

Data Processing & Feature Engineering II

Prepared by Dr. Sumaya Alkubaisi

FNT633
Machine Learning in Finance

Lecture 3



Learning Goals

By the end of this lecture, you will be able to:

- By the end of this lecture, you will be able to:
- Integrate financial data from multiple sources.
- Normalize and scale variables for fair comparisons.
- Detect and handle outliers using residuals.
- Apply feature engineering, transformations, and encoding.
- Prepare finance-ready datasets in Python.

What is Data Integration?

Data Integration is the process of combining data from multiple sources into a single, consistent view.

Why it matters in finance?

- Customer financial profiles may be spread across different banks, credit bureaus, and transaction logs.
- Integrated data = complete view of customer behavior → better predictions.

Examples of Data Integration in Finance

- Merging datasets: Combine loan applications with credit scores.
- Concatenating records: Stack transaction logs from different branches.
- Joining tables: Link customer IDs with loan payment history.

Example:

Table A: Loan applications (name, income, loan amount).

Table B: Credit bureau (name, credit score).

Integration: Merge both on “name” to get a complete profile.

Python Demo: Data Integration (Merge)

```
import pandas as pd
```

```
# Example loan application dataset
```

```
loans = pd.DataFrame({  
    "Customer": ["Ali", "Sara", "John"],  
    "Income": [5000, 7000, 4000],  
    "Loan_Amount": [2000, 3000, 1500]  
})
```

```
# Example credit bureau dataset
```

```
credit = pd.DataFrame({  
    "Customer": ["Ali", "Sara", "John"],  
    "Credit_Score": [650, 720, 580]  
})
```

```
# Merge datasets
```

```
merged = pd.merge(loans, credit, on="Customer")  
print(merged)
```

Python Demo: Data Integration (Merge)

Output:

Combined dataset with income, loan amount, and credit score.



	Customer	Income	Loan_Amount	Credit_Score
0	Ali	5000	2000	650
1	Sara	7000	3000	720
2	John	4000	1500	580

merge() : combines two datasets on a shared column (**Customer**).

Python Demo: Data Integration (Concatentation)

```
# Concatenate two loan datasets
loans_branchA = pd.DataFrame({
    "Customer": ["Ali", "Mona"],
    "Loan_Amount": [2000, 2500]
})
loans_branchB = pd.DataFrame({
    "Customer": ["John", "Laila"],
    "Loan_Amount": [1500, 3000]
})

all_loans = pd.concat([loans_branchA, loans_branchB])
print(all_loans)
```

Python Demo: Data Integration (Concatentation)

Output:

Combined records from both branches into one dataset.



	Customer	Loan_Amount
0	Ali	2000
1	Mona	2500
0	John	1500
1	Laila	3000

concat() : stacks datasets on top of each other (like combining rows).

Data Normalization

What is Data Normalization?

- Rescaling numeric data into a standard range (e.g., 0–1).
- Ensures that no single feature dominates simply because of its scale.
- Makes features comparable, regardless of their original scale.

Example: Income (70,000) vs Age (35).

Why Do We Need It in Finance?

- Financial features are often on different scales (income, loan amount, credit score, age).
- Without normalization → model may give more weight to features with bigger numbers.
- Risk: Predictions become biased (unfair and unreliable).

What Happens if a Feature Dominates?

- Algorithm focuses too much on one feature.
- Other important variables are ignored.
- This causes:
 - Biased predictions → the model favors one feature unfairly (e.g., always trusting income more than credit score).
 - Skewed results → the predictions are “tilted” in one direction, not balanced.

Example:

If income is very large compared to other features, the model may “think” income alone decides loan approval → unfair to people with lower income but excellent credit scores.

Benefits of Normalization

- Ensures fair comparison among features.
- Improves model accuracy and training speed.
- Prevents bias from large-scale features (like loan amount overshadowing credit score).
- Helps financial predictions become balanced and reliable.

Common Normalization/ Standardization Techniques

Min-Max Scaling → Rescales values to 0–1.

Formula: $(x - \min) / (\max - \min)$.

Example: Income 70,000 in range (10,000–100,000) → 0.67.

Z-score Standardization → Rescales values based on mean & standard deviation.

Formula: $(x - \text{mean}) / \text{std}$.

Example: Credit score 700 with mean 600, std 50 → 2.0 (2 std above mean).

Python Demo: Min-Max Normalization

```
from sklearn.preprocessing import MinMaxScaler
```

```
data = pd.DataFrame({
    "Income": [3000, 6000, 9000],
    "Loan_Amount": [1000, 2000, 3000]
})
```

```
scaler = MinMaxScaler()
normalized = scaler.fit_transform(data)
```

```
normalized_df = pd.DataFrame(normalized, columns=data.columns)
print(normalized_df)
```



	Income	Loan_Amount
0	0.0	0.0
1	0.5	0.5
2	1.0	1.0

Python Demo: Z-score Standardization

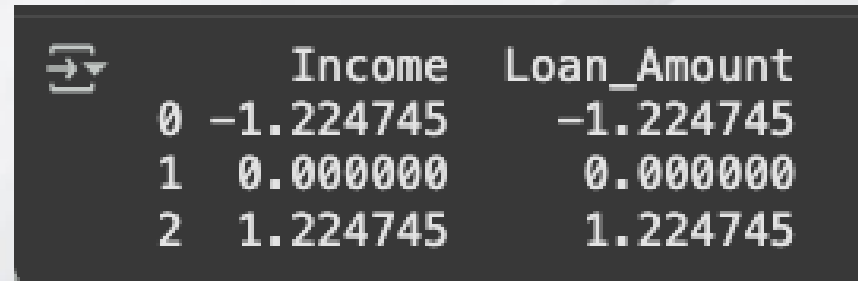
```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
standardized = scaler.fit_transform(data)
```

```
standardized_df = pd.DataFrame(standardized, columns=data.columns)
```

```
print(standardized_df)
```



	Income	Loan_Amount
0	-1.224745	-1.224745
1	0.000000	0.000000
2	1.224745	1.224745

❖ Useful when features have very different ranges (e.g., salary vs. age).

Handling Missing Values (Residual Approach)

- Missing values can distort analysis or model performance.
- Residual approach = an advanced method to estimate missing values using other related features.
- The residual approach doesn't just guess missing values. It predicts them using patterns in the data and then fine-tunes the guess with residuals.
- It is more accurate than simple mean/median imputation.

How the Residual Approach Works?

1. Build a model using complete data

Take the rows that don't have missing values.

Example: We want to fill missing income, so we use customers who have income recorded.

2. Predict missing values

Use other features (like age, education, credit score) to predict income for customers with missing income.

3. Use residuals to adjust the prediction (optional)

Residual = the small difference between the real value and the predicted value (for those customers where we know income).

If we see a pattern in the residuals (like always predicting slightly higher), we can adjust the predicted values to make them more accurate.

Residual Approach Example

Dataset: Loan applications

Problem: Some customers are missing income data.

Step 1: Train a regression model with Age, Education, Credit Score → Income (using customers with full data).

Step 2: For a customer missing income, predict:

Age = 40, Education = Master's, Credit Score = 720 → Predicted Income = \$75,000.

Step 3: Check residuals from training: If the model usually overestimates by \$5,000, adjust to \$70,000.

Why It's Useful?

- Uses relationships between variables to produce more accurate estimates.
- Maintains dataset size, unlike deletion.
- Reduces bias introduced by simple imputation.
- Prepares data better for machine learning models.

Variable Transformation

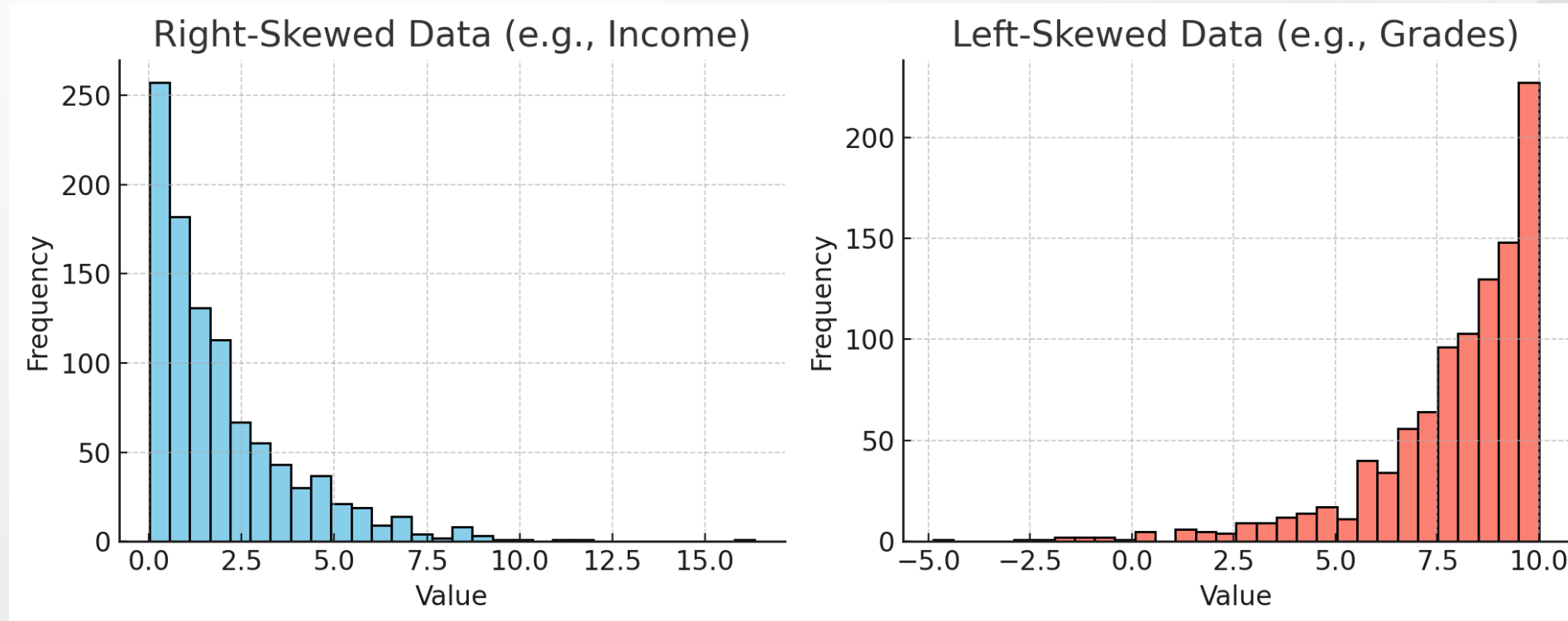
- Adjusts the distribution or relationship of variables to make data more suitable for modeling.
- It is commonly used to normalize skewed data, reduce the influence of extreme values, or reveal hidden patterns.

Examples:

Log transformation: Makes right-skewed data (e.g., income) more normal.

Square root transformation: Reduces the impact of large values.

Polynomial features: Introduces nonlinear relationships for complex patterns.



Visual comparison — the right-skewed data has a long tail to the right (few very large values), while the left-skewed data has a long tail to the left (few very small values)

Feature Encoding

Converts categorical (non-numeric) variables into numeric format so that machine learning algorithms can process them.

Types of Features

1. Numerical (Quantitative):

Values that represent measurable quantities.

Examples: Age, Income, Loan Amount.

Can be continuous (e.g., income = 50,000.75) or discrete (e.g., number of transactions = 7).

2. Categorical (Qualitative):

Values that represent groups or categories.

Nominal: No natural order.

Example: Gender (Male, Female), Country (USA, UK, UAE).

Ordinal: Has a meaningful order or ranking.

Example: Education (High School < Bachelor < Master < PhD), Credit Rating (Low < Medium < High).

Why Encoding Is Needed

Machine learning models only understand numbers, not text labels. So, we convert categorical data into numeric form using encoding.

Nominal → One-Hot Encoding

(each category becomes its own binary column)

Bachelor → (1, 0, 0)

Master → (0, 1, 0)

PhD → (0, 0, 1)

Ordinal → Label Encoding

(replace categories with ordered numbers like 1, 2, 3)

Loan Default: *Yes/No* → *1/0*

Feature Scaling

Adjusts numerical feature values to a common scale, ensuring that no feature dominates others in models sensitive to magnitude (e.g., KNN, SVM).

Scaling Methods:

- Standardization (Z-score): Centers data around mean = 0 and standard deviation = 1.
- Min-Max Scaling: Rescales data to a fixed range, usually [0, 1].

Example:

Without scaling, Age (20–60) and Loan Amount (10,000–200,000) would have very different ranges, causing imbalance in model learning.

Python Demo: Feature Engineering

---- Variable Transformation ----

```
df['log_salary'] = np.log(df['salary'] + 1)
```

---- Feature Encoding ----

```
df = pd.get_dummies(df, columns=['education'], drop_first=True)
```

---- Feature Scaling ----

```
scaler = StandardScaler()
```

```
df['loan_scaled'] = scaler.fit_transform(df[['loan_amount']])
```

Lab Activity: Data preparation 2

```
# =====  
# Lecture 3, Feature Engineering & Data Processing  
# =====  
# Libraries  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt %matplotlib inline from sklearn.preprocessing  
import StandardScaler, MinMaxScaler  
from scipy import stats
```

Lab Activity: Data preparation 2

```
# =====
# Step 1: Create Example Dataset
# =====
data = { "age": [25, 40, None, 35, -5],
        "salary": [50000, 60000, 55000, None, 45000],
        "loan_amount": [20000, 25000, 22000, 18000, 30000],
        "education": ["Bachelor", "Master", "PhD", "Bachelor", "Master"],
        "default": ["No", "Yes", "No", "No", "Yes"] }
df = pd.DataFrame(data)
df['customer_id'] = [1, 2, 3, 4, 5]
print("Original Dataset:")
print(df) Dataset:") print(merged)
```

Lab Activity: Data preparation 2

```
# =====  
# Step 2: Data Integration  
# =====  
risk_data = { "customer_id": [1, 2, 3, 4, 5],  
              "risk_score": [0.2, 0.8, 0.5, 0.1, 0.7] }  
  
risk_df = pd.DataFrame(risk_data)  
  
# Merge both datasets merged = pd.merge(df, risk_df, on="customer_id")  
print("\nMerged Dataset:")  
print(merged)
```

Lab Activity: Data preparation 2

```
# =====  
# Step 3: Handling Missing Values  
# =====  
print("\nMissing values before cleaning:\n",  
merged.isnull().sum())  
  
# Fill missing values  
merged['age'].fillna(merged['age'].median(), inplace=True)  
merged['salary'].fillna(merged['salary'].mean(), inplace=True)  
print("\nMissing values after cleaning:\n",  
merged.isnull().sum()) #
```

Lab Activity: Data preparation 2

```
# =====  
# Step 4: Handling Outliers  
# =====  
plt.figure(figsize=(6,3))  
sns.boxplot(x=merged['age'])  
plt.title("Age Distribution Before Cleaning")  
plt.show()  
  
# Remove unrealistic ages (negative)  
merged = merged[merged['age'] > 0]  
plt.figure(figsize=(6,3))  
sns.boxplot(x=merged['age'])  
plt.title("Age Distribution After Cleaning")  
plt.show()
```

Lab Activity: Data preparation 2

```
# =====  
# Step 5: Variable Transformation (Log Transformation)  
# =====  
# Histogram before log transformation  
sns.histplot(df['salary'], kde=True)  
plt.title("Salary Distribution (Before Log)")  
plt.show()  
  
# Apply log transformation  
df['log_salary'] = np.log(df['salary'] + 1)  
  
# Histogram after log transformation  
sns.histplot(df['log_salary'], kde=True, color="orange")  
plt.title("Salary Distribution (After Log)")  
plt.show()
```

Lab Activity: Data preparation 2

```
# =====
```

```
# Step 6: Feature Encoding
```

```
# =====
```

```
# One-hot encoding “education”
```

```
df = pd.get_dummies(df, columns=['education'], drop_first=True)  
print(df.head())
```

Lab Activity: Data preparation 2

```
# =====  
# Step 7: Feature Scaling  
# =====  
scaler = StandardScaler()  
df['loan_scaled'] = scaler.fit_transform(df[['loan_amount']])  
  
# Plot comparison of original vs scaled loan amounts  
fig, axes = plt.subplots(1, 2, figsize=(12, 5))  
sns.histplot(df['loan_amount'], kde=True, ax=axes[0])  
axes[0].set_title("Original Loan Amount")  
  
sns.histplot(df['loan_scaled'], kde=True, color="green", ax=axes[1])  
axes[1].set_title("Scaled Loan Amount")  
plt.show()
```

Lab Activity: Data preparation 2

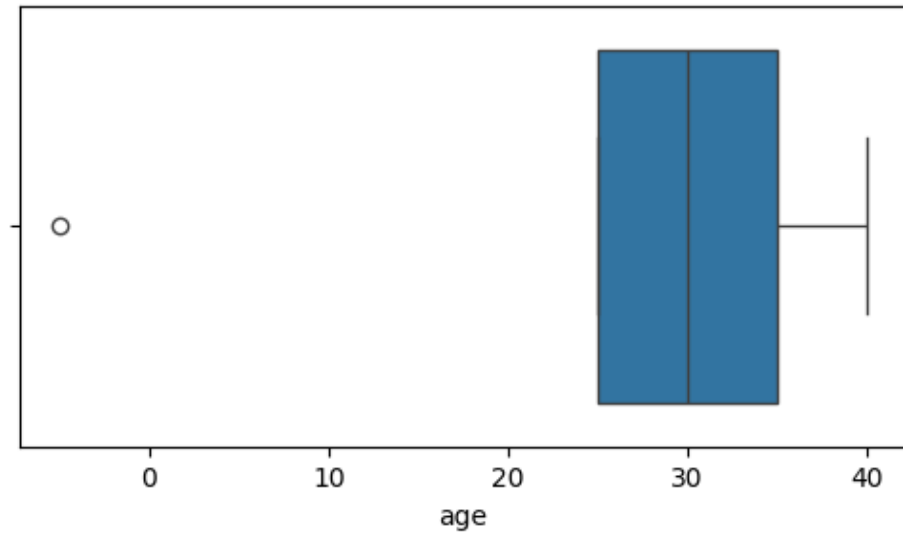
```
# =====  
# Step 8: Final Cleaned Dataset  
# =====
```

```
print("\nFinal Processed Dataset:")  
print(df.round(2))
```

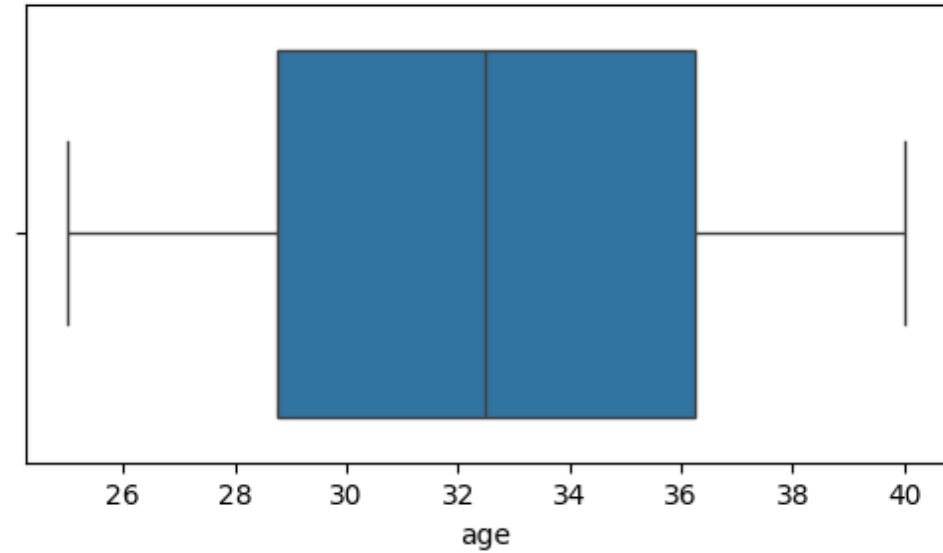
Chapter

1

Age Distribution Before Cleaning

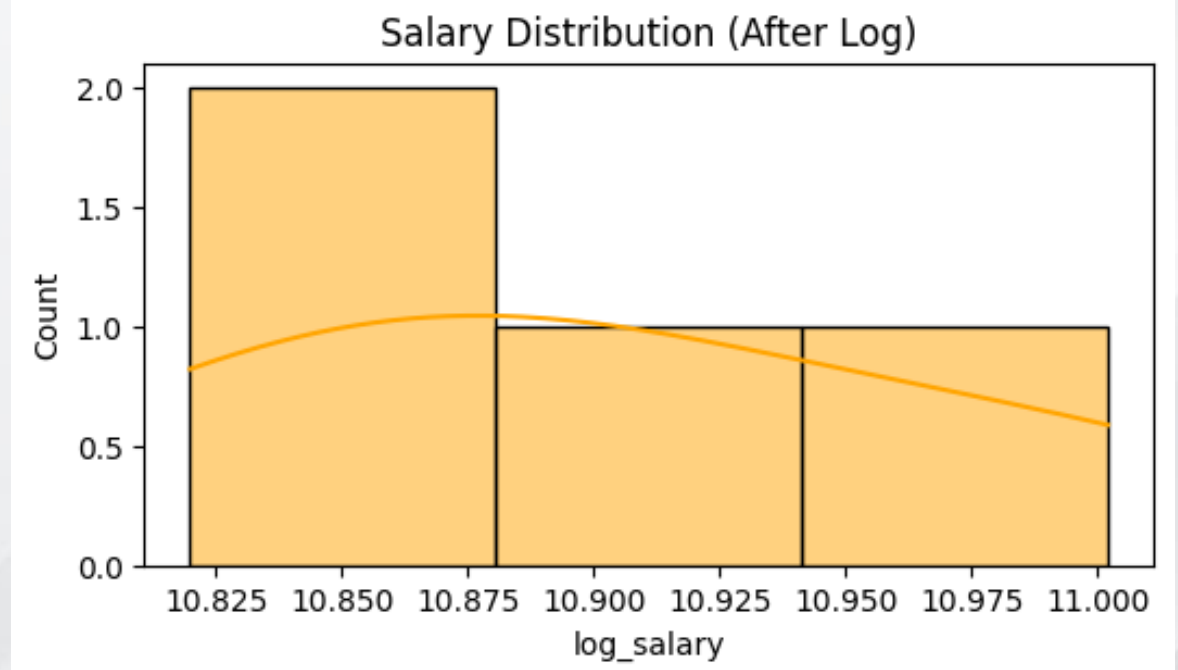
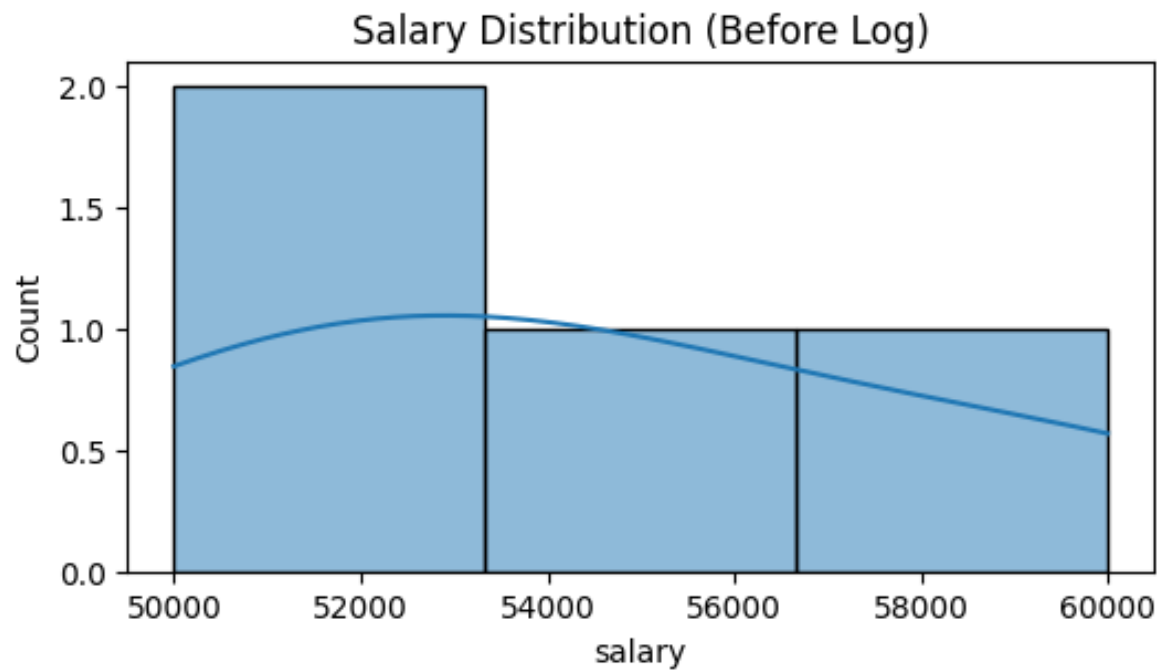


Age Distribution After Cleaning

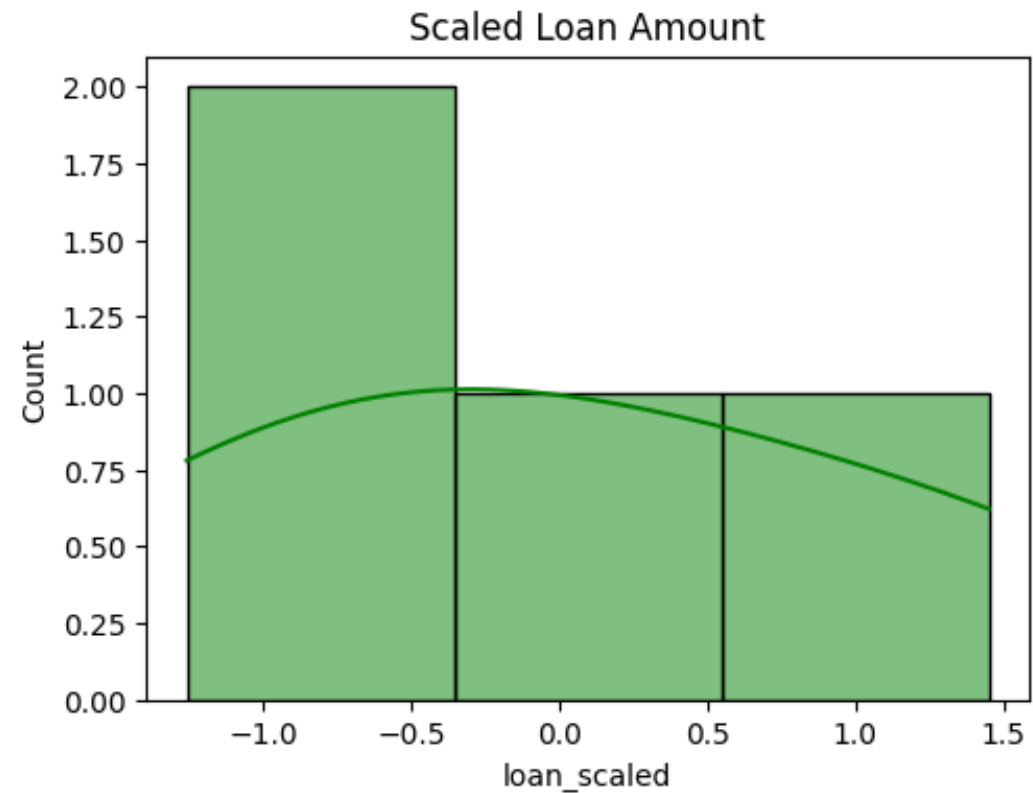
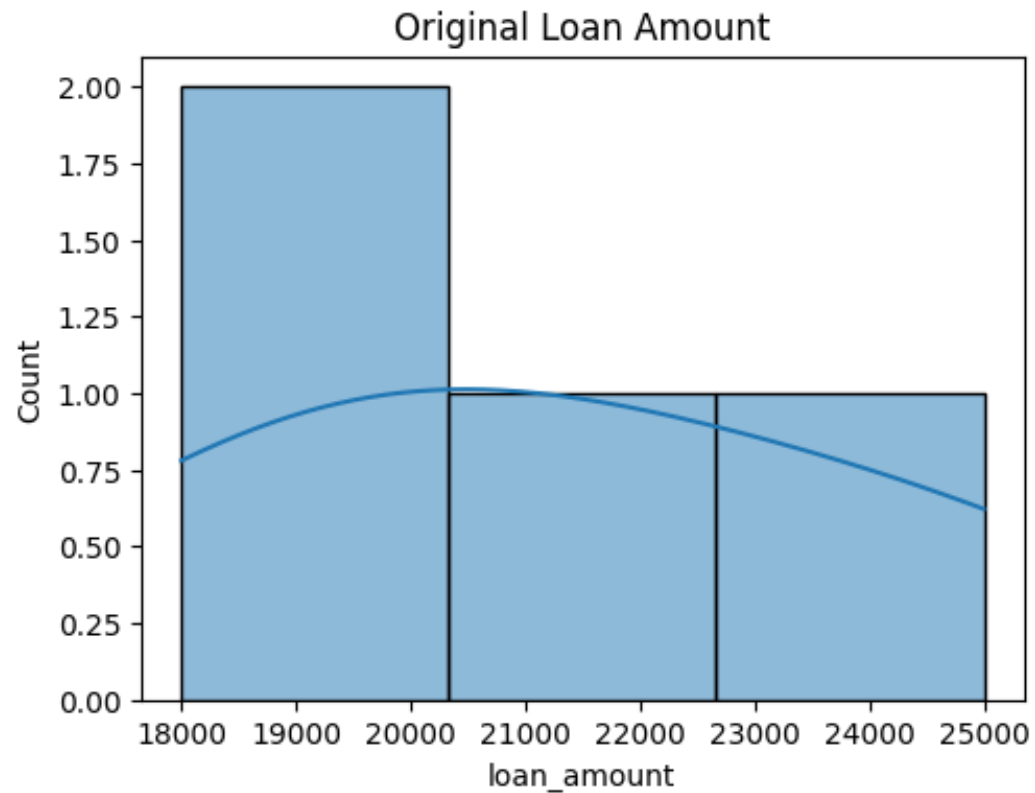


Chapter

1



Chapter 1



Lecture

3

Demo Lab Recap: What We Learned

- Data Integration
- Used `pd.merge()` to combine datasets by a common key (`customer_id`).
- Integrated risk score data with the main loan dataset.
- Handling Missing Values
- Identified missing data using `isnull().sum()`.
- Filled missing entries with mean or median using `fillna()`.
- Outlier Detection and Removal
- Used `sns.boxplot()` to visualize and remove unrealistic values (like negative ages).
- Variable Transformation

Lecture

3

Demo Lab Recap: What We Learned

- Applied logarithmic transformation `np.log()` to reduce skewness in salary distribution.
- Feature Encoding
- Converted categorical data into numerical form using `pd.get_dummies()` for the education variable.
- Feature Scaling
- Standardized the loan amount feature using `StandardScaler()` to normalize the scale of variables.
- Visualization
- Used Seaborn and Matplotlib to visualize before-and-after transformations for clear understanding.

References

- Han, Kamber, & Pei (2012). Data Mining: Concepts and Techniques.
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn & TensorFlow.
- Scikit-learn Documentation: <https://scikit-learn.org/stable/>
- Pandas Documentation: <https://pandas.pydata.org>