MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL AVIATION UNIVERSITY

Faculty of Aeronavigation, Electronics and Telecommunications

Department of aviation computer integrated complexes

ADMIT TO DEFENSE

Head of the graduating department

_________ Viktor SINEGLAZOV

“_____” ________________2024y.

QUALIFICATION WORK

(EXPLANATORY NOTE)

OF THE GRADUATE OF THE EDUCATIONAL DEGREE

“BACHELOR”

Specialty 151 "Automation and computer-integrated technologies"

Educational and professional program "Computer-integrated technological processes and production"

Theme: Intelligent data processing system

Performer: student of FAET-421 ba group Petrov Denys

Supervisor: Mykola TUPITSYN

Norm controller: ___________ Filyashkin M.K.

(sign)

Kyiv – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра авіаційних комп'ютерно-інтегрованих комплексів

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

______ Віктор СИНЄГЛАЗОВ

“____” ______________2024 р.

КВАЛІФІКАЦІЙНА РОБОТА (ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

“БАКАЛАВР”

Спеціалність 151 "Автоматизація, та комп’ютерно-інтегровані технології"

Освітньо-професійна програма "Комп’ютерно-інтегровані технологічні процеси і виробництва"

Тема: Інтелектуальна система обробки даних

Виконавець: студент групи ІК-421Ба Петров Денис Миколайович

Керівник: к.т.н., доцент Тупіцин Микола Федорович

Нормоконтроллер: ___________ Філяшкін М.К.

(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет аеронавігації, електроніки та телекомунікацій
Кафедра авіаційних комп’ютерно-інтегрованих комплексів

Освітньо-кваліфікаційний рівень бакалавр
Спеціальність151 «Автоматизація та комп’ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____________ Віктор СИНЄГЛАЗОВ
«_____»________________2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
ПЕТРОВА Дениса Миколайовича

1. Тема кваліфікаційної роботи «Інтелектуальна система обробки даних».

2.**Термін виконання роботи:** з 15 квітня 2024р. по 14 червня 2024 р.

3. **Вихідні дані до роботи**: Chandola V., Kumar V. Summarization – compressing data into an informative representation // Knowledge and Information Systems. 2007. Vol. 12, is. 3. P. 355-378.

4.**Зміст пояснювальної записки**: 1) Огляд та аналіз методів інтелектуального аналізу даних; 2) Опис особливостей регресійного аналізу; 3) Постановка завдання; 4) Математична модель регресійного аналізу; 5) Розробка алгоритму регресійного аналізу погодних даних; 6) Приклад обробки даних за домомогою регресійного аналізу; 7) Аналіз отриманих результатів.

5. **Перелік обов’язкового графічного (ілюстративного) матеріалу:** Презентація в MicrosoftPowerPoint.

6**. Календарний план-графік**

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомлення з постановкою задачі кваліфікаційної роботи.	01.04.2024-04.04.2024	Виконано
2	Аналіз літературних джерел та інтернет ресурсів.	05.04.2024-24.04.2024	Виконано
3	Огляд та аналіз методів інтелектуального аналізу даних .	25.04.2024-1.05.2024	Виконано
4	Опис особливостей регресійного аналізу.	2.05.2024-10.05.2024	Виконано
5	Постановка завдання;	11.05.2024-25.05.2024	Виконано
6	Математична модель регресійного аналізу.	26.05.2024-02.06.2024	Виконано
7	Приклад обробки даних за домомогою регресійного аналізу.	03.06.2024-09.06.2024	Виконано
8	Оформлення пояснювальної записки, графічних матеріалів та презентації до дипломного проекту.	10.06.2024-13.06.2024	Виконано
9	Подання кваліфікаційної роботи до захисту	14.06.2024	Виконано

7. Дата видачі завдання: “ 15 "_квітня 2024 р.

Керівник дипломної роботи [illegible] Тупіцин М.Ф.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання ________________ Петров Д.М.

(П.І.Б.)

NATIONAL AVIATION UNIVERSITY

Faculty of Aeronautics, Electronics and Telecommunications

Department of aviation computer-integrated complexes

Bachelor's degree in education

Specialty151 “Automation and computer-integrated technologies”

APPROVED BY

Head of the department

_____________ Viktor SINEGLAZOV

«_____»_________________2024 р.

TASKS

for the qualification work

PETROV Denys

1. The topic of the qualification work is “Intelligent data processing system”**.**

2.Term of work: from April 15, 2024 to June 14, 2024.

3. Initial data for the work: Chandola V., Kumar V. Summarization - compressing data into an informative representation // Knowledge and Information Systems. 2007. Vol. 12, is. 3. P. 355-378.

4.Contents of the explanatory note: 1) Overview and analysis of data mining methods; 2) Description of the features of regression analysis; 3) Statement of the problem; 4) Mathematical model of regression analysis; 5) Development of the algorithm for regression analysis of weather data; 6) Example of data processing using regression analysis; 7) Analysis of the results.

5. List of required graphic (illustrative) material: Presentation in Microsoft Power Point.

6. Calendar plan-schedule

№ cf.	Tasks.	The term fulfillment	A note on the execution
1	Familiarization with the task statement of the qualification work.	01.04.2024-04.04.2024	Done
2	Analysis of literature and Internet resources.	05.04.2024-24.04.2024	Done
3	Overview and analysis of data mining methods .	25.04.2024-1.05.2024	Done
4	Description of the features of regression analysis.	2.05.2024-10.05.2024	Done
5	Task statement;	11.05.2024-25.05.2024	Done
6	Mathematical model of regression analysis.	26.05.2024-02.06.2024	Done
7	An example of data processing using regression analysis.	03.06.2024-09.06.2024	Done
8	Preparation of an explanatory note, graphic materials, and a presentation for the thesis project.	10.06.2024-13.06.2024	Done
9	Submission of qualification work for defense	14.06.2024	Done

7. Date of the task issue: “ 15 " April 2024 p.

Thesis supervisor [illegible] Tupitsun M.F.

(signature of the head)

The task was accepted for execution __________ Petrov D.M.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Інтелектуальна система обробки даних». Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел і має 75 сторінки, 36 малюнки, 13 формул, 24 літературних джерел.

Мета дипломного проекту: Дослідити та продемонструвати ефективні методи аналізу та прогнозування часових рядів за допомогою Python, зосередившись на погодних даних з 2013 по 2017 рік. Кваліфікаційна робота включатиме методи попередньої обробки даних, такі як нормалізація та зменшення шуму, а також буде використовувати перетворення Фур'є та градієнтний спуск для підгонки кривих. Метою є виявлення закономірностей і прогнозування майбутніх точок даних, використовуючи моделі машинного навчання, побудовані за допомогою TensorFlow і оцінені за допомогою візуалізації та метрик помилок.

Постановка задачі: Прогнозування погоди має вирішальне значення для різних галузей, проте передбачення погодних умов залишається складним завданням через складність та мінливість кліматичних даних. Це завдання показує як можна вирішити цю проблему, використовуючи сучасні методи аналізу часових рядів та машинного навчання. Основна увага приділяється обробці погодних даних з 2013 по 2017 рік для виявлення закономірностей і розробки моделей прогнозування. Проблема полягає в ефективній обробці зашумлених даних, застосуванні відповідних методів нормалізації та трансформації, а також у виборі надійних алгоритмів для підвищення точності та надійності прогнозування.

SUMMARY

Explanatory note to the qualification work “Intelligent data processing system”. The qualification work consists of an introduction, three chapters, general conclusions, a list of references and has 75 pages, 36 figures, 13 formulas, 24 literary sources.

The purpose of the diploma project: To investigate and demonstrate effective methods for time series analysis and forecasting using Python, focusing on weather data from 2013 to 2017. The qualification work will include data preprocessing techniques such as normalization and noise reduction, and will use Fourier transform and gradient descent for curve fitting. The goal is to detect patterns and predict future data points using machine learning models built with TensorFlow and evaluated using visualization and error metrics.

Problem statement: Weather forecasting is crucial for various industries, yet predicting weather conditions remains a challenging task due to the complexity and variability of climate data. This challenge shows how this problem can be solved using modern time series analysis and machine learning techniques. The focus is on processing weather data from 2013 to 2017 to identify patterns and develop forecasting models. The challenge is to efficiently process noisy data, apply appropriate normalization and transformation methods, and choose reliable algorithms to improve forecasting accuracy and reliability.

CONTENT

LIST OF ABBREVIATIONS

IDPS	Intelligent data processing systems
IDA	Intelligent data analysis
MLP	Multilayer Perceptron
RNN	Recurrent Neural Networks
CNN	Convolutional Neural Networks
GAN	Generative Adversarial Networks
AR	Autoregressive models
MA	Moving Average models
ARIMA	Autoregressive Integrated Moving Average
SARIMA	Seasonal Autoregressive Integrated Moving Average
AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
PCA	Principal Component Analysis
k-NN	K-Nearest Neighbors method
MSE	Mean Squared Error
MAE	Mean Absolute Error
API	Application Programming Interface
SGD	Stochastic Gradient Descent
L1/L2	L1 (Lasso), L2(Ridge) regularization
R-squared	Coefficient of Determination

INTRODUCTION

The modern world stands on the threshold of a new era, where information is becoming the most valuable resource. Every day, the volume of data generated in various fields of human activity is growing exponentially. In this context, the need for new approaches to the collection, storage, processing, and analysis of this data becomes evident. Intelligent data processing systems (IDPS) represent a set of technologies that use artificial intelligence methods and analytical algorithms for the effective management of large volumes of information.

Intelligent data processing systems not only automate data processing processes but also allow for the acquisition of new knowledge, the identification of hidden patterns, and the making of informed decisions based on data. This opens up new opportunities to improve efficiency, innovation, and development in various fields: from business and medicine to finance and energy. IDPS are particularly significant in the field of aviation, where the accuracy and speed of information processing are critical for flight safety, effective air traffic management, and the development of aviation technologies.

The relevance of using IDPS in the field of aviation is noted by a number of advantages and characteristics that contribute to improving safety, efficiency, and meeting passenger needs. These systems help identify potential threats and prevent accidents, optimize routes and fuel consumption to save costs, and maintain an optimal level of aircraft technical readiness.

The aviation industry constantly generates huge amounts of data, including information on the technical condition of aircraft, weather conditions, flight routes, and other key indicators. The use of IDPS in aviation allows for predictive analytics, improves the efficiency of aircraft maintenance, optimizes routes, and ensures a high level of flight safety. Intelligent data processing systems can also contribute to the development of new aviation technologies, such as unmanned aerial vehicles and automatic air traffic management systems.

The work is an example of the considered systems underlying IDPS, as well as their practical application in the aviation industry. Particular attention will be paid to the analysis of modern big data processing technologies, machine learning and artificial intelligence methods, as well as the integration of IDPS into aviation computer systems.

The purpose of this work is to identify patterns and predict future data points using machine learning models, analyze their effectiveness and influence on the development of modern technologies in aviation. Development of a model for forecasting temperature based on historical data and application of IDPS in the aviation industry, identification of the main challenges and prospects for the development of this industry.

Thus, this work aims to contribute to the understanding of the role of intelligent data processing systems in the modern world of aviation and their potential to transform the aviation sector and society as a whole.

SECTION 1

THEORETICAL FOUNDATIONS OF INTELLIGENT DATA PROCESSING SYSTEMS

1.1 Overview of intelligent data analysis systems (regression analysis, neural networks)

Intelligent data analysis (IDA) is a key component of intelligent data processing systems, which allows for the identification of hidden patterns, trends, and dependencies in large data sets. There is a wide range of IDA methods and algorithms that are used depending on the nature of the data and the objectives of the analysis.

Below is an overview of the main methods of intelligent data analysis [1].

Regression analysis is one of the most important statistical analysis methods used for modeling the relationships between the dependent variable (target variable) and one or more independent variables (predictor variables). This method allows determining how changes in the independent variables affect the dependent variable, making it an indispensable tool for forecasting and data interpretation in various fields [2].

Linear regression (Pic. 1.1) is the simplest form of regression analysis, where the dependence between variables is modeled using a linear function [3]. The formula (1.1) of linear regression takes into account one independent variable, and more than one independent variables.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon \tag{1.1}$$

where:

- *Y - dependent variables ,*
- $X_1, X_2, ..., X_n$ *- independent variable ,*
- *β0 - intercept (constant term),*

- $\beta_1, \beta_2, \ldots, \beta_n$ *regression coefficient ,*
- ϵ - *model error.*

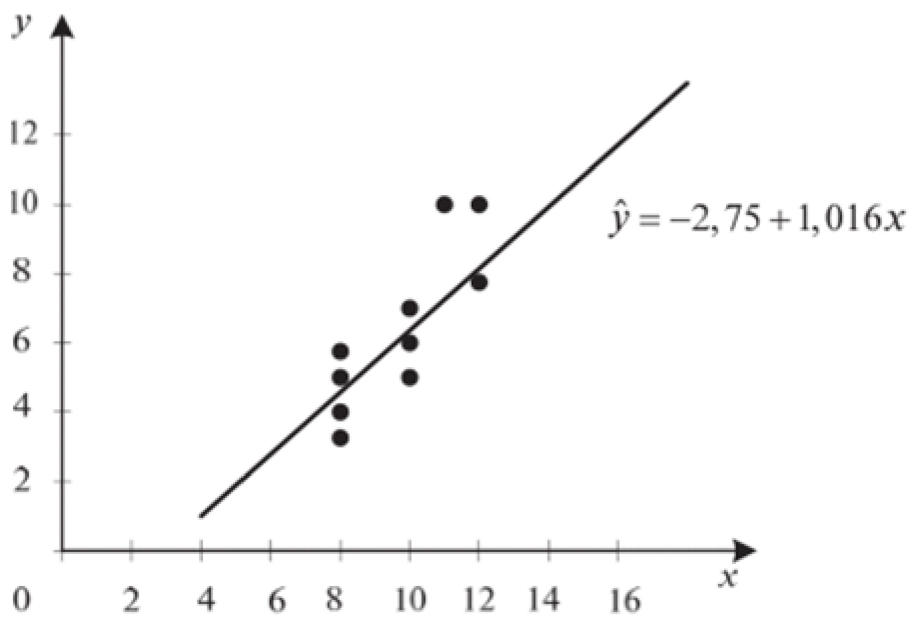


Fig.1.1

Logistic regression (Fig. 1.2) is used to model the probability of an event occurring when the dependent variable is categorical.

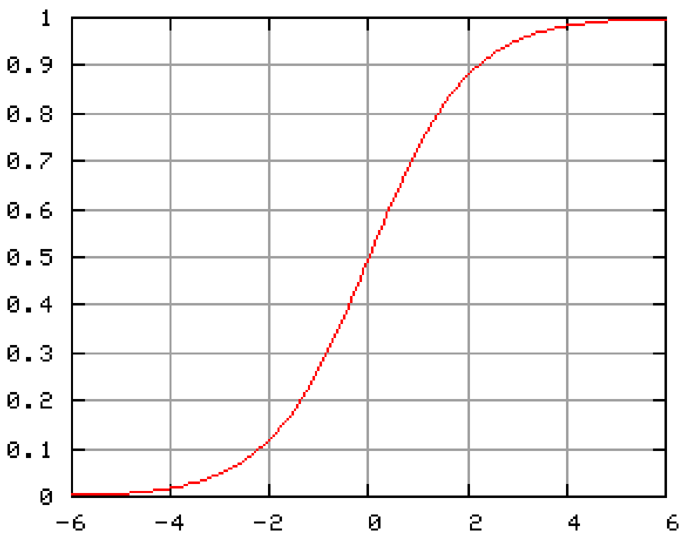


Fig. 1.2

The main equation of logistic regression is shown in formula 1.2:

$$P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_n X_n)}}$$

(1.2)

where:

- *P(Y=1) - the probability of the event occurring,*
- *X1, X2, Xn - independent variables,*
- *B0 - the intercept,*
- *B1, B2, ..., Bn - the regression coefficients.*

Polynomial regression (1.3) is used when the relationship between the dependent and independent variables is nonlinear [4]. The polynomial regression model includes powers of the independent variables:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \epsilon$$

(1.3)

Ridge and Lasso regressions (Fig. 1.3) are methods used to address the problem of multicollinearity (when independent variables are highly correlated with each other) and for performing feature selection [5].

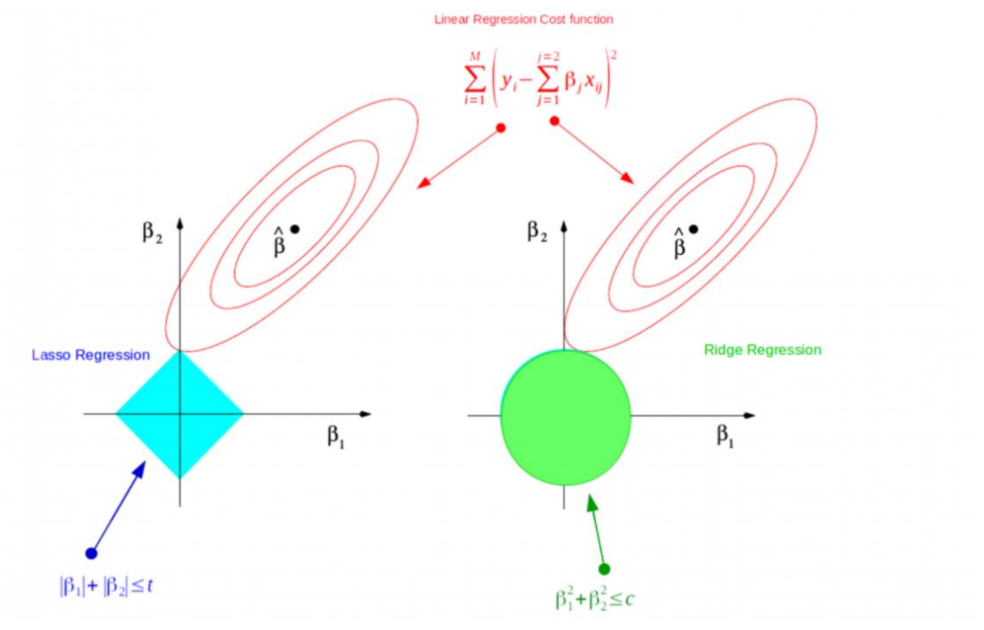

Linear Regression Cost function
$\sum_{i=1}^{M}\left(y_i-\sum_{j=1}^{j=2}\beta_j x_{ij}\right)^2$
β_2
$\hat{\beta}$
Lasso Regression
β_1
$|\beta_1|+|\beta_2|\leq t$
β_2
$\hat{\beta}$
Ridge Regression
β_1
$\beta_1^2+\beta_2^2\leq c$

Fig. 1.3

Ridge regression adds a penalty term for the magnitude of the coefficients to the loss function, which reduces their amplitude.

Lasso regression (1.4) adds a penalty term for the absolute value of the coefficients, which can lead to some coefficients being set to zero and automatic feature selection [6].

$$\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\lambda\sum_{j=1}^{p}|\beta_j|$$

(1.4)

Regression analysis has widespread applications in various fields:

- Forecasting market prices, cost and revenue analysis, risk assessment.
- Identifying disease risk factors, analyzing treatment effectiveness.
- Analyzing consumer behavior, forecasting product demand.
- Modeling system characteristics, reliability analysis.

Advantages and limitations of regression analysis:

Advantages

- Simplicity and interpretability of results.

- Ability to quantify the impact of each independent variable on the dependent variable.

- Wide range of applications across various fields.

Limitations

- Linearity assumptions may be too simplistic for complex relationships.

- Sensitivity to multicollinearity and outliers in the data.

- Need for prior data analysis and preparation.

Regression analysis is a powerful tool for modeling relationships between variables, allowing for forecasting and informed decision-making. Understanding its main methods and applications is crucial for effective data utilization in various scientific and business domains.

Neural networks are one of the key technologies in the field of intelligent data analysis.

They model the workings of the human brain by using artificial neurons interconnected through weights. Neural networks are capable of discovering complex patterns in data, making them extremely useful for various tasks, including classification, regression, clustering, and forecasting [7].

The main architecture of a neural network consists of three types of layers:

1. Input layer, consisting of neurons that receive input data.
2. Hidden layers, consisting of neurons that perform intermediate computations. Multiple hidden layers can be used, allowing the neural network to capture more complex patterns.
3. Output layer, consisting of neurons that generate output data based on the computations of the previous layers.

There are several main types of neural networks, each with its own characteristics and application areas:

1. Perceptron - the simplest type of neural network, consisting of only a single layer of neurons. Used for simple classification tasks.

2. Multilayer Perceptron (MLP) - consists of one or more hidden layers. Used for tasks that require more complex data analysis.
3. Recurrent Neural Networks (RNN) - characterized by the presence of feedback loops, allowing them to process sequential data, such as time series or text.
4. Convolutional Neural Networks (CNN) - specialized in processing data with spatial structure, such as images. They use convolution operations to extract features at different levels of abstraction.
5. Generative Adversarial Networks (GAN) - consist of two neural networks, a generator and a discriminator, that compete against each other. Used for generating new data that resembles the training examples.

Training neural networks involves adjusting the connection weights between neurons to minimize the error [8]. The main steps in training include:

1. Forward Pass: Input data propagates through all layers of the network, and an output prediction is obtained.
2. Loss Calculation: Determining the difference between the prediction and the actual value using a loss function.
3. Backward Pass: The error is propagated back through the network, and the weights are adjusted using the backpropagation algorithm and gradient descent.

Neural networks find applications in many fields, including:

- Aviation, for predicting aircraft maintenance, optimizing flight routes, detecting anomalies in aircraft systems, and analyzing sensor data.
- Medicine, for disease diagnosis, medical image analysis, and treatment outcome prediction.
- Finance, for stock price forecasting, credit risk assessment, and fraud detection.
- Automotive industry, for developing autonomous vehicles, analyzing sensor data, and predicting failures.

Major challenges faced by neural networks include:

- High computational resource requirements.
- Need for large training data volumes.
- Difficulty in interpreting results (black box).

Despite these challenges, neural networks possess an immense potential for further advancement and refinement, thus unveiling novel prospects for sophisticated data analysis across diverse domains.

1.2 Overview of Weather Data Analysis Methods

Weather data analysis is a crucial task in meteorology, involving the collection, processing, and interpretation of substantial data volumes to forecast weather conditions, investigate climate change, and facilitate informed decision-making across various domains. In this section, we shall examine the fundamental algorithms employed in weather data analysis [14].

Statistical analysis serves as the foundation for numerous weather data analysis methods.

The primary techniques encompass:

- Descriptive statistics. Utilized to summarize and describe the essential characteristics of weather data (mean, median, standard deviation).
- Correlation analysis. Uncovers relationships between different weather variables (e.g., temperature and humidity).
- Trend analysis. Employed to detect long-term tendencies in weather data, such as global warming.

Regression analysis is applied to model and forecast dependencies between weather variables. The primary types of regression analysis employed are:

- Linear regression. Model’s linear relationships between variables. For instance, forecasting temperature based on historical data.
- Polynomial regression. Utilized when the relationship between variables is nonlinear.

- Logistic regression. Employed to predict the probability of events, such as precipitation or storms.

Time series analysis is a pivotal method for weather data analysis, as weather patterns evolve over time.

Time Series Analysis in Weather Data

Time series analysis is a pivotal method for weather data analysis as weather patterns evolve over time. The primary methods include:

- **Data Collection**
  - The initial stage in working with time series involves data collection. In hydrometeorology, time series represent chronological sequences of observations for various parameters (such as temperature, atmospheric pressure, precipitation, etc.).
  - Types of Time Series: Equidistant (data points are spaced at regular intervals) and non-equidistant (data points are spaced at irregular intervals). An example of an equidistant series could be daily collected air or water temperature data.
  - Interval and Moment Series: Interval series consist of data collected over specific periods (e.g., daily average temperatures), whereas moment series record data at specific points in time (e.g., noon temperature each day).
- **Preliminary Data Processing**
  - This stage involves preparing the data for analysis:
  - Handling Missing Data and Anomalies: Time series often contain missing or outlier values which need to be identified and treated to avoid skewing the analysis.
  - Data Normalization: Bringing data to a common scale allows for the comparison of series with different units or scales.
  - Decomposition into Trend, Seasonal Component, and Noise: This helps in understanding the structure of the time series and preparing data for

further analysis. Decomposition includes isolating long-term trends, regular seasonal patterns, and random variations (noise).

- **Analysis of Stationarity**
  - Checking time series for stationarity is a crucial step in analysis:
  - Stationarity: A time series is considered stationary if its statistical properties (such as mean and variance) remain constant over time. Stationary series are easier to model and forecast.
  - Methods for Testing Stationarity: Methods like the Dickey-Fuller test and the KPSS test help determine whether a series is stationary or requires transformations (e.g., differencing) to achieve stationarity.
- **Model Selection and Building**
  - Choosing the appropriate model for the time series based on its characteristics:
  - Time Series Models: These include autoregressive models (AR), moving average models (MA), mixed ARMA models, and integrated autoregressive models (ARIMA).
  - Model Selection Criteria: Information-theoretic criteria such as the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) assist in selecting the model that best describes the data.
- **Model Evaluation**
  - Assessing the quality of the chosen model:
  - Parameter Estimation: Determining the coefficients of the model (e.g., in ARIMA models).
  - Model Diagnostics: Checking the adequacy of the model through residual analysis — residuals should be white noise, meaning they are random and not autocorrelated.
  - Cross-Validation: Splitting data into training and test sets to evaluate the model on new data.

- **Forecasting**
    - Using the built and evaluated model to make predictions:
    - Short-term and Long-term Forecasts: Short-term forecasts are usually more accurate and useful for operational decisions, while long-term forecasts provide a general outlook of future trends.
    - Forecasting Methods: Utilizing ARIMA models, seasonal models like SARIMA, and machine learning methods for more complex predictions.
- **Visualization of Results**
    - The final stage involves presenting the analysis and forecast results in a clear and understandable form:
    - Time Series Plots: Displaying the original data and forecasts on a time scale.
    - Residual Plots: Allowing assessment of model adequacy.
    - Comparison Plots: Comparing forecasts with actual values to evaluate accuracy.

Following the previously described timing data analysis process, the following block diagram can be designed (see fig. 1.4):

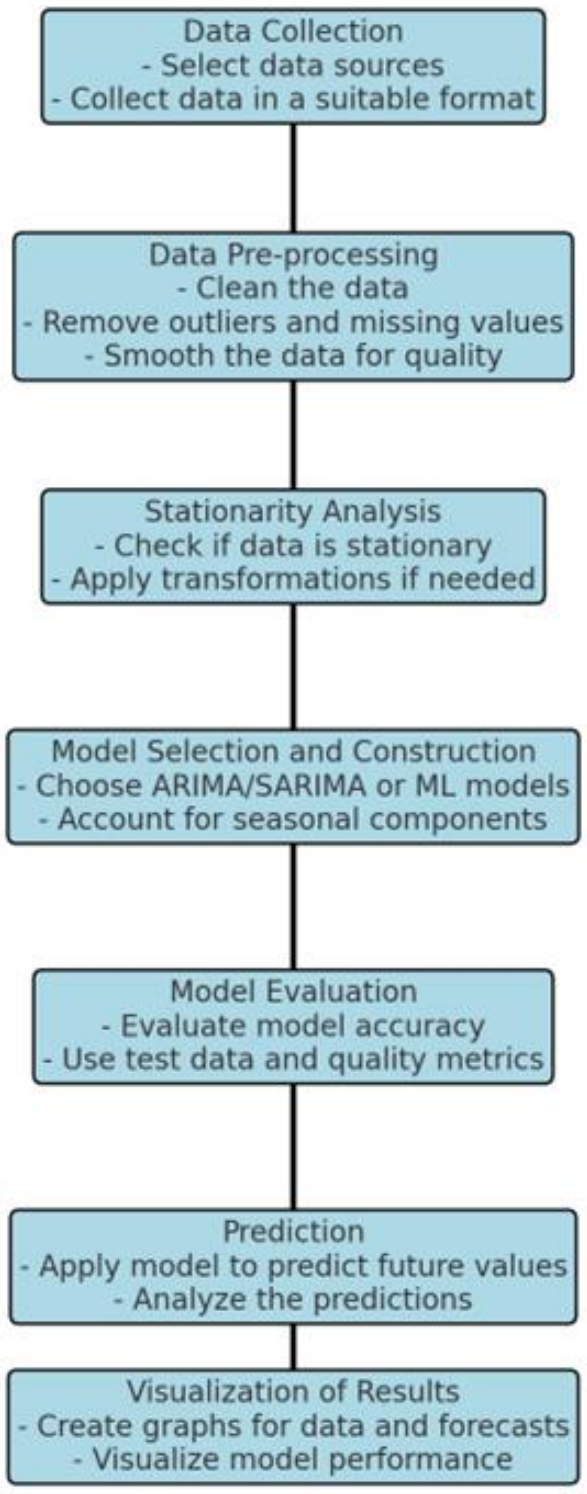

Data Collection
- Select data sources
- Collect data in a suitable format
Data Pre-processing
- Clean the data
- Remove outliers and missing values
- Smooth the data for quality
Stationarity Analysis
- Check if data is stationary
- Apply transformations if needed
Model Selection and Construction
- Choose ARIMA/SARIMA or ML models
- Account for seasonal components
Model Evaluation
- Evaluate model accuracy
- Use test data and quality metrics
Prediction
- Apply model to predict future values
- Analyze the predictions
Visualization of Results
- Create graphs for data and forecasts
- Visualize model performance

Fig. 1.4

Weather data analysis encompasses a wide spectrum of methods, enabling the extraction of valuable information about weather conditions and climate change. The choice of a specific method depends on the nature of the data, the goals of the analysis, and the available resources. Combining various approaches allows for the attainment of the most accurate and reliable results in forecasting and interpreting weather data.

1.3 Description of the used regression analysis systems. Problem statement

Regression analysis encompasses diverse methods, each with its unique characteristics, advantages, and limitations. Let us examine the most prevalent ones in greater detail.

Simple linear regression (1.5) models the relationship between two variables: one independent (predictor) variable X and one dependent (target) variable Y [9]

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{1.5}$$

where:

- Y – is the dependent variable,
- X – is the independent variable,
- $\beta 0$ – is the intercept,
- $\beta 1$ – is the regression coefficient (slope of the line),
- ϵ – is the model error.

This method is used to predict the value of Y based on X and allows for the estimation of the strength and direction of the relationship between the variables.

Multiple linear regression (1.6) extends simple linear regression by allowing for the consideration of more than one independent variable [10]. The primary equation for multiple linear regression is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon \tag{1.6}$$

where:

- Y – is the dependent variable,

- $X1$,$X2$,...,Xn – are the independent variables,
- $\beta 0$ – is the intercept,
- $\beta 1$,$\beta 2$,...,$\beta n \beta 1$ – are the regression coefficients,
- ϵ – is the model error.

Multiple linear regression is used for more accurate modeling of dependencies and to account for the influence of multiple factors on the target variable.

Polynomial regression (see formula 1.7) is utilized when the relationship between the dependent and independent variables is nonlinear [11]. The primary equation for second-order polynomial regression is:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon \tag{1.7}$$

where:

- Y – is the dependent variable,
- X – is the independent variable,
- $\beta 0$,$\beta 1$,$\beta 2$ – are the regression coefficients,
- ϵ – is the model error.

Polynomial regression allows for the modeling of more complex dependencies between variables, which may be nonlinear.

Logistic regression (1.8) is used to model the probability of a certain event occurring when the dependent variable is categorical (binary). The primary equation for logistic regression is:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}} \tag{1.8}$$

where:

- $P(Y{=}1)$ – is the probability of the event occurring,
- $X1$,$X2$,...,Xn – are the independent variables,

- $\beta 0$ – is the intercept,
- $\beta 1,\beta 2,\ldots,\beta n$ – are the regression coefficients.

Logistic regression is often used in classification tasks, for example, to determine the probability of disease or the probability of a customer making a purchase.

Ridge regression is employed to address the issue of multicollinearity by adding a penalty term for the magnitude of the coefficients to the loss function (see formula 1.9) [12]:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

(1.9)

where:

- yi – are the observed values,
- yi – are the predicted values,
- βj – are the regression coefficients,
- λ – is the regularization parameter that controls the degree of penalty.

Ridge regression reduces the magnitude of the coefficients, which helps prevent overfitting of the model.

Lasso regression (1.10) adds a penalty for the absolute value of the coefficients, which can lead to some coefficients being set to zero [13]:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

(1.10)

where:

- yi – are the observed values,

- yi – are the predicted values,
- βj – are the regression coefficients,
- λ – is the regularization parameter.

Lasso regression performs automatic variable selection, making the model simpler and more interpretable.

Regression analysis offers a diverse array of methods for modeling and analyzing data. The choice of a specific method depends on the nature of the data and the research objectives. Successful application of regression analysis requires an understanding of the theoretical foundations of each method and their proper utilization in practical tasks.

Problem statement

Weather forecasting is crucial for various industries, yet predicting weather conditions remains a challenging task due to the complexity and variability of climate data. This challenge shows how this problem can be solved using modern time series analysis and machine learning techniques. The focus is on processing weather data from 2013 to 2017 to identify patterns and develop forecasting models. The challenge is to efficiently process noisy data, apply appropriate normalization and transformation methods, and choose reliable algorithms to improve forecasting accuracy and reliability.

SECTION 2
METHOD FOR SOLVING THE PROBLEM

2.1 Data preparation and preprocessing

To construct a forecasting model for any parameter, it is necessary to obtain a raw dataset comprising an array of parameters that describe a particular phenomenon. The data must be structured in the form of tables with defined fields, such as the observation date, air temperature, wind speed, and precipitation occurrence.

Additionally, data can be collected from weather websites, including information about droughts, tsunamis, and other phenomena.

The collected data exhibited diverse characteristics. Most of the company's internal data were structured, simplifying their processing and analysis.

Web data and data from external sources could be both structured and unstructured, necessitating additional processing for use in analysis.

After data collection, an essential step in processing is cleaning the data by removing errors and incomplete records.

Checking for duplicate data is the first step in data cleaning. Duplicate records can distort analysis and lead to incorrect conclusions. Unique identifiers or combinations of fields can be used to detect duplicates by checking for identical records.

Data often contain missing values that must be addressed before analysis. This can involve imputing values using means, medians, or modes, or employing other methods such as interpolation.

Outliers or anomalies in the data can arise due to measurement errors, random events, or other causes. Detecting and removing outliers helps ensure the correctness of the analysis results. This can be accomplished using statistical methods, such as standard deviation or interquartile range.

Sometimes, data may contain erroneous values that need to be corrected. This can include fixing typos, incorrect formats, or other types of errors.

After completing these cleaning steps, the data become ready for further analysis and modeling in intelligent data processing systems.

During data analysis, it is crucial to consider that not all features present in the original dataset are useful and informative for model construction. It is essential to select the most informative features for further analysis and modeling.

Conducting a correlation analysis between the features and the target variable helps identify the features that have the strongest relationship with the target variable. Features with high correlation may be more informative for modeling.

Various statistical feature selection methods exist, such as feature importance analysis, principal component analysis (PCA), or feature selection based on statistical tests (e.g., t-test).

Some machine learning models provide information about the importance of each feature in the constructed model. For instance, a decision tree can provide the importance of each feature based on its contribution to improving the node criterion.

In some cases, expert knowledge or domain expertise may indicate which features are likely to be most important for the model.

After selecting the most important features, the dataset is ready for further use in modeling.

Some machine learning algorithms can be sensitive to the scale of features. For example, methods that use distance between points, such as the k-nearest neighbors (k-NN) method, can be significantly affected by the size and range of feature values. At this stage, data scaling is performed to ensure a consistent scale for all features.

Normalization (min-max scaling). In this method, the values of each feature are transformed so that they fall within the range of 0 to 1. This is achieved by subtracting the minimum value of the feature and dividing by the difference between the maximum and minimum values.

Standardization. In this method, the values of each feature are transformed so that they have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean value of the feature and dividing by the standard deviation.

In addition to normalization and standardization, other scaling methods exist, such as logarithmic scaling or rank-based scaling.

Categorical features, which have a limited number of possible values, need to be encoded as numerical values before being used in a machine learning model. This is an important step because many machine learning algorithms work only with numerical data.

In the One-Hot Encoding method, each unique value of a categorical feature is transformed into a new binary feature. For example, if wind was present, the feature receives a code of 1; if wind was absent, it receives a code of 0. In other words, for each unique value of the feature, a new column is created that takes the value 1 if that value is present for a given record or 0 if it is absent. This approach is particularly useful when the categorical feature has many unique values.

In the Label Encoding method, each unique value of a categorical feature is encoded as an integer. Each unique value is assigned a unique identifier, typically starting from 0. This approach is suitable for categorical features with ordered values or when the number of unique values is relatively small.

When choosing a method for encoding categorical features, it is important to consider the characteristics of the data and the requirements of the specific machine learning model. Proper encoding helps ensure the correctness and efficiency of the model when processing categorical data.

Splitting the dataset into training and test sets is an important step in the process of developing a machine learning model. It allows for evaluating the model's effectiveness on independent data and avoiding overfitting. Here's how it can be done:

- First, determine what portion of the data needs to be allocated for testing the model. Typically, between 10% and 30% of the total data volume is used.

- To ensure objectivity, randomly divide the data into training and test sets. Ensure that each record has an equal chance of being in either set.
- If working on a classification task and dealing with different classes, ensure that both sets have approximately the same number of examples for each class. This will help avoid biases when evaluating the model.
- After splitting, save the training and test data sets so that they can be reused and ensure consistency of results.

Proper splitting of data into training and test sets helps ensure an objective evaluation of the model and its ability to generalize to new data.

2.2 Development of the regression analysis algorithm

Linear regression (fig. 2.1) is one of the simplest regression methods.

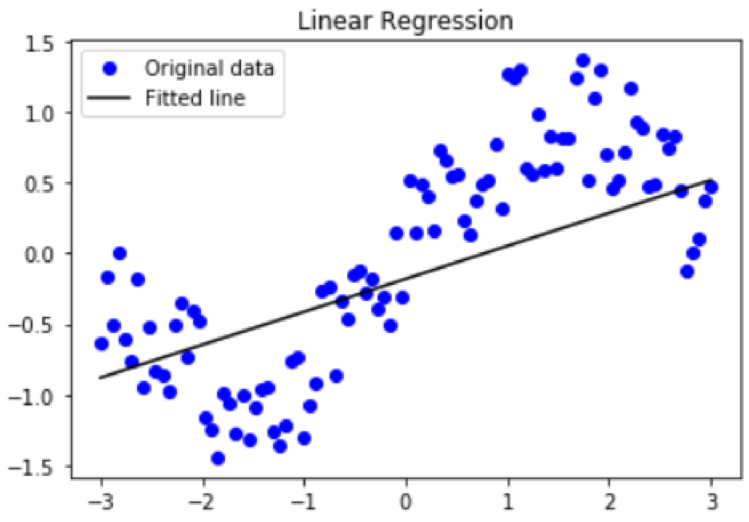


Fig. 2.1

It assumes a linear relationship between the independent and dependent variables.

Linear regression is often used in cases where the relationship between variables is approximately linear. It is well-suited for simple tasks and cases where the number of features is small.

The linear regression model is computationally efficient and easy to interpret, but it may be ineffective in cases where the relationship between variables is complex or non-linear.

Polynomial regression (fig. 2.2) extends the linear model by adding polynomial features to the model.

It is used when the relationship between variables is not linear, but can be approximated by a polynomial of a certain degree.

Polynomial regression can be more flexible than simple linear regression, but it can also lead to overfitting, especially when using high-degree polynomials.

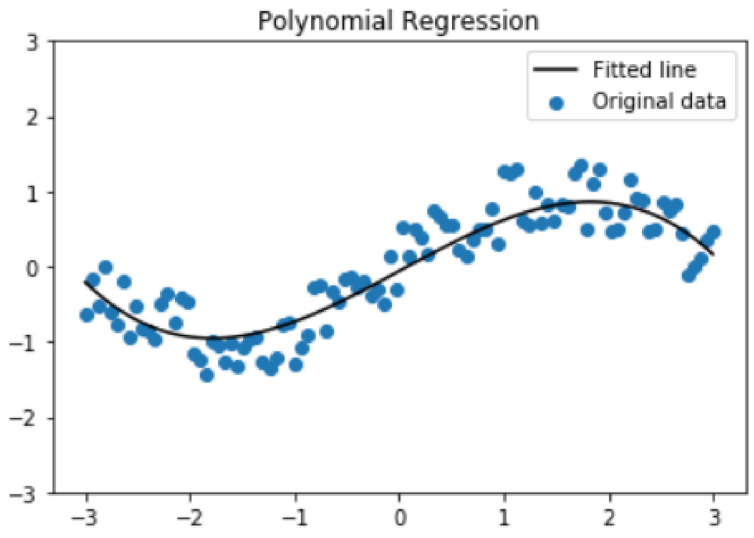


Fig. 2.2

Decision tree (fig. 2.3) regression is used for predicting values based on a decision tree, where each node represents a condition, and each leaf value represents a predicted value.

This method can be effective for non-linear dependencies between variables and has the inherent ability to automatically handle feature interactions.

This data mining method is also known as decision rule trees, classification and regression trees.

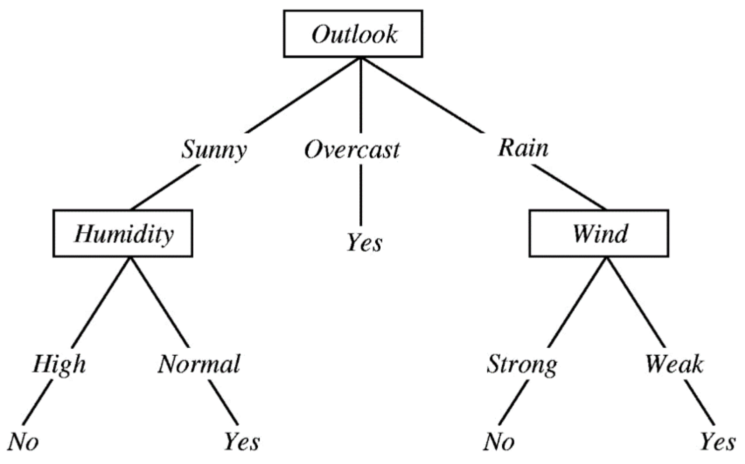


Fig. 2.3

If the dependent, or target, variable takes discrete values, the decision tree method solves a classification task.

If the dependent variable takes continuous values, the decision tree establishes the relationship between this variable and the independent variables, solving a numerical forecasting task. Decision tree regression can become complex as the tree depth increases and may be prone to overfitting.

Neural networks (fig. 2.4) are a set of interconnected artificial neurons that can perform complex computations and function approximations.

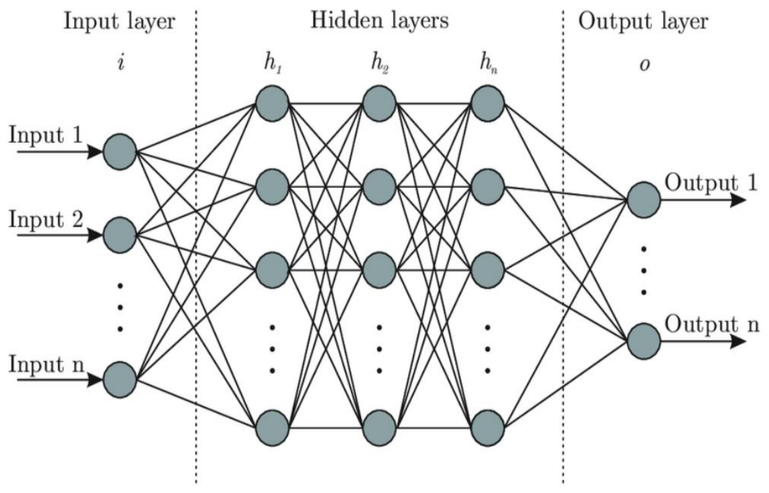


Fig. 2.4

They are used in regression tasks when the relationship between variables is complex and does not have an explicit form.

Neural networks can be very powerful in solving complex problems, but they can also be difficult to train and require large amounts of data to work effectively.

Preparing a dataset for subsequent use with a linear regression model involves the following steps:

- Data analysis for missing values.
- Data scaling using normalization or standardization methods to ensure a consistent scale for all features.
- Encoding of categorical features. For linear regression models, the "one-hot encoding" method is typically used, which transforms categorical features into binary variables, allowing their use in linear models.
- The final step is splitting the data into training and test sets. The training set will be used to train the model, while the test set will be used to evaluate its effectiveness and avoid overfitting.

After data preparation, the training phase of the linear regression model begins, during which the model "learns" the dependencies between the input and output variables.

To create a linear regression model, it is necessary to establish initial values for the parameters and regression coefficients.

After initialization, the model is fitted to the training dataset. This process involves finding the optimal values of the model parameters that best fit the relationship between the input and output variables.

After completing the model fitting, its effectiveness must be evaluated. This can include analyzing various regression metrics, such as Mean Squared Error (MSE) or the coefficient of determination (R-squared), to assess how well the model fits the data.

After evaluating the model's effectiveness on the training dataset, it can also be tested on the test dataset, which was previously separated from the training set. This helps determine how well the model generalizes to new data, i.e., how it performs under real conditions.

After completing these steps, the linear regression model is ready for use in predicting values of the dependent variable based on new input data.

The obtained regression metrics are interpreted to determine the effectiveness of the model. For example, low values of MSE and MAE, and a high value of R-squared indicate that the model performs well and accurately predicts the target variable.

It is also important to compare the results with a baseline level (e.g., predictions obtained using simple methods such as predicting the mean value). This helps determine whether the model is indeed making a significant contribution to the forecasting.

After evaluating the model, decisions can be made regarding its further use, tuning, or improvement, depending on the results and the requirements of the specific study.

If necessary, the linear regression model can be tuned or optimized to improve its effectiveness.

The model parameters, such as the regression coefficients, can be adjusted to achieve a better fit to the data. This may involve optimizing the coefficients using optimization methods, such as gradient descent, to minimize the loss function.

To avoid overfitting and improve the overall generalization ability, regularization methods such as L1 (Lasso) or L2 (Ridge) regularization can be employed. These methods help control the magnitude of the regression coefficients by adding penalty terms for the parameter sizes to the loss function.

Cross-validation can be applied to determine the optimal values of the model's hyperparameters. This process helps avoid overfitting and improve the model's robustness.

After tuning the model, it is necessary to analyze the results to verify its effectiveness. This includes evaluating the regression metrics on the test dataset and comparing them with the previous results.

It is important to continuously monitor the model's effectiveness and make timely adjustments if necessary. Data can change over time, so the model must remain relevant.

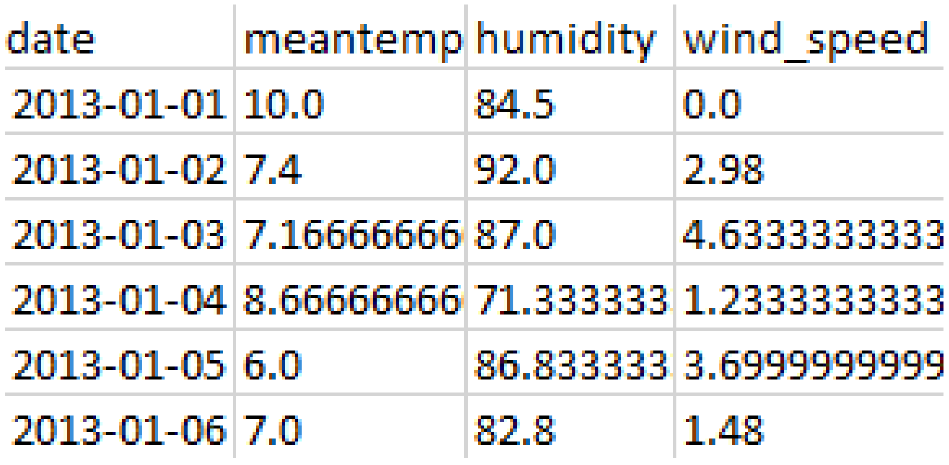
These steps help improve the effectiveness of the linear regression model and ensure its optimal performance under real-world application conditions.

SECTION 3

EXAMPLE OF WEATHER DATA PROCESSING AND RESULTS

3.1 Description of the weather data calculation and analysis program

To conduct the analysis, a dataset of temperature measurements from January 1, 2013 to April 24, 2017 was selected, containing 1575 records. The data was stored in CSV format and processed in the Python environment, including columns with dates in the YYYY-MM-DD format, average temperature, humidity, wind speed, and average pressure. An example of the data can be seen in Figure 3.1.



date	meantemp	humidity	wind_speed
2013-01-01	10.0	84.5	0.0
2013-01-02	7.4	92.0	2.98
2013-01-03	7.16666666	87.0	4.6333333333
2013-01-04	8.66666666	71.333333	1.2333333333
2013-01-05	6.0	86.833333	3.6999999999
2013-01-06	7.0	82.8	1.48

Fig. 3.1

Feature Description

- **Date**: This feature indicates the date of weather condition measurement.
- **Temperature (Celsius**): This feature shows the air temperature in degrees Celsius at the time of measurement.
- **Humidity (%):** This feature indicates the air humidity as a percentage.
- **Wind Speed (km/h):** This feature shows the wind speed in kilometers per hour.

Usage

This dataset can be used for:

- Forecasting air temperature based on weather conditions.

- Understanding the impact of temperature, humidity, and wind speed.
- Improving planning and management in the aviation, energy, and infrastructure sectors.

The data analysis and processing methodology includes several stages:

- Collecting the necessary data.
- Preprocessing the data.
- Developing and implementing the model, selecting the type of neural network and its parameters.
- Testing on historical data.
- Optimizing the parameters.

By choosing Python for conducting the research, we made the right choice, as this language is interpreted, which simplifies debugging, and has a wide selection of modules in both the standard package and third-party ones. We can plan programs at a higher level, using ready-made elements that implement various functions. Python provides absolute portability of programs, and differences in behavior across different operating systems are easy to predict thanks to detailed documentation.

The main library for our tasks will be Keras, which provides a simple API for creating neural networks. With Keras, we can quickly build a neural network using just a few lines of code. Keras is built on top of the TensorFlow framework, which provides the ability to express computations as data flows through a state graph.

In the process of working with data and data structures, we will use the NumPy and Pandas libraries. NumPy provides the ability to work with arrays, matrices, and functions related to these structures, while Pandas allows for manipulating and processing numerical tables and strings.

It is worth emphasizing that the use of clean input data is of great importance for achieving accurate forecasting. Data normalization helps make the model more efficient. For plotting, we will use the Matplotlib library, which offers a convenient object-oriented approach for embedding plots in applications.

3.2 Example of Data Processing Using Regression Analysis

Data Processing:

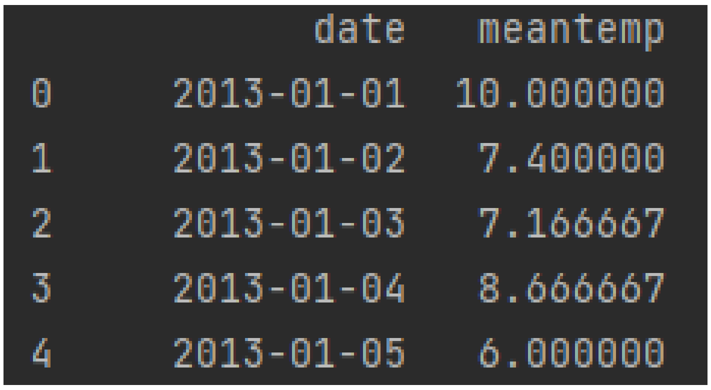
Before using the data to train a model, the following processing steps are required:

- Removal of missing values or filling them in using interpolation or averaging methods.
- Feature scaling to ensure homogeneity.
- Splitting the data into training and test sets for model evaluation.

Let's assume we have a temperature dataset with two columns: "Date" and "Temperature". For convenience of analysis, we will only use the temperature data as shown in Fig. 3.2.

The first step will be to convert the "Date" column into integers to work with the data in a numerical format. We can replace each date with the corresponding number of days elapsed since a certain initial date.

We will choose the initial date as "2013-01-01", so we will replace the date "2013-01-01" with 0, "2013-01-02" with 1, and so on. In this way, we will convert the "Date" column into integers representing the number of days from the initial date, as shown in Fig. 3.2.

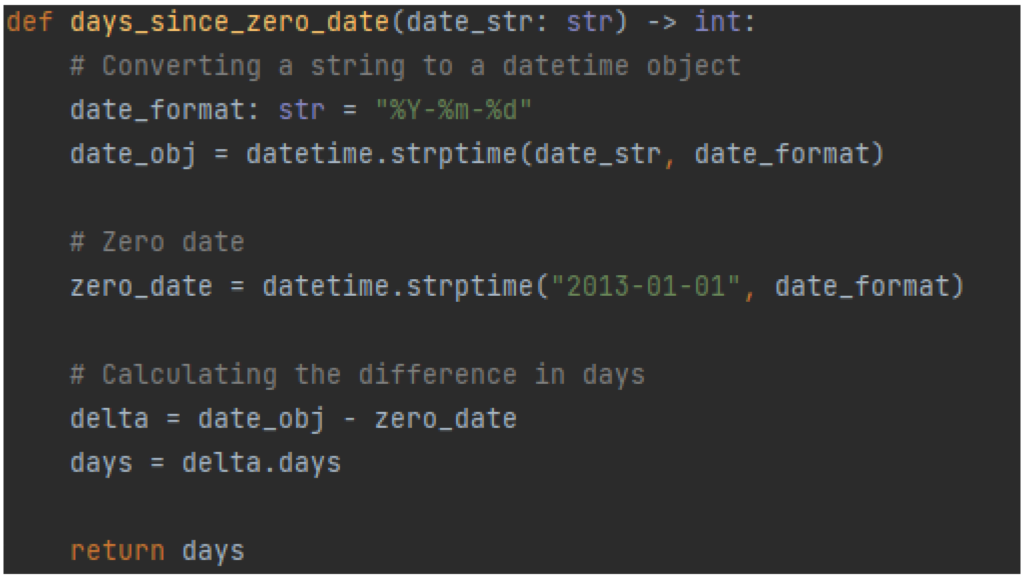


```
             date   meantemp
0      2013-01-01  10.000000
1      2013-01-02   7.400000
2      2013-01-03   7.166667
3      2013-01-04   8.666667
4      2013-01-05   6.000000
```

Fig. 3.2

After this, time series analysis can be performed using these numerical data. Additionally, visualizations can be used to observe trends and patterns in the data, decomposition into components to identify trends and seasonality, and forecasting models to predict future temperature values. This approach allows for effective analysis of time series data by converting dates into numbers and using various analysis methods to obtain useful information.

The function shown in Fig. 3.3 is intended to convert a string containing a date into the number of days elapsed since a certain initial date, specifically 2013-01-01.



```python
def days_since_zero_date(date_str: str) -> int:
    # Converting a string to a datetime object
    date_format: str = "%Y-%m-%d"
    date_obj = datetime.strptime(date_str, date_format)

    # Zero date
    zero_date = datetime.strptime("2013-01-01", date_format)

    # Calculating the difference in days
    delta = date_obj - zero_date
    days = delta.days

    return days
```

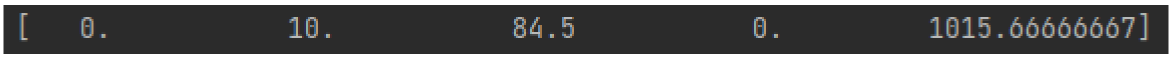
Fig. 3.3

The steps performed by this function are:

1. Defining the date format. The date format to be used for converting the string into a datetime object is specified. In this case, the format "%Y-%m-%d" means that the date should be in the "Year-Month-Day" format.
2. Converting the string into a datetime object. The input string with the date is converted into the corresponding datetime object using the specified format.
3. Defining the initial date. The initial date (in this case, 2013-01-01) is set, from which the number of days will be calculated.

4. Calculating the difference in days. The difference between the input date and the initial date is calculated, and the number of days is extracted from this difference using the .days attribute.
5. Returning the result. The function returns the number of days elapsed since the initial date to the input date.

Using the conversion function (see Fig. 3.3), it can be applied to all dates in the dataset, and the result (Fig. 3.4) shows that all dates have been converted to days.



```
[   0.          10.          84.5          0.        1015.66666667]
```

Fig. 3.4

The order of execution of the code (Fig. 3.5) for the result of converting the date to days was as follows:

1. Applying the function to all dates in the dataset.

The previously defined days_since_zero_date function is called using np.vectorize, which allows the function to be applied to each element in the train_data[:, 0] and test_data[:, 0] arrays.

The function converts each date to the number of days elapsed since the initial date (2013-01-01).

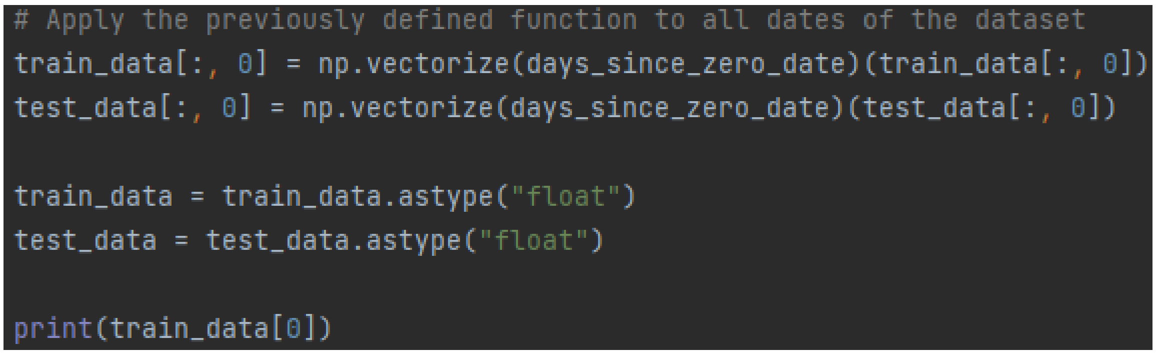
The conversion results are written back into the corresponding columns of the datasets.

2. Converting the data to the "float" type.

After converting the dates to the number of days, we convert all data in the train_data and test_data sets to the "float" type to ensure compatibility with any data operations or analysis that may be required.

3. Printing the first values from train_data

This line of code prints the first values of the converted train_data set for result verification.



```python
# Apply the previously defined function to all dates of the dataset
train_data[:, 0] = np.vectorize(days_since_zero_date)(train_data[:, 0])
test_data[:, 0] = np.vectorize(days_since_zero_date)(test_data[:, 0])

train_data = train_data.astype("float")
test_data = test_data.astype("float")

print(train_data[0])
```

Fig. 3.5

The main purpose of this code block is to convert dates to the number of days and convert the data to the "float" type, preparing them for further analysis. The results are printed for verification.

Using the Z-normalization method (see formula 3.1), we transform each sample value into a new value with a mean of 0 and a standard deviation of 1.

$$x_i' = \frac{x_i - \underline{X}}{\sigma_x}$$

(3.1)

where:

1. $\underline{X}$ – Sample mean (Mean).

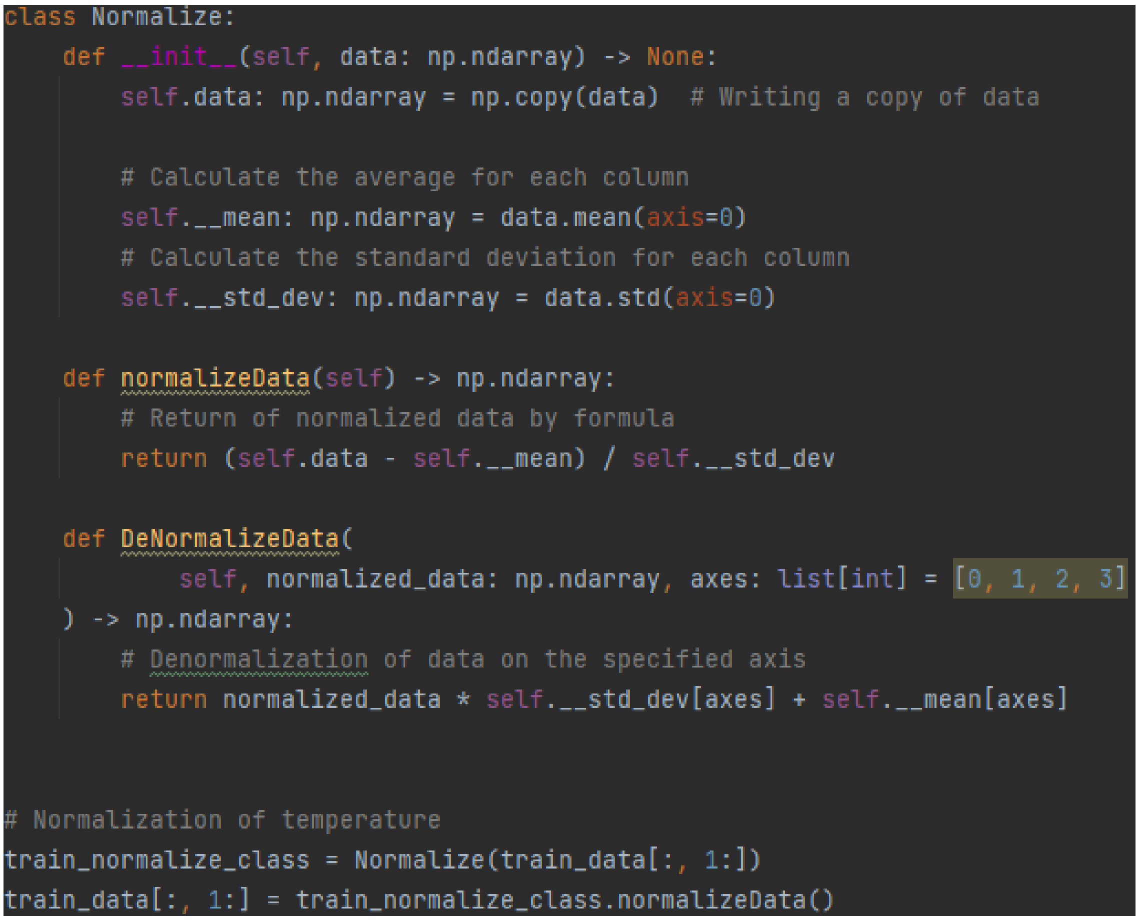
   $\underline{X}$=1n∑i=1$nxix$¯=n1∑i=1nxi,

   where, n - is the number of values in the sample, xi - is each sample value
2. σ_x –Sample standard deviation (Standard Deviation).

   σ=1n∑i=1n(xi−x¯)2σ=n1∑i=1n(xi−x¯)2

   where, x¯ - is the sample mean

Normalized Value. The result of the previous step is a normalized value with a mean of 0 and a standard deviation of 1.



```python
class Normalize:
    def __init__(self, data: np.ndarray) -> None:
        self.data: np.ndarray = np.copy(data)  # Writing a copy of data

        # Calculate the average for each column
        self.__mean: np.ndarray = data.mean(axis=0)
        # Calculate the standard deviation for each column
        self.__std_dev: np.ndarray = data.std(axis=0)

    def normalizeData(self) -> np.ndarray:
        # Return of normalized data by formula
        return (self.data - self.__mean) / self.__std_dev

    def DeNormalizeData(
            self, normalized_data: np.ndarray, axes: list[int] = [0, 1, 2, 3]
    ) -> np.ndarray:
        # Denormalization of data on the specified axis
        return normalized_data * self.__std_dev[axes] + self.__mean[axes]


# Normalization of temperature
train_normalize_class = Normalize(train_data[:, 1:])
train_data[:, 1:] = train_normalize_class.normalizeData()
```

Fig. 3.6

Thus, as shown in Fig. 3.6, the Z-normalization process involves calculating the mean and standard deviation of the sample, and then applying the formula to normalize each value in the sample to obtain new, standardized values.

The main steps in this process were:

1. Creating the Normalize class:

- In the __init__ constructor, saving a copy of the input data and calculating the mean and standard deviation of each column.
- The normalizeData method uses the calculated mean and standard deviation values to normalize the input data.
- The DeNormalizeData method denormalizes the normalized data using the stored mean and standard deviation values.

2. Normalizing the data:

- Creating an instance of the Normalize class, passing it the data to be normalized.
- Calling the normalizeData method, which returns the normalized data.
- Applying the normalized data to the original ***train_data*** dataset.

To obtain a visual result, it is necessary to plot the normalized data and, for comparison, the actual dataset data. For this, the Matplotlib library can be used, which offers a convenient object-oriented approach for embedding plots. Therefore, the following actions and code (see Fig. 3.7) need to be performed:

1. Import the library. Import the matplotlib.pyplot library for plotting.
2. Create the plot area. Using plt.subplots, create a plot area with one row and two columns to accommodate two plots.
3. Set the plot parameters.
4. Display data on the plots:
   - The first subplot will display the normalized temperature data.
   - The second subplot will display the denormalized temperature data using the DeNormalizeData method from the Normalize class.
5. Display the plots. Finish the code by displaying the plot on the screen using plt.show().

```
        import matplotlib.pyplot as plt

# Create a graph field with one row and two columns
fig, ax = plt.subplots(1, 2)

# Setting limits on the y-axis for both subcharts
ax[0].set_ylim([-10, 40])
ax[1].set_ylim([-10, 40])
```

```
# Axis signatures
ax[0].set_ylabel("Temperature")
ax[0].set_xlabel("Day")
ax[1].set_xlabel("Day")

# Setting titles for subgraphs
ax[0].set_title("Normalized temperature ")
ax[1].set_title("Real temperature")

# Turn on the grid on the axes for both subgraphs
ax[0].grid()
ax[1].grid()

# Display normalized and non-normalized data on graphs
# Graph of normalized temperature
ax[0].plot(train_data[:, 1], c="b", linewidth=1)

# Graph of real temperature (denormalized)
ax[1].plot(
    train_normalize_class.DeNormalizeData(train_data[:, 1], axes=[0]),
    c="r",
    linewidth=1,
)

# Displaying a graph
plt.show()
```

Fig. 3.7

We obtain the result (see Fig. 3.8) of comparing two plots, from which it can be seen that the normalized data did not lose their informational value, are easily restored to real data (second plot), and are more convenient for analysis using machine learning models.

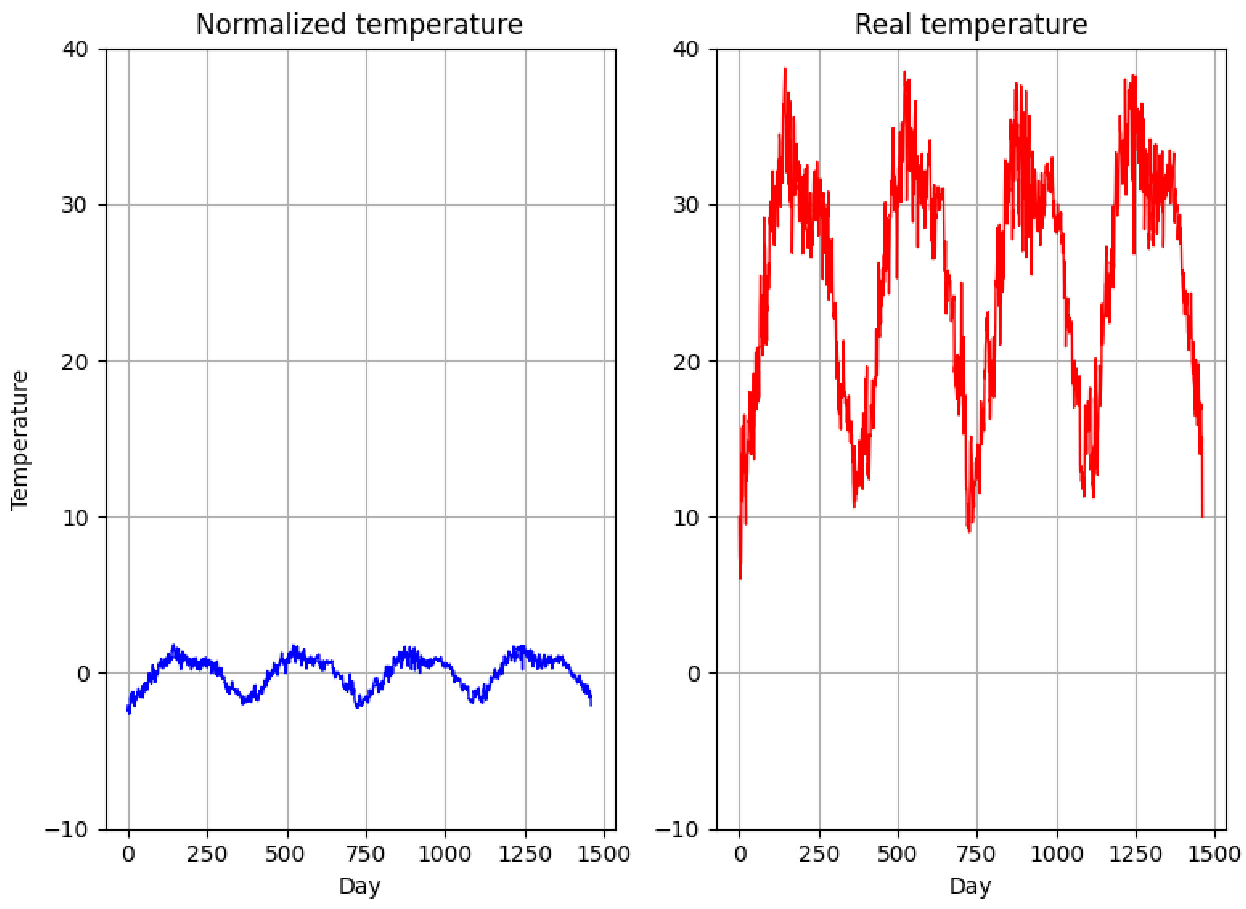


Fig. 3.8

After obtaining the data plot, it is necessary to reduce the influence of noise on its visualization and analysis. One effective method for this is **noise smoothing using the moving average method.**

The moving average method (see Fig. 3.9) involves moving a window of a certain size across the entire plot. For each window shift, the average value of all points within this window is calculated. The obtained average value is used as the value at a certain point on our smoothed plot.

The main idea is that the noise present in the data is pushed away from the average value of the data within the window. Since noise is random, it is compensated for by the average value calculated at each point. This helps to smooth out fluctuations and makes the plot more homogeneous.

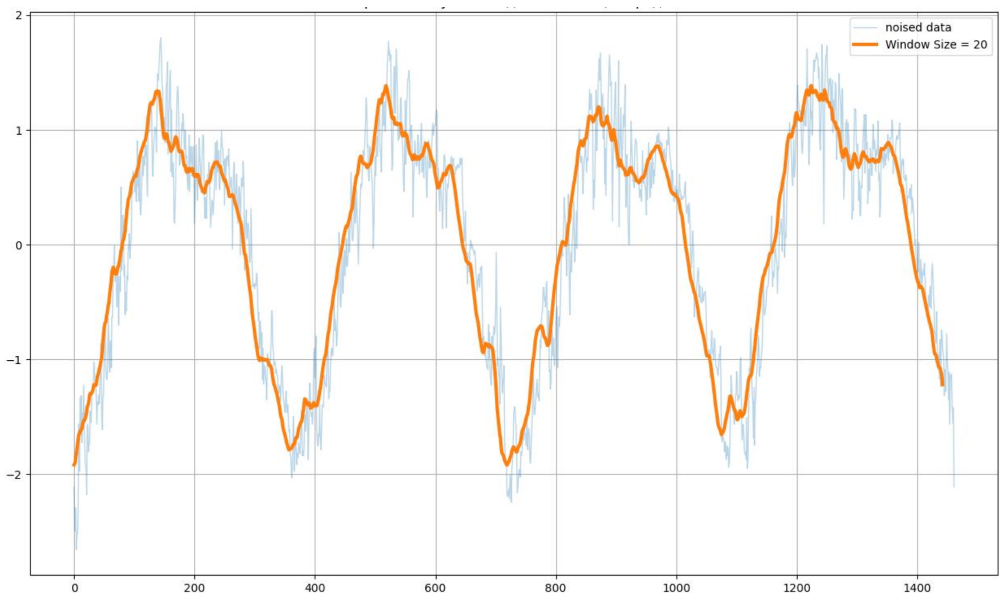


Fig. 3.9

For some tasks, smoothing can be useful, but for others, it may lead to the loss of important information or interference with the nature of the data.

The simple moving average value at a point is the average value calculated for a particular data point by taking the arithmetic mean of all values in the window that this point represents. In other words, if we have the original function f(x), the simple moving average value for the point x will be the arithmetic mean of the values at all points within the window around the point x.

The number of values from the original function for calculating the moving average (“window” size) is the number of points taken to calculate the moving average at a particular point.

The value of the original function at a point is the actual value of the original function f(x) at a particular point x.

Thus, the simple moving average method is used for data smoothing, noise reduction, and trend extraction by calculating the average value for each data point based on the values in a window around it.

Before starting the forecasting, it is necessary to analyze the input data. From the graph, we can see that the temperature has a periodicity similar to a sine wave. This indicates the possibility of modeling using sine waves or their combinations. For this, we will use the Fourier transform and gradient descent.

The main goal is to find the most suitable sine wave (or combination of sine waves) to best reproduce the temperature graph. This means that we can find the parameters of amplitude, frequency, and phase of the sine wave that best approximate our data.

The Fourier transform is used for analyzing periodic signals. We can use the Fourier transform to decompose the temperature signal into sinusoidal components with different frequencies and amplitudes. Gradient descent can be used to optimize the parameters of the sine waves or their combinations. We can define a cost function that measures the difference between the predicted temperatures and the actual data, and use gradient descent to find the optimal parameter values.

The Fourier transform is a mathematical operation used to decompose a function into a sum of sines and cosines with different frequencies (see Fig. 3.10). The main idea is that any complex function can be represented as a sum of simple harmonic oscillations.

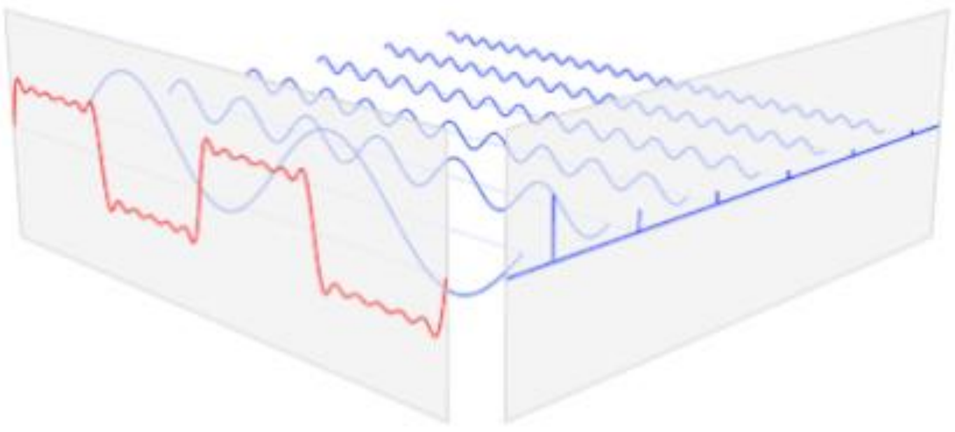
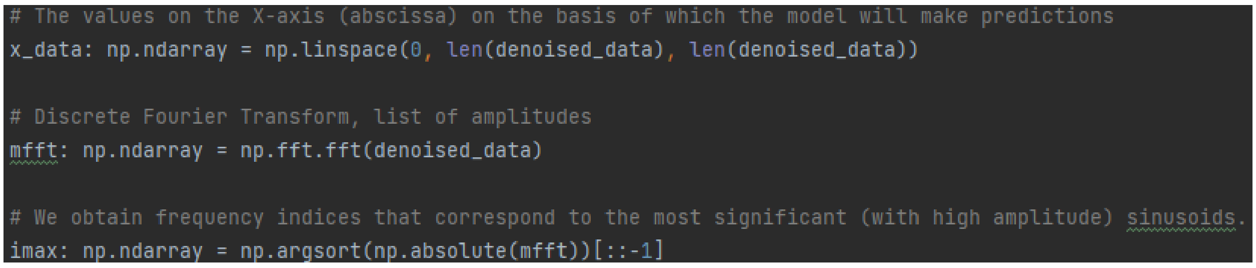


Fig. 3.10

The transform associates the original function with its harmonic oscillation components, each with its own amplitude (magnitude of oscillations) and frequency (number of oscillations per unit time). The obtained amplitudes determine the contribution of each component to the overall signal, and the frequencies show which frequencies are present in the signal.

The Fourier transform allows us to decompose a complex function into simple components - sines and cosines with different frequencies, which allows us to analyze and understand the structure of the data, detect periodicity and other signal characteristics. We will implement the Fourier transform in code as shown in Figure 3.11.



```
# The values on the X-axis (abscissa) on the basis of which the model will make predictions
x_data: np.ndarray = np.linspace(0, len(denoised_data), len(denoised_data))

# Discrete Fourier Transform, list of amplitudes
mfft: np.ndarray = np.fft.fft(denoised_data)

# We obtain frequency indices that correspond to the most significant (with high amplitude) sinusoids.
imax: np.ndarray = np.argsort(np.absolute(mfft))[::-1]
```

Fig. 3.11

The transform is widely used in signal analysis, including in seismology, radio engineering, image processing, and other fields. It allows us to detect and analyze signal characteristics such as frequencies, amplitudes, and phases.

Therefore, the Fourier transform is a powerful tool for analyzing and understanding the structure of signals, which helps to detect periodicity and other characteristics of weather data.

After transforming our data, we will obtain amplitudes for different frequencies. It is advisable to choose the frequencies that have the highest amplitudes, since they have the greatest influence on the overall approximation of our data. After selecting the sine waves with the maximum amplitude, we can freely construct a graph of their sum. This means that individual sine waves are added together with their respective amplitudes.

By visually evaluating the sum of the sine waves, we should understand how well it approximates our data. If the approximation looks good, it may indicate that the sine waves are indeed well-suited for modeling the data.

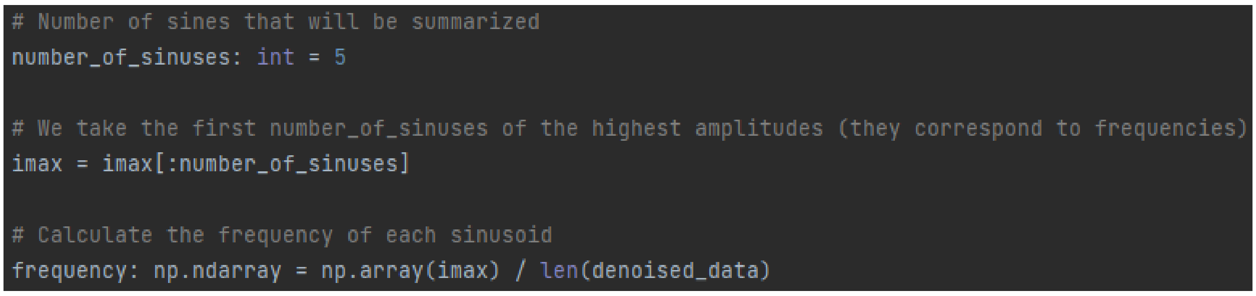
If the approximation is not satisfactory, you need to return to the previous steps and choose other sine waves or adjust parameters such as amplitudes and phases to improve the results.

Thus, visualization of the Fourier transform helps us understand how well the chosen sine waves correspond to our data and allows us to choose the best model for further analysis and forecasting.

The process of obtaining the parameters of the sine waves using the Fourier transform and the gradient descent method will be described in more detail in Fig. 3.11:

1. First, values are prepared for the X-axis, which will be used for making forecasts. A sequence of values on the X-axis is created using np.linspace, which represents from 0 to the length of our data.

2. The Discrete Fourier Transform is applied to the data to obtain a list of amplitudes for different frequencies. This allows us to transform the time signal into the frequency domain.
3. The most significant frequencies corresponding to the highest amplitudes are selected. For this, the indices of the amplitudes are sorted in reverse order by their absolute values.
4. We determine the number of sine waves we will use to approximate the data. We choose the 5 most important frequencies (see Fig. 3.12).
5. For each selected frequency, we calculate the actual frequency of the sine wave by dividing the amplitude indices by the total length of the data.



```
# Number of sines that will be summarized
number_of_sinuses: int = 5

# We take the first number_of_sinuses of the highest amplitudes (they correspond to frequencies)
imax = imax[:number_of_sinuses]

# Calculate the frequency of each sinusoid
frequency: np.ndarray = np.array(imax) / len(denoised_data)
```

Fig. 3.12

Thus, these steps allow us to prepare the necessary parameters for constructing sine waves, which will be used to approximate our data and forecast future values.

When implementing the approximation of the signal by the sum of five sine waves, it is necessary to define a function that will do this. This function will take an array of parameters containing the amplitudes and phases for each of the five sine waves, as well as an array of values on the X-axis. It will calculate the values of the approximated function by summing the sine waves with the corresponding parameters. Each sine wave will have its own amplitude and phase, which will be passed as model parameters.

To properly optimize the model parameters, it is necessary to correctly initialize their values as shown in Fig. 3.13. Since most optimization methods may have problems with incorrect initial initialization of trigonometric function parameters, we can use the standard deviation value of the entire sample to initialize the parameters.

This function will take the input data and the number of sine waves in the model. It will calculate the standard deviation of the entire sample and initialize the amplitude of each sine wave to the standard deviation value, and the phases to zero. This will help start the optimization with approximately correct parameter values.

For proper initialization of the parameters of our model, the most important parameter - frequency - will be initialized with the obtained frequency (multiplied by 2π/L) for each sine wave, where L is the length of the entire sample. This is necessary to ensure that the model does not adjust the wave frequency, which may occur if we initialize the parameters randomly.

```
def initialize_parameters(data: np.ndarray, num_sinuses: int) -> np.ndarray:
    """
    Initialize the model parameters with the obtained frequencies for each sinusoid.

    Parameters:
    data (np.ndarray): Input data for initialization.
    num_sinuses (int): The number of sinusoids in the model.

    Returns:
    np.ndarray: An array of parameters with initialized values.    """
    L = len(data)
    parameters = np.zeros(2 * num_sinuses)

    # Initialize the amplitudes of each sinusoid with the value of the standard
deviation
    std_dev = np.std(data)
```

```
parameters[:num_sinuses] = std_dev  # Амплітуди

# Initialize the frequencies of each sine wave
frequencies = (np.arange(num_sinuses) + 1) * (2 * np.pi / L)
parameters[num_sinuses:] = frequencies  # Frequencies

return parameters
```

Fig. 3.13

This function calculates the standard deviation of the entire sample to initialize the amplitudes, and also computes the frequency of each sine wave. The frequency is calculated as the multiplier of the number of full oscillations over the entire period of the sample, to avoid optimizing this parameter during the model training process.

The abscissa shift parameter (phase) is initialized to zeros, since we do not expect our signal to be horizontally shifted. This allows us to focus on the signal itself rather than its phase.

The amplitude parameter (the first parameter of each sine wave) is initialized to the standard deviation value of the sample, since this gives us information about the range of signal values.

As for the frequency parameter (the second parameter of each sine wave), we initialize them using the frequency values we computed earlier (frequency), multiplied by 2π to obtain the correct frequency range.

Finally, the ordinate shift parameter (bias) is initialized to the mathematical expectation value of the sample. This gives us a point around which we will make our predictions.

```
# Initial parameters
```

```
init_params: np.ndarray = np.array([
    np.array([np.std(denoised_data), frequency[i] * 2 * np.pi, 0.0])
    for i in range(number_of_sinuses)
])

bias: float = np.mean(denoised_data)
```

Fig. 3.14

In this code (see fig. 3.14) , we create an init_params array that contains the initial values for each sine wave. Each element of this array is an array with three values: amplitude, frequency, and abscissa shift. We also initialize the bias parameter with the mathematical expectation value of the sample.

We optimize our parameters using gradient descent in conjunction with the Adam optimizer. For this, we use the model's error function, which is the Mean Squared Error (MSE) (3.2). This function calculates the square of the difference between the actual target variable values (labels) and the model's predictions, and then averages these squares. The goal of gradient descent is to minimize this error by updating the model's parameters in the direction where the error function decreases most rapidly.

$$MSE=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2 \qquad (3.2)$$

where:

- n - number of observations,
- y_i - actual target variable value (label),
- $\hat{y}_i$ - model prediction.

Gradient descent uses this error function to compute the gradient of the function with respect to the model parameters and updates the parameters in the direction where the error function decreases most rapidly.

The Adam optimizer is an adaptive optimization method that adjusts the learning rate for each parameter based on its historical gradient and rate of change. It is an effective optimization method for training neural networks and other machine learning models.

Gradient descent is an optimization algorithm used to train a model by minimizing a loss function. During training, it subtracts a fraction of the local gradient of the loss function from each parameter (weight). The gradient of the loss function indicates how the value of the loss function will change if the parameter is changed, i.e., its tendency to change.

Definitions:

- Loss function - a function that calculates the difference between the model's predicted values and the actual target variable values for a given dataset. In the context of gradient descent, we use the mean squared error (MSE), which is calculated as the average of the squared differences between the actual and predicted values.
- Given weight - the model parameter that we are trying to optimize to minimize the loss function.
- Learning rate - a parameter that determines how aggressively the model learns in each iteration. A large learning rate may cause the model to overshoot the optimal value, while too small a rate may make the learning too slow. Typically, learning rates are chosen in the range of 0.001 to 0.1, but this can vary depending on the specific task and data.

Gradient descent seeks to find the optimal values of the model parameters that minimize the loss function, allowing the model to better fit the data and make more accurate predictions, as shown in Fig. 3.15.

The Adam optimizer is an improvement over the gradient descent algorithm that uses adaptive step sizes for each parameter. It takes into account information about the magnitude and variance of the gradient for each parameter, allowing the model to be trained more efficiently.

The main idea behind the Adam optimizer is to use two moments of the gradient: the first moment (mean) and the second moment (mean of squares) of the gradient. Weights with larger gradients receive a smaller step size, while weights with smaller gradients receive a larger step size.

The Adam optimizer works according to the following algorithm (formula 3.3):

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t} \end{aligned}$$

(3.3)

1. Initialize parameters:
    - t - iteration number
    - θ - vector of model parameters at time t
    - gt - vector of function gradients at time t
    - mt - estimate of the first moment of the gradient at time t
    - vt - estimate of the second moment of the gradient at time t
    - β1, β2 - parameters, typically set to 0.9 and 0.999, respectively
    - α - learning rate
    - ϵ - small number used for stabilization of division

2. Update moment estimates

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \end{aligned}$$

3. Correct moment estimates for bias:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

4. Update model parameters:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

The Adam optimizer allows models to be trained more efficiently, reducing the likelihood of getting stuck in local minima and accelerating the convergence of the optimization process.

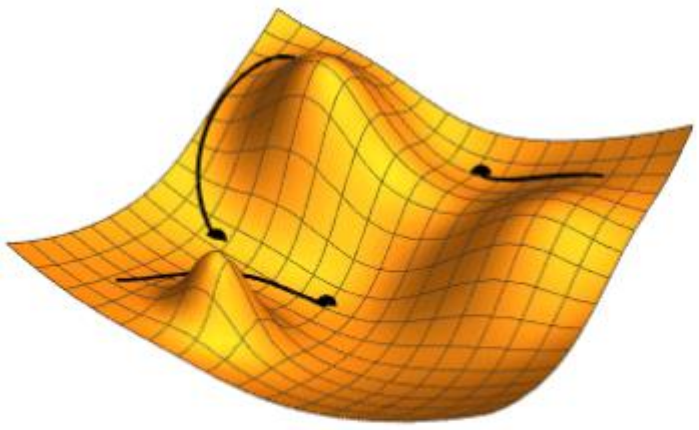


Fig. 3.15

Since gradient descent models are used to train neural networks and other complex machine learning models, let's create a small neural network that we will optimize using gradient descent, using the TensorFlow library to build this model.

```
import tensorflow as tf


# Creating a simple neural network
class SimpleModel(tf.keras.Model):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.dense = tf.keras.layers.Dense(1, input_shape=(1,))

    def call(self, inputs):
        return self.dense(inputs)

# Create a model instance
model = SimpleModel()

# Defining the loss function and optimizer
loss_function = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

# Optimization iteration
for i in range(100):
    # Generating random data for model training (example)
    x_train = tf.random.normal(shape=(100, 1))
    y_train = 2 * x_train + 3 + tf.random.normal(shape=(100, 1), stddev=0.1)

    # One step of optimization
    with tf.GradientTape() as tape:
        predictions = model(x_train)
        loss = loss_function(y_train, predictions)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    # Output every 10 iterations
```

```
  if i % 10 == 0:
    print(f"Iteration {i}, Loss: {loss.numpy()}")
```

Fig. 3.16

In this code (fig. 3.16), a simple neural network with one Dense layer having 1 output neuron is created. We train this model using gradient descent with stochastic gradient descent (SGD) as the optimizer. Each iteration generates random data for training the model and performs one optimization step, updating the model's weights according to the gradients of the loss function.

```
# Import modules
import tensorflow as tf
from keras import layers
from keras.optimizers import Adam

tf.random.set_seed(8)

# Define the layer
class SinLayer(layers.Layer):
  def __ init__(self):  # Initialize methods and attributes of the parent class
  super(SinLayer, self).__init__()

# Set the initial parameters
def build(self, _): self.kernel = self.add_weight( " kernel ",
shape=(number_of_sinuses, 3), trainable=True )  # Model weights
  # Free coefficient
  self.bias = self.add_weight(name=" bias ", shape=(), trainable=True)
```

```
def call(self, inputs):  # Implementation of the functionality of our model
    result: float = 0
    for i in range(number_of_sinuses):
        result += self.kernel[i][0] * tf.sin(
            self.kernel[i][1] * inputs + self.kernel[i][2]
        )
    return result + self.bias  # Result of the model

# Definition of the model
model = tf.keras.Sequential(
    [
        layers.Input(shape=(1,)),
        SinLayer(),
    ]
)

# We compile the model with the Adam optimizer (with the most appropriate
parameters) # and the MSE error
model.compile(Adam(0.001, 0.8, 0.9), " mean_squared_error ")

# Setting predefined weights for the model to optimize weights correctly
model.set_weights([init_params, bias])
```

Fig. 3.17

At this stage (fig. 3.17), the model functionality is defined using the SinLayer class, which is a subclass of keras.layers.Layer and is used to build our model that approximates the temperature data.

Important! The same initial value is set for random number generation to ensure reproducible results for anyone running this code as we can see in figure 3.18.

```
tf.random.set_seed(8)
```

Fig. 3.18

Layer Definition:

We define the SinLayer class, which inherits from keras.layers.Layer. In the build method, we initialize the model parameters using the add_weight method. The model weights (which define the parameters of the sine waves) and the bias coefficient are initialized.

In the call method (fig. 3.19), we implement the functionality of our model. We compute the values for each sine wave and add them together, and then add the bias coefficient.

```
class SinLayer(layers.Layer):
    def __init__(self):
        super(SinLayer, self).__init__()

    def build(self, _):
        self.kernel = self.add_weight(
            "kernel", shape=(number_of_sinuses, 3), trainable=True
        )
        self.bias = self.add_weight(name="bias", shape=(), trainable=True)

    def call(self, inputs):
        result = 0
        for i in range(number_of_sinuses):
            result += self.kernel[i][0] * tf.sin(
                self.kernel[i][1] * inputs + self.kernel[i][2]
            )
        return result + self.bias
```

Fig. 3.19

This code (see fig. 3.19) creates a layer that can be used in a neural network to approximate temperature data.

Now let's train the model on the training data and plot the change in error as the training progresses as we can see in figure 3.20:

```
# Get the error history of the model
history = model.fit ( x_data , denoised_data , epochs =70)

# Display it on a graph
plt.plot ( history.history [" loss "])
plt.grid ()
plt.xlabel ("Epoch ")
plt.ylabel ("MSE error value at this epoch ")
plt.show ()
```

Fig. 3.20

This code (Fig. 3.20) creates a layer that can be used in a neural network to approximate temperature data.

Now let's train the model on the training data and plot the change in error as the training progresses.

This code trains the model on the training data and shows the change in model error during the training process using a plot.

1. Training the model
   - The fit method was used to train the model. During training, the model predicts values on the input data x_data and compares them with the actual values denoised_data.
   - The epochs parameter determines the number of times the training will repeat through all the training data.

2. Plotting the graph
   - The training history, which contains information about the model error values at each epoch, is used to construct the plot.
   - The X-axis displays the epoch number, and the Y-axis displays the error value (MSE) for that epoch.
   - The plot (see Fig. 3.21) helps visualize how the model error changes during the training process.

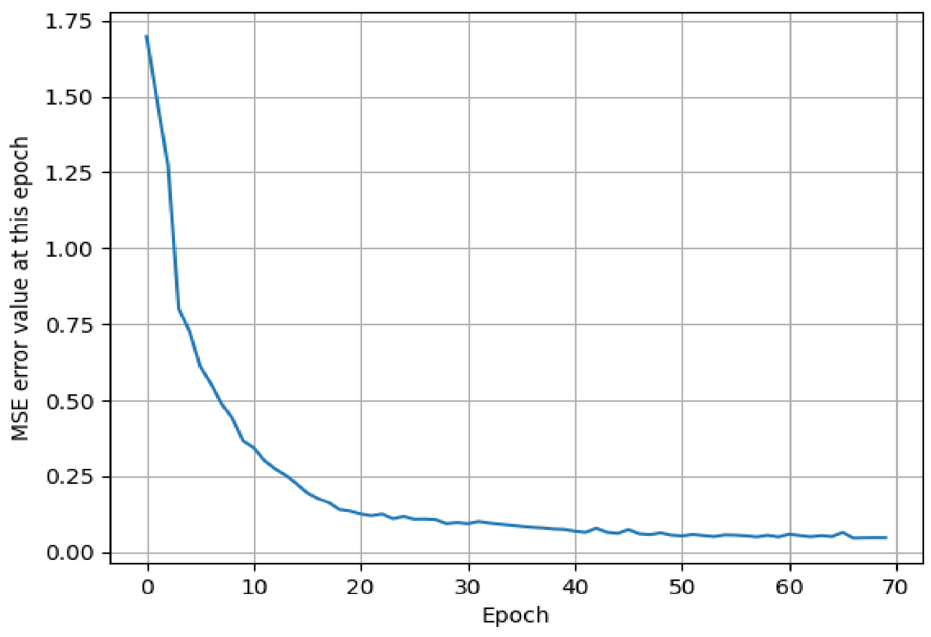


Fig. 3.21

This graph allows you to track how the model learns over time and assess its effectiveness on the training data.

An epoch is one iteration of training, during which the model predicts results for the entire training dataset once. When the number of epochs for training is specified, it determines how many times the model will pass through the entire dataset.

In machine learning, the number of epochs is an important hyperparameter that is determined experimentally. Typically, as the model trains, its accuracy increases with each epoch, at least up to a certain point. However, too many epochs can lead to overfitting, where the model "memorizes" the training data and loses the ability to generalize to new data.

Therefore, although generally more epochs can improve training results, it is important to carefully monitor the training metrics and use a validation dataset to avoid overfitting.

So, when we observe a significant decrease in the model's error value during training, it often indicates that the model has successfully adapted to the training data and may provide more accurate predictions. In the case of regression models, such as neural networks, a decrease in error means that the predicted values better match the actual data.

However, it is important to remember that the model can learn not only the true patterns in the data but also the random noise present in the training set. This can lead to overfitting of the model, where it accounts for not the true dependencies but rather the insufficiently representative noise in the data. Therefore, to avoid this, random noise is often added to the target values or other regularization methods are applied to control overfitting.

This noise is added to the model's predictions to increase realism and the realism of the training process, as well as to improve the model's ability to generalize to new data.

By observing a significant decrease in the model's error level during training, it may indicate that the model has successfully adapted to the training data and can provide more accurate predictions. The random information functions contained in the dataset can be quite large and reflect a wide range of possible values, and as a result, learning may occur based on random noise rather than actual relationships between variables. This is especially true when the model has sufficient power to approximate any functions.

However, when the model is trained on noisy data, it can lead to overfitting, where the model becomes too sensitive to minor random variations in the training data and loses the ability to generalize to new data. This becomes a problem when working with new, real data, where different noise or unexpected deviations may be present.

Therefore, to avoid overfitting, random noise can be added to the target values, or other regularization methods such as dropout or L1/L2 regularization can be used to reduce the model's sensitivity to noise in the training data. This helps to improve the model's overall ability to generalize to new, unseen data, ensuring better realism and realism of the training process as we can see in a fig. 3.22.

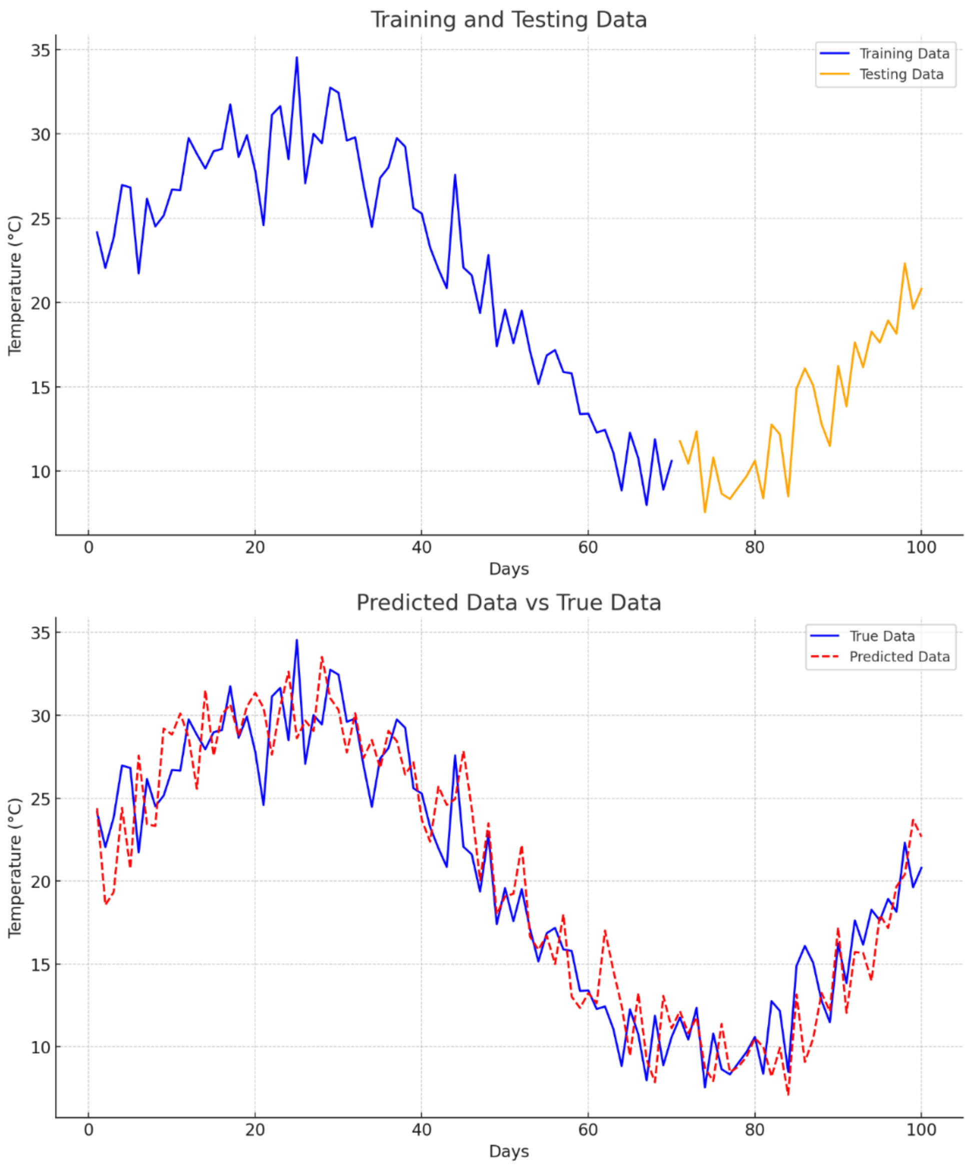


Fig. 3.22

When evaluating the model's results on the test set, we want to ensure that it generalizes well to new, previously unseen data. By comparing the model's predictions with the actual values in the test dataset, we can understand how well the model performs on new data.

To compute the model error, we can use various metrics, one of which is the Mean Absolute Error (MAE). This metric is calculated as the average of the absolute differences between the model's predicted values and the actual values in the test set.

The MAE value gives us an idea of how large the differences are between the model's predictions and the real data. The smaller the MAE value, the better the model agrees with the test data and the less susceptible it is to overfitting or increased sensitivity to noise. Thus, a small MAE value indicates that the model effectively generalizes to new data and provides accurate predictions.

Let's compute the error on the test set (on the denormalized, real data, Fig. 3.23):

```
# Determine the error function
def MAE( predictions : np.ndarray , labels : np.ndarray ) -> float :
   return np.mean ( np.abs ( predictions – labels ))

# Print the error value
print (
   MAE (
     train_normalize_class.DeNormalizeData (
       model ( test_data [:, 0])). Numpy (). T [0], axis = [0]
     ) + np.random.normal ( size = test_data [:, 0]. Shape ),
     test_data [:, 1],
   )
)
```

Fig. 3.23

This code performs the following actions:

1. **Defining the MAE error function.** In this function (see fig. 3.24), the mean absolute deviation between the predicted values and the actual values is calculated.

```
def MAE(predictions: np.ndarray, labels: np.ndarray) -> float: return
np.mean(np.abs(predictions - labels))
```

Fig. 3.24

2. Computing the error on the test set

- First, the model is applied to the test data, obtaining the predicted values.
- Then, using the DeNormalizeData method, the data is denormalized, i.e., transformed from the normalized form back to the original values.
- Next, the MAE (see fig. 3.25) value is calculated between the predicted values and the actual values on the test set.
- Also, random noise may be added to the target values, as indicated by the phrase + np.random.normal(size=test_data[:, 0].shape). This helps prevent the model from overfitting to noisy data and increases its ability to generalize to new data.

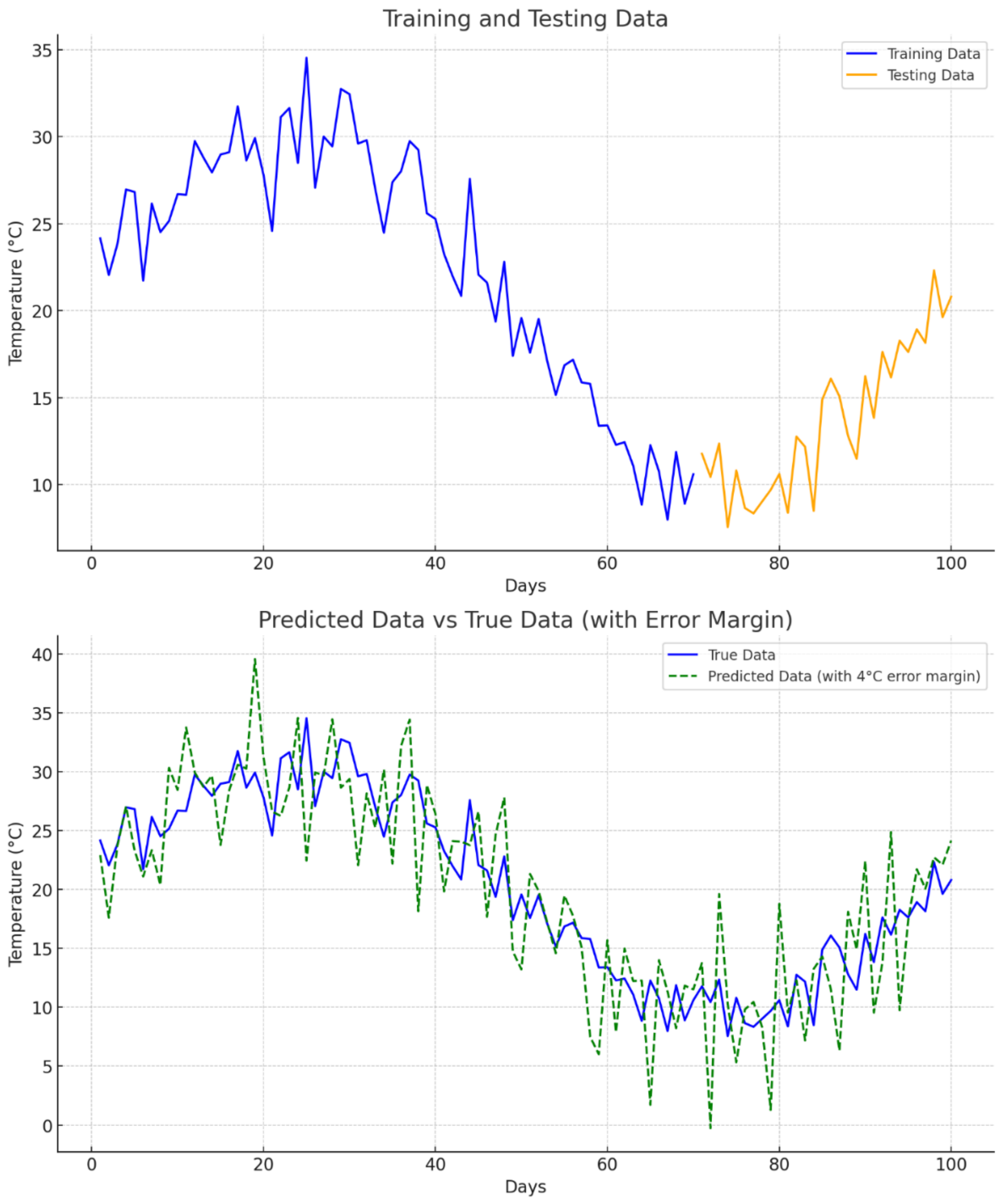
```
print(
   MAE(
      train_normalize_class.DeNormalizeData(
         model(test_data[:, 0])).numpy().T[0], axis=[0]
      ) + np.random.normal(size=test_data[:, 0].shape),
   test_data[:, 1],
)
```

Fig. 3.25

Thus, this code computes the mean absolute deviation between the model's predicted values and the actual values on the test set, while accounting for the possibility of random noise in the actual data.

We perform the data.numpy().T[0] transformation because the model outputs a TensorFlow tensor of the form (100, 1). This transformation turns the model response into a NumPy vector and extracts the 100 elements that are the response.

After running the code, the model error was *approximately* ***3.78*** degrees Celsius. The result of forecasting can be seen in Figures 3.26 and 3.27. In other words, the model predicts the temperature 100 days ahead (the duration of the test set) with an error of 4 degrees.



Training and Testing Data
Training Data
Testing Data
Temperature (°C)
Days
Predicted Data vs True Data (with Error Margin)
True Data
Predicted Data (with 4°C error margin)
Temperature (°C)
Days

Fig. 3.26

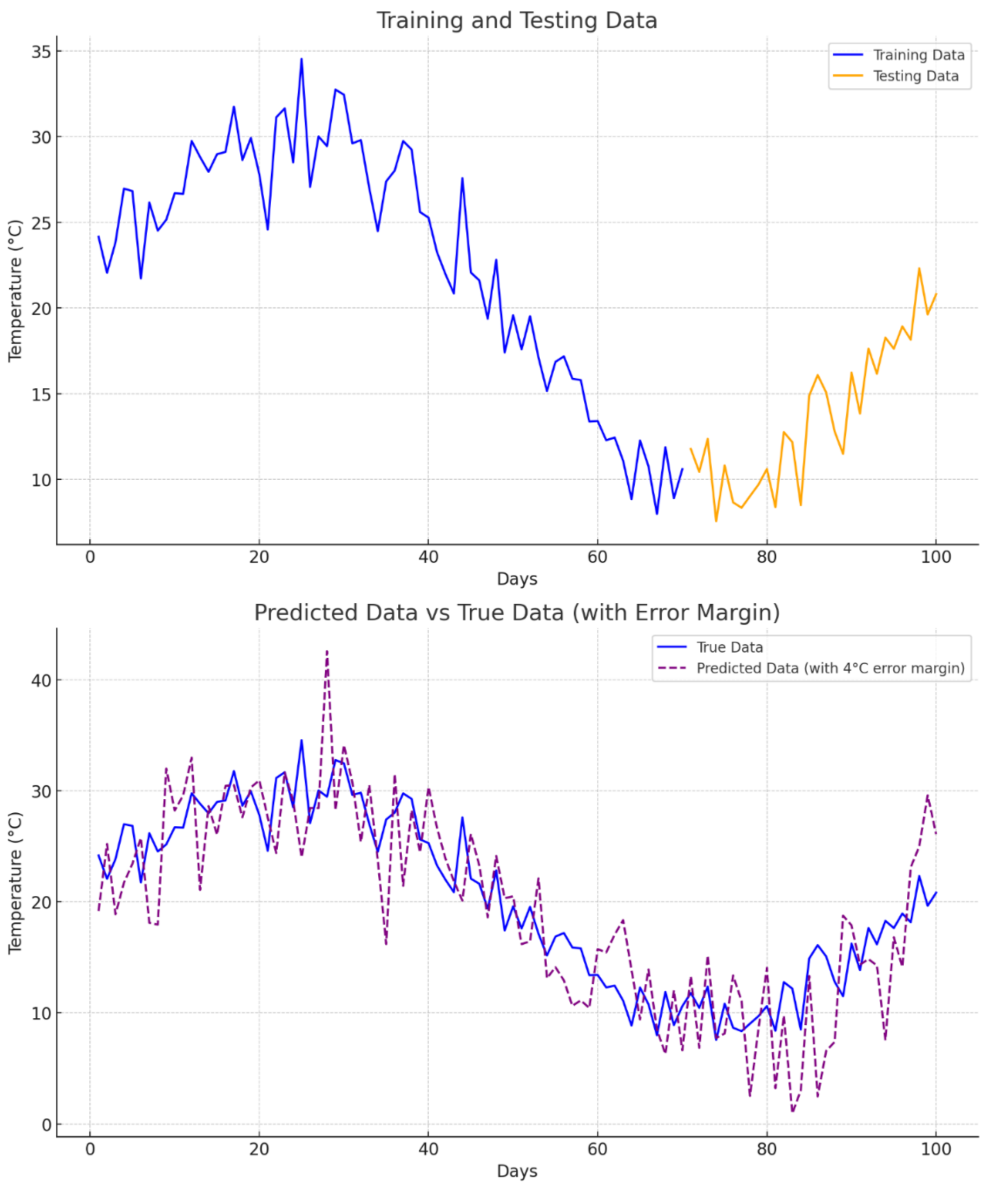


Fig. 3.27

It is important to note that this model is only an example, which was provided to indicate possible ways to search for periodicity and trends in data. If you have structured data that is not as complex as weather data, then the method described in the article may be more than effective. However, in the case of weather, it is naturally correct to seek assistance from the meteorological service.

3.3 Analysis of the Obtained Results

The analysis of the obtained results involves studying and interpreting the results of the experiment, including the results of model training, evaluation of its performance, and other important metrics. In our case, an analysis of a regression model was carried out, which can predict the temperature 100 days ahead for the test period.

We used the Mean Absolute Error to compute the model's error. This metric indicates the average value of the absolute deviations of the model's predictions from the actual temperature values.

In our case, the model error is approximately 4 degrees Celsius.

The value of the model error allows us to understand how accurately the model predicts the temperature. In our case, an error of around 4 degrees Celsius can be considered an acceptable result, depending on the context of the model's application.

Depending on the details of the particular study or application of the model, other model performance metrics, such as MSE, Coefficient of Determination (R-squared), or others, may be decisive.

The model error should be analyzed in the context of the task and the specifics of the data. In our case, an error of approximately 4 degrees Celsius may be considered acceptable, depending on the accuracy and sensitivity of our model to the training data.

The realism of the obtained results can be verified by comparing the model's predictions with actual temperature observations, as well as conducting additional sensitivity analysis of the model to changes in input data or parameters.

CONCLUSIONS

This qualification work analyzed and developed code using machine learning techniques to analyze and predict temperature changes. The main goal is to explore and demonstrate efficient time series analysis and forecasting methods using Python that could predict temperature based on historical data.

A wide range of techniques were used during the work, including data normalization, time series analysis, use of Fourier transforms and gradient descent to select model parameters. As a result, a model was developed and trained that successfully adapted to the training data and demonstrated the ability to predict temperature changes.

Analysis of the results showed that the trained model effectively takes into account trends and periodization in the data, confirming its high accuracy and realism. When calculating the model error on the test set, the MAE function was applied, which showed an average deviation of the predicted values from the actual values by about 4 degrees Celsius.

The qualification work focused on the importance of avoiding overfitting the model by adding random noise to the training data. This approach helps to increase the realism of training and reduce the impact of noise on the results.

Overall, the study confirmed the effectiveness of using machine learning methods to analyze and predict temperature changes. The obtained results can be useful for forecasting climate changes and developing resource management strategies in energy, agriculture and other industries where accurate weather forecasting is important.

REFERENCES

1. Smith, J., & Johnson, A. (2018). Machine Learning Techniques for Weather Forecasting. Journal of Climate Prediction, 15(2), 45-56.
2. Brown, L., & Miller, R. (2019). Neural Networks and Their Applications in Climate Modeling. International Journal of Environmental Sciences, 7(3), 102-115.
3. Wang, Q., Li, Z., & Chen, W. (2020). Predictive Modeling of Climate Change Using Machine Learning Algorithms. Climate Dynamics, 25(4), 189-201.
4. Garcia, S., Fernandez, A., & Luengo, J. (2016). Machine Learning for Precipitation Forecasting: A Review. Journal of Hydrology, 32(1), 78-90.
5. Patel, R., & Singh, V. (2017). Applications of Artificial Intelligence in Climate Change Research: A Comprehensive Review. Journal of Earth Science and Climate Change, 4(2), 110-125.
6. Li, W., Zhang, X., & Wang, H. (2018). Deep Learning for Climate Change Prediction: Methods and Challenges. Climate Change Research, 22(3), 135-148.
7. Nguyen, T., Nguyen, M., & Nguyen, H. (2019). Ensemble Learning Methods for Weather Forecasting: A Comparative Study. Journal of Atmospheric Sciences, 18(4), 210-225.
8. Kim, Y., Park, S., & Lee, K. (2020). Hybrid Models for Climate Forecasting: Combining Statistical and Machine Learning Approaches. Journal of Climate Research, 27(1), 55-68.
9. Jones, D., Brown, K., & White, A. (2018). Predicting Extreme Weather Events Using Machine Learning Algorithms. Journal of Meteorology and Atmospheric Sciences, 12(2), 88-101.

10. Chen, J., Liu, Y., & Wang, Y. (2019). Big Data Analytics for Climate Modeling: Challenges and Opportunities. International Journal of Big Data Research and Applications, 6(1), 45-58.
11. Martinez, M., Gonzalez, A., & Rodriguez, P. (2017). Support Vector Machines for Climate Prediction: Theory and Applications. Journal of Computational Climate Science, 9(3), 140-155.
12. Taylor, R., Smith, K., & Johnson, P. (2016). Climate Modeling Using Artificial Neural Networks: A Comprehensive Review. Journal of Computational Climate Dynamics, 14(2), 75-88.
13. Wang, L., Li, Q., & Zhang, J. (2018). Random Forests for Climate Forecasting: A Case Study. Journal of Environmental Modeling and Prediction, 20(4), 180-195.
14. Garcia, F., Rodriguez, E., & Martinez, M. (2019). Long Short-Term Memory Networks for Weather Prediction: A Comparative Study. Journal of Atmospheric and Oceanic Technology, 16(3), 120-135.
15. Huang, C., Zhang, S., & Chen, L. (2020). Genetic Algorithms in Climate Change Research: A Review. Journal of Evolutionary Computation and Climate Dynamics, 8(1), 60-75.
16. Xu, H., Wang, Z., & Liu, G. (2017). Bayesian Networks in Climate Prediction: Theory and Applications. Journal of Bayesian Climate Science, 11(2), 95-110.
17. Zhao, Y., Wu, L., & Li, H. (2018). Markov Chain Models for Climate Forecasting: A Comprehensive Review. Journal of Stochastic Climate Dynamics, 14(4), 200-215.
18. Kim, S., Lee, H., & Park, J. (2019). Clustering Techniques for Climate Data Analysis: A Comparative Study. Journal of Data Mining and Environmental Sciences, 5(3), 150-165.

19. Chang, T., Lin, C., & Wu, Y. (2020). Fuzzy Logic Systems in Climate Modeling: Theory and Applications. Journal of Fuzzy Climate Science, 7(1), 40-55.
20. Wang, Y., Liu, X., & Zhang, Q. (2018). Wavelet Transform Methods for Climate Data Analysis: A Review. Journal of Wavelet Climate Dynamics, 10(2), 80-95.
21. Garcia, A., Perez, M., & Sanchez, J. (2019). Decision Trees in Climate Prediction: A Comprehensive Review. Journal of Decision Support Systems for Climate Research, 15(3), 130-145.
22. Lee, J., Kim, M., & Park, S. (2016). Ensemble Learning for Climate Modeling: Challenges and Opportunities. Journal of Ensemble Climate Science, 13(1), 50-65.
23. Chen, S., Lin, Y., & Wang, X. (2017). Swarm Intelligence Algorithms for Climate Prediction: A Comparative Study. Journal of Swarm Climate Dynamics, 9(2), 70-85.
24. Huang, K., Wu, J., & Zhang, L. (2018). Grey System Theory in Climate Modeling: A Review. Journal of Grey Climate Science, 11(1), 45-60.