
Laporan Kecerdasan Web dan Big Data A

Tugas 1 - Scraping Data
Tugas 2 - Stream Data Web API

Disusun Oleh:

Sila Ulfania

5024221016

Dosen Pengampu:

Arta Kusuma Hernanda



COMPUTER ENGINEERING

DEPARTMEN TEKNIK KOMPUTER
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SEMESTER GANJIL 2024

Daftar Isi

Daftar Isi	ii
Daftar Gambar	iii
Daftar Tabel	v
1 Scraping Data Web IMDb 1000 Top Movie Menggunakan Selenium Python	1
1.1 Tujuan	1
1.2 Dasar Teori	1
1.3 Alat dan Bahan	3
1.4 Prosedur	3
1.4.1 Bagian 1: Scraping Data Web IMDb 1000 Top Movie	3
1.4.2 Bagian 2: Visualisasi Data IMDb 1000 Top Movie Menggunakan PowerBI	3
1.5 Skenario	3
1.6 Instruksi Tugas 1	4
1.7 Implementasi	4
1.7.1 Bagian 1: Scraping Data Web IMDb 1000 Top Movie	4
1.7.2 Bagian 2: Visualisasi Data IMDb 1000 Top Movie Menggunakan PowerBI	8
1.8 Analisis	11
1.8.1 Hubungan title dengan Tahun Rilis	11
1.8.2 Hubungan title dengan vote	11
1.8.3 Hubungan title dengan metacore	11
1.8.4 Hubungan title dengan duration	11
1.8.5 Hubungan title dengan imdbrating	11
2 Streaming Data Web API IOT Sensor Menggunakan Kafka Python	12
2.1 Tujuan	12
2.2 Dasar Teori	12
2.3 Alat dan Bahan	14
2.4 Prosedur	14

2.5	Skenario: Streaming Data Web API IOT Sensor Monitor Menggunakan Kafka Python	15
2.5.1	Langkah-langkah Instalasi dan Run Apache Kafka di Windows . . .	15
2.6	Langkah-Langkah Streaming Data Sensor IOT Menggunakan Kafka Python	16
2.6.1	Langkah-Langkah Membuat Kafka Consumer	
	Program ini adalah consumer Kafka yang terus mendengarkan topik ssi_data, menerima data JSON, dan menuliskannya ke dalam file CSV.	19
2.7	Prediksi Data Menggunakan Machine Learning LTSM	23
2.7.1	Temperature Sensor 1 dan Temperature Sensor 2	26
2.7.2	Humidity Sensor 1 dan Humidity 2	33
2.7.3	Light Intensity Sensor	39
Daftar Pustaka		43

Daftar Gambar

1.1	Web IMDb 1000 Top Movie	1
1.2	Title dengan Tahun Rilis	9
1.3	Title dengan Vote	9
1.4	Title dengan Metascore	9
1.5	Title dengan Duration	10
1.6	Title dengan IMDb Rating	10
2.1	Web IOT Sensor Monitor	12
2.2	Setting Server	15
2.3	Setting Zookeeper	15
2.4	Start Zookeeper	15
2.5	Start Server	16
2.6	Data Sensor IOT	23
2.7	Data Info Sensor IOT	24
2.8	Visual Semua Data IOT	25
2.9	Data Temperature Sensor 1	26
2.10	Data Temperature Sensor 1	27
2.11	Model LSTM Temperature Sensor 1	29
2.12	Model LSTM Temperature Sensor 2	29
2.13	Hasil Predict Temperature Sensor 1	30
2.14	Hasil Predict Temperature Sensor 2	31
2.15	Grafik Temperature Sensor 1	32
2.16	Grafik Temperature Sensor 2	32
2.17	Data Humidity Sensor 1	33
2.18	Data Humidity Sensor 2	34
2.19	Hasil Predict Humidity Sensor 1	36
2.20	Hasil Predict Humidity Sensor 2	37
2.21	Grafik Humidity Sensor 1	38
2.22	Grafik Humidity Sensor 2	38
2.23	Light Intensity Sensor	39
2.24	Light Intensity Sensor	41
2.25	Hasil Predict Light Intensity Sensor	41

2.26 Grafik Light Intensity Sensor	42
--	----

Daftar Tabel

Tugas 1

Scraping Data Web IMDb 1000 Top Movie Menggunakan Selenium Python

Link Web IMDb: https://www.imdb.com/search/title/?groups=top1000count=250sort=user_rating,desc

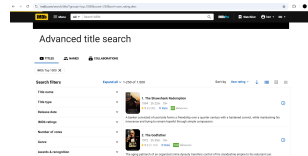


Figure 1.1: Web IMDb 1000 Top Movie

1.1 Tujuan

Memahami cara kerja dan teknik melakukan scraping data menggunakan Selenium Python serta melakukan visualisasi hasil data scraping menggunakan PowerBI membantu menyajikan informasi tersebut secara jelas dan menarik, memudahkan pengambilan keputusan berbasis data. Kombinasi scraping dan visualisasi ini tidak hanya meningkatkan efisiensi dalam pengumpulan dan analisis data, tetapi juga memperkuat kemampuan Anda dalam mengelola pipeline data secara end-to-end, yang sangat penting dalam dunia kerja modern berbasis data.

1.2 Dasar Teori

- **Scraping Data** adalah teknik otomatisasi untuk memperoleh informasi dari sebuah situs web tanpa perlu menyalinnya secara manual. Tujuan utama dari web

scraping adalah menemukan dan mengumpulkan informasi tertentu untuk digunakan dalam situs web lain atau keperluan analisis. Teknik ini berfokus pada proses pengambilan dan ekstraksi data. Manfaat dari web scraping adalah memungkinkan pengumpulan data yang lebih terarah, sehingga memudahkan pencarian informasi yang dibutuhkan. Aplikasi web scraping berfokus pada bagaimana cara mengambil dan mengekstrak data, dengan ukuran data yang dapat bervariasi.

- **Cara Kerja Scraping Data:** Untuk mengekstraksi informasi dan data modern saat ini, cara termudah untuk memperoleh data dari website adalah dengan menggunakan tools data scraping yang telah diprogram oleh developer. Proses utama dalam scraping data meliputi:

- **Request:** Program dimulai dengan proses request ke website menggunakan command GET untuk mengekstrak seluruh data dari halaman yang diinginkan.
- **Parse:** Setelah itu, program akan mencari data spesifik yang telah diidentifikasi untuk diekstraksi pada tools data scraping.
- **Display:** Informasi yang telah di-request kemudian akan diubah menjadi sebuah laporan yang telah dibuat atau disesuaikan.

- **Fungsi Utama Scraping Data:**

- Pengumpulan data dalam jumlah besar dari berbagai sumber online secara otomatis, yang sangat berguna untuk keperluan analisis Big Data.
- Data yang dikumpulkan bisa digunakan untuk riset pasar, tren konsumen, atau analisis kompetitor.
- Scraping digunakan untuk melacak perubahan harga produk atau layanan secara otomatis.
- Membantu dalam mengotomatiskan tugas-tugas rutin, seperti mengumpulkan data dari situs web tertentu secara berkala.

- **IMDb:** IMDb (Internet Movie Database) adalah situs web terbesar dan paling populer yang menyediakan informasi lengkap tentang film, serial TV, video game, dan berbagai konten hiburan lainnya. IMDb didirikan pada tahun 1990 dan kini dimiliki oleh Amazon. Dengan beberapa fitur diantaranya yaitu: Database Film dan Serial, Informasi Pemeran dan Kru, Ulasan dan Rating, Daftar Film Teratas, Trailer dan Media, dan Berita atau Wawancara.

- **Selenium Python:** Selenium adalah library yang dibutuhkan untuk proses otomatisasi pada browser web. Selenium Python mempermudah pengembang untuk melakukan web scraping, pengujian aplikasi web, atau interaksi dengan elemen-elemen halaman web secara otomatis melalui skrip Python.

- **PowerBI:** PowerBI adalah sebuah aplikasi dari Microsoft untuk visualisasi dan analisis data. Dengan PowerBI, pengguna dapat membuat laporan dan dashboard interaktif yang menampilkan data secara visual, sehingga memudahkan pemahaman informasi dan pengambilan keputusan berbasis data.

1.3 Alat dan Bahan

- VS Code
- Laptop
- Jaringan Internet
- Web IMDb 1000 Top Movie
- Library Selenium, Python
- PowerBI

1.4 Prosedur

1.4.1 Bagian 1: Scraping Data Web IMDb 1000 Top Movie

- Instalasi dan konfigurasi Selenium untuk web scraping
- Mengambil data film (judul, rating, tanggal rilis, dll.) dari halaman IMDb.
- Menyimpan data hasil scraping ke dalam format tabular CSV.
- Memastikan data tersimpan dengan struktur yang rapi untuk memudahkan querying dan analisis.

1.4.2 Bagian 2: Visualisasi Data IMDb 1000 Top Movie Menggunakan PowerBI

- Import dan koneksi data dari file CSV ke PowerBI.
- Membersihkan dan mempersiapkan data untuk visualisasi.
- Membuat visualisasi dasar seperti grafik rating, jumlah penonton, dan popularitas film.

1.5 Skenario

1. Web scraping adalah teknik otomatisasi untuk mengambil informasi dari situs web tanpa menyalinnya secara manual.

2. PowerBI, aplikasi ini membantu menyajikan data secara visual sehingga informasi lebih mudah dipahami dan mendukung pengambilan keputusan berbasis data.

1.6 Instruksi Tugas 1

1. Scraping Data Web IMDb 1000 Top Movie:
 - Instalasi dan Konfigurasi Selenium.
 - Scraping Data Film dari IMDb.
 - Menyimpan Data ke dalam Format CSV.
2. Visualisasi Data IMDb 1000 Top Movie Menggunakan PowerBI:
 - Import dan Koneksi Data dari CSV.
 - Membuat Visualisasi Dari Hasil Data.
3. Analisis:
 - Hubungan title dengan Tahun Rilis.
 - Hubungan title dengan vote.
 - Hubungan title dengan metascore.
 - Hubungan title dengan duration.
 - Hubungan title dengan imdbrating.

1.7 Implementasi

1.7.1 Bagian 1: Scraping Data Web IMDb 1000 Top Movie

a. Library yang Digunakan untuk Scraping dan Penyimpanan Data

- Selenium: Selenium digunakan untuk membuka halaman website dari IMDb secara otomatis. Library ini mempermudah proses interaksi dengan elemen-elemen web yang dinamis, yang tidak dapat diakses hanya dengan permintaan HTTP biasa. Dengan Selenium, kita dapat mengontrol browser secara langsung untuk mengakses, menggulir, dan mengambil data dari halaman tertentu di IMDb.

Code

```
1 from selenium import webdriver
2 from bs4 import BeautifulSoup
3 from pandas import DataFrame
4 from selenium.webdriver.common.by import By
5 from selenium.webdriver.support.ui import WebDriverWait
6 from selenium.webdriver.support import expected_conditions as EC
```

```
7 import pandas as pd
8 import time
```

b. Menggunakan WebDriver Chrome untuk Mengakses Halaman IMDb

- Untuk drivanya disini saya memakai Webdriver Chrome menggunakan Selenium. Dan dari link tersebut akan masuk ke web IMDB dan membuka halaman yang menampilkan 250 film dari Top 1000.

Code

```
1 driver = webdriver.Chrome()
2 driver.get('https://www.imdb.com/search/title/?groups=top_1000&count=250&sort
3 =user_rating,desc')
```

c. Menggunakan Fungsi Button untuk Menampilkan Detail Film

- Selanjutnya fungsi button digunakan untuk menampilkan dan mengaktifkan tampilan detailnya.

Code

```
1 button = WebDriverWait(driver, 10).until(
2 EC.element_to_be_clickable((By.ID, 'list-view-option-detailed'))) )
3 button.click()
4 time.sleep(3)
```

d. Menggunakan Loop untuk Memuat Lebih Banyak Film dengan Tombol "250 More"

- while True bertujuan untuk terus-menerus mengecek keberadaan tombol "250 More" di halaman IMDb, yang berfungsi untuk memuat lebih banyak film dalam daftar secara bertahap. Loop ini sangat berguna ketika data film di halaman tersebut tidak ditampilkan sekaligus, melainkan memerlukan beberapa kali klik pada tombol "250 More" untuk memuat seluruh daftar.

Code

```
1 while True:
2     try:
3         # Tunggu sampai tombol "250 More" muncul dan bisa di klik
4         more_button = WebDriverWait(driver, 10).until(
5             EC.element_to_be_clickable((By.XPATH,
6                 ↪ "//button[contains(@class, 'ipc-see-more__button')]"))
7         )
8         print(more_button.is_displayed())
9         print(more_button.is_enabled())
10        print('Bisa')
11        driver.execute_script("arguments[0].click();",more_button)
12        print("clicked")
13        time.sleep(5)
14    except Exception as e:
15        print(e)
16        print("No more '250 more' button found or all content loaded.")
17        break
```

e. Membuat List Kosong

- Beberapa list (Titles, Images, Years, Durations, Imdbratings, Metascores, Votes, Descs) untuk menyimpan data film yang akan diambil .

Code

```
1 Titles = []
2 Images = []
3 Years = []
4 Durations = []
5 Imdb_ratings = []
6 Metascores = []
7 Votes = []
8 Descs
```

f. Mengambil Indeks HTML dan Mencari Daftar Film dengan BeautifulSoup

- `soup = BeautifulSoup(driver.page_source, 'html.parser')` digunakan untuk mengambil indeks HTML.
- `movie_list = soup.find.all('li', { 'class': 'ipc-metadata-list-summary-item' })` digunakan untuk menemukan semua item film dalam daftar.
- Setiap data film diambil satu per satu dari objek `movie_list` dan ditambahkan ke

dalam list yang sesuai.

Code

```
1 soup = BeautifulSoup(driver.page_source, 'html.parser')
2 movie_list = soup.find_all('li', {'class':
    ↳ 'ipc-metadata-list-summary-item'})
3 for movie in movie_list:
4     Title = movie.h3.text if movie.h3 else 'N/A'
5     Titles.append(Title)
6     Image = movie.img['src'] if movie.img else 'N/A'
7     Images.append(Image)
8     Year = movie.find_all('span', {'class': 'sc-ab348ad5-8 cSWcJI
    ↳ dli-title-metadata-item'})[0].text if len(movie.find_all('span',
    ↳ {'class': 'sc-ab348ad5-8 cSWcJI dli-title-metadata-item'})) > 0 else
    ↳ 'N/A'
9     Years.append(Year)
10    Duration = movie.find_all('span', {'class': 'sc-ab348ad5-8 cSWcJI
    ↳ dli-title-metadata-item'})[1].text if len(movie.find_all('span',
    ↳ {'class': 'sc-ab348ad5-8 cSWcJI dli-title-metadata-item'})) > 1
    ↳ else 'N/A'
11    Durations.append(Duration)
12    Imdb_rating = movie.find('span', {'class': 'ipc-rating-star
    ↳ ipc-rating-star--base ipc-rating-star--imdb
    ↳ ratingGroup--imdb-rating'}).text if movie.find('span', {'class':
    ↳ 'ipc-rating-star ipc-rating-star--base ipc-rating-star--imdb
    ↳ ratingGroup--imdb-rating'}) else 'N/A'
13    Imdb_ratings.append(Imdb_rating)
14    Metascore = movie.find('span', {'class': 'sc-b0901df4-0 bXI0oL
    ↳ metacritic-score-box'}).text if movie.find('span', {'class':
    ↳ 'sc-b0901df4-0 bXI0oL metacritic-score-box'}) else 'N/A'
15    Metascores.append(Metascore)
16    Vote = movie.find('span', {'class': 'ipc-rating-star--voteCount'}).text
    ↳ if movie.find('span', {'class': 'ipc-rating-star--voteCount'}) else
    ↳ 'N/A'
17    Votes.append(Vote)
18    Desc = movie.find('div', {'class': 'ipc-html-content-inner-div'}).text
    ↳ if movie.find('div', {'class': 'ipc-html-content-inner-div'}) else
    ↳ 'N/A'
19    Descs.append(Desc)
```

g. Menyimpan Data ke dalam DataFrame

- Data dari list diubah menjadi DataFrame Pandas dengan kolom seperti 'Title', 'Image', 'Year', dan lainnya.

Code

```
1 Data = pd.DataFrame({
2     'Title': Titles,
3     'Image': Images,
4     'Year': Years,
5     'Duration': Durations,
6     'Imdb_rating': Imdb_ratings,
7     'Metascore' : Metascores,
8     'Vote': Votes,
9     'Desc': Descs,
10 })
11 print(Data)
```

h. Menyimpan ke File Excel/CSV

- `Data.to_excel('IMDB3.xlsx', index=False)` menyimpan DataFrame ke file Excel bernama "IMDB3.xlsx". `driver.close()` digunakan untuk menutup browser setelah proses scraping selesai.

Code

```
1 Data.to_excel('IMDB3.xlsx', index=False)
2 print('DataFrame is written to Excel File Successfully.')
3 driver.close()
```

1.7.2 Bagian 2: Visualisasi Data IMDb 1000 Top Movie Menggunakan PowerBI

a. Hasil Data CSV

- Link CSV: <https://drive.google.com/file/d/1KYizbwahlsdssffkFFOXBn10oMQ0vWtr/view?usp=sharing>

b. Visualisasi Dari Hasil Data

- Title terhadap tahun rilis:
- Title terhadap vote:
- Title terhadap metascore:
- Title terhadap durasi:
- Title terhadap IMDb rating:

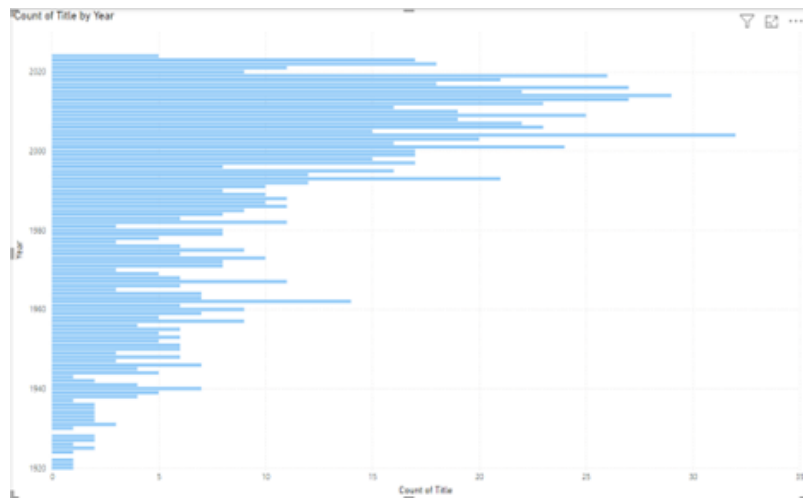


Figure 1.2: Title dengan Tahun Rilis

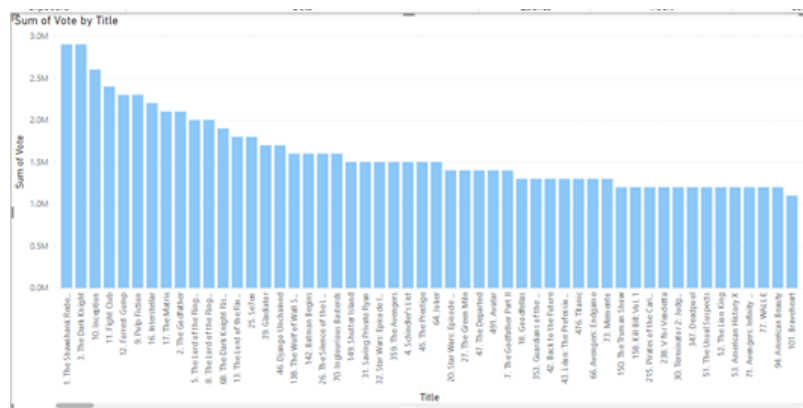


Figure 1.3: Title dengan Vote

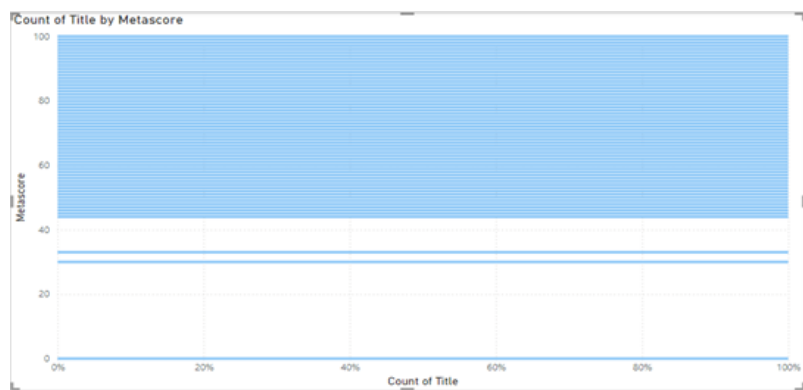


Figure 1.4: Title dengan Metascore

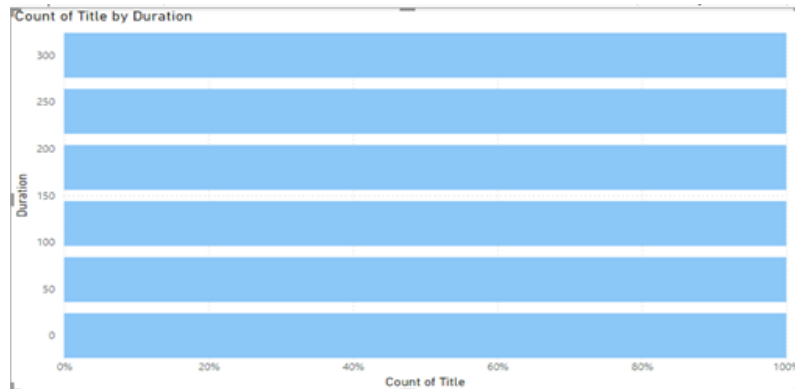


Figure 1.5: Title dengan Duration

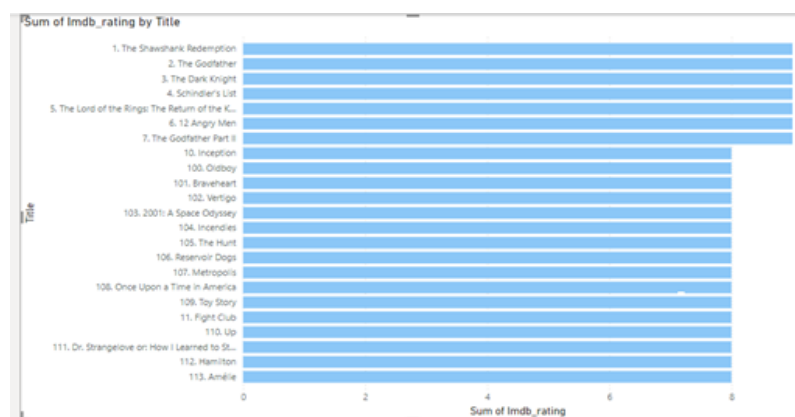


Figure 1.6: Title dengan IMDb Rating

1.8 Analisis

1.8.1 Hubungan title dengan Tahun Rilis

- Film yang dirilis di tahun-tahun tertentu mungkin mencerminkan tren atau genre populer pada masa itu dan sampai sekarang.
- Dari hasil data tahun yang banyak paling merilis film terdapat pada tahun 2004 dengan jumlah film yang dirilis sebanyak 32 title.
- Sedangkan untuk tahun 1920-1924, 1926, dan tahun 1930 dan hanya 1 title saja.

1.8.2 Hubungan title dengan vote

- Jumlah vote mencerminkan popularitas film di kalangan audiens.
- Dari gambar visual total perolehan vote didapatkan 2,9 M dan vote paling sedikit didapatkan 26.000

1.8.3 Hubungan title dengan metaspcore

- Metaspcore adalah indikator dari penilaian kritis. Film dengan metaspcore tinggi biasanya menunjukkan penerimaan positif dari kritikus profesional.
- Metaspcore paling sedikit didapatkan 0 dengan jumlah title terbanyak 161
- Metaspcore terbanyak didapatkan 100 dengan jumlah title sebanyak 16

1.8.4 Hubungan title dengan duration

- Durasi film dapat memengaruhi pengalaman audiens. Film yang lebih panjang biasanya menargetkan cerita yang lebih kompleks atau detail.
- ilm berdurasi panjang (>90 menit) mungkin memiliki rating lebih tinggi jika alur ceritanya menarik. Durasi 120: 477title, 180: 45 title, 240: 2 title, dan 300: 1 title.
- Film pendek (<90 menit) cenderung lebih ringan, tetapi bisa memiliki durasi ideal untuk genre seperti komedi. 60: 474 title.

1.8.5 Hubungan title dengan imdbrating

- IMDb rating adalah refleksi kombinasi dari kritik profesional dan respon audiens.
- Rating tinggi (di atas 8.0) sering kali dimiliki oleh film dengan narasi kuat, akting luar biasa, dan elemen sinematik berkualitas.
- Rating nilai ini berurutan sesuai rating 1-1000 pada gambar visualisasi. Dengan nilai rating skal 1-10.

Tugas 2

Streaming Data Web API IOT Sensor Menggunakan Kafka Python

Link Web Sensor IOT: <https://ssiot.jlbsd.my.id/>



Figure 2.1: Web IOT Sensor Monitor

2.1 Tujuan

Memahami cara kerja dan teknik pengelolaan data yang mengalir secara real-time dari sumber data ke sistem lain sangat penting dalam era digital. Teknik ini memungkinkan pengolahan data secara cepat dan efisien, terutama untuk aplikasi yang memerlukan analisis atau respons waktu nyata, seperti sistem pemantauan, Internet of Things (IoT), atau layanan berbasis lokasi. Selain itu, data yang dikelola dapat dianalisis lebih lanjut atau digunakan dalam pemodelan prediktif dengan algoritma machine learning, seperti LSTM. Pendekatan ini memberikan kemampuan untuk memprediksi pola di masa depan, mendeteksi anomali, dan menghasilkan wawasan relevan yang dapat langsung diimplementasikan.

2.2 Dasar Teori

- **Streaming Data**, streaming adalah proses pengiriman data secara terus menerus, biasanya pengiriman data dilakukan dalam ukuran yang kecil (bukan dalam bentuk batch) dalam aliran yang tetap berkelanjutan saat data dihasilkan. Karena data

yang dikirim adalah data dalam ukuran kecil, maka jika datanya besar harus dikonversi terlebih dahulu menjadi data yang berukuran kecil (biasanya dalam bentuk byte). Dengan streaming data, ini memungkinkan kita untuk dapat menganalisis data secara real time atau nyata dan memberi berbagai macam pengetahuan seperti pengukuran, aktivitas server, geolokasi perangkat dan lain sebagainya. Salah satu tools yang dapat digunakan untuk streaming data adalah Kafka.

- **IoT Sensor Monitor** adalah sistem pemantauan yang menggunakan sensor pintar untuk mengumpulkan data dari lingkungan sekitar secara terus-menerus. Data ini kemudian dikirim ke pusat data untuk dianalisis dan ditampilkan
- **Kafka** merupakan platform Apache untuk streaming data yang didistribusikan. Secara implementasi, Kafka merupakan sistem pengiriman pesan yang mempunyai fitur publish-subscribe terdistribusi yang mana pesan dikelola dalam topik, baik partisi ataupun replikasinya. Secara sederhana, alur penggunaan Kafka melibatkan beberapa komponen yaitu:
 1. **Producer**: Komponen ini bertanggung jawab menghasilkan dan mengirimkan pesan (data) yang akan didistribusikan. Producer menentukan pesan yang akan dikirim dan topik (topic) tujuan pengiriman. Setiap pesan dapat dilampirkan dengan key, sehingga pesan-pesan dengan key yang sama akan diarahkan ke topik atau partisi yang sesuai.
 2. **Consumer**: Komponen ini membaca pesan yang telah disimpan di dalam Topic. Consumer dapat membaca data dari sekumpulan partisi dalam topik tertentu. Selain itu, Consumer dapat dibentuk dalam kelompok (consumer group), di mana setiap grup yang subscribe pada topik yang sama akan menerima pesan secara terdistribusi untuk mencegah redundansi data, sehingga data yang sama tidak diproses lebih dari satu kali.
 3. **Topic**: Merupakan sebuah log yang berfungsi menerima pesan dari Producer dan menyimpan pesan tersebut ke dalam partisinya.
 4. **Partisi** adalah unit penyimpanan dalam Kafka yang terdiri dari serangkaian pesan yang diurutkan secara teratur. Setiap partisi disimpan di satu atau lebih broker dalam cluster Kafka. Partisi memungkinkan Kafka untuk melakukan skalabilitas secara horizontal dan meningkatkan throughput secara keseluruhan.
 5. **Broker** adalah server dalam cluster Kafka yang bertanggung jawab untuk menyimpan dan mengelola partisi-partisi topic. Setiap broker dalam cluster Kafka menyimpan sebagian dari data dan melayani permintaan-producer dan permintaan-consumer.
 6. **Replication** adalah proses menyalin partisi dari satu broker ke broker lain dalam cluster Kafka. Ini adalah bagian dari strategi toleransi kesalahan Kafka, yang memastikan bahwa data tetap tersedia dan tidak hilang meskipun terjadi kegagalan pada salah satu broker.

- **Zookeeper** adalah layanan terdistribusi yang dirancang untuk menyediakan koordinasi dan sinkronisasi antar sistem dalam lingkungan terdistribusi, seperti Kafka. Zookeeper berfungsi sebagai pusat pengelolaan metadata yang memungkinkan komponen-komponen dalam sistem terdistribusi bekerja secara konsisten dan efisien. Fungsi utama Zookeeper dalam konteks Kafka:
 1. Menyimpan informasi penting seperti daftar broker Kafka yang aktif, topik, partisi, dan replikasi.
 2. Mengelola koordinasi antar broker dalam cluster Kafka, termasuk proses pemilihan pemimpin (leader election) untuk partisi tertentu.
 3. Dengan Zookeeper, Kafka dapat tetap berjalan meskipun salah satu broker gagal, karena koordinasi dan informasi cluster tetap tersedia melalui Zookeeper.
- **Long Short-Term Memory (LSTM)** adalah jenis jaringan saraf tiruan (Artificial Neural Network) yang termasuk dalam keluarga Recurrent Neural Network (RNN). LSTM dirancang khusus untuk menangani data sekuensial atau data yang memiliki hubungan temporal, seperti deret waktu (time series), data sensor, atau data sekuensial lainnya. Algoritma ini sangat efektif dalam mempelajari pola jangka panjang dan pendek dari data historis untuk menghasilkan proyeksi atau prediksi di masa depan.

2.3 Alat dan Bahan

- VS Code
- Laptop
- Jaringan Internet
- Web API IOT Sensor Monitor
- Kafka Python
- Zookeeper
- LTSM

2.4 Prosedur

1. Instalasi Apache Kafka
2. Streaming Data Menggunakan Python
3. Prediksi Menggunakan Machine Learning (LTSM)

2.5 Skenario: Streaming Data Web API IOT Sensor Monitor Menggunakan Kafka Python

Kita akan mengimplementasikan Streaming Data Menggunakan Kafka Python dengan Broker Zookeeper.

2.5.1 Langkah-langkah Instalasi dan Run Apache Kafka di Windows

a. Pastikan Kafka diinstal dan tersimpan di directory C atau sejenisnya. **Link Install Kafka:**

<https://kafka.apache.org/downloads> Pilih Scala 2.13, lalu setelah extract all file zip dan simpan di directory.

b. Buka File Kafka untuk melakukan setting, lalu buka folder config >server.properties >klik kanan, pilih Edit in Notepad >ubah log.dirs=c:/kafka/kafka-logs

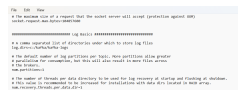


Figure 2.2: Setting Server

c. Selanjutnya setting bagian zookeeper.properties, lalu buka folder config >Zookeeper.properties >klik kanan, pilih Edit in Notepad >ubah dataDir=c:/kafka/zookeeper-

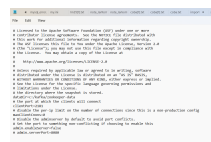


Figure 2.3: Setting Zookeeper

d. Selanjutnya, buka Windows PowerShell atau terminal sejenisnya untuk mengaktifkan Broker Zookeeper dengan menggunakan perintah:

```
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

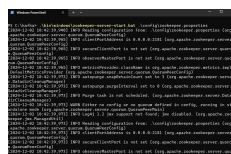


Figure 2.4: Start Zookeeper

e. Selanjutnya, buka lagi Windows PowerShell atau terminal sejenisnya untuk mengaktifkan Kafka Server dengan menggunakan perintah:

```
.\bin\windows\kafka-server-start.bat .\config\kafka.properties
```

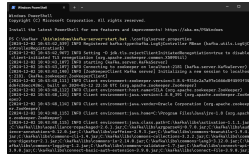


Figure 2.5: Start Server

2.6 Langkah-Langkah Streaming Data Sensor IOT Menggunakan Kafka Python

a. Langkah-Langkah Membuat Kafka Producer

Membuat program untuk mengambil data dari lima endpoint API terkait sensor suhu, kelembaban, dan intensitas cahaya. Data yang diambil dikirimkan ke sebuah topik di Kafka untuk digunakan dalam pipeline streaming data.

b. Buat File Python `kafka_producer.py` menggunakan VS Code ataupun sejenisnya.

c. Mengimport Library yang dibutuhkan, `kafka`: untuk mengintegrasikan Kafka dalam program, `requests`: Untuk melakukan permintaan HTTP ke API, `json`: Untuk mengonversi data API menjadi format JSON yang dapat dibaca Kafka, dan `time`: Untuk memberikan jeda waktu antarpermintaan.

Code

```
1 from kafka import KafkaProducer
2 from kafka.errors import KafkaError
3 import requests
4 import json
5 import time
```

d. ungsi `fetch_data(url)`: untuk mengambil data dari URL API. akan melakukan request HTTP GET ke URL yang diberikan. Memastikan status respons adalah 200 (sukses). Jika gagal, akan memunculkan error. ika terjadi error (misalnya URL salah atau server tidak merespons), pesan error dicetak, dan None dikembalikan.

Code

```
1 def fetch_data(url):
2     try:
3         response = requests.get(url)
4         response.raise_for_status() # Raise error jika bukan 200
5         return response.json()
6     except Exception as e:
7         print(f"Error fetching data from {url}: {e}")
8         return None
```

e. Fungsi `send_to_kafka(producer, topic, data)`: untuk mengirim data ke Kafka topic. Fungsi pengiriman data ke Kafka dimulai dengan mengonversi data JSON dari API menjadi string menggunakan `json.dumps(data)`. String tersebut kemudian dikodekan ke format byte dengan `.encode('utf-8')`, agar kompatibel dengan Kafka, yang hanya menerima data dalam bentuk byte. Setelah itu, data dikirim ke Kafka dengan `producer.send()` ke topik yang telah ditentukan. Untuk mencegah kegagalan proses, fungsi ini memiliki error handling yang akan mencetak pesan kesalahan jika terjadi masalah, membantu pengembang mengidentifikasi dan memperbaiki error dengan mudah.

Code

```
1 def send_to_kafka(producer, topic, data):
2     try:
3         # Convert data ke JSON string untuk Kafka
4         producer.send(topic, value=json.dumps(data).encode('utf-8'))
5         print(f"Data sent to Kafka topic '{topic}': {data}")
6     except Exception as e:
7         print(f"Error sending data to Kafka: {e}")
```

f. Fungsi `main()`: untuk mengatur alur pengambilan dan pengiriman data.

1. Program dimulai dengan mendefinisikan daftar URL yang berisi data sensor yang akan diambil.

2. Konfigurasi Kafka ditentukan dengan alamat broker Kafka (`kafka_broker`) dan topik Kafka (`kafka_topic`) untuk mengirimkan data.
3. Kemudian, producer Kafka diinisialisasi menggunakan `KafkaProducer` untuk memfasilitasi pengiriman data. Program memasuki loop utama, yang mengiterasi melalui daftar URL API satu per satu.
4. Setiap URL diproses dengan memanggil fungsi `fetch_data(url)` untuk mengambil data. Jika data berhasil diambil, fungsi `send_to_kafka()` digunakan untuk mengirimkan data ke topik Kafka yang ditentukan.
5. Setelah memproses satu URL, program akan menunggu selama 1 detik sebelum melanjutkan ke URL berikutnya untuk memastikan alur pengambilan dan pengiriman data berjalan lancar.

Code

```
1  def main():
2      api_urls = [
3          "https://ssiot.jlbsd.my.id/api/temperature1",
4          "https://ssiot.jlbsd.my.id/api/temperature2",
5          "https://ssiot.jlbsd.my.id/api/humidity1",
6          "https://ssiot.jlbsd.my.id/api/humidity2",
7          "https://ssiot.jlbsd.my.id/api/light"
8      ]
9
10     kafka_broker = "localhost:9092"
11     kafka_topic = "ssi_data"
12
13     producer = KafkaProducer(bootstrap_servers=[kafka_broker])
14
15     while True:
16         for url in api_urls:
17             data = fetch_data(url)
18             if data:
19                 send_to_kafka(producer, kafka_topic, data)
20                 time.sleep(1)
```


g. manggilan Fungsi: untuk Menentukan bahwa program ini hanya akan menjalankan fungsi main() jika dijalankan langsung (bukan diimpor sebagai modul).

Code

```
1     if __name__ == "__main__":  
2         main()
```

2.6.1 Langkah-Langkah Membuat Kafka Consumer

Program ini adalah consumer Kafka yang terus mendengarkan topik ssi_data, menerima data JSON, dan menuliskannya ke dalam file CSV.

a. Mengimport library yang dibutuhkan:

Code

```
1     from kafka import KafkaConsumer  
2     import json  
3     import csv  
4     import os
```

b. Konfigurasi untuk Kafka

1. kafka_broker: untuk menyimpan alamat broker Kafka yang digunakan untuk menghubungkan dengan cluster Kafka. Broker Kafka berjalan di localhost pada port 9092.
2. kafka_topic: untuk menentukan nama topik Kafka yang akan didengarkan oleh consumer. Di sini, topik yang digunakan adalah ssi_data.
3. output_csv: Menentukan nama file CSV yang digunakan untuk menyimpan data yang diterima dari Kafka, yaitu kafka_consumer_output.csv.

Code

```
1     kafka_broker = "localhost:9092"  
2     kafka_topic = "ssi_data"  
3     output_csv = "kafka_consumer_output.csv"
```

c. Memeriksa apakah file CSV dengan nama yang ditentukan sudah ada. Fungsi ini akan mengembalikan nilai True jika file tersebut ada, dan False jika tidak ada. Ini digunakan untuk memutuskan apakah perlu menulis header baru pada file CSV.

Code

```
1 file_exists = os.path.exists(output_csv)
```

d. `afkaConsumer(...)`: Membuat objek consumer untuk mendengarkan data dari Kafka. Parameter yang digunakan:

1. `kafka.topic`: Menentukan topik Kafka yang ingin didengarkan.
2. `bootstrap_servers`: Alamat broker Kafka yang digunakan untuk menghubungkan consumer ke Kafka.
3. `auto_offset_reset='earliest'`: Menginstruksikan consumer untuk mulai membaca pesan dari offset awal jika belum ada pesan yang dibaca sebelumnya.
4. `enable_auto_commit=True`: Mengatur Kafka untuk secara otomatis mengelola komitmen offset setelah pesan dibaca.
5. `group_id='ssi_consumer_group'`: Nama grup untuk consumer yang memungkinkan beberapa consumer dapat bekerja bersama untuk membaca data secara paralel.
9. `value_deserializer=lambda x: json.loads(x.decode('utf-8'))`: Mendekode data yang diterima (dalam format JSON) menjadi struktur data Python, yaitu dictionary, setelah pesan di-decode dari byte menjadi string UTF-8.

Code

```
1 consumer = KafkaConsumer(  
2     kafka_topic,  
3     bootstrap_servers=[kafka_broker],  
4     auto_offset_reset='earliest',  
5     enable_auto_commit=True,  
6     group_id='ssi_consumer_group',  
7     value_deserializer=lambda x: json.loads(x.decode('utf-8'))  
8 )
```

e. Program ini dimulai dengan menginisialisasi `KafkaConsumer` yang terhubung ke Kafka broker dan topik yang ditentukan (`ssi_data`). Setelah itu, program membuka file CSV dalam mode `append`, memastikan data baru ditambahkan tanpa menghapus data yang ada. Data yang diterima dari Kafka kemudian di-deserialize dari format JSON menjadi struktur Python (biasanya dictionary) sebelum ditulis ke file CSV. Proses ini memeriksa apakah data yang diterima berupa dictionary dan menuliskannya ke file dengan header yang sesuai. Proses ini berulang hingga proses dihentikan secara manual (misalnya dengan `Ctrl+C`). Alur ini memastikan data yang diterima dari Kafka terus disimpan dalam file CSV untuk analisis lebih lanjut.

Code

```
1 with open(output_csv, mode='a', newline='', encoding='utf-8') as file:
2     csv_writer = None
```

f. Program akan menerima Pesan dari Kafka, Loop yang terus berjalan untuk mendengarkan pesan yang dikirim ke topik Kafka. Setiap pesan yang diterima akan diproses di dalam loop ini. Kemudian mengambil data yang terkandung dalam pesan Kafka. Data ini berupa JSON yang sudah didekodekan ke dalam dictionary Python.

Code

```
1 for message in consumer:
2     data = message.value
3     print(f"Received data: {data}")
```

g. Pastikan bahwa data yang diterima merupakan dictionary, karena hanya data dalam format ini yang akan disimpan ke dalam CSV. Jika data bukan dictionary, data tersebut akan dilewati.

Code

```
1 if not isinstance(data, dict):
2     print(f"Skipped non-dictionary data: {data}")
3     continue
```

h. Variabel `csv_writer` digunakan untuk menyimpan objek penulis CSV yang akan menangani proses penulisan data ke dalam file CSV. Pada awalnya, `csv_writer` diset ke `None` untuk menunjukkan bahwa objek penulis belum diinisialisasi. Kemudian, saat data pertama kali diterima, objek `csv.DictWriter` dibuat untuk menulis dictionary ke dalam file CSV. Jika file CSV masih kosong, header ditentukan berdasarkan kunci data pertama yang diterima (menggunakan `data.keys()`). Jika file sudah ada, header yang ada diambil dari file untuk memastikan konsistensi dalam penulisan kolom data selanjutnya.

Code

```
1     if csv_writer is None:
2     if not file_exists:
3         header = data.keys()
4         csv_writer = csv.DictWriter(file, fieldnames=header)
5         csv_writer.writeheader()
6     else:
7         with open(output_csv, mode='r', encoding='utf-8') as existing_file:
8             existing_header = next(csv.reader(existing_file))
9             csv_writer = csv.DictWriter(file, fieldnames=existing_header)
```

i. Menulis data yang diterima ke dalam file CSV dan memaksa penulisan data ke disk, memastikan data ditulis tanpa menunggu buffer penuh.

Code

```
1     csv_writer.writerow(data)
2     file.flush()
```

j. Menangani sinyal dari keyboard (seperti menekan `Ctrl+C`) untuk menghentikan consumer dengan cara yang bersih tanpa meninggalkan proses yang tidak terkelola.

Code

```
1     except KeyboardInterrupt:
2         print("\nConsumer stopped.")
```

2.7 Prediksi Data Menggunakan Machine Learning LSTM

a. Impor library yang dibutuhkan:

Code

```
1 from datetime import datetime
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sklearn.preprocessing import MinMaxScaler
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import LSTM, Dense
8 from keras.utils import to_categorical
```

b. Kemudian masukkan file CSV yang berisi data sensor IOT, dan akan ditampilkan data-data dari sensor IOT.

Code

```
1 file_path = "kafka_consumer_output.csv" #
2 df = pd.read_csv(file_path)
3 df
```

	sensor	value	unit	timestamp
0	Temperature Sensor 1	20.59	°C	2024-11-27T13:30:41.312Z
1	Temperature Sensor 2	23.20	°C	2024-11-27T13:30:46.316Z
2	Humidity Sensor 1	10.08	%	2024-11-27T13:30:51.318Z
3	Humidity Sensor 2	56.23	%	2024-11-27T13:30:57.320Z
4	Light Intensity Sensor	328.53	lux	2024-11-27T13:31:02.320Z
...
107606	Temperature Sensor 2	24.16	°C	2024-12-04T17:06:32.508Z
107607	Humidity Sensor 1	59.91	%	2024-12-04T17:06:37.509Z
107608	Humidity Sensor 2	41.03	%	2024-12-04T17:06:42.510Z
107609	Light Intensity Sensor	100.75	lux	2024-12-04T17:06:47.510Z
107610	Temperature Sensor 1	26.46	°C	2024-12-04T17:06:53.512Z
107611 rows × 4 columns				

Figure 2.6: Data Sensor IOT

c. Selanjutnya melakukan konversi kolom timestamp ke format datetime64 agar mudah diolah, dan menampilkan data info dari sensor IOT.

```
Code

1 df['timestamp'] = pd.to_datetime(df['timestamp'])
2 print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 107611 entries, 0 to 107610
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   sensor      107611 non-null object
1   value       107611 non-null float64
2   unit        107611 non-null object
3   timestamp   107611 non-null datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](1), float64(1), object(2)
memory usage: 3.3+ MB
None
```

Figure 2.7: Data Info Sensor IOT

d. Menampilkan visualisasi dari masing-masing data Sensor IOT: Temperature 1, Temperature 2, Humidity Sensor 1, Humidity Sensor 2, dan Lihgt Intensity Sensor. Data dipisahkan berdasarkan jenis sensor.

Code

```
1 sensors = df['sensor'].unique()
2 sensor_data = {sensor: df[df['sensor'] == sensor] for sensor in sensors}
3
4 # Plot data untuk setiap sensor
5 plt.figure(figsize=(15, 10))
6 for i, sensor in enumerate(sensors):
7     plt.subplot(len(sensors), 1, i + 1)
8     plt.plot(sensor_data[sensor]['timestamp'], sensor_data[sensor]['value'],
9     ↪ label=sensor)
10    plt.xlabel("Timestamp")
11    plt.ylabel("Value")
12    plt.title(sensor)
13    plt.legend()
14 plt.tight_layout()
15 plt.show()
```

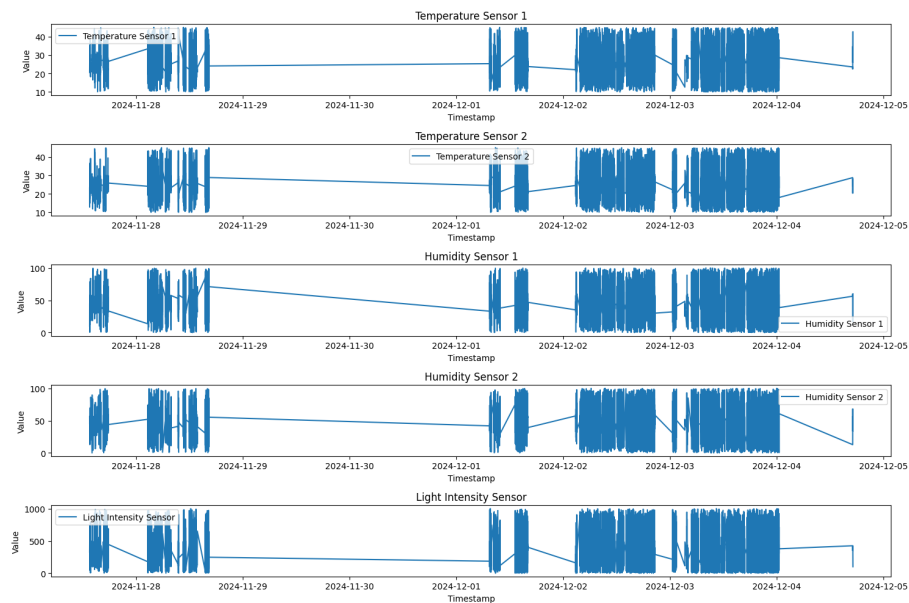


Figure 2.8: Visual Semua Data IOT

2.7.1 Temperature Sensor 1 dan Temperature Sensor 2

a. Memfilter data sensor IOT, untuk data Temperature 1. Kemudian ditampilkan hasil data Temperature.

Code

```
1 temp_sensor_1_df = df[df['sensor'] == "Temperature Sensor 1"]
2 display(temp_sensor_1_df)
```

	sensor	value	unit	timestamp
0	Temperature Sensor 1	20.59	°C	2024-11-27 13:30:41.312000+00:00
5	Temperature Sensor 1	25.85	°C	2024-11-27 13:31:07.321000+00:00
10	Temperature Sensor 1	26.06	°C	2024-11-27 13:31:34.325000+00:00
15	Temperature Sensor 1	25.34	°C	2024-11-27 13:32:01.330000+00:00
20	Temperature Sensor 1	22.29	°C	2024-11-27 13:32:28.339000+00:00
...
107589	Temperature Sensor 1	25.95	°C	2024-12-04 17:05:07.488000+00:00
107595	Temperature Sensor 1	31.36	°C	2024-12-04 17:05:34.495000+00:00
107600	Temperature Sensor 1	26.94	°C	2024-12-04 17:06:00.503000+00:00
107605	Temperature Sensor 1	42.61	°C	2024-12-04 17:06:26.507000+00:00
107610	Temperature Sensor 1	26.46	°C	2024-12-04 17:06:53.512000+00:00

21530 rows × 4 columns

Figure 2.9: Data Temperature Sensor 1

b. Memfilter data sensor IOT, untuk data Temperature 1. Kemudian ditampilkan hasil data Temperature.

Code

```
1 temp_sensor_2_df = df[df['sensor'] == "Temperature Sensor 2"]
2 display(temp_sensor_2_df)
```


	sensor	value	unit	timestamp
1	Temperature Sensor 2	23.20	°C	2024-11-27 13:30:46.316000+00:00
6	Temperature Sensor 2	12.80	°C	2024-11-27 13:31:13.321000+00:00
11	Temperature Sensor 2	25.79	°C	2024-11-27 13:31:40.326000+00:00
16	Temperature Sensor 2	20.41	°C	2024-11-27 13:32:07.332000+00:00
21	Temperature Sensor 2	17.51	°C	2024-11-27 13:32:33.342000+00:00
...
107548	Temperature Sensor 2	23.10	°C	2024-12-04 00:31:46.079000+00:00
107553	Temperature Sensor 2	25.72	°C	2024-12-04 00:31:53.079000+00:00
107558	Temperature Sensor 2	29.48	°C	2024-12-04 00:31:59.079000+00:00
107563	Temperature Sensor 2	20.59	°C	2024-12-04 00:32:05.081000+00:00
107568	Temperature Sensor 2	18.04	°C	2024-12-04 00:32:12.082000+00:00

21521 rows x 4 columns

Figure 2.10: Data Temperature Sensor 1

c. **Normalisasi Data:** Agar data berada pada rentang 0 hingga 1, untuk melatih LSTM karena LSTM bekerja lebih baik pada data dengan skala kecil.

Code

```

1     scaler = MinMaxScaler(feature_range=(0, 1))
2     temp_sensor_1_df['scaled_value'] =
    ↪ scaler.fit_transform(temp_sensor_1_df[['value']])
3
4     scaler = MinMaxScaler(feature_range=(0, 1))
5     temp_sensor_2_df['scaled_value'] =
    ↪ scaler.fit_transform(temp_sensor_2_df[['value']])

```

d. **Membuat Data Sequence:** Membagi data menjadi sekuensial (input x) dan target (y) untuk melatih LSTM.

Code

```

1     def create_sequences(data, sequence_length=50):
2     x, y = [], []
3     for i in range(len(data) - sequence_length):
4         x.append(data[i:i + sequence_length])
5         y.append(data[i + sequence_length])
6     return np.array(x), np.array(y)
7
8     # Membuat urutan
9     sequence_length = 50
10    x, y = create_sequences(temp_sensor_1_df['scaled_value'].values,

```

```

    ↪ sequence_length)
11 x = np.expand_dims(x, axis=-1)
12
13 sequence_length = 50
14 x, y = create_sequences(temp_sensor_2_df['scaled_value'].values,
    ↪ sequence_length)
15 x = np.expand_dims(x, axis=-1) # LSTM expects input of shape (samples,
    ↪ timesteps, features)
16
17 # Membagi menjadi set pelatihan dan pengujian
18 # 80% pelatihan dan 20% pengujian
19 split = int(0.8 * len(x))
20 x_train, x_test = x[:split], x[split:]
21 y_train, y_test = y[:split], y[split:]
22
23 split = int(0.8 * len(x)) # 80% training and 20% testing
24 x_train, x_test = x[:split], x[split:]
25 y_train, y_test = y[:split], y[split:]

```

e. Membuat Model LSTM Temperature 1

Code

```

1 # Output layer
2 model = Sequential([
3     LSTM(165, activation='relu', input_shape=(sequence_length, 1)),
4     Dense(1)
5 ])
6 # Mean Squared Error loss function
7 model.compile(optimizer='adam', loss='mse')
8 model.summary()

```

f. Membuat Model LSTM Temperature 2

Code

```

1 # Output layer
2 model = Sequential([
3     LSTM(165, activation='relu', input_shape=(sequence_length, 1)),
4     Dense(1)
5 ])
6 # Mean Squared Error loss function

```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 165)	110,220
dense_5 (Dense)	(None, 1)	166

Total params: 110,386 (431.20 KB)

Trainable params: 110,386 (431.20 KB)

Non-trainable params: 0 (0.00 B)

Figure 2.11: Model LSTM Temperature Sensor 1

```

7     model.compile(optimizer='adam', loss='mse')
8     model.summary()

```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 165)	110,220
dense_6 (Dense)	(None, 1)	166

Total params: 110,386 (431.20 KB)

Trainable params: 110,386 (431.20 KB)

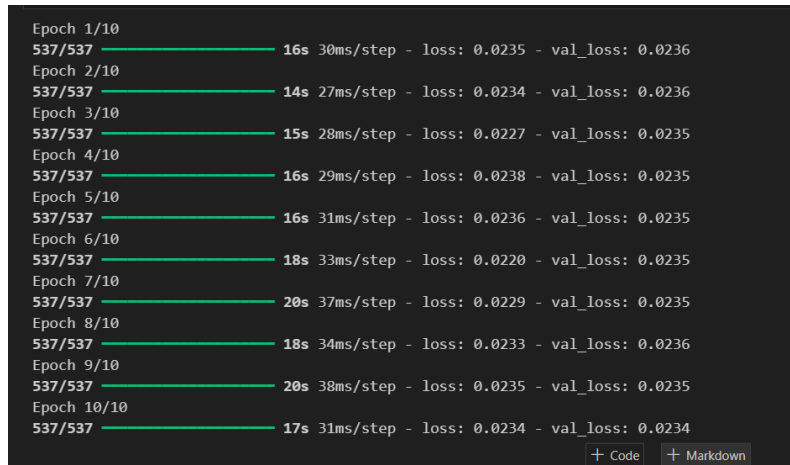
Non-trainable params: 0 (0.00 B)

Figure 2.12: Model LSTM Temperature Sensor 2

g. Kode tersebut digunakan untuk melatih model menggunakan data pelatihan (x_train, y_train) dengan 10 epoch dan batch size 32. Untuk Temperature Sensor 1

Code

```
1 history = model.fit(x_train, y_train, epochs=10, batch_size=32,  
    ↪ validation_data=(x_test, y_test))  
2 predicted = model.predict(x_test)  
3 predicted = scaler.inverse_transform(predicted) # Rescale back to original  
    ↪ values  
4 y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```



```
Epoch 1/10  
537/537 ————— 16s 30ms/step - loss: 0.0235 - val_loss: 0.0236  
Epoch 2/10  
537/537 ————— 14s 27ms/step - loss: 0.0234 - val_loss: 0.0236  
Epoch 3/10  
537/537 ————— 15s 28ms/step - loss: 0.0227 - val_loss: 0.0235  
Epoch 4/10  
537/537 ————— 16s 29ms/step - loss: 0.0238 - val_loss: 0.0235  
Epoch 5/10  
537/537 ————— 16s 31ms/step - loss: 0.0236 - val_loss: 0.0235  
Epoch 6/10  
537/537 ————— 18s 33ms/step - loss: 0.0220 - val_loss: 0.0235  
Epoch 7/10  
537/537 ————— 20s 37ms/step - loss: 0.0229 - val_loss: 0.0235  
Epoch 8/10  
537/537 ————— 18s 34ms/step - loss: 0.0233 - val_loss: 0.0236  
Epoch 9/10  
537/537 ————— 20s 38ms/step - loss: 0.0235 - val_loss: 0.0235  
Epoch 10/10  
537/537 ————— 17s 31ms/step - loss: 0.0234 - val_loss: 0.0234
```

Figure 2.13: Hasil Predict Temperature Sensor 1

h. Kode tersebut digunakan untuk melatih model menggunakan data pelatihan (x_train, y_train) dengan 10 epoch dan batch size 32. Untuk Temperature Sensor 2

Code

```
1 history = model.fit(x_train, y_train, epochs=10, batch_size=32,  
    ↪ validation_data=(x_test, y_test))  
2 predicted = model.predict(x_test)  
3 predicted = scaler.inverse_transform(predicted) # Rescale back to original  
    ↪ values  
4 y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```

Epoch 1/10
537/537 ————— 21s 37ms/step - loss: 0.0339 - val_loss: 0.0228
Epoch 2/10
537/537 ————— 15s 28ms/step - loss: 0.0231 - val_loss: 0.0226
Epoch 3/10
537/537 ————— 15s 27ms/step - loss: 0.0231 - val_loss: 0.0231
Epoch 4/10
537/537 ————— 15s 28ms/step - loss: 0.0229 - val_loss: 0.0226
Epoch 5/10
537/537 ————— 15s 28ms/step - loss: 0.0231 - val_loss: 0.0225
Epoch 6/10
537/537 ————— 16s 30ms/step - loss: 0.0234 - val_loss: 0.0226
Epoch 7/10
537/537 ————— 16s 29ms/step - loss: 0.0229 - val_loss: 0.0227
Epoch 8/10
537/537 ————— 15s 28ms/step - loss: 0.0238 - val_loss: 0.0228
Epoch 9/10
537/537 ————— 18s 33ms/step - loss: 0.0226 - val_loss: 0.0230
Epoch 10/10
537/537 ————— 16s 30ms/step - loss: 0.0231 - val_loss: 0.0225
135/135 ————— 2s 12ms/step

```

Figure 2.14: Hasil Predict Temperature Sensor 2

i. Kode tersebut digunakan untuk visualisasi perbandingan antara nilai aktual dan prediksi Temperature Sensor 1 sepanjang waktu (timestamp).

Code

```

1 plt.figure(figsize=(10, 6))
2 plt.plot(temp_sensor_1_df['timestamp'].iloc[-len(y_test):], y_test,
   ↪ label='Actual')
3 plt.plot(temp_sensor_1_df['timestamp'].iloc[-len(predicted):], predicted,
   ↪ label='Predicted')
4 plt.xlabel("Timestamp")
5 plt.ylabel("Temperature (C)")
6 plt.title("Actual vs Predicted (Temperature Sensor 1)")
7 plt.legend()
8 plt.show()

```

j. Kode tersebut digunakan untuk visualisasi perbandingan antara nilai aktual dan prediksi Temperature Sensor 2 sepanjang waktu (timestamp).

Code

```

1 plt.figure(figsize=(10, 6))
2 plt.plot(temp_sensor_2_df['timestamp'].iloc[-len(y_test):], y_test,
   ↪ label='Actual')
3 plt.plot(temp_sensor_2_df['timestamp'].iloc[-len(predicted):], predicted,
   ↪ label='Predicted')
4 plt.xlabel("Timestamp")
5 plt.ylabel("Temperature (C)")

```

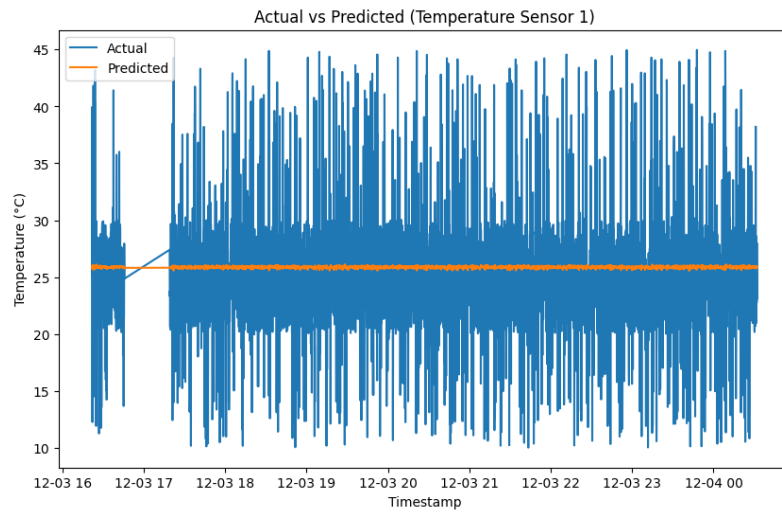


Figure 2.15: Grafik Temperature Sensor 1

```

6 plt.title("Actual vs Predicted (Temperature Sensor 2)")
7 plt.legend()
8 plt.show()

```

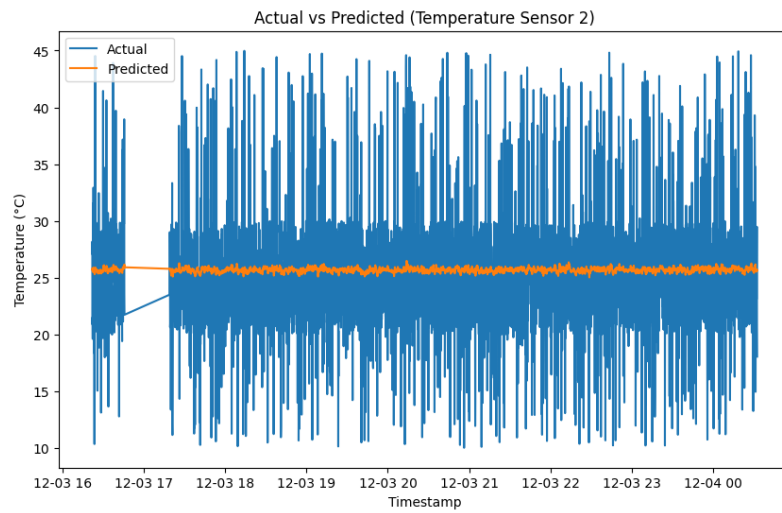


Figure 2.16: Grafik Temperature Sensor 2

2.7.2 Humidity Sensor 1 dan Humidity 2

a. Memfilter data sensor IOT, untuk data Humidity Sensor 1. Kemudian ditampilkan hasil data Humidity .

```
Code

1  humidity_sensor_1_df = df[df['sensor'] == "Humidity Sensor 1"]
2  df
```

	sensor	value	unit	timestamp
2	Humidity Sensor 1	10.08	%	2024-11-27 13:30:51.318000+00:00
7	Humidity Sensor 1	44.81	%	2024-11-27 13:31:18.320000+00:00
12	Humidity Sensor 1	37.36	%	2024-11-27 13:31:45.326000+00:00
17	Humidity Sensor 1	44.19	%	2024-11-27 13:32:12.334000+00:00
22	Humidity Sensor 1	45.24	%	2024-11-27 13:32:39.342000+00:00
...
107549	Humidity Sensor 1	40.18	%	2024-12-04 00:31:47.079000+00:00
107554	Humidity Sensor 1	40.62	%	2024-12-04 00:31:54.078000+00:00
107559	Humidity Sensor 1	59.04	%	2024-12-04 00:32:00.079000+00:00
107564	Humidity Sensor 1	47.11	%	2024-12-04 00:32:07.081000+00:00
107569	Humidity Sensor 1	38.67	%	2024-12-04 00:32:13.082000+00:00

21520 rows x 4 columns

Figure 2.17: Data Humidity Sensor 1

b. Memfilter data sensor IOT, untuk data Humidity Sensor 2. Kemudian ditampilkan hasil data Humidity .

```
Code

1  humidity_sensor_2_df = df[df['sensor'] == "Humidity Sensor 2"]
2  df
```

	sensor	value	unit	timestamp	scaled_value
3	Humidity Sensor 2	56.23	%	2024-11-27 13:30:57.320000+00:00	0.562550
8	Humidity Sensor 2	44.67	%	2024-11-27 13:31:24.322000+00:00	0.446857
13	Humidity Sensor 2	46.68	%	2024-11-27 13:31:51.327000+00:00	0.466974
18	Humidity Sensor 2	54.27	%	2024-11-27 13:32:17.335000+00:00	0.542934
23	Humidity Sensor 2	37.42	%	2024-11-27 13:32:44.344000+00:00	0.374299
...
107550	Humidity Sensor 2	46.13	%	2024-12-04 00:31:49.079000+00:00	0.461469
107555	Humidity Sensor 2	48.83	%	2024-12-04 00:31:55.078000+00:00	0.488491
107560	Humidity Sensor 2	45.46	%	2024-12-04 00:32:01.080000+00:00	0.454764
107565	Humidity Sensor 2	0.66	%	2024-12-04 00:32:08.080000+00:00	0.006405
107570	Humidity Sensor 2	60.62	%	2024-12-04 00:32:14.081000+00:00	0.606485

21511 rows x 5 columns

Figure 2.18: Data Humidity Sensor 2

c. **Normalisasi Data:** Agar data berada pada rentang 0 hingga 1, untuk melatih LSTM karena LSTM bekerja lebih baik pada data dengan skala kecil.

Code

```

1 scaler_1 = MinMaxScaler(feature_range=(0, 1))
2 humidity_sensor_1_df['scaled_value'] =
  ↳ scaler_1.fit_transform(humidity_sensor_1_df[['value']])
3
4 scaler_2 = MinMaxScaler(feature_range=(0, 1))
5 humidity_sensor_2_df['scaled_value'] =
  ↳ scaler_2.fit_transform(humidity_sensor_2_df[['value']])

```

d. **Membuat Data Sequence:** Membagi data menjadi sekuensial (input (x) dan target (y) untuk melatih LSTM.

Code

```

1 def create_sequences(data, sequence_length=50):
2     x, y = [], []
3     for i in range(len(data) - sequence_length):
4         x.append(data[i:i + sequence_length])
5         y.append(data[i + sequence_length])
6     return np.array(x), np.array(y)
7
8     sequence_length = 50
9     x_1, y_1 = create_sequences(humidity_sensor_1_df['scaled_value'].values,
  ↳ sequence_length)
10    x_1 = np.expand_dims(x_1, axis=-1)
11    split_1 = int(0.8 * len(x_1)) # 80% training, 20% testing for Sensor 1

```



```

12     x_1_train, x_1_test = x_1[:split_1], x_1[split_1:]
13     y_1_train, y_1_test = y_1[:split_1], y_1[split_1:]
14
15     sequence_length = 50
16     x, y = create_sequences(temp_sensor_2_df['scaled_value'].values,
17         ↪ sequence_length)
17     x = np.expand_dims(x, axis=-1) # LSTM expects input of shape (samples,
18         ↪ timesteps, features)
18     split_2 = int(0.8 * len(x_2)) # 80% training, 20% testing for Sensor 2
19     x_2_train, x_2_test = x_2[:split_2], x_2[split_2:]
20     y_2_train, y_2_test = y_2[:split_2], y_2[split_2:]

```

e. Membuat Model LSTM Humidity 1

Code

```

1     def build_lstm_model(sequence_length):
2         model = Sequential([
3             LSTM(50, activation='relu', input_shape=(sequence_length, 1)),
4             Dense(1) # Output layer
5         ])
6         model.compile(optimizer='adam', loss='mse') # Mean Squared Error loss
7         ↪ function
8         return model
9
10    model_1 = build_lstm_model(sequence_length)

```

f. Membuat Model LSTM Humidity 2

Code

```

1     def build_lstm_model(sequence_length):
2         model = Sequential([
3             LSTM(50, activation='relu', input_shape=(sequence_length, 1)),
4             Dense(1) # Output layer
5         ])
6         model.compile(optimizer='adam', loss='mse') # Mean Squared Error loss
7         ↪ function
8         return model
9
10    model_2 = build_lstm_model(sequence_length)

```

g. Kode tersebut digunakan untuk melatih model menggunakan data pelatihan (x_train, y_train) dengan 10 epoch dan batch size 32. Untuk Humidity Sensor 1

Code

```
1 history_1 = model_1.fit(x_1_train, y_1_train, epochs=10, batch_size=32,  
    ↪ validation_data=(x_1_test, y_1_test))  
2 predicted_1 = model_1.predict(x_1_test)  
3 predicted_1 = scaler_1.inverse_transform(predicted_1) # Rescale back to  
    ↪ original values  
4 y_1_test = scaler_1.inverse_transform(y_1_test.reshape(-1, 1))
```

```
Epoch 1/10  
537/537 ————— 10s 16ms/step - loss: 0.0348 - val_loss: 0.0243  
Epoch 2/10  
537/537 ————— 6s 12ms/step - loss: 0.0241 - val_loss: 0.0244  
Epoch 3/10  
537/537 ————— 6s 11ms/step - loss: 0.0241 - val_loss: 0.0240  
Epoch 4/10  
537/537 ————— 6s 12ms/step - loss: 0.0241 - val_loss: 0.0239  
Epoch 5/10  
537/537 ————— 6s 11ms/step - loss: 0.0239 - val_loss: 0.0248  
Epoch 6/10  
537/537 ————— 6s 12ms/step - loss: 0.0240 - val_loss: 0.0239  
Epoch 7/10  
537/537 ————— 6s 11ms/step - loss: 0.0238 - val_loss: 0.0239  
Epoch 8/10  
537/537 ————— 6s 12ms/step - loss: 0.0231 - val_loss: 0.0239  
Epoch 9/10  
537/537 ————— 6s 12ms/step - loss: 0.0236 - val_loss: 0.0239  
Epoch 10/10  
537/537 ————— 6s 12ms/step - loss: 0.0240 - val_loss: 0.0238  
135/135 ————— 1s 4ms/step
```

Figure 2.19: Hasil Predict Humidity Sensor 1

h. Kode tersebut digunakan untuk melatih model menggunakan data pelatihan (x_train, y_train) dengan 10 epoch dan batch size 32. Untuk Humidity Sensor 2

Code

```
1 history_2 = model_2.fit(x_2_train, y_2_train, epochs=10, batch_size=32,  
    ↪ validation_data=(x_2_test, y_2_test))  
2 predicted_2 = model_2.predict(x_2_test)  
3 predicted_2 = scaler_2.inverse_transform(predicted_2) # Rescale back to  
    ↪ original values  
4 y_2_test = scaler_2.inverse_transform(y_2_test.reshape(-1, 1))
```

```

Epoch 1/10
537/537 ————— 10s 14ms/step - loss: 0.0465 - val_loss: 0.0243
Epoch 2/10
537/537 ————— 6s 12ms/step - loss: 0.0243 - val_loss: 0.0242
Epoch 3/10
537/537 ————— 6s 11ms/step - loss: 0.0241 - val_loss: 0.0241
Epoch 4/10
537/537 ————— 6s 12ms/step - loss: 0.0240 - val_loss: 0.0243
Epoch 5/10
537/537 ————— 6s 11ms/step - loss: 0.0246 - val_loss: 0.0241
Epoch 6/10
537/537 ————— 6s 12ms/step - loss: 0.0236 - val_loss: 0.0244
Epoch 7/10
537/537 ————— 6s 12ms/step - loss: 0.0244 - val_loss: 0.0243
Epoch 8/10
537/537 ————— 6s 12ms/step - loss: 0.0243 - val_loss: 0.0241
Epoch 9/10
537/537 ————— 7s 12ms/step - loss: 0.0242 - val_loss: 0.0242
Epoch 10/10
537/537 ————— 7s 12ms/step - loss: 0.0240 - val_loss: 0.0241
135/135 ————— 1s 5ms/step

```

Figure 2.20: Hasil Predict Humidity Sensor 2

i. Kode tersebut digunakan untuk visualisasi perbandingan antara nilai aktual dan prediksi Humidity Sensor 1 sepanjang waktu (timestamp).

Code

```

1 plt.figure(figsize=(10, 6))
2 plt.plot(humidity_sensor_1_df['timestamp'].iloc[-len(y_1_test):], y_1_test,
   ↪ label='Actual')
3 plt.plot(humidity_sensor_1_df['timestamp'].iloc[-len(predicted_1):],
   ↪ predicted_1, label='Predicted')
4 plt.xlabel("Timestamp")
5 plt.ylabel("Humidity (%)")
6 plt.title("Actual vs Predicted (Humidity Sensor 1)")
7 plt.legend()
8 plt.show()

```

j. Kode tersebut digunakan untuk visualisasi perbandingan antara nilai aktual dan prediksi Humidity Sensor 2 sepanjang waktu (timestamp).

Code

```

1 plt.figure(figsize=(10, 6))
2 plt.plot(humidity_sensor_2_df['timestamp'].iloc[-len(y_2_test):], y_2_test,
   ↪ label='Actual')
3 plt.plot(humidity_sensor_2_df['timestamp'].iloc[-len(predicted_2):],
   ↪ predicted_2, label='Predicted')
4 plt.xlabel("Timestamp")
5 plt.ylabel("Humidity (%)")

```

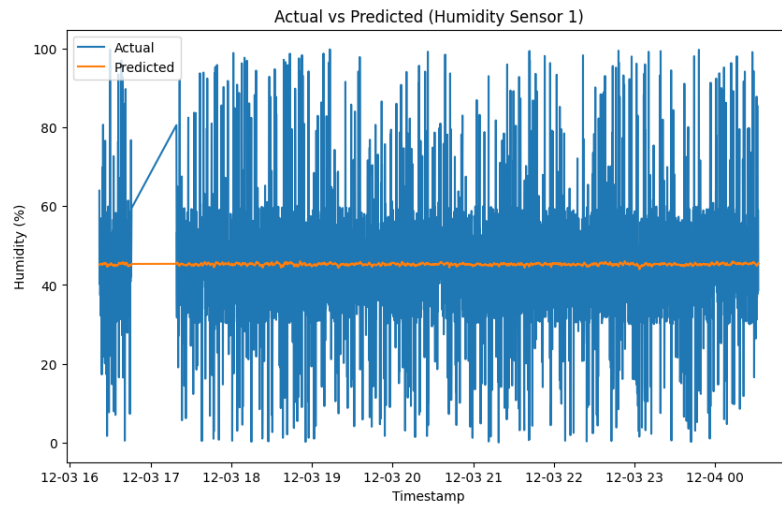


Figure 2.21: Grafik Humidity Sensor 1

```

6 plt.title("Actual vs Predicted (Humidity Sensor 2)")
7 plt.legend()
8 plt.show()

```

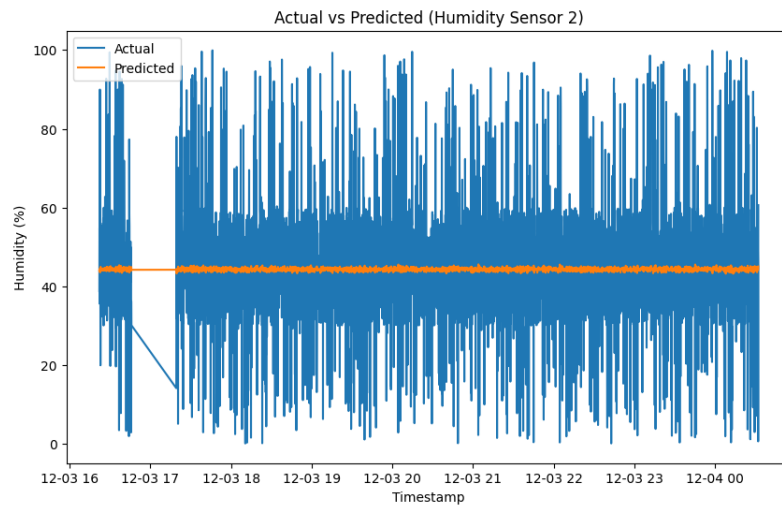


Figure 2.22: Grafik Humidity Sensor 2

2.7.3 Light Intensity Sensor

a. Memfilter data sensor IOT, untuk data Light Intensity Sensor . Kemudian ditampilkan hasil data Humidity .

```
Code

1 light_intensity_sensor_df = df[df['sensor'] == 'Light Intensity Sensor']
2 df
```

	sensor	value	unit	timestamp
4	Light Intensity Sensor	328.53	lux	2024-11-27 13:31:02.320000+00:00
9	Light Intensity Sensor	472.00	lux	2024-11-27 13:31:29.322000+00:00
14	Light Intensity Sensor	499.25	lux	2024-11-27 13:31:56.328000+00:00
19	Light Intensity Sensor	143.43	lux	2024-11-27 13:32:23.337000+00:00
24	Light Intensity Sensor	741.73	lux	2024-11-27 13:32:49.346000+00:00
...
107551	Light Intensity Sensor	259.62	lux	2024-12-04 00:31:50.079000+00:00
107556	Light Intensity Sensor	306.67	lux	2024-12-04 00:31:56.078000+00:00
107561	Light Intensity Sensor	712.89	lux	2024-12-04 00:32:03.081000+00:00
107566	Light Intensity Sensor	364.77	lux	2024-12-04 00:32:09.081000+00:00
107571	Light Intensity Sensor	378.46	lux	2024-12-04 00:32:15.081000+00:00

21499 rows × 4 columns

Figure 2.23: Light Intensity Sensor

b. Normalisasi Data: Agar data berada pada rentang 0 hingga 1, untuk melatih LSTM karena LSTM bekerja lebih baik pada data dengan skala kecil.

```
Code

1 scaler = MinMaxScaler()
2 light_intensity_sensor_df['scaled_value'] =
  ↳ scaler.fit_transform(light_intensity_sensor_df[['value']])
```

c. Membuat Data Sequence: Membagi data menjadi sekuensial (input (x) dan target (y) untuk melatih LSTM.

Code

```
1 def create_sequences(data, sequence_length=50):
2     x, y = [], []
3     for i in range(len(data) - sequence_length):
4         x.append(data[i:i + sequence_length])
5         y.append(data[i + sequence_length])
6     return np.array(x), np.array(y)
7
8     sequence_length = 50
9     x, y = create_sequences(light_intensity_sensor_df['scaled_value'].values,
10                             ↪ sequence_length)
11
12     x = np.expand_dims(x, axis=-1)
13
14     split = int(0.8 * len(x))
15     x_train, x_test = x[:split], x[split:]
16     y_train, y_test = y[:split], y[split:]
```

d. Membuat Model LSTM Light Intensity Sensor

Code

```
1 # Build LSTM Model
2 model = Sequential([
3     LSTM(50, activation='relu', input_shape=(sequence_length, 1)),
4     Dense(1)
5 ])
6 model.compile(optimizer='adam', loss='mse')
7
8 # Tampilkan summary model
9 model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
lstm_8 (LSTM)	(None, 50)	10,400
dense_8 (Dense)	(None, 1)	51

Total params: 10,451 (40.82 KB)

Trainable params: 10,451 (40.82 KB)

Non-trainable params: 0 (0.00 B)

Figure 2.24: Light Intensity Sensor

e. Kode tersebut digunakan untuk melatih model menggunakan data pelatihan (x_train, y_train) dengan 10 epoch dan batch size 32. Untuk Light Intensity Sensor

Code	
1	# Train Model
2	history = model.fit(x_train, y_train, epochs=10, batch_size=32, ↪ validation_data=(x_test, y_test))
3	predicted = model.predict(x_test)
4	predicted = scaler.inverse_transform(predicted)
5	y_test = scaler.inverse_transform(y_test.reshape(-1, 1))

Epoch 1/10	537/537	13s	19ms/step	- loss: 0.0411 - val_loss: 0.0340
Epoch 2/10	537/537	9s	18ms/step	- loss: 0.0335 - val_loss: 0.0345
Epoch 3/10	537/537	7s	13ms/step	- loss: 0.0343 - val_loss: 0.0340
Epoch 4/10	537/537	8s	15ms/step	- loss: 0.0330 - val_loss: 0.0336
Epoch 5/10	537/537	7s	13ms/step	- loss: 0.0336 - val_loss: 0.0337
Epoch 6/10	537/537	6s	12ms/step	- loss: 0.0334 - val_loss: 0.0336
Epoch 7/10	537/537	7s	13ms/step	- loss: 0.0338 - val_loss: 0.0336
Epoch 8/10	537/537	7s	13ms/step	- loss: 0.0331 - val_loss: 0.0335
Epoch 9/10	537/537	7s	13ms/step	- loss: 0.0333 - val_loss: 0.0337
Epoch 10/10	537/537	7s	12ms/step	- loss: 0.0328 - val_loss: 0.0341

Figure 2.25: Hasil Predict Light Intensity Sensor

f. Kode tersebut digunakan untuk visualisasi perbandingan antara nilai aktual dan prediksi Light Intensity Sensor sepanjang waktu (timestamp).

Code

```
1 plt.figure(figsize=(10, 6))
2 plt.plot(light_intensity_sensor_df['timestamp'].iloc[-len(y_test):], y_test,
   ↪ label='Actual')
3 plt.plot(light_intensity_sensor_df['timestamp'].iloc[-len(predicted):],
   ↪ predicted, label='Predicted')
4 plt.xlabel("Timestamp")
5 plt.ylabel("Light Intensity (lux)")
6 plt.title("Actual vs Predicted (Light Intensity Sensor)")
7 plt.legend()
8 plt.show()
```

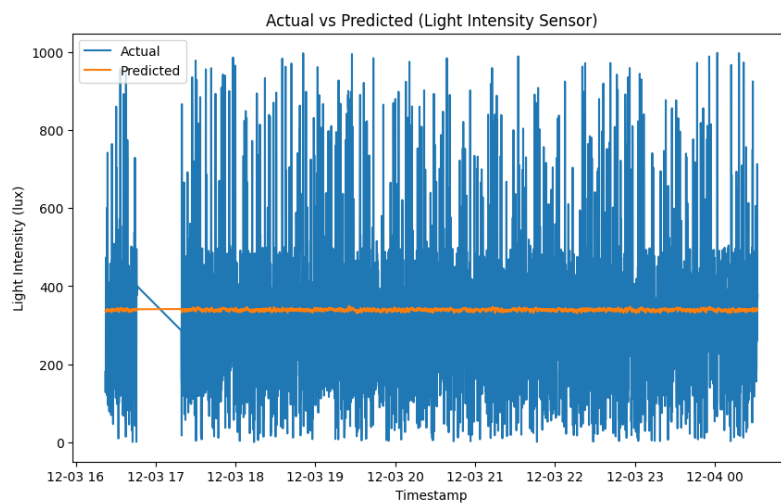


Figure 2.26: Grafik Light Intensity Sensor

Daftar Pustaka

- [1] Yani, Dhita Deviacita A., Helen Sasty Pratiwi, and Hafiz Muhardi. "Implementasi web scraping untuk pengambilan data pada situs marketplace." *JUSTIN (Jurnal Sistem dan Teknologi Informasi)*, vol. 7, no. 4, 2019, pp. 257–262.
- [2] Medium.com, 2024. "Streaming Video Menggunakan Kafka." Available at: <https://medium.com/bisa-ai/streaming-video-menggunakan-kafka-e17aec076bd> [Accessed 27 November 2024].
- [3] Rumah Coding, 2024. "Eksplorasi Data Streaming dengan Apache Kafka dan Python: Memahami dan Menganalisis Data Real-Time." Available at: <https://rumahcoding.co.id/eksplorasi-data-streaming-dengan-apache-kafka-dan-python-memahami-dan-menganalisis-d> [Accessed 27 November 2024]