

## Handling Missing Data in ETL

### SECTION A – THEORETICAL QUESTIONS

#### **Q1. What are the most common reasons for missing data in ETL pipelines?**

**Ans:-** Missing data in ETL pipelines commonly arises from source system limitations, extraction failures, or transformation errors. These issues can skew analytics and decisions if not addressed early. Understanding root causes helps in implementing preventive measures.

##### **Primary Causes**

Incomplete records from source databases often result from user input errors or optional fields left null. Extraction glitches, such as API failures or connection timeouts, lead to omitted rows during data pulls. System schema changes without detection create gaps when pipelines expect certain fields that no longer exist.

##### **Human and Configuration Errors**

Manual data entry mistakes, like omissions or typos, introduce missing values that propagate through pipelines. Incorrect field mappings during transformation misplace data, effectively causing apparent absences. Lack of null checks or default value rules fails to handle incomplete data properly.

##### **Technical Failures**

Network issues or transient errors during extraction drop records silently. Transformation logic errors, including bad joins or data type mismatches, result in null outputs. Poor validation skips freshness checks, allowing stale or partial datasets to load.

#### **Q2. Why is blindly deleting rows with missing values considered a bad practice in ETL?**

**Ans:-** Blindly deleting rows with missing values in ETL pipelines risks significant data loss and bias. This practice discards potentially valuable information without investigation, undermining analysis reliability.

##### **Data Loss Impact**

Deleting incomplete rows eliminates records that may contain useful partial data for other analyses. Financial or transactional datasets suffer most, as missing entries could represent actual zeros or unlogged events rather than errors. Partial records often hold insights in fields that remain intact, reducing dataset volume unnecessarily.

##### **Introduces Bias**

Removal disproportionately affects underrepresented groups or rare events, skewing statistical models and business metrics. Imbalanced datasets lead to flawed predictions, like underestimating customer churn in sparse demographics. Aggregate calculations, such as averages or totals, become inaccurate without accounting for the deleted volume.

## Better Alternatives

Impute values using means, medians, or advanced methods like regression for structured gaps. Route suspect rows to quarantine tables for review instead of permanent deletion. Implement validation rules early to flag issues without data destruction.

### **Q3. Explain the difference between: Listwise deletion Column deletion. Also mention one scenario where each is appropriate.**

**Ans:-** Listwise deletion and column deletion represent distinct strategies for handling missing data in ETL pipelines, each with trade-offs in data retention and analysis impact.

#### **Key Differences**

Listwise deletion removes entire rows containing any missing values across all columns, ensuring only complete records remain for analysis. Column deletion eliminates specific columns with excessive missing values while preserving rows, retaining more observations but losing feature information.

#### **Listwise Deletion Scenario**

This method suits large datasets with minimal missing data, such as less than 5% of rows affected in customer transaction logs. Machine learning model training benefits from uniform complete cases, avoiding imputation biases in regression tasks.

#### **Column Deletion Scenario**

Apply column deletion when a single feature, like optional survey responses, exceeds 70% missing values in demographic data. This preserves row volume for primary metrics while discarding low-value sparse attributes.

### **Q4. Why is median imputation preferred over mean imputation for skewed data such as income?**

**Ans:-** Median imputation outperforms mean imputation for skewed data like income because the median resists outliers and better captures the central tendency. The mean gets pulled toward extreme high values, distorting filled values and downstream analyses.

#### **Robustness to Outliers**

Income distributions typically show right skew from a few high earners, inflating the mean far above most values. Median stays at the middle point, unaffected by these tails, providing realistic imputations. This preserves the original data shape without artificial inflation.

#### **Preserves Distribution Shape**

Mean imputation reduces variance and shifts the distribution toward higher values in skewed sets, biasing models like regressions. Median maintains skewness and spread more accurately for real-world variables. Statistical tests and machine learning perform reliably without distortion.

### Income Scenario Example

For employee salary data with missing entries, median fills gaps at typical pay levels rather than overestimating via mean pulled by executives. This ensures accurate average compensation reports and fair modeling.

### Q5. What is forward fill and in what type of dataset is it most useful?

**Ans:-** Forward fill, also known as last observation carried forward (LOCF), replaces missing values with the most recent prior non-missing value in a sequence. This method propagates the last known data point forward until a new observation appears.

#### How It Works

In a dataset, forward fill scans sequentially and fills each null with the immediately preceding valid entry. For example, in a time series like daily stock prices with gaps, it assumes the price holds steady from the last recorded close.

#### Most Useful Dataset Type

Forward fill excels in time series data where values change slowly or remain stable between observations, such as sensor readings, financial quotes, or inventory levels. It maintains continuity without introducing artificial trends, ideal for sporadic missing points in chronological records.

### Q6. Why should flagging missing values be done before imputation in an ETL workflow?

**Ans:-** Flagging missing values before imputation in ETL workflows preserves transparency about data quality and enables better model performance. This step creates a binary indicator column marking original nulls, allowing downstream analyses to account for missingness patterns without losing that metadata.

#### Tracks Missingness Patterns

Imputation alone overwrites nulls, erasing evidence of gaps that may correlate with other variables, like systematic absences in certain customer segments. Flagging retains this signal as a feature, helping models detect and adjust for biases like MNAR mechanisms.

#### Avoids Downstream Confusion

Without flags, imputed values blend seamlessly with real data, misleading validation or auditing processes about true completeness. Flags support sensitivity analyses, comparing results with and without imputed rows for robust insights.

#### Improves Model Interpretability

Machine learning benefits from missing indicators as predictors, often boosting accuracy by capturing "missingness is meaningful" scenarios, such as non-responses indicating low engagement. This precedes

imputation to ensure flags reflect pre-fill states accurately.

**Q7. Consider a scenario where income is missing for many customers. How can this missingness itself provide business insights?**

**Ans:-** Missing income data for customers often signals behavioral patterns rather than random errors, turning missingness into a proxy feature for business segmentation. Analyzing who skips income disclosure reveals insights like privacy concerns among high-value users or survey fatigue in low-engagement segments.

### Privacy-Sensitive Segments

High-net-worth customers frequently omit income due to privacy fears, especially in financial services apps. Flagging these as a "missing\_income" indicator identifies premium users who avoid profiling, enabling targeted trust-building campaigns without imputation bias.

### Engagement and Churn Signals

Non-responders to income questions correlate with lower activity or higher churn risk, as seen in e-commerce where budget-conscious shoppers skip optional fields. This pattern predicts lifetime value better than averages, guiding retention strategies for at-risk cohorts.

### Product Fit Indicators

Missing income in loan applications flags prospects uncomfortable sharing finances, often indicating poor product fit or competitor loyalty. Businesses use this to refine qualification flows, reducing rejection rates by addressing disclosure barriers early.

## SECTION B – PRACTICAL QUESTIONS

**Q8. Listwise Deletion Remove all rows where Region is missing. Tasks: Identify affected rows Show the dataset after deletion Mention how many records were lost**

**Ans:-** To perform a **Listwise Deletion** in SQL, we use the DELETE statement combined with a WHERE clause that targets NULL values. In our provided dataset, this operation focuses specifically on the **Region** column.

### 1. Identify Affected Rows

In our current dataset, only **one row** matches the criteria for removal:

- **Customer 105 (Amit Verma):** This record contains a NULL value in the Region column.

While other rows (102, 104, 106, 107) have missing data in Monthly\_Sales or Income, they are **not** affected because their Region data is present.

## 2. SQL Query for Deletion

This query will permanently remove the record where the Region is missing:

SQL

```
Select * from sales;
```

```
SET SQL_SAFE_UPDATES = 0;
```

```
DELETE FROM Sales WHERE Region IS NULL;
```

```
SET SQL_SAFE_UPDATES = 1;
```

```
SELECT * FROM Sales;
```

## 3. Dataset After Deletion

After executing the query, the remaining dataset is as follows:

Customer_ID	Name	City	Monthly_Sales	Income	Region
101	Rahul Meheta	Mumbai	12000	65000	West
102	Anjali Rao	Bengaluru	NULL	NULL	South
103	Suresh Iyer	Chennai	15000	72000	South
104	Neha Singh	Dehli	NULL	NULL	South
106	Karan Shah	Ahmedabad	NULL	61000	West
107	Pooja Das	Kolkata	14000	NULL	East
108	Riya Kapoor	Jaipur	16000	69000	North

## 4. Records Lost

- **Total Records Removed:** 1 (Customer\_ID 105)
- **Impact:** Our dataset size decreased from 8 records to 7.

**Q9. Imputation Handle missing values in Monthly\_Sales using: Forward Fill Tasks: Apply forward fill Show before vs after values Explain why forward fill is suitable here**

**Ans:-** To handle missing values using Forward Fill (ffill), we replace a missing value with the most recent previous non-null value in the sequence. This is a common technique in time-series or ordered datasets.

1. Apply Forward Fill

Looking at your dataset, we focus on the Monthly\_Sales column. We look at the rows in order and "carry forward" the last known valid sales figure.

- Row 102 (Anjali Rao): Value is NULL. The previous value (Row 101) is 12000.
- Row 104 (Neha Singh): Value is NULL. The previous value (Row 103) is 15000.
- Row 106 (Karan Shah): Value is NULL. The previous value (Row 105) is 18000.

The SQL Query to Generate those Columns

- Because your original table still has NULL values, you need to use a query like this to see the results of the Forward Fill. This query creates the "Imputed\_Monthly\_Sales" (The After) and a "Status" column (The Source).

```

SELECT
    Customer_ID,
    Name,
    Monthly_Sales AS Before_Value,
    -- This part calculates the 'After' value (Forward Fill)
    COALESCE(Monthly_Sales,
        SELECT S2.Monthly_Sales
        FROM Sales S2
        WHERE S2.Customer_ID < S1.Customer_ID
        AND S2.Monthly_Sales IS NOT NULL
        ORDER BY S2.Customer_ID DESC
        LIMIT 1
    ) AS Imputed_Monthly_Sales,
    'Forward Fill' AS Status
FROM Sales S1
ORDER BY Customer_ID DESC
LIMIT 10;
  
```

```

)) AS After_Value,
-- This part creates the 'Source' column
CASE
    WHEN Monthly_Sales IS NOT NULL THEN 'Original'
    ELSE 'Carried Forward'
END AS Source_of_Value
FROM Sales S1;

```

## 2. Before vs. After Values

Customer_ID	Name	Before (Monthly_Sales)	After (Monthly_Sales)	Source of Value
101	Rahul Meheta	12000	12000	Original
102	Anjali Rao	NULL	12000	Carried from 101
103	Suresh Iyer	15000	15000	Original
104	Neha Singh	NULL	15000	Carried from 103
105	Amit Verma	18000	18000	Original
106	Karan Shah	NULL	18000	Carried from 105
107	Pooja Das	14000	14000	Original
108	Riya Kapoor	16000	16000	Original

---

## 3. Why is Forward Fill suitable here?

Forward fill is typically used under the assumption of Persistence or Local Similarity.

- **Consistency:** It assumes that the missing value is likely similar to the record immediately preceding it.
- **Sequential Logic:** If these records were sorted by date or entry time, forward fill assumes that the status hasn't changed since the last recording.

- No Data Loss: Unlike Listwise Deletion (which would have removed 3 more rows here), Forward Fill allows us to keep the records for Anjali, Neha, and Karan for analysis.

**Q10. Flagging Missing Data Create a flag column for missing Income. Tasks: Create Income\_Missing\_Flag (0 = present, 1 = missing) Show updated dataset Count how many customers have missing income**

**Ans:-** In data management, Flagging is a non-destructive way to handle missing values. Instead of deleting or changing the data, you create an indicator variable (also known as a "Dummy Variable") to highlight where data is incomplete for future analysis.

### 1. Create Income\_Missing\_Flag

To do this in SQL, we use a CASE statement. This evaluates whether the Income column is NULL and assigns a 1 or a 0 accordingly.

#### SQL

```
SELECT
```

```
Customer_ID,
```

```
Name,
```

```
Income,
```

```
CASE
```

```
WHEN Income IS NULL THEN 1
```

```
ELSE 0
```

```
END AS Income_Missing_Flag
```

```
FROM Sales;
```

### 2. Updated Dataset

Based on your current table (after the deletion of Customer 105), here is how the flag column looks:

Customer_ID	Name	Income	Income_Missing_Flag
101	Rahul Meheta	65000	0
102	Anjali Rao	NULL	1

Customer_ID	Name	Income	Income_Missing_Flag
103	Suresh Iyer	72000	0
104	Neha Singh	NULL	1
106	Karan Shah	61000	0
107	Pooja Das	NULL	1
108	Riya Kapoor	69000	0

### 3. Count of Missing Income

To get the total number of customers with missing income, you can run an aggregate query:

SQL

```
SELECT COUNT(*) AS Missing_Income_Count
FROM Sales
WHERE Income IS NULL;
```

**Total Customers with Missing Income: 3**

#### Why is Flagging useful?

- **Audit Trail:** It allows you to keep the original NULL values while still being able to filter or group by "missingness" later.
- **Machine Learning:** Many algorithms cannot handle NULL values directly, but they can use a 0/1 flag as a feature to understand patterns in missing data.
- **Reporting:** It makes it easy to generate a "Data Quality Report" to show stakeholders exactly how much information is missing.