

***FORECASTING KECEPATAN RATA-RATA MENGGUNAKAN
LIGHTGBM***

DATATHON COMPETITION 2023
RISTEK FASILKOM UI



DISUSUN OLEH:

TIM DATA DEVINERS

KRISNA BAYU DHARMA PUTRA
VENUS ANGELA KURNIAWAN
WILDAN DZAKY RAMADHANI

**UNIVERSITAS GADJAH MADA
YOGYAKARTA**

2023

BAB 1: PENDAHULUAN

A. Latar Belakang

Dalam beberapa dekade terakhir, pertumbuhan populasi dan urbanisasi yang cepat telah mengakibatkan peningkatan yang signifikan dalam jumlah kendaraan bermotor yang beroperasi di jalan raya. Dampak dari lonjakan lalu lintas ini secara nyata tercermin dalam bentuk kemacetan lalu lintas yang meresahkan, polusi udara yang meningkat, konsumsi bahan bakar yang semakin besar, serta dampak-dampak negatif lainnya terhadap lingkungan dan kualitas hidup masyarakat di kota-kota besar.

Dalam menghadapi tantangan ini, para peneliti, ahli transportasi, dan pihak pemerintah telah berusaha keras untuk mengembangkan berbagai strategi serta solusi guna meredam dampak buruk dari lalu lintas yang semakin padat. Salah satu alat yang menjadi inti dari upaya ini adalah prediksi kecepatan rata-rata kendaraan setiap jamnya.

Prediksi ini memiliki peran yang sangat penting, karena informasi tentang kecepatan kendaraan rata-rata di berbagai waktu dapat memberikan bantuan berharga kepada para pengemudi, penyedia layanan transportasi, dan pihak berwenang dalam membuat keputusan yang lebih tepat terkait perjalanan yang mereka lakukan.

B. Rumusan Masalah

Rumusan masalah pada makalah ini adalah sebagai berikut:

- Bagaimana cara memprediksi kecepatan rata-rata kendaraan tiap jamnya secara akurat berdasarkan data yang ada?

C. Tujuan

Tujuan dari makalah ini adalah sebagai berikut:

- Menemukan dan mengembangkan model prediksi yang mampu mengestimasi dengan akurat kecepatan rata-rata kendaraan pada setiap jamnya, berdasarkan data yang dianalisis serta data yang dijadikan acuan.

BAB 2: DATA PREPROCESSING

A. Dataset Awal

Dataset awal dari panitia hanya menyediakan 4 kolom fitur, yaitu ['waktu_setempat', 'id_jalan', 'id_titik_mulai', 'id_titik_akhir', 'rerata_kecepatan']. Seperti yang terlihat pada gambar 2.1. Setelah dilakukan pengecekan, terlihat bahwa data ini sudah bersih

dari missing value seperti yang terlihat pada gambar 2.2.

	waktu_setempat	id_jalan	id_titik_mulai	id_titik_akhir	rerata_kecepatan
0	2020-02-01 01:00:00+00:00	691007296	21390008	1425033102	29.126
1	2020-02-01 01:00:00+00:00	47010584	1677092762	579493410	46.576
2	2020-02-01 01:00:00+00:00	22932408	26486694	1930267566	36.587
3	2020-02-01 01:00:00+00:00	142479648	1111592522	3775231113	34.063
4	2020-02-01 01:00:00+00:00	8504977	5940503398	5940503394	38.336
...
398643	2020-02-22 23:00:00+00:00	3691841	18235127	1590448416	41.094
398644	2020-02-22 23:00:00+00:00	3691841	1250564256	18293380	45.902
398645	2020-02-22 23:00:00+00:00	182210371	33139383	33139375	38.918
398646	2020-02-22 23:00:00+00:00	8504977	1623682036	26467191	34.951

gambar 2.1. Dataset awal yang diberikan.

```
train.isnull().sum()

waktu_setempat    0
id_jalan          0
id_titik_mulai    0
id_titik_akhir    0
rerata_kecepatan  0

test.isnull().sum()

id                0
waktu_setempat    0
id_jalan          0
id_titik_mulai    0
id_titik_akhir    0
```

gambar 2.2. Hasil pengecekan missing value

B. Data Eksternal

a. Ekstraksi dari “id_jalan”

Ekstraksi data dari “id_jalan” dilakukan dengan menggunakan API openstreetmap dengan URL "https://www.openstreetmap.org/api/0.6/way/{id_value}" dimana id_value berisi id_jalan pada setiap row. Ekstraksi dilakukan dengan mengambil beberapa data yang menurut kami penting dari referensi tag yang tersedia. Ekstraksi kami lakukan dengan pemanfaatan library XML.

Pada URL, tim kami mengekstrak beberapa data tambahan seperti berikut :

	busway	cycleway	foot	highway	lanes	lit	maxspeed	name	oneway	operator	sidewalk	surface	lanesbackward	lanesforward
0	Not included	lane	Not included	trunk	3	yes	30 mph	Upper Tooting Road	Not included	Transport for London	separate	asphalt	Not included	2
1	Not included	Not included	Not included	primary	Not included	yes	30 mph	High Road	Not included	Not included	Not included	Not included	Not included	Not included
2	Not included	Not included	Not included	secondary	Not included	yes	20 mph	Nightingale Lane	Not included	Not included	Not included	Not included	Not included	Not included
3	Not included	Not included	Not included	primary	3	yes	20 mph	Lavender Hill	Not included	Not included	Not included	asphalt	2	1
4	Not included	Not included	Not included	primary	2	yes	30 mph	Harrow Road	Not included	Not included	Not included	asphalt	Not included	Not included

gambar 2.3 menunjukkan hasil dari ekstraksi URL untuk “id_jalan”

Setelah data terekstrak, data akan dilakukan penggabungan dengan dataset awal yang diberikan.

b. Ekstraksi dari “id_titik_mulai” dan “id_titik_akhir”

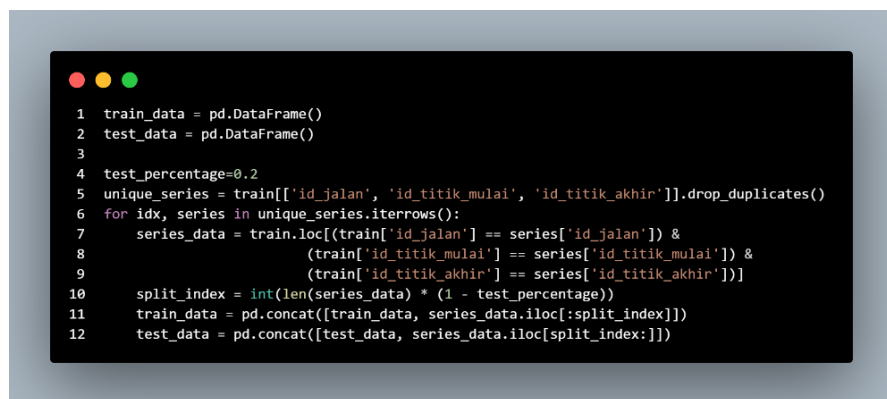
Ekstraksi data dari "id_titik_mulai" dan "id_titik_akhir" dilakukan dengan menggunakan API OpenStreetMap melalui URL "https://www.openstreetmap.org/node/{node_id}/history", di mana node_id mengandung id_titik_mulai atau id_titik_akhir pada setiap baris. Ekstraksi ini dilakukan untuk mendapatkan nilai latitude (garis lintang) dan longitude (garis bujur) dari setiap id. Proses ekstraksi dilaksanakan dengan memanfaatkan

pustaka (library) Beautiful Soup. Pada URL ini, kami mengekstrak nilai latitude dan longitude untuk "id_titik_mulai" dan "id_titik_akhir" dengan mengambil data longitude dan latitude yang terakhir tercatat dalam riwayat URL tersebut. Pendekatan ini diambil untuk mengantisipasi terjadinya data kosong akibat titik lokasi yang mungkin telah dihapus oleh administrator OpenStreetMap.

c. Data Splitting

Data train yang telah ditambahkan data eksternal akan dipecah (*split*) dengan persentase 80% untuk latihan (*train*) dan 20% untuk data uji (*test*).

i. Metode Pertama

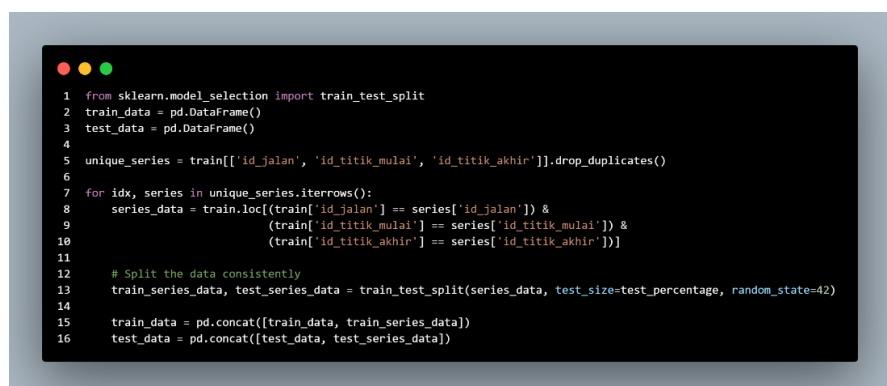


```
1 train_data = pd.DataFrame()
2 test_data = pd.DataFrame()
3
4 test_percentage=0.2
5 unique_series = train[['id_jalan', 'id_titik_mulai', 'id_titik_akhir']].drop_duplicates()
6 for idx, series in unique_series.iterrows():
7     series_data = train.loc[(train['id_jalan'] == series['id_jalan']) &
8                             (train['id_titik_mulai'] == series['id_titik_mulai']) &
9                             (train['id_titik_akhir'] == series['id_titik_akhir'])]
10    split_index = int(len(series_data) * (1 - test_percentage))
11    train_data = pd.concat([train_data, series_data.iloc[:split_index]])
12    test_data = pd.concat([test_data, series_data.iloc[split_index:]])
```

gambar 2.4. Metode pertama yang digunakan

Pada metode pertama, kami temukan inkonsistensi Ketika kami lakukan split ulang datanya, namun kami menemukannya dan menyadarinya di hari akhir lomba dan sudah melakukan tuning, sehingga kami tetap menggunakan metode split ini untuk melakukan train data dan tuning. Dokumentasi program proses tersebut dapat dilihat pada gambar 2.4.

ii. Metode Kedua



```
1 from sklearn.model_selection import train_test_split
2 train_data = pd.DataFrame()
3 test_data = pd.DataFrame()
4
5 unique_series = train[['id_jalan', 'id_titik_mulai', 'id_titik_akhir']].drop_duplicates()
6
7 for idx, series in unique_series.iterrows():
8     series_data = train.loc[(train['id_jalan'] == series['id_jalan']) &
9                             (train['id_titik_mulai'] == series['id_titik_mulai']) &
10                             (train['id_titik_akhir'] == series['id_titik_akhir'])]
11
12    # Split the data consistently
13    train_series_data, test_series_data = train_test_split(series_data, test_size=test_percentage, random_state=42)
14
15    train_data = pd.concat([train_data, train_series_data])
16    test_data = pd.concat([test_data, test_series_data])
```

gambar 2.5. Metode kedua yang digunakan

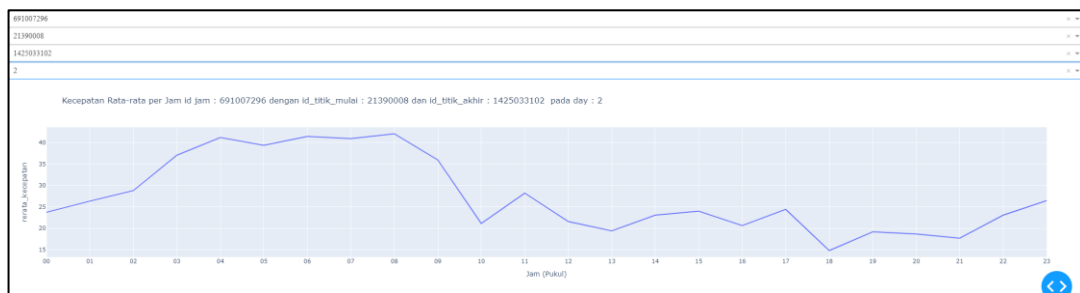
Kami menggunakan metode *split* ini untuk melakukan pengujian dan pembuatan ulang model setelah mendapatkan hasil dari hyperopt.

Selama proses pelatihan, kami membiarkan program berjalan tanpa intervensi. Namun, setelah pelatihan selesai dan karena ketidakaktifan, platform Kaggle secara otomatis menutup dan mematikan kernel kami. Kondisi ini menyebabkan kami belum dapat menyimpan data `train_data` dan `test_data` hasil dari metode split awal. Oleh karena itu, kami mengadopsi metode kedua ini untuk membuat ulang split data latih menjadi 80% `train_data` dan 20% `test_data`. Dokumentasi program terdapat pada Gambar 2.5.

BAB 3: EXPLORATORY DATA ANALYSIS

A. Grafik Garis Rerata Kecepatan Berdasarkan ID

Di sini, kami mencoba memvisualisasikan `rerata_kecepatan` per jam berdasarkan `id_jalan`, `id_titik_mulai`, dan `id_titik_akhir` dalam suatu hari tertentu menggunakan *line graph* dengan bantuan library `dash`.

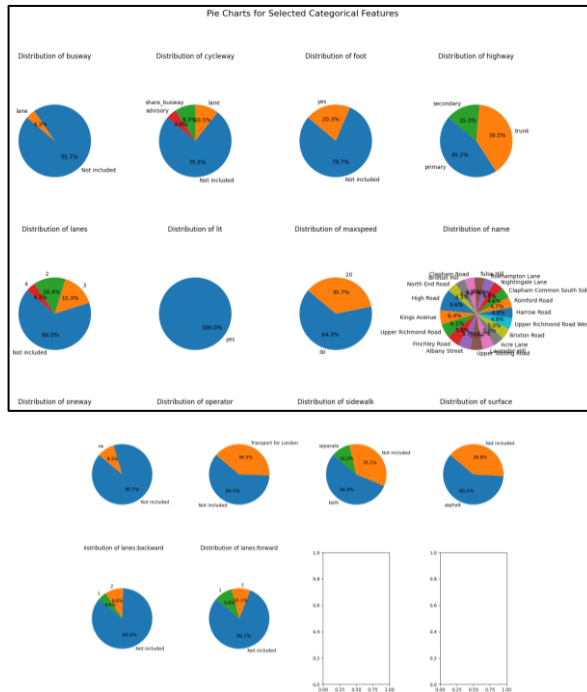


gambar 3.1. Grafik Rerata Kecepatan per Jam

Gambar 3.1 adalah salah satu contoh dari visualisasi kami, pada visualisasi ini, dapat dipilih `id_jalan`, `id_titik_mulai`, `id_titik_akhir`, dan `daynya`. Rata-rata, `rerata_kecepatan` memiliki nilai yang fluktuatif dan kadang ada saat terjadi penurunan yang curam. Ada beberapa hal menarik yang kami temukan, yaitu Ketika dini hari sekitar jam 01.00 sampai 04.00 rerata kecepatan cenderung tinggi dan Ketika pukul 14.00-17.00 kecepatan cenderung rendah. Kemungkinan hal ini disebabkan karena pada pagi hari tidak banyak kendaraan lewat sehingga pengendara dapat bergerak lebih cepat sedangkan saat sore lalu lintas akan lebih padat sehingga `rerata_kecepatan` menurun.

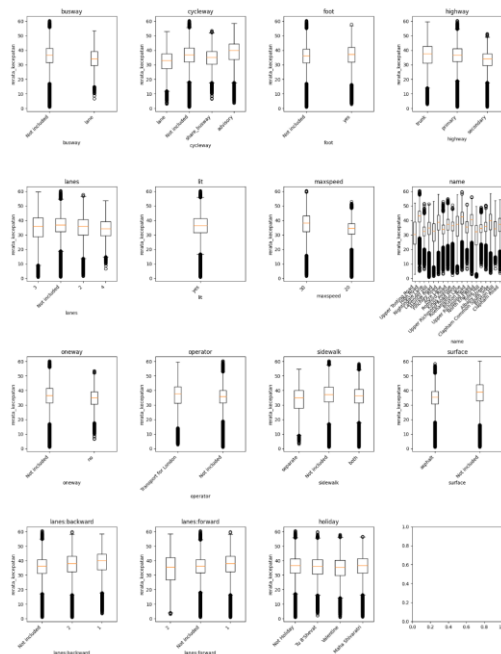
B. Count Plot Data Kategorik

Untuk data kategorik, kami memutuskan untuk melihat persebarannya terlebih dahulu. Berikut adalah hasilnya:



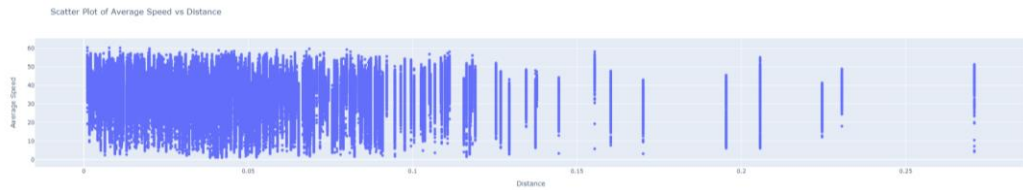
Pada gambar 3.2, terlihat bahwa banyak fitur kategorikal memiliki nilai yang tak seimbang, bahkan ada yang hanya memiliki satu jenis data saja, yaitu fitur “lit”. Fitur ini tidak memberikan informasi apapun sehingga dapat di-drop.

C. Boxplot Data Kategorik vs Rerata Kecepatan



Gambar 3.3 menunjukkan visualisasi *boxplots* data kategorik vs rerata_kecepatan. Pada *boxplot* ini, kita dapat mendapatkan informasi umum persebaran rerata_kecepatan sesuai dengan fitur-fitur tertentu. Informasi yang dapat kita dapatkan antara lain rata-rata dan median rerata_kecepatan berdasarkan fitur berjenis kategori untuk tiap *unique value*-nya.

D. Distance vs Rerata Kecepatan



gambar 3.4. Grafik Distance vs Rerata Kecepatan

Gambar diatas menunjukkan *scatterplot* dari fitur distance dan rerata_kecepatan. Dapat dilihat bahwa persebaran distance mayoritas berada di rentang 0 sampai 0.1, sehingga jarak perekaman rerata_kecepatan secara umum tidak terlalu jauh

BAB 4: FEATURE ENGINEERING

A. Pemisahan Data Tanggal

Panitia hanya menyediakan 4 kolom fitur, yaitu ['waktu_setempat', 'id_jalan', 'id_titik_mulai', 'id_titik_akhir'], sehingga kami memutuskan untuk memisahkan data tanggal menjadi fitur baru. Fitur baru ini diekstrak dari kolom “waktu_setempat” menjadi 5 kolom baru, yaitu ['year', 'month', 'day', 'hour', 'weekday'].

B. Transformasi Sin Cos untuk Fitur Waktu

```
1 def add_sine_cosine_features(df,test, column_name, period):
2     sin_transformer = FunctionTransformer(lambda x: np.sin(2 * np.pi * x / period))
3     cos_transformer = FunctionTransformer(lambda x: np.cos(2 * np.pi * x / period))
4
5     df[f'{column_name}_sin'] = sin_transformer.fit_transform(df[[column_name]])
6     df[f'{column_name}_cos'] = cos_transformer.fit_transform(df[[column_name]])
7
8     # test_df[f'{column_name}_sin'] = sin_transformer.transform(test_df[[column_name]])
9     # test_df[f'{column_name}_cos'] = cos_transformer.transform(test_df[[column_name]])
10    test[f'{column_name}_sin'] = sin_transformer.transform(test[[column_name]])
11    test[f'{column_name}_cos'] = cos_transformer.transform(test[[column_name]])
12    return df,test
13
```

gambar 4.1. Kode Transformasi Sin Cos untuk Fitur Waktu

Gambar 4.1 merupakan code untuk mengubah fitur day, hour, dan weekday menggunakan fungsi cos dan sinus. Hal ini berguna untuk mengetahui pola tertentu setiap satuan waktu. Hasilnya akan berupa float hasil perhitungan rumus:

$$y = \sin\left(2 \times \pi \times \frac{x}{\text{period}}\right)$$

Hasil transformasi sin cos dapat dilihat pada gambar 4.2.

	hour_sin	day_sin	weekday_sin	hour_cos	day_cos	weekday_cos
0	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521
1	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521
2	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521
3	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521
4	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521

gambar 4.2. Hasil Transformasi Sin Com

C. Ekstraksi Hari Spesial

```
1 def holiday(date):
2     if date == 10:
3         return 'Tu B'Shevat'
4     if date == 14:
5         return 'Valentine'
6     if date == 21:
7         return 'Maha Shivaratri'
8     if date == 25:
9         return 'Carnival / Shrove Tuesday / Pancake Day'
10    if date == 26:
11        return 'Carnival / Ash Wednesday'
12    return 'Not Holiday'
13
14 train['holiday'] = train.apply(lambda row: holiday(row['day']), axis=1)
15 test['holiday'] = test.apply(lambda row: holiday(row['day']), axis=1)
```

gambar 4.3. Metode pertama yang digunakan

Kami mencari secara online hari spesial di UK di google. Hasilnya adalah seperti gambar berikut. Selanjutnya, kami akan menambahkan fitur “holiday” sesuai dengan kode disamping.

D. Mencari Fitur Distance (Jarak) dengan Haversine

```
1 def haversine_distance(lat1, lon1, lat2, lon2):
2     R = 6371 # Radius of the Earth in kilometers
3
4     # Convert latitude and longitude from degrees to radians
5     lat1_rad = math.radians(lat1)
6     lon1_rad = math.radians(lon1)
7     lat2_rad = math.radians(lat2)
8     lon2_rad = math.radians(lon2)
9
10    # Differences in coordinates
11    dlat = lat2_rad - lat1_rad
12    dlon = lon2_rad - lon1_rad
13
14    # Haversine formula
15    a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
16    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
17    distance = R * c
18
19    return distance
20
21 train['distance'] = train.apply(lambda row: haversine_distance(row['latitude_soul'],
22 row['longitude_soul'], row['latitude_akhir'], row['longitude_akhir']), axis=1)
23 test['distance'] = test.apply(lambda row: haversine_distance(row['latitude_soul'],
24 row['longitude_soul'], row['latitude_akhir'], row['longitude_akhir']), axis=1)
```

gambar 4.4. Kode Metode Haversine

Kami mencoba untuk menghitung jarak antar dua titik, yaitu titik awal perekaman dan titik akhir. Di sini, kami memanfaatkan fitur latitude dan longitude hasil scraping kemudian menggunakan metode haversine untuk menemukan perkiraan jarak antar dua titik ini.

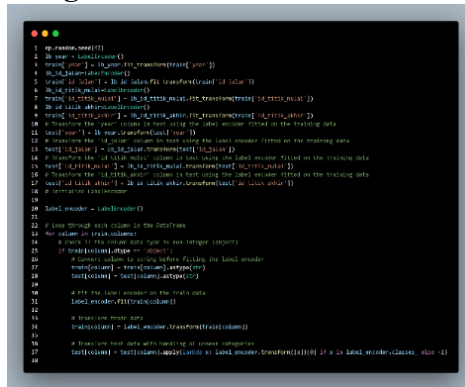
	hour_sin	day_sin	weekday_sin	hour_cos	day_cos	weekday_cos	distance
0	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521	0.007914
1	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521	0.019985
2	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521	0.044685
3	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521	0.082340
4	0.258819	0.21497	-0.974928	0.965926	0.976621	-0.222521	0.029118

gambar 4.5. Hasil Metode Haversine

E. Dropping Feature

Terdapat beberapa fitur yang di-drop. Fitur tersebut adalah rerata_kecepatan, lit, dan waktu_setempat. Fitur rerata_kecepatan di-drop karena merupakan target, fitur lit di-drop karena hanya memiliki satu nilai, dan fitur waktu_setempat tidak dibutuhkan karena sudah diwakili oleh fitur waktu year, month, day, dan hour.

F. Encoding

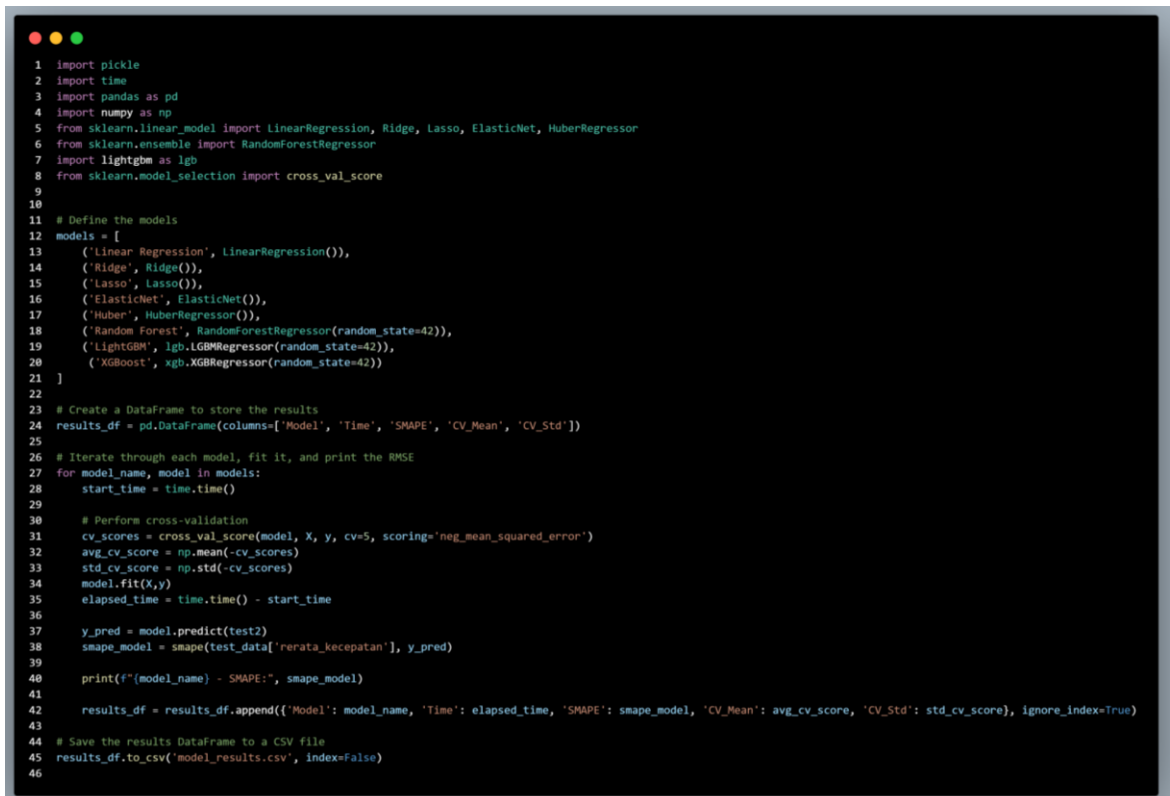


gambar 4.6. Encoding Fitur Kategorik

Kami menggunakan label encoding untuk melakukan encoding fitur-fitur kategorik.

BAB 5: MODELLING

A. Base Model



gambar 5.1. Kode Base Model

Untuk modelling, kami mencoba 8 model untuk data ini. 8 model ini adalah Linear Regression, Ridge Regression, Lasso Regression, ElasticNet, HuberRegressor, RandomForest, LightGBM, dan XGBoost Regressor. Untuk evaluasi metriknya, kami menggunakan dua evaluasi, yaitu SMAPE dan root mean squared error. Selain itu, untuk menguji efektivitas model, kami juga menghitung waktu fitting setiap model. Selanjutnya, setiap model akan dilakukan cross-validation dengan fold sebanyak 5 kali dengan scoring root mean square error. Di sini kamu juga menghitung standar deviasi serta rata-rata dari skoring cross-validation, hal ini bertujuan untuk menguji kekonsistenan model. Berikut adalah hasil dari training 8 model tersebut:

117...	Model	Time	SMAPE	CV_Mean	CV_Std
5	Random Forest	1250.059339	8.551405	36.407689	4.733350
7	XGBoost	308.604759	8.943365	32.834497	3.868161
6	LightGBM	20.095950	10.092260	31.217414	3.774821
0	Linear Regression	1.920546	14.704345	41.490268	4.086563
1	Ridge	0.639740	14.706909	41.332881	4.131875
4	Huber	39.313533	15.306508	47.775440	4.604883
3	ElasticNet	0.885396	16.951606	53.821954	4.088096
2	Lasso	0.809893	17.272168	55.464143	4.172460

gambar 5.2. Hasil Training

Hal ini menunjukkan bahwa kedua model ini memiliki waktu latih terlalu lama sehingga sangat sulit untuk dilakukan hyperparameter tuning. Oleh karena itu, tim kami memutuskan menggunakan LightGBM sebagai model utama untuk dituning. Hal ini dikarenakan LightGBM memiliki waktu latih hanya 20 detik saja sehingga memungkinkan untuk dilakukan tuning lebih lanjut.

Selain itu, CV_mean dan CV_Std dari LightGBM juga lebih tinggi dari Random Forest maupun XGBRegressor yang menandakan bahwa LightGBM lebih konsisten daripada dua model lainnya. Namun, kami juga melakukan tuning pada XGBRegressor dan RandomForest dengan jumlah loop yang lebih sedikit.

B. Hyperparameter Tuning

a. Random Forest

```

1 from hyperopt import fmin, tpe, hp
2 from sklearn.ensemble import RandomForestRegressor
3 import numpy as np
4 # Define the objective function to minimize (SMAPE)
5 def objective(params):
6     model = RandomForestRegressor(
7         n_estimators=int(params['n_estimators']),
8         max_depth=int(params['max_depth']),
9         min_samples_split=int(params['min_samples_split']),
10        min_samples_leaf=int(params['min_samples_leaf']),
11        max_features=params['max_features'],
12        random_state=42
13    )
14    model.fit(X, y)
15    y_pred = model.predict(test2)
16    smape_score = smape(test_data['rata_kecepatan'], y_pred)
17    print(f"SMAPE : {smape_score} params : {params}")
18    return smape_score
19
20 space = {
21     'n_estimators': hp.quniform('n_estimators', 60, 400, 20),
22     'max_depth': hp.quniform('max_depth', 3, 30, 2),
23     'min_samples_split': hp.quniform('min_samples_split', 2, 30, 2),
24     'min_samples_leaf': hp.quniform('min_samples_leaf', 1, 30, 2),
25     'max_features': hp.quniform('max_features', 0.1, 1, 0.1)
26 }
27
28 # Run hyperparameter tuning
29 best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=20)
30
31 print("Best hyperparameters:", best)
32

```

gambar 5.3. Hyperparameter Tuning Random Forest

```

SMAPE : 8.6821130051776 params : {'max_depth': 20.0, 'max_features': 0.3706114618756009, 'min_samples_leaf': 28.0, 'min_samples_split': 16.0, 'n_estimators': 80.0}
SMAPE : 11.566780954217177 params : {'max_depth': 18.0, 'max_features': 0.754351151590066, 'min_samples_leaf': 4.0, 'min_samples_split': 10.0, 'n_estimators': 320.0}
SMAPE : 0.640895097142584 params : {'max_depth': 20.0, 'max_features': 0.3301350808861594, 'min_samples_leaf': 24.0, 'min_samples_split': 8.0, 'n_estimators': 200.0}
SMAPE : 10.292554056893167 params : {'max_depth': 12.0, 'max_features': 0.835780361610905, 'min_samples_leaf': 2.0, 'min_samples_split': 12.0, 'n_estimators': 360.0}
SMAPE : 12.802577031922598 params : {'max_depth': 8.0, 'max_features': 0.5260660151700765, 'min_samples_leaf': 18.0, 'min_samples_split': 6.0, 'n_estimators': 260.0}
SMAPE : 0.244265236720215 params : {'max_depth': 30.0, 'max_features': 0.6560147541800355, 'min_samples_leaf': 4.0, 'min_samples_split': 20.0, 'n_estimators': 80.0}
SMAPE : 13.746242071832455 params : {'max_depth': 6.0, 'max_features': 0.211009217451027, 'min_samples_leaf': 4.0, 'min_samples_split': 6.0, 'n_estimators': 160.0}
SMAPE : 0.380523128010193 params : {'max_depth': 20.0, 'max_features': 0.44782700714826479, 'min_samples_leaf': 8.0, 'min_samples_split': 6.0, 'n_estimators': 320.0}
SMAPE : 0.400942880901425 params : {'max_depth': 20.0, 'max_features': 0.9995299979624503, 'min_samples_leaf': 16.0, 'min_samples_split': 16.0, 'n_estimators': 200.0}
SMAPE : 13.696081312711893 params : {'max_depth': 6.0, 'max_features': 0.32404510463481304, 'min_samples_leaf': 18.0, 'min_samples_split': 26.0, 'n_estimators': 160.0}
SMAPE : 0.329347432217372 params : {'max_depth': 14.0, 'max_features': 0.5801493261438579, 'min_samples_leaf': 12.0, 'min_samples_split': 6.0, 'n_estimators': 200.0}
SMAPE : 14.6354961403577 params : {'max_depth': 4.0, 'max_features': 0.3867006370709196, 'min_samples_leaf': 28.0, 'min_samples_split': 22.0, 'n_estimators': 260.0}
SMAPE : 12.00402025091734 params : {'max_depth': 20.0, 'max_features': 0.1514167181956887, 'min_samples_leaf': 12.0, 'min_samples_split': 6.0, 'n_estimators': 200.0}
SMAPE : 0.294357086020252 params : {'max_depth': 22.0, 'max_features': 0.8939360817788136, 'min_samples_leaf': 2.0, 'min_samples_split': 20.0, 'n_estimators': 200.0}
SMAPE : 0.5309782084159 params : {'max_depth': 18.0, 'max_features': 0.3440624467786747, 'min_samples_leaf': 6.0, 'min_samples_split': 14.0, 'n_estimators': 220.0}
SMAPE : 10.576976208772024 params : {'max_depth': 12.0, 'max_features': 0.36186850722970527, 'min_samples_leaf': 22.0, 'min_samples_split': 26.0, 'n_estimators': 140.0}
SMAPE : 9.425309552551599 params : {'max_depth': 14.0, 'max_features': 0.9789910306832171, 'min_samples_leaf': 6.0, 'min_samples_split': 20.0, 'n_estimators': 360.0}
SMAPE : 13.804612356048203 params : {'max_depth': 8.0, 'max_features': 0.665136353133307, 'min_samples_leaf': 16.0, 'min_samples_split': 18.0, 'n_estimators': 200.0}
SMAPE : 9.44987314724874 params : {'max_depth': 14.0, 'max_features': 0.7085401791868887, 'min_samples_leaf': 28.0, 'min_samples_split': 22.0, 'n_estimators': 220.0}
SMAPE : 0.724889717990535 params : {'max_depth': 20.0, 'max_features': 0.4080249100199169, 'min_samples_leaf': 28.0, 'min_samples_split': 16.0, 'n_estimators': 160.0}
100% 20.00 (11:52:08.00) 250.64s/trial, best loss: 8.244265236720215
Best hyperparameters: {'max_depth': 30.0, 'max_features': 0.6560147541800355, 'min_samples_leaf': 4.0, 'min_samples_split': 20.0, 'n_estimators': 80.0}

```

gambar 5.4. Hasil Hyperparameter Tuning

Dapat dilihat bahwa smape terbaik dari 20 kali looping RandomForest bernilai 8.2442. Nilai ini terbilang cukup bagus mengingat loop yang dilakukan hanya sebanyak 20 kali.

Dapat dilihat bahwa Random Forest memiliki skor SMAPE terbaik, lalu diikuti oleh XGBRegressor dan LightGBM. Namun, Random Forest memiliki waktu latih yang sangat lama, yaitu sekitar 1250 detik atau sekitar 20.8 menit, sedangkan XGBRegressor membutuhkan 308 detik atau sekitar 5.1 menit.

b. XGBRegressor

```
1 from hyperopt import fmin, tpe, hp
2 import xgboost as xgb
3 import numpy as np
4
5 # Define the objective function to minimize (SMAPE)
6 def objective(params):
7     model = xgb.XGBRegressor(
8         n_estimators=params['n_estimators'],
9         max_depth=params['max_depth'],
10        learning_rate=params['learning_rate'],
11        subsample=params['subsample'],
12        colsample_bytree=params['colsample_bytree'],
13        gamma=params['gamma'],
14        reg_alpha=params['reg_alpha'],
15        reg_lambda=params['reg_lambda'],
16        min_child_weight=params['min_child_weight'],
17        random_state=42
18    )
19    model.fit(X, y)
20    y_pred = model.predict(test2)
21    smape_score = smape(test_data['rerata_kecepatan'], y_pred)
22
23    print(f"smape : {smape_score} params : {params}")
24
25    return smape_score
26
27 space = {
28     'n_estimators': hp.quniform('n_estimators', 60, 400, 20),
29     'max_depth': hp.quniform('max_depth', 3, 30, 2),
30     'learning_rate': hp.loguniform('learning_rate', np.log(0.01), np.log(0.3)),
31     'subsample': hp.uniform('subsample', 0.5, 1),
32     'colsample_bytree': hp.uniform('colsample_bytree', 0.5, 1),
33     'gamma': hp.uniform('gamma', 0, 5),
34     'reg_alpha': hp.loguniform('reg_alpha', np.log(1e-10), np.log(1)),
35     'reg_lambda': hp.loguniform('reg_lambda', np.log(1e-10), np.log(1)),
36     'min_child_weight': hp.loguniform('min_child_weight', np.log(1e-3), np.log(10)),
37 }
38
39 # Run hyperparameter tuning
40 best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=20)
41
42 print("Best hyperparameters:", best)
43
```

gambar 5.4. Hyperparameter Tuning XGBRegressor

c. LightGBM

```
1 from hyperopt import fmin, tpe, hp
2 import lightgbm as lgb
3
4 # Define the objective function to minimize (SMAPE)
5 def objective(params):
6     model = lgb.LGBMRegressor(
7         n_estimators=params['n_estimators'],
8         max_depth=params['max_depth'],
9         min_child_samples=params['min_child_samples'],
10        learning_rate=params['learning_rate'],
11        colsample_bytree=params['colsample_bytree'],
12        subsample=params['subsample'],
13        reg_alpha=params['reg_alpha'],
14        reg_lambda=params['reg_lambda'],
15        min_child_weight=params['min_child_weight'],
16        subsample_freq=params['subsample_freq'],
17        num_leaves=params['num_leaves'],
18        min_data_in_leaf=params['min_data_in_leaf'],
19        bagging_fraction=params['bagging_fraction'],
20        bagging_freq=params['bagging_freq'],
21        feature_fraction=params['feature_fraction'],
22        lambda_l1=params['lambda_l1'],
23        lambda_l2=params['lambda_l2'],
24        random_state=42
25    )
26    model.fit(X, y)
27    y_pred = model.predict(test2)
28    smape_score = smape(test_data['rerata_kecepatan'], y_pred)
29    feature_importances = model.get_params()
30
31    print(f"smape : {smape_score} params : {feature_importances}")
32    # predicted = model.predict(test) # Use X_test_selected instead of test
33    # subs['rerata_kecepatan'] = predicted
34
35    # subs_filename = f'kaggle/working/submission_haru.csv'
36    # subs.to_csv(subs_filename, index=False)
37
38    # print(f"Submission saved for submission")
39
40    return smape_score
41
42 space = {
43     'n_estimators': hp.quniform('n_estimators', 10, 1000, 10),
44     'max_depth': hp.quniform('max_depth', 3, 100, 1),
45     'min_child_samples': hp.quniform('min_child_samples', 1, 100, 1),
46     'learning_rate': hp.loguniform('learning_rate', np.log(0.0001), np.log(0.0)),
47     'colsample_bytree': hp.uniform('colsample_bytree', 0.5, 1),
48     'subsample': hp.uniform('subsample', 0.5, 1),
49     'reg_alpha': hp.loguniform('reg_alpha', np.log(1e-10), np.log(1)),
50     'reg_lambda': hp.loguniform('reg_lambda', np.log(1e-10), np.log(1)),
51     'min_child_weight': hp.loguniform('min_child_weight', np.log(1e-3), np.log(10)),
52     'subsample_freq': hp.quniform('subsample_freq', 1, 10, 1),
53     'num_leaves': hp.quniform('num_leaves', 8, 128, 1),
54     'min_data_in_leaf': hp.quniform('min_data_in_leaf', 10, 100, 1),
55     'bagging_fraction': hp.uniform('bagging_fraction', 0.5, 1),
56     'bagging_freq': hp.quniform('bagging_freq', 1, 10, 1),
57     'feature_fraction': hp.uniform('feature_fraction', 0.5, 1),
58     'lambda_l1': hp.loguniform('lambda_l1', np.log(1e-10), np.log(10)),
59     'lambda_l2': hp.loguniform('lambda_l2', np.log(1e-10), np.log(10)),
60 }
61
62 # Run hyperparameter tuning
63 best = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=500)
64
65 print("Best hyperparameters:", best)
66
```

gambar 5.5. Hyperparameter Tuning LightGBM

Pada XGBRegressor, kami menggunakan random_state=42 dan jumlah num_evals sebanyak 20 kali. Space yang kami pilih seperti pada gambar disamping. Hasil dari tuning ini mendapatkan SMAPE sebesar 8.39464. Karena Hasil SMAPE yang didapat cukup jauh dibandingkan RandomForest dan LightGBM, kami memutuskan untuk tidak menggunakan XGBRegressor untuk metode ensembling.

Untuk Tuning LightGBM, kami menggunakan random state=42 dan menggunakan tuning parameter seperti di gambar berikut. Kami melakukan loop sebanyak 500 kali dengan evaluasi metrics adalah SMAPE. Pada tuning ini, kami berhasil mendapatkan model terbaik dengan SMAPE: 8.17719 pada data latih (*test_data*) dengan metode *split* pertama dan SMAPE 7.03 menggunakan model *split* kedua.

BAB 6: EVALUASI

Model LightGBM memiliki nilai evaluasi skor metrics terbaik, sehingga kami memutuskan untuk menggunakan model ini saja. Selain itu, nilai *importance* tiap fitur juga dilihat pada gambar berikut:

	Feature	Importance		Feature	Importance
23	hour	10950	4	cycleway	262
18	latitude_akhir	9915	14	lanes_backward	216
16	latitude_awal	8697	12	sidewalk	197
19	longitude_akhir	8431	8	maxspeed	147
17	longitude_awal	6773	15	lanes_forward	143
22	day	6735	13	surface	116
2	id_titik_akhir	6698	5	foot	96
1	id_titik_mulai	6503	3	busway	62
27	hour_sin	6034	11	operator	23
28	hour_cos	5937	10	oneway	5
26	distance	5925	20	year	0
24	weekday	4444	21	month	0

gambar 5.6. Feature Importance

BAB 7: ANALISIS HASIL PREDIKSI

Model LightGBM mampu melakukan prediksi dengan cukup baik pada dataset kali ini, sehingga model lightGBM dengan *tuning* dapat digunakan untuk menyelesaikan permasalahan pada dataset ini. Di lain sisi, model RandomForest juga memiliki hasil evaluasi SMAPE yang baik, namun RandomForest memiliki waktu latih yang terlalu lama sehingga tidak efektif untuk dilakukan *tuning*. Dengan segala pertimbangan tersebut, tim kami memutuskan bahwa model LightGBM adalah model terbaik dan paling efektif untuk menyelesaikan permasalahan ini.

BAB 8: KESIMPULAN

LightGBM memiliki efektifitas dan skor evaluasi yang baik untuk menyelesaikan permasalahan ini, namun setelah dilakukan tuning selama 500 evaluasi, sangat sulit untuk mendapatkan evaluasi SMAPE menggunakan metode *split* pertama di bawah 8 untuk data latih. Hal ini menunjukkan bahwa kemungkinan masih ada algoritma lain yang lebih efektif untuk permasalahan ini. Namun, menurut tim kami, LightGBM memiliki efisiensi yang cukup baik serta akurasi yang baik pula, sebab rata-rata waktu fitting LightGBM hanya sekitar 20-40 detik saja dan memiliki akurasi yang cukup baik pula, yaitu sebesar 8.17719 untuk metode *splitting* pertama dan 7.03569 untuk metode *splitting* kedua.