**pyspark.sql.Row**

A row in DataFrame. The fields in it can be accessed:

* like attributes (row.key)
* like dictionary values (row[key])

Row can be used to create a row object by using named arguments. It is **not allowed to omit a named argument** to represent that the value is None or missing. This should be explicitly set to None in this case.

|  |
| --- |
| >>>row = Row(name="Alice", age=11)  >>>row  Row(name='Alice', age=11)  >>>row['name'], row['age']  ('Alice', 11)  >>>row.name, row.age  ('Alice', 11)  >>>'name' in row  True  >>>'wrong\_key' in row  False  >>>row1 = Row("Alice", 11)  >>>row2 = Row(name="Alice", age=11)  >>>row1 == row2  True |
| **Methods**   |  |  | | --- | --- | | [**asDict**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Row.asDict.html#pyspark.sql.Row.asDict)([recursive]) | Return as a dict | | **count**(value, /) | Return number of occurrences of value. | | **index**(value[, start, stop]) | Return first index of value. | |

|  |
| --- |
| pyspark.sql.Row.asDict(recursive=False)  If a row contains duplicate field names, e.g., the rows of a join between two DataFrame that both have the fields of same names, one of the duplicate fields will be selected by asDict. \_\_getitem\_\_ will also return one of the duplicate fields, however returned value might be different to asDict.  >>>Row(name="Alice", age=11).asDict() == {'name': 'Alice', 'age': 11}  True  >>>row = Row(key=1, value=Row(name='a', age=2))  >>>row.asDict() == {'key': 1, 'value': Row(name='a', age=2)}  True  >>>row.asDict(True) == {'key': 1, 'value': {'name': 'a', 'age': 2}}  True |

|  |
| --- |
| **Create a Row Object**  from pyspark.sql import Row  row=Row("James",40)  print(row[0] +","+str(row[1]))  🡪James,40  row=Row(name="Alice", age=11)  print(row.name) |
| **Create Custom Class from Row**  We can also create a Row like class, for example “Person” and use it similar to Row object. This would be helpful when you wanted to create real time object and refer it’s properties.  Person = Row("name", "age")  p1=Person("James", 40)  p2=Person("Alice", 35)  print(p1.name +","+p2.name) |
| **Using Row class on PySpark RDD**  from pyspark.sql import SparkSession, Row  spark = SparkSession.builder. getOrCreate()  data = [Row(name="James,,Smith",lang=["Java","Scala","C++"],state="CA"),  Row(name="Michael,Rose,",lang=["Spark","Java","C++"],state="NJ"),  Row(name="Robert,,Williams",lang=["CSharp","VB"],state="NV")]  rdd=spark.sparkContext.parallelize(data)  print(rdd.collect())  collData=rdd.collect()  for row in collData:  print(row.name + "," +str(row.lang)) |
| **Using Row class on PySpark DataFrame**  Row class also can be used with PySpark DataFrame, By default data in DataFrame represent as Row.  Note that Row on DataFrame is not allowed to omit a named argument to represent that the value is None or missing.  df=spark.createDataFrame(data)  df.printSchema()  df.show()  columns = ["name","languagesAtSchool","currentState"]  df=spark.createDataFrame(data).toDF(\*columns)  df.printSchema() |
| **Create Nested Struct Using Row Class**  #Create DataFrame with struct using Row class  from pyspark.sql import Row  data=[Row(name="James",prop=Row(hair="black",eye="blue")),  Row(name="Ann",prop=Row(hair="grey",eye="black"))]  df=spark.createDataFrame(data)  df.printSchema() |
| from pyspark.sql import SparkSession, Row  row=Row("James",40)  print(row[0] +","+str(row[1]))  row2=Row(name="Alice", age=11)  print(row2.name)  Person = Row("name", "age")  p1=Person("James", 40)  p2=Person("Alice", 35)  print(p1.name +","+p2.name)  #PySpark Example  spark = SparkSession.builder. getOrCreate()  data = [Row(name="James,,Smith",lang=["Java","Scala","C++"],state="CA"),  Row(name="Michael,Rose,",lang=["Spark","Java","C++"],state="NJ"),  Row(name="Robert,,Williams",lang=["CSharp","VB"],state="NV")]  #RDD Example 1  rdd=spark.sparkContext.parallelize(data)  collData=rdd.collect()  print(collData)  for row in collData:  print(row.name + "," +str(row.lang))  # RDD Example 2  Person=Row("name","lang","state")  data = [Person("James,,Smith",["Java","Scala","C++"],"CA"),  Person("Michael,Rose,",["Spark","Java","C++"],"NJ"),  Person("Robert,,Williams",["CSharp","VB"],"NV")]  rdd=spark.sparkContext.parallelize(data)  collData=rdd.collect()  print(collData)  for person in collData:  print(person.name + "," +str(person.lang))  #DataFrame Example 1  columns = ["name","languagesAtSchool","currentState"]  df=spark.createDataFrame(data)  df.printSchema()  df.show()  collData=df.collect()  print(collData)  for row in collData:  print(row.name + "," +str(row.lang))    #DataFrame Example 2  columns = ["name","languagesAtSchool","currentState"]  df=spark.createDataFrame(data).toDF(\*columns)  df.printSchema() |

**pyspark.sql.Column**

df.colName

df["colName"]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Methods**   |  |  | | --- | --- | | [**alias**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.alias.html#pyspark.sql.Column.alias)(\*alias, \*\*kwargs) | Returns this column aliased with a new name or names (in the case of expressions that return more than one column, such as explode). | | [**asc**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.asc.html#pyspark.sql.Column.asc)() | Returns a sort expression based on ascending order of the column. | | [**asc\_nulls\_first**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.asc_nulls_first.html#pyspark.sql.Column.asc_nulls_first)() | Returns a sort expression based on ascending order of the column, and null values return before non-null values. | | [**asc\_nulls\_last**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.asc_nulls_last.html#pyspark.sql.Column.asc_nulls_last)() | Returns a sort expression based on ascending order of the column, and null values appear after non-null values. | | [**astype**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.astype.html#pyspark.sql.Column.astype)(dataType) | [**astype()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.astype.html#pyspark.sql.Column.astype) is an alias for [**cast()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.cast.html#pyspark.sql.Column.cast). | | [**between**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.between.html#pyspark.sql.Column.between)(lowerBound, upperBound) | True if the current column is between the lower bound and upper bound, inclusive. | | [**bitwiseAND**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.bitwiseAND.html#pyspark.sql.Column.bitwiseAND)(other) | Compute bitwise AND of this expression with another expression. | | [**bitwiseOR**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.bitwiseOR.html#pyspark.sql.Column.bitwiseOR)(other) | Compute bitwise OR of this expression with another expression. | | [**bitwiseXOR**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.bitwiseXOR.html#pyspark.sql.Column.bitwiseXOR)(other) | Compute bitwise XOR of this expression with another expression. | | [**cast**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.cast.html#pyspark.sql.Column.cast)(dataType) | Casts the column into type dataType. | | [**contains**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.contains.html#pyspark.sql.Column.contains)(other) | Contains the other element. | | [**desc**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.desc.html#pyspark.sql.Column.desc)() | Returns a sort expression based on the descending order of the column. | | [**desc\_nulls\_first**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.desc_nulls_first.html#pyspark.sql.Column.desc_nulls_first)() | Returns a sort expression based on the descending order of the column, and null values appear before non-null values. | | [**desc\_nulls\_last**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.desc_nulls_last.html#pyspark.sql.Column.desc_nulls_last)() | Returns a sort expression based on the descending order of the column, and null values appear after non-null values. | | [**dropFields**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.dropFields.html#pyspark.sql.Column.dropFields)(\*fieldNames) | An expression that drops fields in **StructType** by name. | | [**endswith**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.endswith.html#pyspark.sql.Column.endswith)(other) | String ends with. | | [**eqNullSafe**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.eqNullSafe.html#pyspark.sql.Column.eqNullSafe)(other) | Equality test that is safe for null values. | | [**getField**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.getField.html#pyspark.sql.Column.getField)(name) | An expression that gets a field by name in a **StructType**. | | [**getItem**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.getItem.html#pyspark.sql.Column.getItem)(key) | An expression that gets an item at position ordinal out of a list, or gets an item by key out of a dict. | | [**isNotNull**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.isNotNull.html#pyspark.sql.Column.isNotNull)() | True if the current expression is NOT null. | | [**isNull**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.isNull.html#pyspark.sql.Column.isNull)() | True if the current expression is null. | | [**isin**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.isin.html#pyspark.sql.Column.isin)(\*cols) | A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of the arguments. | | [**like**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.like.html#pyspark.sql.Column.like)(other) | SQL like expression. | | [**name**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.name.html#pyspark.sql.Column.name)(\*alias, \*\*kwargs) | [**name()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.name.html#pyspark.sql.Column.name) is an alias for [**alias()**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.alias.html#pyspark.sql.Column.alias). | | [**otherwise**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.otherwise.html#pyspark.sql.Column.otherwise)(value) | Evaluates a list of conditions and returns one of multiple possible result expressions. | | [**over**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.over.html#pyspark.sql.Column.over)(window) | Define a windowing column. | | [**rlike**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.rlike.html#pyspark.sql.Column.rlike)(other) | SQL RLIKE expression (LIKE with Regex). | | [**startswith**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.startswith.html#pyspark.sql.Column.startswith)(other) | String starts with. | | [**substr**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.substr.html#pyspark.sql.Column.substr)(startPos, length) | Return a [**Column**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.html#pyspark.sql.Column) which is a substring of the column. | | [**when**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.when.html#pyspark.sql.Column.when)(condition, value) | Evaluates a list of conditions and returns one of multiple possible result expressions. | | [**withField**](https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.Column.withField.html#pyspark.sql.Column.withField)(fieldName, col) | An expression that adds/replaces a field in **StructType** by name. | |

|  |
| --- |
| pyspark.sql.Column.alias (\*alias, \*\*kwargs)  Returns this column aliased with a new name or names  >>>df.select(df.age.alias("age2")).collect()  [Row(age2=2), Row(age2=5)]  >>>df.select(df.age.alias("age3", metadata={'max': 99})).schema['age3'].metadata['max']  99 |

|  |
| --- |
| pyspark.sql.Column.asc()  Returns a sort expression based on ascending order of the column.  from pyspark.sql import Row  df = spark.createDataFrame([('Tom', 80), ('Alice', None)], ["name", "height"])  df.select(df.name).orderBy(df.name.asc()).collect() |

|  |
| --- |
| pyspark.sql.Column.asc\_nulls\_first()  Returns a sort expression based on ascending order of the column, and null values return before non-null values. |

|  |
| --- |
| pyspark.sql.Column.asc\_nulls\_last()  Returns a sort expression based on ascending order of the column, and null values appear after non-null values. |

|  |
| --- |
| pyspark.sql.Column.astype(datatype)  astype() is an alias for cast(). |

|  |
| --- |
| pyspark.sql.Column.between (lowerBound, upperBound)  True if the current column is between the lower bound and upper bound, inclusive.  df.select(df.name, df.age.between(2, 4)).show() |

|  |
| --- |
| pyspark.sql.Column.bitwiseAND(other) |

|  |
| --- |
| pyspark.sql.Column.bitwiseOR(other) |

|  |
| --- |
| pyspark.sql.Column.bitwiseXOR(other) |

|  |
| --- |
| pyspark.sql.Column.cast(datatype)  Casts the column into type dataType.  df.select(df.age.cast("string").alias('ages')).collect()  df.select(df.age.cast(StringType()).alias('ages')).collect() |

|  |
| --- |
| pyspark.sql.Column.contains (other)  Contains the other element. Returns a boolean Column based on a string match.  >>>df.filter(df.name.contains('obb')).collect()  [Row(age=5, name='Bobby')] |

|  |
| --- |
| pyspark.sql.Column.desc()  Returns a sort expression based on the descending order of the column.  from pyspark.sql import Row  df = spark.createDataFrame([('Tom', 80), ('Alice', None)], ["name", "height"])  df.select(df.name).orderBy(df.name.desc()).collect() |

|  |
| --- |
| pyspark.sql.Column.desc\_nulls\_first()  Returns a sort expression based on the descending order of the column, and null values appear before non-null values. |

|  |
| --- |
| pyspark.sql.Column.desc\_nulls\_last()  Returns a sort expression based on the descending order of the column, and null values appear after non-null values. |

|  |
| --- |
| pyspark.sql.Column.dropFields(\*fieldNames)  An expression that drops fields in StructType by name. This is a no-op if schema doesn’t contain field name(s).  from pyspark.sql import Row  from pyspark.sql.functions import col, lit  df = spark.createDataFrame([  Row(a=Row(b=1, c=2, d=3, e=Row(f=4, g=5, h=6)))])  df.withColumn('a', df['a'].dropFields('b')).show()  df.withColumn('a', df['a'].dropFields('b', 'c')).show() |

|  |
| --- |
| pyspark.sql.Column.endswith(other)  String ends with. Returns a boolean Column based on a string match.  df.filter(df.name.endswith('ice')).collect()  df.filter(df.name.endswith('ice$')).collect() |

|  |
| --- |
| pyspark.sql.Column.eqNullSafe(other)  Equality test that is safe for null values.  from pyspark.sql import Row  df1 = spark.createDataFrame([  Row(id=1, value='foo'),  Row(id=2, value=None)  ])  df1.select(  df1['value'] == 'foo',  df1['value'].eqNullSafe('foo'),  df1['value'].eqNullSafe(None)  ).show() |

|  |
| --- |
| pyspark.sql.Column.getField(name)  An expression that gets a field by name in a StructType.  from pyspark.sql import Row  df = spark.createDataFrame([Row(r=Row(a=1, b="b"))])  df.select(df.r.getField("b")).show()  +---+  |r.b|  +---+  | b|  +---+ |

|  |
| --- |
| pyspark.sql.Column.getItem(key)  An expression that gets an item at position ordinal out of a list, or gets an item by key out of a dict.  df = spark.createDataFrame([([1, 2], {"key": "value"})], ["l", "d"])  df.select(df.l.getItem(0), df.d.getItem("key")).show() |

|  |
| --- |
| pyspark.sql.Column.isNotNull()  True if the current expression is NOT null.  from pyspark.sql import Row  df = spark.createDataFrame([Row(name='Tom', height=80), Row(name='Alice', height=None)])  df.filter(df.height.isNotNull()).collect() |

|  |
| --- |
| pyspark.sql.Column.isNull()  True if the current expression is null. |

|  |
| --- |
| pyspark.sql.Column.isin(cols)  A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of the arguments.  >>>df[df.name.isin("Bob", "Mike")].collect()  [Row(age=5, name='Bob')]  >>>df[df.age.isin([1, 2, 3])].collect()  [Row(age=2, name='Alice')] |

|  |
| --- |
| pyspark.sql.Column.like(other)  SQL like expression. Returns a boolean Column based on a SQL LIKE match.  df.filter(df.name.like('Al%')).collect() |

|  |
| --- |
| pyspark.sql.Column.name(\*alias, \*\*kwargs)  name() is an alias for alias(). |

|  |
| --- |
| pyspark.sql.Column.otherwise(value)  Evaluates a list of conditions and returns one of multiple possible result expressions. If Column.otherwise() is not invoked, None is returned for unmatched conditions.  from pyspark.sql import functions as F  df.select(df.name, F.when(df.age > 3, 1).otherwise(0)).show() |

|  |
| --- |
| pyspark.sql.Column.over(window)  Define a windowing column. |

|  |
| --- |
| pyspark.sql.Column.rlike(other)  SQL RLIKE expression (LIKE with Regex). Returns a boolean Column based on a regex match.  df.filter(df.name.rlike('ice$')).collect() |

|  |
| --- |
| pyspark.sql.Column.startswith(other)  String starts with. Returns a boolean Column based on a string match.  >>>df.filter(df.name.startswith('Al')).collect()  [Row(age=2, name='Alice')]  >>>df.filter(df.name.startswith('^Al')).collect()  [] |

|  |
| --- |
| pyspark.sql.Column.substr(startPos, length)  Return a Column which is a substring of the column.  >>>df.select(df.name.substr(1, 3).alias("col")).collect()  [Row(col='Ali'), Row(col='Bob')] |

|  |
| --- |
| pyspark.sql.Column.when(condition, value)  Evaluates a list of conditions and returns one of multiple possible result expressions. If Column.otherwise() is not invoked, None is returned for unmatched conditions.  from pyspark.sql import functions as F  df.select(df.name, F.when(df.age > 4, 1).when(df.age < 3, -1).otherwise(0)).show() |

|  |
| --- |
| pyspark.sql.Column.withField(fieldName, col)  An expression that adds/replaces a field in StructType by name.  from pyspark.sql import Row  from pyspark.sql.functions import lit  df = spark.createDataFrame([Row(a=Row(b=1, c=2))])  df.withColumn('a', df['a'].withField('b', lit(3))).select('a.b').show()  +---+  | b|  +---+  | 3|  +---+  df.withColumn('a', df['a'].withField('d', lit(4))).select('a.d').show()  +---+  | d|  +---+  | 4|  +---+ |