

Project Report

Analisis Data Cuaca Kota Mumbai Menggunakan Data Science untuk Tren dan Prediksi Cuaca



Anggota Kelompok 3:

Andrew Kurniawan Gianto - 2210101008

Bryant Nehemia Natanael - 2210101022

Ryo Ferdinand - 2210101039

Sibgah Rabbani Kusuma -2210101031

**Program Studi Teknik Informatika
Universitas Pradita, Tangerang.**

2024

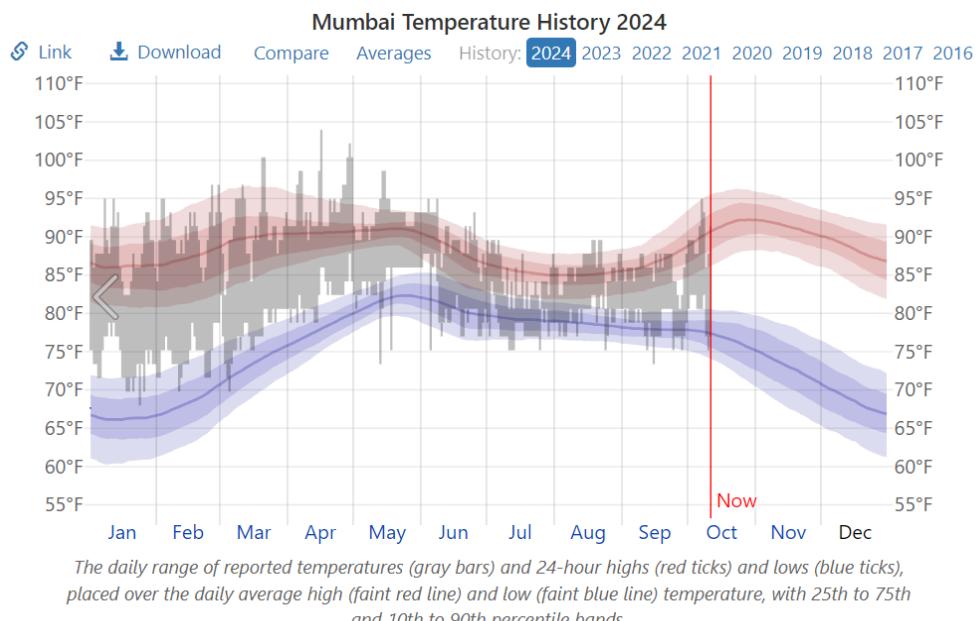
BAB I

PENDAHULUAN

I. Latar Belakang

Di era digital, kemajuan teknologi telah membawa revolusi besar dalam banyak hal, seperti pengumpulan dan analisis data. Dalam konteks cuaca, perangkat seperti sensor cuaca, satelit, dan teknologi Internet of Things (IoT) memungkinkan pengumpulan data cuaca dalam volume besar secara real-time (Heryana, 2013, 30). Dengan adanya perangkat ini, pengumpulan data cuaca menjadi lebih akurat dan presisi. Proses analisis data juga menjadi semakin penting untuk memprediksi fenomena cuaca di masa depan oleh karena adanya teknologi ini serta kemajuan AI dan pembelajaran mesin (Devianto et al., 2020, 20-21).

Data science adalah proses multidisiplin yang melibatkan pengumpulan, pemrosesan, analisis, dan pemodelan data untuk menghasilkan solusi prediktif, otomatisasi, atau wawasan strategis (Ardilla et al., 2022, 2-3)(Kurniawan et al., 2023, 28). Dalam konteks cuaca, data science digunakan untuk mengembangkan model prediktif berdasarkan data historis, mendeteksi anomali cuaca, dan menciptakan sistem prediksi berbasis pembelajaran mesin (machine learning). Metode ini mencakup penggunaan algoritma statistik, teknik pembelajaran mesin, pemrograman, dan pemrosesan data besar (big data) untuk mendapatkan wawasan yang lebih mendalam, proaktif, dan dapat diandalkan (Hartama, 2018, 52)(Kurniawan et al., 2023, 40) (Jayanthi et al., 2022, 1213)(Hamdani et al., 2024, 20-23). Visualisasi data cuaca di Mumbai pada tahun 2024 dapat terlihat pada gambar 1 (Weatherspark, 2024).



Gambar 1. Visualisasi data cuaca di Mumbai pada tahun 2024

Cuaca adalah kondisi atmosfer suatu tempat yang berubah dalam waktu singkat dan dipengaruhi oleh banyak faktor lingkungan (Komal et al., 2019, 352). Faktor utama yang mempengaruhi cuaca meliputi suhu udara, yang menentukan seberapa panas atau dinginnya suatu tempat; kelembapan, yang mencerminkan jumlah uap air di udara; curah hujan, yang mengukur intensitas presipitasi; tekanan udara, yang mempengaruhi pembentukan angin; serta kecepatan angin, yang menunjukkan pergerakan udara (Murmayani & Darwis, 2024, 2-5). Selain itu, tutupan awan dan radiasi matahari juga berperan penting dalam menentukan intensitas sinar matahari yang mencapai bumi dan potensi terjadinya fenomena cuaca ekstrem. Semua faktor ini berinteraksi untuk membentuk pola cuaca yang dapat dianalisis dan diprediksi.

Mumbai, sebagai salah satu kota metropolitan terbesar di India, terletak di pesisir Laut Arab dan sangat dipengaruhi oleh iklim tropis.. Iklim tropis umumnya mempunyai ciri-ciri curah hujan yang tinggi, suhu udara yang tinggi sekitar 20-30°C, dan tekanan udara yang rendah (T et al., 2021, 114-115). Mumbai memiliki dua musim: musim monsun dengan curah hujan tinggi dan musim panas yang panas dan lembap. Suhu rata-rata tahunan di Mumbai berkisar antara 17°C hingga 33°C, dengan puncak musim panas mencapai lebih dari 40°C. Faktor-faktor seperti musim monsun, curah hujan yang tinggi, dan fluktuasi suhu ekstrem menjadikan Mumbai sebagai wilayah yang menarik untuk studi data cuaca (Murmayani & Darwis, 2024, 2-5).

Tren merujuk pada pola atau arah perubahan yang dapat diamati dalam jangka waktu tertentu (Suryanto & Krisbiyantoro, 2018, 294). Dalam konteks analisis cuaca, tren ini memungkinkan untuk identifikasi jika ada kenaikan atau penurunan dalam variabel seperti suhu, curah hujan, atau kelembapan. Tren dibentuk dengan menggunakan teknik data analysis yang melibatkan pengumpulan, pengolahan, dan visualisasi data (Fikri et al., 2022, 737).

Forecasting adalah metode untuk meramalkan kejadian di masa depan berdasarkan data historis yang telah dikumpulkan dan dianalisis (Mulyani et al., 2021, 180). Dalam konteks analisis cuaca, *forecasting* bertujuan untuk memprediksi pola cuaca yang mungkin terjadi di masa depan, seperti rata-rata suhu, curah hujan, atau intensitas angin (Utami & S, 2021). Dalam proses ini, algoritma prediksi dan teknik statistik digunakan untuk menemukan tren dan perubahan yang mungkin terjadi di masa mendatang dengan menggunakan pola dari data sebelumnya.

Dengan latar belakang ini, harapan kami adalah bahwa proyek ini akan memberikan wawasan lebih luas mengenai pembuatan model untuk analisis tren dan prediksi cuaca di suatu wilayah, yang pada akhirnya dapat menjadi referensi bagi peneliti lain dalam mengembangkan proyek terkait. Selain itu, diharapkan analisis ini dapat berkontribusi pada pemahaman yang lebih baik tentang metode forecasting cuaca, sehingga dapat diadaptasi dan diterapkan dalam penelitian-penelitian yang membahas topik serupa di masa mendatang.

II. Motivasi

Motivasi utama dari penelitian ini muncul dari keinginan kuat untuk memanfaatkan pengetahuan dan keterampilan yang telah diperoleh selama studi akademis. Sebagai mahasiswa yang telah mendalami ilmu terkait data science dan cuaca, kami merasa bahwa menganalisis tren cuaca di Mumbai pada tahun 2009 hingga 2024 adalah cara tepat untuk menguji pemahaman kami dan mengaplikasikan teori yang telah dipelajari. Proyek ini tidak hanya menantang kami untuk menyelidiki berbagai variabel cuaca, tetapi juga memberi kesempatan untuk menggabungkan teknologi data dengan fenomena dunia nyata.

Dibuatnya penelitian ini juga dipengaruhi oleh kesadaran akan pentingnya pemahaman yang mendalam terhadap cuaca dan iklim dalam konteks urban yang dinamis seperti Mumbai. Kondisi cuaca yang ekstrem, terutama di wilayah pesisir, berpotensi mempengaruhi kehidupan sehari-hari masyarakat, baik dari sisi ekonomi maupun kesehatan. Oleh karena itu, analisis tren cuaca menjadi krusial dalam mendukung berbagai sektor seperti transportasi, pertanian, dan perencanaan kota.

Selain mitigasi risiko cuaca, penelitian ini juga bertujuan untuk membantu masyarakat yang hidupnya sangat bergantung pada kondisi cuaca, seperti nelayan, petani, dan pekerja sektor informal. Dengan pemahaman yang lebih baik tentang tren cuaca, mereka dapat mengambil keputusan yang lebih tepat untuk melindungi penghidupan mereka dari dampak negatif cuaca yang tidak menentu, sekaligus memanfaatkan peluang cuaca yang menguntungkan.

Dengan penelitian ini, kami berharap dapat memberikan solusi dan wawasan yang berguna bagi para pemangku kepentingan, khususnya dalam pengambilan keputusan terkait mitigasi risiko cuaca ekstrem dan peningkatan kesejahteraan masyarakat yang bergantung pada cuaca. Selain itu, kami juga ingin memberikan kontribusi pada literatur yang relevan dengan cuaca dan iklim, sehingga analisis ini dapat menjadi model bagi studi serupa di masa depan.

III. Tujuan

Berdasarkan latar belakang dan motivasi tersebut, kelompok kami merumuskan tujuan dari proyek kami ini adalah:

1. Mengidentifikasi Tren atau Pola Perubahan Cuaca di Mumbai dari Tahun 2009 hingga 2019

Proyek ini bertujuan untuk melakukan analisis kuantitatif terhadap data cuaca Mumbai dari tahun 2009 hingga 2019 untuk mengidentifikasi tren yang signifikan. Memeriksa variabel-variabel seperti suhu maksimum, suhu minimum, curah hujan bulanan, kelembapan, dan pola angin untuk memahami fluktuasi dan perubahan jangka panjang. Serta mengembangkan visualisasi grafis menggunakan alat seperti Matplotlib untuk menyajikan data dengan cara yang intuitif dan informatif. Menciptakan grafik dan peta yang jelas untuk menunjukkan perubahan iklim dalam bentuk time series line graph

pada data suhu, humiditas, kecepatan angin, pengendapan, index ultraviolet, dan ketutuhan awan, agar lebih mudah dipahami oleh berbagai pemangku kepentingan.

2. *Forecasting/Peramalan Pola dan Kondisi Cuaca di Mumbai dari Tahun 2021 hingga 2024*

Proyek ini bertujuan untuk meramalkan pola cuaca di Mumbai hingga tahun 2024 dengan menggunakan teknik pemodelan statistik dan pembelajaran mesin. Dimulai dengan pengumpulan dan pembersihan data historis cuaca dari tahun 2009 hingga 2019, analisis awal dilakukan untuk mengidentifikasi pola dan anomali. Hasil prediksi ini akan dibandingkan dengan data aktual yang ada untuk mengevaluasi keakuratannya, membantu dalam menentukan kualitas dan validasi model prediksi cuaca dari tahun 2021 hingga 2024.

3. *Peningkatan Kesadaran dan Kontribusi pada Pengetahuan masyarakat*

Proyek ini bertujuan untuk meningkatkan kesadaran masyarakat tentang perubahan iklim dan dampaknya di Mumbai melalui penyusunan materi edukasi yang beragam, termasuk brosur, infografis, dan video. Materi ini dirancang agar mudah dipahami dan relevan dengan berbagai kelompok masyarakat, seperti pelajar dan petani. Selain itu, kampanye kesadaran akan dilaksanakan melalui media sosial untuk menjangkau audiens yang lebih luas, menggunakan testimoni masyarakat lokal untuk meningkatkan dampak emosional. Dokumentasi menyeluruh juga akan dibuat, mencakup metodologi, data, dan temuan, yang dapat diakses oleh peneliti dan akademisi lain sebagai referensi untuk studi serupa. Dengan langkah-langkah ini, proyek ini tidak hanya berfokus pada peningkatan pemahaman masyarakat, tetapi juga berkontribusi secara berkelanjutan pada literatur ilmiah dan basis pengetahuan mengenai analisis cuaca di kawasan urban.

BAB II

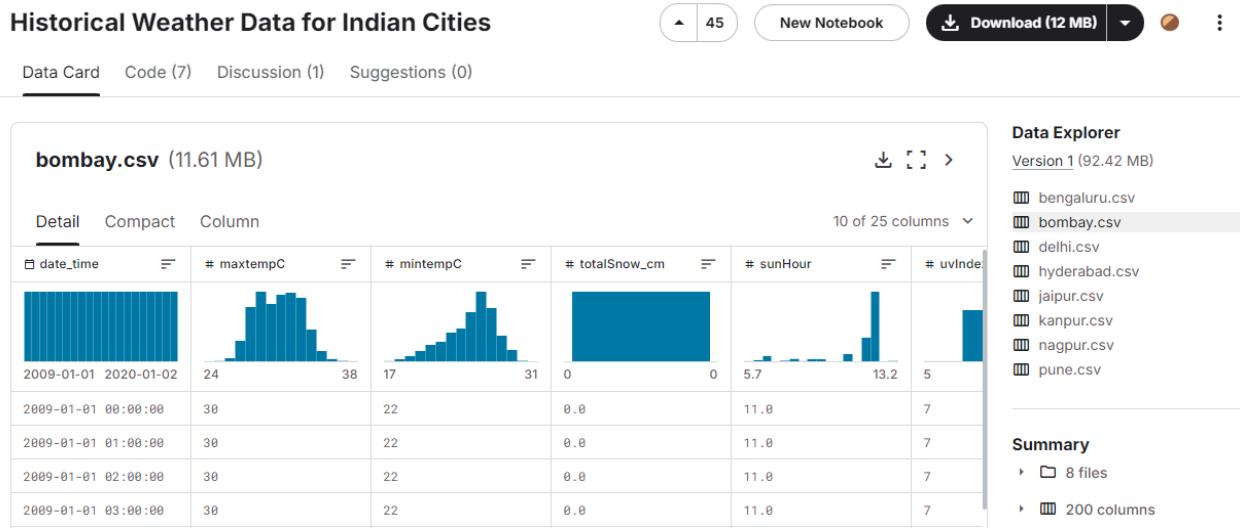
METODE

I. Deskripsi Data

Deskripsi data adalah penjelasan ringkas tentang data yang digunakan dalam suatu analisis atau penelitian, mencakup informasi mengenai struktur, jenis, dan sifat data. Dalam deskripsi data, biasanya dijelaskan elemen-elemen seperti sumber data, jumlah dan tipe variabel, ukuran dataset, dan tingkat kesulitan data. Deskripsi ini membantu pembaca memahami konteks, relevansi, dan kualitas data yang digunakan, serta bagaimana data tersebut mendukung tujuan penelitian atau analisis.

A. Data yang diperoleh dari Kaggle

Dataset yang digunakan dalam proyek ini diperoleh dari Kaggle, sebuah platform yang menyediakan berbagai dataset untuk analisis data, pembelajaran mesin, dan penelitian seperti yang terlihat pada gambar 3. Dataset ini diunggah oleh Soneji (2020) dan berfokus pada data cuaca historis dari Mumbai. Disajikan dalam format CSV (Comma Separated Values), dataset ini termasuk dalam kategori structured data, yang artinya sudah diatur dengan baik dalam baris dan kolom (Soneji, 2020). Setiap kolom mewakili variabel cuaca spesifik seperti suhu, kelembapan, curah hujan, dan kecepatan angin, sedangkan setiap baris mencerminkan satu observasi cuaca pada waktu tertentu.



Gambar 2. Data Cuaca Mumbai dari kaggle

B. Kolom pada dataset

Kolom yang ada pada dataset ada 13 kolom, yaitu: Date_time (Waktu pengambilan data), maxtempC (Suhu maksimum harian dalam derajat Celsius), mintempC (Suhu minimum harian dalam derajat Celsius), sunHour (Jumlah jam matahari bersinar dalam sehari), uvIndex (Indeks ultraviolet yang menunjukkan kekuatan radiasi UV), DewPointC (Titik embun dalam derajat Celsius), FeelsLikeC (Suhu yang dirasakan dalam derajat Celsius) , HeatIndexC (Indeks panas yang memperhitungkan suhu dan kelembapan), cloudcover (Persentase langit yang tertutup awan), humidity (Kelembapan udara dalam persentase), precipMM (Curah hujan dalam milimeter), tempC (Suhu rata-rata harian dalam derajat Celsius), windspeedKmph (Kecepatan angin dalam kilometer per jam). Kolom-kolom tersebut dapat dilihat pada gambar 4.

date_time	maxtempC	mintempC	Humidity	Precip_mm	tempC	windspeedKmph
uvIndex	cloudcover	sunHour	DewPointC	FeelsLikeC	HeatIndexC	-

Gambar 3. Kolom yang ada dalam dataset

C. Jumlah Data untuk project

Dataset ini terdiri dari 96.433 baris, yang mencerminkan pengamatan cuaca yang diambil secara berkala selama lebih dari satu dekade. Jumlah data yang besar ini memberikan dasar yang kuat untuk melakukan analisis tren cuaca, mengidentifikasi pola musiman, dan memprediksi perubahan cuaca di masa depan.

96424	2020-01-0	27	24	0	11	7	7	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96425	2020-01-0	27	24	0	11	7	7	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96426	2020-01-0	27	24	0	11	7	7	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96427	2020-01-0	27	24	0	11	7	7	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96428	2020-01-0	27	24	0	11	7	7	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96429	2020-01-0	27	24	0	11	7	6	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96430	2020-01-0	27	24	0	11	7	1	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96431	2020-01-0	27	24	0	11	7	1	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96432	2020-01-0	27	24	0	11	7	1	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96433	2020-01-0	27	24	0	11	7	1	37	11:38 AM	11:36 PM	07:12 AM	06:12 PM
96434												

Gambar 4. Jumlah data pada data excel

D. Tingkat Kesulitan Data

Dataset ini sebagian besar terdiri dari data numerik, yang membuatnya relatif mudah untuk diproses dan dianalisis. Sebagian besar kolom seperti suhu tempC, maxtempC, mintempC, humidity, date_time, windspeedKmph, dan precipMM, uvIndex, Sunhour, DewpointC, FeelsLikeC, HeatIndexC, Cloudcover berbentuk angka, sehingga cocok untuk berbagai jenis visualisasi dan analisis statistik. Dengan data numerik ini, kita dapat dengan mudah membuat visualisasi

deret waktu (time series) untuk melihat tren perubahan cuaca dari waktu ke waktu.

E. Informasi pada Dataset

Dataset ini berisi 25 kolom dengan total 96.432 entri yang mencakup informasi lengkap terkait kondisi cuaca, seperti parameter meteorologi (maxTemp and minTemp, humidity, pressure, precipMM, windspeedKmph dan winddirDegree), kondisi visual (uvIndex, cloudcover, visibility), dan indikator kenyamanan (FeelsLikeC, HeatIndexC, DewPointC). Selain itu, terdapat data astronomis seperti sunHour, moon_illumination, serta moonrise dan moonset. Dengan cakupan data yang detail, dataset ini dapat digunakan untuk analisis cuaca yang kompleks, seperti peramalan, identifikasi pola cuaca, atau analisis dampak lingkungan berdasarkan parameter meteorologi dan astronomis.

	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	\	
0	2009-01-01 00:00:00	30	22	0.0	11.0	7		
1	2009-01-01 01:00:00	30	22	0.0	11.0	7		
2	2009-01-01 02:00:00	30	22	0.0	11.0	7		
3	2009-01-01 03:00:00	30	22	0.0	11.0	7		
4	2009-01-01 04:00:00	30	22	0.0	11.0	7		
	uvIndex.1	moon_illumination	moonrise	moonset	... WindChillC	\		
0	1	31	10:21 AM	10:20 PM	...	27		
1	1	31	10:21 AM	10:20 PM	...	26		
2	1	31	10:21 AM	10:20 PM	...	26		
3	1	31	10:21 AM	10:20 PM	...	25		
4	1	31	10:21 AM	10:20 PM	...	26		
	WindGustKmph	cloudcover	humidity	precipMM	pressure	tempC	visibility	\
0	11	0	49	0.0	1012	22	10	
1	12	0	50	0.0	1012	22	10	
2	14	0	50	0.0	1012	22	10	
3	15	0	50	0.0	1012	22	10	
4	14	0	49	0.0	1013	22	10	
	winddirDegree	windspeedKmph						
0	20	10						
1	18	11						
2	16	12						
3	14	13						
4	28	12						

Gambar 5. Informasi Dataset

```
[5 rows x 25 columns]
Index(['date_time', 'maxtempC', 'mintempC', 'totalSnow_cm', 'sunHour',
       'uvIndex', 'uvIndex.1', 'moon_illumination', 'moonrise', 'moonset',
       'sunrise', 'sunset', 'DewPointC', 'FeelsLikeC', 'HeatIndexC',
       'WindChillC', 'WindGustKmph', 'cloudcover', 'humidity', 'precipMM',
       'pressure', 'tempC', 'visibility', 'winddirDegree', 'windspeedKmph'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96432 entries, 0 to 96431
```

Gambar 6. Semua Koloms pada Dataset

Data columns (total 25 columns):			
#	Column	Non-Null Count	Dtype
0	date_time	96432 non-null	object
1	maxtempC	96432 non-null	int64
2	mintempC	96432 non-null	int64
3	totalSnow_cm	96432 non-null	float64
4	sunHour	96432 non-null	float64
5	uvIndex	96432 non-null	int64
6	uvIndex.1	96432 non-null	int64
7	moon_illumination	96432 non-null	int64
8	moonrise	96432 non-null	object
9	moonset	96432 non-null	object
10	sunrise	96432 non-null	object
11	sunset	96432 non-null	object
12	DewPointC	96432 non-null	int64
13	FeelsLikeC	96432 non-null	int64
14	HeatIndexC	96432 non-null	int64
15	WindChillC	96432 non-null	int64
16	WindGustKmph	96432 non-null	int64
17	cloudcover	96432 non-null	int64
18	humidity	96432 non-null	int64
19	precipMM	96432 non-null	float64
20	pressure	96432 non-null	int64
21	tempC	96432 non-null	int64
22	visibility	96432 non-null	int64
23	winddirDegree	96432 non-null	int64
24	windspeedKmph	96432 non-null	int64
dtypes: float64(3), int64(17), object(5)			

Gambar 7. Datatype dan Count

F. Outliers dan Missing Value pada Data

Outliers adalah nilai yang menyimpang dari pola distribusi data dan dapat memengaruhi analisis. Untuk mendeteksi outliers pada dataset ini, digunakan metode Rentang Interquartil (IQR) dengan rumus nilai di luar batas atas dan bawah dianggap sebagai outliers. Penanganan outliers dilakukan dengan menghapus dan menjumlahkan seberapa banyak outliers pada kolom seperti maxtempC dengan 336 outliers, precipMM 13,857 outliers, dan sunHour dengan 16,536 outliers.

```
Empty DataFrame
Columns: [date_time, maxtempC, mintempC, humidity, precipMM, tempC, windspeedKmph, uvIndex, sunHour, DewPointC, FeelsLikeC, HeatIndexC]
Index: []
date_time      0
maxtempC      336
mintempC       0
humidity       0
precipMM    13857
tempC        4096
windspeedKmph 2742
uvIndex        0
sunHour     16536
DewPointC     357
FeelsLikeC    3266
HeatIndexC      0
dtype: int64)
```

Gambar 8. Outliers pada Dataset

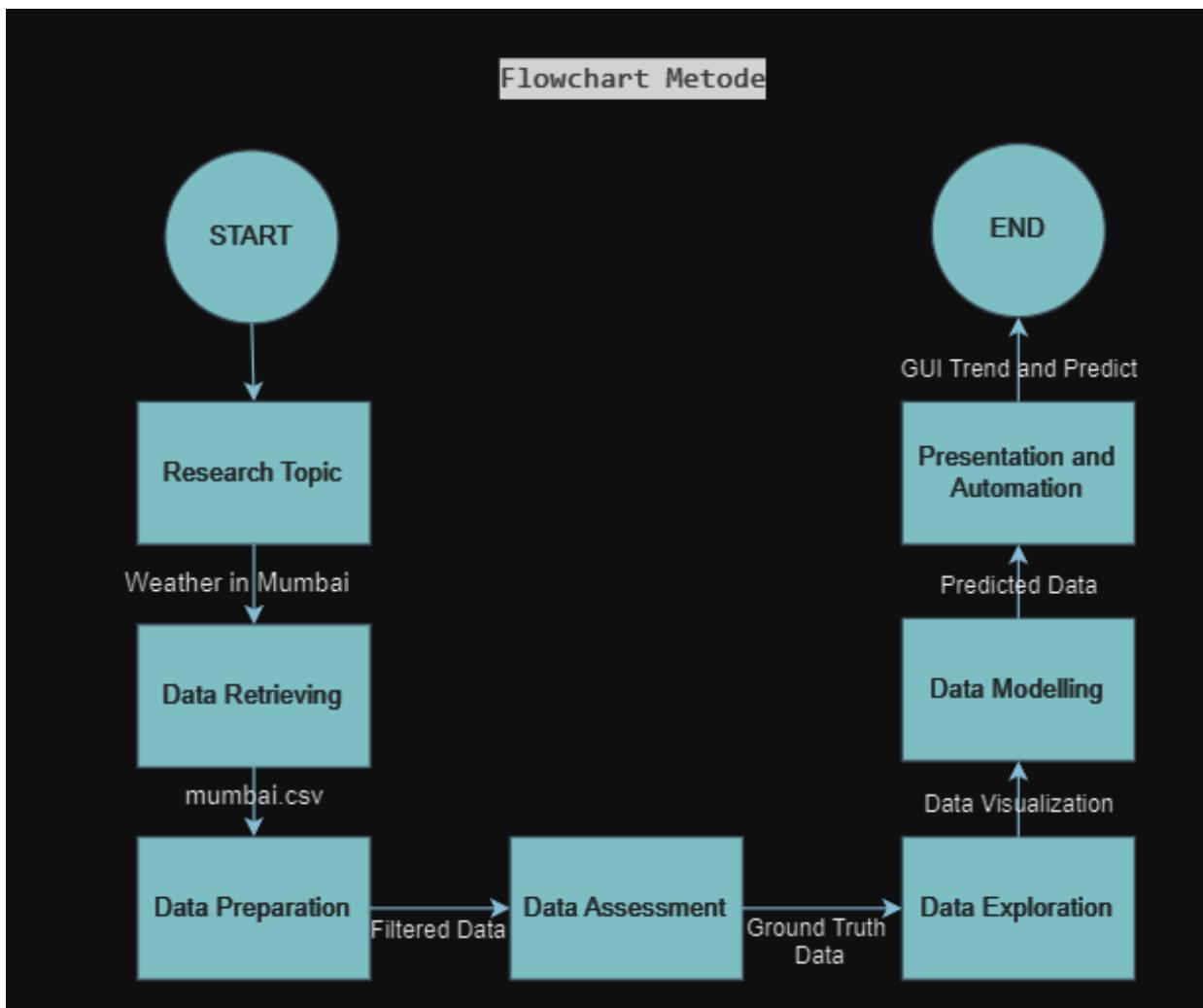
Missing value dapat mengurangi validitas hasil analisis. Identifikasi dilakukan dengan menghitung jumlah nilai hilang per kolom menggunakan metode programatik. Penanganan meliputi imputasi rata-rata (mean) atau median untuk data numerik, serta modus untuk data kategori. Jika proporsi missing value terlalu besar, kolom atau baris terkait dapat dihapus. Dataset menunjukkan kolom suhu tempC, maxtempC, mintempC, humidity, date_time, windspeedKmph, precipMM, uvIndex, Sunhour, DewpointC, FeelsLikeC, dan HeatIndex tidak ada nilai yang hilang. Ini menjadi indikator bahwa dataset yang digunakan sudah cukup bagus.

```
(date_time      0
maxtempC      0
mintempC      0
humidity       0
precipMM      0
tempC         0
windspeedKmph 0
uvIndex        0
sunHour        0
DewPointC     0
FeelsLikeC    0
HeatIndexC    0
dtype: int64,
```

Gambar 9. Missing values pada Dataset

II. Program Python yang telah dibangun

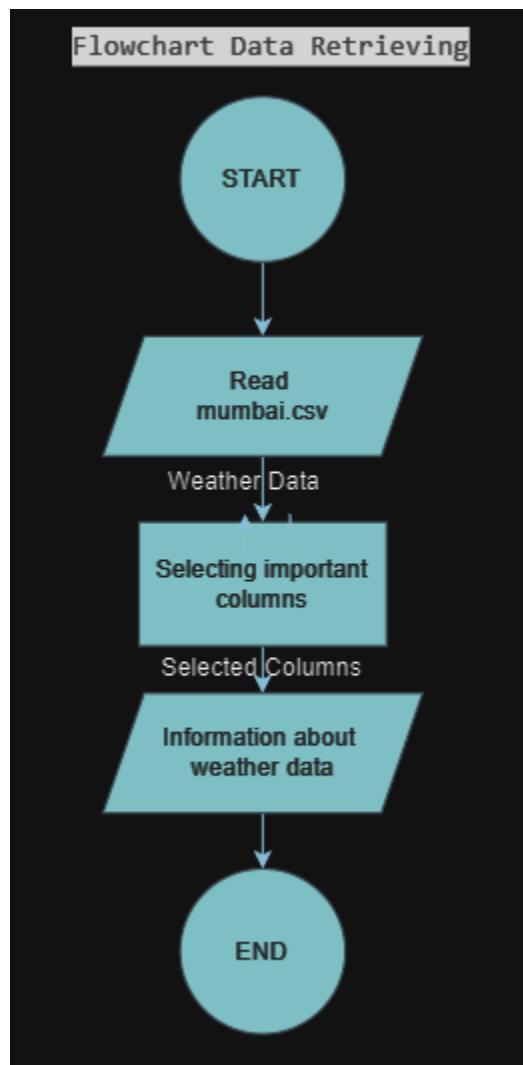
Bagian ini merujuk pada penjelasan mengenai program Python yang telah dibuat untuk memproses dan menganalisis data. Deskripsi ini mencakup rincian tentang bagaimana program tersebut berfungsi, seperti langkah-langkah dalam pengumpulan, pengolahan, visualisasi, modelling, presentation, dan automation data. Bila ada flowchart yang kurang jelas, boleh dizoom agar terlihat lebih jelas dan lebih detail. Berikut adalah flowchart metode yang meliputi 7 tahap penting untuk data science kami:



Gambar 10. Flowchart Keseluruhan Metode

A. Retrieving Data

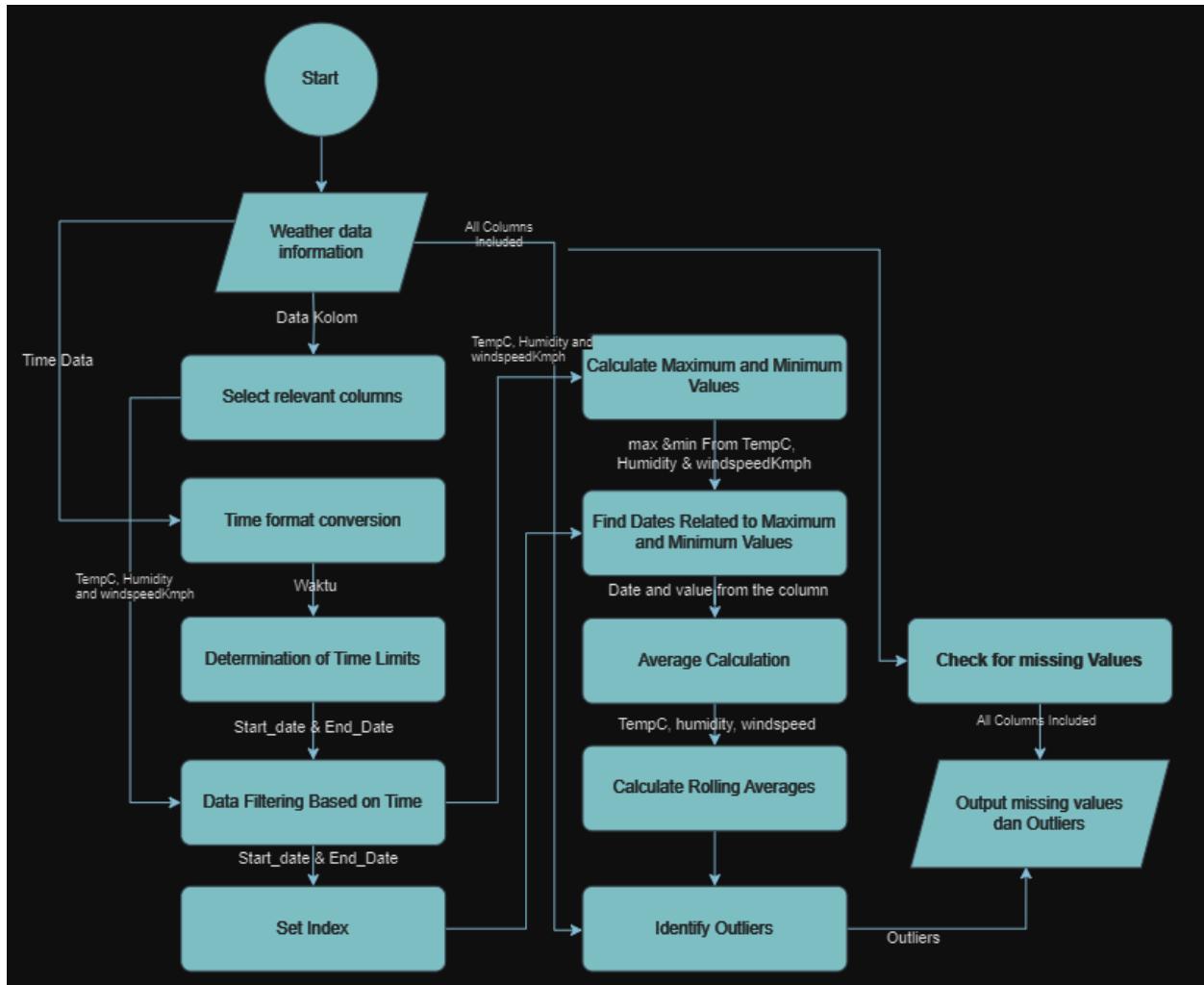
Flowchart dimulai dengan membaca file CSV bernama mumbai.csv dibaca ke dalam DataFrame Pandas. Setelah data dimuat, hanya kolom-kolom tertentu yang terkait dengan cuaca yang dipilih untuk membuat DataFrame baru bernama weather_data. Terakhir, beberapa baris pertama dari data yang telah difilter ditampilkan untuk memastikan bahwa data telah dimuat dan diproses dengan benar, siap untuk analisis lebih lanjut.



Gambar 11. Flowchart Data Retrieving

B. Data Preparation

Flowchart ini dimulai dengan memilih kolom data cuaca yang relevan dan mengonversi kolom waktu (date_time) ke format datetime agar dapat dilakukan pemfilteran berdasarkan waktu. Selanjutnya, data difilter berdasarkan rentang tanggal yang ditentukan (dari 1 Januari 2009 hingga 1 Januari 2020). Setelah itu, dihitung nilai maksimum dan minimum untuk suhu, kelembapan, dan kecepatan angin, serta tanggal terkait dengan nilai-nilai tersebut. Rata-rata bulanan dan mingguan dihitung untuk analisis tren, dan rolling average diterapkan untuk suhu, kelembapan, dan kecepatan angin. Selain itu, dilakukan deteksi nilai yang hilang (missing value) dan pencarian outlier dengan metode Interquartile Range (IQR).



Gambar 12. Flowchart Data Preparation

C. Data Assessment

1. Ground Truth Data

a) WeatherOnline API

Dari gambar 16, dapat ditentukan bahwa menurut Kaggle, data cuaca mumbai yang dipakai untuk proyek ini bersumber dari World Weather Online API, sebuah platform terpercaya dalam penyediaan data cuaca seperti yang terlihat pada gambar 17. World Weather Online API dikenal karena akurasi dan kelengkapan datanya, sehingga menjadikannya pilihan yang ideal untuk pengambilan informasi cuaca yang diperlukan dalam analisis ini. Dataset ini mencakup berbagai variabel meteorologi yang relevan, memungkinkan kami untuk melakukan evaluasi yang mendalam terhadap pola dan kondisi cuaca, serta mendukung pengembangan model prediksi yang lebih efektif.

Content

The dataset was used with the help of the worldweatheronline.com API and the wwo_hist package. The datasets contain hourly weather data from 01-01-2009 to 01-01-2020. The data of each city is for more than 10 years. This data can be used to visualize the change in data due to global warming or can be used to predict the weather for upcoming days, weeks, months, seasons, etc.

Note : The data was extracted with the help of worldweatheronline.com API and I can't guarantee about the accuracy of the data.

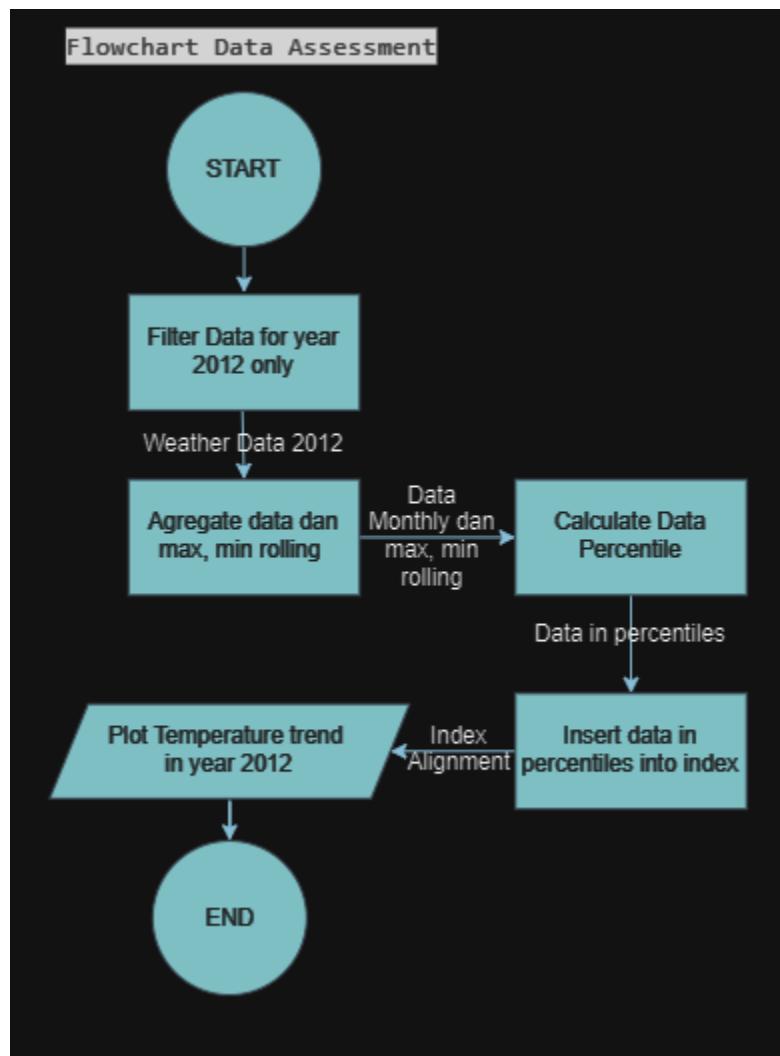
Gambar 13. Sumber menurut kaggle



Gambar 14. World Weather API

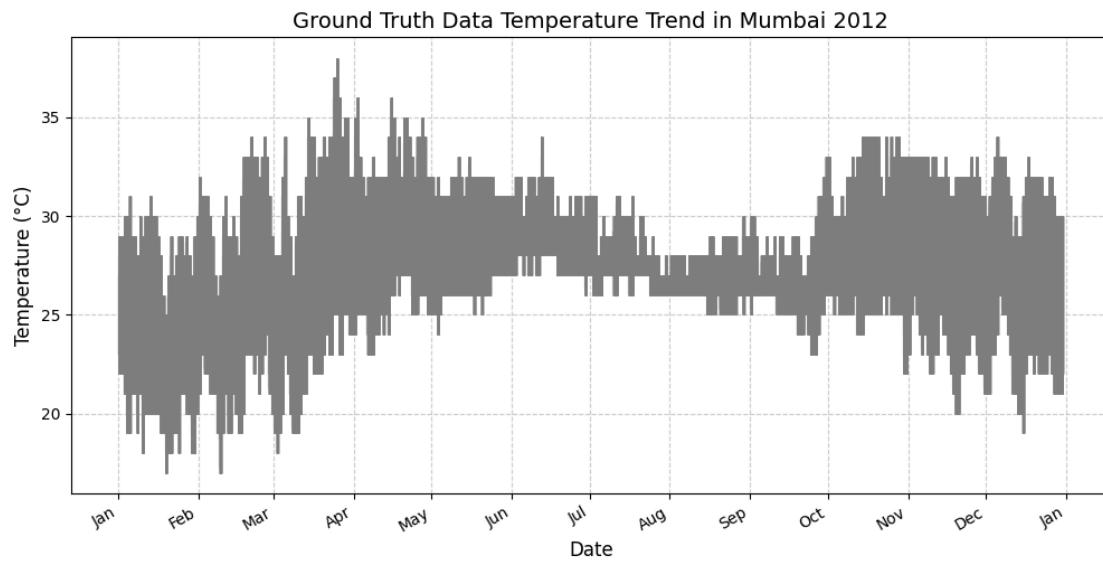
b) Tren Cuaca Tahun 2012

Flowchart dimulai dengan data difilter untuk tahun 2012, kemudian disampling untuk menghitung suhu maksimum, minimum, dan rata-rata bulanan. Lalu, dilanjut dengan menghitung suhu harian yang telah diperhalus menggunakan jendela rolling 7 hari untuk suhu tertinggi dan terendah. Selanjutnya, menghitung persentil suhu (persentil ke-10, 25, 75, dan 90) untuk setiap hari di tahun 2012. Grafik yang dihasilkan menampilkan tren suhu dengan area bayangan yang mewakili berbagai persentil dan garis vertikal yang menunjukkan rentang suhu harian.

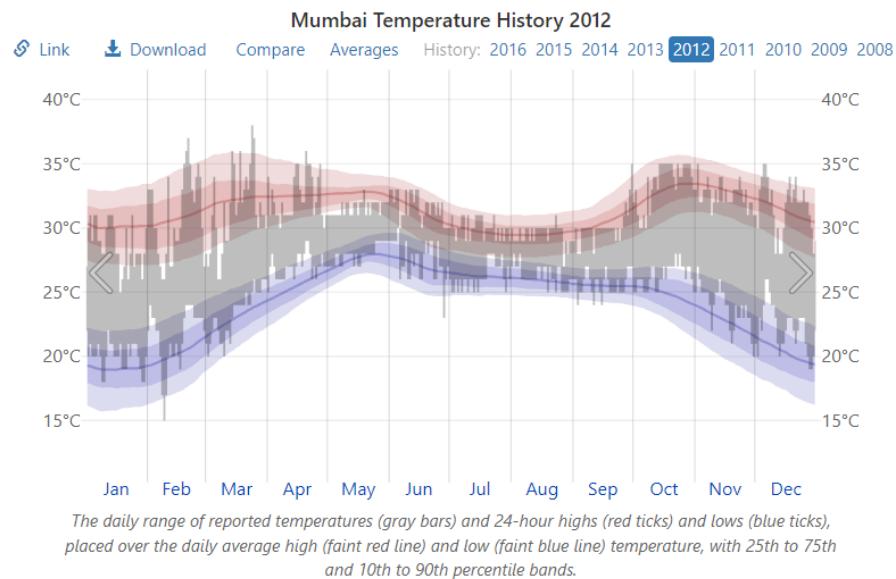


Gambar 14. Flowchart Data Assessment

Setelah kita mendapatkan grafik variasi temperatur cuaca di Mumbai 2012, kita membandingkan grafik tersebut dengan grafik online yang sudah dibuat dengan data asli untuk mencari kebenaran dalam dataset yang dipakai untuk proyek ini. Grafik online seperti yang ada pada gambar 23, diambil dari sebuah website bernama weatherspark, yang merupakan tempat analisis data cuaca yang terpercaya dari banyak wilayah (WeatherSpark, 2012).



Gambar 15. Hasil Output Grafik

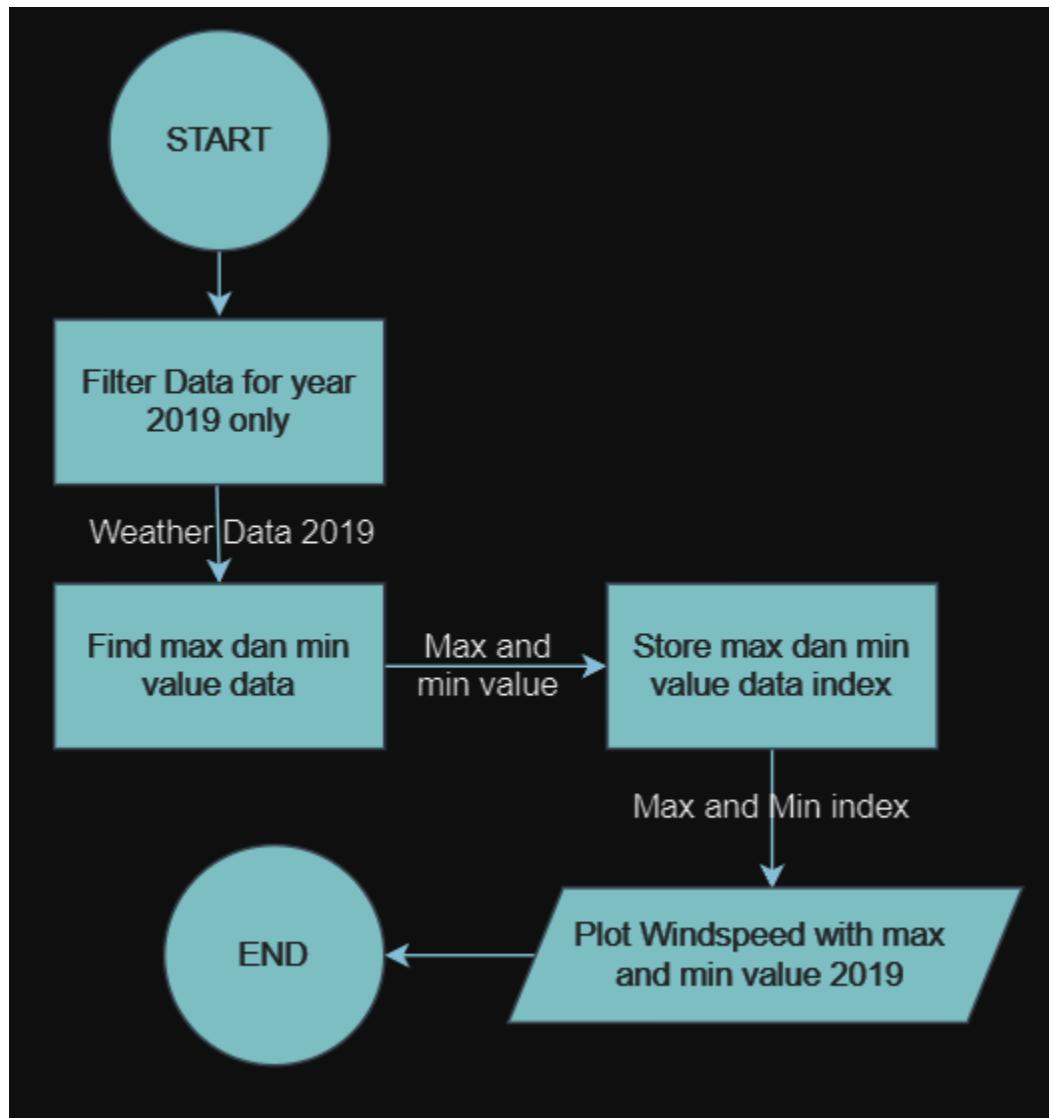


Gambar 16. Grafik Data Cuaca Mumbai 2012 dari WeatherSpark

Setelah membandingkan trend ataupun nilai-nilai temperatur tersebut, dapat kita simpulkan bahwa dataset yang kita pakai untuk proyek ini memiliki kebenaran.

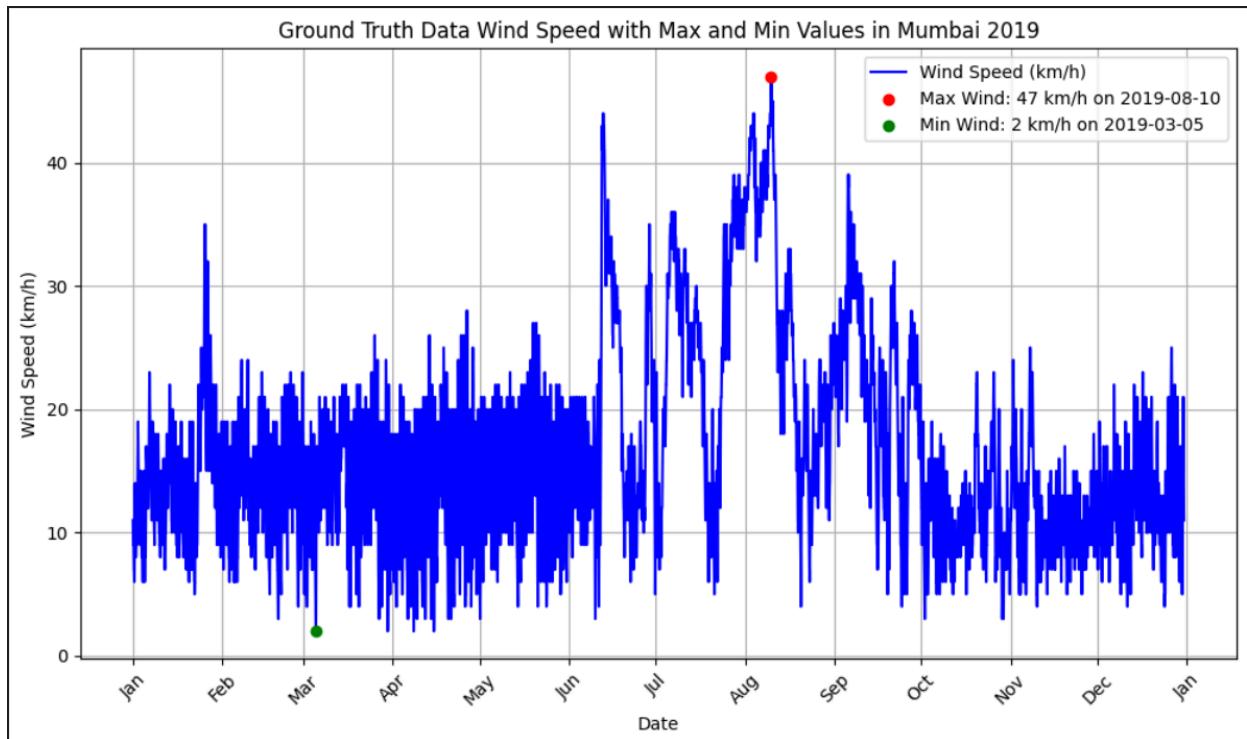
c) **Maksimum data kecepatan angin tahun 2019**

Flowchart dimulai dengan data difilter untuk tahun 2019, kemudian mencari nilai kecepatan angin maksimum dan minimum, serta tanggal terjadinya. Grafik yang akan dihasilkan menunjukkan perubahan kecepatan angin sepanjang tahun dengan sumbu x yang mewakili bulan. Titik maksimum dan minimum ditandai dengan warna merah dan hijau, masing-masing, dan menunjukkan nilai kecepatan angin serta tanggal terjadinya.

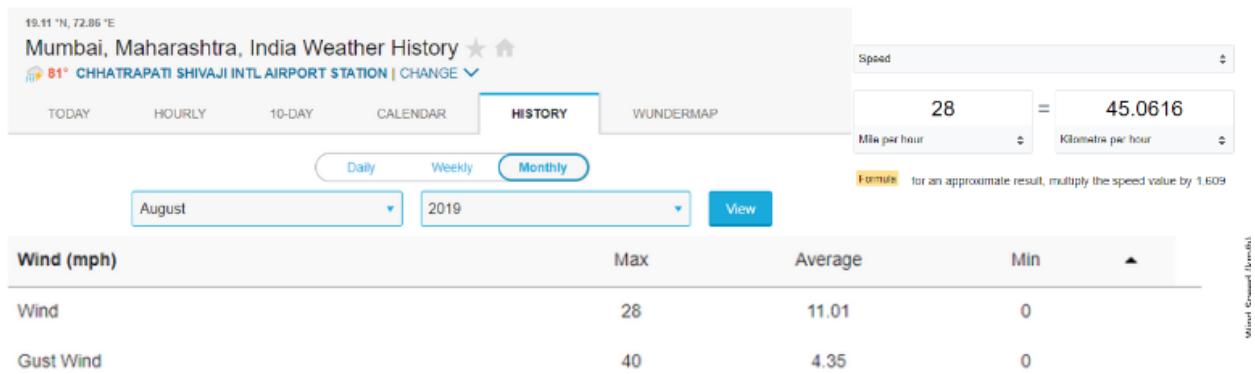


Gambar 17. Flowchart Data Assessment

Setelah kita mendapatkan grafik maksimum dan minimum windspeed di Mumbai 2019, kita membandingkan grafik tersebut dengan tabel online yang sudah dibuat dengan data asli untuk mencari kebenaran dalam dataset yang dipakai untuk proyek ini. Tabel online seperti yang ada pada gambar 28, diambil dari sebuah website bernama wunderground, yang merupakan tempat analisis data cuaca yang terpercaya dari banyak wilayah (Wunderground, 2012).



Gambar 18. Hasil Output Grafik



Gambar 19. Tabel Online Data Cuaca Mumbai 2019 WunderGround

Setelah membandingkan nilai maksimum dan minimum windspeed tersebut, nilainya sangat mirip. Maka dapat kita simpulkan bahwa dataset yang kita pakai untuk proyek ini memiliki kebenaran.

2. Relevansi Data

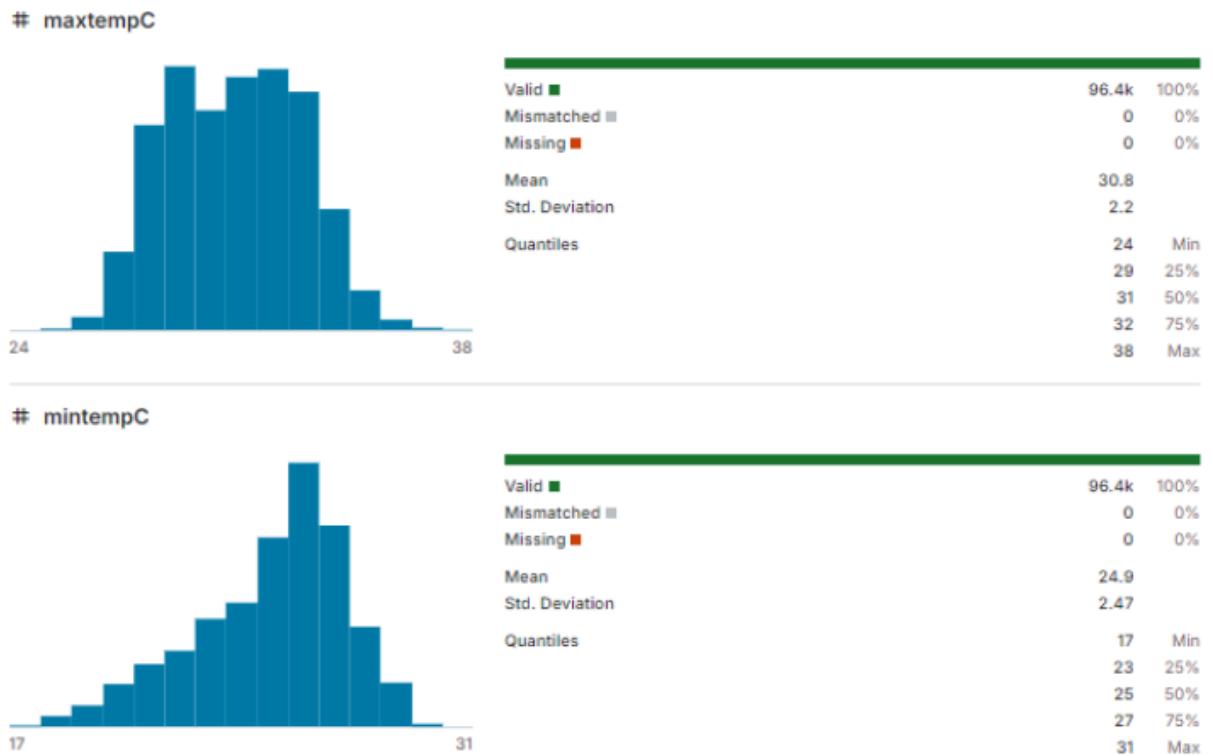
Pada gambar 29, terlihat data yang digunakan dalam proyek ini, seperti maxtempC, mintempC, humidity, precipMM, tempC, windspeedKmph, uvIndex, sunHour, DewPointC, FeelsLikeC, dan HeatIndexC yang sangat relevan untuk analisis tren dan prediksi cuaca. Setiap variabel berperan penting dalam menggambarkan kondisi atmosfer secara menyeluruh, membantu memprediksi fluktuasi suhu, curah hujan, kecepatan angin, radiasi matahari, dan tutupan awan. Kombinasi variabel ini mendukung prediksi cuaca yang lebih akurat dan membantu mengidentifikasi pola cuaca jangka panjang serta risiko cuaca ekstrem.

	date_time	maxtempC	mintempC	humidity	precipMM	tempC	\
0	2009-01-01 00:00:00	30	22	49	0.0	22	
1	2009-01-01 01:00:00	30	22	50	0.0	22	
2	2009-01-01 02:00:00	30	22	50	0.0	22	
3	2009-01-01 03:00:00	30	22	50	0.0	22	
4	2009-01-01 04:00:00	30	22	49	0.0	22	
	windspeedKmph	uvIndex	sunHour	DewPointC	FeelsLikeC	HeatIndexC	
0	10	7	11.0	15	28	28	
1	11	7	11.0	15	27	27	
2	12	7	11.0	15	27	27	
3	13	7	11.0	14	25	26	
4	12	7	11.0	14	26	26	

Gambar 20. Kolom pada data excel

3. Kuantitas Data

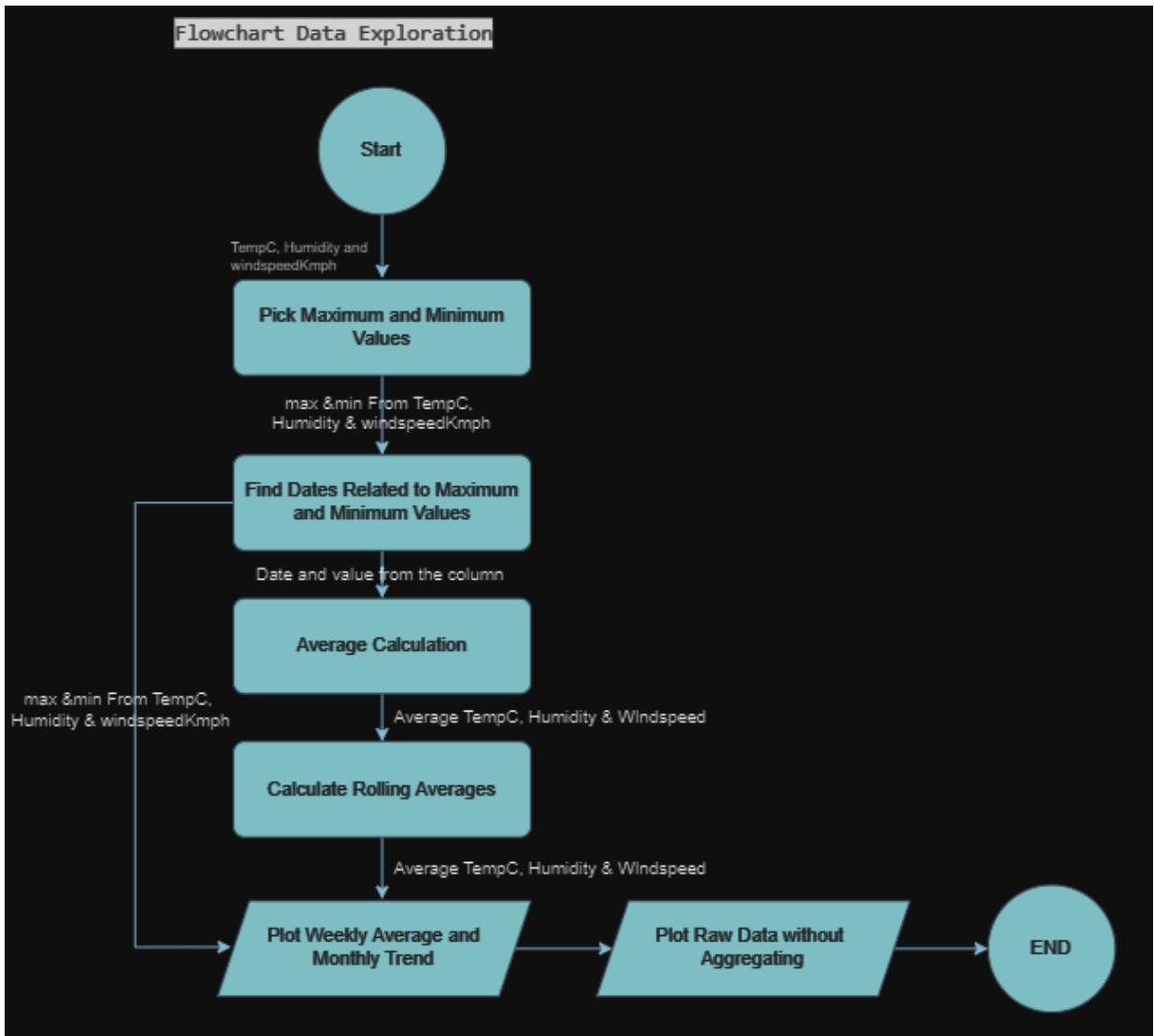
Kuantitas data mencakupi lebih dari 96.000 pengamatan yang diambil selama lebih dari satu dekade seperti yang terlihat pada gambar 30. Jumlah data yang besar ini memberikan dasar yang kuat untuk melakukan analisis tren cuaca, mengidentifikasi pola musiman, dan memprediksi perubahan cuaca di masa depan.



Gambar 21. Kuantitas Data dari Kaggle

D. Data Eksplorasi

Flowchart diawali dengan menghitung rata-rata mingguan dan tren bulanan untuk suhu, kelembapan, dan kecepatan angin serta mencari nilai maksimum dan minimum. Lalu ditampilkan 2 grafik. Pada grafik pertama (Weekly Average and Monthly Trend), ditampilkan suhu mingguan dan rata-rata pergerakan suhu selama 30 minggu. Begitu juga dengan kelembapan dan kecepatan angin yang ditampilkan pada grafik kedua dan ketiga, dengan menambahkan rata-rata pergerakan 30 minggu. Pada grafik kedua (Raw Data without Aggregating), data suhu, kelembapan, dan kecepatan angin harian juga ditampilkan dengan nilai maksimum dan minimum yang disorot, serta rata-rata bulanan untuk setiap variabel.



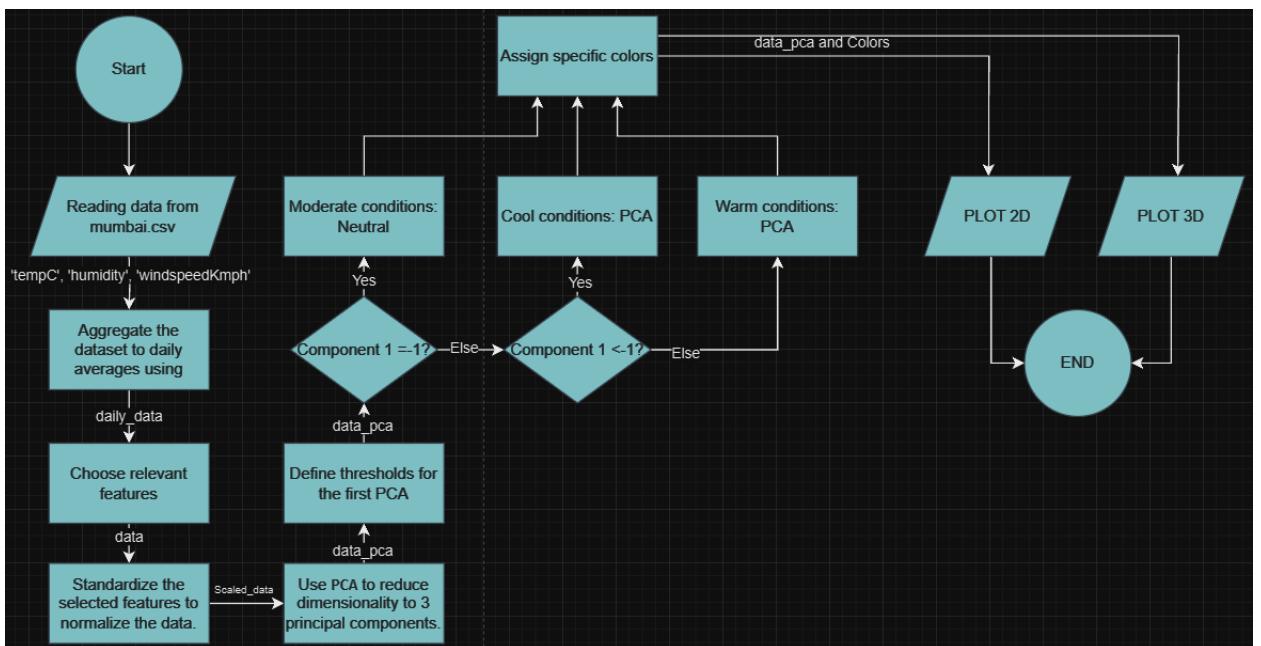
Gambar 22. Flowchart Data Exploration

E. Data Modelling

1. PCA (Principal Component Analysis)

Flowchart ini menggambarkan proses analisis data cuaca dari file mumbai.csv menggunakan Principal Component Analysis (PCA). Proses dimulai dengan membaca data mentah yang mencakup suhu (tempC), kelembapan (humidity), dan kecepatan angin (windspeedKmph). Data kemudian diproses menjadi rata-rata harian untuk mengurangi granularitas dan membentuk dataset daily_data. Fitur relevan dipilih dan data dinormalisasi dengan proses standardisasi untuk membentuk dataset data_scaled. PCA digunakan untuk mengurangi dimensi data menjadi tiga komponen utama yang kemudian dianalisis untuk mengidentifikasi pola cuaca.

Berdasarkan nilai komponen pertama PCA, data dikategorikan menjadi tiga kondisi cuaca: moderat (netral), dingin, dan hangat, dengan warna yang diberikan untuk memudahkan visualisasi. Hasil PCA divisualisasikan dalam dua format, yaitu plot 2D dan 3D, untuk memberikan pemahaman visual tentang pola cuaca di Mumbai. Proses ini menghasilkan wawasan tentang kondisi cuaca yang dapat dianalisis lebih lanjut.

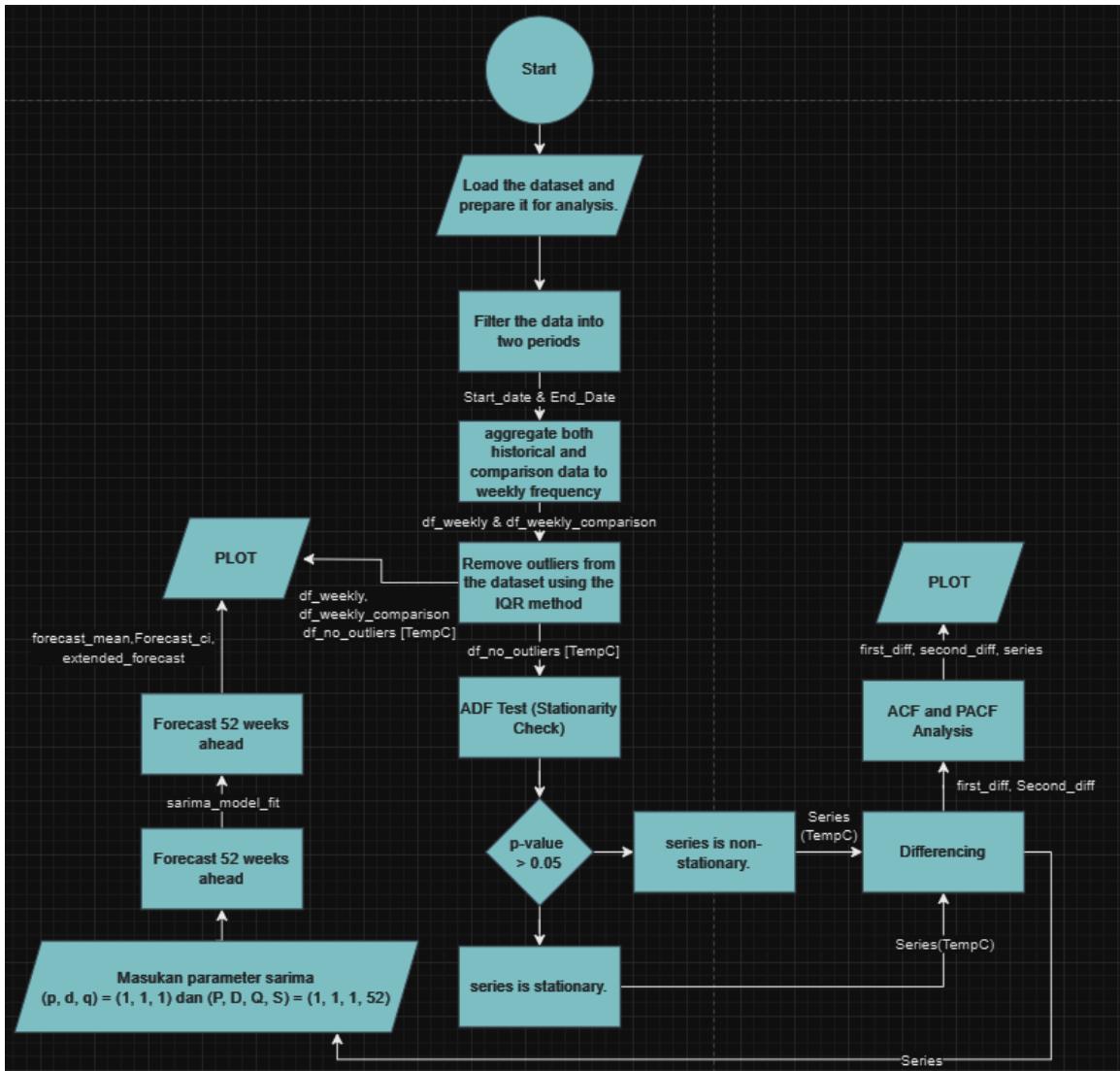


Gambar 23. Flowchart Data Modelling: PCA

2. SARIMA Model

Flowchart ini menggambarkan proses analisis data dan peramalan menggunakan metode SARIMA. Proses dimulai dengan memuat dataset dan mempersiapkannya untuk analisis. Selanjutnya, data difilter menjadi dua periode dan diubah menjadi frekuensi mingguan. Setelah itu, outlier

dihapus dari dataset menggunakan metode IQR. Langkah berikutnya adalah melakukan uji ADF untuk memeriksa stasioneritas data. Jika data tidak stasioner, differencing dilakukan untuk membuatnya stasioner. Setelah data menjadi stasioner, analisis ACF dan PACF dilakukan untuk mengidentifikasi pola dalam data. Hasil differencing kemudian dipetakan, dan peramalan 52 minggu ke depan dilakukan menggunakan model ARIMA. Parameter SARIMA dimasukkan sebagai bagian dari proses peramalan untuk meningkatkan akurasi. Flowchart ini menunjukkan langkah-langkah yang sistematis dalam melakukan analisis data dan peramalan menggunakan metode ARIMA, yang merupakan teknik umum dalam analisis deret waktu.

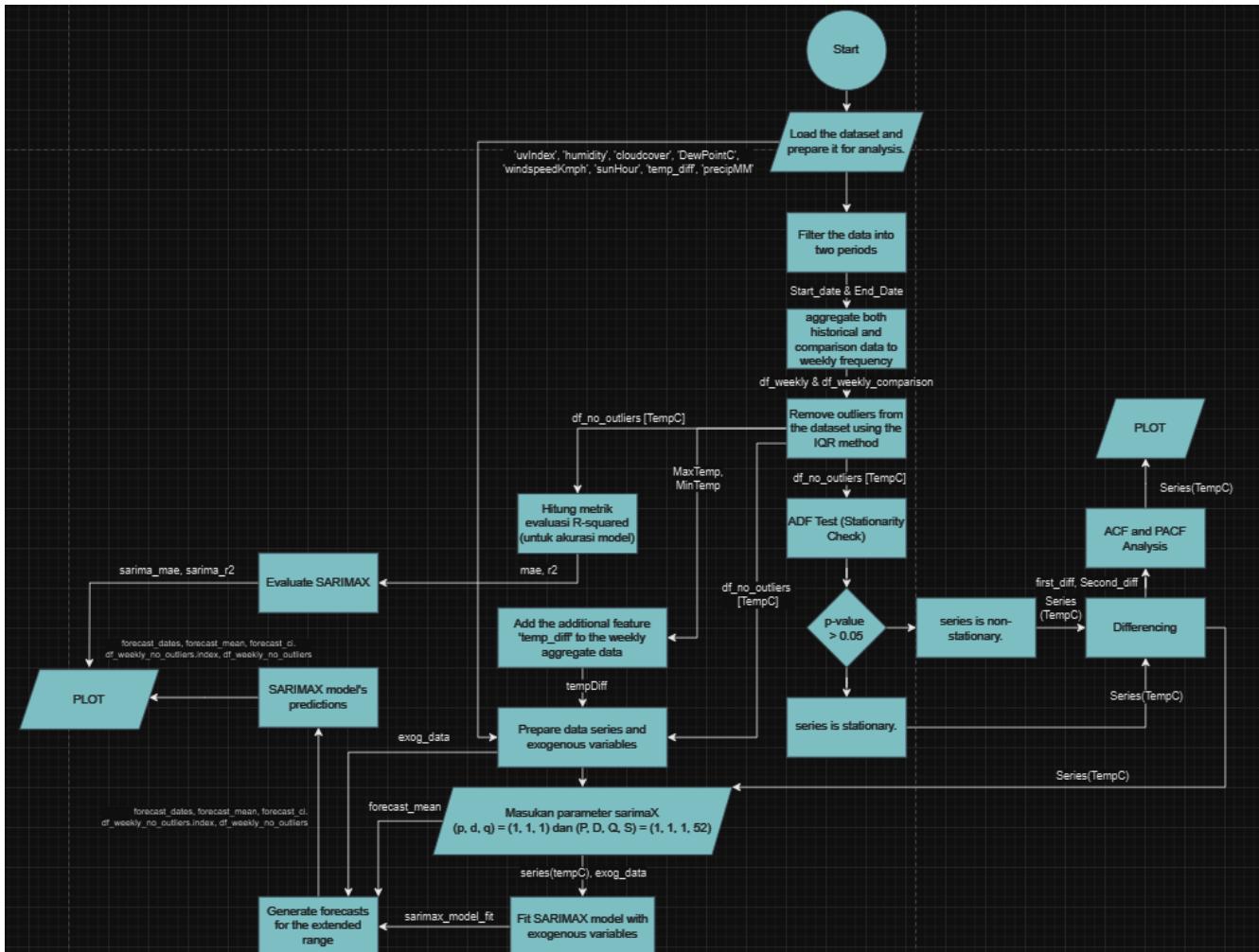


Gambar 24. Flowchart Data Modelling: SARIMA

3. SARIMAX Model

Flowchart ini menjelaskan proses analisis data dan peramalan menggunakan model SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous factors). Proses dimulai dengan memuat dataset, mengubah data menjadi frekuensi mingguan, menghapus outlier menggunakan metode IQR, dan memeriksa stasioneritas data dengan uji ADF. Jika data tidak stasioner, dilakukan differencing hingga data menjadi stasioner. Analisis ACF dan PACF digunakan untuk mengidentifikasi pola dalam data sebelum parameter SARIMAX dimasukkan.

Fitur tambahan ditambahkan ke data mingguan, dan model SARIMAX difit dengan variabel eksogen untuk peramalan. Prediksi dievaluasi menggunakan metrik seperti MAE dan R-squared, lalu dibandingkan dengan data aktual melalui visualisasi. Proses ini memungkinkan analisis mendalam dengan mempertimbangkan komponen musiman dan faktor eksogen, menghasilkan prediksi yang lebih akurat.

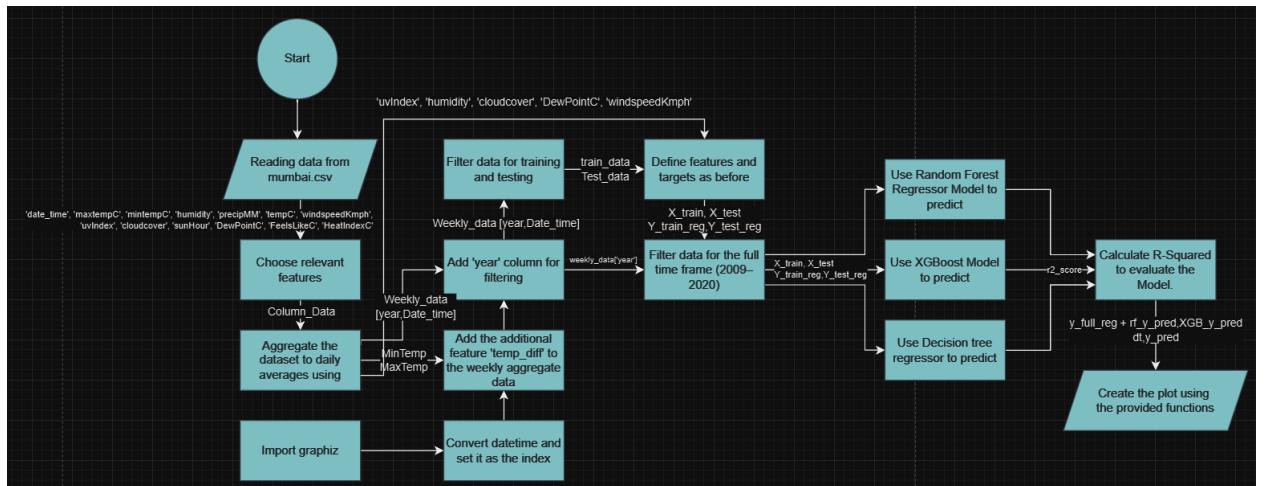


Gambar 25. Flowchart Data Modelling: SARIMAX

4. Random Forest, XGBoost dan Decision tree Regressor

Flowchart ini menggambarkan proses analisis data cuaca dari file "mumbai.csv" untuk memprediksi variabel tertentu menggunakan berbagai model regresi. Proses dimulai dengan membaca data dan memilih fitur relevan seperti suhu maksimum, suhu minimum, kelembapan, kecepatan angin, dan indeks UV. Data kemudian diagregasi menjadi rata-rata harian dan diimpor ke dalam graphiz untuk visualisasi. Selanjutnya, kolom 'year' ditambahkan untuk mempermudah penyaringan data berdasarkan tahun, dan data difilter untuk pelatihan dan pengujian model.

Data yang telah difilter digunakan untuk pelatihan model regresi menggunakan tiga jenis model: Random Forest Regressor, XGBoost, dan Decision Tree Regressor. Hasil prediksi dievaluasi menggunakan nilai R-Squared, dan plot dibuat untuk visualisasi hasil prediksi. Flowchart ini menggambarkan langkah-langkah sistematis dalam analisis data cuaca dan penerapan model regresi untuk menghasilkan prediksi yang akurat, yang penting dalam bidang data science dan machine learning.



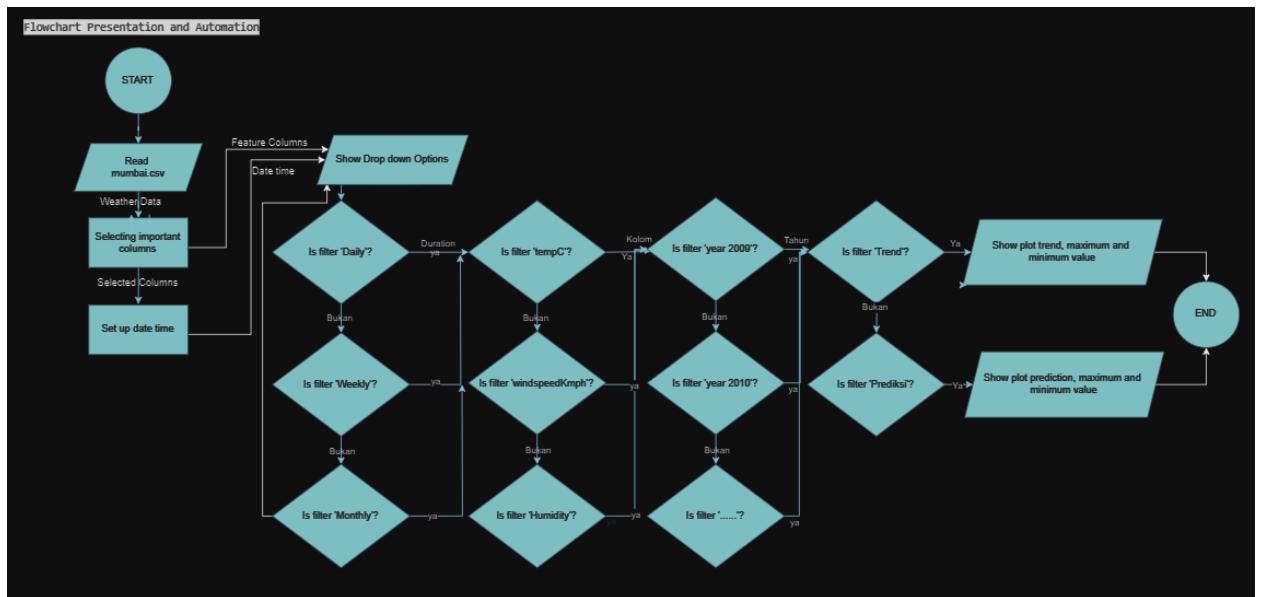
Gambar 25. Flowchart Data Modelling: XGBOOST. Random Forest, Decision Tree Regressor

F. Presentation and Automation

Flowchart ini dirancang untuk memvisualisasikan dan meramalkan data cuaca menggunakan antarmuka grafis pengguna (GUI) yang dibangun dengan Tkinter. Diawali dengan data cuaca diimpor dari file CSV, kemudian diproses dengan memfilter data untuk rentang tanggal yang ditentukan (2009-2019) dan meresample data untuk durasi waktu seperti bulanan, mingguan, atau harian.

Pengguna dapat memilih parameter cuaca (misalnya suhu, kecepatan angin, atau kelembaban), durasi (bulanan, mingguan, atau harian), dan tahun untuk analisis menggunakan menu dropdown di GUI. Selanjutnya, akan memplot parameter cuaca yang dipilih seiring waktu, menyoroti nilai maksimum dan minimum beserta tanggalnya.

Adapun penerapan model SARIMAX (Seasonal AutoRegressive Integrated Moving Average dengan variabel eksogen) untuk meramalkan parameter cuaca, khususnya suhu dan kelembaban. Model SARIMAX dilatih menggunakan data mingguan, dan outlier dihapus dari dataset dengan menggunakan metode Interquartile Range (IQR). Ramalan yang dihasilkan, dengan kelembaban yang dibatasi pada rentang 0 hingga 100, ditampilkan di GUI, di mana pengguna dapat memvisualisasikan tren yang diprediksi bersama dengan interval kepercayaan. Pengguna dapat beralih antara menampilkan tren cuaca dan data ramalan melalui tombol di GUI, dengan plot yang secara dinamis disematkan dalam jendela Tkinter.



Gambar 26. Flowchart Presentation and Automation

BAB III

HASIL DAN PEMBAHASAN

A. Retrieving Data

Output berikut adalah struktur data, nama kolom-kolom, informasi tentang dataset yang kami gunakan, dan juga menampilkan 5 baris pertama dalam dataset. Dataset terdiri dari beberapa kolom seperti date_time untuk merekam waktu pengambilan data, maxtempC dan mintempC untuk suhu maksimum dan minimum, humidity untuk kelembapan udara, serta precipMM yang mencatat curah hujan dalam mm, tempC untuk suhu rata-rata, windspeedKmph untuk menunjukkan kecepatan angin rata-rata (km/jam), uvIndex untuk mengukur tingkat radiasi ultraviolet, sunHour untuk menunjukkan jumlah jam sinar matahari pada hari tersebut.

	date_time	maxtempC	mintempC	humidity	precipMM	tempC	\
0	2009-01-01 00:00:00	30	22	49	0.0	22	
1	2009-01-01 01:00:00	30	22	50	0.0	22	
2	2009-01-01 02:00:00	30	22	50	0.0	22	
3	2009-01-01 03:00:00	30	22	50	0.0	22	
4	2009-01-01 04:00:00	30	22	49	0.0	22	
	windspeedKmph	uvIndex	sunHour	DewPointC	FeelsLikeC	HeatIndexC	
0	10	7	11.0	15	28	28	
1	11	7	11.0	15	27	27	
2	12	7	11.0	15	27	27	
3	13	7	11.0	14	25	26	
4	12	7	11.0	14	26	26	

Gambar 27. Output Data

B. Data Preparation

Output dibawah menunjukkan hasil identifikasi outlier pada dataset selama fase data preparation dalam proses data science, yang mencakup fitur-fitur seperti maxtempC, mintempC, humidity, precipMM, tempC, dan lainnya. Beberapa fitur memiliki jumlah outlier signifikan, seperti precipMM dengan 13.857 outlier, tempC dengan 4.076 outlier, dan sunHour dengan 16.536 outlier, sementara fitur lain seperti date_time, mintempC, humidity, uvIndex, dan cloudcover tidak memiliki outlier. Identifikasi ini bertujuan untuk mendeteksi nilai ekstrem yang dapat memengaruhi kualitas analisis atau pemodelan, sehingga diperlukan langkah penanganan seperti transformasi data atau penghapusan outlier agar dataset menjadi lebih representatif.

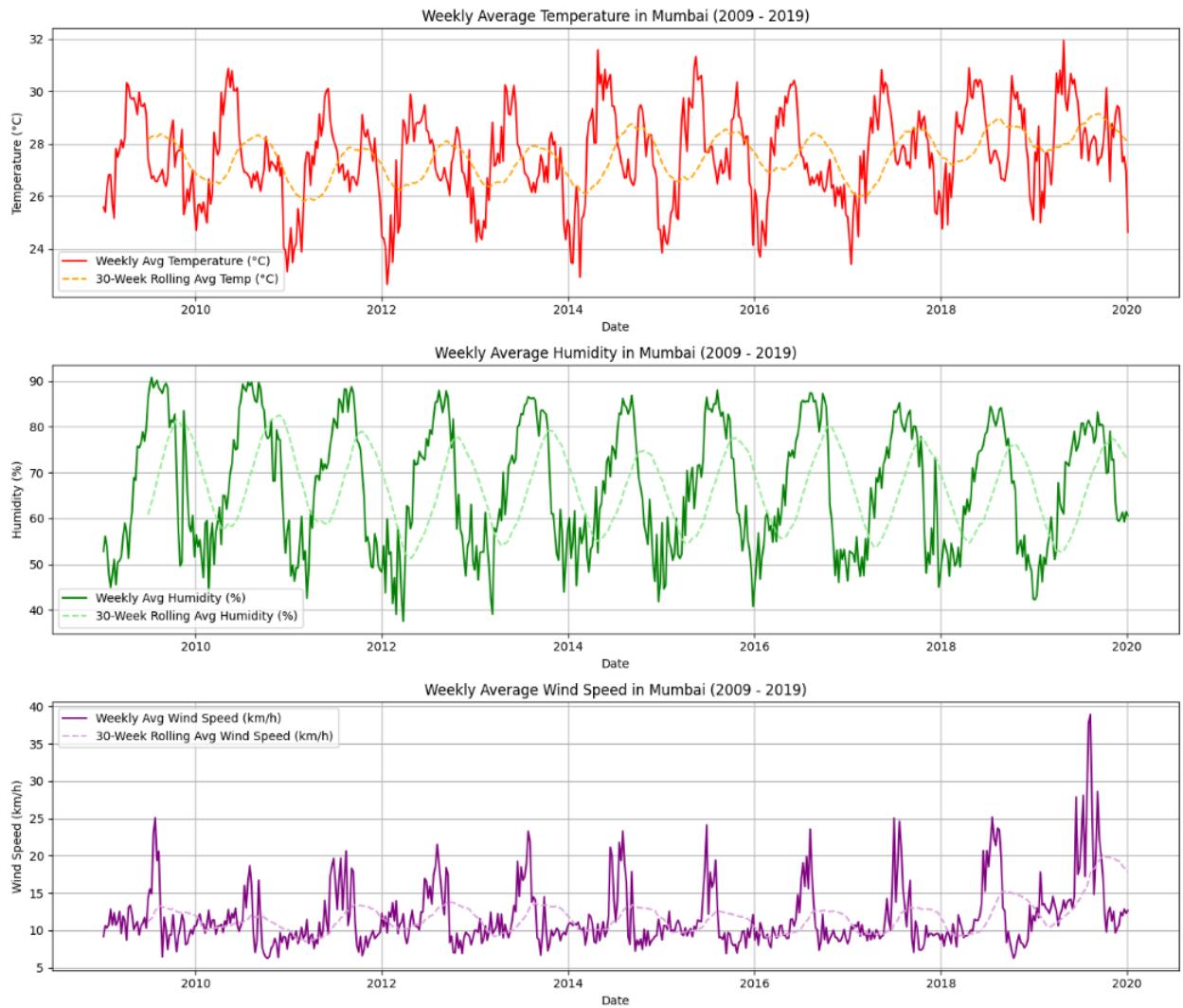
(date_time	0
maxtempC	0
mintempC	0
humidity	0
precipMM	0
tempC	0
windspeedKmph	0
uvIndex	0
cloudcover	0
sunHour	0
DewPointC	0
FeelsLikeC	0
HeatIndexC	0
dtype: int64,	
date_time	0
maxtempC	336
mintempC	0
humidity	0
precipMM	13857
tempC	4076
windspeedKmph	2742
uvIndex	0
cloudcover	0
sunHour	16536
DewPointC	357
FeelsLikeC	3266
HeatIndexC	0
dtype: int64)	

Gambar 28. Data missing value dan outlier

C. Data Eksplorasi

a. Weekly average temperature in Mumbai

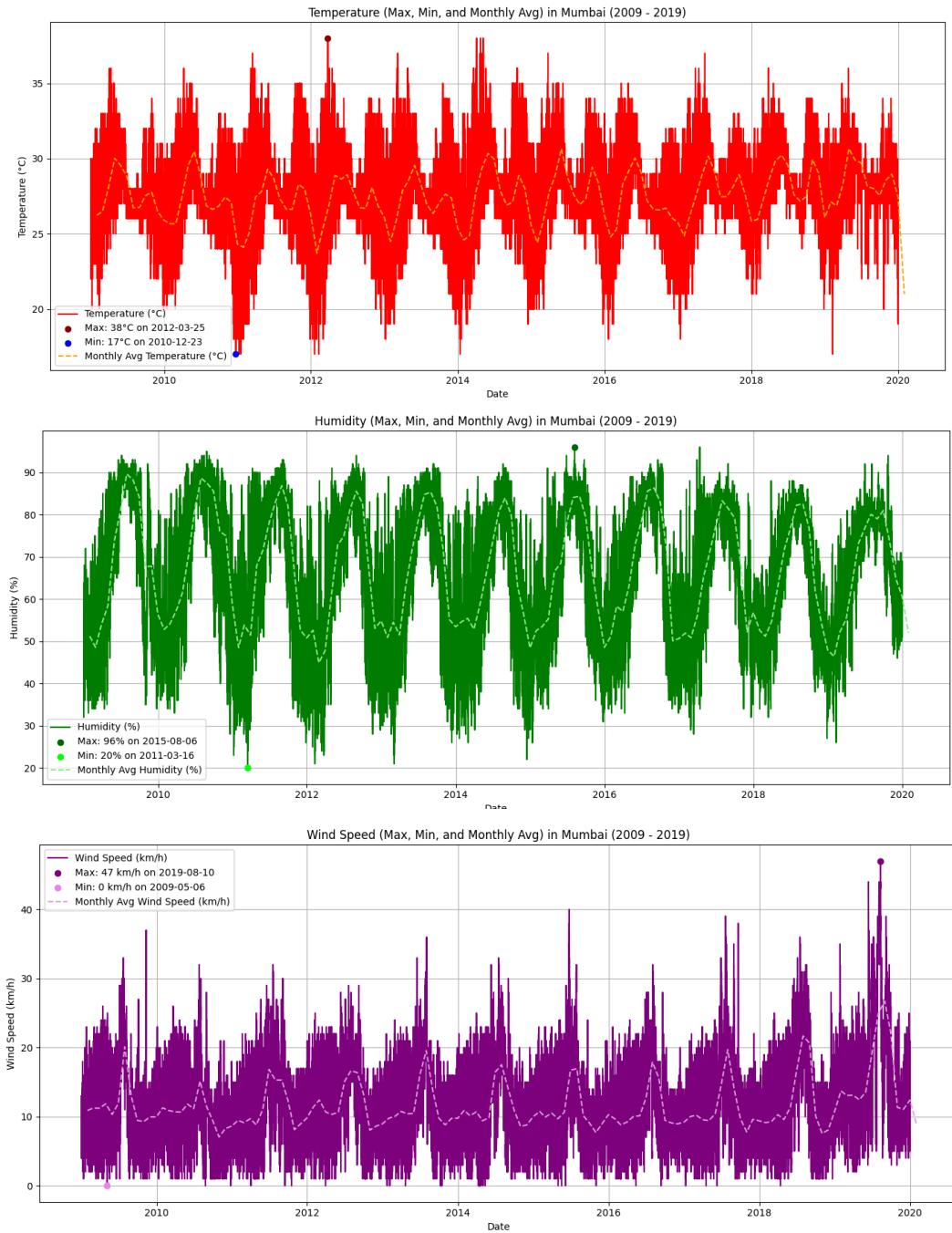
Plot berikut menunjukkan suhu rata-rata mingguan di Mumbai (garis merah) dengan rolling average 30 minggu (garis oranye putus-putus), memperlihatkan fluktuasi musiman dan tren jangka panjang dari 2009 hingga 2019. Selain suhu, plot serupa juga dibuat untuk kelembapan (garis hijau) dan kecepatan angin (garis ungu), menggunakan kolom tempC, humidity, dan windspeedKmph untuk membangun plot ini.



Gambar 29. Plot Grafik Weekly Trend

b. Raw Data without aggregating

Plot pertama menunjukkan suhu di Mumbai dari 2009–2019 tanpa agregasi, dengan garis merah untuk suhu rata-rata, titik biru untuk suhu maksimum, dan titik hitam untuk suhu minimum setiap bulan. Plot ini menggunakan tiga kolom data: suhu maksimum (maxtempC), suhu minimum (mintempC), dan suhu rata-rata (tempC).

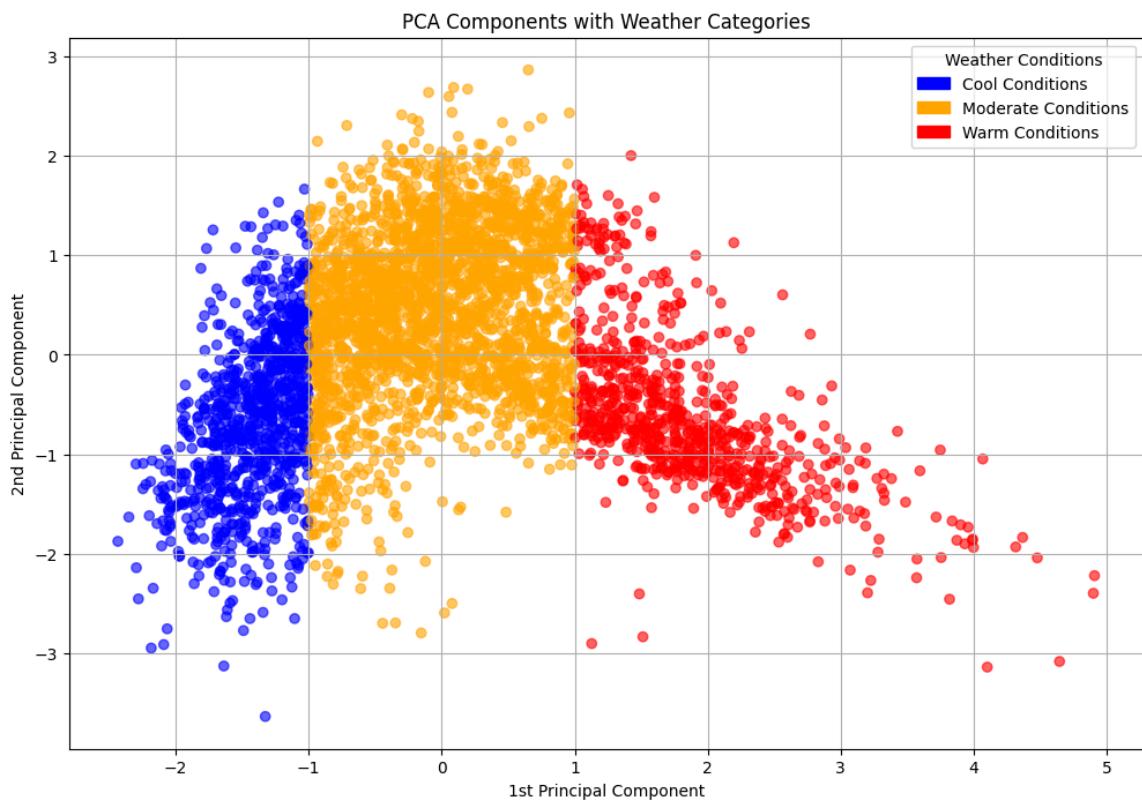


Gambar 30. Plot Grafik Raw Data

D. Data Modelling

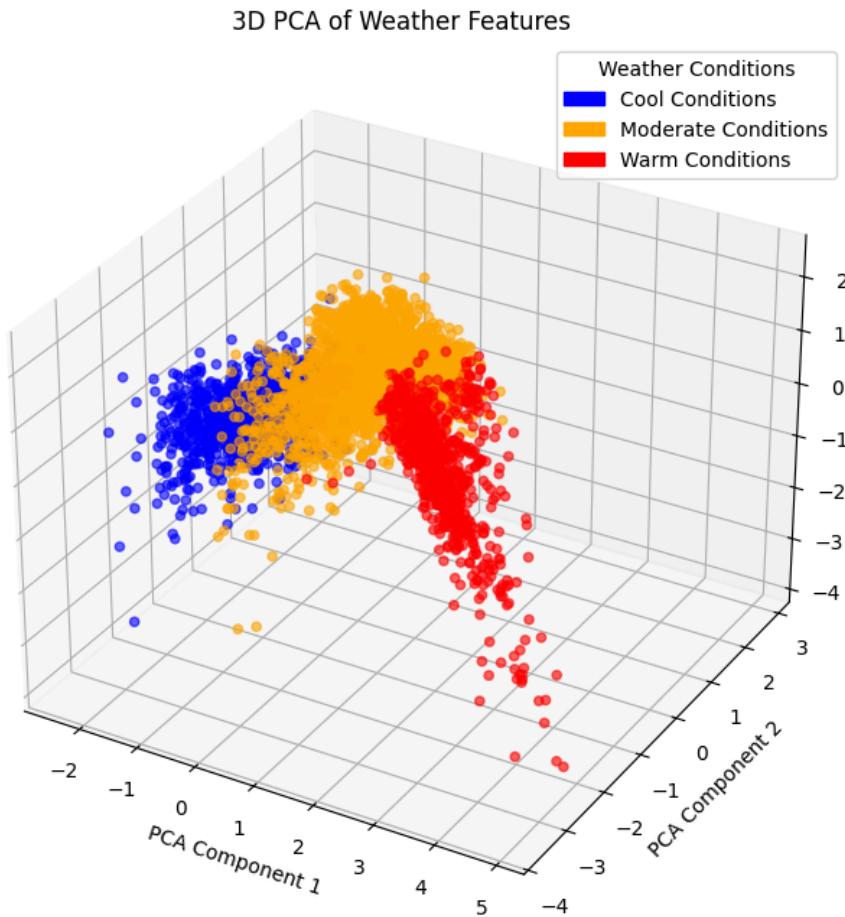
a. PCA (Principal Component Analysis)

Plot PCA (Principal Component Analysis) 2D ini menunjukkan distribusi data cuaca di Mumbai berdasarkan dua komponen utama hasil reduksi dimensi. Data dikelompokkan ke dalam tiga kelas cuaca: Cool Conditions (biru), Moderate Conditions (kuning), dan Warm Conditions (merah). Komponen utama pertama (x-axis) dan kedua (y-axis) merepresentasikan sebagian besar variasi dalam data, sehingga memudahkan visualisasi pola dan klaster cuaca. Pemisahan warna mencerminkan perbedaan suhu dan karakteristik cuaca antar kategori. PCA membantu menyederhanakan analisis dengan tetap mempertahankan informasi penting.



Gambar 31. 2D PCA plot

Gambar dibawah menunjukkan hasil PCA 3D, terdapat tiga komponen utama yang menangkap variasi data terbesar. Titik-titik merepresentasikan sampel data, dengan warna biru, oranye, dan merah menunjukkan kategori kondisi cuaca (Cool, Moderate, dan Warm). Distribusi ini memperlihatkan bahwa PCA berhasil memisahkan kategori secara jelas berdasarkan pola variasi utama, di mana Cool Conditions terkumpul di sisi kiri, Moderate di tengah, dan Warm di sisi kanan, membantu memahami pola tersembunyi dalam data dan mempermudah analisis data.



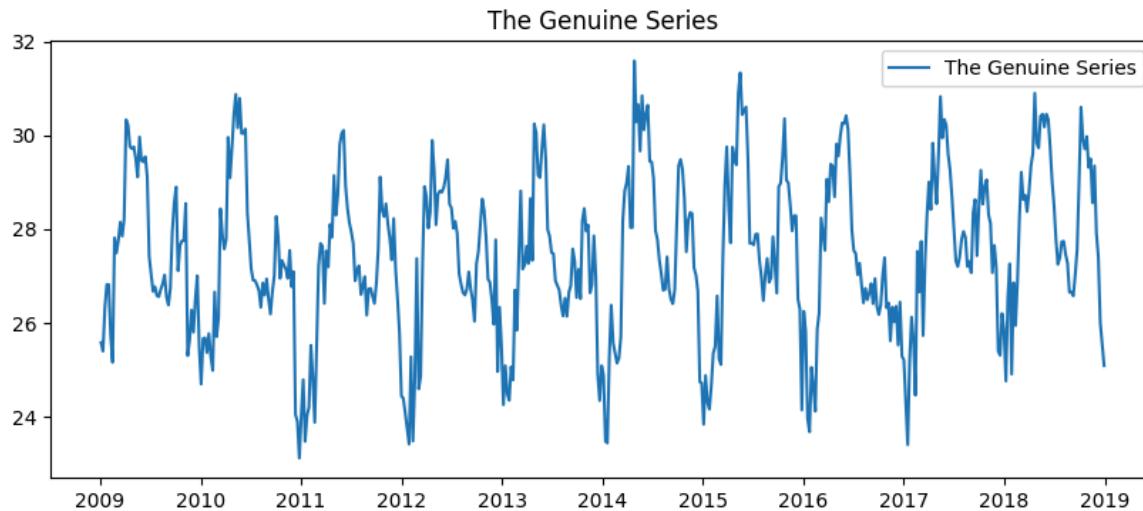
Gambar 32. 3D PCA plot

b. SARIMA Model

Pada model SARIMA, p adalah order AR (autoregressive), d adalah order differencing untuk stasioneritas, q adalah order MA (moving average), dan s adalah periode musiman, yang ditentukan berdasarkan pola musiman data. Gambar dibawah menunjukkan *time series*, yang menggambarkan fluktuasi data dari tahun 2009 hingga 2019. Garis biru mewakili tren perubahan nilai data dari waktu ke waktu, dengan rentang nilai sekitar 24 hingga 32. Pola plot menunjukkan naik-turun yang signifikan dan berulang, menandakan adanya siklus atau variasi musiman.

Plot berikut menunjukkan suhu rata-rata mingguan di Mumbai dari tahun 2009 hingga 2019, yang ditampilkan sebagai "The Genuine Series" dengan garis biru. Sumbu X mewakili rentang waktu dari tahun 2009 hingga 2019, sedangkan sumbu Y menunjukkan suhu rata-rata dalam Celsius, berkisar antara 24°C hingga 32°C. Garis biru pada plot menggambarkan fluktuasi suhu mingguan yang

memperlihatkan pola musiman tahunan dengan puncak dan lembah yang konsisten setiap tahun. Tren ini menunjukkan adanya pola naik turun suhu yang berulang secara berkala, mencerminkan variasi musiman yang stabil selama periode 10 tahun tersebut.



Gambar 33. Genuine Series

Output menunjukkan hasil Uji Augmented Dickey-Fuller (ADF) pada data deret waktu suhu rata-rata mingguan di Mumbai dari tahun 2009 hingga 2019. Uji dilakukan pada data asli, data setelah differencing orde pertama, dan differencing orde kedua. Berdasarkan hasil uji, semua p-value berada jauh di bawah tingkat signifikansi 0,05, menunjukkan bahwa data pada semua tahap analisis sudah stasioner. Dengan demikian, pola musiman yang stabil selama periode 10 tahun tetap terlihat dalam data suhu mingguan tersebut.

```

ADF Test on Original Series:
ADF Statistic: -8.59611002884203
p-value: 7.070496817625539e-14
Series is stationary

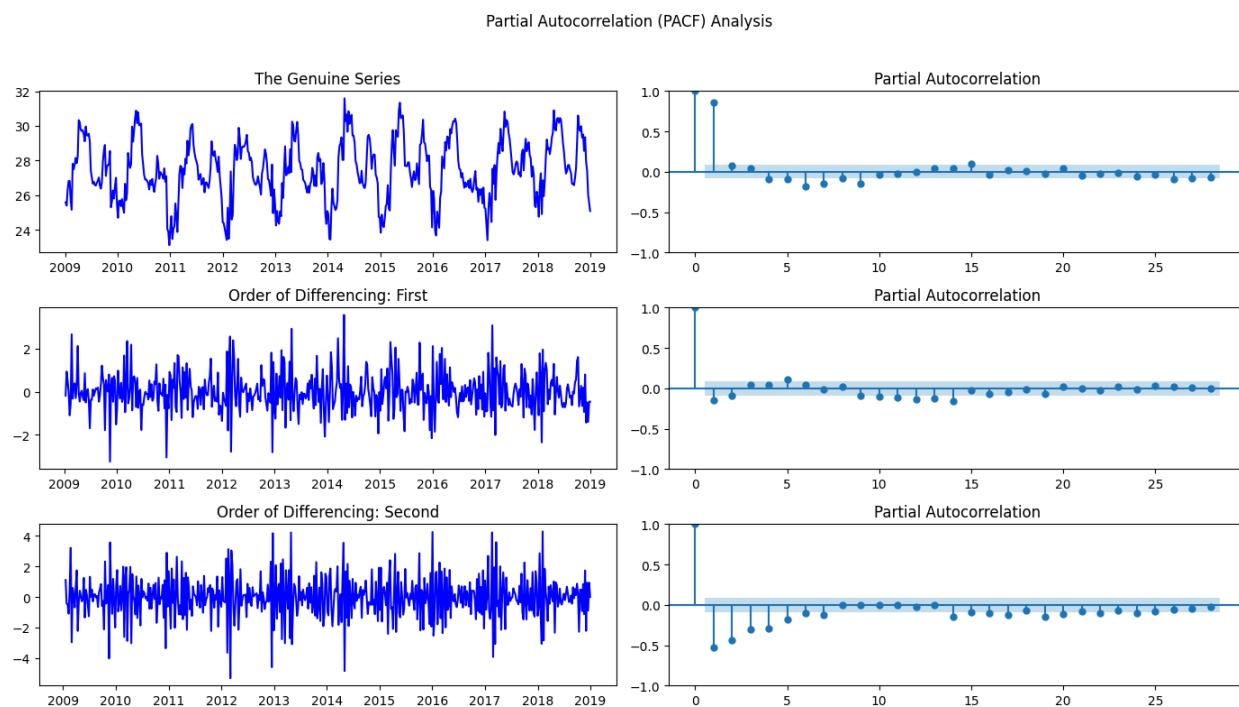
ADF Test on First Order Differencing:
ADF Statistic: -10.292953878557688
p-value: 3.530821790388833e-18
Series is stationary

ADF Test on Second Order Differencing:
ADF Statistic: -10.24963266229337
p-value: 4.521246549232831e-18
Series is stationary

```

Gambar 34. ADF Test output

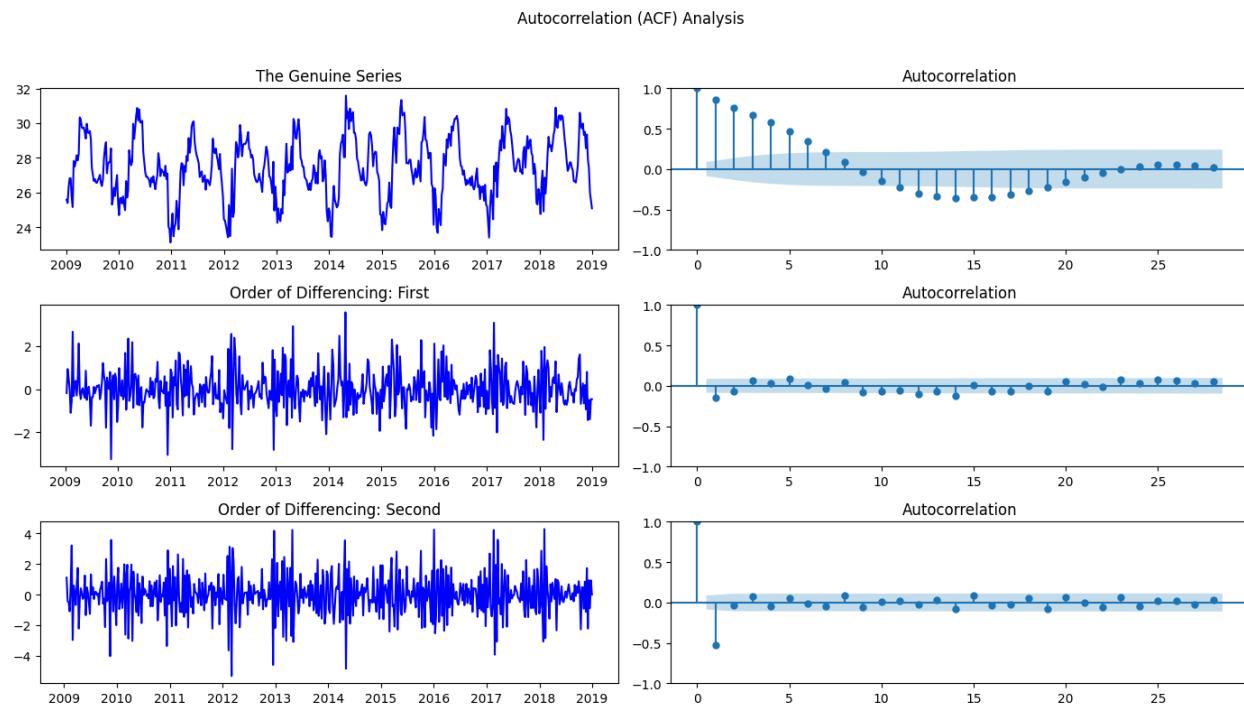
Plot dibawah menunjukkan analisis deret waktu suhu rata-rata mingguan di Mumbai dari tahun 2009 hingga 2019 menggunakan genuine series dan hasil differencing, disertai grafik Partial Autocorrelation Function (PACF). Pada baris pertama, grafik deret waktu asli (*The Genuine Series*) menunjukkan pola musiman yang jelas dengan fluktuasi tahunan. Setelah differencing orde pertama, pola musiman hilang, menghasilkan data yang lebih stasioner. Differencing orde kedua menunjukkan hasil serupa dengan pola lebih datar, mengindikasikan kestasioneran yang lebih kuat. Grafik PACF di setiap tahap menunjukkan penurunan signifikan pada lag awal, menguatkan bahwa data menjadi lebih stasioner setelah proses differencing, yang penting untuk pemodelan deret waktu seperti ARIMA.



Gambar 34. PACF Plot

Plot berikut menampilkan analisis deret waktu suhu rata-rata mingguan di Mumbai dari tahun 2009 hingga 2019 melalui tiga tahap, yaitu genuine series dan hasil differencing orde pertama serta kedua. Grafik genuine series (baris pertama) menunjukkan pola musiman yang konsisten dengan fluktuasi tahunan, mengindikasikan bahwa data belum stasioner. Pada differencing orde pertama (baris kedua), pola musiman berkurang signifikan, menghasilkan data yang lebih mendekati kestasioneran. Differencing orde kedua (baris ketiga) semakin memperkuat kestasioneran dengan pola yang lebih datar. Grafik Autocorrelation Function (ACF) di sebelah kanan setiap tahap menunjukkan penurunan autokorelasi yang signifikan pada lag awal, menegaskan proses differencing

berhasil menghilangkan komponen non-stasioner, yang penting untuk analisis lanjutan seperti pemodelan ARIMA.



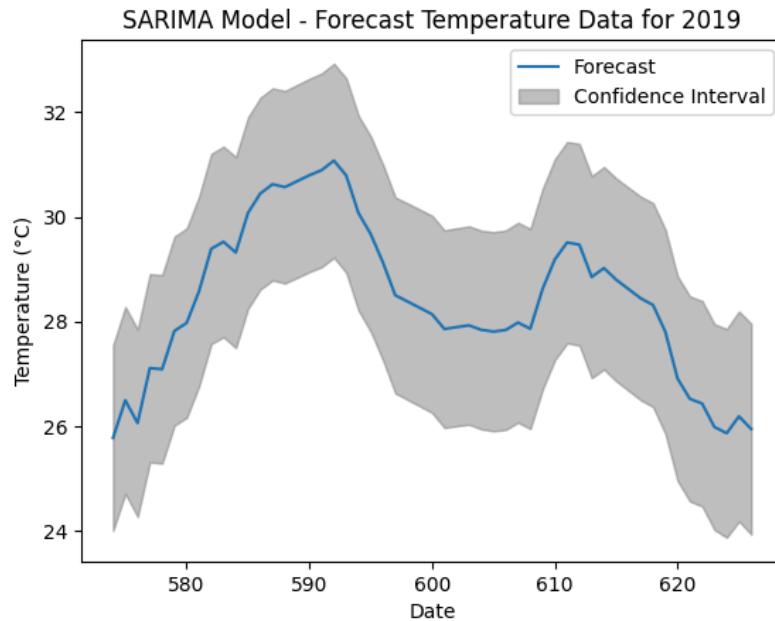
Gambar 35. ACF Plot

Tabel SARIMA menunjukkan hasil estimasi model SARIMA(1, 1, 1)x(1, 1, 1, 52) pada data suhu (tempC) dengan total observasi 572. Fokus pada p-value di tabel menunjukkan bahwa beberapa parameter signifikan secara statistik. Koefisien **ar.L1** (0.000), **ma.L1** (0.000), dan **ma.S.L52** (0.000) memiliki p-value < 0.05, menunjukkan pengaruh signifikan dalam model. Sebaliknya, **ar.S.L52** memiliki p-value sebesar 0.747 (> 0.05), menandakan tidak signifikan secara statistik. Hal ini berarti bahwa komponen musiman AR pada lag 52 tidak memberikan kontribusi signifikan terhadap model. Evaluasi p-value ini penting dalam menentukan parameter yang perlu dipertahankan atau dieliminasi untuk meningkatkan akurasi model SARIMA.

SARIMAX Results						
Dep. Variable:	tempC	No. Observations:	572			
Model:	SARIMAX(1, 1, 1)x(1, 1, 1, 52)	Log Likelihood	-660.974			
Date:	Thu, 12 Dec 2024	AIC	1331.947			
Time:	07:34:22	BIC	1353.207			
Sample:	0 - 572	HQIC	1340.276			
Covariance Type:	opg					
coef	std err	z	P> z	[0.025	0.975]	
ar.L1	0.4036	0.041	9.757	0.000	0.323	0.485
ma.L1	-0.9558	0.016	-59.274	0.000	-0.987	-0.924
ar.S.L52	0.0191	0.059	0.322	0.747	-0.097	0.135
ma.S.L52	-0.8966	0.102	-8.829	0.000	-1.096	-0.698
sigma2	0.6411	0.051	12.654	0.000	0.542	0.740
Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	15.37			
Prob(Q):	0.94	Prob(JB):	0.00			
Heteroskedasticity (H):	1.10	Skew:	0.09			
Prob(H) (two-sided):	0.53	Kurtosis:	3.82			

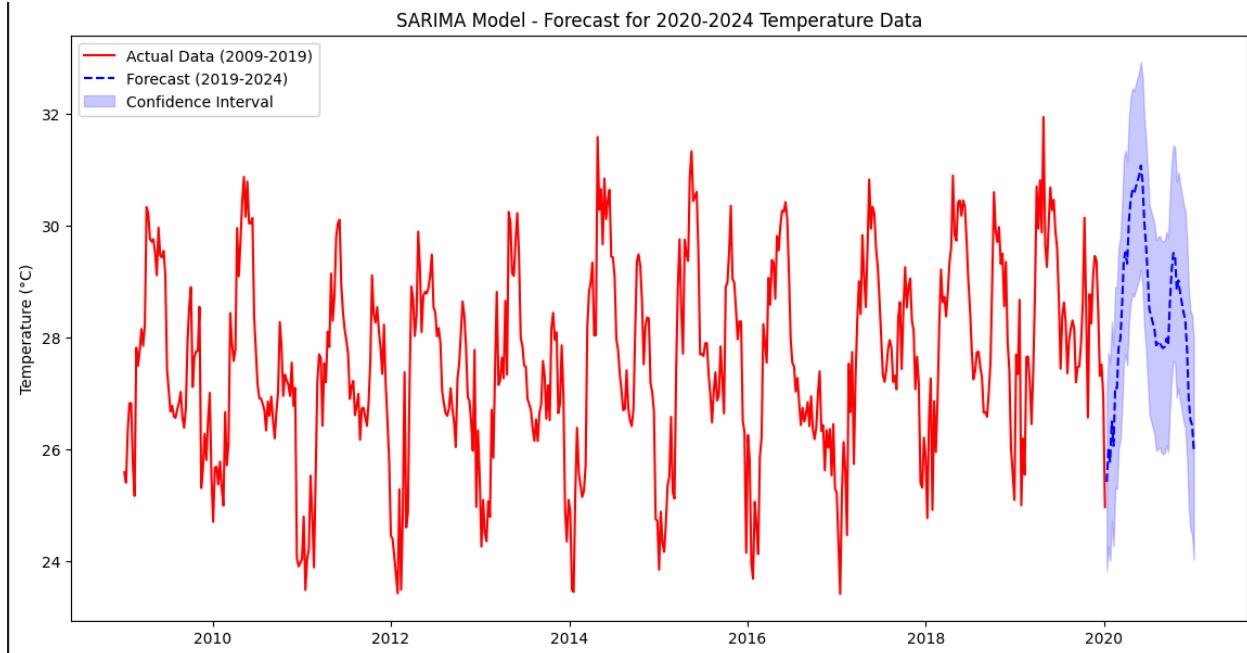
Gambar 36. Tabel SARIMA

Gambar ini menunjukkan hasil prediksi suhu menggunakan **model SARIMA** untuk tahun 2019. Garis biru menggambarkan **prediksi suhu** (Forecast), sedangkan area abu-abu menunjukkan **interval kepercayaan** (Confidence Interval). Secara umum, suhu diprediksi meningkat hingga mencapai puncaknya di sekitar pertengahan periode (590), lalu menurun perlahan di akhir periode. Lebar interval kepercayaan mengindikasikan ketidakpastian prediksi, di mana interval lebih lebar pada titik-titik tertentu, menunjukkan variasi yang lebih tinggi. Tren ini menunjukkan kemampuan model SARIMA dalam menangkap pola musiman, namun tetap menyertakan ketidakpastian dalam hasil ramalan.



Gambar 37. Hasil Plot Prediksi

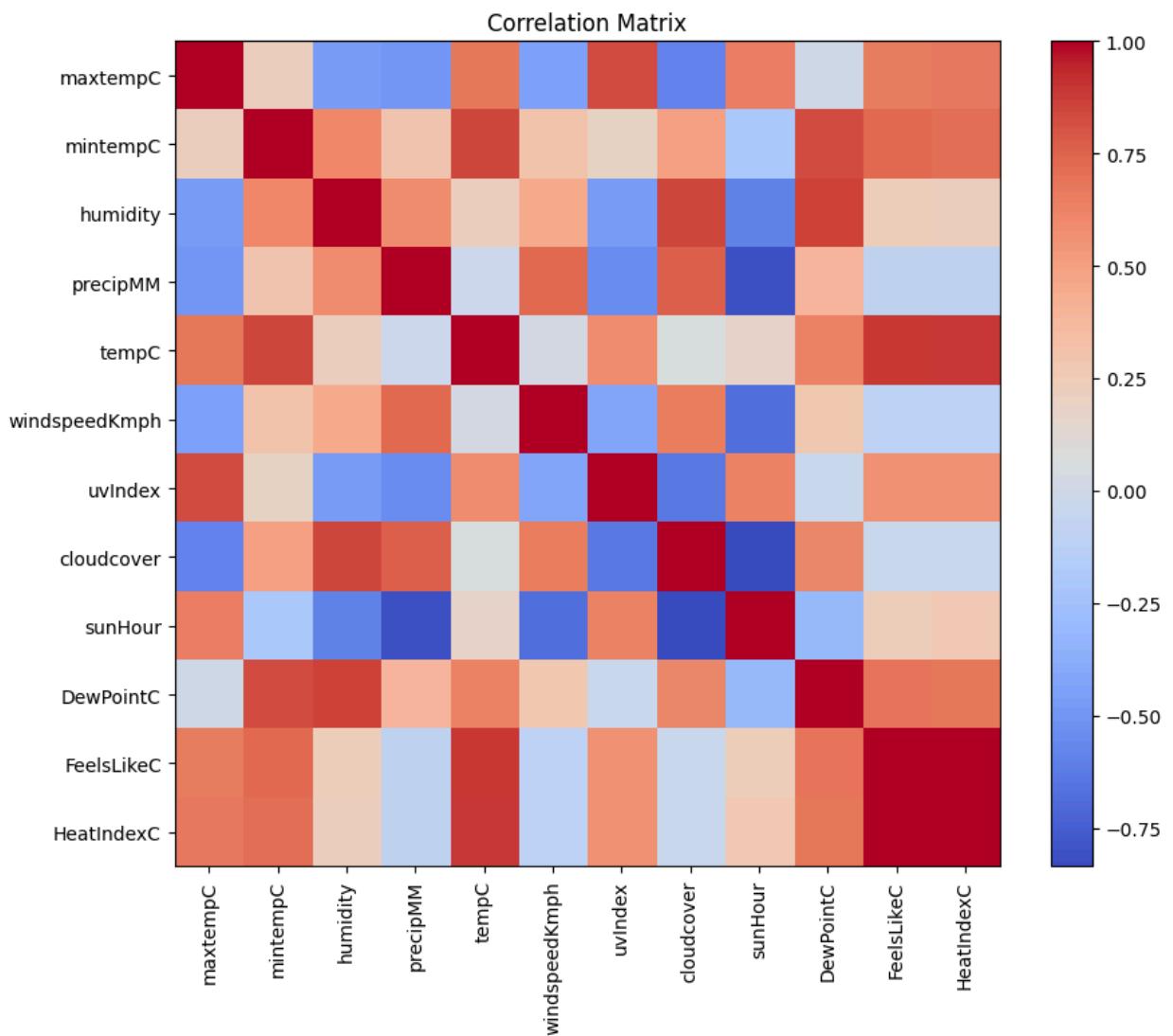
Plot dibawah ini menunjukkan hasil prediksi suhu menggunakan model SARIMA untuk tahun 2020-2024. Garis merah menggambarkan data suhu aktual dari tahun 2009-2019, sementara garis biru putus-putus menunjukkan prediksi suhu (Forecast) untuk tahun 2019-2024. Area berwarna biru pucat mewakili interval kepercayaan (Confidence Interval) yang semakin melebar seiring waktu, menandakan peningkatan ketidakpastian dalam prediksi suhu di masa depan. Tren prediksi menunjukkan adanya fluktuasi musiman yang konsisten dengan pola historis, dengan potensi peningkatan suhu di awal periode ramalan dan kemungkinan penurunan di titik-titik tertentu. Model SARIMA terlihat efektif dalam menangkap pola musiman.



Gambar 38. Hasil Plot Prediksi

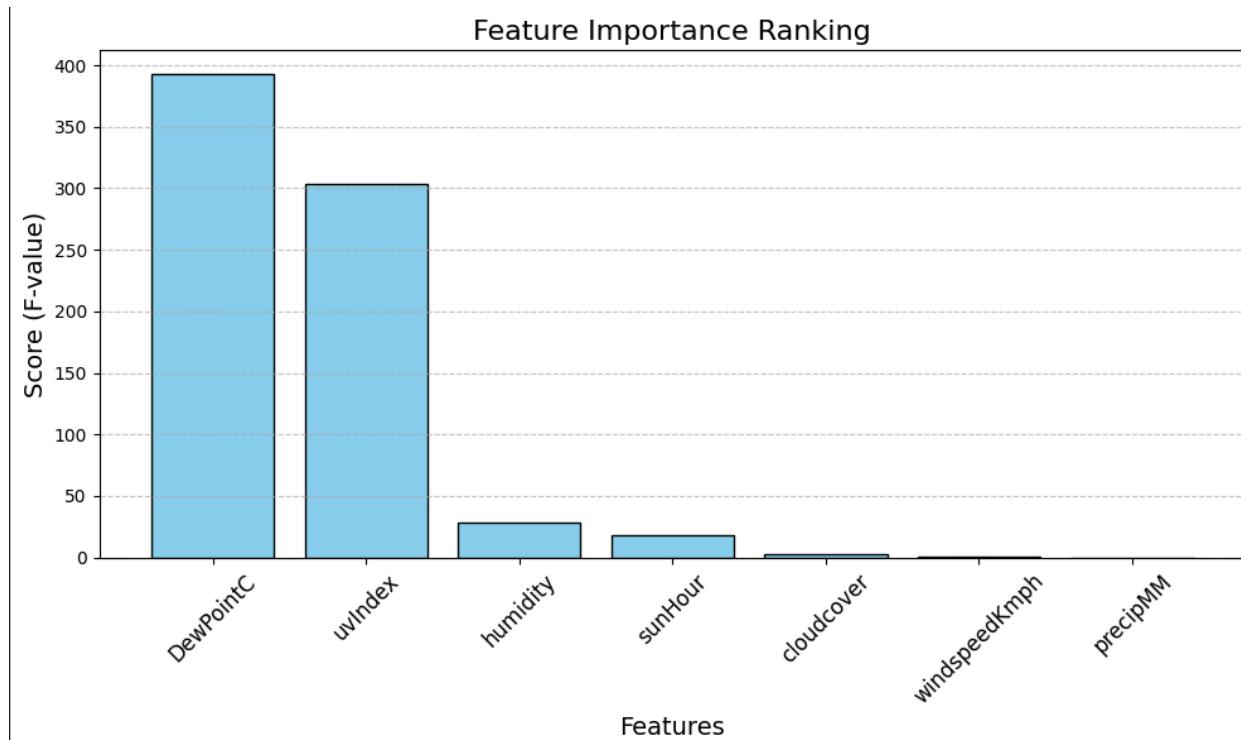
c. XGBoost, Random Forest, Decision Tree Regressor

Gambar dibawah ini menunjukkan *Correlation Matrix* yang menggambarkan hubungan antar variabel cuaca, seperti suhu, kelembapan, kecepatan angin, dan lainnya. Skala warna menunjukkan tingkat korelasi, di mana warna merah menunjukkan korelasi positif kuat (mendekati +1), warna biru menunjukkan korelasi negatif kuat (mendekati -1), dan warna netral (putih) menunjukkan korelasi mendekati nol. Terlihat bahwa variabel seperti **HeatIndexC**, **FeelsLikeC**, **tempC**, dan **DewPointC** memiliki korelasi positif yang tinggi satu sama lain. Sebaliknya, variabel **windspeedKmph** dan **precipMM** memiliki hubungan negatif dengan beberapa variabel suhu. Pola ini membantu mengidentifikasi hubungan antar variabel cuaca dan memberikan wawasan mengenai pengaruh antar parameter.



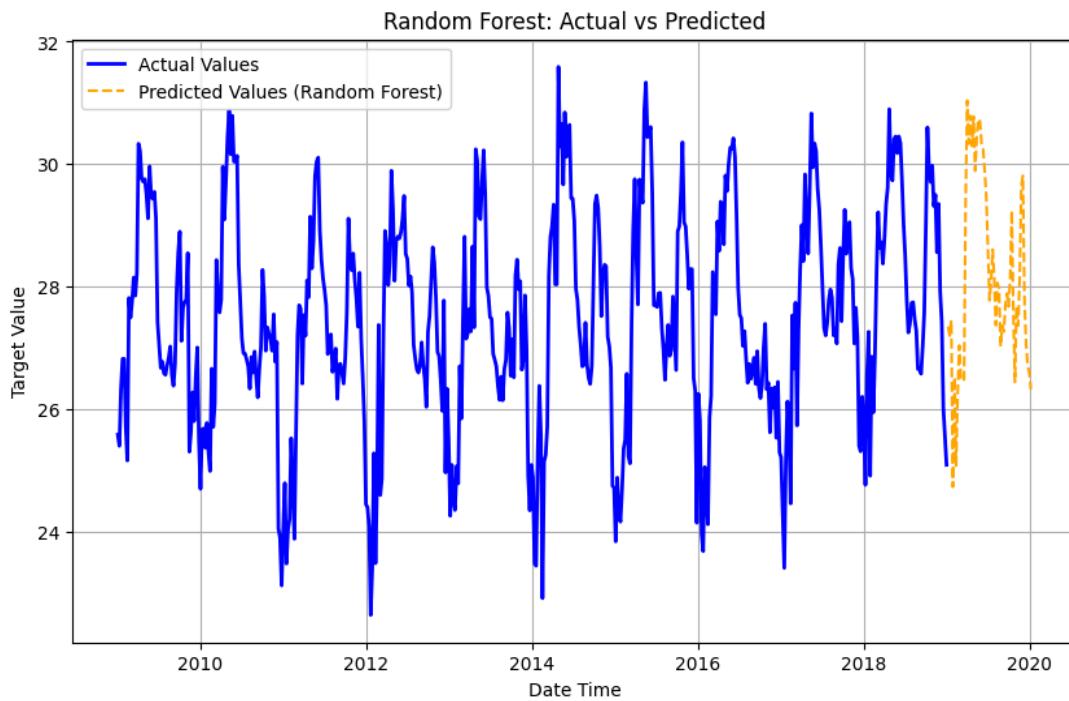
Gambar 39. Plot Correlation Matrix

Gambar dibawah menunjukkan *Feature Importance Ranking*, yang menggambarkan seberapa signifikan setiap fitur dalam mempengaruhi hasil model berdasarkan skor F-Value. **DewPointC** menempati peringkat tertinggi dengan skor mendekati 400, diikuti oleh **uvIndex** yang memiliki skor sekitar 320. Fitur lainnya seperti **humidity** dan **sunHour** memiliki kontribusi yang jauh lebih kecil, sementara **cloudcover**, **windspeedKmph**, dan **precipMM** memiliki pengaruh yang hampir tidak signifikan. Hasil ini menunjukkan bahwa **DewPointC** dan **uvIndex** adalah fitur utama yang paling penting dalam model, sedangkan fitur-fitur lainnya berperan lebih kecil dalam mempengaruhi output prediksi.



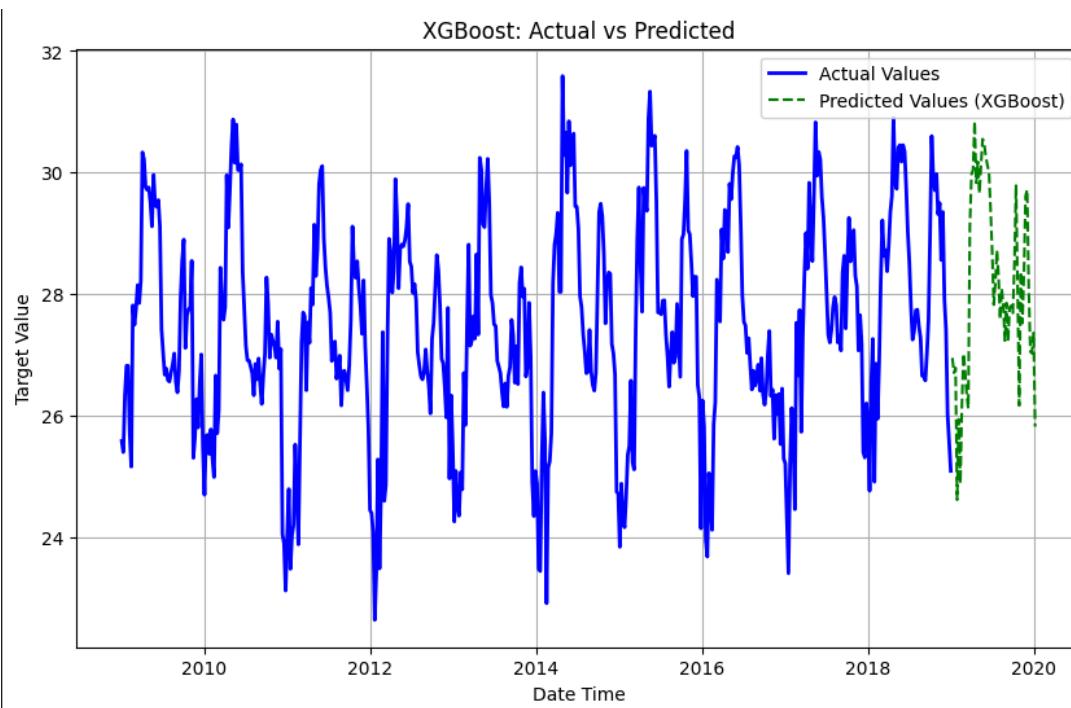
Gambar 40. Plot Feature Importance

Gambar berikut menampilkan perbandingan antara nilai aktual dan nilai prediksi menggunakan model Random Forest dalam rentang waktu tertentu. Garis biru menunjukkan nilai aktual yang berfluktuasi secara signifikan dari tahun 2009 hingga 2018, sementara garis oranye putus-putus merepresentasikan nilai prediksi yang dimulai menjelang tahun 2018 hingga 2020. Pola garis prediksi cenderung mengikuti tren nilai aktual, meskipun terdapat beberapa perbedaan kecil. Grafik ini menunjukkan kemampuan model Random Forest dalam memprediksi data masa depan dengan cukup baik berdasarkan pola historis, meskipun akurasi prediksi bisa menurun seiring waktu akibat variabilitas data yang kompleks.



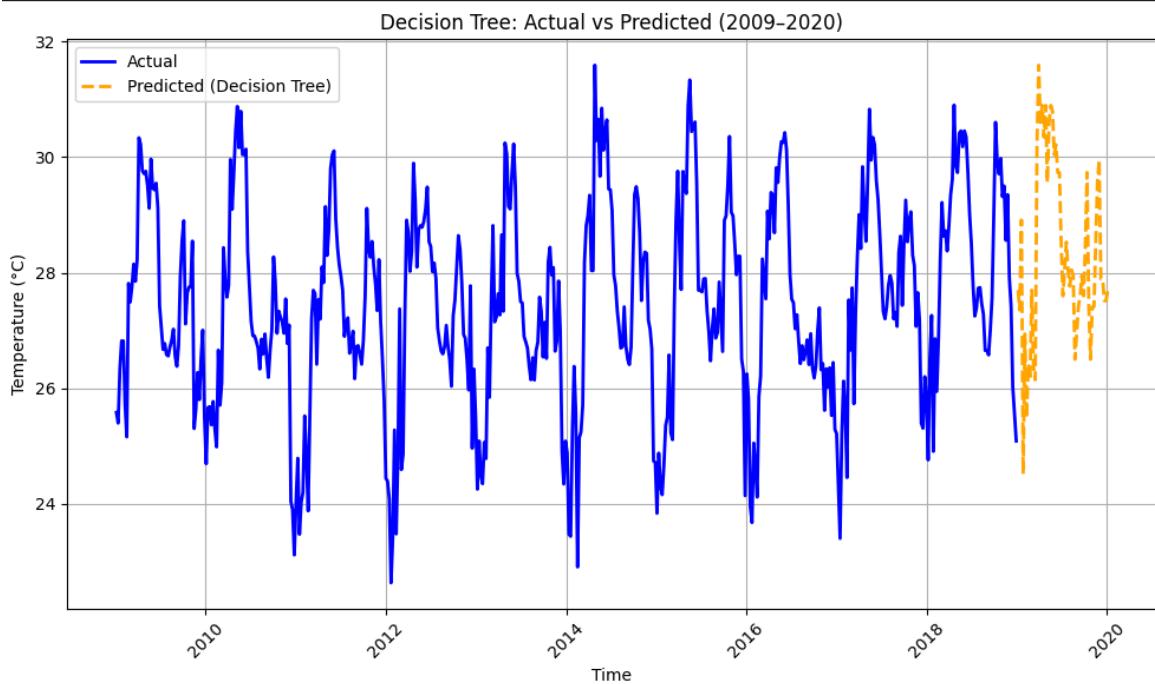
Gambar 41. Plot Predicted Values Random Forest

Plot dibawah menampilkan perbandingan antara nilai aktual dan nilai prediksi menggunakan model XGBoost dalam rentang waktu tertentu. Garis biru menunjukkan nilai aktual yang berfluktuasi secara signifikan dari tahun 2009 hingga 2018, sementara garis hijau putus-putus merepresentasikan nilai prediksi yang dimulai menjelang tahun 2018 hingga 2020. Secara umum, prediksi model XGBoost menunjukkan pola yang cenderung mengikuti tren nilai aktual, meskipun terdapat beberapa deviasi kecil. Grafik ini menggambarkan bahwa model XGBoost mampu memprediksi data masa depan dengan cukup baik berdasarkan pola historis, meskipun akurasi prediksi dapat bervariasi karena kompleksitas data.



Gambar 42. Plot Predicted Values XGBoost

Plot dibawah menunjukkan perbandingan antara nilai aktual dan nilai prediksi menggunakan model *Decision Tree* dalam rentang waktu 2009 hingga 2020. Garis biru merepresentasikan nilai aktual yang berfluktuasi secara signifikan selama periode tersebut, sementara garis oranye putus-putus menunjukkan nilai prediksi yang dimulai menjelang tahun 2018 hingga 2020. Meskipun prediksi mengikuti pola umum tren historis, terdapat beberapa deviasi antara nilai aktual dan prediksi. Grafik ini menggambarkan kemampuan model *Decision Tree* dalam memprediksi tren data masa depan, meskipun tingkat akurasi dapat bervariasi karena pola data yang kompleks dan fluktuatif.



Gambar 43. Plot Predicted Values Decision Tree

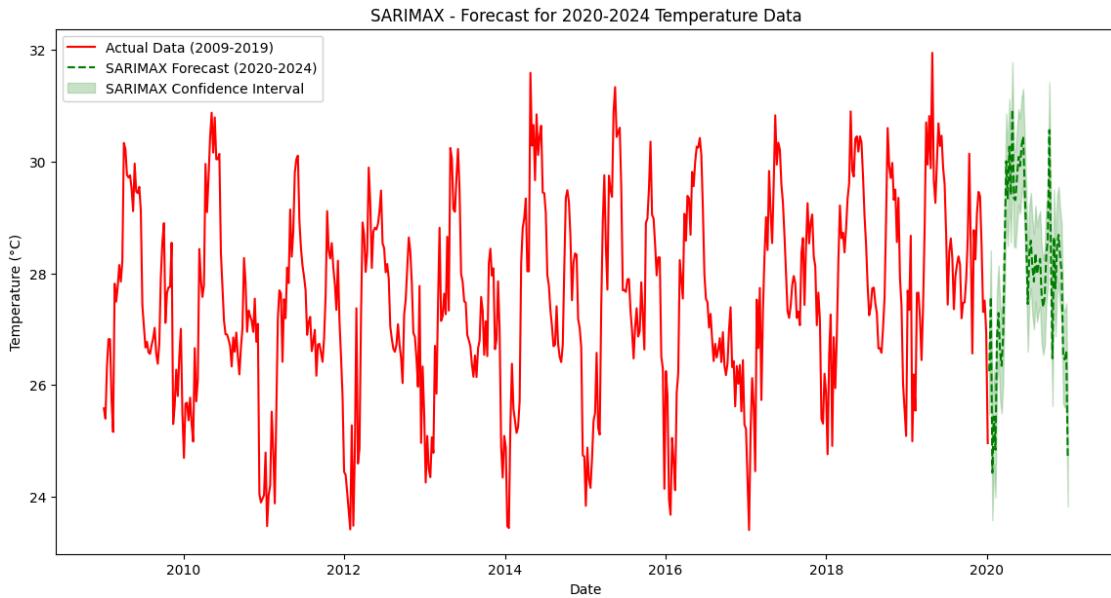
d. SARIMAX Model

Output dibawah ini menunjukkan hasil model SARIMAX untuk memprediksi suhu (tempC) dengan beberapa variabel bebas. Fokus pada p-value menunjukkan bahwa variabel uvIndex, humidity, cloudcover, DewPointC, windspeedKmph, sunHour, dan temp_diff memiliki p-value < 0.05, sehingga signifikan terhadap suhu. Selain itu, komponen AR(1) (ar.L1), MA(1) (ma.L1), komponen musiman (ma.S.L52), dan varians error (sigma2) juga signifikan. Sementara itu, precipMM dan ar.S.L52 memiliki p-value > 0.05, sehingga tidak signifikan dalam memengaruhi prediksi suhu.

SARIMAX Results						
Dep. Variable:	tempC	No. Observations:	572			
Model:	SARIMAX(1, 1, 1)x(1, 1, 1, 52)	Log Likelihood	-283.902			
Date:	Thu, 12 Dec 2024	AIC	593.804			
Time:	08:05:16	BIC	649.078			
Sample:	0 - 572	HQIC	615.459			
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
uvIndex	0.9378	0.076	12.393	0.000	0.789	1.086
humidity	-0.2010	0.008	-25.166	0.000	-0.217	-0.185
cloudcover	0.0448	0.003	14.898	0.000	0.039	0.051
DewPointC	0.6662	0.026	25.609	0.000	0.615	0.717
windspeedKmph	0.0240	0.010	2.334	0.020	0.004	0.044
sunHour	0.2971	0.046	6.513	0.000	0.208	0.386
temp_diff	-0.1253	0.022	-5.764	0.000	-0.168	-0.083
precipMM	-0.1501	0.155	-0.968	0.333	-0.454	0.154
ar.L1	0.4084	0.037	11.079	0.000	0.336	0.481
ma.L1	-0.9920	0.007	-141.884	0.000	-1.006	-0.978
ar.S.L52	-0.0846	0.060	-1.407	0.159	-0.203	0.033
ma.S.L52	-0.7868	0.074	-10.575	0.000	-0.933	-0.641
sigma2	0.1551	0.011	13.842	0.000	0.133	0.177
...						

Gambar 44. SARIMAX Table

Plot dibawah ini menunjukkan hasil prediksi menggunakan model SARIMAX untuk data suhu selama tahun 2020-2024 berdasarkan data aktual dari tahun 2009-2019. Garis merah mewakili data aktual (2009-2019), sedangkan garis hijau putus-putus menunjukkan prediksi model SARIMAX untuk periode 2020-2024. Area berwarna hijau muda mengindikasikan interval kepercayaan prediksi, yang mencerminkan tingkat ketidakpastian model dalam memproyeksikan nilai masa depan. Dari grafik ini, terlihat bahwa prediksi model mencerminkan pola musiman dari data historis dengan fluktuasi yang konsisten, meskipun ada ketidakpastian yang semakin meningkat pada ujung prediksi (2024).



Gambar 45. Plot Predict SARIMAX

e. Perbandingan R-Squared untuk setiap model

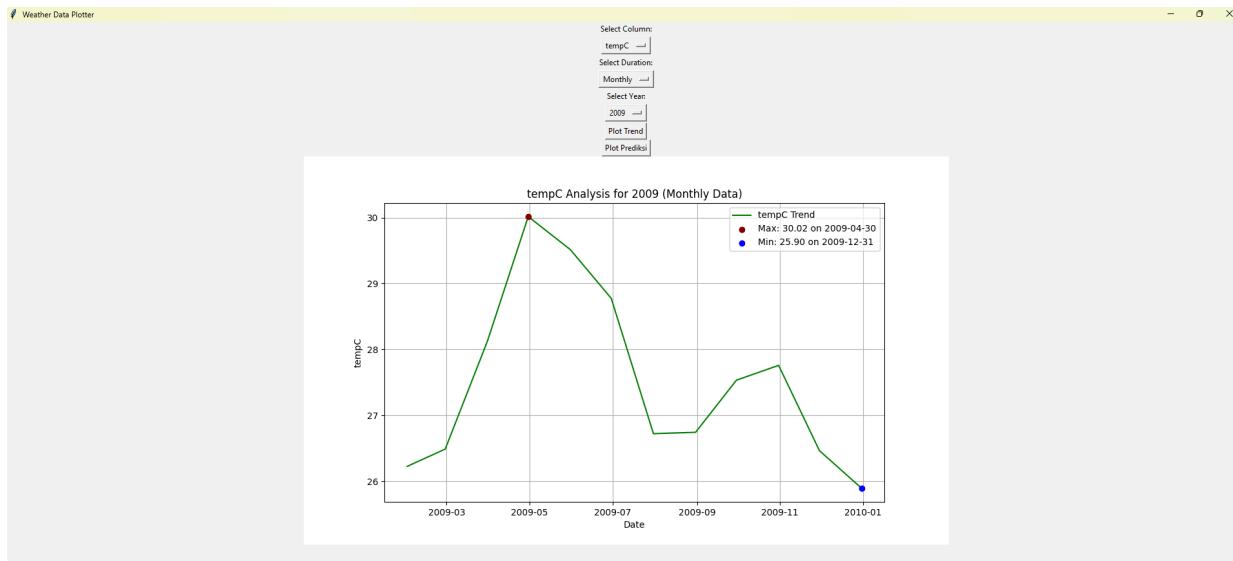
Berdasarkan nilai R-squared yang diperoleh dari berbagai model, Random Forest memiliki nilai 0,82, XGBoost sebesar 0,83, Decision Tree sebesar 0,71, SARIMA sebesar 0,63, dan SARIMAX sebesar 0,87. Di antara semua model yang diuji, SARIMAX menunjukkan performa terbaik dengan nilai R-squared tertinggi, yaitu 0,87, yang menunjukkan kemampuan model ini dalam menjelaskan variabilitas data secara lebih akurat dibandingkan model lainnya. Oleh karena itu, kami memutuskan untuk menggunakan model SARIMAX untuk analisis lebih lanjut, karena model ini memberikan hasil prediksi yang paling andal berdasarkan evaluasi metrik R-squared.



Gambar 46. Perbandingan Setiap Model

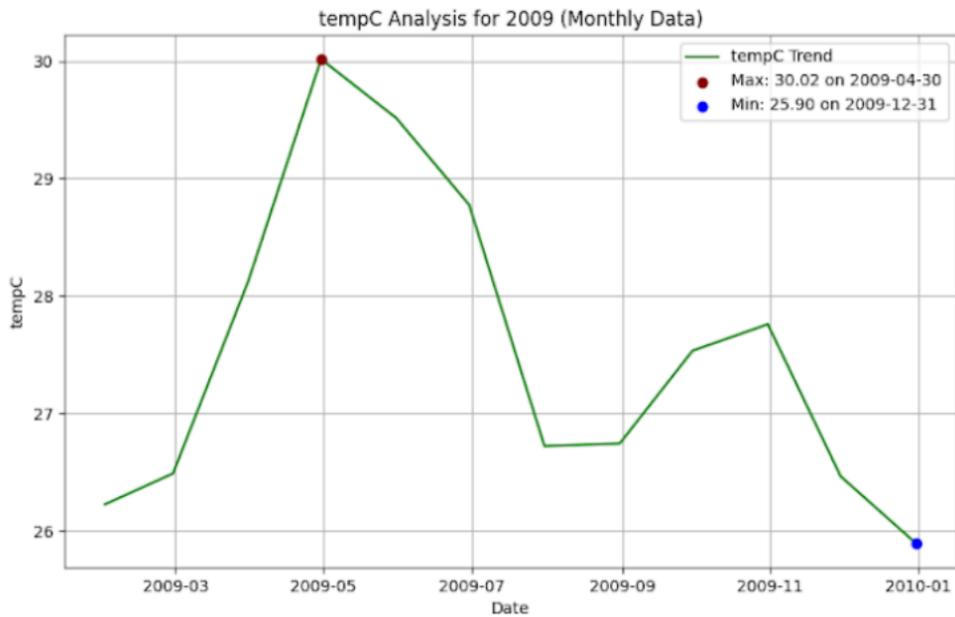
E. Presentation and Automation

GUI ini memiliki tiga dropdown utama untuk memilih data yang akan dianalisis. **Dropdown Select Column** memungkinkan pengguna memilih kolom data, dengan opsi utama `tempC`, `humidity`, dan `windspeedKmph`. **Dropdown Select Duration** menyediakan pilihan durasi analisis, yaitu monthly, weekly, dan daily, sementara **Dropdown Select Year** memungkinkan seleksi tahun dari 2009 hingga 2024. Setelah parameter dipilih, pengguna dapat memilih salah satu dari dua aksi: Plot Tren atau Plot Prediksi. Untuk Plot Tren, semua kolom di **Dropdown Select Column**, semua durasi di **Dropdown Select Duration**, dan tahun antara 2009 hingga 2019 dapat dipilih. Sementara itu, untuk Plot Prediksi, hanya kolom `tempC` dan `humidity` di **Dropdown Select Column**, durasi monthly atau weekly di **Dropdown Select Duration**, serta tahun antara 2020 hingga 2024 yang dapat digunakan. GUI ini dirancang untuk mempermudah eksplorasi dan prediksi tren data cuaca.



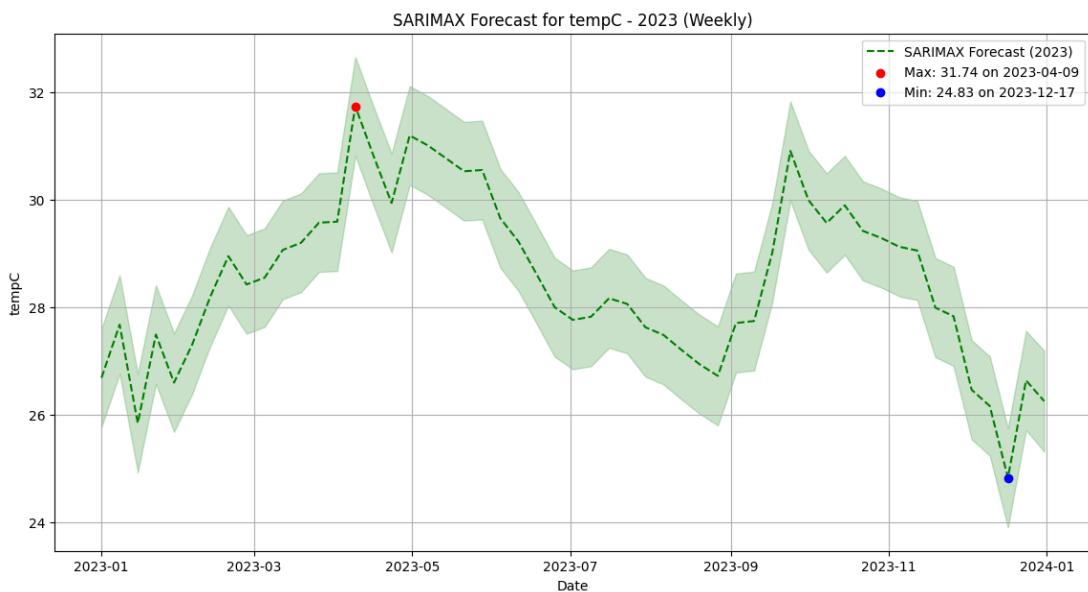
Gambar 47. Bentuk GUI

Plot dibawah ini menunjukkan analisis suhu bulanan pada tahun 2009 dengan tren suhu yang diwakili oleh garis hijau. Titik merah menandai suhu maksimum sebesar 30,02°C yang terjadi pada 30 April 2009, sedangkan titik biru menunjukkan suhu minimum sebesar 25,90°C yang terjadi pada 31 Desember 2009. Dari grafik terlihat bahwa suhu mencapai puncaknya di pertengahan tahun dan menurun secara bertahap hingga akhir tahun, mencerminkan pola musiman dengan fluktuasi yang signifikan sepanjang tahun.



Gambar 48. Bentuk Plot Trend

Grafik tersebut menampilkan prediksi suhu mingguan untuk tahun 2023 menggunakan model SARIMAX. Garis hijau putus-putus menunjukkan nilai prediksi suhu, sedangkan area berwarna hijau muda menggambarkan interval kepercayaan yang mencerminkan tingkat ketidakpastian dalam prediksi. Suhu maksimum diperkirakan terjadi pada 9 April 2023 sebesar 31,74°C (ditandai dengan titik merah), sedangkan suhu minimum diprediksi pada 17 Desember 2023 sebesar 24,83°C (ditandai dengan titik biru). Pola grafik menunjukkan adanya fluktuasi musiman, dengan suhu yang cenderung meningkat di awal tahun hingga mencapainya pada pertengahan tahun, sebelum menurun secara bertahap menjelang akhir tahun.



Gambar 49. Bentuk Plot Prediksi

BAB IV

KESIMPULAN

Proyek ini berhasil mengidentifikasi pola perubahan cuaca di Mumbai dari tahun 2009 hingga 2019 melalui analisis tren historis yang mencakup variabel suhu (tempC), kelembapan (humidity), dan kecepatan angin (windspeed). Analisis ini menunjukkan adanya pola musiman yang konsisten, di mana visualisasi grafik memberikan wawasan mendalam terkait fluktuasi cuaca selama satu dekade. Untuk periode 2021 hingga 2024, prediksi cuaca dilakukan menggunakan model SARIMAX pada variabel suhu dan kelembapan dengan tingkat akurasi yang memadai, sedangkan prediksi kecepatan angin ditiadakan karena performa model yang rendah, yang mengindikasikan perlunya variabel tambahan untuk meningkatkan akurasi.

Hasil penelitian ini tidak hanya memberikan wawasan penting mengenai tren dan pola cuaca, tetapi juga diharapkan mampu meningkatkan kesadaran masyarakat terhadap dampak pola cuaca jangka panjang. Selain itu, penelitian ini berkontribusi pada upaya mitigasi risiko cuaca ekstrem di masa depan, membuktikan bahwa kombinasi analisis tren historis dan model prediksi modern dapat menjadi alat yang andal untuk memahami dan memprediksi perubahan iklim.

DAFTAR PUSTAKA

Ardilla, Y., Guntoro, G., Afnarius, S., & Santoso, A. (2022). *DATA SCIENCE*. WIDINA BHAKTI PERSADA BANDUNG.

<https://repository.penerbitwidina.com/publications/554354/data-science#cite>

Devianto, Yudo, & Dwiasnati, S. (2020, 4). Kerangka Kerja Sistem Kecerdasan Buatan dalam Meningkatkan Kompetensi Sumber Daya Manusia Indonesia. *InComTech : Jurnal Telekomunikasi dan Komputer*, vol. 10(no. 1), 19-24. 10.22441/incomtech.v10i1.7460

Fikri, Akhsanul, & Mustikasari, M. (2022). Trend Analysis in Sales Forecasting and Decision Support Systems AHP Method on the Selection of Types of Motorcycles PT. AHM. *International Journal of Multidisciplinary: Applied Business and Education Research*, 3(no. 5), 737-747. 10.11594/ijmaber.03.05.02

Hamdani, I., Nurhidayat, N., Karman, A., & Fuady, N. (2024). Edukasi dan Pelatihan Data Science dan Data Preprocessing. *Jurnal Inovasi Pengabdian Masyarakat*, 2(1), 19-26. 10.58227/intisari.v2i1.125

Hartama, D. (2018, 7). ANALISA VISUALISASI DATA AKADEMIK MENGGUNAKAN TABLEAU BIG DATA. *Jurnal Riset Sistem Informasi Dan Teknik Informatika (JURASIK)*, 3, 46-55.

Retrieved 10 11, 2024, from <https://tunasbangsa.ac.id/ejurnal/index.php/jurasik/article/view/65/57>

Heryana, A. (2013). Perangkat Lunak Pemantau Cuaca Pada Stasiun Cuaca Cilegon. *NKOM Journal of Informatics, Control Systems, and Computers*, vol. 7,(no. 1). 10.14203/j.inkom.219.

Jayanthi, K. B., Vanitha, K., & Malathi, M. (2022). An Efficient Weather Recognition Algorithm on Highway Roads for Vehicle Guidance. *International journal of health sciences*, 6(no. S8), 1212-1223. Retrieved 10 18, 2024, from <https://www.neliti.com/publications/574483/an-efficient-weather-recognition-algorithm-on-highway-roads-for-vehicle-guidance#cite>

Komal, A., Sonu, S., & Tejswini, S. (2019, 2). IOT Based Weather Monitoring System. *JournalNX*, 352-357. Retrieved 12 7, 2024, from

<https://www.neliti.com/publications/342341/iot-based-weather-monitoring-system#cite>

Kurniawan, J., Hartoto, H., Fahmi, A., Ahyani, H., Hikmah, H., & Ridwan, M. (2023). *ANALISIS DAN VISUALISASI DATA*. CV WIDINA MEDIA UTAMA.

<https://repository.penerbitwidina.com/publications/560454/analisis-dan-visualisasi-data#cite>

Mulyani, S., Hayat, D., & Sari, A. (2021, 3). ANALISIS METODE PERAMALAN (FORECASTING) PENJUALAN SEPEDA MOTOR HONDA DALAM MENYUSUN ANGGARAN

PENJUALAN PADA PT TRIO MOTOR MARTADINATA BANJARMASIN. *Jurnal Ekonomi dan Bisnis*, 14(No.1), 178-188. Retrieved 10 11, 2024, from

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjYvObv0oaJAxWLzjgGHXJwHIMQFnoECDAQAAQ&url=https%3A%2F%2Fstienas-y pb.ac.id%2Fjurnal%2Findex.php%2Fjdeb%2Farticle%2Fdownload%2F320%2F304&usg=AOvV aw1EyoXohRGTw68ROrgoKJv_&

Murmayani, M., & Darwis, D. (2024). *AGROKLIMATOLOGI*. CV WIDINA MEDIA UTAMA.

<https://repository.penerbitwidina.com/publications/586429/agroklimatologi#cite>

Soneji, H. (2020). *Historical Weather Data for Indian Cities*. Kaggle Datasets. Retrieved 10 18, 2024, from

https://weatherspark.com/h/y/107286/2012/Historical-Weather-during-2012-in-Mumbai-Maharas htra-India#google_vignette

Suryanto, J., & Krisbiyantoro, J. (2018, 10). ANALISIS KECENDERUNGAN CURAH HUJAN

KABUPATEN MAGELANG MENGGUNAKAN UJI MANN-KENDALL DAN VARIASI

MODIFIKASI MANN-KENDALL. *Jurnal AGRIFOR*, 17(Nomor 2), 293-304. Retrieved 10 11, 2024, from

<https://www.neliti.com/publications/363333/trend-analysis-of-rainfall-data-in-magelang-district-using-mann-kendall-test-and>

- T, P., A, S., & N, R. (2021). Pemanfaatan Pencahayaan Alami Iklim Tropis pada Bangunan Hotel Resort di Bali. *Jurnal Arsitektur Zonasi*, 4(1), 114-120. <https://doi.org/doi.org/10.17509/jaz.v4i1.27141>
- Utami, & S, A. (2021, 04 14). Weather Forecasting Based on Supervised Learning Using K-Nearest Neighbour Algorithm. *Jurnal Penelitian Ilmu dan Teknologi Komputer*, 13(no. 1). Retrieved 10 18, 2024, from
<https://www.neliti.com/publications/440828/weather-forecasting-based-on-supervised-learning-using-k-nearest-neighbour-algor#cite>
- WeatherSpark. (2024). *2012 Weather History in Mumbai*. Weather History. Retrieved 10 19, 2024, from
<https://weatherspark.com/h/y/107286/2012/Historical-Weather-during-2012-in-Mumbai-Maharashtra-India>
- Weather Spark. (2024). *2024 Weather History in Mumbai*. Weather Spark. Retrieved 10 11, 2024, from
<https://weatherspark.com/h/y/107286/2024/Historical-Weather-during-2024-in-Mumbai-Maharashtra-India#Figures-Temperature>
- Wunderground. (2024). *Mumbai, Maharashtra, India Weather History*. Weather Underground. Retrieved 10 19, 2024, from
<https://www.wunderground.com/history/monthly/in/mumbai/VABB/date/2012-3>

Lampiran

Berikut adalah bagian lampiran berisi screenshot kodingan. Bila ada kode yang kurang kelihatan atau tidak jelas, boleh di zoom agar lebih terlihat dan jelas.

A. Retrieving Data

```
#Retrieving the data
#Import files
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.dates as mdates

# Reading the CSV file
file_path = 'mumbai.csv'
data = pd.read_csv(file_path)
```

B. Data Preparation

```
#Data Preparation
# Select only the relevant columns
weather_data = data[['date_time', 'maxtempC', 'mintempC', 'humidity', 'precipMM', 'tempC', 'windspeedKmph', 'uvIndex', 'cloudcover', 'sunHour', 'DewPointC', 'FeelsLikeC', 'HeatIndexC']]

# Convert 'date_time' to datetime format (removing timezone awareness if it exists)
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'], utc=True).dt.tz_localize(None)

# Ensure comparison dates are timezone-naive (you can adjust the date range based on the dataset)
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2020-01-01')

# Filtering data between the start and end dates
filtered_data = weather_data[(weather_data['date_time'] >= start_date) &
                               | (weather_data['date_time'] <= end_date)]

# Set 'date_time' as the index
filtered_data.set_index('date_time', inplace=True)

# Finding the maximum and minimum values without resampling
temp_max = filtered_data['maxtempC'].max()
temp_min = filtered_data['mintempC'].min()

humidity_max = filtered_data['humidity'].max()
humidity_min = filtered_data['humidity'].min()

wind_max = filtered_data['windspeedKmph'].max()
wind_min = filtered_data['windspeedKmph'].min()
```

```

# Extracting the corresponding dates for max and min values
temp_max_date = filtered_data[filtered_data['maxtempC'] == temp_max].index[0]
temp_min_date = filtered_data[filtered_data['mintempC'] == temp_min].index[0]

humidity_max_date = filtered_data[filtered_data['humidity'] == humidity_max].index[0]
humidity_min_date = filtered_data[filtered_data['humidity'] == humidity_min].index[0]

wind_max_date = filtered_data[filtered_data['windspeedKmph'] == wind_max].index[0]
wind_min_date = filtered_data[filtered_data['windspeedKmph'] == wind_min].index[0]

# Calculating monthly averages
monthly_avg = filtered_data.resample('M').mean()

# Resample into weekly averages
weekly_avg = filtered_data.resample('W').mean()

# Calculating the monthly trend based on weekly averages
monthly_trend = weekly_avg.resample('M').mean()

# Calculating rolling averages for Temperature, Humidity, and Wind Speed on weekly data
weekly_avg['Temp_Rolling_Avg'] = weekly_avg['tempC'].rolling(window=30).mean()
weekly_avg['Humidity_Rolling_Avg'] = weekly_avg['humidity'].rolling(window=30).mean()
weekly_avg['Wind_Rolling_Avg'] = weekly_avg['windspeedKmph'].rolling(window=30).mean()

# Extra Keys Consideration for Data Preparation before Data Exploration
# Re-running outlier detection excluding the 'date_time' column
numerical_columns = ['maxtempC', 'mintempC', 'humidity', 'precipMM', 'tempC', 'windspeedKmph', 'cloudcover', 'uvIndex', 'sunHour', 'DewPointC', 'FeelsLikeC']

Q1 = weather_data[numerical_columns].quantile(0.25)
Q3 = weather_data[numerical_columns].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Finding rows with outliers
outliers = weather_data[(weather_data[numerical_columns] < lower_bound) | (weather_data[numerical_columns] > upper_bound)].count()
# Checking for missing values again
missing_values = weather_data.isnull().sum()

missing_values, outliers

#No Usage of Unbalanced Data because most of our data consists of numerical which uses Outliers instead

```

C. Data Assessment

Ground Truth Data #1

```
# Calculate percentiles for each day in 2012
percentile_25 = filtered_data_2012['tempC'].resample('D').apply(lambda x: np.percentile(x, 25))
percentile_75 = filtered_data_2012['tempC'].resample('D').apply(lambda x: np.percentile(x, 75))
percentile_10 = filtered_data_2012['tempC'].resample('D').apply(lambda x: np.percentile(x, 10))
percentile_90 = filtered_data_2012['tempC'].resample('D').apply(lambda x: np.percentile(x, 90))

# Ensure the index alignment
percentile_10 = percentile_10.reindex(daily_max.index)
percentile_90 = percentile_90.reindex(daily_max.index)
percentile_25 = percentile_25.reindex(daily_max.index)
percentile_75 = percentile_75.reindex(daily_max.index)

plt.figure(figsize=(12, 6))

# Plot the 10th to 90th percentile (gray shading)
plt.fill_between(
    daily_max.index,
    percentile_10,
    percentile_90,
    color='gray',
    alpha=0.3,
    label='10th-90th Percentile'
)
```

```
# Plot the 25th to 75th percentile (darker gray shading)
plt.fill_between(
    daily_max.index,
    percentile_25,
    percentile_75,
    color='gray',
    alpha=0.5,
    label='25th-75th Percentile'
)

# Add daily range (thin vertical bars in gray)
plt.vlines(
    daily_max.index,
    daily_min,
    daily_max,
    color='gray',
    alpha=0.7,
    linewidth=0.5
)

# Adding labels and title
plt.title('Ground Truth Data Temperature Trend in Mumbai 2012', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Temperature (°C)', fontsize=12)

# Formatting the x-axis to show months
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gcf().autofmt_xdate()

# Adding grid and legend
plt.grid(True, linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```

Ground Truth Data #2

```
# Ground Truth Data for Wind Speed Mumbai Data in the year 2019
# Filtering for 2019
start_date_2019 = pd.to_datetime('2019-01-01')
end_date_2019 = pd.to_datetime('2019-12-31')
filtered_data_2019 = filtered_data[(filtered_data.index >= start_date_2019) & (filtered_data.index <= end_date_2019)]

# Find max and min wind speed values
max_wind = filtered_data_2019['windspeedKmph'].max()
min_wind = filtered_data_2019['windspeedKmph'].min()

# Extract corresponding dates for the max and min wind speed
max_wind_date = filtered_data_2019[filtered_data_2019['windspeedKmph'] == max_wind].index[0]
min_wind_date = filtered_data_2019[filtered_data_2019['windspeedKmph'] == min_wind].index[0]

# Plotting with month-based x-axis and max/min points
plt.figure(figsize=(10, 6))
plt.plot(filtered_data_2019.index, filtered_data_2019['windspeedKmph'], label='Wind Speed (km/h)', color='blue')

# Marking max and min points with full date in labels
plt.scatter(max_wind_date, max_wind, color='red', label=f'Max Wind: {max_wind} km/h on {max_wind_date.date()}', zorder=5)
plt.scatter(min_wind_date, min_wind, color='green', label=f'Min Wind: {min_wind} km/h on {min_wind_date.date()}', zorder=5)

# Formatting the plot
plt.title('Ground Truth Data Wind Speed with Max and Min Values in Mumbai 2019')
plt.xlabel('Date')
plt.ylabel('Wind Speed (km/h)')
plt.legend()

# Set x-axis to show months
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())

plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

D. Data Eksplorasi

```

#Data Exploration
#Weekly Average and Monthly Trend in Mumbai (2009 - 2019)
# Plot weekly averages and rolling averages with resampled weekly data
plt.figure(figsize=(14, 12))

# Weekly Temperature and Rolling Average
plt.subplot(3, 1, 1)
plt.plot(weekly_avg.index, weekly_avg['tempC'], label='Weekly Avg Temperature (°C)', color='red')
plt.plot(weekly_avg.index, weekly_avg['Temp_Rolling_Avg'], label='30-Week Rolling Avg Temp (°C)', color='orange', linestyle='--')
plt.title('Weekly Average Temperature in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)

# Weekly Humidity and Rolling Average
plt.subplot(3, 1, 2)
plt.plot(weekly_avg.index, weekly_avg['humidity'], label='Weekly Avg Humidity (%)', color='green')
plt.plot(weekly_avg.index, weekly_avg['Humidity_Rolling_Avg'], label='30-Week Rolling Avg Humidity (%)', color='lightgreen', linestyle='--')
plt.title('Weekly Average Humidity in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.legend()
plt.grid(True)

# Weekly Wind Speed and Rolling Average
plt.subplot(3, 1, 3)
plt.plot(weekly_avg.index, weekly_avg['windspeedKmph'], label='Weekly Avg Wind Speed (km/h)', color='purple')
plt.plot(weekly_avg.index, weekly_avg['Wind_Rolling_Avg'], label='30-Week Rolling Avg Wind Speed (km/h)', color='plum', linestyle='--')
plt.title('Weekly Average Wind Speed in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Wind Speed (km/h)')
plt.legend()
plt.grid(True)

# Adjust layout and display the plot
plt.tight_layout()
plt.show()

# Raw Data without Resampling 2009-2019
# Plotting the Maximum, Minimum, and Monthly Average Data
plt.figure(figsize=(14, 18))

# Max temp, min temp, and monthly avg tempC
plt.subplot(3, 1, 1)
plt.plot(filtered_data.index, filtered_data['tempC'], label='Temperature (°C)', color='red')
plt.scatter(temp_max_date, temp_max, color='darkred', label=f'Max: {temp_max}°C on {temp_max_date.date()}', zorder=5)
plt.scatter(temp_min_date, temp_min, color='blue', label=f'Min: {temp_min}°C on {temp_min_date.date()}', zorder=5)
plt.plot(monthly_avg.index, monthly_avg['tempC'], label='Monthly Avg Temperature (°C)', color='orange', linestyle='--')
plt.title('Temperature (Max, Min, and Monthly Avg) in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)

# Humidity max, min, and monthly avg
plt.subplot(3, 1, 2)
plt.plot(filtered_data.index, filtered_data['humidity'], label='Humidity (%)', color='green')
plt.scatter(humidity_max_date, humidity_max, color='darkgreen', label=f'Max: {humidity_max}% on {humidity_max_date.date()}', zorder=5)
plt.scatter(humidity_min_date, humidity_min, color='lime', label=f'Min: {humidity_min}% on {humidity_min_date.date()}', zorder=5)
plt.plot(monthly_avg.index, monthly_avg['humidity'], label='Monthly Avg Humidity (%)', color='lightgreen', linestyle='--')
plt.title('Humidity (Max, Min, and Monthly Avg) in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Humidity (%)')
plt.legend()
plt.grid(True)

# Wind Speed max, min, and monthly avg
plt.subplot(3, 1, 3)
plt.plot(filtered_data.index, filtered_data['windspeedKmph'], label='Wind Speed (km/h)', color='purple')
plt.scatter(wind_max_date, wind_max, color='purple', label=f'Max: {wind_max} km/h on {wind_max_date.date()}', zorder=5)
plt.scatter(wind_min_date, wind_min, color='violet', label=f'Min: {wind_min} km/h on {wind_min_date.date()}', zorder=5)
plt.plot(monthly_avg.index, monthly_avg['windspeedKmph'], label='Monthly Avg Wind Speed (km/h)', color='plum', linestyle='--')
plt.title('Wind Speed (Max, Min, and Monthly Avg) in Mumbai (2009 - 2019)')
plt.xlabel('Date')
plt.ylabel('Wind Speed (km/h)')
plt.legend()
plt.grid(True)

# Adjust layout and display the plot
plt.tight_layout()
plt.show()

```

E. Data Modelling

a. PCA

PCA (Principal Component Analysis) to analyze data as a whole

Preparation for model

```
[13] import pandas as pd
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt
     import matplotlib.colors as mcolors
     import matplotlib.patches as mpatches
     from mpl_toolkits.mplot3d import Axes3D # Import for 3D plotting

     # Assume filtered_data and weekly resampling
     daily_data = filtered_data.resample('D').mean()

     # Select relevant columns
     features = ['tempC', 'humidity', 'windspeedKmph']
     data = daily_data[features]
```

Standardize, Apply PCA, and Styling for cluster colors

```
[14] # Standardize the data
     scaler = StandardScaler()
     data_scaled = scaler.fit_transform(data)

     # Apply PCA for dimensionality reduction to 3 components
     pca = PCA(n_components=3)
     data_pca = pca.fit_transform(data_scaled)

     # Define thresholds for coloring based on the first PCA component
     cool_threshold = -1 # Cool conditions
     warm_threshold = 1 # Warm conditions

     # Assign specific colors based on thresholds
     colors = [
         'blue' if x < cool_threshold else
         'red' if x > warm_threshold else
         'orange'
         for x in data_pca[:, 0]
     ]
```

Plotting 2D and 3D PCA Plot

```
[15] # 2D PCA Plot with specific colors
    plt.figure(figsize=(12, 8))
    plt.scatter(data_pca[:, 0], data_pca[:, 1], c=colors, alpha=0.6)

    plt.xlabel('1st Principal Component')
    plt.ylabel('2nd Principal Component')
    plt.title('PCA Components with Weather Categories')

    # Create legend patches with specific colors
    cool_patch = mpatches.Patch(color='blue', label='Cool Conditions')
    moderate_patch = mpatches.Patch(color='orange', label='Moderate Conditions')
    warm_patch = mpatches.Patch(color='red', label='Warm Conditions')

    # Add legend and grid
    plt.legend(handles=[cool_patch, moderate_patch, warm_patch], title="Weather Conditions")
    plt.grid()
    plt.show()

# 3D PCA Plot with specific colors
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot each point in 3D with specific colors
scatter = ax.scatter(data_pca[:, 0], data_pca[:, 1], data_pca[:, 2], c=colors, alpha=0.6)

# Set labels and title
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
ax.set_title('3D PCA of Weather Features')

# Create legend for 3D plot
ax.legend(handles=[cool_patch, moderate_patch, warm_patch], title="Weather Conditions")

plt.show()
```

b. SARIMA

Preparation for Model and Functions

```
#SARIMA Model Building
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_predict

# Define the date ranges for historical and comparison data
start_date_2009 = pd.to_datetime('2009-01-01')
end_date_2018 = pd.to_datetime('2018-12-31')
start_date_2019 = pd.to_datetime('2019-01-01')
end_date_2019 = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018 and 2019 separately
filtered_data_actual = filtered_data[(filtered_data.index >= start_date_2009) & (filtered_data.index <= end_date_2018)]
filtered_data_comparison = filtered_data[(filtered_data.index >= start_date_2019) & (filtered_data.index <= end_date_2019)]

# Resample to weekly data
df_weekly = filtered_data_actual.resample('W').mean()
df_weekly_comparison = filtered_data_comparison.resample('W').mean()

# Define the function to find and remove outliers based on IQR
def remove_outliers_iqr(df, numerical_columns):
    # Calculate the IQR for each numerical column
    Q1 = df[numerical_columns].quantile(0.25)
    Q3 = df[numerical_columns].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Removing rows where any column value is outside the IQR range (outliers)
    df_no_outliers = df[~((df[numerical_columns] < lower_bound) | (df[numerical_columns] > upper_bound)).any(axis=1)]

    return df_no_outliers

# List of numerical columns to check for outliers
numerical_columns = ['tempC']

# Remove outliers from both the filtered data sets
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)
df_weekly_comparison_no_outliers = remove_outliers_iqr(df_weekly_comparison, numerical_columns)

# Prepare the data series for further analysis or plotting
series = df_weekly_no_outliers['tempC']
comparison_series_2019 = df_weekly_comparison_no_outliers['tempC']

# ADF Test for checking if data is stationary
# ADF Test function
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    if result[1] > 0.05:
        print("Series is non-stationary")
    else:
        print("Series is stationary")

def adf_pvalue(series):
    result = adfuller(series)
    print('p-value:', result[1])
```

Testing to figure out the best values for p(AutoRegressive), d(differencing), q(MovingAverage)

```
▶ plt.figure(figsize=(10, 4))
plt.plot(series, label='The Genuine Series')
plt.title("The Genuine Series")
plt.legend()
plt.show()

# Original Series and ADF test
print("ADF Test on Original Series:")
adf_test(series)

# First Order Differencing
first_diff = series.diff().dropna()
print("\nADF Test on First Order Differencing:")
adf_test(first_diff)

# Second Order Differencing
second_diff = first_diff.diff().dropna()
print("\nADF Test on Second Order Differencing:")
adf_test(second_diff)

# Determine the p value
# PACF to determine AR terms (p)
# Plotting Original, First Differencing, Second Differencing
fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Partial Autocorrelation (PACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_pacf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_pacf(first_diff, ax=axes[1, 1])

# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_pacf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```

fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Autocorrelation (ACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_acf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_acf(first_diff, ax=axes[1, 1])

# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_acf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Determine the d value
adf_pvalue(series)
adf_pvalue(first_diff)
adf_pvalue(second_diff)

```

Configuring the SARIMA model and plot the predictions

```

p, d, q = 1, 1, 1 # Non-seasonal ARIMA parameters
P, D, Q, S = 1, 1, 1, 52 # Seasonal ARIMA parameters (yearly seasonality)

# Fit SARIMA model
sarima_model = SARIMAX(series, order=(p, d, q), seasonal_order=(P, D, Q, S))
sarima_model_fit = sarima_model.fit(disp=False)
print(sarima_model_fit.summary())

# Find the indices for these dates in the dataset
start_loc = len(df_weekly) - 1
end_loc = start_loc + 52

# Plot the SARIMA model's predictions
plt.figure(figsize=(12, 6))
plot_predict(sarima_model_fit, start=start_loc, end=end_loc, dynamic=False)

# Customize the plot
plt.title("SARIMA Model - Forecast Temperature Data for 2019")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend(["Forecast", "Confidence Interval"])
plt.show()

# Forecast 52 weeks ahead
forecast = sarima_model_fit.get_prediction(start=start_loc, end=end_loc)
forecast_mean = forecast.predicted_mean

# Define forecast dates
forecast_dates = pd.date_range(start=df_weekly.index[-1] + pd.Timedelta(weeks=1), periods=len(forecast_mean), freq='W')

# Plotting
plt.figure(figsize=(12, 6))

# Plot the forecasted values
plt.plot(forecast_dates, forecast_mean, color='blue', linewidth=1.5, label="Forecast (2019)")

# Plot the actual data up to 2018
plt.plot(df_weekly_no_outliers.index, df_weekly_no_outliers['tempc'], color='red', linewidth=1.5, label="Actual Data (2009-2018)")

# Plot the actual data for 2019 separately, without markers for a cleaner look
plt.plot(df_weekly_comparison_no_outliers.index, df_weekly_comparison_no_outliers['tempc'], color='red', linewidth=1.5, alpha=0.5, label="Actual Data (2019)")

# Customize the plot
plt.title("SARIMA Model - Forecast vs. Actual Temperature Data (2009-2019)")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()

```

SARIMA MODEL PREDICTION 2020

Preparation for the model and functions

```
[ ] ##SARIMA Model Building
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_predict
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

## Define the date ranges for historical and comparison data
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2019-12-31')

## Filter data for 2009-2018 and 2019 separately
filtered_data_actual = filtered_data[(filtered_data.index >= start_date) & (filtered_data.index <= end_date)]

## Resample to weekly data
df_weekly = filtered_data_actual.resample('W').mean()

## Define the function to find and remove outliers based on IQR
def remove_outliers_iqr(df, numerical_columns):
    # Calculate the IQR for each numerical column
    Q1 = df[numerical_columns].quantile(0.25)
    Q3 = df[numerical_columns].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Removing rows where any column value is outside the IQR range (outliers)
    df_no_outliers = df[~((df[numerical_columns] < lower_bound) | (df[numerical_columns] > upper_bound)).any(axis=1)]

    return df_no_outliers

## Define a function to calculate evaluation metrics
def evaluate_model(actual, predicted, model_name):
    mae = mean_absolute_error(actual, predicted)
    r2 = r2_score(actual, predicted)

    print(f"== Evaluation Metrics for {model_name} ==")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"R-squared (R2): {r2:.4f}")
    print("=====\\n")

    return mae, r2

## List of numerical columns to check for outliers
numerical_columns = ['tempC']

## Remove outliers from both the filtered data sets
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)

## Prepare the data series for further analysis or plotting
series = df_weekly_no_outliers['tempC']
```

Configuring the SARIMA model, prediction plot and evaluation

```
p, d, q = 1, 1, 1 # Non-seasonal ARIMA parameters
P, D, Q, S = 1, 1, 1, 52 # Seasonal ARIMA parameters (yearly seasonality)

# Fit SARIMA model
sarima_model = SARIMAX(series, order=(p, d, q), seasonal_order=(P, D, Q, S))
sarima_model_fit = sarima_model.fit(disp=False)
print(sarima_model_fit.summary())

# Find the indices for these dates in the dataset
start_loc = len(df_weekly) - 1
end_loc = start_loc + 52

# Plot the SARIMA model's predictions
plt.figure(figsize=(12, 6))
plot_predict(sarima_model_fit, start=start_loc, end=end_loc, dynamic=False)

# Customize the plot
plt.title("SARIMA Model - Forecast Temperature Data for 2019")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend(["Forecast", "Confidence Interval"])
plt.show()

# Define the new extended forecast range
new_forecast_periods = 52 # Forecast for the next 104 weeks (2 years as an example)
forecast_dates = pd.date_range(start=df_weekly.index[-1] + pd.Timedelta(weeks=1), periods=new_forecast_periods, freq='W')

# Generate forecasts for the extended range
extended_forecast = sarima_model_fit.get_forecast(steps=new_forecast_periods)
forecast_mean = extended_forecast.predicted_mean
forecast_ci = extended_forecast.conf_int() # Confidence intervals

# Plot the original data and extended forecast
plt.figure(figsize=(14, 7))

# Plot actual data
plt.plot(df_weekly_no_outliers.index, df_weekly_no_outliers['tempC'], color='red', label="Actual Data (2009-2019)")

# Plot the forecasted values
plt.plot(forecast_dates, forecast_mean, color='blue', linestyle='--', label="Forecast (2019-2024)")

# Plot confidence intervals
plt.fill_between(forecast_dates, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='blue', alpha=0.2, label="Confidence Interval")

# Customize the plot
plt.title("SARIMA Model - Forecast for 2020-2024 Temperature Data")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()

actual_data = series[-new_forecast_periods:] # Adjust to actual test data

# Evaluate SARIMAX
sarima_mae, sarima_r2 = evaluate_model(
    actual=actual_data,
    predicted=forecast_mean[:len(actual_data)],
    model_name="SARIMAX"
)
```

c. Random Forest, XGBoost, Decision Tree Regressor

Random Forest, XGBOOST, Decision Tree Regressor

Correlation Matrix

```
[ ] import pandas as pd
      import matplotlib.pyplot as plt

      # Calculate the correlation matrix
      correlation_matrix = df_weekly.corr()

      # Get the correlation of all features with 'tempC'
      correlation_with_tempC = correlation_matrix['tempC'].sort_values(ascending=False)

      # Display the correlation with tempC
      print("Correlation with tempC:")
      print(correlation_with_tempC)

      # Plot the correlation matrix
      plt.figure(figsize=(10, 8))
      plt.title("Correlation Matrix")
      plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
      plt.colorbar()
      plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=90)
      plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
      plt.show()
```

SelectKBest to find feature importance

```
[ ] import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler

# Pisahkan target (y) dan fitur kandidat (X)
X = df_weekly.drop(columns=['tempC', 'FeelsLikeC', 'HeatIndexC', 'minTempC', 'maxTempC']) # Semua kolom kecuali target dan yang mempunya korelasi terlalu tinggi
y = df_weekly['tempC'] # Kolom target

# Pastikan data bersih dari NaN
X = X.fillna(X.mean())
y = y.fillna(y.mean())

# Standarisasi data (untuk kestabilan model)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Gunakan SelectKBest dengan F-value sebagai metode seleksi
k_best_selector = SelectKBest(score_func=f_regression, k='all') # Pilih semua untuk melihat skor
X_new = k_best_selector.fit_transform(X_scaled, y)

# Tampilkan skor fitur
scores = k_best_selector.scores_
features = X.columns

# Gabungkan fitur dengan skor dalam bentuk DataFrame untuk analisis
feature_scores = pd.DataFrame({'Feature': features, 'Score': scores})
feature_scores = feature_scores.sort_values(by='Score', ascending=False)

# Cetak hasil
print("Ranking Fitur Berdasarkan Skor:")
print(feature_scores)

# Plot skor fitur
plt.figure(figsize=(10, 6))
plt.bar(feature_scores['Feature'], feature_scores['Score'], color='skyblue', edgecolor='black')
plt.title("Feature Importance Ranking", fontsize=16)
plt.xlabel("Features", fontsize=14)
plt.ylabel("Score (F-value)", fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Tampilkan plot
plt.show()
```

Prediksi menggunakan XGBoost, Random Forest, Decision Tree Regressor

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor, plot_tree
import xgboost as xgb
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.metrics import confusion_matrix, classification_report, r2_score
from sklearn import metrics
from IPython.display import Image, SVG
import seaborn as sns
import pydotplus
from io import StringIO
import os

# Reading the CSV file
file_path = 'mumbai.csv'
data = pd.read_csv(file_path)

# Select only the relevant columns
weather_data = data[['date_time', 'maxtempC', 'mintempC', 'humidity', 'precipMM', 'tempC', 'windspeedKmph', 'uvIndex', 'cloudcover', 'sunHour', 'DewPointC', 'FeelsLikeC', 'HeatIndexC']]

# Convert 'date_time' to datetime format (removing timezone awareness if it exists)
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'], utc=True).dt.tz_localize(None)

# Path for Graphviz (adjust if needed)
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'

# convert 'date_time' to datetime and set it as the index
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'])
weather_data.set_index('date_time', inplace=True)

# Resample data to weekly intervals (e.g., mean for temperature, sum for precipitation)
weekly_data = weather_data.resample('W').agg({
    'maxtempC': 'mean', # Average max temperature per week
    'mintempC': 'mean', # Average min temperature per week
    'tempC': 'mean', # Average temperature per week
    'humidity': 'mean', # Average humidity per week
    'precipMM': 'sum', # Total precipitation per week
    'windspeedKmph': 'mean', # Average wind speed per week
    'DewPointC': 'mean', # Average dew point temperature per week
    'HeatIndexC': 'mean', # Average heat index per week
    'cloudcover': 'mean', # Average cloud cover per week
    'uvIndex': 'mean', # Average UV Index per week
    'sunHour': 'mean' # Average sunlight hours per week
})
```

```

# Add the additional feature 'temp_diff' to the weekly resampled data
weekly_data['temp_diff'] = weekly_data['maxtempC'] - weekly_data['mintempC']

# Reset index to bring 'date_time' back as a column
weekly_data.reset_index(inplace=True)

# Add 'year' column for filtering
weekly_data['year'] = weekly_data['date_time'].dt.year

# Filter data for training and testing
train_data = weekly_data[weekly_data['year'].between(2009, 2018)] #80% data training
test_data = weekly_data[weekly_data['year'].between(2019, 2020)] #20% data testing

# Define features and targets as before
feature_cols = ['uvIndex', 'humidity', 'cloudcover', 'DewPointC', 'windspeedKmph']
x_train = train_data[feature_cols]
y_train_reg = train_data['tempC']

x_test = test_data[feature_cols]
y_test_reg = test_data['tempC']

# Filter data for the full time frame (2009-2020)
full_data = weekly_data[weekly_data['year'].between(2009, 2018)]

# Define features and targets
X_full = full_data[feature_cols]
y_full_reg = full_data['tempC']

# -----
# Random Forest Regressor
# -----
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train_reg)
rf_y_pred = rf_reg.predict(X_test)
rf_r2 = r2_score(y_test_reg, rf_y_pred)
print(f"Random Forest R-Squared: {rf_r2:.2f}")

# -----
# XGBoost Regressor
# -----
xgb_reg = XGBRegressor(random_state=42, eval_metric='rmse')
xgb_reg.fit(X_train, y_train_reg)
xgb_y_pred = xgb_reg.predict(X_test)
xgb_r2 = r2_score(y_test_reg, xgb_y_pred)
print(f"XGBoost R-Squared: {xgb_r2:.2f}")

# -----
# Decision Tree Regressor
# -----
dt_reg = DecisionTreeRegressor(max_depth=8, random_state=42)
dt_reg.fit(X_train, y_train_reg)
dt_y_pred = dt_reg.predict(X_test)
dt_r2 = r2_score(y_test_reg, dt_y_pred)
print(f"Decision Tree R-Squared: {dt_r2:.2f}")

```

```
# -----
# Plot 1: Actual vs Predicted (Random Forest)
# -----
plt.figure(figsize=(10, 6))
plt.plot(full_data['date_time'], y_full_reg, label='Actual Values', color='blue', linewidth=2) # Use y_full_reg here
plt.plot(test_data['date_time'], rf_y_pred, label='Predicted Values (Random Forest)', color='orange', linestyle='--')
plt.title('Random Forest: Actual vs Predicted')
plt.xlabel('Date Time')
plt.ylabel('Target Value')
plt.legend()
plt.grid(True)
plt.show()

# -----
# Plot 2: Actual vs Predicted (XGBoost)
# -----
plt.figure(figsize=(10, 6))
plt.plot(full_data['date_time'], y_full_reg, label='Actual Values', color='blue', linewidth=2) # Use y_full_reg here
plt.plot(test_data['date_time'], xgb_y_pred, label='Predicted Values (XGBoost)', color='green', linestyle='--')
plt.title('XGBoost: Actual vs Predicted')
plt.xlabel('Date Time')
plt.ylabel('Target Value')
plt.legend()
plt.grid(True)
plt.show()

# -----
# Plot 3: Actual vs Predicted (DecisionTree)
# -----
plt.figure(figsize=(10, 6))
plt.plot(full_data['date_time'], y_full_reg, label='Actual', color='blue', linewidth=2)
plt.plot(test_data['date_time'], dt_y_pred, label='Predicted (Decision Tree)', color='orange', linestyle='--', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Temperature (°C)')
plt.title('Decision Tree: Actual vs Predicted (2009-2020)')
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

d. SARIMAX

Preparation for the model and functions

```
▶ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define the date ranges for historical and comparison data
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018
filtered_data_actual = filtered_data[(filtered_data.index >= start_date) & (filtered_data.index <= end_date)]

# Augmenting hourly data to weekly data
df_weekly = filtered_data_actual.resample('W').mean()

# Function to remove outliers using IQR
def remove_outliers_iqr(df, numerical_columns):
    Q1 = df[numerical_columns].quantile(0.25)
    Q3 = df[numerical_columns].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df[~((df[numerical_columns] < lower_bound) | (df[numerical_columns] > upper_bound)).any(axis=1)]
    return df_no_outliers

# Define a function to calculate evaluation metrics
def evaluate_model(actual, predicted, model_name):
    mae = mean_absolute_error(actual, predicted)
    r2 = r2_score(actual, predicted)

    print(f"--- Evaluation Metrics for {model_name} ---")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"R-squared (R²): {r2:.4f}")
    print("=====\\n")

    return mae, r2

# List of numerical columns to check for outliers
numerical_columns = ['tempC']

# Remove outliers
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)

# Add the additional feature 'temp_diff' to the weekly resampled data
df_weekly_no_outliers['temp_diff'] = df_weekly_no_outliers['maxtempC'] - df_weekly_no_outliers['mintempC']

# Prepare data series and exogenous variables
series = df_weekly_no_outliers['tempC']
exogenous_columns = ['uvIndex', 'humidity', 'cloudcover', 'DewPointC', 'windspeedKmph', 'sunHour', 'temp_diff', 'precipMM']
exog_data = df_weekly_no_outliers[exogenous_columns]
```

```
Configuring the SARIMAX model, prediction plot, and evaluation
```

```
[ ] # SARIMA model parameters
p, d, q = 1, 1, 1
P, D, Q, S = 1, 1, 1, 52

# Fit SARIMAX model with exogenous variables
sarimax_model = SARIMAX(series, exog=exog_data, order=(p, d, q), seasonal_order=(P, D, Q, S))
sarimax_model_fit = sarimax_model.fit(disp=False)
print(sarimax_model_fit.summary())

# Forecast periods and future exogenous data
new_forecast_periods = 52 # Forecast for a year
forecast_dates = pd.date_range(start=df_weekly.index[-1] + pd.Timedelta(weeks=1), periods=new_forecast_periods, freq='W')

# Generate forecasts for SARIMAX
exog_forecast = exog_data.iloc[-new_forecast_periods:] # Use future exogenous data if available
sarimax_forecast = sarimax_model_fit.get_forecast(steps=new_forecast_periods, exog=exog_forecast)
sarimax_forecast_mean = sarimax_forecast.predicted_mean
sarimax_forecast_ci = sarimax_forecast.conf_int()

# Plot SARIMA and SARIMAX forecasts
plt.figure(figsize=(14, 7))

# Plot actual data
plt.plot(df_weekly_no_outliers.index, df_weekly_no_outliers['tempC'], color='red', label="Actual Data (2009-2019)")

# Plot SARIMAX forecast
plt.plot(forecast_dates, sarimax_forecast_mean, color='green', linestyle='--', label="SARIMAX Forecast (2020-2024)")
plt.fill_between(forecast_dates, sarimax_forecast_ci.iloc[:, 0], sarimax_forecast_ci.iloc[:, 1], color='green', alpha=0.2, label="SARIMAX Confidence Interval")

# Customize the plot
plt.title("SARIMAX - Forecast for 2020-2024 Temperature Data")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.show()

actual_data = series[-new_forecast_periods:] # Adjust to actual test data

# Evaluate SARIMAX
sarimax_mae, sarimax_r2 = evaluate_model(
    actual=actual_data,
    predicted=sarimax_forecast_mean[:len(actual_data)],
    model_name="SARIMAX"
)
```

```
[ ] # -----
# Plot 4: Comparison of R-Squared Values for Different Models
# -----

# Create a list of model names and their corresponding R-squared values
models = ['Random Forest', 'XGBoost', 'Decision Tree', 'SARIMA', 'SARIMAX']
r_squared_values = [rf_r2, xgb_r2, dt_r2, sarima_r2, sarimax_r2]

# Create a bar plot
plt.figure(figsize=(10, 6)) # Set the figure size
bars = plt.bar(models, r_squared_values, color=[ 'lightblue', 'lightgreen', 'lightcoral', 'lightyellow', 'lightpink' ], edgecolor='black')

# Adding title and labels
plt.title('Model Comparison: R-Squared', fontsize=16)
plt.xlabel('Models', fontsize=14)
plt.ylabel('R-Squared Value', fontsize=14)
plt.ylim(0, 1) # Set y-axis limit from 0 to 1

# Add value labels on top of each bar for clarity
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.02, f'{yval:.2f}', ha='center', fontsize=12)

# Display grid for better readability
plt.grid(axis='y', linestyle='---', alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```

Best Model for our weather data : SARIMAX

Finding the best parameter SARIMAX for humidity

```
[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_predict

# Define the date ranges for historical and comparison data
start_date_2009 = pd.to_datetime('2009-01-01')
end_date_2018 = pd.to_datetime('2018-12-31')
start_date_2019 = pd.to_datetime('2019-01-01')
end_date_2019 = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018 and 2019 separately
filtered_data_actual = filtered_data[(filtered_data.index >= start_date_2009) & (filtered_data.index <= end_date_2018)]
filtered_data_comparison = filtered_data[(filtered_data.index >= start_date_2019) & (filtered_data.index <= end_date_2019)]

# Resample to weekly data
df_weekly = filtered_data_actual.resample('W').mean()
df_weekly_comparison = filtered_data_comparison.resample('W').mean()

# Define the function to find and remove outliers based on IQR
def remove_outliers_iqr(df, numerical_columns):
    # Calculate the IQR for each numerical column
    Q1 = df[numerical_columns].quantile(0.25)
    Q3 = df[numerical_columns].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Removing rows where any column value is outside the IQR range (outliers)
    df_no_outliers = df[~((df[numerical_columns] < lower_bound) | (df[numerical_columns] > upper_bound)).any(axis=1)]

    return df_no_outliers

# List of numerical columns to check for outliers
numerical_columns = ['humidity']

# Remove outliers from both the filtered data sets
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)
df_weekly_comparison_no_outliers = remove_outliers_iqr(df_weekly_comparison, numerical_columns)

# Prepare the data series for further analysis or plotting
series = df_weekly_no_outliers['humidity']
comparison_series_2019 = df_weekly_comparison_no_outliers['humidity']
```

```

# ADF TEST FUNCTION
[ ] def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    if result[1] > 0.05:
        print("Series is non-stationary")
    else:
        print("Series is stationary")

def adf_pvalue(series):
    result = adfuller(series)
    print('p-value:', result[1])

# SARIMA Plot

# Plot the original time series
plt.figure(figsize=(10, 4))
plt.plot(series, label='The Genuine Series')
plt.title("The Genuine Series")
plt.legend()
plt.show()

# Original Series and ADF test
print("ADF Test on Original Series:")
adf_test(series)

# First Order Differencing
first_diff = series.diff().dropna()
print("\nADF Test on First Order Differencing:")
adf_test(first_diff)

# Second Order Differencing
second_diff = first_diff.diff().dropna()
print("\nADF Test on Second Order Differencing:")
adf_test(second_diff)

# Determine the p value
# PACF to determine AR terms (p)
# Plotting Original, First Differencing, Second Differencing
fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Partial Autocorrelation (PACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_pacf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_pacf(first_diff, ax=axes[1, 1])

```

```
# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_pacf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Determine the q value
# ACF to determine MA terms (q)
# Plotting Original, First Differencing, Second Differencing
fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Autocorrelation (ACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_acf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_acf(first_diff, ax=axes[1, 1])

# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_acf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Determine the d value
adf_pvalue(series)
adf_pvalue(first_diff)
adf_pvalue(second_diff)
```

P=1, Q=1, D=1

Finding Exogenous variable and SARIMAX model for humidity

```
[ ] import pandas as pd
      import matplotlib.pyplot as plt

      # Calculate the correlation matrix
      correlation_matrix = df_weekly_no_outliers.corr()

      # Get the correlation of all features with 'tempC'
      correlation_with_humidity = correlation_matrix['humidity'].sort_values(ascending=False)

      # Display the correlation with tempC
      print("Correlation with humidity:")
      print(correlation_with_humidity)

      # Plot the correlation matrix
      plt.figure(figsize=(10, 8))
      plt.title("Correlation Matrix")
      plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
      plt.colorbar()
      plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=90)
      plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
      plt.show()
```

```
▶ import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler

# Pisahkan target (y) dan fitur kandidat (X)
X = df_weekly.drop(columns=['humidity']) # Semua kolom kecuali target dan yang mempunya korelasi terlalu tinggi
y = df_weekly['humidity'] # Kolom target

# Pastikan data bersih dari NaN
X = X.fillna(X.mean())
y = y.fillna(y.mean())

# Standarisasi data (untuk kestabilan model)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Gunakan SelectKBest dengan F-value sebagai metode seleksi
k_best_selector = SelectKBest(score_func=f_regression, k='all') # Pilih semua untuk melihat skor
X_new = k_best_selector.fit_transform(X_scaled, y)

# Tampilkan skor fitur
scores = k_best_selector.scores_
features = X.columns

# Gabungkan fitur dengan skor dalam bentuk DataFrame untuk analisis
feature_scores = pd.DataFrame({'Feature': features, 'Score': scores})
feature_scores = feature_scores.sort_values(by='Score', ascending=False)

# Cetak hasil
print("Ranking Fitur Berdasarkan Skor:")
print(feature_scores)

# Plot skor fitur
plt.figure(figsize=(10, 6))
plt.bar(feature_scores['Feature'], feature_scores['Score'], color='skyblue', edgecolor='black')
plt.title("Feature Importance Ranking", fontsize=16)
plt.xlabel("Features", fontsize=14)
plt.ylabel("Score (F-value)", fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Tampilkan plot
plt.show()
```

```

❶ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define the date ranges for historical and comparison data
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018
filtered_data_actual = filtered_data[(filtered_data.index >= start_date) & (filtered_data.index <= end_date)]

# Augmenting hourly data to weekly data
df_weekly = filtered_data_actual.resample('W').mean()

# Define a function to calculate evaluation metrics
def evaluate_model(actual, predicted, model_name):
    mae = mean_absolute_error(actual, predicted)
    r2 = r2_score(actual, predicted)

    print(f"==== Evaluation Metrics for {model_name} ===")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"R-squared (R2): {r2:.4f}")
    print("=====\\n")

    return mae, r2

# List of numerical columns to check for outliers
numerical_columns = ['humidity']

# Remove outliers
df_weekly_no_outliers = df_weekly

# Add the additional feature 'temp_diff' to the weekly resampled data
df_weekly_no_outliers['temp_diff'] = df_weekly_no_outliers['maxtempc'] - df_weekly_no_outliers['mintempc']

# Prepare data series and exogenous variables
series = df_weekly_no_outliers['humidity']
exogenous_columns = ['uvIndex', 'DewPointC', 'cloudcover']
exog_data = df_weekly_no_outliers[exogenous_columns]
# SARIMA model parameters
p, d, q = 1, 1, 1
P, D, Q = 1, 1, 1, 52

# Fit SARIMAX model with exogenous variables
sarimax_model = SARIMAX(series, exog=exog_data, order=(p, d, q), seasonal_order=(P, D, Q, S))
sarimax_model_fit = sarimax_model.fit(disp=False)
print(sarimax_model_fit.summary())

```

```

# Forecast periods and future exogenous data
new_forecast_periods = 208 # Forecast for 2 years (weekly)
forecast_dates = pd.date_range(start=df_weekly.index[-1] + pd.Timedelta(weeks=1), periods=new_forecast_periods, freq='W')

# Generate forecasts for SARIMAX
exog_forecast = exog_data.iloc[-new_forecast_periods:] # Use future exogenous data if available
sarimax_forecast = sarimax_model_fit.get_forecast(steps=new_forecast_periods, exog=exog_forecast)
sarimax_forecast_mean = sarimax_forecast.predicted_mean
sarimax_forecast_ci = sarimax_forecast.conf_int()

actual_data = series[-new_forecast_periods:] # Adjust to actual test data

# Evaluate SARIMAX
sarimax_mae, sarimax_r2 = evaluate_model(
    actual=actual_data,
    predicted=sarimax_forecast_mean[:len(actual_data)],
    model_name="SARIMAX"
)

```

Finding the best parameter SARIMAX for Windspeed

```

[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_predict

# Define the date ranges for historical and comparison data
start_date_2009 = pd.to_datetime('2009-01-01')
end_date_2018 = pd.to_datetime('2018-12-31')
start_date_2019 = pd.to_datetime('2019-01-01')
end_date_2019 = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018 and 2019 separately
filtered_data_actual = filtered_data[(filtered_data.index >= start_date_2009) & (filtered_data.index <= end_date_2018)]
filtered_data_comparison = filtered_data[(filtered_data.index >= start_date_2019) & (filtered_data.index <= end_date_2019)]

# Resample to weekly data
df_weekly = filtered_data_actual.resample('W').mean()
df_weekly_comparison = filtered_data_comparison.resample('W').mean()

# Define the function to find and remove outliers based on IQR
def remove_outliers_iqr(df, numerical_columns):
    # Calculate the IQR for each numerical column
    Q1 = df[numerical_columns].quantile(0.25)
    Q3 = df[numerical_columns].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Removing rows where any column value is outside the IQR range (outliers)
    df_no_outliers = df[~((df[numerical_columns] < lower_bound) | (df[numerical_columns] > upper_bound)).any(axis=1)]

    return df_no_outliers

# List of numerical columns to check for outliers
numerical_columns = ['windspeedKmph']

# Remove outliers from both the filtered data sets
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)
df_weekly_comparison_no_outliers = remove_outliers_iqr(df_weekly_comparison, numerical_columns)

# Prepare the data series for further analysis or plotting
series = df_weekly_no_outliers['windspeedKmph']
comparison_series_2019 = df_weekly_comparison_no_outliers['windspeedKmph']

```

```
# ADF TEST FUNCTION
def adf_test(series):
    result = adfuller(series)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    if result[1] > 0.05:
        print("Series is non-stationary")
    else:
        print("Series is stationary")

def adf_pvalue(series):
    result = adfuller(series)
    print('p-value:', result[1])

# SARIMA Plot

# Plot the original time series
plt.figure(figsize=(10, 4))
plt.plot(series, label='The Genuine Series')
plt.title("The Genuine Series")
plt.legend()
plt.show()

# Original Series and ADF test
print("ADF Test on Original Series:")
adf_test(series)

# First Order Differencing
first_diff = series.diff().dropna()
print("\nADF Test on First Order Differencing:")
adf_test(first_diff)

# Second Order Differencing
second_diff = first_diff.diff().dropna()
print("\nADF Test on Second Order Differencing:")
adf_test(second_diff)

# Determine the p value
# PACF to determine AR terms (p)
# Plotting Original, First Differencing, Second Differencing
fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Partial Autocorrelation (PACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_pacf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_pacf(first_diff, ax=axes[1, 1])
```

```
# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_pacf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Determine the q value
# ACF to determine MA terms (q)
# Plotting Original, First Differencing, Second Differencing
fig, axes = plt.subplots(3, 2, figsize=(14, 8))
fig.suptitle("Autocorrelation (ACF) Analysis")

# Plot the Genuine Series and its ACF
axes[0, 0].plot(series, color='blue')
axes[0, 0].set_title("The Genuine Series")
plot_acf(series.dropna(), ax=axes[0, 1])

# Plot First Order Differencing and its ACF
axes[1, 0].plot(first_diff, color='blue')
axes[1, 0].set_title("Order of Differencing: First")
plot_acf(first_diff, ax=axes[1, 1])

# Plot Second Order Differencing and its ACF
axes[2, 0].plot(second_diff, color='blue')
axes[2, 0].set_title("Order of Differencing: Second")
plot_acf(second_diff, ax=axes[2, 1])

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()

# Determine the d value
adf_pvalue(series)
adf_pvalue(first_diff)
adf_pvalue(second_diff)
```

Finding Exogenous variable and SARIMAX model for wind speed

```
▶ import pandas as pd
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = df_weekly_no_outliers.corr()

# Get the correlation of all features with 'tempC'
correlation_with_windspeedKmph = correlation_matrix['windspeedKmph'].sort_values(ascending=False)

# Display the correlation with tempC
print("Correlation with windspeedKmph:")
print(correlation_with_windspeedKmph)

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
plt.title("Correlation Matrix")
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
plt.colorbar()
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=90)
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.show()
```

```
▶ import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler

# Pisahkan target (y) dan fitur kandidat (X)
X = df_weekly.drop(columns=['windspeedKmph']) # Semua kolom kecuali target dan yang mempunya korelasi terlalu tinggi
y = df_weekly['windspeedKmph'] # Kolom target

# Pastikan data bersih dari NaN
X = X.fillna(X.mean())
y = y.fillna(y.mean())

# Standarisasi data (untuk kestabilan model)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Gunakan SelectKBest dengan F-value sebagai metode seleksi
k_best_selector = SelectKBest(score_func=f_regression, k='all') # Pilih semua untuk melihat skor
X_new = k_best_selector.fit_transform(X_scaled, y)

# Tampilkan skor fitur
scores = k_best_selector.scores_
features = X.columns

# Gabungkan fitur dengan skor dalam bentuk DataFrame untuk analisis
feature_scores = pd.DataFrame({'Feature': features, 'Score': scores})
feature_scores = feature_scores.sort_values(by='Score', ascending=False)

# Cetak hasil
print("Ranking Fitur Berdasarkan Skor:")
print(feature_scores)

# Plot skor fitur
plt.figure(figsize=(10, 6))
plt.bar(feature_scores['Feature'], feature_scores['Score'], color='skyblue', edgecolor='black')
plt.title("Feature Importance Ranking", fontsize=16)
plt.xlabel("Features", fontsize=14)
plt.ylabel("Score (F-value)", fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Tampilkan plot
plt.show()
```

```

❶ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define the date ranges for historical and comparison data
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2019-12-31')

# Filter data for 2009-2018
filtered_data_actual = filtered_data[(filtered_data.index >= start_date) & (filtered_data.index <= end_date)

# Augmenting hourly data to weekly data
df_weekly = filtered_data_actual.resample('W').mean()

# Define a function to calculate evaluation metrics
def evaluate_model(actual, predicted, model_name):
    mae = mean_absolute_error(actual, predicted)
    r2 = r2_score(actual, predicted)

    print(f"==== Evaluation Metrics for {model_name} ===")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"R-squared (R2): {r2:.4f}")
    print("=====\\n")

    return mae, r2

# List of numerical columns to check for outliers
numerical_columns = ['windspeedKmph']

# Remove outliers
df_weekly_no_outliers = df_weekly

# Add the additional feature 'temp_diff' to the weekly resampled data
df_weekly_no_outliers['temp_diff'] = df_weekly_no_outliers['maxtempC'] - df_weekly_no_outliers['mintempC']

# Prepare data series and exogenous variables
series = df_weekly_no_outliers['windspeedKmph']
exogenous_columns = ['cloudcover', 'humidity', 'precipMM', 'uvIndex']
exog_data = df_weekly_no_outliers[exogenous_columns]
# SARIMA model parameters
p, d, q = 1, 1, 1
P, D, Q = 1, 1, 1, 52

# Fit SARIMAX model with exogenous variables
sarimax_model = SARIMAX(series, exog=exog_data, order=(p, d, q), seasonal_order=(P, D, Q, S))
sarimax_model_fit = sarimax_model.fit(disp=False)

```

```

# Forecast periods and future exogenous data
new_forecast_periods = 208 # Forecast for 2 years (weekly)
forecast_dates = pd.date_range(start=df_weekly.index[-1] + pd.Timedelta(weeks=1), periods=new_forecast_periods, freq='W')

# Generate forecasts for SARIMAX
exog_forecast = exog_data.iloc[-new_forecast_periods:] # Use future exogenous data if available
sarimax_forecast = sarimax_model_fit.get_forecast(steps=new_forecast_periods, exog=exog_forecast)
sarimax_forecast_mean = sarimax_forecast.predicted_mean
sarimax_forecast_ci = sarimax_forecast.conf_int()

actual_data = series[-new_forecast_periods:] # Adjust to actual test data

# Evaluate SARIMAX
sarimax_mae, sarimax_r2 = evaluate_model(
    actual=actual_data,
    predicted=sarimax_forecast_mean[:len(actual_data)],
    model_name="SARIMAX"
)

```

Mencoba model lain untuk WindspeedKmph

```

[ ] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor, plot_tree
import xgboost as xgb
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.metrics import confusion_matrix, classification_report, r2_score
from sklearn import metrics
from IPython.display import Image, SVG
import seaborn as sns
import pydotplus
from io import StringIO
import os

# Reading the CSV file
file_path = 'mumbai.csv'
data = pd.read_csv(file_path)

# select only the relevant columns
weather_data = data[['date_time', 'maxtempC', 'mintempC', 'humidity', 'precipMM', 'tempC', 'windspeedKmph', 'uvIndex', 'cloudcover', 'sunHour', 'DewPointC', 'FeelsLikeC', 'HeatIndexC']]

# convert 'date_time' to datetime format (removing timezone awareness if it exists)
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'], utc=True).dt.tz_localize(None)

# Path for Graphviz (adjust if needed)
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'

# convert 'date_time' to datetime and set it as the index
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'])
weather_data.set_index('date_time', inplace=True)

# Resample data to weekly intervals (e.g., mean for temperature, sum for precipitation)
weekly_data = weather_data.resample('W').agg({
    'maxtempC': 'mean', # Average max temperature per week
    'mintempC': 'mean', # Average min temperature per week
    'tempC': 'mean', # Average temperature per week
    'humidity': 'mean', # Average humidity per week
    'precipMM': 'sum', # Total precipitation per week
    'windspeedKmph': 'mean', # Average wind speed per week
    'DewPointC': 'mean', # Average dew point temperature per week
    'HeatIndexC': 'mean', # Average heat index per week
    'cloudcover': 'mean', # Average cloud cover per week
    'uvIndex': 'mean', # Average UV Index per week
    'sunHour': 'mean' # Average sunlight hours per week
})

```

```

[ ] # Reset index to bring 'date_time' back as a column
weekly_data.reset_index(inplace=True)

# Add 'year' column for filtering
weekly_data['year'] = weekly_data['date_time'].dt.year

# Filter data for training and testing
train_data = weekly_data[weekly_data['year'].between(2009, 2018)] #80% data training
test_data = weekly_data[weekly_data['year'].between(2019, 2020)] #20% data testing

# Define features and targets as before
feature_cols = ['cloudcover', 'humidity', 'precipMM', 'uvIndex']
X_train = train_data[feature_cols]
y_train_reg = train_data['windspeedKmph']

X_test = test_data[feature_cols]
y_test_reg = test_data['windspeedKmph']

# Filter data for the full time frame (2009-2020)
full_data = weekly_data[weekly_data['year'].between(2009, 2018)]

# Define features and targets
X_full = full_data[feature_cols]
y_full_reg = full_data['windspeedKmph']

# -----
# Random Forest Regressor
# -----
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train_reg)
rf_y_pred = rf_reg.predict(X_test)
rf_r2 = r2_score(y_test_reg, rf_y_pred)
print(f"Random Forest R-Squared: {rf_r2:.2f}")

# -----
# XGBoost Regressor
# -----
xgb_reg = XGBRegressor(random_state=42, eval_metric='rmse')
xgb_reg.fit(X_train, y_train_reg)
xgb_y_pred = xgb_reg.predict(X_test)
xgb_r2 = r2_score(y_test_reg, xgb_y_pred)
print(f"XGBoost R-Squared: {xgb_r2:.2f}")

# -----
# Decision Tree Regressor
# -----
dt_reg = DecisionTreeRegressor(max_depth=8, random_state=42)
dt_reg.fit(X_train, y_train_reg)
dt_y_pred = dt_reg.predict(X_test)
dt_r2 = r2_score(y_test_reg, dt_y_pred)
print(f"Decision Tree R-Squared: {dt_r2:.2f}")

```

F. Presentation and Automation

```
import pandas as pd
import matplotlib.pyplot as plt
from tkinter import Tk, Label, Button, OptionMenu, StringVar, IntVar, Frame
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from statsmodels.tsa.statespace.sarimax import SARIMAX
import numpy as np
from sklearn.metrics import mean_absolute_error, r2_score

# Reading the CSV file
file_path = 'mumbai.csv'
data = pd.read_csv(file_path)

# Select only the relevant columns
weather_data = data[['date_time', 'maxtempC', 'mintempC', 'humidity', 'precipMM', 'tempC',
                     'windspeedKmph', 'uvIndex', 'cloudcover', 'sunHour', 'DewPointC',
                     'FeelsLikeC', 'HeatIndexC']]

# Convert 'date_time' to datetime format and remove timezone awareness if it exists
weather_data['date_time'] = pd.to_datetime(weather_data['date_time'], utc=True).dt.tz_localize(None)

# Filter data within the specified date range
start_date = pd.to_datetime('2009-01-01')
end_date = pd.to_datetime('2019-12-31')
filtered_data = weather_data[(weather_data['date_time'] >= start_date) & (weather_data['date_time'] <= end_date)]

# Set 'date_time' as the index
filtered_data.set_index('date_time', inplace=True)

# Dropdown options
columns = ['tempC', 'windspeedKmph', 'humidity']
durations = ['Monthly', 'Weekly', 'Daily']
years = list(range(2009, 2025))

def plot_weather_data(column, duration, year):
    # Filter data for the selected year
    year_data = filtered_data[filtered_data.index.year == year]

    # Resample data based on duration
    if duration == "Monthly":
        resampled_data = year_data.resample('M').mean()
    elif duration == "Weekly":
        resampled_data = year_data.resample('W').mean()
    elif duration == "Daily":
        resampled_data = year_data
```

```

# Extract max and min values for the selected column
col_max = resampled_data[column].max()
col_min = resampled_data[column].min()
col_max_date = resampled_data[resampled_data[column] == col_max].index[0]
col_min_date = resampled_data[resampled_data[column] == col_min].index[0]

# Plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(resampled_data.index, resampled_data[column], label=f'{column} Trend', color='green')
ax.scatter(col_max_date, col_max, color='darkred', label=f'Max: {col_max:.2f} on {col_max_date.date()}', zorder=5)
ax.scatter(col_min_date, col_min, color='blue', label=f'Min: {col_min:.2f} on {col_min_date.date()}', zorder=5)
ax.set_title(f'{column} Analysis for {year} ({duration} Data)')
ax.set_xlabel('Date')
ax.set_ylabel(column)
ax.legend()
ax.grid(True)
return fig

# Augmenting hourly data to weekly data
df_weekly = filtered_data.resample('W').mean()

# Function to remove outliers using IQR, excluding windspeedKmph from outlier removal
def remove_outliers_iqr(df, numerical_columns, exclude_columns=[]):
    # Remove specified columns from the outlier check
    columns_to_check = [col for col in numerical_columns if col not in exclude_columns]

    Q1 = df[columns_to_check].quantile(0.25)
    Q3 = df[columns_to_check].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df[~((df[columns_to_check] < lower_bound) | (df[columns_to_check] > upper_bound)).any(axis=1)]
    return df_no_outliers

# List of numerical columns to check for outliers
numerical_columns = ['tempC', 'humidity']

# Remove outliers for the other columns, but not for windspeedKmph
df_weekly_no_outliers = remove_outliers_iqr(df_weekly, numerical_columns)

# Add the additional feature 'temp_diff' to the weekly resampled data
df_weekly_no_outliers['temp_diff'] = df_weekly_no_outliers['maxtempC'] - df_weekly_no_outliers['mintempC']

# Prepare data series and exogenous variables for SARIMAX
exogenous_columns = ['uvIndex', 'humidity', 'cloudcover', 'DewPointC', 'windspeedKmph', 'sunHour', 'temp_diff', 'precipMM']
exog_data = df_weekly_no_outliers[exogenous_columns]

```

```

# Define a function to train SARIMAX models for each target
def train_sarimax_model(target, exogenous_columns, order, seasonal_order):
    exog_data = df_weekly_no_outliers[exogenous_columns]
    model = SARIMAX(df_weekly_no_outliers[target], exog=exog_data, order=order, seasonal_order=seasonal_order)
    model_fit = model.fit(disp=False)
    return model_fit

# SARIMA model parameters
p, d, q = 1, 1, 1
P, D, Q = 1, 1, 1, 52

# Define the correct exogenous variables for each target
exog_columns_tempC = ['uvIndex', 'humidity', 'cloudcover', 'DewPointC', 'windspeedKmph', 'sunHour', 'temp_diff', 'precipMM']
exog_columns_humidity = ['uvIndex', 'DewPointC', 'cloudcover']

# Train models for each target
sarimax_models = {
    'tempC': train_sarimax_model('tempC', exog_columns_tempC, (p, d, q), (P, D, Q, S)),
    'humidity': train_sarimax_model('humidity', exog_columns_humidity, (p, d, q), (P, D, Q, S))
}

# Function to forecast using the trained models and assign a DatetimeIndex
def forecast_sarimax(model_fit, exog_columns=None, steps=260, start_date='2020-01-01', freq='W'):
    if exog_columns:
        exog_forecast = df_weekly_no_outliers[exog_columns].iloc[-steps:] # Use future exogenous data if available
        forecast = model_fit.get_forecast(steps=steps, exog=exog_forecast)
    else:
        forecast = model_fit.get_forecast(steps=steps)

    # Assign a DatetimeIndex to forecast results
    forecast_dates = pd.date_range(start=start_date, periods=steps, freq=freq)
    predicted_mean = pd.Series(forecast.predicted_mean.values, index=forecast_dates)
    conf_int = pd.DataFrame(forecast.conf_int().values, index=forecast_dates, columns=forecast.conf_int().columns)

    return predicted_mean, conf_int

# Function to cap humidity predictions between 0 and 100
def cap_humidity(predicted_mean, conf_int):
    capped_mean = predicted_mean.clip(lower=0, upper=100)
    capped_conf_int = conf_int.clip(lower=0, upper=100)
    return capped_mean, capped_conf_int

# Modify SARIMAX forecasts to include capping for humidity
sarimax_forecasts = {
    target: cap_humidity(*forecast_sarimax(
        sarimax_models[target],
        exog_columns=exog_columns_tempC if target == 'tempC'
    )
)
}

```

```

# Function to plot forecast
def plot_sarimax_forecast(column, duration, year):
    plt.clf() # Clear the current figure
    plt.close() # Close the current figure

    # Determine data and forecast dates based on duration
    if duration == 'Weekly':
        data = df_weekly_no_outliers
        forecast_mean, forecast_ci = sarimax_forecasts[column]
        forecast_dates_resampled = forecast_mean.index # Use the index directly
    elif duration == 'Monthly':
        data = df_weekly_no_outliers.resample('M').mean()
        forecast_mean, forecast_ci = sarimax_forecasts[column]
        # Resample forecasts to monthly frequency
        forecast_mean_resampled = forecast_mean.resample('M').mean()
        forecast_ci_resampled = forecast_ci.resample('M').mean()
        forecast_dates_resampled = forecast_mean_resampled.index # Use resampled index

    # Define the start and end dates for the year
    year_start_date = pd.to_datetime(f'{year}-01-01')
    year_end_date = pd.to_datetime(f'{year}-12-31')

    # Filter data for the specified year
    forecast_mean_filtered = forecast_mean_resampled[
        (forecast_mean_resampled.index >= year_start_date) & (forecast_mean_resampled.index <= year_end_date)
    ] if duration == 'Monthly' else forecast_mean[
        (forecast_mean.index >= year_start_date) & (forecast_mean.index <= year_end_date)
    ]

    forecast_ci_filtered = forecast_ci_resampled[
        (forecast_ci_resampled.index >= year_start_date) & (forecast_ci_resampled.index <= year_end_date)
    ] if duration == 'Monthly' else forecast_ci[
        (forecast_ci.index >= year_start_date) & (forecast_ci.index <= year_end_date)
    ]

    forecast_dates_filtered = forecast_mean_filtered.index # Use the filtered index

    # Find max and min values and their corresponding dates
    max_value = forecast_mean_filtered.max()
    max_date = forecast_mean_filtered.idxmax()
    min_value = forecast_mean_filtered.min()
    min_date = forecast_mean_filtered.idxmin()

```

```

# Plot the forecasts
fig, ax = plt.subplots(figsize=(14, 7))
ax.plot(
    forecast_dates_filtered,
    forecast_mean_filtered,
    color='green', linestyle='--', label=f"SARIMAX Forecast ({year})"
)
ax.fill_between(
    forecast_dates_filtered,
    forecast_ci_filtered.iloc[:, 0],
    forecast_ci_filtered.iloc[:, 1],
    color='green', alpha=0.2
)

# Plot max and min values as scatter points
ax.scatter(max_date, max_value, color='red', label=f'Max: {max_value:.2f} on {max_date.date()}', zorder=5)
ax.scatter(min_date, min_value, color='blue', label=f'Min: {min_value:.2f} on {min_date.date()}', zorder=5)

# Configure plot labels and legend
ax.set_title(f"SARIMAX Forecast for {column} - {year} ({duration})")
ax.set_xlabel("Date")
ax.set_ylabel(column)
ax.legend()
ax.grid(True)

return fig

def on_plot_trend():
    column = column_var.get()
    duration = duration_var.get()
    year = year_var.get()
    fig = plot_weather_data(column, duration, year)

    # Clear the previous plot
    for widget in plot_frame.winfo_children():
        widget.destroy()

    # Embed the new plot in the Tkinter window
    canvas = FigureCanvasTkAgg(fig, master=plot_frame)
    canvas.draw()
    canvas.get_tk_widget().pack()

def on_plot_prediksi():
    column = column_var.get()

```

```
def on_plot_prediksi():
    column = column_var.get()
    duration = duration_var.get()
    year = year_var.get()
    fig = plot_sarimax_forecast(column, duration, year)

    # Clear the previous plot
    for widget in plot_frame.winfo_children():
        widget.destroy()

    # Embed the new plot in the Tkinter window
    canvas = FigureCanvasTkAgg(fig, master=plot_frame)
    canvas.draw()
    canvas.get_tk_widget().pack()

# Create the main window
root = Tk()
root.title("Weather Data Plotter")

# Create and set the message text variable
column_var = StringVar(root)
column_var.set(columns[0]) # default value

duration_var = StringVar(root)
duration_var.set(durations[0]) # default value

year_var = IntVar(root)
year_var.set(years[0]) # default value

# Create and pack the widgets
Label(root, text="Select Column:").pack()
OptionMenu(root, column_var, *columns).pack()

Label(root, text="Select Duration:").pack()
OptionMenu(root, duration_var, *durations).pack()

Label(root, text="Select Year:").pack()
OptionMenu(root, year_var, *years).pack()

Button(root, text="Plot Trend", command=on_plot_trend).pack()
Button(root, text="Plot Prediksi", command=on_plot_prediksi).pack()
```

```
# Create a frame to hold the plot
plot_frame = Frame(root)
plot_frame.pack()

# Run the main loop
root.mainloop()
```