|  |  |
| --- | --- |
|  | **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 4](#_Toc73713982)

[2. Cel i zakres pracy. 5](#_Toc73713983)

[3. Podstawy teoretyczne 6](#_Toc73713984)

[3.1. Eksploracja danych 6](#_Toc73713985)

[3.2. Szeregi czasowe 7](#_Toc73713986)

[3.3. Sztuczne sieci neuronowe 9](#_Toc73713987)

[3.4. Wykorzystane technologie 9](#_Toc73713988)

[3.4.1. Python 9](#_Toc73713989)

[3.4.2. Keras 10](#_Toc73713990)

[3.4.3. PyCharm 10](#_Toc73713991)

[3.4.4. Git 10](#_Toc73713992)

[3.4.5. Java 10](#_Toc73713993)

[3.4.6. InteliJ 10](#_Toc73713994)

[3.4.7. HTML 10](#_Toc73713995)

[3.4.8. Thymeleaf 10](#_Toc73713996)

[4. Praktyczna realizacja systemu 10](#_Toc73713997)

[4.1. Wykorzystany zbiór danych 10](#_Toc73713998)

[4.2. Wstępne przygotowanie danych 11](#_Toc73713999)

[4.3. Przetwarzanie danych na potrzeby modelu LSTMs 14](#_Toc73714000)

[4.4. Implementacja modelu LSTM 17](#_Toc73714001)

[4.5. Proces tworzenia modelu LSTM 20](#_Toc73714002)

[4.6. Zastosowanie modeli predykcyjnych dla aktualnych danych 34](#_Toc73714003)

[4.7. Implementacja aplikacji webowej 36](#_Toc73714004)

[5. Podsumowanie 36](#_Toc73714005)

[Literatura 37](#_Toc73714006)

# Wstęp

Informatyka to obecnie jedna z najszybciej rozwijających się dziedzin nauki. Dostęp do ogromnych ilości danych wymusza poszukiwanie sposobów na poprawne i efektywne ich przechowywanie oraz przetwarzanie, a właśnie tym zajmuje się informatyka. Powstają coraz to nowsze i lepsze technologie, które sprostają tym zadaniom. Ilość generowanych w dzisiejszym świecie informacji doprowadziła do wyodrębnienia takich działów informatyki jak hurtownie danych czy Big Data.

Przechowywanie, przetwarzanie i analiza dużych, zmiennych, różnorodnych zbiorów danych nie jest prostym zadaniem, jednak bardzo wartościowym. Może prowadzić do zdobycia nowych, cennych informacji. Z pomocą w tym zadaniu przychodzi eksploracja danych, która wykorzystuje szybkość komputera do odszukiwania prawidłowości w gromadzonych danych. Dzięki temu możliwe jest wyciąganie wniosków oraz prognozowanie, co znajduje szerokie zastosowanie w wielu dziedzinach np. biznesie czy przemyśle.

W niniejszej pracy eksploracja danych zostanie wykorzystana w meteorologii. Nie jest to łatwa nauka, gdyż warunki pogodowe są bardzo zmienne i zależne od wielu czynników, a prognozowanie pogody jest procesem złożonym i trudnym.

Szybki rozwój technologii umożliwił gromadzenie ogromnych ilości danych. Jednak wydobycie z nich przydanych informacji jest dużo trudniejszym wyzwaniem. Tradycyjne metody analizy danych przestały być wystarczające. To doprowadziło do powstania nowych technologii, które za pomocą specjalistycznych algorytmów przetwarzają duże ilości danych.

# Cel i zakres pracy.

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

Proces realizacji pracy podzielono na kilka etapów:

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

System został stworzony przy użyciu nowoczesnych technologii. Większość pracy wykonano za pomocą języka programowania Python, który jest najczęściej wykorzystywany do zagadnień związanych ze sztuczną inteligencją, eksploracją i analizą danych. Pozwolił on na dokonanie sprawnej obróbki danych oraz zaprojektowanie modelu predykcyjnego. Aby zwizualizować wyniki prognoz zaimplementowano aplikację webową, korzystając z języka programowania Java oraz HTML. Jako środowiska programistyczne wybrano PyCharm oraz InteliJ. Ponadto zastosowano system kontroli wersji Git.

W pracy wyróżniono część teoretyczną i praktyczną. Pierwszą z nich stanowią rozdziały omawiające podstawy teoretyczną niezbędne do wykonania niniejszej pracy, a także technologie i narzędzia wykorzystane do zaprojektowania systemu. Druga część pracy szczegółowo opisuje implementację projektu, wszystkie kroki jego powstawania i ulepszania oraz sposób działania.

# Podstawy teoretyczne

## 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 wyników dla przyszłych 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 przedstawiono etapy KDD, które prowadzą do otrzymania wartościowych informacji z danych wejściowych. Pierwszy etap - wstępne przetwarzanie danych (ang. preprocessing) obejmuje zebranie danych z różnych źródeł, usunięcie duplikatów i niepełnych danych, wybór cech, redukcję wymiarowości, normalizację, wyodrębienie podzbiorów. Często jest to proces żmudny i bardzo czasochłonny. Celem wstępnego przetworzenia danych jest przekształcenie surowych danych w dane odpowiednio sformatowane pod kątem analizy. Drugi etap to eksploracja danych, w ramach której do konretnych danych dobiera się i stosuje odpowiednie algorytmy. Na ostatni krok – końcowe przetwarzanie (ang. postprocessing) składają się filtrowanie wzorców, wizualizacja oraz interpretacja wyników. Stosowane są także statystyczne miary oceniające wydajność wykonanych modeli. Dzięki temu etapowi cały proces znajduje zastosowanie w praktyce – może zostać wykorzystany w aplikacjach biznesowych, ponieważ zapewnia, że tylko poprawne i przydatne informacje zostaną zastosowane w systemie.



Rys. 3.1.1 Proces odkrywania wiedzy z baz danych (KDD) [1].

Eksploracja danych jest wykorzystywana do dwóch typów zadań. Pierwszy typ do 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 w danych, określanie korelacji między danymi, trendów i zauważanie anomalii.

Zgłębianie danych musi sprostać wielu wymaganiom.

## 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 problemów szeregów czasowych wprowadzono standardowe terminy używane do opisywania tego typu danych:

* 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 zastosować szeregi czasowe w uczeniu maszynowym, należy odpowiednio przygotować posiadany zbiór danych. Na rysunku przedstawiono przykładowy zbiór danych, w którym dane są posortowane względem czasu.

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

Restrukturyzacja tego zbioru polega na tym, że wartość poprzedniego kroku jest używana do prognozowania wartości zmiennej w kroku następnym. Na rysunku widać, że poprzedni krok czasowy jest wejściem (X), a następujący po nim krok czasowy jest wyjściem (y). Kolejność obserwacji jest zachowana i musi taka pozostać w trakcie trenowania modelu predykcyjnego. Dla pierwszego wiersza brakuje wartości wejściowej, a dla ostateniego wartości wyjściowej, dlatego należy je usunąć.

|  |  |
| --- | --- |
| X | y |
| NaN | 100 |
| 100 | 50 |
| 50 | 120 |
| 120 | 90 |
| 90 | NaN |

Wykorzystanie wcześniejszych kroków czasowych do przewidywania następnego nazywane jest metodą przesuwnego okna (ang. sliding window method).

Ze względu na ilość obserwacji wyróżniamy dwa rodzaje szeregów czasowych:

* jednowymiarowe (ang. Univariate Time Series) – obserwowana jest tylko jedna zmienna w każdym kroku czasowym,
* wielowymiarowe (ang. Multivariate Time Series) – obserwowane są dwie lub więcej zmiennych w każdym kroku czasowym.

Prognozowanie w przypadku wielowymiarowych zbiorów jest dużo trudniejsze i bardziej złożone niż jednowymiarowych. Na rysunku przedstawiono przykład wielowymiarowego szeregu czasowego.

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X3 | y |
| NaN | NaN | 100 | 200 |
| 100 | 200 | 50 | 150 |
| 50 | 150 | 120 | 60 |
| 120 | 60 | 90 | 30 |
| 90 | 30 | NaN | NaN |

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

* jednoetapowy (ang. One-step Forecast) – przewidywany jest jeden następny krok (przykład z rysunków ),
* wieloetapowe (ang. Multi-step Forecast) – prognozowane są dwa lub więcej kroki czasowe (rys. ).

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | y1 | y |
| NaN | NaN | 100 | 200 |
| 100 | 200 | 50 | 150 |
| 50 | 150 | 120 | 60 |
| 120 | 60 | 90 | 30 |
| 90 | 30 | NaN | NaN |

## Sztuczne sieci neuronowe

## Wykorzystane technologie

### 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. Jest dostosowany do programowania zorientowanego obiektowo, umożliwia definiowanie klas wraz z dziedziczeniem. Wyróżnia się dynamicznym deklarowaniem typów zmiennych. Twórcy Pythona położyli nacisk na czytelność kodu, co zmniejsza koszty jego utrzymania.

Python znajduje szerokie zastosowanie w takich dziedzinach informatyki jak aplikacje webowe, nauka o danych, sztuczna inteligencja.

### Keras

### PyCharm

### Git

### Java

### InteliJ

### HTML

### Thymeleaf

# Praktyczna realizacja systemu

## Wykorzystany zbiór danych

Pierwszym etapem praktycznej realizacji systemu był wybór zbioru danych do analizy. To ważna część pracy, ponieważ im dokładniejsze dane, tym model prognostyczny daje lepsze efekty. Na potrzeby pracy wybrano historyczne dane meteorologiczne udostępniane przez Instytut Meteolorogii i Gospodardki Wodnej. Są one zbierane z polskich stacji klimatologicznych, przechowywane w formacie *csv*. Dostępne dane obejmują okres czasu od 1951 r. Na potrzeby pracy pobrano dane od stycznia 2010 r. do stycznia 2021 r., ponieważ kliemat jest zmienny, w związku z czym najbardziej wartościowe są dane z ostatnich lat. Parametry meteorologiczne zawarte w zbiorze to:

* maksymalna temperatura dobowa [°C],
* minimalna temperatura dobowa [°C],
* średnia temperatura dobowa [°C],
* temperatura minimalna przy gruncie [°C],
* średnia dobowa wilgotność względna [%],
* średnia dobowa prędkość wiatru [m/s],
* suma dobowa opadów [mm],
* rodzaj opadu [S/W/],
* średnie dobowe zachmurzenie ogólne [oktanty],
* wysokość pokrywy śnieżnej [cm].

## Wstępne przygotowanie danych

Pobrane pliki z danymi były podzielone na dane miesięczne. Fragment jednego z plików został przedstawiony na rysunku 4.2.1.



Rys. 4.2.1 Arkusz kalkulacyjny z surowymi danymi.

Aby móc przetwarzać dane, należało scalić wszystkie pliki w jeden zbiór. W tym celu napisano skrypt (listing 4.2.1). 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 pętla for otwiera każdy z plików i nadaje nagłówki kolumnom, a otrzymany plik zapisuje w formacie csv. Kolejno ponownie za pomocą modułu glob wczytywane są pliki – tym razem pliki z nagłówkami. Za pomocą funkcji concat z modułu pandas pliki zostają zscalone. W efekcie działania skryptu powstały dwa różne pliki w formacie csv, różniące się przechowywanymi parametrami meteorologicznymi (rys. 4.2.2, rys. 4.2.3).

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 predkosc 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 4.2.1 Kod źródłowy skryptu do scalania danych.



Rys. 4.2.2 Arkusz kalkulacyjny zawierający zscalone dane – pierwszy rodzaj plików.



Rys. 4.2.3 Arkusz kalkulacyjny zawierający zscalone dane – drugi rodzaj plików.

Do połączenia powyższych dwóch plików skorzystano z programu Miscrosoft SQL Server. Każdy plik został importowany do osobnej tabeli, a następnie za pomocą polecenia select into z klauzulą join (listing 4.2.2) stworzono jedną tabelę, którą wyeksportowano do pliku w formacie csv. Fragment otrzymanego pliku, zawierającego wszystkie dane, przedstawiono na rysunku 4.2.4.

select d.\*, t.[Srednia dobowa wilgotnosc wzgledna % ]

,t.[Srednia dobowa 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 4.2.2 Polecenie *select into*.

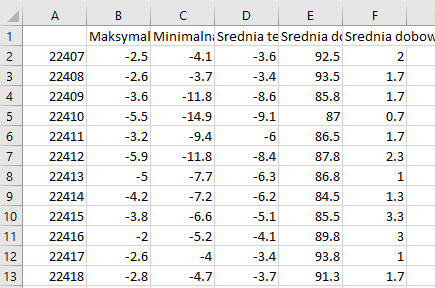


Rys. 4.2.4 Plik zawierający wszystkie zscalone dane.

Kolejno zaimplementowano skrypt służący do sortowania oraz selekcji danych (listing 4.2.3). Został on użyty wielokrotnie podczas procesu tworzenia modelu predykcyjnego. Wczytuje on dane z pliku *csv*, wykorzystując bibliotekę *pandas* i funkcję *read\_csv*. Sortuje dane według kodu stacji, roku, miesiąca i dnia z wykorzystaniem funkcji *sort\_values*, natomiast selekcja jest dostosowywana do potrzeb danego modelu. Wybierane są dane 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. 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 4.2.3 Kod źródłowy skryptu do sortowania i selekcji danych.



Rys. 4.2.1

## Przetwarzanie danych na potrzeby modelu LSTM

Dane, które będą wykorzystywane przez model LSTM są zapisane w pliku o formacie csv, dlatego aby je pozyskać została wykorzystana funkcja read\_csv (listing ). Następnie dla wczytanego zbioru danych wywoływane 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

Kolejnym krokiem przetwarzania danych jest ich normalizacja (listing ), czyli takie przekształcenie zbioru, aby wszystkie wartości były w zakresie od 0 do 1. W tym celu zastosowano MinMaxScaler z biblioteki sklearn. Skaler zostaje zapisany do pliku tak, aby można go później zastosować dla danych aktualnych.

*# normalize features*scaler = MinMaxScaler(feature\_range=(0, 1))  
scaled = scaler.fit\_transform(values)  
dump(scaler, **'scalers/scaler-'** + CITY + **'.joblib'**)

Listing

Na potrzeby modelu predykcyjnego zaimplementowano funkcję series\_to\_supervised (listing ), 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 shitf() tworzy kopię kolumny z wartościami przesuniętymi o 2 w tył (dodając na początku wiersze z wartościami NaN). 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). 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 4.3.1 Implementacja funkcji *series\_to\_supervised*.

Na rysunku 4.3.1 przedstwiono zbiór danych przed wywołaniem funkcji (listing 4.3.2), a na rysunku 4.3.2 zbiór powstały w wyniku działania tej funkcji wywołanej dla tego znormalizowanego zbioru z krokiem opóźnienia 2 oraz krokiem prognozowania 2.

reframed = series\_to\_supervised(scaled, 2, 2)

Listing 4.3.2 Wywołanie funkcji *series\_to\_supervised*.



Rys. 4.3.1 Dane przed przekstałceniem przez funkcję series\_to\_supervised.



Rys. 4.3.2 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 wartością prognozowaną jest jedynie średnia temperatura dobowa, dlatego należy usunąć kolumny, które nie będą prognozowane (listing )

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

Listing 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 ). Jednak później zastosowano losowy podział (listing ), dzięki czemu modele osiągały lepsze wyniki. Kolejno każdy z tych zbiorów zostaje dzielony na wartości wejściowe (wszystkie kolumny oprócz ostatniej) i wartość prognozowaną (ostatnia kolumna). Kod, który posłużył do tego celu przedstawiono na listingu.

train = values[:TRAIN\_SIZE, :]  
test = values[TRAIN\_SIZE:, :]

Listing 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 Podział na zbiór treningowy i testowy.

*# split into input and outputs*train\_X, train\_y = train[:, :-1], train[:, -1]  
test\_X, test\_y = test[:, :-1], test[:, -1]

Listing Podział na wartości wejściowe i prognozowane.

Powstałe zbiory treningowy i testowy są przekształcane do wymiaru 3D – pierwszy wymiar to próbki, drugi kroki czasowe, a trzeci cechy.

*# reshape input to be 3D [samples, timesteps, features]*train\_X = train\_X.reshape((train\_X.shape[0], 1, train\_X.shape[1]))  
test\_X = test\_X.reshape((test\_X.shape[0], 1, test\_X.shape[1]))

Listing Przekształcenie zbioru do wymiaru 3D.

## Implementacja modelu LSTM

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

Pierwszym krokiem jest zdefiniowanie modelu (listing ). Sieć neuronowa w bibliotece Keras to sekwencja warstw. Kontenerem tych warstw jest klasa Sequential, dlatego należy powołać instację tej klasy. Następnie można utworzyć połączone ze sobą warstwy. Warstwy ukryte (do przetwarzania danych) LSTM 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 Zdefiniowanie modelu LSTM.

Po zdefiniowaniu modelu należy go skompilować (listing ). Ten proces ma na celu przeszktałcenie 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 Kompilacja modelu LSTM.

Gdy model LSTM zostanie skompilowany, może zostać wytrenowany (listing ) na zbiorze treningowym. W tym celu należy podać macierz danych wejściowym 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 Trenowanie modelu LSTM.

Po wytrenowaniu modelu zostaje on zapisany do pliku (listing ), 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 Zapisanie modelu LSTM do pliku.

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

Aby wynik prognozy był czytelny konieczny jest powrót z wartości znormalizowanych do wartości pierwotnych (listing ). Zbiór trójwymiarowy należy przekształcić na dwuwymiarowy, a dla wcześniej użytego skalera wywołać funkcję *inverse\_transform* zarówno dla wartości otrzymanych przez model (zmienna *inv\_yhat*) jak i wartości ze zbioru testowego (zmienna *inv\_y*).

*# make a prediction*yhat = model.predict(test\_X)

Listing Predykcja.

test\_X = test\_X.reshape((test\_X.shape[0], test\_X.shape[2]))  
*# invert scaling for forecast*inv\_yhat = concatenate((yhat, test\_X[:, 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[:, 1:]), axis=1)  
inv\_y = scaler.inverse\_transform(inv\_y)  
inv\_y = inv\_y[:, 0]

Listing

W celu wizualizacji wyników (listing ) napisano pętlę *for*, która powoduje wyświetlenie wartości prognozowanej i oczekiwanej dla pierwszych stu próbek. Stworzono także wykresy przedstawiające porównanie wartości prognozowanych i oczekiwanych. Pierwszy wykres obejmuje wszystkie próbki, drugi tylko część próbek dla lepszej czytelności. Obliczono także pierwiastek błędu średniokwadratowego (RMSE), 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 dobowa °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 dobowa °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 Wizualizacja wyników predykcji.

## Proces tworzenia modelu LSTM

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

Pierwszy wstępny model miał na celu przede wszystkim zapoznanie się z Long – Short Term Memory Networks i naukę jego poprawnej implementacji. 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 dobowa temperatura 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 przedstawiono wykresy wartości prognozowanych w stosunku do oczekiwanych. Widać na nich, że wartości te pokrywają się tylko częściowo. Na rysunku 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. 4.3.4 Wykres wartości prognozowanej średniej temperatury dobowej – Warszawa – Filtry.



Rys. 4.5.1 Wykres wartości prognozowanej średniej temperatury dobowej dla wybranych próbek – Warszawa – Filtry.



W modelu drugim również zastosowano dane ze stacji klimatologicznej Warszawa – Filtry z jednego dnia, ale zebrano wszystkie dostępne parametry meteorologiczne: Maksymalna temperatura dobowa [°C], Minimalna temperatura dobowa [°C], Średnia temperatura dobowa [°C], Temperatura minimalna przy gruncie [°C], Suma dobowa opadow [mm], Wysokość pokrywy śnieżnej [cm], Średnia dobowa wilgotność względna [%], Średnia dobowa predkość 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 również można odczytać, że pokrycie jest trochę gorsze niż w przypadku pierwszego modelu.



Wykres wartości prognozowanej średniej temperatury dobowej – Warszawa – Filtry.



Model trzeci wykonano dla danych z innej stacji pogodowej niż wcześniej. 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 przedstawiono wykresy…



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 kolumnami. Dokładność tego modelu znacząco wzrosła w stosunku do poprzednich, co pokazuje jak ważne jest rozpatrywanie szeregów czasowych w zagadnień 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 (rys) także widać, że wartości prognozowane dużo lepiej pokrywają się z wartościami oczekiwanymi.



Piąty model predykcyjny został zaprojektowany tak, aby był dostosowany do danych aktualnych pobieranych z API. Szereg czasowy rozszerzono do pięciu dni, ponieważ z tylu 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 dobowa [°C], Minimalna temperatura dobowa [°C], Średnia temperatura dobowa [°C], Średnia dobowa wilgotność względna [%], Średnia dobowa prędkość wiatru [m/s]. Zmieniono także stację pogodową, ponieważ zauważono, że dla Warszawy – Filtry w kolumnie „Średnia dobowa wilgotność względna [%]” 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 przedstawiono wykresy, na których także można zaobserwować dużą dokładność modelu.



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 najaktualniejsze, które zostało wykorzystane do trenowania modelu. Powyższa teza okazała się trafna, ponieważ model osiągnął lepszy wynik po wprowadzeniu takiego podziału. Wartość RMSE osiągnęła 0.203. Na wykresach (rys) 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.



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 ulepszyło jego działanie. Wyniki osiągane przez model są bardzo 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 de facto 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. Z tego względu dla każdej ze stacji zaprojektowano modele, które będą prognozować temperaturę na jutro. Aby to osiągnąć, wystarczyło wprowadzić w implementacji modelu tylko jedną zmianę. Zamiast tworzyć kopię kolumny przesunięto o 1, należało utworzyć kopię kolumny przesuniętą o 2.

W tabeli (rys ) przedstawiono wartości pierwiastka błędu średniokwadratowego dla wszystkich sześciu modeli. Najlepszy wynik osiągnął model prognozujący temperaturę na jutro dla stacji Borucino – 0.164, najgorszy na jutro dla Warszawy – Bielany – 0.399, natomiast 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 |

Na rysunkach zaprezentowano wykresy dla wszystkich zaprojektowanych 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.



Warszawa dziś



Warszawa – jutro



Kraków dziś



Kraków jutro



Gdańsk dziś



Gdańsk jutro

## Zastosowanie modeli predykcyjnych dla aktualnych danych

Aktualne dane meteorologiczne są pobierane z wykorzystaniem Open Weather Map API. Interfejs ten jest płatny, jeśli chce się z niego korzystać bez ograniczeń, natomiast zachowując limity zapytań możliwe jest korzystanie z niego całkowicie za darmo. Wystarczy założyć konto na stronie internetowej: <https://rapidapi.com/marketplace> oraz wygenerować klucz dostępu do wybranego API. Na potrzeby pracy dostępna darmowa pula zapytań jest wystarczająca, dlatego nie było konieczne uiszczenie opłat.

Open Weather Map API pozwala 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 dobowa, dane pobierane z API są godzinne.

Na listingu przedstawiono kod źródłowy funkcji, którą zaimplementowano w celu pobierania danych z API oraz przekształcania ich do takiej formy, aby można było wykonać dla nich predykcję. Funkcja 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 iteratorem 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()*, a wczytywana do zmiennej data funkcją *read()*. Dane zostają dekodowane i zapisywane do zmiennej typu słownik. Ponieważ dane 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 iterator 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. Gdy pętla *for* kończy swoje działanie obliczana jest średnia temepratura dobowa, średnia wilgotność oraz średnia prędkość wiatru. Wartości średniej, minimalnej i maksymalnej temperatury dobowej są zamieniane na stopnie Celsjusza, ponieważ API zwraca wartości w skali Kelwina. Obliczone wartości zostają wstawione do wcześniej zdefiniowanej listy. Po wykonaniu procesu dla pięciu dni do tyłu lista zostaje wypełniona wszystkimi dwudziestoma pięcioma wartościami potrzebnymi do pracy modelu predykcyjnego. Taka lista jest zwracana przez funkcję.

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'**: **"f7cbc33b86msh15d74892fcf8b61p14f6c8jsn5ac8c369d4d2"**,  
 **'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

Aby dokonać predykcji dla aktualnych warunków meteorologicznych zaimplementowano skrypt przedstawiony na listingu. Wartości prognozowane zostaną naniesione na zdjęcia mapki Polski, dlatego skrypt rozpoczyna swoje działanie od otwarcia 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 miejsce początku położenia napisu. 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). Jako parametry podawane są współrzędne geograficzne miasta. Funkcja zwraca próbkę danych w postaci listy. Aby mogła zostać użyta przez model, należy ją odpowiedno przekształcić. Najpierw konwertowana jest

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/result.png"**)  
my\_image2.save(**"E:/Kinga/Studies-mgr/Semestr 3/Praca dyplomowa/System/web-app/src/main/resources/static/result2.png"**)

## Implementacja aplikacji webowej

# Podsumowanie

# 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] <https://www.edureka.co/blog/interview-questions/python-interview-questions/> 21 stycznia 2021

[5] <https://www.python.org/doc/essays/blurb/> ostatni dostęp 6 maja 2021

[6] <https://www.badania-statystyczne.pl/data-mining> ostatni dostęp 12 maja 2021

[7]<https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/> 28 sierpnia 2020, ostatni dostęp 02.06.2021