

# Cheat Sheet for PySpark

Wenqiang Feng

E-mail: [von198@gmail.com](mailto:von198@gmail.com), Web: <http://web.utk.edu/~wfeng1>; <https://runawayhorse001.github.io/LearningApacheSpark>

## Spark Configuration

```
from pyspark.sql import SparkSession
spark = SparkSession.builder
    .appName("Python Spark regression example")
    .config("config.option", "value").getOrCreate()
```

## Loading Data

### From RDDs

```
# Using parallelize()
df = spark.sparkContext.parallelize([(('1', 'Joe', '70000', '1'),
    ('2', 'Henry', '80000', None))]
    .toDF(['Id', 'Name', 'Salary', 'DepartmentId'])
# Using createDataFrame()
df = spark.createDataFrame([(('1', 'Joe', '70000', '1'),
    ('2', 'Henry', '80000', None))]
    ['Id', 'Name', 'Salary', 'DepartmentId'])
```

Id	Name	Salary	DepartmentId
1	Joe	70000	1
2	Henry	80000	null

### From Data Sources

#### ▷ From .csv

```
df = spark.read.format('com.databricks.spark.csv').
    options(header='true', inferschema='true').
    load("Advertising.csv", header=True);
```

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4

#### ▷ From .json

```
df = spark.read.json('/home/feng/Desktop/data.json')
```

id	location	timestamp
29572562021	[72.1, DE, 8086, 52...]	[2019-02-23 22:36:52]
29572562031	[598.5, BG, 3963, 42...]	[2019-02-23 22:36:52]

#### ▷ From Database

```
user = 'username'; pw = 'password'
table_name = 'table_name'
url = 'jdbc:postgresql://##.###.###.###:5432/dataset?user='
    +user+'&password='+pw
p = 'driver': 'org.postgresql.Driver', 'password': pw, 'user': user
df = spark.read.jdbc(url=url, table=table_name, properties=p)
```

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4

#### ▷ From HDFS

```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext
sc = SparkContext('local', 'example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://###/user/data/file_name")
```

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4

## Auditing Data

### Checking schema

```
df.printSchema()
root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

### Checking missing value

```
from pyspark.sql.functions import count
def my_count(df):
    df.agg([count(c).alias(c) for c in df_in.columns]).show()
my_count(df_raw)
```

InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
541909	541909	541909	541909	541909	406829	541909

### Checking statistical results

```
# function form my pyspark
df_raw.describe().show()
```

summary	TV	Radio	Newspaper
count	200	200	200
mean	147.0425	23.264000000000002	30.553999999999996
stddev	85.85423631490805	14.846809176168728	21.77862083852283
min	0.7	0.0	0.3
max	296.4	49.6	114.0

## Manipulating Data (More details on next page)

### Fixing missing value

Function	Description
----------	-------------

df.na.fill() #Replace null values

df.na.drop() #Dropping any rows with null values.

### Joining data

Description	Function
-------------	----------

#Data join left.join(right, key, how='\*') \* = left, right, inner, full

### Wrangling with UDF

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
# user defined function
def complexFun(x):
    return results
Fn = F.udf(lambda x: complexFun(x), DoubleType())
df.withColumn('2col', Fn(df.col))
```

### Reducing features

df.select(featureNameList)

## Modeling Pipeline

### Deal with categorical feature and label data

```
# Deal with categorical feature data
from pyspark.ml.feature import VectorIndexer
featureIndexer = VectorIndexer(inputCol="features",
    outputCol="indexedFeatures",
    maxCategories=4).fit(data)
featureIndexer.transform(data).show(2, True)
```

features	label	indexedFeatures
[29, [1, 11, 14, 16, 1...]]	no	[29, [1, 11, 14, 16, 1...]]

```
# Deal with categorical label data
labelIndexer = StringIndexer(inputCol='label',
    outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(2, True)
```

features	label	indexedLabel
[29, [1, 11, 14, 16, 1...]]	no	0.0

### Splitting the data to training and test data sets

(trainingData, testData) = data.randomSplit([0.6, 0.4])

### Importing the model

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='indexedFeatures',
    labelCol='indexedLabel')
```

### Converting indexed labels back to original labels

```
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction",
    outputCol="predictedLabel",
    labels=labelIndexer.labels)
```

### Wrapping Pipeline

pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
 lr, labelConverter])

### Training model and making predictions

```
model = pipeline.fit(trainingData)
predictions = model.transform(testData)
predictions.select("features", "label", "predictedLabel").show(2)
```

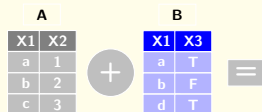
features	label	predictedLabel
[29, [0, 11, 13, 16, 1...]]	no	no

### Evaluating

```
from pyspark.ml.evaluation import *
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel",
    predictionCol="prediction", metricName="accuracy")
accu = evaluator.evaluate(predictions)
print("Test Error: %g, AUC: %g"%(1-accu, Summary.areaUnderROC))
Test Error: 0.0986395, AUC: 0.886664269877
```

## Data Wrangling: Combining DataFrame

### Mutating Joins



#### Result

X1	X2	X3
a	1	T
b	2	F
c	3	T

#### Function

```
#Join matching rows from B to A
#dplyr::left_join(A, B, by = "x1")
A.join(B,'X1',how='left')
.orderBy('X1', ascending=True).show()
```

X1	X2	X3
a	1	T
b	2	F
c	3	T

```
#Join matching rows from A to B
#dplyr::right_join(A, B, by = "x1")
A.join(B,'X1',how='right')
.orderBy('X1', ascending=True).show()
```

X1	X2	X3
a	1	T
b	2	F

```
#Retain only rows in both sets
#dplyr::inner_join(A, B, by = "x1")
A.join(B,'X1',how='inner')
.orderBy('X1', ascending=True).show()
```

X1	X2	X3
a	1	T
b	2	F
c	3	null
d	null	T

```
#Retain all values,all rows
#dplyr::full_join(A, B, by = "x1")
A.join(B,'X1',how='full')
.orderBy('X1', ascending=True).show()
```

### Filtering Joins

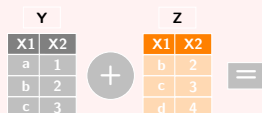
X1	X2
a	1
b	2

```
#All rows in A that have a match in B
#dplyr::semi_join(A, B, by = "x1")
a.join(b,'X1',how='left_semi')
.orderBy('X1', ascending=True).show()
```

X1	X2
C	3

```
#All rows in A, don't have a match in B
#dplyr::anti_join(A, B, by = "x1")
A.join(B,'X1',how='left_anti')
.orderBy('X1', ascending=True).show()
```

## DataFrame Operations



#### Result

X1	X2
b	2
c	3

#### Function

```
#Rows that appear in both Y and Z
#dplyr::intersect(Y, Z)
Y.intersect(Z).show()
```

X1	X2
a	1
b	2
c	3
d	4

```
#Rows that appear in either or both Y and Z
#dplyr::union(Y, Z)
Y.union(Z).dropDuplicates()
.orderBy('X1', ascending=True).show()
```

X1	X2
a	1

```
#Rows that appear in Y but not Z
#dplyr::setdiff(Y, Z)
Y.subtract(Z).show()
```

### Binding

X1	X2
a	1
b	2
c	3

```
#Append Z to Y as new rows
#dplyr::bind_rows(Y, Z)
Y.union(Z)
.orderBy('X1', ascending=True).show()
```

X1	X2	X1	X2
a	1	b	2
b	2	c	3
c	3	d	4

```
#Append Z to Y as new columns
#Caution: zipDataFrames form my package
#dplyr::bind_cols(Y, Z)
zipDataFrames(Y,Z).show()
```

## Data Wrangling: Reshaping Data

### Splitting

#### Change

key	value
a	1,2,3
b	2,3,4

#### Function

```
#ArrayType() tidy::separate ::separate one column into several
df.select("key", df.value[0], df.value[1], df.value[2]).show()
#StructType()
df2.select('key', 'value.*').show()
```

key	value
a	1,2,3
b	2,3,4

```
#Splitting one column into rows
df.select("key",F.split("values", ",").alias("values"),
F.posexplode(F.split("values",","),").alias("pos", "val")
).drop("val")
.select("key",F.expr("values[pos]").alias("val")).show()
```

key	value
a	1
a	2
a	3
b	2
b	3
b	4

```
#Gather columns into rows
def to_long(df, by):
    cols, dtypes = zip(*(c,t) for (c, t) in df.dtypes if c not in by))
    # Spark SQL supports only homogeneous columns
    assert len(set(dtypes))==1,"all columns have to be of the same type"
    # Create and explode an array of (column_name, column_value) structs
    kvs = explode(array(
        struct(lit(c).alias("key"), col(c).alias("val")) for c in cols
    )).alias("kvs")
    return df.select(by + [kvs]).select(by + ["kvs.key", "kvs.val"])
```

### Pivot

key	value
a	1
a	2
a	3
b	2
b	3
b	4

```
#Spread rows into columns
df.groupBy(['key'])
.pivot('col1').sum('col1').show()
```

### Subset Observations (Rows)



#### Function

df.na.drop()

#### Description

#Omitting rows with null values

df.where()

#Filters rows using the given condition

df.filter()

#Filters rows using the given condition

df.distinct()

#Returns distinct rows in this DataFrame

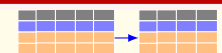
df.sample()

#Returns a sampled subset of this DataFrame

df.sampleBy()

#Returns a stratified sample without replacement

### Subset Variables (Columns)



#### Function

df.select()

#### Description

#Applies expressions and returns a new DataFrame

### Make New Variables



#### Function

df.withColumn() df.withColumn('new',1/df.col)

#### Examples

```
df.withColumn('new',F.log(df.col))
df.withColumn('id', psf.monotonically_increasing_id())
df.withColumn("new", Fn('col')) #Fn:F.udf()
df.withColumn('new', F.when((df.c1>1)&(df.c2<2),1)
.when((df.c3>3),2).otherwise(3))
```

## Data Wrangling: Reshaping Data

### Summarise Data



#### Function

df.describe()

#### Description

#Computes simple statistics

Correlation.corr(df)

#Computes the correlation matrix

df.count()

#Count the number of rows



#### Description

#Sum

#### Demo

df.agg(F.max(df.c)).head()[0]#Similar for: F.min,max,avg,stddev

### Group Data



A	min_b	max_b	avg_c
m	1	2	4.5
n	3	4	7.5

```
df.groupBy(['A'])
.agg(F.min('B').alias('min_b'),
F.max('B').alias('max_b'),
F.avg('C').alias('avg_c')).show()
```

A	min_b	max_b	list_c
m	1	2	[2,4,3,4,7]
n	3	4	[7,3,5,7,7]

```
def quant_pd(val_list):
    quant = np.round(np.percentile(val_list,
[20,50,75]),2)
    return list(map(float,quant))
Fn = F.udf(quant_pd,ArrayType(FloatType()))
#GroupBy and aggregate
df.groupBy(['A'])
.agg(F.min('B').alias('min_b'),
F.max('B').alias('max_b'),
Fn(F.collect_list(col('C'))).alias('list_c'))
```

### Windows



#### Result

A	B	C	D
a	m	1	0
b	m	2	1
c	n	3	0
d	n	6	3

```
from pyspark.sql import Window
#Define windows for difference
w = Window.partitionBy(df.B)
D = df.C - F.max(df.C).over(w)
df.withColumn('D',D).show()
```

A	B	C	D
a	m	1	1
b	m	2	2
c	n	3	3
d	n	6	4

```
df = df.withColumn("D",
F.monotonically_increasing_id())
#Define windows for row_num
w = Window.orderBy("D")
df.withColumn("D", F.row_number().over(w))
```

A	B	C	D
a	m	2	1
a	m	1	2
d	n	6	1
c	n	3	2

```
#Define windows for rank
w = Window.partitionBy('B')
.orderBy(df.C.desc())
df.withColumn("D",rank().over(w)).show()
```

### Rename Variables



#### Function

df.withColumnRenamed()#Renaming an existing column