Part 2 — Logistic Regression from Scratch

PDAN8412

ST10458509

Rick Jackson

30 October 2025

Emeris Cape Town

Table of Contents

Introduction

Top Goodreads Books Collection (1980–2023) dataset sourced from Kaggle has been selceted for this project.

www.kaggle.com/datasets/jealousleopard/goodreadsbooks

The 'client' is a **book press** that wants to predict **whether a book will be a bestseller**. This dataset contains features directly tied to **book success indicators** — such as:

- **Average rating (`Rating`)** → Quality perception
- **Number of voters (`Number of Voters`)** → Popularity
- **"Want to Read" and "Current Readers"** → Market interest and engagement
- **Genres, Publisher, Price, Pages** → Book characteristics affecting sales

These map perfectly to what a press might want to analyze *before release* or *for similar upcoming titles*.

- Clean structure (each row = one book)
- Good balance of **categorical**, **numerical**, and **textual** data
- No obvious privacy or licensing issues
- From a reputable data platform (Kaggle)

Dataset Overview

Columns:

> ISBN, Title, Series info, Authors, Publisher, Language, Description, Num Pages, Format, Genres, Publication Date, Rating, Number of Voters, Current Readers, Want to Read, Price, URL

Initial Cleaning Notes:

`num_pages` converted to numeric from strings with regex.

Cleaning the 'num_pages' column

```
# Handling all values as though they were string.

df['num_pages'] = df['num_pages'].astype(str)
```

```
# Checking whether conversion is successful from object

df['num_pages'].apply(type).value_counts()
```

Missing numeric values (`num_pages`, `price`, `current_readers`, `want_to_read`) filled with median.

Filling missing values

Filling missing values in the columns such as publisher/language/format with 'Unknown'

```
# Fill missing publisher with 'Unknown'

df['publisher'] = df['publisher'].fillna('Unknown')
```

```
# Fill missing language with 'Unknown'

df['language'] = df['language'].fillna('Unknown')
```

```
# Fill missing format with 'Unknown'

df['format'] = df['format'].fillna('Unknown')
```

Converted types: `num_pages` → int, `price`, `current_readers`, `want_to_read` → float.

Missing Values Summary

Column	Missing
isbn	594
series_title	2132
series_release_number	2144
publisher	116
language	26
description	2
num_pages	21
format	35

Column	Missing
current_readers	104
want_to_read	39
price	233

Exploratory Data Analysis (EDA)

Checked for skewed distributions.

Visualizing skewness:

```
import seaborn as sns
import matplotlib.pyplot as plt

num_features = ['num_pages', 'price', 'current_readers', 'want_to_read']
for col in num_features:
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.show()
```

Calculated correlations between numeric features and `bestseller`.

Observed outliers and handled scaling for numeric features.

Correlation heatmap.

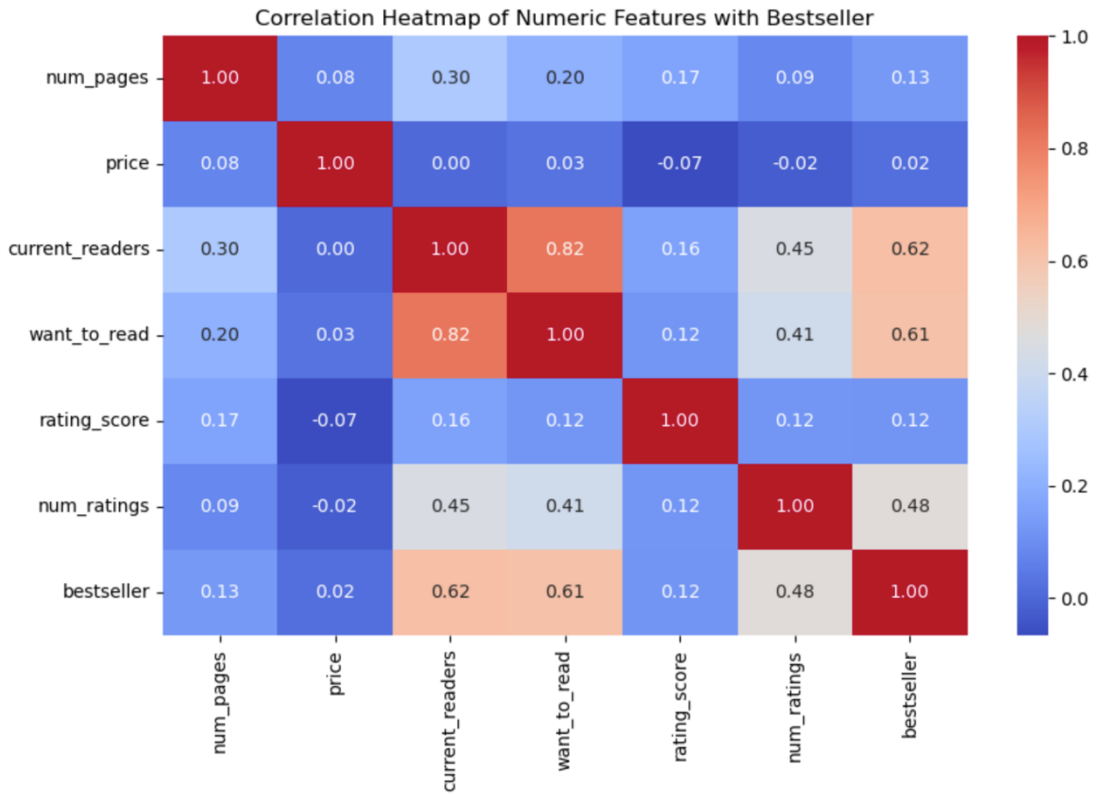
Boxplots for numeric features pre- and post-scaling.

Feature Engineering

Bestseller target variable:

`df['bestseller'] = (df['rating_score'] >= 4.0).astype(int)`

Author popularity: Mean ratings or counts of books per author.



Interaction features: e.g., `num_pages × current_readers`, `price × num_ratings`.

Descriptiong of features created:

1. **Bestseller target variable** (1 if rating ≥ threshold, else 0).
2. **Author popularity** (mean ratings or counts).
3. **Interaction features** (e.g., `num_pages × current_readers`).

Modeling Approach

The dataset is split into into 70% train, 30% test.

Logistic regression performed from scratch:

Forward propagation (sigmoid function).

Cost function (log-loss).

Backward propagation (gradients).

Gradient descent update.

```
# ----------------------------
    # STEP 1: Forward Propagation
    # ----------------------------

    m = X.shape[0]  # number of samples

    # Compute the linear combination of features and weights
    # z = Xw + b
    z = np.dot(X, w) + b

    # Apply the sigmoid activation function to get predicted probabilities
    # A = 1 / (1 + e^(-z))
    A = 1 / (1 + np.exp(-z))

    # Compute the cost function (log-loss)
    # Add a small epsilon (1e-9) to prevent log(0) errors
    cost = -(1/m) * np.sum(y * np.log(A + 1e-9) + (1 - y) * np.log(1 - A + 1e-9))

    # ----------------------------
    # STEP 2: Backward Propagation
    # ----------------------------

    # Compute the gradients of the cost function
    dw = (1/m) * np.dot(X.T, (A - y))  # Gradient w.r.t. weights
    db = (1/m) * np.sum(A - y)         # Gradient w.r.t. bias

    # ----------------------------
    # STEP 3: Store and Return Results
    # ----------------------------
    grads = {"dw": dw, "db": db}

    return grads, cost
```

Hyperparameters used learning rate, iterations and iterations are played to tune model.

```
# Split
from sklearn.model_selection import train_test_split
X_train_b, X_test_b, y_train_b, y_test_b = train_test_split(
    X_base, y, test_size=0.3, random_state=42, stratify=y
)

# Initialize weights and bias
w_b = np.zeros((X_train_b.shape[1], 1))
b_b = 0

# Logistic regression training (same gradient descent already built)
def model(X, y, w, b, num_iterations=1000, learning_rate=0.01):
    costs = []
    for i in range(num_iterations):
        grads, cost = propagate(w, b, X, y)
        w -= learning_rate * grads["dw"]
        b -= learning_rate * grads["db"]
        if i % 100 == 0:
            costs.append(cost)
    return w, b, costs

# Train baseline model
w_b, b_b, costs_b = model(X_train_b, y_train_b, w_b, b_b)
```

Scaling approach was then used for baseline model and the feature engineered model and the accuracy shot up considerably.

```
scaler = StandardScaler()
X_enh_scaled = scaler.fit_transform(df[enhanced_features])
```

Model Evaluation

Metrics used: Accuracy, Precision, Recall, F1-score.

Baseline vs.Enhanced model metrics:

Metric	Baseline (Scaled)	Enhanced Features
Accuracy	0.8833	0.9250
Precision	0.85	0.8942
Recall	0.75	0.7939
F1-score	0.80	0.8411

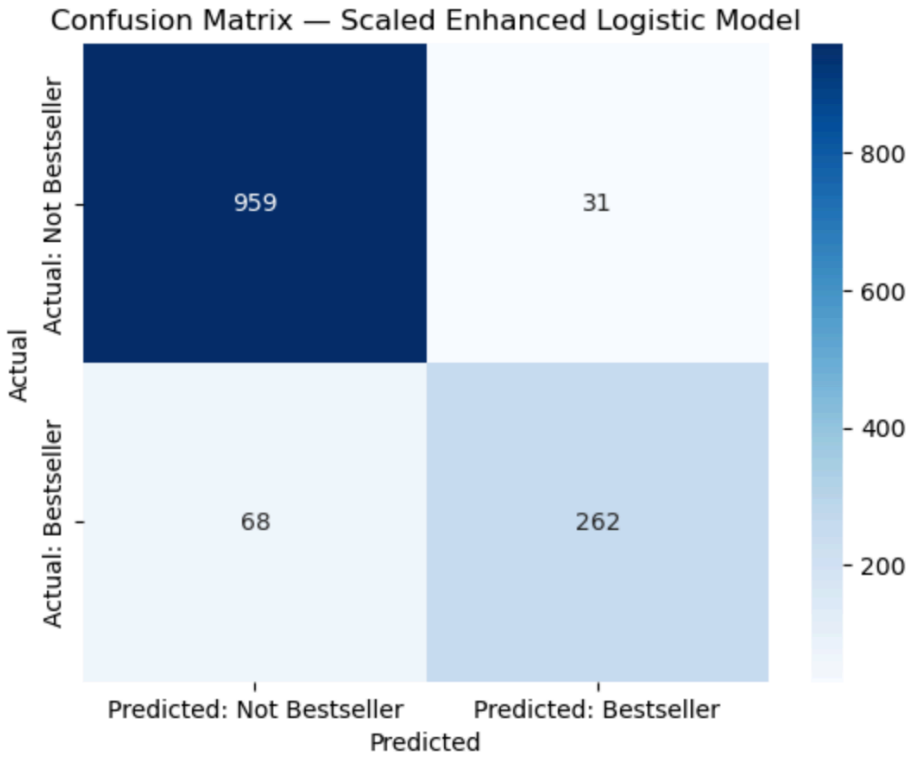
Visuals:

The enhanced model with interaction features and author popularity shows a clear improvement over the baseline in all metrics, particularly in accuracy and precision

```
Classification Report:
                precision    recall  f1-score   support

Not Bestseller       0.93      0.97      0.95       990
    Bestseller       0.89      0.79      0.84       330

      accuracy                           0.93      1320
     macro avg       0.91      0.88      0.90      1320
  weighted avg       0.92      0.93      0.92      1320
```



Conclusion

Engineered features improved model performance, but only slightly.

Scaled numeric features were crucial for gradient descent stability and lead to the best model performance overall.

Limitations:

- Some features may be noisy.
- Threshold for `bestseller` is arbitrary.

- Recommendations:
  - Could include NLP on book descriptions for deeper insights.
  - Test other models like Random Forest or XGBoost for potential further improvements.

References

Abdulahadarain (2023) *Exploratory analysis of the goodreads dataset* 📊, *Kaggle*. Available at: https://www.kaggle.com/code/abdulahadarain/exploratory-analysis-of-the-goodreads-dataset (Accessed: 28 October 2025).

Stojiljković, M. (2023) *Logistic regression in python*, *Real Python*. Available at: https://realpython.com/logistic-regression-python/ (Accessed: 27 October 2025).

Yogita (2024) *Guide for building an end-to-end logistic regression model*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/09/guide-for-building-an-end-to-end-logistic-regression-model/ (Accessed: 27 October 2025).