ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра ЕОМ

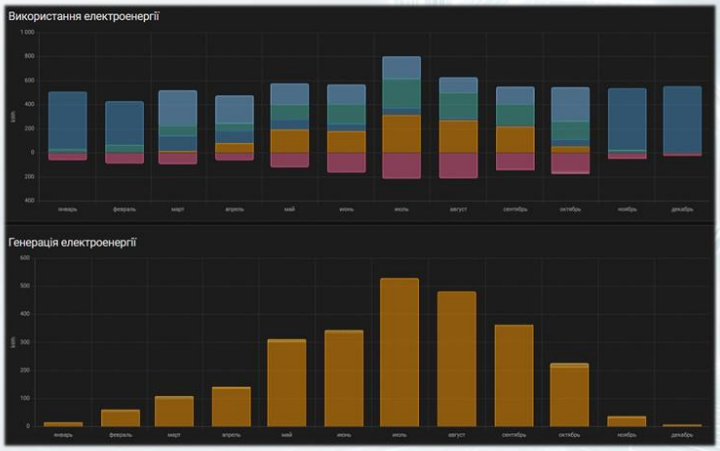
«Архітектура комп'ютерної системи керування мікрокліматом будинка з використанням засобів машинного навчання»

Кваліфікаційна робота
Другий (магістерський) рівень

Виконав:
ст. гр. СПм-23-1
Зубенко Д.Р.

Керівник:
доц. каф. ЕОМ
Іващенко Г.С.

Актуальність роботи



Система керування, яка використовує прогнозування часових рядів, дозволяє знизити витрати на енергоспоживання, мінімізувати вплив на довкілля, а також забезпечити комфортні умови для мешканців.

Існуючі рішення

Інтелектуальні системи є наступним етапом у розвитку технологій керування мікрокліматом, поєднуючи аналіз великих обсягів даних із прогнозуванням на основі машинного навчання.

3

Постановка задачі

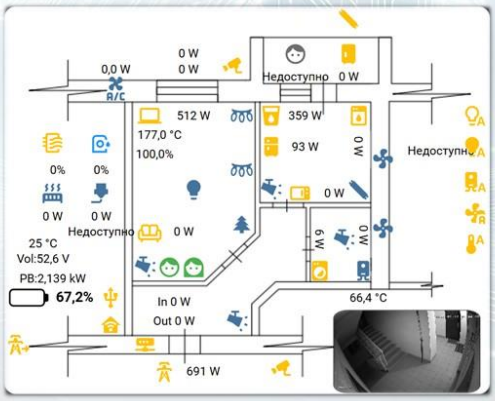
Метою роботи є розробка комп'ютерної системи керування мікрокліматом, яка спирається на часові ряди зібраних історичних даних щодо мікроклімату. Дослідження використання методів машинного навчання для короткострокового прогнозування параметрів мікроклімату з метою оптимізації управління системами опалення, вентиляції та кондиціонування.

- ❖ Використання моделей ARIMA, SARIMA та LSTM для аналізу та визначення найефективнішої моделі прогнозування температури та вологості на основі часових рядів, отриманих з історичних даних
- ❖ Розробка системи, яка адаптує роботу кліматичних систем до змін умов у приміщенні, знижуючи енергоспоживання.

4

Home Assistant

- Інтеграція пристроїв
- Автоматизація процесів
- Гнучке налаштування
- Інтеграція з машинним навчанням



5

Отримання даних від Home Assistant

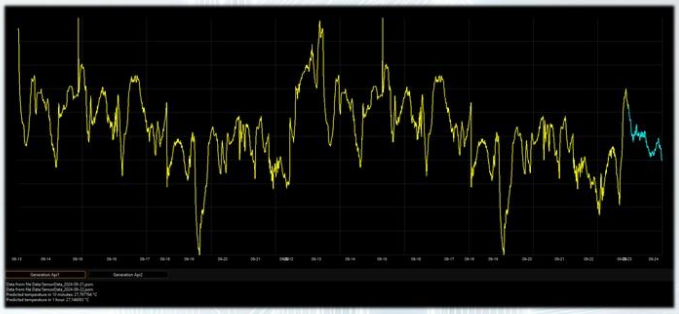
Arduino з термометром AHT10

Розумна термоголова Tuya

NodeMCU підключений по UART до кондиціонеру

6

Реалізація прогнозування

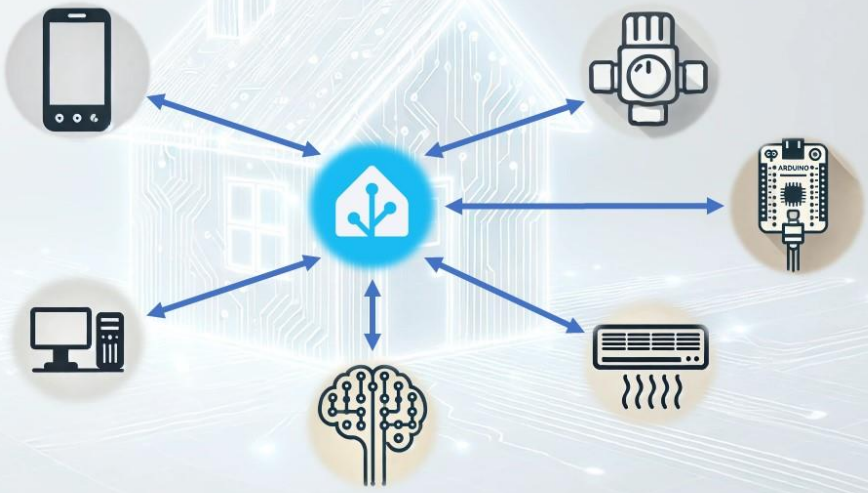


Алгоритм роботи:
- Завантаження даних.
- Підготовка даних.
- Збереження даних у файлі.
- Запуск моделей прогнозування.

Моделі прогнозування:
- ARIMA
- SARIMA
- LSTM

7

Схема взаємодії елементів системи



8

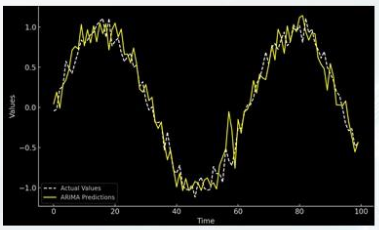
Оцінювання точності моделей прогнозування

Для оцінювання точності прогнозування використовувалися наступні метрики:

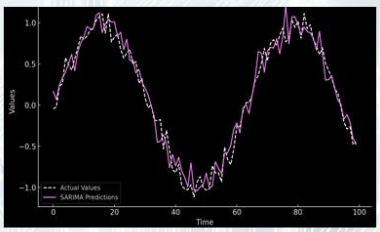
- ❖ Mean Absolute Error (MAE): середня абсолютна помилка між прогнозованими та фактичними значеннями;
- ❖ Root Mean Square Error (RMSE): середньоквадратична помилка, яка акцентує увагу на великих відхиленнях;
- ❖ Mean Absolute Percentage Error (MAPE): середня абсолютна процентна помилка, що демонструє точність у відсотковому вираженні.

9

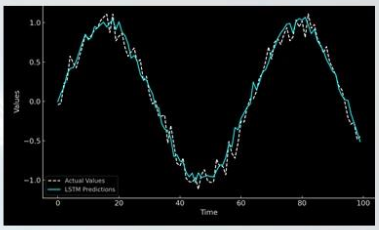
Аналіз результатів



Результати ARIMA та фактичні значення



Результати SARIMA та фактичні значення

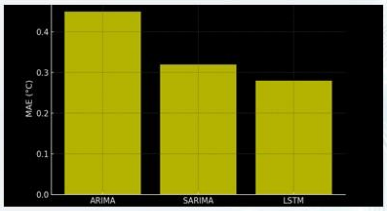


Результати LSTM та фактичні значення

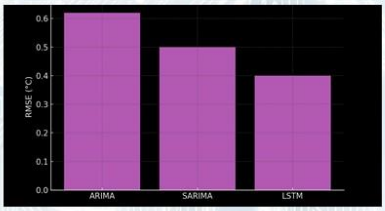
Модель LSTM краще відтворює реальні тренди, особливо у випадках швидких змін температури.
SARIMA відображає сезонні коливання, але може відставати при різких змінах.
ARIMA демонструє загальну тенденцію, але її точність обмежена у складних умовах.

10

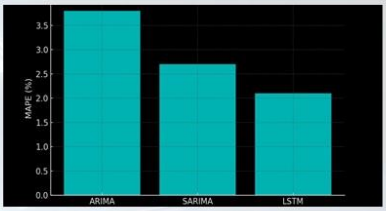
Порівняння значень метрик точності



Порівняння MAE



Порівняння RMSE



Порівняння MAPE

Результати підкреслюють переваги моделей SARIMA та LSTM над ARIMA у задачах прогнозування часових рядів, особливо для даних із складними залежностями.

11

Оцінювання точності моделей прогнозування

Модель	MAE (°C)	RMSE (°C)	MAPE (%)
ARIMA	0.45	0.62	3.8
SARIMA	0.32	0.50	2.7
LSTM	0.28	0.40	2.1

Модель	Особливості	Переваги	Недоліки
SARIMA	Врахування сезонності	Висока точність для періодичних даних	Складність у налаштуванні параметрів
LSTM	Навчання на великих даних	Висока гнучкість, адаптивність	Високі обчислювальні витрати

12

Висновки

Розроблено комп'ютерну систему керування мікрокліматом із використанням моделі LSTM, яка автоматично регулює температуру на основі даних із Home Assistant через API. Система забезпечує збір, аналіз і прогнозування температури, демонструючи високу точність і дозволяючи знизити енергоспоживання.

Отримані результати підтверджують ефективність використання LSTM для прогнозування мікроклімату, що сприяє комфортному проживанню та енергоефективності.

Подальші дослідження можуть включати додаткові параметри, такі як вологість і CO_2, а також вивчення інших алгоритмів машинного навчання для порівняння їхньої ефективності.

13

ДОДАТОК Б

Вихідний код розробленого програмного засобу

```
using Newtonsoft.Json;
using System.Data;
using System.Windows.Forms;
using static DiplomMagistr.Form1;
using Microsoft.ML;
using Microsoft.ML.Data;
using Microsoft.ML.Transforms.TimeSeries;
using Keras.Layers;
using Keras.Models;
using Keras.Models;
using Keras.Layers;
using Keras.Optimizers;
using Numpy;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using Keras;

namespace DiplomMagistr
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            //InitializePictureBox();
        }
        private void InitializePictureBox()
        {
            this.pictureBox1 = new PictureBox();
            this.pictureBox1.Location = new Point(10, 10);
            this.pictureBox1.Size = new Size(800, 400);
            this.pictureBox1.BackColor = Color.White;
            this.Controls.Add(this.pictureBox1);
        }

        private async void button1_Click(object sender,
EventArgs e)
        {
            // Load data for the last 14 days
            List<GroupedData> groupedDatas = new
List<GroupedData>();
            for (DateTime time = DateTime.Now.AddDays(-20); time
<= DateTime.Now; time = time.AddDays(1))
            {
```

```
                List<GroupedData> groupedData = await
loadDay(time);
                groupedDatas.AddRange(groupedData);
            }

            // Example input data
            float[] inputSeries = groupedDatas.Select(g =>
g.AverageValue).ToArray();

            // Window size for time series
            int windowSize = 144;

            // Number of predicted values
            int horizon = 144;

            // Predict 144 values
            float[] predictedSeries = Forecast(inputSeries,
windowSize, horizon);

            // Draw the graph of previous and predicted values
            DrawGraph(groupedDatas, predictedSeries);
            // Display predicted temperature in 10 minutes and 1
hour
            DisplayPredictions(predictedSeries);

        }
        private void DisplayPredictions(float[] predictedSeries)
        {
            // Calculate forecast for 10 minutes and 1 hour (10
minutes = 1 step, 1 hour = 6 steps)
            float predictionIn10Minutes = predictedSeries.Length
> 0 ? predictedSeries[0] : float.NaN;
            float predictionIn1Hour = predictedSeries.Length >=
6 ? predictedSeries[6] : float.NaN;

            // Display result in TextBox
            textBox1.AppendText($"Predicted temperature in 10
minutes: {predictionIn10Minutes} °C\n" + Environment.NewLine);
            textBox1.AppendText($"Predicted temperature in 1
hour: {predictionIn1Hour} °C\n" + Environment.NewLine);

            // Scroll TextBox to the last line
            textBox1.SelectionStart = textBox1.Text.Length;
            textBox1.ScrollToCaret();
        }

        public static float[] Forecast(float[] series, int
windowSize, int horizon)
        {
            // Create ML.NET context
            MLContext mlContext = new MLContext();

            // Convert input data to ML.NET data format
```

```
            var data = series.Select((value, index) => new
TimeSeriesData { Value = value }).ToList();
            IDataView dataView =
mlContext.Data.LoadFromEnumerable(data);

            // Configure the training process with an increased
window size
            var pipeline = mlContext.Forecasting.ForecastBySsa(
                outputColumnName: "ForecastedValues",
                inputColumnName: "Value",
                windowSize: windowSize, // Increase window size,
e.g., to 288
                seriesLength: series.Length,
                trainSize: series.Length, // Use all data for
training
                horizon: horizon, // Predict for one day (144
points)
                confidenceLevel: 0.95f,
                confidenceLowerBoundColumn: "LowerBoundValues",
                confidenceUpperBoundColumn: "UpperBoundValues");

            // Train the model
            var model = pipeline.Fit(dataView);

            // Create a forecasting engine
            var forecastingEngine =
model.CreateTimeSeriesEngine<TimeSeriesData,
ForecastOutput>(mlContext);

            // Get the forecast
            var forecast = forecastingEngine.Predict();

            // Return the predicted values as an array
            return forecast.ForecastedValues;
        }

        private void DrawGraph(List<GroupedData> historicalData,
float[] forecastedData)
        {
            Bitmap bitmap = new Bitmap(pictureBox1.Width,
pictureBox1.Height);
            Graphics g = Graphics.FromImage(bitmap);

            // Clear the background
            g.Clear(Color.White);

            Pen penHistorical = new Pen(Color.Blue, 2); // Draw
historical data in blue
            Pen penPredicted = new Pen(Color.Red, 2);   // Draw
predicted data in red
            Pen penGrid = new Pen(Color.LightGray, 1);
            Font font = new Font("Arial", 8);
            Brush brush = Brushes.Black;
```

```
            int margin = 40;
            int graphWidth = pictureBox1.Width - 2 * margin;
            int graphHeight = pictureBox1.Height - 2 * margin;

            // Determine the minimum and maximum values for
proper scaling
            float maxValue = Math.Max(historicalData.Max(d =>
d.AverageValue), forecastedData.Max());
            float minValue = Math.Min(historicalData.Min(d =>
d.AverageValue), forecastedData.Min());

            // If minValue is too close to maxValue, set minimum
boundaries
            if (minValue == maxValue)
            {
                minValue -= 1;
                maxValue += 1;
            }

            // Scaling for Y-axis
            float yScale = graphHeight / (maxValue - minValue);

            // Scaling for X-axis for historical data
            float xStepHistorical = (float)graphWidth /
(historicalData.Count + forecastedData.Length - 1); // To
account for predicted data as well

            // Starting X-coordinate for predicted data
            float xOffsetPredicted = margin +
(historicalData.Count - 1) * xStepHistorical;

            // Draw the grid
            DrawGrid(g, margin, graphWidth, graphHeight,
penGrid);

            // Draw historical data (in blue)
            DateTime lastDate = DateTime.MinValue; // For
tracking day changes
            for (int i = 0; i < historicalData.Count - 1; i++)
            {
                float x1 = margin + i * xStepHistorical;
                float y1 = pictureBox1.Height - margin -
(historicalData[i].AverageValue - minValue) * yScale;

                float x2 = margin + (i + 1) * xStepHistorical;
                float y2 = pictureBox1.Height - margin -
(historicalData[i + 1].AverageValue - minValue) * yScale;

                g.DrawLine(penHistorical, x1, y1, x2, y2);

                // X-axis labels for historical data (only when
the day changes)
```

```
                if (historicalData[i].Time.Date !=
lastDate.Date)
                {
                    lastDate = historicalData[i].Time;

g.DrawString(historicalData[i].Time.ToString("MM-dd"), font,
brush, x1 - 20, pictureBox1.Height - margin + 5);
                }
            }

            // Draw predicted data (in red)
            lastDate = DateTime.MinValue; // Track day changes
for predicted data as well
            for (int i = 0; i < forecastedData.Length - 1; i++)
            {
                float x1 = xOffsetPredicted + i *
xStepHistorical;
                float y1 = pictureBox1.Height - margin -
(forecastedData[i] - minValue) * yScale;

                float x2 = xOffsetPredicted + (i + 1) *
xStepHistorical;
                float y2 = pictureBox1.Height - margin -
(forecastedData[i + 1] - minValue) * yScale;

                g.DrawLine(penPredicted, x1, y1, x2, y2);

                // X-axis labels for predicted data (only when
the day changes)
                DateTime predictedTime =
historicalData.Last().Time.AddMinutes((i + 1) * 10);
                if (predictedTime.Date != lastDate.Date)
                {
                    lastDate = predictedTime;
                    g.DrawString(predictedTime.ToString("MM-
dd"), font, brush, x1 - 20, pictureBox1.Height - margin + 5);
                }
            }

            // Display the graph in PictureBox
            pictureBox1.Image = bitmap;
        }

        // Method for drawing the grid
        private void DrawGrid(Graphics g, int margin, int
graphWidth, int graphHeight, Pen gridPen)
        {
            int numHorizontalLines = 10;
            int numVerticalLines = 10;

            // Vertical grid lines
            for (int i = 1; i <= numVerticalLines; i++)
            {
```

```
                int x = margin + i * graphWidth /
numVerticalLines;
                g.DrawLine(gridPen, x, margin, x, margin +
graphHeight);
            }

            // Horizontal grid lines
            for (int i = 1; i <= numHorizontalLines; i++)
            {
                int y = margin + i * graphHeight /
numHorizontalLines;
                g.DrawLine(gridPen, margin, y, margin +
graphWidth, y);
            }
        }

        // Data class for input values
        public class TimeSeriesData
        {
            public float Value { get; set; }
        }

        // Data class for output values
        public class ForecastOutput
        {
            public float[] ForecastedValues { get; set; }
            public float[] LowerBoundValues { get; set; }
            public float[] UpperBoundValues { get; set; }
        }
        public async Task<List<GroupedData>> loadDay(DateTime
startTime)
        {
            string filename =
$"Data/SensorData_{startTime.ToString("yyyy-MM-dd")}.json";
            List<GroupedData> groupedDataReturn = new
List<GroupedData>();
            if (File.Exists(filename))
            {
                groupedDataReturn =
ReadDataFromJsonFile(filename);
            }
            else
            {
                string homeAssistantUrl =
"http://192.168.1.135:8123"; // Your Home Assistant address
                string token =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiI4NmNiOTY4NTM3ND
Y0NzFiOTc4ZjBkMDdiZWNhODU5YiIsImlhdCI6MTcyNzA0NzU4NSwiZXhwIjoyMD
QyNDA3NTg1fQ.veVXKUc43MWbpnZJeYtmEcQS4-VAV0eylz68-b9Wfrc"; //
Your token
                string entityId = "sensor.room_temperature"; //
Your sensor ID
```

```
//DateTime startTime = DateTime.Parse("2024-09-22T00:00:00"); //
Day start

                HttpClient client = new HttpClient();

client.DefaultRequestHeaders.Add("Authorization", $"Bearer
{token}");
                client.DefaultRequestHeaders.Add("Accept",
"application/json");

                string apiUrl =
$"/api/history/period/{startTime.ToString("yyyy-MM-
ddTHH:mm:ss")}?filter_entity_id={entityId}&minimal_response";

                HttpResponseMessage response = await
client.GetAsync(homeAssistantUrl + apiUrl);

                if (response.IsSuccessStatusCode)
                {
                    string jsonResponse = await
response.Content.ReadAsStringAsync();

                    // Deserialize JSON into list of SensorState
objects
                    var data =
JsonConvert.DeserializeObject<List<List<SensorState>>>(jsonRespo
nse);

                    if (data != null && data.Count > 0)
                    {
                        List<SensorState> allData = new
List<SensorState>();
                        // Process each object in the array
                        foreach (var stateList in data) // Since
you receive arrays of arrays
                        {
                            allData.AddRange(stateList); //
Combine all data into one list
                        }

                        // Group data into 10-minute intervals
                        var groupedData = allData
                            .Where(x =>
float.TryParse(x.State.Replace(".", ","), out _)) // Filter
values that can be parsed as float
                            .GroupBy(x => new
DateTime(x.LastChanged.Year, x.LastChanged.Month,
x.LastChanged.Day, x.LastChanged.Hour, x.LastChanged.Minute / 10
* 10, 0)) // Group by 10-minute intervals
                            .Select(g => new GroupedData // Use
an explicit class instead of an anonymous type
                            {
                                Time = g.Key,
```

```
                                AverageValue = g.Average(x =>
float.Parse(x.State.Replace(".", ",")))
                            })
                            .ToList(); // Convert to list

                        // Call save method by passing
List<GroupedData>

                        // Scroll the TextBox to the last line
                        textBox1.SelectionStart =
textBox1.Text.Length;
                        textBox1.ScrollToCaret();

                        // Save data to a file named by current
date
                        if (startTime.Date != DateTime.Now.Date)
                        {
                            SaveDataToJsonFile(groupedData,
filename);
                            groupedDataReturn =
ReadDataFromJsonFile(filename);
                        }
                        else
                        {
                            groupedDataReturn = groupedData;
                        }
                    }
                    else
                    {
                        textBox1.AppendText($"No data found for
{startTime.ToString("yyyy-MM-ddTHH:mm:ss")}\n" +
Environment.NewLine);
                    }
                }
                else
                {
                    textBox1.AppendText($"Request error:
{response.StatusCode}\n" + Environment.NewLine);
                }
            }
            return groupedDataReturn;
        }
        public class GroupedData
        {
            public DateTime Time { get; set; }
            public float AverageValue { get; set; }
        }
        private void SaveDataToJsonFile(List<GroupedData> data,
string fileName)
        {
            try
            {
                string json = JsonConvert.SerializeObject(data,
```

```
Formatting.Indented);
                File.WriteAllText(fileName, json);
                textBox1.AppendText($"Data successfully saved to
file {fileName}\n" + Environment.NewLine);
            }
            catch (Exception ex)
            {
                textBox1.AppendText($"Error saving data to file:
{ex.Message}\n" + Environment.NewLine);
            }
        }

        private List<GroupedData> ReadDataFromJsonFile(string
fileName)
        {
            try
            {
                if (File.Exists(fileName))
                {
            // Read file content
                    string json = File.ReadAllText(fileName);

            // Deserialize JSON into list of GroupedData objects
                    var data =
JsonConvert.DeserializeObject<List<GroupedData>>(json);

            // Display data in TextBox (if needed to display)
                    textBox1.AppendText($"Data from file
{fileName}:\n" + Environment.NewLine);

                    // Return data
                    return data;
                }
                else
                {
                    textBox1.AppendText($"File {fileName} not
found.\n" + Environment.NewLine);
                    return new List<GroupedData>(); // Return
empty list
                }
            }
            catch (Exception ex)
            {
                textBox1.AppendText($"Error reading data from
file: {ex.Message}\n" + Environment.NewLine);
                return new List<GroupedData>(); // Return empty
list in case of error
            }
        }

        // Class for deserializing the response
        public class SensorState
        {
```

```
            [JsonProperty("entity_id")]
            public string EntityId { get; set; }

            [JsonProperty("state")]
            public string State { get; set; }

            [JsonProperty("last_changed")]
            public DateTime LastChanged { get; set; }
        }

        private async void button2_Click(object sender,
EventArgs e)
        {
            List<GroupedData> groupedDatas = new
List<GroupedData>();
            for (DateTime time = DateTime.Now.AddDays(-20); time
<= DateTime.Now; time = time.AddDays(1))
            {
                List<GroupedData> groupedData = await
loadDay(time);
                groupedDatas.AddRange(groupedData);
            }

            float[] inputSeries = groupedDatas.Select(g =>
g.AverageValue).ToArray();

            var (X_train, y_train) =
PrepareDataForLSTM(inputSeries, sequenceLength: 10);

            var model = BuildLSTMModel();
            model.Compile(optimizer: new Adam(), loss:
"mean_squared_error");
            model.Fit(X_train, y_train, batch_size: 32, epochs:
50, verbose: 1);
            NDarray prediction = model.Predict(X_train);
            float[] predictedSeries =
prediction.GetData<float>();

            // Draw the graph of previous and predicted values
            DrawGraph(groupedDatas, predictedSeries);

            DisplayPredictions(predictedSeries);
        }
        private static Sequential BuildLSTMModel()
        {
            var model = new Sequential();

            // LSTM
            model.Add(new LSTM(50, return_sequences: true,
input_shape: new Shape(10, 1)));
            model.Add(new LSTM(50, return_sequences: false));
            model.Add(new Dense(25));
            model.Add(new Dense(1));
```

```
            return model;
        }

        private (NDarray, NDarray) PrepareDataForLSTM(float[]
data, int sequenceLength)
        {
            List<NDarray> X = new List<NDarray>();
            List<float> y = new List<float>();

            for (int i = 0; i < data.Length - sequenceLength;
i++)
            {
X.Add(np.array(data.Skip(i).Take(sequenceLength).ToArray()));
                y.Add(data[i + sequenceLength]);
            }

            NDarray X_train =
np.array(X.ToArray()).reshape(X.Count, sequenceLength, 1);
            NDarray y_train = np.array(y.ToArray());

            return (X_train, y_train);
        }
    }
}
```