# 10 Null Handling -KirkYagami

Null values are common in datasets, representing missing, unknown, or undefined data. Properly handling these null values is crucial for accurate data analysis and processing. This lecture will cover various aspects of null handling in PySpark DataFrames and SQL, providing practical examples using the `disney_raw` dataset.

## 1. Understanding Null Values in PySpark

- **Null Values**: Represent the absence of a value in a column. They are different from empty strings or zero values.
- **Implications**: Null values can impact aggregations, filtering, joins, and other data operations, leading to inaccurate results if not handled properly.

## 2. Detecting Null Values

- **Using `isNull` and `isNotNull`:**
  These functions are used to filter rows based on null or non-null values in a specific column.
  - **Example: Filtering Rows with Nulls in the `director` Column**

    ```
    disney_raw.filter(disney_raw["director"].isNull()).show(truncate=False)
    ```

    This filters and displays rows where the `director` column has null values.

  - **Example: Filtering Rows with Non-Nulls in the `director` Column**

    ```
    disney_raw.filter(disney_raw["director"].isNotNull()).show(truncate=False)
    ```

    This filters and displays rows where the `director` column is not null.

- **Counting Null Values:**
  To count the number of null values in a column, you can use the `isNull` function in combination with `count`.

  - **Example: Counting Nulls in the `metascore` Column**

    ```
    from pyspark.sql.functions import col

    disney_raw.filter(col("metascore").isNull()).count()
    ```

    This returns the number of rows with null values in the `metascore` column.

## 3. Replacing Null Values

- **Using `fillna` or `na.fill`:**
  PySpark provides the `fillna` function to replace null values with a specified value. This can be done for a single column or multiple columns.
  - **Example: Replacing Nulls in the `metascore` Column with 0**

```
disney_filled = disney_raw.fillna({"metascore": 0})
disney_filled.show(truncate=False)
```

This replaces all null values in the `metascore` column with `0`.

- **Example: Replacing Nulls in Multiple Columns**

```
disney_filled = disney_raw.fillna({"metascore": 0, "imdb_rating": 5.0})
disney_filled.show(truncate=False)
```

This replaces nulls in `metascore` with `0` and in `imdb_rating` with `5.0`.

- **Using `replace`:**
  The `replace` function allows for more flexible replacements, including replacing specific values.

  - **Example: Replacing Nulls in the `rated` Column with "Unrated"**

```
disney_replaced = disney_raw.replace({None: "Unrated"}, subset=["rated"])
disney_replaced.show(truncate=False)
```

This replaces null values in the `rated` column with "Unrated".

## 4. Dropping Rows with Null Values

- **Using `dropna`:**
  The `dropna` function removes rows with null values. You can specify how this is done using the `how` and `subset` parameters.
  - **Example: Dropping Rows with Any Null Values**

```
disney_no_nulls = disney_raw.dropna()
disney_no_nulls.show(truncate=False)
```

This drops rows that contain any null values.

- **Example: Dropping Rows with Nulls in Specific Columns**

```
disney_no_nulls_specific = disney_raw.dropna(subset=["director", "writer"])
disney_no_nulls_specific.show(truncate=False)
```

This drops rows where either the `director` or `writer` column contains a null value.

- **Using `how` Parameter:**
  The `how` parameter can take `any` (default) or `all`. The `any` option drops rows if any specified column has a null value, while the `all` option drops rows only if all specified columns are null.

  - **Example: Dropping Rows Where All Specified Columns are Null**

```
disney_no_nulls_all = disney_raw.dropna(how="all", subset=["director", "writer"])
disney_no_nulls_all.show(truncate=False)
```

This drops rows only where both `director` and `writer` are null.

## 5. Handling Nulls in Aggregations

- **Using `na.drop` in GroupBy:**
  Null values can cause issues in aggregations, especially when grouping data. You can drop nulls before performing aggregations.
  - **Example: Counting Movies by Genre Excluding Nulls**

```
from pyspark.sql.functions import desc

disney_raw.filter(disney_raw["genre"].isNotNull()) \
    .groupBy("genre") \
    .count() \
    .orderBy(desc("count")) \
    .show(10, truncate=False)
```

This counts the number of movies by genre, excluding null genres, and sorts the results by count in descending order.

- **Using `na.fill` in GroupBy:**
  Alternatively, you can replace nulls before grouping.
  - **Example: Replacing Null Genres with "Unknown" Before Counting**

```
disney_raw.fillna({"genre": "Unknown"}) \
    .groupBy("genre") \
    .count() \
    .orderBy(desc("count")) \
    .show(10, truncate=False)
```

This replaces null genres with "Unknown" before counting and sorting.

## 6. Handling Nulls in Joins

- **Inner Join:**
  An inner join automatically excludes rows with null values in the join columns.
  - **Example:**

Suppose we have another DataFrame with additional data:

```
additional_data = ss.createDataFrame([
    ("tt0147800", "Popular"),
    ("tt7019028", "Not Popular")
], ["imdb_id", "popularity"])

joined_df = disney_raw.join(additional_data, on="imdb_id", how="inner")
joined_df.show(truncate=False)
```

This performs an inner join on the `imdb_id` column, excluding rows where `imdb_id` is null.

- **Left Join:**
  A left join retains all rows from the left DataFrame, filling nulls for missing values in the right DataFrame.

  - **Example:**

```
joined_df = disney_raw.join(additional_data, on="imdb_id", how="left")
joined_df.show(truncate=False)
```

    This performs a left join, retaining all rows from `disney_raw` and filling nulls for missing `popularity` values.

## 7. Handling Nulls in Conditional Expressions

- **Using `when` and `otherwise`:**
  The `when` and `otherwise` functions allow conditional handling of null values.
  - **Example: Assigning a Default Value Based on Null Check**

```
from pyspark.sql.functions import when

disney_conditional = disney_raw.withColumn(
    "metascore_filled",
    when(col("metascore").isNull(), 50).otherwise(col("metascore"))
)
disney_conditional.show(truncate=False)
```

    This creates a new column `metascore_filled` where null values in `metascore` are replaced with `50`.

## 8. Null Handling in PySpark SQL

- **SQL Queries for Null Handling:**
  PySpark allows SQL-like syntax for more complex null handling scenarios.
  - **Example: Filtering Null Values in SQL**

```
ss.sql("""
    SELECT *
    FROM disney_view
    WHERE director IS NOT NULL
""").show(truncate=False)
```

    This query selects all rows where the `director` column is not null.

  - **Example: Replacing Nulls with Default Values in SQL**

```
ss.sql("""
    SELECT *,
    COALESCE(metascore, 50) AS metascore_filled
    FROM disney_view
""").show(truncate=False)
```

    This query replaces null values in the `metascore` column with `50` using the `COALESCE` function.

## 9. Real-World Scenarios and Best Practices

- **Scenario 1: Preparing Data for Machine Learning**
  In many machine learning workflows, nulls must be handled appropriately. You might choose to replace nulls with mean, median, or a specific value, depending on the column.

- **Scenario 2: Reporting and Analytics**
  Null handling is crucial in reporting to ensure accurate calculations. For instance, summing a column with nulls may require replacing nulls with zero.

- **Best Practices:**
  - Always assess the impact of null values on your analysis or processing tasks.
  - Choose the appropriate method for handling nulls (e.g., dropping, replacing) based on the context.
  - Consider the implications of the chosen null-handling method on the overall data quality and analysis results. For example, replacing nulls with a default value might introduce bias, while dropping rows with nulls could lead to data loss.

## 10. Advanced Null Handling Techniques

- **Using Window Functions with Nulls:**
  Window functions can be used to handle nulls in a more advanced way, such as filling null values based on the values in surrounding rows.
  - **Example: Filling Nulls with the Previous Value Using a Window Function**

```
from pyspark.sql.window import Window
from pyspark.sql.functions import last

window_spec = Window.orderBy("released_at").rowsBetween(Window.unboundedPreceding,
Window.currentRow)

disney_filled = disney_raw.withColumn(
    "metascore_filled",
    last("metascore", True).over(window_spec)
)
disney_filled.show(truncate=False)
```

  This fills null values in the `metascore` column with the last non-null value encountered in the `released_at` order.

- **Using UDFs (User-Defined Functions) for Custom Null Handling:**
  Sometimes, custom logic is required to handle nulls, which can be achieved using UDFs.

  - **Example: Custom UDF to Replace Nulls Based on Complex Conditions**

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def custom_null_handler(genre):
    if genre is None:
        return "Unspecified"
    elif "Comedy" in genre:
        return "Comedy-related"
```

```
    else:
        return genre

handle_nulls_udf = udf(custom_null_handler, StringType())

disney_custom_filled = disney_raw.withColumn("genre_custom",
handle_nulls_udf(col("genre")))
disney_custom_filled.show(truncate=False)
```

This example uses a UDF to replace null values in the `genre` column with "Unspecified," and if the genre contains "Comedy," it is replaced with "Comedy-related."

## 11. Conclusion

Null handling is a critical aspect of data processing and analysis in PySpark. Depending on the context, different strategies such as dropping nulls, filling nulls, and advanced techniques like using window functions or UDFs can be employed. Properly handling null values ensures data integrity and reliability, leading to more accurate and meaningful insights.

By incorporating these techniques and practices into your PySpark workflows, you'll be better equipped to manage null values effectively, improving the quality of your data processing and analysis.