05 Selecting Renaming Adding Dropping Columns - KirkYagami 🧑‍💻🕵️

Loading the Dataset

To load the dataset into a Spark DataFrame:

```python
disney_raw = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .option("multiline", "true") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .load("disney_plus_shows.csv")
```

This creates a DataFrame named `disney_raw` with the following columns:
['imdb_id', 'title', 'plot', 'type', 'rated', 'year', 'released_at', 'added_at', 'runtime', 'genre', 'director', 'writer', 'actors', 'language', 'country', 'awards', 'metascore', 'imdb_rating', 'imdb_votes']

Basic DataFrame Operations

1. Viewing the schema:

```python
disney_raw.printSchema()
```

2. Displaying the first few rows:

```python
disney_raw.show(5)
```

3. Getting basic statistics:

```python
disney_raw.describe().show()
```

Selecting and Filtering Data

1. Selecting specific columns:

```python
disney_raw.select("title", "year", "imdb_rating").show(5)
```

2. Filtering data:

```python
disney_raw.filter(disney_raw.year > 2020).show(5)
```

3. Combining selection and filtering:

```
disney_raw.select("title", "year", "imdb_rating") \
    .filter(disney_raw.imdb_rating > 8.0) \
    .show(5)
```

Aggregations and Grouping

1. Counting the number of shows by type:

```
disney_raw.groupBy("type").count().show()
```

2. Average IMDB rating by year:

```
disney_raw.groupBy("year") \
    .agg({"imdb_rating": "avg"}) \
    .orderBy("year") \
    .show(5)
```

Adding New Columns

1. Adding a column for the decade:

```
from pyspark.sql.functions import col, floor

disney_with_decade = disney_raw.withColumn("decade", floor(col("year") / 10) * 10)
disney_with_decade.select("title", "year", "decade").show(5)
```

To add, rename, and drop columns in a Spark DataFrame, you can use the following methods:

Adding Columns

To add a new column to a DataFrame, use the `withColumn` method. This method allows you to create a new column or replace an existing one with the same name.

Example: Adding a new column for the length of the `plot` description

```
from pyspark.sql.functions import length

disney_with_plot_length = disney_raw.withColumn("plot_length", length(col("plot")))
disney_with_plot_length.select("title", "plot", "plot_length").show(5)
```

Renaming Columns

To rename a column, use the `withColumnRenamed` method. This method takes two arguments: the existing column name and the new column name.

Example: Renaming the `imdb_rating` column to `rating`

```
disney_renamed = disney_raw.withColumnRenamed("imdb_rating", "rating")
disney_renamed.select("title", "rating").show(5)
```

Dropping Columns

To drop a column, use the `drop` method. You can drop one or more columns by passing their names as arguments.

Example: Dropping the `plot` column

```
disney_dropped = disney_raw.drop("plot")
disney_dropped.show(5)
```

Example: Dropping multiple columns (`plot` and `awards`)

```
disney_dropped_multiple = disney_raw.drop("plot", "awards")
disney_dropped_multiple.show(5)
```

Filtering Rows

Filtering rows allows you to select a subset of data based on specific conditions. You use the `filter` or `where` method for this purpose.

Examples:

1. **Filtering based on a single condition:**

   ```
   # Filter shows released after the year 2020
   disney_filtered = disney_raw.filter(disney_raw.year > 2020)
   disney_filtered.show(5)
   ```

2. **Filtering using multiple conditions:**

   ```
   # Filter shows with IMDb rating greater than 8.0 and released after 2019
   disney_filtered = disney_raw.filter((disney_raw.imdb_rating > 8.0) & (disney_raw.year >
   2019))
   disney_filtered.sho w(5)
   ```

3. **Filtering with `where` (an alias for `filter`):**

   ```
   # Same filter condition using `where`
   disney_filtered = disney_raw.where((disney_raw.imdb_rating > 8.0) & (disney_raw.year >
   2019))
   disney_filtered.show(5)
   ```

4. **Filtering using SQL expressions:**

```
# Filter using SQL-like expressions
disney_filtered = disney_raw.filter("imdb_rating > 8.0 AND year > 2019")
disney_filtered.show(5)
```

Dropping Rows

Dropping rows can be based on specific conditions or to remove rows with null values.

Examples:

1. **Dropping rows with null values in any column:**

```
# Drop rows with any null values
disney_cleaned = disney_raw.dropna()
disney_cleaned.show(5)
```

2. **Dropping rows with null values in specific columns:**

```
# Drop rows where 'title' or 'imdb_rating' columns have null values
disney_cleaned = disney_raw.dropna(subset=["title", "imdb_rating"])
disney_cleaned.show(5)
```

3. **Dropping rows based on a condition:**

```
# Drop rows where IMDb rating is less than or equal to 5.0
disney_filtered = disney_raw.filter(disney_raw.imdb_rating > 5.0)
disney_filtered.show(5)
```

4. **Dropping duplicate rows:**

```
# Drop duplicate rows based on all columns
disney_no_duplicates = disney_raw.dropDuplicates()
disney_no_duplicates.show(5)
```

Drop duplicates based on specific columns:

```
# Drop duplicates based on 'title' column
disney_no_duplicates = disney_raw.dropDuplicates(["title"])
disney_no_duplicates.show(5)
```

Handling Missing Data

1. Checking for null values:

```python
from pyspark.sql.functions import col, count, when

disney_raw.select([count(when(col(c).isNull(), c)).alias(c) for c in
disney_raw.columns]).show()
```

2. Dropping rows with null values:

```python
disney_clean = disney_raw.dropna()
```

Saving the DataFrame

To save the processed DataFrame:

```python
disney_clean.write.format("parquet").save("disney_clean.parquet")
```