Working with Different Types of Data		
Keywork	**Explanation**	**Example**
lit	converts a type in another lang intoSpark type	df.select(lit(5), lit("five"), lit(5.0))
=== or =	wokring w/ boolean - equality	df.where(col("colName") === intOrStr) df.where("InvoiceNo = 536373")
<>	wokring w/ boolean - difference	df.where("InvoiceNo <> 536365")
.contains isin	wokring w/ boolean - > & contains	val priceFilter = col("ColE") > 600 val descripFilter = col("ColNameA").contains("WORD") df.where(col("ColNameB").isin("DOT")).where(priceFilter.or(descripFilter)).show()
withCol expr	wokring w/ boolean -	df.withColumn("isExpensive", expr("NOT colA <= 250")) .filter("isExpensive") .select("ColNameA", "ColE").show(5)
power * +	wokring w/ nb - numerical expr	df.selectExpr("ColNameC", "(POWER((ColD * ColE), 2.0) + 5) as realColD").show(2)
round	wokring w/ nb - rounded float	df.select(round(col("ColE"), 1).alias("rounded"), col("ColE"))
corr	wokring w/ nb - the correlation of two columns	df.select(corr("ColD", "ColE")).show() spark.sql("SELECT corr(ColD, ColE) FROM dfTable").show()
describe	wokring w/ nb - compute summary statistics	df.describe().show()
approxQuantile	wokring w/ nb -	val colName = "ColE" val quantileProbs = Array(0.5) val relError = 0.05 df.stat.approxQuantile("ColE", quantileProbs, relError)
stat.crosstab freqit	wokring w/ nb - cross-tabulation or frequent item pairs	df.stat.crosstab("ColNameB", "ColD").show(2) df.stat.freqItems(Seq("ColNameB", "ColD")).show()
monotically_incr	wokring w/ nb - add a unique ID to each row	df.select(col("InvoiceNo"), monotonically_increasing_id()).show(5)
initcap	wokring w/ str - capitalize every words separated by a space	df.select(initcap(col("ColNameA"))).show(2, false)
lower / upper	cast strings in uppercase and lowercase	df.select(col("ColNameA"), lower(col("ColNameA")), upper(lower(col("ColNameA")))).show(2)
l/r/trim l/r/pad	adding or removing spaces around a string	df.select(ltrim(lit(" HELLO ")).as("ltrim"), rtrim(lit(" HELLO ")).as("rtrim"), trim(lit(" HELLO ")).as("trim"), lpad(lit("HELLO"), 3, " ").as("lp"), rpad(lit("HELLO"), 10, " ").as("rp")).show(2)
regexp_replace	wokring w/ str - RegEx	val simpleColors = Seq("black", "white", "red", "green", "blue") val regexString = simpleColors.map(_.toUpperCase).mkString("|") // the | signifies `OR` in regular expression syntax df.select(regexp_replace(col("ColNameA"), regexString, "COLOR").alias("color_clean"), col("ColNameA")).show()
translate	wokring w/ str - replace given characters with other	df.select(translate(col("ColNameA"), "LEET", "1337"), col("ColNameA")) .show(2)
regexp_extract	wokring w/ str - map	val regexString = simpleColors.map(_.toUpperCase).mkString("(", "|", ")") df.select(regexp_extract(col("ColNameA"), regexString, 1).alias("color_clean"), col("ColNameA")).show(2)
	wokring w/ str - check for their existence	val containsBlack = col("ColNameA").contains("BLACK") val containsWhite = col("ColNameA").contains("WHITE") df.withColumn("hasSimpleColor", containsBlack.or(containsWhite)) .where("hasSimpleColor") .select("ColNameA").show(3, false)
range	working w/ dates & timestps - create DF	val dateDF = spark.range(10) .withColumn("today", current_date()) .withColumn("now", current_timestamp())
date_sub / add	working w/ dates - add 5 days	dateDF.select(date_sub(col("today"), 5), date_add(col("today"), 5))

to_date	working w/ dates - to_date	dateDF.select(to_date(lit("2016-01-01")).alias("start"), to_date(lit("2017-05-22")).alias("end")) .select(months_between(col("start"), col("end"))).show(1)
to_date	working w/ dates	val dateFormat = "yyyy-dd-MM" val cleanDateDF = spark.range(1).select(to_date(lit("2017-12-11"), dateFormat).alias("date"), to_date(lit("2017-20-12"), dateFormat).alias("date2"))
		cleanDateDF.select(to_timestamp(col("date"), dateFormat))
filter	filtering	cleanDateDF.filter(col("date2") > "'2017-12-12'").show()
coalesce	working w/ nulls - Coalesce - function to allow you to select the first non-null value from a set of columns	df.select(coalesce(col("ColNameA"), col("ColNameC")))
na.drop	working w/ nulls - drop	df.na.drop("any") df.na.drop("all")
na.drop seq	working w/ nulls - apply this to certain sets of columns	df.na.drop("all", Seq("ColNameB", "colNameA"))
na.fill	working w/ nulls - fill	df.na.fill("All Null values become this string")
struct	complex types - creation	val complexDF = df.select(struct("ColNameA", "InvoiceNo").alias("complex"))
select	complex types - selection	complexDF.select("complex.ColNameA").show(2)
split	complex types - split	df.select(split(col("ColNameA"), " ")).show(2)
split	complex types - split	df.select(split(col("ColNameA"), " ").alias("array_col")).selectExpr("array_col[0]")
size	complex types - size	df.select(size(split(col("ColNameA"), " "))).show(2)
array_contains	complex types - array_contains	df.select(array_contains(split(col("ColNameA"), " "), "WHITE"))
explode	complex types - explode	df.withColumn("splitted", split(col("ColNameA"), " ")) .withColumn("exploded", explode(col("splitted")))
map	complex types - map	df.select(map(col("ColNameA"), col("InvoiceNo")).alias("complex_map"))
spark.udf.register	UDF - User defined functions	val udfExampleDF = spark.range(5).toDF("num") def power3(number:Double):Double = number * number * number -> power3(2.0) spark.udf.register("power3", power3(_:Double):Double) udfExampleDF.selectExpr("power3(num)").show(2)