Project Title : Improving Data Integration Quality For Multi Source Analytics

Phase 2: Solution Architecture for Data Integration Quality

1. Overview of Data Integration and Quality Analysis (Harshita M Jain)

This phase focuses on improving the integration quality of data from multiple sources while ensuring high data quality standards. Exploratory Data Analysis (EDA) and Python-based automation are leveraged to identify, clean, and integrate data effectively.

Objectives:

1. Develop methods to identify and resolve data quality issues across sources.
2. Establish a framework for schema mapping, deduplication, and standardization.
3. Use Python for profiling, monitoring, and enhancing data integration workflows.

2. Data Cleaning and Preparation

2.1 Handling Missing Values

Approach: Address missing data in both numerical and categorical features to maintain the integrity of the analysis.

Implementation:

1. Numerical Features: Imputed using the mean or median.
2. Categorical Features: Assigned a placeholder like "Unknown" or imputed based on mode.

Code :

```
# Import necessary libraries

import pandas as pd

import numpy as np
```

```
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.decomposition import PCA


#  Import CSV Files

# Load datasets from local system

customer_df =
pd.read_csv(r"C:\Users\shett\Downloads\NAGA\Large_Customers_Dataset.csv"
)

transaction_df =
pd.read_csv(r"C:\Users\shett\Downloads\NAGA\Large_Transactions_Dataset.cs
v")

support_df =
pd.read_csv(r"C:\Users\shett\Downloads\NAGA\Large_Support_Tickets_Datase
t.csv")
```

2.2 Managing Outliers

1. Approach: Detect and handle outliers to prevent skewed results during integration.
2. Detection: Use boxplots and Z-score analysis.
3. Treatment: Apply Winsorization or exclude corrupted rows.

Code :

```
# Data Cleaning and Preparation

# Handling missing values (Simulated by inserting NaN)

customer_df.loc[:, 'Phone'] = customer_df['Phone'].fillna("Unknown")

transaction_df.loc[:, 'Amount'] =
transaction_df['Amount'].fillna(transaction_df['Amount'].median())
```

Remove duplicates

customer_df = customer_df.drop_duplicates()

transaction_df = transaction_df.drop_duplicates()

support_df = support_df.drop_duplicates()

2.3 Resolving Duplicates and Inconsistencies

Approach: Remove duplicate records and resolve logical inconsistencies across datasets.

Code :

Data Integration

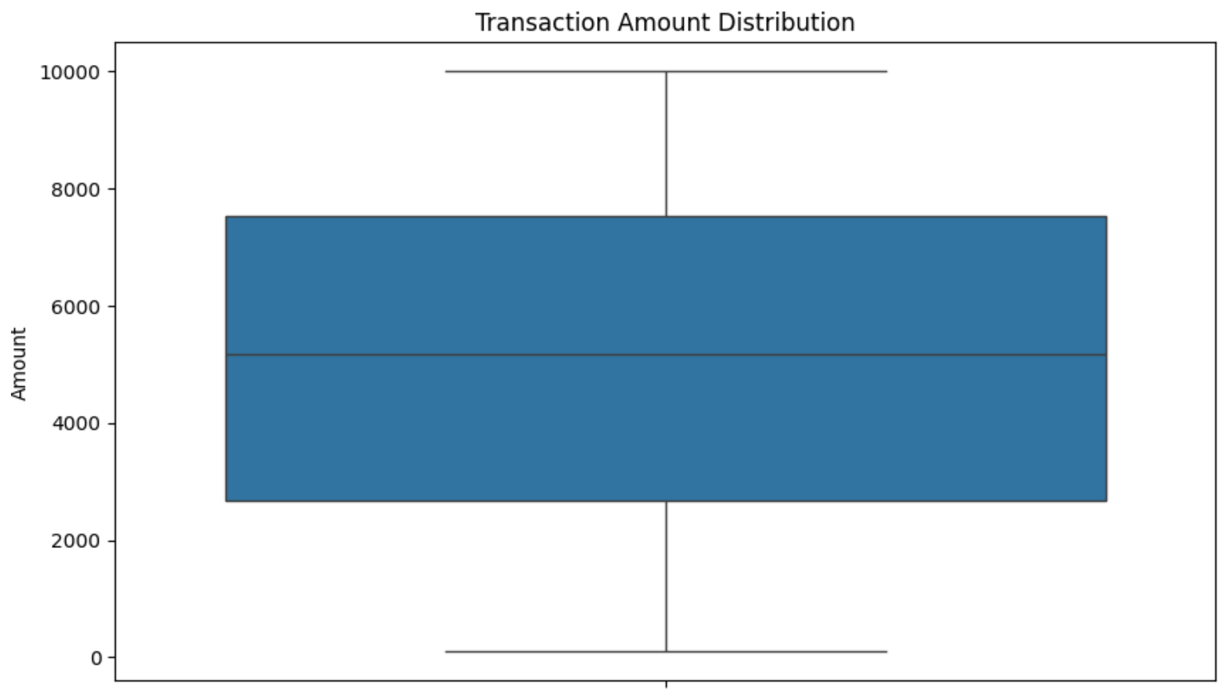
Merging datasets

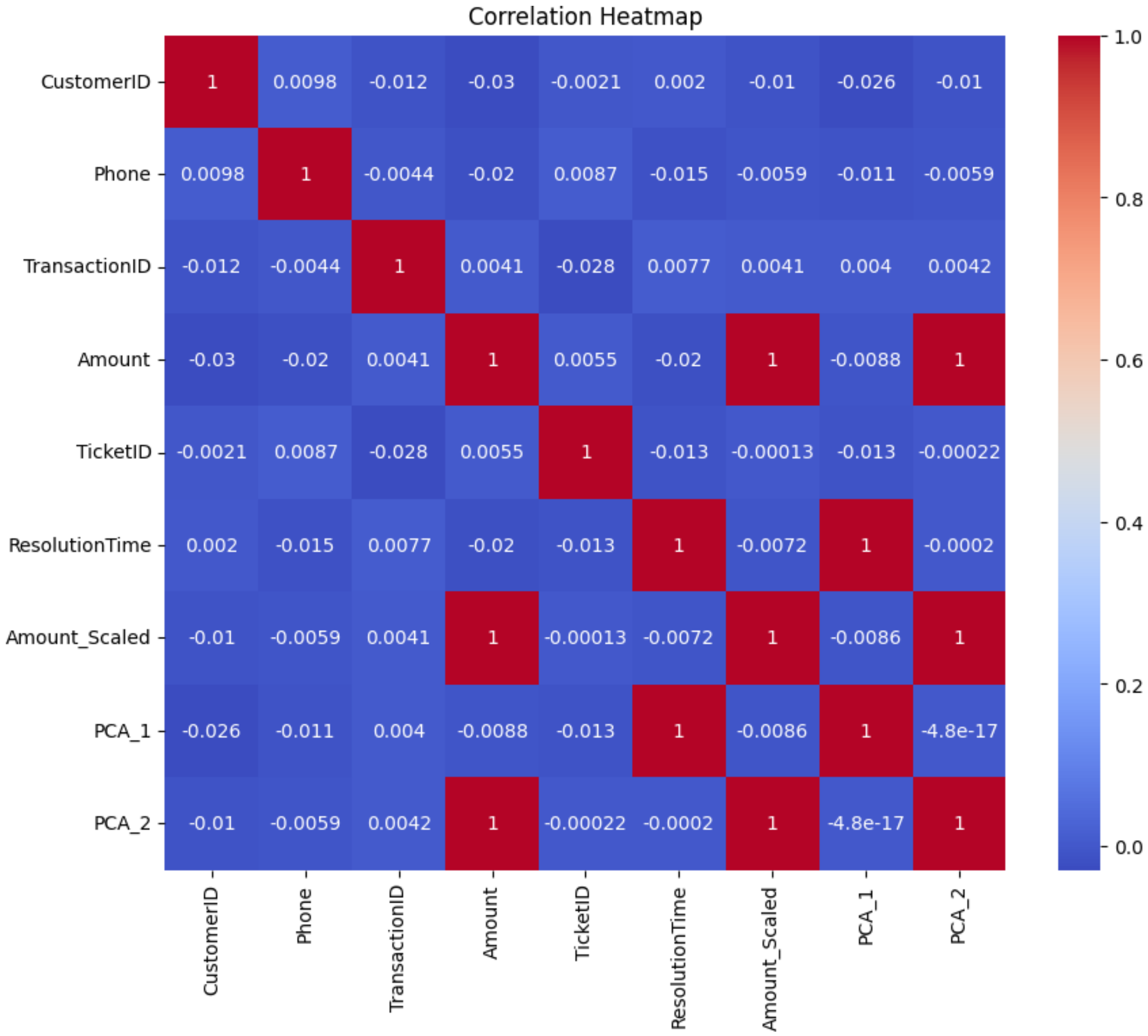
merged_df = pd.merge(customer_df, transaction_df, on='CustomerID', how='outer')

final_df = pd.merge(merged_df, s

upport_df, on='CustomerID', how='outer')

3. Data Visualization (Nagayashas A B)





3.1 Tools for Visualization

1. Matplotlib: Static visualizations for basic analysis.

2. Seaborn: Heatmaps for feature correlations.

3. Plotly: Interactive exploration of anomalies.

3.2 Key Visualizations and Insights

1. Schema Differences: Highlight variations between datasets.

2. Anomaly Detection: Identify problematic records using scatterplots and boxplots.

Code :

```
# Feature Engineering
# Scaling Amount and Resolution Time
scaler = StandardScaler()
if 'Amount' in final_df.columns:
    final_df['Amount_Scaled'] = scaler.fit_transform(final_df[['Amount']].fillna(0))
else:
    final_df['Amount_Scaled'] = 0


# Dimensionality Reduction using PCA
if 'ResolutionTime' in final_df.columns:
    pca = PCA(n_components=2)
    pca_features = pca.fit_transform(final_df[['Amount_Scaled', 
'ResolutionTime']].fillna(0))
    final_df['PCA_1'] = pca_features[:, 0]
    final_df['PCA_2'] = pca_features[:, 1]
else:
    final_df['PCA_1'] = 0
    final_df['PCA_2'] = 0
```

4. Schema Mapping and Data Integration (Isha Srivastava)

4.1 Schema Alignment

Align schemas from different sources to a unified structure.

4.2 Key Matching

Use fuzzy matching to identify similar records across datasets.

Code:

```
# Data Type Conversion for Correlation Analysis
# Ensure only numeric columns are used for correlation analysis
numeric_cols = final_df.select_dtypes(include=['float64', 'int64']).columns
if numeric_cols.any():
    final_numeric_df = final_df[numeric_cols].copy()
else:
    final_numeric_df = pd.DataFrame()
```

4.3 Merging Data (Isha Srivastava)

Combine data sources into a single dataset using Python.

Code :

```
# Data Visualization
# Correlation Heatmap
if not final_numeric_df.empty:
    plt.figure(figsize=(10, 8))
    sns.heatmap(final_numeric_df.corr(), annot=True, cmap='coolwarm')
    plt.title("Correlation Heatmap")
    plt.show()
else:
    print("No numeric data available for correlation heatmap.")


# Distribution of Transaction Amount
if 'Amount' in final_df.columns:
    plt.figure(figsize=(10, 6))
    sns.boxplot(final_df['Amount'])
    plt.title("Transaction Amount Distribution")
```

```python
    plt.show()
else:
    print("Transaction Amount column not found.")

# Scatterplot of PCA Components
if 'PCA_1' in final_df.columns and 'PCA_2' in final_df.columns:
    plt.figure(figsize=(10, 6))
    plt.scatter(final_df['PCA_1'], final_df['PCA_2'], alpha=0.6)
    plt.title("PCA Component Scatterplot")
    plt.xlabel("PCA 1")
    plt.ylabel("PCA 2")
    plt.show()
else:
    print("PCA components not available for scatterplot.")

# Save final integrated dataset
final_df.to_csv("Integrated_Dataset.csv", index=False)
print("Integrated dataset saved as 'Integrated_Dataset.csv'.")
```

5. Data Transformation and Feature Engineering

5.1 Feature Scaling

Apply scaling techniques to normalize data across sources.

5.2 Encoding Categorical Variables

Convert categorical data into numeric format using one-hot encoding.

6. Feasibility Assessment

6.1 Results from Integration and Cleaning

1. Evaluated schema differences and resolved inconsistencies.
2. Analyzed and visualized integrated data for actionable insights.

6.2 Metrics for Success

1. Completeness and accuracy scores for data integration.
2. Performance benchmarks for integrated datasets.

7. Conclusion

This phase established a framework for multi-source data integration, focusing on schema alignment, cleaning, and deduplication. Python-based tools streamlined the process, ensuring scalable and reliable integration.

Lessons Learned:

Profiling and EDA are essential for identifying integration challenges.

Iterative cleaning improved the reliability of integrated datasets.