Lab 2 — Exploring Bias, Variance, and Model Evaluation

Time: 60–90 minutes
Difficulty: Beginner → Intermediate
What you'll learn

- Train/test/validation split (why not just one split?)
- Understand underfitting vs overfitting through polynomial regression
- Compare models with different complexity
- Evaluate using multiple metrics (MSE, R^2)
- Visualize learning curves

Prerequisites

- Completion of Lab 1
- Google Colab account (or local Python with `scikit-learn`, `matplotlib`, `numpy`, `pandas` installed)

Deliverables

1. A Colab notebook with executed cells.
2. Plots showing underfit/overfit behavior.
3. A short reflection (5–8 bullet points) on bias, variance, and generalization.

Step-by-step Instructions

Step 0 — Notebook setup

Open a new notebook in Colab and rename it:

`ML-AI-Foundations-Section2-Lab2-YourName`

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import mean_squared_error, r2_score

SEED = 42
np.random.seed(SEED)
```

Step 1 — Generate synthetic data

We'll create noisy quadratic data so learners can experiment with model complexity.

```
# True function: y = 0.5x^2 + x + 2 + noise
X = np.linspace(-3, 3, 100).reshape(-1, 1)
y = 0.5*X**2 + X + 2 + np.random.randn(100, 1)

plt.scatter(X, y, alpha=0.7)
plt.title("Synthetic Data (Quadratic with Noise)")
plt.show()
```

Step 2 — Train/test split

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

Step 3 — Build a helper function for training/evaluation

```
def train_and_evaluate(degree):
    model = Pipeline([
        ("poly", PolynomialFeatures(degree=degree, include_bias=False)),
        ("scaler", StandardScaler()),
        ("linreg", LinearRegression())
    ])

    model.fit(X_train, y_train)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_pred_train)
    test_mse = mean_squared_error(y_test, y_pred_test)
    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)

    print(f"Degree {degree}: Train MSE={train_mse:.3f}, Test MSE={test_mse:.3f

    # Plot predictions vs data
    X_range = np.linspace(-3, 3, 100).reshape(-1, 1)
    y_pred_range = model.predict(X_range)
    plt.scatter(X_train, y_train, label="Train", alpha=0.6)
    plt.scatter(X_test, y_test, label="Test", alpha=0.6)
    plt.plot(X_range, y_pred_range, color="red", linewidth=2, label=f"Model (c
    plt.legend()
    plt.title(f"Polynomial Regression (degree {degree})")
    plt.show()
```

Step 4 — Try different model complexities

```
for d in [1, 2, 10]:
    train_and_evaluate(d)
```

- **Degree 1 (linear):** underfits → poor fit, low R^2
- **Degree 2:** good fit → close to true function
- **Degree 10:** overfits → train perfect, test poor

Step 5 — Learning curve visualization

Let's see how model performance evolves with more data.

```
train_sizes, train_scores, test_scores = learning_curve(
    Pipeline([
        ("poly", PolynomialFeatures(degree=10, include_bias=False)),
        ("scaler", StandardScaler()),
        ("linreg", LinearRegression())
    ]),
    X, y.ravel(), cv=5,
    scoring="neg_mean_squared_error",
    train_sizes=np.linspace(0.1, 1.0, 10),
    random_state=SEED
)

train_mean = -train_scores.mean(axis=1)
test_mean = -test_scores.mean(axis=1)

plt.plot(train_sizes, train_mean, "o-", label="Training error")
plt.plot(train_sizes, test_mean, "o-", label="Validation error")
plt.xlabel("Training set size")
plt.ylabel("MSE")
plt.title("Learning Curve (Degree=10)")
plt.legend()
```

```
plt.show()
```

Expected: Training error very low, validation error high → variance problem.

Step 6 — Compare metrics (MSE vs R^2)

Add a markdown cell:

- **MSE (Mean Squared Error):** lower is better; penalizes large errors.
- **R^2 (coefficient of determination):** closer to 1 = better; <0 means worse than baseline.

Step 7 — Reflection Questions

Answer in markdown:

1. How did **degree 1 vs 2 vs 10** models differ in fit?
2. Which one underfit? Which overfit? Why?
3. What did the learning curve show about bias vs variance?
4. How does **MSE** compare to **R^2** as an evaluation metric?
5. What steps could reduce overfitting? (e.g., regularization, more data, simpler model)

Step 8 — Save and submit

- Save notebook to Google Drive.
- Export `.ipynb` and upload as per course instructions.