# Data Preprocessing - Reading Assignment

Name – Maneesha Bogahawatta

Date – 30/08/2025

**Abstract**

This report presents the preprocessing steps applied to a dataset of electric vehicles (EVs) to prepare it for predictive modeling. The dataset contains both numerical features (e.g., battery capacity, range, mileage) and categorical features (e.g., make, model, region, vehicle type). The target variable, **Resale_Value_USD**, represents the resale value of EVs in US dollars. Preprocessing steps included handling missing values, categorizing features, encoding categorical variables, detecting outliers, scaling numerical data, and preparing the final machine-learning-ready dataset. These steps ensure the dataset is clean, consistent, and suitable for regression-based modeling.

**Table of Contents**

## 1. Introduction

The dataset contains detailed information about **electric vehicles (EVs)**, including their technical specifications, performance metrics, operating costs, and categorical attributes such as make, model, region, vehicle type, and usage type.
It includes both **numerical features** (e.g., Battery_Capacity_kWh, Range_km, Mileage_km, Charging_Time_hr) and **categorical features** (e.g., Make, Model, Region, Vehicle_Type, Usage_Type).

The main target variable identified for prediction tasks is:
**Resale_Value_USD**
This represents the resale value of an electric vehicle in US dollars, making this dataset useful for regression-based modeling.

Link to dataset (Kaggle) - Electric Vehicle Analytics Dataset.

## 2. Preprocessing Steps

### 2.1 Handling Missing Values

- Checked the dataset for missing/null values.
- Found that most features had no missing values.
- Any missing values (if present in real-world application) would be handled by either **imputation** (mean/median for numerical, mode for categorical) or by dropping rows with excessive missing data.

```python
# Basic data handling & math
import pandas as pd
import numpy as np

# For preprocessing
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

# For encoding categorical data
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```python
# Load dataset (upload first in Colab or mount Google Drive)
df = pd.read_csv("electric_vehicle_analytics.csv")

# Preview the dataset
print(df.shape)
df.head()
```

(3000, 25)

| | Vehicle_ID | Make | Model | Year | Region | Vehicle_Type | Battery_Capacity_kWh | Battery_Health_% | Range_km | Charging_Power_kW | ... | Max_Speed_kmh | Acceleration_0_100_kmh_sec | Temperature_C | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | Nissan | Leaf | 2021 | Asia | SUV | 101.7 | 75.5 | 565 | 153.6 | ... | 233 | 8.10 | -9.0 | |
| 1 | 2 | Nissan | Leaf | 2020 | Australia | Sedan | 30.1 | 99.8 | 157 | 157.2 | ... | 221 | 9.83 | 1.6 | |
| 2 | 3 | Hyundai | Kona Electric | 2021 | North America | SUV | 118.5 | 84.0 | 677 | 173.6 | ... | 138 | 3.60 | 1.5 | |
| 3 | 4 | Audi | Q4 e-tron | 2022 | Europe | Hatchback | 33.1 | 97.3 | 149 | 169.3 | ... | 192 | 8.97 | 12.5 | |
| 4 | 5 | Tesla | Model 3 | 2022 | Australia | Truck | 81.3 | 85.6 | 481 | 212.8 | ... | 189 | 7.03 | -3.0 | |

5 rows × 25 columns

```python
# Data types & missing values
df.info()

# Summary statistics
df.describe(include="all").T

# Missing value count
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Vehicle_ID                      3000 non-null   int64
 1   Make                            3000 non-null   object
 2   Model                           3000 non-null   object
 3   Year                            3000 non-null   int64
 4   Region                          3000 non-null   object
 5   Vehicle_Type                    3000 non-null   object
 6   Battery_Capacity_kWh            3000 non-null   float64
 7   Battery_Health_%                3000 non-null   float64
 8   Range_km                        3000 non-null   int64
 9   Charging_Power_kW               3000 non-null   float64
 10  Charging_Time_hr                3000 non-null   float64
 11  Charge_Cycles                   3000 non-null   int64
 12  Energy_Consumption_kWh_per_100km 3000 non-null  float64
 13  Mileage_km                      3000 non-null   int64
 14  Avg_Speed_kmh                   3000 non-null   float64
 15  Max_Speed_kmh                   3000 non-null   int64
 16  Acceleration_0_100_kmh_sec      3000 non-null   float64
 17  Temperature_C                   3000 non-null   float64
 18  Usage_Type                      3000 non-null   object
 19  CO2_Saved_tons                  3000 non-null   float64
```

Variables Terminal

```
 20  Maintenance_Cost_USD            3000 non-null   int64
 21  Insurance_Cost_USD              3000 non-null   int64
 22  Electricity_Cost_USD_per_kWh    3000 non-null   float64
 23  Monthly_Charging_Cost_USD       3000 non-null   float64
 24  Resale_Value_USD                3000 non-null   int64
dtypes: float64(11), int64(9), object(5)
memory usage: 586.1+ KB
```

| | 0 |
|---|---|
| Vehicle_ID | 0 |
| Make | 0 |
| Model | 0 |
| Year | 0 |
| Region | 0 |
| Vehicle_Type | 0 |
| Battery_Capacity_kWh | 0 |
| Battery_Health_% | 0 |
| Range_km | 0 |
| Charging_Power_kW | 0 |
| Charging_Time_hr | 0 |
| Charge_Cycles | 0 |
| Energy_Consumption_kWh_per_100km | 0 |
| Mileage_km | 0 |
| Avg_Speed_kmh | 0 |
| Max_Speed_kmh | 0 |
| Acceleration_0_100_kmh_sec | 0 |
| Temperature_C | 0 |
| Usage_Type | 0 |
| CO2_Saved_tons | 0 |
| Maintenance_Cost_USD | 0 |
| Insurance_Cost_USD | 0 |
| Electricity_Cost_USD_per_kWh | 0 |
| Monthly_Charging_Cost_USD | 0 |
| Resale_Value_USD | 0 |

dtype: int64

### 2.2 Feature Categorization

- The dataset contained two types of variables:
  - **Numerical features:** Battery_Capacity_kWh, Battery_Health_%, Range_km, Charging_Power_kW, Mileage_km, Resale_Value_USD, etc.
  - **Categorical features:** Make, Model, Region, Vehicle_Type, Usage_Type.

```
# Separate numerical & categorical columns
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = df.select_dtypes(include=["object"]).columns.tolist()

print("Numerical columns:", num_cols)
print("Categorical columns:", cat_cols)

# Imputation strategy:
# - Numerical: Replace missing with median
# - Categorical: Replace missing with most frequent value

num_imputer = SimpleImputer(strategy="median")
cat_imputer = SimpleImputer(strategy="most_frequent")

df[num_cols] = num_imputer.fit_transform(df[num_cols])
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

# Verify
df.isnull().sum()
```

```
Numerical columns: ['Vehicle_ID', 'Year', 'Battery_Capacity_kWh', 'Battery_Health_%', 'Range_km', 'Charging_Power_kW', 'Charging_Time_hr', 'Charge_Cycles', 'Energy_Consumption_kWh_per_100km',
Categorical columns: ['Make', 'Model', 'Region', 'Vehicle_Type', 'Usage_Type']
```

| | 0 |
|---|---|
| Vehicle_ID | 0 |
| Make | 0 |
| Model | 0 |
| Year | 0 |
| Region | 0 |
| Vehicle_Type | 0 |
| Battery_Capacity_kWh | 0 |
| Battery_Health_% | 0 |
| Range_km | 0 |
| Charging_Power_kW | 0 |
| Charging_Time_hr | 0 |
| Charge_Cycles | 0 |
| Energy_Consumption_kWh_per_100km | 0 |
| Mileage_km | 0 |
| Avg_Speed_kmh | 0 |
| Max_Speed_kmh | 0 |
| Acceleration_0_100_kmh_sec | 0 |
| Temperature_C | 0 |
| Usage_Type | 0 |

```
[5] print("Before removing duplicates:", df.shape)
    df = df.drop_duplicates()
    print("After removing duplicates:", df.shape)
```

```
Before removing duplicates: (3000, 25)
After removing duplicates: (3000, 25)
```

### 2.3 Encoding Categorical Features

- One-hot encoding was applied to categorical columns (Make, Model, Region, Vehicle_Type, Usage_Type).
- This resulted in additional dummy columns such as Make_Tesla, Model_Model 3, Region_Europe, etc.
- One-hot encoding avoids introducing ordinal relationships into non-ordinal categories.

```python
from sklearn.preprocessing import OneHotEncoder

# Use sparse_output=False for newer scikit-learn versions
encoder = OneHotEncoder(drop="first", sparse_output=False)

# Fit and transform categorical columns
encoded = pd.DataFrame(
    encoder.fit_transform(df[cat_cols]),
    columns=encoder.get_feature_names_out(cat_cols)
)

# Drop original categorical columns & join encoded ones
df = df.drop(cat_cols, axis=1).reset_index(drop=True)
df = pd.concat([df, encoded], axis=1)

df.head()
```

| | Vehicle_ID | Year | Battery_Capacity_kWh | Battery_Health_% | Range_km | Charging_Power_kW | Charging_Time_hr | Charge_Cycles | Energy_Consumption_kWh_per_100km | Mileage_km | ... | Model_i4 | Model_iX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 30.99 | 2021.0 | 101.700 | 75.5 | 565.00 | 153.6 | 0.82 | 1438.0 | 12.76 | 117727.0 | ... | 0.0 | 0.0 |
| 1 | 30.99 | 2020.0 | 31.199 | 99.8 | 157.00 | 157.2 | 0.27 | 1056.0 | 15.79 | 161730.0 | ... | 0.0 | 0.0 |
| 2 | 30.99 | 2021.0 | 118.500 | 84.0 | 667.01 | 173.6 | 0.84 | 1497.0 | 24.34 | 244931.0 | ... | 0.0 | 0.0 |
| 3 | 30.99 | 2022.0 | 33.100 | 97.3 | 149.00 | 169.3 | 0.25 | 1613.0 | 14.70 | 57995.0 | ... | 0.0 | 0.0 |
| 4 | 30.99 | 2022.0 | 81.300 | 85.6 | 481.00 | 212.8 | 0.43 | 1078.0 | 22.77 | 17185.0 | ... | 0.0 | 0.0 |

5 rows × 59 columns

### 2.4 Outlier Detection and Treatment

- Numerical features were checked for extreme outliers using **boxplots** and **z-score analysis**.
- Outliers were not removed at this stage, but flagged for later model-based handling if necessary.

```python
[6] # Cap outliers at 1st and 99th percentile for each numeric column
    for col in num_cols:
        lower = df[col].quantile(0.01)
        upper = df[col].quantile(0.99)
        df[col] = np.clip(df[col], lower, upper)
```

### 2.5 Feature Scaling

- Applied **StandardScaler** (z-score normalization) to scale numerical features.
- This ensures features such as Range_km and Charging_Time_hr are on comparable scales, which benefits algorithms like regression, SVMs, and neural networks.

```python
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])

df.head()
```

| | Vehicle_ID | Year | Battery_Capacity_kWh | Battery_Health_% | Range_km | Charging_Power_kW | Charging_Time_hr | Charge_Cycles | Energy_Consumption_kWh_per_100km | Mileage_km | ... | Model_i4 | Model_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1.697882 | 0.526882 | 1.045994 | -1.110270 | 1.394959 | 0.353739 | -0.276295 | 0.648384 | -1.548581 | -0.106322 | ... | 0.0 | |
| 1 | -1.697882 | 0.175705 | -1.696342 | 1.720576 | -1.589695 | 0.406146 | -0.690655 | -0.100005 | -0.743725 | 0.518584 | ... | 0.0 | |
| 2 | -1.697882 | 0.526882 | 1.699478 | -0.120057 | 2.141196 | 0.644888 | -0.261227 | 0.763972 | 1.527405 | 1.700158 | ... | 0.0 | |
| 3 | -1.697882 | 0.878058 | -1.622398 | 1.429337 | -1.648218 | 0.582291 | -0.705723 | 0.991232 | -1.033260 | -0.954602 | ... | 0.0 | |
| 4 | -1.697882 | 0.878058 | 0.252478 | 0.066337 | 0.780472 | 1.215539 | -0.570114 | -0.056905 | 1.110367 | -1.534163 | ... | 0.0 | |

5 rows × 59 columns

### 2.6 Final Dataset Preparation

- After preprocessing, the dataset was transformed into a machine-learning–ready format with:
    - **Cleaned numerical features (scaled).**
    - **One-hot encoded categorical features.**
    - **Target column (Resale_Value_USD).**

```
df.columns
```

```
Index(['Vehicle_ID', 'Year', 'Battery_Capacity_kWh', 'Battery_Health_%',
       'Range_km', 'Charging_Power_kW', 'Charging_Time_hr', 'Charge_Cycles',
       'Energy_Consumption_kWh_per_100km', 'Mileage_km', 'Avg_Speed_kmh',
       'Max_Speed_kmh', 'Acceleration_0_100_kmh_sec', 'Temperature_C',
       'CO2_Saved_tons', 'Maintenance_Cost_USD', 'Insurance_Cost_USD',
       'Electricity_Cost_USD_per_kWh', 'Monthly_Charging_Cost_USD',
       'Resale_Value_USD', 'Make_BMW', 'Make_Chevrolet', 'Make_Ford',
       'Make_Hyundai', 'Make_Kia', 'Make_Mercedes', 'Make_Nissan',
       'Make_Tesla', 'Make_Volkswagen', 'Model_Bolt EUV', 'Model_Bolt EV',
       'Model_EQC', 'Model_EQS', 'Model_EV6', 'Model_F-150 Lightning',
       'Model_ID.3', 'Model_ID.4', 'Model_Ioniq 5', 'Model_Kona Electric',
       'Model_Leaf', 'Model_Model 3', 'Model_Model S', 'Model_Model X',
       'Model_Model Y', 'Model_Mustang Mach-E', 'Model_Niro EV',
       'Model_Q4 e-tron', 'Model_e-tron', 'Model_i3', 'Model_i4', 'Model_iX',
       'Region_Australia', 'Region_Europe', 'Region_North America',
       'Vehicle_Type_SUV', 'Vehicle_Type_Sedan', 'Vehicle_Type_Truck',
       'Usage_Type_Fleet', 'Usage_Type_Personal'],
      dtype='object')
```

```
# update with the correct target column name
target = "Resale_Value_USD"

# Separate features and target
X = df.drop(columns=[target])
y = df[target]

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (2400, 58)
y_train shape: (2400,)
```

```python
df.to_csv("ev_preprocessed.csv", index=False)
print("✅ Preprocessed dataset saved as ev_preprocessed.csv")
```

```
✅ Preprocessed dataset saved as ev_preprocessed.csv
```

```
[ ] Start coding or generate with AI.
```

Link to processed data set (GitHub) –

https://github.com/maneesha-bogahawatta/Data-Preprocessing-Reading-Assignment_FDM.git

## 3. Conclusion

The preprocessing ensured that the dataset is free of inconsistencies and ready for use in predictive modeling.
By encoding categorical features, scaling numerical variables, and handling missing values, the dataset can now be effectively used for **machine learning tasks such as resale value prediction of electric vehicles**.

**4.References**


- Kaggle Data Set - Electric Vehicle Analytics Dataset.