A3 - Predicting Car Prices

- Anushka Ojha st126222
- Githublink: https://github.com/anushkaojha11/carprediction_a3

Importing the libraries

In [405…
```
#Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn import preprocessing
warnings.filterwarnings('ignore')
```

Loading data

In [406…
```
# Load the dataset using pandas and storing them in a dataframe variable
df = pd.read_csv('Cars.csv')
```

In [407…
```
# Displaying the first few rows of the dataframe
df.head()
```

Out[407…

	name	year	selling_price	km_driven	fuel	seller_type	transmission	ow
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	F Ow
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Seco Ow
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Th Ow
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	F Ow
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	F Ow

In [408…
```
#Checking the statistical summary of the dataframe
df.describe()
```

Out[408…

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

In [409…

```
#Checking the data types and non-null values in the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [410…

```
#Checking the number of rows and columns in the dataframe
df.shape
```

Out[410…

```
(8128, 13)
```

In [411…

```
#Checking the column names in the dataframe
df.columns
```

Out[411…

```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_typ
e',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [412…

```
#Checking for unique values in "owner" column
df['owner'].unique()
```

Out[412…
```
array(['First Owner', 'Second Owner', 'Third Owner',
       'Fourth & Above Owner', 'Test Drive Car'], dtype=object)
```

Task 1: Preparing the datasets

Create a mapping dictionary for the 'Owner' column

In [413…
```
#Mapping the owner column to numerical values.
ownermap = {
    "First Owner":1,
    "Second Owner":2,
    "Third Owner":3,
    "Fourth & Above Owner":4,
    "Test Drive Car":5
    }
```

In [414…
```
#Checking if the owner column exists in the dataframe and then mapping it
if 'owner' in df.columns:
    df['owner'] = df['owner'].map(ownermap)
```

In [415…
```
#Checking if mapping is done correctly
df.owner.unique()
```

Out[415…
```
array([1, 2, 3, 4, 5])
```

Removing the rows of 'Fuel' column with values of LPG and CNG

In [416…
```
#Getting the count of unique values in the 'Fuel' column
df['fuel'].unique()
```

Out[416…
```
array(['Diesel', 'Petrol', 'LPG', 'CNG'], dtype=object)
```

In [417…
```
#Storing the unique values to remove in an array
fuel_to_remove = ['CNG', 'LPG']
```

In [418…
```
#Removing the unwanted fuel types from the dataframe
df = df[~df['fuel'].isin(fuel_to_remove)]
```

In [419…
```
#checking if the unwanted fuel types are removed
df['fuel'].unique()
```

Out[419…
```
array(['Diesel', 'Petrol'], dtype=object)
```

Removing "kmpl" and converting the column to numerical type for feature mileage

In [420…
```
#Getting the values of mileage column
df['mileage'].head()
```

Out[420…
```
0     23.4 kmpl
1    21.14 kmpl
2     17.7 kmpl
3     23.0 kmpl
4     16.1 kmpl
Name: mileage, dtype: object
```

In [421…
```
#Removing the "kmpl" from the mileage column
df['mileage']= df['mileage'].str.split( ).str[0]
```

In [422…
```
#Converting the mileage column to float type
df['mileage'] = df['mileage'].astype(float)
```

In [423…
```
#Checking if the conversation is done correctly
df['mileage'].dtype
```

Out[423…
```
dtype('float64')
```

Removing "CC" and converting the column to numerical type for feature engine

In [424…
```
#Getting the values of 'engine' feature column
df['engine'].head()
```

Out[424…
```
0    1248 CC
1    1498 CC
2    1497 CC
3    1396 CC
4    1298 CC
Name: engine, dtype: object
```

In [425…
```
#Removing the 'CC' from the engine column using str.split() method
df['engine'] = df['engine'].str.split( ).str[0]
```

In [426…
```
#Verifying the changes in the engine column
df['engine'].head()
```

Out[426…
```
0    1248
1    1498
2    1497
3    1396
4    1298
Name: engine, dtype: object
```

In [427…
```
#converting the data type of engine column to float
df['engine'] = df['engine'].astype(float)
```

In [428…
```
#verifying the data type of engine column
df['engine'].dtype
```

Out[428…
```
dtype('float64')
```

Removing "bhp" and converting the column to numerical type for feature max_power

In [429…
```
#Getting the values of the column 'max_power'
df['max_power'].head()
```

Out[429…
```
0        74 bhp
1    103.52 bhp
2        78 bhp
3        90 bhp
4      88.2 bhp
Name: max_power, dtype: object
```

In [430…
```
#Removing the 'bhp' from the max_power column using str.split() method
df['max_power'] = df['max_power'].str.split( ).str[0]
```

In [431…
```
#Converting the data type of max_power column to float
df['max_power'] = df['max_power'].astype(float)
```

In [432…
```
#Verifying the changes
df['engine'].head()
```

Out[432…
```
0    1248.0
1    1498.0
2    1497.0
3    1396.0
4    1298.0
Name: engine, dtype: float64
```

In [433…
```
#Converting the mileage column to float type
df['mileage'] = df['mileage'].astype(float)
```

In [434…
```
#Verifying the changes
df['engine'].dtypes
```

Out[434…
```
dtype('float64')
```

Taking only the first word for the feature brand

In [435…
```
df.columns
```

Out[435…
```
Index(['name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_typ
e',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [436…
```
#Renaming the column 'name' to 'brand'
df.rename(columns={'name':'brand'}, inplace=True)
#checking the changes
df.columns
```

Out[436…
```
Index(['brand', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_ty
pe',
       'transmission', 'owner', 'mileage', 'engine', 'max_power', 'torqu
e',
       'seats'],
      dtype='object')
```

In [437…
```
#Getting information about the column 'brand'
df['brand'].head()
```

Out[437…
```
0          Maruti Swift Dzire VDI
1    Skoda Rapid 1.5 TDI Ambition
2        Honda City 2017-2020 EXi
3       Hyundai i20 Sportz Diesel
4          Maruti Swift VXI BSIII
Name: brand, dtype: object
```

In [438…
```
#Only taking the first word from the brand column by using the str.split(
df['brand'] = df['brand'].str.split( ).str[0]
```

In [439…
```
#Verifying the changes
df['brand'].head()
```

Out[439…
```
0     Maruti
1      Skoda
2      Honda
3    Hyundai
4     Maruti
Name: brand, dtype: object
```

Dropping the feature torque

In [440…
```
#Removing the feature 'torque' from the dataframe using the drop() method
df = df.drop(['torque'], axis=1)
```

Removing the 'Test Drive Cars' i.e. owner = 5 from the data set.

In [441…
```
#Removing the 'Owner=5' from the dataframe using the query() method
df = df.query("owner != '5'")
```

In [442…
```
#Verifying the changes on the owner column
df['owner'].unique()
```

Out[442…
```
array([1, 2, 3, 4, 5])
```

In [443…
```
#Checking the cleaned dataframe
df.head()
```

Out[443…

	brand	year	selling_price	km_driven	fuel	seller_type	transmission	owne
0	Maruti	2014	450000	145500	Diesel	Individual	Manual	
1	Skoda	2014	370000	120000	Diesel	Individual	Manual	
2	Honda	2006	158000	140000	Petrol	Individual	Manual	
3	Hyundai	2010	225000	127000	Diesel	Individual	Manual	
4	Maruti	2007	130000	120000	Petrol	Individual	Manual	

Changing the columns 'brand', 'fuel, 'seller_type' and 'trasmission' to numerical format.

In [444…
```
#Importing the LabelEncoder from sklearn library
from sklearn.preprocessing import LabelEncoder

#Initializing the LabelEncoder
label_encoder_brand = LabelEncoder()
```

```
label_encoder_fuel = LabelEncoder()
label_encoder_seller_type = LabelEncoder()
label_encoder_transmission = LabelEncoder()

#Transforming the categorical columns using the fit_transform() method
df['brand'] = label_encoder_brand.fit_transform(df['brand'])
original_brands = label_encoder_brand.classes_.tolist()
df['fuel'] = label_encoder_fuel.fit_transform(df['fuel'])
df['seller_type'] = label_encoder_seller_type.fit_transform(df['seller_ty
df['transmission'] = label_encoder_transmission.fit_transform(df['transmi
```

In [445…

```
#Checking for the changes made
df.head()
```

Out[445…

	brand	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	20	2014	450000	145500	0	1	1	1
1	27	2014	370000	120000	0	1	1	2
2	10	2006	158000	140000	1	1	1	3
3	11	2010	225000	127000	0	1	1	1
4	20	2007	130000	120000	1	1	1	1

In [446…

```
#Creating a csv of the encoded data set.
df.to_csv('le_cars_data.csv', sep = ',', index=False, encoding='utf-8 ')
```

Exploratory Data Analysis

In [447…

```
#Seperating the numerical and categorical columns from the dataframe
#Excluding the column 'selling_price' from the numerical columns using df

df_copy = df.loc[:, df.columns != 'selling_price']

#Seperating the numerical and categorical columns from the dataframe

num_cols = df_copy.select_dtypes(include=['int64', 'float64'])
cat_cols = df_copy.select_dtypes(exclude=['int64', 'float64'])
```

In [448…

```
#Verifying the changes made to the dataset
num_cols.columns, cat_cols.columns
```

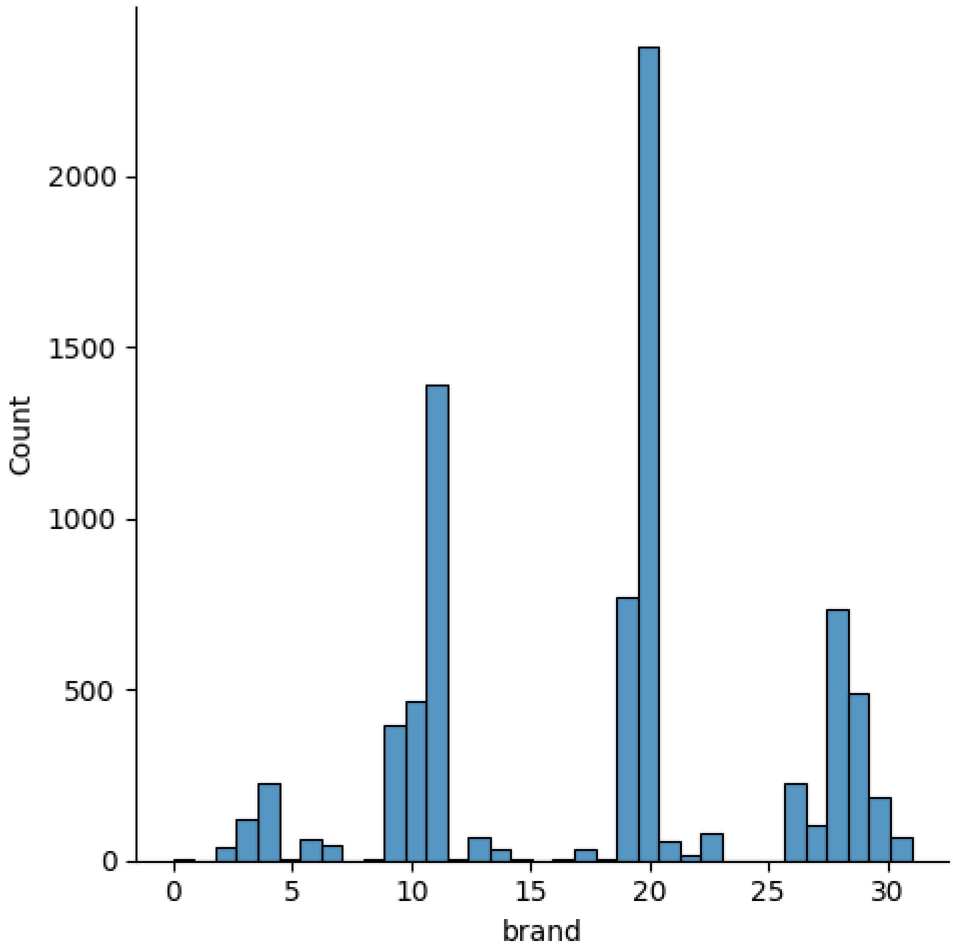
Out[448…

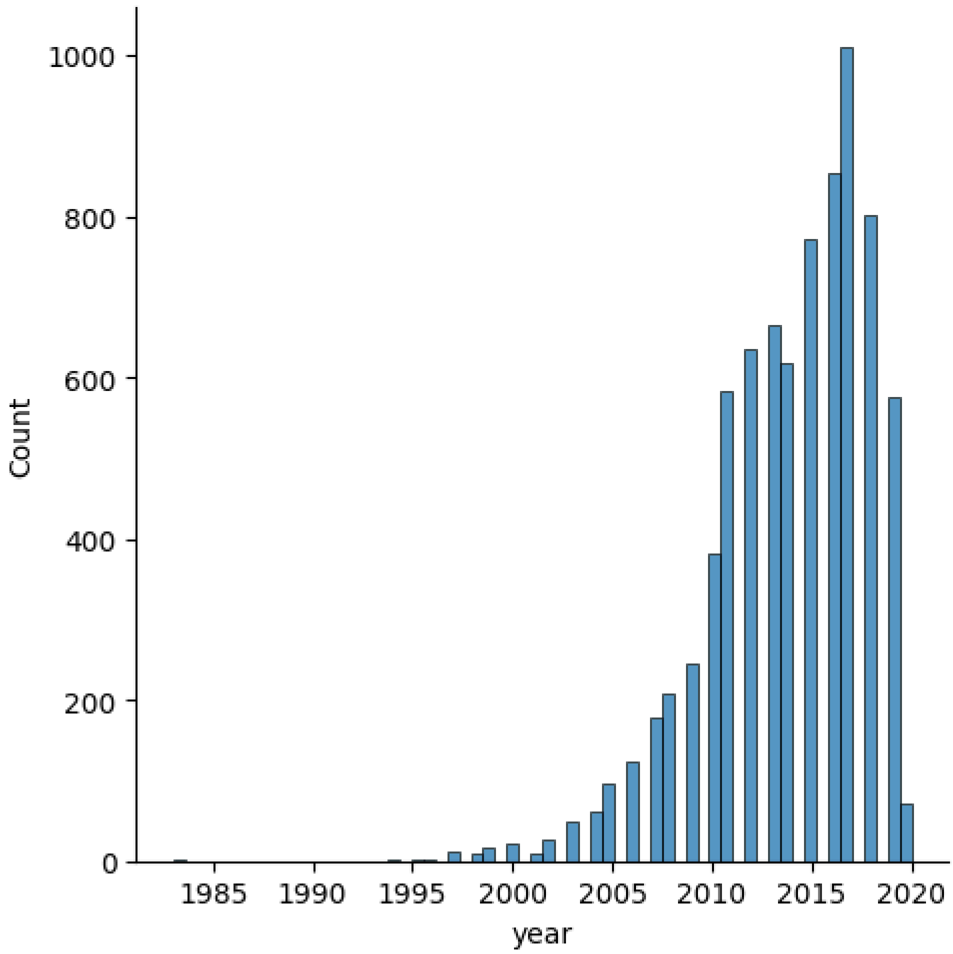
```
(Index(['brand', 'year', 'km_driven', 'fuel', 'seller_type', 'transmissi
on',
        'owner', 'mileage', 'engine', 'max_power', 'seats'],
       dtype='object'),
 Index([], dtype='object'))
```

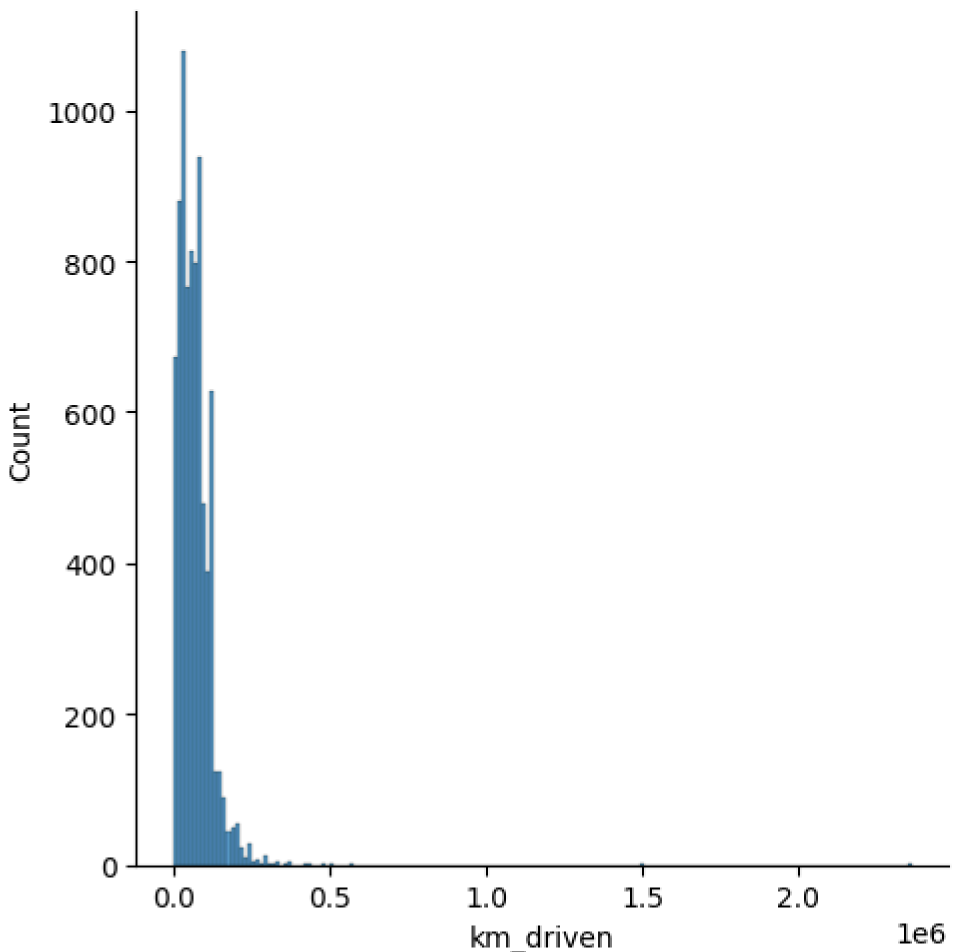
Univariate Analysis

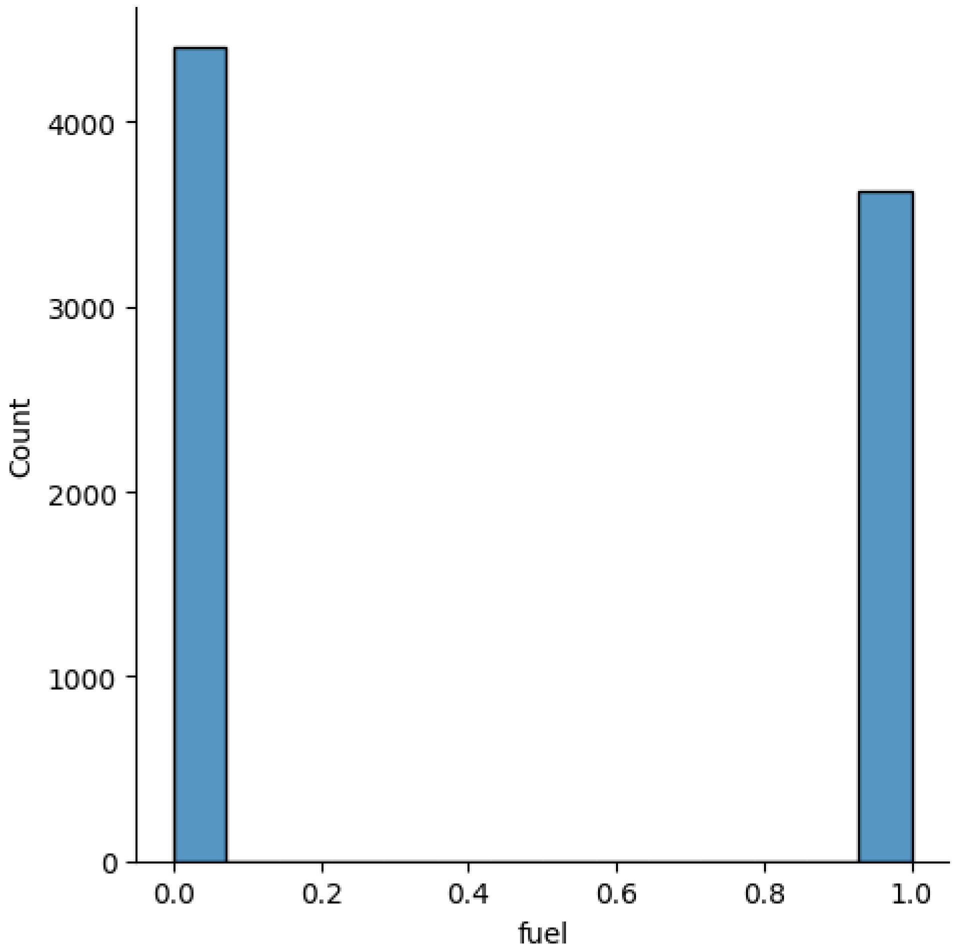
Distribution plot

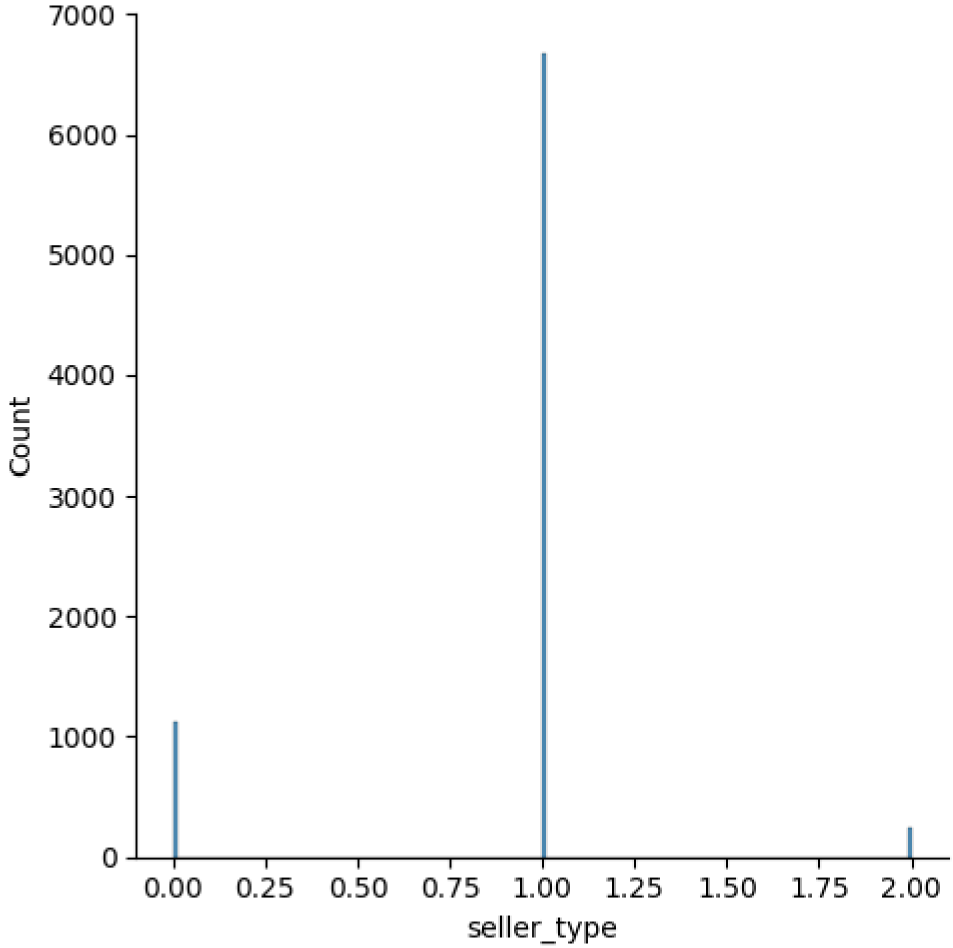
In [449…
```
#Analysing the count of all nummerical colums using the distribution plot
for col in num_cols.columns:
    sns.displot(df, x=df[col])
```

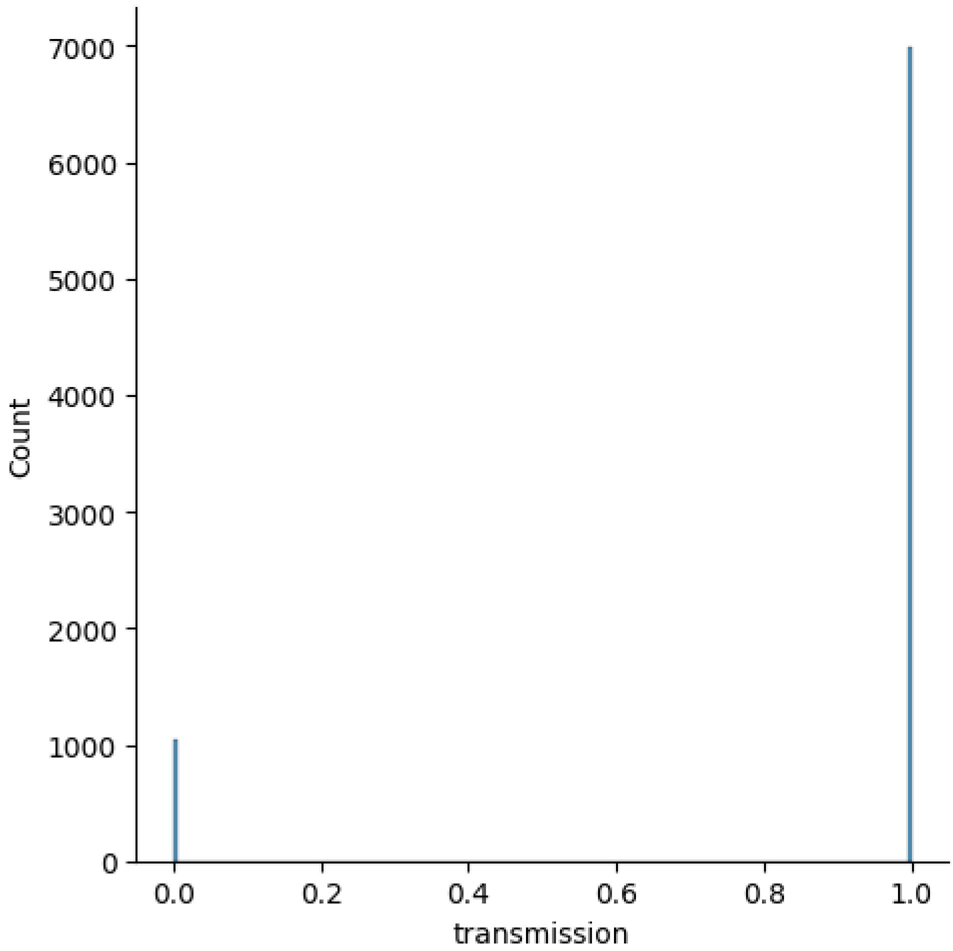


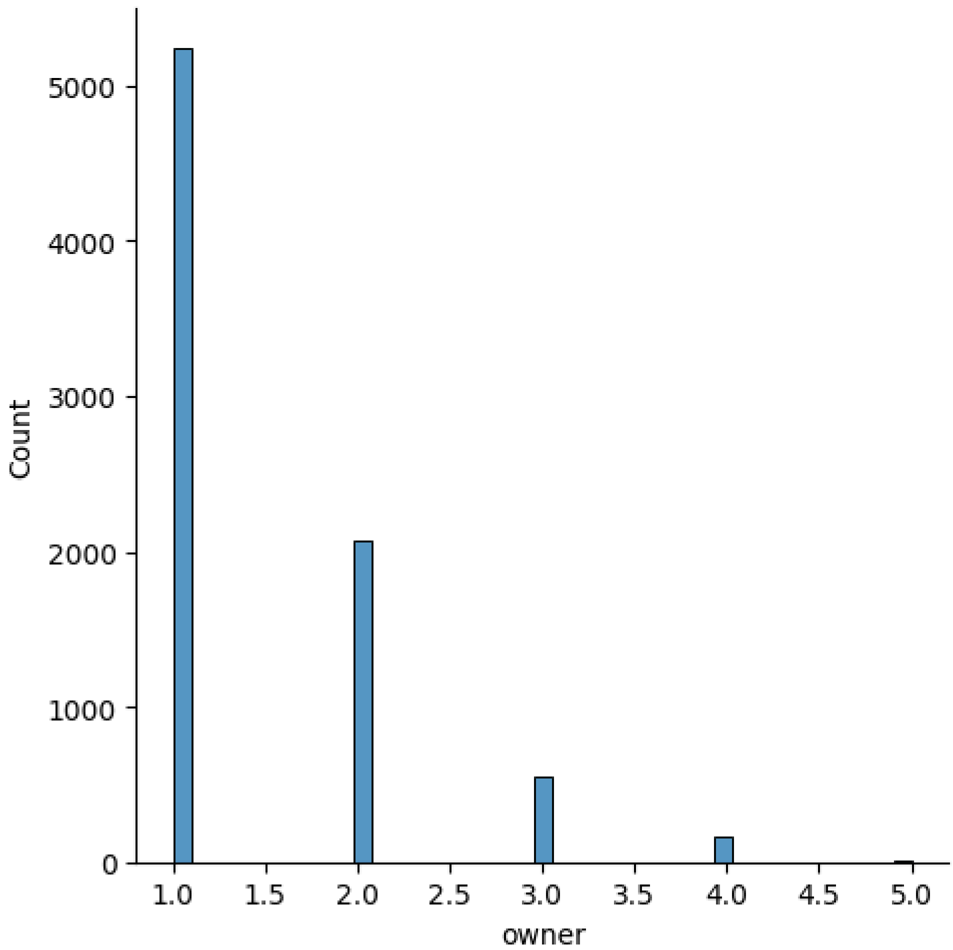

1000
800
600
400
200
0
Count
1985
1990
1995
2000
2005
2010
2015
2020
year

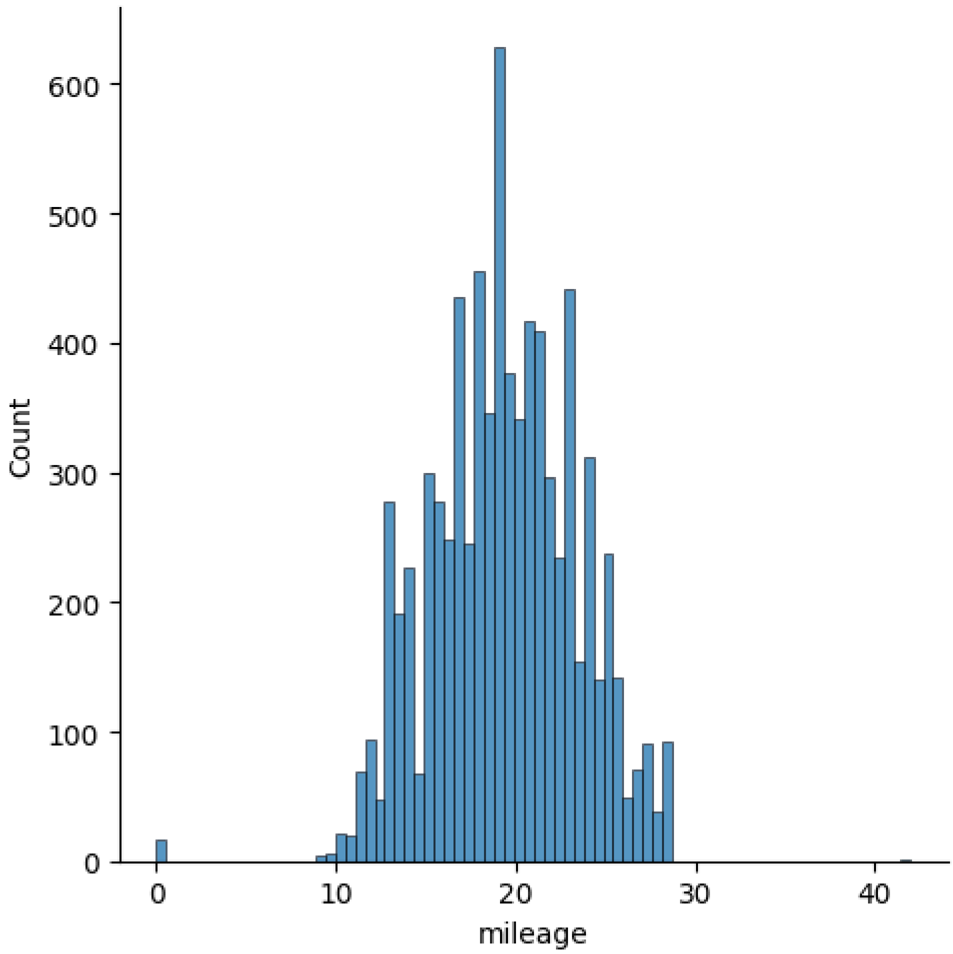

1000
800
600
400
200
0
Count
0.0
0.5
1.0
1.5
2.0
1e6
km_driven

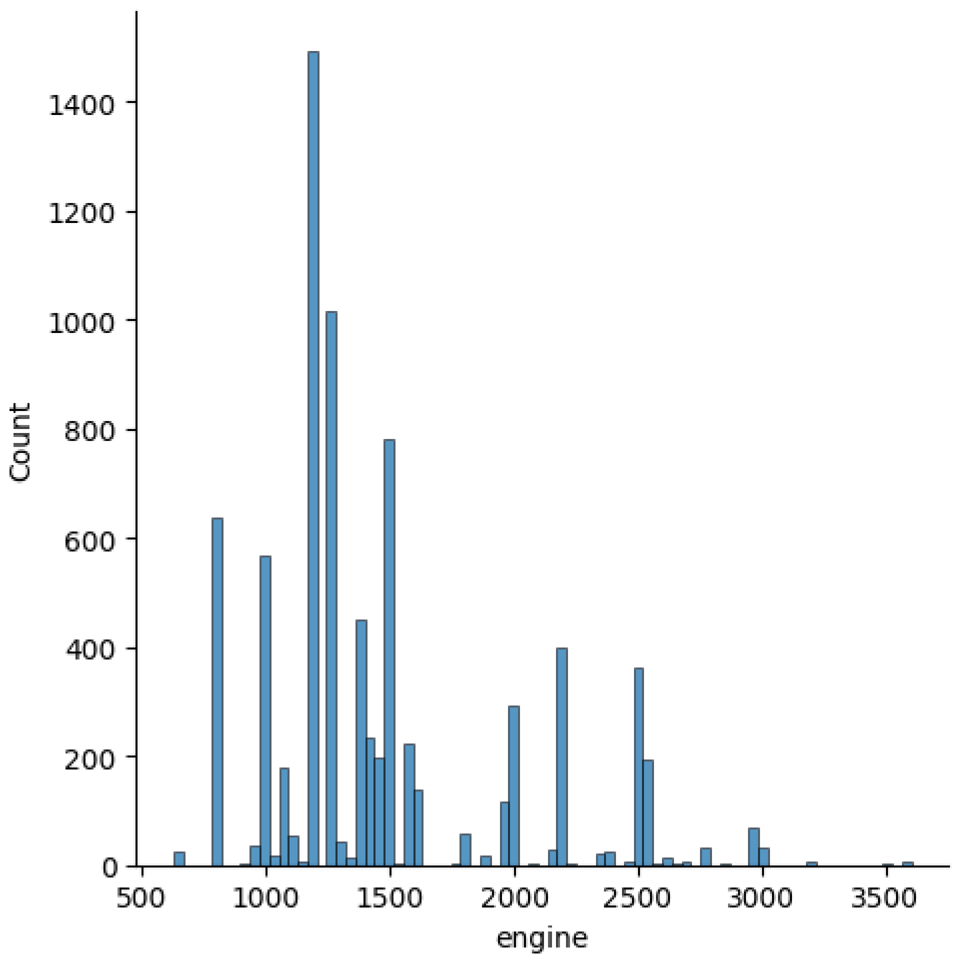

Count
0
1000
2000
3000
4000
0.0
0.2
0.4
0.6
0.8
1.0
fuel

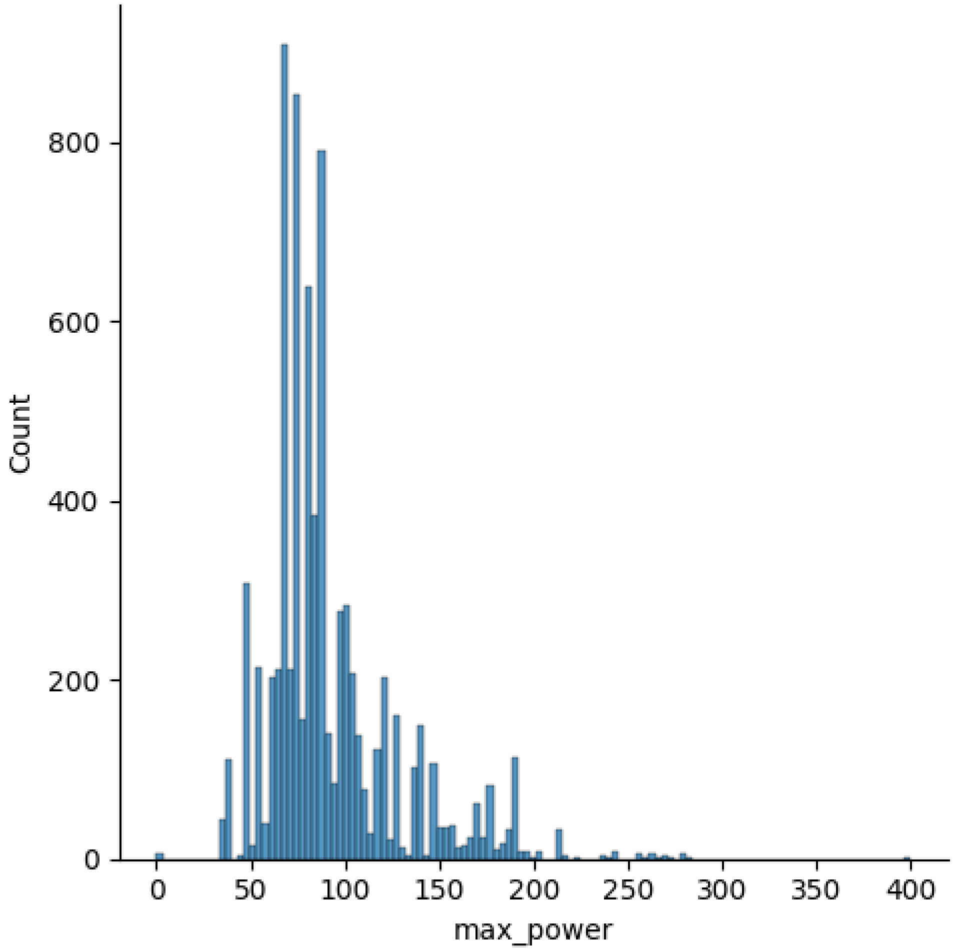

Count
0
1000
2000
3000
4000
5000
6000
7000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
seller_type


7000
6000
5000
4000
3000
2000
1000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
transmission

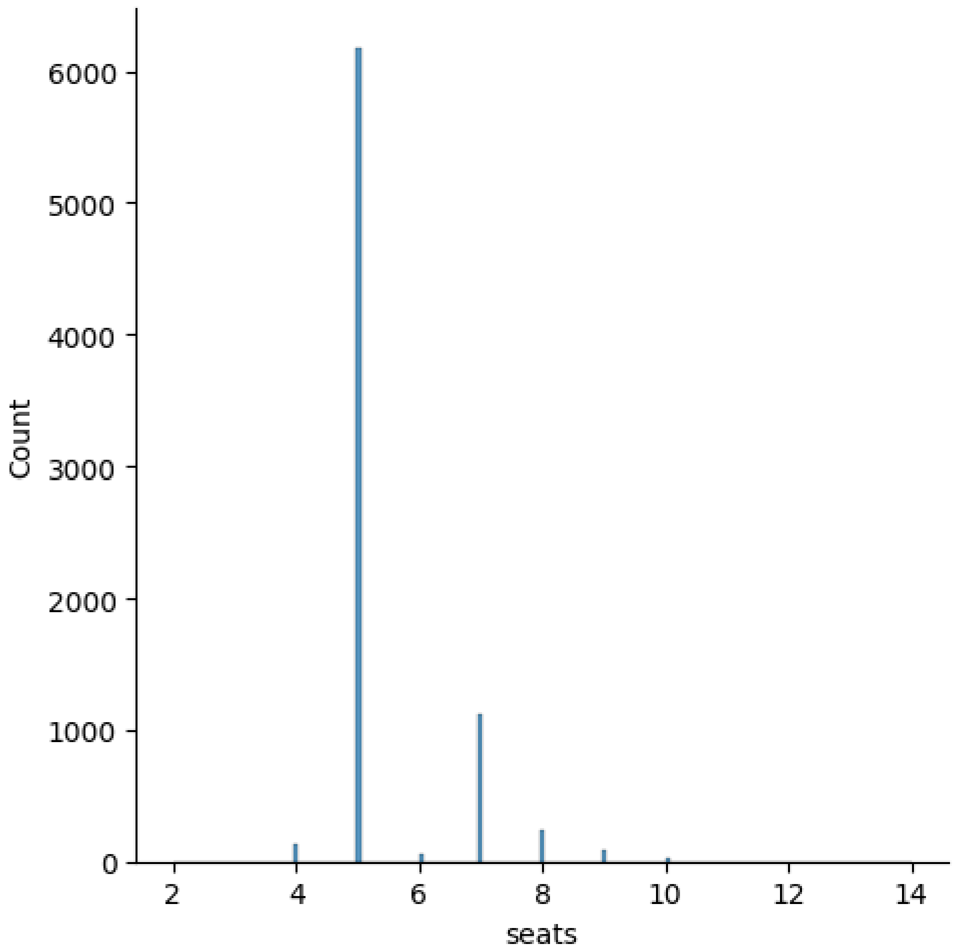

5000
4000
3000
2000
1000
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
owner


Count
mileage
0
100
200
300
400
500
600
0
10
20
30
40


Count
engine
0
200
400
600
800
1000
1200
1400
500
1000
1500
2000
2500
3000
3500



Count
max_power
0
200
400
600
800
0
50
100
150
200
250
300
350
400



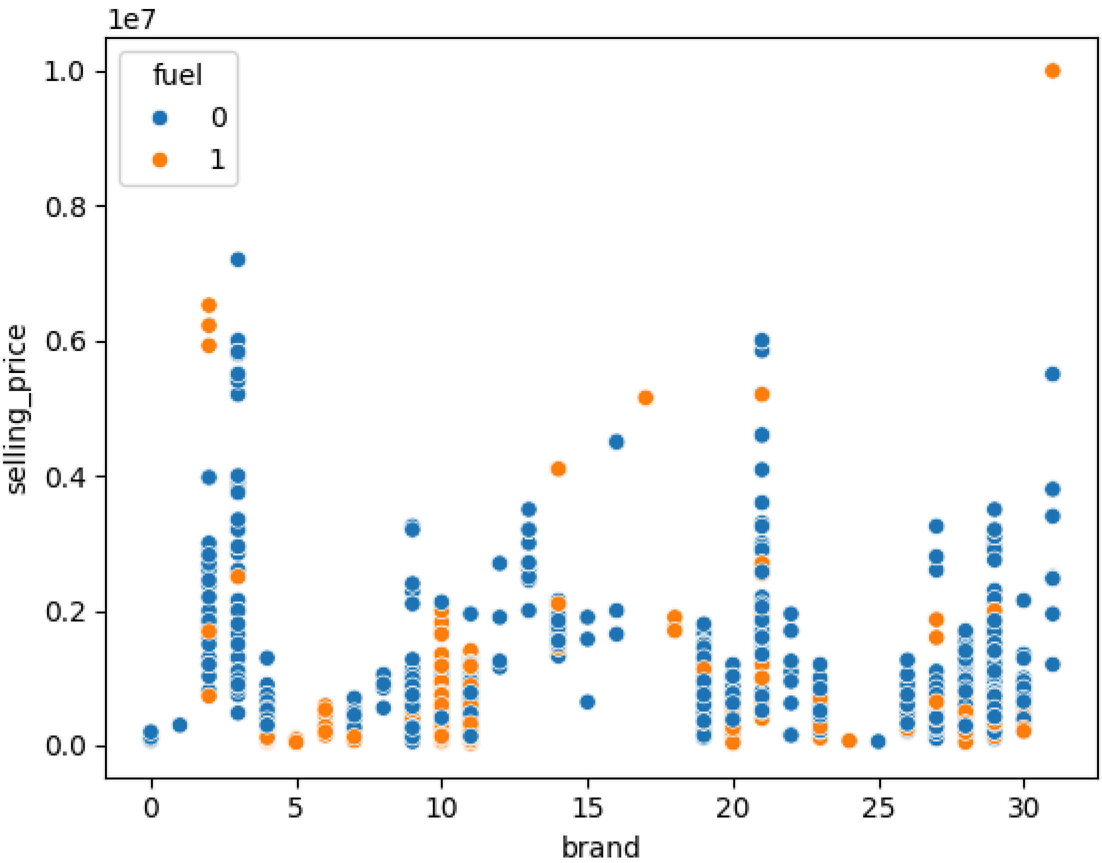
Count
seats
0
1000
2000
3000
4000
5000
6000
2
4
6
8
10
12
14

Multivariate Analysis

Scatterplot

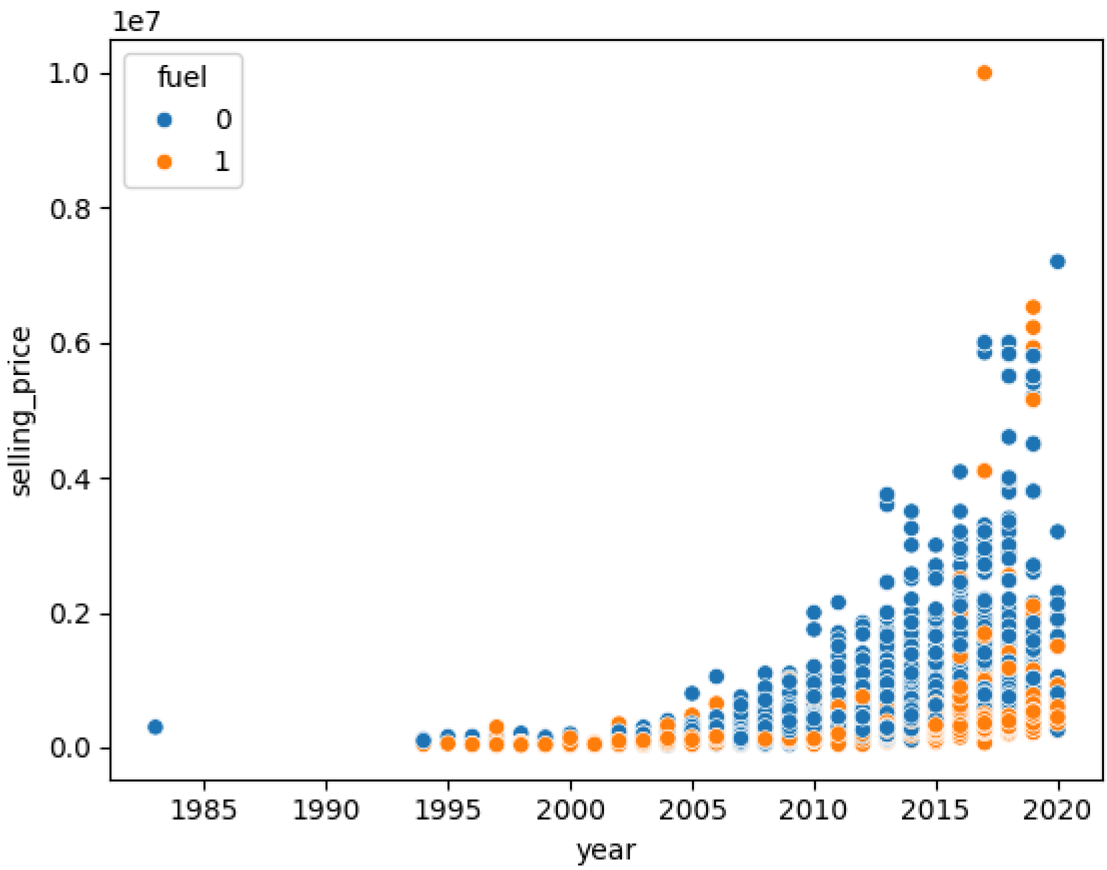
In [450…
```
#Plotting the features in the scatter plot to see the relationship betwee
sns.scatterplot(x=df['brand'], y=df['selling_price'], hue=df['fuel'])
```

Out[450… <Axes: xlabel='brand', ylabel='selling_price'>



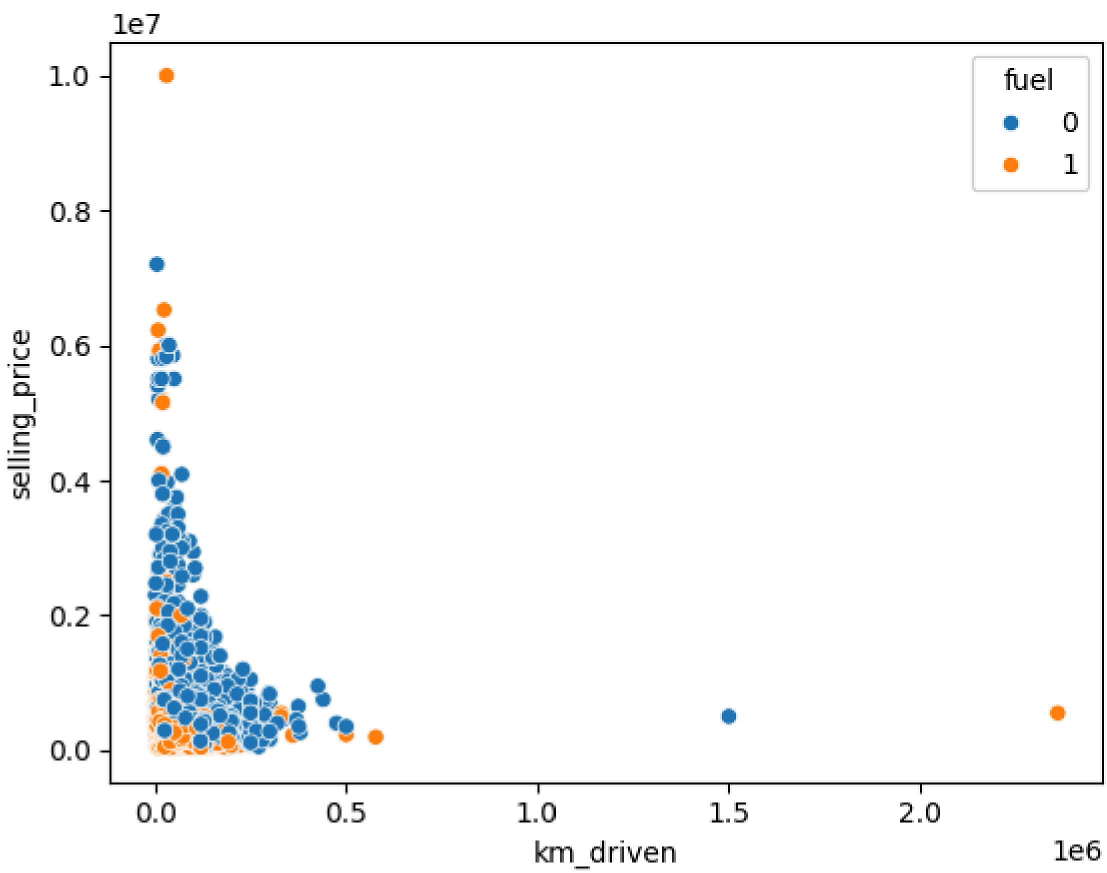
In [451…
```
sns.scatterplot(x=df['year'], y=df['selling_price'], hue=df['fuel'])
```

Out[451… <Axes: xlabel='year', ylabel='selling_price'>



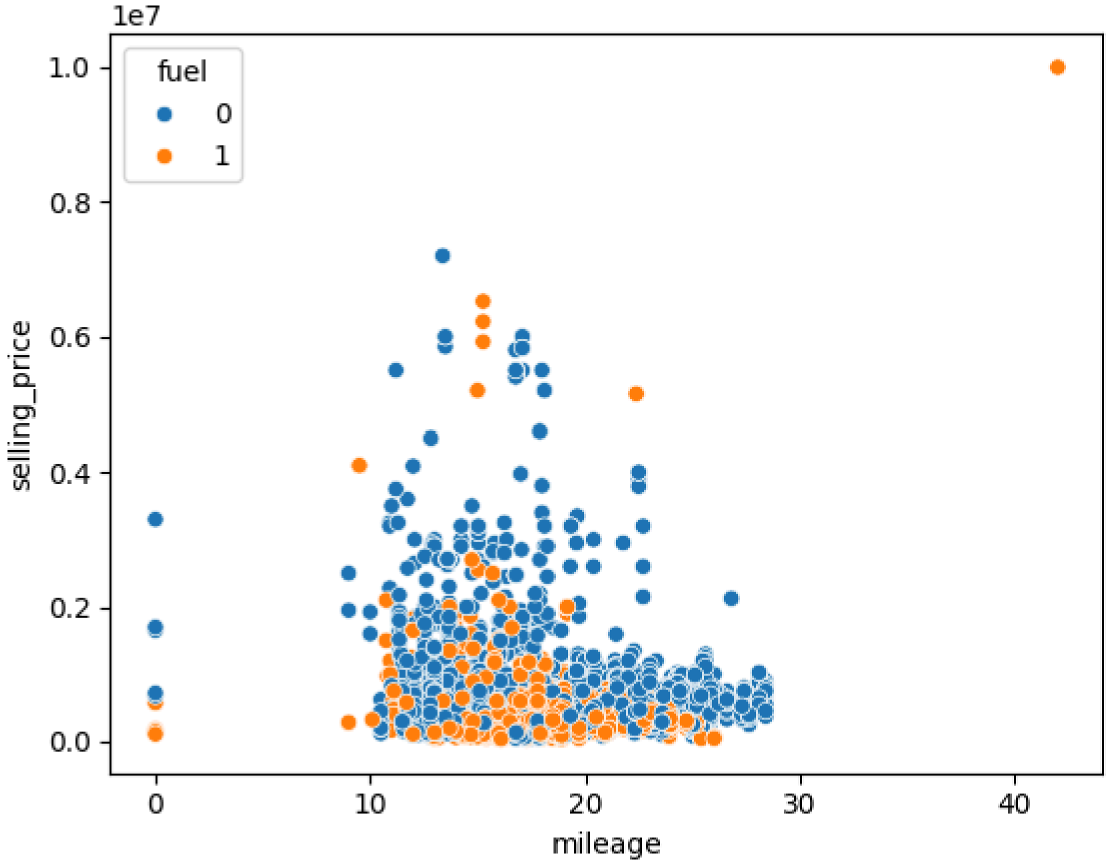
In [452…
```
sns.scatterplot(x=df['km_driven'], y=df['selling_price'], hue=df['fuel'])
```

Out[452…
```
<Axes: xlabel='km_driven', ylabel='selling_price'>
```



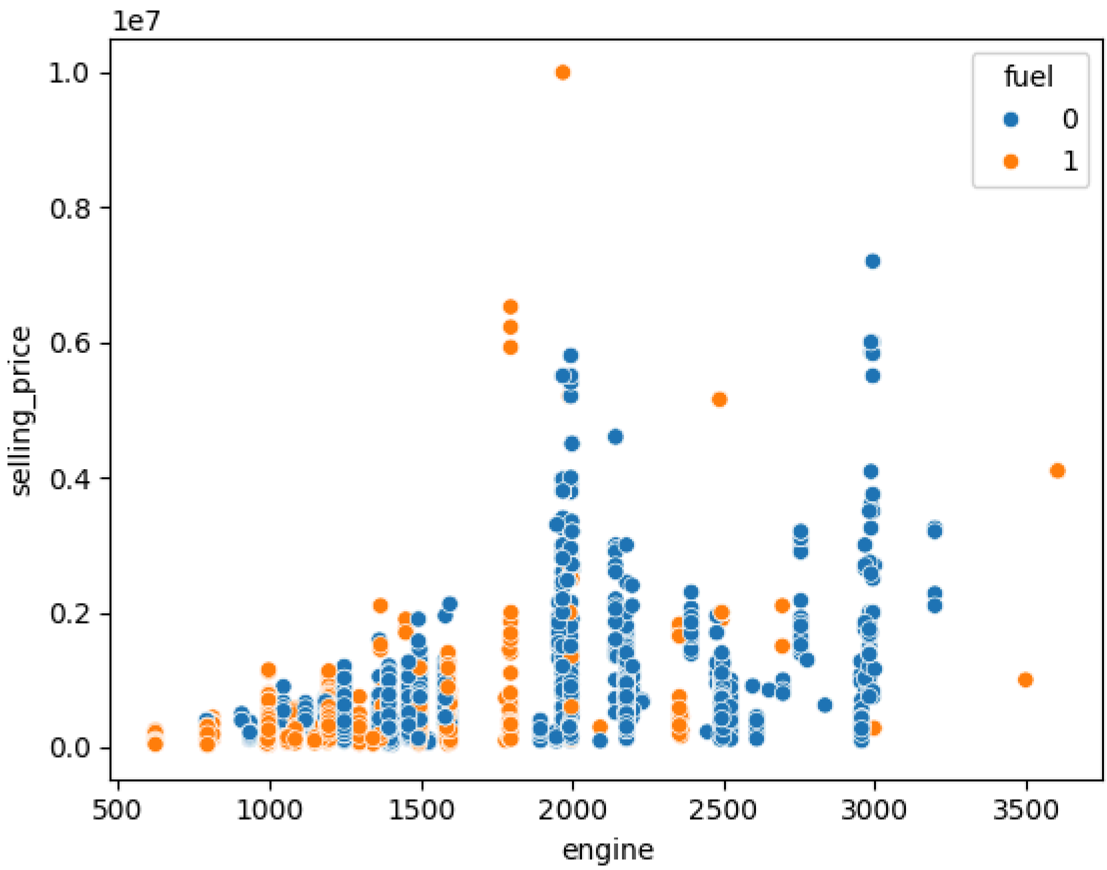
In [453…
```
sns.scatterplot(x=df['mileage'], y=df['selling_price'], hue=df['fuel'])
```

Out[453… <Axes: xlabel='mileage', ylabel='selling_price'>



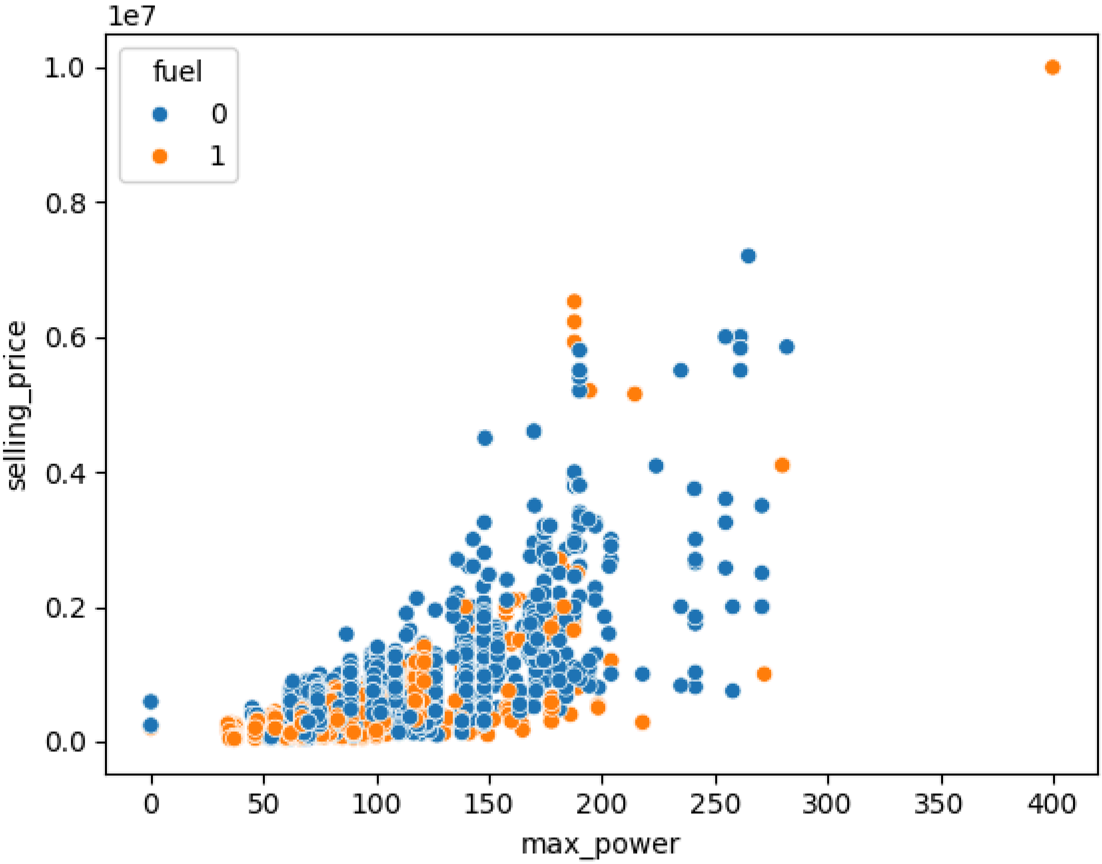
In [454…
```
sns.scatterplot(x=df['engine'], y=df['selling_price'], hue=df['fuel'])
```

Out[454… <Axes: xlabel='engine', ylabel='selling_price'>



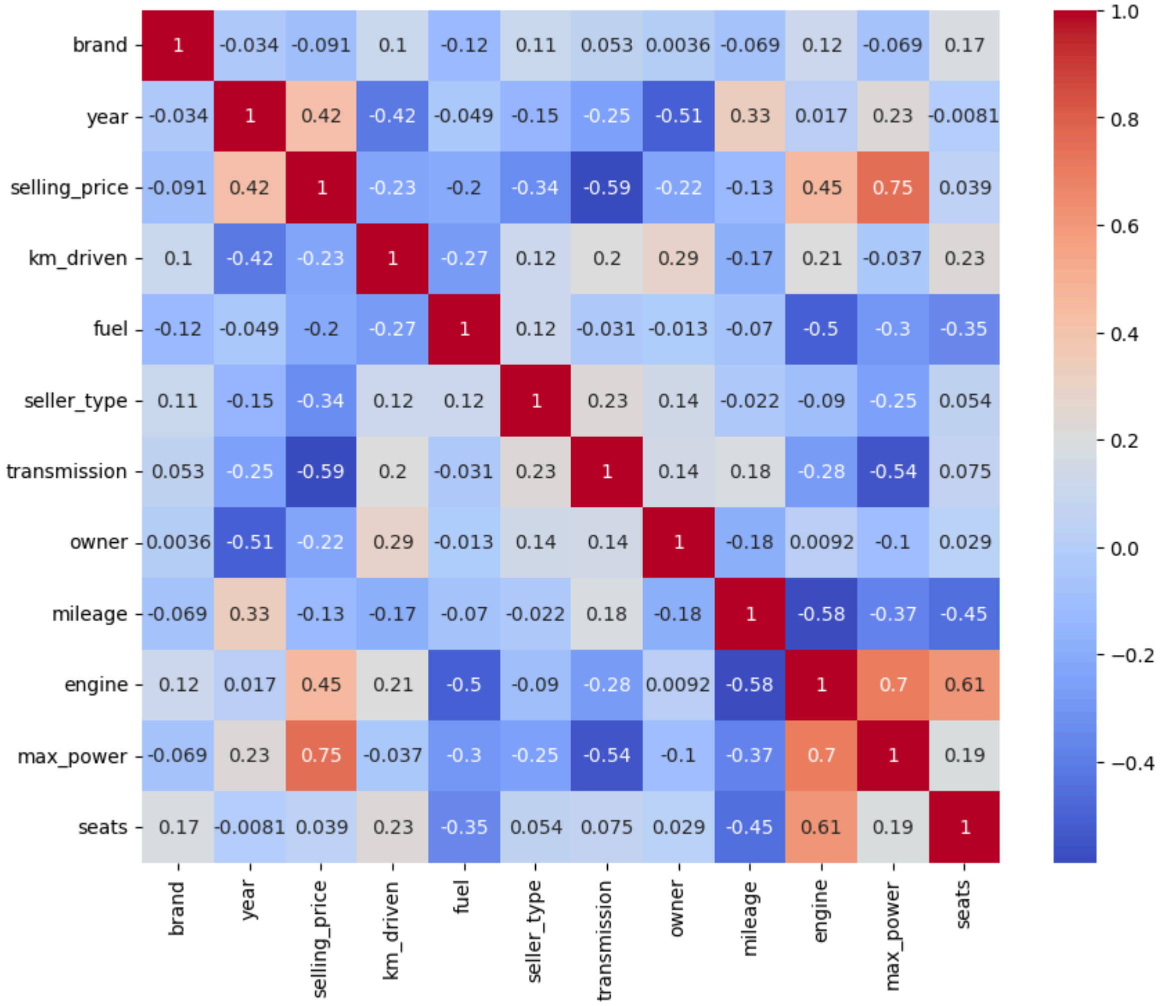
In [455…
```
sns.scatterplot(x=df['max_power'], y=df['selling_price'], hue=df['fuel'])
```

Out[455…
```
<Axes: xlabel='max_power', ylabel='selling_price'>
```



Correlation matrix

In [456…
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Feature Selection

In [457…

```
df['selling_price'] = pd.qcut(x=df['selling_price'], q=4, labels=range(0,
df.head()
```

Out[457…

	brand	year	selling_price	km_driven	fuel	seller_type	transmission	owner
0	20	2014	1	145500	0	1	1	1
1	27	2014	1	120000	0	1	1	2
2	10	2006	0	140000	1	1	1	3
3	11	2010	0	127000	0	1	1	1
4	20	2007	0	120000	1	1	1	1

In [458…

```
#The important features that were selected based on the correlation and p

# Main features = max power, mileage, engine and brand

X = df[['max_power', 'mileage', 'year', 'brand']]

#Selling price is stored in variable Y, which is the target variable. Her
y = df['selling_price']
```

In [459…
```
k = len(set(y))
k
```

Out[459…
```
4
```

Data Splitting

We are splitting the values into train set and test set.

In [460…
```
from sklearn.model_selection import train_test_split
#Splitting the dataset into training and testing sets using train_test_sp
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

Preprocessing

Checking for null values in X_train

In [461…
```
#Checking for null values in the training and testing sets
X_train[['max_power', 'mileage', 'year', 'brand']].isnull().sum()
```

Out[461…
```
max_power    146
mileage      151
year           0
brand          0
dtype: int64
```

In [462…
```
X_test[['max_power', 'mileage', 'year', 'brand']].isnull().sum()
```

Out[462…
```
max_power    62
mileage      63
year          0
brand         0
dtype: int64
```

In [463…
```
#Checking for null values in y_train
y_train.isnull().sum()
```

Out[463…
```
np.int64(0)
```

In [464…
```
#Checking for null values in y_train
y_test.isnull().sum()
```

Out[464…
```
np.int64(0)
```

Finding the mean, median, and mode of the features to fill up the null values

Here only max_power and mileage have null values among the features.

In [465…
```
X_train['max_power'].median() #Finding the median of max_power because th
```

Out[465…
```
np.float64(83.1)
```

In [466…
```
X_train['mileage'].mean() #Finding the mean of mileage because the distri
```

Out[466…
```
np.float64(19.35297697368421)
```

In [467…
```
X_test['max_power'].median()
```

Out[467…
```
np.float64(82.0)
```

In [468…
```
X_test['mileage'].mean()
```

Out[468…
```
np.float64(19.47756710694504)
```

Filling the missing numerical values

Using the obtained median and mean of test and train dataset to fill in the null values.

In [469…
```
X_train['max_power'].fillna(X_train['max_power'].median(), inplace=True)
X_train['mileage'].fillna(X_train['mileage'].mean(), inplace=True)
X_test['max_power'].fillna(X_test['max_power'].median(), inplace=True)
X_test['mileage'].fillna(X_test['mileage'].mean(), inplace=True)
```

In [470…
```
#Verifying if the null values are filled in training set
X_train[['max_power', 'mileage', 'brand', 'year']].isnull().sum()
```

Out[470…
```
max_power    0
mileage      0
brand        0
year         0
dtype: int64
```

In [471…
```
#Verifying if the null values are filled in test set
X_test[['max_power', 'mileage', 'brand', 'year']].isnull().sum()
```

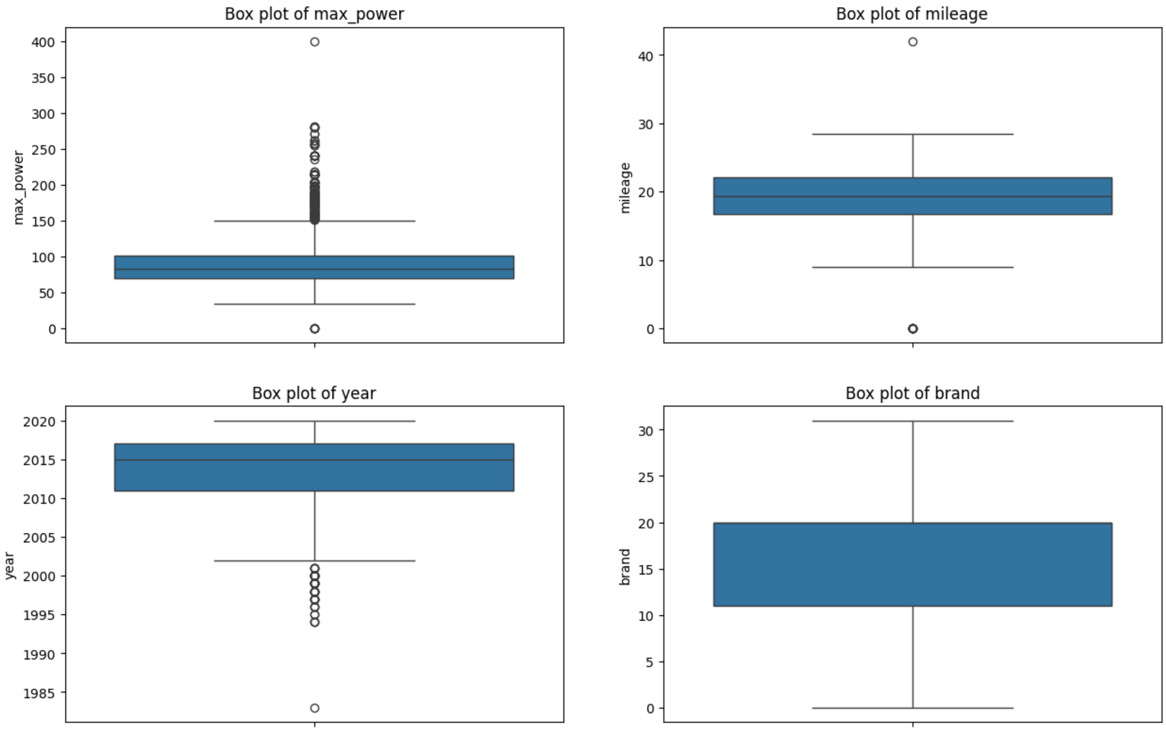
Out[471…
```
max_power    0
mileage      0
brand        0
year         0
dtype: int64
```

In [472…
```
#Verifying if the null values are filled in y_train and y_test
y_train.isnull().sum(), y_test.isnull().sum()
```

Out[472…
```
(np.int64(0), np.int64(0))
```

Checking for outliers

In [473…
```
#Checking the outliers in the training set using box plot
#Creating a dictionary to store the columns and their respective values

col_dict = {'max_power': 1, 'mileage': 2, 'year': 3, 'brand': 4}

#Using boxplot to visualize the outliers in the training set
plt.figure(figsize=(15,20))
for i, col in col_dict.items():
    plt.subplot(4,2,col)
    sns.boxplot(X_train[i])
    plt.title(f'Box plot of {i}')
plt.show()
```



In [474…
```
#To check the impact of outliers on the efficiency of the model, we will

def calculate_outliers(col, data=X_train):
    q75 = np.percentile(data[col], 75)
    q25 = np.percentile(data[col], 25)
    iqr = q75 - q25
    min_val = q25 - (1.5 * iqr)
    max_val = q75 + (1.5 * iqr)

    calculate_outliers = len(np.where((data[col] < min_val) | (data[col]
    percentage_outliers = round(calculate_outliers / len(data) * 100, 2)

    if calculate_outliers > 0:
        print(f'Feature {col} has {calculate_outliers} outliers which is
    else:
        print(f'Feature {col} has no outliers')
```

In [475…
```
for col in X_train.columns:
    calculate_outliers(col)
```

```
Feature max_power has 412 outliers which is 7.33% of the data
Feature mileage has 13 outliers which is 0.23% of the data
Feature year has 55 outliers which is 0.98% of the data
Feature brand has no outliers
```

Since the percentage of the outliers are low in the dataset, we are neglecting them for analysis

Scaling the dataset

In [476…

```
#Importing the library from sklearn for scaling the features, excluding t

from sklearn.preprocessing import StandardScaler
num_cols = ['max_power', 'mileage', 'year']
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

In [477…

```
#Verifying the changes made
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5623, 4)
(2410, 4)
(5623,)
(2410,)
```

In [478…

```
#Encoding of Y_train
Y_train_encoded = pd.get_dummies(y_train)
```

In [479…

```
X_train = X_train.to_numpy()
X_test  = X_test.to_numpy()
Y_train_encoded = Y_train_encoded.to_numpy()
y_test  = y_test.to_numpy()
```

In [480…

```
#Checking the shape
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ", Y_train_encoded.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (5623, 4)
Shape of X_test:  (2410, 4)
Shape of y_train:  (5623, 4)
Shape of y_test:  (2410,)
```

In [481…

```
import time
from sklearn.metrics import classification_report
import mlflow
import os
```

In [482…

```
os.environ["MLFLOW_TRACKING_USERNAME"] = "admin"
os.environ["MLFLOW_TRACKING_PASSWORD"] = "password"
```

In [483…

```
mlflow.set_tracking_uri("https://mlflow.ml.brain.cs.ait.ac.th/")
mlflow.set_experiment("st126222-a3")
```

Out[483…

```
<Experiment: artifact_location='mlflow-artifacts:/442617570942485080', c
reation_time=1759469947454, experiment_id='442617570942485080', last_upd
ate_time=1759469947454, lifecycle_stage='active', name='st126222-a3', ta
gs={'mlflow.experimentKind': 'custom_model_development'}>
```

In [484…

```
class LogisticRegression:

    def __init__(self, regularization, k, n, method, alpha=0.001, max_ite
        self.regularization = regularization
        self.k = int(k)
        self.n = int(n)
        self.alpha = alpha
        self.max_iter = max_iter
        self.method = method
        self.W = None  # Will be initialized later
        self.losses = []

    def fit(self, X, Y):
        self.W = np.random.rand(self.n, self.k)
        params = {
            "reg": type(self).__name__,
            "method": self.method,
            "k": int(self.k),
            "n": int(self.n),
            "alpha": self.alpha,
            "max_iter": self.max_iter
        }
        mlflow.log_params(params=params)

        if self.method == "batch":
            start_time = time.time()
            for i in range(self.max_iter):
                loss, grad = self.gradient(X, Y)
                self.losses.append(loss)
                self.W = self.W - self.alpha * grad
                if i % 500 == 0:
                    print(f"Loss at iteration {i}", loss)
                    mlflow.log_metric(key="train_loss", value=loss, step=
            print(f"time taken: {time.time() - start_time}")

        elif self.method == "minibatch":
            start_time = time.time()
            batch_size = int(0.3 * X.shape[0])
            for i in range(self.max_iter):
                ix = np.random.randint(0, X.shape[0])  # With replacement
                batch_X = X[ix:ix + batch_size]
                batch_Y = Y[ix:ix + batch_size]
                loss, grad = self.gradient(batch_X, batch_Y)
                self.losses.append(loss)
                self.W = self.W - self.alpha * grad
                if i % 500 == 0:
                    print(f"Loss at iteration {i}", loss)
                    mlflow.log_metric(key="train_loss", value=loss, step=
            print(f"time taken: {time.time() - start_time}")

        elif self.method == "sto":
```

```
            start_time = time.time()
            list_of_used_ix = []
            for i in range(self.max_iter):
                idx = np.random.randint(X.shape[0])
                while i in list_of_used_ix:
                    idx = np.random.randint(X.shape[0])
                X_train = X[idx, :].reshape(1, -1)
                Y_train = Y[idx]
                loss, grad = self.gradient(X_train, Y_train)
                self.losses.append(loss)
                self.W = self.W - self.alpha * grad

                list_of_used_ix.append(i)
                if len(list_of_used_ix) == X.shape[0]:
                    list_of_used_ix = []
                if i % 500 == 0:
                    print(f"Loss at iteration {i}", loss)
                    mlflow.log_metric(key="train_loss", value=loss, step=
            print(f"time taken: {time.time() - start_time}")

        else:
            raise ValueError('Method must be one of the followings: "batc

    def gradient(self, X, Y):
        # X: (m, n), Y: (m, k) one-hot
        m = X.shape[0]
        h = self.h_theta(X, self.W)                # (m, k)

        # Safe log to avoid log(0)
        eps = 1e-15
        h_safe = np.clip(h, eps, 1 - eps)

        # Average cross-entropy loss
        loss = -np.sum(Y * np.log(h_safe)) / m

        # Add regularization to the LOSS as well
        if self.regularization:
            loss += self.regularization(self.W) / m

        # Properly averaged gradient
        error = h - Y                              # (m, k)
        grad = (X.T @ error) / m                   # (n, k)

        # Add regularization to the GRAD
        if self.regularization:
            grad += self.regularization.derivation(self.W) / m
        return loss, grad

    def softmax(self, theta_t_x):
        theta_t_x = theta_t_x - np.max(theta_t_x, axis=1, keepdims=True)
        exp_scores = np.exp(theta_t_x)
        return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    def softmax_grad(self, X, error):
        return X.T @ error

    def h_theta(self, X, W):
        return self.softmax(X @ W)

    def predict(self, X_test):
```

```
        return np.argmax(self.h_theta(X_test, self.W), axis=1)

    def plot(self):
        plt.plot(np.arange(len(self.losses)), self.losses, label="Train L
        plt.title("Losses")
        plt.xlabel("epoch")
        plt.ylabel("losses")
        plt.legend()

    def accuracy(self, y_test, y_pred):
        correct_predictions = np.sum(y_test == y_pred)
        total_predictions = y_test.shape[0]
        return correct_predictions / total_predictions

    def precision(self, y_test, y_pred, c=0):
        true_positives = np.sum((y_test == c) & (y_pred == c))
        false_positives = np.sum((y_test != c) & (y_pred == c))
        if true_positives + false_positives == 0:
            return 0
        else:
            return true_positives / (true_positives + false_positives)

    def recall(self, y_test, y_pred, c=0):

        true_positives = np.sum((y_test == c) & (y_pred == c))
        false_negatives = np.sum((y_test == c) & (y_pred != c))
        if true_positives + false_negatives == 0:
            return 0
        else:
            return true_positives / (true_positives + false_negatives)

    def f1_score(self, y_test, y_pred, c=0):

        precision = self.precision(y_test, y_pred, c)
        recall = self.recall(y_test, y_pred, c)
        if precision + recall == 0:
            return 0
        else:
            return 2 * precision * recall / (precision + recall)

    def macro_precision(self, y_test, y_pred):

        precisions = [self.precision(y_test, y_pred, c) for c in range(se
        return np.sum(precisions) / self.k

    def macro_recall(self, y_test, y_pred):

        recalls = [self.recall(y_test, y_pred, c) for c in range(self.k)]
        return np.sum(recalls) / self.k

    def macro_f1(self, y_test, y_pred):

        f1s = [self.f1_score(y_test, y_pred, c) for c in range(self.k)]
        return np.sum(f1s) / self.k

    def weighted_precision(self, y_test, y_pred):

        class_counts = [np.count_nonzero(y_test == c) for c in range(self
        precisions = [class_counts[c] / len(y_test) * self.precision(y_te
        return np.sum(precisions)
```

```
    def weighted_recall(self, y_test, y_pred):

        class_counts = [np.count_nonzero(y_test == c) for c in range(self
        recalls = [class_counts[c] / len(y_test) * self.recall(y_test, y_
        return np.sum(recalls)

    def weighted_f1(self, y_test, y_pred):

        class_counts = [np.count_nonzero(y_test == c) for c in range(self
        f1s = [class_counts[c] / len(y_test) * self.f1_score(y_test, y_pr
        return np.sum(f1s)

    def classification_report(self, y_test, y_pred):

        cols = ["precision", "recall", "f1-score"]
        idx = list(range(self.k)) + ["accuracy", "macro", "weighted"]

        report = [[self.precision(y_test, y_pred, c),
                   self.recall(y_test, y_pred, c),
                   self.f1_score(y_test, y_pred, c)] for c in range(self.

        report.append(["", "", self.accuracy(y_test, y_pred)])

        report.append([self.macro_precision(y_test, y_pred),
                       self.macro_recall(y_test, y_pred),
                       self.macro_f1(y_test, y_pred)])

        report.append([self.weighted_precision(y_test, y_pred),
                       self.weighted_recall(y_test, y_pred),
                       self.weighted_f1(y_test, y_pred)])

        return pd.DataFrame(report, index=idx, columns=cols)
```

In [485…

```
class RidgePenalty:

    def __init__(self, l):
        self.l = l

    def __call__(self, theta):
        return self.l * np.sum(np.square(theta))

    def derivation(self, theta):
        return self.l * 2 * theta

class Ridge(LogisticRegression):

    def __init__(self, l, k, n, method, alpha=0.001, max_iter=5000):
        regularization = RidgePenalty(l)
        super().__init__(regularization, k, n, method, alpha, max_iter)

class Normal(LogisticRegression):

    def __init__(self, k, n, method, alpha=0.001, max_iter=5000):
        super().__init__(regularization=None, k=k, n=n, method=method, al
```

In [486…

```
import sys
```

```
def str_to_class(classname):
    return getattr(sys.modules[__name__], classname)
```

In [354…

```
regs = ["Normal", "Ridge"]
methods = ["batch", "minibatch", "sto"]
alphas = [0.01, 0.001, 0.0001]
best_model = None
best_train_loss = float('inf')
best_reg_name = ""
best_method = ""

for reg in regs:
    for method in methods:
        for alpha in alphas:
            if reg == "Normal":
                params = {"k": k, "n":X_train.shape[1], "method": method,
            else:
                params = {"k": k, "n":X_train.shape[1], "method": method,

            with mlflow.start_run(run_name=f"reg-{reg}-method-{params['me
                print("="*30)
                print(reg, method)
                print(f"alpha: {alpha}")
                print("="*30)
                type_of_regression = str_to_class(reg)    #Normal, Ridge
                model = type_of_regression(**params)
                model.fit(X_train, Y_train_encoded)
                yhat = model.predict(X_test)

                accuracy = model.accuracy(y_test, yhat)
                mlflow.log_metric(key="accuracy", value=accuracy)

                for c in range(k):
                    f1 = model.f1_score(y_test, yhat, c)
                    precision = model.precision(y_test, yhat, c)
                    recall = model.recall(y_test, yhat, c)
                    mlflow.log_metric(key=f"class_{c}_f1", value=f1)
                    mlflow.log_metric(key=f"class_{c}_recall", value=reca
                    mlflow.log_metric(key=f"class_{c}_precision", value=p

                # Get the final training loss
                final_train_loss = model.losses[-1]
                print(f"Final Training Loss: {final_train_loss}")

                # Compare the current model's training loss with the best
                if final_train_loss < best_train_loss:
                    best_train_loss = final_train_loss
                    best_model = model
                    best_reg_name = reg
                    best_method = method

                signature = mlflow.models.infer_signature(X_train, model.
                mlflow.sklearn.log_model(model, name='model', signature=s

        mlflow.end_run()

mlflow.end_run()
```

```
==============================
Normal batch
alpha: 0.01
==============================
Loss at iteration 0 6.917298901322403
Loss at iteration 500 1.0143461901860302
Loss at iteration 1000 0.9299357607137992
Loss at iteration 1500 0.8909880660457434
Loss at iteration 2000 0.8679449034338557
Loss at iteration 2500 0.8526838356632752
Loss at iteration 3000 0.8418716310023614
Loss at iteration 3500 0.8338574077618752
Loss at iteration 4000 0.8277225033431471
Loss at iteration 4500 0.8229118746927121
Loss at iteration 5000 0.8190688430941218
Loss at iteration 5500 0.8159529980620436
Loss at iteration 6000 0.8133960640651283
Loss at iteration 6500 0.8112766220748742
Loss at iteration 7000 0.8095048551949441
Loss at iteration 7500 0.8080129341883356
Loss at iteration 8000 0.8067487321002725
Loss at iteration 8500 0.8056715861081603
Loss at iteration 9000 0.8047493633401864
Loss at iteration 9500 0.8039563827846886
time taken: 14.303831815719604
Final Training Loss: 0.8032731865964094
2025/10/04 14:07:37 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
🏃 View run reg-Normal-method-batch-alpha-0.01 at: https://mlflow.ml.brai
n.cs.ait.ac.th/#/experiments/442617570942485080/runs/bd618604107f4c2e96869
25cd8cda407
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Normal batch
alpha: 0.001
==============================
Loss at iteration 0 4.429272435394426
Loss at iteration 500 1.534130239198065
Loss at iteration 1000 1.4005921487978101
Loss at iteration 1500 1.302760724965179
Loss at iteration 2000 1.2315492381861748
Loss at iteration 2500 1.1786696375458574
Loss at iteration 3000 1.1381853902575918
Loss at iteration 3500 1.1062127047223258
Loss at iteration 4000 1.080252926031754
Loss at iteration 4500 1.0586746260617226
Loss at iteration 5000 1.040384408786265
Loss at iteration 5500 1.0246280018423528
Loss at iteration 6000 1.0108702157951959
Loss at iteration 6500 0.9987212490219259
Loss at iteration 7000 0.9878903632365262
Loss at iteration 7500 0.9781560191771285
Loss at iteration 8000 0.9693461399816586
Loss at iteration 8500 0.9613247572044875
Loss at iteration 9000 0.9539827749314705
Loss at iteration 9500 0.9472314509276526
time taken: 14.081402063369751
Final Training Loss: 0.9410097014802694
```

```
2025/10/04 14:08:05 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
🏃 View run reg-Normal-method-batch-alpha-0.001 at: https://mlflow.ml.brai
n.cs.ait.ac.th/#/experiments/442617570942485080/runs/edfdcc682ebc4a199bdd9
b9f1eed38d4
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Normal batch
alpha: 0.0001
==============================
Loss at iteration 0 5.0628310515969295
Loss at iteration 500 1.7056071149785714
Loss at iteration 1000 1.1866432517143946
Loss at iteration 1500 1.1767684860473273
Loss at iteration 2000 1.1718215540936037
Loss at iteration 2500 1.167012482893219
Loss at iteration 3000 1.1623339018576175
Loss at iteration 3500 1.1577813558379917
Loss at iteration 4000 1.153350517623576
Loss at iteration 4500 1.1490371885658002
Loss at iteration 5000 1.144837299683124
Loss at iteration 5500 1.1407469120111862
Loss at iteration 6000 1.1367622162772821
Loss at iteration 6500 1.1328795319768201
Loss at iteration 7000 1.129095305927807
Loss at iteration 7500 1.1254061103763546
Loss at iteration 8000 1.1218086407221328
Loss at iteration 8500 1.1182997129278134
Loss at iteration 9000 1.1148762606712612
Loss at iteration 9500 1.1115353322937134
time taken: 14.099228858947754
Final Training Loss: 1.1082805323903167
2025/10/04 14:08:34 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Normal-method-batch-alpha-0.0001 at: https://mlflow.ml.bra
in.cs.ait.ac.th/#/experiments/442617570942485080/runs/4d59a5bf5c05477b8753
7b11d8eae5c4
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Normal minibatch
alpha: 0.01
==============================
Loss at iteration 0 10.90744479868169
Loss at iteration 500 1.0280713938704982
Loss at iteration 1000 0.9420780019301521
Loss at iteration 1500 0.8813364847869282
Loss at iteration 2000 0.8558070443586259
Loss at iteration 2500 0.8724673320679531
Loss at iteration 3000 0.8454954103973392
Loss at iteration 3500 0.829464357592458
Loss at iteration 4000 0.8200245059245889
Loss at iteration 4500 0.8368025014831096
Loss at iteration 5000 0.7970344245365778
Loss at iteration 5500 0.7975979653151596
Loss at iteration 6000 0.799381161793619
Loss at iteration 6500 0.833872777050089
Loss at iteration 7000 0.7750621982265061
Loss at iteration 7500 0.8113207171798568
Loss at iteration 8000 0.7958638274609966
Loss at iteration 8500 0.8224639221132657
Loss at iteration 9000 0.8107879424142306
Loss at iteration 9500 0.7997750259381546
time taken: 12.293530225753784
Final Training Loss: 0.8089428695173724
```

```
2025/10/04 14:09:00 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Normal-method-minibatch-alpha-0.01 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/affa6ea96a9a4b42938895d7753ae526
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Normal minibatch
alpha: 0.001
==============================
Loss at iteration 0 5.1327979154524686
Loss at iteration 500 1.33286152545923
Loss at iteration 1000 1.2489763741118283
Loss at iteration 1500 1.1936696647700564
Loss at iteration 2000 1.137996961887521
Loss at iteration 2500 1.131040962250183
Loss at iteration 3000 1.0822537662146994
Loss at iteration 3500 1.067433342213668
Loss at iteration 4000 1.051521771790507
Loss at iteration 4500 1.0113901585865508
Loss at iteration 5000 1.032100504452786
Loss at iteration 5500 1.013592178578278
Loss at iteration 6000 0.999784961744841
Loss at iteration 6500 0.9952340065221146
Loss at iteration 7000 0.9765744215897627
Loss at iteration 7500 0.9460504690716921
Loss at iteration 8000 0.9760493170071984
Loss at iteration 8500 0.9380624917557167
Loss at iteration 9000 0.9515891635940661
Loss at iteration 9500 0.9441968633966522
time taken: 12.551079988479614
Final Training Loss: 0.9230023533309908
```

```
2025/10/04 14:09:27 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Normal-method-minibatch-alpha-0.001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/f8eea23cc7d14d8e9fa8c4fedfdfa5cb
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Normal minibatch
alpha: 0.0001
==============================
Loss at iteration 0 7.772469757383197
Loss at iteration 500 2.213064839868054
Loss at iteration 1000 1.39967963548177
Loss at iteration 1500 1.3732718169910139
Loss at iteration 2000 1.365041771693345
Loss at iteration 2500 1.3597009098283197
Loss at iteration 3000 1.3460935553667859
Loss at iteration 3500 1.340324906311059
Loss at iteration 4000 1.337797897073048
Loss at iteration 4500 1.326818248542427
Loss at iteration 5000 1.3157731880379473
Loss at iteration 5500 1.2892705959312585
Loss at iteration 6000 1.2964375163944966
Loss at iteration 6500 1.2940783446099333
Loss at iteration 7000 1.2726949205760434
Loss at iteration 7500 1.2741180821492724
Loss at iteration 8000 1.265221931771682
Loss at iteration 8500 1.2078911359871818
Loss at iteration 9000 1.2508541937842848
Loss at iteration 9500 1.2401245787754311
time taken: 12.48028016090393
Final Training Loss: 1.239884841063593
```

```
2025/10/04 14:09:53 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Normal-method-minibatch-alpha-0.0001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/a4526992769943869944a7cc445abc63
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Normal sto
alpha: 0.01
==============================
Loss at iteration 0 14.47907600763082
Loss at iteration 500 7.160658441445908
Loss at iteration 1000 0.10367187644855222
Loss at iteration 1500 0.011207901941646939
Loss at iteration 2000 2.013287076567818
Loss at iteration 2500 0.6985559193675035
Loss at iteration 3000 2.8787520534926996
Loss at iteration 3500 9.707146791795244e-05
Loss at iteration 4000 0.0008326373951922771
Loss at iteration 4500 1.043217833368112
Loss at iteration 5000 0.009884232877145845
Loss at iteration 5500 6.204402390588408
Loss at iteration 6000 2.5927103809747694
Loss at iteration 6500 14.497478589038547
Loss at iteration 7000 3.1220885944562324
Loss at iteration 7500 0.7464406231574275
Loss at iteration 8000 1.1216158554108413
Loss at iteration 8500 0.01768534476118212
Loss at iteration 9000 0.089886553856916
Loss at iteration 9500 0.00197000931929626
time taken: 11.776821851730347
Final Training Loss: 5.601577689611762e-07
```

```
2025/10/04 14:10:19 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Normal-method-sto-alpha-0.01 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/186ed2caa9b1443e95df4bd54c43b312
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Normal sto
alpha: 0.001
==============================
Loss at iteration 0 4.898484200549846
Loss at iteration 500 1.9642139947164412
Loss at iteration 1000 1.330328708127214
Loss at iteration 1500 1.203295934464154
Loss at iteration 2000 1.1845541754217526
Loss at iteration 2500 1.6219222492149377
Loss at iteration 3000 1.9402202666259614
Loss at iteration 3500 1.420056018717042
Loss at iteration 4000 1.4326536612239853
Loss at iteration 4500 0.5938870772490652
Loss at iteration 5000 0.669960511973135
Loss at iteration 5500 0.9385892715774695
Loss at iteration 6000 1.7425802741320344
Loss at iteration 6500 1.2364561714547453
Loss at iteration 7000 1.325410825067555
Loss at iteration 7500 1.3619551827708745
Loss at iteration 8000 1.082950555083717
Loss at iteration 8500 2.828943633057848
Loss at iteration 9000 0.516164147639763
Loss at iteration 9500 1.0149590554797272
time taken: 11.283135890960693
Final Training Loss: 1.2038135402002372
```

```
2025/10/04 14:10:45 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Normal-method-sto-alpha-0.001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/ee626e3847654c69b03786dbaa00ae90
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Normal sto
alpha: 0.0001
==============================
Loss at iteration 0 0.6412379263929402
Loss at iteration 500 0.9808376510929507
Loss at iteration 1000 1.708474922573344
Loss at iteration 1500 1.3944912169597803
Loss at iteration 2000 1.0396514706315845
Loss at iteration 2500 1.0663213031123995
Loss at iteration 3000 0.8588906323923475
Loss at iteration 3500 1.1527231711266988
Loss at iteration 4000 1.0055746741933282
Loss at iteration 4500 1.6524775088528025
Loss at iteration 5000 0.5183312689157279
Loss at iteration 5500 1.0642325445257954
Loss at iteration 6000 1.4907762664214823
Loss at iteration 6500 1.403307927721781
Loss at iteration 7000 1.408151856251542
Loss at iteration 7500 1.2225149747798771
Loss at iteration 8000 1.4081416878835435
Loss at iteration 8500 1.063811826457288
Loss at iteration 9000 0.18348028888079787
Loss at iteration 9500 1.3629544872103414
time taken: 11.327151775360107
Final Training Loss: 0.46515843301440074
```

```
2025/10/04 14:11:10 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Normal-method-sto-alpha-0.0001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/96904f5d8a124dd68ab95b9473671220
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Ridge batch
alpha: 0.01
==============================
Loss at iteration 0 8.268645503394069
Loss at iteration 500 0.9983044832854678
Loss at iteration 1000 0.9258591698410275
Loss at iteration 1500 0.8900087511346797
Loss at iteration 2000 0.8679915143114286
Loss at iteration 2500 0.8530965192755062
Loss at iteration 3000 0.84241188633967
Loss at iteration 3500 0.8344354134559457
Loss at iteration 4000 0.8283051021238231
Loss at iteration 4500 0.8234881200832764
Loss at iteration 5000 0.8196364051842971
Loss at iteration 5500 0.8165126831372942
Loss at iteration 6000 0.8139496137898302
Loss at iteration 6500 0.811825861482819
Loss at iteration 7000 0.8100513952400282
Loss at iteration 7500 0.808558097363128
Loss at iteration 8000 0.8072935625633495
Loss at iteration 8500 0.806216887192436
Loss at iteration 9000 0.8052957397631956
Loss at iteration 9500 0.8045042793246384
time taken: 14.282635927200317
Final Training Loss: 0.8038229169778999
```

```
2025/10/04 14:11:39 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Ridge-method-batch-alpha-0.01 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/734ad5f90f5d488ea995d8af74a9ed25
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Ridge batch
alpha: 0.001
==============================
Loss at iteration 0 5.622067063428297
Loss at iteration 500 1.2766175002143592
Loss at iteration 1000 1.2095888332063085
Loss at iteration 1500 1.1592884347862897
Loss at iteration 2000 1.1209450128724217
Loss at iteration 2500 1.0909014700599042
Loss at iteration 3000 1.0666622488122248
Loss at iteration 3500 1.0465852751459568
Loss at iteration 4000 1.0295846181214474
Loss at iteration 4500 1.0149263616242739
Loss at iteration 5000 1.002100721833561
Loss at iteration 5500 0.9907434169617048
Loss at iteration 6000 0.9805869612858764
Loss at iteration 6500 0.9714299472485814
Loss at iteration 7000 0.9631172355463361
Loss at iteration 7500 0.9555268748690867
Loss at iteration 8000 0.9485612608075241
Loss at iteration 8500 0.9421410242492946
Loss at iteration 9000 0.9362007158222947
Loss at iteration 9500 0.9306856971790787
time taken: 14.124035120010376
Final Training Loss: 0.9255597798770612
```

```
2025/10/04 14:12:07 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Ridge-method-batch-alpha-0.001 at: https://mlflow.ml.brai
n.cs.ait.ac.th/#/experiments/442617570942485080/runs/413d2c71f4d243119bfb5
f866e0b10ff
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Ridge batch
alpha: 0.0001
==============================
Loss at iteration 0 8.549323930170276
Loss at iteration 500 1.5248285570885958
Loss at iteration 1000 1.4218118105762656
Loss at iteration 1500 1.4087699581842883
Loss at iteration 2000 1.396160048663823
Loss at iteration 2500 1.3839624952777108
Loss at iteration 3000 1.3721667250012728
Loss at iteration 3500 1.3607618296167514
Loss at iteration 4000 1.3497366371556192
Loss at iteration 4500 1.3390797830872012
Loss at iteration 5000 1.3287797789476243
Loss at iteration 5500 1.3188250771082972
Loss at iteration 6000 1.309204130706716
Loss at iteration 6500 1.2999054481057608
Loss at iteration 7000 1.2909176415768184
Loss at iteration 7500 1.2822294701870718
Loss at iteration 8000 1.2738298770903569
Loss at iteration 8500 1.265708021563281
Loss at iteration 9000 1.2578533061955166
Loss at iteration 9500 1.2502553996482808
time taken: 14.734291076660156
Final Training Loss: 1.2429187179078987
```

```
2025/10/04 14:12:36 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Ridge-method-batch-alpha-0.0001 at: https://mlflow.ml.brai
n.cs.ait.ac.th/#/experiments/442617570942485080/runs/d493277d823f4bbd8c214
65397c44a61
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Ridge minibatch
alpha: 0.01
==============================
Loss at iteration 0 3.2947995847929628
Loss at iteration 500 1.0335598827168548
Loss at iteration 1000 0.9443844171863498
Loss at iteration 1500 0.9065062183502525
Loss at iteration 2000 0.8757239896848216
Loss at iteration 2500 0.8675484896711521
Loss at iteration 3000 0.8643846211388986
Loss at iteration 3500 0.8264557809465825
Loss at iteration 4000 0.8451872390014101
Loss at iteration 4500 0.7147107270074782
Loss at iteration 5000 0.8259656356839437
Loss at iteration 5500 0.836118433593005
Loss at iteration 6000 0.8276204536581032
Loss at iteration 6500 0.790779674219894
Loss at iteration 7000 0.8237387358864362
Loss at iteration 7500 0.8329101206697517
Loss at iteration 8000 0.8196857856060971
Loss at iteration 8500 0.8328473953747925
Loss at iteration 9000 0.8187625375444824
Loss at iteration 9500 0.8170310973474563
time taken: 12.348836183547974
Final Training Loss: 0.8301512720318726
```

```
2025/10/04 14:13:02 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Ridge-method-minibatch-alpha-0.01 at: https://mlflow.ml.br
ain.cs.ait.ac.th/#/experiments/442617570942485080/runs/b137b0e43dc34dbb863
5406d35a60a02
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Ridge minibatch
alpha: 0.001
==============================
Loss at iteration 0 7.145428930213771
Loss at iteration 500 1.423786411447122
Loss at iteration 1000 1.33665789435921
Loss at iteration 1500 1.250840605853631
Loss at iteration 2000 1.1920075251043163
Loss at iteration 2500 1.1488358549447069
Loss at iteration 3000 1.1255257843591324
Loss at iteration 3500 1.0904297474932645
Loss at iteration 4000 1.0541895315563066
Loss at iteration 4500 1.0386160989648023
Loss at iteration 5000 1.0413296788941155
Loss at iteration 5500 1.015754081379649
Loss at iteration 6000 1.0032605598103608
Loss at iteration 6500 0.9786369544244691
Loss at iteration 7000 0.9682373816669821
Loss at iteration 7500 0.9680782072510751
Loss at iteration 8000 0.9525113686243092
Loss at iteration 8500 0.9506828767893704
Loss at iteration 9000 0.9605786893966968
Loss at iteration 9500 0.9542033258303284
time taken: 12.171257972717285
Final Training Loss: 0.9427107191383616
```

```
2025/10/04 14:13:29 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Ridge-method-minibatch-alpha-0.001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/6fded595fe254ff48324d00444d91ffd
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Ridge minibatch
alpha: 0.0001
==============================
Loss at iteration 0 6.898940846807
Loss at iteration 500 2.1444505807908736
Loss at iteration 1000 1.8979632891202418
Loss at iteration 1500 1.9474497531011459
Loss at iteration 2000 1.854317105387652
Loss at iteration 2500 1.8331563097369388
Loss at iteration 3000 1.8194510924059182
Loss at iteration 3500 1.7901594981911684
Loss at iteration 4000 1.8710722140235951
Loss at iteration 4500 1.7946483074391189
Loss at iteration 5000 1.7351028285929366
Loss at iteration 5500 1.7758701991870827
Loss at iteration 6000 1.6943434757208693
Loss at iteration 6500 1.7414862422797854
Loss at iteration 7000 1.6545420906896873
Loss at iteration 7500 1.7408793454261466
Loss at iteration 8000 1.6139964405201868
Loss at iteration 8500 1.6092078104698448
Loss at iteration 9000 1.5969765103711995
Loss at iteration 9500 1.5689564669009395
time taken: 12.157469987869263
Final Training Loss: 1.5507590036248704
```

```
2025/10/04 14:13:56 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Ridge-method-minibatch-alpha-0.0001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/325895b3175f4247ad22fcd62ddd9ae8
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Ridge sto
alpha: 0.01
==============================
Loss at iteration 0 0.9993026317105491
Loss at iteration 500 10.489390018926683
Loss at iteration 1000 1.4203763085942924
Loss at iteration 1500 5.726732909250548
Loss at iteration 2000 2.3486277252351284
Loss at iteration 2500 2.2477786802569995
Loss at iteration 3000 3.88823410399902
Loss at iteration 3500 1.138984485017201
Loss at iteration 4000 1.242009118725569
Loss at iteration 4500 4.202999975494487
Loss at iteration 5000 4.278773766368874
Loss at iteration 5500 2.0671943058798754
Loss at iteration 6000 2.260231515157784
Loss at iteration 6500 0.18673608256621357
Loss at iteration 7000 0.3142998239398418
Loss at iteration 7500 5.371419114452236
Loss at iteration 8000 1.8212181902999802
Loss at iteration 8500 5.568765429315798
Loss at iteration 9000 0.20265882096376966
Loss at iteration 9500 5.670491273863316
time taken: 11.208280086517334
Final Training Loss: 1.7752092087779763
2025/10/04 14:14:21 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
```

```
🏃 View run reg-Ridge-method-sto-alpha-0.01 at: https://mlflow.ml.brain.c
s.ait.ac.th/#/experiments/442617570942485080/runs/8c587752b581458886bd85d8
cd4b1273
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/
442617570942485080
==============================
Ridge sto
alpha: 0.001
==============================
Loss at iteration 0 1.2560601037217949
Loss at iteration 500 1.5205979031746086
Loss at iteration 1000 2.227230594490324
Loss at iteration 1500 1.102088799172141
Loss at iteration 2000 1.0015757811670776
Loss at iteration 2500 1.1552328314964022
Loss at iteration 3000 1.7644323218660778
Loss at iteration 3500 2.037404140650165
Loss at iteration 4000 2.9092928657983084
Loss at iteration 4500 1.5476062178043763
Loss at iteration 5000 1.6020126609419125
Loss at iteration 5500 0.725838685770164
Loss at iteration 6000 1.419686029418791
Loss at iteration 6500 1.2611958616318282
Loss at iteration 7000 0.9477671796727164
Loss at iteration 7500 1.4822704470603114
Loss at iteration 8000 1.4987325001333776
Loss at iteration 8500 0.5080731215941929
Loss at iteration 9000 0.7504516561524677
Loss at iteration 9500 0.61901128117732
time taken: 11.238692045211792
Final Training Loss: 0.3622137868491073
```

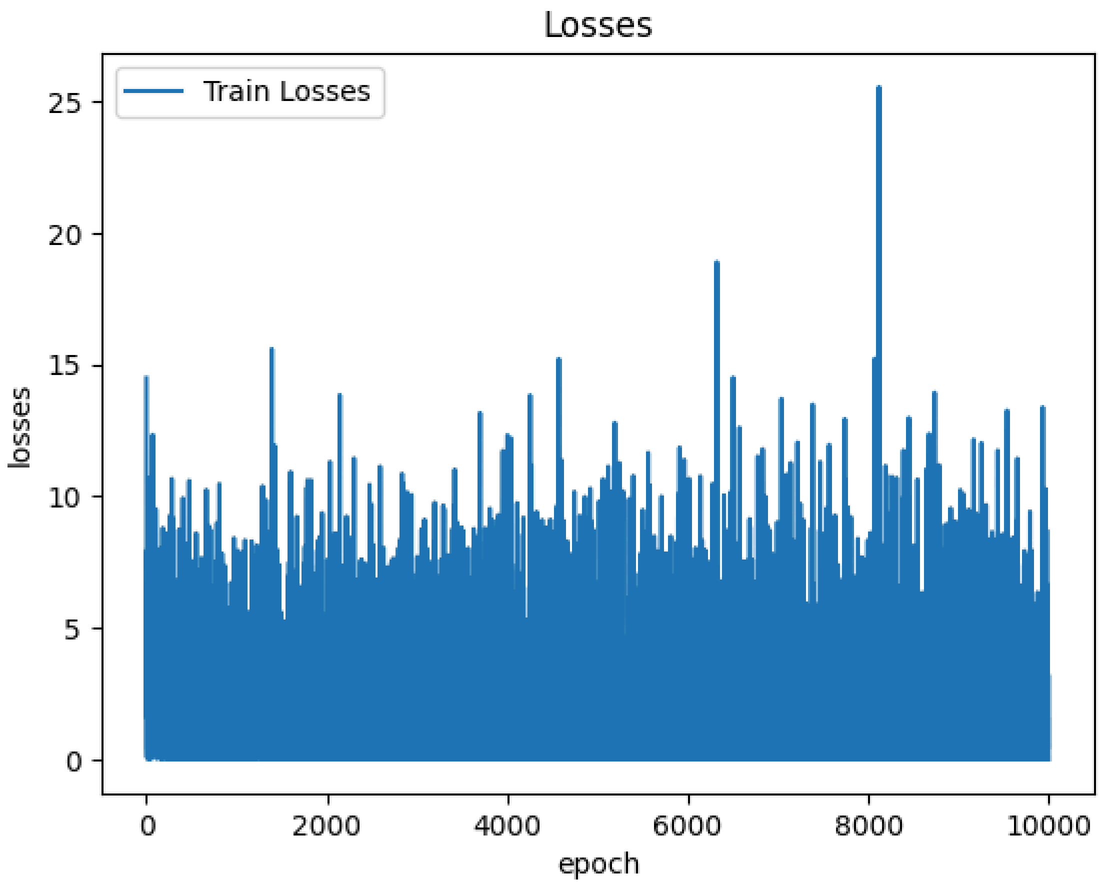
```
2025/10/04 14:14:47 WARNING mlflow.utils.environment: Failed to resolve in
stalled pip version. ``pip`` will be added to conda.yaml environment spec
without a version specifier.
```

```
🏃 View run reg-Ridge-method-sto-alpha-0.001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/f3e590380dbf4f82b45b352857d75a3e
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
==============================
Ridge sto
alpha: 0.0001
==============================
Loss at iteration 0 10.571643340563895
Loss at iteration 500 4.3920423402704865
Loss at iteration 1000 1.8356866504638107
Loss at iteration 1500 1.9561487104064166
Loss at iteration 2000 1.7330803285590592
Loss at iteration 2500 2.0429464576723744
Loss at iteration 3000 1.5463616007931669
Loss at iteration 3500 2.0285284085653776
Loss at iteration 4000 2.0416637900771306
Loss at iteration 4500 1.5759686043837018
Loss at iteration 5000 1.6153383348447015
Loss at iteration 5500 1.8667757406076324
Loss at iteration 6000 1.3651766154416767
Loss at iteration 6500 1.657881396936143
Loss at iteration 7000 1.3576559637138024
Loss at iteration 7500 1.1906085814400995
Loss at iteration 8000 1.9016578033280558
Loss at iteration 8500 1.7280063533487424
Loss at iteration 9000 2.0228815565584934
Loss at iteration 9500 1.3980982490800662
time taken: 11.342077732086182
Final Training Loss: 1.6059627172030972
2025/10/04 14:15:12 WARNING mlflow.utils.environment: Failed to resolve installed pip version. ``pip`` will be added to conda.yaml environment spec without a version specifier.
🏃 View run reg-Ridge-method-sto-alpha-0.0001 at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/2fbcc7fbe95e4e5195a5a18bdd0ea357
🧪 View experiment at: https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080
```

In [487…

```python
# Plotting the loss graph the graph of the best model
print("Best Model:")
print(f"Regularization: {best_reg_name}")
print(f"Method: {best_method}")
print(f"Final Training Loss: {best_train_loss}")
best_model.plot()
plt.show()
```

```
Best Model:
Regularization: Normal
Method: sto
Final Training Loss: 5.601577689611762e-07
```



Configure MLflow for experiment tracking in a Dockerized setup, and setting the experiment name and overriding the default user ID.

In [488…

```
from sklearn.metrics import classification_report as sklearn_classificati

# Assuming best_model is already defined based on the previous code

# Use the best model to predict
yhat = best_model.predict(X_test)

# Custom classification report
custom_classification_report = best_model.classification_report(y_test, y
print("*" * 5, "Custom Classification report", "*" * 5)
print(custom_classification_report)

# Sklearn's classification report
sklearn_report = sklearn_classification_report(y_test, yhat)
print("\n")
print("*" * 5, "Sklearn's Classification report", "*" * 5)
print(sklearn_report)
```

```
***** Custom Classification report *****
          precision    recall  f1-score
0          0.782828   0.72205  0.751212
1          0.494413  0.282748  0.359756
2               0.5  0.001767  0.003521
3          0.394231       1.0  0.565517
accuracy                       0.504979
macro      0.542868  0.501641  0.420002
weighted   0.548935  0.504979  0.429704


***** Sklearn's Classification report *****
              precision    recall  f1-score   support

           0       0.78      0.72      0.75       644
           1       0.49      0.28      0.36       626
           2       0.50      0.00      0.00       566
           3       0.39      1.00      0.57       574

    accuracy                           0.50      2410
   macro avg       0.54      0.50      0.42      2410
weighted avg       0.55      0.50      0.43      2410
```

In [489…

```python
import pickle

filename = 'Model/A3_car_prediction.model'
pickle.dump(model, open(filename, 'wb'))

scaler_path = 'Model/A3_prediction_scalar.model'
pickle.dump(scaler, open(scaler_path, 'wb'))

label_path = 'Model/A3_brand_label.model'
pickle.dump(original_brands, open(label_path, 'wb'))
```

In [490…

```python
print(model)
```

```
<ModelVersion: aliases=[], creation_timestamp=1759571825827, current_stage
='None', deployment_job_state=<ModelVersionDeploymentJobState: current_tas
k_name='', job_id='', job_state='DEPLOYMENT_JOB_CONNECTION_STATE_UNSPECIFI
ED', run_id='', run_state='DEPLOYMENT_JOB_RUN_STATE_UNSPECIFIED'>, descrip
tion='', last_updated_timestamp=1759571825827, metrics=None, model_id=Non
e, name='st126222-a3-model', params=None, run_id='affa6ea96a9a4b42938895d7
753ae526', run_link='', source='models:/m-31174258c90242009ca0654a2dcdf5e
3', status='READY', status_message=None, tags={}, user_id='', version='1'>
```

https://mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/affa6ea96

In [398…

```python
## Registering the model with best run id
import mlflow
from mlflow.tracking import MlflowClient

# MLflow client
client = MlflowClient()

# Your new model name
model_name = "st126222-a3-model"

# Best run info
best_run_id = "affa6ea96a9a4b42938895d7753ae526"
```

```
model_uri = f"runs:/{best_run_id}/model"  # Assuming your model artifact

# Check if the model already exists and delete if needed
try:
    client.delete_registered_model(name=model_name)
    print(f"Deleted existing registered model '{model_name}'")
except Exception as e:
    print(f"No existing model to delete or error: {e}")

# Register the new model
registered_model = mlflow.register_model(model_uri=model_uri, name=model_
print(f"Registered new model '{model_name}' with version: {registered_mod
```

```
Deleted existing registered model 'st126222-a3-model'
Successfully registered model 'st126222-a3-model'.
2025/10/04 16:57:11 WARNING mlflow.tracking._model_registry.fluent: Run wi
th id affa6ea96a9a4b42938895d7753ae526 has no artifacts at artifact path
'model', registering model based on models:/m-31174258c90242009ca0654a2dcd
f5e3 instead
2025/10/04 16:57:11 INFO mlflow.store.model_registry.abstract_store: Waiti
ng up to 300 seconds for model version to finish creation. Model name: st1
26222-a3-model, version 1
Created version '1' of model 'st126222-a3-model'.
Registered new model 'st126222-a3-model' with version: 1
```

In [491…

```
from mlflow.tracking import MlflowClient
import time

client = MlflowClient()
model_name = "st126222-a3-model"
version = 1   # whichever version you just registered

while True:
    mv = client.get_model_version(name=model_name, version=version)
    print(f"Version {version} status: {mv.status}")
    if mv.status == "READY":
        print("✅ Model is READY to use!")
        break
    elif mv.status == "FAILED_REGISTRATION":
        print("❌ Model failed to register")
        break
    else:
        print("⏳ Still waiting...")
        time.sleep(2)
```

```
Version 1 status: READY
✅ Model is READY to use!
```

In [492…

```
from mlflow.tracking import MlflowClient

client = MlflowClient()
for model in client.get_registered_model("st126222-a3-model").latest_vers
    print(model)
    #Find model in staging

    if(model.current_stage == "Staging"):
        version = model.version
        client.transition_model_version_stage(
            name="st126222-a3-model", version = version, stage= "Producti
        )
```

```
<ModelVersion: aliases=[], creation_timestamp=1759571825827, current_stage
='None', deployment_job_state=<ModelVersionDeploymentJobState: current_tas
k_name='', job_id='', job_state='DEPLOYMENT_JOB_CONNECTION_STATE_UNSPECIFI
ED', run_id='', run_state='DEPLOYMENT_JOB_RUN_STATE_UNSPECIFIED'>, descrip
tion='', last_updated_timestamp=1759571825827, metrics=None, model_id=Non
e, name='st126222-a3-model', params=None, run_id='affa6ea96a9a4b42938895d7
753ae526', run_link='', source='models:/m-31174258c90242009ca0654a2dcdf5e
3', status='READY', status_message=None, tags={}, user_id='', version='1'>
```

In [493…

```
from mlflow.tracking import MlflowClient

client = MlflowClient()

# Promote version 1 of your model to Production
client.transition_model_version_stage(
    name="st126222-a3-model",
    version=1,
    stage="Production",
    archive_existing_versions=True
)

print("🚀 Model promoted to Production!")
```

```
---------------------------------------------------------------------------
-
ResponseError                             Traceback (most recent call las
t)
ResponseError: too many 500 error responses

The above exception was the direct cause of the following exception:

MaxRetryError                             Traceback (most recent call las
t)
File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/requests/adapters.py:644,
in HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    643 try:
--> 644     resp = conn.urlopen(
    645         method=request.method,
    646         url=url,
    647         body=request.body,
    648         headers=request.headers,
    649         redirect=False,
    650         assert_same_host=False,
    651         preload_content=False,
    652         decode_content=False,
    653         retries=self.max_retries,
    654         timeout=timeout,
    655         chunked=chunked,
    656     )
    658 except (ProtocolError, OSError) as err:

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/urllib3/connectionpool.p
y:942, in HTTPConnectionPool.urlopen(self, method, url, body, headers, ret
ries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chu
nked, body_pos, preload_content, decode_content, **response_kw)
    941     log.debug("Retry: %s", url)
--> 942     return self.urlopen(
    943         method,
    944         url,
    945         body,
    946         headers,
    947         retries=retries,
    948         redirect=redirect,
    949         assert_same_host=assert_same_host,
    950         timeout=timeout,
    951         pool_timeout=pool_timeout,
    952         release_conn=release_conn,
    953         chunked=chunked,
    954         body_pos=body_pos,
    955         preload_content=preload_content,
    956         decode_content=decode_content,
    957         **response_kw,
    958     )
    960 return response

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/urllib3/connectionpool.p
y:942, in HTTPConnectionPool.urlopen(self, method, url, body, headers, ret
ries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chu
nked, body_pos, preload_content, decode_content, **response_kw)
    941     log.debug("Retry: %s", url)
```

```
--> 942     return self.urlopen(
    943         method,
    944         url,
    945         body,
    946         headers,
    947         retries=retries,
    948         redirect=redirect,
    949         assert_same_host=assert_same_host,
    950         timeout=timeout,
    951         pool_timeout=pool_timeout,
    952         release_conn=release_conn,
    953         chunked=chunked,
    954         body_pos=body_pos,
    955         preload_content=preload_content,
    956         decode_content=decode_content,
    957         **response_kw,
    958     )
    960 return response

    [... skipping similar frames: HTTPConnectionPool.urlopen at line 942
(4 times)]

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/urllib3/connectionpool.p
y:942, in HTTPConnectionPool.urlopen(self, method, url, body, headers, ret
ries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chu
nked, body_pos, preload_content, decode_content, **response_kw)
    941     log.debug("Retry: %s", url)
--> 942     return self.urlopen(
    943         method,
    944         url,
    945         body,
    946         headers,
    947         retries=retries,
    948         redirect=redirect,
    949         assert_same_host=assert_same_host,
    950         timeout=timeout,
    951         pool_timeout=pool_timeout,
    952         release_conn=release_conn,
    953         chunked=chunked,
    954         body_pos=body_pos,
    955         preload_content=preload_content,
    956         decode_content=decode_content,
    957         **response_kw,
    958     )
    960 return response

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/urllib3/connectionpool.p
y:932, in HTTPConnectionPool.urlopen(self, method, url, body, headers, ret
ries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chu
nked, body_pos, preload_content, decode_content, **response_kw)
    931 try:
--> 932     retries = retries.increment(method, url, response=response, _p
ool=self)
    933 except MaxRetryError:

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/urllib3/util/retry.py:51
9, in Retry.increment(self, method, url, response, error, _pool, _stacktra
```

```
ce)
    518     reason = error or ResponseError(cause)
--> 519     raise MaxRetryError(_pool, url, reason) from reason  # type: i
gnore[arg-type]
    521 log.debug("Incremented Retry for (url='%s'): %r", url, new_retry)

MaxRetryError: HTTPSConnectionPool(host='mlflow.ml.brain.cs.ait.ac.th', po
rt=443): Max retries exceeded with url: /api/2.0/mlflow/model-versions/tra
nsition-stage (Caused by ResponseError('too many 500 error responses'))

During handling of the above exception, another exception occurred:

RetryError                                Traceback (most recent call las
t)
File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/utils/rest_utils.p
y:230, in http_request(host_creds, endpoint, method, max_retries, backoff_
factor, backoff_jitter, extra_headers, retry_codes, timeout, raise_on_stat
us, respect_retry_after_header, retry_timeout_seconds, **kwargs)
    229 try:
--> 230     return _get_http_response_with_retries(
    231         method,
    232         url,
    233         max_retries,
    234         backoff_factor,
    235         backoff_jitter,
    236         retry_codes,
    237         raise_on_status,
    238         headers=headers,
    239         verify=host_creds.verify,
    240         timeout=timeout,
    241         respect_retry_after_header=respect_retry_after_header,
    242         **kwargs,
    243     )
    244 except requests.exceptions.Timeout as to:

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/utils/request_util
s.py:237, in _get_http_response_with_retries(method, url, max_retries, bac
koff_factor, backoff_jitter, retry_codes, raise_on_status, allow_redirect
s, respect_retry_after_header, **kwargs)
    235 allow_redirects = env_value if allow_redirects is None else allow_
redirects
--> 237 return session.request(method, url, allow_redirects=allow_redirect
s, **kwargs)

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/requests/sessions.py:589,
in Session.request(self, method, url, params, data, headers, cookies, file
s, auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert, j
son)
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/requests/sessions.py:703,
in Session.send(self, request, **kwargs)
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
```

```
    705 # Total elapsed time of the request (approximately)

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/requests/adapters.py:668,
in HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    667 if isinstance(e.reason, ResponseError):
--> 668     raise RetryError(e, request=request)
    670 if isinstance(e.reason, _ProxyError):

RetryError: HTTPSConnectionPool(host='mlflow.ml.brain.cs.ait.ac.th', port=
443): Max retries exceeded with url: /api/2.0/mlflow/model-versions/transi
tion-stage (Caused by ResponseError('too many 500 error responses'))

During handling of the above exception, another exception occurred:

MlflowException                           Traceback (most recent call las
t)
Cell In[493], line 6
      3 client = MlflowClient()
      5 # Promote version 1 of your model to Production
----> 6 client.transition_model_version_stage(
      7     name=                   ,
      8     version=1,
      9     stage=            ,
     10     archive_existing_versions=True
     11 )
     13 print("🚀 Model promoted to Production!")

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/utils/annotations.
py:186, in deprecated.<locals>.deprecated_decorator.<locals>.deprecated_fu
nc(*args, **kwargs)
    183 @wraps(obj)
    184 def deprecated_func(*args, **kwargs):
    185     warnings.warn(notice, category=FutureWarning, stacklevel=2)
--> 186     return obj(*args, **kwargs)

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/tracking/client.p
y:4595, in MlflowClient.transition_model_version_stage(self, name, versio
n, stage, archive_existing_versions)
   4523 """
   4524 Update model version stage.
   4525 
   (...)   4592     Stage: Staging
   4593 """
   4594 self._raise_if_prompt(name)
-> 4595 return self._get_registry_client().transition_model_version_stage(
   4596     name, version, stage, archive_existing_versions
   4597 )

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/tracking/_model_re
gistry/client.py:332, in ModelRegistryClient.transition_model_version_stag
e(self, name, version, stage, archive_existing_versions)
    330 if stage.strip() == "":
    331     raise MlflowException("The stage must not be an empty strin
g.")
--> 332 return self.store.transition_model_version_stage(
    333     name=name,
```

```
    334     version=version,
    335     stage=stage,
    336     archive_existing_versions=archive_existing_versions,
    337 )

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/store/model_regist
ry/rest_store.py:329, in RestStore.transition_model_version_stage(self, na
me, version, stage, archive_existing_versions)
    305 """
    306 Update model version stage.
    307
   (...)    319
    320 """
    321 req_body = message_to_json(
    322     TransitionModelVersionStage(
    323         name=name,
   (...)    327     )
    328 )
--> 329 response_proto = self._call_endpoint(TransitionModelVersionStage,
req_body)
    330 return ModelVersion.from_proto(response_proto.model_version)

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/store/model_regist
ry/base_rest_store.py:42, in BaseRestStore._call_endpoint(self, api, json_
body, call_all_endpoints, extra_headers)
     40 else:
     41     endpoint, method = self._get_endpoint_from_method(api)
---> 42     return call_endpoint(
     43         self.get_host_creds(), endpoint, method, json_body, respon
se_proto, extra_headers
     44     )

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/utils/rest_utils.p
y:552, in call_endpoint(host_creds, endpoint, method, json_body, response_
proto, extra_headers, retry_timeout_seconds)
    550 else:
    551     call_kwargs["json"] = json_body
--> 552     response = http_request(**call_kwargs)
    554 response = verify_rest_response(response, endpoint)
    555 response_to_parse = response.text

File ~/Library/CloudStorage/OneDrive-AsianInstituteofTechnology/AIT/Machin
e_Learning/A3/.venv/lib/python3.12/site-packages/mlflow/utils/rest_utils.p
y:253, in http_request(host_creds, endpoint, method, max_retries, backoff_
factor, backoff_jitter, extra_headers, retry_codes, timeout, raise_on_stat
us, respect_retry_after_header, retry_timeout_seconds, **kwargs)
    251     raise InvalidUrlException(f"Invalid url: {url}") from iu
    252 except Exception as e:
--> 253     raise MlflowException(f"API request to {url} failed with excep
tion {e}")

MlflowException: API request to https://mlflow.ml.brain.cs.ait.ac.th/api/
2.0/mlflow/model-versions/transition-stage failed with exception HTTPSConn
ectionPool(host='mlflow.ml.brain.cs.ait.ac.th', port=443): Max retries exc
eeded with url: /api/2.0/mlflow/model-versions/transition-stage (Caused by
ResponseError('too many 500 error responses'))
```

In [376...
```
# Best run id
run_id = "affa6ea96a9a4b42938895d7753ae526"

# Build the model URI (artifact path was "model" when you logged it)
model_uri = f"runs:/{run_id}/model"

# Load the model
model = mlflow.pyfunc.load_model(model_uri)
```

In [377...
```
print(type(model))
```

```
<class 'mlflow.pyfunc.PyFuncModel'>
```

In [378...
```
import pandas as pd
precited_selling_price= model.predict(pd.DataFrame(X_test))
precited_selling_price[:10]
```

Out[378...
```
array([2, 2, 2, 1, 3, 2, 3, 3, 3, 1])
```

In [379...
```
y_test[:10]
```

Out[379...
```
array([2, 2, 1, 1, 3, 1, 2, 3, 1, 3])
```

In [385...
```
sample_df = pd.DataFrame([[107, 100,2017, 'BMW']],
                         columns=['max_power', 'mileage', 'year', 'brand'
sample_df
```

Out[385...

	max_power	mileage	year	brand
0	107	100	2017	BMW

In [386...
```
sample_df['brand'] = label_encoder_brand.transform(sample_df['brand'])
```

In [387...
```
sample_df
```

Out[387...

	max_power	mileage	year	brand
0	107	100	2017	3

In [389...
```
sample_df = sample_df.astype("float64")
output = model.predict(sample_df)
print(output)
```

```
[3]
```

Report

The car prediction model is a basic machine learning model which predicts the price of a car based on the values of the features that the user has selected.

The initial data set contained the features like: name(brand), year, km_driven, fuel, setter_type, trainsmission, owner, mileage, engine, max_power, torque, and seats. For the analysis we dropped and cleaned the data first for the analysis. We separated the string from the numerical values for the features fuel, mileage, engine, and maximum power. After performing the initial cleaning, an explanatory data analysis was performed to understand the nature of the features and their interdependency. A univariate analysis was performed using a distribution plot to observe the distribution of the data sets. Scatter plot was used to see the relationship between features and selling price.

A correlation matrix was used to see the relation of the features with the selling price and select the most influential features. After analyzing the correlation matrix, maximum power, mileage, year, and brand were selected as important features for the analysis. After feature selection, the datasets were separated into train and test sets. About 30% of the dataset were separated into the test set. During the preprocessing step, median was used to fill the missing values of the maximum power and mean was used to fill the missing values of mileage. Our test set has no null values of the features that were selected.

Feature Selection

The features that were selected for analysis were as follows:

- Brand: The selling price of a car depends on the brand to some influential extent. The selling price of some luxury brands like BMW, Mercedes,etc. are higher than other brands.
- Year of manufacture: It influences the selling price of the car because newly manufactured cars are sold for higher selling price than old price.
- Maximum power: The cars with high power are priced higher due to their better performance and the use of higher level equipment.
- Mileage: Cars which provide good mileage and better fuel efficiency can increase the demand of the product, and hence create a rise in its price.

The other features like seats, seller type, and owner have lower influence than that of the selected features on the selling price.

Changing selling price to discrete values

The selling price variable was transformed into discrete categories by placing values into four buckets labeled 0, 1, 2, 3, which correspond to [“Cheap”, “Average”, “Expensive”, “Very Expensive”]. Thus, a 4-class classification problem was created.

Models

The LogisticRegression() class was modified to include custom implementations of key classification metrics:

- Accuracy: computed as the ratio of correctly predicted samples to total predictions.
- Precision, Recall, and F1-Score: implemented manually for each class using the standard formulas involving True Positives (TP), False Positives (FP), and False Negatives (FN).
- Macro Precision, Recall, and F1: calculated by averaging the respective metric values across all four classes.
- Weighted Precision, Recall, and F1: computed similarly to macro averages but with weights proportional to class frequencies to handle imbalanced data.

The LogisticRegression() class was further extended to include Ridge (L2) regularization. A new option was added to allow users to toggle the use of this penalty.

Model Training and Evaluation

A grid search–like procedure was implemented to systematically evaluate different regularization techniques, optimization methods, and learning rates (α) for the logistic regression models. The following configurations were explored: • Regularization types: Normal (no penalty) and Ridge (L2 penalty) • Optimization methods: batch, minibatch, and stochastic (sto) gradient descent • Learning rates (α): 0.01, 0.001, and 0.0001

For each combination of regularization, method, and learning rate, a model instance was created and trained using the training dataset (X_train, Y_train_encoded). When Ridge regularization was used, an additional hyperparameter $\lambda = 0.1$ was included to control the strength of the penalty term.

Each experiment was tracked using MLflow, where the following metrics were logged: • Overall accuracy of the model on the test set • Class-specific metrics including precision, recall, and F1-score for each of the four price categories (Cheap, Average, Expensive, Very Expensive) • Final training loss, obtained from the last iteration of the model's loss history

Best Model from MLflow Tracking

From the MLflow experiment tracking, the best-performing model was identified as:

- Model: Logistic Regression (Normal)
- Optimization Method: Mini-batch Gradient Descent
- Learning Rate (α): 0.01
- Regularization: None (Normal)

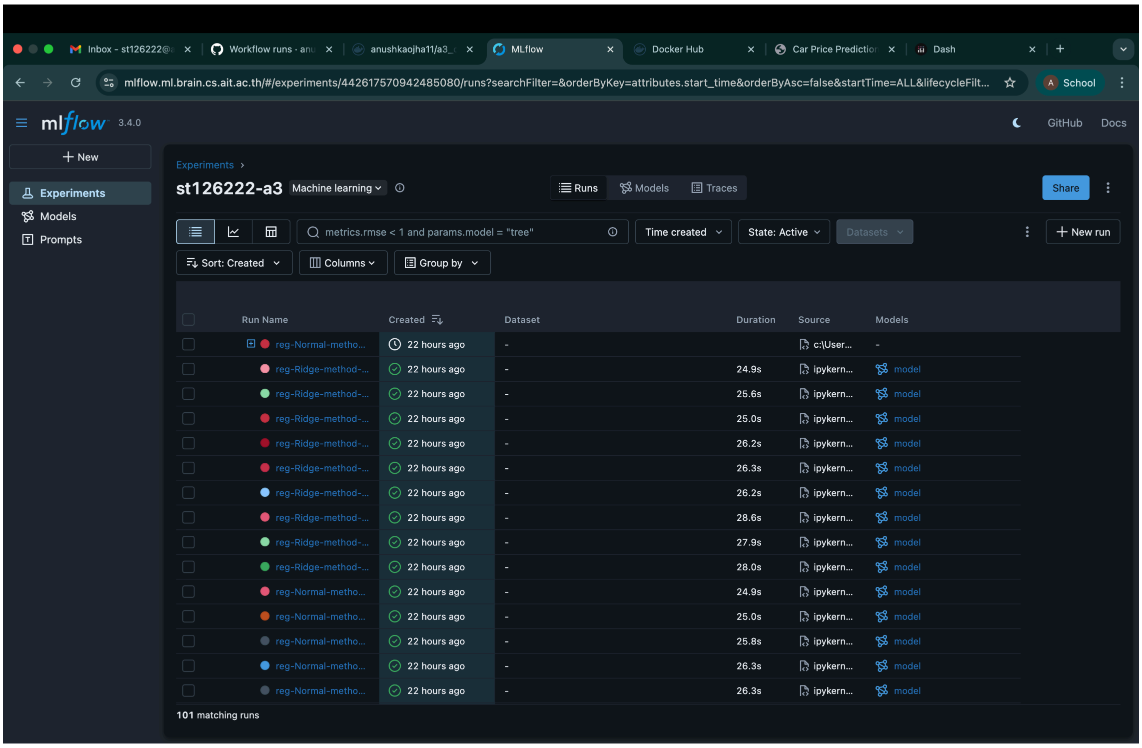
Performance Metrics

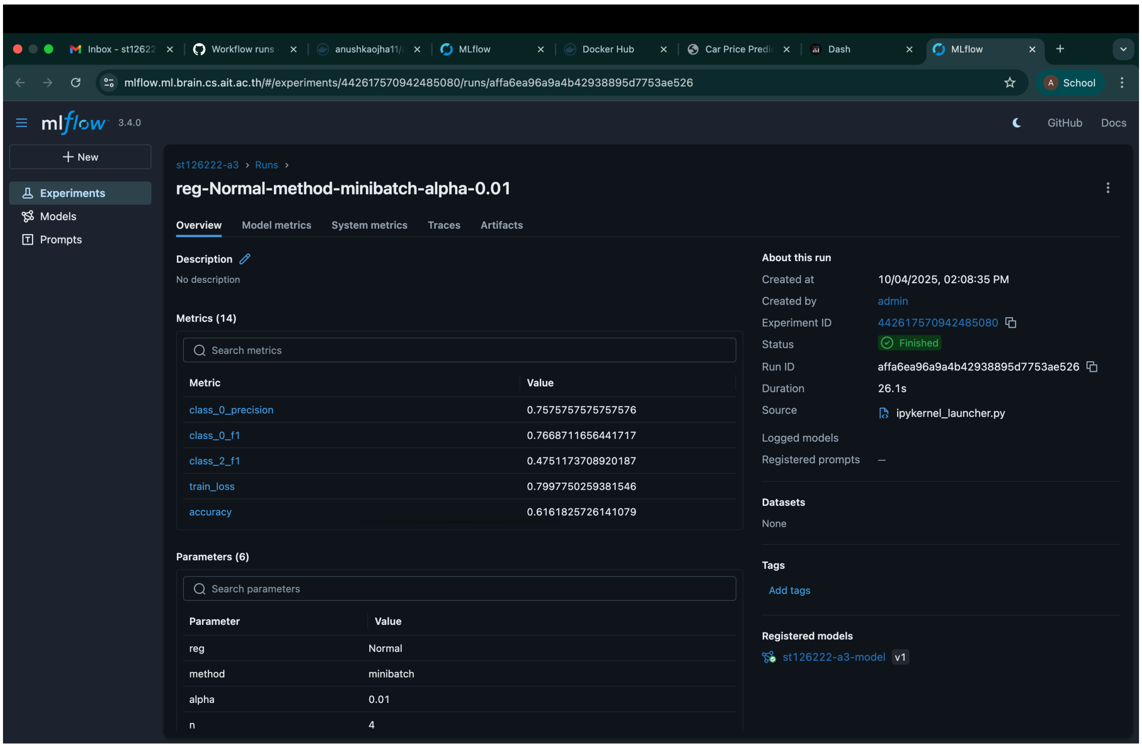
Metric	Value
Accuracy	0.6162
Training Loss	0.7998
Class 0 Precision	0.7576
Class 0 F1-Score	0.7669
Class 2 F1-Score	0.4751

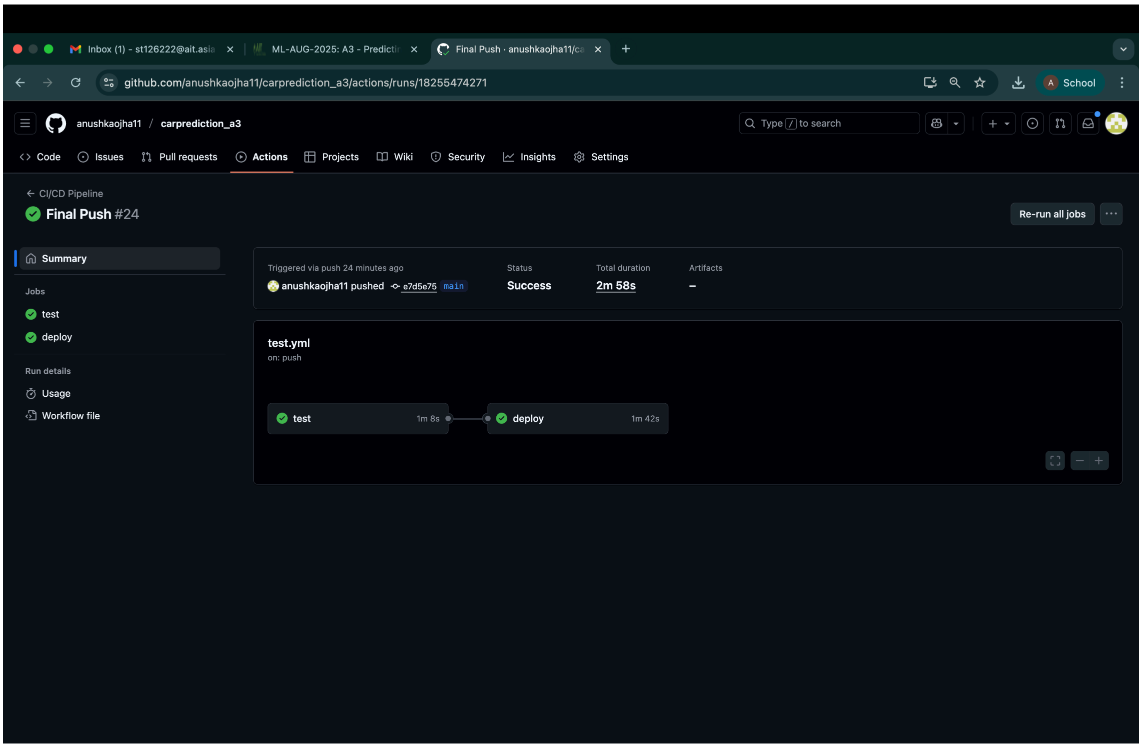
Screenshots

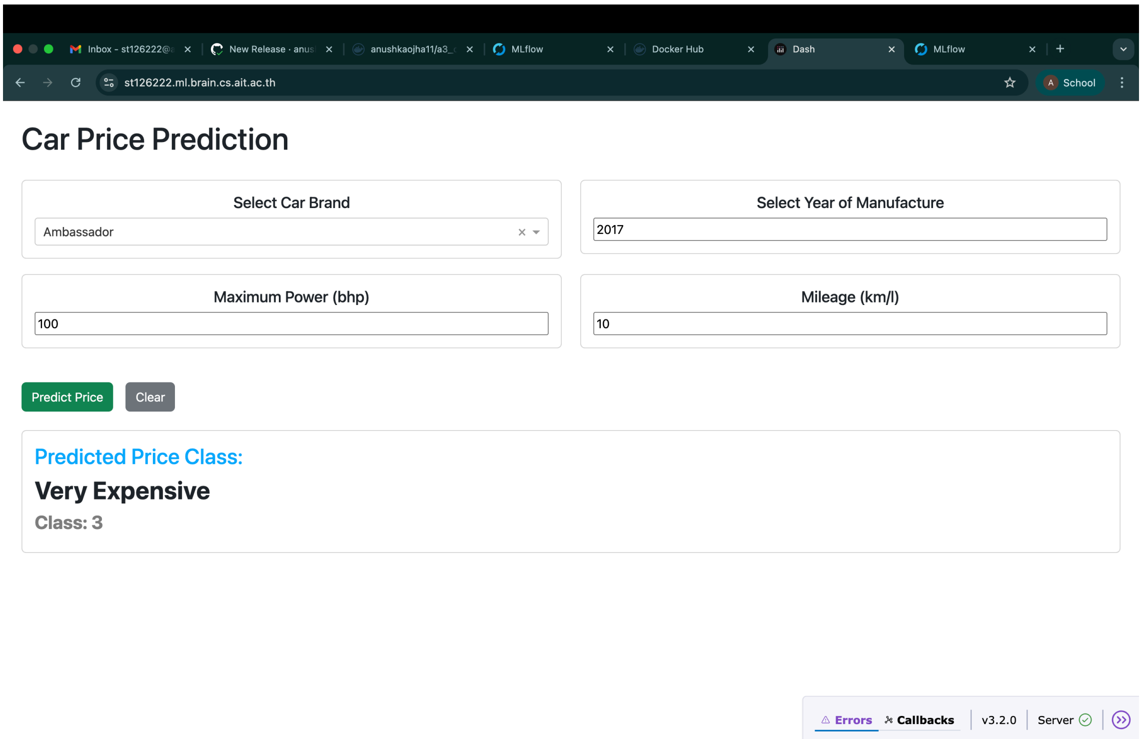
In [497…

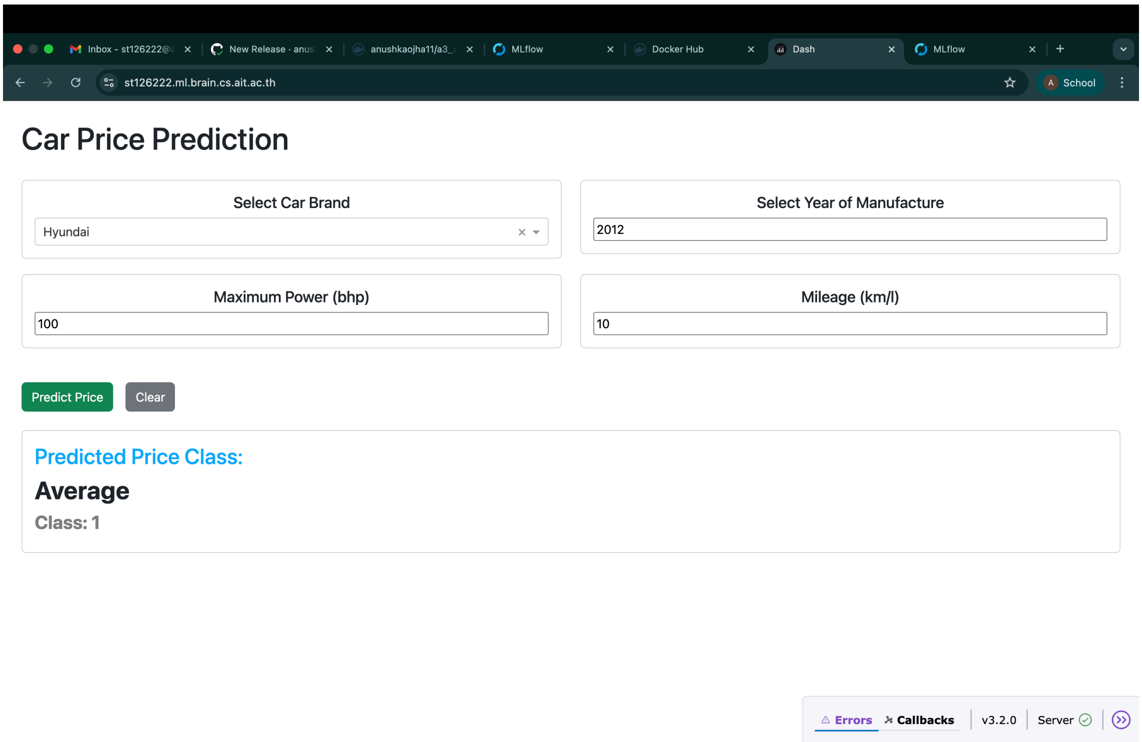
```
import base64
from IPython.display import display, HTML

# List all the image filenames
image_files = [
    "../images/MLflow1.png",
    "../images/MLflow2.png",
    "../images/Deploy1.png",
    "../images/Deploy3.png",
    "../images/Deploy4.png",
    "../images/Deploy5.png"
]

# Generate base64 <img> tags and display them one by one
html_images = ""
for path in image_files:
    with open(path, "rb") as f:
        data = base64.b64encode(f.read()).decode()
    name = path.split("/")[-1].split(".")[0]
    html_images += f'<img src="data:image/png;base64,{data}" alt="{name}"

# Display all images in the notebook
display(HTML(html_images))
```

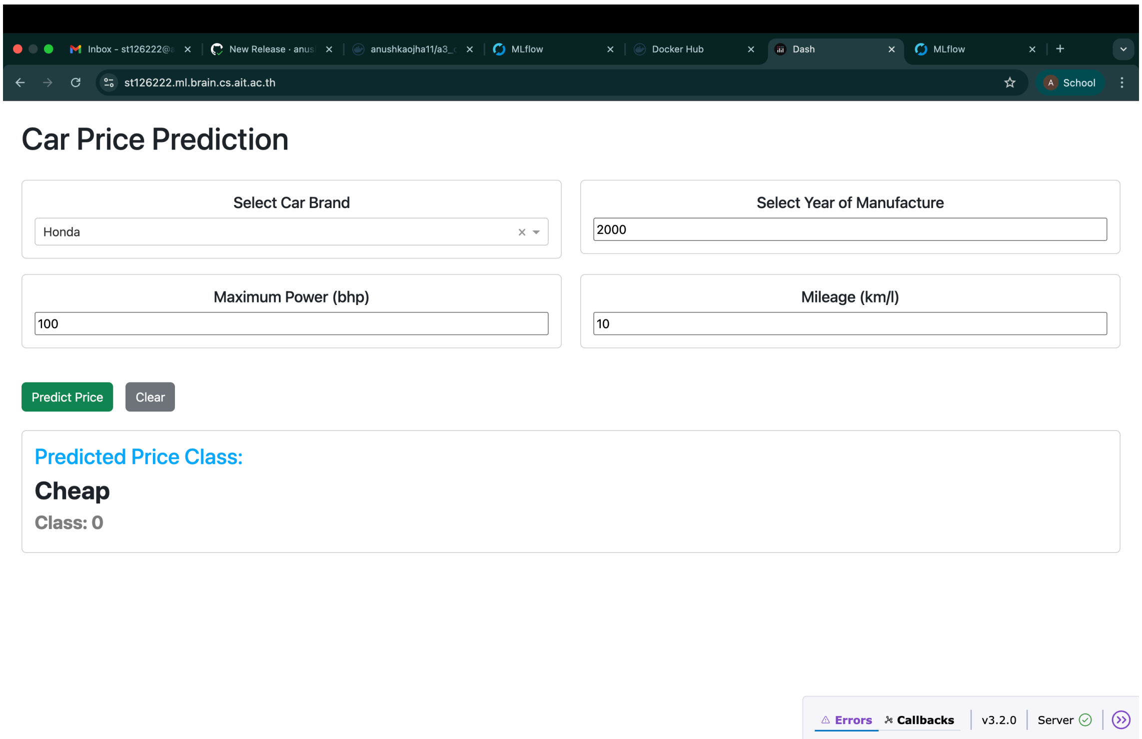

mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs?searchFilter=&orderByKey=attributes.start_time&orderByAsc=false&startTime=ALL&lifecycleFilt...
mlflow 3.4.0
GitHub
Docs
New
Experiments
Models
Prompts
Experiments
st126222-a3
Machine learning
Runs
Models
Traces
Share
metrics.rmse < 1 and params.model = "tree"
Time created
State: Active
Datasets
New run
Sort: Created
Columns
Group by
Run Name
Created
Dataset
Duration
Source
Models
reg-Normal-metho...
22 hours ago
c:\User...
reg-Ridge-method-...
ipykern...
model
24.9s
25.6s
25.0s
26.2s
26.3s
28.6s
27.9s
28.0s
25.8s
101 matching runs


mlflow.ml.brain.cs.ait.ac.th/#/experiments/442617570942485080/runs/affa6ea96a9a4b42938895d7753ae526
mlflow 3.4.0
GitHub
Docs
New
Experiments
Models
Prompts
st126222-a3 › Runs ›
reg-Normal-method-minibatch-alpha-0.01
Overview
Model metrics
System metrics
Traces
Artifacts
Description
No description
Metrics (14)
Search metrics
Metric
Value
class_0_precision
0.7575757575757576
class_0_f1
0.7668711656441717
class_2_f1
0.4751173708920187
train_loss
0.7997750259381546
accuracy
0.6161825726141079
Parameters (6)
Search parameters
Parameter
Value
reg
Normal
method
minibatch
alpha
0.01
n
4
About this run
Created at
10/04/2025, 02:08:35 PM
Created by
admin
Experiment ID
442617570942485080
Status
Finished
Run ID
affa6ea96a9a4b42938895d7753ae526
Duration
26.1s
Source
ipykernel_launcher.py
Logged models
Registered prompts
Datasets
None
Tags
Add tags
Registered models
st126222-a3-model
v1


github.com/anushkaojha11/carprediction_a3/actions/runs/18255474271
anushkaojha11 / carprediction_a3
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
CI/CD Pipeline
Final Push #24
Re-run all jobs
Summary
Jobs
test
deploy
Run details
Usage
Workflow file
Triggered via push 24 minutes ago
anushkaojha11 pushed e7d5e75 main
Status
Success
Total duration
2m 58s
Artifacts
–
test.yml
on: push
test 1m 8s
deploy 1m 42s


st126222.ml.brain.cs.ait.ac.th
Car Price Prediction
Select Car Brand
Ambassador
Select Year of Manufacture
2017
Maximum Power (bhp)
100
Mileage (km/l)
10
Predict Price
Clear
Predicted Price Class:
Very Expensive
Class: 3
Errors
Callbacks
v3.2.0
Server


st126222.ml.brain.cs.ait.ac.th
Car Price Prediction
Select Car Brand
Hyundai
Select Year of Manufacture
2012
Maximum Power (bhp)
100
Mileage (km/l)
10
Predict Price
Clear
Predicted Price Class:
Average
Class: 1
Errors
Callbacks
v3.2.0
Server


st126222.ml.brain.cs.ait.ac.th
Car Price Prediction
Select Car Brand
Honda
Select Year of Manufacture
2000
Maximum Power (bhp)
100
Mileage (km/l)
10
Predict Price
Clear
Predicted Price Class:
Cheap
Class: 0
Errors
Callbacks
v3.2.0
Server