PySpark is a Python API for Apache Spark that lets you process big data efficiently on clusters. This cheatsheet covers key operations from creating DataFrames to advanced functions, and we'll go through each major section step by step with simple explanations, line-by-line breakdowns, multiple examples, and real use cases.[1]

Creating DataFrames

DataFrames are like tables in Spark—think Excel sheets but for huge datasets. Start by creating a SparkSession, the entry point to Spark.

Default Schema Example 1 (Simple lists):

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [[1, "Alice", 29], [2, "Bob", 35]]
df = spark.createDataFrame(data, ["id", "name", "age"])
df.show()
```

- `from pyspark.sql import SparkSession`: Imports the SparkSession class.
- `spark = SparkSession.builder.appName("Example").getOrCreate()`: Creates or gets a Spark session named "Example". Use this once per program.
- `data = [[1, "Alice", 29], [2, "Bob", 35]]`: List of lists (rows).
- `df = spark.createDataFrame(data, ["id", "name", "age"])`: Makes DataFrame; Spark guesses types (all strings here).
- `df.show()`: Prints first 20 rows. Output: table with id, name, age.[1]

Use when you have small in-memory data like test cases. Changes: Add more rows for larger employee lists.

Explicit Schema Example 2 (Define types):

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True)
])
df = spark.createDataFrame(data, schema)
```

```
df.printSchema()
df.show()
```

- `from pyspark.sql.types import ...`: Imports type definitions.
- `schema = StructType([...])`: Defines columns with names, types (IntegerType for numbers), nullable=True (allows nulls).
- `df.printSchema()`: Shows structure like `root |-- id: integer (nullable = true)`.
  Use for precise control, like ensuring age is integer not string. Case: Financial data where salary must be float.[1]

Dict List Example 3:

```
data = [{"id": 1, "name": "Alice", "age": 29}, {"id": 2, "name": "Bob", "age": 35}]
df = spark.createDataFrame(data)
df.show()
```

Simpler for JSON-like data. Use when reading APIs. Change: Add {"dept": "HR"} for departments.[1]

Reading Files

Load real data from files—CSV for tables, JSON for nested data.

CSV Example 1 (Basic):

```
df = spark.read.format("csv").load("path/to/sample.csv")
```

- Loads CSV without headers; all as strings. Use for simple exports. Case: Log files without headers.[1]

CSV with Header/Infer Example 2:

```
df = spark.read.option("header", True).csv("path/to/sample.csv")
df = spark.read.option("inferSchema", True).option("delimiter",
",").csv("path/to/sample.csv")
```

- `header, True`: First row as column names.
- `inferSchema, True`: Guesses types (int, string).
  Use for standard CSVs like sales data. Multiple options chain for pipe-delimited (|).[1]

JSON Example 3:

```
df = spark.read.format("json").load("path/to/sample.json")
df = spark.read.option("multiline", True).json("path/to/sample.json")
```

- Basic for single-line JSONs.
- `multiline, True`: For pretty-printed JSON files.
  Use for API responses. Schema example: Define like CSV for nested JSON.[1]

Column Operations

Manipulate columns after creating/reading DataFrames.

Select Example 1:

```
df.select("name").show()  # Single column
df.select("name", "age").show()  # Multiple
columns_to_select = ["name", "department"]
df.select(columns_to_select).show()
```

Picks specific columns. Use to reduce data for analysis, like just names for reports.[1]

Rename Example 2:

```
df = df.withColumnRenamed("name", "fullname")
df = df.withColumnRenamed("oldcol1", "newcol1").withColumnRenamed("oldcol2", "newcol2")
```

- Renames one/multiple columns.
  Use when cleaning messy data, e.g., "emp_name" to "employee_name".[1]

Add Columns Example 3:

```
from pyspark.sql.functions import col, lit, expr, when
df = df.withColumn("country", lit("USA"))  # Constant
df = df.withColumn("salary_after_bonus", col("salary") * 1.1)  # Math
df = df.withColumn("high_earner", when(col("salary") > 55000, "Yes").otherwise("No"))
```

- `lit("USA")`: Adds constant.
- `col("salary") * 1.1`: References column for calc.

- `when().otherwise()`: If-else logic.
  Use for derived fields like bonuses in payroll. Multiple conditions: Chain `.when(col("salary") > 60000, "Low").when(... , "Medium").otherwise("High")`.[1]

Drop Columns:

```
df = df.drop("temp_col")
```

Removes unnecessary columns to save memory.[1]

Filtering Data

Narrow down rows based on conditions.

Basic Filter Example 1:

```
df.filter(col("age") > 30).show()
```

Keeps rows where age > 30. Use for subset queries like adults only.[1]

Multiple/String/Null Example 2:

```
df.filter((col("age") > 25) & (col("dept") == "IT")).show()  # AND with &
df.filter(col("name").contains("Ali")).show()  # String contains
df.filter(col("age").isNull()).show()  # Nulls
```

- `&` for AND, `|` for OR (use parens!).
- `contains()` for partial matches.
  Use for cleaning: Remove null ages, find "IT" employees.[1]

Filter from List Example 3:

```
from pyspark.sql.functions import col
depts = ["IT", "HR"]
df.filter(col("dept").isin(depts)).show()
```

Filters multiple values. Case: Specific departments.[1]

Grouping and Aggregations

Summarize data by groups.

Basic Group Example 1:

```
from pyspark.sql.functions import count, sum, avg
df.groupBy("dept").sum("salary").show()
```

Groups by dept, sums salaries. Use for dept totals.[1]

Multiple Aggs Example 2:

```
df.groupBy("dept").agg(
    count("name").alias("EmployeeCount"),
    avg("salary").alias("AverageSalary"),
    max("salary").alias("MaxSalary")
).show()
```

Multiple stats per group. Filter after: `.filter(col("TotalSalary") > 8000)`. Case: HR reports.[1]

Common functions: `min()`, `countDistinct()`, `collect_list()` (all values as list).[1]

Joins

Combine DataFrames like SQL.

Join Type	Syntax	When to Use	
Inner	`df1.join(df2, "id", "inner")`	Matching rows only (e.g., sales with customers).	
Left	`df1.join(df2, "id", "left")`	All from left, nulls from right (keep all employees).	
Right	`df1.join(df2, "id", "right")`	All from right.	
Full Outer	`df1.join(df2, "id", "outer")`	All rows, nulls where no match.	
Left Anti	`df1.join(df2, "id", "left_anti")`	Left rows without right match (inactive users).	[1]

Example:

```
df1.join(df2, on="id", how="left").select("df1.*", "df2.state").show()
```

Multiple cols: `df1.join(df2, (df1.col1 == df2.col2) & (df1.region == df2.region), "inner")`.
Broadcast small tables: `df1.join(broadcast(df2), "id")`.[1]

Key Functions

Date/Time Example (String to Date):

```
from pyspark.sql.functions import to_date
df.withColumn("date_parsed", to_date("datestr", "yyyy-MM-dd")).show()
```

Converts "2025-01-25" to date type. Formats: "dd-MMM-yyyy". Use for reports.[1]

String Functions:

- `trim(" col ")`: Removes spaces.
- `upper()`, `lower()`, `concat()`.[1]

Window Functions:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import sum
window_spec = Window.partitionBy("dept").orderBy("salary").rowsBetween(-2, 0)
df.withColumn("rolling_sum", sum("salary").over(window_spec)).show()
```

Running totals per group. Use for trends.[1]

Array Functions:

```
from pyspark.sql.functions import array, explode
df.withColumn("fruits_arr", array("fruit1", "fruit2")).withColumn("fruit",
explode("fruits_arr")).show()
```

Handles lists. Explode expands arrays to rows.

Writing Files

Save DataFrames back to files. Always specify mode like "overwrite" to avoid errors.

CSV Write Example 1 (Basic):

```
df.write.csv("path/to/output.csv")  # No header
df.write.option("header", True).csv("path/to/output.csv")  # With header
```

- Writes to folder of part files (Spark splits big data).
  Use for exporting clean data. Change: Add `.option("delimiter", "|")` for pipe format.[2]

Advanced Modes Example 2:

```
df.write.mode("overwrite").option("header", True).csv("path/to/output.csv")
df.write.mode("append").option("header", True).csv("path/to/output.csv")
```

- `overwrite`: Replace folder.
- `append`: Add to existing.
  Use append for logs, overwrite for daily reports.[2]

Delta/Partition Example 3:

```
df.write.format("delta").option("overwriteSchema",
true).mode("overwrite").save("path/to/output_delta")
df.write.format("delta").partitionBy("name").save("path/to/output_delta")
```

Delta for ACID tables. Partition speeds queries by column.[2]

Data Cleansing

Clean dirty data—duplicates, nulls.

Duplicates/Nulls Example:

```
df.distinct()  # Remove full row duplicates
df.dropDuplicates(["name", "age"])  # Specific columns
df.dropna()  # Drop any null row
df.dropna(subset=["email", "age"])  # Specific columns
df.fillna("NA")  # Fill all nulls
df.fillna({"age": 0, "country": "Unknown"})  # Column-specific
```

Use `distinct` for unique records, `fillna` for reports. Case: Clean customer data before analysis.[2]

Date/Time Functions

Convert/parse dates. Import `todate`, `totimestamp`.

String to Date Example 1:

```
from pyspark.sql.functions import to_date
df.withColumn("date_parsed", to_date("datestr", "yyyy-MM-dd")).show()  # "2025-01-25" → date
df.withColumn("date_parsed2", to_date("datestr", "dd-MMM-yyyy")).show()  # "25-Jan-2025"
```

Formats match input string. Use for sales dates.[2]

Timestamp Example 2:

```
from pyspark.sql.functions import to_timestamp
df.withColumn("ts_parsed", to_timestamp("timestampstr", "yyyy-MM-dd HH:mm:ss")).show()
```

Handles time too. Case: Log timestamps.[2]

Date Operations Example 3:

```
from pyspark.sql.functions import current_date, date_add, date_sub, datediff, year, month
df.withColumn("today", current_date()).withColumn("days_diff", datediff(current_date(), "date")).show()
df.withColumn("next_day", date_add("date", 7)).show()  # +7 days
```

`datediff`: Days between dates. Use for aging reports.[2]

String Functions

Manipulate text.

Basic Example 1:

```
from pyspark.sql.functions import upper, lower, concat, split, trim
df.withColumn("name_upper", upper("name")).withColumn("full_name", concat("first", lit(" "), "last")).show()
df.withColumn("parts", split("fullname", " ")).withColumn("last", split("fullname", " ")[^2_1]).show()
```

`split`: Breaks "John Doe" → array. `trim`: Removes spaces.[2]

Trim/Pad Example 2:

```
df.withColumn("cleaned", trim("name")).withColumn("padded", lpad("name", 10,
"0")).show()
```

`lpad`: "Alice" → "00000Alice". Use for fixed-width files.[2]

Math Functions

Simple calculations.

```
from pyspark.sql.functions import round, abs, sqrt
df.withColumn("salary_rounded", round("salary", 0)).withColumn("bonus", col("salary") *
0.1).show()
```

Arithmetic: `col("a") + col("b")`. Use for financial calcs.[2]

Window Functions

Advanced: Calculations over groups/windows.

Basic Ranking Example 1:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, rank, lag, sum
window_spec = Window.partitionBy("dept").orderBy(col("salary").desc())
df.withColumn("rank", rank().over(window_spec)).withColumn("prev_salary",
lag("salary").over(window_spec)).show()
```

`partitionBy`: Groups like GROUP BY. `orderBy`: Sorts within group. `rank`: 1,1,3 (gaps); `row_number`: 1,2,3.[2]

Rows Between Example 2 (Rolling):

```
window_spec2 = Window.partitionBy("dept").orderBy("salary").rowsBetween(-2, 0)  # Last 2
+ current
df.withColumn("rolling_sum", sum("salary").over(window_spec2)).show()
```

`rowsBetween(Window.unboundedPreceding, 0)`: Running total from start.[2]

Use for trends like 3-day sales average.

Array Functions

Handle lists/arrays.

Create/Manipulate Example 1:

```
from pyspark.sql.functions import array, explode, array_contains, size
df.withColumn("fruits_arr", array(lit("apple"), lit("banana"))).withColumn("has_apple",
array_contains("fruits_arr", "apple")).show()
df.withColumn("fruit", explode("fruits_arr")).show()  # Expands to rows
```

`explode`: 1 row → many rows. `size`: Array length.[2]

Elements/Modify Example 2:

```
df.withColumn("second_item", element_at("letters", 2)).withColumn("unique",
array_distinct("numbers")).show()
```

`element_at(arr, 1)`: Index 1 (0-based).[2]

SQL Queries

Use SQL on DataFrames.

Temp View Example:

```
df.createOrReplaceTempView("employees")
spark.sql("SELECT * FROM employees WHERE age > 30").show()
```

Register DF as table. Use for complex queries.[2]

Direct SQL Example:

```
spark.sql("SELECT name, avg(salary) FROM employees GROUP BY name").show()
```