#Importing the Libraries and data

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from numpy import asarray

from sklearn.preprocessing import OrdinalEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Import the CSV file

ev = pd.read\_csv('ElectricCarData\_Clean.csv')

ev.head()

ev.columns

ev.info()

ev.shape

ev.isnull().sum()

ev.dtypes

ev.nunique()

ev.describe()

#Analysis range by EV Brand

ax= plt.figure(figsize=(20,5))

sns.barplot(x='Brand',y='Range\_Km',data=ev,palette='hls')

plt.grid(axis='y')

plt.title('Electric Vehicle Range VS EV Brand')

plt.xlabel('Brand')

plt.ylabel('Range per Km')

plt.xticks(rotation=45)

# model with highest range

range\_df = ev.sort\_values(by=['Range\_Km'], ascending=False)

range\_df[['Brand','Model','Range\_Km']].head(n=1)

#Analysis acceleration by EV Brand

ax= plt.figure(figsize=(20,5))

sns.barplot(x='Brand',y='AccelSec',data=ev,palette='coolwarm')

plt.grid(axis='y')

plt.title('Electric Vehicle Acceleration VS EV Brand')

plt.xlabel('Brand')

plt.ylabel('AccelSec')

plt.xticks(rotation=45)

df = ev.sort\_values(by=['AccelSec'], ascending=True)

df[['Brand','Model','AccelSec']].head(n=1)

#Analysis top speed by EV Brand

ax= plt.figure(figsize=(20,5))

sns.barplot(x='Brand',y='TopSpeed\_KmH',data=ev,palette='husl')

plt.grid(axis='y')

plt.title('Electric Vehicle Top Speed VS EV Brand')

plt.xlabel('Brand')

plt.ylabel('TopSpeed\_KmH')

plt.xticks(rotation=45)

speed\_df = ev.sort\_values(by=['TopSpeed\_KmH'], ascending=False)

speed\_df[['Brand','Model','TopSpeed\_KmH']].head(n=1)

#Analysis efficiency by EV Brand

ax= plt.figure(figsize=(20,5))

sns.barplot(x='Brand',y='Efficiency\_WhKm',data=ev,palette='Paired')

plt.grid(axis='y')

plt.title('Electric Vehicle efficiency VS EV Brand')

plt.xlabel('Brand')

plt.ylabel('Efficiency\_WhKm')

plt.xticks(rotation=45)

eff\_df = ev.sort\_values(by=['Efficiency\_WhKm'], ascending=False)

eff\_df[['Brand','Model','Efficiency\_WhKm']].head(n=1)

seat\_df = ev.sort\_values(by=['Seats'], ascending=False)

seat\_df[['Brand','Model','Range\_Km', 'Seats']].head(n=1)

##Distribution of range with PowerTrain

fig, axs = plt.subplots(1,2)

sns.catplot(x="AccelSec", y="Range\_Km", data=ev)

plt.close(1)

ev.columns

ev1 = ev[['Brand', 'Model', 'PowerTrain', 'RapidCharge', 'PlugType', 'BodyStyle', 'Segment']]

print(ev1)

encoder = OrdinalEncoder()

result = encoder.fit\_transform(ev1)

print(result)

temp = ['Brand', 'Model', 'PowerTrain', 'RapidCharge', 'PlugType', 'BodyStyle', 'Segment']

ev.drop(temp,axis=1,inplace=True)

result = pd.DataFrame(result)

result.columns=["Brand", "Model", "PowerTrain", 'RapidCharge', "PlugType", "BodyStyle", "Segment"]

result.index+=1

result

ev1 = ev

ev1.index+=1

ev1[['Brand', 'Model', 'PowerTrain', 'RapidCharge', 'PlugType', 'BodyStyle', 'Segment']] = result[['Brand', 'Model', 'PowerTrain', 'RapidCharge', 'PlugType', 'BodyStyle', 'Segment']]

import numpy as np

features = ['TopSpeed\_KmH', 'AccelSec', 'Efficiency\_WhKm', 'FastCharge\_KmH']

for column in features:

ev = ev[ev[column] != '-']

X = ev[['TopSpeed\_KmH', 'AccelSec', 'Efficiency\_WhKm', 'FastCharge\_KmH']].values

y = ev['Range\_Km']

#Training and Test Data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 42)

print("X\_train : ",X\_train.shape)

print("X\_test : ",X\_test.shape)

print("y\_train : ",y\_train.shape)

print("y\_test : ",y\_test.shape)

# Create Linear Model

lr = LinearRegression()

lr.fit(X\_train, y\_train)

y\_pred = lr.predict(X\_test)

y\_pred[0:5]

import joblib

joblib.dump(lr, r"C:\Users\kskor\OneDrive\Desktop\SHRUTI\Sem 6\EDI\linear\_regression\_model.pkl")

loaded\_lr\_model = joblib.load('linear\_regression\_model.pkl')

#real - time data

import joblib

lr\_model = joblib.load(r"C:\Users\kskor\OneDrive\Desktop\SHRUTI\Sem 6\EDI\linear\_regression\_model.pkl")

top\_speed = 160

acceleration = 15.0

efficiency = 100

fast\_charge = 250

real\_time\_features = np.array([[top\_speed, acceleration, efficiency, fast\_charge]])

predicted\_range = lr\_model.predict(real\_time\_features)

print(predicted\_range)

#Predict the range of this EV using linear regression model.

import joblib

lr = joblib.load(r"C:\Users\kskor\OneDrive\Desktop\SHRUTI\Sem 6\EDI\linear\_regression\_model.pkl")

import numpy as np

top\_speed = 170

acceleration = 20

efficiency = 150

fast\_charge = 270

features = np.array([[top\_speed, acceleration, efficiency, fast\_charge]])

predicted\_range = lr.predict(features)

print("Predicted Range (Km):", predicted\_range[0])

total\_capacity\_kwh = float(input("Enter the total capacity of the battery pack in kilowatt-hours (KWh): "))

efficiency\_wh\_km = float(input("Enter the efficiency of the battery pack in percentage (%): "))

consumed\_energy = efficiency\_wh\_km \* predicted\_range / 1000 # Convert to KWh

# Calculate SoC

current\_soc = 100 - (consumed\_energy / total\_capacity\_kwh) \* 100

print(f"Estimated State of Charge (SoC): {current\_soc[0]:.2f}%")

def charging\_discharging\_decision(predicted\_range, current\_soc, min\_soc\_threshold, max\_soc\_threshold):

"""

Decide whether to charge, discharge, or maintain the current state based on predicted range and current SoC.

"""

if current\_soc < min\_soc\_threshold:

decision = 'discharge'

elif current\_soc > max\_soc\_threshold:

decision = 'charge'

else:

decision = 'maintain'

return decision

predicted\_range

current\_soc

min\_soc\_threshold = 20

max\_soc\_threshold = 80

decision = charging\_discharging\_decision(predicted\_range, current\_soc, min\_soc\_threshold, max\_soc\_threshold)

print("Charging/Discharging Decision:", decision)

import numpy as np

def regenerative\_braking(current\_speed, deceleration\_rate, efficiency\_wh\_km):

recovered\_energy = 0.5 \* (current\_speed 2) / deceleration\_rate \* efficiency\_wh\_km / 1000 # Convert to KWh

return recovered\_energy

lr = joblib.load(r"C:\Users\kskor\OneDrive\Desktop\SHRUTI\Sem 6\EDI\linear\_regression\_model.pkl")

top\_speed = 170

acceleration = 20

efficiency = 150

fast\_charge = 270

features = np.array([[top\_speed, acceleration, efficiency, fast\_charge]])

predicted\_range = lr.predict(features)

print("Predicted Range (Km):", predicted\_range[0])

total\_capacity\_kwh = float(input("Enter the total capacity of the battery pack in kilowatt-hours (KWh): "))

consumed\_energy = efficiency\_wh\_km \* predicted\_range / 1000 # Convert to KWh

# Calculate SoC

current\_soc = 100 - (consumed\_energy / total\_capacity\_kwh) \* 100

print(f"Estimated State of Charge (SoC): {current\_soc[0]:.2f}%")

# Example values for regenerative braking

current\_speed = 40

deceleration\_rate = 2

recovered\_energy = regenerative\_braking(current\_speed, deceleration\_rate, efficiency\_wh\_km)

print(f"Recovered Energy from Regenerative Breaking: {recovered\_energy:.2f} KWh")

current\_soc += 100 - (recovered\_energy / total\_capacity\_kwh) \* 100

print(f"Estimated State of Charge (SoC) after regenerative breaking: {current\_soc[0]:.2f}%")

max\_soc\_threshold = 80

if current\_soc > max\_soc\_threshold:

current\_soc = max\_soc\_threshold

min\_soc\_threshold = 20

if current\_soc < min\_soc\_threshold:

current\_soc = min\_soc\_threshold

adjusted\_decision = charging\_discharging\_decision(predicted\_range, current\_soc, min\_soc\_threshold, max\_soc\_threshold)

print("Charging/Discharging Decision after Regenerative Breaking:", adjusted\_decision)

# Get the coefficients

coefficients = lr.coef\_

# Print the coefficients for each feature

for feature, coefficient in zip(features, coefficients):

print(f"{feature}: {coefficient}")

lr.coef\_

lr.intercept\_

import matplotlib.pyplot as plt

import seaborn as sns

# Define the number of subplots and their layout

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

fig.subplots\_adjust(hspace=0.5)

# List of features to plot

features = ["TopSpeed\_KmH", "AccelSec", "Efficiency\_WhKm", "FastCharge\_KmH"]

# Create scatter plots with regression lines for each feature

for i, feature in enumerate(features):

row, col = divmod(i, 2)

sns.regplot(x=feature, y="Range\_Km", data=ev, ax=axes[row, col])

axes[row, col].set\_ylim(0)

# Show the plot

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Assuming you have imported and prepared your dataset

# ev = pd.read\_csv('your\_dataset.csv')

# Check the data types of the columns in your DataFrame

print(ev.dtypes)

# Define the number of subplots and their layout

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

fig.subplots\_adjust(hspace=0.5)

# List of features to plot

features = ["TopSpeed\_KmH", "AccelSec", "Efficiency\_WhKm", "FastCharge\_KmH"]

# Create scatter plots with regression lines for each feature

for i, feature in enumerate(features):

row, col = divmod(i, 2)

sns.regplot(x=feature, y="Range\_Km", data=ev, ax=axes[row, col])

axes[row, col].set\_ylim(0)

# Show the plot

plt.show()

ax1 = sns.kdeplot(y\_test, color="r", label="Actual Value")

sns.kdeplot(y\_pred, color="b", label="Fitted Values", ax=ax1)

plt.title('Actual vs Fitted value for Range')

plt.xlabel('Range(in Km)')

plt.ylabel('Proportion of Cars')

plt.show()

plt.close()

#Model Evaluation (Regression Metrics)

# Calculate the score for Training Data

lr.score(X\_train, y\_train)

print("R2 for Traing Data: ", lr.score(X\_train, y\_train))

# Calculate the score (R^2 for Regression) for Testing Data

lr.score(X\_test, y\_test)

print("R2 for Testing Data: ", lr.score(X\_test, y\_test))

#Calculate Mean Squared Error

mean\_squared\_error(y\_test, y\_pred)

print("MSE: ", mean\_squared\_error(y\_test, y\_pred))

#Calculate Mean Absolute Error(MAE)

mean\_absolute\_error(y\_test, y\_pred)

print("MAE: ",mean\_absolute\_error(y\_test, y\_pred))

#Calculate Root Mean Squared Error(RMSE)

print("RMSE: ",np.sqrt(mean\_squared\_error(y\_test,y\_pred)))

Explanation:  
  
This Jupyter Notebook file seems to be an analysis and modeling pipeline for electric vehicle (EV) data.

1. Importing Libraries and Data: This section imports necessary libraries such as NumPy, Pandas, Seaborn, Matplotlib, and scikit-learn. It also loads the dataset `ElectricCarData\_Clean.csv` using Pandas and displays some basic information about the dataset such as columns, data types, shape, null values, unique values, and descriptive statistics.

2. Data Visualization and Analysis: This part contains several visualizations using Seaborn and Matplotlib to analyze different aspects of the electric vehicle data. It plots bar charts to compare EV range, acceleration, top speed, and efficiency by brand. It also analyzes the distribution of range with powertrain type and segments. Additionally, it visualizes the relationship between range and various features using scatter plots with regression lines.

3. Data Preprocessing: In this section, the code preprocesses the data before training the machine learning model. It encodes categorical variables using ordinal encoding and removes any rows where certain features have missing or invalid values.

4. Model Training: This part splits the data into training and testing sets using `train\_test\_split` from scikit-learn. It then trains a linear regression model using the training data.

5. Model Evaluation and Saving: After training the model, the code evaluates its performance using various regression metrics such as R-squared, Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). It also saves the trained model using joblib for future use.

6. Real-Time Prediction and Decision Making: This section demonstrates how to use the trained model for real-time prediction. It defines a function to make charging/discharging decisions based on predicted range and current state of charge (SoC) of the battery. It also includes an example of regenerative braking and adjusts the charging/discharging decision accordingly.

7. Visualization of Model Performance: Finally, the code visualizes the actual vs. predicted values of range using a kernel density plot (KDE plot) to assess the model's performance visually.

Overall, this Jupyter Notebook file provides a comprehensive analysis of electric vehicle data, trains a machine learning model to predict EV range based on various features, and evaluates the model's performance using regression metrics. It also demonstrates how to use the trained model for real-time prediction and decision-making in practical scenarios.