



POLITECHNIKA ŁÓDZKA

**Wydział Elektrotechniki, Elektroniki, Informatyki
i Automatyki**

Instytut Mechatroniki i Systemów Informatycznych

Praca dyplomowa magisterska

na temat:

**System prognozowania warunków meteorologicznych z
wykorzystaniem algorytmów eksploracji danych**

**(System for forecasting meteorological conditions using data mining
algorithms)**

Imię i Nazwisko: **Kinga Sochacka**
Nr albumu: **234005**
Specjalność: **Inteligentne systemy baz danych**
Kierunek: **Informatyka**

Opiekun pracy:
prof. dr hab. inż. **Adam Pelikant**

Łódź, wrzesień 2021

SPIS TREŚCI

1. Wstęp	5
2. Cel i zakres pracy	6
3. Podstawy teoretyczne	8
3.1. Eksploracja danych	8
3.2. Szeregi czasowe	9
3.3. Sztuczne sieci neuronowe	12
3.3.1. Rekurencyjne sieci neuronowe	14
3.3.2. Long Short – Term Memory Network	14
4. Wykorzystane technologie	16
4.1. Python	16
4.2. Java	18
4.3. Spring Framework	19
4.4. HTML	19
4.4.1. Bootstrap	20
4.5. Thymeleaf	20
4.6. Microsoft SQL Server	21
4.7. PyCharm i IntelliJ	22
4.8. Git	23
5. Praktyczna realizacja systemu	26
5.1. Wykorzystany zbiór danych	26
5.2. Wstępne przygotowanie danych	26
5.3. Przetwarzanie danych na potrzeby modelu LSTM	31
5.4. Implementacja modelu LSTM	34
5.5. Proces tworzenia modelu LSTM	37
5.6. Zastosowanie modeli predykcyjnych dla aktualnych danych	50

5.7. Implementacja aplikacji webowej	55
6. Podsumowanie	61
Streszczenie	63
Summary	64
Literatura	65
Spis rysunków.....	67
Spis kodów.	69
Spis tabel.....	70

1. Wstęp

Informatyka to stosunkowo młoda dziedzina nauki. Została wyodrębniona jako osobna dyscyplina w latach 60-tych XX wieku, co było kamieniem milowym w rozwoju cywilizacji. Potęga informatyki bierze się z tego, że zajmuje się przetwarzaniem danych, które okazały się być nowym złotem. W dzisiejszych czasach największe firmy na świecie zawdzięczają swój sukces informacjom, które posiadają. Jednak to nie wszystko. Dane same w sobie byłyby bezużyteczne, gdyby nie zostały odpowiednio przetworzone, co jest wielkim wyzwaniem. Doprowadziło to do rozwoju takich dziedzin jak hurtownie danych czy Big Data, które radzą sobie ze zbieraniem różnorodnych danych z wielu źródeł oraz pozwalają je efektywnie przetwarzać. Wykonanie analizy tych danych jest procesem trudnym, ale i bardzo wartościowym, ponieważ prowadzi do zdobycia nowych informacji.

W celu przeprowadzenia analizy bardzo często stosuje się eksplorację danych, która wykorzystuje szybkość komputera do odszukiwania prawidłowości w zgromadzonych zbiorach. Podczas procesu zgłębiania stosowane są wyspecjalizowane algorytmy, które umożliwiają wyciąganie wniosków oraz prognozowanie na podstawie posiadanych informacji. Często wykorzystywane są techniki z obszaru sztucznej inteligencji. Symulowanie ludzkiej inteligencji przez maszyny pozwala przeprowadzić proces uczenia na bazie posiadanych informacji.

Dobrze wykonane analizy są niezwykle przydatne np. w biznesie. Prowadzą do zwiększenia zysków firmy, pozwalają ocenić ryzyko inwestycyjne. Czasami informacje pozyskane dzięki eksploracji mogą okazać się zaskakujące np. firma Kaggle Inc., zrzeszająca naukowców zajmujących się uczeniem maszynowym, wykazała, że najlepiej kupić używane auto w kolorze pomarańczowym, bo właściciele takich aut bardziej o nie dbają. Ta drobna informacja może się okazać przydatna, a bez pomocy komputerów wyciągnięcie takich wniosków byłoby bardzo trudne.

W niniejszej pracy eksploracja zostanie wykorzystana w meteorologii, czyli nauce zajmującej się badaniem warunków pogodowych. Nie jest to łatwa dziedzina, gdyż warunki meteorologiczne są bardzo zmienne i zależne od wielu czynników. Prognozowanie pogody jest procesem złożonym i trudnym, dlatego wykorzystanie komputerów i wyspecjalizowanych algorytmów jest nieocenioną pomocą w tym zadaniu.

2. Cel i zakres pracy

Celem pracy jest wykonanie analizy historycznych danych meteorologicznych w celu prognozowania warunków pogodowych z wykorzystaniem algorytmów eksploracji danych.

Proces realizacji pracy podzielono na cztery etapy:

- wybór zbioru danych do analizy,
- przygotowanie historycznych danych meteorologicznych,
- wykonanie modelu predykcyjnego,
- implementacja aplikacji wykorzystującej model prognostyczny dla aktualnych danych.

System został stworzony przy użyciu nowoczesnych technologii. W celu wstępnego przygotowania i scalenia danych wykorzystano język programowania Python oraz system zarządzania bazami danych – Microsoft SQL Server. Aby przekształcić zbiór na potrzeby modelu predykcyjnego, zastosowano język Python, który posiada takie moduły jak *pandas* czy *numpy*, które są przeznaczone specjalnie do pracy z danymi. Jako model predykcyjny wybrano odmianę sztucznej sieci neuronowej – *Long Short – Term Memory Network*, która jest dostosowana do problemów sekwencyjnych. Do implementacji tej sieci skorzystano z biblioteki Keras.

Celem modelu jest prognozowanie średniej dobowej temperatury na dziś i na następny dzień dla trzech miast – Gdańska, Warszawy i Krakowa. Dokładność zastosowanego rozwiązania została zmierzona przez obliczenie pierwiastka błędu średniokwadratowego, a porównanie otrzymanych wyników z oczekiwanymi zilustrowano na wykresach wykonanych z pomocą modułu *matplotlib*.

Został także napisany skrypt w Pythonie wywołujący model prognostyczny dla aktualnych danych meteorologicznych, które są pobierane z Open Weather Map API. Aby zwizualizować wyniki prognoz, zaimplementowano aplikację webową. Do wykonania backendu zastosowano język programowania Java oraz technologię Spring Framework. Za frontend aplikacji odpowiada język znaczników HTML oraz silnik szablonów Thymeleaf. W celu poprawienia wizualnej oprawy aplikacji skorzystano z biblioteki Bootstrap. Jako środowiska programistyczne wybrano PyCharm oraz IntelliJ. Do sprawnego zarządzania projektem wykorzystano system kontroli wersji Git.

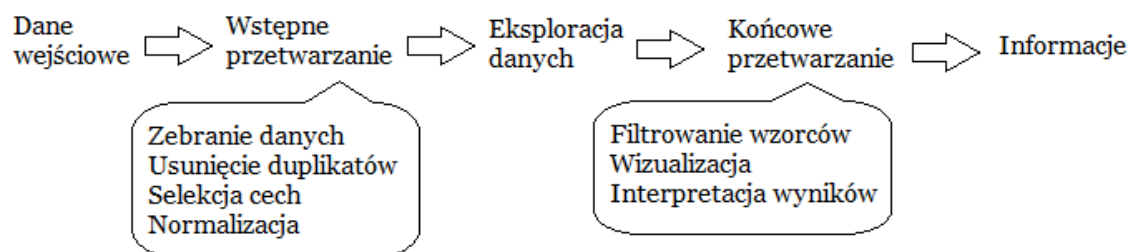
W pracy wyróżniono część teoretyczną i praktyczną. Pierwszą z nich stanowią rozdziały omawiające podstawy teoretyczne niezbędne do wykonania niniejszej pracy, czyli takie zagadnienia jak eksploracja danych, szeregi czasowe oraz sztuczne sieci neuronowe. W tej części opisano także technologie i narzędzia wykorzystane do zaprojektowania systemu. Druga część pracy szczegółowo opisuje implementację systemu, wszystkie kroki jego powstawania, ewaluację dokładności wyników oraz sposób działania. Zaprezentowano i objaśniono kod źródłowy systemu. Wyniki osiągane przez model zilustrowano na wykresach. Przedstawiono efekt końcowy pracy – aplikację webową. Rezultaty uzyskane w pracy zostały podsumowane.

3. Podstawy teoretyczne

3.1. Eksploracja danych

Eksploracja danych (ang. *data mining*, inaczej zgłębianie, ekstrakcja danych) to proces pozyskiwania nowej wiedzy z posiadanych danych. Wykorzystuje szybkość komputera oraz wyspecjalizowane algorytmy w celu odkrywania wzorców i prawidłowości w zbiorach danych. Daje także możliwość prognozowania wartości zmiennych na podstawie zebranych obserwacji.

Zgłębianie danych jest integralną częścią odkrywania wiedzy z baz danych (ang. *Knowledge discovery in databases* – KDD), czyli procesu przekształcania surowych danych w przydatne informacje. Na rysunku 3.1.1 przedstawiono etapy KDD, które prowadzą do otrzymania wartościowych informacji z danych wejściowych.



Rys. 3.1.1 Proces odkrywania wiedzy z baz danych (KDD).

Pierwszy etap - wstępne przetwarzanie danych (ang. *preprocessing*) obejmuje zebranie danych z różnych źródeł, usunięcie duplikatów i niepełnych danych, selekcję cech, wyodrębnienie podzbiorów. Bardzo ważne jest przeprowadzenie normalizacji, czyli ujednolicenia danych. Wstępne przetwarzanie to często proces żmudny i czasochłonny, ale niezwykle istotny. Jego celem jest przekształcenie surowych informacji w dane odpowiednio sformatowane pod kątem ich zgłębiania. Bez tego kroku wykonanie analizy byłoby niemożliwe.

Drugi etap to właściwa eksploracja danych, w ramach której do konkretnych danych dobiera się i stosuje odpowiednie algorytmy. Często są to techniki z obszaru sztucznej inteligencji, uczenia maszynowego lub statystyki np. klasyfikacja, klasteryzacja, asocjacja, regresja czy sztuczne sieci neuronowe.

Na ostatni krok KDD – końcowe przetwarzanie (ang. *postprocessing*) składają się filtrowanie wzorców, wizualizacja oraz interpretacja wyników. W celu dokonania

oceny wydajności modelu stosowane są miary statystyczne. Do najbardziej popularnych należy pierwiastek błędu średniokwadratowego (ang. *Root Mean Square Error* – RMSE), który jest sposobem pomiaru średniego błędu modelu w prognozowaniu wartości. Na rysunku 3.1.2 przedstawiono wzór pozwalający obliczyć RMSE.

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(p_i - y_i)^2}{n}}$$

p_1, p_2, \dots, p_n – wartości prognozowane

y_1, y_2, \dots, y_n – wartości oczekiwane

n – liczba obserwacji

Rys. 3.1.2 Wzór na błąd średniokwadratowy RMSE.

Dzięki końcowemu przetwarzaniu proces KDD znajduje zastosowanie w praktyce – może zostać wykorzystany w aplikacjach biznesowych, ponieważ zapewnia, że tylko poprawne i przydatne informacje zostaną zawarte w systemie.

Zgłębianie danych jest wykorzystywane do dwóch typów problemów. Pierwszy rodzaj to zadania predykcyjne, których celem jest przewidywanie wartości określonego atrybutu na podstawie wartości innych atrybutów. Drugi typ to zadania opisywania danych, czyli szukanie wzorców i prawidłowości, określanie korelacji, trendów i zauważanie anomalii.

Obszarów zastosowań eksploracji danych jest wiele. Do najbardziej popularnych należą:

- bankowość (np. analiza rynku, ocena ryzyka kredytowego, inwestycyjnego, detekcja oszustw),
- marketing (np. dopasowanie oferty do klientów),
- medycyna (np. diagnostyka). [1]

3.2. Szeregi czasowe

Tradycyjne zbiory danych używane w uczeniu maszynowym to kolekcja informacji, w której każda próbka jest traktowana tak samo. Inaczej jest w przypadku szeregów czasowych, które są ciągiem obserwacji uporządkowanych w czasie ze stałym krokiem. Wprowadza to dodatkową informację, która może być istotna w przypadku

wielu problemów prognostycznych. Z drugiej strony przetwarzanie takich danych jest trudniejsze i wymaga zastosowania specjalnych metod.

Na potrzeby rozwiązywania tego typu problemów wprowadzono standardowe terminy używane do opisywania szeregów czasowych:

- t : aktualny czas, który stanowi punkt odniesienia,
- $t - n$: czas opóźniony, przeszłość w stosunku do czasu aktualnego,
- $t + n$: czas przyszły w stosunku do czasu aktualnego.

Aby móc stosować szeregi czasowe, należy odpowiednio przygotować posiadane dane. W tabeli 3.2.1 przedstawiono przykładowy zbiór, w którym obserwacje są posortowane względem czasu.

czas	zmienna
10-05-2021	100
11-05-2021	50
12-05-2021	120
13-05-2021	90

Tabela 3.2.1 Zbiór danych posortowanych względem czasu.

Restrukturyzacja powyższego zbioru polega na tym, że wartość poprzedniego kroku jest używana do prognozowania wartości zmiennej w kroku następnym. W tabeli 3.2.2 widać, że poprzedni krok czasowy jest wejściem (X), a następujący po nim krok czasowy jest wyjściem (y). Dla pierwszego wiersza brakuje wartości wejściowej, a dla ostatniego wartości wyjściowej. Na potrzeby modelu predykcyjnego należy je usunąć. Wykorzystanie wcześniejszych kroków czasowych do przewidywania następnego nazywane jest metodą przesuwnego okna (ang. *sliding window method*).

$X(t)$	$y(t+1)$
NaN	100
100	50
50	120
120	90
90	NaN

Tabela 3.2.2 Zbiór danych po restrukturyzacji.

Ze względu na ilość obserwacji wyróżniamy dwa rodzaje szeregów czasowych:

- jednowymiarowe (ang. *Univariate Time Series*) – obserwowana jest tylko jedna zmienna w każdym kroku czasowym (tak jak w tabeli 3.2.2),

- wielowymiarowe (ang. *Multivariate Time Series*) – obserwowane są dwie lub więcej zmiennych w każdym kroku czasowym i/lub prognozowane są dwie lub więcej zmiennych. Prognozowanie w tym przypadku jest dużo trudniejsze i bardziej złożone. W tabeli 3.2.3 przedstawiono przykład wielowymiarowego szeregu czasowego.

X1(t)	X2(t)	X3(t)	y1(t+1)	y2(t+1)
NaN	NaN	NaN	200	50
100	200	50	150	120
50	150	120	60	90
120	60	90	30	5
90	30	5	NaN	NaN

Tabela 3.2.3 Wielowymiarowy szereg czasowy.

Ze względu na liczbę przewidywanych kroków czasowych wyróżniamy dwa rodzaje prognozowania:

- jednoetapowy (ang. *One - step Forecast*) – przewidywany jest jeden następny krok (tak jak w tabelach 3.2.2 i 3.2.3),
- wieloetapowy (ang. *Multi - step Forecast*) – prognozowane są dwa lub więcej kroki czasowe (tabela 3.2.4).

X1(t-1)	X2(t)	y(t+1)	y(t+2)
NaN	NaN	100	50
NaN	100	50	120
100	50	120	90
50	120	90	NaN
120	90	NaN	NaN

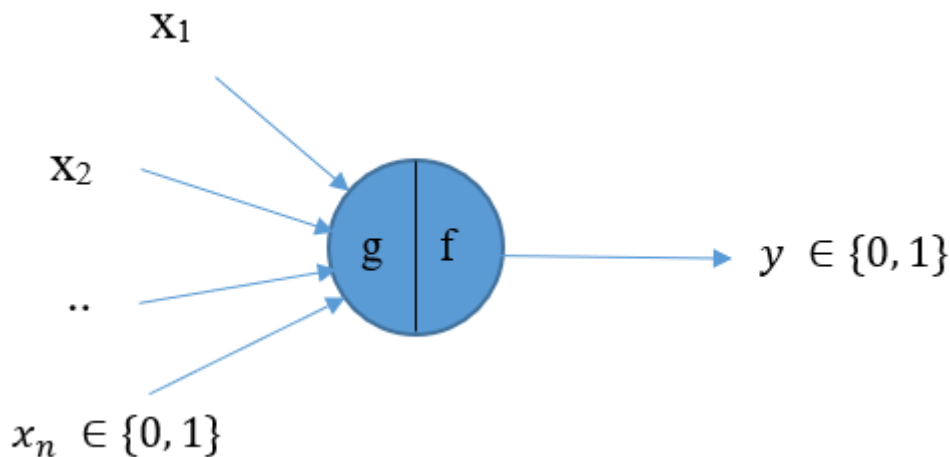
Tabela 3.2.4 Wieloetapowy szereg czasowy.

Celem stosowania szeregów czasowych może być opisanie zbioru danych bądź prognozowanie. Analiza szeregów czasowych polega na projektowaniu modeli, które jak najlepiej opisują obserwowany zbiór danych po to, żeby zrozumieć przyczyny badanego zjawiska. Prognozowanie polega na projektowaniu modeli dopasowanych do historycznych danych, a następnie wykorzystaniu go do przewidywania przyszłych wartości. Analiza szeregu czasowego może pomóc tworzyć lepsze predykcje, ale nie jest konieczna zwłaszcza, że może wymagać dużych nakładów pracy. [2]

3.3. Sztuczne sieci neuronowe

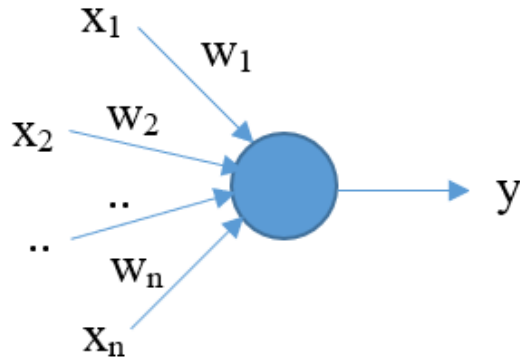
Sztuczna sieć neuronowa (ang. *artificial neural network*, ANN) to system komputerowy symulujący działanie ludzkiego mózgu. Składa się z elementów przetwarzających zwanych neuronami i połączeń między nimi, do których przypisane są wagi. Połączenia te tworzą strukturę neuronową, z którą związane są algorytmy uczenia. [5]

Neuron to podstawowa jednostka ANN, która odbiera dane wejściowe, przetwarza je, przekazuje do funkcji aktywacji i zwraca dane wyjściowe. Pierwszy model sztucznego neuronu został zaproponowany przez McCullocha i Pittsa w 1943r. (rys. 3.3.1). Pierwsza część neuronu (g) odbiera wejście (typu logicznego – 0 lub 1) i dokonuje agregacji. Druga część (f) podejmuje decyzję na podstawie zagregowanej wartości. Zwraca 0, jeśli wartość zagregowanej funkcji nie przekroczy tzw. parametru progowego. W przeciwnym razie zwraca 1. [6]



Rys. 3.3.1 Model neuronu zaproponowany przez Pittsa i McCullocha.

W 1969 Minsky i Papert udoskonallili powyższy model, wprowadzając koncepcję wag, które są miarą ważności danego wejścia (rys. 3.3.2). Ponadto wejście nie musi być wartością logiczną. Podczas agregacji wartości wejściowe mnożone są przez przypisane im wagi. Funkcja wyjściowa zwraca 1, gdy suma iloczynu danych wejściowych i ich wag jest większa lub równa parametrowi progowemu. W przeciwnym wypadku zwracana jest wartość 0. [7]

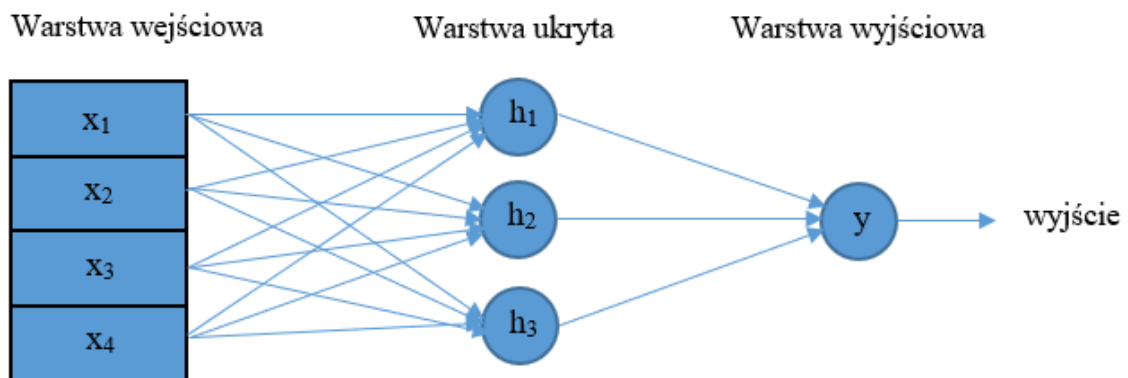


$$\sum_{i=1}^n w_i * x_i \geq \theta \Rightarrow y = 1$$

$$\sum_{i=1}^n w_i * x_i < \theta \Rightarrow y = 0$$

Rys. 3.3.2 Model neuronu zaproponowany przez Minsky'ego i Paperta.

Sieć neuronowa może składać się z jednego neuronu i jest wtedy nazywana siecią jednowarstwową. Jednak daje ona małe możliwości, ponieważ na wyjściu mogą być tylko dwie wartości. Znacznie szersze zastosowanie znajduje sieć MLP (ang. *Multilayer perceptron*), czyli sieć wielowarstwowa z wieloma neuronami. Sieć taka posiada warstwę wejściową, która gromadzi informacje. Warstwa ukryta przetwarza dane, a warstwa wyjściowa oblicza wyjście. Ponadto stosowany jest algorytm propagacji wstecznej (ang. *backpropagation algorithm*). W pierwszej fazie funkcje aktywacji przechodzą z warstwy wejściowej do wyjściowej. W fazie drugiej błędy między aktualnie obserwowaną wartością, a pożądaną wartością są przekazywane z warstwy wyjściowej do wejściowej po to, aby zmodyfikować wagi. Dzięki temu wagi wejść są dostosowane tak, aby sieć osiągała jak najlepsze wyniki. Na rysunku 3.3.3 przedstawiono model wielowarstwowej sieci neuronowej. [8]



Rys. 3.3.3 Model wielowarstwowej sztucznej sieci neuronowej.

3.3.1. Rekurencyjne sieci neuronowe

Rekurencyjne sieci neuronowe (ang. *Recurrent Neural Network*, RNN) to typ sieci neuronowych, który został zaprojektowany na potrzeby problemów sekwencyjnych. Wprowadzono w nich połączenia rekurencyjne, które dodają do sieci stan lub pamięć. Pozwala to wykorzystywać uporządkowany charakter obserwacji w sekwencjach wejściowych. Pamięć wewnętrzna powoduje, że wartość wyjściowa jest uzależniona od aktualnej sekwencji wejściowej. Sieć śledzi po jednej obserwacji na raz i może sprawdzać, jakie obserwacje ją poprzedzają i w jakim stopniu są one istotne przy dokonywaniu prognozy. Zaletą rekurencyjnych sieci neuronowych jest to, że może się nauczyć zależności czasowych i informacji kontekstowych. [3]

3.3.2. Long Short – Term Memory Network

Long Short – Term Memory Network (LSTM) to specjalny rodzaj sieci neuronowej. Należy do typu MPL, czyli składa się z wielu warstw neuronów, przez które propagowane są informacje wejściowe w celu dokonywania predykcji. Jest to rodzaj sieci rekurencyjnej, a więc LSTM posiada rekurencyjne połączenia, które umożliwiają uwzględnianie poprzednich kroków czasowych przy formułowaniu wartości wyjściowych. To co wyróżnia ją od innych RNN to unikalna formuła pozwalająca uniknąć przetrenowania i niewyskalowania modelu. Dzięki temu osiąga świetne wyniki, dlatego jest bardzo popularnym typem sieci neuronowej.

W standardowych rekurencyjnych sieciach neuronowych zauważano problem związany z trenowaniem sieci. Procedura aktualizacji wag może powodować, że wagi szybko stają się tak niskie, że obserwacja traci znaczenie (ang. *vanishing gradients*) lub tak wysokie, że powodują zbyt duże zmiany (ang. *exploding gradients*). LSTM pokonuje ten problem dzięki swojej architekturze. Sieć ta składa się z jednostek obliczeniowych nazywanych komórkami pamięci (ang. *memory cell*). Do komórek tych przypisane są po trzy wartości:

- waga wejściowa (ang. *input weights*) – oznaczająca wagę danej wejściowej dla aktualnego kroku czasowego,
- waga wyjściowa (ang. *output weights*) – waga danej wyjściowej dla ostatniego kroku czasowego,

- stan wewnętrzny (ang. *internal state*) – używany do obliczania wartości wyjściowej aktualnego kroku czasowego.

Każda komórka pamięci posiada także trzy bramy:

- brama zapominania (ang. *forget gate*) – decyduje, jakie informacje usunąć z komórki,
- brama wejściowa (ang. *input gate*) – decyduje, jakimi informacjami wejściowymi aktualizować stan pamięci,
- brama wyjściowa (ang. *output gate*) – decyduje o wartości wyjściowej, bazując na danej wejściowej oraz pamięci komórki.

Właśnie dzięki zastosowaniu bram oraz zapewnieniu spójnego przepływu danych sieć LSTM pokonuje problem trenowania sieci RNN. Dzięki temu, że sieć jest wyposażona w pamięć, radzi sobie z sekwencjami wejściowymi, które mają długoterminowe zależności czasowe. Ponadto dopuszczalne jest stosowanie różnych długości wejść i wyjść, ponieważ sieć przetwarza dane krok po kroku.

Możliwości sieci LSTM są imponujące. Jest ona wykorzystywana nie tylko w problemach sekwencyjnych, ale też np. do automatycznego tłumaczenia tekstu i generowania pisma odręcznego. Sprawdza się najlepiej przy trudnych i skomplikowanych problemach. W sytuacjach prostszych zazwyczaj lepszym wyborem jest zastosowanie zwykłej sieci MLP, ponieważ model ten będzie bardziej wydajny. [3]

4. Wykorzystane technologie

4.1. Python

Python jest interpretowanym językiem programowania wysokiego poziomu. Nie wymaga kompilacji przed uruchomieniem. Pisanie kodu w tym języku jest szybkie, ale uruchomienie zazwyczaj wolniejsze niż w przypadku języków kompilowanych. Twórcy Pythona położyli nacisk na czytelność oraz zwięzłość kodu, co ułatwia pracę programistów, a co za tym idzie zmniejsza koszty utrzymania programów napisanych w tym języku. Jest jednym z najpopularniejszych języków programowania, posiada dobrą dokumentację, a w sieci dostępnych jest wiele materiałów na jego temat, dzięki czemu nawet początkujący programista z łatwością może go zastosować.

Python wyróżnia się dynamicznym deklarowaniem typów zmiennych, czyli przypisywanie typów odbywa się dopiero w trakcie wykonywania programu. Programista nie musi sam deklarować typu zmiennej, robi to interpreter w momencie przypisywania wartości. Deklarowanie dynamiczne jest bardziej elastyczne niż statyczne, zmniejsza ilość kodu oraz ułatwia naukę programowania. [11]

W większości języków programowania grupowanie wyrażeń odbywa się za pomocą nawiasów klamrowych, natomiast tutaj, do tego celu używane są wcięcia.

Python wspiera wiele paradygmatów programowania: proceduralne, funkcyjne oraz bardzo popularne programowanie zorientowane obiektowo, z którego przeniósł wiele koncepcji takich jak dziedziczenie, enkapsulacja, abstrakcja i polimorfizm.

Jedną z największych zalet Pythona jest elastyczność. Świetnie sprawdza się zarówno w prostych, krótkich skryptach jak i rozbudowanych systemach, zapewniając złożone struktury danych i obsługę błędów. Dzięki temu znajduje szerokie zastosowanie w takich dziedzinach informatyki jak skrypty automatyzacyjne, aplikacje internetowe, aplikacje biznesowe. W obszarze Data Science, sztucznej inteligencji i uczeniu maszynowym jest zdecydowanie najczęściej wybieranym językiem programowania, ponieważ posiada moduły przeznaczone do przetwarzania danych oraz biblioteki implementujące algorytmy sztucznej inteligencji, które łatwo dostosować do swoich potrzeb.

Python oferuje szeroką gamę bibliotek, które znacząco ułatwiają pracę. Aby z nich korzystać potrzebne jest środowisko do zarządzania zależnościami nazywane *venv*

(ang. *virtual environment*). Posiada ono własny interpreter, moduły i biblioteki. Instalacji bibliotek dokonuje się najczęściej za pomocą programów *pip* lub *conda*. Poniżej opisano kilka najważniejszych bibliotek Pythona:

- *Pandas* – wykorzystywana do pracy z danymi, oferując przechowywanie danych w obiektach *DataFrame*. Umożliwia szybką i efektywną manipulację danymi. Pozwala odczytywać i zapisywać informacje z plików tekstowych, baz danych oraz z plików w formacie *csv* i *HDF5*. Oferuje elastyczne przekształcanie danych i wydajne skalanie zbiorów. Pomaga również w pracy z szeregami czasowymi np. funkcja *shift()* tworzy kopię kolumny z opóźnionym krokiem czasowym. Bardzo przydaje się do przekształcania danych na potrzeby analiz.[12]
- *Numpy* – zapewnia obsługę dużych, wielowymiarowych macierzy. Podstawowym obiektem tej biblioteki jest *ndarray*, czyli macierz *n* – wymiarowa o obiektach tego samego typu. Umożliwia między innymi wykonywanie operacji matematycznych i transformację macierzy. Jest wykorzystywana do obliczeń naukowych, a także do przygotowywania danych na potrzeby Data Science. Biblioteka ta jest bardzo wydajna dzięki temu, że korzysta ze skompilowanego kodu napisanego w języku C. [13]
- *Scikit – learn* – dostarcza implementację algorytmów uczenia maszynowego, zarówno nadzorowanego jak i nienadzorowanego np. klasyfikacji, klasteryzacji, regresji. Zapewnia narzędzia służące do przetwarzania danych, trenowania modelu i jego ewaluacji. [14]
- *Matplotlib* – biblioteka do tworzenia statycznych, animowanych i interaktywnych wizualizacji danych. Pozwala sporządzać przeróżne rodzaje wykresów. Stosując domyślne ustawienia odbywa się to niezwykle szybko. Posiada również wiele narzędzi, które pozwalają zwizualizować dane w sposób dostosowany do indywidualnych preferencji. [15]
- *Keras* – biblioteka typu open-source. Zapewnia interfejs Pythona dla sztucznych sieci neuronowych. Implementuje elementy stosowane w ANN takie jak warstwy, funkcje aktywacji. Wspiera standardowe, splotowe i rekurencyjne sieci neuronowe. [16]

Mimo wielu korzyści Python nie jest językiem pozbawionym wad. Decydując się na wybór tej technologii w swoim systemie, należy wziąć pod uwagę jego ograniczenia.

Po pierwsze wysokie zużycie pamięci, a także niską wydajność na urządzeniach mobilnych i przeglądarkach internetowych oraz słabo rozwinięte biblioteki obsługi baz danych. Co więcej, prostota tego języka choć z jednej strony jest wielką zaletą, z drugiej powoduje, że po kilku latach programowania tylko w tym języku ciężko jest posługiwać się innym.

4.2. Java

Java jest zorientowanym obiektowo językiem programowania. To również platforma komputerowa używana do rozwijania aplikacji. Java jest jednym z najszybszych, najbezpieczniejszych i najbardziej niezawodnych języków programowania, stąd zyskała ogromną popularność. Język ten jest ciągle rozwijany. Tworzone są nowe wersje, dodające nowe funkcjonalności. [17]

Java została zaprojektowana tak, aby była łatwa w użyciu – pisaniu, kompilowaniu i debugowaniu kodu. Ponieważ jest językiem obiektowym, pozwala tworzyć modułowe programy, a projektowanie klas powoduje, że ten sam kod może zostać wykorzystany wielokrotnie, co upraszcza jego strukturę i ułatwia utrzymanie.

Java jest niezależna od platformy, w której zostaje uruchomiona, a to za sprawą wirtualnej maszyny JVM (ang. *Java Virtual Machine*), która kompiluje kod do postaci bajtowej, a nie maszynowej. [18]

Niezwykle przydatną cechą Javy jest mechanizm automatycznego zarządzania pamięcią. Do tego celu został stworzony *garbage collector*, który dynamicznie przydziela i zwalnia pamięć.

Java znajduje szerokie zastosowanie w rozwijaniu aplikacji desktopowych, webowych i mobilnych. Jest używana również do analizy Big Data, obliczeń naukowych, programowania urządzeń czy technologii serwerowych.

Mimo wielu zalet nie jest to język idealny. Jego największą wadą jest wysokie zużycie pamięci, więc decydując się na programowanie w tym języku, należy mieć to na uwadze. [19]

4.3. Spring Framework

Spring Framework jest szkieletem projektowania aplikacji dla platformy Java EE. Zapewnia kompleksowy model programowania i konfiguracji. Jest bogaty w gotowe elementy wykorzystywane do tworzenia aplikacji, dzięki czemu programiści nie muszą implementować całego programu od podstaw, za to mogą skupić się na logice biznesowej. [20]

Spring Framework stosuje paradygmat AOP (ang. *Aspect – oriented programming*), czyli programowanie zorientowane aspektowo. Jest to podobne podejście do popularnego OOP (ang. *Object – oriented programming*) – programowanie zorientowane obiektowo, jednak podstawową jednostką kodu nie jest klasa, a aspekt rozumiany jako wydzielony problem. Takie podejście pomaga oddzielić problemy przekrojowe od obiektów, na które mają wpływ.

Do najważniejszych cech technologii Spring Framework należą:

- budowa modułowa,
- model MVC (*model – view – controller*),
- mechanizm odwracania zależności,
- rozwijanie aplikacji bazujących na klasach POJO (ang. *Plain Old Java Object*), dzięki czemu do uruchamiania aplikacji nie jest potrzebny serwer aplikacyjny, tylko kontener serwletów np. Tomcat,
- osobny kontener przeznaczony na potrzeby testowania, dzięki czemu jest ono znacznie prostsze,
- spójny interfejs zarządzania transakcjami, który można wyskalować zarówno na potrzeby lokalnych jak i globalnych transakcji. [21]

Dużą zaletą tej technologii jest niskie zużycie pamięci oraz zasobów procesora, dzięki czemu aplikacje projektowane z wykorzystaniem Spring Framework są bardzo wydajne i znalazły szerokie zastosowanie w biznesie. [22]

4.4. HTML

HTML (ang. *Hypertext Markup Language*) to język znaczników przeznaczony do tworzenia stron internetowych. HTML jest oficjalnym standardem webowym utrzymywany i rozwijany przez konsorcjum W3C. Służy do projektowania

struktury dokumentów dzięki zastosowaniu znaczników, które określają styl dokumentu. Niosą one informacje, w jaki sposób mają być wyświetlane takie elementy jak tekst, pliki multimedialne czy hiperłącza.

Każda strona internetowa składa się z serii elementów. Trzy główne części elementu to:

- znacznik otwierający – określa początek elementu,
- zawartość – dane wyjściowe widoczne dla użytkowników,
- znacznik zamykający – określa koniec elementu.

HTML jest całkowicie darmowy i wspierany przez wszystkie przeglądarki. Ma prostą składnię, więc może być stosowany nawet przez początkujących programistów. Można go łatwo zintegrować z językami programowania takimi jak Java, PHP, Node.js. Jego wadą jest to, że nie umożliwia tworzenia dynamicznej funkcjonalności. Z tego też powodu nie jest uznawany za język programowania. [23]

4.4.1. Bootstrap

Bootstrap to biblioteka HTML, CSS i JavaScript bogata w narzędzia ułatwiające tworzenie interfejsu graficznego stron internetowych. Jest to najbardziej popularna biblioteka tego typu. Można korzystać z niej całkowicie bezpłatnie. Jest kompatybilna z większością przeglądarek internetowych. Posiada responsywną siatkę CSS automatycznie dostosowującą się do rozdzielczości urządzenia.

Bootstrap oferuje szeroką gamę estetycznych i funkcjonalnych elementów np. przyciski, tabele, formularze. Wykorzystanie gotowych elementów znacząco skraca projektowanie stron internetowych. Natomiast wadą stosowania tej biblioteki jest to, że zazwyczaj wiąże się to z dołączeniem kodu, który nie jest używany, co utrudnia i wydłuża proces optymalizacji. [24]

4.5. Thymeleaf

Thymeleaf to silnik szablonów Java służący do tworzenia i przetwarzania plików HTML, CSS, XML i JavaScript. Działa w środowiskach internetowych i samodzielnych. Zapewnia integrację dla Spring Framework. W pełni zastępuje

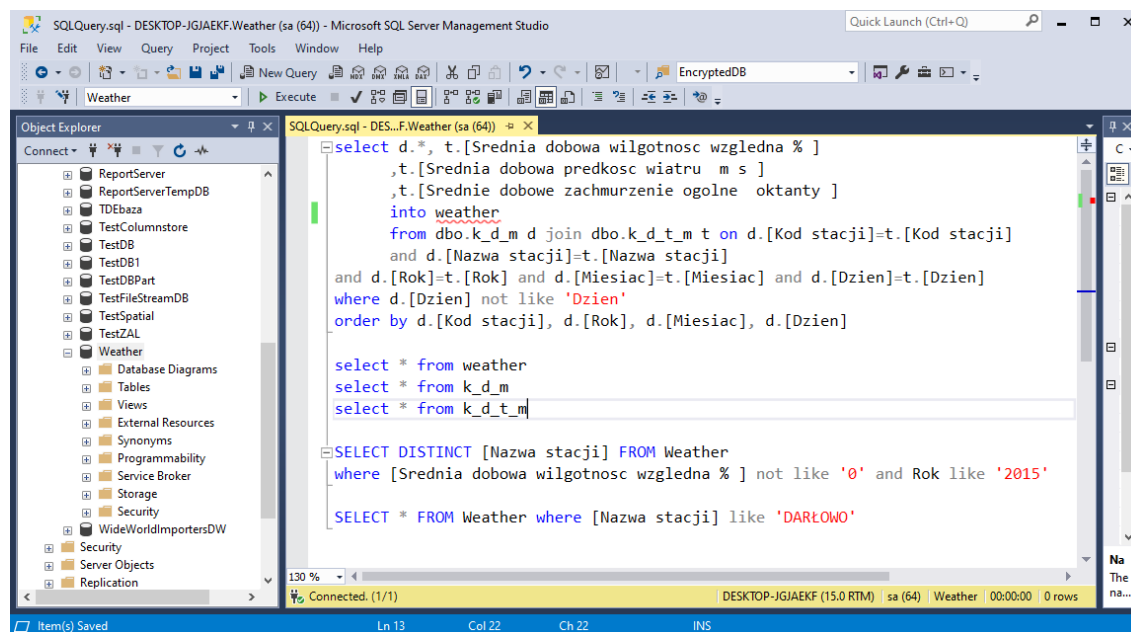
JavaServer Pages (JSP). Jest oparty na modułowych zestawach cech zwanych dialektami. Są one stosowane poprzez wprowadzenie znaczników i atrybutów szablonu. Dostępny jest dialekt standardowy oraz dialekt przeznaczony dla Spring Framework.[25] Możliwe jest także tworzenie własnych dialektów. Thymeleaf posiada konfigurowalną i bardzo wydajną pamięć podręczną analizowanych szablonów, która ogranicza ilość danych wejściowych i wyjściowych do minimum. [26]

4.6. Microsoft SQL Server

Microsoft SQL Server to system zarządzania relacyjną bazą danych, czyli taką, która przechowuje dane w tabelach. [27] Składają się one z wierszy, które zawierają rekordy, natomiast kolumny zawierają atrybuty danych. Tabele można łączyć ze sobą poprzez relacje jeden do jednego, jeden do wielu lub wiele do wielu. Model relacyjny zapewnia spójność danych oraz uporządkowaną strukturę.

Podstawę systemu SQL Server stanowi silnik bazodanowy, który jest odpowiedzialny za przechowywanie, przetwarzanie i bezpieczeństwo danych. [28]

Językiem zapytań stosowanym w tym systemie jest T-SQL. Tak jak podstawowa wersja SQL umożliwia tworzenie, modyfikowanie baz, wstawianie i pobieranie danych, natomiast rozszerza tę funkcjonalność o takie możliwości jak np.: definiowanie elementów proceduralnych, obsługa wyjątków, definiowanie typów i zmiennych.



Rys. 4.6.1 Interfejs graficzny środowiska Microsoft SQL Server 18.

W celu zarządzania systemem SQL Server zostało stworzone środowisko SQL Server Management Studio. Zapewnia ono dostęp do wszystkich komponentów systemu SQL Server, umożliwia konfigurowanie go, zarządzanie i administrację. Posiada różnorodne narzędzia graficzne przeznaczone do pracy z danymi np. służące do importu i eksportu plików. Na rysunku 4.6.1 przedstawiono interfejs graficzny środowiska Microsoft SQL Server 18. [29]

4.7. PyCharm i IntelliJ

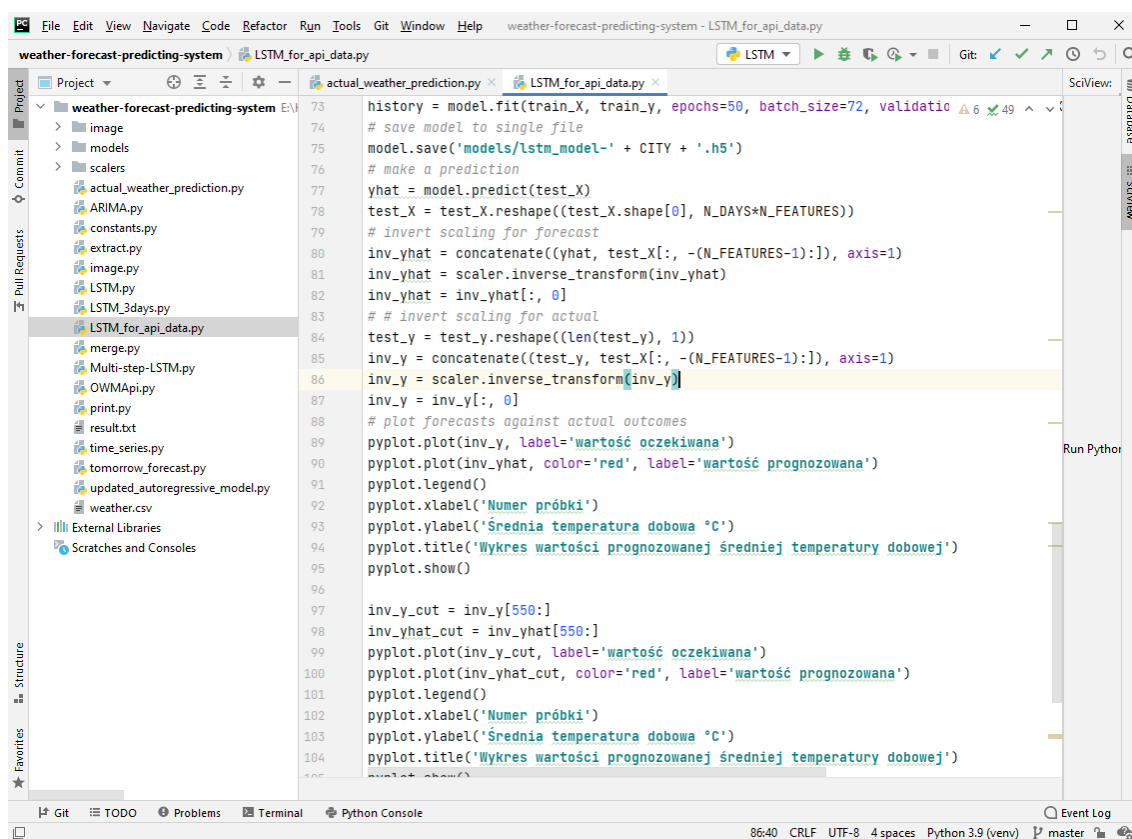
PyCharm i IntelliJ to środowiska programistyczne firmy JetBrains. Pierwsze z nich zapewnia wsparcie dla Pythona, drugi dla Javy. Mimo że są przeznaczone dla dwóch znacznie różniących się języków programowania są do siebie bardzo podobne. Działają w systemach operacyjnych Windows, Linux i macOS. Umożliwiają edycję kodu źródłowego oraz pozwalają uruchomić napisane programy. [30]

Posiadają szereg cech ułatwiających pracę programisty:

- sugestia naprawy błędów w kodzie,
- wykrywanie zduplikowanych fragmentów kodu,
- wygodne skróty klawiszowe,
- graficzny debugger, który pozwala śledzić działanie programu krok po kroku, co pomaga wychwycić błędy,
- zintegrowany tester jednostkowy,
- wsparcie dla systemu kontroli wersji.

Pomimo tego, że PyCharm jest przeznaczony do pracy z Pythonem, a IntelliJ do Javy, możliwa jest także edycja takich plików jak HTML, CSS, JavaScript czy CoffeeScript. Oba środowiska pozwalają pracować z kilkoma systemami baz danych.[31]

Interfejs graficzny obu środowisk jest niemalże identyczny, bardzo czytelny i przejrzysty (rys. 4.7.1).



Rys. 4.7.1 Interfejs graficzny środowiska programistycznego PyCharm.

4.8. Git

Git to darmowy, rozproszony system kontroli wersji przeznaczony do zarządzania projektami, przede wszystkim zespołowymi, ale w pracy indywidualnej również bardzo dobrze się sprawdza. Umożliwia przechowywanie kodu źródłowego w serwisie internetowym GitHub (rys. 4.8.1), co chroni przed utratą wyników pracy. Dzięki temu serwisowi można także w łatwy sposób udostępnić swój projekt innym, a także pracować nad jednym projektem na kilku różnych urządzeniach.

Git pozwala dzielić swoją pracę na części i zatwierdzać każdą istotną zmianę. Możliwy jest także szybki powrót do wcześniejszych wersji projektu. W serwisie GitHub widoczna jest cała historia zmian. Na rysunku 4.8.2 przedstawiono funkcję przeglądania zmian w kodzie źródłowym systemu.

System Git umożliwia wyodrębnianie tzw. gałęzi. Dzięki temu można np. stworzyć gałąź developerską służącą do pracy nad zmianami oraz produkcyjną, do której trafiają ostateczne zmiany. [32]

🔒 Allariana / WeatherForecastSystem (Private) Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Security Insights Settings

🔑 master Go to file Add file Code About ⚙️

File	Description	Time
Allariana write summary		22 hours ago 59
weather-forecast-predic...	write about charts	3 days ago
web-app	write about web app	2 days ago
README.md	Create README.md	3 months ago
SQLQuery.sql	write about LSTM implementation	20 days ago
Sochacka_Praca_magiste...	write summary	22 hours ago

No description, website, or topics provided.

Readme

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages

Python 88.1% Java 7.6% HTML 4.3%

Rys. 4.8.1 Wygląd repozytorium projektu w serwisie GitHub.

predict for actual data Browse files

🔑 master

Allariana committed on Apr 25 1 parent 96e34a9 commit 2224b6497cfb2005f77cbc4b3eae19d1edd18393

Showing 2 changed files with 11 additions and 7 deletions. Unified Split

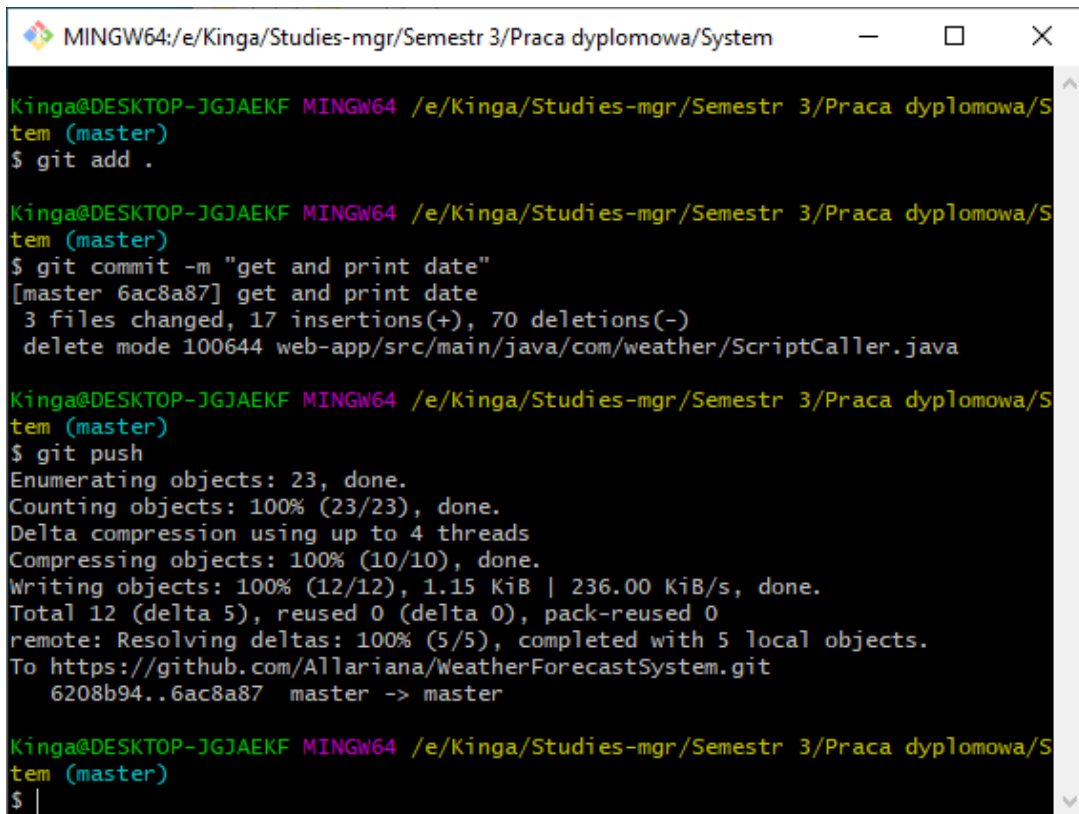
```

18 weather-forecast-predicting-system/actual_weather_prediction.py
... @@ -1,22 +1,26 @@
1 1 import OWMApi
2 - import pickle
3 2 from keras.models import load_model
4 3 from joblib import load
5 4 from pandas import DataFrame
6 + import numpy as np
7 + from constants import *
6 7
7 8 actual_data = []
8 9 actual_data = OWMApi.get_actual_weather_data()
9 10 df = DataFrame(actual_data).transpose()
10 11 values = df.values
11 12 values = values.astype('float32')
12 - # scaler = load('scaler.bin')
13 - # scaler = pickle.load(open('scaler.pkl', 'rb'))
14 13 scaler = load('scaler.joblib')
15 - # scaled = scaler.fit_transform(values)
16 - scaled = scaler.transform(values)
14 + values_all = values[:, 0:5]
15 + values_all = scaler.transform(values_all)
16 + for i in range(5, 21, 5):
17 +     values_cut = values[:, i:(i+5)]
18 +     values_cut = scaler.transform(values_cut)
19 +     values_all = np.concatenate([values_all, values_cut], axis=1)

```

Rys. 4.8.2 Przeglądanie zmian kodu w serwisie GitHub.

Git działa bardzo szybko, a to dlatego, że zużywa mało zasobów systemu. Jest łatwy w obsłudze. Został wyposażony w konsolę – Git Bash (rys. 4.8.2), która poprzez wpisywanie komend, umożliwia pracę nad projektem.



```
MINGW64:/e/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/System

Kinga@DESKTOP-JGJAEKF MINGW64 /e/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/S
tem (master)
$ git add .

Kinga@DESKTOP-JGJAEKF MINGW64 /e/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/S
tem (master)
$ git commit -m "get and print date"
[master 6ac8a87] get and print date
3 files changed, 17 insertions(+), 70 deletions(-)
delete mode 100644 web-app/src/main/java/com/weather/ScriptCaller.java

Kinga@DESKTOP-JGJAEKF MINGW64 /e/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/S
tem (master)
$ git push
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 1.15 KiB | 236.00 KiB/s, done.
Total 12 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/Allariana/WeatherForecastSystem.git
6208b94..6ac8a87 master -> master

Kinga@DESKTOP-JGJAEKF MINGW64 /e/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/S
tem (master)
$ |
```

Rys. 4.8.3 Okienko konsoli Git Bash.

5. Praktyczna realizacja systemu

5.1. Wykorzystany zbiór danych

Pierwszym etapem praktycznej realizacji systemu był wybór zbioru danych do analizy. Jest to niezwykle istotna część pracy, ponieważ im dokładniejsze dane, tym model prognostyczny daje lepsze efekty. Na potrzeby pracy wybrano historyczne dane meteorologiczne udostępniane przez Instytut Meteorologii i Gospodarki Wodnej. Są one zbierane z polskich stacji klimatologicznych i przechowywane w formacie *csv*. Dostępne dane obejmują okres od 1951 r. do chwili obecnej. Na potrzeby pracy pobrano tylko najnowsze dane (od stycznia 2010r. do stycznia 2021r.), ponieważ klimat się zmienia, w związku z czym najbardziej wartościowe, wiarygodne są dane z ostatnich lat.

Parametry meteorologiczne zawarte w wybranym zbiorze to:

- maksymalna temperatura dobową [$^{\circ}\text{C}$],
- minimalna temperatura dobową [$^{\circ}\text{C}$],
- średnia temperatura dobową [$^{\circ}\text{C}$],
- temperatura minimalna przy gruncie [$^{\circ}\text{C}$],
- średnia dobową wilgotność względną [%],
- średnia dobową prędkość wiatru [m/s],
- suma dobową opadów [mm],
- rodzaj opadu [S/W/],
- średnie dobowe zachmurzenie ogólne [oktanty],
- wysokość pokrywy śnieżnej [cm].

5.2. Wstępne przygotowanie danych

Pobrane pliki z danymi były podzielone na dane miesięczne. Fragment jednego z plików został przedstawiony na rysunku 5.2.1.

2.49E+08	PSZCZYNA	2010	1	1	0.8	0	8	1	8
2.49E+08	PSZCZYNA	2010	1	2	-2.6	0	8	1	8
2.49E+08	PSZCZYNA	2010	1	3	-5.5	0	8	0	8
2.49E+08	PSZCZYNA	2010	1	4	-9.3	0	8	0	6.3
2.49E+08	PSZCZYNA	2010	1	5	-5.5	0	8	0	8
2.49E+08	PSZCZYNA	2010	1	6	-5.4	0	8	0.7	8
2.49E+08	PSZCZYNA	2010	1	7	-3.7	0	8	0.7	8
2.49E+08	PSZCZYNA	2010	1	8	-4.1	0	8	2	8
2.49E+08	PSZCZYNA	2010	1	9	0	0	8	3	8
2.49E+08	PSZCZYNA	2010	1	10	-1.4	0	8	1.7	8
2.49E+08	PSZCZYNA	2010	1	11	-3.9	0	8	0.3	8
2.49E+08	PSZCZYNA	2010	1	12	-3.5	0	8	0	8
2.49E+08	PSZCZYNA	2010	1	13	-3.2	0	8	1	8
2.49E+08	PSZCZYNA	2010	1	14	-4.5	0	8	0.3	8

Rys. 5.2.1 Arkusz kalkulacyjny z surowymi danymi.

Aby móc przetwarzać dane, należało scalić wszystkie pliki w jeden zbiór. W tym celu napisano skrypt, który zamieszczono w listingu 5.2.1. Pierwotne dane były zapisane w dwóch rodzajach plików. Każdy rodzaj zapisano w osobnych folderach. Skrypt wczytuje dane, wykorzystując moduł *glob*, który służy do rekurencyjnego pobierania ścieżek z katalogów, co umożliwia wczytanie wszystkich plików znajdujących się w jednym folderze za pomocą jednej linijki kodu. Następnie w pętli *for* otwierany jest każdy z plików i nadawane są nagłówki kolumnom, a otrzymane pliki zapisywane są w formacie *csv*. W następnej kolejności ponownie za pomocą modułu *glob* wczytywane są pliki – tym razem nowo utworzone, z nagłówkami. Za pomocą funkcji *concat* z modułu *pandas* pliki zostają scalone. W efekcie działania skryptu powstały dwa różne pliki w formacie *csv*, różniące się przechowywanymi parametrami meteorologicznymi (rys. 5.2.2, rys. 5.2.3).

```
import os
import glob
import pandas as pd

from constants import *

all_files = glob.glob(os.path.join(SOURCE_DATA, "*.csv"))
all_files2 = glob.glob(os.path.join(SOURCE_DATA2, "*.csv"))

i = 0
for f in all_files:
    df = pd.read_csv(f, header=None, encoding='windows-1250')
    df.to_csv(DATA_WITH_HEADERS + "\k_" + str(i) + ".csv",
    header=["Kod stacji", "Nazwa stacji",
    "Rok", "Miesiac", "Dzien", "Srednia dobowa temperatura",
    "Status pomiaru TEMP",
    "Srednia dobowa wilgotnosc wzgledna[%]", "Status pomiaru
    WLGS", "Srednia dobowa predkosci wiatru [m/s]",
    "Status pomiaru FWS", "Srednie dobowe zachmurzenie ogolne
    [oktanty]", "Status pomiaru NOS"],
    encoding='windows-1250')
    i += 1
```

```

i = 0
for f in all_files2:
    df = pd.read_csv(f, header=None, encoding='windows-1250')
    df.to_csv(DATA_WITH_HEADERS2 + "\k_" + str(i) + ".csv",
header=["Kod stacji", "Nazwa stacji", "Rok",
"Miesiac", "Dzien", "Maksymalna temperatura dobowa [°C]",
"Status pomiaru TMAX",
"Minimalna temperatura dobowa [°C]", "Status pomiaru TMIN",
"Średnia temperatura dobowa [°C]",
"Status pomiaru STD", "Temperatura minimalna przy gruncie
[°C]", "Status pomiaru TMNG",
"Suma dobowa opadów [mm]", "Status pomiaru SMDB", "Rodzaj
opadu [S/W/ ]",
"Wysokość pokrywy śnieżnej [cm]", "Status pomiaru PKSN"],
encoding='windows-1250')
    i += 1

files = glob.glob(os.path.join(DATA_WITH_HEADERS, "*.csv"))
files2 = glob.glob(os.path.join(DATA_WITH_HEADERS2, "*.csv"))
df_from_each_file = (pd.read_csv(f, sep=',', header=None,
encoding='windows-1250') for f in files)
df_from_each_file2 = (pd.read_csv(f2, sep=',', header=None,
encoding='windows-1250') for f2 in files2)
df_merged = pd.concat(df_from_each_file)
df_merged2 = pd.concat(df_from_each_file2)
df_merged.to_csv(DESTINATION_FOLDER + "\k_d_t_m.csv",
encoding='windows-1250')
df_merged2.to_csv(DESTINATION_FOLDER + "\k_d_m.csv",
encoding='windows-1250')

```

Listing 5.2.1 Kod źródłowy skryptu do scalania danych.

Kod stacji	Nazwa sta	Rok	Miesiac	Dzien	Maksymal	Minimaln	Srednia te	Temperat	Suma dob	Wysokosc	pokrywy	śnieżnej [cm]
2.49E+08	PSZCZYNA	2010	1	1	3	-0.9	0.8	-0.5	2.8	0		
2.49E+08	PSZCZYNA	2010	1	2	0.8	-4.1	-2.6	-4.6	2.4	0		
2.49E+08	PSZCZYNA	2010	1	3	-3.5	-6.9	-5.5	-9.2	2.3	3		
2.49E+08	PSZCZYNA	2010	1	4	-3.8	-13.4	-9.3	-13.9	0.1	4		
2.49E+08	PSZCZYNA	2010	1	5	-3	-8.4	-5.5	-13.7	0.5	4		
2.49E+08	PSZCZYNA	2010	1	6	-2.6	-7.2	-5.4	-7.7	2.8	5		
2.49E+08	PSZCZYNA	2010	1	7	-1	-6.4	-3.7	-8.7	1.6	8		
2.49E+08	PSZCZYNA	2010	1	8	-1	-8.2	-4.1	-8.7	17.8	9		
2.49E+08	PSZCZYNA	2010	1	9	1.5	-1.4	0	-4.6	8.5	13		
2.49E+08	PSZCZYNA	2010	1	10	0.2	-2.4	-1.4	-2.5	9.4	10		
2.49E+08	PSZCZYNA	2010	1	11	-2.2	-4.9	-3.9	-5.3	0	17		
2.49E+08	PSZCZYNA	2010	1	12	-1.9	-4.9	-3.5	-4.1	2.3	16		
2.49E+08	PSZCZYNA	2010	1	13	-1.3	-4.4	-3.2	-4.1	0	18		
2.49E+08	PSZCZYNA	2010	1	14	-3.4	-5.4	-4.5	-7.2	0	16		
2.49E+08	PSZCZYNA	2010	1	15	-4	-5.9	-4.8	-7.5	0	15		

Rys. 5.2.2 Arkusz kalkulacyjny zawierający scalone dane – pierwszy rodzaj plików.

Kod stacji	Nazwa stacji	Rok	Miesiac	Dzien	Srednia d	Srednia d	Srednia d	Srednie dobowe zachmurzenie ogolne [oktanty]
2.49E+08	PSZCZYNA	2010	1	1	0.8	0	1	8
2.49E+08	PSZCZYNA	2010	1	2	-2.6	0	1	8
2.49E+08	PSZCZYNA	2010	1	3	-5.5	0	0	8
2.49E+08	PSZCZYNA	2010	1	4	-9.3	0	0	6.3
2.49E+08	PSZCZYNA	2010	1	5	-5.5	0	0	8
2.49E+08	PSZCZYNA	2010	1	6	-5.4	0	0.7	8
2.49E+08	PSZCZYNA	2010	1	7	-3.7	0	0.7	8
2.49E+08	PSZCZYNA	2010	1	8	-4.1	0	2	8
2.49E+08	PSZCZYNA	2010	1	9	0	0	3	8
2.49E+08	PSZCZYNA	2010	1	10	-1.4	0	1.7	8
2.49E+08	PSZCZYNA	2010	1	11	-3.9	0	0.3	8
2.49E+08	PSZCZYNA	2010	1	12	-3.5	0	0	8
2.49E+08	PSZCZYNA	2010	1	13	-3.2	0	1	8
2.49E+08	PSZCZYNA	2010	1	14	-4.5	0	0.3	8
2.49E+08	PSZCZYNA	2010	1	15	-4.8	0	0.7	7.3
2.49E+08	PSZCZYNA	2010	1	16	-4.6	0	0.7	8

Rys. 5.2.3 Arkusz kalkulacyjny zawierający scalone dane – drugi rodzaj plików.

Do scalenia powyższych dwóch plików skorzystano z programu Microsoft SQL Server. Każdy plik został importowany za pomocą narzędzia graficznego do osobnej tabeli, a następnie za pomocą polecenia *select into* z klauzulą *join* (listing 5.2.2) połączono te tabele w jedną. Została ona przedstawiona na rysunku 5.2.4 po wyświetleniu pierwszych rekordów poleceniem *select* w programie Microsoft SQL Server Management studio. Tabelę wyeksportowano następnie do pliku w formacie *csv*. Fragment otrzymanego pliku zawierającego wszystkie dane przedstawiono na rysunku 5.2.5.

```

select d.*, t.[Srednia dobowo wilgotnosc wzgledna % ]
,t.[Srednia dobowo predkosc wiatru m s ]
,t.[Srednie dobowe zachmurzenie ogolne oktanty ]
into weather
from dbo.k_d_m d join dbo.k_d_t_m t on d.[Kod stacji]=t.[Kod stacji]
and d.[Nazwa stacji]=t.[Nazwa stacji]
and d.[Rok]=t.[Rok] and d.[Miesiac]=t.[Miesiac] and d.[Dzien]=t.[Dzien]
where d.[Dzien] not like 'Dzien'
order by d.[Kod stacji], d.[Rok], d.[Miesiac], d.[Dzien]

```

Listing 5.2.2 Polecenie *select into* scalające dane.

	Kod stacji	Nazwa stacji	Rok	Miesiac	Dzien	Maksymalna temperatura dobowo °C	Minimalna temperatura dobowo °C	Srednia temperatura dobowo °C	Temperatura minimalna przy gruncie °C
1	253180020	RADOSTOWO	2013	10	25	14	2.8	7.1	0.2
2	253180020	RADOSTOWO	2013	10	26	15.6	7	12.1	5.9
3	253180020	RADOSTOWO	2013	10	27	16	9.8	12.7	9.9
4	253180020	RADOSTOWO	2013	10	28	17.2	8.6	13.3	8
5	253180020	RADOSTOWO	2013	10	29	17.7	9	12.3	8.6
6	253180020	RADOSTOWO	2013	10	30	13.2	5.4	8.2	3.9
7	253180020	RADOSTOWO	2013	10	31	12.2	4.1	6.8	2.1
8	253180090	ŚLIWICE	2013	10	1	10.8	-1.9	4.1	-1.2
9	253180090	ŚLIWICE	2013	10	2	9.2	-0.1	2.9	-2.5
10	253180090	ŚLIWICE	2013	10	3	9.8	-4.9	1	-5.8
11	253180090	ŚLIWICE	2013	10	4	10	-4.3	1.5	-6.6
12	253180090	ŚLIWICE	2013	10	5	12.3	0.3	5.3	-0.2
13	253180090	ŚLIWICE	2013	10	6	13.8	4.2	9.3	1.5
14	253180090	ŚLIWICE	2013	10	7	16.4	7.4	11	4.1
15	253180090	ŚLIWICE	2013	10	8	14.2	3.5	8.9	0.9
16	253180090	ŚLIWICE	2013	10	9	15.9	6	10.4	3
17	253180090	ŚLIWICE	2013	10	10	16.6	7.3	10.8	4.2
18	253180090	ŚLIWICE	2013	10	11	17.6	7.4	11.5	2.4
19	253180090	ŚLIWICE	2013	10	12	12	8.9	9.9	8.3
20	253180090	ŚLIWICE	2013	10	13	12	9.1	10.7	11.2

Rys. 5.2.4 Widok tabeli w programie Microsoft SQL Server Management Studio.

Kod stacji	Nazwa sta	Rok	Miesiac	Dzien	Maksymal	Minimaln	Srednia te	Temperat	Suma dob	Wysokosc	Srednia d	Srednia d	Srednie dobowe zach
2.53E+08	RADOSTO	2013	10	25	14	2.8	7.1	0.2	0	0	0	0	0
2.53E+08	RADOSTO	2013	10	26	15.6	7	12.1	5.9	1.1	0	0	0	0
2.53E+08	RADOSTO	2013	10	27	16	9.8	12.7	9.9	3.6	0	0	0	0
2.53E+08	RADOSTO	2013	10	28	17.2	8.6	13.3	8	0	0	0	0	0
2.53E+08	RADOSTO	2013	10	29	17.7	9	12.3	8.6	0	0	0	0	0
2.53E+08	RADOSTO	2013	10	30	13.2	5.4	8.2	3.9	0	0	0	0	0
2.53E+08	RADOSTO	2013	10	31	12.2	4.1	6.8	2.1	0	0	0	0	0
2.53E+08	ŚLIWICE	2013	10	1	10.8	-1.9	4.1	-1.2	0	0	81	1.3	4.7
2.53E+08	ŚLIWICE	2013	10	2	9.2	-0.1	2.9	-2.5	0	0	78.3	1.7	4
2.53E+08	ŚLIWICE	2013	10	3	9.8	-4.9	1	-5.8	0	0	80.3	1.7	6
2.53E+08	ŚLIWICE	2013	10	4	10	-4.3	1.5	-6.6	0	0	78.3	3.3	5.7
2.53E+08	ŚLIWICE	2013	10	5	12.3	0.3	5.3	-0.2	0	0	76	3.3	5.3
2.53E+08	ŚLIWICE	2013	10	6	13.8	4.2	9.3	1.5	0	0	80	2	6
2.53E+08	ŚLIWICE	2013	10	7	16.4	7.4	11	4.1	0	0	82	2	6
2.53E+08	ŚLIWICE	2013	10	8	14.2	3.5	8.9	0.9	0	0	82.3	2	6
2.53E+08	ŚLIWICE	2013	10	9	15.9	6	10.4	3	0	0	79.8	2.7	5.3
2.53E+08	ŚLIWICE	2013	10	10	16.6	7.3	10.8	4.2	0	0	83.8	2.3	5

Rys. 5.2.5 Plik zawierający wszystkie scalone dane.

Kolejnym etapem pracy była implementacja skryptu służącego do sortowania oraz selekcji danych (listing 5.2.3). Został on użyty wielokrotnie podczas procesu tworzenia modelu predykcyjnego. Skrypt wczytuje dane z pliku *csv*, wykorzystując bibliotekę *pandas* i funkcję *read_csv*. Sortuje je według kodu stacji, roku, miesiąca i dnia z wykorzystaniem funkcji *sort_values*, natomiast selekcja jest dostosowywana do potrzeb danego modelu. Wybierane są rekordy z podanej stacji oraz określone kolumny przechowujące parametry meteorologiczne. Następnie wyekstrahowane dane zostają zapisane do pliku *csv* za pomocą funkcji *to_csv*. Przykładowy plik powstały w wyniku działania skryptu został przedstawiony na rysunku 5.2.6. Zawiera dane ze stacji pogodowej Warszawa – Bielany z wybranymi pięcioma kolumnami.

```
import pandas as pd
from constants import *

df = pd.read_csv("weather.csv", encoding='windows-1250', squeeze=True)
df_sort = df.sort_values(by=['Kod stacji', 'Rok', 'Miesiac', 'Dzien'])
df_city = df_sort.loc[df_sort['Nazwa stacji'] == CITY]
df_city_chosen = df_city.iloc[:, [5, 6, 7, 11, 12]]
df_city_chosen.to_csv(DESTINATION_FOLDER + FILENAME,
encoding='windows-1250')
```

Listing 5.2.3 Kod źródłowy skryptu do sortowania i selekcji danych.

	Maksymal	Minimaln	Srednia te	Srednia d	Srednia dobow
22407	-2.5	-4.1	-3.6	92.5	2
22408	-2.6	-3.7	-3.4	93.5	1.7
22409	-3.6	-11.8	-8.6	85.8	1.7
22410	-5.5	-14.9	-9.1	87	0.7
22411	-3.2	-9.4	-6	86.5	1.7
22412	-5.9	-11.8	-8.4	87.8	2.3
22413	-5	-7.7	-6.3	86.8	1
22414	-4.2	-7.2	-6.2	84.5	1.3
22415	-3.8	-6.6	-5.1	85.5	3.3

Rys. 5.2.6 Plik z wybranymi danymi ze stacji pogodowej Warszawa – Bielany.

5.3. Przetwarzanie danych na potrzeby modelu LSTM

Dane, które będą wykorzystywane przez model LSTM są zapisane w pliku o formacie *csv*. Aby je pozyskać została wykorzystana funkcja *read_csv* (listing 5.3.1). Następnie dla wczytanego zbioru danych wywoływana jest funkcja *values*, która zwraca listę wszystkich wartości w zbiorze oraz funkcja *astype()* przyjmująca jako argument 'float32', co powoduje konwersję danych do typu *float*.

```
dataset = read_csv(DESTINATION_FOLDER + FILENAME, header=0,
index_col=0)
values = dataset.values

# ensure all data is float
values = values.astype('float32')
```

Listing 5.3.1 Wczytanie i konwersja danych.

Kolejnym krokiem przetwarzania danych jest ich normalizacja (listing 5.3.2), czyli takie przekształcenie zbioru, aby wszystkie wartości były przeskalowane do zakresu od 0 do 1. W tym celu zastosowano obiekt *MinMaxScaler* z biblioteki *sklearn*. Obiekt scaler zostaje zapisany do pliku po to, żeby móc później znormalizować dane aktualne w ten sam sposób.

```
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
dump(scaler, 'scalers/scaler-' + CITY + '.joblib')
```

Listing 5.3.2 Normalizacja danych.

Na potrzeby modelu predykcyjnego zaimplementowano funkcję *series_to_supervised* (listing 5.3.3), której zadaniem jest tworzenie szeregu czasowego. Jej parametry wejściowe to zbiór danych, liczba kroków opóźnionych, liczba kroków prognozowanych oraz wartość logiczna wskazująca, czy funkcja ma usuwać wiersze z brakującymi wartościami. Tworzenie szeregu czasowego odbywa się dzięki funkcji *shift()*, która tworzy kopię kolumn przesuniętych w przód lub wstecz o zadany krok czasowy. Funkcja zostaje wywoływana w pętli *for* i zaczyna swoje działanie, przyjmując jako parametr liczbę kroków opóźnionych. Dla przykładu gdy liczba kroków opóźnionych jest równa 2, funkcja *shift()* tworzy kopię kolumny z wartościami przesuniętymi o 2 w tył (dodając na początku wiersze z wartościami NaN). Ponadto każdej utworzonej kolumnie nadawany jest nagłówek „var” z numerem cechy oraz oznaczeniem kroku czasowego – w tym przypadku „t-2”. Przy następnym obiegu pętli funkcja *shift()* przyjmuje wartość 1, tworząc kopię kolumny z wartościami

przesuniętymi o 1 w tył. Dodawanie wartości prognozowanej odbywa się na podobnej zasadzie z tą różnicą, że licznik w pętli *for* jest inkrementowany, a argument funkcji *shift()* przyjmuje wartość przeciwną do licznika (np. -2), dzięki czemu tworzona jest kopia kolumny z wartościami przesuniętymi w przód (dodając na końcu wiersze z wartościami NaN). Oznaczenie kroków czasowych w przypadku wartości prognozowanych rozpoczyna się od „t”, później „t+1” aż do „t+n”. Po utworzeniu wszystkich kopii kolumn zostają one połączone w jeden zbiór oraz usuwane są wiersze, które zawierają wartości NaN.

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in
range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Listing 5.3.3 Implementacja funkcji *series_to_supervised*.

Na rysunku 5.3.1 przedstawiono przykładowy zbiór danych przed wywołaniem funkcji *series_to_supervised* (listing 5.3.4), a na rysunku 5.3.2 zbiór powstały w wyniku działania tej funkcji wywołanej dla znormalizowanego zbioru danych z krokiem opóźnienia 2 oraz krokiem prognozowania 2.

```
reframed = series_to_supervised(scaled, 2, 2)
```

Listing 5.3.4 Wywołanie funkcji *series_to_supervised*.

÷ 0	÷ 1
0.30105	0.06033
0.30526	0.03839
0.20632	0.00183
0.17895	0.00000
0.23579	0.00000
0.22316	0.08775
0.26526	0.00000
0.24632	0.15722
0.29684	0.12980
0.30737	0.18282

Rys. 5.3.1 Dane przed przekształceniem przez funkcję *series_to_supervised*.

÷ var1(t-2)	÷ var2(t-2)	÷ var1(t-1)	÷ var2(t-1)	÷ var1(t)	÷ var2(t)	÷ var1(t+1)	÷ var2(t+1)
0.30105	0.06033	0.30526	0.03839	0.20632	0.00183	0.17895	0.00000
0.30526	0.03839	0.20632	0.00183	0.17895	0.00000	0.23579	0.00000
0.20632	0.00183	0.17895	0.00000	0.23579	0.00000	0.22316	0.08775
0.17895	0.00000	0.23579	0.00000	0.22316	0.08775	0.26526	0.00000
0.23579	0.00000	0.22316	0.08775	0.26526	0.00000	0.24632	0.15722
0.22316	0.08775	0.26526	0.00000	0.24632	0.15722	0.29684	0.12980
0.26526	0.00000	0.24632	0.15722	0.29684	0.12980	0.30737	0.18282
0.24632	0.15722	0.29684	0.12980	0.30737	0.18282	0.31368	0.01097
0.29684	0.12980	0.30737	0.18282	0.31368	0.01097	0.30526	0.00914
0.30737	0.18282	0.31368	0.01097	0.30526	0.00914	0.24632	0.00000

Rys. 5.3.2 Dane po przekształceniu przez funkcję *series_to_supervised*.

Funkcja *series_to_supervised* tworzy kopię wszystkich dostępnych kolumn z cechami. Dane opóźnione są potrzebne jako dane wejściowe, jednak nie wszystkie parametry mają być prognozowane, dlatego należy usunąć niepotrzebne kolumny, zostawiając jedynie tę, która zawiera prognozowaną wielkość (listing 5.3.5).

```
reframed.drop(reframed.columns[[25, 26, 28, 29]], axis=1,
inplace=True)
```

Listing 5.3.5 Usunięcie kolumn, które nie będą prognozowane.

Zbiór danych należy podzielić na dane treningowy służące do trenowania modelu oraz dane testowe służące do przetestowania modelu i oceny jego dokładności. Przy tworzeniu wstępnych modeli do zbioru treningowego przydzielano pierwsze 3500 wierszy, a pozostałe dane do zbioru testowego (listing 5.3.6). Jednak później zastosowano losowy podział (listing 5.3.7), co sprawiło, że modele osiągały wyższą dokładność, dlatego ten rodzaj podziału wdrożono w modelu ostatecznym.

```
train = values[:TRAIN_SIZE, :]
test = values[TRAIN_SIZE:, :]
```

Listing 5.3.6 Podział na zbiór treningowy i testowy.

```
X = values[:, :N_OBS]
y = values[:, -N_FEATURES]
train_X, test_X, train_y, test_y = train_test_split(X, y,
test_size=0.15, random_state=42)
```

Listing 5.3.7 Losowy podział na zbiór treningowy i testowy.

Kolejnym krokiem przetwarzania danych był podział na wartości wejściowe (wszystkie kolumny oprócz ostatniej) i wartość prognozowaną (ostatnia kolumna). Kod, który posłużył do tego celu przedstawiono na listingu 5.3.8. Model LSTM wymaga, aby zbiór danych był w wymiarze 3D, dlatego zbiory treningowy i testowy przekształcono do takiego wymiaru – pierwszy wymiar to próbki, drugi kroki czasowe, a trzeci cechy (listing 5.3.9).

```
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
```

Listing 5.3.8 Podział na wartości wejściowe i prognozowane.

```
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], N_DAYS, N_FEATURES))
test_X = test_X.reshape((test_X.shape[0], N_DAYS, N_FEATURES))
```

Listing 5.3.9 Przekształcenie zbioru do wymiaru 3D.

5.4. Implementacja modelu LSTM

Dla danych przygotowanych według kroków opisanych w poprzednim podrozdziale, można przystąpić do tworzenia modelu LSTM, wykorzystując bibliotekę Keras.

Pierwszym krokiem jest zdefiniowanie modelu (listing 5.4.1). Sieć neuronowa w bibliotece Keras to sekwencja warstw. Kontenerem tych warstw jest klasa *Sequential*, dlatego należy powołać jej instancję. Następnie można utworzyć połączone ze sobą warstwy. Warstwy ukryte LSTM (do przetwarzania danych) składają się z jednostek pamięci. Warstwa wyjściowa wykorzystywana do predykcji nazywa się *Dense()*. W pracy zdefiniowano ukrytą warstwę LSTM składającą się z 50 komórek pamięci. Jako argument wejściowy *input_shape* podano liczbę kroków czasowych oraz liczbę cech. Następnie zdefiniowano warstwę wyjściową z jednym neuronem.

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
```

Listing 5.4.1 Zdefiniowanie modelu LSTM.

Po zdefiniowaniu modelu należy go skompilować (listing 5.4.2). Ten proces ma na celu przekształcenie prostej sekwencji warstw w wysoce wydajną serię macierzy o formacie możliwym do uruchomienia przez procesor lub kartę graficzną. Kompilacja

wymaga podania parametrów – *optimizer* określa algorytm optymalizacji, a *loss* funkcję straty.

```
model.compile(loss='mae', optimizer='adam')
```

Listing 5.4.2 Kompilacja modelu LSTM.

Gdy model LSTM zostanie skompilowany, może zostać wytrenowany (listing 5.4.3) na zbiorze treningowym. W tym celu należy podać macierz danych wejściowych oraz odpowiadającą mu listę danych wyjściowych. Sieć jest trenowana z wykorzystaniem algorytmu wstecznej propagacji, a jej optymalizacja przebiega zgodnie z algorytmem zdefiniowanym podczas kompilacji. Algorytm wstecznej propagacji wymaga wytrenowania na określonej liczbie okresów definiowanych przez parametr *epochs*. Każdy okres jest dzielony na grupy par wzorców wejścia – wyjścia zwanych partiami. Określa to ilość wzorców wystawionych przez sieć przed aktualizacją wag w okresie. Poprawia to także wydajność, zapewniając aby do pamięci nie było załadowane zbyt dużo wzorców. Przy trenowaniu modelu użytego w pracy zdefiniowano rozmiar okresu *batch_size* równy 72. Ustawienie argumentu *verbose* na wartość równą 2 powoduje redukcję ilości wyświetlanych informacji w konsoli podczas trenowania modelu. Z kolei argument *shuffle* ustawiony na wartość *false* oznacza, że kolejność próbek ma zostać zachowana.

```
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72,
validation_data=(test_X, test_y), verbose=2, shuffle=False)
```

Listing 5.4.3 Trenowanie modelu LSTM.

Po wytrenowaniu modelu zostaje on zapisany do pliku (listing 5.4.4), aby móc go później użyć do prognozowania na podstawie aktualnych danych meteorologicznych.

```
# save model to single file
model.save('models/lstm_model-' + CITY + '.h5')
```

Listing 5.4.4 Zapisanie modelu LSTM do pliku.

Przygotowany model można użyć do predykcji (listing 5.4.5). Wykorzystuje się do tego celu funkcję *predict*, podając jako argument dane, dla których ma być wykonana prognoza.

```
# make a prediction
yhat = model.predict(test_X)
```

Listing 5.4.5 Wykonanie predykcji z wykorzystaniem utworzonego modelu.

Aby wynik prognozy był czytelny konieczny jest powrót z wartości znormalizowanych do wartości pierwotnych (listing 5.4.6). W tym celu należy

wykorzystać wcześniej utworzony obiekt scaler. Jednak aby było to możliwe konieczne jest przekształcenie zbioru tak, aby był w takiej samej formie co zbiór danych poddawany wcześniej normalizacji. Najpierw trójwymiarowy zbiór testowy z wartościami wejściowymi należy zamienić na dwuwymiarowy. Następnie łączone są wartości prognozowane przez model z kolumnami reprezentującymi pozostałe parametry meteorologiczne. W ten sposób powstaje zbiór *inv_yhat* składający się z pięciu kolumn przechowujących pięć parametrów meteorologicznych wykorzystywanych przez model, w którym nie ma szeregu czasowego. Różnica między tym zbiorem, a pierwotnym jest taka, że kolumna z średnią temperaturą dobową została zamieniona na wartość tej temperatury prognozowaną przez model. Dla takiego zbioru możliwe jest wywoływanie funkcji *inverse_transform*, co powoduje zmianę wartości znormalizowanych na pierwotne. Na koniec wycinana jest kolumna z prognozowaną średnią temperaturą, gdyż tylko ona będzie potrzebna, pozostałe konieczne były jedynie do zastosowania skalera. Transformacji należy również dokonać dla wartości oczekiwanych ze zbioru testowego. Odbywa się to w sposób analogiczny z tą różnicą, że zamiast wartości prognozowanej używane są wartości oczekiwane.

```
test_X = test_X.reshape((test_X.shape[0], N_DAYS*N_FEATURES))

# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(N_FEATURES-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]
# # invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -(N_FEATURES-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]
```

Listing 5.4.6 Przekształcenie wartości znormalizowanych w pierwotne.

W celu wizualizacji wyników (listing 5.4.7) napisano pętlę *for*, która powoduje wyświetlenie wartości prognozowanej i oczekiwanej dla pierwszych stu próbek. Z pomocą modułu *pyplot* z biblioteki *matplotlib* stworzono wykresy przedstawiające porównanie wartości prognozowanych i oczekiwanych. Pierwszy wykres obejmuje wszystkie próbki, drugi tylko część próbek w celu poprawy jego czytelności. Obliczono także pierwiastek błędu średniokwadratowego (RMSE), wykorzystując do tego celu moduł *mean_squared_error* z biblioteki *sklearn*, a wynik wyświetlono w konsoli.

```
for t in range(100):
    print('predicted=%f, expected=%f' % (inv_yhat[t], inv_y[t]))
# plot forecasts against actual outcomes
pyplot.plot(inv_y, label='wartość oczekiwana')
pyplot.plot(inv_yhat, color='red', label='wartość prognozowana')
pyplot.legend()
```

```

pyplot.xlabel('Numer próbki')
pyplot.ylabel('Średnia temperatura dobową °C')
pyplot.title('Wykres wartości prognozowanej średniej temperatury
dobowej')
pyplot.show()

inv_y_cut = inv_y[350:]
inv_yhat_cut = inv_yhat[350:]
pyplot.plot(inv_y_cut, label='wartość oczekiwana')
pyplot.plot(inv_yhat_cut, color='red', label='wartość prognozowana')
pyplot.legend()
pyplot.xlabel('Numer próbki')
pyplot.ylabel('Średnia temperatura dobową °C')
pyplot.title('Wykres wartości prognozowanej średniej temperatury
dobowej')
pyplot.show()
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

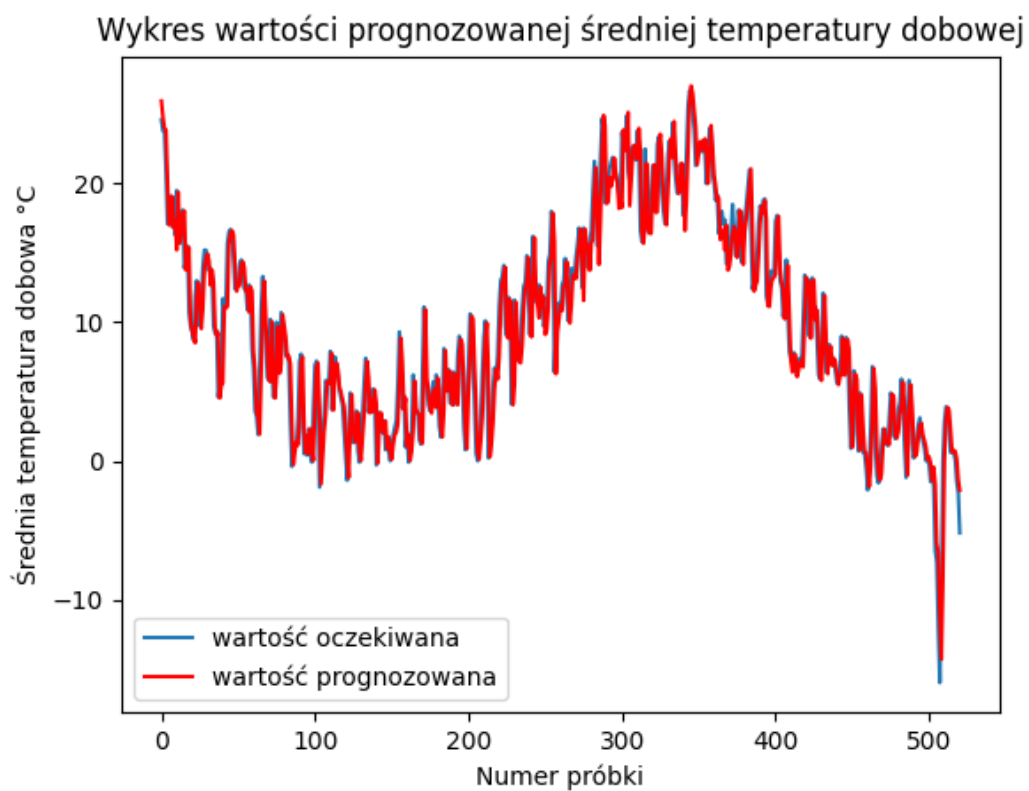
```

Listing 5.4.7 Wizualizacja wyników predykcji.

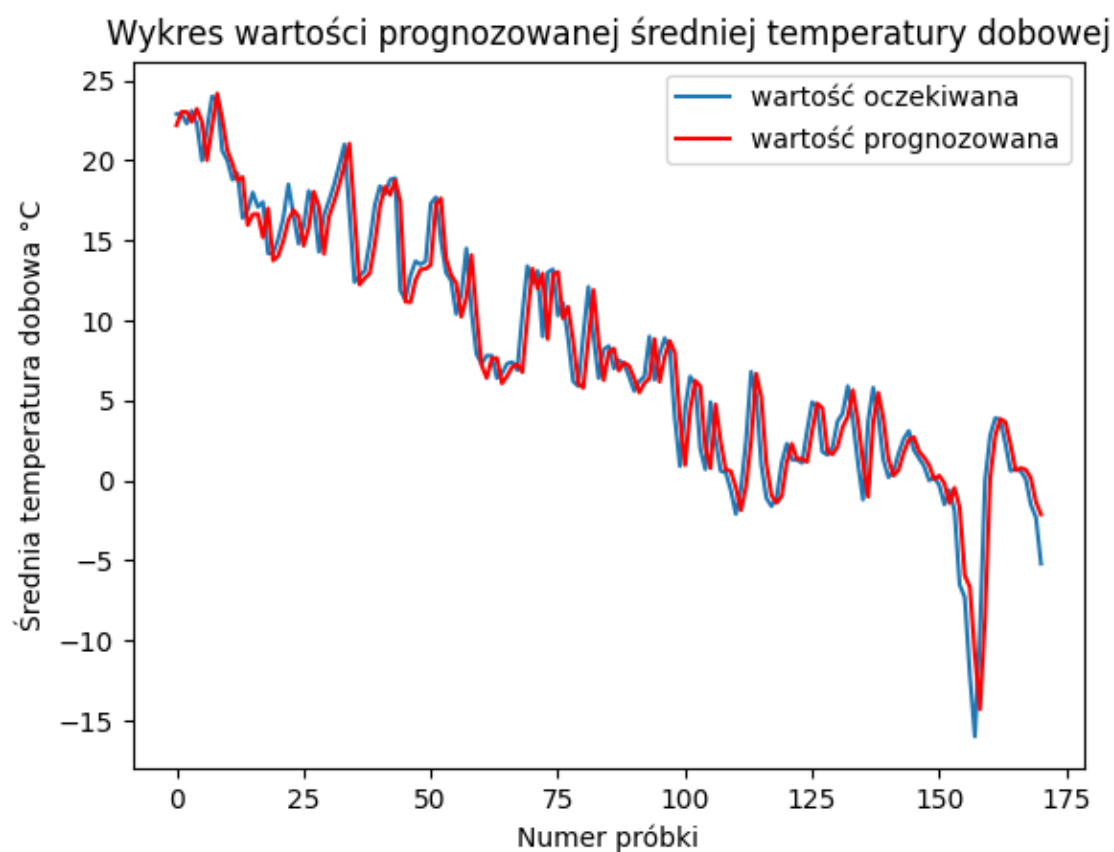
5.5. Proces tworzenia modelu LSTM

Niniejszy rozdział przedstawia proces tworzenia modelu predykcyjnego, który osiągnie wysoką dokładność oraz będzie dostosowany zarówno do historycznych danych meteorologicznych jak i danych aktualnych pobieranych z API.

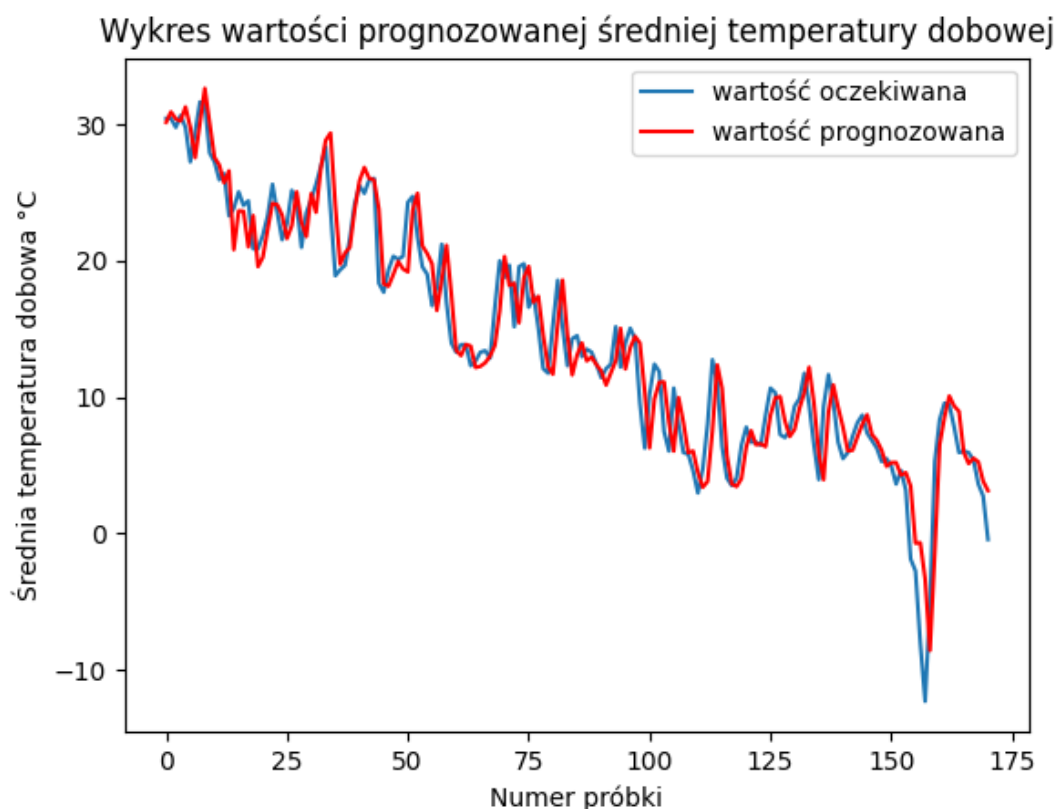
Pierwszy wstępnie wykonany model miał na celu przede wszystkim zapoznanie się z *Long – Short Term Memory Networks* oraz naukę poprawnej implementacji tej sieci. Wykorzystano dane ze stacji pogodowej Warszawa – Filtry, wybierając tylko dwie kolumny: średnią dobową temperaturę [°C] oraz sumę dobową opadów [mm]. Predykcji dokonano na podstawie danych z jednego dnia, a wartością prognozowaną była średnia dobową temperaturę na następny dzień. Przy powyższych danych wartość RMSE była równa 2.319, co przy tak małej liczbie danych wejściowych jest stosunkowo dobrym wynikiem. Na rysunkach 5.5.1, 5.5.2 przedstawiono wykresy wartości prognozowanych w stosunku do oczekiwanych. Widać na nich, że pokrywają się tylko częściowo. Na rysunku 5.5.3 zilustrowano fragment wydruku konsoli wyświetlający porównanie wartości prognozowanych i oczekiwanych. Między pierwszymi wyświetlonymi wynikami jest duża różnica – aż 5 °C, dla kolejnych wierszy różnice są mniejsze.



Rys. 5.5.1 Wykres wartości prognozowanej średniej temperatury dobowej – Warszawa – Filtry.



Rys. 5.5.2 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Warszawa – Filtry.

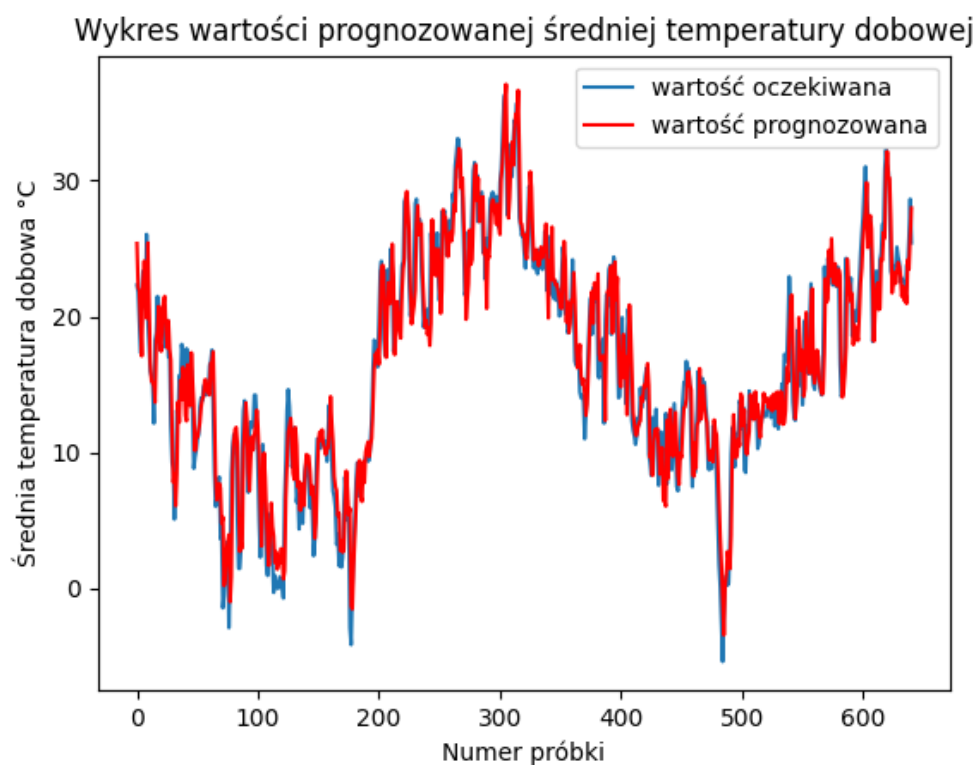


Rys. 5.5.5 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Warszawa – Filtry (wszystkie parametry).

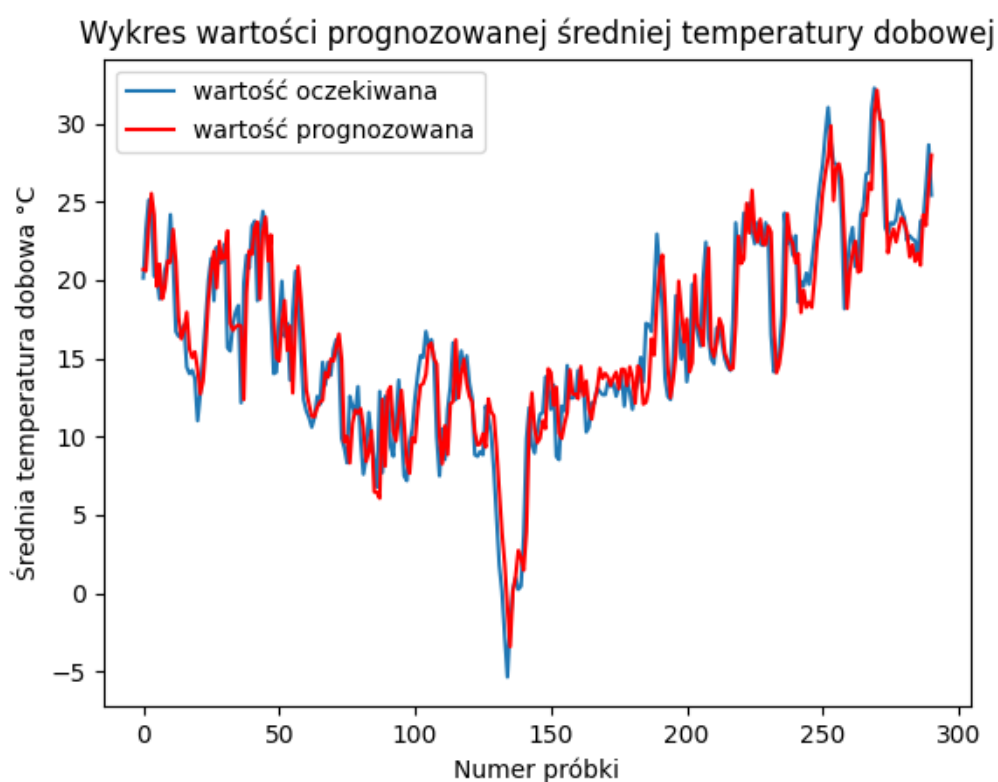
W modelu drugim również zastosowano dane ze stacji klimatologicznej Warszawa – Filtry z jednego dnia z tą różnicą, że zebrano wszystkie dostępne parametry meteorologiczne: maksymalną temperaturę dobową [°C], minimalną temperaturę dobową [°C], średnią temperaturę dobową [°C], temperaturę minimalną przy gruncie [°C], sumę dobową opadów [mm], wysokość pokrywy śnieżnej [cm], średnią dobową wilgotność względną [%], średnią dobową prędkość wiatru [m/s], średnie dobowe zachmurzenie ogólne [oktanty]. Po zmierzeniu dokładności modelu za pomocą pierwiastka błędu średniokwadratowego okazało się, że działa on gorzej od poprzedniego. Wartość RMSE wyniosła 2.553. Z wykresów wartości prognozowanych i oczekiwanych (rys. 5.5.4, 5.5.5) również można odczytać, że pokrycie jest trochę gorsze niż w przypadku pierwszego modelu.

Model trzeci wykonano dla danych z innej stacji pogodowej niż poprzednio. Wybrano dane z Sieradza ze wszystkimi dostępnymi parametrami (tak jak w poprzednim modelu) z jednego dnia. Model ten osiągnął bardzo podobną dokładność (RMSE: 2.494) do tego, który pracował z takimi samymi danymi wejściowymi, ale dla stacji Warszawa – Filtry. Na rysunkach 5.5.6, 5.5.7 przedstawiono wykresy

prognozowanej średniej temperatury dobowej w porównaniu do wartości oczekiwanych dla tego modelu.

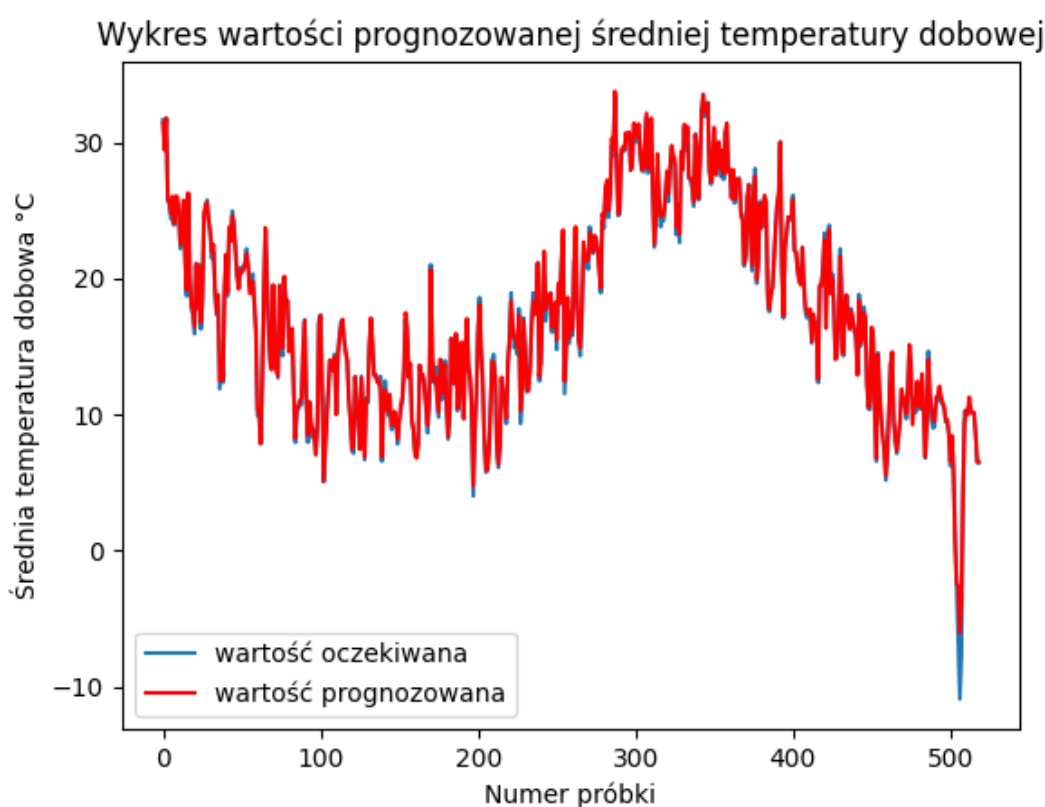


Rys. 5.5.6 Wykres wartości prognozowanej średniej temperatury dobowej – Sieradz.

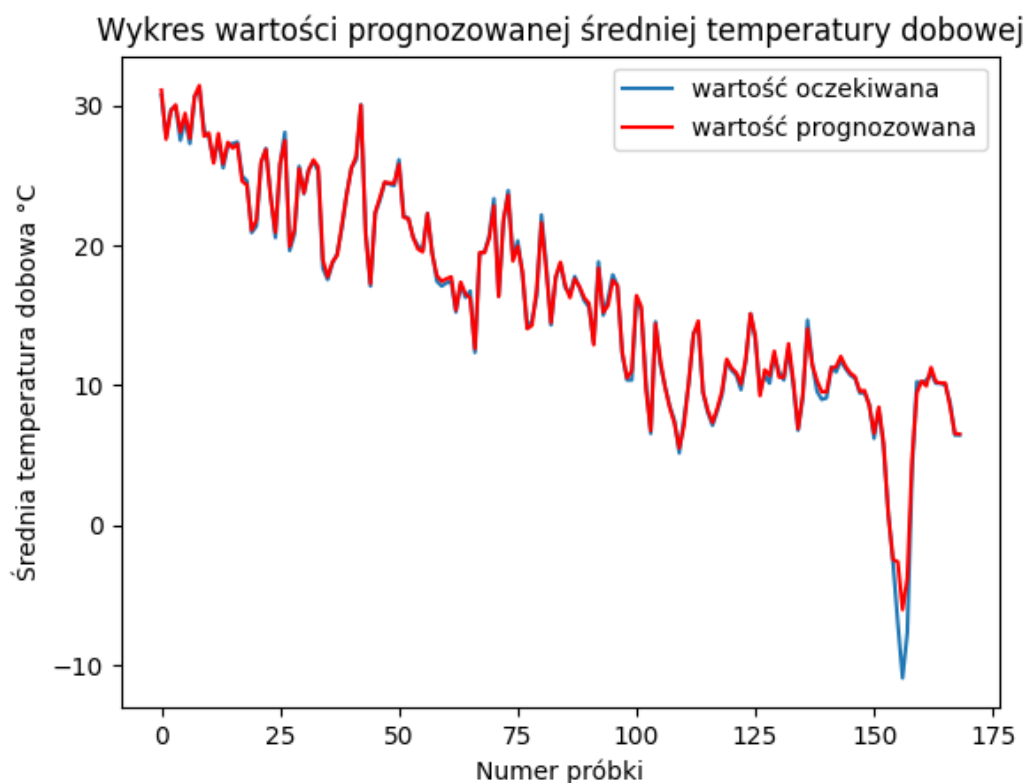


Rys. 5.5.7 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Sieradz.

W modelu czwartym wprowadzono szereg czasowy – jedna próbka danych obejmuje trzy dni. Wykorzystano ten sam zbiór co w modelu drugim, czyli dane ze stacji Warszawa – Filtry ze wszystkimi dostępnymi parametrami. Dokładność tego modelu znacząco wzrosła w stosunku do poprzednich, co pokazuje jak ważne jest rozpatrywanie szeregów czasowych w zagadnieniach z zakresu meteorologii. Pierwiastek błędu średniokwadratowego wyniósł 0.481, czyli wynik jest o ok. 2 jednostki lepszy niż w przypadku modeli, które uwzględniają dane tylko z jednego dnia. Na wykresach zilustrowanych na rysunkach 5.5.8 i 5.5.9 także widać, że wartości prognozowane dużo lepiej pokrywają się z wartościami oczekiwanymi.

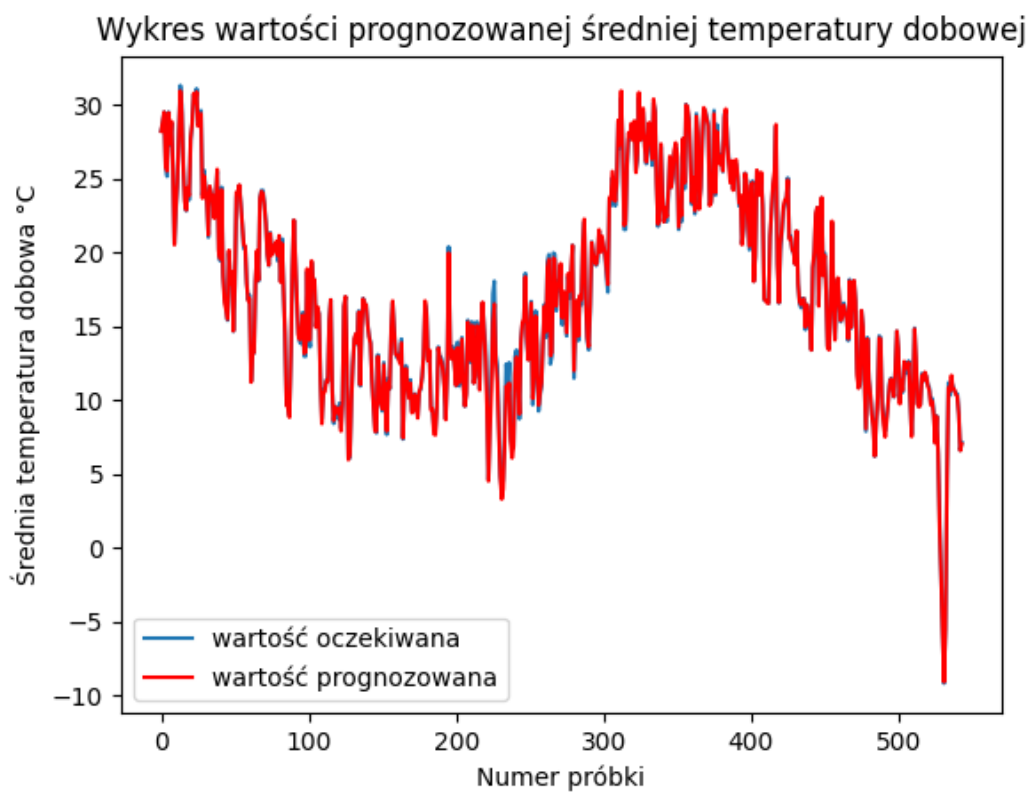


Rys. 5.5.8 Wykres wartości prognozowanej średniej temperatury dobowej dla zbioru z szeregiem czasowym – Warszawa – Filtry.

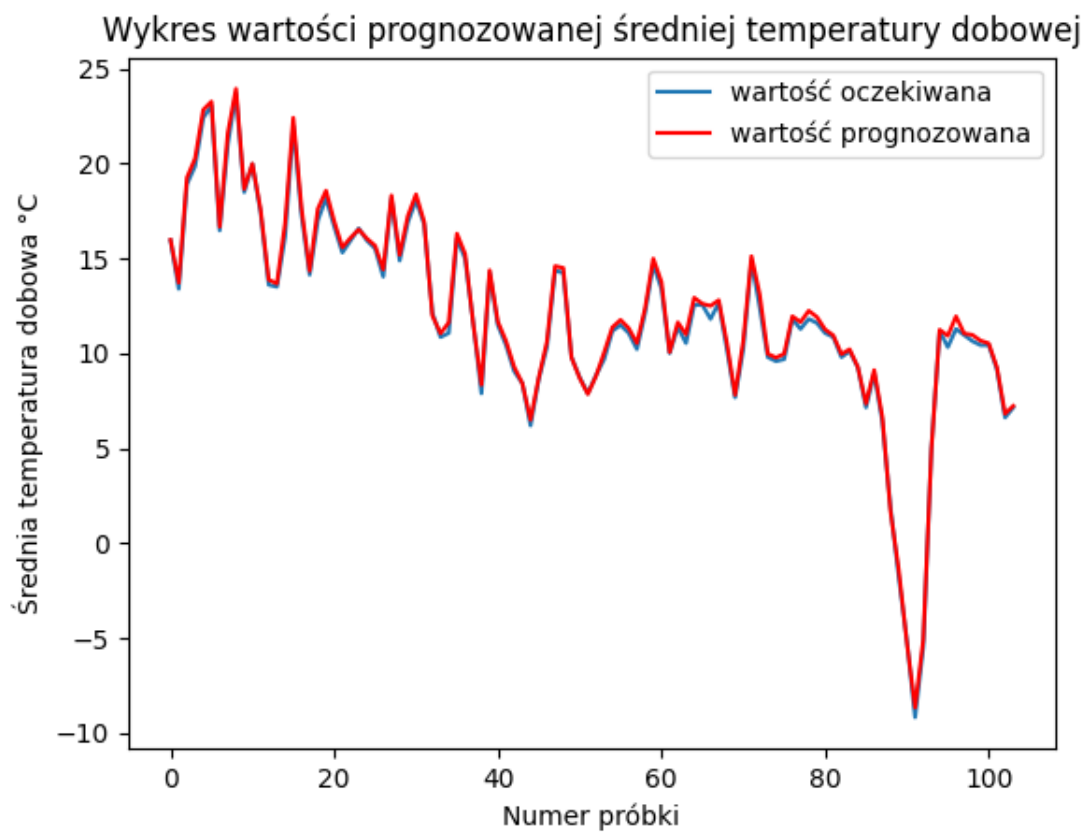


Rys. 5.5.9 Wykres wartości prognozowanej średniej temperatury dobowej dla zbioru z szeregiem czasowym – Warszawa – Filtry (wybrane próbki).

Piąty model predykcyjny został zaprojektowany tak, aby był dostosowany do danych aktualnych pobieranych z API. Szereg czasowy rozszerzono do pięciu dni, ponieważ z tyłu maksymalnie dni do tyłu możliwe jest pobranie danych aktualnych. Ponadto zredukowano ilość parametrów meteorologicznych, ponieważ dane aktualne nie zawierają takich samych informacji co dane historyczne. Po wyciągnięciu części wspólnej z obu zbiorów otrzymano następujące parametry, na których bazuje model: maksymalna temperatura dobową [°C], minimalna temperatura dobową [°C], średnia temperatura dobową [°C], średnia dobową wilgotność względną [%], średnia dobową prędkość wiatru [m/s]. Zmieniono także stację pogodową, ponieważ zauważono, że dla Warszawy – Filtry w kolumnie „Średnia dobową wilgotność względną [%]” wszędzie są wartości równe zero. Wybrano rejon Warszawa – Bielany, ponieważ parametry wymagane do pracy modelu są dla tej stacji kompletne. Dokładność modelu mierzona za pomocą pierwiastka błędu średniokwadratowego wyniosła 0.3, czyli o ok. 0.2 lepszy wynik niż w przypadku poprzedniego modelu. Wynika to zapewne z rozszerzenia szeregu czasowego z trzech do pięciu dni. Na rysunkach 5.5.10, 5.5.11 przedstawiono wykresy, na których można zaobserwować wysoce dokładne pokrycie wartości prognozowanych z oczekiwanymi.

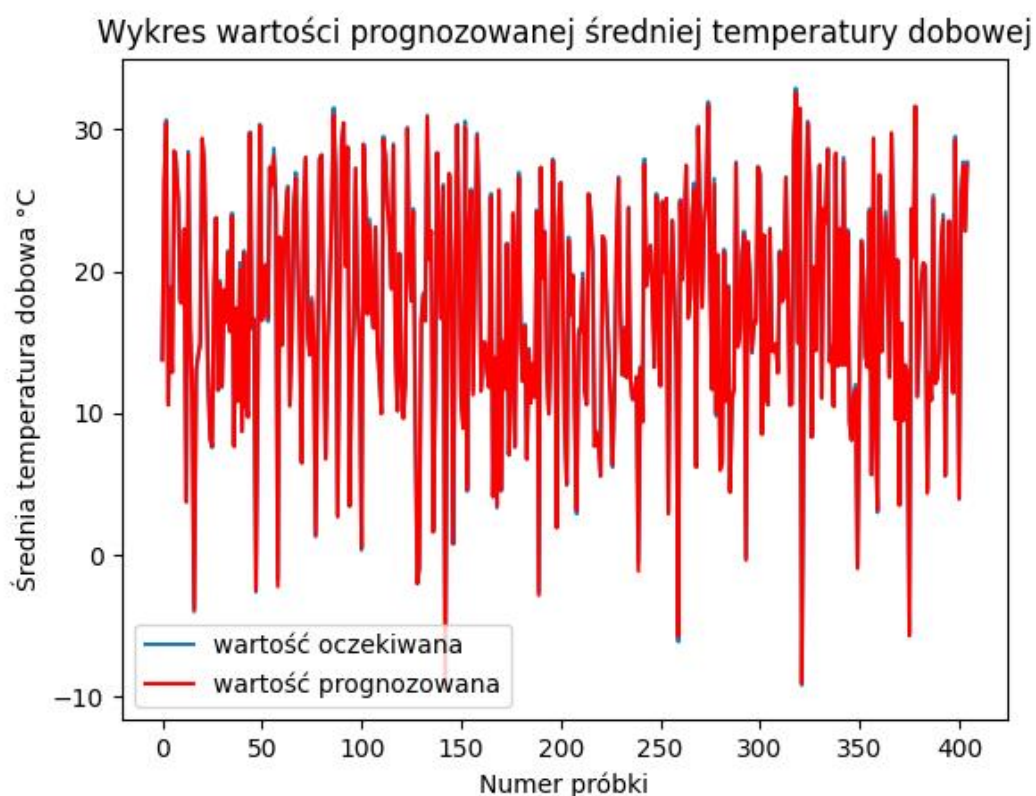


Rys. 5.5.10 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu dostosowanego do danych zwracanych przez API – Warszawa – Bielany.

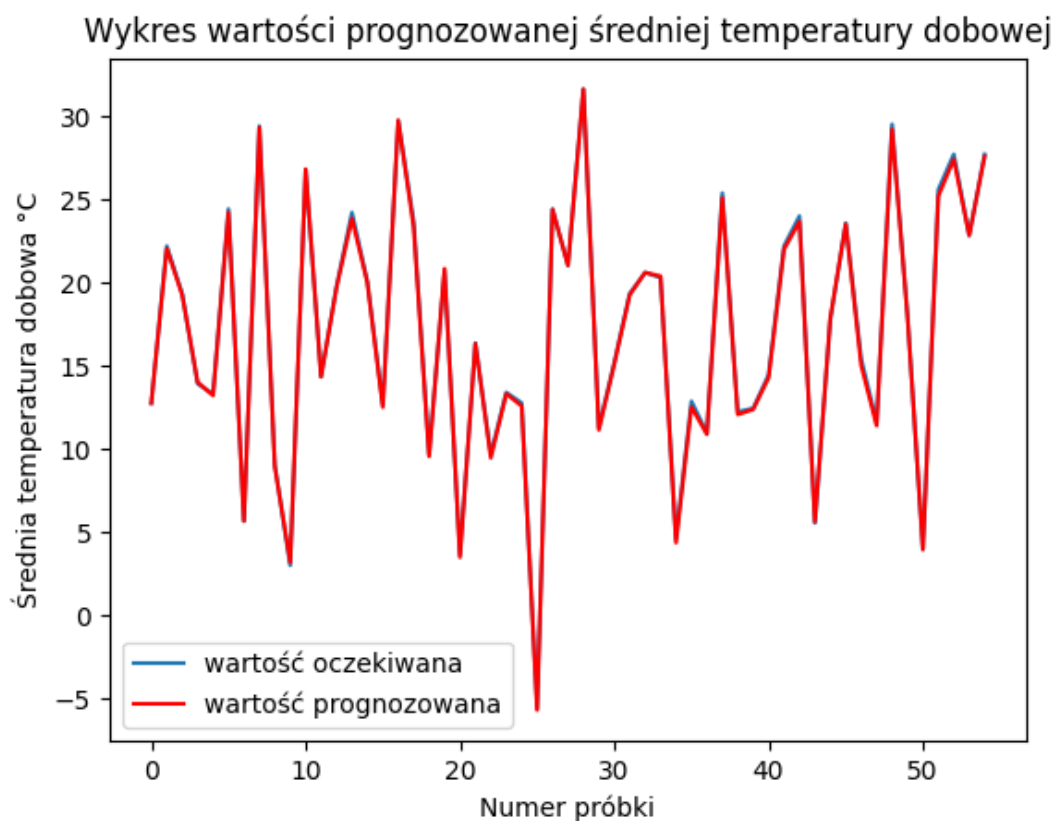


Rys. 5.5.11 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu dostosowanego do danych zwracanych przez API – Warszawa – Bielany (wybrane próbki).

We wszystkich powyższych modelach stosowano taki podział danych na zbiór treningowy i testowy, w którym pierwsze wiersze należą do zbioru treningowego, a ostatnie wiersze do testowego. Dane były sortowane chronologicznie, jednak zauważono, że na wszystkich wykresach pojawiają się nagłe spadki temperatur. Po zweryfikowaniu pierwotnych danych, nie odkryto tam takiego zjawiska, co oznacza, że próbki musiały zostać przemieszane w trakcie tworzenia modelu. Nie udało się ustalić, na jakim etapie to się dzieje, natomiast postanowiono wprowadzić losowy podział próbek już na początku przy tworzeniu zbiorów treningowych i testowych. Stwierdzono, że może się to okazać lepszym podejściem, ponieważ przy wcześniejszym podziale wszystkie najnowsze dane znajdowały się w zbiorze testowym, a z racji tego, że klimat się zmienia warto, aby w zbiorze treningowym znalazły się też dane najbardziej aktualne. Powyższa teza okazała się trafna, ponieważ model osiągnął lepszy wynik po wprowadzeniu takiego podziału. Wartość RMSE osiągnęła 0.203. Na wykresach (rys. 5.5.12, 5.5.13) również widać, że model jest bardzo dokładny. Wykresy te różnią się od poprzednich, ponieważ w tym przypadku próbki są całkowicie wymieszane, a więc pojawiają się obok siebie wartości bardzo rozbieżne.



Rys. 5.5.12 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu z wprowadzonym losowym podziałem na zbiory treningowy i testowy.



Rys. 5.5.13 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu z wprowadzonym losowym podziałem na zbiory treningowy i testowy (wybrane próbki).

Powyższy model osiągnął bardzo dobrą dokładność. Co więcej, jest dopasowany zarówno do danych historycznych, jak i danych aktualnych. Dodatkowo, wprowadzenie losowego podziału na zbiór treningowy i testowy jeszcze bardziej poprawiło jego działanie. Wyniki osiągnięte przez model są satysfakcjonujące, dlatego zdecydowano, że będzie to model ostateczny.

Na identycznej zasadzie jak dla danych ze stacji klimatologicznej Warszawa – Bielany stworzono modele dla danych z Krakowa oraz Borucino (okolice Gdańska) tak, aby uzyskać modele dla północy, centrum i południa Polski.

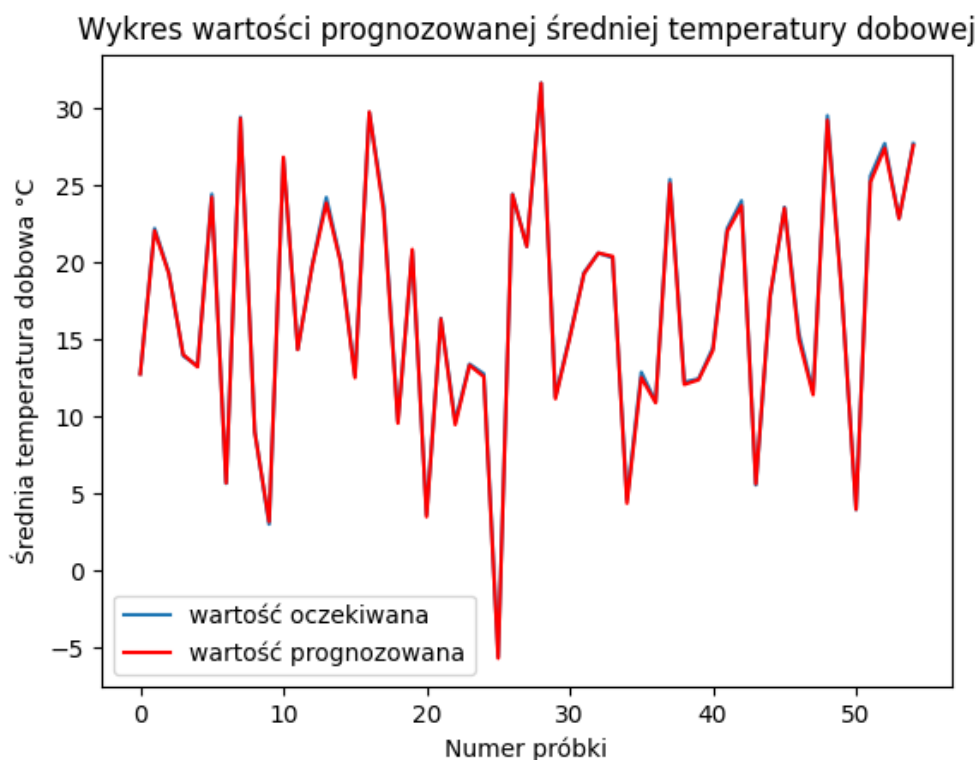
Powyższe modele pozwalają prognozować średnią dobową temperaturę na dzisiaj. Wynika to z tego, że pracują z danymi uśrednionymi dobowo, a więc można je uruchomić dopiero po zebraniu danych z całego poprzedniego dnia. Postanowiono zaprojektować dodatkowe modele prognozujące temperaturę na następny dzień. Aby to osiągnąć, wystarczyło wprowadzić w implementacji poprzedniego modelu tylko jedną zmianę. Zamiast tworzyć kopię kolumny z wartościami średniej temperatury dobowej przesuniętą o 1 dzień do przodu, należało utworzyć kopię kolumny przesuniętą o 2 dni.

W tabeli 5.5.1 przedstawiono wartości pierwiastka błędu średniokwadratowego dla wszystkich sześciu modeli – dla trzech miast na dziś i dzień następny. Najlepszy wynik osiągnął model prognozujący temperaturę na jutro dla stacji Borucino – 0.164, najgorszy na jutro dla Warszawy – Bielany – 0.399. Wszystkie otrzymane wyniki są zadowalające i oscylują wokół wartości 0.2 – 0.3.

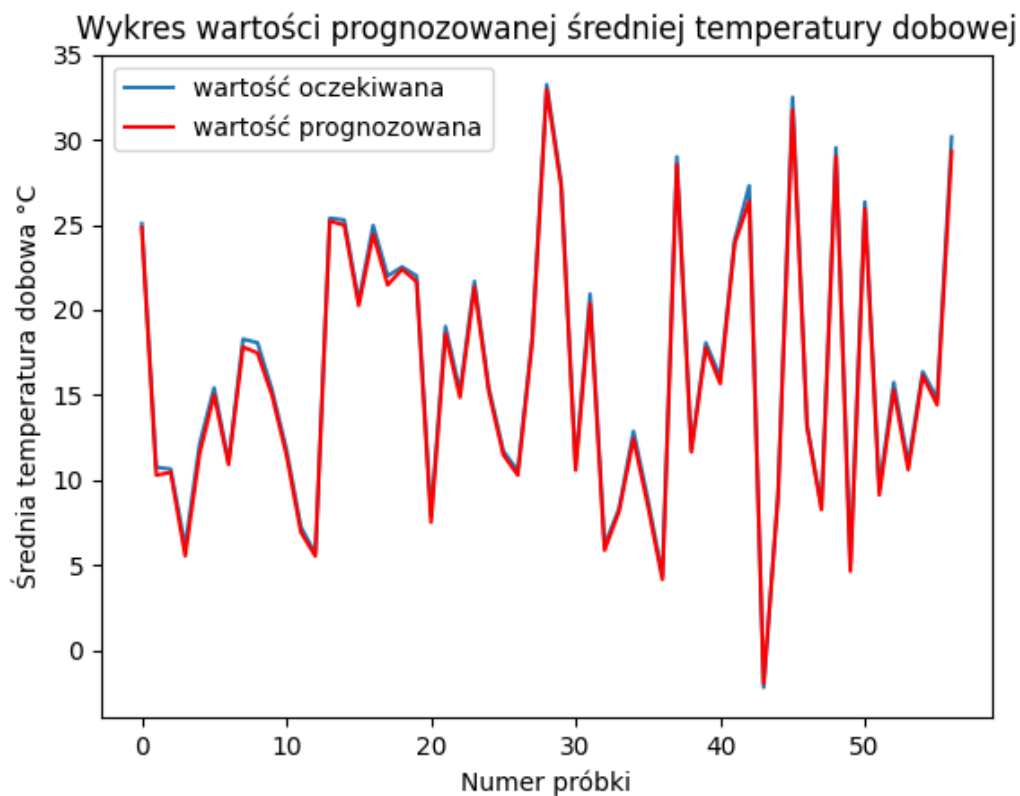
Stacja\Dzień	Dzisiaj	Jutro
Warszawa - Bielany	RMSE: 0.203	RMSE: 0.399
Kraków	RMSE: 0.187	RMSE: 0.224
Borucino	RMSE: 0.241	RMSE: 0.164

Tabela 5.5.1 Tabela z wartościami RMSE dla wykonanych modeli.

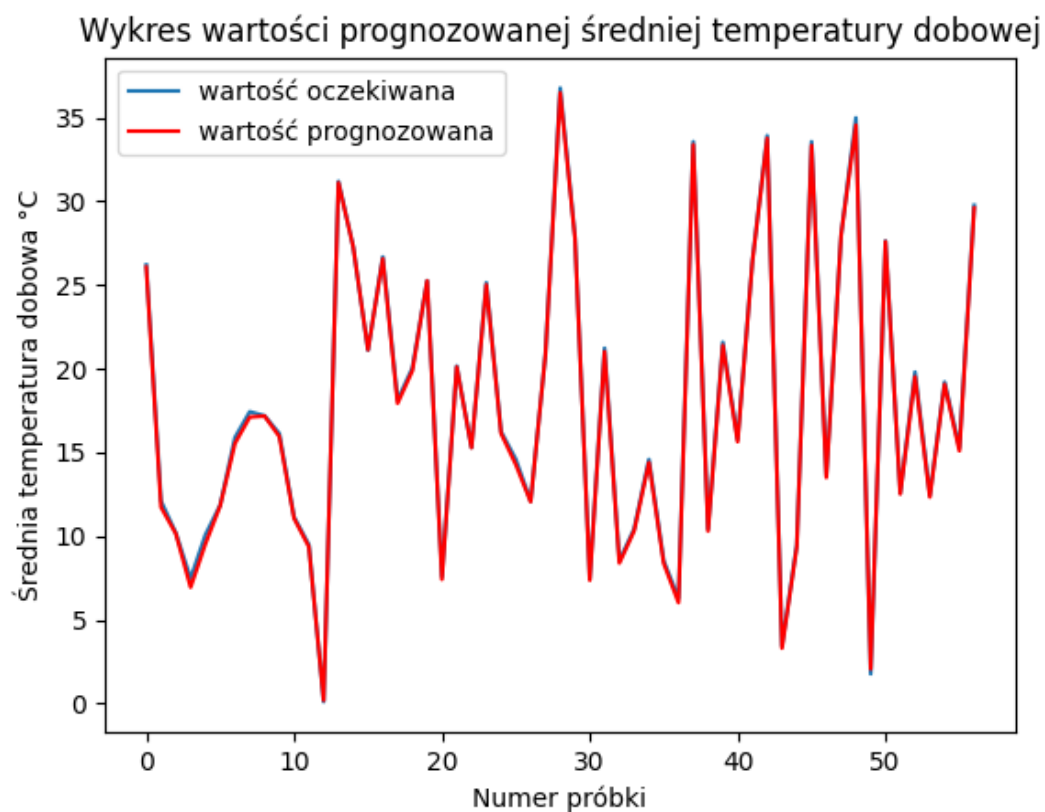
Na rysunkach 5.5.14-19 zaprezentowano wykresy dla wszystkich ostatecznie wykorzystanych w pracy modeli. Analizując je można zauważyć, że rzeczywiście najgorzej sprawdził się model prognozy na jutro dla Warszawy – Bielany, widać lekkie rozbieżności wartości między prognozowanymi a oczekiwanymi w kilku miejscach wykresu. W przypadku pozostałych wykresów wartości prognozowane niemalże całkowicie pokrywają się z wartościami oczekiwanymi.



Rys. 5.5.14 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Warszawa – Bielany.

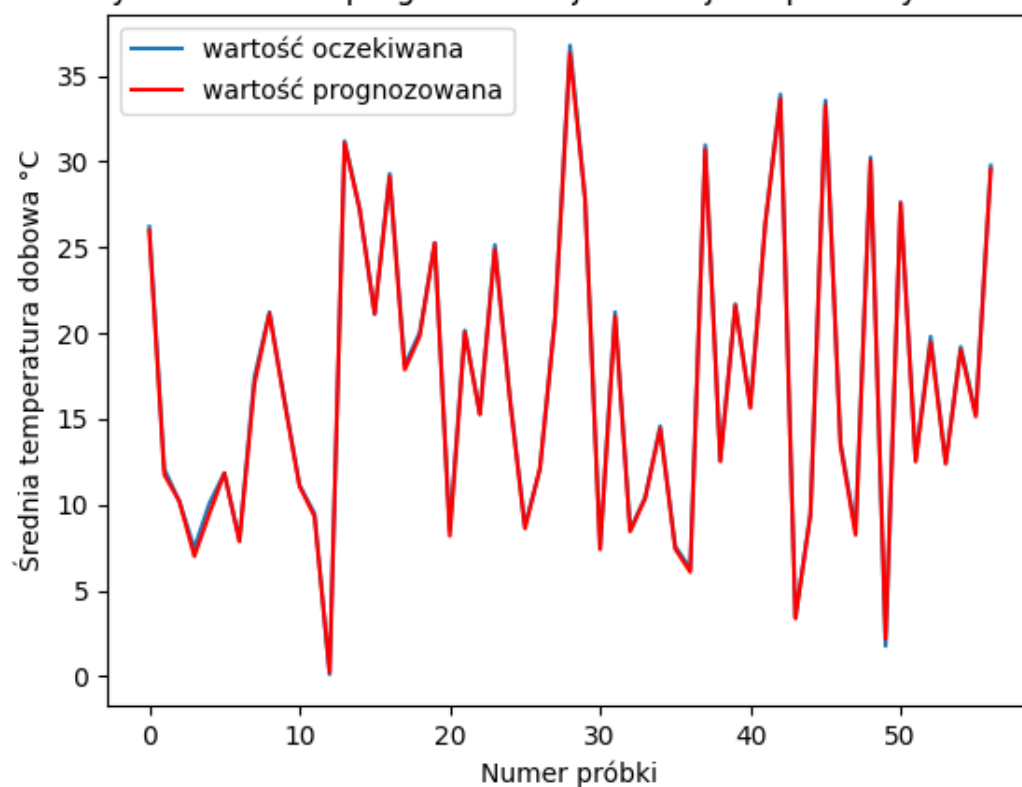


Rys. 5.5.15 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Warszawa – Bielany.



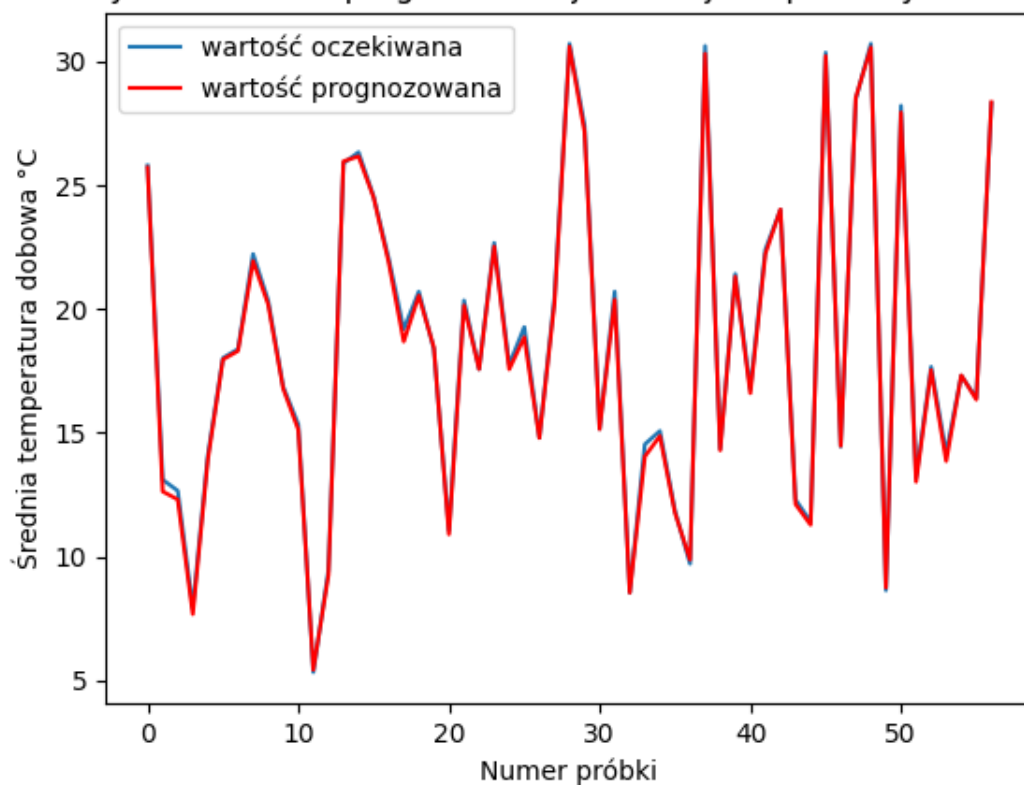
Rys. 5.5.16 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Kraków.

Wykres wartości prognozowanej średniej temperatury dobowej

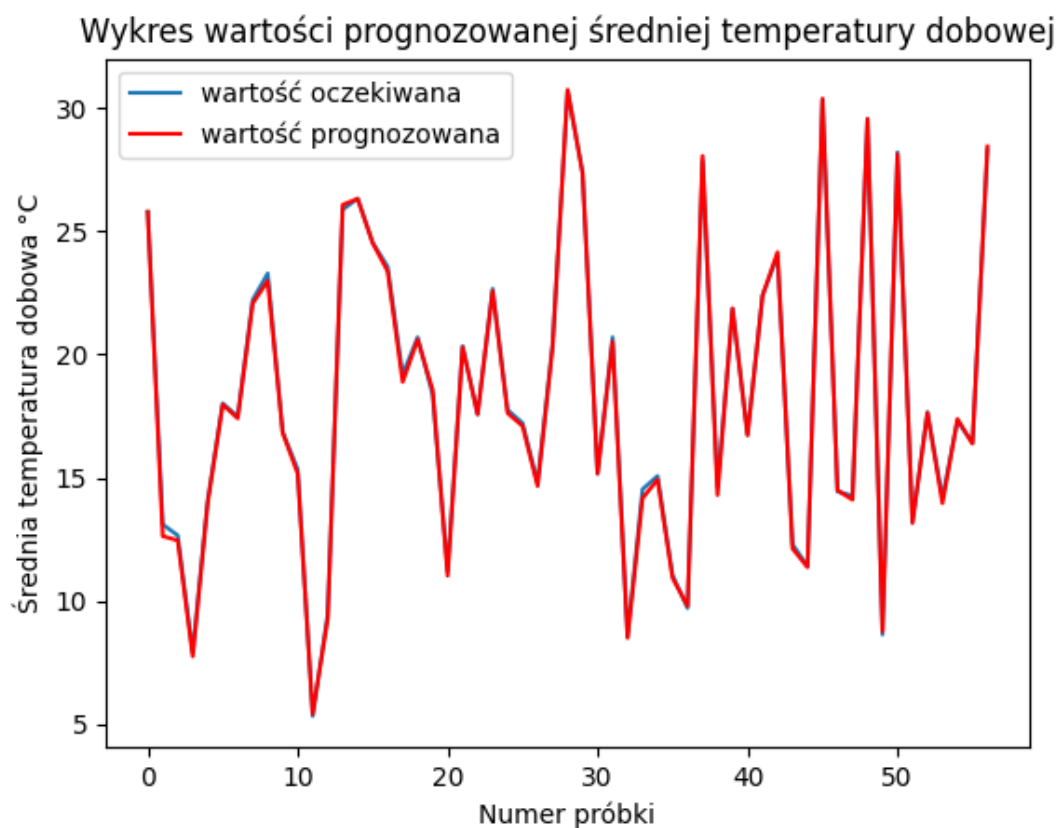


Rys. 5.5.17 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Kraków.

Wykres wartości prognozowanej średniej temperatury dobowej



Rys. 5.5.18 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Gdańsk.



Rys. 5.5.19 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Gdańsk.

5.6. Zastosowanie modeli predykcyjnych dla aktualnych danych

Aktualne dane meteorologiczne są pobierane z wykorzystaniem Open Weather Map API. Pozwala ono uzyskać dane meteorologiczne z całego świata z maksymalnie pięciu poprzednich dni. Są one zwracane w formacie JSON. Parametry meteorologiczne udostępniane przez API to temperatura, temperatura odczuwalna, wilgotność względna, ciśnienie, kierunek i prędkość wiatru. W odróżnieniu od historycznych danych wykorzystywanych w pracy, które są uśrednione dobowo, dane pobierane z API są godzinne.

Na listingu 5.6.1 przedstawiono kod źródłowy funkcji *get_actual_weather_data*, którą zaimplementowano w celu pobierania danych z API oraz przekształcenia ich do takiej formy, aby można było wykonać dla nich predykcję.

```

import http.client
import ast
from datetime import datetime, timedelta, timezone

def get_actual_weather_data(lat, lon):
    list = []
    conn = http.client.HTTPSConnection("community-open-weather-
map.p.rapidapi.com")

    headers = {
        'x-rapidapi-key':
"f7cbbc33b86msh15d74892fcf8b61p14f6c8jsn5ac8c369d4d2",
        'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com"
    }
    for j in range(5, 0, -1):
        day_before = datetime.now() - timedelta(j)
        timestamp =
int(day_before.replace(tzinfo=timezone.utc).timestamp())

        conn.request("GET", "/onecall/timemachine?lat=" + str(lat) +
"&lon=" + str(lon) + "&dt=" + str(timestamp),
                    headers=headers)

        res = conn.getresponse()
        data = res.read()
        dict = data.decode("utf-8")
        dict = ast.literal_eval(dict)
        min_temp = 330
        max_temp = 0
        sum = 0
        humidity_sum = 0
        wind_speed_sum = 0
        for i in range(0, 24):
            sum += dict["hourly"][i]["temp"]
            humidity_sum += dict["hourly"][i]["humidity"]
            wind_speed_sum += dict["hourly"][i]["wind_speed"]
            if (dict["hourly"][i]["temp"] < min_temp):
                min_temp = dict["hourly"][i]["temp"]
            if (dict["hourly"][i]["temp"] > max_temp):
                max_temp = dict["hourly"][i]["temp"]

        avg = sum / 24
        min_temp = min_temp - 273.15
        max_temp = max_temp - 273.15
        avg = avg - 273.15
        avg_humidity = humidity_sum / 24
        avg_wind_speed = wind_speed_sum / 24
        list += [max_temp, min_temp, avg, avg_humidity,
avg_wind_speed]
    return list

```

Listing 5.6.1 Kod źródłowy funkcji `get_actual_weather_data`.

Funkcja `get_actual_weather_data` przyjmuje dwa parametry wejściowe *lat* i *lon*, określające współrzędne geograficzne miejsca, dla którego będą pobierane warunki pogodowe. Na początku działania funkcji definiowana jest pusta lista, która będzie przechowywać dane meteorologiczne. Następnie tworzone jest połączenie do Open

Weather Map API za pomocą modułu *http.client*. Zadeklarowany zostaje nagłówek przechowujący klucz do API oraz informacje o hoście.

Kolejno została stworzona pętla *for*, w której są pobierane dane pogodowe z kolejnych dni. Pętla zaczyna swoje działanie z licznikiem przyjmującym wartość 5. W kolejnych iteracjach wartość licznika jest zmniejszana o 1 aż osiągnie wartość 0 po to, żeby pobrać dane z pięciu dni. Na początku działania pętli pobierana jest data – od czasu systemowego odejmowana jest ilość dni wskazywanych przez licznik np. przy pierwszym obiegu pętli uzyskiwana jest data sprzed pięciu dni, w kolejnym sprzed czterech. Następnie tworzone jest żądanie do API typu GET, w którym należy podać współrzędne geograficzne miejsca, datę oraz wcześniej zdefiniowany nagłówek. Odpowiedź z API pobierana jest za pomocą funkcji *getresponse()* i wczytywana do zmiennej o nazwie *data* za pomocą funkcji *read()*. Dane zostają dekodowane i zapisane do struktury słownikowej.

Ponieważ dane zwracane przez API są godzinne, niezbędne jest obliczenie wartości dobowych, a konkretniej minimalnej, maksymalnej i średniej temperatury dobowej, średniej wilgotności i średniej prędkości wiatru. W tym celu stworzono drugą pętlę *for*, której licznik przyjmuje wartości z zakresu od 0 do 24. Zadaniem pętli jest zsumowanie wartości temperatury, wilgotności i prędkości wiatru z całego dnia. Ponadto w każdym obiegu pętli za pomocą instrukcji warunkowej *if* porównywane są wartości minimalnej i maksymalnej temperatury tak, aby znaleźć wartość najmniejszą i największą w ciągu dnia. Gdy pętla *for* kończy swoje działanie obliczana jest średnia temperatura dobową, średnia wilgotność powietrza oraz średnia prędkość wiatru. Wartości średniej, minimalnej i maksymalnej temperatury dobowej są przeliczane na stopnie Celsjusza, ponieważ API zwraca wartości w skali Kelwina. Obliczone wartości zostają wstawione do wcześniej zdefiniowanej listy.

Po wykonaniu procesu dla pięciu dni wstecz lista zostaje wypełniona wszystkimi dwudziestoma pięcioma wartościami potrzebnymi do pracy modelu predykcyjnego. Taka lista jest zwracana przez funkcję.

```
import OWMApi
from keras.models import load_model
from joblib import load
from pandas import DataFrame
import numpy as np
from numpy import concatenate
from PIL import Image, ImageFont, ImageDraw
from constants import *

my_image = Image.open("E:/Kinga/Studies-mgr/Semestr 3/Praca
```

```

dyplomowa/System/"
                                "weather-forecast-predicting-
system/image/map3.png")
title_font = ImageFont.truetype('image/OrelegaOne-Regular.ttf', 50)
image_editable = ImageDraw.Draw(my_image)

my_image2 = Image.open("E:/Kinga/Studies-mgr/Semestr 3/Praca
dyplomowa/System/"
                                "weather-forecast-predicting-
system/image/map3a.png")
image_editable2 = ImageDraw.Draw(my_image2)

city = ['warsaw', 'KRAKÓW-OBSERWATORIUM', 'BORUCINO']
d = {'lat': [52.29958465640118, 50.0649722906938, 54.36538821401951],
     'lon': [20.927704121901673, 19.988522784913144,
18.592691014184325],
     'x': [348, 319, 170], 'y': [240, 420, 6]}
df2 = DataFrame(data=d)

for j in range(0, 3):
    actual_data = []
    actual_data = OWMApi.get_actual_weather_data(df2['lat'][j],
df2['lon'][j])
    df = DataFrame(actual_data).transpose()
    values = df.values
    values = values.astype('float32')
    scaler = load('scalers/scaler-' + city[j] + '.joblib')
    values_all = values[:, 0:5]
    values_all = scaler.transform(values_all)

    for i in range(5, 21, 5):
        values_cut = values[:, i:(i+5)]
        values_cut = scaler.transform(values_cut)
        values_all = np.concatenate([values_all, values_cut], axis=1)

    values_all = values_all.reshape((values_all.shape[0], N_DAYS,
N_FEATURES))
    # load model from single file
    model = load_model('models/lstm_model-' + city[j] + '.h5')
    # make predictions
    yhat = model.predict(values_all)
    values_all_reshape = values_all.reshape((values_all.shape[0],
N_DAYS*N_FEATURES))
    # invert scaling for forecast
    inv_yhat = concatenate((yhat, values_all_reshape[:, -(N_FEATURES-
1):]), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, 0]
    result = int(round(inv_yhat[0]))
    print(result)
    image_editable.text((df2['x'][j], df2['y'][j]), "%d °C" % result,
(0, 0, 0), font=title_font)
    # load model from single file
    model2 = load_model('models/tomorrow_model-' + city[j] + '.h5')
    # make predictions
    yhat2 = model2.predict(values_all)
    values_all = values_all.reshape((values_all.shape[0],
N_DAYS*N_FEATURES))
    # invert scaling for forecast
    inv_yhat2 = concatenate((yhat2, values_all[:, -(N_FEATURES-1):]),
axis=1)

```

```

    inv_yhat2 = scaler.inverse_transform(inv_yhat2)
    inv_yhat2 = inv_yhat2[:, 0]
    result2 = int(round(inv_yhat2[0]))
    print(result2)
    image_editable2.text((df2['x'][j], df2['y'][j]), "%d °C" %
result2, (0, 0, 0), font=title_font)

my_image.save("E:/Kinga/Studies-mgr/Semestr 3/Praca
dyplomowa/System/web-app/src/main/resources/static/today.png")
my_image2.save("E:/Kinga/Studies-mgr/Semestr 3/Praca
dyplomowa/System/web-app/src/main/resources/static/tomorrow.png")

```

Listing 5.6.2 Kod źródłowy skryptu przeznaczonego do predykcji aktualnych warunków meteorologicznych.

Aby dokonać predykcji dla aktualnych warunków meteorologicznych zaimplementowano skrypt przedstawiony na listingu 5.6.2. Wartości prognozowane zostaną naniesione na zdjęcia mapy Polski, dlatego skrypt rozpoczyna swoje działanie od otworzenia obrazków, wykorzystując do tego celu funkcję *open* z modułu *Image*. Definiowana jest czcionka, za pomocą której będą wypisywane prognozowane wartości. Aby edycja obrazków była możliwa, konieczne jest wprowadzenie ich w tryb edycji za pomocą modułu *ImageDraw* i funkcji *Draw*. Następnie definiowane są listy zawierające stałe, które będą potrzebne później – lista z nazwami miast, współrzędnymi geograficznymi oraz lista przechowująca położenie pikseli na obrazkach, które wskazują umiejscowienie napisów.

Kolejny etap działania skryptu to pętla *for*, która jest wykonywana w trzech iteracjach. W każdej z nich działa dla innego miasta. Na początku wywoływana jest funkcja *get_actual_weather_data* opisana wyżej (listing 5.6.1). Jako parametry podawane są współrzędne geograficzne danego miasta. Funkcja zwraca próbkę danych w postaci listy. Aby mogła zostać użyta przez model, należy ją odpowiednio przekształcić. Najpierw konwertowana jest do typu *DataFrame*, a następnie przeprowadzana jest transpozycja (zamiana wierszy na kolumny, a kolumny na wiersze). Następnie za pomocą funkcji *values* z danych w formacie *DataFrame* pobierane są wartości w reprezentacji *Numpy* i konwertowane do typu *float*. Kolejny krok to wczytanie z pliku skalera, który za pomocą funkcji *transform* normalizuje dane w taki sam sposób, w jaki były normalizowane dane historyczne wykorzystane do trenowania modelu predykcyjnego. Proces ten odbywa się w pięciu iteracjach, ponieważ skaler został stworzony dla danych niezmiennych na szereg czasowy, czyli dla danych z jednego dnia zawierającego pięć kolumn z poszczególnymi parametrami wejściowymi. Skaler można stosować tylko na zbiorze, który będzie posiadał odpowiadające kolumny, więc próbka składająca się z danych z pięciu dni musiała zostać podzielona

na dane z pojedynczych dni, znormalizowana, a następnie ponownie połączona. Kolejno próbka z danymi jest przekształcana do wymiaru 3D.

Tak przygotowana próbka jest gotowa do wykorzystania przez model, który zostaje wczytany z pliku, a następnie dokonuje prognozy z wykorzystaniem funkcji *predict*. Następnie dane są przekształcane z powrotem do postaci pierwotnej – dwuwymiarowej, nieznormalizowanej. Wynik predykcji jest zaokrąglany za pomocą funkcji *round* do liczby całkowitej, a wartość jest nanoszona na mapę Polski.

Kolejno zostaje załadowany model prognozujący temperaturę na dzień następny. Predykcja odbywa się na takiej samej zasadzie jak na dzień dzisiejszy, a otrzymana wartość jest nanoszona na drugi obraz Polski.

Po wykonaniu wszystkich iteracji pętli *for* (dla każdego z miast) efektem końcowym są dwie mapy Polski – pierwsza z nich przedstawia prognozowaną średnią temperaturę dobową na dziś dla Gdańska, Warszawy i Krakowa, a druga temperaturę na jutro dla tych miast. Gotowe obrazy zostają zapisane do folderu, aby możliwe było wczytanie ich przez aplikację webową.

5.7. Implementacja aplikacji webowej

W celu udostępnienia użytkownikom wyników prognoz, zaprojektowano aplikację webową. W środowisku IntelliJ utworzono projekt typu Spring Boot, korzystając również z narzędzia Maven. W pliku *pom.xml* zdefiniowano zależności niezbędne do działania aplikacji (listing 5.7.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.5</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.weather</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>weather</name>
  <description>Project</description>
  <properties>
    <java.version>11</java.version>
  </properties>
```

```

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-exec</artifactId>
    <version>1.3</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

Listing 5.7.1 Plik pom.xml.

W pliku application.properties ustawiono port uruchamiania aplikacji na 8080 oraz przedrostek i przyrostek adresu strony internetowej (listing 5.7.2).

```

server.port=8080
spring.view.prefix: /
spring.view.suffix: .html

```

Listing 5.7.2 Plik application.properties.

Na listingu 5.7.3 przedstawiono kod klasy `WeatherApplication.java`, która odpowiada za uruchomienie aplikacji. Została ona wygenerowana automatycznie podczas tworzenia projektu. Adnotacja `@SpringBootApplication` wprowadza mechanizm automatycznej konfiguracji, powoduje wykonywanie skanowania pakietu, w którym znajduje się aplikacja oraz zapewnia możliwość definiowania dodatkowych klas konfiguracyjnych. W klasie `WeatherApplication` znajduje się funkcja `main()`, która jest punktem wyjścia do wykonywania programu. Wewnątrz niej zostaje wywołana funkcja `run()`, która powoduje uruchomienie aplikacji.

```
package com.weather;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class WeatherApplication {

    public static void main(String[] args) {
        SpringApplication.run(WeatherApplication.class, args);
    }

}
```

Listing 5.7.3 Kod źródłowy klasy `WeatherApplication.java`.

W celu pobierania daty dzisiejszej i jutrzejszej utworzono klasę `BaseController` (listing 5.7.4). Użyto adnotacji `Controller`, która pozwala na automatyczne wykrywanie klas implementacyjnych poprzez skanowanie ich ścieżek. W klasie `BaseController` znajduje się metoda `getDate` oznaczona adnotacją `RequestMapping`, dzięki czemu metoda może mapować żądania webowe do metod obsługiwanych przez kontrolery Spring. W nawiasie podaje się wartość, która oznacza z jaką podstroną związana jest dana metoda oraz typ żądania. W tym przypadku jest to GET, co oznacza, że dana metoda pobiera wartość i przekazuje ją do strony internetowej. Wewnątrz metody `getDate` tworzone są obiekty klasy `Date`, które służą do pobrania aktualnej daty systemowej oraz daty jutrzejszej w porównaniu do systemowej. Powołany zostaje także obiekt klasy `SimpleDateFormat`, aby przedstawić datę w postaci dzień-miesiąc-rok. Kolejno daty zostają przekonwertowane do formatu String. Następnie tworzony jest obiekt klasy `ModelAndView` jako argument podając plik `index.html`. Do tego obiektu dodawane są pozyskane wcześniej daty za pomocą metody `addObject`, dzięki czemu będzie można je pobierać z poziomu strony internetowej.

```

package com.weather;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import java.text.SimpleDateFormat;
import java.util.Date;

@Controller
public class BaseController {

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public ModelAndView getDate(HttpServletRequest request) throws
Exception {
        SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-
yyyy");
        Date date = new Date(System.currentTimeMillis());
        Date date2 = new Date(System.currentTimeMillis() + (1000 * 60
* 60 * 24));

        String today = formatter.format(date);
        String tomorrow = formatter.format(date2);

        ModelAndView model = new ModelAndView("index.html");
        model.addObject("today", today);
        model.addObject("tomorrow", tomorrow);
        return model;
    }
}

```

Listing 5.7.4 Kod źródłowy klasy BaseController.java.

Po skonfigurowaniu wszystkich ustawień aplikacji przystąpiono do projektu strony internetowej. Utworzono plik header.html (listing 5.7.5), który odpowiada za wygląd nagłówka. Zdefiniowano odniesienia do silnika szablonów Thymeleaf oraz biblioteki Bootstrap. W ciele pliku wstawiono panel nawigacyjny za pomocą znacznika `<nav class>`, określając jednocześnie kolory tła i czcionki. Wykorzystując znacznik `<a href>`, utworzono hiperłącze w postaci tekstu „Twoja pogoda”, powodujące odświeżanie strony głównej.

```

<html xmlns:th="http://www.thymeleaf.org">

<head>
    <div th:fragment="header-css">
        <!-- this is header-css -->
        <!-- <link
href="webjars/bootstrap/4.1.3/css/bootstrap.min.css" rel="stylesheet">
-->
        <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap
.min.css"
        integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
        crossorigin="anonymous">
    </div>

```

```

</head>

<body>
<div th:fragment="header">
    <!-- this is header -->
    <nav class="navbar navbar-expand-md navbar-dark bg-info mb-4">
        <a href="/" class="navbar-brand"> Twoja pogoda </a>
    </nav>
</div>
</body>

</html>

```

Listing 5.7.5 Kod pliku header.html.

Za zawartość strony odpowiada plik index.html (listing 5.7.6). W nim również zdefiniowano odniesienia do silnika Thymeleaf oraz biblioteki Bootstrap. Dodano znacznik <title>, ustawiając tytuł zakładki w przeglądarce na „Pogoda”. Następnie wstawiono wcześniej utworzony nagłówek strony. W ciele pliku wstawiono tabelę. W jej pierwszym wierszu znajdują się dwie kolumny – jedna z datą dzisiejszą, druga z jutrzejszą, które stanowią podpis zawartości drugiego wiersza tabeli przechowującego obrazki z mapami Polski. Pobieranie tych dat odbywa się dzięki kontrolerowi *BaseController*, który został opisany wcześniej.

Umieszczanie obrazków na stronie jest możliwe dzięki znacznikowi „img”. Obrazki są zapisywane do folderu domyślnie przeznaczonego do tego celu, stąd nie jest wymagane podawanie całej ścieżki.

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
springsecurity3"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<html>
<head>
    <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/css/bootstrap.min.css"
        rel="stylesheet"
        integrity="sha384-
giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHYTfAAsrekwKmp1"
        crossorigin="anonymous">
    <meta charset="utf-8">
    <title>Pogoda</title>
    <div th:replace="fragments/header.html :: header-css"/>
</head>
<body>
<div th:replace="fragments/header.html :: header"/>
<table class="table">
    <tr style="text-align: center; vertical-align: middle; font-size:
25px; font-weight: bold">
        <td th:text="{today}">wartosc</td>
        <td th:text="{tomorrow}">wartosc</td>
    </tr>

```

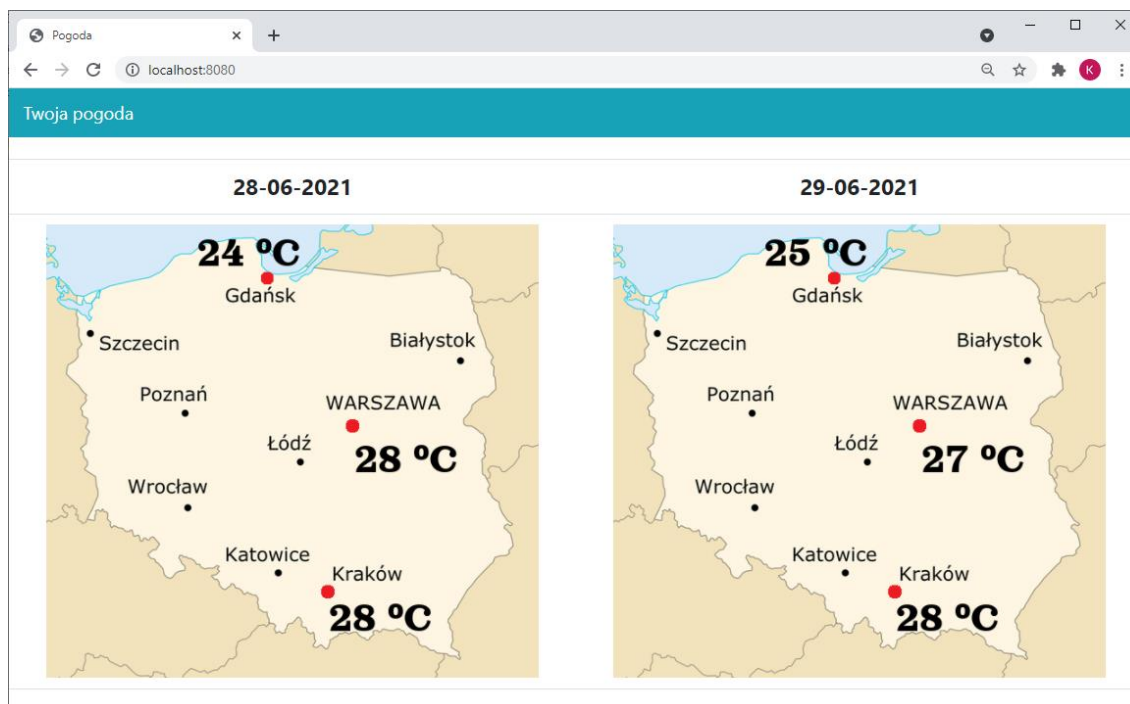
```

<tr>
  <td>
    <center></center>
  </td>
  <td>
    <center></center>
  </td>
</tr>
</table>
</body>
</html>

```

Listing 5.7.6 Kod pliku index.html.

Na rysunku 5.7.1 przedstawiono zrzut ekranu po uruchomieniu aplikacji webowej w przeglądarce internetowej. Aplikacja przedstawia dwie mapy Polski. Pierwsza z nich ilustruje prognozowaną średnią temperaturę dobową na dziś dla Gdańska, Warszawy i Krakowa, a druga predykcję na dzień następny. Nad mapami wyświetlane są daty, wskazujące na jaki dzień wykonana jest prognoza.



Rys. 5.7.1 Aplikacja webowa.

6. Podsumowanie

Efektem końcowym zrealizowanej pracy jest system prognozujący średnią temperaturę dobową na dziś i jutro dla trzech miast Polski.

Do realizacji projektu wykorzystano historyczne dane meteorologiczne udostępniane przez Instytut Meteorologii i Gospodarki Wodnej. Język programowania Python pozwolił przygotować zbiór na potrzeby analizy. Jako model predykcyjny zastosowano sztuczną sieć neuronową typu *Long Short – Term Memory Networks*. Pierwszy wstępnie zaprojektowany model miał na celu zapoznanie się z sieciami LSTM. Kolejne modele były udoskonalane tak, aby osiągnąć jak najlepszą dokładność. Ostateczny model został dostosowany do danych historycznych, na bazie których odbywa się trenowanie modelu oraz do danych aktualnych pobieranych z API, dla których wykorzystywany jest w praktyce. Model ten prognozuje średnią temperaturę dobową na dziś dla Warszawy – Bielany. Na takiej samej zasadzie zaprojektowano modele dla Krakowa oraz Gdańska. Następnie utworzono modele dla tych trzech miast prognozujące temperaturę na następny dzień. W efekcie powstało sześć modeli predykcyjnych. Ich dokładność mierzono za pomocą pierwiastka błędu średniokwadratowego (RMSE). Wartość tej metryki dla wykonanych modeli oscyluje wokół wartości 0.2 – 0.3, co świadczy o bardziej dużej dokładności.

Aktualne dane meteorologiczne pobierane są z Open Weather Map API. Za pomocą skryptu przekształcane są do takiej formy, aby móc wywołać dla nich model predykcyjny. Prognozowane wartości temperatur zostają naniesione na mapę Polski, a następnie wyświetlane są w aplikacji webowej wykonanej z użyciem technologii Spring Framework oraz HTML.

Najtrudniejszą częścią pracy było wykonanie modelu predykcyjnego. Przygotowanie danych na potrzeby modelu okazało się zadaniem czasochłonnym. Niełatwa była także implementacja sieci LSTM. Zanim osiągnięto wystarczająco dobry model, konieczne było stworzenie kilku modeli, co wymagało dużego nakładu pracy. Wykorzystanie języka Python do tej części systemu było dobrym wyborem, ponieważ dużą pomocą były gotowe moduły przeznaczone do przekształcania danych oraz biblioteka Keras implementująca elementy sieci LSTM.

Założony cel pracy został osiągnięty. Wykonany system umożliwia prognozowanie warunków meteorologicznych. Projekt pokazał, że eksploracja danych

to bardzo wartościowa dziedzina, która prowadzi do zdobycia nowej, przydatnej wiedzy. Dzięki wykorzystaniu szybkości komputera i wyspecjalizowanej sztucznej sieci neuronowej możliwe jest dokonywanie predykcji. Dla człowieka wykonanie analizy tak dużego zbioru danych jak użyty w pracy jest praktycznie niemożliwe.

Zaprojektowany system można rozbudować. Aktualnie prognozowana jest tylko jedna z pięciu wartości wejściowych. Na takiej samej zasadzie można dokonać predykcji pozostałych wartości. Np. prognozując wilgotność powietrza, można wnioskować o prawdopodobieństwie wystąpienia opadów deszczu.

Streszczenie

Celem pracy była analiza historycznych danych meteorologicznych w celu prognozowania warunków pogodowych z wykorzystaniem algorytmów eksploracji danych.

W pierwszej części pracy omówiono wiedzę teoretyczną niezbędną do wykonania pracy, a także technologie i narzędzia wykorzystane do jej realizacji. Przedstawiono takie zagadnienia jak eksploracja danych, szeregi czasowe oraz sztuczne sieci neuronowe. Opisano języki programowania Python oraz Java, język znaczników HTML, technologię Spring Framework, środowiska programistyczne PyCharm i IntelliJ oraz system kontroli wersji Git.

W części praktycznej przedstawiono cały proces powstawania systemu z naciskiem na etap tworzenia modelu predykcyjnego. Omówiono kod źródłowy projektu potrzebny do jego implementacji. Zaprezentowano wyniki osiągane przez model w postaci wykresów. Przedstawiono aplikację webową, która umożliwia wykorzystanie modelu w praktyce.

Słowa kluczowe: eksploracja danych, szereg czasowy, sztuczna sieć neuronowa, Python, aplikacja webowa

Summary

The purpose of this diploma was the analysis of historical meteorological data to predict weather conditions using data mining algorithms.

The first part describes theoretical knowledge which was necessary to design the project. Technologies and tools used to create the system are also presented. Described data mining, timeseries and artificial neural network. The programming languages Python and Java, markup language HTML, Spring Framework, development environments PyCharm and IntelliJ, version control system – Git are described.

In the practical part the whole process of creating the system is presented with emphasis on designing predictive model. Described source code which is used to implementation. The results of the model are shown in the form of charts. The web application which allows to use the model in practise is presented.

Keywords: data mining, timeseries, artificial neural network, Python, web application

Literatura

- [1] Tan, Steinbach, Kumar, Introduction to Data Mining, Pearson, 2014
- [2] Jason Brownlee, Introduction to Time Series Forecasting with Python, Machine Learning Mastery, 2020
- [3] Jason Brownlee, Long Short-Term Memory Networks With Python, Machine Learning Mastery, 2017
- [4] Jason Brownlee, Deep Learning for Time Series Forecasting, Machine Learning Mastery, 2018
- [5] Subana Shanmuganathan, Sandhya Samarasinghe, Artificial Neural Network Modelling, Springer, 2016
- [6] <https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1> 24 lipca 2018, ostatni dostęp 14 czerwca 2021
- [7] <https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d> 12 sierpnia 2018, ostatni dostęp 14 czerwca 2021
- [8] <https://medium.com/analytics-steps/understanding-the-perceptron-model-in-a-neural-network-2b3737ed70a2> 27 stycznia 2020, ostatni dostęp 14 czerwca 2021
- [9] <https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp> 28 sierpnia 2020, ostatni dostęp 12 czerwca 2021
- [10] https://www.researchgate.net/figure/Multilayer-perceptron-neural-network-model_fig1_314176848 marzec 2017, ostatni dostęp 14 czerwca 2021
- [11] <https://www.python.org/doc/essays/blurp/> ostatni dostęp 6 maja 2021
- [12] <https://pandas.pydata.org/about/> ostatni dostęp 15 czerwca 2021
- [13] <https://numpy.org/doc/stable/user/whatisnumpy.html> ostatni dostęp 19 czerwca 2021
- [14] https://scikit-learn.org/stable/getting_started.html ostatni dostęp 19 czerwca 2021
- [15] <https://matplotlib.org/> ostatni dostęp 24 lipca 2021
- [16] <https://keras.io/> ostatni dostęp 24 lipca 2021
- [17] <https://www.guru99.com/java-platform.html> ostatni dostęp 3 lipca 2021
- [18] <https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java> ostatni dostęp 3 lipca 2021
- [19] <https://data-flair.training/blogs/pros-and-cons-of-java/> ostatni dostęp 3 lipca 2021
- [20] <https://spring.io/projects/spring-framework> ostatni dostęp 23 czerwca 2021

- [21] https://www.tutorialspoint.com/spring/spring_overview.htm ostatni dostęp 24 czerwca 2021
- [22] <https://www.roseindia.net/spring/advantages-of-spring-framework.shtml> ostatni dostęp 23 czerwca 2021
- [23] <https://www.hostinger.com/tutorials/what-is-html> 15 stycznia 2021, ostatni dostęp 16 czerwca 2021
- [24] <https://mansfeld.pl/webdesign/bootstrap-co-to-jest-czy-warto-uzywac/> 13 maja 2019, ostatni dostęp 24 czerwca 2021
- [25] <https://www.baeldung.com/thymeleaf-in-spring-mvc> 28 grudnia 2020, ostatni dostęp 28 czerwca 2021
- [26] <https://www.thymeleaf.org/> 21 grudnia 2020, ostatni dostęp 28 czerwca 2021
- [27] <https://searchdatamanagement.techtarget.com/definition/SQL-Server> ostatni dostęp 24 lipca 2021
- [28] <https://learnsql.com/blog/microsoft-sql-server-pros-and-cons/> ostatni dostęp 24 lipca 2021
- [29] <https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15> ostatni dostęp 24 lipca 2021
- [30] <https://www.jetbrains.com/idea/features/> ostatni dostęp 23 czerwca 2021
- [31] <https://hackr.io/blog/what-is-pycharm> 6 sierpnia 2020, ostatni dostęp 16 czerwca 2021
- [32] <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F> ostatni dostęp 20 lipca 2021

Spis rysunków.

Rys. 3.1.1 Proces odkrywania wiedzy z baz danych (KDD).....	8
Rys. 3.1.2 Wzór na błąd średniokwadratowy RMSE.	9
Rys. 3.3.1 Model neuronu zaproponowany przez Pittsa i McCullocha.....	12
Rys. 3.3.2 Model neuronu zaproponowany przez Minsky’ego i Paperta.	13
Rys. 3.3.3 Model wielowarstwowej sztucznej sieci neuronowej.....	13
Rys. 4.6.1 Interfejs graficzny środowiska Microsoft SQL Server 18.....	21
Rys. 4.7.1 Interfejs graficzny środowiska programistycznego PyCharm.....	23
Rys. 4.8.1 Wygląd repozytorium projektu w serwisie GitHub.....	24
Rys. 4.8.2 Przeglądanie zmian kodu w serwisie GitHub.....	24
Rys. 4.8.3 Okienko konsoli Git Bash.....	25
Rys. 5.2.1 Arkusz kalkulacyjny z surowymi danymi.	27
Rys. 5.2.2 Arkusz kalkulacyjny zawierający scalone dane – pierwszy rodzaj plików...	28
Rys. 5.2.3 Arkusz kalkulacyjny zawierający scalone dane – drugi rodzaj plików.....	29
Rys. 5.2.4 Widok tabeli w programie Microsoft SQL Server Management Studio.	29
Rys. 5.2.5 Plik zawierający wszystkie scalone dane.	30
Rys. 5.2.6 Plik z wybranymi danymi ze stacji pogodowej Warszawa – Bielany.....	30
Rys. 5.3.1 Dane przed przekształceniem przez funkcję <i>series_to_supervised</i>	33
Rys. 5.3.2 Dane po przekształceniu przez funkcję <i>series_to_supervised</i>	33
Rys. 5.5.1 Wykres wartości prognozowanej średniej temperatury dobowej – Warszawa – Filtry.....	38
Rys. 5.5.2 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Warszawa – Filtry.	38
Rys. 5.5.3 Wydruk z konsoli – porównanie wartości prognozowanych z oczekiwanymi.	39
Rys. 5.5.4 Wykres wartości prognozowanej średniej temperatury dobowej – Warszawa – Filtry (wszystkie parametry).....	39
Rys. 5.5.5 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Warszawa – Filtry (wszystkie parametry).	40
Rys. 5.5.6 Wykres wartości prognozowanej średniej temperatury dobowej – Sieradz..	41
Rys. 5.5.7 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Sieradz.....	41

Rys. 5.5.8 Wykres wartości prognozowanej średniej temperatury dobowej dla zbioru z szeregiem czasowym – Warszawa – Filtry.	42
Rys. 5.5.9 Wykres wartości prognozowanej średniej temperatury dobowej dla zbioru z szeregiem czasowym – Warszawa – Filtry (wybrane próbki).	43
Rys. 5.5.10 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu dostosowanego do danych zwracanych przez API – Warszawa – Bielany.	44
Rys. 5.5.11 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu dostosowanego do danych zwracanych przez API – Warszawa – Bielany (wybrane próbki).	44
Rys. 5.5.12 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu z wprowadzonym losowym podziałem na zbiory treningowy i testowy.	45
Rys. 5.5.13 Wykres wartości prognozowanej średniej temperatury dobowej dla modelu z wprowadzonym losowym podziałem na zbiory treningowy i testowy (wybrane próbki).	46
Rys. 5.5.14 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Warszawa – Bielany.	47
Rys. 5.5.15 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Warszawa – Bielany.	48
Rys. 5.5.16 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Kraków.	48
Rys. 5.5.17 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Kraków.	49
Rys. 5.5.18 Wykres wartości prognozowanej średniej temperatury dobowej na dzisiaj – Gdańsk.	49
Rys. 5.5.19 Wykres wartości prognozowanej średniej temperatury dobowej na jutro – Gdańsk.	50
Rys. 5.7.1 Aplikacja webowa.	60

Spis kodów.

Listing 5.2.1 Kod źródłowy skryptu do scalania danych.....	27
Listing 5.2.2 Polecenie <i>select into</i>	29
Listing 5.2.3 Kod źródłowy skryptu do sortowania i selekcji danych.....	30
Listing 5.3.1 Wczytanie i konwersja danych.....	31
Listing 5.3.2 Normalizacja danych.....	31
Listing 5.3.3 Implementacja funkcji <i>series_to_supervised</i>	32
Listing 5.3.4 Wywołanie funkcji <i>series_to_supervised</i>	32
Listing 5.3.5 Usunięcie kolumn, które nie będą prognozowane.....	33
Listing 5.3.6 Podział na zbiór treningowy i testowy.....	33
Listing 5.3.7 Losowy podział na zbiór treningowy i testowy.....	33
Listing 5.3.8 Podział na wartości wejściowe i prognozowane.....	34
Listing 5.3.9 Przekształcenie zbioru do wymiaru 3D.....	34
Listing 5.4.1 Zdefiniowanie modelu LSTM.....	34
Listing 5.4.2 Kompilacja modelu LSTM.....	35
Listing 5.4.3 Trenowanie modelu LSTM.....	35
Listing 5.4.4 Zapisanie modelu LSTM do pliku.....	35
Listing 5.4.5 Wykonanie predykcji z wykorzystaniem utworzonego modelu.....	35
Listing 5.4.6 Przekształcenie wartości znormalizowanych w pierwotne.....	36
Listing 5.4.7 Wizualizacja wyników predykcji.....	36
Listing 5.6.1 Kod źródłowy funkcji <i>get_actual_weather_data</i>	51
Listing 5.6.2 Kod źródłowy skryptu przeznaczonego do predykcji aktualnych warunków meteorologicznych.....	52
Listing 5.7.1 Plik pom.xml.....	55
Listing 5.7.2 Plik application.properties.....	56
Listing 5.7.3 Kod źródłowy klasy WeatherApplication.java.....	57
Listing 5.7.4 Kod źródłowy klasy BaseController.java.....	58
Listing 5.7.5 Kod pliku header.html.....	58
Listing 5.7.6 Kod pliku index.html.....	59

Spis tabel.

Tabela 3.2.1 Zbiór danych posortowanych względem czasu.	10
Tabela 3.2.2 Zbiór danych po restrukturyzacji.	10
Tabela 3.2.3 Wielowymiarowy szereg czasowy.	11
Tabela 3.2.4 Wieloetapowy szereg czasowy.	11
Tabela 5.5.1 Tabela z wartościami RMSE dla wykonanych modeli.	47