In-Class Exercises: Solutions

Instructor copy – do not distribute before the exercise session.

Data Science with Python – BSc Course

Exercise 1: OLS by Hand — Solution

(a) Means

$$\bar{x} = \frac{1+2+3+4+5}{5} = \frac{15}{5} = 3$$

$$\bar{y} = \frac{2.0+3.5+5.0+6.2+7.8}{5} = \frac{24.5}{5} = 4.9$$

(b) Working table

x_i	y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
1	2.0	−2	−2.9	5.8	4
2	3.5	−1	−1.4	1.4	1
3	5.0	0	0.1	0.0	0
4	6.2	1	1.3	1.3	1
5	7.8	2	2.9	5.8	4
Sum		0	0	**14.3**	**10**

(c) Estimates

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{14.3}{10} = 1.43$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} = 4.9 - 1.43 \times 3 = 4.9 - 4.29 = 0.61$$

Fitted line: $\hat{y} = 0.61 + 1.43x$.

(d) Prediction at $x = 6$

$$\hat{y}(6) = 0.61 + 1.43 \times 6 = 0.61 + 8.58 = 9.19$$

If the true value is $y_6 = 9.2$, the residual is

$$e_6 = y_6 - \hat{y}(6) = 9.2 - 9.19 = 0.01.$$

The line predicts the sixth point almost exactly — the training data was generated from a nearly linear relationship.

Key takeaway

OLS picks the unique $(\hat{\beta}_0, \hat{\beta}_1)$ that minimizes the sum of squared vertical residuals. The closed-form formulas give the answer in a single pass with no iteration.

Exercise 2: Read the Coefficient Path — Solution

Answers reference the ridge path plot from slide 22. The model answers below describe the qualitative behaviour typical of such plots.

(a) Direction as λ grows: All coefficients shrink *toward zero* as λ increases. The ridge penalty $\lambda \sum \beta_j^2$ penalizes large coefficient magnitudes, so larger λ pushes all $\hat{\beta}_j$ closer to zero. At $\lambda = 0$ the solution equals OLS; at $\lambda \to \infty$ all coefficients approach zero.

(b) Largest absolute coefficient at small λ: The coefficient with the largest magnitude corresponds to the feature that (1) has strong correlation with the target and (2) is not absorbed by other correlated features. In practice this is the feature most strongly related to y *after accounting for the others.*

(c) When do all coefficients vanish? There is no finite λ at which Ridge sets coefficients *exactly* to zero. Practically, for λ large enough (often $\lambda > 10^3$ times the range of the data), coefficients become indistinguishable from zero. The resulting model predicts the *mean of* y for every input — a constant intercept with no feature dependence.

(d) Ridge vs. Lasso in practice:

- Ridge smoothly shrinks coefficients but keeps all features in the model. Use when all features are believed relevant but you want to reduce variance.
- Lasso drops features by setting coefficients exactly to zero. The L1 penalty has sharp "corners" at zero in the constraint region, so the optimum often lies on an axis (one or more $\beta_j = 0$). Use when you expect many irrelevant features and want automatic feature selection.

Discussion note

The coefficient path is the best visual tool to build intuition for regularization. Point out: the order in which Lasso coefficients become non-zero (as λ decreases from large to small) gives a "priority ranking" of features.

Exercise 3: Compute Precision, Recall, F1 — Solution

(a) Metrics

Given TP = 80, FP = 20, FN = 15, TN = 885.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{80}{80 + 20} = \frac{80}{100} = 0.80$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{80}{80 + 15} = \frac{80}{95} \approx 0.842$$

$$\text{F1} = 2 \cdot \frac{0.80 \times 0.842}{0.80 + 0.842} = 2 \cdot \frac{0.6737}{1.642} \approx 0.82$$

$$\text{Accuracy} = \frac{TP + TN}{\text{total}} = \frac{80 + 885}{1000} = \frac{965}{1000} = 0.965$$

Summary: Precision = 0.80, Recall $\approx$ 0.84, F1 $\approx$ 0.82, Accuracy = 0.965.

(b) Baseline: always predict "legit"

Predicting every transaction as legitimate gives:

- TP = 0, FP = 0, FN = 95, TN = 905
- Accuracy = $905/1000 = 0.905 = 90.5\%$
- Precision = $0/(0+0)$ — *undefined*; usually reported as 0.
- Recall = $0/(0+95) = 0$
- F1 = 0

The classifier's 96.5% accuracy beats the baseline's 90.5% by only 6 percentage points. The real improvement lies in F1 (from 0 to 0.82), which reflects the model's ability to *actually find the fraud*.

(c) High recall vs. high precision

False negatives cost 10× more than false positives, so missing fraud is the dominant concern. → **Prefer high recall.**

In practice:

- Lower the decision threshold to flag more transactions as fraud. This raises recall (catches more fraud) at the cost of precision (more false alarms).
- The *threshold-moving* slide in L28 shows this trade-off directly.
- An alternative summary metric is the F_β score with $\beta > 1$, which weights recall more heavily than precision.

Key takeaway

Accuracy hides the cost structure of the problem. On imbalanced data with asymmetric errors, precision and recall must be reported *separately*, and the decision threshold should be tuned to match the business cost function.

Exercise 4: Which Model? — Solution

	Scenario	Best Method	Reasoning
A	Continuous return, small clean feature set, linear	**Linear Regression**	Exactly the setting OLS was designed for; small set means low variance without regularization
B	Loan default, explainability required	**Logistic Regression** or **Decision Tree**	Both produce human-readable rules; logistic gives coefficients per feature, tree gives a splitting chart
C	Fraud detection, 1% positive class	**SMOTE + XGBoost**	SMOTE rebalances the training data; XGBoost delivers strong performance on tabular, mildly non-linear signals
D	Revenue prediction, 50 correlated features	**Lasso**	L1 penalty selects a sparse subset from correlated candidates, avoids OLS instability

Bonus: Why plain logistic regression fails in scenario C

With a 1% positive class, plain logistic regression (with default threshold 0.5) rarely emits a positive prediction — the maximum likelihood objective is dominated by the 99% majority. The model converges to a classifier that almost always says "legit," achieving 99% accuracy but ≈ 0 recall.

What SMOTE fixes: SMOTE (Synthetic Minority Oversampling Technique) creates synthetic fraud examples by interpolating between existing fraud cases. After SMOTE, the training set is (for example) 50% fraud / 50% legit. The classifier now has enough positive examples to learn the fraud patterns.

Why XGBoost on top: XGBoost builds shallow trees sequentially, each focusing on the residuals from previous trees. For fraud detection:

- Captures non-linear feature interactions better than logistic regression.
- Handles mixed numeric and categorical features natively.
- Built-in regularization via γ and λ hyper- parameters.
- Typically the strongest single-model performer on tabular competitions.

Alternative answers worth full credit

- Scenario A: Ridge regression is also acceptable if there is any correlation among the factors.

- Scenario B: Random Forest can be acceptable if paired with SHAP values for explainability.
- Scenario C: class-weighted logistic regression or threshold- moving also work; students who suggest these deserve credit.
- Scenario D: ElasticNet is equally valid — it combines L1 (sparse selection) and L2 (handles correlated groups).

Exercise 5: Build a Classifier Pipeline — Solution

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures  # (1)(2)
from sklearn.linear_model import LogisticRegression                   # (3)
from sklearn.model_selection import train_test_split
import numpy as np

np.random.seed(42)
X = np.random.randn(500, 3)
y = (X[:, 0] + X[:, 1] - X[:, 2] + 0.3 * np.random.randn(500) > 0).astype(int)

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

pipe = Pipeline([
    ('scaler', StandardScaler()),                                     # (4)
    ('poly',   PolynomialFeatures(degree=2, include_bias=False)),     # (5)
    ('clf',    LogisticRegression(C=1.0, max_iter=1000)),             # (6)
])

pipe.fit(X_train, y_train)                                            # (7)

train_acc = pipe.score(X_train, y_train)
test_acc  = pipe.score(X_test,  y_test)
print(f"Train accuracy: {train_acc:.3f}")
print(f"Test accuracy : {test_acc:.3f}")
```

Blank answers

Blank	Answer
(1)	`StandardScaler`
(2)	`PolynomialFeatures`
(3)	`LogisticRegression`
(4)	`StandardScaler()`
(5)	`PolynomialFeatures(degree=2, include_bias=False)`
(6)	`LogisticRegression(C=1.0, max_iter=1000)`
(7)	`fit`

(b) Why order matters

If you apply `PolynomialFeatures` *before* `StandardScaler`, the squared and interaction terms $(x_1^2, x_1x_2, \ldots)$ are computed on raw, unscaled inputs. These squared terms can have wildly different magnitudes from the linear terms, and the logistic regression fits on that distorted scale. After `StandardScaler` runs first, all linear features have mean 0 and standard deviation 1, so the resulting polynomial terms stay in a comparable range. Numerically, this improves optimizer convergence and avoids one squared feature dominating the gradient.

(c) Regularization strength in `LogisticRegression`

`C` is the *inverse* of the regularization strength λ. Therefore:

- `C = 0.01` $\Rightarrow$ strong regularization (coefficients shrunk aggressively).
- `C = 100` $\Rightarrow$ weak regularization (nearly unregularized logistic regression).

Mnemonic: "low C = lots of shrinkage."

Exercise 6: Random Forest Feature Importance — Solution

(a) Top features

The target was generated as

$$\text{logits} = 2.5 \cdot \text{debt\_ratio} - 0.01 \cdot (\text{credit\_score} - 700) + \text{noise}.$$

So only two features actually drive `default`: `debt_ratio` and `credit_score`.

Running the code, the Random Forest's `feature_importances_` ranks these two at the top, typically with values in the range 0.25–0.45 each, while the unrelated features (age, employ_yrs, dependents, income) each sit near 0.05–0.15. The forest correctly recovers the generative structure.

(b) Low importance despite many splits

`age` and `dependents` are random noise with respect to `default`. The trees will sometimes split on them because:

- Each split is chosen to minimize impurity on the *current node*, and a random split can yield a marginal impurity gain on a small subsample.
- Random Forest uses `max_features` (sqrt of total features by default), so unrelated features are forced into some splits.

But the *gain* at those splits is small, so the feature importance (summed impurity decrease, weighted by the number of samples at each split) stays low.

(c) Fairer alternatives

Default "Gini" feature importance inflates:

- High-cardinality numeric features (income, credit_score) that offer many possible splits.
- Features correlated with other important features.

Better alternatives:

1. **Permutation importance**: randomly shuffle one column at a time and measure how much test accuracy drops. Model- agnostic and unbiased by cardinality. (`sklearn.inspection.permutation_imp`
2. **SHAP values**: compute each feature's marginal contribution to each prediction via cooperative game theory. Introduces per-sample, per-feature attributions.
3. **Drop-column importance**: refit the model without each feature and measure performance drop. Accurate but expensive.

L26 and L32 of the course cover these alternatives; the slide on "Feature importance caveats" is the anchor reference.

Key takeaway

Random Forest feature importance is a fast first-pass ranking, but for high-stakes decisions (credit, medical, legal), cross-check with permutation importance or SHAP before drawing conclusions about "which features matter."