CAR PRICE PREDICTION - CASE STUDY (REGRESSION)

```
================================
1 Import Libraries
================================
import pandas as pd,import numpy as np import matplotlib.pyplot as plt ,
import seaborn as sns , import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler, LabelEncoder,
OneHotEncoder
from sklearn.model_selection import train_test_split, KFold,
cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
import pickle

================================
 2 Load Dataset
 ================================
df = pd.read_csv("Cars.csv") , df.head() , df.shape , df.info()
df.describe() , df.isnull().sum()
 ================================
3 Data Cleaning & Feature Engineering
 ================================
# Mapping 'owner' column
owner_mapping = {"First Owner": 1, "Second Owner": 2, "Third Owner":3,
                 "Fourth & Above Owner":4, "Test Drive Car": 5}
df['owner'] = df['owner'].map(owner_mapping)

# Filter fuel column to remove LPG and CNG
df = df[~df['fuel'].isin(['LPG','CNG'])]
# Extract numeric part of mileage and convert to float
df['mileage'] = df['mileage'].str.split().str[0].astype(float)
# Rename 'name' to 'brand'
df = df.rename(columns={"name": "brand"})
# Drop unnecessary columns
df.drop(columns=['torque'], inplace=True)

# Remove Test Drive Cars
df = df[df['owner'] != 5]

# Getting categorical and numerical
```

```
cat_col = df.select_dtypes(exclude=['int64','float64'])

num_col = df.select_dtypes(include = ['int64','float64'])

================================
4 Encode Categorical Columns
================================
label_brand = LabelEncoder()
df['brand'] = label_brand.fit_transform(df['brand'])

================================
5 Explore & Visualize Data
================================
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")

# Optional: Boxplots to detect outliers
col_dict = {"brand":1, "year":2,"fuel":3,"mileage":4,"max_power":5}
plt.figure(figsize=(20,30))
for variable, i in col_dict.items():
    plt.subplot(5,11,i)
    plt.boxplot(df[variable])
    plt.title(variable)

================================
6 Prepare Features and Target
================================
X = df[['year','max_power','brand','mileage','fuel']]
y = np.log(df['selling_price'])  # log-transform target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fill missing numeric values
X_train['mileage'].fillna(X_train['mileage'].mean(), inplace=True)
X_train['max_power'].fillna(X_train['max_power'].median(), inplace=True)
X_test['mileage'].fillna(X_test['mileage'].mean(), inplace=True)
X_test['max_power'].fillna(X_test['max_power'].median(), inplace=True)
# Fill missing values with mode
X_test['category_col'].fillna(X_test['category_col'].mode()[0],
inplace=True)
# Get distribution of existing categories
probs = X_test['category_col'].value_counts(normalize=True)
# Fill missing values randomly based on this distribution
X_test['category_col'] = X_test['category_col'].apply(
```

```
    lambda x: np.random.choice(probs.index, p=probs.values) if
pd.isna(x) else x )

================================
7️⃣ Scale Numerical Columns
================================
num_cols = ['max_power','mileage']
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])

================================
8️⃣ Train Random Forest Model
================================
rfr = RandomForestRegressor(random_state=1)
rfr.fit(X_train, y_train)

yhat = rfr.predict(X_test)
print("MSE:", mean_squared_error(y_test, yhat))
print("R2:", r2_score(y_test, yhat))

# Feature importance
feature_importances = rfr.feature_importances_
print("Feature Importances:", feature_importances)

================================
9️⃣ Compare Multiple Models with Cross-Validation
================================
algorithms = [LinearRegression(), SVR(), KNeighborsRegressor(),
              DecisionTreeRegressor(random_state=0),
RandomForestRegressor(n_estimators=100, random_state=0)]
algorithm_names = ["Linear Regression", "SVR", "KNeighbors Regressor",
"Decision-Tree Regressor", "Random-Forest Regressor"]

kfold = KFold(n_splits=5, shuffle=True, random_state=1)

for i, model in enumerate(algorithms):
    scores = cross_val_score(model, X_train, y_train, cv=kfold,
scoring='neg_mean_squared_error')
    print(f"{algorithm_names[i]} - Score: {-scores}; Mean:
{-scores.mean()}")

================================
🔟 Grid Search for Random Forest
================================
param_grid = {'max_depth': [5, 10, None],
```

```
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 20,
30, 100]}

grid = GridSearchCV(estimator=RandomForestRegressor(random_state=1),
                    param_grid=param_grid,
                    cv=kfold,
                    n_jobs=-1,
                    return_train_score=True,
                    refit=True,
                    scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)

best_params = grid.best_params_
best_mse = -grid.best_score_
print("Best Parameters:", best_params)
print("Best MSE:", best_mse)

# Predict on test set with best model
yhat_best = grid.predict(X_test)
print("Test MSE (best):", mean_squared_error(y_test, yhat_best))
print("Test R2 (best):", r2_score(y_test, yhat_best))

================================
11 Feature Importance (Permutation)
================================
perm_importance = permutation_importance(grid.best_estimator_, X_test,
y_test)
sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(X_train.columns[sorted_idx],
perm_importance.importances_mean[sorted_idx])
plt.xlabel("Permutation Feature Importance")

================================
12 Save Models and Objects
================================
pickle.dump(grid, open('Model/car-prediction.model', 'wb'))
pickle.dump(label_brand, open('Model/brand-label.model', 'wb'))
pickle.dump(scaler, open('Model/car-scaler.model', 'wb'))
pickle.dump(rfr, open('Model/feature_importance.model', 'wb'))

# Load model example
loaded_model = pickle.load(open('Model/car-prediction.model', 'rb'))
predicted_price = loaded_model.predict(X_test[:1])
print("Predicted Price (first sample):", np.exp(predicted_price))
```