

# 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
<b>Basics</b>		
Read in a csv file	<code>pd.read_csv("path.csv")</code>	<code>spark.read \</code> <code>.option("header", True) \</code> <code>.csv("path.csv")</code>
Write to a csv file	<code>pd.to_csv("path.csv")</code>	<code>df.write.csv("path.csv", header = True)</code>
Convert between a pandas DataFrame and a PySpark DataFrame	<code>spark.createDataFrame(pd)</code>	<code>df.toPandas()</code>
Get table dimensions	<code>df.shape</code>	<code>df.count()</code> for row count <code>len(df.columns)</code> for column count
Display the first <code>n</code> rows	<code>df.head(n)</code>	<code>df.show(5)</code>
Check for null values by column	<code>df.isnull().sum()</code>	<code>df.select([</code> <code>count(</code> <code>    when(isnan(c)   col(c).isNull(), c)</code> <code>).alias(c) \</code> <code>for c in df.columns</code> <code>])</code>
Check for type information	<code>df.dtypes</code>	<code>df.dtypes</code>
List columns	<code>df.columns</code>	<code>df.columns</code>
<b>Statistics</b>		
Get summary statistics (mean, standard deviation, minimum, etc.)	<code>df.describe()</code>	<code>df.describe().show()</code>
Get quantiles of a numeric column	<code>df["c"].quantile(q = [0.5])</code>	<code>df.approxQuantile(</code> <code>    "c",</code> <code>    probabilities = [0.5],</code> <code>    relativeError = 0*</code> <code>)</code> * <code>relativeError</code> can be set to a value greater than 0 for faster, approximate results
Compute pairwise correlations	<code>df[["x", "y"]].corr()</code>	<code>df.stat.corr("x", "y")</code>

Command	Pandas	PySpark
Compute correlation matrix	<code>df.corr()</code>	<pre> 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() </pre>
Get unique values in a column	<code>df["c"].unique()</code>	<code>df.select("c").distinct()</code>
Count unique values in a column	<code>df["c"].nunique()</code>	<code>df.select("c").distinct().count()</code>
Return count of unique values for a column	<code>df["c"].value_counts()</code>	<code>df.groupBy("c").count()</code>
Pairwise frequencies of two categorical columns	<code>pd.crosstab(df["x"], df["y"])</code>	<code>df.crosstab("x", "y")</code>
Create pivot table	<pre> pd.pivot_table(     data = df,     values = "z",     index = "x",     columns = "y",     aggfunc = "sum" ) </pre>	<pre> df.groupBy("x") \     .pivot("y") \     .sum("z") </pre>
Get histogram bins and counts	<code>df["c"].value_counts(bins = n)</code>	<pre> bins, counts = df.select("c") \     .rdd \     .flatMap(lambda x: x) \     .histogram(n) </pre>
<b>Queries</b>		
Group data by values of a column	<code>df.groupby("c")</code>	<code>df.groupBy("c")</code>
Select a column	<code>df["c"]</code>	<code>df.select("c")</code>
Subset rows based on membership in a list	<code>df[df["c"].isin(values)]</code>	<code>df[df["c"].isin(values)]</code>
Subset rows based on a numeric threshold	<code>df[df["c"] &gt; value]</code>	<code>df[df["c"] &gt; value]</code>
Subset rows based on string patterns	<code>df[df["c"].str.startswith(pattern)]</code>	<code>df[df["c"].startswith(pattern)]</code>
Select values based on a filter condition	<code>df.loc[df["x"] == value, "y"]</code>	<code>df.filter(df["x"] == value).select("y")</code>
<b>Data Wrangling</b>		
Sort data by a column	<pre> df.sort_values(     by = "x",     ascending = False ) </pre>	<code>df.orderBy(df.x.desc())</code>
Cast to a different data type	<code>df.c.astype("float64")</code>	<pre> df.withColumn(     "c",     df["c"].cast("double") ) </pre>

Command	Pandas	PySpark
Select a sample of observations (without replacement)	<pre>df.sample(     frac = frac,     replace = False )</pre>	<pre>df.sample(     withReplacement = False,     fraction = frac )</pre>
Remove duplicated entries	<pre>df.drop_duplicates(     subset = ["x", "y"] )</pre>	<pre>df.dropDuplicates(subset = ["x", "y"])</pre>
Remove rows with missing values	<pre>df.dropna(how = "any")</pre>	<pre>df.dropna(how = "any")</pre>
Replace missing values with a constant	<pre>df["x"].fillna(-1)</pre>	<pre>df.fillna(-1, subset = ["x"])</pre>
Remove columns	<pre>df.drop(columns = ["x", "y"])</pre>	<pre>df.drop("x", "y")</pre>
Rename a column	<pre>df.rename(     columns = {"old_c": "new_c"} )</pre>	<pre>df.withColumnRenamed("old_c", "new_c")</pre>
Replace values according to a condition	<pre>df.loc[df.x == "c", "y"] = "d"</pre>	<pre>df.withColumn(     "y",     when(         df["x"] == "c", "d"     ).otherwise(df["y"]) )</pre>
Add a new column	<pre>df["z"] = values</pre>	<pre>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</pre>
Bin values into discrete intervals	<pre>df["x_grouped"] = pd.cut(     df.x, bins = intervals_list )</pre>	<pre>from pyspark.ml.feature import Bucketizer bucketizer = Bucketizer(     splits = intervals_list,     inputCol = "x",     outputCol = "x_grouped" ) df = bucketizer.transform(df)</pre>
Merge two tables	<pre>df1.merge(     df2, on = ["x"], how = "left" )</pre>	<pre>df1.join(     df2, on = ["x"], how = "left" )</pre>
Append a table to the end of another table	<pre>df1.append(df2)</pre>	<pre>df1.union(df2) (need to have same schema)</pre>
Concatenate two tables by column	<pre>pd.concat([df1, df2], axis = 1)</pre>	<pre>df1 = df1.withColumn(     "id", monotonically_increasing_id() ) df2 = df2.withColumn(     "id", monotonically_increasing_id() ) df1.join(     df2, on = "id", how = "outer" ).drop("id")</pre>
Apply custom functions	<pre>df.x.apply(foo).rename("x_new")</pre>	<pre>from pyspark.sql.types import FloatType foo_udf = udf(foo, FloatType()) df.select(foo_udf("x").alias("x_new"))</pre>