

Analisis dan Prediksi Jumlah Curah Hujan Harian menggunakan Metode eXtreme Gradient Boosting (XGBoost) dan Light Gradient Boosted Machine (LightGBM)

Samatha Marhaendra Putra, Rizky Alif Ramadhan, Daffa Bil Nadzary

1. Pendahuluan

1.1 Latar Belakang

Sektor pertanian menjadi salah satu sektor yang vital dalam hal ketahanan pangan. Dalam usaha untuk meningkatkan dan memertahankan produktivitas di sektor tersebut, diperlukan suplai dengan kualitas yang sesuai. Salah satu hal yang menjadi suplai utama di sektor ini yakni air. Akan tetapi, ketidakpastian curah hujan yang ada memiliki dampak negatif terhadap kualitas air yang tersedia dan produktivitas di sektor agrikultur. Sektor ini bergantung pada curah hujan dan ketersediaan air setiap harinya [1]. Maka dari itu, diperlukan suatu sistem yang mampu secara akurat memprediksi jumlah curah hujan harian untuk dapat mengelola air hujan dengan baik untuk sektor tersebut dan juga untuk ketersediaan air.

Untuk memprediksi curah hujan harian, beberapa riset telah dilakukan dengan menggunakan *data mining* dan *machine learning* pada data cuaca harian dari berbagai wilayah. Beberapa faktor lingkungan dapat mempengaruhi jumlah curah hujan yakni seperti temperatur, kelembapan, radiasi matahari, tekanan udara, tingkat evaporasi, dan lain-lain [2].

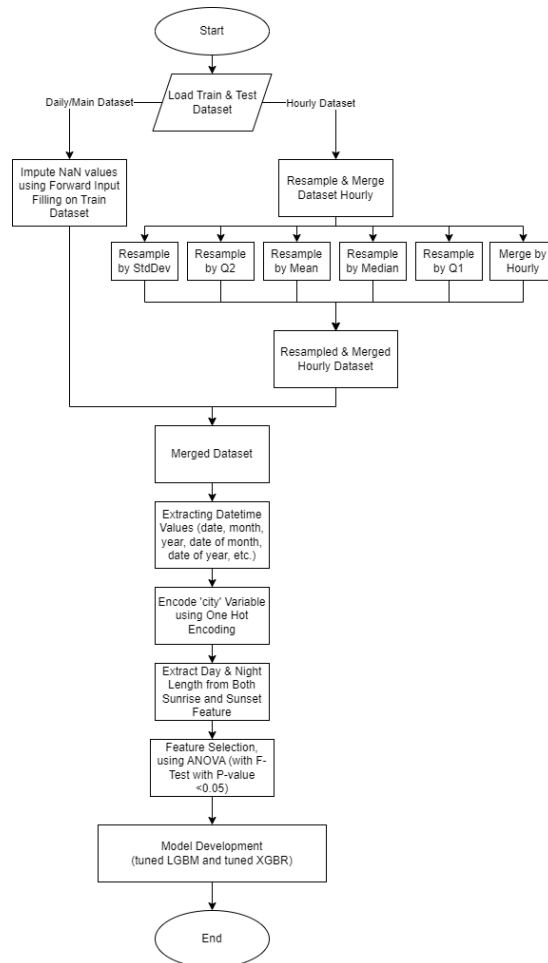
1.2 Tujuan

Tujuan dilakukannya penelitian ini adalah:

- a) Melakukan analisis curah hujan terhadap waktu di berbagai kota
- b) Menganalisis fitur-fitur yang memiliki dampak baik langsung maupun tidak langsung terhadap jumlah curah hujan harian
- c) Membuat suatu model yang mampu menerjemahkan fitur-fitur yang ada sehingga mampu digunakan untuk dapat memprediksi jumlah curah hujan harian dengan akurat.

2. Metodologi

2.1 Alur Pengerjaan



Gambar 2.1 *Flowchart* Pengerjaan Submisi.

Alur pengerjaan yang dilakukan pada pengerjaan ini diawali dari pengumpulan dan pemrosesan data seperti pembersihan dan penggabungan data harian dan data setiap jam. Selanjutnya yakni dilanjutkan dengan melakukan beberapa visualisasi pada tahap *exploratory data analysis* yang kemudian visualisasi tersebut dipakai untuk menentukan langkah pemrosesan data selanjutnya. Berikutnya yakni dilakukan ekstraksi dan pembuatan fitur baru, lalu diikuti dengan melakukan *feature selection* menggunakan beberapa pengujian. Berikutnya yaitu dilakukan pembagian data latih untuk keperluan *modelling* dan *cross-validation* untuk melihat performa dari model yang dibuat. Tahap terakhir yakni melakukan proses *modelling* itu sendiri, juga penyusunan berkas submisi yang kemudian dikumpulkan pada *platform* Kaggle InClass Competition.

2.2 Pengumpulan Data

Dataset yang dipakai dalam pengerjaan ini memakai data yang telah disediakan oleh penyelenggara kompetisi. Data yang diberikan yaitu data harian dan data setiap jam terkait riwayat curah hujan di beberapa kota, beserta dengan fitur-fitur lain yang selanjutnya dilakukan analisis keterkaitan dengan fitur yang diprediksi, yakni jumlah curah hujan.

2.3 Preprocessing

Data *training* serta *testing* yang sudah dikumpulkan terbagi atas dua jenis, yakni data harian serta data per jam. Pada data per jam, dilakukan metode *resampling* data dari per jam menjadi per hari, dengan melakukan agregasi data untuk mencari baik rerata (*mean*), median, kuartil 1, kuartil 2, serta standar deviasi dari data. Selain melakukan *resample*, dilakukan juga *merge* data per jam untuk dijadikan sebagai fitur tambahan pada *dataset*.

Selanjutnya yaitu dilakukan visualisasi pada *dataset* harian yang termasuk ke dalam tahap *exploratory data analysis*. Visualisasi ini bertujuan untuk menentukan langkah pemrosesan data selanjutnya. Pada *dataset* harian, terdapat beberapa baris yang mengandung nilai NaN yang di-*impute* menggunakan teknik *forward input filling*. Namun, pada fitur target, yakni 'rain_sum (mm)', apabila terdapat nilainya yang NaN maka nilai tersebut tidak di-*impute*, melainkan dilakukan *drop* baris data tersebut.

Dari *dataset* harian yang sudah dibersihkan sementara dan *dataset* per jam yang sudah dilakukan agregasi, kedua *dataset* tersebut kemudian di-*merge* menjadi satu untuk memperbanyak serta memperkaya fitur yang dapat digunakan dalam proses pelatihan model.

2.4 Exploratory Data Analysis

EDA atau *Exploratory Data Analysis* adalah kegiatan untuk menganalisis sekumpulan data untuk melihat karakteristik utamanya. EDA bertujuan untuk melihat intisari dan pola-pola dari data yang bermanfaat dalam proses modeling atau eksperimen pada data [3]. Pada penelitian ini, peneliti menggunakan visualisasi data dalam proses EDA, untuk melihat pola-pola atau kesimpulan tertentu, utamanya yang berhubungan dengan curah hujan.

2.5 Feature Engineering

Tahap *feature engineering* dilakukan untuk memilih, mengevaluasi serta mengubah data agar lebih sesuai sebelum dimasukkan ke dalam proses pemodelan dengan memilih hanya beberapa fitur dengan tingkat *importance* yang sesuai. Pada tahap ini, setelah *dataset* harian dan per jam digabung, terdapat beberapa fitur yang masih dapat dipecah menjadi beberapa fitur lain, seperti fitur tanggal, kota, waktu matahari terbit dan terbenam (*sunrise* dan *sunset*), dan lain-lain. Adapun tahap lain yang dilakukan yaitu melakukan normalisasi data pada fitur dengan sebaran yang tidak seimbang (*skewed*) serta melakukan uji multikolinearitas dari fitur dengan menggunakan uji ANOVA F-Test untuk memilih fitur-fitur yang penting untuk dimasukkan ke dalam model. Berikut merupakan formula yang dipakai dalam pengujian ini.

$$r_{reg} = \Sigma[(X[:, i] - \text{mean}(X[:, i])) * (y - \text{mean}(y))] / (\text{std}(X[:, i]) * \text{std}(y))$$

Kemudian, hasil dari perhitungan menggunakan formula di atas diubah ke dalam *F-Score* dan dilanjut ke *p-value*.

2.6 Modelling

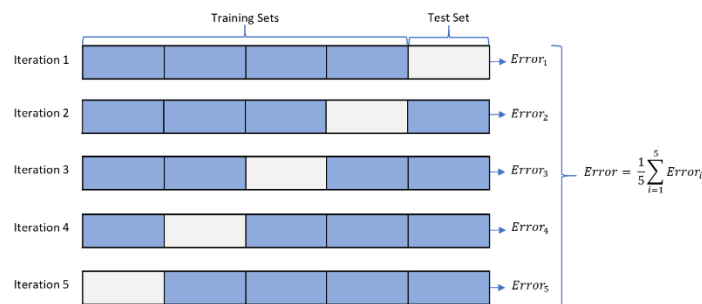
Proses *modelling* yang dilakukan pada penelitian ini adalah menggunakan dua model yang berbeda, yaitu eXtreme Gradient Boost (XGBoost), dan Light Gradient-Boosting Machine (LightGBM). XGBoost merupakan model *machine learning* berbasis *gradient-boosted decision tree* (GBDT) yang bersifat *scalable*. Sedangkan LightGBM merupakan juga model yang berbasis GBDT namun dengan implementasi teknik *gradient-based one-side sampling*. Kedua model ini cocok digunakan untuk permasalahan regresi, klasifikasi, dan permasalahan *supervised learning* lainnya.

Kedua model di-*fit* menggunakan data yang sudah diubah dari proses *feature engineering* sebelumnya, dan di-*tuning* lebih lanjut untuk memperoleh hasil yang maksimal. Pada kedua model, proses *tuning* dilakukan dengan membuat sebuah *search space* yang berisikan parameter-parameter yang akan di-*tuning*. Adapun proses *hyperparameter tuning* yang dilakukan adalah dengan menggunakan pendekatan Bayesian Optimization

dengan mengatur *delta stopper* untuk menghasilkan hasil kombinasi parameter yang terbaik.

2.7 Validation

Validasi yang dilakukan pada penelitian kali ini adalah menggunakan *5-fold cross-validation* dengan ukuran evaluasinya adalah *mean squared error* (MSE). *Cross-validation* atau validasi silang adalah teknik yang digunakan untuk mengevaluasi model *machine learning* dengan data yang dibagi menjadi 2 bagian, di mana bagian pertama digunakan untuk pelatihan model dan bagian kedua digunakan untuk melakukan validasi performa model. Dalam validasi silang, *set* pelatihan dan validasi harus saling bersilangan dalam putaran yang berurutan sehingga setiap titik data memiliki peluang untuk divalidasi [4]. Validasi silang diperlukan untuk melihat apakah model yang dibuat *overfitting* atau tidak. Gambar 2.2 menunjukkan contoh skema dari validasi silang tersebut.



Gambar 2.2 Skema 5-fold Cross-validation

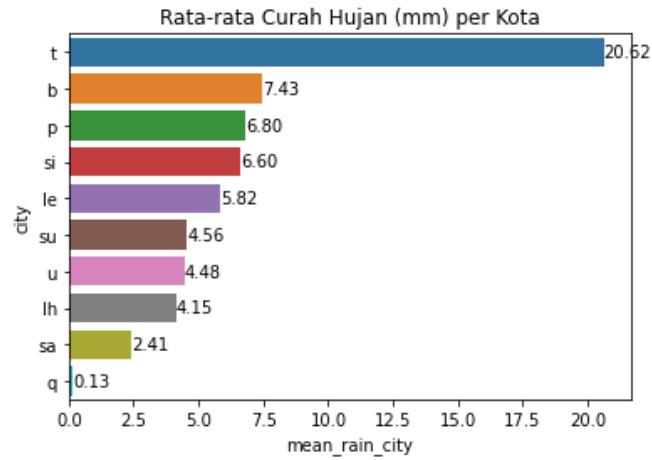
3. Pembahasan

3.1 Preprocessing

Hasil dari proses *preprocessing* yang dilakukan baik pada data harian maupun data per jam di-merge menjadi satu dengan jumlah *feature* sebanyak 959, dengan detail yaitu: 16 fitur dari data harian (termasuk prediktor ‘rain_sum (mm)’), 5 fitur hasil *resampling* dari data per jam, serta sisanya yang merupakan fitur hasil agregat data per jam. Nilai NaN pada data harian juga sudah di-impute dengan menggunakan metode *forward input filling*. Teknik *forward fill* dipilih dikarenakan jenis dataset yang termasuk ke dalam jenis *time series data*.

3.2 Exploratory Data Analysis

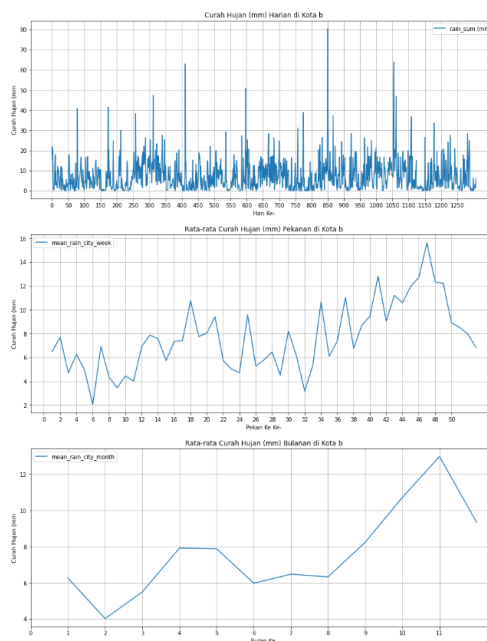
- **Rata-rata Curah Hujan setiap Kota**



Gambar 3.1 Rata-rata Curah Hujan (mm) per Kota

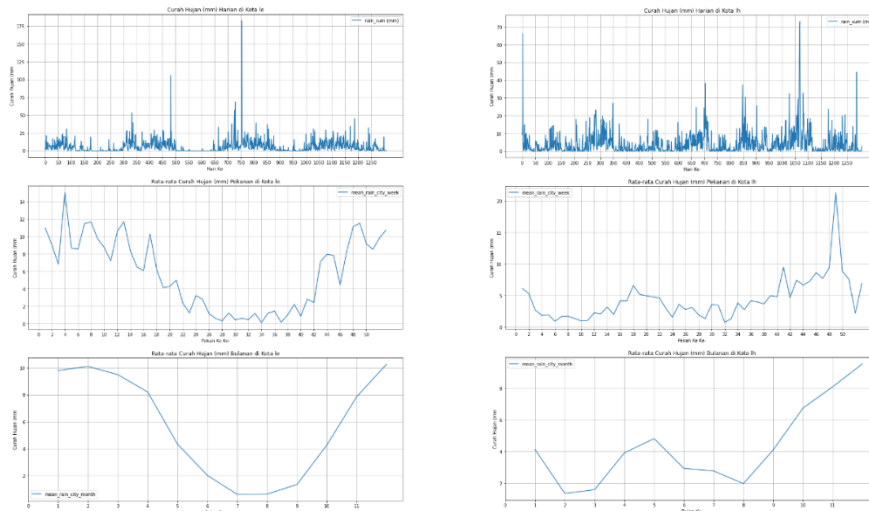
Dapat dilihat pada Gambar 3.1 bahwa kota dengan curah hujan yang paling tinggi adalah kota t dengan rata-rata curah hujan mencapai 20.62 mm per harinya. Sementara kota q adalah kota dengan rata-rata curah hujan paling rendah, yaitu hanya 0.13 mm per harinya. **Hal ini mengindikasikan bahwa beberapa kota dari kota-kota tersebut tidak dalam suatu negara karena curah hujannya tidak merata karena terdapat perbedaan karakteristik curah hujan antar kota.**

- ***Analisis Curah Hujan Setiap Kota***



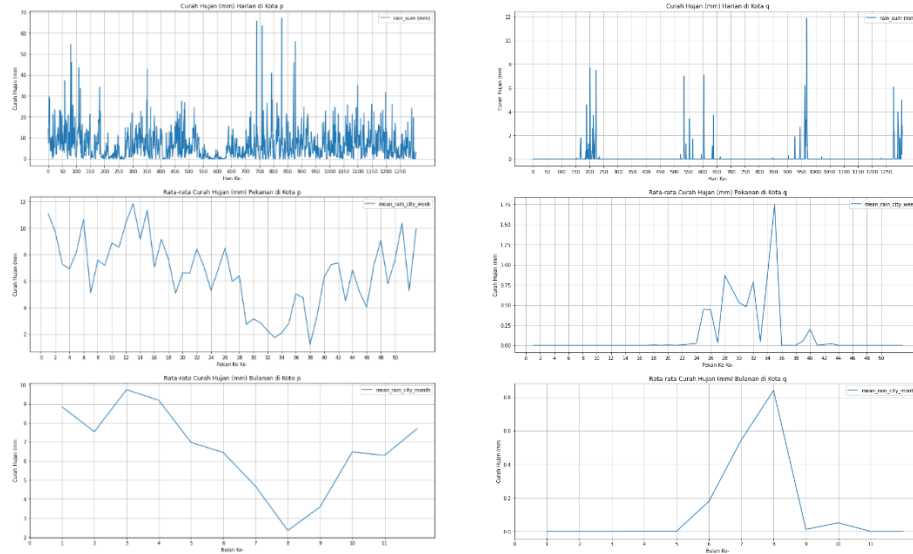
Gambar 3.2 Curah Hujan di Kota b

Menurut Gambar 3.2, curah hujan di kota b kebanyakan 0 sampai 20 mm. Kita dapat melihat pula bahwa di kota b siklus berulang kurang lebih 300 hari sekali (10-11 Bulan). **Rata-rata curah hujan tertinggi pada kota b terdapat pada pekan ke 32-48 atau pada bulan Agustus sampai November.**



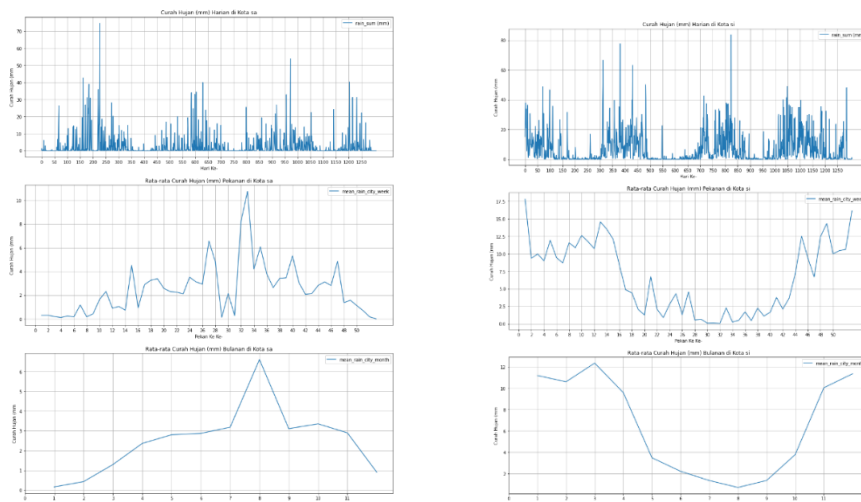
Gambar 3.3 Curah Hujan di Kota le dan lh

Menurut Gambar 3.3, curah hujan di kota le kebanyakan 0 sampai 10 mm. Kita dapat melihat pula bahwa di kota le siklus berulang kurang lebih 350-an hari sekali (11-12 Bulan). **Rata-rata curah hujan tertinggi pada kota le terdapat pada pekan ke 42 sampai pekan ke 16 tahun setelahnya atau pada bulan Oktober sampai April, dengan puncak musim ujan terjadi pada bulan Januari.** Menurut Gambar 3.3 pula, curah hujan di kota lh kebanyakan 0 sampai 20 mm. Kita dapat melihat pula bahwa di kota le siklus berulang kurang lebih 350-an hari sekali (11-12 Bulan). **Rata-rata curah hujan tertinggi pada kota lh terdapat pada pekan ke 44 sampai pekan ke 50 atau pada bulan September sampai Desember, dengan puncak musim ujan terjadi pada bulan Desember.**



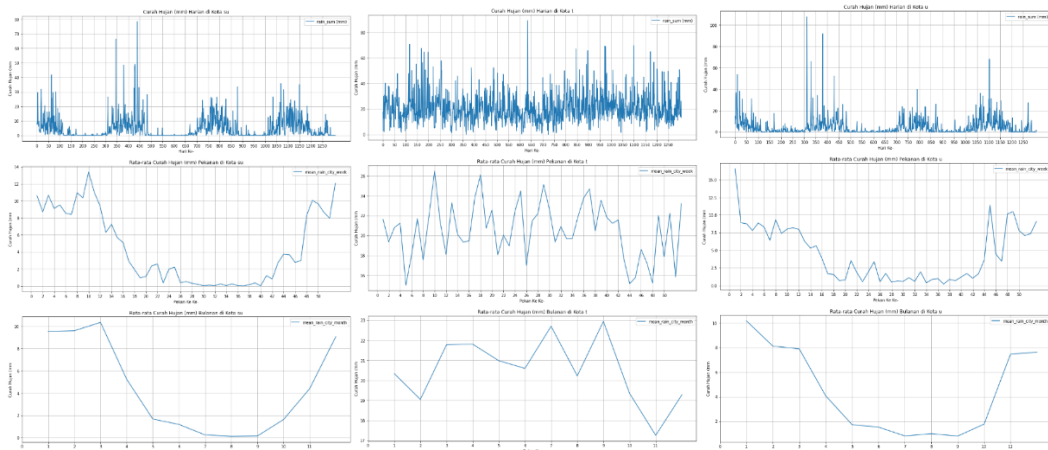
Gambar 3.4 Curah Hujan di Kota p dan q

Menurut Gambar 3.4, curah hujan di kota p kebanyakan 0 sampai 10 mm. Kita dapat melihat pula bahwa di kota p siklus berulang kurang lebih 300 hari sekali (10-11 Bulan). **Rata-rata curah hujan tertinggi pada kota b terdapat pada pekan ke 8-14 atau pada bulan Februari sampai Maret.** Sedangkan pada kota q, curah hujan kebanyakan berada di 0 – 1 mm. Kita dapat melihat pula bahwa di kota q siklus berulang kurang lebih 300 hari sekali. **Rata-rata curah hujan tertinggi pada kota q terdapat pada pekan ke 26-36 atau pada bulan Juli – September.**



Gambar 3.5 Curah Hujan di Kota sa dan si

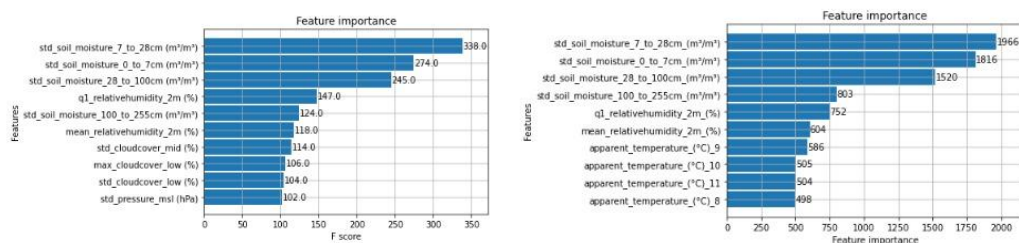
Menurut Gambar 3.5, curah hujan di kota sa kebanyakan 0 sampai 10 mm. Kita dapat melihat pula bahwa di kota sa siklus berulang kurang lebih 300 hari sekali (10-11 Bulan). **Rata-rata curah hujan tertinggi pada kota sa terdapat pada pekan ke 34-36 atau pada bulan Agustus-September.** Sedangkan pada kota si, curah hujan kebanyakan 0 – 12.5 mm, dengan siklus yang berulang kurang lebih 300 hari sekali. **Rata-rata curah hujan tertinggi pada kota si terdapat pada pekan ke 2-14 (Februari – April).**



Gambar 3.6 Curah Hujan di Kota su, t, dan u

Menurut Gambar 3.6, curah hujan di kota su kebanyakan 0 sampai 20 mm. Kita dapat melihat pula bahwa di kota su siklus berulang kurang lebih 300 hari sekali (10-11 Bulan). **Rata-rata curah hujan tertinggi pada kota b terdapat pada pekan ke 4 – 12 atau pada bulan Februari sampai April.** Hal yang sama juga terjadi pada kota t dan u.

- **Hubungan terhadap Fitur Target**



Gambar 3.7 Plot hubungan fitur terhadap fitur target (*importance plot*).

Selanjutnya dilakukan juga plot hubungan dari tiap fitur hasil *feature engineering* dan *preprocessing* terhadap fitur prediktor (rain_sum (mm)) untuk melihat *importance* dari masing-masing fitur. Plot yang pertama menunjukkan hasil analisis menggunakan ANOVA F-Test, **di mana diperoleh bahwa fitur soil_mosture, relativehumidity, serta std_cloudcover memperoleh skor yang tinggi dibandingkan dengan fitur yang lain.** Sedangkan pada plot yang kedua menunjukkan hasil *plot importance* dari model yang sudah di-*train* menggunakan dataset *training*. Pada plot tersebut juga **menunjukkan bahwa fitur soil_mosture serta relative_humidity memiliki importance yang besar terhadap hasil prediksi model.**

3.3 Feature Engineering

Hasil dari proses *feature engineering* adalah *dataset* yang sudah dilakukan manipulasi sedemikian rupa yang telah siap untuk dimasukkan ke dalam model untuk proses *training* serta *fitting*. Pada fitur tanggal, data pertama kali diubah ke dalam format *datetime* terlebih dahulu agar mempermudah *handling*. Pada komponen tanggal, beberapa fitur yang diekstrak yaitu hari, bulan, tahun, seperempat tahun (*quarter*), hari dalam minggu (*day of week*), hari dalam tahun (*day of year*), dan lain-lain. Hal ini dilakukan agar fitur waktu dapat lebih merepresentasikan setiap data dengan lebih baik.

Selanjutnya, pada fitur kota, di mana sebelumnya masih berupa fitur kategorikal. Pada fitur ini, dilakukan *feature encoding* dengan teknik *one hot encoding* untuk 10 kota unik yang ada pada fitur. Kemudian dilakukan juga ekstraksi fitur durasi hari serta malam dari fitur ‘sunrise (iso8601)’ dan ‘sunset (iso8601)’. Hasil ekstraksi berupa durasi dalam satuan menit.

Dari fitur-fitur yang sudah diekstrak lebih lanjut, dilakukan proses penskalaan ulang pada data dengan menggunakan *method* MinMaxScaler() yang diambil dari pustaka *scikit-learn*. Hasilnya berupa data yang sudah *rescaled* sehingga distribusi data yang ada di setiap fitur tersebut tidak lagi *skewed*. Proses terakhir yaitu dilakukan proses *feature selection* dengan menggunakan uji ANOVA F-Test dengan P-Value < 0.05. Dari hasil *feature*

selection, jumlah akhir fitur yang akan digunakan ke dalam model adalah sebanyak 938 fitur. Terbukti hasil dari *feature engineering* menempati 3 urutan dengan *f_score* tertinggi. **Hal ini mengindikasikan bahwa *feature engineering* berhasil dilakukan.**

	Variabel	f score	p value
218	std_soil_moisture_7_to_28cm (m ³ /m ³)	11054.981011	0.0
219	std_soil_moisture_28_to_100cm (m ³ /m ³)	8564.577258	0.0
217	std_soil_moisture_0_to_7cm (m ³ /m ³)	6879.652060	0.0
498	relativehumidity_2m (%)_13	6190.249811	0.0
497	relativehumidity_2m (%)_12	6133.490480	0.0
499	relativehumidity_2m (%)_14	6034.605556	0.0
42	q1_relativehumidity_2m (%)	5935.415809	0.0
496	relativehumidity_2m (%)_11	5892.484537	0.0
500	relativehumidity_2m (%)_15	5838.139024	0.0
495	relativehumidity_2m (%)_10	5635.151845	0.0
501	relativehumidity_2m (%)_16	5618.314218	0.0
12	min_relativehumidity_2m (%)	5150.830648	0.0
78	mean_cloudcover (%)	5047.595098	0.0
48	q1_cloudcover (%)	5018.681045	0.0
502	relativehumidity_2m (%)_17	4890.733151	0.0

Gambar 3.8 Hasil F-Test

3.4 Modelling

Model XGBoost dan LightGBM di-*train* menggunakan 13198 data dan 938 fitur dengan objektif yaitu *mean squared error*. Pada proses *hyperparameter tuning* dengan Bayesian Optimization, dilakukan pembuatan *search space* terlebih dahulu yang berisikan rentang nilai dari setiap parameter yang ingin dicari kombinasi mana yang mampu menghasilkan *mean squared error* yang paling baik. Potongan kode di bawah menunjukkan *search space* dari masing-masing model.

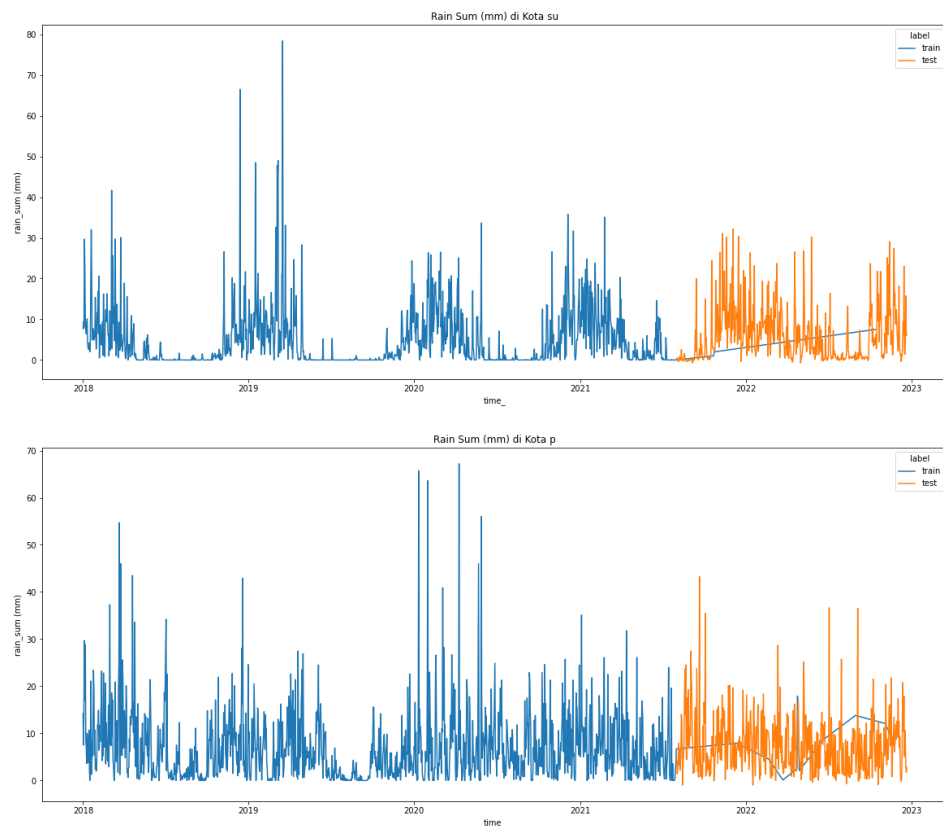
```
search_spaces = {
    'learning_rate': Real(0.01, 1.0, 'uniform'),
    'max_depth': Integer(2, 12),
    'subsample': Real(0.1, 1.0, 'uniform'),
    'colsample_bytree': Real(0.1, 1.0, 'uniform'), # subsample ratio of columns by tree
    'reg_lambda': Real(1e-9, 100., 'uniform'), # L2 regularization
    'reg_alpha': Real(1e-9, 100., 'uniform'), # L1 regularization
    'n_estimators': Integer(50, 5000)
}
```

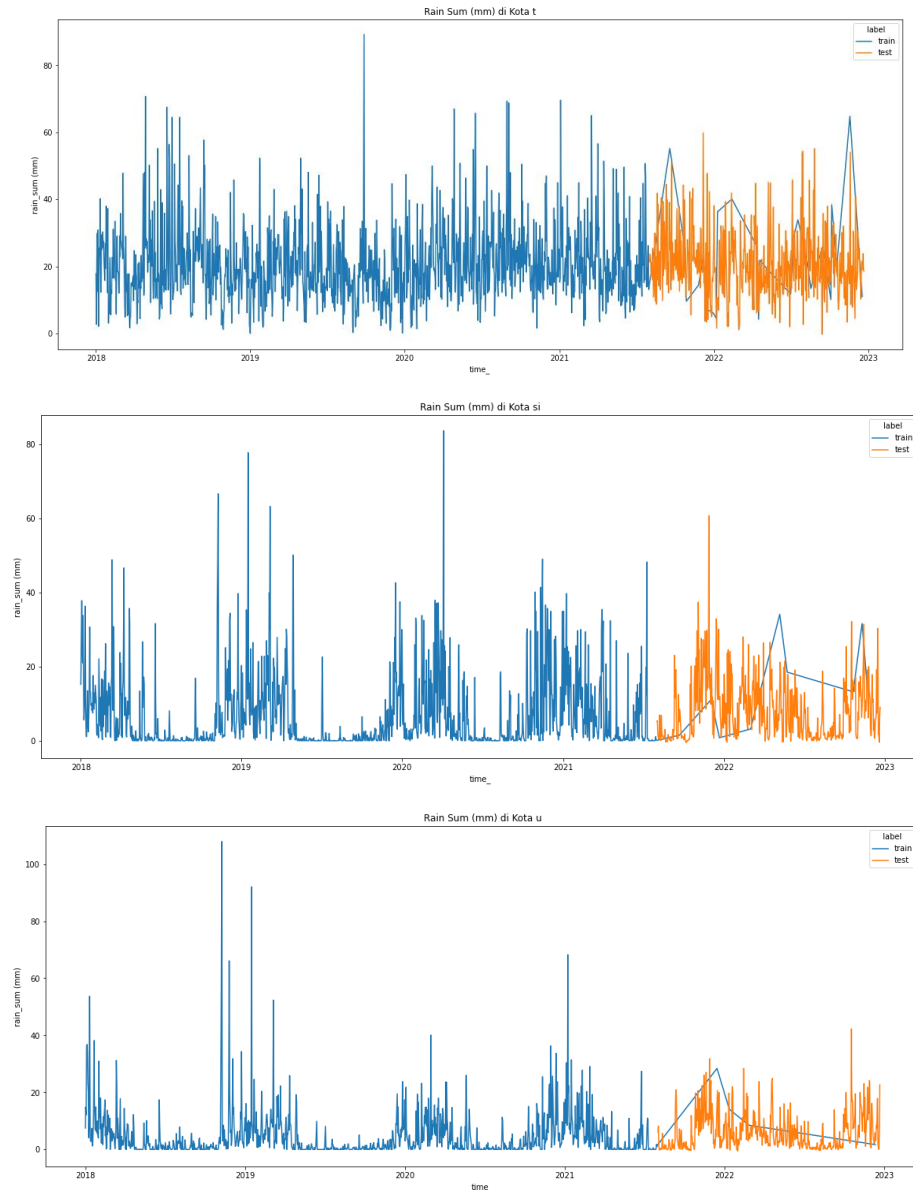
Search space dari parameter model XGBoost.

```
search_spaces = {
    'learning_rate': Real(0.01, 1.0, 'log-uniform'), # Boosting learning rate
    'n_estimators': Integer(30, 5000), # Number of boosted trees to fit
    'num_leaves': Integer(2, 512), # Maximum tree leaves for base learners
    'max_depth': Integer(-1, 256), # Maximum tree depth for base learners, <=0 means
    'subsample': Real(0.01, 1.0, 'uniform'), # Subsample ratio of the training instance
    'subsample_freq': Integer(1, 10), # Frequency of subsample, <=0 means no enable
    'colsample_bytree': Real(0.01, 1.0, 'uniform'), # Subsample ratio of columns when constructing
    'reg_lambda': Real(1e-9, 100.0, 'log-uniform'), # L2 regularization
    'reg_alpha': Real(1e-9, 100.0, 'log-uniform'), # L1 regularization
}
```

Search space dari parameter model LightGBM.

Dari proses *hyperparameter tuning* yang dilakukan, diperoleh kombinasi parameter terbaik yang digunakan pada proses *training* sebenarnya. Gambar 3.9 menunjukkan hasil plot dari data *train* yang digabung dengan data *test* hasil prediksi dari beberapa kota menggunakan model XGBoost.





Gambar 3.9 Plot data *train* dan hasil prediksi pada data *test* di beberapa kota

3.5 Validation

Hasil cross-validation yang dilakukan sekaligus melakukan tuning pada model memperlihatkan **bahwa xgboost lebih memiliki sedikit MSE dibanding dengan LGBM**. Pada XGBoost didapat skor validasi 15.0076 dan pada LGBM didapat skor validasi sebesar 15.740.

4. Kesimpulan dan Saran

- Terdapat tren *seasonality* yang cukup jelas untuk tiap kota dengan beberapa titik yang mengalami perubahan jumlah curah hujan yang cukup tinggi.

- Dari hasil analisis plot *importance* dari masing-masing fitur, terlihat bahwa fitur *soil_moisture*, *relativehumidity*, serta *cloudcover* memiliki peranan yang cukup besar dibandingkan dengan fitur target yaitu curah hujan.
- Dari hasil prediksi cuaca hujan harian dengan menggunakan kedua model, diperoleh bahwa model **XGBoost dapat menghasilkan skor MSE yang lebih rendah dibandingkan model LightGBM, yakni dengan MSE sebesar 15.0076 dan RMSE sebesar 3.87, serta MSE sebesar 24.22976** pada *public leaderboard* submisi Kaggle (30% dari data tes).
- **Peramalan cuaca sangat bermanfaat bagi pemerintah kota untuk melakukan mitigasi bencana dan dalam menyusun rencana antisipasi terhadap curah hujan yang tinggi** misalnya dengan membangun fasilitas tertentu yang bermanfaat bagi masyarakat kota.

5. Referensi

- [1] Chowdari K.K, Girisha R, and K. C. Gouda, "A study of rainfall over India using data mining," in 2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT), Mandya, India, Dec. 2015, pp. 44–47. doi: 10.1109/ERECT.2015.7498985.
- [2] C. M. Liyew and H. A. Melese, "Machine learning techniques to predict daily rainfall amount," J Big Data, vol. 8, no.1, p. 153, Dec. 2021, doi: 10.1186/s40537-021-00545-4.
- [3] Chatfield, C. Problem Solving: A Statistician's Guide (2nd ed.). Chapman and Hall. 1995. ISBN 978-0412606304.
- [4] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection". Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann. 1995. 2 (12): 1137–1143. CiteSeerX 10.1.1.48.529.

▼ Notebook TheLastDance | Datavidia

Samatha Marhaendra Putra
Rizky Alif Ramadhan
Daffa Bil Nadzary

Notebook Link: <https://colab.research.google.com/drive/1AQM1oF1rV93OJ97xCRc9JNIf4IMWh3wq?usp=sharing>

EDA Notebook Link: <https://colab.research.google.com/drive/1Vw3mpmmdUaeSr-D0SYjQfu1EPQYtGGIT?usp=sharing>

▼ 1. Preparation

```
!pip install -q kaggle
```

```
from google.colab import files
```

```
!mkdir ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle/
```

```
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle competitions download -c 'weather-forecasting-datavidia'
```

```
Downloading weather-forecasting-datavidia.zip to /content
61% 11.0M/17.9M [00:00<00:00, 112MB/s]
100% 17.9M/17.9M [00:00<00:00, 150MB/s]
```

```
!unzip -n /content/weather-forecasting-datavidia.zip
```

```
Archive: /content/weather-forecasting-datavidia.zip
  inflating: sample_submission.csv
  inflating: test.csv
  inflating: test_hourly.csv
  inflating: train.csv
  inflating: train_hourly.csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from time import time
import pprint
from functools import partial
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from xgboost import XGBRegressor
import lightgbm as lgb

from sklearn.metrics import mean_absolute_error, make_scorer, mean_squared_error
```

```
train = pd.read_csv('/content/train.csv')
train_hourly = pd.read_csv('/content/train_hourly.csv')

test = pd.read_csv('/content/test.csv')
test_hourly = pd.read_csv('/content/test_hourly.csv')

submission = pd.read_csv('/content/sample_submission.csv')
```

▼ 2. Data Preprocessing

▼ 2.1. Missing Value Handling

```
# Excluding rows that have null value on target feature
del_time = train[train['rain_sum (mm)'].isnull()]['time']
train = train[~train['time'].isin(del_time)]
```

```
# Creating function to inspect total missing values of each column and its percentage
def missing_percentage(df):
    """
    This function takes a DataFrame(df) as input and returns two columns,
    total missing values and total missing values percentage.
    """
    total = df.isnull().sum().sort_values(ascending=False)
    percent = round((df.isnull().sum().sort_values(ascending=False) / len(df) * 100), 2)
    col_miss = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

    return col_miss
```

```
train_miss = missing_percentage(train).reset_index()
train_miss = train_miss[train_miss['Total'] > 0]['index']
missing_percentage(train).reset_index()
```

	index	Total	Percent
0	winddirection_10m_dominant (°)	406	3.08
1	time	0	0.00
2	temperature_2m_max (°C)	0	0.00
3	temperature_2m_min (°C)	0	0.00
4	apparent_temperature_max (°C)	0	0.00
5	apparent_temperature_min (°C)	0	0.00
6	sunrise (iso8601)	0	0.00
7	sunset (iso8601)	0	0.00
8	shortwave_radiation_sum (MJ/m²)	0	0.00
9	rain_sum (mm)	0	0.00
10	snowfall_sum (cm)	0	0.00
11	windspeed_10m_max (km/h)	0	0.00
12	windgusts_10m_max (km/h)	0	0.00
13	et0_fao_evapotranspiration (mm)	0	0.00
14	elevation	0	0.00
15	city	0	0.00


```
train_h_miss = missing_percentage(train_hourly).reset_index()
train_h_miss = train_h_miss[train_h_miss['Total'] > 0]['index']
missing_percentage(train_hourly)
```


	Total	Percent	
winddirection_10m (°)	602	0.19	
winddirection_100m (°)	347	0.11	
windspeed_10m (km/h)	170	0.05	
soil_moisture_100_to_255cm (m³/m³)	170	0.05	
soil_moisture_7_to_28cm (m³/m³)	170	0.05	
soil_moisture_0_to_7cm (m³/m³)	170	0.05	
soil_temperature_100_to_255cm (°C)	170	0.05	
soil_temperature_28_to_100cm (°C)	170	0.05	
soil_temperature_7_to_28cm (°C)	170	0.05	
soil_temperature_0_to_7cm (°C)	170	0.05	
vapor_pressure_deficit (kPa)	170	0.05	
et0_fao_evapotranspiration (mm)	170	0.05	
windgusts_10m (km/h)	170	0.05	
windspeed_100m (km/h)	170	0.05	
temperature_2m (°C)	170	0.05	
soil_moisture_28_to_100cm (m³/m³)	170	0.05	
diffuse_radiation (MJ/m²)	170	0.05	

```
test_miss = missing_percentage(test).reset_index()
test_miss = test_miss[test_miss['Total'] > 0]['index']
missing_percentage(test).reset_index()
```

	index	Total	Percent	
0	id	0	0.0	
1	time	0	0.0	
2	temperature_2m_max (°C)	0	0.0	
3	temperature_2m_min (°C)	0	0.0	
4	apparent_temperature_max (°C)	0	0.0	
5	apparent_temperature_min (°C)	0	0.0	
6	sunrise (iso8601)	0	0.0	
7	sunset (iso8601)	0	0.0	
8	shortwave_radiation_sum (MJ/m²)	0	0.0	
9	snowfall_sum (cm)	0	0.0	
10	windspeed_10m_max (km/h)	0	0.0	
11	windgusts_10m_max (km/h)	0	0.0	
12	winddirection_10m_dominant (°)	0	0.0	
13	et0_fao_evapotranspiration (mm)	0	0.0	
14	elevation	0	0.0	
15	city	0	0.0	

```
test_h_miss = missing_percentage(test_hourly).reset_index()
test_h_miss = test_h_miss[test_h_miss['Total'] > 0]['index']
missing_percentage(test_hourly).reset_index()
```

	index	Total	Percent	
0	winddirection_100m (°)	56	0.05	
1	time	0	0.00	
2	temperature_2m (°C)	0	0.00	
3	soil_moisture_100_to_255cm (m³/m³)	0	0.00	
4	soil_moisture_28_to_100cm (m³/m³)	0	0.00	
5	soil_moisture_7_to_28cm (m³/m³)	0	0.00	
6	soil_moisture_0_to_7cm (m³/m³)	0	0.00	
7	soil_temperature_100_to_255cm (°C)	0	0.00	
8	soil_temperature_28_to_100cm (°C)	0	0.00	
9	soil_temperature_7_to_28cm (°C)	0	0.00	
10	soil_temperature_0_to_7cm (°C)	0	0.00	
11	vapor_pressure_deficit (kPa)	0	0.00	
12	et0_fao_evapotranspiration (mm)	0	0.00	
13	windgusts_10m (km/h)	0	0.00	
14	winddirection_10m (°)	0	0.00	
15	windspeed_100m (km/h)	0	0.00	
16	windspeed_10m (km/h)	0	0.00	
17	direct_normal_irradiance (W/m²)	0	0.00	
18	diffuse_radiation (W/m²)	0	0.00	
19	direct_radiation (W/m²)	0	0.00	
20	shortwave_radiation (W/m²)	0	0.00	
21	cloudcover_high (%)	0	0.00	
22	cloudcover_mid (%)	0	0.00	

2.2. Changing data type of date-related columns

```
train['time_'] = pd.to_datetime(train['time'])
train['sunrise (iso8601)'] = pd.to_datetime(train['sunrise (iso8601)'])
train['sunset (iso8601)'] = pd.to_datetime(train['sunset (iso8601)'])

test['time_'] = pd.to_datetime(test['time'])
test['sunrise (iso8601)'] = pd.to_datetime(test['sunrise (iso8601)'])
test['sunset (iso8601)'] = pd.to_datetime(test['sunset (iso8601)'])

train_hourly['time_'] = pd.to_datetime(train_hourly['time'])
train_hourly['date'] = pd.to_datetime(train_hourly['time']).dt.date

test_hourly['time_'] = pd.to_datetime(test_hourly['time'])
test_hourly['date'] = pd.to_datetime(test_hourly['time']).dt.date
```

2.3. Imputation

```
# Sorting the data
train = train.sort_values(['city', 'time_'])
train_hourly = train_hourly.sort_values(['city', 'time_'])
test = test.sort_values(['city', 'time_'])
test_hourly = test_hourly.sort_values(['city', 'time_'])

# Creating new unique id
train['id_'] = train['time_'] + train['city']
train_hourly['id_'] = train_hourly['date'].astype(str)+train_hourly['city'].astype(str)
test['id_']= test['time_'] + test['city']
test_hourly['id_'] = test_hourly['date'].astype(str)+test_hourly['city'].astype(str)

# Imputation using `ffill`
for x in train_miss :
    train[x] = train[x].ffill()

for x in train_h_miss :
    train_hourly[x] = train_hourly[x].ffill()
```

```

for x in test_miss :
    test[x] = test[x].ffill()

for x in test_h_miss :
    test_hourly[x] = test_hourly[x].ffill()

```

▼ 2.4. Creating Aggregated Features

```

# Creating temporary dataframes to store aggregated values of each predictor
min_train = train_hourly.groupby(['id_']).min().reset_index().add_prefix('min_')
q1_train = train_hourly.groupby(['id_']).quantile(0.25).reset_index().add_prefix('q1_')
mean_train = train_hourly.groupby(['id_']).mean().reset_index().add_prefix('mean_')
median_train = train_hourly.groupby(['id_']).median().reset_index().add_prefix('median_')
q3_train = train_hourly.groupby(['id_']).quantile(0.75).reset_index().add_prefix('q3_')
max_train = train_hourly.groupby(['id_']).max().reset_index().add_prefix('max_')
std_train = train_hourly.groupby(['id_']).std().reset_index().add_prefix('std_')

```

```

# Merging temporary dataframes to the daily data
train_ = pd.merge(train, min_train, left_on=['id_'], right_on=['min_id_'], how='inner')
train_ = pd.merge(train_, q1_train, left_on=['id_'], right_on=['q1_id_'], how='inner')
train_ = pd.merge(train_, mean_train, left_on=['id_'], right_on=['mean_id_'], how='inner')
train_ = pd.merge(train_, median_train, left_on=['id_'], right_on=['median_id_'], how='inner')
train_ = pd.merge(train_, q3_train, left_on=['id_'], right_on=['q3_id_'], how='inner')
train_ = pd.merge(train_, max_train, left_on=['id_'], right_on=['max_id_'], how='inner')
train_ = pd.merge(train_, std_train, left_on=['id_'], right_on=['std_id_'], how='inner')

train_ = train_.drop(columns=['min_id_', 'q1_id_', 'mean_id_', 'median_id_', 'q3_id_', 'max_id_', 'std_id_'])

```

```

# Creating temporary dataframes to store aggregated values of each predictor
mean_test = test_hourly.groupby(['id_']).mean().reset_index().add_prefix('mean_')
median_test = test_hourly.groupby(['id_']).median().reset_index().add_prefix('median_')
min_test = test_hourly.groupby(['id_']).min().reset_index().add_prefix('min_')
max_test = test_hourly.groupby(['id_']).max().reset_index().add_prefix('max_')
q1_test = test_hourly.groupby(['id_']).quantile(0.25).reset_index().add_prefix('q1_')
q3_test = test_hourly.groupby(['id_']).quantile(0.75).reset_index().add_prefix('q3_')
std_test = test_hourly.groupby(['id_']).std().reset_index().add_prefix('std_')

```

```

# Merging temporary dataframes to the daily data

test_ = pd.merge(test, min_test, left_on=['id_'], right_on=['min_id_'], how='inner')
test_ = pd.merge(test_, q1_test, left_on=['id_'], right_on=['q1_id_'], how='inner')
test_ = pd.merge(test_, mean_test, left_on=['id_'], right_on=['mean_id_'], how='inner')
test_ = pd.merge(test_, median_test, left_on=['id_'], right_on=['median_id_'], how='inner')
test_ = pd.merge(test_, q3_test, left_on=['id_'], right_on=['q3_id_'], how='inner')
test_ = pd.merge(test_, max_test, left_on=['id_'], right_on=['max_id_'], how='inner')
test_ = pd.merge(test_, std_test, left_on=['id_'], right_on=['std_id_'], how='inner')

test_ = test_.drop(columns=['min_id_', 'q1_id_', 'mean_id_', 'median_id_', 'q3_id_', 'max_id_', 'std_id_'])

```

▼ 2.5. Merging Daily Data and Hourly Data

```

# Creating temporary feature called `label` to label each row

train_['label'] = 'train'
test_['label'] = 'test'

```

```

# Set the time column as the index for `df_train_hourly` and `df_test_hourly`
df_train_hourly = train_hourly.set_index('time')
df_test_hourly = test_hourly.set_index('time')

# Convert the index to a datetime index
df_train_hourly.index = pd.to_datetime(df_train_hourly.index)
df_test_hourly.index = pd.to_datetime(df_test_hourly.index)

```

```

df_train_hourly = df_train_hourly.reset_index()
df_test_hourly = df_test_hourly.reset_index()

df_train_hourly['date'] = df_train_hourly.time.dt.date
df_train_hourly['hour'] = df_train_hourly.time.dt.hour
df_test_hourly['date'] = df_test_hourly.time.dt.date
df_test_hourly['hour'] = df_test_hourly.time.dt.hour

```

```
df_train_hourly.drop(['time'], axis=1, inplace=True)
df_test_hourly.drop(['time'], axis=1, inplace=True)
```

```
df_train_hourly = pd.pivot_table(df_train_hourly, index=['date', 'city'], columns='hour')
df_test_hourly = pd.pivot_table(df_test_hourly, index=['date', 'city'], columns='hour')
```

```
df_train_hourly.columns = [col[0] + "_" + str(col[1]) for col in df_train_hourly.columns]
df_test_hourly.columns = [col[0] + "_" + str(col[1]) for col in df_test_hourly.columns]
```

```
df_train_hourly = df_train_hourly.reset_index()
df_test_hourly = df_test_hourly.reset_index()
```

```
# Set the time column as the index for `df_train`
df_train = train_.set_index('time')
```

```
# Convert the index to a datetime index
df_train.index = pd.to_datetime(df_train.index)
```

```
df_train = df_train.reset_index()
```

```
df_train['date'] = df_train.time.dt.date
```

```
df_train.drop(['time'], axis=1, inplace=True)
```

```
df_train_merged = pd.merge(df_train, df_train_hourly, left_on=['date', 'city'], right_on=['date', 'city'])
```

```
# Set the time column as the index for `df_test`
df_test = test_.set_index('time')
```

```
# Convert the index to a datetime index
df_test.index = pd.to_datetime(df_test.index)
```

```
df_test = df_test.reset_index()
```

```
df_test['date'] = df_test.time.dt.date
```

```
df_test.drop(['time'], axis=1, inplace=True)
```

```
df_test_merged = pd.merge(df_test, df_test_hourly, left_on=['date', 'city'], right_on=['date', 'city'])
```

```
train = df_train_merged
test = df_test_merged
```

```
train.shape, test.shape
```

```
((13198, 959), (4972, 959))
```

▼ 3. Feature Engineering

▼ 3.1. Extracting Date Properties

```
# Concatenating the train and test data
df = pd.concat([train, test]).sort_values(by=['time_', 'city']).reset_index(drop=True)
```

```
# Extracting various date properties, then assign its value to new columns
df['dayofweek'] = df['time_'].dt.dayofweek.astype(int)
df['is_weekend'] = ((df['time_'].dt.dayofweek.astype(int)) > 5).map({True:1, False:0})
df['dayofyear'] = df['time_'].dt.dayofyear.astype(int)
df['weakofyear'] = df['time_'].dt.weekofyear.astype(int)
df['week'] = df['time_'].dt.week.astype(int)
df['day'] = df['time_'].dt.day.astype(int)
df['month'] = df['time_'].dt.month.astype(int)
df['year'] = df['time_'].dt.year.astype(int)
df['quarter'] = df['time_'].dt.quarter.astype(int)
df['is_month_start'] = df['time_'].dt.is_month_start.astype(int)
df['is_month_end'] = df['time_'].dt.is_month_end.astype(int)
df['is_quarter_end'] = df['time_'].dt.is_quarter_end.astype(int)
df['is_quarter_start'] = df['time_'].dt.is_quarter_start.astype(int)
df['is_year_end'] = df['time_'].dt.is_year_end.astype(int)
df['is_year_start'] = df['time_'].dt.is_year_start.astype(int)
df['is_leap_year'] = df['time_'].dt.is_leap_year.astype(int)
```

```
# Creating `day_length` and `night_length` features in minutes
df['day_length'] = (df['sunset (iso8601)'] - df['sunrise (iso8601)']).dt.seconds.astype(int) / 60

q_day = df[(df['day_length'] != 0) & (df['city'] == 'q')][['time_', 'day_length']].sort_values(by='time_')
q_day['time_'] = q_day['time_']
q_day = q_day.set_index('time_')

q_day['quarter'] = q_day['time_'].dt.quarter.astype(int)
mean_day_length = q_day.groupby('quarter')['day_length'].mean().reset_index()

index_0 = df[df['day_length'] == 0].index

for i in np.array(index_0):
    if df.loc[i, 'quarter'] == 1:
        df.loc[i, 'day_length'] = 561.147368
    elif df.loc[i, 'quarter'] == 2:
        df.loc[i, 'day_length'] = 1108.078947
    elif df.loc[i, 'quarter'] == 3:
        df.loc[i, 'day_length'] = 946.060870
    elif df.loc[i, 'quarter'] == 4:
        df.loc[i, 'day_length'] = 415.848837

df['night_length'] = 1440 - df['day_length']
```

```
# Checking for correlations between each predictor and target feature
df.corr()['rain_sum (mm)'].reset_index().sort_values(by='rain_sum (mm)', ascending=False)
```

	index	rain_sum (mm)
5	rain_sum (mm)	1.000000
219	std_soil_moisture_7_to_28cm (m³/m³)	0.675172
220	std_soil_moisture_28_to_100cm (m³/m³)	0.627361
218	std_soil_moisture_0_to_7cm (m³/m³)	0.585394
499	relativehumidity_2m (%)_13	0.565076
...
766	surface_pressure (hPa)_16	-0.486928
763	surface_pressure (hPa)_13	-0.487056
764	surface_pressure (hPa)_14	-0.487280
765	surface_pressure (hPa)_15	-0.487439
942	id	NaN

961 rows × 2 columns

```
train_ = df[df['label'] == 'train']
test_ = df[df['label'] == 'test']
```

```
train = train_.sample(train_.shape[0])
test = test_.sample(test_.shape[0])
```

3.2. Encoding Categorical Feature

```
# One hot encoding on `city` feature
cat_features = ['city']

for feature in cat_features:
    a = pd.get_dummies(train[feature], prefix=feature)
    frames = [train, a]
    train = pd.concat(frames, axis=1)

    b = pd.get_dummies(test[feature], prefix=feature)
    frames = [test, b]
    test = pd.concat(frames, axis=1)
```

```
# Saving preprocessed train and test data as csv files
train.to_csv('train_clean.csv', index=False)
test.to_csv('test_clean.csv', index=False)
```

```
X = train.select_dtypes(include=['float', 'int'])
X = X.drop(columns=['rain_sum (mm)', 'id'])
```

```
X_test = test.select_dtypes(include=['float', 'int'])
X_test = X_test.drop(columns=['id'])

y = train[['rain_sum (mm)']]

X_test = X_test[X.columns]
```

▼ 3.3. Normalization

```
from sklearn.preprocessing import MinMaxScaler
```

```
scale = MinMaxScaler()
```

```
X = pd.DataFrame(scale.fit_transform(X.values), columns=X.columns, index=X.index)
X_test = pd.DataFrame(scale.transform(X_test.values), columns=X_test.columns, index=X_test.index)
```

```
X_test_ = X_test
```

▼ 3.4. Feature Selection

```
# ANOVA test
from sklearn.feature_selection import f_regression

f_score, p_value = f_regression(X, y)
f_test = pd.DataFrame()
f_test['Variabel'] = X.columns
f_test['f_score'] = f_score
f_test['p_value'] = p_value
f_test = f_test.sort_values(by=['f_score'], ascending=False)
f_test.head(20)
```

	Variabel	f_score	p_value	
218	std_soil_moisture_7_to_28cm (m³/m³)	11054.981011	0.0	
219	std_soil_moisture_28_to_100cm (m³/m³)	8564.577258	0.0	
217	std_soil_moisture_0_to_7cm (m³/m³)	6879.652060	0.0	
498	relativehumidity_2m (%)_13	6190.249811	0.0	
497	relativehumidity_2m (%)_12	6133.490480	0.0	
499	relativehumidity_2m (%)_14	6034.605556	0.0	
42	q1_relativehumidity_2m (%)	5935.415809	0.0	
496	relativehumidity_2m (%)_11	5892.484537	0.0	
500	relativehumidity_2m (%)_15	5838.139024	0.0	
495	relativehumidity_2m (%)_10	5635.151845	0.0	
501	relativehumidity_2m (%)_16	5618.314218	0.0	
12	min_relativehumidity_2m (%)	5150.830648	0.0	
78	mean_cloudcover (%)	5047.595098	0.0	
48	q1_cloudcover (%)	5018.681045	0.0	
502	relativehumidity_2m (%)_17	4890.733151	0.0	
108	median_cloudcover (%)	4849.630126	0.0	
494	relativehumidity_2m (%)_9	4584.365279	0.0	
72	mean_relativehumidity_2m (%)	4501.843289	0.0	
10	elevation	4237.063897	0.0	
503	relativehumidity_2m (%)_18	4115.277117	0.0	

```
# Selecting features that have p-value below .05
sel = np.array(f_test[(f_test['p_value'] < 0.05) & (~f_test['p_value'].isnull())]['Variabel'])

X = X[sel]
X_test = X_test[sel]
```

```
X.shape, X_test.shape
```



```
def cv_mse(model, X, y):  
    mse = (-cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=kf))  
    return (mse)
```

▼ 4.2. XGBRegressor

```
# Setting the basic regressor  
reg = XGBRegressor(objective='reg:squarederror', tree_method='gpu_hist')
```

```
# Setting the search space  
search_spaces = {  
    'learning_rate': Real(0.01, 1.0, 'uniform'),  
    'max_depth': Integer(2, 12),  
    'subsample': Real(0.1, 1.0, 'uniform'),  
    'colsample_bytree': Real(0.1, 1.0, 'uniform'), # subsample ratio of columns by tree  
    'reg_lambda': Real(1e-9, 100., 'uniform'), # L2 regularization  
    'reg_alpha': Real(1e-9, 100., 'uniform'), # L1 regularization  
    'n_estimators': Integer(50, 5000)  
}
```

```
opt = BayesSearchCV(estimator=reg,  
                    search_spaces=search_spaces,  
                    scoring=scoring,  
                    cv=kf,  
                    n_jobs=1,  
                    n_iter=20,  
                    return_train_score=False,  
                    refit=True,  
                    iid=False, # if not iid it optimizes on the cv score  
                    optimizer_kwargs={'base_estimator': 'GP'}, # optimizer parameters: we use Gaussian Process (GP)  
                    random_state=0,  
                    verbose=100)
```

```
# Running the optimizer  
overdone_control = DeltaYStopper(delta=0.0001) # We stop if the gain of the optimization becomes too small  
  
best_params_xgb = report_perf(opt, X, y, 'XGBRegressor',  
                              callbacks=[overdone_control])
```



```
[CV 5/5; 1/1] END colsample_bytree=1.0, learning_rate=0.045389466723668, max_depth=9, n_estimators=1158, reg_alpha=93.5296681373
[CV 4/5; 1/1] START colsample_bytree=1.0, learning_rate=0.045389466723668, max_depth=9, n_estimators=1158, reg_alpha=93.5296681373
[CV 4/5; 1/1] END colsample_bytree=1.0, learning_rate=0.045389466723668, max_depth=9, n_estimators=1158, reg_alpha=93.5296681373
[CV 5/5; 1/1] START colsample_bytree=1.0, learning_rate=0.045389466723668, max_depth=9, n_estimators=1158, reg_alpha=93.5296681373
[CV 5/5; 1/1] END colsample_bytree=1.0, learning_rate=0.045389466723668, max_depth=9, n_estimators=1158, reg_alpha=93.5296681373
XGBoost took 3478.55 seconds, candidates checked: 20, best CV score: -15.008 ± 2.218
Best parameters:
OrderedDict([('colsample_bytree', 0.5737547253606028),
             ('learning_rate', 0.07767866742287785),
             ('max_depth', 5),
             ('n_estimators', 1085),
             ('reg_alpha', 23.459722486222752),
             ('reg_lambda', 35.87270784309438),
             ('subsample', 0.6807161976101201)])
```

```
# Setting up the best params
```

```
best_params_xgb = {
    'learning_rate': 0.07767866742287785,
    'max_depth': 5,
    'subsample': 0.6807161976101201,
    'colsample_bytree': 0.5737547253606028,
    'reg_lambda': 35.87270784309438,
    'reg_alpha': 23.459722486222752,
    'n_estimators': 1085}
```

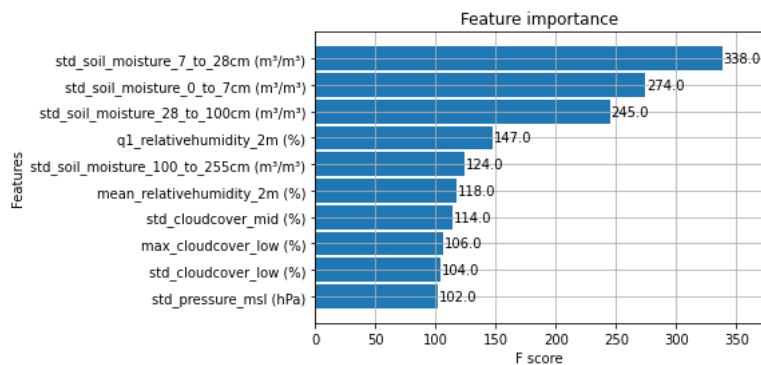
```
# Transferring the best parameters to our basic regressor
```

```
xgb_tuned = XGBRegressor(**best_params_xgb, tree_method='gpu_hist')
```

```
from xgboost import plot_importance, plot_tree
```

```
xgb_tuned.fit(X, y)
```

```
_ = plot_importance(xgb_tuned, height=0.9, max_num_features=10)
```



```
score = cv_mse(xgb_tuned, X, y)
print('XGB Tuned: {:.4f} ({:.4f})\n'.format(score.mean(), score.std()))
```

```
XGB Tuned: 15.0076 (2.2177)
```

4.3. LGBMRegressor

```
# Setting the basic regressor
```

```
reg2 = lgb.LGBMRegressor(boosting_type='gbdt',
                          objective='regression',
                          metric='rmse',
                          device='gpu',
                          n_jobs=1,
                          verbose=-1,
                          random_state=0)
```

```
# Setting the search space
```

```
search_spaces = {
    'learning_rate': Real(0.01, 1.0, 'log-uniform'), # Boosting learning rate
    'n_estimators': Integer(30, 5000), # Number of boosted trees to fit
    'num_leaves': Integer(2, 512), # Maximum tree leaves for base learners
    'max_depth': Integer(-1, 256), # Maximum tree depth for base learners, <=0 means no limit
    'subsample': Real(0.01, 1.0, 'uniform'), # Subsample ratio of the training instance
    'subsample_freq': Integer(1, 10), # Frequency of subsample, <=0 means no enable
    'colsample_bytree': Real(0.01, 1.0, 'uniform'), # Subsample ratio of columns when constructing each tree
    'reg_lambda': Real(1e-9, 100.0, 'log-uniform'), # L2 regularization
    'reg_alpha': Real(1e-9, 100.0, 'log-uniform'), # L1 regularization
}
```

```
# Wrapping everything up into the Bayesian optimizer
opt = BayesSearchCV(estimator=reg,
                    search_spaces=search_spaces,
                    scoring=scoring,
                    cv=kf,
                    n_iter=20,                                # max number of trials
                    n_jobs=1,                                # number of jobs
                    iid=False,                               # if not iid it optimizes on the cv score
                    return_train_score=False,
                    refit=False,
                    optimizer_kwargs={'base_estimator': 'GP'},    # optimizer parameters: we use Gaussian Process (GP)
                    random_state=0,
                    verbose=100)                                # random state for replicability

# Running the optimizer
overdone_control = DeltaYStopper(delta=0.0001)                # We stop if the gain of the optimization becomes too small

best_params_lgbm = report_perf(opt, X, y, 'LGBMRegressor',
                                callbacks=[overdone_control])

[CV 5/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=562, num_leaves=512, reg_alpha=1e-09, reg_
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5; 1/1] START colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 1/5; 1/1] END colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 2/5; 1/1] START colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 2/5; 1/1] END colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 3/5; 1/1] START colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 3/5; 1/1] END colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 4/5; 1/1] START colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 4/5; 1/1] END colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 5/5; 1/1] START colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
[CV 5/5; 1/1] END colsample_bytree=1.0, learning_rate=1.0, max_depth=255, n_estimators=30, num_leaves=512, reg_alpha=0.057271001
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5; 1/1] START colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_
[CV 1/5; 1/1] END colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_l
[CV 2/5; 1/1] START colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_
[CV 2/5; 1/1] END colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_l
[CV 3/5; 1/1] START colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_
[CV 3/5; 1/1] END colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_l
[CV 4/5; 1/1] START colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_
[CV 4/5; 1/1] END colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_l
[CV 5/5; 1/1] START colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_
[CV 5/5; 1/1] END colsample_bytree=0.6815430729193241, learning_rate=0.2038089563661785, max_depth=143, n_estimators=3398, num_l
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, r
[CV 1/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, reg_
[CV 2/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, r
[CV 2/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, reg_
[CV 3/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, r
[CV 3/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, reg_
[CV 4/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, r
[CV 4/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, reg_
[CV 5/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, r
[CV 5/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=2732, num_leaves=2, reg_alpha=100.0, reg_
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, r
[CV 1/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, reg_
[CV 2/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, r
[CV 2/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, reg_
[CV 3/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, r
[CV 3/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, reg_
[CV 4/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, r
[CV 4/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, reg_
[CV 5/5; 1/1] START colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, r
[CV 5/5; 1/1] END colsample_bytree=0.01, learning_rate=0.01, max_depth=-1, n_estimators=3914, num_leaves=2, reg_alpha=1e-09, reg_
LGBMRegressor took 9432.00 seconds, candidates checked: 20, best CV score: -15.740 ± 3.483
Best parameters:
OrderedDict([('colsample_bytree', 0.5334048246546964),
            ('learning_rate', 0.013055565482435798),
            ('max_depth', 87),
            ('n_estimators', 982),
            ('num_leaves', 125),
            ('reg_alpha', 5.799589814357769e-06),
            ('reg_lambda', 0.011556350804882232),
            ('subsample', 0.4144872817307234),
            ('subsample_freq', 8)])

# Setting up the best params
best_params_lgbm = {
    'colsample_bytree': 0.5334048246546964,
    'learning_rate': 0.013055565482435798,
    'max_depth': 87,
    'n_estimators': 982,
```

```
'num_leaves': 125,
'reg_alpha': 5.799589814357769e-06,
'reg_lambda': 0.011556350804882232,
'subsample': 0.4144872817307234,
'subsample_freq': 8}
```

```
# Transferring the best parameters to our basic regressor
lgbm_tuned = lgb.LGBMRegressor(boosting_type='gbdt',
                                objective='regression',
                                metric='mse',
                                device='gpu',
                                n_jobs=1,
                                verbose=-1,
                                random_state=0,
                                **best_params_lgbm)
```

```
score = cv_mse(lgbm_tuned, X, y)
print("LGBM Tuned: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

▼ 4.4. Creating Submission File

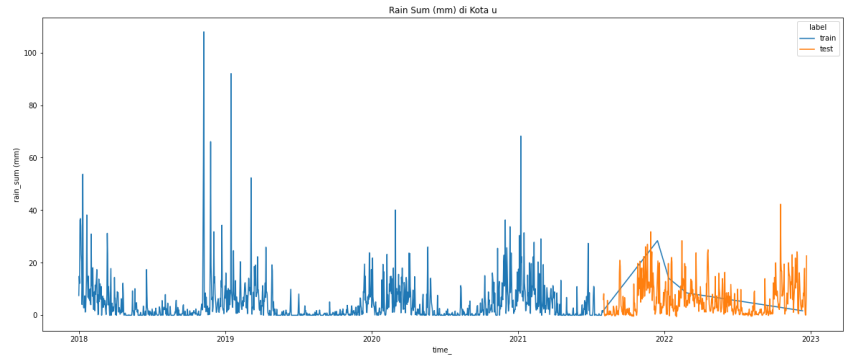
▼ 4.4.1 XGBRegressor

```
y_pred = xgb_tuned.predict(X_test)
```

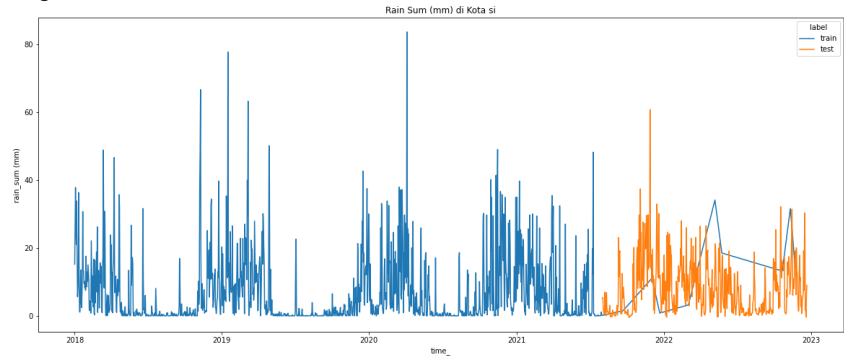
```
test['rain_sum (mm)'] = y_pred
```

```
df_ = pd.concat([train, test]).sort_values(by='time_')
```

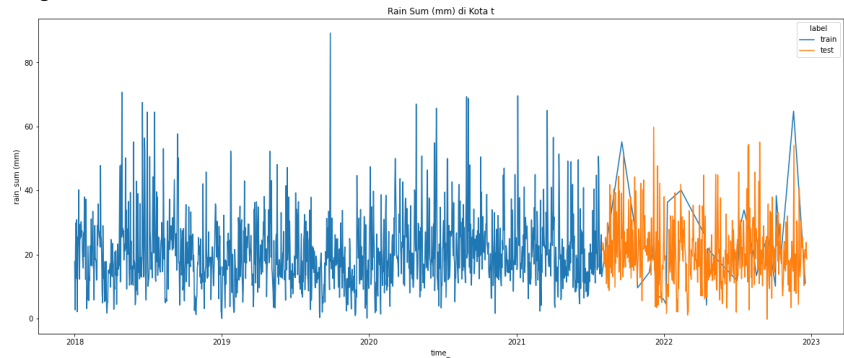
```
for x in df_['city'].unique() :
    df_city = df_[df_['city'] == x]
    fig, ax = plt.subplots(figsize=(20, 8))
    sns.lineplot(x='time_', y='rain_sum (mm)', data=df_city, hue='label')
    plt.title('Rain Sum (mm) at Kota '+str(x))
    plt.show()
    plt.savefig('Rain Sum (mm) at Kota '+str(x)+'.png')
```



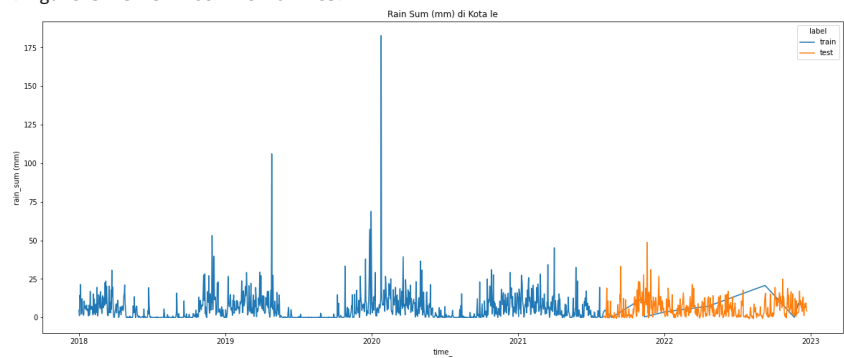
<Figure size 432x288 with 0 Axes>



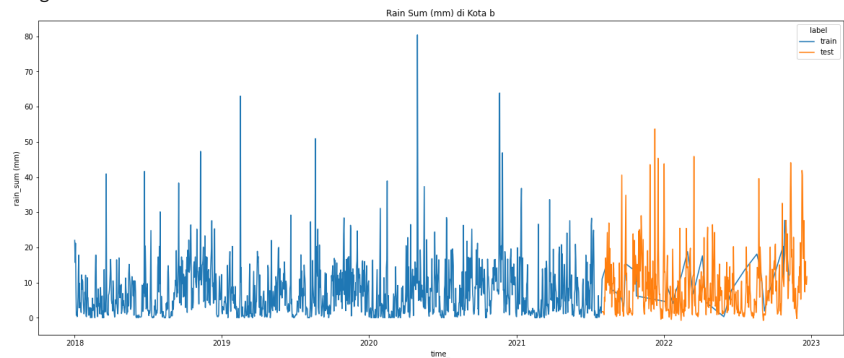
<Figure size 432x288 with 0 Axes>



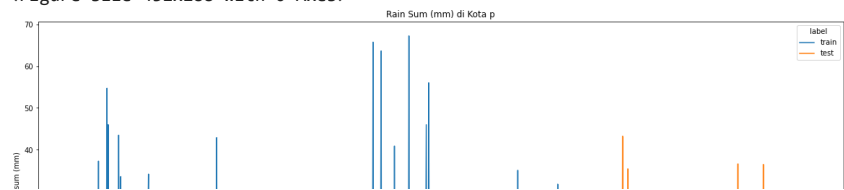
<Figure size 432x288 with 0 Axes>

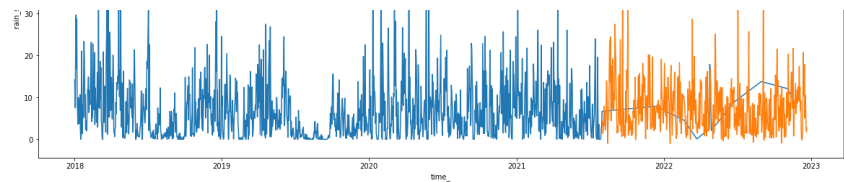


<Figure size 432x288 with 0 Axes>

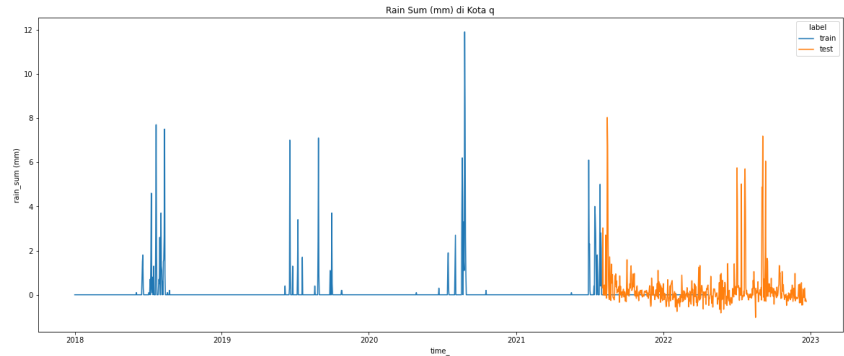


<Figure size 432x288 with 0 Axes>

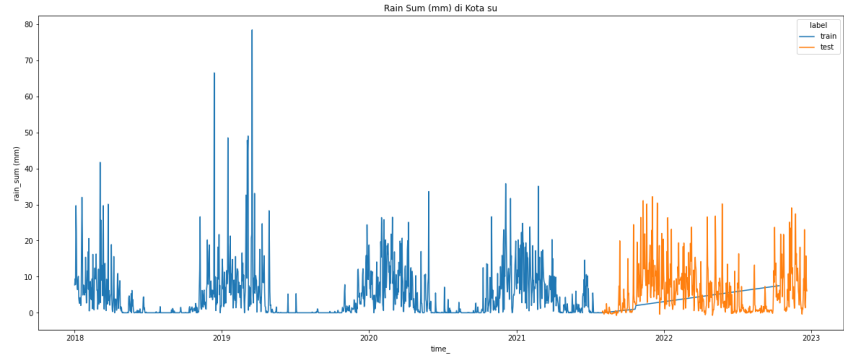




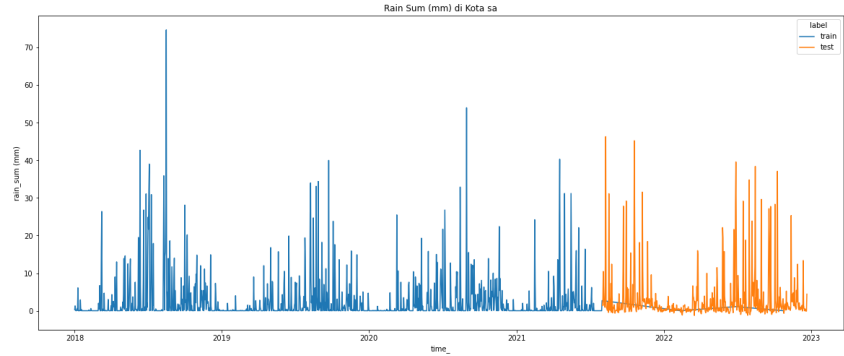
<Figure size 432x288 with 0 Axes>



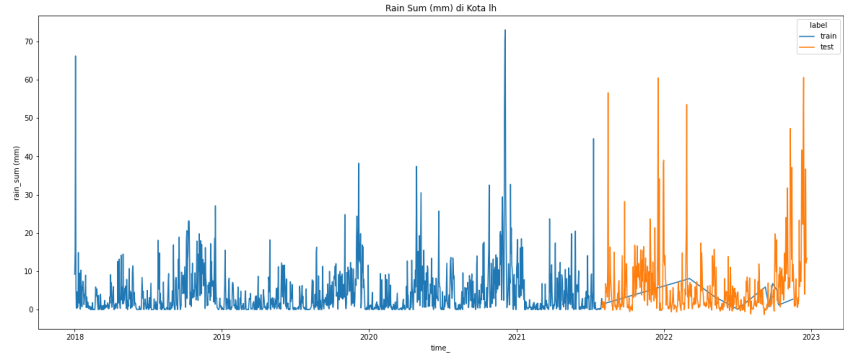
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

```
submission_ = test[['id', 'rain_sum (mm)']]
submission_ = submission_[~submission_['id'].isnull()].reset_index(drop=False).sort_values(by='id')
```

```
submission = submission_.reset_index(drop=True)
submission['id'] = submission['id'].astype(int)
submission = submission.drop(columns='index')
```

```
submission
```

	id	rain_sum (mm)
0	0	0.223396
1	1	0.535051
2	2	0.037786
3	3	-0.111958
4	4	-0.386251
...
4967	4967	0.301898
4968	4968	-0.289670
4969	4969	-0.172741
4970	4970	-0.228900
4971	4971	-0.301656

4972 rows × 2 columns

```
submission.to_csv('submission_xgbr.csv', index=False)
```

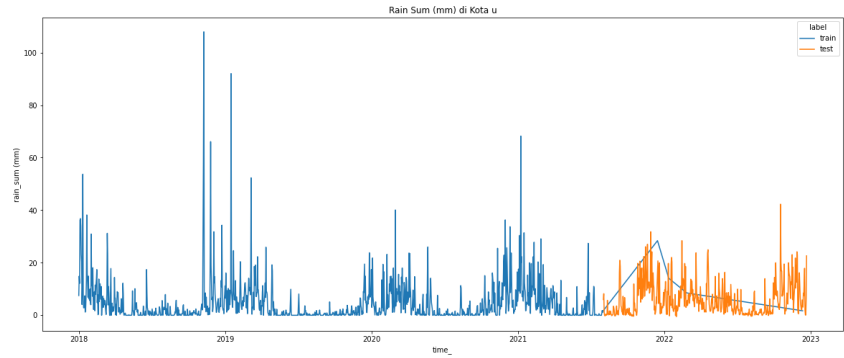
▼ 4.4.2 LGBMRegressor

```
y_pred = lgbm_tuned.predict(X_test)
```

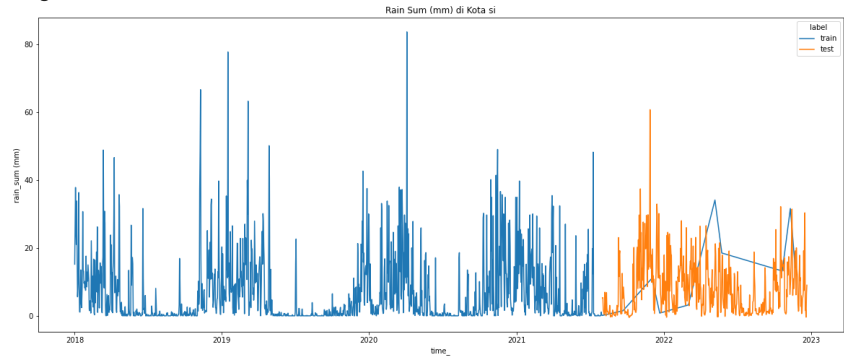
```
test['rain_sum (mm)'] = y_pred
```

```
df_ = pd.concat([train, test]).sort_values(by='time_')
```

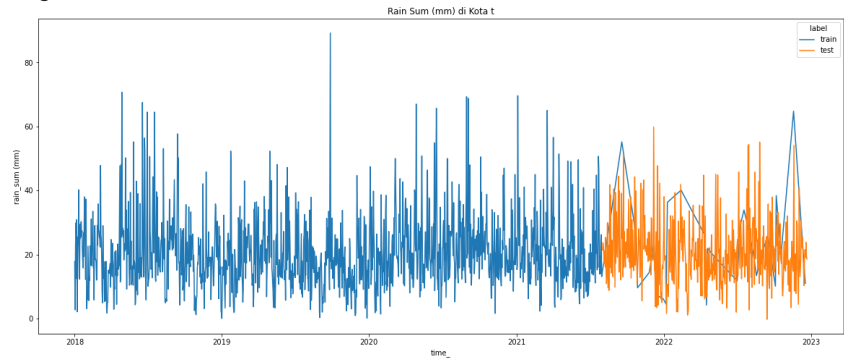
```
for x in df_['city'].unique() :
    df_city = df_[df_['city'] == x]
    fig, ax = plt.subplots(figsize=(20, 8))
    sns.lineplot(x='time_', y='rain_sum (mm)', data=df_city, hue='label')
    plt.title('Rain Sum (mm) at Kota '+str(x))
    plt.show()
    plt.savefig('Rain Sum (mm) at Kota '+str(x)+'.png')
```



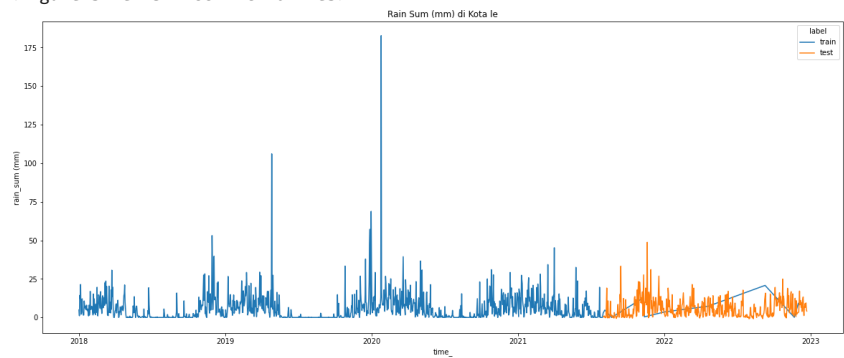
<Figure size 432x288 with 0 Axes>



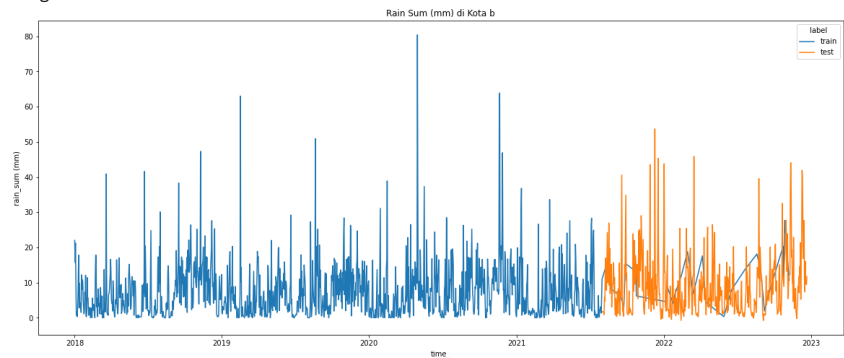
<Figure size 432x288 with 0 Axes>



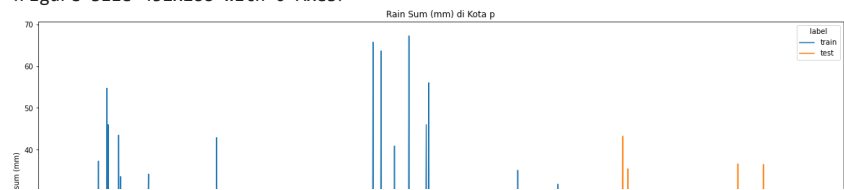
<Figure size 432x288 with 0 Axes>

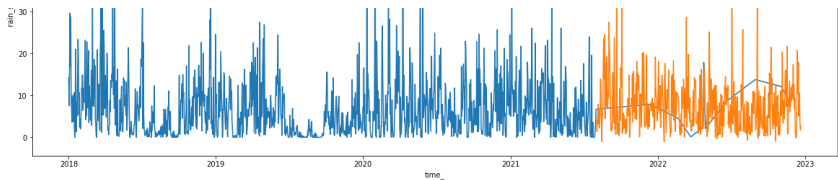


<Figure size 432x288 with 0 Axes>

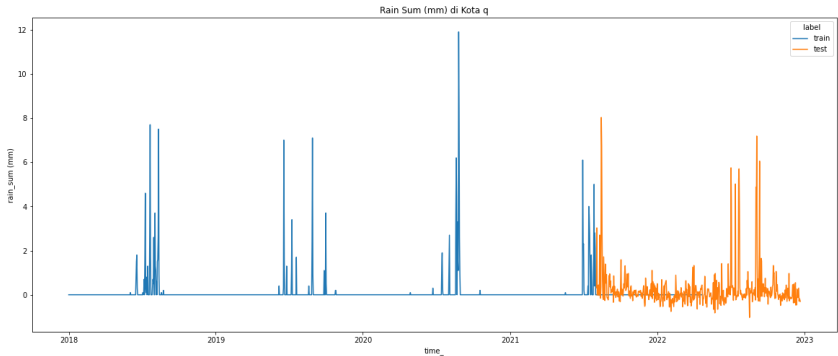


<Figure size 432x288 with 0 Axes>

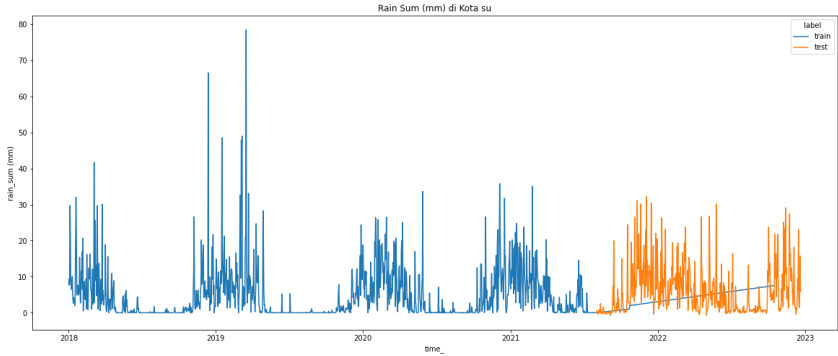




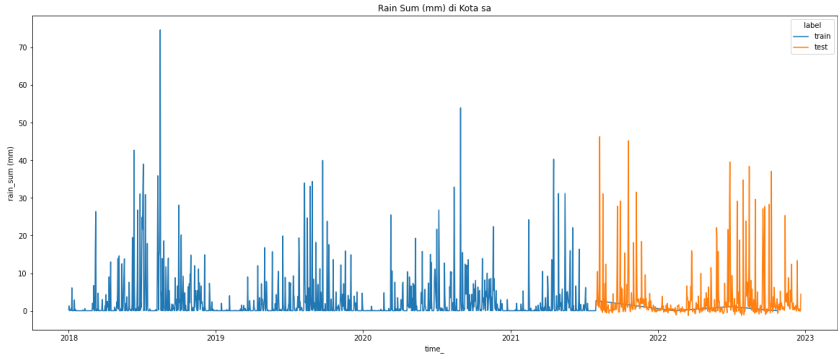
<Figure size 432x288 with 0 Axes>



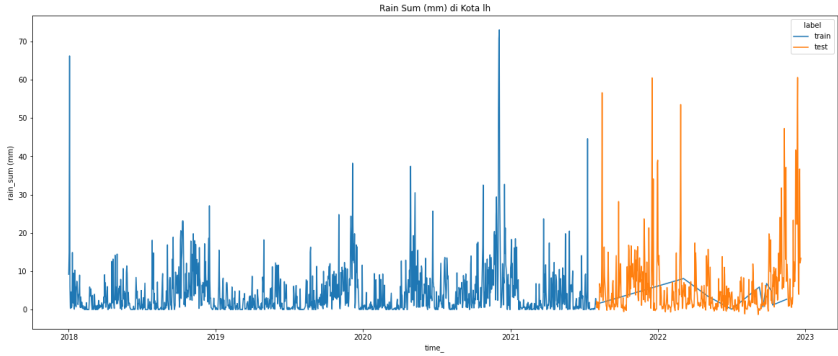
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>