AI Basics

PySpark Cheatsheet

```
## Big Data Processing
from pyspark import *
```

#Python

/sumitkhanna

Python

Follow me on
Sumit Khanna for more updates

Comprehensive Guide on PySpark Framework

PySpark is the Python API for Apache Spark, an open-source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. PySpark allows Python developers to harness the simplicity of Python and the power of Apache Spark to process large datasets efficiently.

Cheat Sheet Table

Method Name	Definition
`spark.read`	Returns a DataFrameReader that provides methods for reading data from various data sources.
`df.show`	Displays the content of the DataFrame in a tabular format.
`df.select`	Selects a set of columns from a DataFrame.
`df.filter`	Filters rows that meet a specified condition.
`df.groupBy`	Groups the DataFrame using the specified columns.
`df.agg`	Applies aggregate functions to the DataFrame.
`df.join`	Joins two DataFrames based on a common column(s).
`df.withColumn`	Adds a new column or replaces an existing column in the DataFrame.
`df.drop`	Drops specified columns from the DataFrame.
`df.distinct`	Returns a new DataFrame with distinct rows.
`df.union`	Returns a new DataFrame containing union of rows from two DataFrames.
`df.intersect`	Returns a new DataFrame containing intersection of rows from two DataFrames.

Method Name	Definition
`df.except`	Returns a new DataFrame containing rows that are in one DataFrame but not the other.
`df.sort`	Sorts the DataFrame based on specified columns.
`df.orderBy`	Orders the DataFrame based on specified columns.
`df.cache`	Caches the DataFrame in memory for quicker access.
`df.persist`	Persists the DataFrame with specified storage level.
`df.repartition`	Repartitions the DataFrame into specified number of partitions.
`df.coalesce`	Reduces the number of partitions in the DataFrame.
`df.na.drop`	Drops rows containing null values.
`df.na.fill`	Fills null values with specified value.
`df.na.replace`	Replaces specified values in the DataFrame.
`df.describe`	Computes basic statistics for numeric columns.
`df.summary`	Provides a summary of the DataFrame.
`df.crosstab`	Computes a cross-tabulation of two columns.
`df.stat.corr`	Computes correlation between two columns.
`df.stat.cov`	Computes covariance between two columns.
`df.stat.freqItems`	Finds frequent items for columns.
`df.stat.approxQuantile`	Calculates approximate quantiles.
`df.stat.sampleBy`	Returns a stratified sample without replacement based on the fraction given on each stratum.
`df.withColumnRenamed`	Renames an existing column in the DataFrame.
`df.dropDuplicates`	Drops duplicate rows based on specified columns.
`df.sample`	Returns a sampled subset of the DataFrame.
`df.randomSplit`	Randomly splits the DataFrame into multiple DataFrames.
`df.take`	Returns the first `n` rows of the DataFrame.
`df.head`	Returns the first row or `n` rows of the DataFrame.
`df.first`	Returns the first row of the DataFrame.
`df.collect`	Returns all rows as a list of Row objects.

Method Name	Definition
`df.toPandas`	Converts the DataFrame to a Pandas DataFrame.
`df.write`	Interface for saving the content of the DataFrame out into external storage systems.
`df.schema`	Returns the schema of the DataFrame.
`df.dtypes`	Returns a list of tuples containing the column name and data type.
`df.columns`	Returns a list of column names.
`df.count`	Returns the number of rows in the DataFrame.
`df.isEmpty`	Returns whether the DataFrame is empty.
`df.sample`	Returns a random sample of rows from the DataFrame.
`df.fillna`	Fills null values with specified value.
`df.replace`	Replaces values with specified values.
`df.dropna`	Drops rows with null values.
`df.cube`	Computes aggregates with specified columns and aggregators, using cube computation.
`df.rollup`	Computes aggregates with specified columns and aggregators, using rollup computation.

Explanation and Usage of Each Method

`spark.read`

Definition:

Returns a DataFrameReader that provides methods for reading data from various data sources.

Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("PySpark Guide").getOrCreate()

# Reading a CSV file
df = spark.read.csv("path/to/csvfile.csv", header=True, inferSchema=True)
df.show()
```

`df.show`

Definition:

Displays the content of the DataFrame in a tabular format.

Example:

```
# Show the first 20 rows of the DataFrame
df.show()

# Show the first 10 rows
df.show(10)
```

`df.select`

Definition:

Selects a set of columns from a DataFrame.

Example:

```
# Select specific columns
selected_df = df.select("column1", "column2")
selected_df.show()
```

`df.filter`

Definition:

Filters rows that meet a specified condition.

Example:

```
# Filter rows where the value in column1 is greater than 100
filtered_df = df.filter(df.column1 > 100)
filtered_df.show()
```

`df.groupBy`

Definition:

Groups the DataFrame using the specified columns.

Example:

```
# Group by column1 and calculate the count
grouped_df = df.groupBy("column1").count()
grouped_df.show()
```

`df.agg`

Definition:

Applies aggregate functions to the DataFrame.

Example:

```
from pyspark.sql.functions import avg

# Calculate the average value of column1
agg_df = df.agg(avg("column1"))
agg_df.show()
```

`df.join`

Definition:

Joins two DataFrames based on a common column(s).

Example:

```
# Join df1 and df2 on column1
joined_df = df1.join(df2, df1.column1 == df2.column1)
joined_df.show()
```

`df.withColumn`

Definition:

Adds a new column or replaces an existing column in the DataFrame.

Example:

```
from pyspark.sql.functions import lit

# Add a new column with a constant value
new_df = df.withColumn("new_column", lit("constant_value"))
new_df.show()
```

`df.drop`

Definition:

Drops specified columns from the DataFrame.

Example:

```
# Drop column1
dropped_df = df.drop("column1")
dropped_df.show()
```

`df.distinct`

Definition:

Returns a new DataFrame with distinct rows.

Example:

```
# Get distinct rows
distinct_df = df.distinct()
distinct_df.show()
```

`df.union`

Definition:

Returns a new DataFrame containing union of rows from two DataFrames.

Example:

```
# Union of df1 and df2
union_df = df1.union(df2)
union_df.show()
```

`df.intersect`

Definition:

Returns a new DataFrame containing intersection of rows from two DataFrames.

Example:

```
# Intersection of df1 and df2
intersect_df = df1.intersect(df2)
intersect_df.show()
```

`df.except`

Definition:

Returns a new DataFrame containing rows that are in one DataFrame but not the other.

Example:

```
# Except of df1 and df2
except_df = df1.except(df2)
except_df.show()
```

`df.sort`

Definition:

Sorts the DataFrame based on specified columns.

Example:

```
# Sort by column1
sorted_df = df.sort("column1")
sorted_df.show()
```

`df.orderBy`

Definition:

Orders the DataFrame based on specified columns.

Example:

```
# Order by column1
ordered_df = df.orderBy("column1")
ordered_df.show()
```

`df.cache`

Definition:

Caches the DataFrame in memory for quicker access.

Example:

```python
# Cache the DataFrame
df.cache()
df.show()
```

`df.persist`

Definition:

Persists the DataFrame with specified storage level.

Example:

```python
from pyspark import StorageLevel



# Persist the DataFrame
df.persist(StorageLevel.MEMORY_ONLY)
df.show()
```

`df.repartition`

Definition:

Repartitions the DataFrame into specified number of partitions.

Example:

```python
# Repartition to 10 partitions
repartitioned_df = df.repartition(10)
repartitioned_df.show()
```

`df.coalesce`

Definition:

Reduces the number of partitions in the DataFrame.

Example:

```
# Coalesce to 2 partitions
coalesced_df = df.coalesce(2)
coalesced_df.show()
```

`df.na.drop`

Definition:

Drops rows containing null values.

Example:

```
# Drop rows with any null values
df_na_dropped = df.na.drop()
df_na_dropped.show()
```

`df.na.fill`

Definition:

Fills null values with specified value.

Example:

```
# Fill null values with 0
df_na_filled = df.na.fill(0)
df_na_filled.show()
```

`df.na.replace`

Definition:

Replaces specified values in the DataFrame.

Example:

```
# Replace 'old_value' with 'new_value'
df_na_replaced = df.na.replace('old_value', 'new_value')
df_na_replaced.show()
```

`df.describe`

Definition:

Computes basic statistics for numeric columns.

Example:

```
# Describe the DataFrame
df.describe().show()
```

`df.summary`

Definition:

Provides a summary of the DataFrame.

Example:

```
# Summary of the DataFrame
df.summary().show()
```

`df.crosstab`

Definition:

Computes a cross-tabulation of two columns.

Example:

```
# Crosstab of column1 and column2
crosstab_df = df.crosstab("column1", "column2")
crosstab_df.show()
```

`df.stat.corr`

Definition:

Computes correlation between two columns.

Example:

```
# Correlation between column1 and column2
correlation = df.stat.corr("column1", "column2")
print(correlation)
```

`df.stat.cov`

Definition:

Computes covariance between two columns.

Example:

```
# Covariance between column1 and column2
covariance = df.stat.cov("column1", "column2")
print(covariance)
```

`df.stat.freqItems`

Definition:

Finds frequent items for columns.

Example:

```
# Frequent items for column1
freq_items_df = df.stat.freqItems(["column1"])
freq_items_df.show()
```

`df.stat.approxQuantile`

Definition:

Calculates approximate quantiles.

Example:

```
# Approximate quantiles for column1
quantiles = df.stat.approxQuantile("column1", [0.25, 0.5, 0.75], 0.01)
print(quantiles)
```

`df.stat.sampleBy`

Definition:

Returns a stratified sample without replacement based on the fraction given on each stratum.

Example:

```
# Stratified sample by column1
sampled_df = df.stat.sampleBy("column1", fractions={"A": 0.1, "B": 0.2}, seed=0)
sampled_df.show()
```

`df.withColumnRenamed`

Definition:

Renames an existing column in the DataFrame.

Example:

```
# Rename column1 to new_column1
renamed_df = df.withColumnRenamed("column1", "new_column1")
renamed_df.show()
```

`df.dropDuplicates`

Definition:

Drops duplicate rows based on specified columns.

Example:

```
# Drop duplicates based on column1
deduped_df = df.dropDuplicates(["column1"])
deduped_df.show()
```

`df.sample`

Definition:

Returns a sampled subset of the DataFrame.

Example:

```
# Sample 10% of the DataFrame
sampled_df = df.sample(fraction=0.1, seed=0)
sampled_df.show()
```

`df.randomSplit`

Definition:

Randomly splits the DataFrame into multiple DataFrames.

Example:

```
# Split the DataFrame into training (70%) and testing (30%) sets
train_df, test_df = df.randomSplit([0.7, 0.3], seed=0)
train_df.show()
test_df.show()
```

`df.take`

Definition:

Returns the first `n` rows of the DataFrame.

Example:

```
# Take the first 5 rows
rows = df.take(5)
for row in rows:
    print(row)
```

`df.head`

Definition:

Returns the first row or `n` rows of the DataFrame.

Example:

```
# Head of the DataFrame
head_row = df.head()
print(head_row)
```

`df.first`

Definition:

Returns the first row of the DataFrame.

Example:

```
# First row of the DataFrame
first_row = df.first()
print(first_row)
```

`df.collect`

Definition:

Returns all rows as a list of Row objects.

Example:

```
# Collect all rows
all_rows = df.collect()
for row in all_rows:
    print(row)
```

`df.toPandas`

Definition:

Converts the DataFrame to a Pandas DataFrame.

Example:

```
# Convert to Pandas DataFrame
pandas_df = df.toPandas()
print(pandas_df.head())
```

`df.write`

Definition:

Interface for saving the content of the DataFrame out into external storage systems.

Example:

```
# Write DataFrame to CSV
df.write.csv("path/to/output.csv")
```

`df.schema`

Definition:

Returns the schema of the DataFrame.

Example:

```
# Schema of the DataFrame
schema = df.schema
print(schema)
```

`df.dtypes`

Definition:

Returns a list of tuples containing the column name and data type.

Example:

```
# Data types of the DataFrame
dtypes = df.dtypes
print(dtypes)
```

`df.columns`

Definition:

Returns a list of column names.

Example:

```
# Columns of the DataFrame
columns = df.columns
print(columns)
```

`df.count`

Definition:

Returns the number of rows in the DataFrame.

Example:

```
# Count of rows
row_count = df.count()
print(row_count)
```

`df.isEmpty`

Definition:

Returns whether the DataFrame is empty.

Example:

```
# Check if DataFrame is empty
is_empty = df.isEmpty()
print(is_empty)
```

Additional Examples

Example 1: Word Count

```
from pyspark.sql.functions import explode, split

text_df = spark.read.text("path/to/textfile.txt")
word_df = text_df.select(explode(split(text_df.value, " ")).alias("word"))
word_count_df = word_df.groupBy("word").count()
word_count_df.show()
```

Example 2: Join DataFrames

```
df1 = spark.read.csv("path/to/csvfile1.csv", header=True, inferSchema=True)
df2 = spark.read.csv("path/to/csvfile2.csv", header=True, inferSchema=True)

joined_df = df1.join(df2, df1.id == df2.id, "inner")
joined_df.show()
```

Example 3: Aggregations

```
from pyspark.sql.functions import sum, avg

agg_df = df.groupBy("category").agg(sum("sales").alias("total_sales"), avg("sales").alias("average_sal
agg_df.show()
```

Example 4: Handling Missing Data

```
# Fill missing values
filled_df = df.na.fill({"column1": 0, "column2": "unknown"})
filled_df.show()

# Drop rows with missing values
dropped_df = df.na.drop()
dropped_df.show()
```

Example 5: Creating UDFs

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

def my_upper(s):
    return s.upper()

my_upper_udf = udf(my_upper, StringType())

df_with_udf = df.withColumn("upper_column", my_upper_udf(df.column1))
df_with_udf.show()
```

Example 6: Reading and Writing Parquet Files

```
# Reading Parquet file
parquet_df = spark.read.parquet("path/to/parquetfile.parquet")
parquet_df.show()

# Writing Parquet file
df.write.parquet("path/to/output.parquet")
```

Example 7: DataFrame Transformation

```
transformed_df = df.withColumn("new_column", df.column1 * 2)
transformed_df.show()
```

Example 8: Filtering Data

```
filtered_df = df.filter((df.column1 > 100) & (df.column2 == "value"))
filtered_df.show()
```

Example 9: Date and Time Operations

```
from pyspark.sql.functions import current_date, current_timestamp

date_df = df.withColumn("current_date", current_date()).withColumn("current_timestamp", current_timest
date_df.show()
```

Example 10: Working with JSON Data

```
# Reading JSON file
json_df = spark.read.json("

path/to/jsonfile.json")
json_df.show()

# Writing JSON file
df.write.json("path/to/output.json")
```

Example 11: SQL Queries

```
df.createOrReplaceTempView("table")
result_df = spark.sql("SELECT * FROM table WHERE column1 > 100")
result_df.show()
```

Example 12: Using Window Functions

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

window_spec = Window.partitionBy("category").orderBy("sales")
ranked_df = df.withColumn("rank", rank().over(window_spec))
ranked_df.show()
```

Example 13: Pivoting Data

```
pivot_df = df.groupBy("category").pivot("year").sum("sales")
pivot_df.show()
```

Example 14: Unpivoting Data

```
unpivot_df = pivot_df.selectExpr("category", "stack(3, '2020', `2020`, '2021', `2021`, '2022', `2022`)
unpivot_df.show()
```

Example 15: Sampling Data

```
sampled_df = df.sample(fraction=0.1, seed=0)
sampled_df.show()
```

Example 16: Creating a DataFrame from RDD

```
rdd = spark.sparkContext.parallelize([(1, "a"), (2, "b"), (3, "c")])
rdd_df = spark.createDataFrame(rdd, ["id", "value"])
rdd_df.show()
```

Example 17: Reading Avro Files

```
avro_df = spark.read.format("avro").load("path/to/avrofile.avro")
avro_df.show()
```

Example 18: Writing Avro Files

```
df.write.format("avro").save("path/to/output.avro")
```

Example 19: Handling Complex Types

```
from pyspark.sql.functions import explode

complex_df = df.withColumn("exploded_column", explode(df.complex_column))
complex_df.show()
```

Example 20: Machine Learning with PySpark

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

# Prepare data for ML
assembler = VectorAssembler(inputCols=["column1", "column2"], outputCol="features")
ml_df = assembler.transform(df)

# Train a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="label")
lr_model = lr.fit(ml_df)

# Make predictions
predictions = lr_model.transform(ml_df)
predictions.show()
```

Follow me on

Sumit Khanna for more updates