```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score

# Load dataset
df = pd.read_csv("C:/Users/KIIT/Downloads/ev_charging_patterns1.csv")

# Drop high-cardinality or irrelevant columns
df.drop(columns=['User ID'], inplace=True)

# Convert time features into numeric (hour, minute)
for col in ['Charging Start Time', 'Charging End Time']:
    df[col] = pd.to_datetime(df[col])
    df[f'{col}_hour'] = df[col].dt.hour
    df[f'{col}_minute'] = df[col].dt.minute
    df.drop(columns=[col], inplace=True)

# Separate categorical and numerical columns
categorical_cols = ['Vehicle Model', 'Charging Station ID', 'Charging Station Location',
                    'Time of Day', 'Day of Week', 'Charger Type', 'User Type']
numerical_cols = [col for col in df.columns if col not in categorical_cols + ['Charging Duration
(hours)']]

# One-Hot Encode categorical features
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Handle missing values (Fill numerical columns with median)
df.fillna(df.median(numeric_only=True), inplace=True)

# Define features and target
target_column = "Charging Duration (hours)"
X = df.drop(columns=[target_column])
y = df[target_column]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Define models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, random_state=42),
    "Support Vector Regressor": SVR(),
    "KNN Regressor": KNeighborsRegressor(n_neighbors=5)
}

# Initialize results dictionary
results = {"Stage": [], "Model": [], "R² Score": []}

# **Step 1: Before Normalization**
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = r2_score(y_test, y_pred)
    results["Stage"].append("Before Normalization")
    results["Model"].append(name)
    results["R² Score"].append(score)

# **Step 2: After Normalization**
scaler = StandardScaler()
X_train[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test[numerical_cols] = scaler.transform(X_test[numerical_cols])

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = r2_score(y_test, y_pred)
    results["Stage"].append("After Normalization")
    results["Model"].append(name)
    results["R² Score"].append(score)

# **Step 3: After Tuning**
tuned_models = {
    "Random Forest": GridSearchCV(RandomForestRegressor(random_state=42),
                                  param_grid={"n_estimators": [100, 200], "max_depth": [10, None]},
                                  cv=3, n_jobs=-1),
    "Gradient Boosting": GridSearchCV(GradientBoostingRegressor(random_state=42),
                                      param_grid={"n_estimators": [100, 200], "learning_rate": [0.05, 0.1]},
                                      cv=3, n_jobs=-1),
    "Support Vector Regressor": GridSearchCV(SVR(),
```

```
                                param_grid={"C": [0.1, 1, 10], "gamma": ["scale", "auto"]},
                                cv=3, n_jobs=-1),
    "KNN Regressor": GridSearchCV(KNeighborsRegressor(),
                                param_grid={"n_neighbors": [3, 5, 7]},
                                cv=3, n_jobs=-1),
    "Decision Tree": GridSearchCV(DecisionTreeRegressor(random_state=42),
                                param_grid={"max_depth": [5, 10, None]},
                                cv=3, n_jobs=-1),
    "Linear Regression": LinearRegression()
}

for name, model in tuned_models.items():
    if isinstance(model, GridSearchCV):
        model.fit(X_train, y_train)
        best_model = model.best_estimator_
    else:
        best_model = model
        best_model.fit(X_train, y_train)

    y_pred = best_model.predict(X_test)
    score = r2_score(y_test, y_pred)

    results["Stage"].append("After Tuning")
    results["Model"].append(name)
    results["R² Score"].append(score)

# Convert results to DataFrame
results_df = pd.DataFrame(results)
print(results_df)
 code
```