***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 I: PENDAHULUAN**

1. **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.

1. **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?

1. **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 II: DATA PREPROCESSING**

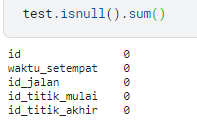
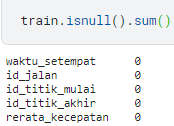
1. **Dataset awal**

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.

gambar 2.1. Dataset awal yang diberikan.

* Check null

Terlihat bahwa data ini sudah bersih dari missing value seperti yang terlihat pada gambar 2.2.

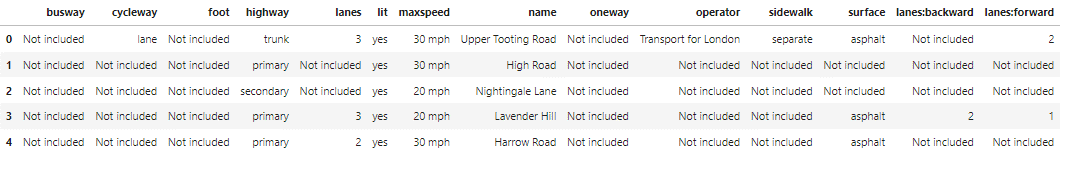


gambar 2.2. Hasil pengecekan missing value

1. **Dataset Eksternal**
2. **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](https://www.openstreetmap.org/api/0.6/way/%7Bid_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 ini, tim kami mengekstrak beberapa data tambahan seperti berikut :



ggambar 2.3 menunjukkan hasil dari ekstraksi URL untuk “id\_jalan”

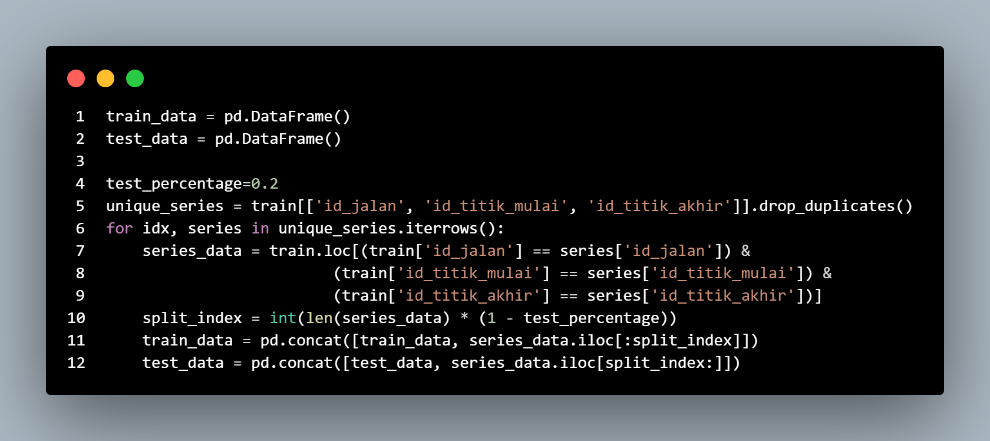
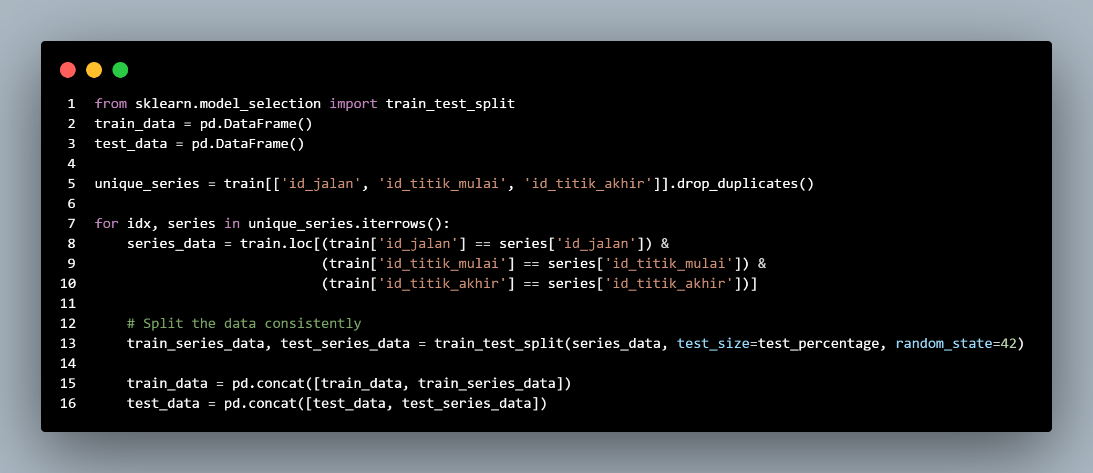
Setelah data terekstrak, data akan dilakukan penggabungan dengan dataset awal yang diberikan.

1. **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.

1. **Data splitting**

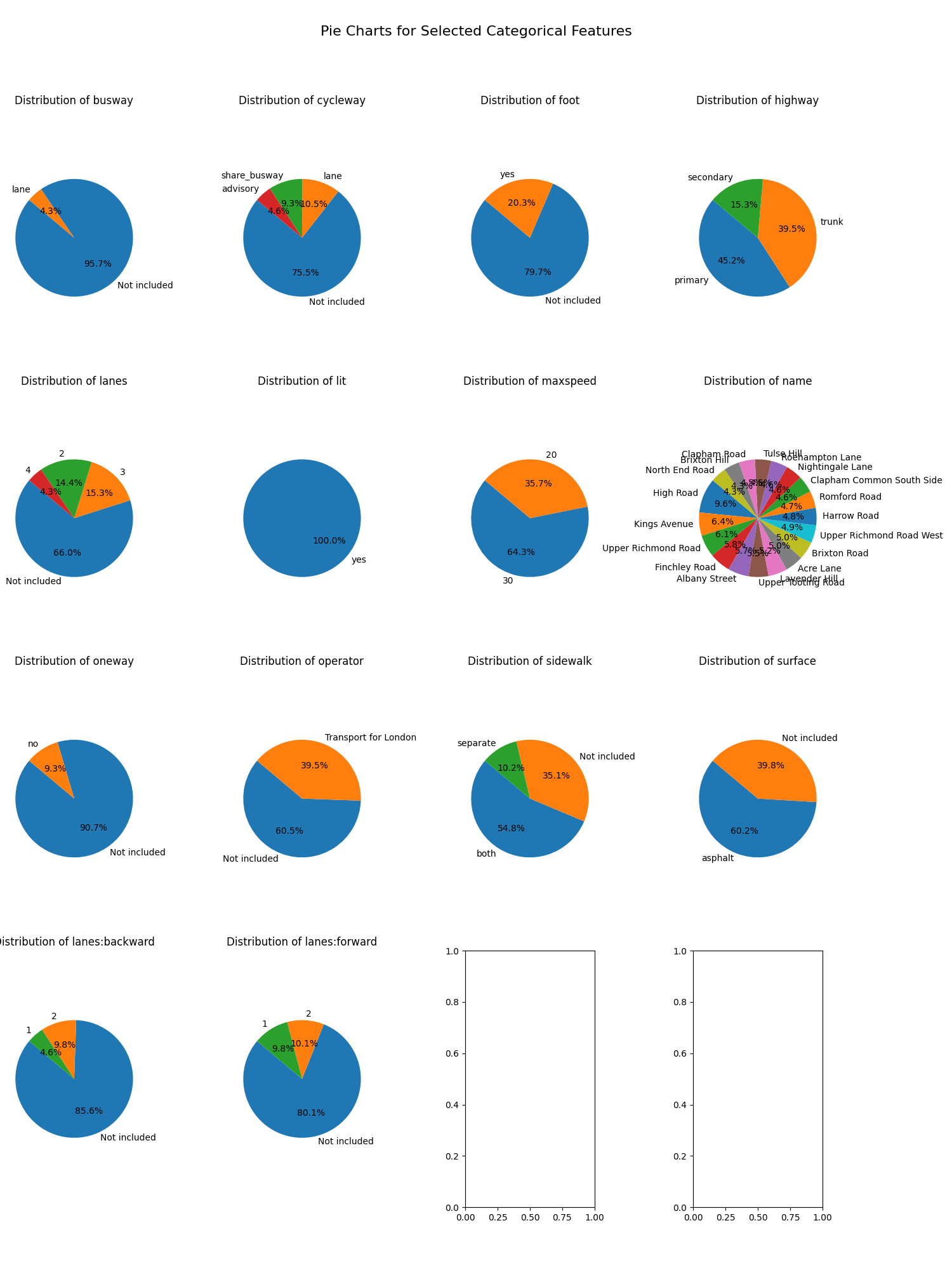
Data train yang telah ditambahkan data eksternal akan dipecah (*split*) dengan persentase 80% untuk latih (*train*) dan 20% untuk data uji (*test*).

1. **Metode Pertama**  
     
    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.
2. **Metode Kedua  
   ** 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.

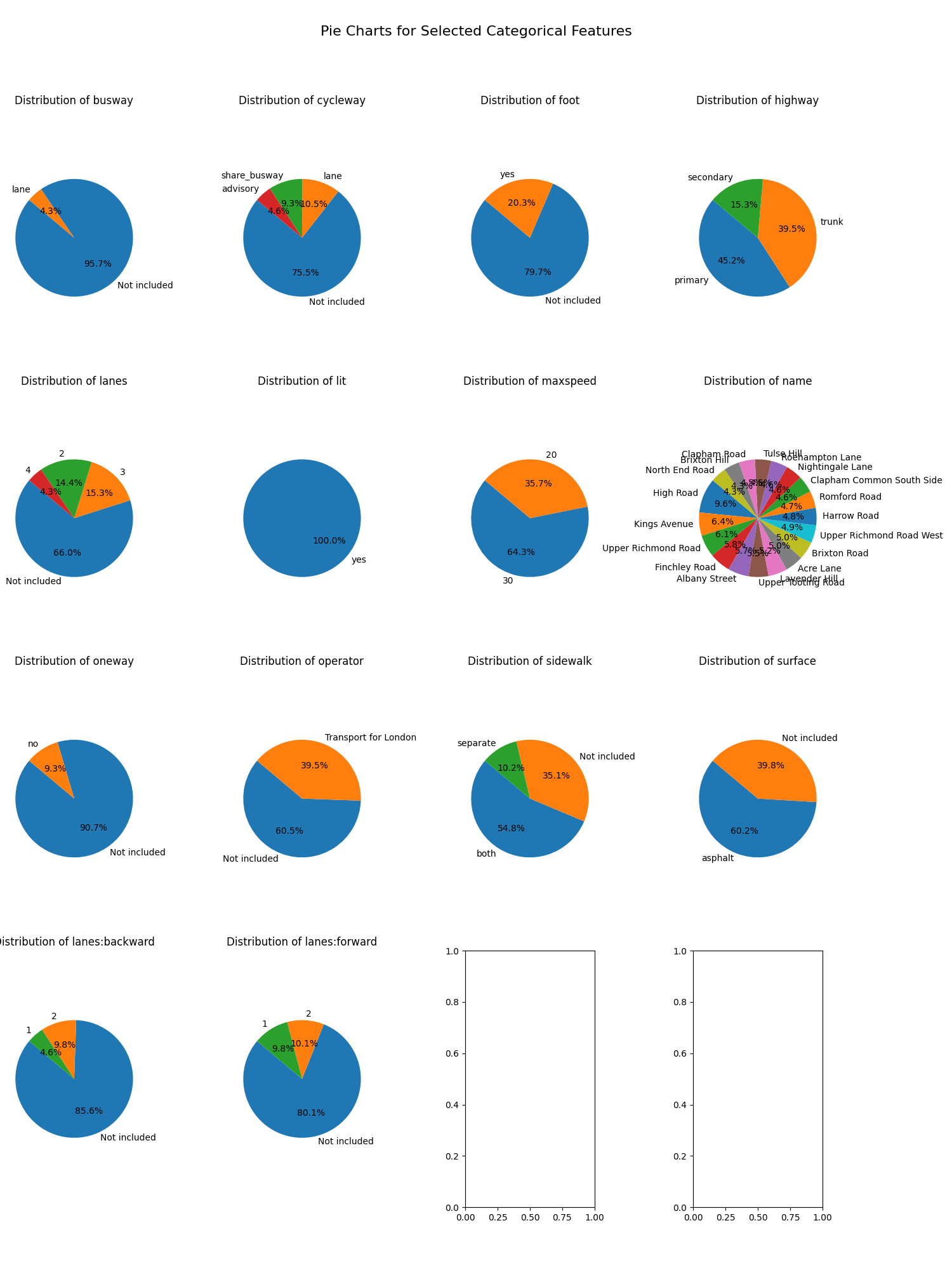
**BAB III: EXPLORATORY DATA ANALYSIS**

1. **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.

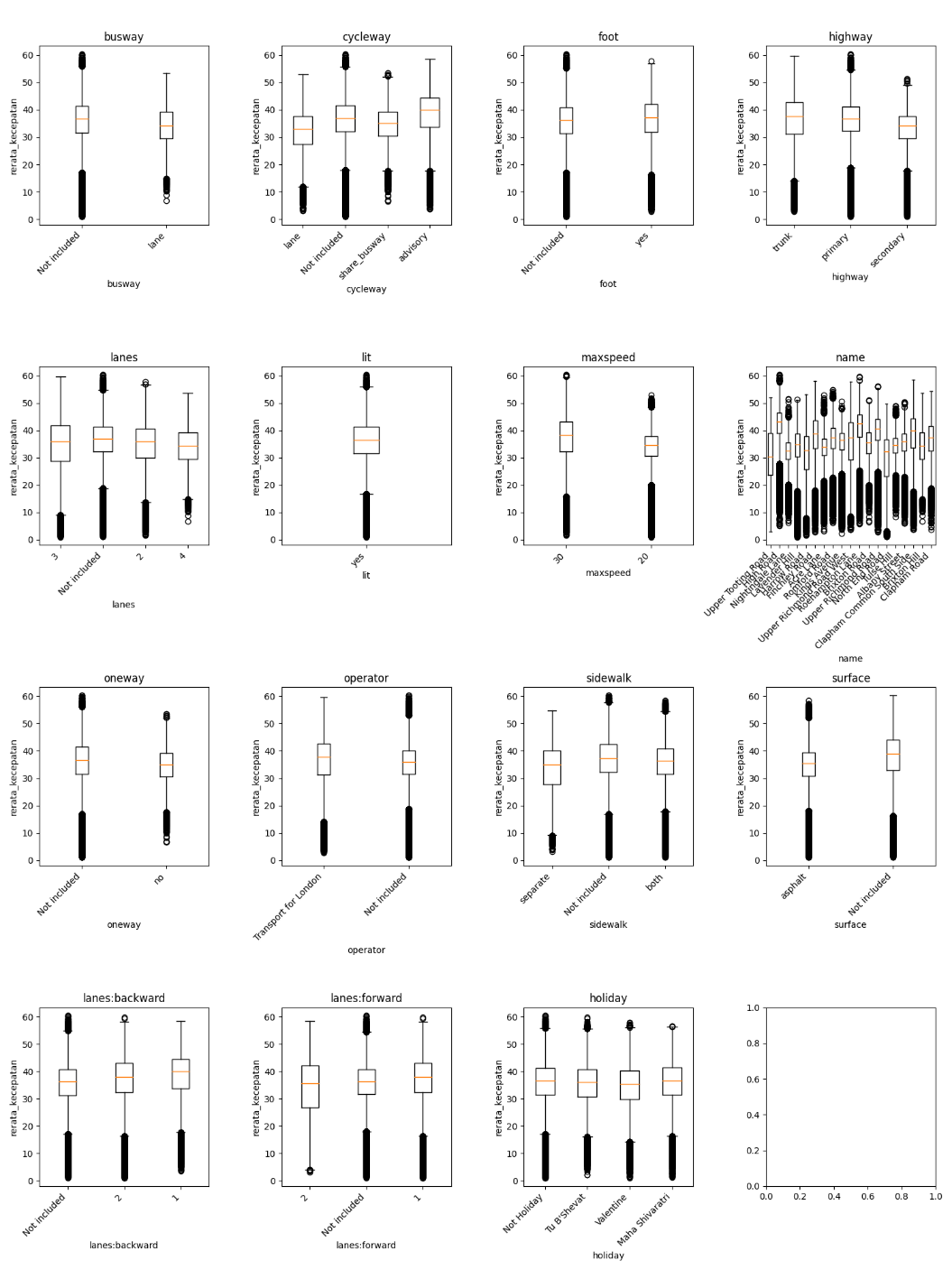
1. **Count Plot Data Kategorik** Untuk data kategorik, kami memutuskan untuk melihat persebarannya terlebih dahulu. Berikut adalah hasilnya :   
   ****

gambar 3.2. Count Plot Data Kategorik

****gambar 3.3. Count Plot Data Kategorik

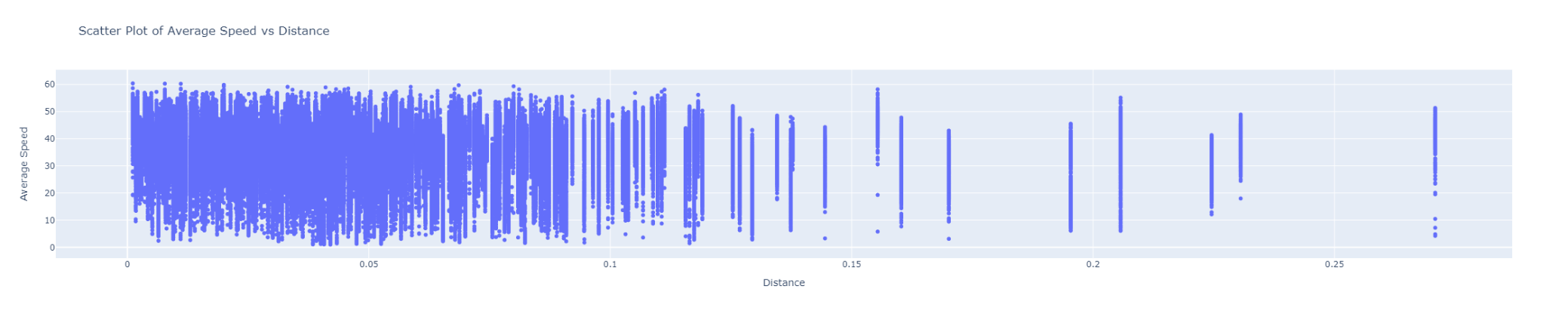
Pada gambar 3.2 dan 3.3, terlihat bahwa banyak fitur kategorikal memiliki nilai yang tak seimbang, bahkan ada yang hanya memiliki satu jenis data saja, yaitu fitur “lit”. Fitur ini tidak memberikan informasi apapun sehingga dapat di-*drop*.

1. **Boxplot data kategorik vs rerata\_kecepatan**

****gambar 3.4. Boxplot Data Kategorik vs Rerata Kecepatan

Gambar 3.4 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.

1. **Distance vs rerata\_kecepatan**



gambar 3.5. Metode pertama yang digunakan

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

**BAB IV: FEATURE ENGINEERING**

1. **Pemisahan Data Tangga**

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'].

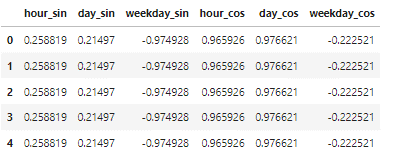
1. **Transformasi Sin Cos untuk Fitur Waktu**

****

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 :

Hasil transformasi sin cos dapat dilihat pada gambar 4.2.



gambar 4.2. Hasil Transformasi Sin Com

1. **Ekstraksi Hari Special**

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

A screen shot of a computer program

Description automatically generated

gambar 4.3. Metode pertama yang digunakan

1. **Mencari fitur distance/jarak dengan haversine**

Di sini 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.

A computer screen shot of a program code

Description automatically generated

gambar 4.4. Kode Metode Haversine

A screenshot of a graph

Description automatically generated

gambar 4.5. Hasil Metode Haversine

1. **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.

1. **Encoding**

A screen shot of a computer screen

Description automatically generated

Di sini kami menggunakan label encoding untuk melakukan encoding fitur-fitur kategorik.

gambar 4.6. Encoding Fitur Kategorik

**BAB V: MODELLING**

1. **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 metriksnya, 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 :

A screenshot of a graph

Description automatically generated

gambar 5.2. Hasil Training

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. 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.

1. **HyperParameter Tuning   
    1. Random Forest**

**A screen shot of a computer program

Description automatically generated** Kami melakukan hyperparameter tuning menggunakan library hyperopt dengan space seperti gambar disamping. Agar kode bisa direproduksi kami menggunakan random\_state=42. Karena RandomForest memiliki waktu training yang sangat amat lama, kami hanya melakukan hyperparameter tuning dengan num\_evals sebanyak 20 buah. Hal ini kami Lakukan demi menghemat waktu. Untuk hasil terbaik dan parameternya dapat dilihat pada gambar 5.3.

gambar 5.3. Hyperparameter Tuning Random Forest

A close-up of a screen

Description automatically generated

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.

1. **XgBRegressor**

****

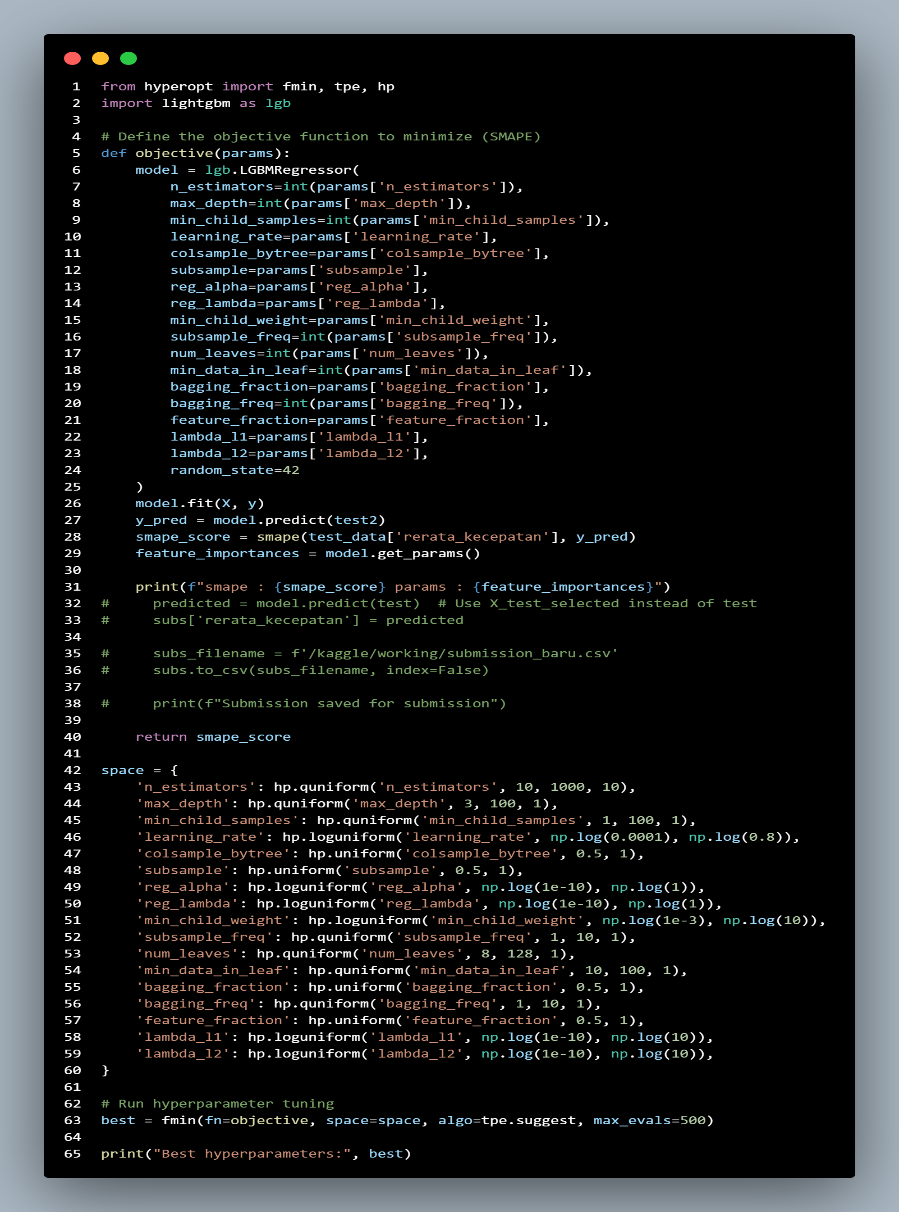
Pada XGBRegressor, kami menggunakan random\_state=42 dan jumlah num\_evals sebanyak 20 kali. Space yang kami pilih seperti pada gambar disamping. Hasil dari tuning ini mendapatkan smape sebesar : 8.39464



Karena Hasil SMAPE yang didaptkan cukup jauh dibandingkan RandomForest dan LightGBM, kami memutuskan untuk tidak menggunakan XGBRegressor untuk metode ensembling.

gambar 5.4. Hyperparameter Tuning XGBRegressor

1. **LightGBM**

****

gambar 5.5. Hyperparameter Tuning LightGBM

Untuk Tuning LightGBM, kami menggunakan random state=42 dan mengunakan tuning parameter seperti di gambar berikut. Kami melakukan loop sebanyak 500 kali dengan evaluasi metrics adalah SMAPE. Pada tuningan ini, kami berhasil mendapatkan model terbaik dengan smape :

A white background with black numbers and letters

Description automatically generated

8.17719 pada data latih(test\_data) dengan metode split pertama dan smape 7.03 menggunakan model split kedua.

A close-up of a computer screen

Description automatically generated

**BAB VI: 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:

|  |  |
| --- | --- |
|  |  |
| gambar 5.6. Feature Importance | |

Terlihat bahwa fitur hour, latitude, longitudinal, day, hour\_sin, hour\_cos, dan weekday memiliki nilai *importance* yang tinggi. Hal tersebut berarti *feature engineering* yang dilakukan berhasil karena dapat memengaruhi hasil pemodelan.

**BAB VII: 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 VIII: 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.