select()

function is used to select single, multiple, column by index, all columns from the list and the nested columns from a DataFrame, PySpark select() is a transformation function hence it returns a new DataFrame with the selected columns.

| import pyspark  from pyspark.sql import SparkSession  spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()  data = [("James","Smith","USA","CA"),  ("Michael","Rose","USA","NY"),  ("Robert","Williams","USA","CA"),  ("Maria","Jones","USA","FL")  ]  columns = ["firstname","lastname","country","state"]  df = spark.createDataFrame(data = data, schema = columns)  df.show(truncate=False) |
| --- |

## **Select Single & Multiple Columns From PySpark**

| df.select("firstname","lastname").show()  df.select(df.firstname,df.lastname).show()  df.select(df["firstname"],df["lastname"]).show()  #By using col() function  from pyspark.sql.functions import col  df.select(col("firstname"),col("lastname")).show()  #Select All  df.select("\*").show() |
| --- |

## **Select Columns by Index**

| #Selects first 3 columns and top 3 rows  df.select(df.columns[:3]).show(3)  #Selects columns 2 to 4 and top 3 rows  df.select(df.columns[2:4]).show(3) |
| --- |

## **Select Nested Struct Columns from PySpark**

| data = [  (("James",None,"Smith"),"OH","M"),  (("Anna","Rose",""),"NY","F"),  (("Julia","","Williams"),"OH","F"),  (("Maria","Anne","Jones"),"NY","M"),  (("Jen","Mary","Brown"),"NY","M"),  (("Mike","Mary","Williams"),"OH","M")  ]  from pyspark.sql.types import StructType,StructField, StringType  schema = StructType([  StructField('name', StructType([  StructField('firstname', StringType(), True),  StructField('middlename', StringType(), True),  StructField('lastname', StringType(), True)  ])),  StructField('state', StringType(), True),  StructField('gender', StringType(), True)  ])  df2 = spark.createDataFrame(data = data, schema = schema)  df2.printSchema()  df2.show(truncate=False) # shows all columns |
| --- |

| df2.select("name").show(truncate=False) |
| --- |

| df2.select("name.firstname","name.lastname").show(truncate=False) |
| --- |

# **withColumn()**

is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column

| data = [('James','','Smith','1991-04-01','M',3000),  ('Michael','Rose','','2000-05-19','M',4000),  ('Robert','','Williams','1978-09-05','M',4000),  ('Maria','Anne','Jones','1967-12-01','F',4000),  ('Jen','Mary','Brown','1980-02-17','F',-1)  ]  columns = ["firstname","middlename","lastname","dob","gender","salary"]  from pyspark.sql import SparkSession  spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()  df = spark.createDataFrame(data=data, schema = columns) |
| --- |

## **Change DataType using PySpark withColumn()**

By using PySpark withColumn() on a DataFrame, we can cast or change the data type of a column. In order to change data type, you would also need to use cast() function along with withColumn(). The below statement changes the datatype from Long to Double for the salary column.

| data = [('James','','Smith','1991-04-01','M',3000),  ('Michael','Rose','','2000-05-19','M',4000),  ('Robert','','Williams','1978-09-05','M',4000),  ('Maria','Anne','Jones','1967-12-01','F',4000),  ('Jen','Mary','Brown','1980-02-17','F',-1)  ]  columns = ["firstname","middlename","lastname","dob","gender","salary"]  from pyspark.sql import SparkSession  spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()  df = spark.createDataFrame(data=data, schema = columns) |
| --- |

| ddf = df.withColumn("salary",col("salary").cast("Double")) |
| --- |

## **Update The Value of an Existing Column**

| udf = df.withColumn("salary",col("salary")\*100) |
| --- |

## **Create a Column from an Existing**

To add/create a new column, specify the first argument with a name you want your new column to be and use the second argument to assign a value by applying an operation on an existing column.

| ncol = df.withColumn("CopiedColumn",col("salary")\* -1)  ncol.show() |
| --- |

## **Add a New Column using withColumn() with constant value**

In order to create a new column, pass the column name you wanted to the first argument of withColumn() transformation function. Make sure this new column is not already present on DataFrame, if it presents it updates the value of that column.

In the below snippet, the PySpark lit() function is used to add a constant value to a DataFrame column. We can also chain in order to add multiple columns.

| from pyspark.sql.functions import col,lit  df.withColumn("Country", lit("USA")).show() |
| --- |

## **Rename Column Name**

| df.withColumnRenamed("gender","sex") \  .show(truncate=False) |
| --- |

filter()

function is used to filter the rows from RDD/DataFrame based on the given condition or SQL expression, you can also use where() clause instead of the filter() if you are coming from an SQL background, both these functions operate exactly the same.

| from pyspark.sql.types import StructType,StructField  from pyspark.sql.types import StringType, IntegerType, ArrayType  data = [  (("James","","Smith"),["Java","Scala","C++"],"OH","M"),  (("Anna","Rose",""),["Spark","Java","C++"],"NY","F"),  (("Julia","","Williams"),["CSharp","VB"],"OH","F"),  (("Maria","Anne","Jones"),["CSharp","VB"],"NY","M"),  (("Jen","Mary","Brown"),["CSharp","VB"],"NY","M"),  (("Mike","Mary","Williams"),["Python","VB"],"OH","M")  ]    schema = StructType([  StructField('name', StructType([  StructField('firstname', StringType(), True),  StructField('middlename', StringType(), True),  StructField('lastname', StringType(), True)  ])),  StructField('languages', ArrayType(StringType()), True),  StructField('state', StringType(), True),  StructField('gender', StringType(), True)  ])  df = spark.createDataFrame(data = data, schema = schema)  df.printSchema()  df.show(truncate=False) |
| --- |

## **DataFrame filter() with Column Condition**

Use Column with the condition to filter the rows from DataFrame, using this you can express complex condition by referring column names using dfObject.colname

| df.filter(df.state == "OH").show(truncate=False)  # not equals condition  df.filter(df.state != "OH") \  .show(truncate=False)  df.filter(~(df.state == "OH")) \  .show(truncate=False) |
| --- |

Same example can also written as below. In order to use this first you need to import from pyspark.sql.functions import col

| #Using SQL col() function  from pyspark.sql.functions import col  df.filter(col("state") == "OH") \  .show(truncate=False) |
| --- |

## **DataFrame filter() with SQL Expression**

If you are coming from SQL background, you can use that knowledge in PySpark to filter DataFrame rows with SQL expressions.

| #Using SQL Expression  df.filter("gender == 'M'").show()  #For not equal  df.filter("gender != 'M'").show()  df.filter("gender <> 'M'").show() |
| --- |

## **PySpark Filter with Multiple Conditions**

| //Filter multiple condition  df.filter( (df.state == "OH") & (df.gender == "M") ) \  .show(truncate=False) |
| --- |

## **Filter Based on List Values**

If you have a list of elements and you wanted to filter that is not in the list or in the list, use isin() function of Column class and it doesn’t have isnotin() function but you do the same using not operator (~)

| li=["OH","CA","DE"]  df.filter(df.state.isin(li)).show() |
| --- |

| # Filter NOT IS IN List values  #These show all records with NY (NY is not part of the list)  df.filter(~df.state.isin(li)).show()  df.filter(df.state.isin(li)==False).show() |
| --- |

## **Filter Based on Starts With, Ends With, Contains**

You can also filter DataFrame rows by using startswith(), endswith() and contains() methods of the Column class.

| # Using startswith  df.filter(df.state.startswith("N")).show()  #using endswith  df.filter(df.state.endswith("H")).show()  #contains  df.filter(df.state.contains("H")).show() |
| --- |

## **Filter like and rlike**

If you have SQL background you must be familiar with like and rlike (regex like), PySpark also provides similar methods in Column class to filter similar values using wildcard characters. You can use rlike() to filter by checking values case insensitive.

| data2 = [(2,"Michael Rose"),(3,"Robert Williams"),  (4,"Rames Rose"),(5,"Rames rose")  ]  df2 = spark.createDataFrame(data = data2, schema = ["id","name"])  # like - SQL LIKE pattern  df2.filter(df2.name.like("%rose%")).show() |
| --- |

## **Filter on an Array column**

When you want to filter rows from DataFrame based on value present in an array collection column, you can use the first syntax. The below example uses array\_contains() from Pyspark SQL functions which checks if a value contains in an array if present it returns true otherwise false.

| from pyspark.sql.functions import array\_contains  df.filter(array\_contains(df.languages,"Java")) \  .show(truncate=False) |
| --- |

## **Filtering on Nested Struct columns**

If your DataFrame consists of nested struct columns, you can use any of the above syntaxes to filter the rows based on the nested column.

| //Struct condition  df.filter(df.name.lastname == "Williams") \  .show(truncate=False) |
| --- |