

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**З В І Т**

**Лабораторна робота №7  
з дисципліни  
«Комп'ютерні системи  
штучного інтелекту»**

Виконавець:

студент групи КІ-22м

Косей М.П.

Керівник:

викладач

Саяпін В.Г.

2023

## Лабораторна робота №7

**Тема:** Прогнозування поведінки часових рядів із застосуванням нейромережевих структур

**Мета:** Одержані уміння побудови авторегресійних прогнозуючих моделей та практичні навички їх застосування

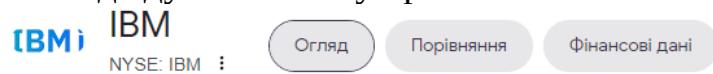
### ХІД РОБОТИ

#### 1) Ознайомитись з теоретичними відомостями до лабораторної роботи

Часовий ряд представляє собою послідовність вимірів змінної, взятих у рівномірні проміжки часу.

Приклади часових рядів:

- щоденні курси валют на міжнародному валютному ринку протягом десятиліть;
- місячна кількість проданих автомобілів в конкретному регіоні;
- щоденні температури повітря в місті протягом року;
- щомісячна кількість виконаних авіарейсів компанією протягом останніх п'яти років;
- щогодинні рівні загального обсягу продажів у роздрібних магазинах на протязі десятиліть;
- щоквартальні звіти прибутків підприємства за останні 20 років;
- щоденні витрати на газ та електрику для домогосподарств у певному місті за останні 5 років;
- місячна кількість нових користувачів певної соціальної мережі за останні 3 роки;
- щомісячна кількість запитів на конкретний товар в інтернет-магазині за останні 2 роки;
- щоденні відвідування сайту протягом останніх 6 міс.



Підсумок даних ринку > IBM

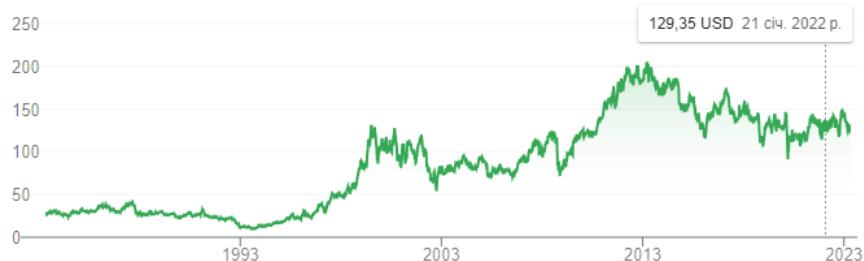
126,41 USD

+ Підписатися

+98,47 (352,43%) ↑ увесь час

Закрито: 28 квіт., 19:59 GMT-4 • Застереження  
Після торгів 126,40 -0,010 (0,0079%)

1 дн. | 5 д. | 1 м. | 6 м. | ВПР | 1 р. | 5 р. | Макс.



**Рис. 2** Приклад часового ряду – ціна акцій компанії IBM

Головною особливістю часових рядів є те, що в класичних задачах аналізу

і прогнозування передбачається незалежність спостережень, а при прогнозуванні часових рядів, навпаки, ми сподіваємося, що значення ряду в минулому містять інформацію про його поведінку в майбутньому.

Прогнозування часових рядів полягає у побудові моделі для передбачення майбутніх подій на основі відомих минулих подій.

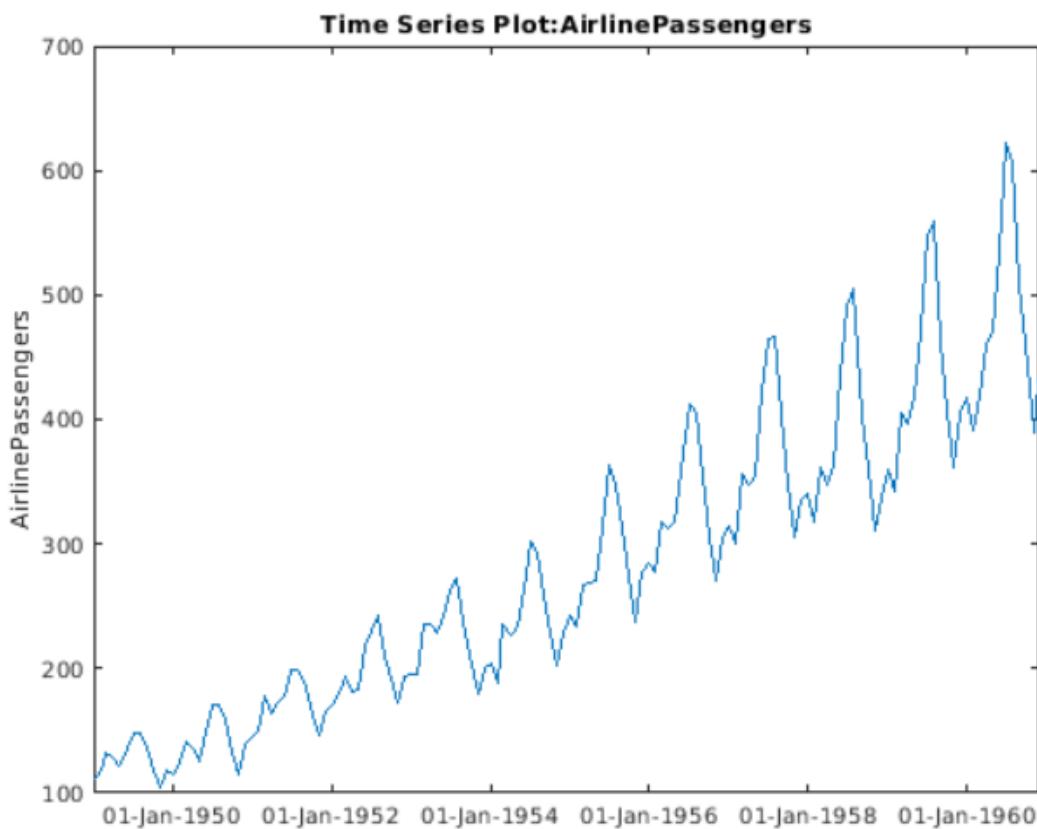
Часові ряди можуть бути характеризовані трендом, циклічністю, сезонністю та нерегулярністю.

**Тренд(trend)** - це довгостроковий зміщений рух у певному напрямку, що спостерігається в часовому ряді, зазвичай пов'язаний з економічними чи соціальними змінами. Наприклад, збільшення кількості населення, зростання економіки або зменшення середнього віку населення.

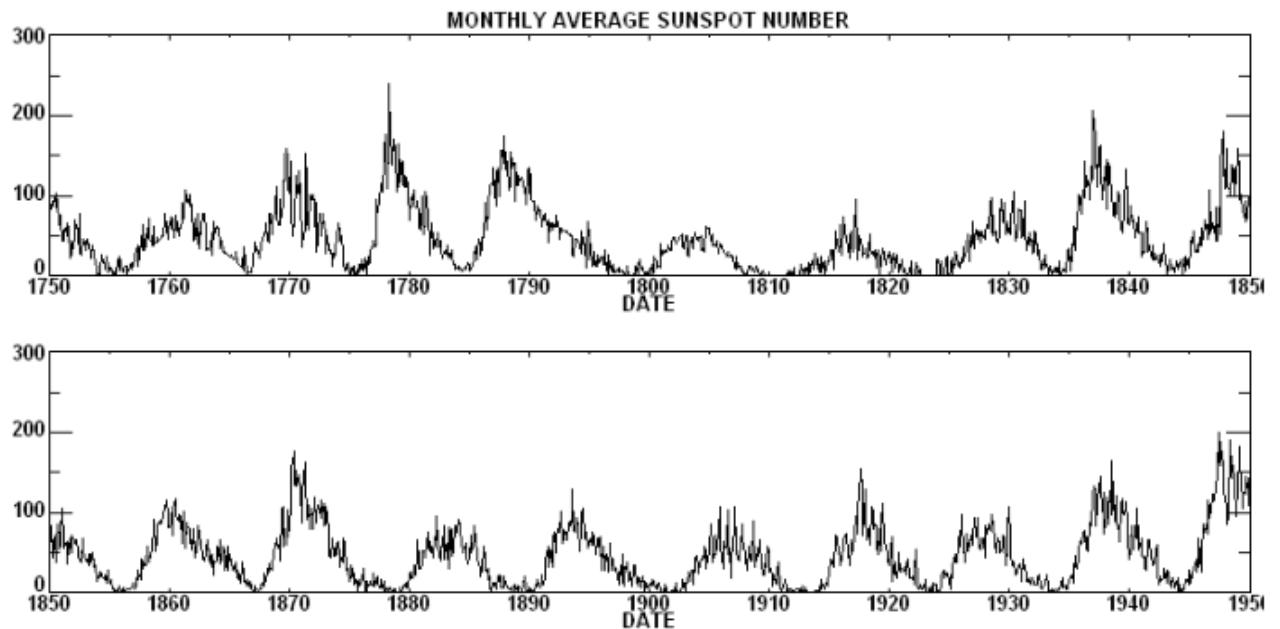
**Циклічність** - це зміна часового ряду, що повторюється через певний період часу. Ці цикли можуть мати довжину від декількох місяців до декількох років. Циклічність може бути пов'язана з економічними циклами, сезонними впливами або іншими факторами.

**Сезонність(seasonality)** - це зміна часового ряду, що повторюється через певний період часу, зазвичай річний, квартальний, місячний або тижневий цикл. Сезонність може бути пов'язана з погодними умовами, святами або іншими регулярними подіями.

**Нерегулярність(residual)** - це випадкові зміни, що не повторюються в часовому ряді, які можуть бути спричинені непередбачуваними подіями, такими як природні катаklізми, зміни в політиці чи в економіці.



**Рис. 2** Тренд, річна сезонність на прикладі кількості пасажирів авіакомпанії



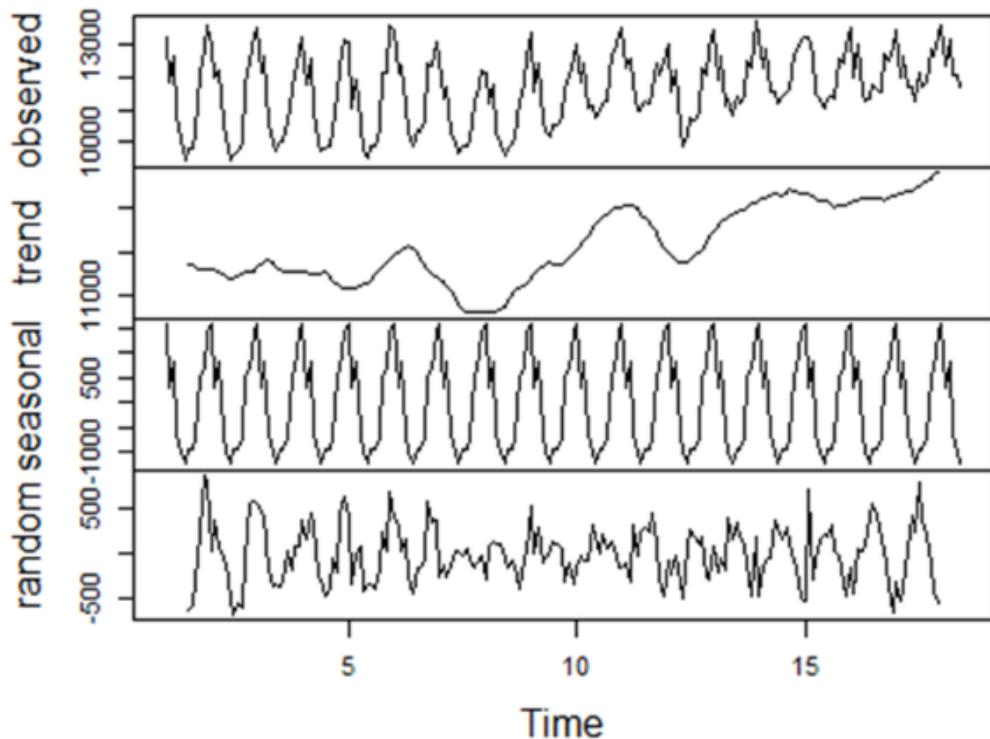
**Рис. 3** Циклічність на прикладі кількості сонячних плям на Сонці.

Часовий ряд - це комбінація його складових: тренду, сезонності та залишкових компонентів. Ця комбінація може мати адитивну або мультиплікативну форму.

У адитивній моделі ці компоненти додаються лінійно. Така модель підходить для часових рядів зі сталою або мало зростаючою залежністю сезонної складової від часу.

$$Y(t) = \text{trend} + \text{seasonality} + \text{residual}$$

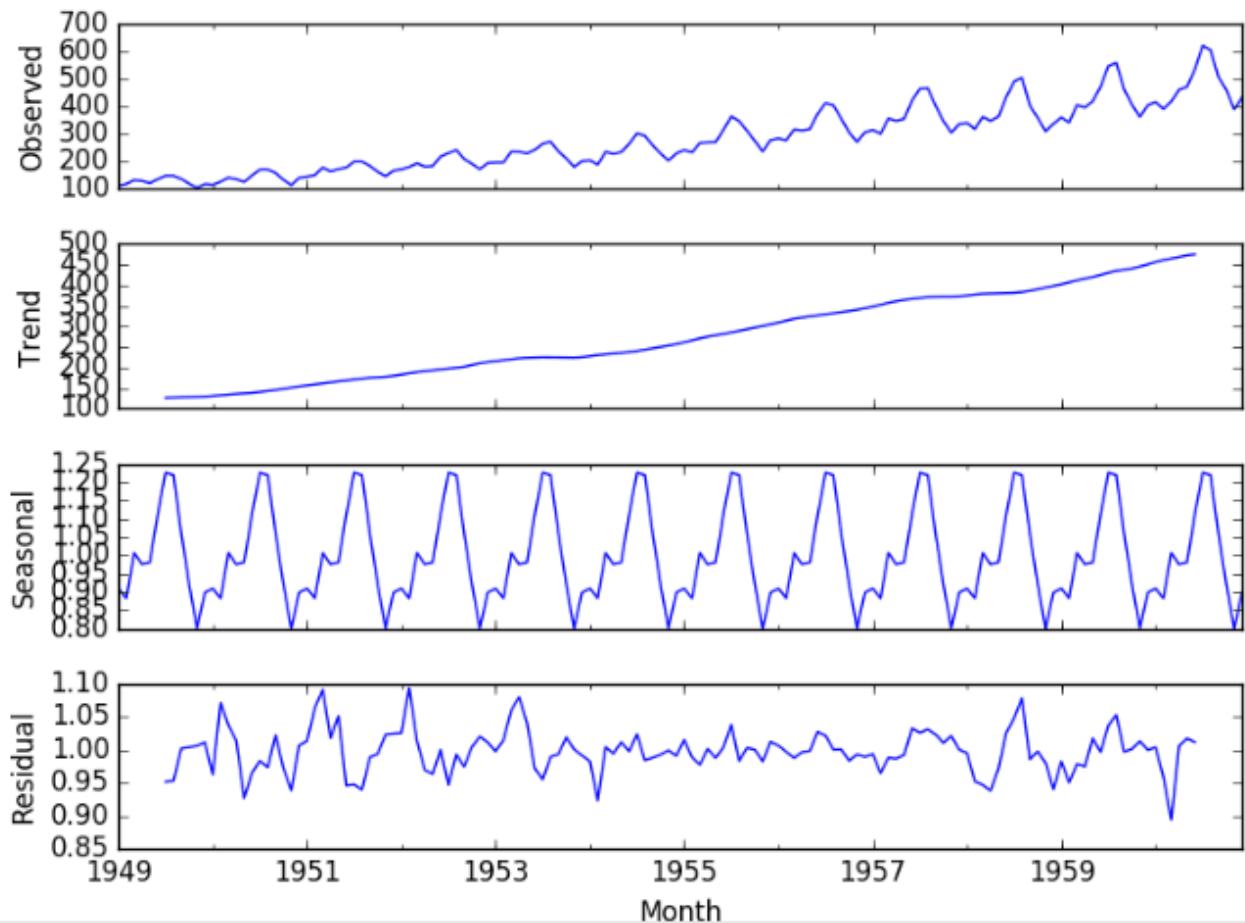
### Decomposition of additive time series



**Рис. 4** Приклад розкладення ряду в адитивній моделі .

В мультиплікативній моделі ці компоненти перемножуються між собою. Така модель підходить для часових рядів, де вплив сезонної складової збільшується з часом.

$$Y(t) = \text{trend} * \text{seasonality} * \text{residual}$$



**Рис. 5** Приклад розкладення ряду в мультиплікативній моделі .

Існують різні типи моделей, такі як авторегресійні моделі **AR (AutoRegressive Model)**, моделі з ковзним середнім **MA (Moving Average Model)**, моделі **ARMA (AutoRegressive Moving Average Model)**, **ARIMA (AutoRegressive Integrated Moving Average Model)**, **SARIMA(Seasonal AutoRegressive Integrated Moving Average Model)** та інші.

Кожен з цих типів моделей має свої особливості та призначення.

Наприклад, **AR**-модель використовується для моделювання часового ряду з авторегресійним ефектом, тобто коли значення ряду залежать від його попередніх значень.

**MA**-модель використовується для моделювання ряду з ковзним середнім ефектом, тобто коли значення ряду залежать від середнього значення попередніх значень.

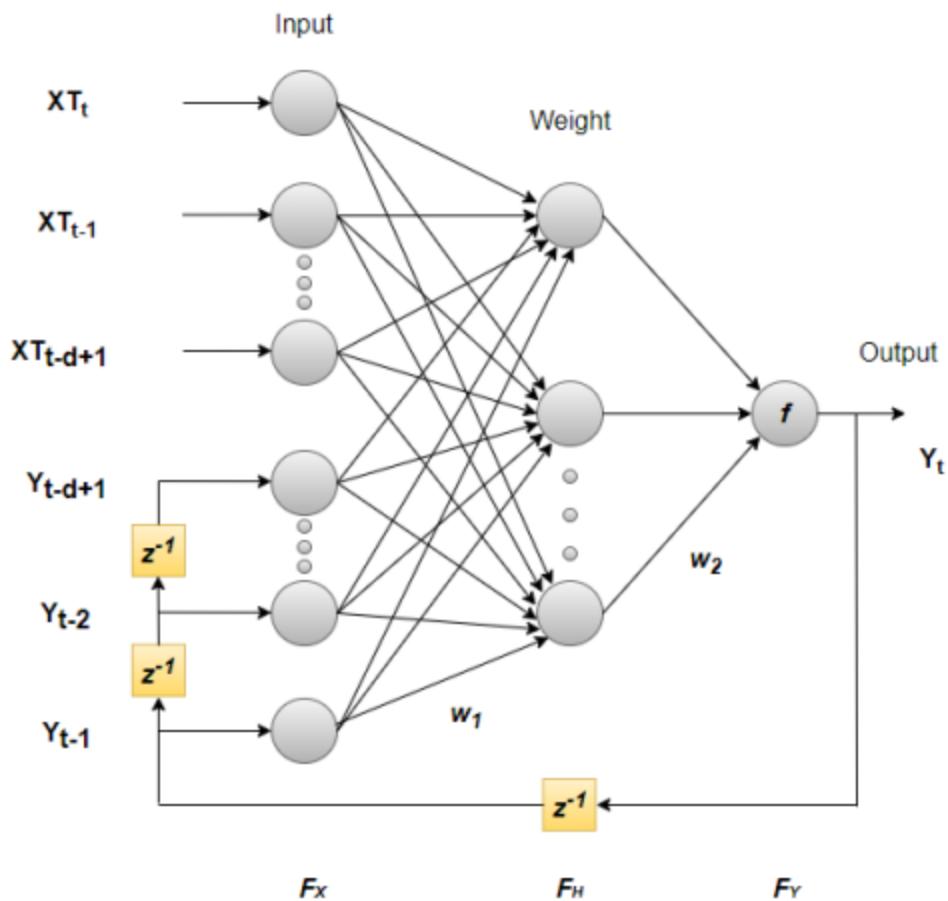
**ARIMA**-модель є розширеною версією **AR**- та **MA**-моделей, яка дозволяє моделювати часові ряди з трендом та сезонністю.

**SARIMA**-модель, у свою чергу, дозволяє моделювати ряди зі сезонністю та нерегулярною компонентою.

Нейромережі також можуть бути використані для моделювання та прогнозування часових рядів. Основна перевага нейромереж у порівнянні з традиційними методами полягає у їх здатності автоматично виявляти складні залежності в даних та адаптуватись до змін у часовому ряді.

Нейромережі типу **TLRN** (Time-Lagged Recurrent Network) та модель **NNARX** (Nonlinear Nonparametric AutoRegressive model with eXogenous inputs) є підтипом рекурентних нейронних мереж (RNN), які зазвичай використовуються для моделювання часових рядів, зокрема використовуються для прогнозування наступного елемента часового ряду на основі попередніх елементів. **TLRN** дозволяє враховувати затримки в часі у вхідних даних та зберігати попередні стани мережі для кращого прогнозування майбутніх значень.

На відміну від штучної нейронної мережі з прямим поширенням, де дані обробляються в одному напрямку від вхідного до вихідного шару без зворотного зв'язку, **RNN** зберігає інформацію про попередні стани даних за допомогою зворотних зв'язків, що дозволяє моделі зберігати попередні стани даних та використовувати їх для подальшої обробки даних.



**Рис. 6 Архітектура трьохшарової моделі мережі NARX.**

Пізніше були запропоновані рекурентні нейромережі типу **LSTM** (Long Short-Term Memory) та **GRU** (Gated Recurrent Unit).

Архітектура **LSTM** була спеціально розроблена для вирішення проблеми зникнення градієнту (**vanishing gradient problem**), яка може виникати під час алгоритму зворотного поширення помилок в традиційних рекурентних

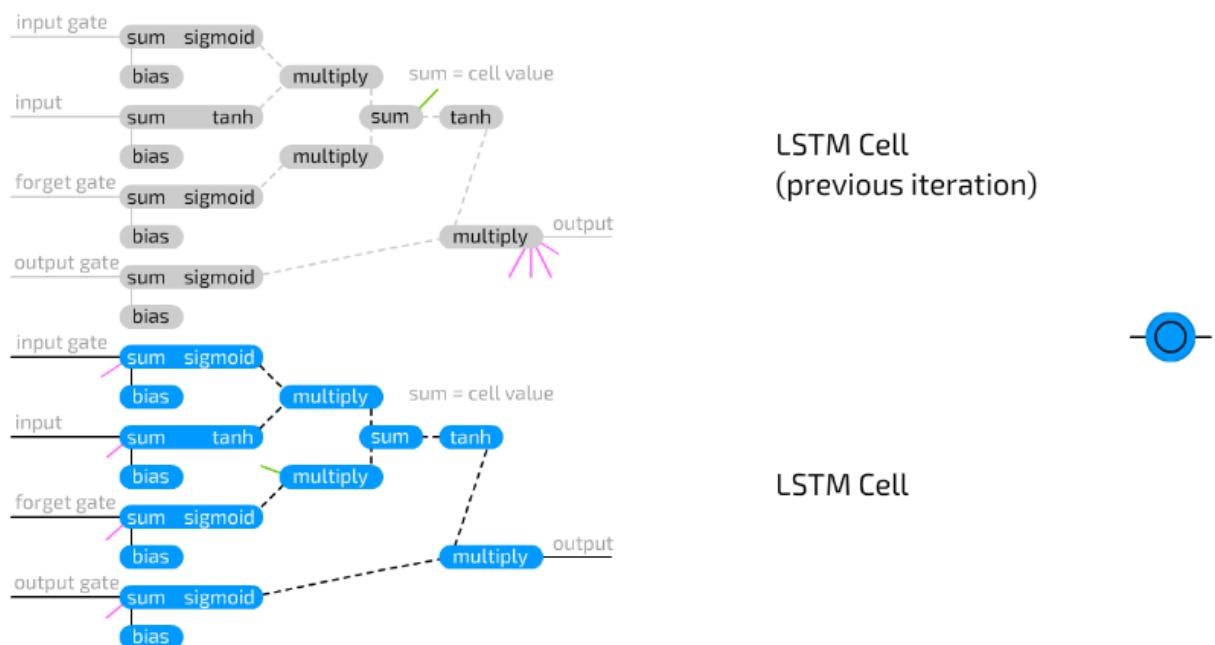
нейронних мережах.

Проблема зникнення градієнту виникає, коли сигнал градієнту, який використовується для оновлення ваг ів під час зворотного поширення помилок (**backpropagation**), стає дуже малим, що призводить до повільного навчання нейромережі і навіть до відсутності навчання.

Ця проблемма вирішується шляхом введення механізму гейтів, що дозволяє мережі вибірково зберігати або забувати інформацію протягом часу.

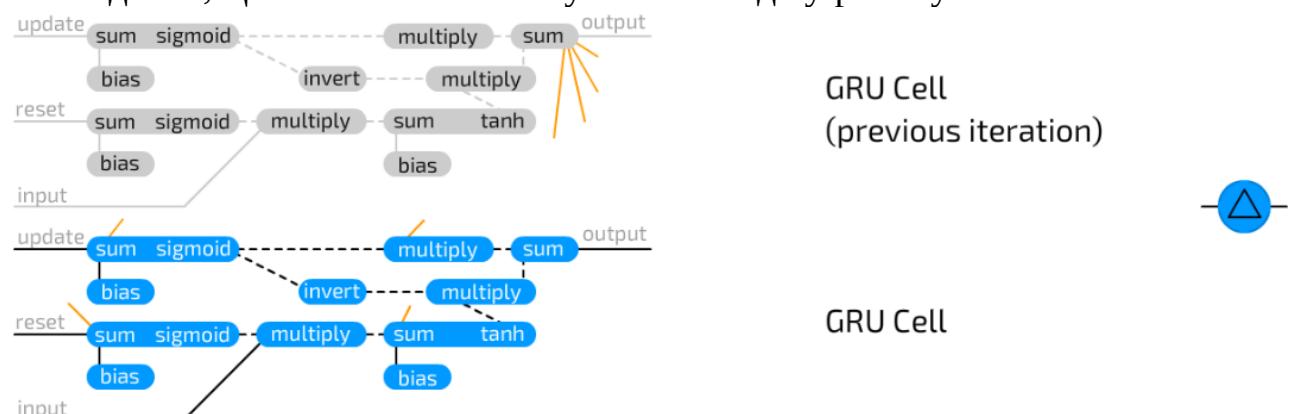
Архітектура включає три гейти: вхідний, вихідний та забування.

Гейтингові блоки можуть читися відкриватися або закриватися на основі вхідних даних та попереднього стану пам'яті, дозволяючи мережі вибірково зберігати або відкидати інформацію протягом часу.



**Рис. 7 Архітектура LTSM Cell.**

**GRU** - є спрощеною версією **LSTM**, з меншою кількістю гейтів: оновлення та скидання, що зазвичай забезпечує їхню швидшу роботу.



**Рис. 8 Архітектура GRU Cell.**

## 2) Зробити передбачення температури повітря

Прогнозування погоди - це процес прогнозування метеорологічних умов на певний час в майбутньому, зазвичай на декілька годин, днів або навіть тижнів. Це важливий процес, який дозволяє людям планувати свої дії, зокрема планувати подорожі, вирощувати рослини, займатися спортом, приймати рішення щодо безпеки на дорозі, прикладати зусилля для запобігання природним катастрофам, таким як урагани, торнадо, повені тощо.

Прогнозування погоди є складним завданням, оскільки атмосфера є дуже складною системою з багатьма факторами, що взаємодіють між собою.

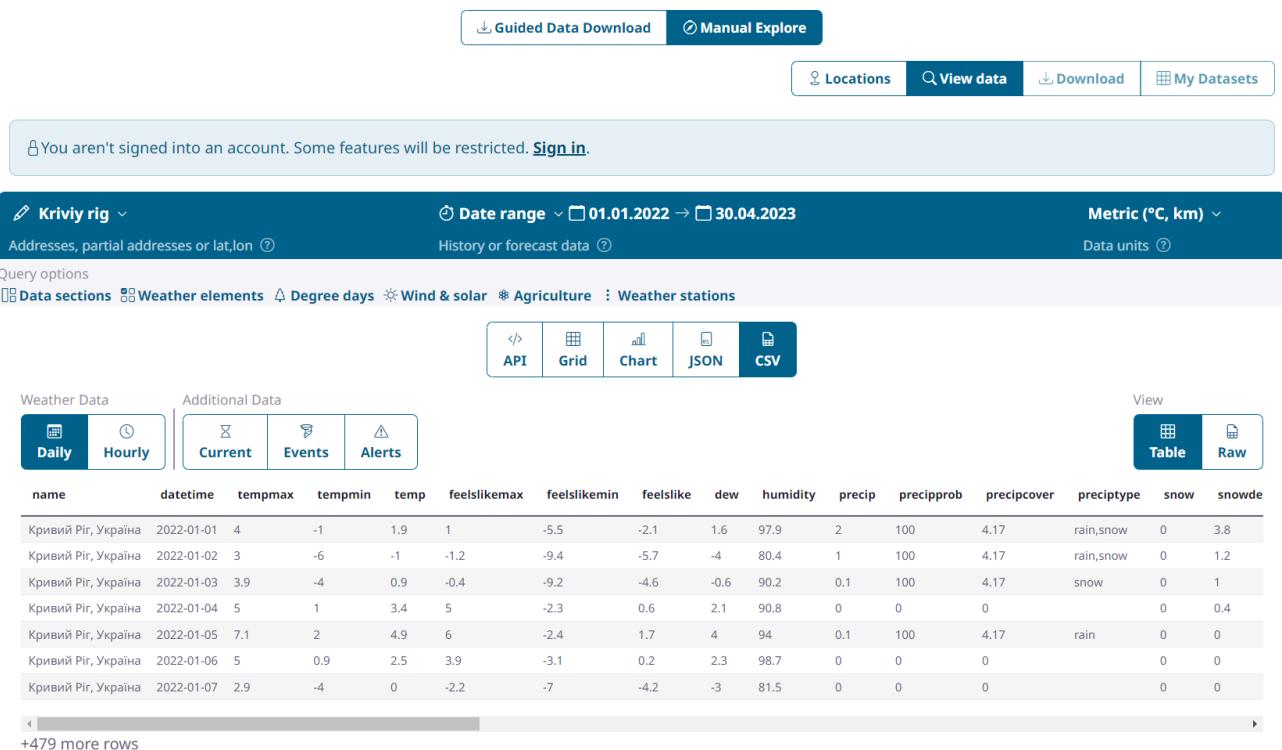
Також погода є хаотичним явищем природи, що означає, що навіть дуже малі зміни в початкових умовах можуть привести до значних різниць у погодних умовах в майбутньому. Це називається "ефектом метелика", який підкреслює важливість точності вимірювання та спостереження метеорологічних даних для отримання більш точних прогнозів погоди.

Згідно з інформацією зі статті розташованої на сайті [BBC](#) завдяки технічному прогресу, період часу, на який вдається більш-менш точно передбачити погоду, збільшувався на день в середньому кожні 10 років, прогноз погоди на найближчі п'ять днів зараз так само надійним, як і прогноз на завтра 40 років тому.

Спочатку отримаємо історичні данні про погоду з сайту <https://www.visualcrossing.com/> у форматі CSV за період: з 1 вересня 2020 року – по 30 квітня 2023 року.

### Weather Query Builder

 Legacy version



The screenshot shows the Weather Query Builder interface. At the top, there are buttons for 'Guided Data Download' and 'Manual Explore'. Below that are buttons for 'Locations', 'View data', 'Download', and 'My Datasets'. A message box says 'You aren't signed into an account. Some features will be restricted.' with a 'Sign in' link. The main search area has 'Kriviy rig' selected as the location, a date range from '01.01.2022' to '30.04.2023', and 'Metric (\*C, km)' selected. There are also 'Data units' and 'Query options' dropdowns. Below the search area are buttons for 'API', 'Grid', 'Chart', 'JSON', and 'CSV' (which is highlighted). The bottom section shows a table of weather data for Kriviy Rig, Ukraine, from January 1 to April 30, 2023. The table includes columns for name, datetime, tempmax, tempmin, temp, feelslikemax, feelslikemin, feelslike, dew, humidity, precip, precipprob, precipcover, preciptype, snow, and snowde. The data shows various weather conditions like rain, snow, and rain/snow mix over the period.

name	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	precip	precipprob	precipcover	preciptype	snow	snowde
Кривий Ріг, Україна	2022-01-01	4	-1	1.9	1	-5.5	-2.1	1.6	97.9	2	100	4.17	rain,snow	0	3.8
Кривий Ріг, Україна	2022-01-02	3	-6	-1	-1.2	-9.4	-5.7	-4	80.4	1	100	4.17	rain,snow	0	1.2
Кривий Ріг, Україна	2022-01-03	3.9	-4	0.9	-0.4	-9.2	-4.6	-0.6	90.2	0.1	100	4.17	snow	0	1
Кривий Ріг, Україна	2022-01-04	5	1	3.4	5	-2.3	0.6	2.1	90.8	0	0	0		0	0.4
Кривий Ріг, Україна	2022-01-05	7.1	2	4.9	6	-2.4	1.7	4	94	0.1	100	4.17	rain	0	0
Кривий Ріг, Україна	2022-01-06	5	0.9	2.5	3.9	-3.1	0.2	2.3	98.7	0	0	0		0	0
Кривий Ріг, Україна	2022-01-07	2.9	-4	0	-2.2	-7	-4.2	-3	81.5	0	0	0		0	0

+479 more rows

Використовуємо мову програмування **Python** та проект [JupyterLab](#).



Імпортуюмо необхідні бібліотеки

Weather-Prediction.ipynb +

File + X New Code Cell

```
[13]: # імпортуюмо бібліотеки
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA
from neuralprophet import NeuralProphet
```

Завантажуємо датасет

```
# завантажуємо датасет
df = pd.read_csv('weather.csv')

df.head()
```

			name	datetime	tempmax	tempmin	temp	feelslikemax	feelslikemin	feelslike	dew	humidity	...	solarenergy	uvindex
0	Kryvyi Rih	2020-09-01			35.0	17.0	26.7	32.6	17.0	25.7	6.0	29.7	...	22.9	8
1	Kryvyi Rih	2020-09-02			35.9	18.0	27.0	33.8	18.0	26.1	7.3	30.3	...	21.4	8

```

# удаляємо зайві стовпчики
df = df.loc[:, ['datetime', 'temp']]

# встановимо дату як індекс
df = df.set_index('datetime')

df.head()

```

	temp
	datetime
2020-09-01	26.7
2020-09-02	27.0
2020-09-03	27.8
2020-09-04	24.3
2020-09-05	19.9

Виводимо графік температури повітря.

```

# створимо графік
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(df.index, df['temp'])

# встановимо частоту місячних міток
month_loc = mdates.MonthLocator(bymonthday=1)
ax.xaxis.set_major_locator(month_loc)

# встановимо частоту місячних міток
month_loc = mdates.MonthLocator(bymonthday=1)
ax.xaxis.set_major_locator(month_loc)

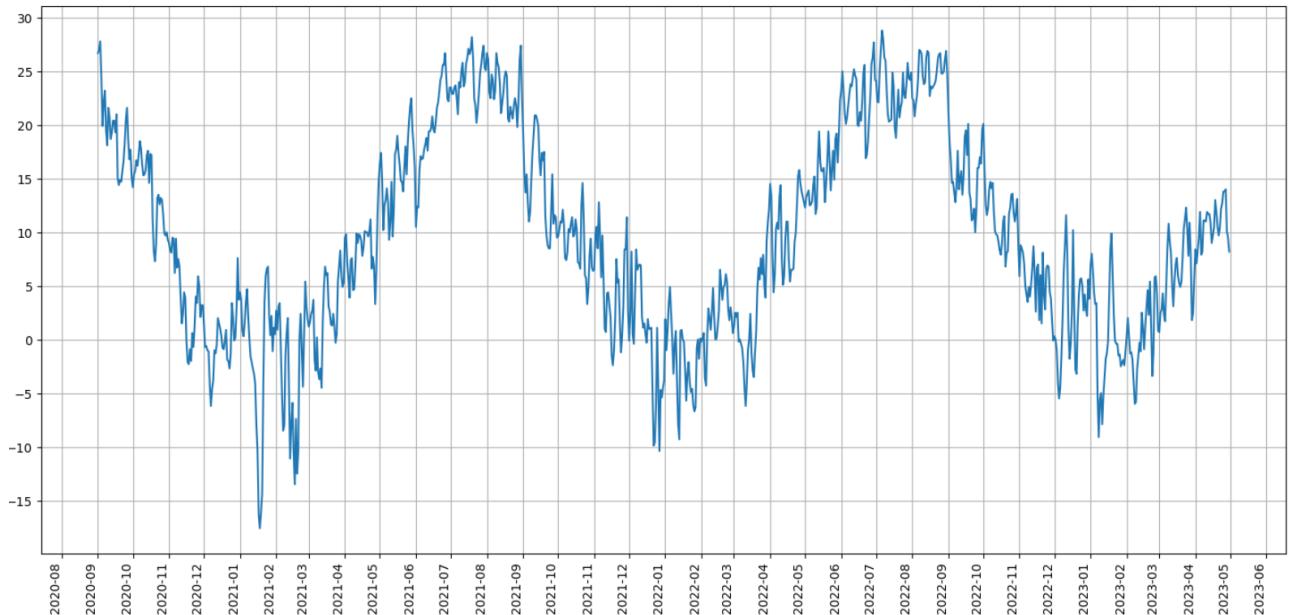
# повернемо текст міток на 90 градусів
fig.autofmt_xdate(rotation=90)

# встановимо мітки на осі у через 5 градусів
y_loc = mticker.MultipleLocator(5)
ax.yaxis.set_major_locator(y_loc)

# додамо сітку
ax.grid(True)

# відобразимо графік
plt.show()

```

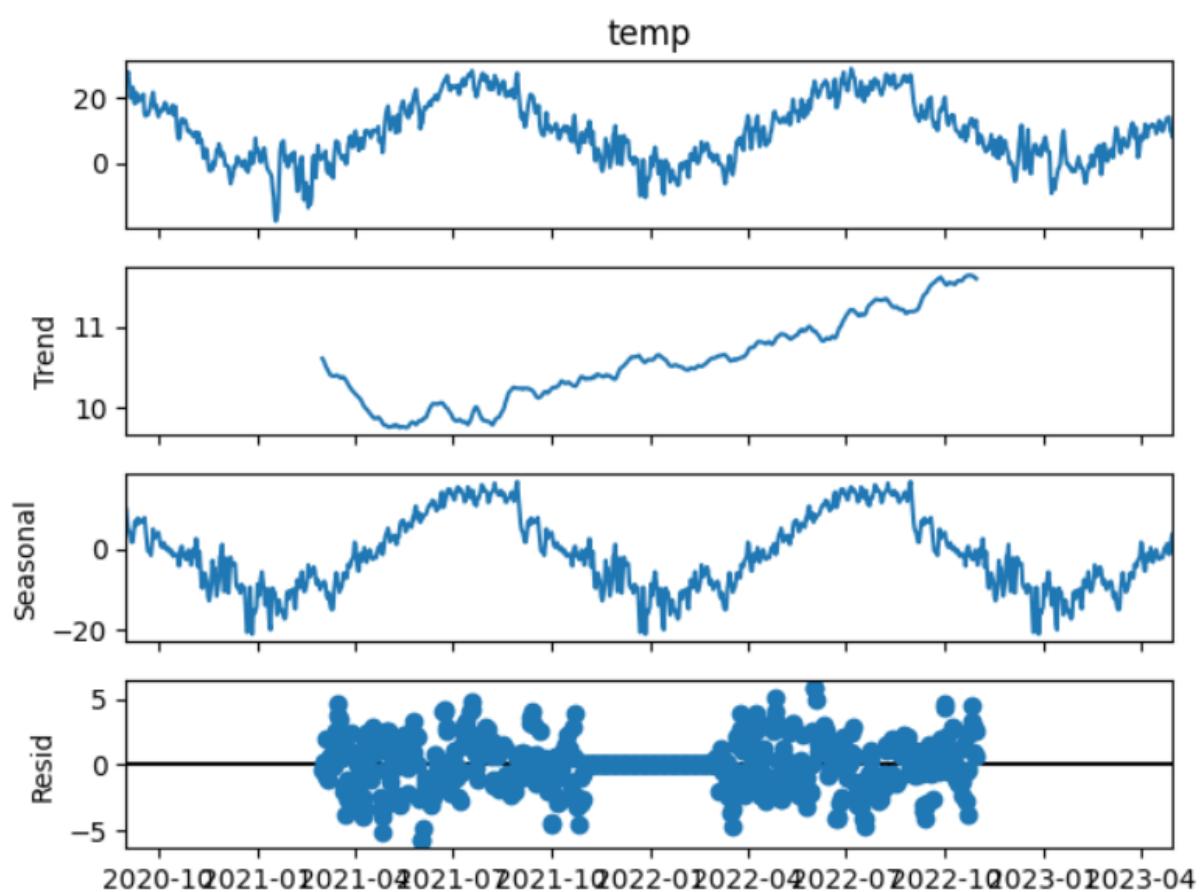


Ряд має сезонність з періодом 1 рік – 365 діб.

Розкладемо ряд на компоненти.

Прогноз зробимо на один місяць – квітень 2023 року.

```
#Розкладемо ряд на компоненти
a = seasonal_decompose(df["temp"], model = "add", period=365)
a.plot();
```



```

#Поділимо датасет на тренувальний та тестовий
test_start = pd.Timestamp('2023-04-01')
test_end = pd.Timestamp('2023-04-30')

#В тренувальний датасет входять дані до першого квітня 2023 року
train_df = df.loc[df.index < test_start]

#В тестовий - квітень 2023 року
test_df = df.loc[(df.index >= test_start) & (df.index <= test_end)]

```

Для прогнозування температури використаємо бібліотеку <https://facebook.github.io/Prophet/>.

**Prophet** - це відкрита бібліотека для прогнозування часових рядів, розроблена командою Facebook.

Ця бібліотека побудована на основі адитивної моделі.

**Prophet** дозволяє прогнозувати часові ряди з додаванням додаткових змінних, наприклад, погодних умов, свяtkovих днів та інших факторів, які можуть вплинути на поведінку часового ряду.

**Prophet** вимагає на вхід датафрейм з двома стовпчиками:

- **ds**: стовпець дат
- **y**: числовий стовпець, який представляє вимірювання, яке ми хочемо передбачити.

```

: # Для використання з Prophet перейменуємо стовпці в ds та у відповідно
train_df_prophet = train_df.reset_index(drop=False)
train_df_prophet.columns = ['ds', 'y']

```

```

test_df_prophet = test_df.reset_index(drop=False)
test_df_prophet.columns = ['ds', 'y']

```

```

: #інтервал надійності 95 % за замовчуванням дорівнює 80%
prophet = Prophet(interval_width=0.95, seasonality_mode='additive')

# Додаємо сезонність на рівні року
prophet.add_seasonality(name='yearly', period=365.25, fourier_order=10)

model = prophet.fit(train_df_prophet, ) #навчання моделі
future = prophet.make_future_dataframe(periods=30)

```

```

forecast = prophet.predict(future)

# Побудова графіку
results = forecast.copy()
results = results.loc[:, ['ds', 'yhat']]
results = results.tail(30).reset_index(drop=True)
results = pd.merge(results, test_df_prophet, on='ds', how='left')

# Обрати прогнозні значення та реальні значення
y_pred = results['yhat']
y_true = results['y']

# Обчислити MSE
mse_prophet = mean_squared_error(y_true, y_pred)

# Побудова графіку
fig, ax = plt.subplots(figsize=(15, 7))

# Графік температури з дадасету
ax.plot(results['ds'], results['y'], label='Temp', color='blue')

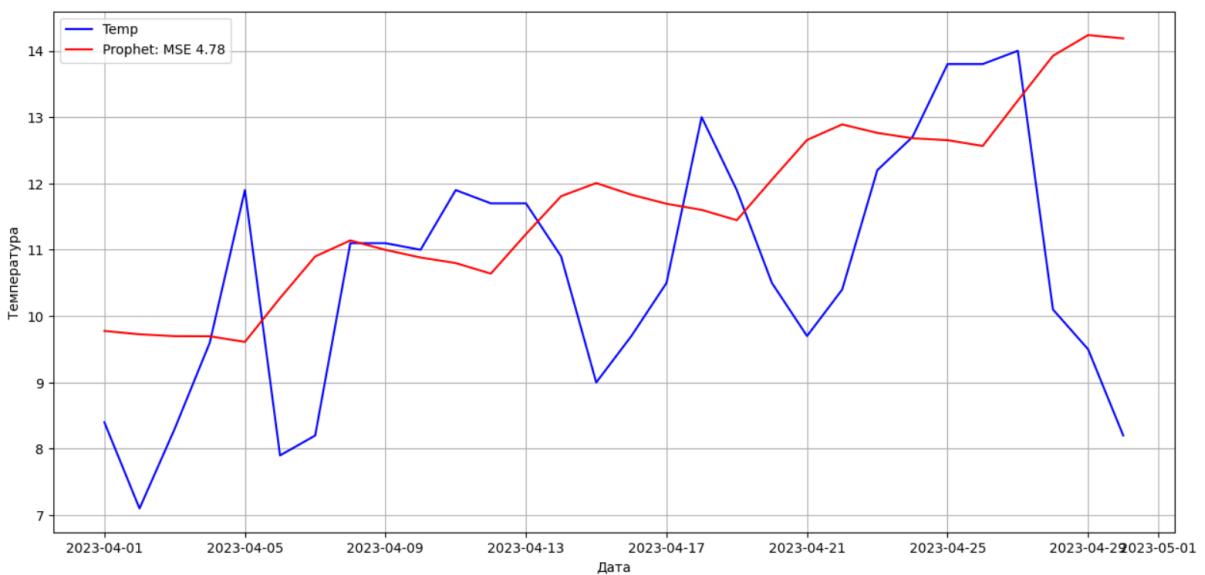
# Графік передбачення Prophet
ax.plot(results['ds'], results['yhat'], label='Prophet: MSE {:.2f}'.format(mse_prophet), color='red')

# Налаштування легенд та осей
ax.set_xlabel('Дата')
ax.set_ylabel('Температура')
ax.legend(loc='upper left')

# додамо сітку
ax.grid(True)

# відобразимо графік
plt.show()

```



Зробимо прогноз температури використовуючи бібліотеку [NeuralProphet](#).

**NeuralProphet** - це відкрита бібліотека на мові **Python** для прогнозування часових рядів з використанням нейронних мереж. Вона є продуктом співпраці команд **Facebook** та **Amazon**, і має великий потенціал в сфері прогнозування різних часових рядів, зокрема в прогнозуванні погоди, фінансових даних, продажів і багатьох інших.

**NeuralProphet** спрощує прогнозування часових рядів і включає в себе модель авторегресії з нейронною мережею **AR-Net**.

**NeuralProphet** вимагає на вхід датафрейм з двома стовпчиками:

- **ds**: стовпець дат

➤ **у:** числовий стовпець, який представляє вимірювання, яке ми хочемо передбачити.

```
# Перетворення індексу на стовпець 'ds', а значення - на стовпець 'y'
# Для використання з NeuralProphet перейменуємо стовпці в ds та у відповідно
train_df_nprophet = train_df.reset_index(drop=False)
train_df_nprophet.columns = ['ds','y']

test_df_nprophet = test_df.reset_index(drop=False)
test_df_nprophet.columns = ['ds','y']
```

```
# Визначення конфігурації моделі NeuralProphet
model = NeuralProphet(
    seasonality_mode='additive', # Режим сезонності. "additive" для адитивної сезонності,
                                # "multiplicative" для мультиплікативної сезонності.
    seasonality_reg=0.1,       # Коефіцієнт регуляризації для сезонності
    learning_rate=0.01,        # Швидкість навчання
    epochs=1000,               # Кількість епох навчання
    batch_size=32,              # Розмір пакета для стохастичного градієнтного спуску
    loss_func='Huber',          # Функція втрат
    yearly_seasonality=True,    # Включення річної сезонності
)
```

```
: # Навчання моделі та отримання прогнозів
metrics = model.fit(train_df_nprophet)

WARNING - (NP.forecaster.fit) - When Global modeling with local normalization, metrics are displayed in normalized scale.
INFO - (NP.df_utils._infer_frequency) - Major frequency D corresponds to 99.894% of the data.
INFO - (NP.df_utils._infer_frequency) - Dataframe freq automatically defined as D
INFO - (NP.config.init_data_params) - Setting normalization to global as only one dataframe provided for training.
INFO - (NP.utils.set_auto_seasonalities) - Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to override this.

Epoch 1000: 100% [1000/1000 [00:00<00:00, 5714.51it/s, loss=0.00231, v_num=14, MAE=2.640, RMSE=3.350, Loss=0.00233, RegLoss=4.6

: future = model.make_future_dataframe(train_df_nprophet, periods=30)

INFO - (NP.df_utils._infer_frequency) - Major frequency D corresponds to 99.894% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - D
INFO - (NP.df_utils._return_df_in_original_format) - Returning df with no ID column

: forecast = model.predict(future)

INFO - (NP.df_utils._infer_frequency) - Major frequency D corresponds to 96.667% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - D
INFO - (NP.df_utils._infer_frequency) - Major frequency D corresponds to 96.667% of the data.
INFO - (NP.df_utils._infer_frequency) - Defined frequency is equal to major frequency - D

Predicting DataLoader 0: 100% [1/1 [00:00<00:00, 130.8

INFO - (NP.df_utils._return_df_in_original_format) - Returning df with no ID column
```

```
# Побудова графіку
results_np = forecast.copy()
results_np = results_np.loc[:, ['ds', 'yhat1']]
results_np = pd.merge(results_np, test_df_prophet, on='ds', how='left')

# Обрати прогнозні значення та реальні значення
y_pred_np = results_np['yhat1']
y_true_np = results_np['y']

# Обчислити MSE
mse_neural_prophet = mean_squared_error(y_true_np, y_pred_np)

# Побудова графіку
fig, ax = plt.subplots(figsize=(15, 7))

# Графік температури з дадасету
ax.plot(results['ds'], results['y'], label='Temp', color='blue')

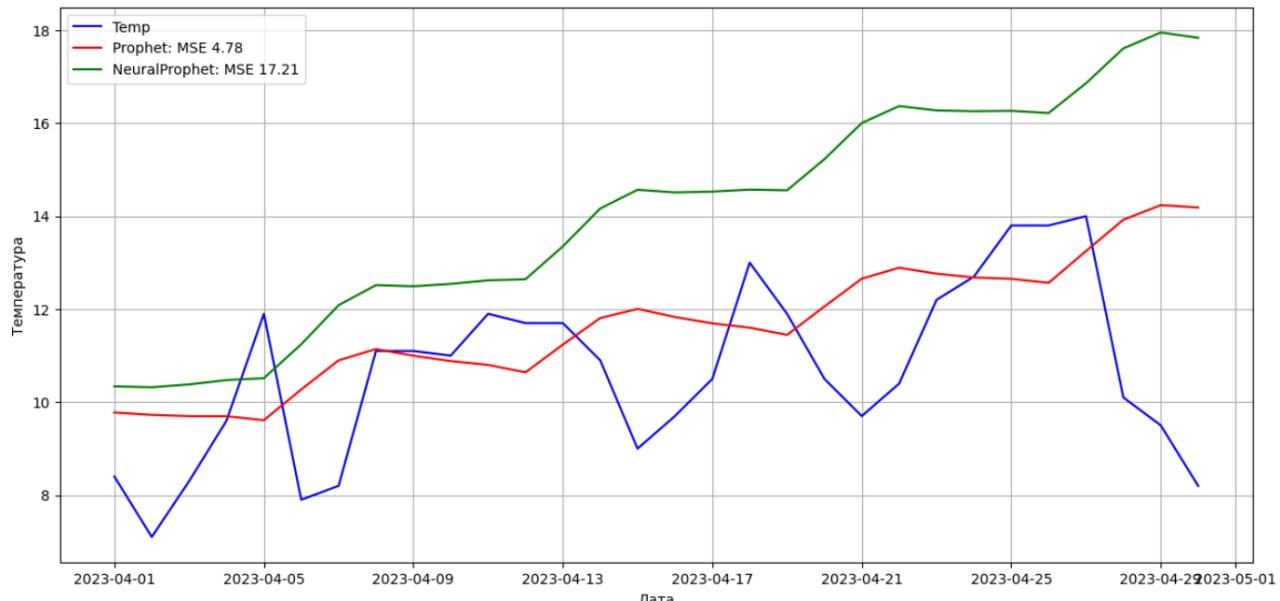
# Графік передбачення Prophet
ax.plot(results['ds'], results['yhat'], label='Prophet: MSE {:.2f}'.format(mse_prophet), color='red')

# Графік передбачення NeuralProphet
ax.plot(results['ds'], results_np['yhat1'], label='NeuralProphet: MSE {:.2f}'.format(mse_neural_prophet), color='green')

# Налаштування легенди та осей
ax.set_xlabel('Дата')
ax.set_ylabel('Температура')
ax.legend(loc='upper left')

# додамо сітку
ax.grid(True)

# відобразимо графік
plt.show()
```



Зробимо прогноз температури використовуючи нейромережу типу **LSTM**.

#Підготовка даних для LSTM моделі

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
```

# Нормалізація даних

```
data = df['temp'].values.reshape(-1, 1)
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

# Розділення даних на навчальний та тестовий набори

```
train_size = int(len(scaled_data) - 40)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]
```

```

# Побудова моделі LSTM
model = Sequential()
model.add(LSTM(64, input_shape=(window_size, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

lstm_model.summary()

```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 300)	362400
dense_1 (Dense)	(None, 1)	301
<hr/>		
Total params: 362,701		
Trainable params: 362,701		
Non-trainable params: 0		

```

# Функція для створення даних зі зсувом для LSTM
def create_dataset(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

```

```

# Параметри моделі та тренування
window_size = 10
train_X, train_y = create_dataset(train_data, window_size)
test_X, test_y = create_dataset(test_data, window_size)

# Побудова та навчання моделі LSTM
model = Sequential()
model.add(LSTM(300, input_shape=(window_size, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(train_X, train_y, epochs=30, batch_size=16)

# Прогнозування температури для тестового набору
predictions = model.predict(test_X)

# Відновлення оригінальних значень температури
predictions = scaler.inverse_transform(predictions)

```

```
58/58 [=====] - 1s 24ms/step - loss: 0.0033
Epoch 22/30
58/58 [=====] - 1s 24ms/step - loss: 0.0030
Epoch 23/30
58/58 [=====] - 1s 26ms/step - loss: 0.0028
Epoch 24/30
58/58 [=====] - 2s 26ms/step - loss: 0.0027
Epoch 25/30
58/58 [=====] - 2s 27ms/step - loss: 0.0027
Epoch 26/30
58/58 [=====] - 1s 25ms/step - loss: 0.0027
Epoch 27/30
58/58 [=====] - 1s 25ms/step - loss: 0.0027
Epoch 28/30
58/58 [=====] - 1s 24ms/step - loss: 0.0026
Epoch 29/30
58/58 [=====] - 2s 29ms/step - loss: 0.0026
Epoch 30/30
58/58 [=====] - 2s 27ms/step - loss: 0.0027
```

```
# Побудова графіку
results_np = forecast.copy()
results_np = results_np.loc[:, ['ds', 'yhat1']]
results_np = pd.merge(results_np, test_df_prophet, on='ds', how='left')

results_LTSM = pd.DataFrame(predictions, columns=['y_LTSM'])

# Обчислити MSE
mse_LTSM = mean_squared_error(y_true_np, y_pred_np)

# Побудова графіку
fig, ax = plt.subplots(figsize=(15, 7))

# Графік температури з дадасету
ax.plot(results['ds'], results['y'], label='Temp', color='blue')

# Графік передбачення Prophet
ax.plot(results['ds'], results['yhat'], label='Prophet: MSE {:.2f}'.format(mse_prophet), color='red')

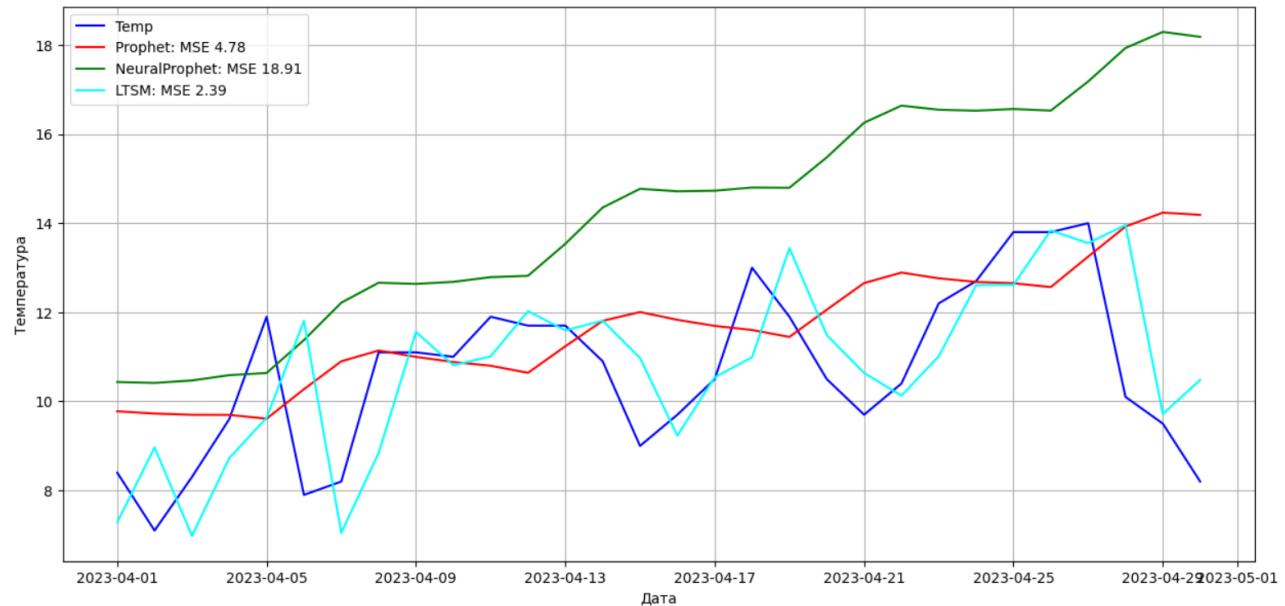
# Графік передбачення NeuralProphet
ax.plot(results['ds'], results_np['yhat1'], label='NeuralProphet: MSE {:.2f}'.format(mse_neural_prophet), color='green')

# Графік передбачення LTSM
ax.plot(results['ds'], results_LTSM['y_LTSM'], label='LTSM: MSE {:.2f}'.format(mse_LTSM), color='cyan')

# Налаштування легенди та осей
ax.set_xlabel('Дата')
ax.set_ylabel('Температура')
ax.legend(loc='upper left')

# додамо сітку
ax.grid(True)

# відобразимо графік
plt.show()
```



## ВИСНОВКИ

В результаті виконаної лабораторної роботи побудовані авторегресійні прогнозуючі моделі та виконано прогнозування часових рядів за допомогою нейромережевих структур.

Усі матеріали викладені у репозіторії GitHub, за посиланням <https://github.com/Max11mus/Artifition-Intelect-Lab7>.