# Methods and Functions in PySpark

## Data Aggregations and Filtering

### Filter

The **filter()** function in PySpark is used to select rows from a DataFrame based on specified conditions. It is equivalent to the SQL WHERE clause.

Example: Find flights in **December 2023**:

df\_dec\_2023 = df.filter((col("MONTH\_NUM") == 12) & (col("YEAR") == 2023))

df\_dec\_2023.show(5)

### GroupBy

The **groupBy()** function is used to group data by one or more columns, similar to GROUP BY in SQL. It is often used with **aggregate functions** like sum(), count(), or avg().

#Ordering the number of Airports by State

df\_december.groupBy("STATE\_NAME").count().orderBy(desc("count")).show()

### OrderBy

The **orderBy()** function is used to sort the DataFrame based on one or more columns.

df\_sorted = df\_grouped.orderBy(col("Total Flights").desc()) df\_sorted.show(5)

### Renaming Columns – alias()

The **alias()** function is used to rename a column temporarily. This is useful when performing aggregations.

df\_grouped = df.groupBy("STATE\_NAME").agg(sum("TOTAL\_FLIGHTS").alias("Total Flights"))

### Combining Filter-Group By-Order By-Alias

df\_december = df.filter(col("MONTH\_NUM") == 12)

df\_top\_countries = df\_december.groupBy("STATE\_NAME").agg(

sum("TOTAL\_FLIGHTS").alias("Total December Flights")

).orderBy(col("Total December Flights").desc())

### countDistinct

This allows us to get unique values in our column.

# Count unique airports per state df\_airports\_per\_state = df\_december.groupBy("STATE\_NAME").agg( countDistinct("APT\_ICAO").alias("Number of Airports") )

## Data Inspection

### .printSchema()

Use .printSchema() to display the schema of your DataFrame, including column names and data types.

df\_december\_Türkiye.printSchema()

### .describe()

This method gives **summary statistics** (e.g., mean, standard deviation, min, max) for numeric columns.

df\_december\_Türkiye.describe().show()

### .dtypes

Use .dtypes to get the data types of each column in your DataFrame.

**Example:**

python

print(df\_december\_Türkiye.dtypes)

### .columns

Use .columns to get the list of **column names** in the DataFrame.

**Example:**

python

print(df\_december\_Türkiye.columns)

🔹 **Explanation**: Returns a list of all the column names in the DataFrame.

### .count()

This function counts the total number of rows in the DataFrame.

**Example:**

python

row\_count = df\_december\_Türkiye.count()

print(f"Number of rows: {row\_count}")

🔹 **Explanation**: Returns the number of rows in the DataFrame.

### .take()

.take(n) returns the first **n rows** of the DataFrame as a list of Row objects.

**Example:**

python

first\_three\_rows = df\_december\_Türkiye.take(3)

print(first\_three\_rows)

🔹 **Explanation**: Returns the first 3 rows from the DataFrame.

### .isEmpty()

Checks if the DataFrame is empty (i.e., contains no rows).

**Example:**

python

is\_empty = df\_december\_Türkiye.isEmpty()

print(f"Is the DataFrame empty? {is\_empty}")

🔹 **Explanation**: Returns True if the DataFrame is empty, False otherwise.

### .cache()

Caches the DataFrame to memory, which can speed up repeated operations on the DataFrame.

**Example:**

python

df\_december\_Türkiye.cache()

🔹 **Explanation**: Caches the DataFrame, useful for performance optimization when working with large datasets.

### .distinct()

Gets the **unique rows** in the DataFrame.

**Example:**

df\_unique\_airports = df\_december\_Türkiye.select("APT\_NAME").distinct()

df\_unique\_airports.show()

🔹 **Explanation**: Returns a DataFrame with distinct values for the APT\_NAME column.

### .sample()

Use .sample() to randomly sample a fraction of the rows.

**Example:**

python

sample\_data = df\_december\_Türkiye.sample(fraction=0.1)

sample\_data.show()

🔹 **Explanation**: Samples **10%** of the rows in the DataFrame. You can change the fraction to suit your needs.

### .first()

Returns the **first row** of the DataFrame as a Row object.

**Example:**

python

first\_row = df\_december\_Türkiye.first()

print(first\_row)

🔹 **Explanation**: Retrieves and prints the first row of the DataFrame.

### .limit()

Limits the DataFrame to a specified number of rows.

**Example:**

python

limited\_data = df\_december\_Türkiye.limit(10)

limited\_data.show()

🔹 **Explanation**: Limits the DataFrame to the first **10 rows**.

## Data Cleaning

### isNull()

You can check for **null values** in a column using the isNull() function combined with filter().

**Example: Check if a Column has Null Values (isNull())**

python

df\_december\_Türkiye.filter(col("APT\_NAME").isNull()).show()

🔹 **Explanation**: This will filter the rows where APT\_NAME is null and display them. If there are any rows with null values in APT\_NAME, they will be shown.

**To Count Rows with Null Values**

python

null\_count = df\_december\_Türkiye.filter(col("APT\_NAME").isNull()).count()

print(f"Number of null values in APT\_NAME: {null\_count}")

🔹 **Explanation**: This counts how many rows in the APT\_NAME column have null values.

### dropna()

You can drop rows that contain missing (null) values in one or more columns.

**Example: Drop Rows with Any Missing Values**

python

df\_december\_Türkiye = df\_december\_Türkiye.dropna()

🔹 **Explanation**: Drops rows with any null values in the DataFrame.

You can drop rows with missing values in specific columns only.

**Example: Drop Rows Where FLT\_TOT\_1 is Missing**

python

df\_december\_Türkiye = df\_december\_Türkiye.dropna(subset=["FLT\_TOT\_1"])

🔹 **Explanation**: Drops rows where FLT\_TOT\_1 has null values.

### fillna()

You can replace missing values with a specified value (like 0, mean, or mode).

**Example: Fill Missing Values in FLT\_TOT\_1 with 0**

python

df\_december\_Türkiye = df\_december\_Türkiye.fillna({"FLT\_TOT\_1": 0})

🔹 **Explanation**: Fills missing values in the FLT\_TOT\_1 column with 0.

You can also fill multiple columns at once.

**Example: Fill Missing Values for Multiple Columns**

python

df\_december\_Türkiye = df\_december\_Türkiye.fillna({"FLT\_DEP\_1": 0, "FLT\_ARR\_1": 0})

🔹 **Explanation**: Fills missing values in both FLT\_DEP\_1 and FLT\_ARR\_1 with 0.

### dropDuplicates()

#### a. Remove Duplicate Rows (dropDuplicates())

This function removes all rows that are exactly the same across all columns.

Example: Remove All Duplicates

df\_december\_Türkiye = df\_december\_Türkiye.dropDuplicates()

🔹 Explanation: Removes rows where all columns have identical values.

#### b. Remove Duplicates Based on Specific Columns

You can also remove duplicates based on specific columns.

**Example: Remove Duplicates Based on APT\_NAME**

python

df\_december\_Türkiye = df\_december\_Türkiye.dropDuplicates(["APT\_NAME"])

🔹 **Explanation**: Removes duplicates where the APT\_NAME column has identical values.

### filter()

#### a. Filter Invalid Data (filter() or where())

You can use filter() or where() to keep only valid rows based on conditions.

**Example: Filter Rows with FLT\_TOT\_1 Greater Than 0**

df\_december\_Türkiye = df\_december\_Türkiye.filter(col("FLT\_TOT\_1") > 0)

🔹 **Explanation**: Filters out rows where FLT\_TOT\_1 is less than or equal to 0.

#### b. Filter Invalid Data with Multiple Conditions

You can filter rows based on multiple conditions.

**Example: Filter Rows Where APT\_NAME is Not Null and FLT\_TOT\_1 > 0**

df\_december\_Türkiye = df\_december\_Türkiye.filter(

col("APT\_NAME").isNotNull() & (col("FLT\_TOT\_1") > 0)

)

🔹 **Explanation**: Keeps rows where APT\_NAME is not null and FLT\_TOT\_1 is greater than 0.

### Cast() and withColumn()

Sometimes, the data type of a column may need to be changed for correct processing.

#### a. Cast Columns to Different Data Types (cast())

You can use the cast() function to change the data type of a column.

**Example: Cast FLT\_TOT\_1 to Integer**

df\_december\_Türkiye = df\_december\_Türkiye.withColumn(

"FLT\_TOT\_1", col("FLT\_TOT\_1").cast("integer")

)

🔹 **Explanation**: Changes the data type of FLT\_TOT\_1 to **integer**.

#### b. Cast Multiple Columns to New Data Types

If you need to change the types of several columns, you can do it in one go.

**Example: Cast Multiple Columns to Integer**

df\_december\_Türkiye = df\_december\_Türkiye.withColumn(

"FLT\_DEP\_1", col("FLT\_DEP\_1").cast("integer")

).withColumn(

"FLT\_ARR\_1", col("FLT\_ARR\_1").cast("integer")

)

🔹 **Explanation**: Changes FLT\_DEP\_1 and FLT\_ARR\_1 to **integer** data type.

### withColumnRenamed()

You can rename columns using this method.

**Example: Rename APT\_NAME to Airport\_Name**

df\_december\_Türkiye = df\_december\_Türkiye.withColumnRenamed(

"APT\_NAME", "Airport\_Name"

)

🔹 **Explanation**: Renames the APT\_NAME column to Airport\_Name.

### Dealing with Outliers

Outliers can significantly impact your analysis, so identifying and dealing with them is crucial.

**a. Remove Outliers Using filter()**

You can filter out rows where values exceed certain thresholds, effectively removing outliers.

**Example: Remove Rows Where FLT\_TOT\_1 is Greater Than 1000**

df\_december\_Türkiye = df\_december\_Türkiye.filter(col("FLT\_TOT\_1") <= 1000)

🔹 **Explanation**: Filters out rows where FLT\_TOT\_1 is greater than 1000 (removing outliers).

### Handling Categorical Data

In some cases, categorical data may need to be cleaned or transformed into numerical values (e.g., encoding categories).

**a. Replace Values (replace())**

You can replace specific values in a column.

**Example: Replace NULL with 'Unknown' in APT\_NAME**

df\_december\_Türkiye = df\_december\_Türkiye.replace(

None, "Unknown", subset=["APT\_NAME"]

)

🔹 **Explanation**: Replaces None (null) values with "Unknown" in the APT\_NAME column.

## Window Functions

A window function requires a **window specification**, which consists of:

1. **Partitioning (PARTITION BY)** - Defines the subset of data for the function.
2. **Ordering (ORDER BY)** - Specifies the order of rows within each partition.
3. **Frame Specification (Optional)** - Defines a subset of rows within the partition for computation.

### **Importing Required Libraries**

To use window functions, you need to import the necessary modules:

from pyspark.sql.window import Window

from pyspark.sql.functions import row\_number, rank, dense\_rank, percent\_rank, ntile, lag

### Types of Window Functions

#### A. Ranking Functions

* row\_number()

Each row gets a **unique** number, starting from 1 within each partition.

window\_spec = Window.partitionBy("STATE\_NAME").orderBy(col("FLT\_TOT\_1").desc())

df\_window = df\_december\_Türkiye.withColumn("row\_number", row\_number().over(window\_spec))

df\_window.show()

🔹 **Example Output:**

diff

CopyEdit

+--------+-----------+--------------+

|APT\_NAME|STATE\_NAME |row\_number |

+--------+-----------+--------------+

|Airport1| Türkiye | 1 |

|Airport2| Türkiye | 2 |

|Airport3| Türkiye | 3 |

+--------+-----------+--------------+

✅ **Key Points:** No duplicates in row numbers; ordering determines assignment.

* rank()

Ranks rows, but if values are **tied**, it assigns the same rank and skips numbers.

df\_window = df\_december\_Türkiye.withColumn("rank", rank().over(window\_spec))

df\_window.show()

🔹 **Example Output:**

+--------+-----------+------+

|APT\_NAME|STATE\_NAME |rank |

+--------+-----------+------+

|Airport1| Türkiye | 1 |

|Airport2| Türkiye | 2 |

|Airport3| Türkiye | 2 |

|Airport4| Türkiye | 4 |

+--------+-----------+------+

✅ **Key Points:** Ties get the same rank, skipping the next number.

* dense\_rank()

Similar to rank(), but **doesn’t skip numbers** after ties.

df\_window = df\_december\_Türkiye.withColumn("dense\_rank", dense\_rank().over(window\_spec))

df\_window.show()

🔹 **Example Output:**

r

CopyEdit

+--------+-----------+----------+

|APT\_NAME|STATE\_NAME |dense\_rank|

+--------+-----------+----------+

|Airport1| Türkiye | 1 |

|Airport2| Türkiye | 2 |

|Airport3| Türkiye | 2 |

|Airport4| Türkiye | 3 |

+--------+-----------+----------+

✅ **Key Points:** Ties get the same rank, but the next rank is not skipped.

* percent\_rank()

Ranks each row as a percentage of total rows in the partition.

python

CopyEdit

df\_window = df\_december\_Türkiye.withColumn("percent\_rank", percent\_rank().over(window\_spec))

df\_window.show()

✅ **Key Points:** Useful for **normalized ranking** based on percentages.

* ntile()

#### **Distributes Rows into Buckets**

Divides rows into **n equal groups** (quartiles, deciles, etc.).

python

CopyEdit

df\_window = df\_december\_Türkiye.withColumn("quartile", ntile(4).over(window\_spec))

df\_window.show()

✅ **Key Points:** Useful for **percentile-based** analysis.