Mejores Practicas de PySpark:  
\*Usar pyspark.sqlfunctions o funciones predeterminadas  
\*Usar la misma versión de Python y paqueterías en todo el Cluster!  
\*Revisar el UI de Spark  
\*Revisar Spark MLlib (Sklearn de PySpark)  
\*Usar Spark-submit

NOT TO DO:  
\*No iterar por renglones  
\*df.toPandas.head() ->major usar ->df.limit(5).toPandas()

**\*Start PySpark on Python\****from pyspark import SparkConf, SparkContext  
conf = SparkConf().setMaster("local").setAppName("ContadorRatings")  
sc = SparkContext(conf = conf)*

|  |  |
| --- | --- |
| **Python Pandas** | **PySpark** |
| **1-Read File** | |
| df = pd.read\_csv(‘mtcars.csv, delimiter=’|’) | df = spark.read \  .options(header=True, inferSchema=True) \  .csv(‘mtcars.csv’) |
| 1.1 – Create 2 column Dataframe | |
|  | Df = sc.parallelize(  (‘honda’,12),  (‘susuki’, 293),  (‘honda’,39)  ) #Puede repetirse la llave |
| Create DataFrame with dictionary | |
|  | Df = sc.parallelize(  (123, {‘nombre:’jose’, ‘apellido’:’perez’}),  (124, {‘nombre:’eli, ‘apellido’:diaz’})  ) |
| 1.1 Transformación: reduceByKey | |
|  | A = sc.parallelize([(1,3), (3,100), (1,5),(3,2)])  A.reduceByKey(lambda x,y : x\*y).collect()  *# Out: [(1,15), (3,200)]* |
| 1.1 Transformación: groupByKey | |
|  | A = sc.parallelize([(1,3), (3,100), (1,5),(3,2)])  A.groupByKey().map(lambda k,iter: (k,[x for x in iter]) )  *# Out: [(1,[ 3,5]), (3,[100,2])]* |
| 1.1 Acción: countByKey, lookup | |
|  | A = sc.parallelize([(1,3), (3,100), (1,5),(3,2)])  A.countByKey()  *# Out: {1:2, 3:2}*  A.lookup(3)  *# Out: [100,2]*  A.collectAcMap()  *# Out: [1:[3,5], 3[100,2]]* |
| **1.2 DataFrame details** | |
| df  df.head(15)  df.columns  df.dtypes  df.describe() | df.show()  df.show(15)  df.columns  df.dtypes  df.describe().show() |
| **1.3 Alter Table** | |
| df.columns=[‘col1’,’col2’,’col3’]  df.rename(columns={‘old’:’new’})  df.drop(‘col2’,axis=1) | df.toDF(‘col1’,’col2’,’col3’)  df.withColumnRenamed(‘old’,’new’)  df.drop(‘col2’) |
| **1.4 New Column (with Function and Conditional)** | |
| import numpy as np  df[‘logvar’]=np.log(df.var)  df[‘cond’]=df.aaply(lambda r:  1 if r.var1<20 else  2 if r.var2==5 else  3, axis=1)  df[‘cond\_2’] = df.var.apply(lambda x: x+1) | import pyspark.sql.functions as F  df.withColumn(‘logvar’, F.log(df.var))  df.withColumn(‘cond’, \  F.when(df.var1<20, 1) \  .when(df.var2==5, 2) \  .otherwise(3)  from pyspark.sql.types import DoubleType  fn = F.udf(lambda x: x+1, DoubleType()) #UserDefineFunction  df.withColumn(‘cond\_2’, fn(df.var)) |
| **1.4 Filtering** | |
| df[(df.col1<100) & (df.col2==5)] | df[(df.col1<100) & (df.col2==5)] |
| **1.5 Add Column** | |
| df[‘new\_col’]= 1/df.col2 | df.withColumn(‘new\_col’,1/df.col2) |
| **1.6 Fill NAs – Pandas has more options** | |
| df.fillna(0) | df.fillna(0) |
| **1.7 Aggregation with GroupBy** | |
| df.groupby([‘cat1’,’cat2’]) \  .agg({‘mpg’:’sum’, ‘disp’:’min’}) | df.groupby([‘cat1’,’cat2’]) \  .agg({‘mpg’:’sum’, ‘disp’:’min’}) |
| **1.8 Merge / SQL Join DataFrames** | |
| df\_left.merge(df\_right, on=’key’)  df\_left.merge(df\_right, left\_on=’a’,right\_on=’b’) | df\_left.join(df\_right, on=’key’)  df\_left.join(right, df\_left.a == df\_right.b) |
| **1.9 Pivot Table** | |
| pd.pivot\_table(df,values=’D’, \  index=[‘A’,’B’], columns=[‘C’], \  aggfunc = np.sum) | df.groupBy(‘A’,’B’).pivot(‘C’).sum(‘D’) |
| **1.20 DataFrame Plot** | |
| df.hist() | df.sample(False, 0.1).toPandas().hist() |
| **1.21 SQL** | |
| N/A | df.createOrReplaceTempView(‘foo’)  df2 = spark.sql( ‘select \* from foo’ ) |
|  |  |

