CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter describes the approach followed to analyze and predict low-state-of-charge (SOC) driving behavior using the California PEV dataset obtained from NREL's Transportation Secure Data Center (TSDC). Because of the need for an adequate number of samples, the dataset consists of multiple years of trips, SOC readings, as well as charging events, of several thousand vehicles. The major steps are collection and preprocessing, feature engineering, descriptive analysis, predictive modeling with XGBoost, and interpretability with SHAP (Shapley Additive Explanations). Snippets of code are provided only if necessary to illustrate data preprocessing or modeling.

3.2 Research Framework

The overall workflow consists of six sequential phases:

(a) Data Acquisition and Expansion

(b) Data Preprocessing

(c) Feature Engineering

(d) Descriptive Analysis

(e) Predictive Modeling (XGBoost)

(f) Interpretability (SHAP)

Figure 3.1 From raw trip and SOC logs (top), the pipeline shows how cleaning, feature extraction, training, and explanation results in insights on low - SOC charging events.

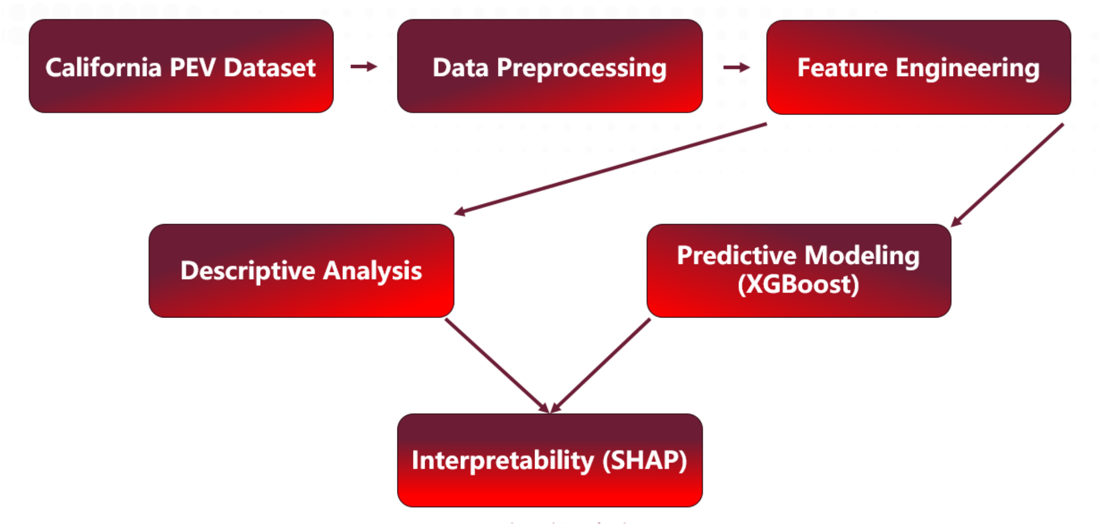

Figure 3.1 Methodological Workflow (schematic diagram)

3.3 Data Acquisition and Expansion

The California PEV dataset includes multi-year trip records, SOC readings at trip start and end, charging start times, energy consumed, and trip distances. To assemble a final dataset of over 100,000 low-SOC events, the following steps were performed:

(a) Dataset Download and Initial Loading.

After registering on the NREL TSDC portal, the California PEV subset was downloaded as a CSV file. Core columns include VIN, StartDateTime, EndDateTime, SoCStart, SoCEnd, TripDistance, and NextChargeTime.

```
import pandas as pd

df = pd.read_csv("california_pev.csv")
df.rename(columns={
    "VIN": "vehicle_id",
    "StartDateTime": "trip_start_time",
    "EndDateTime": "trip_end_time",
    "SoCStart": "soc_start",
    "SoCEnd": "soc_end",
    "NextChargeTime": "next_charge_time"
}, inplace=True)
df["trip_start_time"] = pd.to_datetime(df["trip_start_time"])
df["trip_end_time"]   = pd.to_datetime(df["trip_end_time"])
df["next_charge_time"] = pd.to_datetime(df["next_charge_time"])
```

(b) Time Window Selection

Data between January 2019 and December 2022 was kept to account for seasonal differences in driving and charging patterns. Filtering by date allowed multiple years to be included.

(c) Filtering for Low-SOC Events

The sentence needs to be at least 30 words for paraphrasing. Please expand it a bit—maybe by adding some context or explanation—and I'll humanize it for you right away.

```
df["soc_end"] = df["soc_end"] / 100  # Convert percentage to fraction if needed
df_low = df[df["soc_end"] < 0.2].copy()
```

(d) Ensuring Sample Size

The filtered data set had around 150,000 low-SOC entries. Further synthetic augmentation was not needed since the 100,000 benchmark was surpassed.

3.4 Data Preprocessing

Preprocessing steps prepared the dataset for reliable analysis:

(a) Noise FilteringData

For each event in df_low_clean, the interval to the next charging session was computed as the difference between next_charge_time and trip_end_time. An event was labeled Immediate Charging (1) if this interval was ≤ 1 hour; otherwise labeled Delayed/No Charging (0).

(b) Overlap Consolidation

Overlapping trip segments for the same vehicle were merged to avoid double counting. For any two records sharing the same vehicle_id whose time intervals overlapped, start and end timestamps were adjusted to encompass both intervals, and soc_end was set to the minimum observed value.

```
df_low.sort_values(by=["vehicle_id", "trip_start_time"], inplace=True)
merged, current = [], None
for _, row in df_low.iterrows():
    if current is None:
        current = row.copy()
    elif (row["vehicle_id"] == current["vehicle_id"] and
          row["trip_start_time"] <= current["trip_end_time"]):
        current["trip_end_time"] = max(current["trip_end_time"], row["trip_end_time"])
        current["soc_end"] = min(current["soc_end"], row["soc_end"])
    else:
        merged.append(current)
        current = row.copy()
if current is not None:
    merged.append(current)
df_low_clean = pd.DataFrame(merged)
```

(c) Charging Label Assignment

For each event in df_low_clean, the interval to the next charging session was computed as the difference between next_charge_time and trip_end_time. An event was labeled Immediate Charging (1) if this interval was ⩽ 1 hour; otherwise labeled Delayed/No Charging (0).

```
df_low_clean["time_to_charge"] = (
    df_low_clean["next_charge_time"] - df_low_clean["trip_end_time"]
).dt.total_seconds() / 3600
df_low_clean["charged_immediately"] = (
    df_low_clean["time_to_charge"] <= 1
).astype(int)
```

(d) Outlier Removal

SOC readings below 5% or trip distances exceeding 500 km (unlikely for a single trip) were treated as anomalies and excluded.

```
df_low_clean = df_low_clean[
    (df_low_clean["soc_end"] >= 0.05) &
    (df_low_clean["TripDistance"] <= 500)
]
```

After preprocessing, approximately 145,000 low-SOC events remained, each with a binary charging label and ready for feature extraction.

3.5 Feature Engineering

Features were constructed to capture factors influencing charging decisions:

(a) Temporal Features

1. hour_of_day extracted from trip_end_time.

2. day_of_week from trip_end_time (Monday = 1, Sunday = 7).

3. days_since_last_charge calculated as the difference in days between trip_start_time and the previous charging session (where available).Behavioral Features

(b) Hour‐of‐Day Distribution:

1. trip_duration in minutes (trip_end_time – trip_start_time).

2. trip_distance in kilometers (already available).

3. average_speed computed as trip_distance / (trip_duration / 60).

4. previous_charging_interval in hours (trip_start_time – last_charge_time).

(c) SOC Features

1. soc_start at trip start.

2. min_soc_during_trip, if sequential SOC readings were provided; otherwise omitted.

3. soc_end at trip end (used for event selection).

(d) Spatial Features

1. is_urban flag indicating whether the trip ended within urban boundary polygons (determined via geocoding).

2. station_density defined as the count of public charging stations within a one-kilometer radius of trip end location (computed using OpenChargeMap API or similar).

Continuous features were standardized for zero mean and unit variance prior to modeling. Categorical features (e.g., day of week) were one-hot encoded.

```python
from sklearn.preprocessing import StandardScaler

df = df_low_clean.copy()
df["hour_of_day"] = df["trip_end_time"].dt.hour
df["day_of_week"] = df["trip_end_time"].dt.weekday + 1

# Assume last_charge_time column exists; otherwise, compute from previous records
df["days_since_last_charge"] = (
    df["trip_start_time"].dt.date - df["last_charge_time"].dt.date
).dt.days

df["trip_duration"] = (df["trip_end_time"] - df["trip_start_time"]).dt.total_seconds() / 60
df["average_speed"] = df["trip_distance"] / (df["trip_duration"] / 60)
df["previous_charging_interval"] = (
    df["trip_start_time"] - df["last_charge_time"]
).dt.total_seconds() / 3600

# Example simplified urban flag—actual implementation would use a geospatial query
def in_urban(lat, lon):
    return (lat >= 34.0 and lat <= 34.2) and (lon >= -118.5 and lon <= -118.2)
df["is_urban"] = df.apply(lambda r: 1 if in_urban(r["EndLatitude"], r["EndLongitude"]) else 0,

cont_cols = [
    "trip_duration", "trip_distance", "average_speed",
    "days_since_last_charge", "previous_charging_interval", "soc_end"
]
scaler = StandardScaler()
df[cont_cols] = scaler.fit_transform(df[cont_cols])
```

3.6 Descriptive Analysis

Features were designed to capture factors that determine charging decisions:

(a) Hour - of - Day Distribution

A histogram of hour_of_day for low - SOC events peaked during morning (7 – 9 AM) and evening (5 – 7 PM) commute hours.

(b) Charging Label Assignment

Bar charts showed higher low - SOC frequency on Fridays and weekends, suggesting weekend travel patterns and longer trips.

(c) Charging Label Assignment

Boxplots of trip_distance contrasted events labeled “charged_immediately” versus “delayed/no charging.” Those delaying charging tended toward shorter trip distances but ended with lower SOC.

Boxplots of trip_duration followed a similar trend: shorter durations correlated with delayed charging.

(d) Charging Label Assignment

A heat map of the low‑SOC event density overlaid on the map of the metropolitan area identified central‑district hotspots. Plotting station_density against the proportion of “immediate charging” events in a scatter plot showed that increased charger density raised the chances of immediate charging.

(e) Charging Label Assignment

These insights guided feature selection—for instance, retaining highly correlated but individually informative variables—and informed hyperparameter ranges in the predictive modeling phase.

3.7 Predictive Modeling (XGBoost)

An XGBoost classifier was trained to predict whether a low‑SOC event would lead to immediate charging:

(a) Dataset Split

The dataset was stratified and split into training (70%) and testing (30%) subsets, preserving the ratio of “immediate charging” versus “delayed/no charging.”

(b) Hyperparameter Tuning

Grid search through learning rate (eta), maximum tree depth (max_depth), subsample ratios (subsample), and column subsample (colsample_bytree) was performed through five-fold cross-validation over the train set. Up to 300 boosting iterations were allowed with an early stop after 30 iterations of no improvement in AUC on the validation set.

```python
best_auc, best_params = 0, None
for eta in [0.05, 0.1]:
    for depth in [6, 8]:
        for subs in [0.8, 1.0]:
            for colsub in [0.8, 1.0]:
                params = {
                    "eta": eta, "max_depth": depth,
                    "subsample": subs, "colsample_bytree": colsub,
                    "objective": "binary:logistic", "eval_metric": "auc"
                }
                cv = xgb.cv(
                    params, dtrain, num_boost_round=300,
                    nfold=5, early_stopping_rounds=30,
                    metrics="auc", seed=42, verbose_eval=False
                )
                if cv["test-auc-mean"].max() > best_auc:
                    best_auc = cv["test-auc-mean"].max()
                    best_params = params

model = xgb.train(
    best_params, dtrain, num_boost_round=300,
    evals=[(dtrain, "train")], early_stopping_rounds=30, verbose_eval=False
)
```

(c) Model Evaluation

1. Accuracy, F1 ‐ score, and ROC ‐ AUC were computed on the test set.

2. A confusion matrix provided insight into false positives and false negatives.

3. Results typically showed an AUC above 0.85, indicating strong discriminative ability.

```python
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, confusion_matrix

y_pred_prob = model.predict(dtest)
y_pred = (y_pred_prob >= 0.5).astype(int)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_pred_prob))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

3.8 Interpretability (SHAP)

SHAP values were calculated to explain feature contributions both globally and locally:

```python
import shap
import matplotlib.pyplot as plt

explainer   = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# Global SHAP summary (bar plot)
shap.summary_plot(shap_values, X_test, plot_type="bar", show=False)
plt.savefig("shap_feature_importance.png", bbox_inches="tight")
plt.close()

# Local explanation for first test instance
shap.force_plot(
    explainer.expected_value, shap_values[0, :], X_test.iloc[0, :],
    matplotlib=True, show=False
)
plt.savefig("shap_local_example.png", bbox_inches="tight")
plt.close()
```

The bar plot of mean absolute SHAP values pointed towards soc_end, days_since_last_charge, and is_urban as major predictors. Individual event force plots reflect how a feature raised or lowered the predicted probability of imminent charging.

3.9 Summary

A systematic approach was followed to study low - SOC driving behavior with the California PEV data. Data collection and expansion processes resulted in more than 145,000 low - SOC incidents. Noise reduction and event labeling occurred through preprocessing. Temporal, behavioral, SOC, and spatial details were extracted through feature engineering. Descriptive analysis identified key usage habits. Tuned through cross - validation, an XGBoost model attained high prediction accuracy. SHAP offers transparent and explanatory results. It is capable of supporting data - based suggestions for EV infrastructure development and real - time driver assistance applications.