Project Data Science Prediksi Harga Mobil

Daftar isi :

1. Menentukan Label Data

Dataset yang digunakan untuk machine learning regresi prediksi, Label/targetnya adalah kolom 'Price" dan Fiturnya adalah kolom selain 'Price'. variabel-variabel yang digunakan pada penelitian ini adalah sebagai berikut:

1. ID
2. Price: price of the car (Target Column)
3. Levy
4. Manufacturer
5. Model
6. Prod. year
7. Category
8. Leather interior
9. Fuel type
10. Engine volume
11. Mileage
12. Cylinders
13. Gear box type
14. Drive wheels
15. Doors
16. Wheel
17. Color
18. Airbags

2. Mengumpulkan Data

Penelitian ini menggunakan data sekunder yang berasal dari data kaggle.com. Data yang digunakan merupakan data harga mobil berdasarkan berbagai atribut dan fitur, terdiri dari 19237 barus dan 18 kolom. Tujuan dari dataset adalah untuk menentukan prediksi harga mobil berdasarkan fitur-fitur yang dimiliki oleh dataset.

Dataset : https://www.kaggle.com/datasets/deepcontractor/car-price-prediction-challenge/data

3. Menelaah Data

Data Visualisasi

Import Library

```
# Step 1: Import relevant libraries--------------------------------------------------------

#Standard libraries for data analysis:----------------------

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm, skew
from scipy import stats
import statsmodels.api as sm


# sklearn modules for data preprocessing-------------------------------------

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


#sklearn modules for Model Selection--------------------------------------

from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble, discriminant_analysis, gaussian_process
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


#sklearn modules for Model Evaluation & Improvement---------------------------

from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import f1_score, precision_score, recall_score, fbeta_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import KFold

from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics

from sklearn.metrics import classification_report, precision_recall_curve
from sklearn.metrics import auc, roc_auc_score, roc_curve
from sklearn.metrics import make_scorer, recall_score, log_loss
from sklearn.metrics import average_precision_score


#Standard libraries for data visualization---------------------

import seaborn as sn
from matplotlib import pyplot
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import matplotlib
%matplotlib inline
color = sn.color_palette()
import matplotlib.ticker as mtick
from IPython.display import display
pd.options.display.max_columns = None
from pandas.plotting import scatter_matrix
from sklearn.metrics import roc_curve


#Miscellaneous Utilitiy Libraries--------------------------------------

import random
import os
import re
import sys
import timeit
import string
import time
from datetime import datetime
from time import time
from dateutil.parser import parse
import joblib
```

import Dataset

```
from google.colab import drive
drive.mount('/content/drive')
file = '/content/drive/MyDrive/Colab Notebooks/car_price_prediction.csv'
dataset = pd.read_csv(file)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
dataset.head()
```

	ID	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Doors	Wheel	Color	Airbags
0	45654403	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005 km	6.0	Automatic	4x4	04-May	Left wheel	Silver	12
1	44731507	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3	192000 km	6.0	Tiptronic	4x4	04-May	Left wheel	Black	8
2	45774419	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000 km	4.0	Variator	Front	04-May	Right-hand drive	Black	2
3	45769185	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966 km	4.0	Automatic	4x4	04-May	Left wheel	White	0
4	45809263	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901 km	4.0	Automatic	Front	04-May	Left wheel	Silver	4

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                19237 non-null  int64  
 1   Price             19237 non-null  int64  
 2   Levy              19237 non-null  object 
 3   Manufacturer      19237 non-null  object 
 4   Model             19237 non-null  object 
 5   Prod. year        19237 non-null  int64  
 6   Category          19237 non-null  object 
 7   Leather interior  19237 non-null  object 
 8   Fuel type         19237 non-null  object 
 9   Engine volume     19237 non-null  object 
 10  Mileage           19237 non-null  object 
 11  Cylinders         19237 non-null  float64
 12  Gear box type     19237 non-null  object 
 13  Drive wheels      19237 non-null  object 
 14  Doors             19237 non-null  object 
 15  Wheel             19237 non-null  object 
 16  Color             19237 non-null  object 
 17  Airbags           19237 non-null  int64  
dtypes: float64(1), int64(4), object(13)
memory usage: 2.6+ MB
```

```
dataset.shape
```

```
(19237, 18)
```

```
dataset.columns
```

```
Index(['ID', 'Price', 'Levy', 'Manufacturer', 'Model', 'Prod. year',
       'Category', 'Leather interior', 'Fuel type', 'Engine volume', 'Mileage',
       'Cylinders', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel', 'Color',
```

```
           'Airbags'],
          dtype='object')
```

```
# deskripsi data
dataset.describe()
```

	ID	Price	Prod. year	Cylinders	Airbags
count	1.923700e+04	1.923700e+04	19237.000000	19237.000000	19237.000000
mean	4.557654e+07	1.855593e+04	2010.912824	4.582991	6.582627
std	9.365914e+05	1.905813e+05	5.668673	1.199933	4.320168
min	2.074688e+07	1.000000e+00	1939.000000	1.000000	0.000000
25%	4.569837e+07	5.331000e+03	2009.000000	4.000000	4.000000
50%	4.577231e+07	1.317200e+04	2012.000000	4.000000	6.000000
75%	4.580204e+07	2.207500e+04	2015.000000	4.000000	12.000000
max	4.581665e+07	2.630750e+07	2020.000000	16.000000	16.000000

```
dataset.dtypes
```

```
ID                    int64
Price                 int64
Levy                 object
Manufacturer         object
Model                object
Prod. year            int64
Category             object
Leather interior     object
Fuel type            object
Engine volume        object
Mileage              object
Cylinders           float64
Gear box type        object
Drive wheels         object
Doors                object
Wheel                object
Color                object
Airbags               int64
dtype: object
```

```
dataset.columns.to_series().groupby(dataset.dtypes).groups
```

```
{int64: ['ID', 'Price', 'Prod. year', 'Airbags'], float64: ['Cylinders'], object: ['Levy', 'Manufacturer', 'Model', 'Category', 'Leather interior', 'Fuel type', 'Engine volume', 'Mileage', 'Gear box type', 'Drive wheels', 'Doors', 'Wheel', 'Color']}
```

```
# Chek Missing values
dataset.isna().any()
```

```
ID                  False
Price               False
Levy                False
Manufacturer        False
Model               False
Prod. year          False
Category            False
Leather interior    False
Fuel type           False
Engine volume       False
Mileage             False
Cylinders           False
Gear box type       False
Drive wheels        False
Doors               False
Wheel               False
Color               False
Airbags             False
dtype: bool
```

```
# check Missing value
dataset.isna().sum()
```

```
ID                  0
Price               0
Levy                0
Manufacturer        0
Model               0
Prod. year          0
Category            0
Leather interior    0
Fuel type           0
Engine volume       0
Mileage             0
Cylinders           0
Gear box type       0
Drive wheels        0
Doors               0
Wheel               0
Color               0
Airbags             0
dtype: int64
```

```
# Check the number of unique
dataset.nunique()
```

```
ID                  18924
Price                2315
Levy                  559
Manufacturer           65
Model                1590
Prod. year             54
Category               11
Leather interior        2
Fuel type               7
Engine volume         107
Mileage              7687
Cylinders              13
Gear box type           4
Drive wheels            3
Doors                   3
Wheel                   2
Color                  16
Airbags                17
dtype: int64
```

4. Memvalidasi Data

```
# Check Duplikasi
dataset.duplicated().sum()
```

```
313
```

Column Levy

```
dataset['Levy'].unique()
```

```
           '1646', '259', '609', '697', '585', '475', '690', '308', '1823',
           '1361', '1273', '924', '584', '2078', '831', '1172', '893', '1872',
           '1885', '1266', '447', '2148', '1730', '730', '289', '502', '333',
           '1325', '247', '879', '1342', '1327', '1598', '1514', '1058',
           '738', '1935', '481', '1522', '1282', '456', '880', '900', '798',
           '1277', '442', '1051', '790', '1292', '1047', '528', '1211',
           '1493', '1793', '574', '930', '1998', '271', '706', '1481', '1677',
           '1661', '1286', '1408', '1090', '595', '1451', '1267', '993',
           '1714', '878', '641', '749', '1511', '603', '353', '877', '1236',
           '1141', '397', '784', '1024', '1357', '1301', '770', '922', '1438',
           '753', '607', '1363', '638', '490', '431', '565', '517', '833',
           '489', '1760', '986', '1841', '1620', '1360', '474', '1099', '978',
           '1624', '1946', '1268', '1307', '696', '649', '666', '2151', '551',
           '800', '971', '1323', '2377', '1845', '1083', '694', '463', '419',
           '345', '1515', '1505', '2056', '1203', '729', '460', '1356', '876',
           '911', '1190', '780', '448', '2410', '1848', '1148', '834', '1275',
           '1028', '1197', '724', '890', '1705', '505', '789', '2959', '518',
           '461', '1719', '2858', '3156', '2225', '2177', '1968', '1888',
           '1308', '2736', '1103', '557', '2195', '843', '1664', '723',
           '4508', '562', '501', '2018', '1076', '1202', '3301', '691',
           '1440', '1869', '1178', '418', '1820', '1413', '488', '1304',
           '363', '2108', '521', '1659', '87', '1411', '1528', '3292', '7058',
```

```
'1195', '1132', '1768', '1384', '1080', '7063', '1817', '1452',
'1975', '1368', '702', '1974', '1781', '1036', '944', '663', '364',
'1539', '1345', '1680', '2209', '741', '1575', '695', '1317',
'294', '1525', '424', '997', '1473', '1552', '2819', '2188',
'1668', '3057', '799', '1502', '2606', '552', '1694', '1759',
'1110', '399', '1470', '1174', '5877', '1474', '1688', '526',
'686', '5908', '1107', '2070', '1468', '1246', '1685', '556',
'1533', '1917', '1346', '732', '692', '579', '421', '362', '3505',
'1855', '2711', '1586', '3739', '681', '1708', '2278', '1701',
'722', '1482', '928', '827', '832', '527', '604', '173', '1341',
'3329', '1553', '859', '167', '916', '828', '2082', '1176', '1108',
'975', '3008', '1516', '2269', '1699', '2073', '1031', '1503',
'2364', '1030', '1442', '5666', '2715', '1437', '2067', '1426',
'2908', '1279', '866', '4283', '279', '2658', '3015', '2004',
'1391', '4736', '748', '1466', '644', '683', '2705', '1297', '731',
'1252', '2216', '3141', '3273', '1518', '1723', '1588', '972',
'682', '1094', '668', '175', '967', '402', '3894', '1960', '1599',
'2000', '2084', '1621', '714', '1109', '3989', '873', '1572',
'1163', '1991', '1716', '1673', '2562', '2874', '965', '462',
'605', '1948', '1736', '3518', '2054', '2467', '1681', '1272',
'1205', '750', '2156', '2566', '115', '524', '3184', '676', '1678',
'612', '328', '955', '1441', '1675', '3965', '2909', '623', '822',
'867', '3025', '1993', '792', '636', '4057', '3743', '2337',
'2570', '2418', '2472', '3910', '1662', '2123', '2628', '3208',
'2080', '3699', '2913', '864', '2505', '870', '7536', '1924',
'1671', '1064', '1836', '1866', '4741', '841', '1369', '5681',
'3112', '1366', '2223', '1198', '1039', '3811', '3571', '1387',
'1171', '1365', '1531', '1590', '11706', '2308', '4860', '1641',
```

Karena pada column 'Levy' terdapat nilai (-) dan tipe datanya 'object' maka dilakukan penggantian untuk (-) menjadi (0) dan tipe datanya menjadi 'int'.

5. Menentukan Objek Data

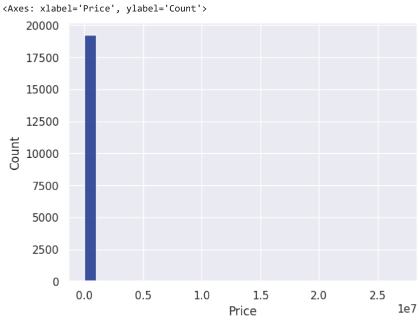
Objek data pada penelitian ini yaitu variabel "Price", sehingga pada bagian ini akan melihat lebih mendalam mengenai variabel tersebut

```
dataset['Price'].describe()
```

```
count    1.923700e+04
mean     1.855593e+04
std      1.905813e+05
min      1.000000e+00
25%      5.331000e+03
50%      1.317200e+04
75%      2.207500e+04
max      2.630750e+07
Name: Price, dtype: float64
```

```
import seaborn as sns
sns.histplot(dataset, x="Price",binwidth = 1000000)
```

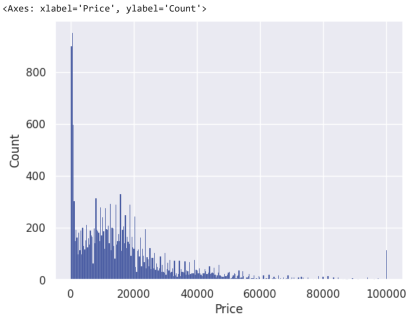
```
<Axes: xlabel='Price', ylabel='Count'>
```




```
# Drop columns 'ID', dan 'Doors' karena tidak dipakai dalam model

dataset=dataset.drop(['ID','Doors'],axis=1)
```

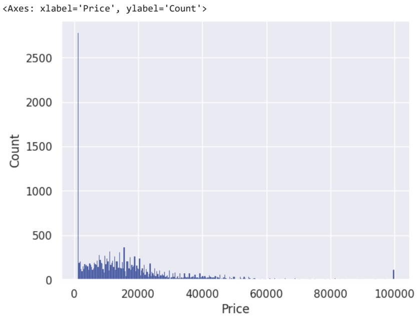
```
# karena plot diatas menghasilkan data yang tidak balance, maka harus dilakukan balance data
# nilai-nilai yang melebihi threshold pada kolom 'Price' dengan nilai threshold itu sendiri (dalam hal ini, 100,000)
# kemudian menampilkan histogram dari kolom 'Price' setelah penggantian menggunakan Seaborn

threshold = 100000
dataset['Price']= np.where(dataset["Price"] > threshold  , threshold , dataset["Price"])
sns.histplot(dataset, x="Price",binwidth =300)
```

```
<Axes: xlabel='Price', ylabel='Count'>
```



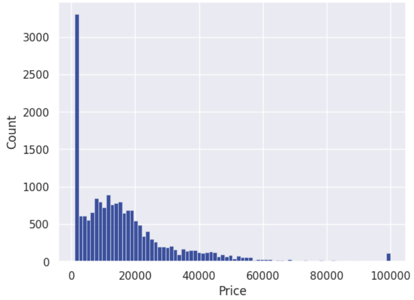

```
# apabila menentukan batas bawahnya juga yang <1000
threshold_low = 1000
threshold_high = 100000
dataset['Price'] = np.where(dataset['Price'] < threshold_low, threshold_low, dataset['Price'])
dataset['Price'] = np.where(dataset['Price'] > threshold_high, threshold_high, dataset['Price'])
sns.histplot(dataset, x="Price", binwidth=300)
```

```
<Axes: xlabel='Price', ylabel='Count'>
```




```
sns.set_theme(palette='dark')
sns.histplot(dataset['Price'])
```

```
<Axes: xlabel='Price', ylabel='Count'>
```




```
dataset.head()
```

	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Wheel	Color	Airbags
0	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005 km	6.0	Automatic	4x4	Left wheel	Silver	12
1	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3	192000 km	6.0	Tiptronic	4x4	Left wheel	Black	8
2	8467	-	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000 km	4.0	Variator	Front	Right-hand drive	Black	2
3	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966 km	4.0	Automatic	4x4	Left wheel	White	0
4	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901 km	4.0	Automatic	Front	Left wheel	Silver	4

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19237 entries, 0 to 19236
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Price             19237 non-null  int64
 1   Levy              19237 non-null  object
 2   Manufacturer      19237 non-null  object
 3   Model             19237 non-null  object
 4   Prod. year        19237 non-null  int64
 5   Category          19237 non-null  object
 6   Leather interior  19237 non-null  object
 7   Fuel type         19237 non-null  object
 8   Engine volume     19237 non-null  object
 9   Mileage           19237 non-null  object
 10  Cylinders         19237 non-null  float64
 11  Gear box type     19237 non-null  object
 12  Drive wheels      19237 non-null  object
 13  Wheel             19237 non-null  object
 14  Color             19237 non-null  object
 15  Airbags           19237 non-null  int64
dtypes: float64(1), int64(3), object(12)
memory usage: 2.3+ MB
```

6. Membersihkan Data

```
# karena ada duplikasi maka 'drop' duplikasi
dataset.drop_duplicates(inplace= True)
```

```
# Chek kembali Duplikasi
dataset.duplicated().sum()
```

```
0
```

```
# kolom levy ada yang missing value, dan diganti dengan 0 (nol)
dataset["Levy"] = np.where(dataset["Levy"] == "-" ,0 , dataset["Levy"]).astype(int)
```

7. Mengkonstruksi Data

```
dataset.head()
```

	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Wheel	Color	Airbags
0	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005 km	6.0	Automatic	4x4	Left wheel	Silver	12
1	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3	192000 km	6.0	Tiptronic	4x4	Left wheel	Black	8
2	8467	0	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000 km	4.0	Variator	Front	Right-hand drive	Black	2
3	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966 km	4.0	Automatic	4x4	Left wheel	White	0
4	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901 km	4.0	Automatic	Front	Left wheel	Silver	4

Column 'Mileage'

```
# menghapus 'km' dan mengubahnya menjadi tipe data 'int'
dataset['Mileage'] = dataset['Mileage'].astype(str).str.replace('km', '')
dataset['Mileage'] = pd.to_numeric(dataset['Mileage'], errors='coerce')
```

```
dataset.Mileage.head()
```

```
0    186005
1    192000
2    200000
3    168966
4     91901
Name: Mileage, dtype: int64
```

Column 'Engine volume'

```
# # menghapus kata 'turbo' dan mengubah tipenya menjadi float
# dataset['Engine volume'] = pd.to_numeric(dataset['Engine volume'].str.replace('Turbo', ''), errors='coerce')
```

```
dataset['Turbo'] = dataset['Engine volume'].str.contains('Turbo', case=False, na=False)
dataset['Engine volume']=dataset['Engine volume'].str.replace('Turbo','')
dataset["Engine volume"]=dataset["Engine volume"].astype('float64')
```

```
dataset['Engine volume'].unique()
```

```
array([ 3.5,  3. ,  1.3,  2.5,  2. ,  1.8,  2.4,  4. ,  1.6,  3.3,  2.2,
        4.7,  1.5,  4.4,  1.4,  3.6,  2.3,  5.5,  2.8,  3.2,  3.8,  4.6,
        1.2,  5. ,  1.7,  2.9,  0.5,  1.9,  2.7,  4.8,  5.3,  0.4,  1.1,
        2.1,  0.7,  5.4,  3.7,  1. ,  2.6,  0.8,  0.2,  5.7,  6.7,  6.2,
        3.4,  6.3,  4.3,  4.2,  0. , 20. ,  0.3,  5.9,  5.6,  6. ,  0.6,
        6.8,  4.5,  7.3,  0.1,  3.1,  6.4,  3.9,  0.9,  5.2,  5.8])
```

```
dataset["Turbo"]=dataset["Turbo"].astype('float64')
```

```
dataset["Turbo"].unique()
```

```
array([0., 1.])
```

```
dataset[(dataset["Engine volume"]==0) | (dataset["Engine volume"]>10)]
```

	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Wheel	Color	Airbags	Turbo
2010	53941	87	TESLA	Model X	2018	Sedan	Yes	Petrol	0.0	81907	6.0	Automatic	4x4	Left wheel	Silver	12	0.0
2357	10036	5603	HYUNDAI	Sonata	2014	Sedan	Yes	LPG	20.0	333686	4.0	Automatic	Front	Left wheel	Silver	4	0.0
3105	2430	87	MERCEDES-BENZ	C 250	2013	Coupe	Yes	Petrol	0.0	121600	4.0	Automatic	Rear	Left wheel	White	12	0.0
3516	27356	87	HYUNDAI	Elantra	2016	Sedan	Yes	LPG	0.0	65004	4.0	Automatic	Front	Left wheel	White	4	0.0
4814	17663	87	TOYOTA	Aqua	2012	Hatchback	Yes	Petrol	0.0	118000	4.0	Automatic	Front	Left wheel	Grey	4	0.0
7685	47076	87	SSANGYONG	REXTON	2016	Jeep	Yes	Diesel	0.0	73968	4.0	Automatic	Front	Left wheel	Black	4	0.0
10603	12231	87	TOYOTA	Prius	2010	Hatchback	No	Hybrid	0.0	0	4.0	Automatic	Front	Left wheel	Golden	0	0.0
12917	1000	87	MERCEDES-BENZ	E 350	2016	Sedan	Yes	Petrol	0.0	33600	6.0	Automatic	Rear	Left wheel	White	12	0.0
14642	1000	87	PORSCHE	Panamera	2011	Sedan	Yes	Petrol	0.0	196800	6.0	Automatic	Rear	Left wheel	Black	12	0.0
17375	1000	87	MERCEDES-BENZ	CLS 550	2014	Sedan	Yes	Petrol	0.0	92800	8.0	Automatic	Rear	Left wheel	Black	12	0.0

```
dataset.drop_duplicates(inplace=True)
```

```
engine_update = {
    2010: 0,
    2357: 2.0,
    3105: 1.8,
    3516: 1.8,
    4814: 1.5,
    7685: 2.0,
    10603: 1.8,
    12917: 3.5,
    14642: 3.6,
    17375: 4.7}
```

```
for baris, nilai_engine_baru in engine_update.items():
    dataset.loc[baris, 'Engine volume'] = nilai_engine_baru
```

```
dataset.head()
```

	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Wheel	Color	Airbags	Turbo
0	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Hybrid	3.5	186005	6.0	Automatic	4x4	Left wheel	Silver	12	0.0
1	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3.0	192000	6.0	Tiptronic	4x4	Left wheel	Black	8	0.0
2	8467	0	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000	4.0	Variator	Front	Right-hand drive	Black	2	0.0
3	3607	862	FORD	Escape	2011	Jeep	Yes	Hybrid	2.5	168966	4.0	Automatic	4x4	Left wheel	White	0	0.0
4	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901	4.0	Automatic	Front	Left wheel	Silver	4	0.0

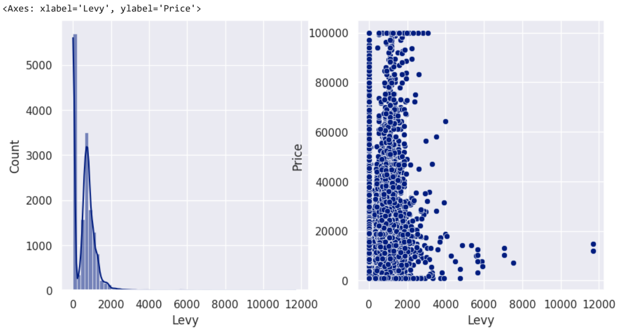
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15704 entries, 0 to 19236
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Price             15704 non-null  int64  
 1   Levy              15704 non-null  int64  
 2   Manufacturer      15704 non-null  object 
 3   Model             15704 non-null  object 
 4   Prod. year        15704 non-null  int64  
 5   Category          15704 non-null  object 
 6   Leather interior  15704 non-null  object 
 7   Fuel type         15704 non-null  object 
 8   Engine volume     15704 non-null  float64
 9   Mileage           15704 non-null  int64  
 10  Cylinders         15704 non-null  float64
 11  Gear box type     15704 non-null  object 
 12  Drive wheels      15704 non-null  object 
 13  Wheel             15704 non-null  object 
 14  Color             15704 non-null  object 
 15  Airbags           15704 non-null  int64  
 16  Turbo             15704 non-null  float64
dtypes: float64(3), int64(5), object(9)
memory usage: 2.7+ MB
```

Visualisasi setiap column

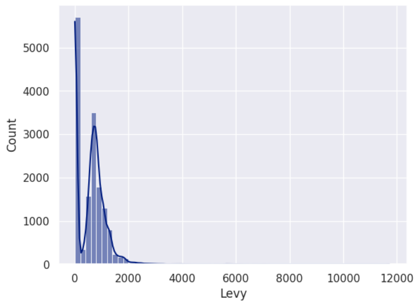
```
# Plot Column Levy

fig, ax = plt.subplots(1,2, figsize = (10,5))
sns.histplot(dataset, ax=ax[0],x="Levy",binwidth = 200, kde=True)
sns.scatterplot(dataset,ax=ax[1], x='Levy', y='Price')
```

```
<Axes: xlabel='Levy', ylabel='Price'>
```

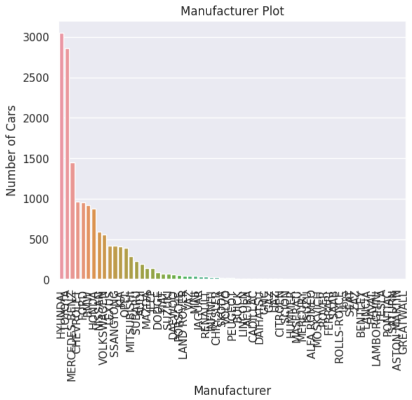



```
sns.histplot(dataset,x="Levy",binwidth = 200, kde=True)
```

```
<Axes: xlabel='Levy', ylabel='Count'>
```

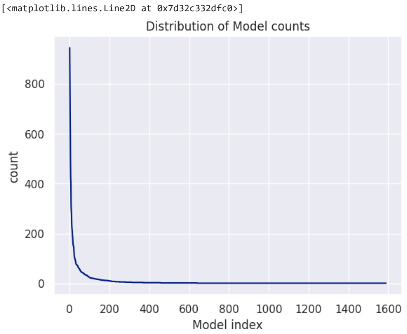



```
# Plot Column Manufacturer

Manufacturer_plot = dataset.Manufacturer.value_counts()


plt.title("Manufacturer Plot")
a=sns.barplot(x = Manufacturer_plot.index,y= Manufacturer_plot);
a.set_xticklabels(Manufacturer_plot.index ,rotation=90)
a.set(xlabel='Manufacturer', ylabel='Number of Cars')
plt.show()
```

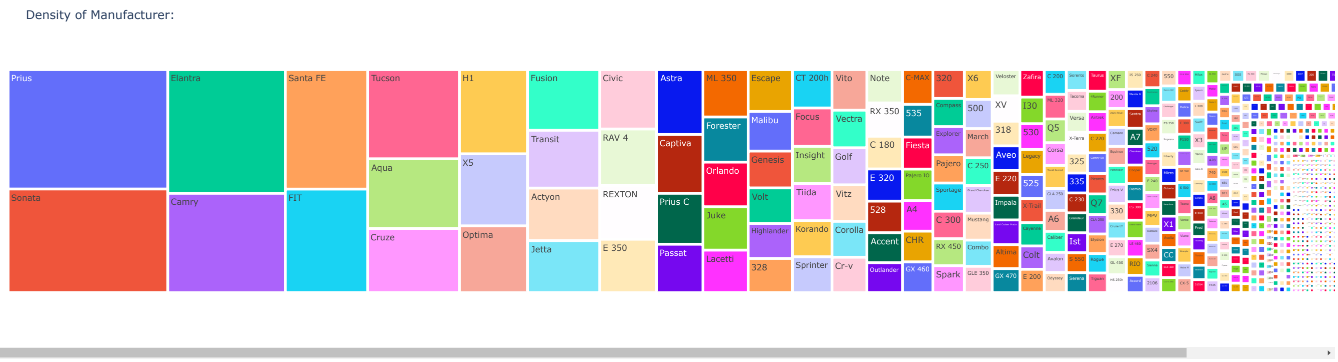



```
# Plot Column Model

Models_counts = dataset["Model"].value_counts()
inf = Models_counts.values
plt.title('Distribution of Model counts')
plt.xlabel('Model index')
plt.ylabel('count')
plt.plot(inf)
```

```
[<matplotlib.lines.Line2D at 0x7d32c332dfc0>]
```

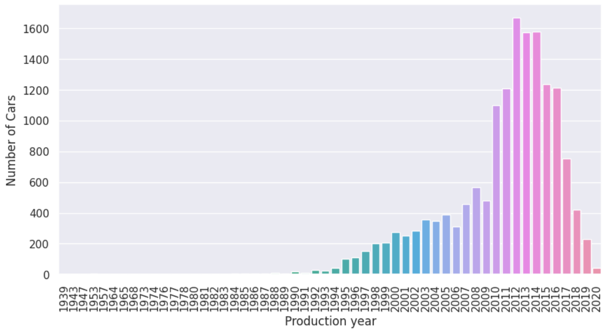



```
import plotly.express as px
px.treemap(dataset,path=["Model"],title="Density of Manufacturer:")
```



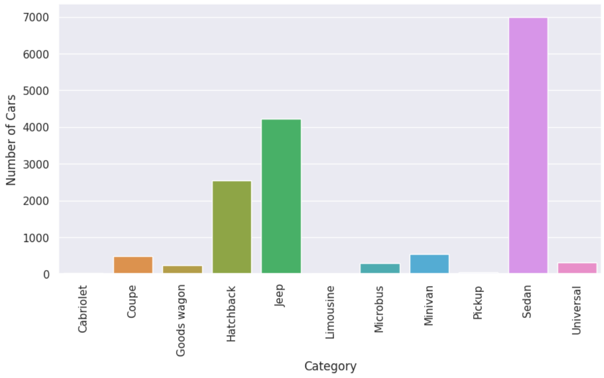

```
# Plot Column Prod. year

plt.figure(figsize=(10,5))
yearly_production =  dataset.groupby(['Prod. year']).size().reset_index().rename(columns = {0:'Counts'})
a1=sns.barplot(x = 'Prod. year', y = 'Counts',data = yearly_production, width = 0.8 );
a1.set_xticklabels(yearly_production['Prod. year'],rotation=90)
a1.set(xlabel='Production year', ylabel='Number of Cars')
```

```
[Text(0.5, 0, 'Production year'), Text(0, 0.5, 'Number of Cars')]
```



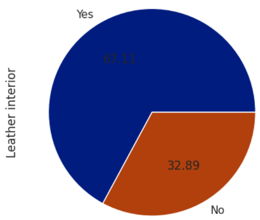

```
# Plot Column Category

plt.figure(figsize=(10,5))
Category_product =  dataset.groupby(['Category']).size().reset_index().rename(columns = {0:'Counts'})
a2=sns.barplot(x = 'Category', y = 'Counts',data = Category_product, width = 0.8 );
a2.set_xticklabels(Category_product['Category'],rotation=90)
a2.set(xlabel='Category', ylabel='Number of Cars')
```

```
[Text(0.5, 0, 'Category'), Text(0, 0.5, 'Number of Cars')]
```



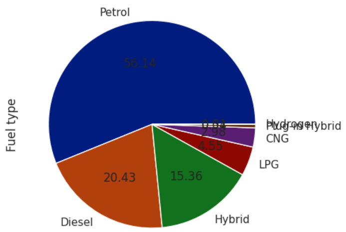

```
# Plot Column Leather interior

dataset["Leather interior"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Leather interior'>
```



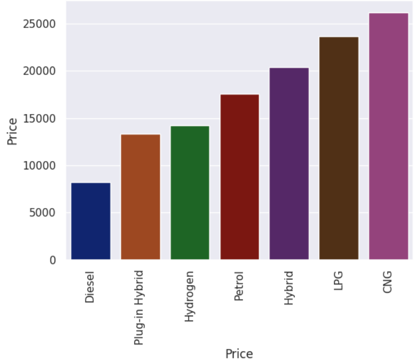

```
# Plot Column Fuel type

dataset["Fuel type"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Fuel type'>
```




```
Fuel = dataset.groupby(['Fuel type'])['Price'].mean()
Fuel = Fuel.sort_values(ascending=False)
a2=sns.barplot(x = Fuel.index, y = Fuel,data = Fuel, width = 0.8 )
a2.set_xticklabels(Fuel.index,rotation=90)
```

```
[Text(0, 0, 'Diesel'),
 Text(1, 0, 'Plug-in Hybrid'),
 Text(2, 0, 'Hydrogen'),
 Text(3, 0, 'Petrol'),
 Text(4, 0, 'Hybrid'),
 Text(5, 0, 'LPG'),
 Text(6, 0, 'CNG')]
```




```
dataset['Electric source'] = dataset['Fuel type'].str.contains('Hybrid', case=False, na=False)
```

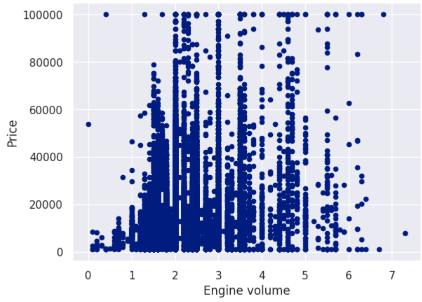
```
dataset['Fuel type'] = dataset['Fuel type'].replace(to_replace=r'.*Hybrid.*', value='Petrol', regex=True)
```

```
dataset["Electric source"]=dataset["Electric source"].astype('float64')
```

```
# Plot Column Engine volume
```

```
dataset.plot(kind = 'scatter', x = 'Engine volume', y= 'Price')
```

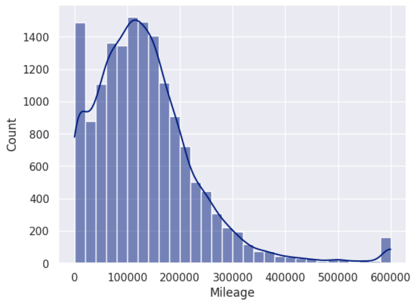
```
<Axes: xlabel='Engine volume', ylabel='Price'>
```




```
# Plot Column Mileage
```

```
dataset["Mileage"] = np.where(dataset["Mileage"] > 0.6e6 , 0.6e6 , dataset["Mileage"])
sns.histplot(dataset,x="Mileage",binwidth = 20000, kde=True)
```

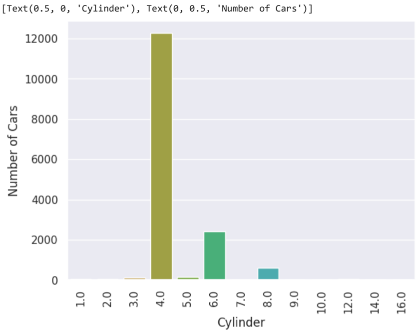
```
<Axes: xlabel='Mileage', ylabel='Count'>
```




```
# Plot Column Cylinder
```

```
Cylinder =  dataset.groupby(['Cylinders']).size().reset_index().rename(columns = {0:'Counts'})
a1=sns.barplot(x= Cylinder.Cylinders, y= Cylinder.Counts);
a1.set_xticklabels(Cylinder.Cylinders ,rotation=90)
a1.set(xlabel='Cylinder', ylabel='Number of Cars')
```

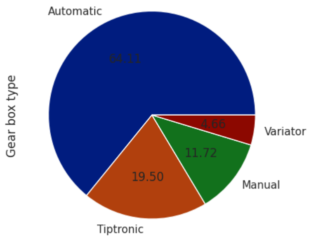
```
[Text(0.5, 0, 'Cylinder'), Text(0, 0.5, 'Number of Cars')]
```




```
# Plot Column Gear box type
```

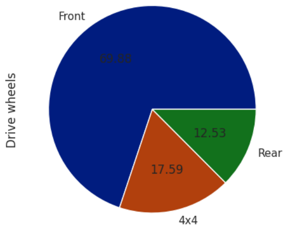
```
dataset["Gear box type"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Gear box type'>
```



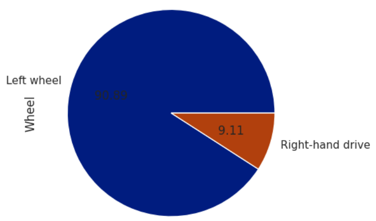

```
# Plot Column Drive wheels

dataset["Drive wheels"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Drive wheels'>
```



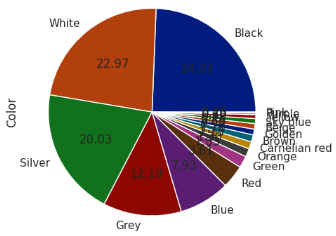

```
# Plot Column Wheel

dataset["Wheel"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Wheel'>
```



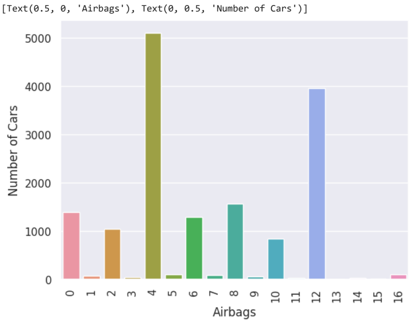

```
# Plot Column Color

dataset["Color"].value_counts().plot(kind = "pie" , autopct = "%.2f")
```

```
<Axes: ylabel='Color'>
```



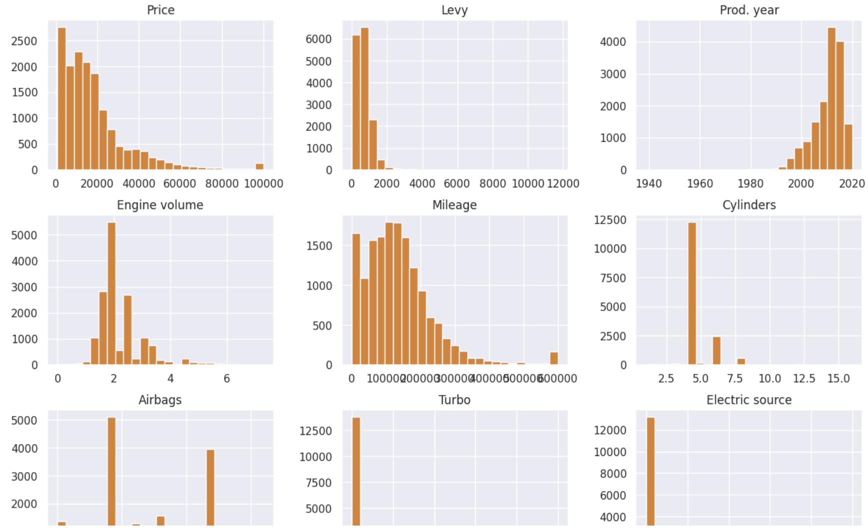

```
# Plot Column Airbags

Airbag = dataset.groupby(['Airbags']).size().reset_index().rename(columns = {0:'Counts'})
a1=sns.barplot(x = Airbag.Airbags, y= Airbag.Counts);
a1.set_xticklabels(Airbag['Airbags'] ,rotation=90)
a1.set(xlabel='Airbags', ylabel='Number of Cars')
```

```
[Text(0.5, 0, 'Airbags'), Text(0, 0.5, 'Number of Cars')]
```

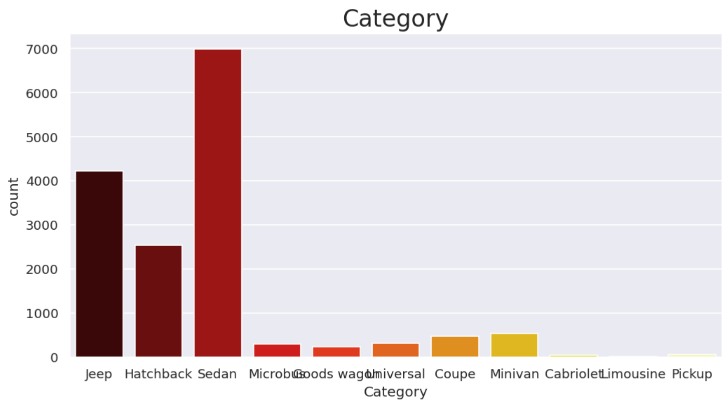


Visualisasi pada seluruh data

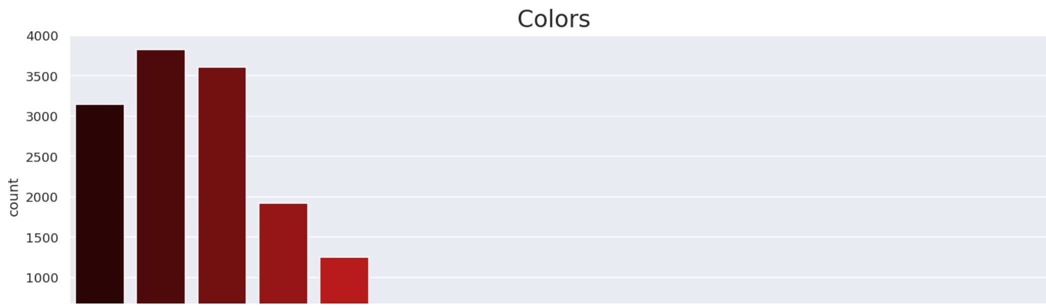
```
dataset.hist(bins=25,figsize=(15,10),color='peru')
plt.show()
```



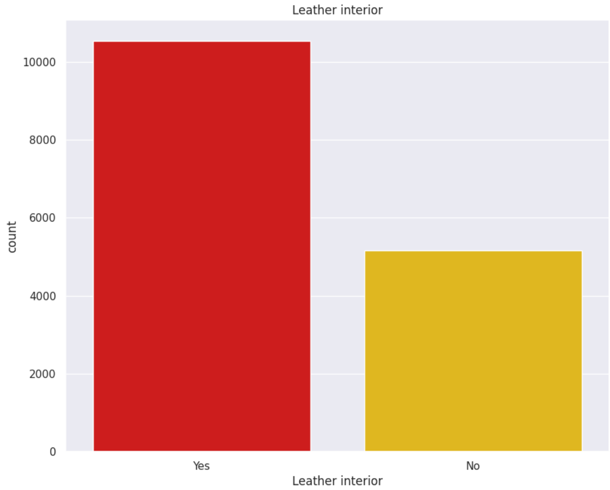

```
plt.subplots(figsize=(10,5), dpi=120)
sns.countplot(data= dataset, x='Category',palette='hot')
plt.title("Category",fontsize=20)
plt.show()
```

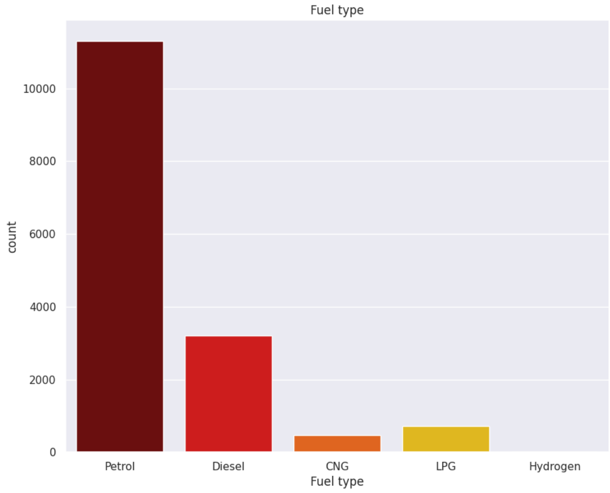


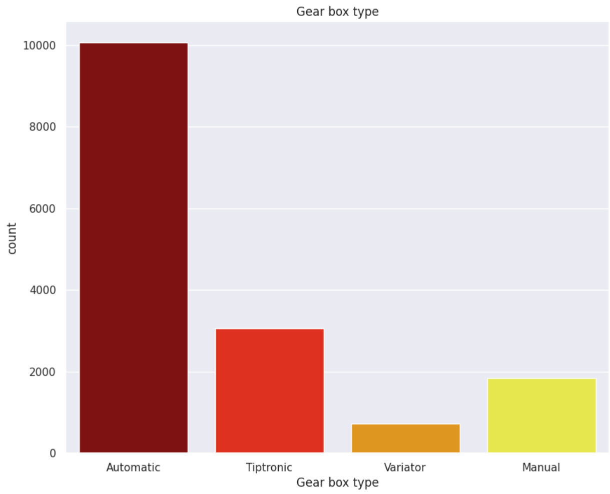

```
plt.subplots(figsize=(15,5), dpi=120)
sns.countplot(data= dataset, x='Color',palette='hot')
plt.title("Colors ",fontsize=20)
plt.show()
```

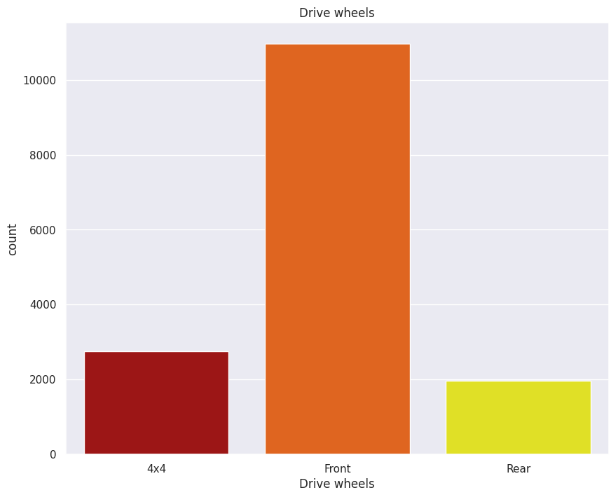


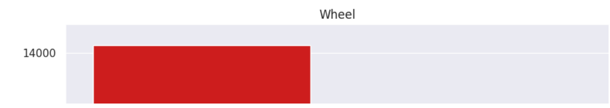

```
columns =['Leather interior','Fuel type','Gear box type','Drive wheels','Wheel']
for col in columns:
    plt.figure(figsize=(10,8))
    #top10 = data[col].value_counts()[:10]
    sns.countplot(data=dataset,x=col,palette='hot')
    plt.title(col)
    plt.show()
```

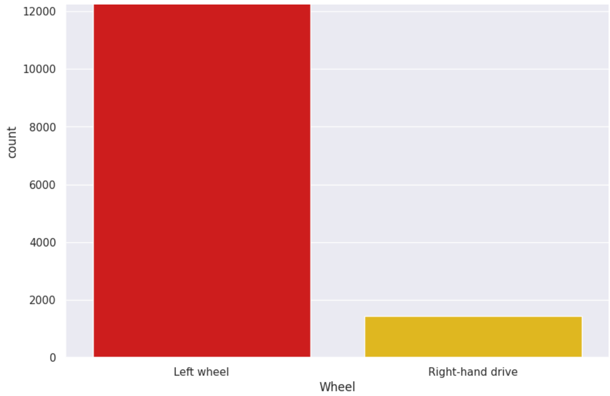

Leather interior
count
0
2000
4000
6000
8000
10000
Yes
No
Leather interior


Fuel type
count
0
2000
4000
6000
8000
10000
Petrol
Diesel
CNG
LPG
Hydrogen
Fuel type


Gear box type
count
0
2000
4000
6000
8000
10000
Automatic
Tiptronic
Variator
Manual
Gear box type


Drive wheels
count
0
2000
4000
6000
8000
10000
4x4
Front
Rear
Drive wheels

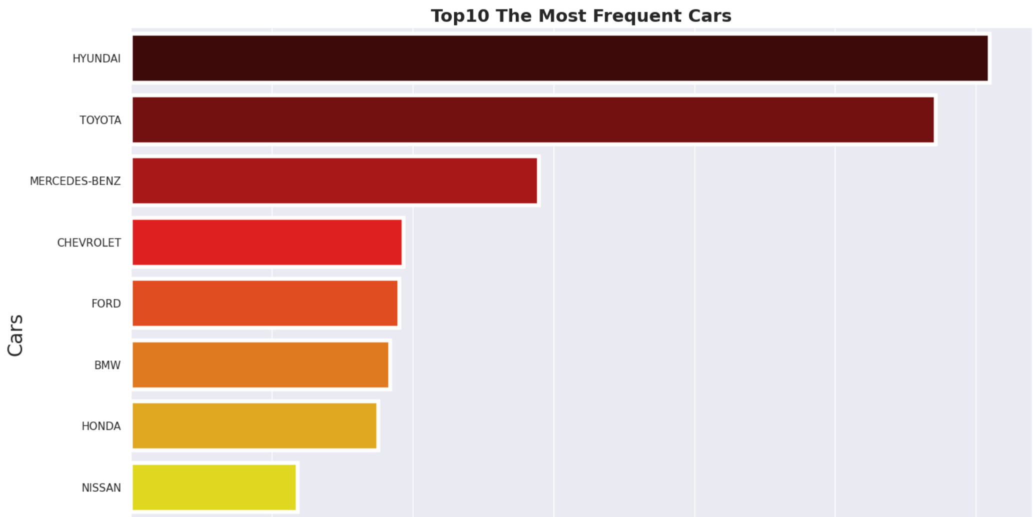

Wheel
14000



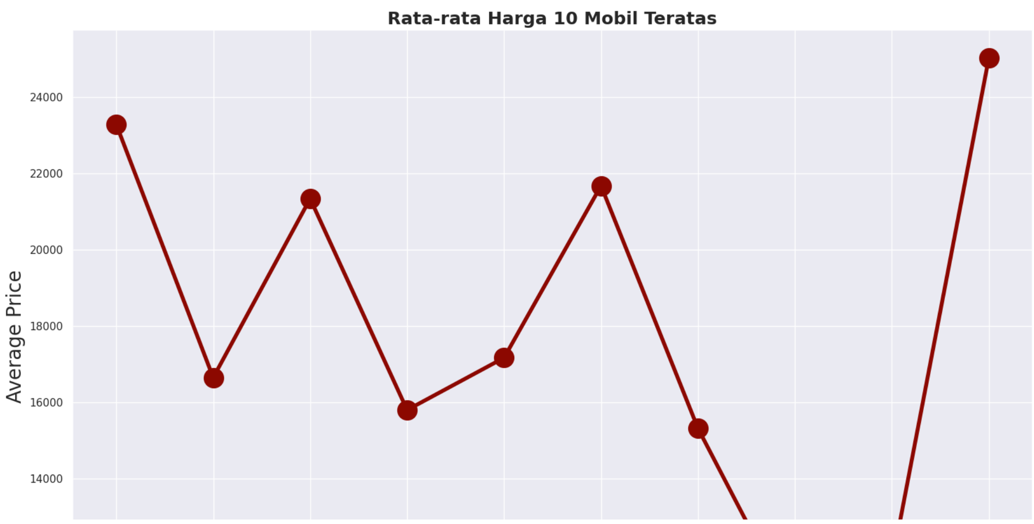

```
top_10_cars = dataset.Manufacturer.value_counts().sort_values(ascending=False)[:10]
top_10_cars
```

```
    HYUNDAI          3048
    TOYOTA           2857
    MERCEDES-BENZ    1450
    CHEVROLET         968
    FORD              953
    BMW               922
    HONDA             879
    NISSAN            594
    VOLKSWAGEN        556
    LEXUS             419
    Name: Manufacturer, dtype: int64
```

```
plt.figure(figsize=(15, 10))
sns.barplot(x=top_10_cars, y=top_10_cars.index,palette='hot',linewidth = 4)
plt.title('Top10 The Most Frequent Cars',loc='center',fontweight='bold',fontsize=18)
plt.xlabel('Frequency',fontsize=20)
plt.ylabel('Cars',fontsize=20)
plt.tight_layout()
plt.show()
```




```
# Rata-rata 10 Mobil teratas
top_10_cars_means_prices = [dataset[dataset['Manufacturer']==i]['Price'].mean() for i in list(top_10_cars.index)]


plt.figure(figsize=(15,10))
plt.plot(top_10_cars.index,top_10_cars_means_prices,color='r',
         linewidth = 4,marker='o',markersize = 20)
plt.title('Rata-rata Harga 10 Mobil Teratas',loc='center',fontweight='bold',fontsize=18)
plt.ylabel('Average Price',fontsize=20)
plt.xlabel('Cars',fontsize=20)
plt.tight_layout()
plt.show()
```



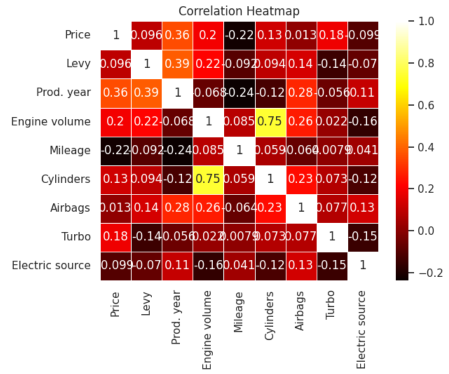

```
plt.figure(figsize=(10, 8), dpi=120)
sns.scatterplot(data=dataset, x='Price', y='Levy', hue="Color", palette="hsv_r")
```

```
<Axes: xlabel='Price', ylabel='Levy'>
```


```
# melihat korelasi
cor= dataset.select_dtypes(exclude=object).corr()
cor
```

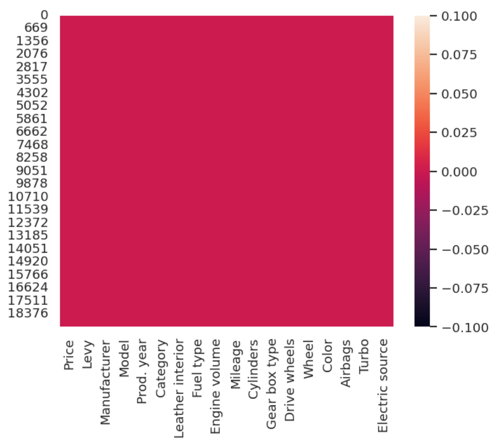
	Price	Levy	Prod. year	Engine volume	Mileage	Cylinders	Airbags	Turbo	Electric source
Price	1.000000	0.096135	0.355969	0.195559	-0.218897	0.127837	0.013024	0.183840	-0.098563
Levy	0.096135	1.000000	0.385373	0.219607	-0.091860	0.093648	0.138737	-0.142493	-0.069741
Prod. year	0.355969	0.385373	1.000000	-0.067598	-0.236744	-0.124423	0.283594	-0.056176	0.109517
Engine volume	0.195559	0.219607	-0.067598	1.000000	0.085065	0.746722	0.264782	0.021863	-0.162708
Mileage	-0.218897	-0.091860	-0.236744	0.085065	1.000000	0.058565	-0.063964	0.007876	0.040801
Cylinders	0.127837	0.093648	-0.124423	0.746722	0.058565	1.000000	0.233105	0.073376	-0.119528
Airbags	0.013024	0.138737	0.283594	0.264782	-0.063964	0.233105	1.000000	0.076610	0.130829
Turbo	0.183840	-0.142493	-0.056176	0.021863	0.007876	0.073376	0.076610	1.000000	-0.145262
Electric source	-0.098563	-0.069741	0.109517	-0.162708	0.040801	-0.119528	0.130829	-0.145262	1.000000

```
sns.heatmap(cor, annot= True, linewidths= 0.5,cmap='hot')
plt.title('Correlation Heatmap')
plt.show()
```




```
plt.figure(dpi=120)
sns.heatmap(dataset.isna())
```

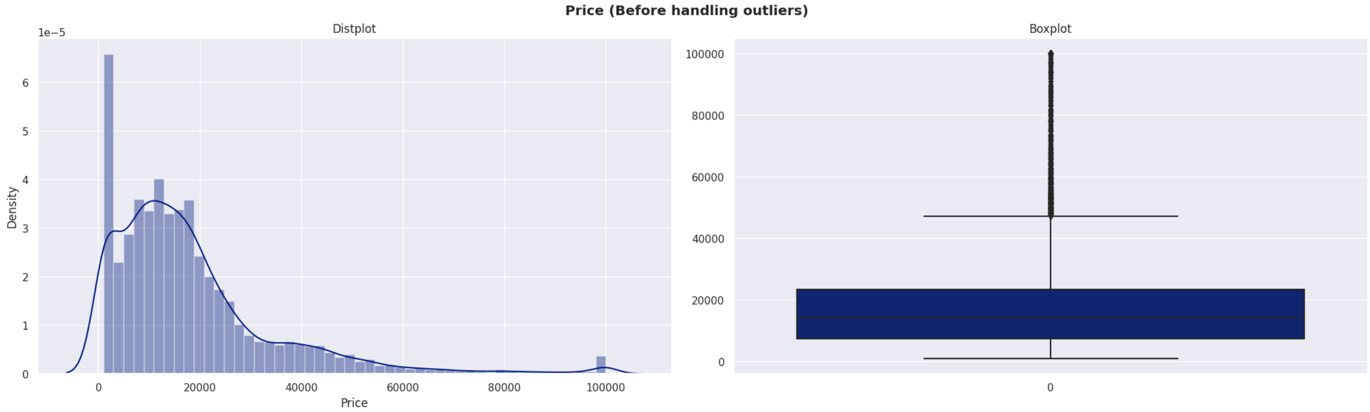
```
<Axes: >
```



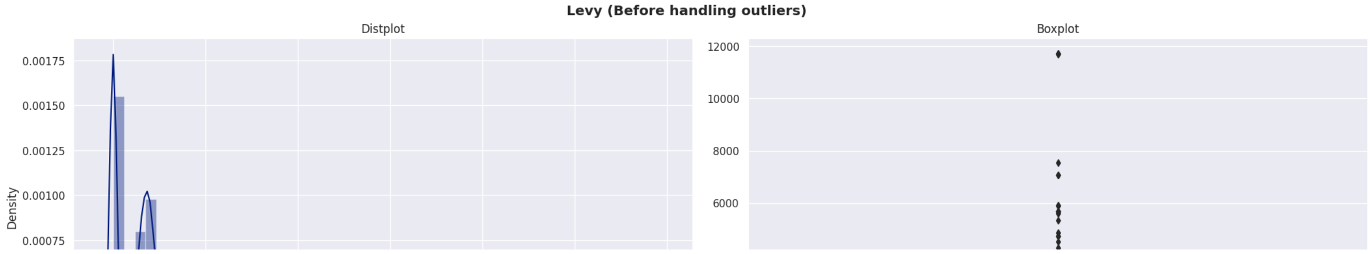

```
# Detect OutLier
numeric_data = dataset.select_dtypes(exclude=object)

for col in numeric_data:
    fig, ax =plt.subplots(1,2, constrained_layout=True)
    fig.set_size_inches(20, 6)
    sns.distplot(dataset[col], ax=ax[0]).set(title="Distplot")
    sns.boxplot(dataset[col], ax=ax[1]).set(title="Boxplot")
    plt.suptitle(f'{col.title()} (Before handling outliers)',weight='bold')
    fig.show()
```

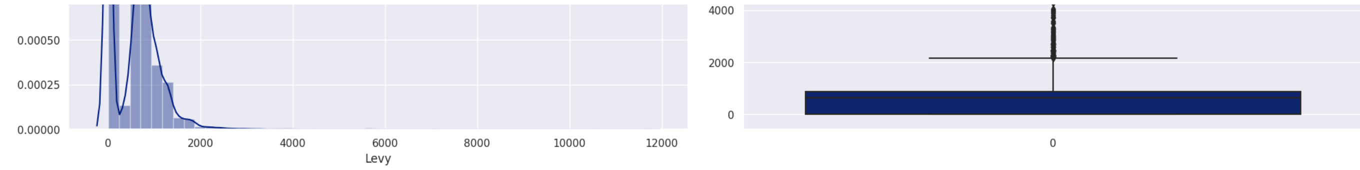
```
<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


<ipython-input-294-c3bafe3bb87c>:4: UserWarning:


`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

Price (Before handling outliers)

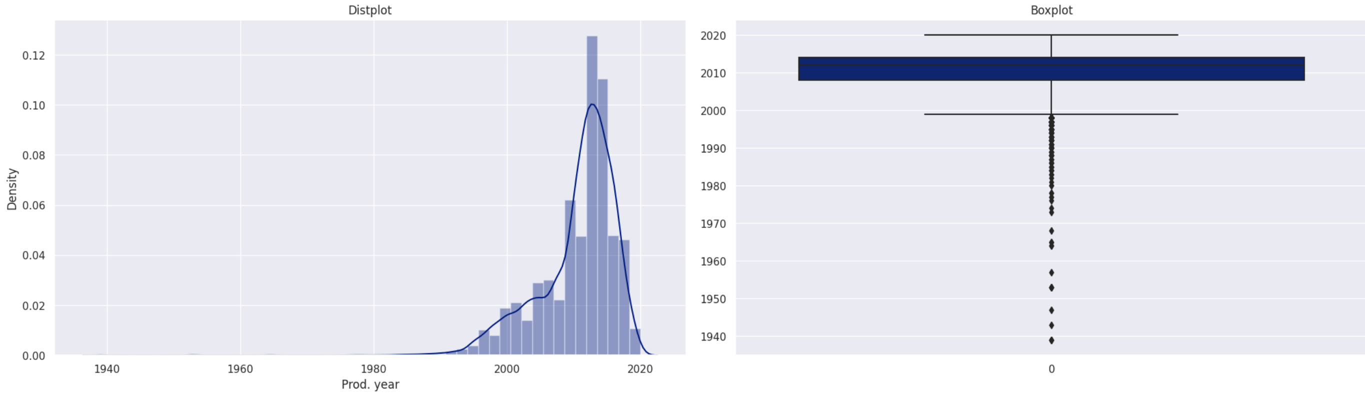


Levy (Before handling outliers)

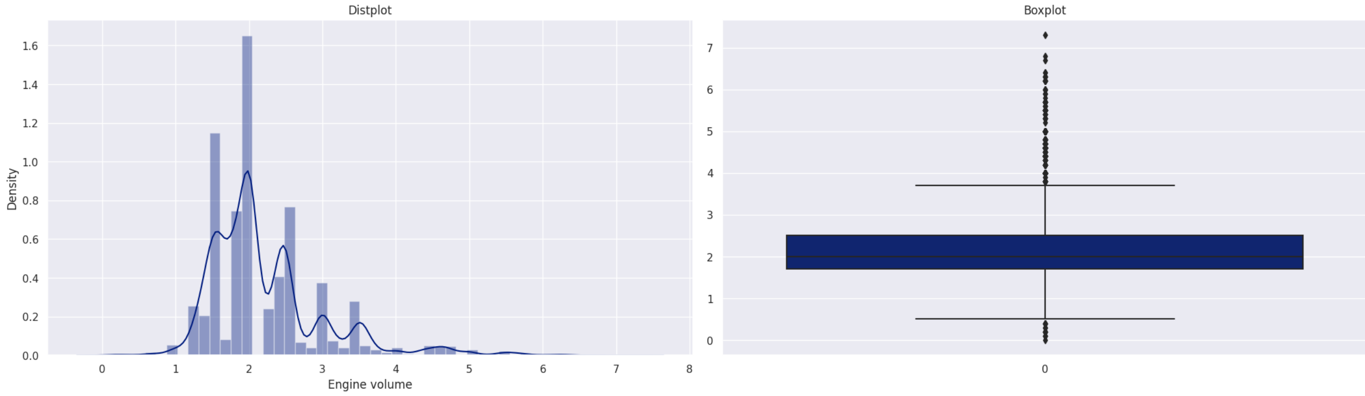




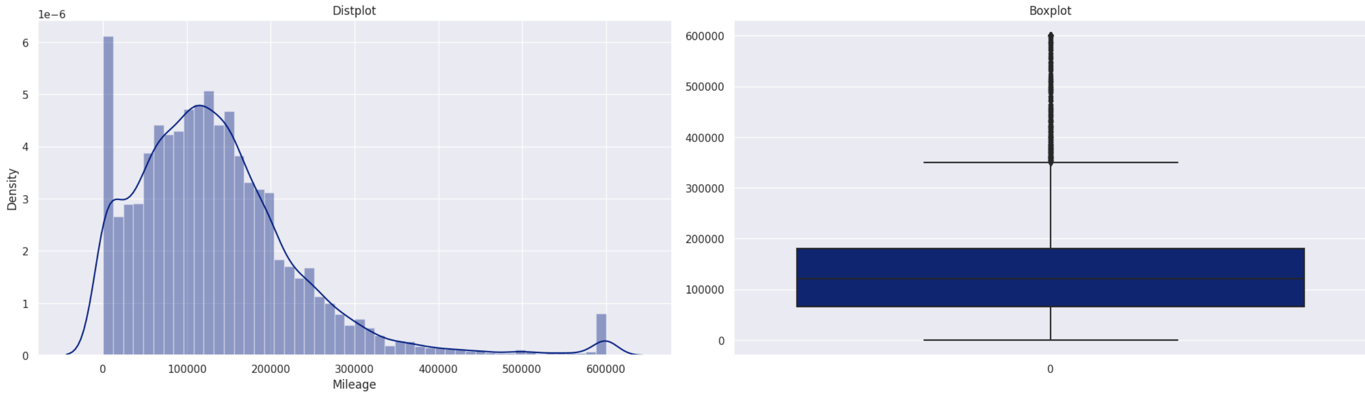
Prod. Year (Before handling outliers)



Engine Volume (Before handling outliers)



Mileage (Before handling outliers)



Cylinders (Before handling outliers)


```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
#copy dulu ke temporary dataframe
df_tmp = dataset.copy()
```

```
df_tmp.head()
```

	Price	Levy	Manufacturer	Model	Prod. year	Category	Leather interior	Fuel type	Engine volume	Mileage	Cylinders	Gear box type	Drive wheels	Wheel	Color	Airbags	Turbo	Electric source
0	13328	1399	LEXUS	RX 450	2010	Jeep	Yes	Petrol	3.5	186005.0	6.0	Automatic	4x4	Left wheel	Silver	12	0.0	1.0
1	16621	1018	CHEVROLET	Equinox	2011	Jeep	No	Petrol	3.0	192000.0	6.0	Tiptronic	4x4	Left wheel	Black	8	0.0	0.0
2	8467	0	HONDA	FIT	2006	Hatchback	No	Petrol	1.3	200000.0	4.0	Variator	Front	Right-hand drive	Black	2	0.0	0.0
3	3607	862	FORD	Escape	2011	Jeep	Yes	Petrol	2.5	168966.0	4.0	Automatic	4x4	Left wheel	White	0	0.0	1.0
4	11726	446	HONDA	FIT	2014	Hatchback	Yes	Petrol	1.3	91901.0	4.0	Automatic	Front	Left wheel	Silver	4	0.0	0.0

```
obdata = dataset.select_dtypes(include=object)
numdata = dataset.select_dtypes(exclude=object)
```

```
from sklearn.preprocessing import LabelEncoder
lab = LabelEncoder()
```

```
for i in range(0,obdata.shape[1]):
    obdata.iloc[:,i] = lab.fit_transform(obdata.iloc[:,i])
```

```
<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`

<ipython-input-300-162b463d48c9>:2: DeprecationWarning:

In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newvals)`
```

```
data = pd.concat([obdata,numdata],axis=1)
```

```
# telah mengubah data ke bentuk binary
data.head()
```

	Manufacturer	Model	Category	Leather interior	Fuel type	Gear box type	Drive wheels	Wheel	Color	Price	Levy	Prod. year	Engine volume	Mileage	Cylinders	Airbags	Turbo	Electric source
0	32	1242	4	1	4	0	0	0	12	13328	1399	2010	3.5	186005.0	6.0	12	0.0	1.0
1	8	658	4	0	4	2	0	0	1	16621	1018	2011	3.0	192000.0	6.0	8	0.0	0.0
2	21	684	3	0	4	3	1	1	1	8467	0	2006	1.3	200000.0	4.0	2	0.0	0.0
3	16	661	4	1	4	0	0	0	14	3607	862	2011	2.5	168966.0	4.0	0	0.0	1.0
4	21	684	3	1	4	0	1	0	12	11726	446	2014	1.3	91901.0	4.0	4	0.0	0.0

```
#Bagi datanya menjadi Variabel independen dan dependen
X = data.drop(['Price'], axis=1)
y = data['Price']
```

```
X.head()
```

	Manufacturer	Model	Category	Leather interior	Fuel type	Gear box type	Drive wheels	Wheel	Color	Levy	Prod. year	Engine volume	Mileage	Cylinders	Airbags	Turbo	Electric source
0	32	1242	4	1	4	0	0	0	12	1399	2010	3.5	186005.0	6.0	12	0.0	1.0
1	8	658	4	0	4	2	0	0	1	1018	2011	3.0	192000.0	6.0	8	0.0	0.0
2	21	684	3	0	4	3	1	1	1	0	2006	1.3	200000.0	4.0	2	0.0	0.0
3	16	661	4	1	4	0	0	0	14	862	2011	2.5	168966.0	4.0	0	0.0	1.0
4	21	684	3	1	4	0	1	0	12	446	2014	1.3	91901.0	4.0	4	0.0	0.0

```
y.head()
```

```
0    13328
1    16621
2     8467
3     3607
4    11726
Name: Price, dtype: int64
```

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
X = pd.DataFrame(X)
```

```
X.tail(5)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
15699	0.909091	0.409904	-0.8	0.0	0.0	0.0	0.0	0.0	0.454545	0.006961	-0.166667	-0.25	1.642954	0.0	0.750	0.0	1.0
15700	0.242424	-0.647868	-1.2	0.0	-4.0	1.0	1.0	0.0	0.454545	-0.741299	-2.166667	0.00	1.578365	0.0	-0.125	1.0	0.0
15701	-0.151515	0.657497	0.4	0.0	0.0	2.0	0.0	0.0	0.363636	0.222738	-0.166667	0.50	0.357998	0.0	0.250	0.0	0.0
15702	-0.151515	0.806052	-0.6	0.0	-3.0	0.0	0.0	0.0	0.000000	0.228538	-0.333333	0.00	-0.040870	0.0	-0.250	0.0	0.0
15703	-0.151515	0.657497	0.4	0.0	0.0	0.0	0.0	0.0	0.636364	0.132251	0.000000	0.50	0.581288	0.0	0.750	0.0	1.0

8. Membangun Model

Pada penelitian ini, model yang digunakan yaitu:

1. Linear Regression
2. Random Forest Regressor
3. Logistic Regression
4. XGBoost
5. LightGBM

```
X.describe()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
count	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000	15704.000000
mean	0.145609	0.009928	-0.158227	-0.328897	-0.777636	0.647160	-0.050560	0.091123	0.076269	-0.090015	-0.247400	0.266198	0.121799	0.483698	0.068040	0.120288	0.159004
std	0.547823	0.580893	0.563413	0.469827	1.325311	0.947717	0.546498	0.287794	0.484026	0.649164	1.002433	0.990748	0.888286	1.134630	0.508107	0.325308	0.365692
min	-0.848485	-1.177442	-1.400000	-1.000000	-4.000000	0.000000	-1.000000	0.000000	-0.636364	-0.741299	-12.166667	-2.500000	-1.066939	-3.000000	-0.750000	0.000000	0.000000
25%	-0.212121	-0.482806	-0.600000	-1.000000	-1.000000	0.000000	0.000000	0.000000	-0.545455	-0.741299	-0.666667	-0.375000	-0.479757	0.000000	-0.250000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.787879	0.517194	0.400000	0.000000	0.000000	1.000000	0.000000	0.000000	0.454545	0.258701	0.333333	0.625000	0.520243	0.000000	0.750000	0.000000	0.000000
max	1.090909	1.008253	0.600000	0.000000	0.000000	3.000000	1.000000	1.000000	0.727273	12.848028	1.333333	6.625000	4.223669	12.000000	1.250000	1.000000	1.000000

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15704 entries, 0 to 15703
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       15704 non-null  float64
 1   1       15704 non-null  float64
 2   2       15704 non-null  float64
 3   3       15704 non-null  float64
 4   4       15704 non-null  float64
 5   5       15704 non-null  float64
 6   6       15704 non-null  float64
```

```
 7   7       15704 non-null  float64
 8   8       15704 non-null  float64
 9   9       15704 non-null  float64
 10  10      15704 non-null  float64
 11  11      15704 non-null  float64
 12  12      15704 non-null  float64
 13  13      15704 non-null  float64
 14  14      15704 non-null  float64
 15  15      15704 non-null  float64
 16  16      15704 non-null  float64
dtypes: float64(17)
memory usage: 2.0 MB
```

```
#Split menjadi Train dan Test (Data Pre-Processing)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.1, random_state = 0)
```

9. Mengevaluasi Hasil Pemodelan

Linear Regression

```
from sklearn.linear_model import LinearRegression

#Buat model Linear Regressor
lm = LinearRegression()

#Fit data ke Model untuk di Training
lm.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lm.intercept_
```

```
16331.741434587864
```

Linear Regression Model Evaluation

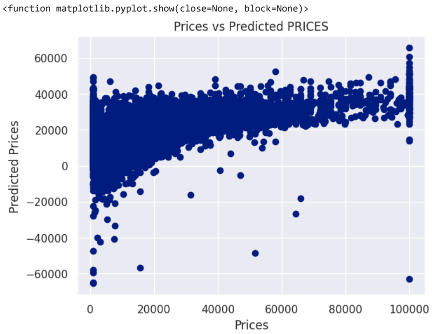
```
#Model Evaluation
#Bandingkan hasil prediksi dengan train data
y_pred = lm.predict(X_train)
```

```
#Model Evaluation
print("R^2: ", metrics.r2_score(y_train, y_pred))
# print("Adjusted R^2: ", 1 - (1 - metrics.r2_score(y_train, y_pred))
print("MAE: ", metrics.mean_absolute_error(y_train, y_pred))
print("MSE: ", metrics.mean_squared_error(y_train, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2:  0.33043189597449085
MAE:  9492.130665498604
MSE:  187769025.58357227
RMSE:  13702.883841862349
```

```
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted Prices")
plt.title("Prices vs Predicted PRICES")
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
#Prediksi model dalam data test
y_pred = lm.predict(X_test)
```

```
#Model Evaluation Test
acc_linreg = metrics.r2_score(y_test, y_pred)
print("R^2: ", acc_linreg)
print("MAE: ", metrics.mean_absolute_error(y_test, y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
R^2:  0.3651295485683653
MAE:  9594.380156458226
MSE:  188304891.2939018
RMSE:  13722.422938165906
```

Random Forest Regressor

```
# Import Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor
reg = RandomForestRegressor()

# Train the model using the training sets
reg.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
# Model prediction on train data
y_pred = reg.predict(X_train)
```

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9602532622009331
MAE: 1837.681841467003
MSE: 11146298.907888541
RMSE: 3338.607330592884
```

```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```


```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
```

```
# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8013063760321597
MAE: 4631.355210751257
MSE: 58933253.51287758
RMSE: 7676.799692116343
```

Logistic Regression

XGBoost

```
from xgboost import XGBRegressor

reg = XGBRegressor()
reg.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```
#Prediksi model dalam data
y_pred = reg.predict(X_train)
```

```
#Model Evaluation
print("R^2: ", metrics.r2_score(y_train, y_pred))
# print("Adjusted R^2: ", 1 - (1 - metrics.r2_score(y_train, y_pred))
print("MAE: ", metrics.mean_absolute_error(y_train, y_pred))
print("MSE: ", metrics.mean_squared_error(y_train, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2:  0.9098029457786557
MAE:  3389.003413893299
MSE:  25294235.014822666
RMSE:  5029.337432984852
```

```
#Prediksi hasil test
y_test_pred = reg.predict(X_test)
```

```
#Model Evaluation
acc_xgb = metrics.r2_score(y_test, y_test_pred)
print("R^2: ", acc_xgb)
# print("Adjusted R^2: ", 1 - (1 - metrics.r2_score(y_test, y_test_pred))
print("MAE: ", metrics.mean_absolute_error(y_test, y_test_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_test_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2:  0.7921218604521009
MAE:  4860.898914490134
MSE:  61657414.33023825
RMSE:  7852.223527781049
```

LightGBM

```
from lightgbm import LGBMRegressor

reg = LGBMRegressor()
reg.fit(X_train, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.002054 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 985
[LightGBM] [Info] Number of data points in the train set: 14133, number of used features: 17
[LightGBM] [Info] Start training from score 18306.709757
LGBMRegressor
LGBMRegressor()
```

```
#Prediksi model dalam data
y_pred = reg.predict(X_train)
```

```
#Model Evaluation
print("R^2: ", metrics.r2_score(y_train, y_pred))
# print("Adjusted R^2: ", 1 - (1 - metrics.r2_score(y_train, y_pred))
print("MAE: ", metrics.mean_absolute_error(y_train, y_pred))
print("MSE: ", metrics.mean_squared_error(y_train, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2:  0.8239617971518581
MAE:  4472.955779370543
MSE:  49366930.138326205
RMSE:  7026.160412225599
```

```
#Prediksi hasil test
y_test_pred = reg.predict(X_test)
```

```
#Model Evaluation
acc_lgb = metrics.r2_score(y_test, y_test_pred)
print("R^2: ", acc_lgb)
# print("Adjusted R^2: ", 1 - (1 - metrics.r2_score(y_test, y_test_pred))
print("MAE: ", metrics.mean_absolute_error(y_test, y_test_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_test_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2:  0.7819210663300206
MAE:  5011.9994025320175
MSE:  64683007.069573104
RMSE:  8042.574654274159
```

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor(random_state = 0)
reg.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

```
y_pred = reg.predict(X_train)
```

```
print("R^2: ", metrics.r2_score(y_train, y_pred))
print("MAE: ", metrics.mean_absolute_error(y_train, y_pred))
print("MSE: ", metrics.mean_squared_error(y_train, y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2:  0.9960720831585175
MAE:  82.26894502228826
MSE:  1101517.6999386777
RMSE:  1049.5321338285348
```

```
#Prediksi hasil test
y_test_pred = reg.predict(X_test)
```

```
#Model Evaluation
acc_dt = metrics.r2_score(y_test, y_test_pred)
print("R^2: ", acc_lr)
print("MAE: ", metrics.mean_absolute_error(y_test, y_test_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_test_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2:  0.6207371111776337
MAE:  5907.016634839804
MSE:  112490756.01244502
RMSE:  10606.165943093905
```

Pada penelitian ini untuk mendapatkan hasil model terbaik diperoleh dari hasil nilai R2 dan RMSE dari setiap model yang ada.

```
models = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost','LightGBM','Decision Tree'],
    'R-squared Score': [acc_linreg*100, acc_rf*100, acc_xgb*100,acc_lgb*100,acc_dt*100]})
model = models.sort_values(by='R-squared Score', ascending=False).reset_index(drop=True)
model
```

	Model	R-squared Score
0	Random Forest	80.130638
1	XGBoost	79.212186
2	LightGBM	78.192107
3	Decision Tree	62.073711
4	Linear Regression	36.512955

Pada kolom hasil sintak diatas, diperoleh bahwa model yang terbaik untuk digunakan dalam data 'car price prediction' yaitu model Random Forest karena pada model ini mendapatkan nilai R2 terbesar yaitu sebesar 80,06 dan nilai RMSE nya sebesar 7690.

Hyper Parameter Tuning

Random Forest Regressor

```
# params=
#  {'bootstrap': True,
#   'ccp_alpha': 0.0,
#   'criterion': 'squared_error',
#   'max_depth': None,
#   'max_features': 1.0,
#   'max_leaf_nodes': None,
#   'max_samples': None,
#   'min_impurity_decrease': 0.0,
#   'min_samples_leaf': 1,
#   'min_samples_split': 2,
#   'min_weight_fraction_leaf': 0.0,
#   'n_estimators': 100,
#   'n_jobs': None,
#   'oob_score': False,
#   'random_state': None,
#   'verbose': 0,
#   'warm_start': False}

# Create a Random Forest Regressor
reg = RandomForestRegressor(n_estimators = 200,
                            criterion= 'squared_error',
                            random_state= 100)
```

```
# Train the model using the training sets
reg.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(n_estimators=200, random_state=100)
```

```
# Model prediction on train data
y_pred = reg.predict(X_train)
```

```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.9611429654288371
MAE: 1821.5661513862167
MSE: 10896796.718107192
RMSE: 3301.0296451421323
```

```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```


```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
```

```
# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8050294360616986
MAE: 4600.371137026894
MSE: 57828980.32995959
```

XGBoost

```
# Params=
# {'objective': 'reg:squarederror',
#  'base_score': None,
#  'booster': None,
#  'callbacks': None,
#  'colsample_bylevel': None,
#  'colsample_bynode': None,
#  'colsample_bytree': None,
#  'device': None,
#  'early_stopping_rounds': None,
#  'enable_categorical': False,
#  'eval_metric': None,
#  'feature_types': None,
#  'gamma': None,
#  'grow_policy': None,
#  'importance_type': None,
#  'interaction_constraints': None,
#  'learning_rate': None,
#  'max_bin': None,
#  'max_cat_threshold': None,
#  'max_cat_to_onehot': None,
#  'max_delta_step': None,
#  'max_depth': None,
#  'max_leaves': None,
#  'min_child_weight': None,
#  'missing': nan,
#  'monotone_constraints': None,
#  'multi_strategy': None,
#  'n_estimators': None,
#  'n_jobs': None,
#  'num_parallel_tree': None,
#  'random_state': None,
#  'reg_alpha': None,
#  'reg_lambda': None,
#  'sampling_method': None,
#  'scale_pos_weight': None,
#  'subsample': None,
#  'tree_method': None,
#  'validate_parameters': None
```