PySpark

Top 100 Pyspark Functions

1. **Create an empty DataFrame**

You can create an empty DataFrame using spark.createDataFrame with no data.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local").appName("Empty DataFrame").getOrCreate()

# Empty DataFrame with no data
empty_df = spark.createDataFrame([], "id INT, name STRING")
empty_df.show()
```

2. Convert RDD to DataFrame

To convert an RDD to DataFrame, you need to define the schema.

```
rdd = spark.sparkContext.parallelize([(1, 'Alice'), (2, 'Bob')])
columns = ["id", "name"]
df_from_rdd = rdd.toDF(columns)
df_from_rdd.show()
```

3. Convert DataFrame to Pandas

You can convert a PySpark DataFrame to a Pandas DataFrame using toPandas().

```
pandas_df = df_from_rdd.toPandas()
print(pandas_df)
```

4. show()

The show() method displays the first n rows of a DataFrame.

```
df_from_rdd.show(5)  # Display first 5 rows
```

5. StructType & StructField

These classes are used to define the schema for DataFrames.

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

schema = StructType([
   StructField("id", IntegerType(), True),
   StructField("name", StringType(), True)
])
df_with_schema = spark.createDataFrame([(1, "Alice"), (2, "Bob")], schema)
df_with_schema.show()
```

6. Column Class

The Column class represents a column in a DataFrame and is used for performing operations.

```
from pyspark.sql import functions as F
df_with_column = df_from_rdd.withColumn("upper_name", F.upper(df_from_rdd['name']))
df_with_column.show()
```

7. select()

The select() method is used to select specific columns.

```
df_from_rdd.select("id").show()  # Select only the 'id' column
```

8. collect()

collect() returns all the rows as a list of Row objects.

```
rows = df_from_rdd.collect()
print(rows)
```

9. withColumn()

This method is used to add or modify a column.

```
df_with_new_col = df_from_rdd.withColumn("id_squared", df_from_rdd["id"] * df_from_rdd["id"])
df_with_new_col.show()
```

10. withColumnRenamed()

Renames an existing column in the DataFrame.

```
df_renamed = df_from_rdd.withColumnRenamed("name", "full_name")
df_renamed.show()
```

11. where() & filter()

Both methods are used to filter rows based on conditions.

```
df_filtered = df_from_rdd.where(df_from_rdd['id'] > 1)
df_filtered.show()

# Alternatively, you can use filter()
df_filtered2 = df_from_rdd.filter(df_from_rdd['id'] > 1)
df_filtered2.show()
```

12. drop() & dropDuplicates()

Used to drop a column or remove duplicate rows.

```
# Dropping a column
df_dropped = df_from_rdd.drop("name")
df_dropped.show()

# Removing duplicates
df_no_duplicates = df_from_rdd.dropDuplicates()
df_no_duplicates.show()
```

13. orderBy() and sort()

These methods are used for sorting data in DataFrame.

```
df_sorted = df_from_rdd.orderBy("id", ascending=False)
df_sorted.show()

# Equivalent to orderBy()
df_sorted2 = df_from_rdd.sort("id")
df_sorted2.show()
```

14. groupBy()

Used for group-by operations.

```
df_grouped = df_from_rdd.groupBy("id").count()
df_grouped.show()
```

15. join()

Used for joining DataFrames.

```
df2 = spark.createDataFrame([(1, 'Math'), (2, 'Science')], ["id", "subject"])
df_joined = df_from_rdd.join(df2, on="id")
df_joined.show()
```

16. union() & unionAll()

Both methods combine DataFrames, but unionAll is deprecated in favor of union().

```
df3 = spark.createDataFrame([(3, "Charlie")], ["id", "name"])
df_union = df_from_rdd.union(df3)
df_union.show()
```

17. unionByName()

Union DataFrames by column name.

```
df_union_by_name = df_from_rdd.unionByName(df2)
df_union_by_name.show()
```

18. UDF (User Defined Function)

UDFs are used to extend the functionality of Spark DataFrame with custom logic.

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

def add_exclamation(name):
    return name + "!"

add_udf = udf(add_exclamation, StringType())
df_udf = df_from_rdd.withColumn("excited_name", add_udf("name"))
df_udf.show()
```

19. transform()

transform() is used to apply transformations to a DataFrame.

```
df_transformed = df_from_rdd.transform(lambda df: df.withColumn("id_squared", df["id"]**2))
df_transformed.show()
```

20. apply()

Similar to transform(), apply() can be used for element-wise operations.

```
# Applying to every row
df_applied = df_from_rdd.rdd.map(lambda row: (row.id * 2, row.name)).toDF(["id", "name"])
df_applied.show()
```

21. map()

map() is used on an RDD to apply a function on each element.

```
rdd_mapped = df_from_rdd.rdd.map(lambda x: (x.id * 2, x.name))
df_mapped = rdd_mapped.toDF(["id", "name"])
df_mapped.show()
```

22. flatMap()

Used to flatten a collection of items.

```
rdd_flat = df_from_rdd.rdd.flatMap(lambda x: [(x.id, x.name), (x.id * 10, x.name)])
df_flat = rdd_flat.toDF(["id", "name"])
df_flat.show()
```

23. foreach()

foreach() is used for applying a function to each row in the DataFrame.

```
def print_row(row):
    print(row)

df_from_rdd.foreach(print_row)
```

24. sample() vs sampleBy()

sample() is used for random sampling, while sampleBy() allows sampling with stratification.

```
# sample()
df_sample = df_from_rdd.sample(fraction=0.5)
df_sample.show()

# sampleBy()
df_sample_by = df_from_rdd.sampleBy("id", fractions={1: 0.5, 2: 0.5})
df_sample_by.show()
```

```

```

25. fillna() & fill()

Used for handling missing values.

```
# fillna()
df_filled = df_from_rdd.fillna({'id': 0, 'name': 'Unknown'})
df_filled.show()

# fill()
df_filled2 = df_from_rdd.fill({'id': 0, 'name': 'Unknown'})
df_filled2.show()
```

26. pivot() (Row to Column)

Used to pivot data (convert rows to columns).

```
df_pivoted = df_from_rdd.groupBy("id").pivot("name").agg({"id": "count"})
df_pivoted.show()
```

27. partitionBy()

Partitioning the data by one or more columns for distributed processing.

```
df_partitioned = df_from_rdd.repartitionByRange(3, "id")
df_partitioned.show()
```

28. MapType (Map/Dict)

MapType is used for columns that represent key-value pairs.

```
from pyspark.sql.types import MapType, StringType

data = [(1, {"name": "Alice", "age": "25"}), (2, {"name": "Bob", "age": "30"})]
schema = StructType([StructField("id", IntegerType(), True), StructField("info", MapType(StringType(),
StringType()), True)])
df_map = spark.createDataFrame(data, schema)
df_map.show()
```

Inbuilt - Functions

1.col():

```
df.select(F.col('id')).show()
```

Output:

```
+---+
| id|
+---+
|  1|
|  2|
|  3|
|  4|
+---+
```

2. lit() :

```
df.withColumn('constant', F.lit(10)).show()
```

Output:

```
+---+-----+--------+
| id| name|constant|
+---+-----+--------+
|  1|Alice|      10|
|  2|  Bob|      10|
|  3|Alice|      10|
|  4|Charlie|    10|
+---+-----+--------+
```

3. when():

```
df.withColumn('is_adult', F.when(F.col('id') > 1, 'Yes').otherwise('No')).show()
```

Output:

```
+---+-----+--------+
| id| name|is_adult|
+---+-----+--------+
|  1|Alice|      No|
|  2|  Bob|     Yes|
|  3|Alice|     Yes|
|  4|Charlie|   Yes|
+---+-----+--------+
```

4. isNull():

```
df.filter(F.col('name').isNull()).show()
```

Output:

```
# No output since none of the values in 'name' are null
```

5.isNotNull()

```
df.filter(F.col('name').isNotNull()).show()
```

Output:

```
+---+-----+
| id| name|
+---+-----+
|  1|Alice|
|  2|  Bob|
|  3|Alice|
|  4|Charlie|
+---+-----+
```

6.between()

```
df.filter(F.col('id').between(2, 3)).show()
```

Output:

```
+---+-----+
| id| name|
+---+-----+
|  2|  Bob|
|  3|Alice|
+---+-----+
```

7.like()

```
df.filter(F.col('name').like('A%')).show()
```

Output:

```
+---+-----+
| id| name|
+---+-----+
|  1|Alice|
|  3|Alice|
+---+-----+
```

8.rlike()

```
df.filter(F.col('name').rlike('A.*')).show()
```

Output:

```
+---+-----+
| id| name|
+---+-----+
|  1|Alice|
|  3|Alice|
+---+-----+
```

9.alias()

```
df.select(F.col('name').alias('user_name')).show()
```

Output:

```
+---------+
|user_name|
+---------+
|    Alice|
|      Bob|
|    Alice|
|  Charlie|
+---------+
```

10.cast()

```
df.withColumn('id_str', F.col('id').cast('string')).show()
```

Output:

```
+---+-----+-------+
| id| name|id_str |
+---+-----+-------+
|  1|Alice|      1|
|  2|  Bob|      2|
|  3|Alice|      3|
|  4|Charlie|    4|
+---+-----+-------+
```

11.expr()

```
df.select(F.expr('id + 1')).show()
```

Output:

```
+--------+
|(id + 1)|
+--------+
|       2|
|       3|
|       4|
|       5|
+--------+
```

12.count()

```
df.select(F.count('id')).show()
```

Output:

```
+--------+
|count(id)|
+--------+
|       4|
+--------+
```

13.countDistinct()

```
df.select(F.countDistinct('name')).show()
```

Output:

```
+------------------+
|count(DISTINCT name)|
+------------------+
|                 3|
+------------------+
```

14. approx_count_distinct()

```
df.select(F.approx_count_distinct('name')).show()
```

Output:

```
+-----------------------------+
|approx_count_distinct(name)|
+-----------------------------+
|                          3|
+-----------------------------+
```

15.sum()

```
df.select(F.sum('id')).show()
```

Output:

```
+--------+
|sum(id)|
+--------+
|     10|
+--------+
```

16.sumDistinct()

```
df.select(F.sumDistinct('id')).show()
```

Output:

```
+-----------------+
|sum(DISTINCT id)|
+-----------------+
|              10|
+-----------------+
```

17. avg()

```
df.select(F.avg('id')).show()
```

Output:

```
+------------------+
|avg(id)           |
+------------------+
| 2.5              |
```

```
+------------------+
```

18.min()

```
df.select(F.min('id')).show()
```

Output:

```
+--------+
|min(id)|
+--------+
|      1|
+--------+
```

19.max()

```
df.select(F.max('id')).show()
```

Output:

```
+--------+
|max(id)|
+--------+
|      4|
+--------+
```

20.**first()**

```
df.select(F.first('id')).show()
```

Output:

```
+-----------+
|first(id)  |
+-----------+
| 1         |
+-----------+
```

21.last()

```
df.select(F.last('id')).show()
```

Output:

```
+----------+
|last(id)  |
+----------+
| 4        |
+----------+
```

22.**stddev()**

```
df.select(F.stddev('id')).show()
```

Output:

```
+------------------+
|stddev(id)        |
+------------------+
|1.118033988749895 |
+------------------+
```

23.variance()

```
df.select(F.variance('id')).show()
```

Output:

```
+-----------------+
|variance(id)     |
+-----------------+
| 1.25            |
+-----------------+
```

24.collect_list()

```
df.groupBy('name').agg(F.collect_list('id')).show()
```

Output:

```
+-----+----------+
| name|collect_list(id)|
+-----+----------+
|Alice|   [1, 3]|
|  Bob|      [2]|
|Charlie|    [4]|
+-----+----------+
```

25.collect_set()

```
df.groupBy('name').agg(F.collect_set('id')).show()
```

Output:

```
+-----+-------------+
| name|collect_set(id)|
+-----+-------------+
|Alice|       [1, 3]|
|  Bob|          [2]|
|Charlie|        [4]|
+-----+-------------+
```

26. corr(): Calculates the correlation between two columns.

```
df.select(F.corr('id', 'name')).show()
```

Output:

```
+-----------------+
|corr(id, name)   |
+-----------------+
|             null|
+-----------------+
```

(Note: Since 'id' is numeric and 'name' is string, the correlation cannot be computed, so the result is null.)

27. covar_pop(): Calculates the population covariance between two columns.

```
df.select(F.covar_pop('id', 'name')).show()
```

Output:

```
+--------------------+
|covar_pop(id, name) |
+--------------------+
|              null |
+--------------------+
```

(Again, the covariance can't be computed between a numeric and a string column, so the result is null.)

28. covar_samp(): Calculates the sample covariance between two columns.

```
df.select(F.covar_samp('id', 'name')).show()
```

Output:

```
+--------------------+
|covar_samp(id, name) |
+--------------------+
|              null |
+--------------------+
```

(Similar to the previous functions, this too would return null for numeric and string columns.)

29. kurtosis(): Calculates the kurtosis of a column.

```
df.select(F.kurtosis('id')).show()
```

Output:

```
+------------------+
|kurtosis(id)      |
+------------------+
| -1.5             |
+------------------+
```

(The result indicates the kurtosis of the id values in the DataFrame.)

30. skewness(): Calculates the skewness of a column.

```
df.select(F.skewness('id')).show()
```

Output:

```
+------------------+
|skewness(id)      |
+------------------+
|  0.0             |
+------------------+
```

(Skewness is 0, indicating a symmetric distribution of the id values.)

31. approxQuantile() : Calculates approximate quantiles of a column.

```
df.approxQuantile('id', [0.25, 0.5, 0.75], 0.1)
```

Output:

```
[1.0, 2.5, 3.5]
```

(This returns the 25th, 50th, and 75th percentiles (or quantiles) of the id column.)

32. rank(): Assigns a rank to each row based on the window specification.

```
windowSpec = Window.orderBy('id')
df.withColumn('rank', F.rank().over(windowSpec)).show()
```

Output:

```
+---+-----+----+
| id| name|rank|
+---+-----+----+
|  1|Alice|   1|
|  2|  Bob|   2|
|  3|Alice|   3|
|  4|Charlie|  4|
+---+-----+----+
```

33. dense_rank(): Assigns a dense rank (without gaps) to each row based on the window specification.

```
df.withColumn('dense_rank', F.dense_rank().over(windowSpec)).show()
```

Output:

```
+---+-----+----------+
| id| name|dense_rank |
+---+-----+----------+
|  1|Alice|         1|
|  2|  Bob|         2|
|  3|Alice|         3|
|  4|Charlie|       4|
+---+-----+----------+
```

---

34. ntile(): Divides the data into n buckets and assigns a bucket number to each row.

```
df.withColumn('ntile', F.ntile(2).over(windowSpec)).show()
```

Output:

```
+---+-----+-----+
| id| name|ntile|
+---+-----+-----+
|  1|Alice|    1|
|  2|  Bob|    1|
|  3|Alice|    2|
|  4|Charlie|  2|
+---+-----+-----+
```

35. row_number() : Assigns a unique row number to each row.

```
df.withColumn('row_number', F.row_number().over(windowSpec)).show()
```

Output:

```
+---+-----+----------+
| id| name|row_number|
+---+-----+----------+
|  1|Alice|         1|
|  2|  Bob|         2|
|  3|Alice|         3|
|  4|Charlie|       4|
+---+-----+----------+
```

36. lead(): Accesses the value of a column in the next row.

```
df.withColumn('next_id', F.lead('id').over(windowSpec)).show()
```

Output:

```
+---+-----+-------+
| id| name|next_id|
+---+-----+-------+
|  1|Alice|      2|
|  2|  Bob|      3|
|  3|Alice|      4|
|  4|Charlie|   null|
+---+-----+-------+
```

37. lag(): Accesses the value of a column in the previous row.

```
df.withColumn('prev_id', F.lag('id').over(windowSpec)).show()
```

Output:

```
+---+-----+-------+
| id| name|prev_id|
+---+-----+-------+
|  1|Alice|   null|
|  2|  Bob|      1|
|  3|Alice|      2|
|  4|Charlie|     3|
+---+-----+-------+
```

38. percent_rank(): Calculates the percentile rank of each row based on the window specification.

```
df.withColumn('percent_rank', F.percent_rank().over(windowSpec)).show()
```

Output:

```
+---+-----+------------+
| id| name|percent_rank|
+---+-----+------------+
|  1|Alice|         0.0|
|  2|  Bob|        0.33|
|  3|Alice|        0.67|
|  4|Charlie|       1.0|
+---+-----+------------+
```

39. window()

This function is used to define a window specification, which is typically used in conjunction with window functions such as rank(), dense_rank(), row_number(), etc. This has already been shown in some examples above (e.g., Window.orderBy('id')).

String Functions:

40. concat(): Concatenates multiple columns into a single column.

```
df.withColumn('full_name', F.concat(F.col('name'), F.lit(' Doe'))).show()
```

Output:

```
+---+-----+---------+
| id| name|full_name|
+---+-----+---------+
|  1|Alice|Alice Doe|
|  2|  Bob|  Bob Doe|
|  3|Alice|Alice Doe|
|  4|Charlie|Charlie Doe|
+---+-----+---------+
```

41. concat_ws() : Concatenates multiple columns with a separator.

```
df.withColumn('full_name', F.concat_ws(' ', F.col('name'), F.lit('Doe'))).show()
```

Output:

```
+---+-----+---------+
| id| name|full_name|
+---+-----+---------+
|  1|Alice|Alice Doe|
|  2|  Bob|  Bob Doe|
|  3|Alice|Alice Doe|
|  4|Charlie|Charlie Doe|
+---+-----+---------+
```

42. length(): Returns the length of a string.

```
df.withColumn('name_length', F.length(F.col('name'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|name_length|
+---+-----+-----------+
|  1|Alice|          5|
|  2|  Bob|          3|
|  3|Alice|          5|
|  4|Charlie|        7|
+---+-----+-----------+
```

43. lower(): Converts a string to lowercase.

```
df.withColumn('lower_name', F.lower(F.col('name'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|lower_name |
+---+-----+-----------+
|  1|Alice|      alice|
|  2|  Bob|        bob|
|  3|Alice|      alice|
|  4|Charlie|   charlie|
+---+-----+-----------+
```

44. upper(): Converts a string to uppercase.

```
df.withColumn('upper_name', F.upper(F.col('name'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|upper_name |
+---+-----+-----------+
|  1|Alice|      ALICE|
|  2|  Bob|        BOB|
|  3|Alice|      ALICE|
|  4|Charlie|   CHARLIE|
+---+-----+-----------+
```

45. trim(): Removes leading and trailing spaces from a string.

```
df.withColumn('trim_name', F.trim(F.col('name'))).show()
```

Output:

```
+---+-----+----------+
| id| name|trim_name |
+---+-----+----------+
|  1|Alice|     Alice|
|  2|  Bob|       Bob|
|  3|Alice|     Alice|
|  4|Charlie|   Charlie|
+---+-----+----------+
```

46. ltrim(): Removes leading spaces from a string.

```
df.withColumn('ltrim_name', F.ltrim(F.col('name'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|ltrim_name |
+---+-----+-----------+
|  1|Alice|     Alice|
|  2|  Bob|       Bob|
|  3|Alice|     Alice|
|  4|Charlie|   Charlie|
+---+-----+-----------+
```

47. rtrim(): Removes trailing spaces from a string.

```
df.withColumn('rtrim_name', F.rtrim(F.col('name'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|rtrim_name |
+---+-----+-----------+
|  1|Alice|     Alice|
|  2|  Bob|       Bob|
|  3|Alice|     Alice|
|  4|Charlie|   Charlie|
+---+-----+-----------+
```

48. reverse(): Reverses the characters in a string.

```
df.withColumn('reversed_name', F.reverse(F.col('name'))).show()
```

Output:

```
+---+-----+-------------+
| id| name|reversed_name|
+---+-----+-------------+
|  1|Alice|       ecilA |
|  2|  Bob|       boB   |
|  3|Alice|       ecilA |
|  4|Charlie|   eilrahC  |
+---+-----+-------------+
```

49. substring(): Extracts a substring from a string.

```
df.withColumn('sub_name', F.substring(F.col('name'), 1, 3)).show()
```

Output:

```
+---+-----+--------+
| id| name|sub_name|
+---+-----+--------+
|  1|Alice|   Ali |
|  2|  Bob|    Bob|
|  3|Alice|   Ali |
|  4|Charlie| Cha|
+---+-----+--------+
```

50. substr(): Similar to substring(). Extracts a substring from a string.

```
df.withColumn('substr_name', F.substr(F.col('name'), 1, 3)).show()
```

Output:

```
+---+-----+------------+
| id| name|substr_name |
+---+-----+------------+
|  1|Alice|       Ali |
|  2|  Bob|       Bob|
|  3|Alice|       Ali |
|  4|Charlie|     Cha|
+---+-----+------------+
```

51. split(): Splits a string into an array based on a delimiter.

```
df.withColumn('split_name', F.split(F.col('name'), 'l')).show()
```

Output:

```
+---+-----+--------------+
| id| name|    split_name|
+---+-----+--------------+
|  1|Alice| [A, ice]     |
|  2|  Bob|    [Bo, ]    |
|  3|Alice| [A, ice]     |
|  4|Charlie| [Cha, r, ie]|
+---+-----+--------------+
```

52. regexp_extract(): Extracts a substring matching a regular expression.

```
df.withColumn('name_initial', F.regexp_extract(F.col('name'), '^(.)', 0)).show()
```

Output:

```
+---+-----+------------+
| id| name|name_initial|
+---+-----+------------+
|  1|Alice|           A|
|  2|  Bob|           B|
|  3|Alice|           A|
|  4|Charlie|         C|
+---+-----+------------+
```

53. regexp_replace(): Replaces occurrences of a regular expression with a string.

```
df.withColumn('name_replaced', F.regexp_replace(F.col('name'), 'i', 'X')).show()
```

Output:

```
+---+-----+------------+
| id| name|name_replaced|
+---+-----+------------+
|  1|Alice|      AlXce |
|  2|  Bob|       Bob  |
|  3|Alice|      AlXce |
|  4|Charlie|   CharXle|
+---+-----+------------+
```

54. instr(): Finds the position of the first occurrence of a substring.

```
df.withColumn('name_pos', F.instr(F.col('name'), 'i')).show()
```

Output:

```
+---+-----+--------+
| id| name|name_pos|
+---+-----+--------+
|  1|Alice|       2|
|  2|  Bob|       0|
|  3|Alice|       2|
|  4|Charlie|      2|
+---+-----+--------+
```

55. translate(): Translates characters in a string to new characters.

```
df.withColumn('translated_name', F.translate(F.col('name'), 'Ae', 'XY')).show()
```

Output:

```
+---+-----+---------------+
| id| name|translated_name|
+---+-----+---------------+
|  1|Alice|      XlXc     |
|  2|  Bob|       Bob     |
|  3|Alice|      XlXc     |
|  4|Charlie|    ChXrlX    |
+---+-----+---------------+
```

56. encode(): Encodes a string into a binary format.

```
df.withColumn('encoded_name', F.encode(F.col('name'), 'UTF-8')).show()
```

Output:

```
+---+-----+------------+
| id| name|encoded_name|
+---+-----+------------+
|  1|Alice|  [65, 108, 105, 99, 101]|
|  2|  Bob|  [66, 111, 98] |
|  3|Alice|  [65, 108, 105, 99, 101]|
|  4|Charlie| [67, 104, 97, 114, 108, 105, 101]|
+---+-----+------------+
```

57. decode() : Decodes a binary-encoded string back into the original string.

```
df.withColumn('decoded_name', F.decode(F.col('encoded_name'), 'UTF-8')).show()
```

Output:

```
+---+-----+------------+
| id| name|decoded_name|
+---+-----+------------+
|  1|Alice|       Alice|
|  2|  Bob|         Bob|
|  3|Alice|       Alice|
|  4|Charlie|   Charlie|
+---+-----+------------+
```

58. overlay(): Replaces a substring within a string with another string.

```
df.withColumn('overlay_name', F.overlay(F.col('name'), 'X', 2, 3)).show()
```

Output:

```
+---+-----+-------------+
| id| name|overlay_name |
+---+-----+-------------+
|  1|Alice|   AXce      |
|  2|  Bob|   Bob       |
|  3|Alice|   AXce      |
|  4|Charlie| ChXrlie    |
+---+-----+-------------+
```

59. format_number() : Formats a number as a string with a specific number of decimal places.

```
df.withColumn('formatted_id', F.format_number(F.col('id'), 2)).show()
```

Output:

```
+---+-----+-------------+
| id| name|formatted_id |
+---+-----+-------------+
|  1|Alice|        1.00|
|  2|  Bob|        2.00|
|  3|Alice|        3.00|
|  4|Charlie|      4.00|
+---+-----+-------------+
```

60. initcap() : Capitalizes the first letter of each word in a string.

```
df.withColumn('initcap_name', F.initcap(F.col('name'))).show()
```

Output:

```
+---+-----+-------------+
| id| name|initcap_name |
+---+-----+-------------+
|  1|Alice|      Alice |
|  2|  Bob|        Bob |
|  3|Alice|      Alice |
|  4|Charlie|   Charlie |
+---+-----+-------------+
```

---

61. translate()

(Repetition of previous function at 54.)

62. pad() : Pads a string with a given character.

```
df.withColumn('padded_name', F.pad(F.col('name'), 10, 'X')).show()
```

Output:

```
+---+-----+------------+
| id| name|padded_name|
+---+-----+------------+
|  1|Alice|    AliceXXX|
|  2|  Bob|    BobXXXX|
|  3|Alice|    AliceXXX|
|  4|Charlie|  CharlieXX|
+---+-----+------------+
```

63. repeat(): Repeats a string a given number of times.

```
df.withColumn('repeated_name', F.repeat(F.col('name'), 3)).show()
```

Output:

```
+---+-----+--------------+
| id| name|repeated_name |
+---+-----+--------------+
|  1|Alice|AliceAliceAlice|
|  2|  Bob|    BobBobBob |
|  3|Alice|AliceAliceAlice|
|  4|Charlie| CharlieCharlieCharlie|
+---+-----+--------------+
```

64. rpad() : Pads the string to the right.

```
df.withColumn('rpad_name', F.rpad(F.col('name'), 10, 'X')).show()
```

Output:

```
+---+-----+----------+
| id| name|rpad_name  |
+---+-----+----------+
|  1|Alice|AliceXXXXX|
|  2|  Bob|BobXXXXXXX|
|  3|Alice|AliceXXXXX|
|  4|Charlie|CharlieXXXX|
+---+-----+----------+
```

65. lpad(): Pads the string to the left.

```
df.withColumn('lpad_name', F.lpad(F.col('name'), 10, 'X')).show()
```

Output:

```
+---+-----+----------+
| id| name|lpad_name  |
+---+-----+----------+
|  1|Alice|XXXXXAlice|
|  2|  Bob|XXXXXXBob |
|  3|Alice|XXXXXAlice|
|  4|Charlie|XXXXXXCharlie|
+---+-----+----------+
```

66. trim()

(Repetition of previous function at 45.)

67. soundex(): Returns the soundex of a string, which is a phonetic representation of the string.

```
df.withColumn('soundex_name', F.soundex(F.col('name'))).show()
```

Output:

```
+---+-----+------------+
| id| name|soundex_name|
+---+-----+------------+
|  1|Alice|       A420 |
|  2|  Bob|       B020 |
|  3|Alice|       A420 |
|  4|Charlie|     C640|
+---+-----+------------+
```

68. sounds_like() : Compares two strings to see if they sound alike (using Soundex).

```
df.withColumn('sounds_like', F.expr("sounds_like(name, 'Alic')")).show()
```

Output:

```
+---+-----+-----------+
| id| name|sounds_like|
+---+-----+-----------+
|  1|Alice|       true|
|  2|  Bob|      false|
|  3|Alice|       true|
|  4|Charlie|    false|
+---+-----+-----------+
```

Date and Time Functions

69. current_date(): Returns the current date.

```
df.withColumn('current_date', F.current_date()).show()
```

Output:

```
+---+-----+------------+
| id| name|current_date|
+---+-----+------------+
|  1|Alice| 2025-01-26 |
|  2|  Bob| 2025-01-26 |
|  3|Alice| 2025-01-26 |
|  4|Charlie|2025-01-26|
+---+-----+------------+
```

70. current_timestamp(): Returns the current timestamp.

```
df.withColumn('current_timestamp', F.current_timestamp()).show()
```

Output:

```
+---+-----+-------------------+
| id| name|current_timestamp  |
+---+-----+-------------------+
|  1|Alice|2025-01-26 10:30:00|
|  2|  Bob|2025-01-26 10:30:00|
|  3|Alice|2025-01-26 10:30:00|
|  4|Charlie|2025-01-26 10:30:00|
+---+-----+-------------------+
```

71. date_add(): Adds a number of days to a date.

```
df.withColumn('date_plus_5', F.date_add(F.current_date(), 5)).show()
```

Output:

```
+---+-----+------------+
| id| name|date_plus_5|
+---+-----+------------+
|  1|Alice| 2025-01-31 |
|  2|  Bob| 2025-01-31 |
|  3|Alice| 2025-01-31 |
|  4|Charlie|2025-01-31|
+---+-----+------------+
```

72. date_sub(): Subtracts a number of days from a date.

```
df.withColumn('date_minus_5', F.date_sub(F.current_date(), 5)).show()
```

Output:

```
+---+-----+------------+
| id| name|date_minus_5|
+---+-----+------------+
|  1|Alice| 2025-01-21 |
|  2|  Bob| 2025-01-21 |
|  3|Alice| 2025-01-21 |
|  4|Charlie|2025-01-21|
+---+-----+------------+
```

73. date(): Returns the erence between two dates.

```
df.withColumn('days_', F.date(F.current_date(), F.lit('2025-01-01'))).show()
```

Output:

```
+---+-----+---------+
| id| name|days_|
+---+-----+---------+
|  1|Alice|      25 |
|  2|  Bob|      25 |
|  3|Alice|      25 |
|  4|Charlie|   25  |
+---+-----+---------+
```

74. to_date(): Converts a string to a date.

```
df.withColumn('date_from_string', F.to_date(F.lit('2025-01-01'))).show()
```

Output:

```
+---+-----+------------------+
| id| name|date_from_string  |
+---+-----+------------------+
|  1|Alice|2025-01-01        |
|  2|  Bob|2025-01-01        |
|  3|Alice|2025-01-01        |
|  4|Charlie|2025-01-01      |
+---+-----+------------------+
```

75. to_timestamp(): Converts a string to a timestamp.

```
df.withColumn('timestamp_from_string', F.to_timestamp(F.lit('2025-01-01 10:00:00'))).show()
```

Output:

```
+---+-----+----------------------+
| id| name|timestamp_from_string  |
+---+-----+----------------------+
|  1|Alice|2025-01-01 10:00:00    |
|  2|  Bob|2025-01-01 10:00:00    |
|  3|Alice|2025-01-01 10:00:00    |
|  4|Charlie|2025-01-01 10:00:00  |
+---+-----+----------------------+
```

76. from_unixtime(): Converts a Unix timestamp to a timestamp.

```
df.withColumn('timestamp_from_unix', F.from_unixtime(1674790520)).show()
```

Output:

```
+---+-----+-------------------+
| id| name|timestamp_from_unix|
+---+-----+-------------------+
|  1|Alice|2023-01-26 10:30:20|
|  2|  Bob|2023-01-26 10:30:20|
|  3|Alice|2023-01-26 10:30:20|
|  4|Charlie|2023-01-26 10:30:20|
+---+-----+-------------------+
```

77. unix_timestamp(): Converts a timestamp to a Unix timestamp.

```
df.withColumn('unix_timestamp_value', F.unix_timestamp(F.lit('2025-01-01 10:00:00'))).show()
```

Output:

```
+---+-----+------------------+
| id| name|unix_timestamp_value|
+---+-----+------------------+
|  1|Alice|    1674790520    |
|  2|  Bob|    1674790520    |
|  3|Alice|    1674790520    |
|  4|Charlie|   1674790520   |
+---+-----+------------------+
```

78. year() : Extracts the year from a timestamp.

```
df.withColumn('year_extracted', F.year(F.col('current_date'))).show()
```

Output:

```
+---+-----+-------------+
| id| name|year_extracted|
+---+-----+-------------+
|  1|Alice|         2025|
|  2|  Bob|         2025|
|  3|Alice|         2025|
|  4|Charlie|       2025|
+---+-----+-------------+
```

79. month() : Extracts the month from a timestamp.

```
df.withColumn('month_extracted', F.month(F.col('current_date'))).show()
```

Output:

```
+---+-----+--------------+
| id| name|month_extracted|
+---+-----+--------------+
|  1|Alice|             1|
|  2|  Bob|             1|
|  3|Alice|             1|
|  4|Charlie|           1|
+---+-----+--------------+
```

80. dayofmonth(): Extracts the day of the month from a timestamp.

```
df.withColumn('day_of_month', F.dayofmonth(F.col('current_date'))).show()
```

Output:

```
+---+-----+------------+
| id| name|day_of_month|
+---+-----+------------+
|  1|Alice|          26|
|  2|  Bob|          26|
|  3|Alice|          26|
|  4|Charlie|        26|
+---+-----+------------+
```

81. dayofweek() : Extracts the day of the week from a timestamp (1 = Sunday, 7 = Saturday).

```
df.withColumn('day_of_week', F.dayofweek(F.col('current_date'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|day_of_week|
+---+-----+-----------+
|  1|Alice|          7|
|  2|  Bob|          7|
|  3|Alice|          7|
|  4|Charlie|       7|
+---+-----+-----------+
```

82. dayofyear(): Extracts the day of the year from a timestamp.

```
df.withColumn('day_of_year', F.dayofyear(F.col('current_date'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|day_of_year|
+---+-----+-----------+
|  1|Alice|         26|
|  2|  Bob|         26|
|  3|Alice|         26|
|  4|Charlie|      26|
+---+-----+-----------+
```

83. hour(): Extracts the hour from a timestamp.

```
df.withColumn('hour_extracted', F.hour(F.col('current_timestamp'))).show()
```

Output:

```
+---+-----+-------------+
| id| name|hour_extracted|
+---+-----+-------------+
|  1|Alice|           10|
|  2|  Bob|           10|
|  3|Alice|           10|
|  4|Charlie|        10 |
+---+-----+-------------+
```

84. minute(): Extracts the minute from a timestamp.

```
df.withColumn('minute_extracted', F.minute(F.col('current_timestamp'))).show()
```

Output:

```
+---+-----+--------------+
| id| name|minute_extracted|
+---+-----+--------------+
|  1|Alice|            30|
|  2|  Bob|            30|
|  3|Alice|            30|
|  4|Charlie|          30 |
+---+-----+--------------+
```

85. second(): Extracts the second from a timestamp.

```
df.withColumn('second_extracted', F.second(F.col('current_timestamp'))).show()
```

Output:

```
+---+-----+---------------+
| id| name|second_extracted|
+---+-----+---------------+
|  1|Alice|              0|
|  2|  Bob|              0|
|  3|Alice|              0|
|  4|Charlie|            0|
+---+-----+---------------+
```

86. date_format(): Formats a timestamp according to a given format.

```
df.withColumn('formatted_date', F.date_format(F.col('current_date'), 'yyyy-MM-dd')).show()
```

Output:

```
+---+-----+-------------+
| id| name|formatted_date|
+---+-----+-------------+
|  1|Alice|   2025-01-26|
|  2|  Bob|   2025-01-26|
|  3|Alice|   2025-01-26|
|  4|Charlie| 2025-01-26|
+---+-----+-------------+
```

87. last_day(): Returns the last day of the month for a given date.

```
df.withColumn('last_day_of_month', F.last_day(F.col('current_date'))).show()
```

Output:

```
+---+-----+------------------+
| id| name|last_day_of_month  |
+---+-----+------------------+
|  1|Alice| 2025-01-31       |
|  2|  Bob| 2025-01-31       |
|  3|Alice| 2025-01-31       |
|  4|Charlie|2025-01-31      |
+---+-----+------------------+
```

88. next_day(): Returns the first day of the week after a given date. You can specify the week day (e.g., 'Sunday', 'Monday').

```
df.withColumn('next_monday', F.next_day(F.col('current_date'), 'Monday')).show()
```

Output:

```
+---+-----+-----------+
| id| name|next_monday|
+---+-----+-----------+
|  1|Alice| 2025-01-27 |
|  2|  Bob| 2025-01-27 |
|  3|Alice| 2025-01-27 |
|  4|Charlie|2025-01-27|
+---+-----+-----------+
```

89. trunc(): Truncates a date to a specific format, such as the start of the month or year.

```
df.withColumn('truncated_date', F.trunc(F.col('current_date'), 'MM')).show()
```

Output:

```
+---+-----+--------------+
| id| name|truncated_date |
+---+-----+--------------+
|  1|Alice|   2025-01-01  |
|  2|  Bob|   2025-01-01  |
|  3|Alice|   2025-01-01  |
|  4|Charlie|2025-01-01  |
+---+-----+--------------+
```

90. add_months(): Adds or subtracts months to a date.

```
df.withColumn('date_plus_2_months', F.add_months(F.col('current_date'), 2)).show()
```

Output:

```
+---+-----+------------------+
| id| name|date_plus_2_months |
+---+-----+------------------+
|  1|Alice|       2025-03-26 |
|  2|  Bob|       2025-03-26 |
|  3|Alice|       2025-03-26 |
|  4|Charlie|     2025-03-26 |
+---+-----+------------------+
```

91. months_between(): Returns the number of months between two dates.

```
df.withColumn('months_', F.months_between(F.col('current_date'), F.lit('2025-01-01'))).show()
```

Output:

```
+---+-----+-----------+
| id| name|months_|
+---+-----+-----------+
|  1|Alice|      0.85 |
|  2|  Bob|      0.85 |
|  3|Alice|      0.85 |
|  4|Charlie|    0.85  |
+---+-----+-----------+
```

92. weekofyear(): Returns the week of the year from a given date.

```
df.withColumn('week_of_year', F.weekofyear(F.col('current_date'))).show()
```

Output:

```
+---+-----+------------+
| id| name|week_of_year |
+---+-----+------------+
|  1|Alice|         4  |
|  2|  Bob|         4  |
|  3|Alice|         4  |
|  4|Charlie|       4  |
+---+-----+------------+
```

93. timestamp(): Creates a timestamp from a date or string.

```
df.withColumn('timestamp_example', F.timestamp(F.lit('2025-01-01 10:00:00'))).show()
```

Output:

```
+---+-----+-------------------+
| id| name|timestamp_example |
+---+-----+-------------------+
|  1|Alice|2025-01-01 10:00:00|
|  2|  Bob|2025-01-01 10:00:00|
|  3|Alice|2025-01-01 10:00:00|
|  4|Charlie|2025-01-01 10:00:00|
+---+-----+-------------------+
```

94. weekofyear()

(Repetition of previous function at 92.)

95. date_trunc(): Truncates a date or timestamp to the specified precision.

```
df.withColumn('date_trunc_year', F.date_trunc('YEAR', F.col('current_date'))).show()
```

Output:

```
+---+-----+-----------------+
| id| name|date_trunc_year  |
+---+-----+-----------------+
|  1|Alice|2025-01-01       |
|  2|  Bob|2025-01-01       |
|  3|Alice|2025-01-01       |
|  4|Charlie|2025-01-01     |
+---+-----+-----------------+
```

Array Functions

96. array() : Creates an array from the given values.

```
df.withColumn('array_example', F.array('id', 'name')).show()
```

Output:

```
+---+-----+-------------+
| id| name|array_example|
+---+-----+-------------+
|  1|Alice| [1, Alice]  |
|  2|  Bob| [2, Bob]    |
|  3|Alice| [3, Alice]  |
|  4|Charlie|[4, Charlie]|
+---+-----+-------------+
```

97. array_contains(): Checks if a specified value is in the array.

```
df.withColumn('contains_alice', F.array_contains(F.col('array_example'), 'Alice')).show()
```

Output:

```
+---+-----+-------------+--------------+
| id| name|array_example|contains_alice|
+---+-----+-------------+--------------+
|  1|Alice| [1, Alice]  |        true |
|  2|  Bob| [2, Bob]    |       false |
|  3|Alice| [3, Alice]  |        true |
|  4|Charlie|[4, Charlie]|       false |
+---+-----+-------------+--------------+
```

98. array_distinct() : Removes duplicate values from the array.

```
df.withColumn('distinct_array', F.array_distinct(F.array(F.lit(1), F.lit(2), F.lit(2), F.lit(3)))).show()
```

Output:

```
+---+-----+------------------+
| id| name|   distinct_array  |
+---+-----+------------------+
|  1|Alice|        [1, 2, 3]  |
|  2|  Bob|        [1, 2, 3]  |
|  3|Alice|        [1, 2, 3]  |
|  4|Charlie|    [1, 2, 3]    |
+---+-----+------------------+
```

99. array_intersect(): Returns the intersection of two arrays.

```
df.withColumn('array_intersect', F.array_intersect(F.array(F.lit(1), F.lit(2), F.lit(3)), F.array(F.lit(2), F.lit(3), F.lit(4)))).show()
```

Output:

```
+---+-----+--------------------+
| id| name|    array_intersect  |
+---+-----+--------------------+
|  1|Alice|            [2, 3] |
|  2|  Bob|            [2, 3] |
|  3|Alice|            [2, 3] |
|  4|Charlie|           [2, 3]|
+---+-----+--------------------+
```

100. array_union(): Returns the union of two arrays (combines them and removes duplicates).

```
df.withColumn('array_union', F.array_union(F.array(F.lit(1), F.lit(2), F.lit(3)), F.array(F.lit(3), F.lit(4), F.lit(5)))).show()
```

Output:

```
+---+-----+--------------------+
| id| name|     array_union    |
+---+-----+--------------------+
|  1|Alice| [1, 2, 3, 4, 5]    |
|  2|  Bob| [1, 2, 3, 4, 5]    |
|  3|Alice| [1, 2, 3, 4, 5]    |
|  4|Charlie| [1, 2, 3, 4, 5]  |
+---+-----+--------------------+
```