```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import os
import joblib
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import (
    r2_score,
    mean_absolute_error,
    mean_squared_error,
    silhouette_score,
    confusion_matrix,
    ConfusionMatrixDisplay
)

path = '/content/drive/My Drive/Car_Data'
files = os.listdir(path)
print(files)
```

```
['CarStory (1).csv']
```

```
file_path = '/content/drive/My Drive/Car_Data/CarStory (1).csv'
df = pd.read_csv(file_path)
df.head()
```

	Unnamed: 0	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	tax(£)
0	0	Fiesta	2015	6998	Manual	37376	Petrol	125.0	54.3	1.2	NaN
1	1	Astra	2017	10998	Manual	19068	Diesel	150.0	72.4	1.6	NaN
2	2	Grandland X	2019	22995	Automatic	5084	Diesel	145.0	57.7	1.5	NaN
3	3	Yaris	2018	9990	Manual	14615	Petrol	150.0	58.9	1.5	NaN
4	4	5 Series	2019	33900	Semi-Auto	3000	Diesel	145.0	56.5	3.0	NaN

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108540 entries, 0 to 108539
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Unnamed: 0    108540 non-null  int64  
 1   model         108540 non-null  object 
 2   year          108540 non-null  int64  
 3   price         108540 non-null  int64  
 4   transmission  108540 non-null  object 
 5   mileage       108540 non-null  int64  
 6   fuelType      108540 non-null  object 
 7   tax           94327 non-null   float64
 8   mpg           99187 non-null   float64
 9   engineSize    108540 non-null  float64
 10  tax(£)        4860 non-null    float64
dtypes: float64(4), int64(4), object(3)
memory usage: 9.1+ MB
```

	Unnamed: 0	year	price	mileage	tax	mpg	engineSize	tax(£)
count	108540.000000	108540.000000	108540.000000	108540.000000	94327.000000	99187.000000	108540.000000	4860.000000
mean	54269.500000	2017.098028	16890.124046	23025.928469	120.256183	55.166825	1.661644	121.147119
std	31332.943446	2.130057	9756.266820	21176.423684	63.404805	16.138522	0.557058	58.003289
min	0.000000	1970.000000	450.000000	1.000000	0.000000	0.300000	0.000000	0.000000
25%	27134.750000	2016.000000	10229.500000	7491.750000	125.000000	47.100000	1.200000	125.000000
50%	54269.500000	2017.000000	14698.000000	17265.000000	145.000000	54.300000	1.600000	145.000000
75%	81404.250000	2019.000000	20940.000000	32236.000000	145.000000	62.800000	2.000000	145.000000
max	108539.000000	2060.000000	159999.000000	323000.000000	580.000000	470.800000	6.600000	555.000000

```
complete_rows = df.dropna()
if not complete_rows.empty:
    print(complete_rows.head(5))
else:
    print("No rows with complete data found.")
```

```
No rows with complete data found.
```

Here the Idea now is to Understand the data while cleaning it more.

- Assume this is an model to find a prediction of tax based on the car.
- Hence, drop the ID, and model columns.
- There is the year 2060, so that will make the model biased; hence, I will remove it or fix it with ppl in charge (for exam case, I will remove it)
- after looking in 5 complete rows, the tax and tax £ are similar (I thought maybe its a USD or currency difference or there might be a different country tax like having the car in the US tax or UK tax but for simplifying reason I will assume its currency and there is no noticeable difference, so I will fill the missing data from tax £ to tax)

```
df['tax'] = df['tax'].fillna(df['tax(£)'])
df.drop(columns=['tax(£)', 'Unnamed: 0', 'model'], inplace=True)

df.info()
df.head()

#  Fill remaining missing values in 'tax' based on the median of 'engineSize' and 'year'
median_tax_by_group = df.groupby(['engineSize', 'year'])['tax'].median()

def fill_tax(row):
    if pd.isnull(row['tax']):
        return median_tax_by_group.get((row['engineSize'], row['year']), df['tax'].median())
    else:
        return row['tax']

df['tax'] = df.apply(fill_tax, axis=1)

# Verify the final result
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108540 entries, 0 to 108539
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   year          108540 non-null  int64  
 1   price         108540 non-null  int64  
 2   transmission  108540 non-null  object 
 3   mileage       108540 non-null  int64  
 4   fuelType      108540 non-null  object 
 5   tax           99187 non-null   float64
 6   mpg           99187 non-null   float64
 7   engineSize    108540 non-null  float64
dtypes: float64(3), int64(3), object(2)
memory usage: 6.6+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108540 entries, 0 to 108539
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   year          108540 non-null  int64  
 1   price         108540 non-null  int64  
 2   transmission  108540 non-null  object 
 3   mileage       108540 non-null  int64  
 4   fuelType      108540 non-null  object 
 5   tax           108533 non-null  float64
 6   mpg           99187 non-null   float64
 7   engineSize    108540 non-null  float64
dtypes: float64(3), int64(3), object(2)
memory usage: 6.6+ MB
```

	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	2015	6998	Manual	37376	Petrol	125.0	54.3	1.2
1	2017	10998	Manual	19068	Diesel	150.0	72.4	1.6
2	2019	22995	Automatic	5084	Diesel	145.0	57.7	1.5
3	2018	9990	Manual	14615	Petrol	150.0	58.9	1.5
4	2019	33900	Semi-Auto	3000	Diesel	145.0	56.5	3.0

Make sure every thing is clean and Normalise the data

```
df.describe()
```

	year	price	mileage	tax	mpg	engineSize
count	108540.000000	108540.000000	108540.000000	108533.000000	99187.000000	108540.000000
mean	2017.098028	16890.124046	23025.928469	120.650102	55.166825	1.661644
std	2.130057	9756.266820	21176.423684	62.296856	16.138522	0.557058
min	1970.000000	450.000000	1.000000	0.000000	0.300000	0.000000
25%	2016.000000	10229.500000	7491.750000	125.000000	47.100000	1.200000
50%	2017.000000	14698.000000	17265.000000	145.000000	54.300000	1.600000
75%	2019.000000	20940.000000	32236.000000	145.000000	62.800000	2.000000
max	2060.000000	159999.000000	323000.000000	580.000000	470.800000	6.600000

```
from datetime import datetime
current_year = datetime.now().year
df = df[df['year'] <= current_year]
df.describe()
```

	year	price	mileage	tax	mpg	engineSize
count	108539.000000	108539.000000	108539.000000	108532.000000	99186.000000	108539.000000
mean	2017.097633	16890.219820	23025.635661	120.649325	55.166950	1.661646
std	2.126083	9756.260741	21176.301513	62.296617	16.138556	0.557060
min	1970.000000	450.000000	1.000000	0.000000	0.300000	0.000000
25%	2016.000000	10230.000000	7491.500000	125.000000	47.100000	1.200000
50%	2017.000000	14698.000000	17265.000000	145.000000	54.300000	1.600000
75%	2019.000000	20940.000000	32236.000000	145.000000	62.800000	2.000000
max	2020.000000	159999.000000	323000.000000	580.000000	470.800000	6.600000

```
# For 'tax' and 'mpg', fill missing values with the median without inplace
df['tax'] = df['tax'].fillna(df['tax'].median())
df['mpg'] = df['mpg'].fillna(df['mpg'].median())

# 'year', 'price', 'mileage', and 'engineSize' should be numeric and look fine
# 'transmission' and 'fuelType' should be categorical
df['transmission'] = df['transmission'].astype('category')
df['fuelType'] = df['fuelType'].astype('category')

# Standardize text columns for consistency
df['transmission'] = df['transmission'].str.lower()
df['fuelType'] = df['fuelType'].str.lower()

# Verify the result
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 108539 entries, 0 to 108539
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   year          108539 non-null  int64
 1   price         108539 non-null  int64
 2   transmission  108539 non-null  object
 3   mileage       108539 non-null  int64
 4   fuelType      108539 non-null  object
 5   tax           108539 non-null  float64
 6   mpg           108539 non-null  float64
 7   engineSize    108539 non-null  float64
dtypes: float64(3), int64(3), object(2)
memory usage: 7.5+ MB
```

	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	2015	6998	manual	37376	petrol	125.0	54.3	1.2
1	2017	10998	manual	19068	diesel	150.0	72.4	1.6
2	2019	22995	automatic	5084	diesel	145.0	57.7	1.5
3	2018	9990	manual	14615	petrol	150.0	58.9	1.5
4	2019	33900	semi-auto	3000	diesel	145.0	56.5	3.0

```
missing_values = df.isnull().sum()
if missing_values.sum() == 0:
    print("No missing data")
else:
    print("Missing values in each column:")
    print(missing_values[missing_values > 0])

# Save the processed DataFrame to an Excel file in Google Drive
output_path = '/content/drive/My Drive/Processed_Excel_Car.xlsx'
df.to_excel(output_path, index=False)

print("File saved to Google Drive as 'Processed_Excel_Car.xlsx'")
```

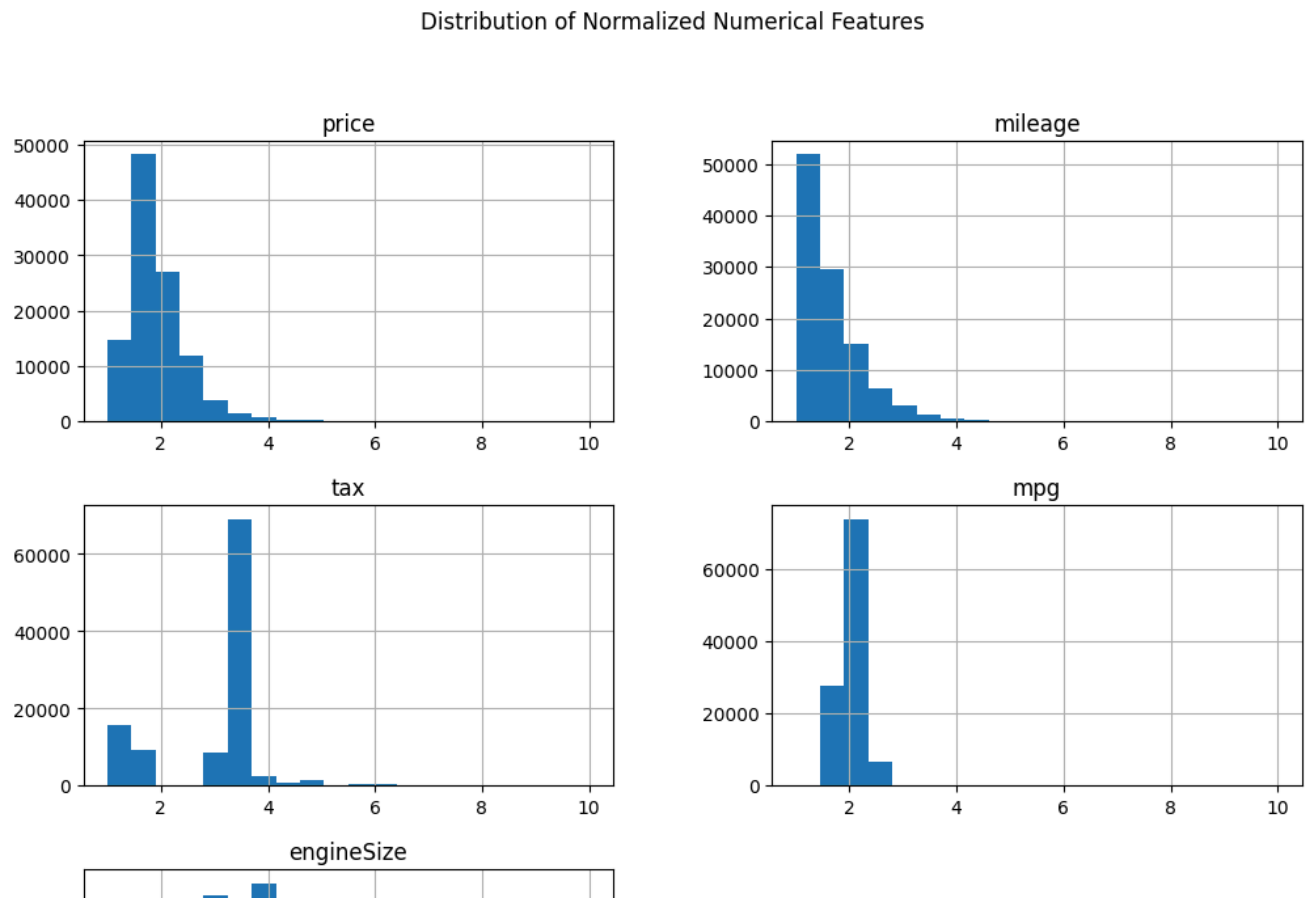
```
No missing data
File saved to Google Drive as 'Processed_Excel_Car.xlsx'
```

```
# Normalise the data form 1 to 10  as  price, mileage, tax, mpg, and engineSize  as I will assume later thae score is from 10 easier to

numeric_cols = ['price', 'mileage','tax','mpg', 'engineSize']

# Initialize the scaler with a feature range from 1 to 10
scaler = MinMaxScaler(feature_range=(1, 10))
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

# Save the normalized DataFrame Drive
output_path = '/content/drive/My Drive/Car_Normalized_1_to_10.xlsx'
df.to_excel(output_path, index=False)

print("Normalized data (1 to 10) saved to Google Drive as 'Car_Normalized_1_to_10.xlsx'")
```

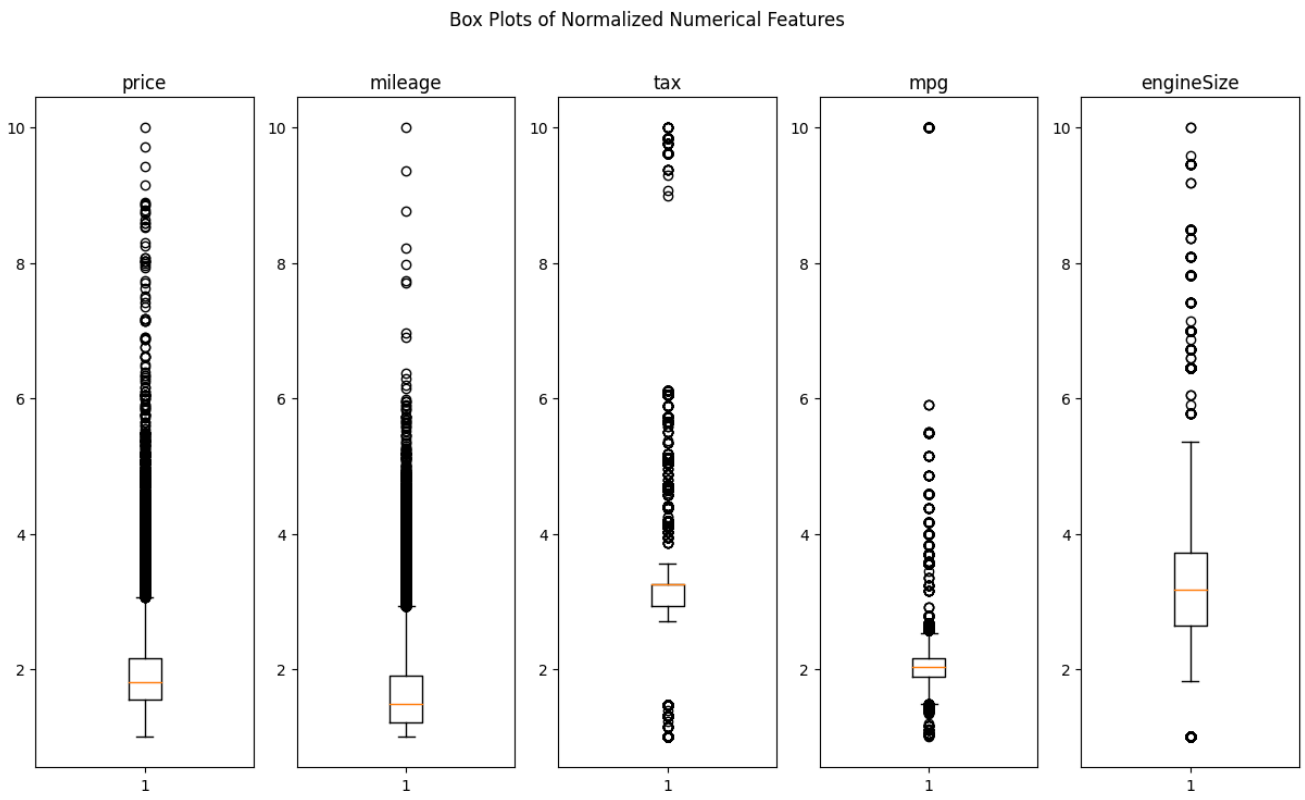
```
Normalized data (1 to 10) saved to Google Drive as 'Car_Normalized_1_to_10.xlsx'
```

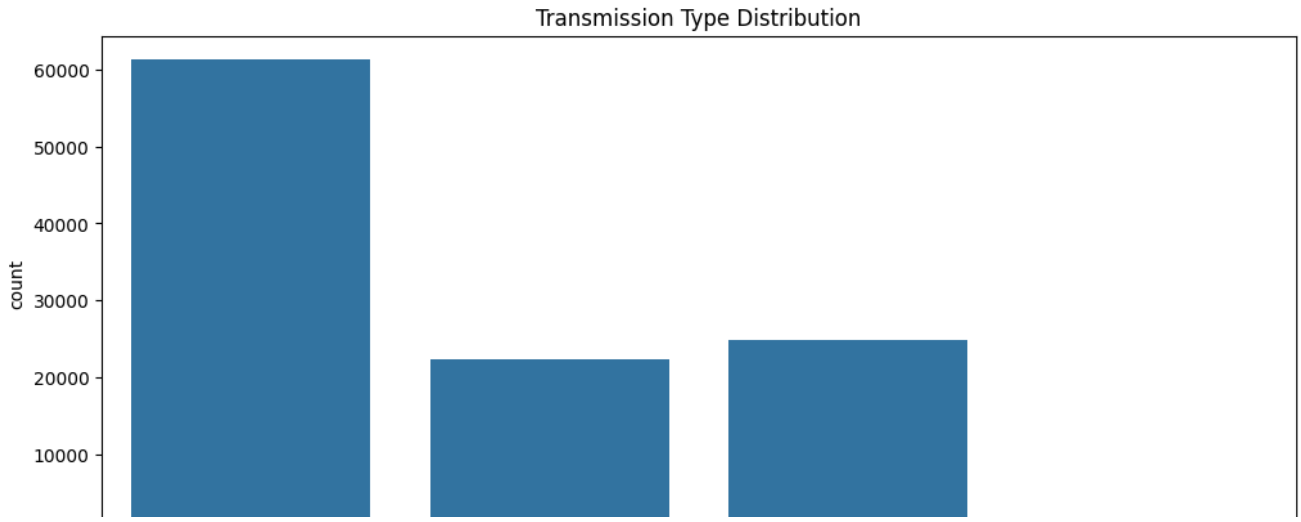
```
df = pd.read_excel('/content/drive/My Drive/Car_Normalized_1_to_10.xlsx')
print(df[['price', 'mileage', 'tax', 'mpg', 'engineSize']].describe())

df[['price', 'mileage', 'tax', 'mpg', 'engineSize']].hist(bins=20, figsize=(12, 10))
plt.suptitle('Distribution of Normalized Numerical Features')
plt.show()
```

```
                price        mileage            tax            mpg  \
count  108539.000000  108539.000000  108539.000000  108539.000000
mean        1.927376       1.641555       2.872169       2.048098
std         0.550341       0.590054       0.966645       0.295144
min         1.000000       1.000000       1.000000       1.000000
25%         1.551680       1.208714       2.939655       1.895218
50%         1.803715       1.481042       3.250000       2.032944
75%         2.155820       1.898192       3.250000       2.168757
max        10.000000      10.000000      10.000000      10.000000

          engineSize
count  108539.000000
mean        3.265881
std         0.759627
min         1.000000
25%         2.636364
50%         3.181818
75%         3.727273
max        10.000000
```

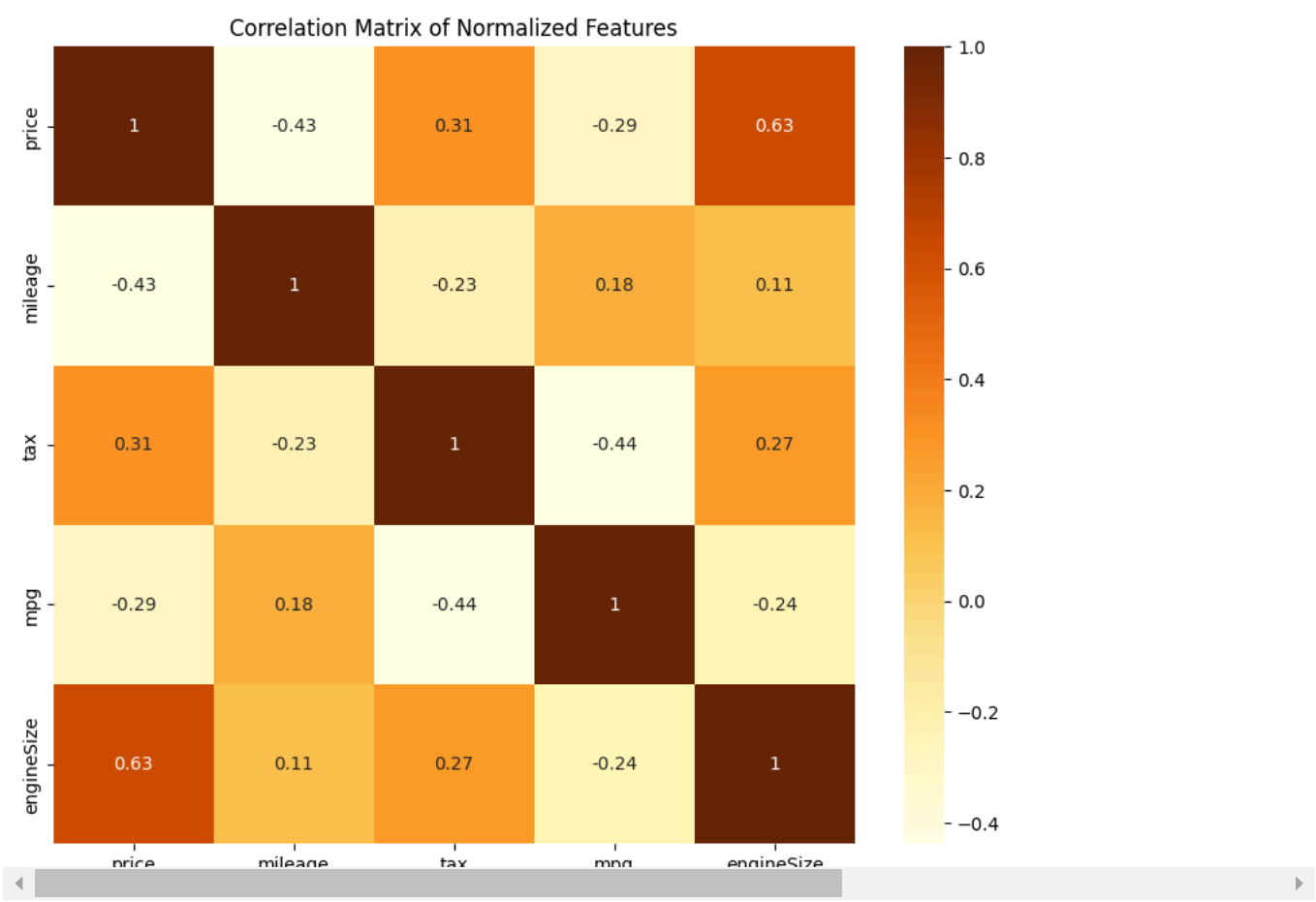



```
plt.figure(figsize=(15, 8))
for i, col in enumerate(['price', 'mileage', 'tax', 'mpg', 'engineSize'], 1):
    plt.subplot(1, 5, i)
    plt.boxplot(df[col].dropna())
    plt.title(col)

plt.suptitle('Box Plots of Normalized Numerical Features')
plt.show()

# Transmission type distribution
plt.figure(figsize=(12, 5))
sns.countplot(data=df, x='transmission')
plt.title('Transmission Type Distribution')
plt.show()

# Fuel type distribution
plt.figure(figsize=(12, 5))
sns.countplot(data=df, x='fuelType')
plt.title('Fuel Type Distribution')
plt.show()
```






```
# Correlation matrix for normalized features with yellow-brown
plt.figure(figsize=(10, 8))
sns.heatmap(df[['price', 'mileage', 'tax', 'mpg', 'engineSize']].corr(), annot=True, cmap='YlOrBr')
plt.title('Correlation Matrix of Normalized Features')
plt.show()
```



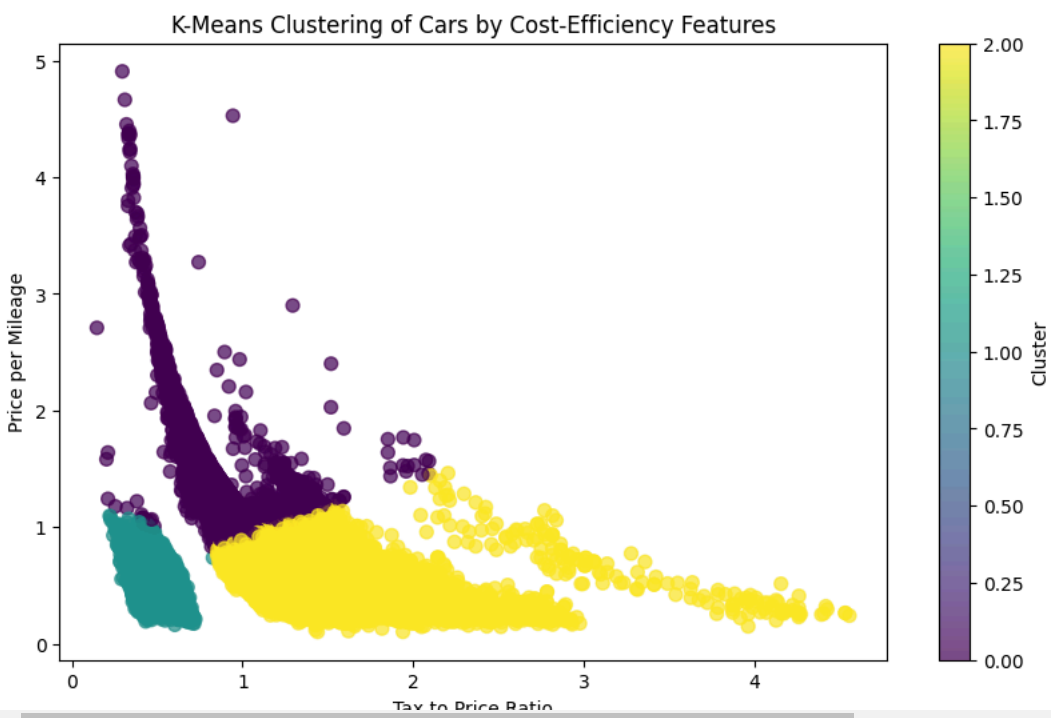

```
# Load the normalized data from Excel
df = pd.read_excel('/content/drive/My Drive/Car_Normalized_1_to_10.xlsx')

# Derive the required features
df['price_per_mileage'] = df['price'] / (df['mileage'] + 1)  # Add 1 to avoid division by zero
df['tax_to_price_ratio'] = df['tax'] / (df['price'] + 1)  # Add 1 to avoid division by zero

# Prepare data for clustering with only the derived features
features_for_clustering = df[['tax_to_price_ratio', 'price_per_mileage']]
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features_for_clustering)



# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=0)  # Adjust n_clusters as needed
kmeans_labels = kmeans.fit_predict(features_scaled)
kmeans_score = silhouette_score(features_scaled, kmeans_labels)

# Store the K-Means results in the DataFrame
df['kmeans_cluster'] = kmeans_labels

print(f"K-Means Silhouette Score: {kmeans_score}")
```

```
K-Means Silhouette Score: 0.5700146570389054
```

```
# Load the normalized data from Excel
"""df = pd.read_excel('/content/drive/My Drive/Car_Normalized_1_to_10.xlsx')

# Derive the required features
df['price_per_mileage'] = df['price'] / (df['mileage'] + 1)  # Add 1 to avoid division by zero
df['tax_to_price_ratio'] = df['tax'] / (df['price'] + 1)  # Add 1 to avoid division by zero

# Prepare data for clustering with only the derived features
features_for_clustering = df[['tax_to_price_ratio', 'price_per_mileage']]
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features_for_clustering)

from sklearn.cluster import DBSCAN

# Apply DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Adjust eps and min_samples as needed
dbscan_labels = dbscan.fit_predict(features_scaled)
```

```
# Filter out noise points (label -1) for silhouette score calculation
filtered_labels = dbscan_labels[dbscan_labels != -1]
filtered_features = features_scaled[dbscan_labels != -1]
dbscan_score = silhouette_score(filtered_features, filtered_labels) if len(set(filtered_labels)) > 1 else -1

# Store the DBSCAN results in the DataFrame
df['dbscan_cluster'] = dbscan_labels

print(f"DBSCAN Silhouette Score: {dbscan_score}")

results_table = pd.DataFrame({
    'Clustering Method': ['K-Means', 'DBSCAN'],
    'Silhouette Score': [kmeans_score, dbscan_score]
})

print("Comparison of Clustering Methods:")
print(results_table)

# Choose the best method based on the highest Silhouette Score
best_method = results_table.sort_values(by="Silhouette Score", ascending=False).iloc[0]['Clustering Method']
print(f"\nBest clustering method based on Silhouette Score: {best_method}")

# Use the chosen method's cluster labels for visualization
df['best_cluster'] = df['kmeans_cluster'] if best_method == 'K-Means' else df['dbscan_cluster']

plt.figure(figsize=(10, 6))
plt.scatter(df['tax_to_price_ratio'], df['price_per_mileage'], c=df['best_cluster'], cmap='viridis', marker='o', s=50, alpha=0.7)
plt.xlabel('Tax to Price Ratio')
plt.ylabel('Price per Mileage')
plt.title(f'Clustering of Cars by {best_method} (Using Derived Features)')
plt.colorbar(label='Cluster')
plt.show()
```

```
# Plot the K-Means Clustering result
plt.figure(figsize=(10, 6))
plt.scatter(df['tax_to_price_ratio'], df['price_per_mileage'], c=df['kmeans_cluster'], cmap='viridis', marker='o', s=50, alpha=0.7)
plt.xlabel('Tax to Price Ratio')
plt.ylabel('Price per Mileage')
plt.title('K-Means Clustering of Cars by Cost-Efficiency Features')
plt.colorbar(label='Cluster')
plt.show()
```




```
df = pd.read_excel('/content/drive/My Drive/Car_Normalized_1_to_10.xlsx')

# Derive additional features for regression
df['price_per_mileage'] = df['price'] / (df['mileage'] + 1)  # Avoid division by zero
df['tax_to_price_ratio'] = df['tax'] / (df['price'] + 1)

# We will use the derived features and other relevant columns for the regression model
```

```
X = df[['price', 'mileage', 'mpg', 'engineSize', 'price_per_mileage', 'tax_to_price_ratio']]
y = df['tax']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
# Evaluate the model performance
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("Linear Regression Performance:")
print(f"R-squared Score: {r2}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
```

```
Linear Regression Performance:
R-squared Score: 0.9742825480133924
Mean Absolute Error (MAE): 0.0917348369520035
Mean Squared Error (MSE): 0.02387230754715723
```

```
df = pd.read_excel('/content/drive/My Drive/Car_Normalized_1_to_10.xlsx')

# Derive additional features for regression
df['price_per_mileage'] = df['price'] / (df['mileage'] + 1)  # Avoid division by zero
df['tax_to_price_ratio'] = df['tax'] / (df['price'] + 1)

# Select features and target variable
X = df[['price', 'mileage', 'mpg', 'engineSize', 'price_per_mileage', 'tax_to_price_ratio']]
y = df['tax']  # Target variable: tax

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Initialize and train the Random Forest Regressor
# n_estimators: Number of trees in the forest (default is 100; try increasing for potentially better accuracy)
# max_depth: Maximum depth of the trees. Setting a lower value can help prevent overfitting.
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("Random Forest Regression Performance:")
print(f"R-squared Score: {r2}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")

model_path = '/content/drive/My Drive/random_forest_model.joblib'
joblib.dump(model, model_path)
print(f"Model saved to Google Drive at '{model_path}'")

''' In case of hyber_para then we can adjust
param_grid_rf = {
    'n_estimators': [50, 100, 200],            # Number of trees
    'max_depth': [None, 10, 20, 30],           # Max depth of each tree
    'min_samples_split': [2, 5, 10],           # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],             # Minimum samples required at a leaf node
    'max_features': ['auto', 'sqrt', 'log2']   # Number of features to consider when looking for the best split
}'''
```

```
Random Forest Regression Performance:
R-squared Score: 0.9974376033012835
Mean Absolute Error (MAE): 0.004513936543338572
Mean Squared Error (MSE): 0.002378552979565614
Model saved to Google Drive at '/content/drive/My Drive/random_forest_model.joblib'
' In case of hyber_para then we can adjust \nparam_grid_rf = {\n    'n_estimators': [50, 100, 200],            # Number of trees\n
'max_depth': [None, 10, 20, 30],           # Max depth of each tree\n    'min_samples_split': [2, 5, 10],           # Minimum sampl
```

Random Forest model are way better LOLZ 💩

```
!pip install flask-ngrok
!pip install Flask
```

```
Collecting flask-ngrok
  Downloading flask_ngrok-0.0.25-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.10/dist-packages (from flask-ngrok) (2.2.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from flask-ngrok) (2.32.3)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (3.0.4)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (8.1.7)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (3.4
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (2024.8.30
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->Flask>=0.8->flask-ngrok
Downloading flask_ngrok-0.0.25-py3-none-any.whl (3.1 kB)
Installing collected packages: flask-ngrok
Successfully installed flask-ngrok-0.0.25
Requirement already satisfied: Flask in /usr/local/lib/python3.10/dist-packages (2.2.5)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask) (3.0.4)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask) (8.1.7)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->Flask) (3.0.2)
```

```
!pip install flask-ngrok
```

```
Requirement already satisfied: flask-ngrok in /usr/local/lib/python3.10/dist-packages (0.0.25)
Requirement already satisfied: Flask>=0.8 in /usr/local/lib/python3.10/dist-packages (from flask-ngrok) (2.2.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from flask-ngrok) (2.32.3)
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (3.0.4)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (3.1.4)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (2.2.0)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (from Flask>=0.8->flask-ngrok) (8.1.7)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (3.4
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->flask-ngrok) (2024.8.30
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=3.0->Flask>=0.8->flask-ngrok
```

```
!pip install pyngrok --quiet
```

```
!ngrok authtoken usr_2o0yg4006WojW7oGSGqbJFWnvwp
```

```
Authtoken saved to configuration file: /root/.config/ngrok/ngrok.yml
```