Notebook Assignment 1: Data Prep Methods & RMSE Impact Study

Outliers · Missingness · Encoding · Transforms · Binning

AIM: 20LPA++

Deliverable: **One Jupyter Notebook**

Goal

Build a **single Jupyter notebook** that demonstrates, *one method per cell*, how classic data preparation techniques change downstream **Linear Regression (LR)** performance. For **each** method:

1. Add a **Markdown heading** (method name + short one-line purpose).
2. Write **code in its own cell** to apply the method on the training data (fit) and consistently transform validation/test.
3. Train LR on the transformed features and **print RMSE** (train/valid/test).
4. Log your result into a small **tracking table** (method/config → RMSE, #features).

Dataset and Protocol

- Use the provided tabular regression dataset (numeric + categorical + some dates + missing values + a few outliers).
- Split once: **train/valid/test = 60/20/20** with a fixed random seed. **Fit only on train**; apply to valid/test.
- Primary metric: **RMSE**. Also log MAE and R^2 if helpful.
- Baseline: **raw LR** (no scaling/encoding) to anchor comparisons.

Methods to Implement (one code cell each)

Outlier Detection/Handling

- Visual inspection (boxplot, scatter) *[plot + brief comment]*
- Z-Score rule
- IQR rule
- Percentile capping (winsorization)

Missing Value Techniques

- Listwise deletion (drop rows)

- Drop columns (threshold)
- Mean / Median / Mode imputation
- Constant value imputation (+ optional `was_missing` flag)
- k-NN imputation

Feature Engineering

- New feature creation
- Feature interactions (e.g., $X_i \times X_j$)
- Date/time extraction
- Aggregations (group-based stats) *[fit on train only]*

Encoding (Categoricals)

- Label encoding
- One-hot encoding
- Ordinal encoding
- Target/mean encoding (out-of-fold)
- Frequency / Count encoding

Transformations & Scaling

- Log transform
- Square-root
- Box–Cox
- Yeo–Johnson
- Z-score standardization
- Min–Max scaling
- Power transformation

Binning

- Equal-width bins
- Quantile bins
- Binary binning (high/low)

Notebook Structure (recommended)

1. Imports, seeding, small utility helpers.
2. Data load + single split (train/valid/test).
3. Baseline raw LR + RMSE.
4. For each method (section above): Markdown heading → method cell → LR + RMSE → append to tracker DataFrame.
5. Final comparison table + short commentary (what helped/hurt and why).

Result Logging Template

After each run, append a row to a shared results table:

Method (and config)	#Features	RMSE (Train)	RMSE (Valid)	RMSE (Test)
Baseline (raw LR)	87	9.42	9.57	9.61
Mean Imputation (numeric)	87	9.38	9.52	9.58
...	...	...	...	...

Sample Code (pattern to follow for every method)

Below is one complete example you should mirror for each technique.

Example method: Mean Value Imputation (Numeric) + LR

Listing 1: Sample cell: Mean Imputation + Linear Regression RMSE

```
# %% [markdown]
# ### Mean Value Imputation (Numeric) -> Linear Regression RMSE
# Compute numeric means on TRAIN, fill NA in train/valid/test consistently.
# Then train a simple LR (no scaling/encoding) and log RMSEs.

import numpy as np, pandas as pd
from typing import Dict, List
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# ---- assumptions from earlier cells ----
# df_train, df_valid, df_test  (each includes target column 'y')
# num_cols                     (list of numeric feature names)

def fit_numeric_means(df: pd.DataFrame, numeric_cols: List[str]) -> Dict[str, float
    ]:
    return {c: float(df[c].mean()) for c in numeric_cols}

def apply_mean_impute(df: pd.DataFrame, means: Dict[str, float]) -> pd.DataFrame:
    out = df.copy()
    for c, m in means.items():
        out[c] = out[c].fillna(m)
    return out

# --- 1) learn means on train ---
means = fit_numeric_means(df_train, num_cols)

# --- 2) transform splits identically ---
Xtr = apply_mean_impute(df_train[num_cols], means)
Xva = apply_mean_impute(df_valid[num_cols], means)
Xte = apply_mean_impute(df_test[num_cols],  means)
ytr, yva, yte = df_train['y'], df_valid['y'], df_test['y']

# --- 3) train + evaluate raw LR ---
lr = LinearRegression()
lr.fit(Xtr, ytr)

def rmse(m, X, y):
    preds = m.predict(X)
    return mean_squared_error(y, preds, squared=False)

```

```
rmse_train = rmse(lr, Xtr, ytr)
rmse_valid = rmse(lr, Xva, yva)
rmse_test  = rmse(lr, Xte, yte)

print({"method":"MeanImpute(numeric)",
       "rmse_train":rmse_train, "rmse_valid":rmse_valid, "rmse_test":rmse_test})

# --- 4) append to tracker (results_df) ---
row = {"method":"MeanImpute(numeric)",
       "n_features_after_prep": Xtr.shape[1],
       "rmse_train":rmse_train, "rmse_valid":rmse_valid, "rmse_test":rmse_test}
results_df = pd.concat([results_df, pd.DataFrame([row])], ignore_index=True)
```

Replicate for all other methods. For each technique (e.g., Z-Score, IQR, kNN imputation, target encoding, Box–Cox, binning, etc.), keep the *same pattern*:

1. Fit method-specific statistics on **train**.
2. Apply to valid/test.
3. Train LR (no scaling/encoding), compute RMSEs, append to `results_df`.