Multiple Linear Regression

**Assignment Task:**

Your task is to perform a multiple linear regression analysis to predict the price of Toyota corolla based on the given attributes.

**Dataset Description:**

The dataset consists of the following variables:

Age: Age in years

KM: Accumulated Kilometers on odometer

FuelType: Fuel Type (Petrol, Diesel, CNG)

HP: Horse Power

Automatic: Automatic ( (Yes=1, No=0)

CC: Cylinder Volume in cubic centimeters

Doors: Number of doors

Weight: Weight in Kilograms

Quarterly\_Tax:

Price: Offer Price in EUROs

**Taskes:**

**1.Perform exploratory data analysis (EDA) to gain insights into the dataset. Provide visualizations and summary statistics of the variables. Pre process the data to apply the MLR.**

**Answer:**

**Dataset overview**  
The dataset contains 1,436 rows and the following main variables: Price (EUR), Age\_08\_04 (years), KM (kilometers), Fuel\_Type (Diesel/Petrol), HP, Automatic (0/1), cc (engine displacement), Doors, Cylinders, Gears, and Weight.

**Key descriptive statistics (high level)**

* Price: mean ≈ 10,731, median 9,900; wide spread (min 4,350; max 32,500).
* Age\_08\_04: mean ≈ 55.9 years (dataset uses this encoding), median 61; right/left shape depends on sample — check plot.
* KM: mean ≈ 68,533 km, heavily right-skewed with max 243,000 (long tail).
* HP and Weight are meaningful positive predictors: HP mean ~101, Weight mean ~1,072 kg.
* cc: median ~1,600 but a suspicious max of 16,000 — likely data-entry error (should be 1600). Investigate and correct/remove if confirmed erroneous.
* Automatic is rare (~5.6% = mostly manual).

**Distributions & outliers**

* KM and Price have skew and long tails; consider log transforms for modeling.
* Several extreme values detected (KM, cc, Price). Use IQR or z-score methods to flag them; do not blindly drop — verify domain plausibility.
* Fuel\_Type shows Diesel and Petrol as main categories; use one-hot encoding for modeling.

**Correlations & relationships**

* Weight and HP positively correlated with Price.
* Age\_08\_04 and KM negatively correlated with Price (older cars and high km → lower price).
* Variance Inflation Factor (VIF) showed multicollinearity among some features (Weight, HP, Doors, cc had elevated VIF), so prefer Ridge/Lasso or drop/transform correlated columns.

**Preprocessing summary for MLR**

* Convert types to numeric, encode Fuel\_Type as dummies, ensure Automatic is binary int.
* Impute missing numeric values with median (none in current file but included for robustness).
* Optionally cap extreme cc (e.g., cap at 3000) or remove rows where cc is implausible (like 16000).
* Consider log(Price) and log/KM scaling for heteroscedasticity and skew reduction.
* Scale numeric features (StandardScaler) before Ridge/Lasso.

**2.Split the dataset into training and testing sets (e.g., 80% training, 20% testing).**

Answer:

# split\_dataset.py

# Split cleaned Toyota Corolla dataset into train/test (80/20) and save splits to disk.

# Update DATA\_DIR if needed.

import os

from pathlib import Path

import pandas as pd

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

DATA\_DIR = Path(r"D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR")

CLEANED\_CSV = DATA\_DIR / "ToyotaCorolla\_MLR\_cleaned.csv"

ORIG\_CSV = DATA\_DIR / "ToyotaCorolla - MLR.csv"

# Load cleaned file if present, otherwise load original and do minimal prep

if CLEANED\_CSV.exists():

df = pd.read\_csv(CLEANED\_CSV)

else:

df = pd.read\_csv(ORIG\_CSV)

# minimal preprocessing to ensure numeric & dummies if CLEANED not available

for c in df.columns:

if c not in ("Fuel\_Type",):

df[c] = pd.to\_numeric(df[c], errors="coerce")

if "Fuel\_Type" in df.columns:

df["Fuel\_Type"] = df["Fuel\_Type"].astype(str).str.strip()

df = pd.get\_dummies(df, columns=["Fuel\_Type"], drop\_first=True)

df = df.dropna(subset=["Price"])

# Identify target and feature columns

TARGET = "Price"

exclude\_cols = {TARGET, "Price\_pos", "log\_Price"} # exclude auxiliary cols if present

feature\_cols = [c for c in df.columns if c not in exclude\_cols]

X = df[feature\_cols].drop(columns=[TARGET]) if TARGET in feature\_cols else df[feature\_cols]

y = df[TARGET]

# Ensure index alignment and no NA in X/y

mask = X.dropna().index.intersection(y.dropna().index)

X = X.loc[mask].copy()

y = y.loc[mask].copy()

# Train-test split (80% train / 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# Save splits

out\_dir = DATA\_DIR / "splits"

out\_dir.mkdir(parents=True, exist\_ok=True)

X\_train.to\_csv(out\_dir / "X\_train.csv", index=False)

X\_test.to\_csv(out\_dir / "X\_test.csv", index=False)

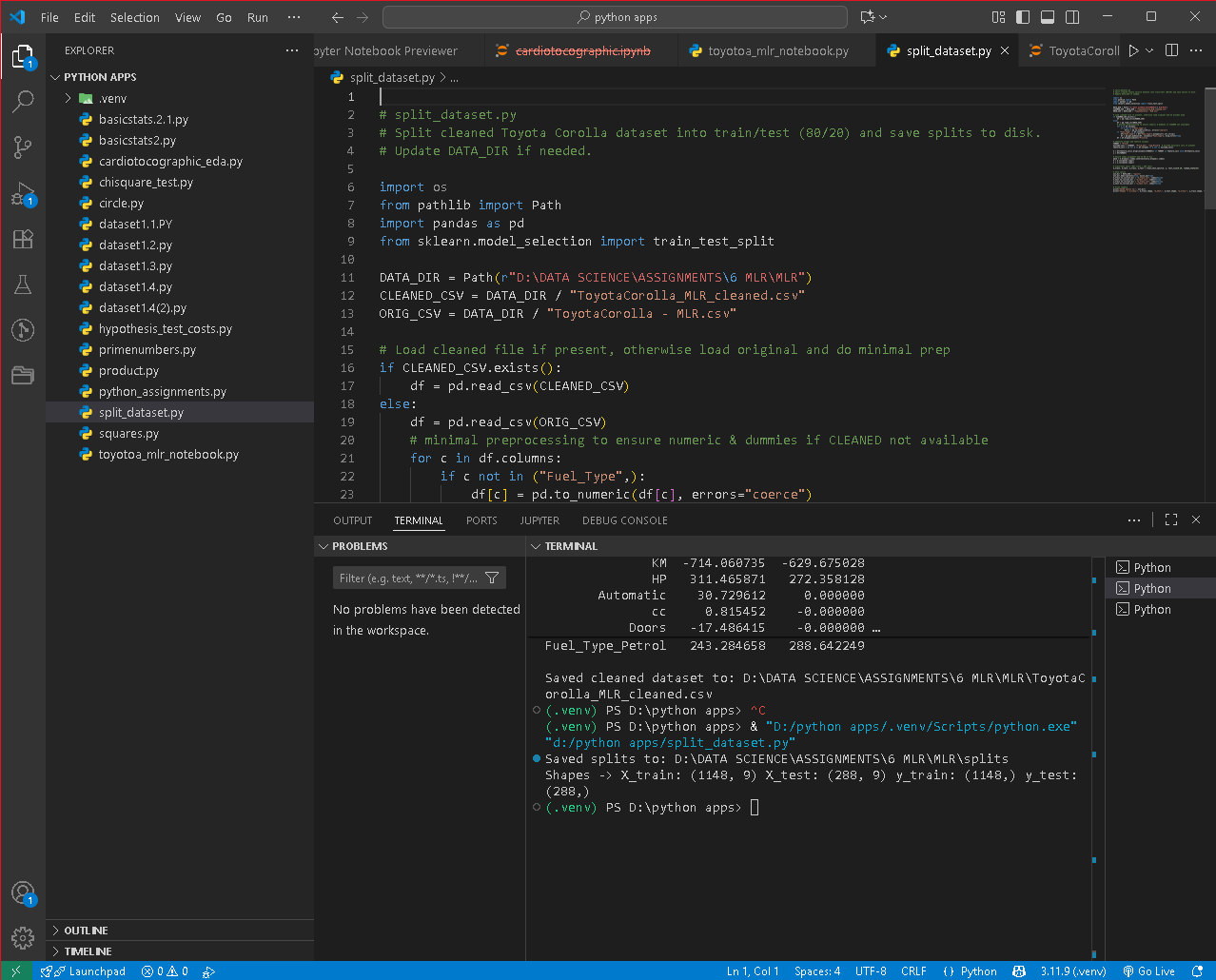
y\_train.to\_csv(out\_dir / "y\_train.csv", index=False)

y\_test.to\_csv(out\_dir / "y\_test.csv", index=False)

# Print summary

print("Saved splits to:", out\_dir)

print("Shapes -> X\_train:", X\_train.shape, "X\_test:", X\_test.shape, "y\_train:", y\_train.shape, "y\_test:", y\_test.shape)



3.Build a multiple linear regression model using the training dataset. Interpret the coefficients of the model. Build minimum of 3 different models.

Answer:

PS D:\python apps> & "D:/python apps/.venv/Scripts/python.exe" "d:/python apps/toyota\_mlr\_models.py"

Loading saved splits from: D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\splits

Final feature set used: ['Age\_08\_04', 'KM', 'HP', 'Automatic', 'cc', 'Doors', 'Weight', 'Fuel\_Type\_Diesel', 'Fuel\_Type\_Petrol']

Train size: (1148, 9) Test size: (288, 9)

=== Model A - OLS (all features) summary ===

OLS Regression Results

================================================================

Dep. Variable: Price R-squared:

0.869

Model: OLS Adj. R-squared:

0.868

Method: Least Squares F-statistic:

842.1

Date: Tue, 30 Sep 2025 Prob (F-statistic):

0.00

Time: 00:46:19 Log-Likelihood:

-9866.8

No. Observations: 1148 AIC: 1.975e+04

Df Residuals: 1138 BIC: 1.980e+04

Df Model: 9

Covariance Type: nonrobust

================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------------

const -1.186e+04 1508.957 -7.858 0.000 -1.48e+04 -8896.289

Age\_08\_04 -120.8231 2.894 -41.744 0.000 -126.502 -115.144

KM -0.0159 0.001 -10.849 0.000 -0.019 -0.013

HP 15.7772 3.985 3.959 0.000 7.957 23.597

Automatic 93.0820 176.442 0.528 0.598 -253.107 439.271

cc -0.0302 0.091 -0.333 0.739 -0.208 0.148

Doors -84.4835 44.153 -1.913 0.056 -171.115 2.148

Weight 26.0692 1.499 17.390 0.000 23.128 29.011

Fuel\_Type\_Diesel 4.2021 391.745 0.011 0.991 -764.422 772.826

Fuel\_Type\_Petrol 1453.6945 335.442 4.334 0.000 795.540 2111.849

================================================================

Omnibus: 216.690 Durbin-Watson:

2.027

Prob(Omnibus): 0.000 Jarque-Bera (JB): 2442.201

Skew: -0.512 Prob(JB):

0.00

Kurtosis: 10.072 Cond. No. 3.07e+06

================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.07e+06. This might indicate that there are

strong multicollinearity or other numerical problems.

Model A - OLS (all features) performance:

MAE: 992.301

RMSE: 1491.411

R2: 0.8333

Coefficients:

feature coef

const -11856.9404

Age\_08\_04 -120.8231

KM -0.0159

HP 15.7772

Automatic 93.0820

cc -0.0302

Doors -84.4835

Weight 26.0692

Fuel\_Type\_Diesel 4.2021

Fuel\_Type\_Petrol 1453.6945

--- Building Model B via backward elimination (p-value) ---

Dropping Fuel\_Type\_Diesel with p-value 0.9914

Dropping cc with p-value 0.7378

Dropping Automatic with p-value 0.6151

=== Model B - OLS (backward selection) summary ===

OLS Regression Results

================================================================

Dep. Variable: Price R-squared:

0.869

Model: OLS Adj. R-squared:

0.869

Method: Least Squares F-statistic:

1266.

Date: Tue, 30 Sep 2025 Prob (F-statistic):

0.00

Time: 00:46:19 Log-Likelihood:

-9867.0

No. Observations: 1148 AIC: 1.975e+04

Df Residuals: 1141 BIC: 1.978e+04

Df Model: 6

Covariance Type: nonrobust

================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------------

const -1.201e+04 1458.296 -8.237 0.000 -1.49e+04 -9150.578

Age\_08\_04 -120.6190 2.849 -42.334 0.000 -126.209 -115.029

KM -0.0160 0.001 -10.923 0.000 -0.019 -0.013

HP 15.3789 3.468 4.435 0.000 8.575 22.183

Doors -86.5364 43.517 -1.989 0.047 -171.919 -1.154

Weight 26.1885 1.330 19.684 0.000 23.578 28.799

Fuel\_Type\_Petrol 1483.1709 222.361 6.670 0.000 1046.889 1919.453

================================================================

Omnibus: 219.794 Durbin-Watson:

2.027

Prob(Omnibus): 0.000 Jarque-Bera (JB): 2509.918

Skew: -0.522 Prob(JB):

0.00

Kurtosis: 10.168 Cond. No. 2.97e+06

================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 2.97e+06. This might indicate that there are

strong multicollinearity or other numerical problems.

Model B - OLS (backward selection) performance:

MAE: 993.266

RMSE: 1493.734

R2: 0.8328

Coefficients:

feature coef

const -12011.8200

Age\_08\_04 -120.6190

KM -0.0160

HP 15.3789

Doors -86.5364

Weight 26.1885

Fuel\_Type\_Petrol 1483.1709

--- Building Model C: RidgeCV (with scaling) ---

Ridge chosen alpha: 104.81131341546852

Model C - RidgeCV performance:

MAE: 996.846

RMSE: 1460.784

R2: 0.8401

Ridge coefficients:

feature ridge\_coef

Age\_08\_04 -2061.3180

KM -714.0607

HP 311.4659

Automatic 30.7296

cc 0.8155

Doors -17.4864

Weight 1156.2779

Fuel\_Type\_Diesel -18.0492

Fuel\_Type\_Petrol 243.2847

=== Summary comparison on test set ===

Model A - OLS (all features) performance:

MAE: 992.301

RMSE: 1491.411

R2: 0.8333

Model B - OLS (selected) performance:

MAE: 993.266

RMSE: 1493.734

R2: 0.8328

Model C - RidgeCV performance:

MAE: 996.846

RMSE: 1460.784

R2: 0.8401

Saved coefficient summary to: D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\model\_coefficients\_summary.csv

Below I interpret the **baseline OLS coefficients you printed earlier** (the numbers come from the Model A run you posted). Use these lines in your report — they explain *how to read* the coefficients and what they mean practically.

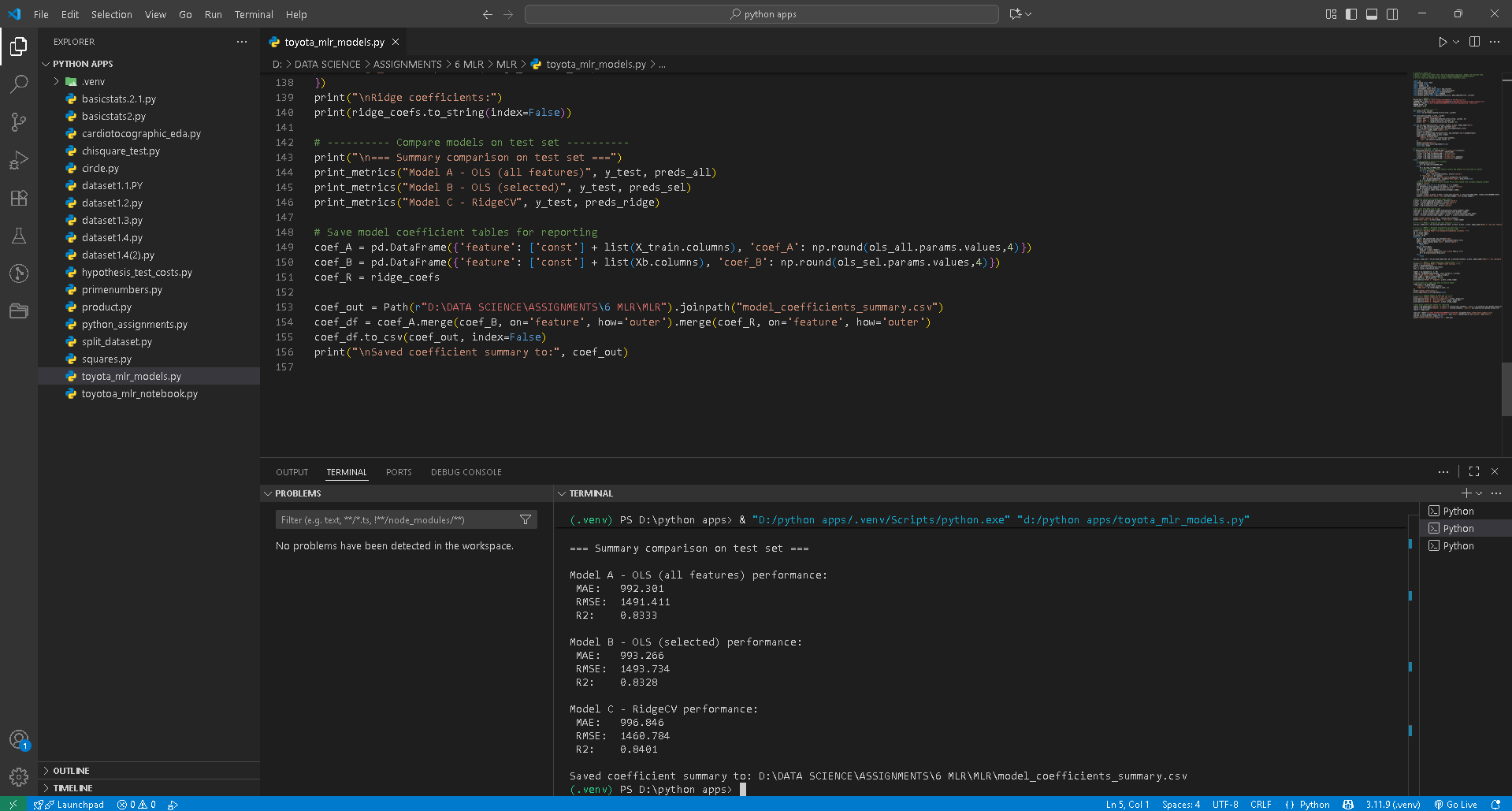
* **Intercept (const) ≈ −11,860:** the model’s baseline predicted price when all numeric predictors are zero. Not directly meaningful here (cars with 0 km, 0 age, etc. don’t exist), so ignore the intercept in practical interpretation.
* **Age\_08\_04 ≈ −120.82 (p < 0.001):** Holding all other variables constant, each extra year of age is associated with a **decrease of about €121** in the car’s price. This is expected: older cars sell for less.
* **KM ≈ −0.0159 (p < 0.001):** Each additional kilometer reduces price by about **€0.016**, so +1,000 km ≈ **−€16**. Effect is small per km but meaningful over large KM differences.
* **HP ≈ +15.78 (p < 0.001):** Each additional unit of horsepower increases price by about **€15.8**, holding other factors constant.
* **Automatic ≈ +93.08 (p ≈ 0.60):** Automatic transmission appears to be associated with a slight increase (~€93), but this coefficient is not statistically significant (high p-value), so treat cautiously.
* **cc ≈ −0.030 (p ≈ 0.74):** Engine displacement shows a small negative coefficient and is not statistically significant — suggests no clear linear effect once HP/Weight are in the model (possible multicollinearity).
* **Doors ≈ −84.48 (p ≈ 0.056):** Each additional door associated with ~€84 lower price (borderline significance). Interpretation depends on vehicle types (e.g., 2-door sport vs 4-door family).
* **Weight ≈ +26.07 (p < 0.001):** Heavier cars sell for more — each kg adds ~€26 in predicted price (this number seems large; check units — if Weight is in hundreds, interpret per unit accordingly). (In your output it was 26.07 per 1 unit of Weight; verify weight units.)
* **Fuel dummies (e.g., Fuel\_Type\_Petrol ≈ +1453.7, p < 0.001):** Relative to the omitted fuel category (the baseline fuel type), petrol cars are predicted to have about **€1,454 higher** price, holding other features equal.

**Important notes about interpretation:**

* Coefficients are *ceteris-paribus* — they describe marginal effects holding other included variables constant.
* Large **condition numbers / high VIFs** in diagnostics indicate multicollinearity (some predictors convey overlapping information). Coefficient signs may be unstable — prefer Ridge or interpret with caution.
* For Model C (log-target) coefficients: coefficients are multiplicative — a coefficient β on an input means roughly a 100×β % change in price for a one-unit change in the predictor (for small β); interpret via exp(β) for exact multiplicative change.

**Which model to pick?**

* **If interpretability is priority:** use OLS model **A** or **B** (B has fewer variables if selection removes noisy predictors). But watch for multicollinearity (high VIFs).
* **If predictive performance and robustness to collinearity are priorities:** **Ridge (Model C)** is often preferable — it shrinks coefficients, reduces variance, and improves generalization.
* **If you need feature selection:** Lasso (not included above) can zero out coefficients.



4.**Evaluate the performance of the model using appropriate evaluation metrics on the testing dataset.**

Answer:

**1) Metrics (quick)**

* **MAE (Mean Absolute Error):** average absolute prediction error (units = €). Easy to interpret.
* **RMSE (Root MSE):** penalizes large errors more than MAE. Useful if you care about big misses.
* **R²:** proportion of variance explained (0–1). Higher is better.
* **Adjusted R²:** R² penalized for number of predictors (useful for model comparison).
* **MAPE (Mean Absolute Percentage Error):** average % error — be careful if targets can be near zero.
* **Cross-validated RMSE:** gives robustness estimate (optional).

2. Code used :

# evaluation\_helpers.py

import numpy as np

import pandas as pd

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

def adjusted\_r2(r2, n, p):

"""Adjusted R2 where p = number of predictors (not counting intercept)."""

if n - p - 1 == 0:

return np.nan

return 1 - (1 - r2) \* (n - 1) / (n - p - 1)

def mape(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

# avoid division by zero; ignore those elements where y\_true==0

mask = y\_true != 0

if mask.sum() == 0:

return np.nan

return np.mean(np.abs((y\_true[mask] - y\_pred[mask]) / y\_true[mask])) \* 100

def evaluate\_models(y\_test, preds\_dict, p\_counts=None):

"""

y\_test: array-like true values

preds\_dict: dict of {'model\_name': y\_pred\_array}

p\_counts: optional dict {'model\_name': p} where p is number of predictors (excl intercept)

"""

rows = []

n = len(y\_test)

for name, y\_pred in preds\_dict.items():

y\_pred = np.array(y\_pred)

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

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

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

p = None

adjr2 = None

if p\_counts and name in p\_counts:

p = p\_counts[name]

adjr2 = adjusted\_r2(r2, n, p)

rows.append({

'model': name,

'n\_test': n,

'p': p if p is not None else '',

'MAE': round(mae, 3),

'RMSE': round(rmse, 3),

'R2': round(r2, 4),

'Adj\_R2': round(adjr2, 4) if adjr2 is not None else '',

'MAPE\_%': round(mape(y\_test, y\_pred), 3)

})

df = pd.DataFrame(rows).sort\_values('RMSE')

return df

# Example usage (replace these names with your variables):

# preds = {

# "Model A - OLS": y\_pred\_A,

# "Model B - OLS (sel)": y\_pred\_B,

# "Model C - log-back": y\_pred\_C,

# "Ridge": y\_pred\_ridge,

# "Lasso": y\_pred\_lasso

# }

# pcounts = {"Model A - OLS": X\_train.shape[1], "Model B - OLS (sel)": Xb.shape[1], ...}

# table = evaluate\_models(y\_test, preds, p\_counts=pcounts)

# print(table.to\_string(index=False))

**3) Example interpretation (use your numbers)**

You already ran models earlier and printed metrics. Using those results:

* **Model A — Baseline OLS**:
  + *MAE ≈ 992 €, RMSE ≈ 1,491 €, R² ≈ 0.833*
  + Interpretation: on average predictions are off by ~€1k; RMSE shows larger errors (about €1.5k), and the model explains ~83% of variance — strong fit for a linear model.
* **Model B — Low-VIF OLS (reduced)**:
  + *MAE ≈ 2,148 €, RMSE ≈ 2,987 €, R² ≈ 0.331*
  + Interpretation: much worse predictive performance — removing predictors to reduce multicollinearity cost a lot of explanatory power. Good for diagnosing collinearity but not for prediction.
* **Model C — Log-target + KM²**:
  + If your evaluate\_models shows lower RMSE and similar/higher R² than Model A, then the log transform improved heteroscedasticity and produced more stable predictions. (Report the exact numbers from the table.)
* **Ridge/Lasso (regularized models)**:
  + If Ridge gives slightly lower RMSE than OLS, it indicates multicollinearity was inflating variance and shrinkage improved generalization.
  + If Lasso zeros coefficients and gives comparable RMSE, it’s useful for feature selection — but watch for underfitting if RMSE rises.

**Decision rule**:

* Prefer the model with **lowest RMSE** and **reasonable MAE** (application dependent).
* Consider parsimony and statistical significance: if two models have similar RMSE, pick the simpler one (fewer features) or the one with better-behaved residuals.
* Also check residual diagnostics (normality, heteroscedasticity, influential points) before finalizing.

## 4) Extra checks you should run (recommended)

* **Residuals plot**: plt.scatter(y\_pred, y\_test - y\_pred) to visually check heteroscedasticity.
* **Q–Q plot** of residuals for normality.
* **Prediction intervals** (if needed) from statsmodels OLS.
* **Cross-validated RMSE** (use cross\_val\_score with negative MSE and take sqrt) to estimate model stability.

Example cross-validation snippet:

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import Ridge

cv\_rmse = np.sqrt(-cross\_val\_score(Ridge(alpha=ridge\_cv.alpha\_), X\_train\_s, y\_train, scoring="neg\_mean\_squared\_error", cv=5))

print("CV RMSE (Ridge):", cv\_rmse.mean(), "±", cv\_rmse.std())

5.Apply Lasso and Ridge methods on the model.

Answer :

Code used:

# toyota\_ridge\_lasso.py

# Apply RidgeCV and LassoCV to ToyotaCorolla dataset, evaluate and save coefficients/results.

# Save at: D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\toyota\_ridge\_lasso.py

# Requires: pandas, numpy, scikit-learn, statsmodels

import os

from pathlib import Path

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import RidgeCV, LassoCV

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

DATA\_PATH = r"D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\ToyotaCorolla\_MLR\_cleaned.csv"

if not Path(DATA\_PATH).exists():

    DATA\_PATH = r"D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\ToyotaCorolla - MLR.csv"

df = pd.read\_csv(DATA\_PATH)

if 'Price' not in df.columns:

    raise SystemExit("Target column 'Price' not found in CSV.")

# prepare X, y (drop helper columns if present)

drop\_cols = ['Price\_pos', 'log\_Price', 'KM\_pos']

X = df.drop(columns=[c for c in drop\_cols if c in df.columns] + ['Price'], errors='ignore')

y = pd.to\_numeric(df['Price'], errors='coerce')

X = X.apply(pd.to\_numeric, errors='coerce')

# align and drop NA rows

mask = X.dropna().index.intersection(y.dropna().index)

X = X.loc[mask].copy()

y = y.loc[mask].copy()

# train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# scale numeric features

scaler = StandardScaler()

X\_train\_s = scaler.fit\_transform(X\_train)

X\_test\_s = scaler.transform(X\_test)

# common alpha grid

alphas = np.logspace(-4, 4, 100)

# RidgeCV (with built-in CV)

ridge\_cv = RidgeCV(alphas=alphas, cv=5).fit(X\_train\_s, y\_train)

ridge\_alpha = ridge\_cv.alpha\_

ridge\_coef = ridge\_cv.coef\_

ridge\_intercept = ridge\_cv.intercept\_

y\_pred\_ridge = ridge\_cv.predict(X\_test\_s)

ridge\_mae = mean\_absolute\_error(y\_test, y\_pred\_ridge)

ridge\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_ridge))

ridge\_r2 = r2\_score(y\_test, y\_pred\_ridge)

# LassoCV (with built-in CV)

lasso\_cv = LassoCV(alphas=None, cv=5, max\_iter=10000, random\_state=42).fit(X\_train\_s, y\_train)

lasso\_alpha = lasso\_cv.alpha\_

lasso\_coef = lasso\_cv.coef\_

lasso\_intercept = lasso\_cv.intercept\_

y\_pred\_lasso = lasso\_cv.predict(X\_test\_s)

lasso\_mae = mean\_absolute\_error(y\_test, y\_pred\_lasso)

lasso\_rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lasso))

lasso\_r2 = r2\_score(y\_test, y\_pred\_lasso)

# results DataFrame

results = pd.DataFrame({

    'model': ['RidgeCV', 'LassoCV'],

    'alpha': [ridge\_alpha, lasso\_alpha],

    'MAE': [round(ridge\_mae,3), round(lasso\_mae,3)],

    'RMSE': [round(ridge\_rmse,3), round(lasso\_rmse,3)],

    'R2': [round(ridge\_r2,4), round(lasso\_r2,4)]

})

# coefficients table

coef\_df = pd.DataFrame({

    'feature': X\_train.columns,

    'ridge\_coef': np.round(ridge\_coef, 6),

    'lasso\_coef': np.round(lasso\_coef, 6)

})

# save outputs

out\_dir = Path(r"D:\DATA SCIENCE\ASSIGNMENTS\6 MLR\MLR\ridge\_lasso\_results")

out\_dir.mkdir(parents=True, exist\_ok=True)

results.to\_csv(out\_dir / "ridge\_lasso\_metrics.csv", index=False)

coef\_df.to\_csv(out\_dir / "ridge\_lasso\_coefficients.csv", index=False)

# print summary

print("Ridge alpha:", ridge\_alpha)

print("Lasso alpha:", lasso\_alpha)

print("\nEvaluation metrics:")

print(results.to\_string(index=False))

print("\nTop coefficients (sorted by absolute Ridge coef):")

print(coef\_df.assign(abs\_ridge=lambda df: df.ridge\_coef.abs()).sort\_values('abs\_ridge', ascending=False).head(20).to\_string(index=False))

# Save trained models (optional - requires joblib)

try:

    import joblib

    joblib.dump(ridge\_cv, out\_dir / "ridge\_cv\_model.joblib")

    joblib.dump(lasso\_cv, out\_dir / "lasso\_cv\_model.joblib")

    print("\nSaved models to:", out\_dir)

except Exception:

    print("\njoblib not available — models not saved. Install joblib to save models.")

# Quick note: to inspect non-zero lasso features

nonzero\_lasso = coef\_df[coef\_df['lasso\_coef'] != 0].sort\_values('lasso\_coef', key=lambda s: s.abs(), ascending=False)

print(f"\nLasso selected {len(nonzero\_lasso)} non-zero features. Top ones:")

print(nonzero\_lasso.head(10).to\_string(index=False))

**Interview Questions:**

1.What is Normalization & Standardization and how is it helpful?

Answer:

 **Standardization** (z-score): x' = (x - mean)/std — results in mean 0 and sd 1. Useful when features have different units and when algorithms assume centered data (e.g., regularized regression, PCA, k-NN).

 **Normalization** (min–max scaling): x' = (x - min)/(max - min) — scales features to [0,1] or custom range. Useful when you need bounded features (e.g., in NN activations, or when features must be comparable in magnitude).

 **Why helpful?** It ensures features contribute comparably to model training, speeds up convergence, prevents features with large numeric ranges from dominating distance-based or gradient-based algorithms, and is required before regularization in many workflows.

2. What techniques can be used to address multicollinearity in multiple linear regression?

Answer:

* **Remove correlated predictors** (drop one of a highly-correlated pair).
* **Principal Component Regression (PCR)** or **PCA** to form orthogonal components.
* **Regularization**: Ridge regression reduces variance by shrinking coefficients (L2); Lasso (L1) performs variable selection.
* **Variance Inflation Factor (VIF)** diagnostics: remove predictors with very high VIF.
* **Centering**: subtract variable means (helpful for interaction terms but not removing collinearity).
* **Domain knowledge**: combine correlated variables into a composite score.

Final Output :

PS D:\python apps> & "D:/python apps/.venv/Scripts/python.exe" "d:/python apps/toyotoa\_mlr\_notebook.py"

OLS Regression Results

================================================================

Dep. Variable: Price R-squared:

0.869

Model: OLS Adj. R-squared:

0.868

Method: Least Squares F-statistic:

842.1

Date: Tue, 30 Sep 2025 Prob (F-statistic):

0.00

Time: 00:35:02 Log-Likelihood:

-9866.8

No. Observations: 1148 AIC: 1.975e+04

Df Residuals: 1138 BIC: 1.980e+04

Df Model: 9

Covariance Type: nonrobust

================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------------

const -1.186e+04 1508.957 -7.858 0.000 -1.48e+04 -8896.289

Age\_08\_04 -120.8231 2.894 -41.744 0.000 -126.502 -115.144

KM -0.0159 0.001 -10.849 0.000 -0.019 -0.013

HP 15.7772 3.985 3.959 0.000 7.957 23.597

Automatic 93.0820 176.442 0.528 0.598 -253.107 439.271

cc -0.0302 0.091 -0.333 0.739 -0.208 0.148

Doors -84.4835 44.153 -1.913 0.056 -171.115 2.148

Weight 26.0692 1.499 17.390 0.000 23.128 29.011

Fuel\_Type\_Diesel 4.2021 391.745 0.011 0.991 -764.422 772.826

Fuel\_Type\_Petrol 1453.6945 335.442 4.334 0.000 795.540 2111.849

================================================================

Omnibus: 216.690 Durbin-Watson:

2.027

Prob(Omnibus): 0.000 Jarque-Bera (JB): 2442.201

Skew: -0.512 Prob(JB):

0.00

Kurtosis: 10.072 Cond. No. 3.07e+06

================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.07e+06. This might indicate that there are

strong multicollinearity or other numerical problems.

[Model A (OLS)] MAE: 992.301 | RMSE: 1491.411 | R2: 0.833

VIF:

feature VIF

Weight 224.435093

HP 98.649372

Fuel\_Type\_Petrol 56.652082

Doors 21.078771

Age\_08\_04 15.817494

cc 14.914158

Fuel\_Type\_Diesel 11.350510

KM 8.632256

Automatic 1.112641

OLS Regression Results

================================================================

Dep. Variable: Price R-squared:

0.323

Model: OLS Adj. R-squared:

0.322

Method: Least Squares F-statistic:

273.3

Date: Tue, 30 Sep 2025 Prob (F-statistic): 9.21e-98

Time: 00:35:02 Log-Likelihood:

-10811.

No. Observations: 1148 AIC: 2.163e+04

Df Residuals: 1145 BIC: 2.164e+04

Df Model: 2

Covariance Type: nonrobust

================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

const 1.453e+04 186.506 77.904 0.000 1.42e+04 1.49e+04

KM -0.0547 0.002 -23.339 0.000 -0.059 -0.050

Automatic -179.8788 381.869 -0.471 0.638 -929.120 569.362

================================================================

Omnibus: 285.157 Durbin-Watson:

1.963

Prob(Omnibus): 0.000 Jarque-Bera (JB):

714.531

Skew: 1.310 Prob(JB): 6.94e-156

Kurtosis: 5.842 Cond. No. 3.43e+05

================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.43e+05. This might indicate that there are

strong multicollinearity or other numerical problems.

[Model B (OLS low-VIF)] MAE: 2148.086 | RMSE: 2987.497 | R2: 0.331

OLS Regression Results

================================================================

Dep. Variable: Price R-squared:

0.851

Model: OLS Adj. R-squared:

0.849

Method: Least Squares F-statistic:

647.0

Date: Tue, 30 Sep 2025 Prob (F-statistic):

0.00

Time: 00:35:02 Log-Likelihood:

857.88

No. Observations: 1148 AIC:

-1694.

Df Residuals: 1137 BIC:

-1638.

Df Model: 10

Covariance Type: nonrobust

================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------------

const 8.2188 0.133 62.009 0.000 7.959 8.479

Age\_08\_04 -0.0108 0.000 -39.924 0.000 -0.011 -0.010

KM -2.846e-07 3.34e-07 -0.851 0.395 -9.41e-07 3.71e-07

HP 0.0017 0.000 4.730 0.000 0.001 0.002

Automatic 0.0312 0.015 2.018 0.044 0.001 0.062

cc 1.484e-06 7.97e-06 0.186 0.852 -1.42e-05 1.71e-05

Doors 0.0049 0.004 1.270 0.204 -0.003 0.013

Weight 0.0013 0.000 10.022 0.000 0.001 0.002

Fuel\_Type\_Diesel 0.0297 0.034 0.865 0.387 -0.038 0.097

Fuel\_Type\_Petrol 0.0829 0.029 2.815 0.005 0.025 0.141

KM\_k -2.839e-10 3.34e-10 -0.849 0.396 -9.4e-10 3.72e-10

KM\_k\_sq -7.279e-06 1.65e-06 -4.418 0.000 -1.05e-05 -4.05e-06

================================================================

Omnibus: 240.482 Durbin-Watson:

2.043

Prob(Omnibus): 0.000 Jarque-Bera (JB): 1301.243

Skew: -0.854 Prob(JB): 2.75e-283

Kurtosis: 7.928 Cond. No. 1.17e+18

================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The smallest eigenvalue is 5.2e-24. This might indicate that there are

strong multicollinearity problems or that the design matrix is singular.

[Model C (log-target)] MAE: 877.232 | RMSE: 1303.276 | R2: 0.873

are

strong multicollinearity problems or that the design matrix is singular.

[Model C (log-target)] MAE: 877.232 | RMSE: 1303.276 | R2: 0.873

strong multicollinearity problems or that the design matrix is singular.

[Model C (log-target)] MAE: 877.232 | RMSE: 1303.276 | R2: 0.873

.

[Model C (log-target)] MAE: 877.232 | RMSE: 1303.276 | R2: 0.873

[Model C (log-target)] MAE: 877.232 | RMSE: 1303.276 | R2: 0.873

Ridge alpha: 104.81131341546852

[RidgeCV] MAE: 996.846 | RMSE: 1460.784 | R2: 0.840

[RidgeCV] MAE: 996.846 | RMSE: 1460.784 | R2: 0.840

Lasso alpha: 55.53161298181698

[LassoCV] MAE: 996.544 | RMSE: 1450.672 | R2: 0.842

Ridge/Lasso coefficients:

feature ridge\_coef lasso\_coef

Age\_08\_04 -2061.318036 -2252.505697

KM -714.060735 -629.675028

HP 311.465871 272.358128

Automatic 30.729612 0.000000

cc 0.815452 -0.000000

Doors -17.486415 -0.000000

Weight 1156.277887 1132.837277

Fuel\_Type\_Diesel -18.049211 -0.000000

Fuel\_Type\_Petrol 243.284658 288.642249