasowego opisującego indeks średnich cen konsumpcyjnych w miastach, generowanych za pomocą modeli ARIMA i LSTM.



Rysunek 4.20 Porównanie prognoz dla szeregu czasowego opisującego wartość sprzedaży detalicznej materiałów budowlanych oraz sprzętu i artykułów ogrodniczych, generowanych za pomocą modeli ARIMA i LSTM.

W przypadku trzeciego szeregu czasowego (rysunek 4.20) model ARIMA uchwycił trend i sezonowość danych. Obszar ufności odzwierciedla większą niepewność danych w dłuższym horyzoncie czasowym. Model LSTM doskonale odwzorowuje szczegóły, uwzględniając zarówno trend i sezonowość, jak i krótkoterminowe fluktuacje w danych. Model ten trafniej prognozuje dane.

Model ARIMA uwzględnia trend długoterminowy i ilustruje niepewność za pomocą przedziału ufności. Jest to szczególnie przydatne przy analizie stabilnych i bardziej przewidywalnych danych. Z kolei LSTM lepiej odwzorowuje złożone wzorce i zmiany krótkoterminowe, przez co jest adekwatny w przypadku danych o dużej zmienności.

Podsumowanie

Praca dotyczy prognozowania szeregów czasowych przy użyciu: klasycznego modelu ARIMA oraz nowoczesnego modelu LSTM opartego na uczeniu maszynowym. Głównym celem pracy było wskazanie mocnych i słabych stron obydwu modeli oraz ocena ich efektywności podczas prognozowania szeregów czasowych. Obie metody mają swoje zalety i wady, a ich wybór zależy od charakterystyki danych i celu analizy.

Model ARIMA jest przydatny w analizie danych z wyraźnym trendem i umiarkowaną zmiennością. Przedział ufności wizualizuje niepewność prognozy w miarę wydłużania horyzontu czasowego, co jest istotnym aspektem w kontekście podejmowania decyzji. ARIMA jest więc odpowiednim wyborem w przypadku analiz, gdzie stabilność i prostota prognozowania mają kluczowe znaczenie.

Model LSTM pozwala na uchwycenie złożonych, nieliniowych wzorców i zależności w danych. Dzięki temu lepiej radzi sobie z analizą krótkoterminowych fluktuacji oraz nieregularnych danych, co w porównaniu z ARIMA, czyni go bardziej elastycznym narzędziem. LSTM, choć jest precyzyjny w prognozach krótkoterminowych, może być trudniejszy w interpretacji i mniej stabilny. Niemniej jednak, jego zdolność do adaptacji sprawia, że jest wyjątkowo skuteczny w analizie niestandardowych danych, które nie wpisują się w proste wzorce trendów i sezonowości.

Podsumowując, dzięki elastyczności i zdolności do uchwycenia niestandardowych zależności, stosowanie modeli uczenia maszynowego, w tym LSTM, stanowi obiecujący kierunek rozwoju analizy danych czasowych. Takie modele pozwalają na tworzenie precyzyjnych prognoz, co czyni je niezwykle wartościowym narzędziem w szybko zmieniającym się świecie analityki danych.

Literatura

[1] Nielsen A., *Szeregi czasowe. Praktyczna Analiza i Predykcja z Wykorzystaniem Statystyki i Uczenia Maszynowego*. Helion, 2020.

[2] Suchwałko A., Zagdański A., *Analiza i Prognozowanie Szeregów Czasowych.* Wydawnictwo Naukowe PWN, Warszawa, 2015.

[3] Brockwell P.J., Davis R.A., *Introduction to Time Series and Forecasting*. Springer Nature Switzerland AG, 2016. https://doi.org/10.1007/978-3-319-29854-2

[4] Albeladi K., Zafar B., Mueen A., *Time series forecasting using LSTM and ARIMA*. International Journal of Advanced Computer Science and Applications (IJACSA), 14 (1), pp. 313–320, 2023. http://dx.doi.org/10.14569/IJACSA.2023.0140133

[5] Zhang M., *Time Series: Autoregressive models AR, MA, ARMA, ARIMA*. (Online presentation), University of Pittsburgh, 2018. https://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class16.pdf

[6] Shumway R.H., Stoffer D.S., *Time Series Analysis and Its Applications*, Springer International Publishing AG, 2017. https://doi.org/10.1007/978-3-319-52452-8

[7] Géron A., *Uczenie Maszynowe z Użyciem Scikit-Learn, Keras i TensorFlow*. Wydanie III, Helion, 2023.

[8] Agarwal H., Mahajan G., Shrotriya A., Shekhawat D., *Predictive Data Analysis: Leveraging RNN and LSTM Techniques for Time Series Dataset*. International Conference on Machine Learning and Data Engineering 2023, Procedia Computer Science 235, pp. 979–989, 2024. https://doi.org/10.1016/j.procs.2024.04.093

[9] Starmer J., *The StatQuest Illustrated Guide to Machine Learning*, StatQuest Publications, 2022.

[10] Wolter M., *Deep Learning*. (Wykład) Instytut Fizyki Jądrowej PAN, 2020. https://indico.ifj.edu.pl/event/503/attachments/1343/2069/DeepLearningLecture2020_1.pdf

[11] Horzyk, A., *Knowledge-based computational intelligence and data mining in Biomedicine*. Akademia Górniczo-Hutnicza w Krakowie, 2021. https://home.agh.edu.pl/~horzyk/lectures/ahdydkcidmb.php

[12] Hochreiter S., Schmidhuber J., *Long Short-Term Memory*. Neural Computation 9 (8), pp. 1735–1780, 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[13] Brownlee J., *Long Short-Term Memory Networks With Python*. *Develop Sequence Prediction Models with Deep Learning*. (E-Book) Machine Learning Mastery, 2017. https://machinelearningmastery.com/lstms-with-python/

[14] Hu J., Wang X., Zhang Y., Zhang D., Zhang M., Xue J., *Time series prediction method based on variant LSTM recurrent neural Network.* Neural Processing Letters 52, pp. 1485–1500, 2020. https://doi.org/10.1007/s11063-020-10319-3

[15] Rasamoelina A.D., Adjailia F., Sinčák P., *A Review of Activation Function for Artificial Neural Network*. 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), Herlany, Slovakia, 2020, pp. 281-286, doi: 10.1109/SAMI48414.2020.9108717.

[16] Smola A., Vishwanathan S.V.N., *Introduction to Machine Learning*. Cambridge University Press, 2008. https://alex.smola.org/drafts/thebook.pdf

[17] Werbos P.J., *Backpropagation through time: what it does and how to do it.* In Proceedings of the IEEE, 78 (10), pp. 1550-1560, 1990. Doi: 10.1109/5.58337.

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2025
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

STRESZCZENIE PRACY DYPLOMOWEJ

Tytuł: Prognozowanie szeregów czasowych: metody klasyczne i elementy uczenia maszynowego
Autor: Aldona Świrad, FS-DI 169854
Promotor: dr hab. Liliana Rybarska-Rusinek , prof. PRz
Słowa kluczowe: szeregi czasowe, analiza danych, predykcje, ARIMA, LSTM

Praca dotyczy prognozowania szeregów czasowych przy użyciu klasycznego modelu ARIMA oraz modelu LSTM opartego na uczeniu maszynowym. Celem pracy było porównanie efektywności tych modeli w analizie i predykcji wybranych szeregów czasowych (bez trendu i sezonowości, z trendem, z trendem i sezonowością). Zastosowano odpowiednie techniki doboru parametrów modeli, w odniesieniu do danych rzeczywistych. Wyniki wskazują jednoznacznie, że model LSTM pozwala na uchwycenie złożonych, nieliniowych wzorców w danych. Model ARIMA może natomiast stanowić odpowiedni wybór w przypadku, gdy prostota prognozowania ma przeważające znaczenie.

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2025

FACULTY OF MATHEMATICS AND APPLIED PHYSICS

DIPLOMA THESIS ABSTRACT

Title: Time series forecasting: classical methods and elements of machine learning
Author: Aldona Świrad, FS-DI 169854
Supervisor: dr hab. Liliana Rybarska-Rusinek, prof. PRz
Key words: time series, data analysis, prediction, ARIMA, LSTM

The thesis concerns forecasting time series using the classical ARIMA model and the LSTM model based on machine learning. The aim of the work was to compare the effectiveness of these models in the analysis and prediction of selected time series (without trend and seasonality, with trend, with trend and seasonality). Appropriate techniques for selecting model parameters, in relation to real data, were employed. The results clearly indicate that the LSTM model is capable of capturing complex, nonlinear patterns in the data. On the other hand, the ARIMA model may be a better choice when simplicity in prediction is crucial.