Pre-Processing

Import Dependencies

```
# Libraries for data manipulation and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pickle

# Data preprocessing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Model evaluation and building
from lightgbm import LGBMClassifier
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, mean_squared_error

# Additional libraries
import re
import sys
import warnings
import os

# Configuration
np.set_printoptions(threshold=sys.maxsize)
warnings.filterwarnings("ignore")

# Kaggle specific
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Read the dataset
df = pd.read_csv("/kaggle/input/dataslayer/train.csv/train.csv")
```

```
/kaggle/input/dataslayer/train.csv/train.csv
/kaggle/input/dataslayer/test.csv/test.csv
```

Penyesuaian Nama Variabel

```
df["Vehicle"] = df["Vehicle Class"]
df["Engine"] = df["Engine Size(L)"]
df["Fuel"] = df["Fuel Type"]
df["City"] = df["Fuel Consumption City"]
df["Hwy"] = df["Fuel Consumption Hwy"]
df["Comb"] = df["Fuel Consumption Comb"]
df["Emissions"] = df["CO2 Emissions(g/km)"]

# Membuang variabel dengan nama yang lama
drop = ["Id", "Vehicle Class", "Engine Size(L)", "Fuel Type", "Fuel
Consumption City", "Fuel Consumption Hwy", "Fuel Consumption Comb",
"CO2 Emissions(g/km)"]
df = df.drop(drop, axis=1)
```

Handling Missing Values

```
for column in df.columns:
    unique_values = df[column].unique()
    print(f"Jumlah nilai unik di kolom '{column}':
{len(unique_values)}")
    if len(unique_values)<=50:
        print(f"Nilai unik di kolom '{column}': {unique_values}")
    print()
```

```
Jumlah nilai unik di kolom 'Make': 21
Nilai unik di kolom 'Make': ['MITSU' 'TOYOTI' 'MATSUDA' 'CHEVO' 'DOGE'
'BMV' 'LECUS' 'KIO' 'FOLD'
 'JIPU' 'NIRRAN' 'CADILUXE' 'FOLKSWA' 'BARUSU' 'GONDA' 'LAND CRAWLER'
 'RYUNDAI' 'TOLVO' 'FIAR' 'ASURA' 'LAMBOGI']

Jumlah nilai unik di kolom 'Cylinders': 15
Nilai unik di kolom 'Cylinders': ['4.0' '6.0' '8.0' 'unknown' 'na'
'3.0' '10.0' 'not-recorded'
 'unspecified' '12.0' nan '5.0' 'missing' 'not-available'
'unestablished']

Jumlah nilai unik di kolom 'Transmission': 33
Nilai unik di kolom 'Transmission': ['AV8' 'A5' 'AS6' 'A6' 'M6' 'M5'
'A9' 'AS8' 'AV' 'AM6' 'AS10' 'A8' 'AS9'
 'unestablished' 'AM8' 'AM7' 'AV7' 'A4' 'not-recorded' 'AV6' 'missing'
 'unspecified' 'AV10' 'AS5' 'M7' 'A10' 'na' 'AS7' 'not-available'
 'unknown' nan 'A7' 'AM9']

Jumlah nilai unik di kolom 'Vehicle': 24
Nilai unik di kolom 'Vehicle': ['SUV - SMALL' 'PICKUP TRUCK - SMALL'
'COMPACT' 'VAN - PASSENGER'
 'MID-SIZE' 'SUV - STANDARD' 'STATION WAGON - SMALL' 'FULL-SIZE'
 'TWO-SEATER' 'PICKUP TRUCK - STANDARD' 'SUBCOMPACT' nan
```

```
 'STATION WAGON - MID-SIZE' 'MINICOMPACT' 'MINIVAN' 'not-available'
 'not-recorded' 'SPECIAL PURPOSE VEHICLE' 'missing' 'unestablished'
'na'
 'unknown' 'unspecified' 'VAN - CARGO']

Jumlah nilai unik di kolom 'Engine': 50
Nilai unik di kolom 'Engine': ['1.5' 'not-available' '2.0' 'unknown'
'1.8' '2.4' '3.5' '2.7' '1.6' 'na'
 '1.2' '2.5' '3.2' '5.0' '8.4' '1.4' '3.8' 'unestablished' '6.2' nan
'4.4'
 '3.0' '3.6' '5.7' '5.2' '6.4' '2.3' 'not-recorded' '3.3' '5.3'
'missing'
 '6.5' '1.0' '4.0' '3.7' '1.3' 'unspecified' '4.6' '4.8' '5.6' '2.8'
'6.0'
 '4.2' '4.3' '2.2' '6.6' '5.4' '3.4' '5.8' '6.8']

Jumlah nilai unik di kolom 'Fuel': 13
Nilai unik di kolom 'Fuel': ['X' 'Z' nan 'E' 'missing' 'not-recorded'
'not-available' 'D'
 'unspecified' 'unknown' 'na' 'unestablished' 'N']

Jumlah nilai unik di kolom 'City': 1792

Jumlah nilai unik di kolom 'Hwy': 1235

Jumlah nilai unik di kolom 'Comb': 9939

Jumlah nilai unik di kolom 'Emissions': 414
```

Ada yang unik disini data yang hilang tidak hanya berupa NaN, akan tetapi ada jenis NaN lain yang "bersembunyi".

Disini kita mendapatkan values ini ['unknown', 'na', 'not-available', 'not-recorded', 'missing', 'unspecified', 'unestablished'] yang seharusnya diidentifikasi sebagai NaN dalam jumlah besar.

```
missing_values = ['unknown', 'na', 'not-available', 'not-recorded',
'missing', 'unspecified', 'unestablished']

def count_missing_values(df, missing_values):
    count_dict = {}

    for column in df.columns:
        counts = {value: (df[column] == value).sum() for value in
missing_values}
        count_dict[column] = counts

    counts_df = pd.DataFrame(count_dict)

    nan_counts = df.isna().sum()
```

```
    nan_counts.name = 'NaN'
    counts_df = pd.concat([counts_df, nan_counts.to_frame().T])

    return counts_df

count_missing_values(df, missing_values)
```

```
                Make  Cylinders  Transmission  Vehicle  Engine  Fuel
City  \
unknown            0        403           129      188     474   189
0
na                 0        409           147      220     454   163
0
not-available      0        399           137      189     485   162
402
not-recorded       0        819           253      372     980   330
0
missing            0        441           145      167     462   168
0
unspecified        0        423           133      189     543   214
0
unestablished      0        431           129      187     473   170
0
NaN                0       1198           410      539    1393   546
1568

                 Hwy  Comb  Emissions
unknown            0     0          0
na                 0     0          0
not-available    423   434          0
not-recorded       0     0          0
missing            0     0          0
unspecified        0     0          0
unestablished      0     0          0
NaN             1701  1674          0
```

Mengidentifikasi nilai tersebut sebagai NaN

```
missing_values = ['unknown', 'na', 'not-available', 'not-recorded',
'missing', 'unspecified', 'unestablished']

for column in df.columns:
    df[column] = df[column].replace(missing_values, np.nan)

count_missing_values(df, missing_values)
```

```
                Make  Cylinders  Transmission  Vehicle  Engine  Fuel
City  \
unknown            0          0             0        0       0     0
0
```

```
na                0          0              0        0       0     0
0
not-available     0          0              0        0       0     0
0
not-recorded      0          0              0        0       0     0
0
missing           0          0              0        0       0     0
0
unspecified       0          0              0        0       0     0
0
unestablished     0          0              0        0       0     0
0
NaN               0       4523           1483     2051    5264  1942
1970

                Hwy  Comb  Emissions
unknown           0     0          0
na                0     0          0
not-available     0     0          0
not-recorded      0     0          0
missing           0     0          0
unspecified       0     0          0
unestablished     0     0          0
NaN            2124  2108          0
```

Sekarang semua nilai yang seharusnya menjadi missing values sudah diidentifikasi sebagai NaN

Penyetaraan Satuan

Untuk Variabel City, Hwy, Comb

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54937 entries, 0 to 54936
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Make          54937 non-null  object
 1   Cylinders     50414 non-null  object
 2   Transmission  53454 non-null  object
 3   Vehicle       52886 non-null  object
 4   Engine        49673 non-null  object
 5   Fuel          52995 non-null  object
 6   City          52967 non-null  object
 7   Hwy           52813 non-null  object
 8   Comb          52829 non-null  object
 9   Emissions     54937 non-null  int64 
```

```
dtypes: int64(1), object(9)
memory usage: 4.2+ MB
```

```
def find_units(data):
    units = set()
    for column in data.columns:
        for value in data[column]:
            matches = re.findall(r'\b\d+(\.\d+)?\s*(L\/10km|L\/100km|
km\/L|mpg Imp\.|MPG \(AS\)|liters per 100 km)\b', str(value))
            if matches:
                units.update([match[1] for match in matches])
    return units

found_units = find_units(df[['Hwy', 'City', 'Comb']])

print(found_units)
```

```
{'L/100km', 'km/L', 'L/10km', 'liters per 100 km'}
```

Disini kita menemukan beberapa satuan yang harus dikonversi {'L/10km', 'L/100km', 'litersper100km', 'km/L'} namun ketika dicari manual, kami menemukan faktor satuan konversi lainnya yaitu 'kmperl', 'mpgimp.', 'mpg'

```
# Faktor Konversi
KM_PER_MILE = 1.60934
LITERS_PER_GALLON_IMP = 4.54609
LITERS_PER_GALLON_US = 3.78541

def convert_fuel_consumption(value):
    if pd.isnull(value) or value == '-1' or value == '0':
        return np.nan  # Memastikan tidak terjadi error
    try:
        value = value.lower().replace(' ', '')  # Merubah data menjadi
tulisan nonkapital dan menghapu spasi
        if 'litersper100km' in value:
            return float(value.split('litersper100km')[0])
        elif 'l/100km' in value:
            return float(value.split('l/100km')[0])
        elif 'l/10km' in value:
            return float(value.split('l/10km')[0]) * 10
        elif 'kmperl' in value:
            return 100 / float(value.split('kmperl')[0])
        elif 'mpgimp.' in value:
            return 100 / (float(value.split('mpgimp.')[0]) *
KM_PER_MILE / LITERS_PER_GALLON_IMP)
        elif 'mpg' in value and 'mpgimp.' not in value:
            return 100 / (float(value.split('mpg')[0]) * KM_PER_MILE /
LITERS_PER_GALLON_US)
        else:
            return np.nan  # jika satuan tidak diketahui, maka kita
```

```
ubah menjadi NaN
    except (ValueError, IndexError):
        return np.nan

df['City'] = df['City'].apply(convert_fuel_consumption)
df['Hwy'] = df['Hwy'].apply(convert_fuel_consumption)
df['Comb'] = df['Comb'].apply(convert_fuel_consumption)
```

Jika berhasil mengkonversi semua maka data type yang sebelumnya object akan berubah menjadi float

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54937 entries, 0 to 54936
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Make          54937 non-null  object 
 1   Cylinders     50414 non-null  object 
 2   Transmission  53454 non-null  object 
 3   Vehicle       52886 non-null  object 
 4   Engine        49673 non-null  object 
 5   Fuel          52995 non-null  object 
 6   City          45897 non-null  float64
 7   Hwy           45868 non-null  float64
 8   Comb          45808 non-null  float64
 9   Emissions     54937 non-null  int64  
dtypes: float64(3), int64(1), object(6)
memory usage: 4.2+ MB
```

Disini karena variabel city, hwy, dan comb sudah memiliki data type float yang mengidentifikasikan bahwa proses konversi sudah terjadi pada semua nilai dalam data.

Imputasi Missing Values

Karena sebelumnya terlihat bahwa masih banyak variabel-variabel yang memiliki missing value, jadi disini kita akan melakukan imputasi terhadap variabel-variabel ini. Dengan 3 pendekatan Imputasi:

- Variabel yang saling berhubungan (City, Hwy, Comb)
- Variabel Kategorik
- Variabel Numerik

Adapun disini yang dikategorikan sebagai:

- variabel kategorik adalah ['Transmission','Make', 'Vehicle', 'Fuel']]
- variabel Numerik adalah ['City', 'Hwy', 'Comb']

Imputasi variabel yang saling berhubungan (City, Hwy, Comb)

Merujuk pada pernyataan ini

> Fuel Consumption Comb (Konsumsi Bahan Bakar Gabungan) Kolom ini menampilkan rating konsumsi bahan bakar yang dihitung sebagai campuran 55% berkendara di kota (Fuel Consumption City) dan 45% berkendara di jalan tol (Fuel Consumption Hwy)

imputasi 'City' jika 'Hwy' dan 'Comb' tidak NaN

```
df.loc[df['City'].isna() & df['Hwy'].notna() & df['Comb'].notna(),
'City'] = (df['Comb'] - 0.45 * df['Hwy']) / 0.55
```

imputasi 'Hwy' jika 'City' dan 'Comb' tidak NaN

```
df.loc[df['Hwy'].isna() & df['City'].notna() & df['Comb'].notna(),
'Hwy'] = (df['Comb'] - 0.55 * df['City']) / 0.45
```

imputasi 'Comb' jika 'City' dan 'Hwy' tidak NaN

```
df.loc[df['Comb'].isna() & df['City'].notna() & df['Hwy'].notna(),
'Comb'] = 0.55 * df['City'] + 0.45 * df['Hwy']

df[['City','Hwy','Comb']].isna().sum()
```

```
City    2831
Hwy     2825
Comb    2852
dtype: int64
```

Disini karena masing-masing variabel City, Hwy, Comb masih memiliki missing values, hal ini kemungkinan disebabkan dalam 1 baris mungkin ada 2 yang NaN, atau malah ketiganya. Oleh karena itu dalam proses selanjutnya akan kembali dilakukan imputasi terhadap variabel ini sebagai variabel numerik.

Imputasi Variabel Kategorikal

['Transmission','Make', 'Vehicle', 'Fuel']]

```
df['Fuel'] = df['Fuel'].astype('category')
df['Transmission'] = df['Transmission'].astype('category')
df['Vehicle'] = df['Vehicle'].astype('category')
df['Make'] = df['Make'].astype('category')
df['Engine'] = pd.to_numeric(df['Engine'], errors='coerce')
df['Cylinders'] = pd.to_numeric(df['Cylinders'], errors='coerce')
```

Encoding Variabel Kategorikal sebelum imputasi

```
def encode_categorical_columns(df, categorical_columns):
    encoders = {}
    for col in categorical_columns:
```

```
        if df[col].dtype == 'object':
            encoder = LabelEncoder()
            df[col] = encoder.fit_transform(df[col].astype(str))
            encoders[col] = encoder
    return df, encoders
```

Proses Imputasi Variabel Kategorikal

1. Pilihan Variabel untuk Imputasi:

LightGBM menggunakan kolom-kolom lain dalam categorical_columns sebagai variabel untuk mengimputasi nilai yang hilang dalam setiap kolom yang dipilih untuk diimputasi. Misalnya, jika 'Vehicle' sedang diimputasi, 'Fuel', 'Transmission', dan 'Make' dapat digunakan sebagai variabel untuk mengisi nilai yang hilang dalam 'Vehicle'.

1. Mekanisme Imputasi:

LightGBM membangun model untuk setiap kolom yang ingin diimputasi ('Vehicle', 'Fuel', 'Transmission'). Setelah memilih kolom-kolom lain sebagai variabel, ia menggunakan model LGBMClassifier yang telah disesuaikan dengan variabel-variabel tersebut untuk memprediksi nilai yang hilang dalam kolom yang sedang diimputasi.

Setelah proses imputasi selesai, LightGBM menghasilkan nilai yang diimputasi untuk setiap kolom yang ditentukan. Kolom yang telah diimputasi tersebut menjadi bagian dari dataframe yang baru, dengan nilai-nilai yang sebelumnya kosong sekarang terisi.

1. Iterasi pada Imputasi:

LightGBM akan mengulangi proses yang sama untuk setiap kolom yang hendak diimputasi, membangun model berbeda untuk setiap kolom tersebut dengan menggunakan variabel dari kolom-kolom lainnya yang ada dalam categorical_columns.

Apabila suatu kolom sudah terisi setelah proses imputasi, maka kolom tersebut tidak lagi diikutsertakan dalam proses imputasi untuk kolom-kolom selanjutnya. Kolom yang diimputasi sebelumnya tetap menggunakan data asli tanpa imputasi. Sebagai contoh, jika kolom 'Vehicle' telah diimputasi, nilai-nilainya tidak akan ikut dipertimbangkan saat mengimputasi kolom 'Fuel' atau 'Transmission'. Setiap kolom diimputasi secara terpisah dan independen dari kolom lainnya.

```
def impute_categorical_data_with_lightgbm(df, column_to_impute,
categorical_columns, include_emissions=False):
    # Gabungkan data yang hilang dan yang tidak hilang
    combined_df = pd.concat([df[df[column_to_impute].isna()],
df[~df[column_to_impute].isna()]])

    # Encode categorical columns
    combined_encoded, encoders =
encode_categorical_columns(combined_df.copy(), categorical_columns)

    # Pisahkan kembali data yang hilang dan yang tidak hilang
    df_with_missing_encoded =
```

```
combined_encoded[combined_encoded[column_to_impute].isna()]
    df_without_missing_encoded =
combined_encoded[~combined_encoded[column_to_impute].isna()]

    columns_to_drop = [column_to_impute]
    if not include_emissions and 'Emissions' in df.columns:
        columns_to_drop.append('Emissions')

    X = df_without_missing_encoded.drop(columns=columns_to_drop)
    y = df_without_missing_encoded[column_to_impute]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

    # Membangun dan melatih model LightGBM
    model = LGBMClassifier(
        num_leaves=36,
        learning_rate=0.04434692015762989,
        n_estimators=31,
        min_child_samples=52,
        subsample=0.9212693252334468,
        colsample_bytree= 0.8977274711608758,
        random_state=0
    )


    model.fit(X_train, y_train, categorical_feature=[col for col in
categorical_columns if col != column_to_impute and col in X.columns])

    # Evaluasi model dengan F1-Score
    y_pred = model.predict(X_test)
    f1_result = f1_score(y_test, y_pred, average='macro')

    # Mengimputasi data yang hilang
    imputed_values =
model.predict(df_with_missing_encoded.drop(columns=columns_to_drop))

    # Mengisi nilai yang diimputasi ke dalam dataframe asli
    df.loc[df[column_to_impute].isna(), column_to_impute] =
imputed_values

    return df, f1_result
```

Disini kita mengidentifikasi kolom kategorikal sehingga kita dimasukkan ke fungsi, fungsi bisa langsung mengenali kolom mana yang perlu di encoding.

Disini kita juga perlu mengidentifikasi kolom yang akan di imputasi untuk melakukan prediksi.

```
# Kolom kategorikal dalam dataset
categorical_columns = ['Transmission','Make', 'Vehicle', 'Fuel']
```

```
# Kolom yang akan diimputasi
columns_to_impute = ['Vehicle', 'Fuel', 'Transmission']

# Melakukan imputasi untuk setiap kolom
imputation_results_with_lightgbm = {}
for column in columns_to_impute:
    df, f1_result = impute_categorical_data_with_lightgbm(df, column,
categorical_columns, include_emissions=True)
    imputation_results_with_lightgbm[column] = f1_result

imputation_results_with_lightgbm
```

```
{'Vehicle': 0.7068414144756947,
 'Fuel': 0.7327560206498307,
 'Transmission': 0.6782549642237186}
```

Didapatkan F1 Score untuk masing-masing variabel lumayan tinggi yaitu kurang lebih diatas 70%

```
df[['Transmission','Make', 'Vehicle', 'Fuel']].isna().sum()
```

```
Transmission    0
Make            0
Vehicle         0
Fuel            0
dtype: int64
```

Tidak ada lagi kolom yang kosong dalam variabel kategorik

Imputasi Variabel Numerik

```
for column in df[['Engine', 'Cylinders','City', 'Hwy', 'Comb',
'City']]:
    unique_values = df[column].unique()
    print(f"Jumlah nilai unik di kolom '{column}':
{len(unique_values)}")
    if len(unique_values)<=50:
        print(f"Nilai unik di kolom '{column}': {unique_values}")
    print()
```

```
Jumlah nilai unik di kolom 'Engine': 43
Nilai unik di kolom 'Engine': [1.5 nan 2.  1.8 2.4 3.5 2.7 1.6 1.2 2.5
3.2 5.  8.4 1.4 3.8 6.2 4.4 3.
 3.6 5.7 5.2 6.4 2.3 3.3 5.3 6.5 1.  4.  3.7 1.3 4.6 4.8 5.6 2.8 6.
4.2
 4.3 2.2 6.6 5.4 3.4 5.8 6.8]

Jumlah nilai unik di kolom 'Cylinders': 8
Nilai unik di kolom 'Cylinders': [ 4.  6.  8. nan  3. 10. 12.  5.]
```

```
Jumlah nilai unik di kolom 'City': 5960

Jumlah nilai unik di kolom 'Hwy': 5591

Jumlah nilai unik di kolom 'Comb': 11455

Jumlah nilai unik di kolom 'City': 5960
```

```
def round_to_nearest_valid_cylinder(value):
    valid_values = np.array([3,4, 5,6, 8, 10, 12])
    closest_index = np.abs(valid_values - value).argmin()
    return valid_values[closest_index]
```

Pemilihan untuk memperlakukan kolom "cylinders" sebagai kolom numerik dan membulatkannya ke nilai terdekat seperti 4, 6, 8, 10, atau 12 dan dilakukan regresi dengan LightGBM didasarkan pada karakteristik dari data itu sendiri dan dinilai lebih efisien.

Meskipun kolom "cylinders" sesungguhnya berupa data rasio yang secara teoritis dapat memiliki nilai desimal, dalam konteks kendaraan, jumlah silinder tidak mungkin berupa nilai desimal. Misalnya, mobil biasanya memiliki jumlah silinder bulat seperti 4, 6, 8, atau 12; jumlah silinder tidak mungkin berupa 4,5 atau 7,2 dalam kendaraan produksi.

Mengapa kita membulatkannya? Karena interpretasi nilai silinder dalam kehidupan nyata tidak memungkinkan adanya jumlah silinder yang tidak berupa bilangan bulat, seperti 5,5 atau 7,2. Dalam prakteknya, kendaraan hanya akan memiliki jumlah silinder yang berupa bilangan bulat. Oleh karena itu, membulatkannya ke nilai terdekat seperti 4, 6, 8, 10, atau 12 dianggap lebih tepat dan sesuai dengan karakteristik data yang dihadapi dalam domain kendaraan.

Memperlakukan variabel "cylinders" seperti ini dapat memberikan hasil yang lebih konsisten dan masuk akal dalam konteks dunia nyata, meskipun secara teknis variabel ini bisa berupa data rasio. Dalam kasus ini, pengambilan keputusan didasarkan pada pengetahuan domain khusus kendaraan dan praktik umum dalam penggunaan variabel tersebut.

```
def impute_numeric_data_with_lightgbm(df, column_to_impute,
categorical_columns, include_emissions=False):
    label_encoders = {}

    # Encode categorical variables
    for col in categorical_columns:
        if df[col].dtype == 'object':
            le = LabelEncoder()
            df[col] = le.fit_transform(df[col])
            label_encoders[col] = le

    # Menyaring kolom yang memiliki data hilang
    df_with_missing = df[df[column_to_impute].isna()]
    df_without_missing = df[~df[column_to_impute].isna()]
```

```
    # Tentukan kolom untuk dihapus
    columns_to_drop = [column_to_impute]
    if not include_emissions and 'Emissions' in df.columns:
        columns_to_drop.append('Emissions')

    # Memisahkan data menjadi fitur dan target
    X = df_without_missing.drop(columns=columns_to_drop)
    y = df_without_missing[column_to_impute]

    # Membagi data untuk training dan testing
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

    # Membangun dan melatih model LightGBM
    model = lgb.LGBMRegressor(
        num_leaves=48,
        learning_rate=0.08016715021489174,
        n_estimators=98,
        min_child_samples=92,
        subsample=0.7250831317078859,
        colsample_bytree=0.9002789518022707,
        reg_alpha=0.007965333485909299,
        reg_lambda=0.00644566847407712,
        random_state=0
    )
    model.fit(X_train, y_train)

    # Evaluasi model
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    # Mengimputasi data yang hilang
    imputed_values =
model.predict(df_with_missing.drop(columns=columns_to_drop))

    # Memproses imputasi khusus untuk kolom 'Cylinders' dan 'Gears'
    if column_to_impute == 'Cylinders':
        imputed_values = np.array([round_to_nearest_valid_cylinder(x)
for x in imputed_values])

    # Mengisi nilai yang diimputasi ke dalam dataframe asli
    df.loc[df[column_to_impute].isna(), column_to_impute] =
imputed_values

    # Mengembalikan data kategorikal ke bentuk asli
    for col, le in label_encoders.items():
        df[col] = le.inverse_transform(df[col])

    return df, rmse
```

```python
# Kolom kategorikal yang akan diikutsertakan dalam proses imputasi
categorical_columns = ["Transmission", 'Make', 'Vehicle', 'Fuel']

# Kolom numerik yang akan diimputasi
numeric_columns_to_impute = ['Engine', 'Cylinders', "City", "Hwy"]

# Melakukan imputasi untuk setiap kolom numerik
imputation_results_with_lightgbm = {}
for column in numeric_columns_to_impute:
    df, rmse = impute_numeric_data_with_lightgbm(df, column,
categorical_columns, include_emissions=True)
    imputation_results_with_lightgbm[column] = rmse

# Menghitung ulang 'Comb' sesuai aturan setelah imputasi 'City' dan
'Hwy'
df.loc[df['Comb'].isna() & df['City'].notna() & df['Hwy'].notna(),
'Comb'] = 0.55 * df['City'] + 0.45 * df['Hwy']
#imputation_results_with_lightgbm['Comb'] = np.nan  # Tidak ada RMSE
untuk 'Comb' karena dihitung dari 'City' dan 'Hwy'

# Menampilkan hasil imputasi
imputation_results_with_lightgbm
```

```
{'Engine': 0.18143299954859768,
 'Cylinders': 0.06135224353639902,
 'City': 0.41771425759621955,
 'Hwy': 0.2288179169879608}
```

```python
df[['Engine', 'Cylinders','City', 'Hwy', 'Comb', 'City']].describe()
```

```
             Engine     Cylinders          City           Hwy
Comb  \
count  54937.000000  54937.000000  54937.000000  54937.000000
54937.000000
mean       2.977998      5.292845     12.578604      9.126729
11.023319
std        1.300677      1.599776      3.827754      2.340302
2.904889
min        1.000000      3.000000      4.119417      3.988652
4.380000
25%        2.000000      4.000000      9.999350      7.500000
8.940139
50%        2.500000      4.000000     11.800000      8.598452
10.408314
75%        3.600000      6.000000     15.097896     10.400000
12.890000
max        8.400000     12.000000     30.500000     20.500000
25.316456

               City
count  54937.000000
```

```
mean      12.578604
std        3.827754
min        4.119417
25%        9.999350
50%       11.800000
75%       15.097896
max       30.500000
```

RMSE dari variabel secara keseluruhan cenderung melakukan prediksi yang sangat dekat dengan nilai sebenarnya.

1. Variabel Engine: 0.18143299954859768

Dalam konteks ini, jika RMSE untuk variabel 'Engine' adalah 0.18 dengan rentang nilai 1 hingga 8.4, itu berarti jika kita memiliki sebuah data yang memiliki nilai 'Engine' sebenarnya sebesar 1 liter, model prediktif kita kemungkinan memprediksi nilainya sekitar 1.1 liter.

Sebuah RMSE sekecil itu menunjukkan bahwa model secara keseluruhan cenderung melakukan prediksi yang sangat dekat dengan nilai sebenarnya. Dalam kasus ini, kesalahan prediksi hanya sekitar 0.1 liter, yang mungkin dianggap sebagai kesalahan yang cukup kecil dalam memperkirakan ukuran mesin suatu kendaraan.

1. Variabel Cylinders: 0.06135224353639902

Dalam konteks ini, jika RMSE untuk variabel 'Cylinders' adalah 0.06 dengan rentang nilai 3-12, itu berarti jika kita memiliki sebuah data yang memiliki 'Cylinders' sebenarnya sebesar 3 maka model prediktif kita memprediksi nilainya sekitar 3,06. Namun hal ini dianulir dengan penggunaan fungsi yang membulatkan kenilai terdekat. Jadi nilai yang diimputasi ke nilai terdekat, nilai 3,06 akan dijadikan menjadi 3

1. Variabel City dan Hwy: {0.41771425759621955 dan 0.2288179169879608}

Untuk variabel 'City' dan 'Highway', RMSE masing-masing adalah 0.4177 dan 0.2288. Ini menunjukkan seberapa dekat model memprediksi konsumsi bahan bakar di kota dan jalan raya dengan nilai sebenarnya. Misalnya, jika model memprediksi konsumsi bahan bakar kota sebesar 8 liter per 100 km, nilai sebenarnya kemungkinan akan sekitar 8 ± 0.4177 liter per 100 km, demikian juga untuk konsumsi bahan bakar di jalan raya.

Secara keseluruhan, RMSE yang kecil untuk setiap variabel menunjukkan bahwa model secara umum memiliki kemampuan yang baik dalam memprediksi nilai-nilai ini. Hal ini mengindikasikan bahwa model cenderung memberikan prediksi yang sangat dekat dengan nilai sebenarnya untuk setiap variabel yang diamati. Namun, penting untuk diingat bahwa interpretasi ini didasarkan pada RMSE saja dan tidak memberikan informasi tentang potensi kecenderungan atau pola kesalahan spesifik yang mungkin dimiliki model.

```
 df[['Engine', 'Cylinders','City', 'Hwy', 'Comb',
'City']].isna().sum()
```

```
Engine       0
Cylinders    0
City         0
```

```
Hwy           0
Comb          0
City          0
dtype: int64
```

```
for column in df[['Engine', 'Cylinders','City', 'Hwy', 'Comb',
'City']]:
    unique_values = df[column].unique()
    print(f"Jumlah nilai unik di kolom '{column}':
{len(unique_values)}")
    if len(unique_values)<=50:
        print(f"Nilai unik di kolom '{column}': {unique_values}")
    print()
```

```
Jumlah nilai unik di kolom 'Engine': 4308

Jumlah nilai unik di kolom 'Cylinders': 7
Nilai unik di kolom 'Cylinders': [ 4.  6.  8.  3. 10. 12.  5.]

Jumlah nilai unik di kolom 'City': 8777

Jumlah nilai unik di kolom 'Hwy': 8363

Jumlah nilai unik di kolom 'Comb': 14286

Jumlah nilai unik di kolom 'City': 8777
```

```
df.isna().sum()
```

```
Make             0
Cylinders        0
Transmission     0
Vehicle          0
Engine           0
Fuel             0
City             0
Hwy              0
Comb             0
Emissions        0
dtype: int64
```

Sekarang semua variabel sudah selesai di imputasi

Penambahan Variabel

Efisiensi Mesin yang merupakan rasio dari ukuran mesin yang digunakan dalam liter (yang direpresentasikan oleh variabel 'Engine') dibagi jumlah silinder (yang direpresentasikan oleh variabel 'Cylinders')

https://repository.its.ac.id/56496/1/03211650010012-Master_Thesis.pdf

https://repository.its.ac.id/56496/

"Efisiensi mesin kendaraan merupakan faktor penting dalam emisi CO2 karena efisiensi mesin yang tinggi dapat mengurangi konsumsi bahan bakar, sehingga menghasilkan emisi CO2 yang lebih rendah. Beberapa penelitian menunjukkan bahwa efisiensi mesin dan efisiensi pembakaran mempengaruhi emisi gas buang, termasuk emisi CO2, dari kendaraan bermotor. Sebagai contoh, kendaraan dengan rasio kompresi mesin yang tinggi cenderung lebih efisien dan menghasilkan emisi CO2 yang lebih rendah dibandingkan dengan model mesin sebelumnya. Selain itu, konsumsi bahan bakar merupakan faktor terpenting dalam pengukuran gas buang karena mempengaruhi emisi, dan efisiensi mesin mempengaruhi konsumsi bahan bakar. Oleh karena itu, peningkatan efisiensi mesin kendaraan dapat berkontribusi pada pengurangan emisi CO2 secara signifikan."

Oleh karena itu, diperlukan variabel baru yaitu efisiensi mesin ('Power To Weight').

```
df['Power_to_Weight'] = df['Engine'] / df['Cylinders']
```

Variable Selection

Variabel "Fuel Consumption Comb" (Konsumsi Bahan Bakar Gabungan) sudah merepresentasikan konsumsi bahan bakar kendaraan secara keseluruhan, karena merupakan rating konsumsi bahan bakar yang dihitung sebagai campuran 55% berkendara di kota dan 45% berkendara di jalan tol. Oleh karena itu, variabel ini sudah mencakup konsumsi bahan bakar saat berkendara di dalam kota maupun di jalan tol, sehingga secara implisit sudah merepresentasikan "Fuel Consumption City" (Konsumsi Bahan Bakar Kota) dan "Fuel Consumption Hwy" (Konsumsi Bahan Bakar Jalan To

```
drop = ["Hwy", "City"]
df = df.drop(drop, axis=1)

df.to_csv("Data_Clean.csv")
```

Modelling

Data yang sudah siap kita modelling adalah sebagai berikut 👇

```
df
```

```
          Make  Cylinders Transmission                    Vehicle
Engine  \
0        MITSU        4.0          AV8                SUV - SMALL
1.500000
1       TOYOTI        6.0           A5     PICKUP TRUCK - SMALL
3.913726
2      MATSUDA        4.0          AS6                    COMPACT
2.000000
3        CHEVO        8.0           A6          VAN - PASSENGER
```

```
5.727763
4       TOYOTI        4.0           M6                    COMPACT
1.800000
...        ...        ...          ...                       ...
...
54932    CHEVO        8.0         AS10                SUBCOMPACT
6.200000
54933    CHEVO        6.0           M6                SUBCOMPACT
3.600000
54934     FOLD        6.0          AM7                TWO-SEATER
3.500000
54935    CHEVO        8.0           A8  PICKUP TRUCK - STANDARD
6.200000
54936  RYUNDAI        4.0          AS6                 FULL-SIZE
2.400000

      Fuel       Comb  Emissions  Power_to_Weight
0        X   9.800000        208         0.375000
1        X  11.960000        325         0.652288
2        X   8.894258        170         0.500000
3        X  14.780000        362         0.715970
4        X   8.010000        180         0.450000
...    ...        ...        ...              ...
54932    Z  10.505362        318         0.775000
54933    X  16.323044        303         0.600000
54934    Z  15.630000        410         0.583333
54935    Z  14.520000        466         0.775000
54936    X   8.620126        192         0.600000

[54937 rows x 9 columns]
```

Disini kita memisahkan variabel independen dan variabel dependen

- Variabel Independen:
  - Make
  - Cylinders
  - Transmission
  - Vehicle
  - Fuel
  - Comb
  - Power_to_Weight
- Variabel Dependen:
  - Emission

```
# Memisahkan variabel independen dan dependen
X = df.drop(columns=['Emissions'])
y = df['Emissions']
```

Holdout

Karena disini data kita tidak terlalu besar maka kita akan menggunakan data untuk train sebesar 80% dan test 20%. Merujuk sumber-sumber kredibel:

https://statmodeling.stat.columbia.edu/2016/11/22/30560/

https://towardsdatascience.com/exploring-best-test-size-number-of-folds-and-repeated-hold-out-bbf773f370b6

```
# Membagi data menjadi train dan tes
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Melakukan modelling dengan menggunakan lightgbm

```
# Memastikan dataset yang akan diproses oleh lightgbm.
train_data = lgb.Dataset(X_train, label=y_train)
test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)
train_data2 = lgb.Dataset(X, label=y)

# Parameter Lightgbm yang digunakan dari hasil tunning hyperparameter
params = {
    'objective': 'regression',
    'metric': 'mse',
    'boosting_type': 'gbdt',
    'num_leaves': 383,
    'learning_rate':  0.043473888565665246,
    'feature_fraction': 0.7062548984003787,
    'bagging_fraction': 0.569170227011523,
    'bagging_freq': 3,
    'min_child_samples': 200,
    'force_col_wise': True
}

# Melatih Model LightGBM
num_round = 1000
bst = lgb.train(params, train_data, num_round, valid_sets=[test_data])
bst2 = lgb.train(params, train_data2, num_round)

# Menyimpan model yang dilatih ke dalam file .pkl
pickle.dump(bst2, open('trained_model_full_data.pkl', 'wb'))  #
Menyimpan model bst2

# Melakukan Prediksi Pada Dataset
y_pred = bst.predict(X_test, num_iteration=bst.best_iteration)

# Mengevaluasi model dengan metrik RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

```
print(f'Mean Squared Error on the test set: {mse}')
print(f'Root Mean Squared Error on the test set: {rmse}')
```

```
Mean Squared Error on the test set: 333.0478268089643
Root Mean Squared Error on the test set: 18.249597990338426
```

Submission

```
# Membaca file csv dataset test yang telah disiapkan
test_data_path = "/kaggle/input/test-tentan-prepro/Test_Tentan.csv"  # Adjust the path as needed
df_test = pd.read_csv(test_data_path)

# Memastikan variabel kategorik yang digunakan
categorical_features = ['Make', 'Transmission', 'Vehicle', 'Fuel']
for feature in categorical_features:
    df_test[feature] = df_test[feature].astype('category')

# Menyiapkan variabel yang akan digunakan
features = [col for col in df_test.columns if col not in ('Unnamed: 0', 'CO2 Emissions(g/km)')]

# Melakukan prediksi
y_pred_test = bst2.predict(df_test[features], num_iteration=bst2.best_iteration)

# Membaca sample submission file
sample_submission_path = "/kaggle/input/dataslayer/sample_submission.csv"  # Adjust the path as needed
df_submission = pd.read_csv(sample_submission_path)

# Mengisi nilai pada variabel 'CO2 Emissions(g/km)' menggunakan hasil prediksi yang diberikan model y_pred_test
df_submission['CO2 Emissions(g/km)'] = y_pred_test

# Menyimpan hasil modifikasi submission kedalam file csv baru
updated_submission_path = "Updated_Submission12.csv"
df_submission.to_csv(updated_submission_path, index=False)
print('The predictions have been saved to Updated_Submission12.csv')
```

```
The predictions have been saved to Updated_Submission12.csv
```