Assignment 2 — Part 1: Linear Regression (HEAPO)

This notebook part 1 fits the specified linear model:

Consumption (kWh) = $\theta 0$ + $\theta 1$·Temperature_avg + $\theta 2$·Humidity_avg + $\theta 3$·Sunshine_Hours

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

DATA_PATH = 'data/heapo_cleaned_dataset.csv'

df = pd.read_csv(DATA_PATH, parse_dates=['date'])
df.head()
```

	date	temperature_avg	humidity_avg	sunshine_hours	consumption_kWh
0	2019-03-02	7.5	77.1	1.0	18.33
1	2019-03-03	9.9	60.3	6.3	15.03
2	2019-03-04	8.3	57.4	0.5	16.69
3	2019-03-05	7.4	60.2	6.7	29.52
4	2019-03-06	6.0	68.2	2.7	16.81

```
# Features and label as specified in the prompt
X = df[['temperature_avg', 'humidity_avg', 'sunshine_hours']]
y = df['consumption_kWh']

linreg = LinearRegression()
linreg.fit(X, y)

coef_names = ['theta_1 (Temp_avg)', 'theta_2 (Humidity_avg)', 'theta_3 (Sunshine_Hours)']
coefs = pd.Series(linreg.coef_, index=coef_names)
intercept = linreg.intercept_

print('θ0 (intercept):', round(intercept, 4))
coefs.round(4)
```

```
θ0 (intercept): 27.4531

theta_1 (Temp_avg)          -0.9496
theta_2 (Humidity_avg)       0.0404
theta_3 (Sunshine_Hours)    -0.5234
dtype: float64
```

Interpretation (part a)

- θ0 is the expected daily consumption when all features are 0.
- θ1 is the change in kWh per 1°C increase in average temperature (holding others fixed).
- θ2 is the change in kWh per +1% humidity.
- θ3 is the change in kWh per additional sunshine hour.

```
# Part (b): 90/10 split, train on 90%, evaluate on both
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.10, random_state=4300
)

linreg_split = LinearRegression().fit(X_train, y_train)

# Predictions
train_pred = linreg_split.predict(X_train)
test_pred = linreg_split.predict(X_test)

# Metrics
mse_train = mean_squared_error(y_train, train_pred)
r2_train = r2_score(y_train, train_pred)

mse_test = mean_squared_error(y_test, test_pred)
r2_test = r2_score(y_test, test_pred)

print('Training MSE:', round(mse_train, 4), '| R^2:', round(r2_train, 4))
print('Test     MSE:', round(mse_test, 4), '| R^2:', round(r2_test, 4))
```

```
Training MSE: 96.6871 | R^2: 0.4342
Test     MSE: 111.1049 | R^2: 0.2362
```

Part (b)

- Training: MSE = 96.6871, R^2 = 0.4342.
- Test: MSE = 111.1049, R^2 = 0.2362.

The model performs worse on the test set than on the training set (higher MSE, lower R^2). This indicates overfitting and that a simple linear specification misses seasonal and household/behavioral factors; for time-ordered data, a chronological split is more appropriate than a random split.

Part 2: Logistic Regression (Pima Indians Diabetes)

We will train Logistic Regression models with C ∊ {0.01, 0.1, 1, 10, 100} for both L1 and L2 penalties using a standardized feature pipeline, then report performance and coefficient behavior. this is part (a)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report

# Load
PIMA_PATH = 'data/Pima_Indians_Diabetes.csv'
pima = pd.read_csv(PIMA_PATH)

X = pima.drop(columns=['Outcome'])
y = pima['Outcome']

# Train/test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=4300, stratify=y
)

Cs = [0.01, 0.1, 1, 10, 100]
penalties = ['l1', 'l2']

results = []
coefs = {}

for pen in penalties:
    # 'liblinear' supports both L1 and L2 for binary classification
    solver = 'liblinear'
    for C in Cs:
        pipe = Pipeline([
            ('scaler', StandardScaler()),
            ('clf', LogisticRegression(penalty=pen, C=C, solver=solver, max_iter=2000, random_
        ])
        pipe.fit(X_train, y_train)

        # Metrics
        y_train_pred = pipe.predict(X_train)
        y_test_pred = pipe.predict(X_test)
        y_train_proba = pipe.predict_proba(X_train)[:, 1]
        y_test_proba = pipe.predict_proba(X_test)[:, 1]

        acc_train = accuracy_score(y_train, y_train_pred)
        acc_test = accuracy_score(y_test, y_test_pred)
        auc_train = roc_auc_score(y_train, y_train_proba)
        auc_test = roc_auc_score(y_test, y_test_proba)

        results.append({'penalty': pen, 'C': C,
                        'acc_train': acc_train, 'acc_test': acc_test,
                        'auc_train': auc_train, 'auc_test': auc_test})

        # Store coefficients (in original feature order)
        clf = pipe.named_steps['clf']
        coefs[(pen, C)] = pd.Series(clf.coef_.ravel(), index=X.columns)

results_df = pd.DataFrame(results).sort_values(by=['penalty', 'C']).reset_index(drop=True)
results_df
```

	penalty	C	acc_train	acc_test	auc_train	auc_test
0	l1	0.01	0.701954	0.714286	0.784352	0.804074
1	l1	0.10	0.773616	0.792208	0.838411	0.837037
2	l1	1.00	0.783388	0.779221	0.840841	0.831296
3	l1	10.00	0.783388	0.779221	0.840643	0.830926
4	l1	100.00	0.783388	0.779221	0.840584	0.830741
5	l2	0.01	0.775244	0.746753	0.833914	0.823519
6	l2	0.10	0.778502	0.785714	0.840432	0.831296
7	l2	1.00	0.783388	0.779221	0.840689	0.830926
8	l2	10.00	0.783388	0.779221	0.840572	0.830741
9	l2	100.00	0.783388	0.779221	0.840584	0.830741

```
# Summarize best per penalty by test AUC, then by accuracy
best_by_penalty = results_df.sort_values(['penalty','auc_test','acc_test'], ascending=[True, F
                              .groupby('penalty').head(1).reset_index(drop=True)
best_by_penalty
```

	penalty	C	acc_train	acc_test	auc_train	auc_test
0	l1	0.1	0.773616	0.792208	0.838411	0.837037
1	l2	0.1	0.778502	0.785714	0.840432	0.831296

```
# Show coefficient magnitudes vs C for L1 and L2 to illustrate sparsity/regularization strengt
coef_summary = []
for pen in penalties:
    for C in Cs:
        s = coefs[(pen, C)].abs()
        coef_summary.append({'penalty': pen, 'C': C,
                             'nonzero': int((coefs[(pen, C)] != 0).sum()),
                             'L1_norm': s.sum(), 'L2_norm': np.sqrt((coefs[(pen, C)]**2).sum()
coef_df = pd.DataFrame(coef_summary).sort_values(['penalty','C'])
coef_df
```

	penalty	C	nonzero	L1_norm	L2_norm
0	l1	0.01	1	0.223137	0.223137
1	l1	0.10	6	2.157978	1.106992
2	l1	1.00	8	3.045049	1.412557
3	l1	10.00	8	3.165268	1.454367
4	l1	100.00	8	3.177538	1.458661
5	l2	0.01	8	1.438794	0.647056
6	l2	0.10	8	2.650539	1.225419
7	l2	1.00	8	3.110030	1.428465
8	l2	10.00	8	3.171722	1.455960
9	l2	100.00	8	3.178121	1.458815

Part b

- Best L1 (by test AUC then accuracy): C = 0.1 with AUC ≈ 0.8370 and Accuracy ≈ 0.7922.
- Best L2 (by test AUC then accuracy): C = 0.1 with AUC ≈ 0.8313 and Accuracy ≈ 0.7857.
- As C increases (weaker regularization), both L1 and L2 generally increase coefficient magnitudes; L1 drives sparsity (fewer nonzero weights), L2 keeps all features with smaller magnitudes. Smaller C (stronger regularization) shrinks weights more and can improve generalization; overly large C can overfit.

Part 3: Model Evaluation — Confusion and Cost

We compute accuracy for M1 and M2 and total cost using the provided cost matrix.

```
import numpy as np

# These are the confusion matrices as given:
# For both, assume rows are Actual [class=1, class=0], columns are Predicted [class=1, class=0
M1 = np.array([[150, 40],
               [ 60,250]])
M2 = np.array([[250, 45],
               [  5,200]])

# Accuracy
acc_M1 = (M1[0,0] + M1[1,1]) / M1.sum()
acc_M2 = (M2[0,0] + M2[1,1]) / M2.sum()

# Cost matrix (rows: actual [pos, neg]; columns: predicted [pos, neg])
# cost(actual=Positive, predicted=Positive) = -1
# cost(actual=Positive, predicted=Negative) = 100
# cost(actual=Negative, predicted=Positive) = 1
```

```
# cost(actual=Negative, predicted=Negative) = 0
C = np.array([[-1, 100],
              [ 1,   0]])

total_cost_M1 = float((M1 * C).sum())
total_cost_M2 = float((M2 * C).sum())

acc_M1, acc_M2, total_cost_M1, total_cost_M2
```

```
(np.float64(0.8), np.float64(0.9), 3910.0, 4255.0)
```

Part 3 — Direct answers

(a) Accuracy

- M1: 0.80 (400/500)
- M2: 0.90 (450/500)

(b) Total cost (using provided cost matrix)

- M1: 3910
- M2: 4255

(c) Why higher accuracy but worse cost for M2?

- M2 favors predicting positives, which raises accuracy but racks up many false positives. Given the cost matrix, those extra false positives and the remaining false negatives outweigh its gains, so total cost is higher.

(d) When is accuracy misleading, and what would you look at instead?

- Rare disease screening: A model can be “accurate” by calling almost everyone healthy. I’d care most about catching the sick people, so I’d focus on recall and a sensible balance with precision.
- Spam filtering: A model can look accurate but flag a lot of legitimate emails as spam. Here I’d emphasize precision for the spam class to avoid false alarms, while keeping recall reasonable.