IN THE NAME OF GOD

Mohaddeseh Feizi – 40007933

Miniproject1

1.1.

The dataset is a **customer churn dataset** for a credit card service.

Objective: Predict whether a customer will churn (stop using the credit card service). This helps the bank proactively retain customers by offering personalized services.

Only 16.07% of customers churned, indicating a significant class **imbalance.**

Features: numerical and categorical data, Target: churn

Challenges in Analyzing:

- **Class Imbalance:** Models tend to favor the majority class (non-churners), making it difficult to accurately predict churners.

- **High Cardinality or Redundancy:** Features like credit card category might have many unique values or may not contribute much to the prediction if poorly encoded or irrelevant.

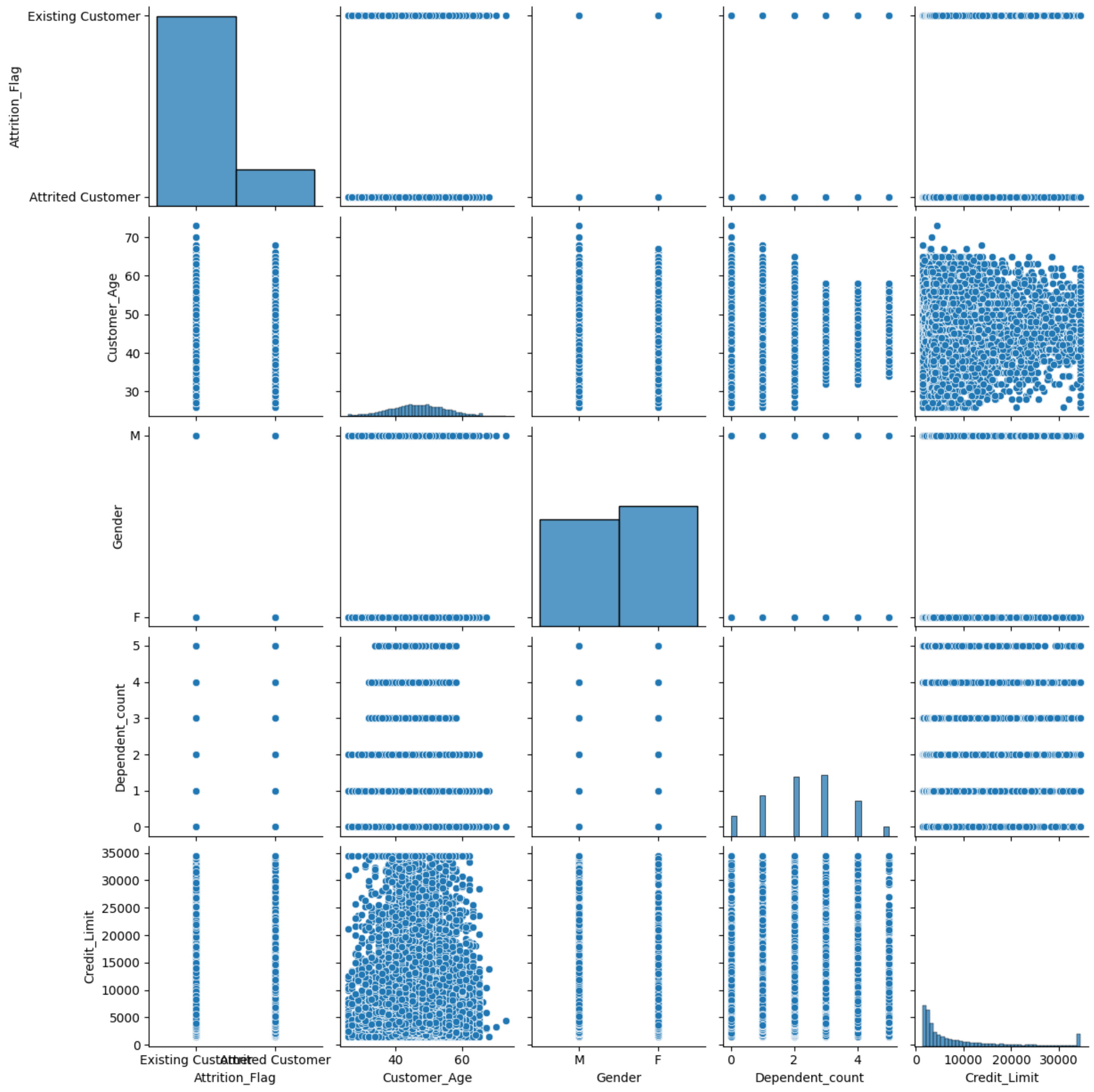
- **Feature Relevance:** Not all features may have predictive power for churn. Identifying which features matter is key.

Features: Attrition Flag, Customer Age, Gender, Dependent count, Education Level, Marital Status, Income Category, Card Category, Months on book, Total Relationship Count, Months Inactive 12 mon, Contacts Count 12 mon, Credit Limit, Total Revolving Bal, Average Open To Buy, Total Amount Change Q4 Q1, Total Trans Amount, Total Trans Ct, Total Ct Change Q4 Q1, Average Utilization Ratio, Naïve Bayes Classifier Attrition Flag Card Category Contacts Count 12 mon Dependent count Education Level Months Inactive 12 mon 1, Naïve Bayes Classifier Attrition Flag Card Category Contacts Count 12 mon Dependent count Education Level Months Inactive 12 mon 2

This data has 23 columns and 10127 rows → 232898

1.2.

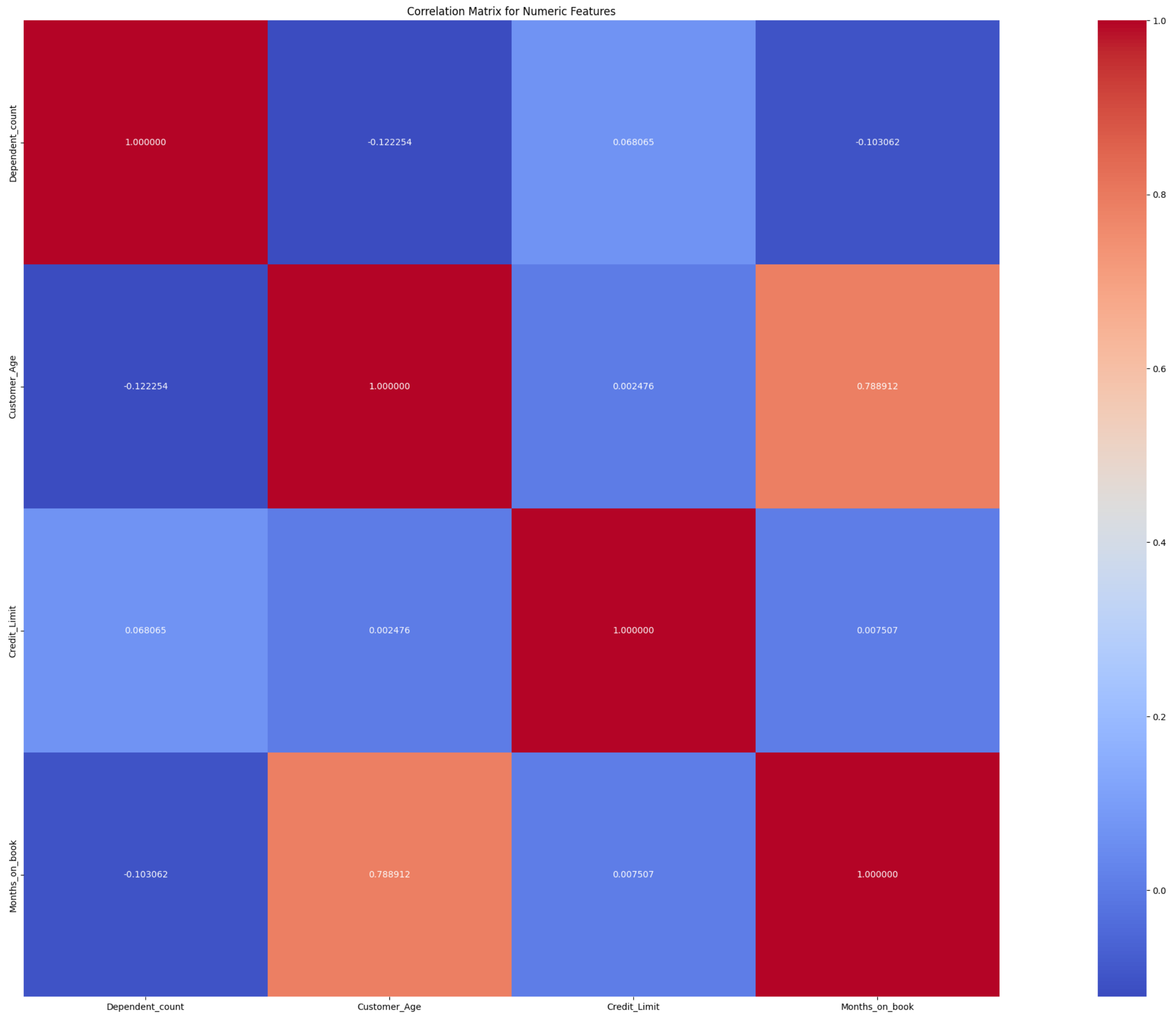
Selected columns: Attrition Flag, Customer Age, Gender, Dependent count, Credit Limit

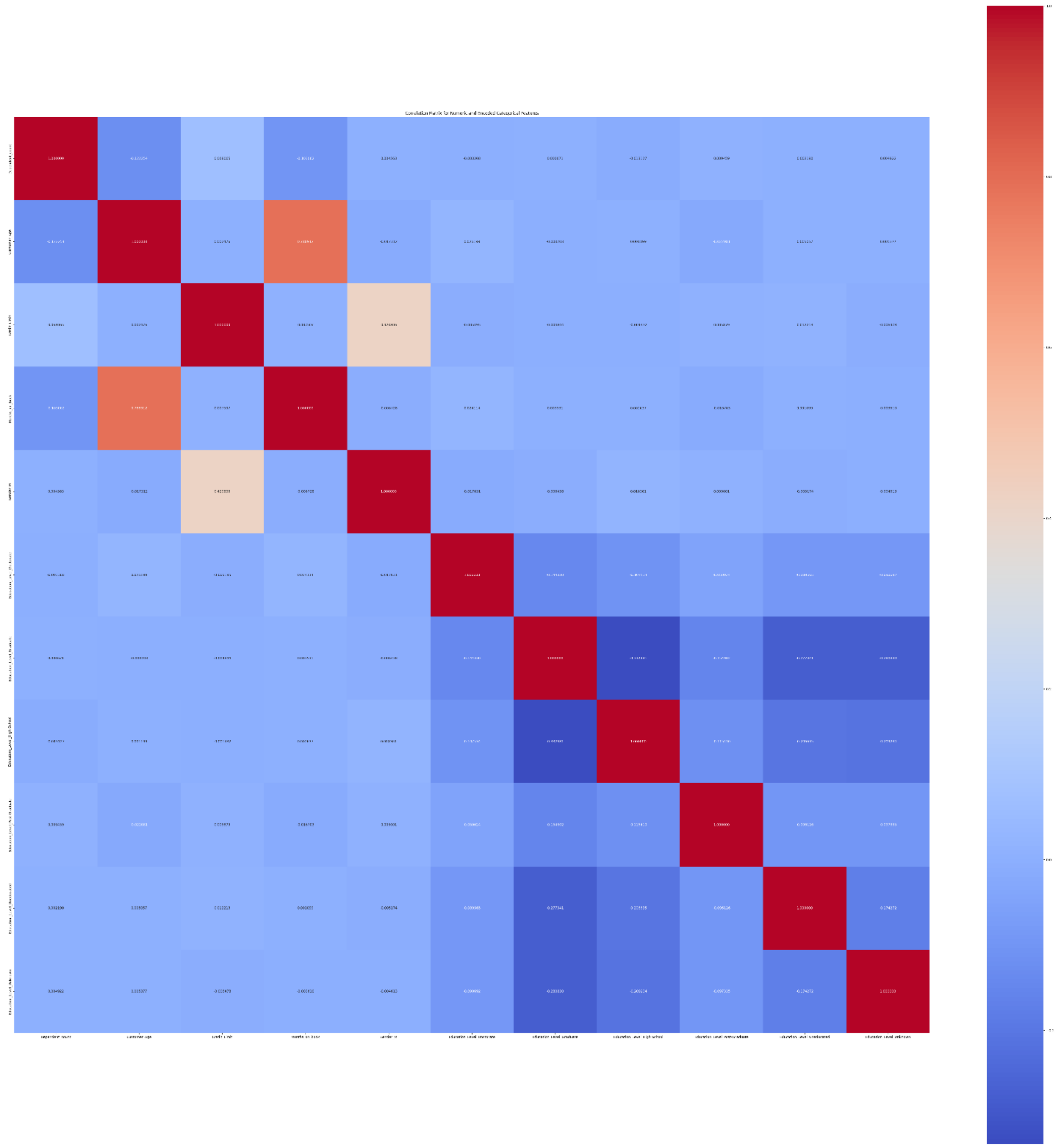


1.3.

Selected numerical columns: Dependent Count, Customer Age, Credit Limit, Months on book

Selected categorical columns: Gender, Education Level

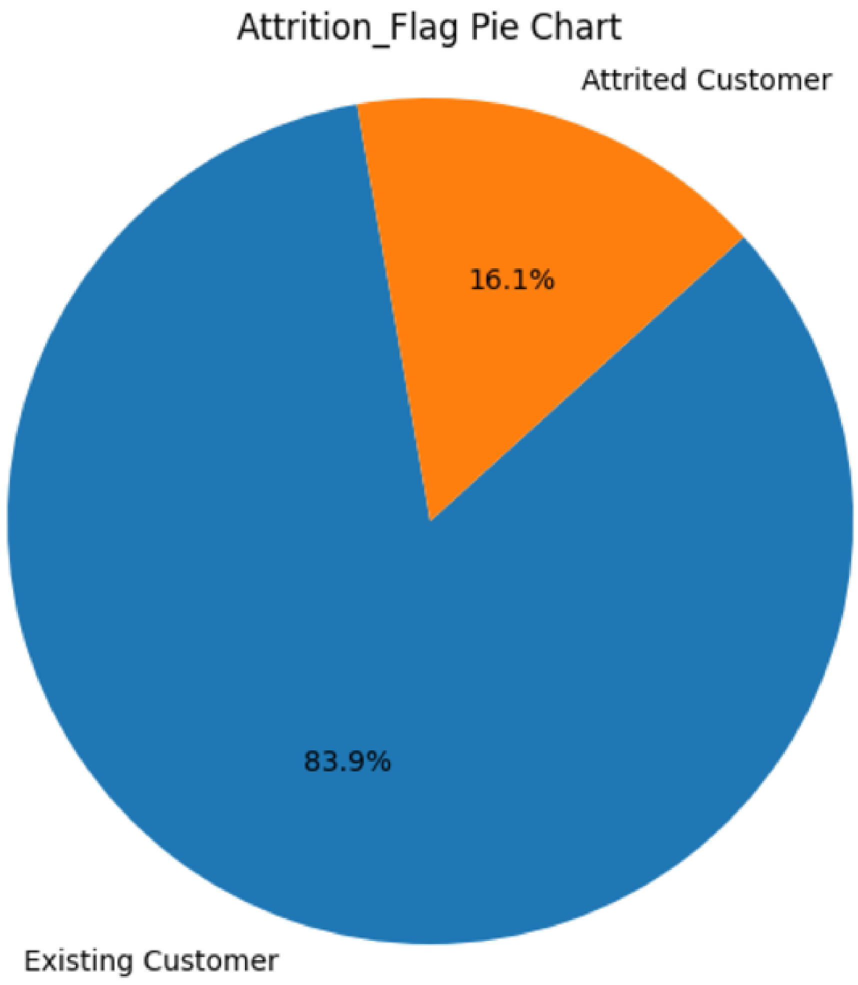




1.4. There is no Nan data.

1.5.

2 class of Attrition Flag: Existing Customer, Attrited Customer



Dataset imbalance can significantly affect the performance of the final model:

1. **Bias toward the Majority Class:** The model tends to favor the majority class because it sees more of it during training. This can result in high accuracy but poor performance on the minority class, leading to unhelpful predictions for underrepresented cases.

2. **Poor Generalization:** The model may fail to learn meaningful patterns for the minority class, leading to overfitting to the majority class and under-fitting to the minority.

3. **Misleading Metrics:** Accuracy can appear high even when the model performs poorly on the minority class. For example, in a dataset with 90% majority class, predicting only the majority class gives 90% accuracy, even though the model fails to classify the minority correctly.

4. Precision and recall for the minority class often drop significantly in imbalanced datasets. F1-score, which balances precision and recall, also degrades for the minority class.

5. The matrix often shows poor true positives for the minority class compared to the majority.

To fix the negative effects of dataset imbalance and improve the performance of the final model, you can apply a combination of **data-level**, **algorithm-level**, and **evaluation-focused** strategies:

1. **Data-Level Fixes:**
   ✓ **Oversampling the Minority Class:**
   - **Duplicate existing data:** Simply duplicate samples from the minority class.
   - **Generate synthetic data:**
   - Randomly remove samples from the majority class to balance the dataset:
     - ➢ **SMOTE (Synthetic Minority Oversampling Technique):** Creates synthetic data points by interpolating between minority class samples.
     - ➢ **ADASYN (Adaptive Synthetic Sampling):** Focuses on generating synthetic samples for harder-to-classify regions.

   ✓ **Undersampling the Majority Class:**
   Randomly **remove** samples from the majority class to balance the dataset. (Avoid excessive undersampling, as it may lead to loss of important information.)

2. **Algorithm-Level Fixes:**
   ✓ **Assign Class Weights:**

- Most machine learning frameworks allow specifying **class weights**. Errors in the minority class are penalized more heavily than errors in the majority class.

✓ **Use Custom Loss Functions:**
Design loss functions that give more importance to the minority class.
- **Focal Loss:** Reduces the impact of well-classified examples, focusing on harder cases.
- **Weighted Cross-Entropy Loss:** Adjusts for class imbalance by adding weights based on class distribution.

✓ **Use Robust Algorithms:**
Certain algorithms are inherently better at handling imbalanced data, such as:
- **XGBoost** and **LightGBM:** Provide built-in handling of imbalance.
- **Balanced Random Forests** and **EasyEnsemble:** Designed specifically for imbalanced problems.

3. **Evaluation-Focused Adjustments:**
Change how you evaluate the model to account for imbalance and better understand its performance.

✓ **Use Appropriate Metrics:**
Avoid relying on accuracy alone. Instead, focus on:
- **F1-Score:** Balances precision and recall, especially useful for minority classes.
- **Confusion Matrix Analysis:** Helps visualize true positives, false positives, and other class-specific metrics.

When handling data imbalance, you should balance the data **only in the training set, not the test set.** The test set should reflect the original, real-world distribution of the data. This ensures that the model's evaluation is realistic and unbiased.

Balancing the training set helps the model learn equally from all classes. However, evaluating it on the original class distribution of the test set ensures it can generalize to real-world, imbalanced scenarios.

If the test set is balanced, metrics like accuracy, precision, recall, and F1-score might not accurately represent how the model performs in real-world applications.

So: Split first, balance second.

1.6.

Validation Set Comparison:

Confusion Matrix:

Metric	Unbalanced	Balanced
Class 0 (No Attrition)	Predicted all as Class 1 (0/227 correct)	Better coverage (154/227 correct)
Class 1 (Attrition)	Perfect recall (1109/1109 correct)	Slight drop (783/1109 correct)

Key metrics:

Metric	Unbalanced	Balanced
Accuracy	83%	70%
Class 0 Precision	0%	32%
Class 0 Recall	0%	68%
Class 1 Precision	83%	91%
Class 1 Recall	100%	71%
Macro Avg. F1-Score	45%	62%

Test Set Comparison:

Confusion Matrix:

Metric	Unbalanced	Balanced
Class 0 (No Attrition)	Predicted all as Class 1 (0/429 correct)	Better coverage (269/429 correct)
Class 1 (Attrition)	Perfect recall (2286/2286 correct)	Slight drop (1836/2286 correct)

Key metrics:

Metric	Unbalanced	Balanced
Accuracy	84%	67%
Class 0 Precision	0%	27%
Class 0 Recall	0%	63%
Class 1 Precision	84%	91%
Class 1 Recall	100%	68%
Macro Avg. F1-Score	46%	58%

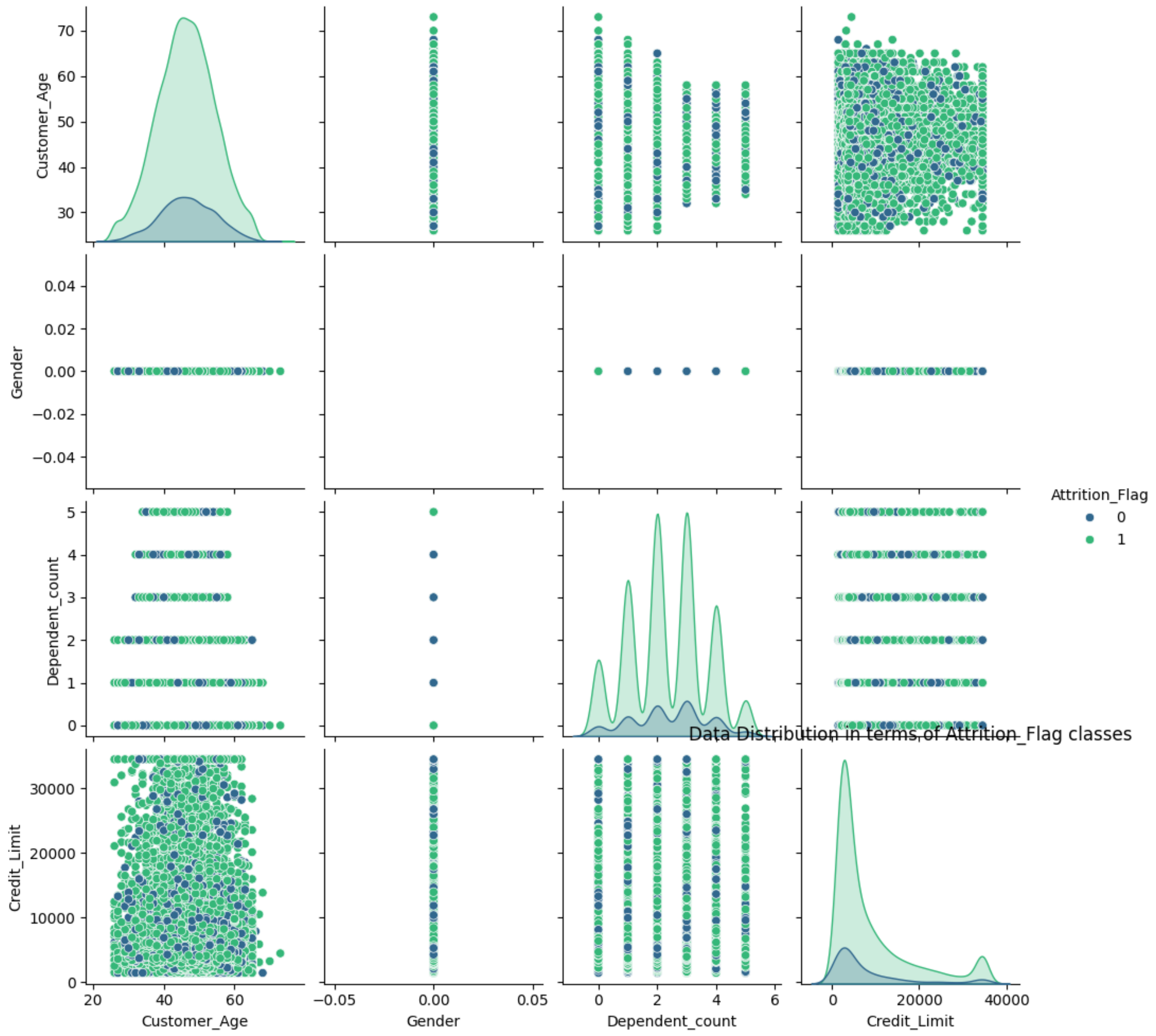
Key Observations:

1. **Class 0 (No Attrition)**:
   - **Unbalanced**: The model nearly ignores Class 0, with almost no correct predictions.
   - **Balanced**: Significant improvement in recall and precision, with more true positives and reduced bias toward Class 1.
2. **Class 1 (Attrition)**:
   - **Unbalanced**: Excellent performance due to the model focusing entirely on this majority class.
   - **Balanced**: Slight drop in recall but gains better generalization by improving Class 0 predictions.
3. **Macro Averages:**
   - **Unbalanced**: Low F1-score for Class 0 skews the macro average, indicating poor overall balance.
   - **Balanced**: Improved macro averages indicate the model performs better across both classes.
4. **Accuracy:**
   - Accuracy is misleading for the unbalanced model since it heavily favors the majority class.

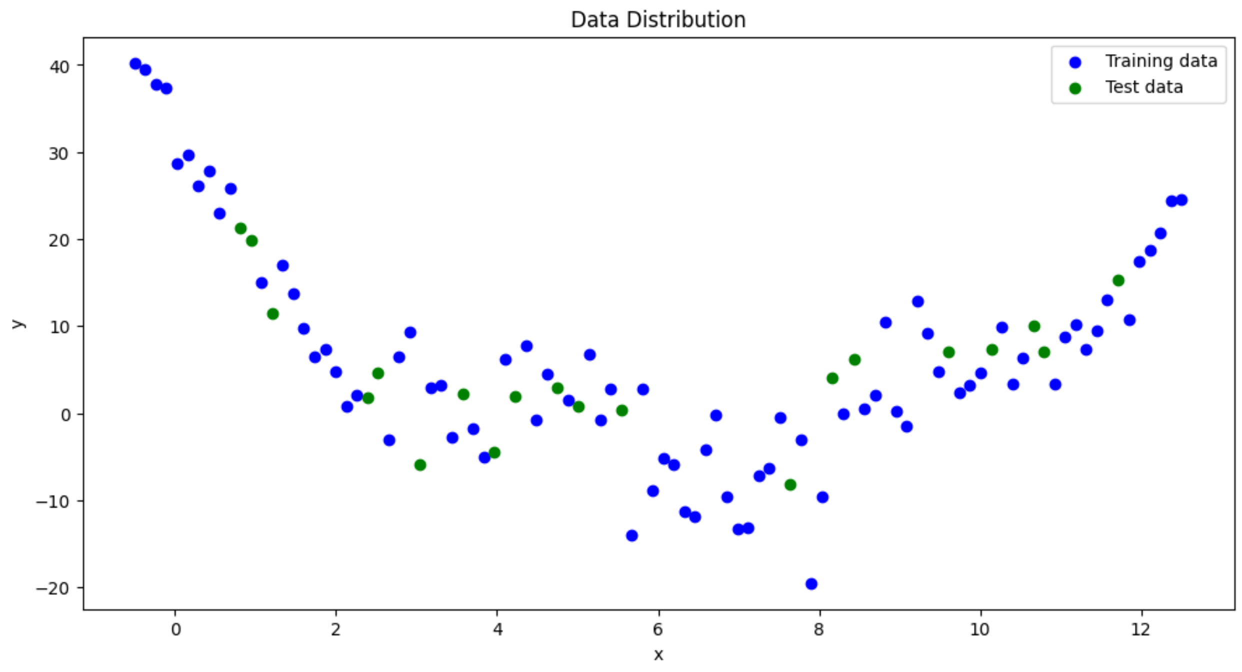
Conclusion:

1. **Unbalanced Model**:
   - Performs well only for the majority class (Attrition) but completely fails on the minority class (No Attrition).
   - High accuracy hides poor performance for Class 0.
2. **Balanced Model**:
   - Strikes a better balance between the classes, significantly improving recall and precision for Class 0 (No Attrition).
   - Slight trade-off in Class 1 performance, but this is acceptable for better overall balance.

*Data Distribution in terms of Attrition Flag classes:



2.1.



2.2.

1. **Mean Absolute Error (MAE)**:
   This metric measures the average absolute errors between predicted values and actual values. MAE is straightforward to understand and represents the average deviation of predictions from actual outcomes. The formula is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

   Where y_i are the actual values and $\hat{y}_i$are the predicted values.

2. **Mean Squared Error (MSE)**:
   This metric calculates the average of the squared differences between predicted and actual values, placing a heavier penalty on larger errors. It helps identify models with significant prediction errors. The formula is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. **R-squared (Coefficient of Determination)**:
   This metric indicates the proportion of variance in the dependent variable that can be explained by the independent variables. R-squared values range from 0 to 1, with values closer to 1 indicating a better fit of the model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

   Where $\bar{y}_i$ is the mean of the actual values.

2.3.

Gradient Descent for Linear Regression:

1. **Objective Function**:
   The goal of linear regression is to minimize the cost function, typically the Mean Squared Error (MSE):

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{n}\left(h_\beta(x^{(i)}) - y^{(i)}\right)^2$$

Where:

- $J(\beta)$ is the cost function.
- m is the number of training examples.
- $h_\beta(x^{(i)})$ is the hypothesis function (model prediction) for input $x^{(i)}$:

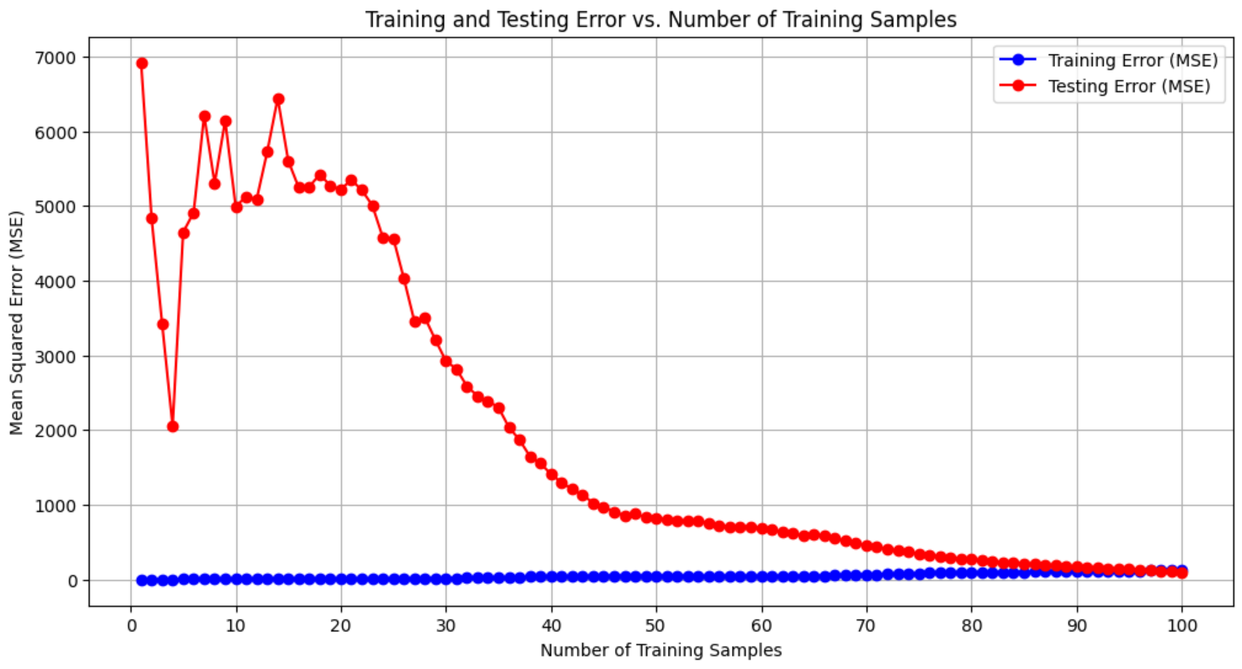
$$h_\beta x = \beta_0 + \beta_1 x$$

$$g = -\frac{\partial}{\partial\beta}\left(\left(y_i - y_{pred_i}\right)^2\right) = -2\left(y_i - y_{pred_i}\right).\frac{\partial y_{pred_i}}{\partial\beta}$$

$$y_{pred_i} = x_i.\beta \rightarrow x_i = \frac{\partial y_{pred_i}}{\partial\beta} \rightarrow gradient = -2\left(y_i - y_{pred_i}\right).x_i$$

The relationship between x and y is not linear: the model may not estimate the data accurately. More advanced models, such as polynomial regression or other nonlinear models, may be needed.

2.4.



As the size of the training dataset increases, the model has more data to learn from, which allows it to fit the training data better. As a result, the training error decreases initially. Eventually, the training error stabilizes at a certain level. This happens because the model's capacity (its ability to fit data) is fixed, and adding more data won't make it "fit better" than it inherently can.

When the training dataset is small, the model might over-fit the training data, leading to a high testing error. As the training dataset grows, the model generalizes better, and the testing error decreases. With a sufficiently large training dataset, the testing error also stabilizes at a level that reflects the model's capacity and how well it can generalize to unseen data.

2.5.

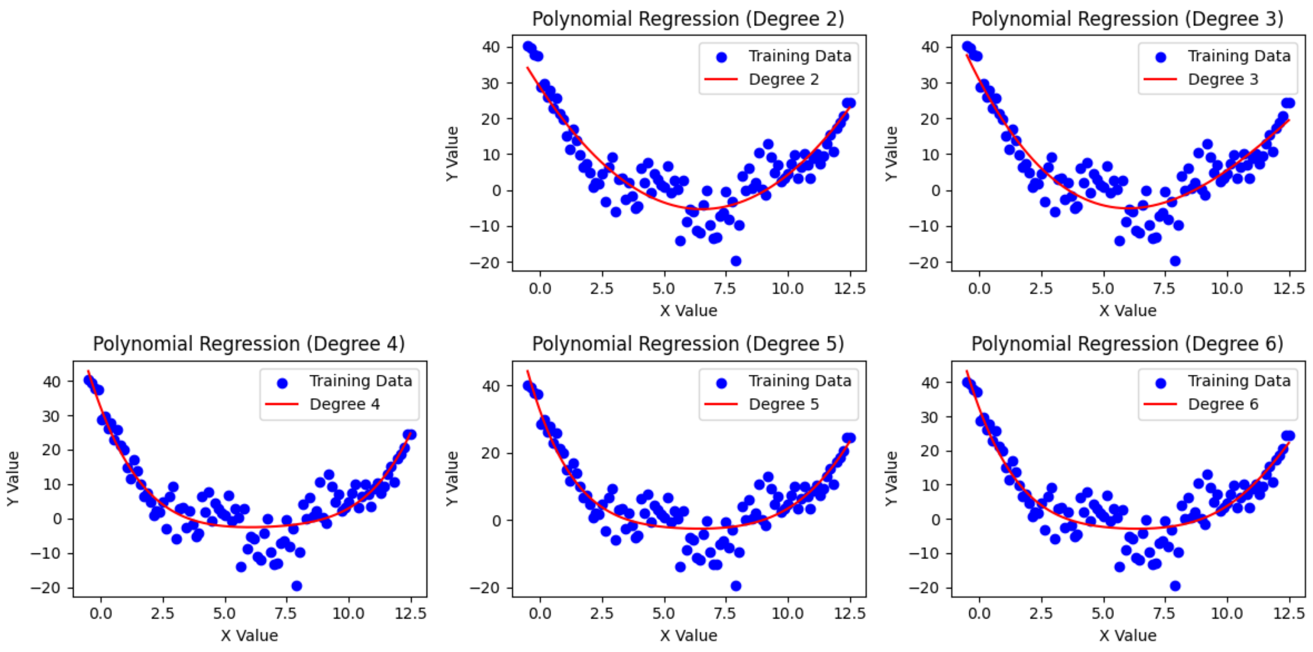
Yes, But Only Under Certain Conditions**:**

The model must have enough capacity to capture the complexity of the task. If the ML model is too simple (under-fitting), it can’t capture the underlying patterns in the data, no matter how much training data is provided. In this case, increasing the model's complexity is necessary.

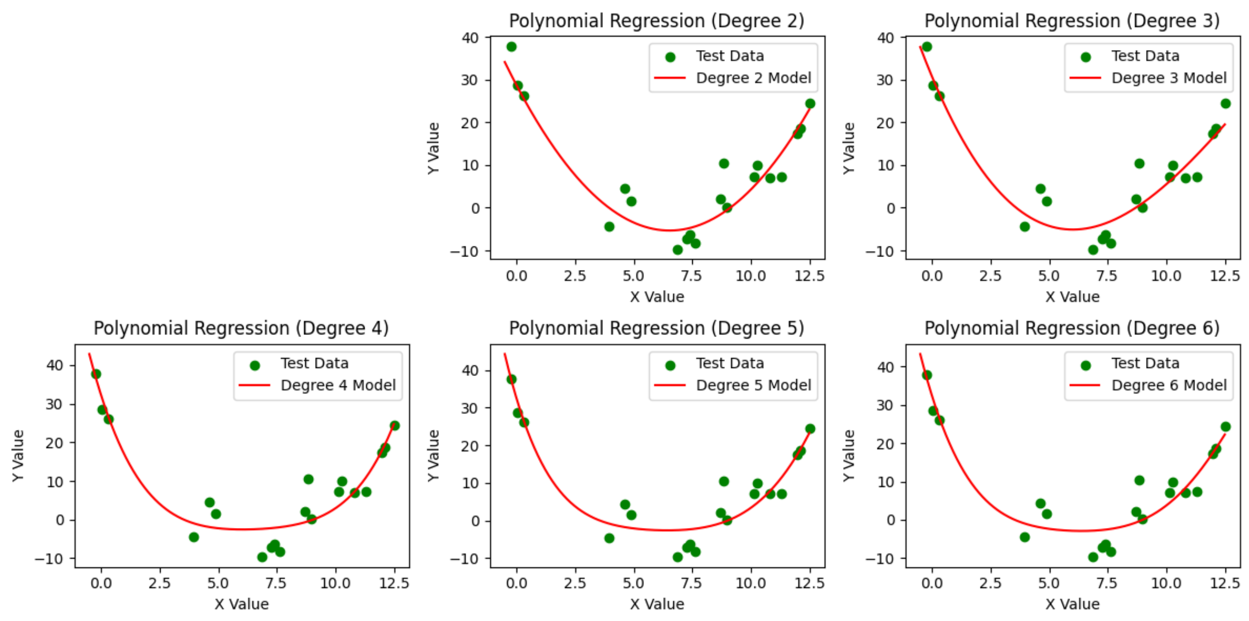
The data must be abundant, diverse, and of high quality. Merely adding more data won't help if the data is noisy, biased, or not representative of the real-world task. High-quality and diverse data is crucial for improving the model's performance.

The task itself must be solvable to a human-level standard by an algorithm.

2.6.



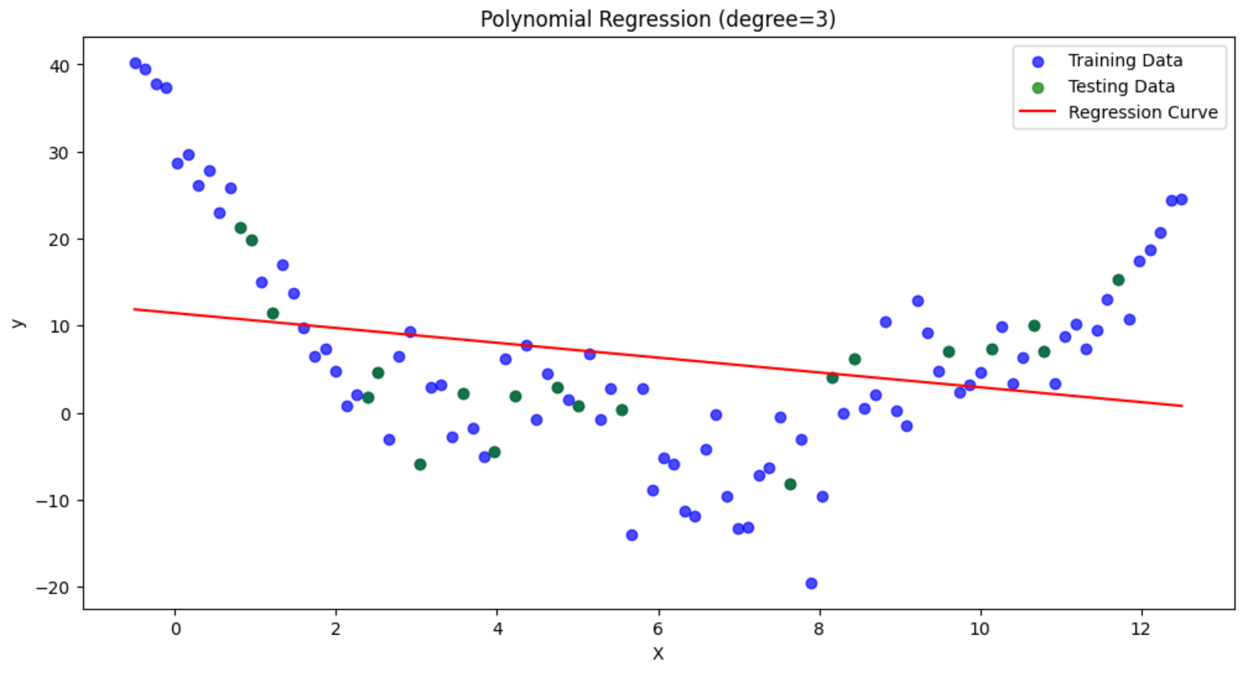
Training Data Errors (MSE): 28.43
MAE = 4.25
R-squared = 0.8



Testing Data Errors (MSE): 0.89
MAE = 3.47
R-squared = 0.89

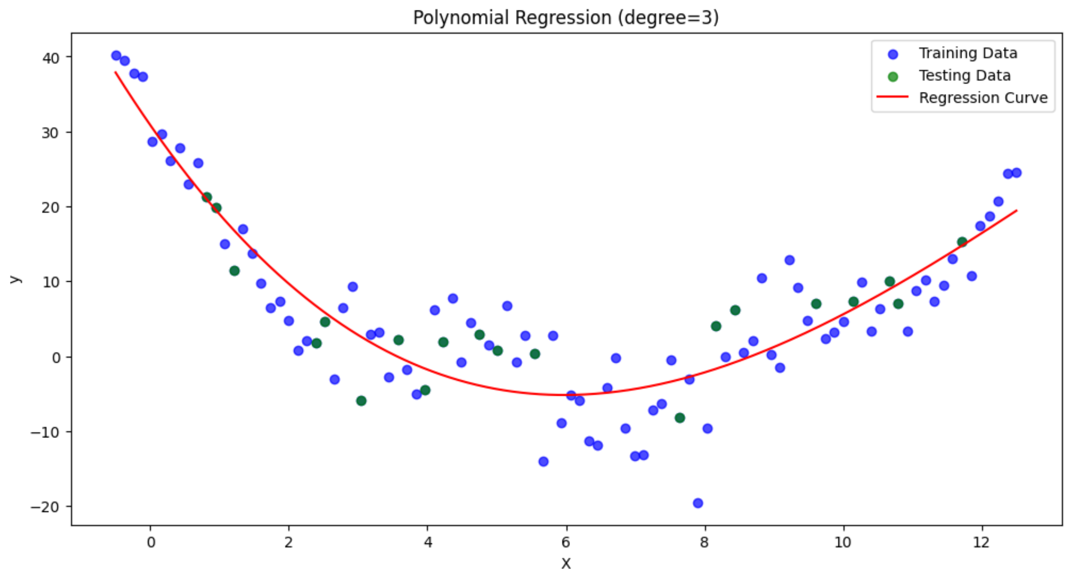
2.7.

Linear Regression: fits a straight line to the data by minimizing the Mean Squared Error (MSE) between the predicted and actual values. It assumes a linear relationship between the independent variables X and the target y.



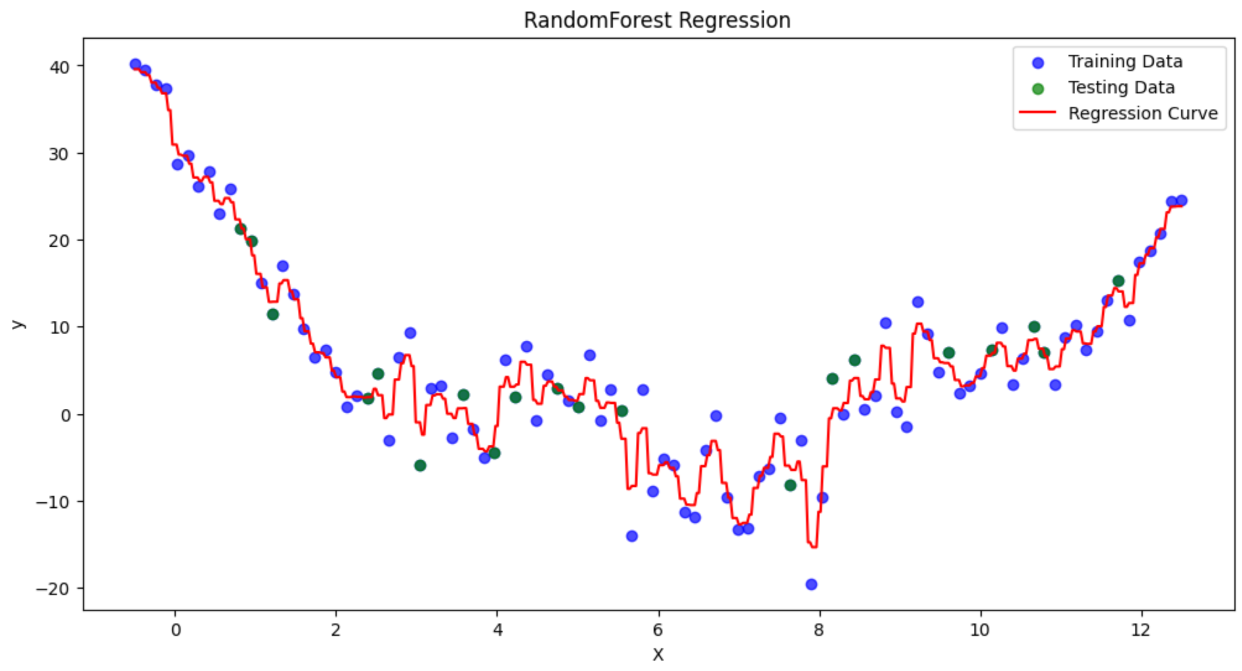
Training Data Errors (MSE): 134.35
Testing Data Errors (MSE): 65.60

Polynomial Regression: Extends linear regression by creating polynomial features, allow the model to fit non-linear relationships.



Training Data Errors (MSE): 26.24
Testing Data Errors (MSE): 19.58

Random Forest Regression: Combines predictions from multiple decision trees (ensemble learning) to improve accuracy and reduce overfitting. Each tree is trained on a random subset of data and features.



Training Data Errors (MSE): 3.64
Testing Data Errors (MSE): 3.62

The error of these regression model decreases respectively.