PySpark Pandas Commands Lookup

Initialize Spark session:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.appName("some-app").config("some-values").getOrCreate()
```

Note that many `PySpark` operations are transformations (rather than actions), which means they are not executed until an action has been called. To show results, use `df.show(n)` or `df.take(n)` to see the first `n` results, and `df.collect()` to persist all elements (may not fit into memory).

Import pandas package:

```
import pandas as pd
```

Command	Pandas	PySpark
Basics		

Read in a csv file	`pd.read_csv("path.csv")`	`spark.read \`
		`  .option("header", True) \`
		`  .csv("path.csv")`

Write to a csv file	`pd.to_csv("path.csv")`	`df.write.csv("path.csv", header = True)`
Convert between a `pandas DataFrame` and a `PySpark DataFrame`	`spark.createDataFrame(pd)`	`df.toPandas()`
Get table dimensions	`df.shape`	`df.count()` for row count `len(df.columns)` for column count
Display the first `n` rows	`df.head(n)`	`df.show(5)`

Check for null values by column	`df.isnull().sum()`	`df.select([`
		`  count(`
		`  when(isnan(c) | col(c).isNull(), c)`
		`  ).alias(c) \`
		`  for c in df.columns`
		`])`

Check for type information	`df.dtypes`	`df.dtypes`
List columns	`df.columns`	`df.columns`
Statistics		
Get summary statistics (mean, standard deviation, minimum, etc.)	`df.describe()`	`df.describe().show()`

Get quantiles of a numeric column	`df["c"].quantile(q = [0.5])`	`df.approxQuantile(`
		`  "c",`
		`  probabilities = [0.5],`
		`  relativeError = 0*`
		`)`
		* `relativeError` can be set to a value greater than 0 for faster, approximate results

Compute pairwise correlations	`df[["x", "y"]].corr()`	`df.stat.corr("x", "y")`

Command	Pandas	PySpark

Compute correlation matrix	`df.corr()`	`from pyspark.ml.stat import Correlation`
		`from pyspark.ml.feature import VectorAssembler`
		`vector_col = "features"`
		`assembler = VectorAssembler(`
		`  inputCols = df.columns,`
		`  outputCol = vector_col`
		`)`
		`df_vec = assembler.transform(df) \`
		`  .select(vector_col)`
		`Correlation.corr(df_vec, vector_col) \`
		`  .head()`

Get unique values in a column	`df["c"].unique()`	`df.select("c").distinct()`
Count unique values in a column	`df["c"].nunique()`	`df.select("c").distinct().count()`
Return count of unique values for a column	`df["c"].value_counts()`	`df.groupBy("c").count()`
Pairwise frequencies of two categorical columns	`pd.crosstab(df["x"], df["y"])`	`df.crosstab("x", "y")`

Create pivot table	`pd.pivot_table(`
	`  data = df,`
	`  values = "z",`
	`  index = "x",`
	`  columns = "y",`
	`  aggfunc = "sum"`
	`)`
		`df.groupBy("x") \`
		`  .pivot("y") \`
		`  .sum("z")`

Get histogram bins and counts	`df["c"].value_counts(bins = n)`	`bins, counts = df.select("c") \`
		`  .rdd \`
		`  .flatMap(lambda x: x) \`
		`  .histogram(n)`

Queries		
Group data by values of a column	`df.groupby("c")`	`df.groupBy("c")`
Select a column	`df["c"]`	`df.select("c")`
Subset rows based on membership in a list	`df[df["c"].isin(values)]`	`df[df["c"].isin(values)]`
Subset rows based on a numeric threshold	`df[df["c"] > value]`	`df[df["c"] > value]`
Subset rows based on string patterns	`df[df["c"].str.startswith(pattern)]`	`df[df["c"].startswith(pattern)]`
Select values based on a filter condition	`df.loc[df["x"] == value, "y"]`	`df.filter(df["x"] == value).select("y")`
Data Wrangling		

Sort data by a column	`df.sort_values(`
	`  by = "x",`
	`  ascending = False`
	`)`
		`df.orderBy(df.x.desc())`

Cast to a different data type	`df.c.astype("float64")`	`df.withColumn(`
		`  "c",`
		`  df["c"].cast("double")`
		`)`

Command	Pandas	PySpark

Select a sample of observations (without replacement)	`df.sample(`
	`  frac = frac,`
	`  replace = False`
	`)`
		`df.sample(`
		`  withReplacement = False,`
		`  fraction = frac`
		`)`

Remove duplicated entries	`df.drop_duplicates(`
	`  subset = ["x", "y"]`
	`)`
		`df.dropDuplicates(subset = ["x", "y"])`

Remove rows with missing values	`df.dropna(how = "any")`	`df.dropna(how = "any")`
Replace missing values with a constant	`df["x"].fillna(-1)`	`df.fillna(-1, subset = ["x"])`
Remove columns	`df.drop(columns = ["x", "y"])`	`df.drop("x", "y")`

Rename a column	`df.rename(`
	`  columns = {"old_c": "new_c"}`
	`)`
		`df.withColumnRenamed("old_c", "new_c")`

Replace values according to a condition	`df.loc[df.x == "c", "y"] = "d"`	`df.withColumn(`
		`  "y",`
		`  when(`
		`  df["x"] == "c", "d"`
		`  ).otherwise(df["y"])`
		`)`

Add a new column	`df["z"] = values`	`df.withColumn("z", values*)` * This works if `values` is a function of another column in `df` (e.g. `2*col("x")`) or if it is a constant value (i.e. `lit(value)`). Otherwise, need to apply `join` as done when concatenating two tables by column

Bin values into discrete intervals	`df["x_grouped"] = pd.cut(`
	`  df.x, bins = intervals_list`
	`)`
		`from pyspark.ml.feature import Bucketizer`
		`bucketizer = Bucketizer(`
		`  splits = intervals_list,`
		`  inputCol = "x",`
		`  outputCol = "x_grouped"`
		`)`
		`df = bucketizer.transform(df)`

Merge two tables	`df1.merge(`
	`  df2, on = ["x"], how = "left"`
	`)`
		`df1.join(`
		`  df2, on = ["x"], how = "left"`
		`)`

Append a table to the end of another table	`df1.append(df2)`	`df1.union(df2)` (need to have same schema)

Concatenate two tables by column	`pd.concat([df1, df2], axis = 1)`	`df1 = df1.withColumn(`
		`  "id", monotonically_increasing_id()`
		`)`
		`df2 = df2.withColumn(`
		`  "id", monotonically_increasing_id()`
		`)`
		`df1.join(`
		`  df2, on = "id", how = "outer"`
		`).drop("id")`

Apply custom functions	`df.x.apply(foo).rename("x_new")`	`from pyspark.sql.types import FloatType`
		`foo_udf = udf(foo, FloatType())`
		`df.select(foo_udf("x").alias("x_new"))`