**Car Price prediction**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

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

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

# Load the dataset

df = pd.read\_csv('car\_data.csv') # Replace with your dataset path

# Display the first few rows

print(df.head())

# Check for missing values

print(df.isnull().sum())

# Handle missing values (example: drop or fill)

df = df.dropna() # or df.fillna(method='ffill', inplace=True)

# Convert categorical variables using OneHotEncoder

categorical\_features = ['brand', 'model'] # replace with actual categorical columns

numerical\_features = ['horsepower', 'mileage', 'year'] # replace with actual numerical columns

# Define the preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('num', 'passthrough', numerical\_features),

('cat', OneHotEncoder(), categorical\_features)

])

# Split the dataset into features and target

X = df.drop('price', axis=1) # Assuming 'price' is the target variable

y = df['price']

# Split the data into training and testing sets

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

# Create a pipeline with preprocessing and model

pipeline = Pipeline(steps=[

('preprocessor', preprocessor),

('model', RandomForestRegressor(random\_state=42)) # You can try LinearRegression() here too

])

# Train the model

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

# Make predictions

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

# Evaluate the model

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

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

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Absolute Error: {mae}')

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

# Plot predicted vs actual prices

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')

plt.grid()

plt.show()