Statistic Fair Data Science UNS

Data Science Forecasting Weather Total Rainfall Day in Singapore

---

Overview

In recent years, the world has witnessed an increase in the frequency and intensity of extreme weather event. Singapore, as a highly urbanized tropical city-state, has also experienced significant weather related challenges, including intense rainfall leading to localized flooding and prolonged heatwaves affecting public health and urban livability.

The Meteorological Service Singapore (**MSS**). the national authority responsible for weather monitoring and forecasting, faces substantial challenges in delivering accurate and timely weather predictions. Despite operating **44 weather observation stations** across Singapore and continuously collecting daily data. Weather forecasting remains inherently complex due to dynamic atmospheric conditions and numerous influencing variables. Additionally, Singapore's compact geography, characterized by high-rise buildings, creates localized microclimates that demand highly precise, location-specific predictive models.

A critical issue in improving weather prediction lies in the underutilization of historical weather data stored in visual formats, as time series plots embedded in PDF reports, scientific articles, or archival documents. Extracting and digitizing this data can significantly enhance the quality and quantity of datasets available. for training advanced forecasting models.

Description

a. **Background**

Dalam beberapa tahun terakhir, negara menghadapi peningkatan frekuensi dengan intensitas cuaca ekstrem seperti banjir lokal dan gelombang panas, mendorong Badang Meteorologi Negara MSS Singapore, untuk mengembanagkan sistem prediksi yang lebih akurat meskipun memiliki 44 stasiun pengamatan, tantangan kondisi geografis yang kompleks dengan microclimate bervariasi dan sifat cuaca yang dinamis, dengan memanfaatkan data time series dalam format gambar plot cuaca historisis dari da dokumen csv tambahan, atau artikel ilmiah belum sepenuhnya tergarap untuk mendukung pemodelan prediksi yang lebih canggih oleh karena itu :

i. Banyak instansi meteorologi di Indonesia, seperti BMKG, memiliki arsip laporan cuaca dalam format PDF, buku, atau gambar. Dengan pendekatan yang sama, data ini dapat diubah menjadi dataset digital yang bisa digunakan untuk analisis jangka panjang.
ii. Indonesia memiliki variasi mikro iklim di setiap wilayah. Solusi ini bisa diadaptasi untuk meningkatkan presisi prakiraan cuaca di daerah-daerah rawan bencana seperti Jakarta, Bandung, Medan, dan Jayapura.
iii. Dengan prediksi cuaca yang lebih akurat, pemerintah daerah bisa lebih cepat merespons potensi bencana seperti banjir, tanah longsor, atau kekeringan, terutama di daerah pedesaan yang biasanya kurang terlayani oleh sistem peringatan dini.

iv. engembangan solusi ini akan mendorong peningkatan kapasitas SDM Indonesia dalam bidang teknologi digital, khususnya AI, machine learning, dan computer vision.

b. Tujuan

i. Banyak data cuaca historis hanya tersedia dalam bentuk visual seperti grafik pada dokumen PDF atau gambar statis. Dengan menggunakan teknik OCR dan computer vision, data tersebut dapat diekstraksi menjadi format numerik yang siap digunakan untuk analisis lebih lanjut.

ii. Dengan menambah jumlah dataset historis yang dapat diakses, model prediksi cuaca dapat dilatih dengan lebih baik, meningkatkan akurasi dan kemampuan generalisasi model dalam memprediksi kondisi cuaca ekstrem di masa depan.

iii. Melalui ekstraksi dan pemanfaatan data cuaca historis, pemerintah dan pemangku kebijakan dapat lebih proaktif dalam merancang mitigasi bencana cuaca, serta menyiapkan infrastruktur dan layanan publik yang lebih tangguh terhadap perubahan iklim.

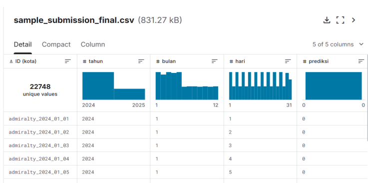
c. Requirement

Dataset ini berisi data cuaca historis dari 44 lokasi berbeda di seluruh Singapura, yang dikumpulkan dari data publik dari tahun 1980 hingga 2025. Pembagian Data
Train: Data cuaca historis untuk training model (1980-2023)
Test: Data cuaca untuk evaluasi model (2024-2025)
sample_submission.csv

i. Input & Output

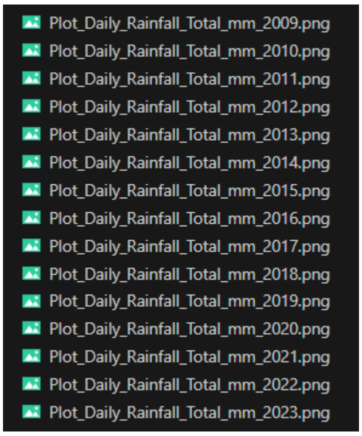
1. Input :
   a. Gambar plot data historisis
   b. Data CSV
2. Output :
   a. Prediksi regresi curah hujan masing masing kota (mm)
      format :

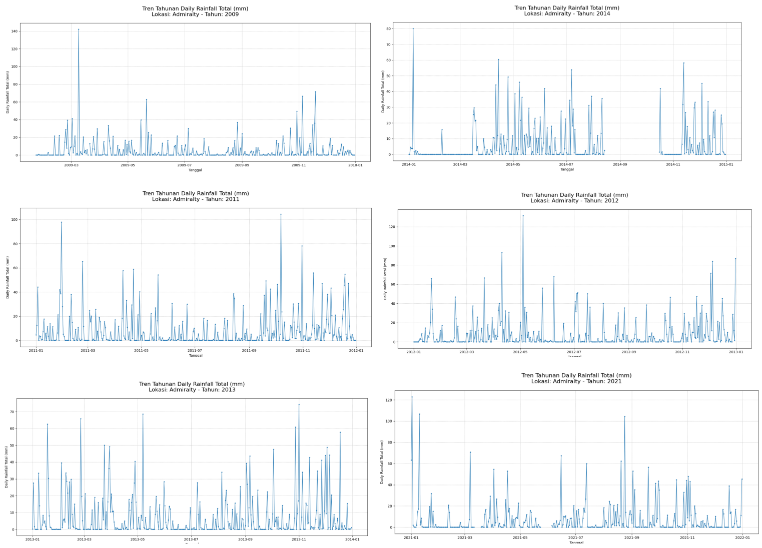
```
ID (kota),tahun,bulan,hari,prediksi
admiralty_2024_01_01,2024,1,1,4.43
admiralty_2024_01_02,2024,1,2,4.01
admiralty_2024_01_03,2024,1,3,4.32
admiralty_2024_01_04,2024,1,4,5.07
admiralty_2024_01_05,2024,1,5,5.05
admiralty_2024_01_06,2024,1,6,4.98
admiralty_2024_01_07,2024,1,7,6.56
```



ii. Features

1. Gambar Plot Historisis masing masing Kota (1980 - 2023)



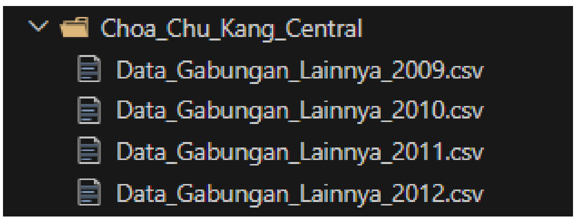


Feature Extract :

a. Feature Target (Daily Rainfall Total (mm))
Total curah hujan harian dalam mm untuk masing masing kota (44 kota) di negara Singapura menggunakan format data floating point.

b. Feature Waktu (Tahun-Bulan-Hari)
Waktu yang sesuai dengan prediksi curah hujan tersebut daily rainfall total dalam mm. untuk training dari tahun 1980 - 2023, sedangkan untuk testing dari 2024 - 2025 untuk prediksi curah hujan masing masing daerah untuk rainfall total.

2. Data tabular CSV



- Data Eksternal
  - AirQualityIndex_Google Trends.csv
  - Dipole Mode Index (DMI).csv
  - OceanicNinoIndex (ONI).csv
  - RelativeHumidityMonthlyMean.csv

```
Date,Highest 30 Min Rainfall (mm),Highest 60 Min
Rainfall (mm),Highest 120 Min Rainfall (mm),Mean
Temperature (°C),Maximum Temperature (°C),Minimum
Temperature (°C),Mean Wind Speed (km/h),Max Wind
Speed (km/h)
2015-01-01,0.0,0.0,0.0,25.8,27.2,24.7,11.5,32.8
2015-01-02,0.0,0.0,0.0,26.6,29.5,24.7,13.9,38.2
2015-01-03,0.0,0.0,0.0,26.8,30.3,24.2,13.2,36.4
```

a. Feature Data “Data_Gabungan_Lainnya_(tahun)”

Feature	Description	Unit	Data Type
Date	tanggal pengamatan	YYYY-MM-DD	Date
Highest 30 min Rainfall	curah hujan tertinggi 30 menit	milimeter	floating
Highest 60 min Rainfall	Curah hujan tertinggi dalam 60 menit	milimeter	floating
Highest 120 min Rainfall	Curah hujan tertinggi dalam 60 menit	milimeter	floating
Mean Temperature	Suhu rata-rata harian	Celsius	floating
Maximum Temperature	Suhu maksimum harian	Celsius	floating
Minimum Temperature	Suhu minimum harian	Celsius	floating
Mean Wind Speed	Kecepatan angin rata-rata	km/jam	floating
Max Wind Speed	Kecepatan angin maksimum	km/jam	floating

b. Feature Data “Data Eksternal/(DMI, ONI, Mean_RH)”

Feature	Description	Unit	Data Type
Date	tanggal pengamatan	YYYY-MM-DD	Date
AQI	Air Quality Index Google Trends	aqi	float
DMI	Dipole Mode Index	DMI	float
ONI	Oceanic Nino Index	ONI	float
Mean_rh	Relative Humidity Monthly Mean	mean_rh	float

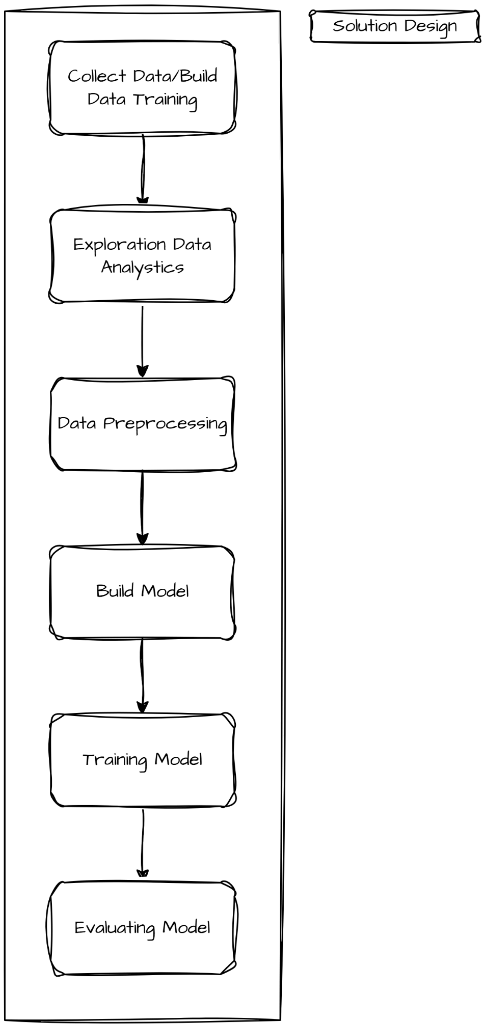
iii. Evaluation

Regression, menggunakan matriks evaluasi MSE, yang digunakan untuk mengukur error bias atau varian dari model terhadap prediksi data testing dengan testing pada kaggle, digunakan untuk memprediksi seakurat mungkin terhadap target

$$\mathrm{MSE} = \frac{1}{n} \sum (Y_i - \hat{Y}_i)^2$$

Solution Design

Alur solusi yang akan dilakukan dalam menangani masalah tersebut menggunakan machine learning dan deeplearning dari model, dimana proses untuk melakukan solution tersebut pada gambaran berikut :



Process yang akan dilakukan mulai dari pengumpulan/pembangunan data training atas bermacam macam source (plot gambar, data gabungan lainnya(csv), data eksternal(csv)). Pemodelan yang dilakukan menggunakan 2 model machine learning yang akan dilakukan perbandingan model

Solution

a. **Data Collect - Pipelinening**

(Training Data)

i. Read Data Plotting CSV

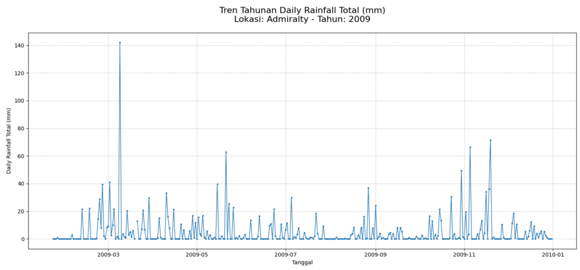
Membaca data plotting menggunakan OpenCV untuk pembacaan prediksi curah hujan Target (Daily_Rainfall_Total), mengekstrak dari gambar plot historisis data target masing masing kota dalam masing masing tahun

Read_historisis_opencv.ipynb
`def process_image(img_path, city_name, year):`

```
    img = cv2.imread(img_path)
    if img is None: return None

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    height, width = gray.shape

    mask = cv2.inRange(img, np.array([165,105,20]),
np.array([195, 135,40]))
    points = cv2.findNonZero(mask)

    plot_area_mask = cv2.inRange(img, np.array([200, 200,
200]), np.array([255, 255, 255]))

    rain_data = np.full(365, np.nan)

    X_LEFT, X_RIGHT = 100, width - 100
    Y_TOP, Y_BOTTOM = 50, height - 80
    x_scale = (X_RIGHT - X_LEFT)/365
    y_max_rain = 150

    baseline = Y_BOTTOM

    if points is not None:
        points = points[:,0,:]
        for x, y in points:
            if X_LEFT <= x <= X_RIGHT and Y_TOP <= y <=
Y_BOTTOM:
                day = int((x - X_LEFT) / x_scale)
                if 0 <= day < 365:
                    rainfall = ((baseline - y) / (baseline -
Y_TOP)) * y_max_rain
                    rainfall = max(0, rainfall)
                    if np.isnan(rain_data[day]) or rainfall >
rain_data[day]:
                        rain_data[day] = rainfall

    for day in range(365):
        if np.isnan(rain_data[day]):
            x = int(X_LEFT + day * x_scale)
            if X_LEFT <= x <= X_RIGHT:
                vertical_line = plot_area_mask[Y_TOP:Y_BOTTOM,
x]
                if np.any(vertical_line > 0):
                    rain_data[day] = 0

    start_date = datetime(year, 1, 1)
    return [
        [f"{city_name}_{(start_date +
timedelta(days=i)).strftime('%Y_%m_%d')}",
         (start_date +
timedelta(days=i)).strftime("%Y-%m-%d"),
         round(rain_data[i], 2) if not np.isnan(rain_data[i])
else np.nan,
         city_name]
        for i in range(365)
    ]

combined_data = []

for city_folder in os.listdir(src):
```

```
    city_path = os.path.join(src, city_folder)
    if not os.path.isdir(city_path): continue

    city_name = city_folder.lower()
    for img_file in sorted(os.listdir(city_path)):
        if img_file.startswith("Plot_Daily_Rainfall_") and
img_file.endswith(".png"):
            try:
                year =
int(img_file.split("_")[-1].split(".")[0])
                if data :=
process_image(os.path.join(city_path, img_file), city_name,
year):
                    combined_data.extend(data)
            except ValueError:
                continue

if combined_data:
    train_df = pd.DataFrame(combined_data, columns=['ID',
'date', 'prediksi', 'city'])
    train_df.to_csv('train_data.csv', index=False)
```

Input :



Output :

```
ID,Date,month,city,prediksi
admiralty_2009_01_10,2009-01-10,1,admiralty,0,1.32
admiralty_2009_01_11,2009-01-11,1,admiralty,0,1.32
admiralty_2009_01_12,2009-01-12,1,admiralty,0,1.32
admiralty_2009_01_13,2009-01-13,1,admiralty,0,2.11
admiralty_2009_01_14,2009-01-14,1,admiralty,0,2.11
admiralty_2009_01_15,2009-01-15,1,admiralty,0,1.32
```

ii. Filter Data

Melakukan filter data yang digunakan dengan melihat feature yang utuh yang dapat digunakan mulai tahun berapa, dimana pada kasus ini yang digunakan adalah data tahun >1982 yang dimana mengacu kepada kelengkapan feature yang dapat digunakan dengan memangkas dua tahun yang tidak dapat digunakan dikarenakan untuk feature **DMI, ONI** yang dimulai pada tahun **1982**

Filtering_data.ipynb
`train_df = pd.read_csv('train_data.csv')`

```
train_df['date'] = pd.to_datetime(train_df['date'])
train_df = train_df[train_df['date'].dt.year >= 1982]
train_df['month'] = train_df['date'].dt.month
train_df.to_csv('train_data.csv', index=False
```

iii. Aggregate / Concat dengan “Data_Gabungan_Lainnya”

Menambahkan atau memasukkan data gabungan lainnya yang berisikan feature feature yang ada dalam tabel diatas dengan memasukkannya ke dalam data training yang berisikan target dimana, akan digunakan sebagai kebutuhan data training untuk memuat banyak feature

Aggregate_Concatenate_Data_Gabungan_Lainnya.ipynb

```
train_df=pd.read_csv('train_data.csv')
train_df['date'] = pd.to_datetime(train_df['date'])
weather_data_list = []

for city_folder in os.listdir(src):
    city_path=os.path.join(src, city_folder)
    if not os.path.isdir(city_path):continue
    city_name = city_folder.lower()

    for filename in os.listdir(city_path):
        if filename.startswith('Data_Gabungan_Lainnya_') and
filename.endswith('.csv'):
            try:
                year =
int(filename.split('_')[-1].split('.')[0])
                file_path = os.path.join(city_path, filename)
                weather_df = pd.read_csv(file_path)

                weather_df.columns = [col.replace('min',
'Min') for col in weather_df.columns]
                weather_df.columns = [col.strip() for col in
weather_df.columns]

                # Convert date and add merge keys
                weather_df['Date'] =
pd.to_datetime(weather_df['Date'])
                weather_df['city'] = city_name
                weather_df['year'] =
weather_df['Date'].dt.year
                weather_df['month'] =
weather_df['Date'].dt.month
                weather_df['day'] = weather_df['Date'].dt.day

                weather_data_list.append(weather_df)

            except Exception as e:
                print(f"Error processing {filename}:
{str(e)}")
                continue

if weather_data_list:
    all_weather_data = pd.concat(weather_data_list,
ignore_index=True)
    duplicate_cols =
```

```
all_weather_data.columns[all_weather_data.columns.duplicated()
]
    if len(duplicate_cols)>0:
        all_weather_data =
all_weather_data.loc[:,~all_weather_data.columns.duplicated()]

    train_df['year'] = train_df['date'].dt.year
    train_df['month'] = train_df['date'].dt.month
    train_df['day'] = train_df['date'].dt.day

    merge_df = pd.merge(
        train_df,
        all_weather_data,
        how='left',
        on=['city','year','month','day']
    )

    merge_df = merge_df.drop(columns='Date')
    dup_cols = set([col for col in merge_df.columns if
merge_df.columns.tolist().count(col) > 1])
    if dup_cols:
        print(f"Duplicate columns in final data: {dup_cols}")
        merge_df =
merge_df.loc[:,~merge_df.columns.duplicated()]

    merge_df.to_csv('train_data.csv', index=False)

train_df = pd.read_csv('train_data.csv')
train_df['date']=pd.to_datetime(train_df['date'])
train_df['year_month']=train_df['date'].dt.to_period('M')

dmi=pd.read_csv('Data Eksternal/Dipole Mode Index (DMI).csv')
dmi['Date'] = pd.to_datetime(dmi['Date'])
dmi['year_month'] = dmi['Date'].dt.to_period('M')
dmi = dmi.rename(columns={' DMI HadISST1.1  missing value
-9999
https://psl.noaa.gov/data/timeseries/month/':'DMI'})[['year_mo
nth','DMI']]

oni=pd.read_csv('Data Eksternal/OceanicNinoIndex (ONI).csv')
oni['Date'] = pd.to_datetime(oni['Date'], format='%d/%m/%Y')
oni['year_month'] = oni['Date'].dt.to_period('M')
oni = oni.rename(columns={'
ONI':'ONI'})[['year_month','ONI']]

humi=pd.read_csv('Data
Eksternal/RelativeHumidityMonthlyMean.csv')
humi['year_month']=pd.to_datetime(humi['month']).dt.to_period(
'M')
humi = humi[['year_month','mean_rh']]

climate_data = dmi.merge(oni, on='year_month', how='outer')
climate_data = climate_data.merge(humi, on='year_month',
how='outer')
train_df = train_df.merge(climate_data, on='year_month',
how='left')

train_df = train_df.drop(columns='year_month')
train_df.to_csv('train_data.csv', index=False)
```

```
train_df.info()
```

Output :

	ID	Date	month	city	city_encoded	Highest_30min_Rainfall_mm	Highest_60min_Rainfall_mm	Highest_120min_Rainfall_mm	Mean_Temperature_C	Max_Temperature_C
0	admiralty_2009_01_10	2009-01-10	1	admiralty	0	NaN	NaN	NaN	NaN	NaN
1	admiralty_2009_01_11	2009-01-11	1	admiralty	0	NaN	NaN	NaN	NaN	NaN
2	admiralty_2009_01_12	2009-01-12	1	admiralty	0	NaN	NaN	NaN	NaN	NaN
3	admiralty_2009_01_13	2009-01-13	1	admiralty	0	NaN	NaN	NaN	NaN	NaN
4	admiralty_2009_01_14	2009-01-14	1	admiralty	0	NaN	NaN	NaN	NaN	NaN

	ID	Date	month	city	city_encoded	Highest_30min_Rainfall_mm	Highest_60min_Rainfall_mm	Highest_120min_Rainfall_mm	Mean_Temperature_C	Max_Temperatu
398322	whampoa_2023_12_27	2023-12-27	12	whampoa	43	3.6	3.8	4.8	NaN	N
398323	whampoa_2023_12_28	2023-12-28	12	whampoa	43	22.8	25.8	31.8	NaN	N
398324	whampoa_2023_12_29	2023-12-29	12	whampoa	43	1.2	1.8	2.6	NaN	N
398325	whampoa_2023_12_30	2023-12-30	12	whampoa	43	1.0	1.0	1.0	NaN	N
398326	whampoa_2023_12_31	2023-	12	whampoa	43	1.8	1.8	2.0	NaN	N

```
ID,Date,month,city,city_encoded,Highest_30min_Rainfall_mm,High
est_60min_Rainfall_mm,Highest_120min_Rainfall_mm,Mean_Temperat
ure_C,Max_Temperature_C,Min_Temperature_C,Mean_Wind_Speed_kmh,
Max_Wind_Speed_kmh,DMI,ONI,mean_rh,prediksi
admiralty_2009_01_10,2009-01-10,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,1.32
admiralty_2009_01_11,2009-01-11,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,1.32
admiralty_2009_01_12,2009-01-12,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,1.32
admiralty_2009_01_13,2009-01-13,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,2.11
admiralty_2009_01_14,2009-01-14,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,2.11
admiralty_2009_01_15,2009-01-15,1,admiralty,0,,,,,,,,0.031,-0
.85,79.1,1.32
```

Data training yang akan digunakan berjumlah 398.327 baris dengan 18 feature yang telah dikumpulkan. Data yang terkumpul masih berbentuk raw dimana gabungan antara data pembacaan opencv yang dilakukan pada data plot dan juga gabungan pada data “gabungan_lainya” pada masing masing kota, serta Data Eksternal seperti ONI, DMI, Mean_rh yang diamati setiap bulannya

(Test_Data)
Dikarenakan pada data testing yang diberikan belum berbentuk keutuhan atau kesatuan maka dilakukan pengcollect dan pembangunan data testing :

i. Concat / Aggregate “Data_Gabungan_Lainnya”
Melakukan penggabungan data “Data_Gabungan_Lainnya_(Tahun).csv” untuk tiap tiap kota pada negara Singapore yang akan digunakan sebagai keutuhan data testing berjumlah 22.750 baris yang akan dilakukan prediksi terhadap “Daily_Rainfall_Total(mm)” menggunakan Regression

```
Concat_Aggregate.ipynb
```

```
def preprocess_dmi_oni(input_file, output_file,
value_col, date_col='Date', missing_val=-9999):
    df = pd.read_csv(input_file)

    if df[date_col].str.contains('/').any():
        df[date_col] = pd.to_datetime(df[date_col],
format='%d/%m/%Y')
    else:
        df[date_col] = pd.to_datetime(df[date_col])

    df[value_col] = df[value_col].replace(missing_val,
np.nan)

    df['year'] = df[date_col].dt.year
    df['month'] = df[date_col].dt.month
    df['month_sin'] = np.sin(2 * np.pi *
df['month']/12)
    df['month_cos'] = np.cos(2 * np.pi *
df['month']/12)

    imputer = KNNImputer(n_neighbors=5)
    features = ['year', 'month_sin', 'month_cos',
value_col]

    temp_df = df[features].copy()
    imputed_values = imputer.fit_transform(temp_df)

    df[value_col] = imputed_values[:, -1]

    df.to_csv(output_file, index=False,
columns=[date_col, value_col])
    print(f"Data berhasil diproses dan disimpan di
{output_file}")

preprocess_dmi_oni('Data Eksternal/Dipole Mode Index
(DMI).csv', 'DMI_imputed.csv', ' DMI HadISST1.1
missing value -9999
https://psl.noaa.gov/data/timeseries/month/')
preprocess_dmi_oni('Data Eksternal/OceanicNinoIndex
(ONI).csv', 'ONI_imputed.csv', '  ONI',
date_col='Date')

all_test_data = []

for city_folder in os.listdir(src_test):
    city_path = os.path.join(src_test, city_folder)
    if not os.path.isdir(city_path): continue

    for filename in os.listdir(city_path):
        if
filename.startswith('Data_Gabungan_Lainnya_') and
filename.endswith('.csv'):
            file_path = os.path.join(city_path,
filename)
            year=filename.split('_')[-1].split('.')[0]
```

```
            try:
                test_df=pd.read_csv(file_path)
                test_df.columns =
[col.replace('min','Min').strip() for col in
test_df.columns]

                test_df['city'] = city_folder.lower()
                test_df['year'] = year

                if 'Date' in test_df.columns:
                    test_df['Date'] =
pd.to_datetime(test_df['Date'])

                all_test_data.append(test_df)

            except Exception as e:
                print(f"{filename}:{str(e)}")
                continue

if all_test_data:
    test_df=pd.concat(all_test_data,
ignore_index=True)
    test_df.to_csv('test_data.csv', index=False)

test_df.info()

test_df = pd.read_csv('test_data.csv')
test_df['year_month'] =
pd.to_datetime(test_df['Date']).dt.to_period('M')

dmi = pd.read_csv('DMI_imputed.csv')
dmi['Date'] = pd.to_datetime(dmi['Date'])
dmi['year_month'] = dmi['Date'].dt.to_period('M')
dmi = dmi.rename(columns={' DMI HadISST1.1  missing
value -9999
https://psl.noaa.gov/data/timeseries/month/':'DMI'})[[
'year_month','DMI']]

oni = pd.read_csv('ONI_imputed.csv')
oni['Date'] = pd.to_datetime(oni['Date'])
oni['year_month'] = oni['Date'].dt.to_period('M')
oni = oni.rename(columns={'
ONI':'ONI'})[['year_month','ONI']]

humi = pd.read_csv('Data
Eksternal/RelativeHumidityMonthlyMean.csv')
humi['year_month'] =
pd.to_datetime(humi['month']).dt.to_period('M')
humi = humi[['year_month', 'mean_rh']]

climate_data = dmi.merge(oni, on='year_month',
how='outer')
climate_data = climate_data.merge(humi,
on='year_month', how='outer')
```

```
test_df = test_df.merge(climate_data, on='year_month',
how='left')

test_df.to_csv('test_data.csv', index=False)
test_df.info()
```

Output :

	Date	month	city	city_encoded	Highest_30min_Rainfall_mm	Highest_60min_Rainfall_mm	Highest_120min_Rainfall_mm	Mean_Temperature_C	Max_Temperature_C	Min_Temperature_C
0	2024-01-01	1	admiralty	0	2.0	2.0	3.0	25.6	28.4	24.8
1	2024-01-02	1	admiralty	0	0.5	0.5	1.5	27.1	32.2	24.5
2	2024-01-03	1	admiralty	0	1.5	1.5	1.5	27.3	30.9	25.3
3	2024-01-04	1	admiralty	0	10.6	13.8	22.2	24.8	26.6	23.3
4	2024-01-05	1	admiralty	0	2.2	3.2	3.6	26.0	29.4	24.2

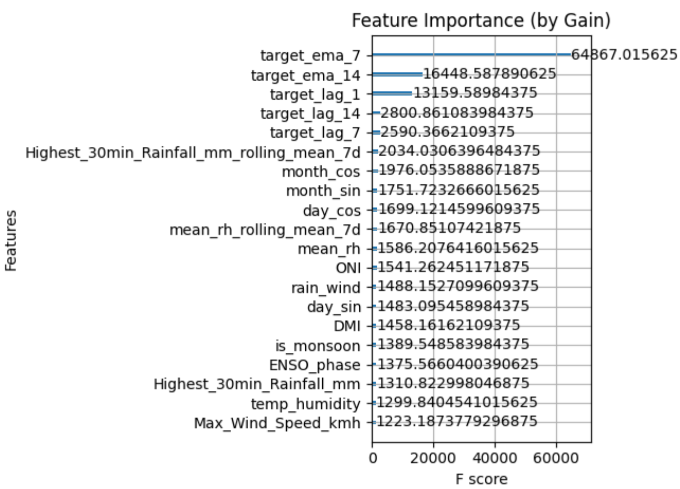
	Date	month	city	city_encoded	Highest_30min_Rainfall_mm	Highest_60min_Rainfall_mm	Highest_120min_Rainfall_mm	Mean_Temperature_C	Max_Temperature_C	Min_Temperature_C
0	2024-01-01	1	admiralty	0	2.0	2.0	3.0	25.6	28.4	24.8
1	2024-01-02	1	admiralty	0	0.5	0.5	1.5	27.1	32.2	24.5
2	2024-01-03	1	admiralty	0	1.5	1.5	1.5	27.3	30.9	25.3
3	2024-01-04	1	admiralty	0	10.6	13.8	22.2	24.8	26.6	23.3
4	2024-01-05	1	admiralty	0	2.2	3.2	3.6	26.0	29.4	24.2

Pembuatan data testing dengan jumlah total data berjumlah 22758 data yang akan dilakukan prediksi dan dilakukan penilaian pada submission kaggle nantinya, dimana dari feature feature yang akan digunakan, dengan menggabungkan data eksternal (DMI, ONI, dan Mean_rh)

b. Exploration Data Analytics

Melakukan eksplorasi terhadap data train yang akan digunakan nantinya, untuk dilakukan eksplorasi untuk mengetahui persebaran data, seperti nilai standar deviasi tinggi atau rendah, important feature, mengecek data null atau kosong, melakukan imputasi terhadap data yang kosong.

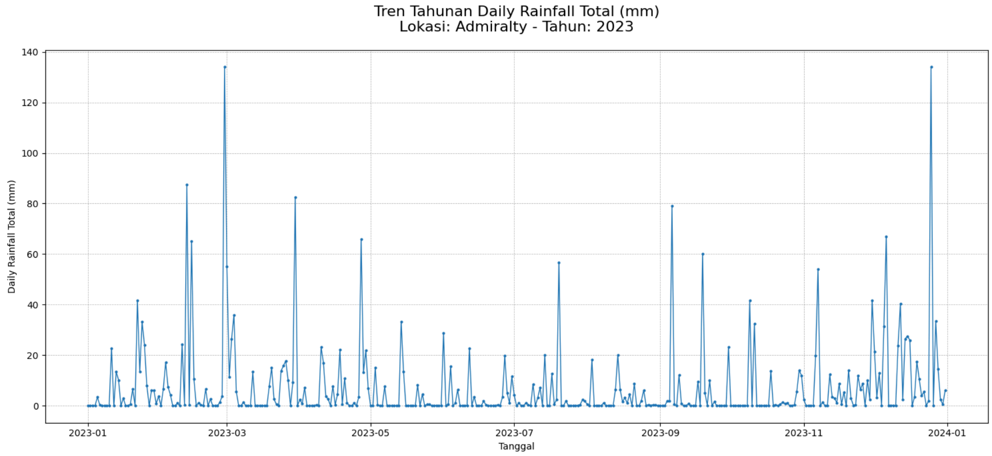
i. Feature Importance

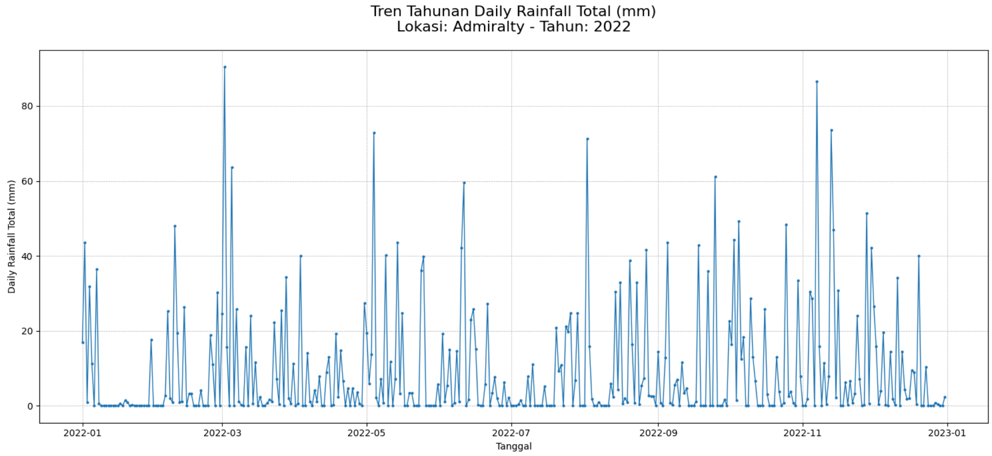


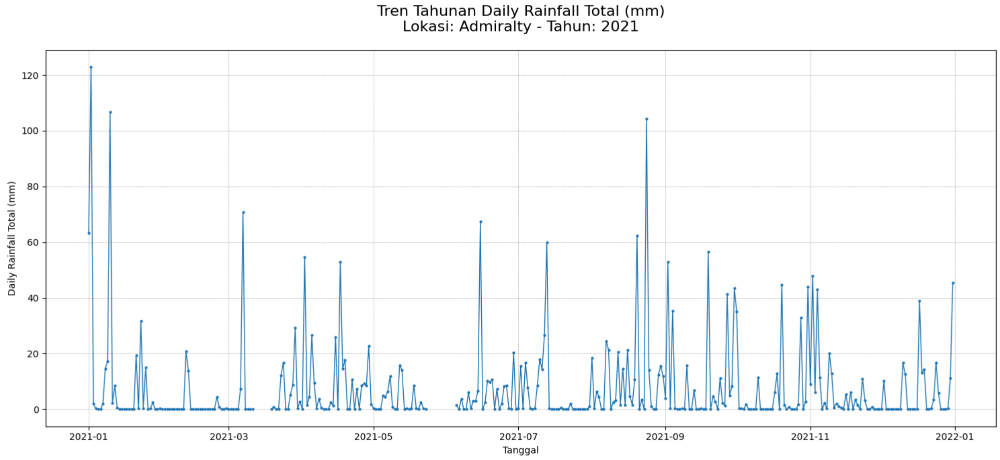
kala

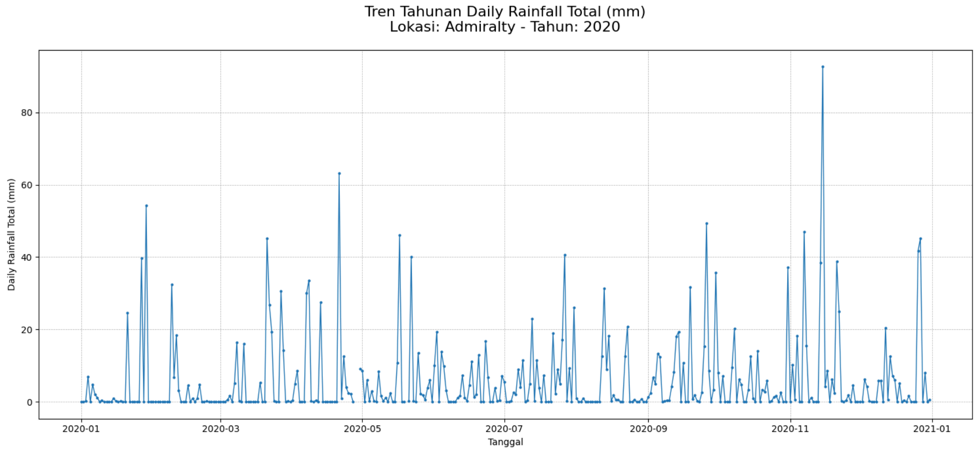
hasil dari feature importance pada feature yang digunakan, dimana untuk feature engineering yang dihasilkan mendaptkan peringat targer_ema_7 dengan importance yang sangat tinggi dimana merupkan

ii. Distribution Data









iii. Visualization Distribution

Data pada prediction rainfall singapore memiliki visualisasi seperti tidak sebaik itu, data yang digunakan lebih banyak yang tidak tersedia atau NULL sehingga data yang akan dilakukan testing ini tidak lengkap sehingga mengakibatkan perbedaan prediksi yang sangat besar. pada kaggle memiliki perbedaan hingga 60 ~ 70 MSE.

c. Preprocessing Data

(Train_data)

i. Mengubah simbol simbol menjadi NaN

train_preprocessing.ipynb

```
columns_with_nulls = [
    'Highest 30 Min Rainfall (mm)',
    'Highest 60 Min Rainfall (mm)',
    'Highest 120 Min Rainfall (mm)',
    'Mean Temperature (°C)',
    'Maximum Temperature (°C)',
    'Minimum Temperature (°C)',
    'Mean Wind Speed (km/h)',
    'Max Wind Speed (km/h)'
]

for col in columns_with_nulls:
    train_df[col] = train_df[col].replace(['', ' ',
'NA', 'N/A', '-', 'NaN', 'null'], np.nan)
    train_df[col] = pd.to_numeric(train_df[col],
errors='coerce')
```

Mengubah nilai pada nilai nilai simbol unik pada kolom, menjadi nilai NaN, atau ‘’, untuk digunakan sebagai training, pada data data yang memiliki banyak nilai unique.

ii. Mengubah Data Type

train_preprocessing

```
col_to_convert = [
    'prediksi',
    'Highest 30 Min Rainfall (mm)',
    'Highest 60 Min Rainfall (mm)',
    'Highest 120 Min Rainfall (mm)',
    'Mean Temperature (°C)',
    'Maximum Temperature (°C)',
    'Minimum Temperature (°C)',
    'Mean Wind Speed (km/h)',
    'Max Wind Speed (km/h)'
]
def convert_to_float(x):
    return float(x)
```

Mengubah data_type data training menjadi floating point untuk kolom prediksi (Daily_Rainfall_Total(mm)), Highest 30 Min Rainfall(mm), Highest 60 Min Rainfall(mm), Highest 120 Min Rainfall(mm), Mean Temperature, Maximum Temperature, Minimum Temperature, Maximum Temperature, Mean Wind, Max Wind. Data Type berpengaruh dalam pemprediksian yang akan digunakan sebagai training model.

iii. Melakukan Penghapusan & Penginputan Data NaN

train_preprocessing

```
for col in col_to_convert:
    train_df[col] = train_df[col].apply(convert_to_float)

null_sum = train_df.isnull().sum()
null_percent = (train_df.isnull().mean()*100).round(2)
null_report = pd.DataFrame({
    'Null Count' : null_sum,
    'Null Percentage' : null_percent
}).sort_values('Null Percentage', ascending=False)
print(null_report[null_report['Null Count']>0])
```

Penghapusan nilai NaN atau data hilang, yang akan dilakukan imputasi menggunakan metode KNN yang telah didefinisikan diawal dengan 5 tahun sebelum dan sesudah.

Function KNN Imputer 5 tahun sebelum dan sesudah

KNNImputer.ipynb

```
from sklearn.impute import KNNImputer

class RainfallImputer:
    def __init__(self, strategy=None, n_neighbors=3):
        self.strategy = strategy or {
            'Highest_30min_Rainfall_mm': 'median',
            'Highest_60min_Rainfall_mm': 'median',
            'Highest_120min_Rainfall_mm': 'median',
```

```
            'Mean_Temperature_C': 'mean',
            'Max_Temperature_C': 'mean',
            'Min_Temperature_C': 'mean',
            'Mean_Wind_Speed_kmh': 'median',
            'Max_Wind_Speed_kmh': 'median'
        }
        self.n_neighbors = n_neighbors
        self.impute_values = {}
        self.global_values = {}
        self.fitted = False

    def fit(self, df):
        df = df.copy()
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year
        self.min_year = df['year'].min()
        self.max_year = df['year'].max()

        for col in self.strategy.keys():
            print(f"Preparing historical data for KNN
imputation: {col}")
            col_impute_values = {}

            for y in range(self.min_year, self.max_year + 1):
                for w in range(1, 6):
                    hist_year = y - w
                    if hist_year < self.min_year:
                        continue

                    df_hist = df[df['year'] ==
hist_year][['city', 'month', col]].dropna()
                    if df_hist.empty:
                        continue

                    key = f"{hist_year}-{y}"
                    if key not in col_impute_values:
                        col_impute_values[key] =
df_hist.copy()
                    else:
                        col_impute_values[key] =
pd.concat([col_impute_values[key], df_hist],
ignore_index=True)

            self.impute_values[col] = col_impute_values

        # Fallback values
        for col, method in self.strategy.items():
            city_month_values = df.groupby(['city',
'month'])[col].agg(method).reset_index()
            self.impute_values[col]['city_month'] =
city_month_values
            self.global_values[col] = df[col].median() if
method == 'median' else df[col].mean()

        self.fitted = True
        return self

    def transform(self, df):
        if not self.fitted:
            raise ValueError("Imputer belum di-fit. Panggil
```

```
fit() terlebih dahulu.")

        df = df.copy()
        df['month'] = df['Date'].dt.month
        df['year'] = df['Date'].dt.year

        for col in self.strategy.keys():
            print(f"Applying KNN historical imputation for:
{col}")
            filled_col = df[col].copy()

            for y in df['year'].unique():
                for w in range(1, 6):
                    hist_year = y - w
                    key = f"{hist_year}-{y}"

                    if key not in self.impute_values[col]:
                        continue

                    hist_df = self.impute_values[col][key]
                    if hist_df.empty:
                        continue

                    mask_target = (df['year'] == y) &
(df[col].isnull())
                    target_df = df.loc[mask_target, ['city',
'month']]

                    if target_df.empty:
                        continue

                    # Gabungkan historical + target
                    combined = pd.concat([
                        hist_df,
                        pd.DataFrame({'city':
target_df['city'], 'month': target_df['month'], col: np.nan})
                    ], ignore_index=True)

                    combined_encoded =
pd.get_dummies(combined[['city', 'month']])
                    combined_encoded[col] =
combined[col].values

                    imputer =
KNNImputer(n_neighbors=self.n_neighbors)
                    imputed_array =
imputer.fit_transform(combined_encoded)

                    imputed_values =
imputed_array[-len(target_df):, -1]  # Ambil hasil prediksi
                    filled_col.loc[mask_target] =
imputed_values

            df[col] = filled_col

        # Fallback: city + month
        for col in self.strategy.keys():
            print(f"Applying fallback imputation for: {col}")
            city_month_values =
self.impute_values[col]['city_month']
```

```
            df = df.merge(city_month_values, on=['city',
'month'], how='left', suffixes=('', '_impute'))
            df[col] = df[col].fillna(df[f'{col}_impute'])
            df.drop(columns=[f'{col}_impute'], inplace=True)

            # Fallback: global
            df[col] = df[col].fillna(self.global_values[col])

        print("Missing values after imputation:")
        print(df.isnull().sum()[df.isnull().sum() > 0])

        return df

    def save_imputer(self, filepath):
        if not self.fitted:
            raise ValueError("Imputer belum di-fit. Panggil
fit() terlebih dahulu.")

        data_to_save = {
            'strategy': self.strategy,
            'n_neighbors': self.n_neighbors,
            'global_values': self.global_values,
            'impute_values': {
                col: {
                    key: df.to_dict('records') if
isinstance(df, pd.DataFrame) else df
                    for key, df in hist.items()
                }
                for col, hist in self.impute_values.items()
            }
        }

        with open(filepath, 'w') as f:
            json.dump(data_to_save, f)

    @classmethod
    def load_imputer(cls, filepath):
        with open(filepath, 'r') as f:
            data = json.load(f)

        imputer = cls(strategy=data['strategy'],
n_neighbors=data['n_neighbors'])
        imputer.global_values = data['global_values']
        imputer.impute_values = {
            col: {
                key: pd.DataFrame(records) if
isinstance(records, list) else records
                for key, records in hist.items()
            }
            for col, hist in data['impute_values'].items()
        }
        imputer.fitted = True
        return imputer
```

Membuat function untuk imputer menggunakan library KNN imputer, yang dibentuk menggunakan 5 tahun sebelum dan sesudah untuk dilakukan KNN imputer untuk imputer yang lebih stabil dan lebih

iv. Renaming & Order Data

train_preprocessing.ipynb

```
def detect_outliers(series, method='iqr', threshold = 1.5):
    if method == 'iqr':
        Q1 = series.quantile(0.25)
        Q3 = series.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold*IQR
        upper_bound = Q3 - threshold*IQR
        return (series<lower_bound) | (series>upper_bound)
    elif method == 'zscore':
        z_score = np.abs(stats.zscore(series))
        return z_score>threshold

def handle_outliers(train_df, col, method='cap', **kwargs):
    if method == 'cap':
        Q1 = train_df[col].quantile(0.25)
        Q3 = train_df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - kwargs.get('threshold', 1.5)*IQR
        upper_bound = Q3 + kwargs.get('threshold', 1.5)*IQR
        train_df[col] = train_df[col].clip(lower_bound,
upper_bound)
    elif method=='remove':
        outliers=detect_outliers(train_df[col], **kwargs)
        train_df = train_df[~outliers]
    elif method=='transform':
        train_df[col]=np.log1p(train_df[col])
    return train_df

outlier_strategy = {
    'Mean_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Max_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Min_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Max_Wind_Speed_kmh': {'method': 'cap', 'threshold': 3,
'detect_method': 'iqr'}
}

for col, params in outlier_strategy.items():
    is_outlier = detect_outliers(
        train_df[col],
        method=params['detect_method'],
        threshold=params['threshold']
    )
    print(f"Jumlah outlier ditemukan: {is_outlier.sum()}")

    plt.figure(figsize=(10, 4))
    plt.boxplot(train_df[col].dropna())
    plt.title(f'Before Outlier Handling - {col}')
    plt.show()

    train_df = handle_outliers(train_df, col, **params)
    plt.figure(figsize=(10, 4))
    plt.boxplot(train_df[col].dropna())
    plt.title(f'After Outlier Handling - {col}')
    plt.show()
```

v. Pendeteksian dan Penghapusan Outlier

train_preprocessing

```
def detect_outliers(series, method='iqr', threshold = 1.5):
    if method == 'iqr':
        Q1 = series.quantile(0.25)
        Q3 = series.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - threshold*IQR
        upper_bound = Q3 - threshold*IQR
        return (series<lower_bound) | (series>upper_bound)
    elif method == 'zscore':
        z_score = np.abs(stats.zscore(series))
        return z_score>threshold

def handle_outliers(train_df, col, method='cap', **kwargs):
    if method == 'cap':
        Q1 = train_df[col].quantile(0.25)
        Q3 = train_df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - kwargs.get('threshold', 1.5)*IQR
        upper_bound = Q3 + kwargs.get('threshold', 1.5)*IQR
        train_df[col] = train_df[col].clip(lower_bound,
upper_bound)
    elif method=='remove':
        outliers=detect_outliers(train_df[col], **kwargs)
        train_df = train_df[~outliers]
    elif method=='transform':
        train_df[col]=np.log1p(train_df[col])
    return train_df

outlier_strategy = {
    'Mean_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Max_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Min_Temperature_C': {'method': 'cap', 'threshold': 2.5,
'detect_method': 'zscore'},
    'Max_Wind_Speed_kmh': {'method': 'cap', 'threshold': 3,
'detect_method': 'iqr'}
}

for col, params in outlier_strategy.items():
    is_outlier = detect_outliers(
        train_df[col],
        method=params['detect_method'],
        threshold=params['threshold']
    )
    print(f"Jumlah outlier ditemukan: {is_outlier.sum()}")

    plt.figure(figsize=(10, 4))
    plt.boxplot(train_df[col].dropna())
    plt.title(f'Before Outlier Handling - {col}')
    plt.show()

    train_df = handle_outliers(train_df, col, **params)
    plt.figure(figsize=(10, 4))
    plt.boxplot(train_df[col].dropna())
    plt.title(f'After Outlier Handling - {col}')
    plt.show()
```

Melakukan handling outlier menggunakan metode IQR untuk dilakukan pendeteksian dan melakukan penghapusan outlier pada data trainingnya. dengan menggunakan penghapusan quartile.

(Test_data)

i. Mengubah Data Type

test_preprocessing

```
columns_with_nulls = [
    'Highest 30 Min Rainfall (mm)',
    'Highest 60 Min Rainfall (mm)',
    'Highest 120 Min Rainfall (mm)',
    'Mean Temperature (°C)',
    'Maximum Temperature (°C)',
    'Minimum Temperature (°C)',
    'Mean Wind Speed (km/h)',
    'Max Wind Speed (km/h)',
    'DMI', 'ONI', 'mean_rh'
]

for col in columns_with_nulls:
    test_df[col] = test_df[col].replace(['', ' ', 'NA', 'N/A',
'-', 'NaN', 'null'], np.nan)
    test_df[col] = pd.to_numeric(test_df[col],
errors='coerce')
```

ii. Convert Data Types

test_preprocessing.ipynb

```
col_to_convert = [
    'Highest 30 Min Rainfall (mm)',
    'Highest 60 Min Rainfall (mm)',
    'Highest 120 Min Rainfall (mm)',
    'Mean Temperature (°C)',
    'Maximum Temperature (°C)',
    'Minimum Temperature (°C)',
    'Mean Wind Speed (km/h)',
    'Max Wind Speed (km/h)'
]
def convert_to_float(x):
    return float(x)
```

iii. Mengubah simbol simbol menjadi NaN

test_preprocessing.ipynb

```
columns_with_nulls = [
    'Highest 30 Min Rainfall (mm)',
    'Highest 60 Min Rainfall (mm)',
    'Highest 120 Min Rainfall (mm)',
    'Mean Temperature (°C)',
    'Maximum Temperature (°C)',
    'Minimum Temperature (°C)',
    'Mean Wind Speed (km/h)',
```

```
    'Max Wind Speed (km/h)',
    'DMI', 'ONI', 'mean_rh'
]

for col in columns_with_nulls:
    test_df[col] = test_df[col].replace(['', ' ', 'NA', 'N/A',
'-', 'NaN', 'null'], np.nan)
    test_df[col] = pd.to_numeric(test_df[col],
errors='coerce')
```

iv. Renaming & Order Features

test_preprocessing.ipynb

```
test_df = test_df.rename(columns={'Date':'Date'})
test_df = test_df.rename(columns={'Highest 30 Min Rainfall
(mm)':'Highest_30min_Rainfall_mm'})
test_df = test_df.rename(columns={'Highest 60 Min Rainfall
(mm)':'Highest_60min_Rainfall_mm'})
test_df = test_df.rename(columns={'Highest 120 Min Rainfall
(mm)':'Highest_120min_Rainfall_mm'})
test_df = test_df.rename(columns={'Mean Temperature
(°C)':'Mean_Temperature_C'})
test_df = test_df.rename(columns={'Maximum Temperature
(°C)':'Max_Temperature_C'})
test_df = test_df.rename(columns={'Minimum Temperature
(°C)':'Min_Temperature_C'})
test_df = test_df.rename(columns={'Mean Wind Speed
(km/h)':'Mean_Wind_Speed_kmh'})
test_df = test_df.rename(columns={'Max Wind Speed
(km/h)':'Max_Wind_Speed_kmh'})

desired_order = [
    'Date', 'month', 'city', 'city_encoded',
    'Highest_30min_Rainfall_mm', 'Highest_60min_Rainfall_mm',
'Highest_120min_Rainfall_mm',
    'Mean_Temperature_C', 'Max_Temperature_C',
'Min_Temperature_C',
    'Mean_Wind_Speed_kmh', 'Max_Wind_Speed_kmh',
    'DMI', 'ONI', 'mean_rh'
]
```

Melakukan renaming dan order untuk masing masing feature yang akan digunakan untuk prediksi agar sama dengan training.

(General)
Feature Engineering :

i. Temporal Features

feature_engineering.ipynb

```
df['month'] = df['Date'].dt.month
    df['day_of_year'] = df['Date'].dt.dayofyear
    df['week_of_year'] =
df['Date'].dt.isocalendar().week
    df['is_monsoon'] =
df['month'].isin([6,7,8,9]).astype(int)
```

Menangkap pola musiman dan temporal dlaam data, menggunakan domain knowledge antara lain waktu yaitu (bulan, minggu, dan musim hujan). Pendefinisian is in [6,7,8,9] digunakan dalam rentang musim kemarau negara negara asia (Singapore) yang digunakan sebagai temporal feature untuk menambahkan feature musim kemarau yang terjadi dalam rentang bulan bulan tersebut dan untuk bulan penghujan yaitu dalam rentang (11,12,1,2,3) sedangkan bulan lainnya atau sisa sisa bulan lain adalah sebagai masuk pada kategori transisi.

ii. Cyclical Encoding

feature_engineering.ipynb

```
df['month_sin'] = np.sin(2 * np.pi * df['month']/12)
df['month_cos'] = np.cos(2 * np.pi * df['month']/12)
```

Mengubah fitur siklik (bulan) menjadi format yang lebih bermakna untuk model, untuk menangkap sifat periodik Desember dan Januari yang berdekatan dll

iii. Weather Interactions

feature_engineering.ipynb

```
df['temp_range'] = df['Max_Temperature_C'] -
df['Min_Temperature_C']
    df['temp_humidity'] = df['Mean_Temperature_C'] *
df['mean_rh']
    df['rain_wind'] = df['Highest_30min_Rainfall_mm'] *
df['Max_Wind_Speed_kmh']
```

Menangkap interaksi kompleks antara variable cuaca. menggunakan domain knowledge meteorologi tentang bagimana variable-variable ini berinteraksi

iv. Climate Indices

feature_engineering.ipynb

```
df['ENSO_phase'] = np.where(df['ONI'] > 0.5, 1,
np.where(df['ONI'] < -0.5, -1, 0))
```

Mengkategorikan indeks iklim menjadi fase-fase yang bermakna, menggunakan pengetahuan ilmiah tentang fenomena ENSO (El Nino-Southern Oscillation)

v. Rolling Statistic with multiple windows

feature_engineering.ipynb

```
for col in ['Highest_30min_Rainfall_mm',
'Mean_Temperature_C', 'Max_Wind_Speed_kmh']:
        for window in [3, 7, 14]:
            df[f'{col}_rolling_mean_{window}d'] =
df.groupby('city')[col].transform(
                lambda x: x.rolling(window,
min_periods=1).mean())
            df[f'{col}_rolling_max_{window}d'] =
df.groupby('city')[col].transform(
                lambda x: x.rolling(window,
min_periods=1).max())
```

Menangkap tren jangka pendek dan menengah dalam data. mengasumsikan bahwa kondisi cuaca terakhir (3-30 haru) mempengaruhi kondisi saat ini

vi. Lag Features

feature_engineering.ipynb

```
if is_training:
        for lag in [1, 2, 3, 7, 14]:
            df[f'target_lag_{lag}'] =
df.groupby('city')['target'].shift(lag)
    else:
        for lag in [1, 2, 3, 7, 14]:
            df[f'target_lag_{lag}'] = 0
```

Memberikan bobot lebih pada data terbaru dalam menghitung rata rata, mengasumsikan data terbaru lebih relevan daripada data lama

d. Build Machine Learning

i. XGBoost

xgboost_model.ipynb

```
model = XGBRegressor(
        **best_params,
        early_stopping_rounds=100,
        eval_metric='rmse',
        random_state=42
    )

    model.fit(
        X_train, y_train,
        eval_set=[(X_val, y_val)],
        verbose=100
    )
```

e. Training

i. Hyperparameter Tuning

Training_model.ipynb

```
def tune_hyperparameters(X_train, y_train, X_val, y_val):
    def objective(trial):
        params = {
            'objective': 'reg:squarederror',
            'tree_method': 'hist',
            'n_estimators': trial.suggest_int('n_estimators',
500, 2000),
            'learning_rate':
trial.suggest_float('learning_rate', 0.001, 0.1, log=True),
            'max_depth': trial.suggest_int('max_depth', 3,
12),
            'subsample': trial.suggest_float('subsample', 0.6,
```

```
1.0),
            'colsample_bytree':
trial.suggest_float('colsample_bytree', 0.6, 1.0),
            'gamma': trial.suggest_float('gamma', 0, 0.5),
            'reg_alpha': trial.suggest_float('reg_alpha', 0,
10),
            'reg_lambda': trial.suggest_float('reg_lambda', 0,
10),
            'min_child_weight':
trial.suggest_int('min_child_weight', 1, 10),
            'early_stopping_rounds': 100,  # Moved here
            'eval_metric': 'rmse',  # Moved here
        }

        model = XGBRegressor(**params)
        model.fit(
            X_train, y_train,
            eval_set=[(X_val, y_val)],
            verbose=False
        )

        val_pred = model.predict(X_val)
        return np.sqrt(mean_squared_error(y_val, val_pred))

    study = optuna.create_study(direction='minimize')
    study.optimize(objective, n_trials=50, timeout=3600)

best_params = tune_hyperparameters(X_train, y_train, X_val,
y_val)
```

f. Evaluating

i. MSE

Evaluating.ipynb

```
val_pred = model.predict(X_val)
    mae = mean_absolute_error(y_val, val_pred)
    rmse = np.sqrt(mean_squared_error(y_val, val_pred))
    print(f"\nValidation MAE: {mae:.4f}, RMSE: {rmse:.4f}"
```

Result & Validation

```
[I 2025-07-17 03:00:44,642] A new study created in memory with name: no-name-69d0c4c0-c78a-472a-ac37-840c65cf1835
Tuning hyperparameters...
[I 2025-07-17 03:01:52,765] Trial 0 finished with value: 10.808419439721165 and parameters: {'n_estimators': 1182,
[I 2025-07-17 03:02:36,760] Trial 1 finished with value: 10.237116606839772 and parameters: {'n_estimators': 1365,
[I 2025-07-17 03:03:20,207] Trial 2 finished with value: 11.03835100324436 and parameters: {'n_estimators': 1437,
[I 2025-07-17 03:04:13,008] Trial 3 finished with value: 9.957321136212377 and parameters: {'n_estimators': 1759,
[I 2025-07-17 03:04:32,618] Trial 4 finished with value: 9.952383427913666 and parameters: {'n_estimators': 1110,
[I 2025-07-17 03:05:27,782] Trial 5 finished with value: 9.897705345517302 and parameters: {'n_estimators': 1169,
[I 2025-07-17 03:05:40,719] Trial 6 finished with value: 9.907339291030896 and parameters: {'n_estimators': 1670,
[I 2025-07-17 03:07:01,731] Trial 7 finished with value: 10.121056774032356 and parameters: {'n_estimators': 1849,
[I 2025-07-17 03:07:39,330] Trial 8 finished with value: 9.883984141672297 and parameters: {'n_estimators': 1616,
[I 2025-07-17 03:09:09,429] Trial 9 finished with value: 9.971833237405455 and parameters: {'n_estimators': 995,
[I 2025-07-17 03:09:21,457] Trial 10 finished with value: 11.077391719175745 and parameters: {'n_estimators': 562,
[I 2025-07-17 03:09:59,918] Trial 11 finished with value: 9.935855139424827 and parameters: {'n_estimators': 900,
[I 2025-07-17 03:10:52,994] Trial 12 finished with value: 9.857898858097325 and parameters: {'n_estimators': 1564,
[I 2025-07-17 03:12:34,636] Trial 13 finished with value: 10.021163761952362 and parameters: {'n_estimators': 1568,
[I 2025-07-17 03:12:44,766] Trial 14 finished with value: 9.95962604009283 and parameters: {'n_estimators': 1973,
[I 2025-07-17 03:14:42,203] Trial 15 finished with value: 9.997068929386652 and parameters: {'n_estimators': 1550,
[I 2025-07-17 03:14:59,774] Trial 16 finished with value: 9.938524554266634 and parameters: {'n_estimators': 1373,
[I 2025-07-17 03:15:38,323] Trial 17 finished with value: 9.870358442700677 and parameters: {'n_estimators': 1975,
[I 2025-07-17 03:17:00,504] Trial 18 finished with value: 9.93290824424781 and parameters: {'n_estimators': 1942,
[I 2025-07-17 03:19:05,687] Trial 19 finished with value: 9.833698945248067 and parameters: {'n_estimators': 1770,
[I 2025-07-17 03:21:32,684] Trial 20 finished with value: 10.405371052705126 and parameters: {'n_estimators': 1749,
[I 2025-07-17 03:23:05,833] Trial 21 finished with value: 9.83564669909753 and parameters: {'n_estimators': 1851,
[I 2025-07-17 03:24:41,904] Trial 22 finished with value: 9.82821421504818 and parameters: {'n_estimators': 1788,
[I 2025-07-17 03:26:20,632] Trial 23 finished with value: 9.842471130373607 and parameters: {'n_estimators': 1824,
[I 2025-07-17 03:27:29,657] Trial 24 finished with value: 9.951758669561794 and parameters: {'n_estimators': 1862,
...
[I 2025-07-17 03:56:22,784] Trial 43 finished with value: 9.817371407257886 and parameters: {'n_estimators': 1160,
[I 2025-07-17 03:58:10,365] Trial 44 finished with value: 9.846987838671481 and parameters: {'n_estimators': 1130,
[I 2025-07-17 04:00:28,917] Trial 45 finished with value: 9.933465536644107 and parameters: {'n_estimators': 1020,
[I 2025-07-17 04:01:18,014] Trial 46 finished with value: 9.829382767144034 and parameters: {'n_estimators': 848,
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Best parameters: {'n_estimators': 1786, 'learning_rate': 0.0037583561071173237, 'max_depth': 10, 'subsample': 0.832
Training final model...
```

```
Training final model...
[0]     validation_0-rmse:20.93059     validation_0-mae:14.64031
[100]   validation_0-rmse:16.86054     validation_0-mae:11.79237
[200]   validation_0-rmse:14.18130     validation_0-mae:9.91595
[300]   validation_0-rmse:12.49381     validation_0-mae:8.69615
[400]   validation_0-rmse:11.47852     validation_0-mae:7.91109
[500]   validation_0-rmse:10.87200     validation_0-mae:7.40486
[600]   validation_0-rmse:10.50118     validation_0-mae:7.06898
[700]   validation_0-rmse:10.26406     validation_0-mae:6.83947
[800]   validation_0-rmse:10.10836     validation_0-mae:6.68062
[900]   validation_0-rmse:10.00411     validation_0-mae:6.56732
[1000]  validation_0-rmse:9.93549      validation_0-mae:6.49017
[1100]  validation_0-rmse:9.89266      validation_0-mae:6.43675
[1200]  validation_0-rmse:9.86390      validation_0-mae:6.40077
[1300]  validation_0-rmse:9.84554      validation_0-mae:6.37553
[1400]  validation_0-rmse:9.83134      validation_0-mae:6.35614
[1500]  validation_0-rmse:9.82057      validation_0-mae:6.34174
[1600]  validation_0-rmse:9.81431      validation_0-mae:6.33175
[1700]  validation_0-rmse:9.81377      validation_0-mae:6.32583
[1785]  validation_0-rmse:9.81136      validation_0-mae:6.32023

Validation MAE: 6.3202, RMSE: 9.8114
```

val_rmse	9.81236
val_mae	6.32

Dari hasil yang didapatkan atas data testing yang telah kita build mendapatkan nilai yang cukup baik walau belum sempurna, dengan hasil **validation rmse** dengan nilai pada **angka**

9.81326, sedangkan untuk **validation mae** pada angka : **6.32** yang dimana merupakan hasil dari training machine learning, dikarenakan data yang memiliki nilai NULL sangat banyak mengakibatkan error ini bernilai 6 atau pun ada faktor lain yaitu untuk nilai dari plt yang di historisiskan atau digambarkan tidak semuanya ada atau tersedia sehingga NULL yang sangat banyak atau penggunaan metode fill data ini sangatlah berpengaruh, dimana pada penggunaan metode fill ini menggunakan KNN imputer.

Conclusion

Pada pembangunan machine learning kali ini untuk data historis serta opencv untuk plotting dan data nya terbilang masih kurang dikarenakan beberapa faktor berikut :

1. Data kosong atau NULL sangat banyak atau sekitar pada angka 60% keatas. itu pada 6 feature dari total feature berjumlah 12 feature atau bisa dibilang 50% nya feature adalah data yang tidak lengkap
2. Build Testing yang dimana kurang tepat dengan apa yang diharapkan pada kaggle yang dimana faktor imputer pada testing yang sangat berbeda menimbulkan nilai MAE pada data testing yang saya buat dengan kompetisi berbeda sehinggan mendapatkan nilai MAE yang besar pada Kaggle.

Dari kedua kesimpulan tersebut bisa disimpulkan kunci dalam pembuatan machine learning pada kasus ini, terdapat pada pembuatan atau collecting atau bisa disebut pembuatan data training dan testing yang tepat menjadikan kunci pada kasus ini untuk didapakan prediksi yang sangatlah tepat dan akurat.

Source

Competition Kaggle :
https://www.kaggle.com/competitions/sebelas-maret-statistics-data-science-2025/overview