
Data Science with PySpark

David Kearney

Aug 22, 2020

CONTENTS

1	RDDs and Schemas and Data Types with Pyspark	3
2	Window functions and Pivot Tables with Pyspark	5
3	Dataframe Filitering and Operations with Pyspark	9
4	Linear Regression and Random Forest/GBT Classification with Pyspark	11
5	Pyspark Regression with Fiscal Data	15
5.1	Bring in needed imports	15
5.2	Describing the Data	16
6	Dataframes, Formatting, Casting Data Type and Correlation with Pyspark	19
7	Handling Missing Data with Pyspark	21
8	Group By and Aggregation with Pyspark	23

Data Science with PySpark, written by [David R. Kearney](#).

Note: Data Science with PySpark

RDDS AND SCHEMAS AND DATA TYPES WITH PYSPARK

```
from pyspark.sql import SparkSession

# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.show()
```

```
df.printSchema()
```

```
df.columns
```

```
df.describe()
```

```
from pyspark.sql.types import StructField, StringType, IntegerType, StructType
```

```
data_schema = [
    StructField("_c0", IntegerType(), True)
    ,StructField("province", StringType(), True)
    ,StructField("specific", IntegerType(), True)
    ,StructField("general", IntegerType(), True)
    ,StructField("year", IntegerType(), True)
    ,StructField("gdp", IntegerType(), True)
    ,StructField("fdi", IntegerType(), True)
    ,StructField("rnr", IntegerType(), True)
    ,StructField("rr", IntegerType(), True)
    ,StructField("i", IntegerType(), True)
    ,StructField("fr", IntegerType(), True)
    ,StructField("reg", StringType(), True)
    ,StructField("it", IntegerType(), True)
]
```

```
final_struct = StructType(fields=data_schema)
```

```
df = spark.read.format("CSV").schema(final_struct).load(file_location)
```

```
df.printSchema()
```

```
df.show()
```

```
df['fr']
```

```
type(df['fr'])
```

```
df.select('fr')
```

```
type(df.select('fr'))
```

```
df.select('fr').show()
```

```
df.head(2)
```

```
df.select(['reg', 'fr'])
```

```
df.select(['reg', 'fr']).show()
```

```
df.withColumn('fiscal_revenue', df['fr']).show()
```

```
df.show()
```

```
df.withColumnRenamed('fr', 'new_fiscal_revenue').show()
```

```
df.withColumn('double_fiscal_revenue', df['fr']*2).show()
```

```
df.withColumn('add_fiscal_revenue', df['fr']+1).show()
```

```
df.withColumn('half_fiscal_revenue', df['fr']/2).show()
```

```
df.withColumn('half_fr', df['fr']/2)
```

```
df.createOrReplaceTempView("economic_data")
```

```
sql_results = spark.sql("SELECT * FROM economic_data")
```

```
sql_results
```

```
sql_results.show()
```

```
spark.sql("SELECT * FROM economic_data WHERE fr=634562").show()
```


WINDOW FUNCTIONS AND PIVOT TABLES WITH PYSPARK

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField,StringType,IntegerType,StructType,␣
↳DoubleType, FloatType
from pyspark.sql.functions import *

data_schema = [
    StructField("_c0", IntegerType(), True)
    ,StructField("province", StringType(), True)
    ,StructField("specific", DoubleType(), True)
    ,StructField("general", DoubleType(), True)
    ,StructField("year", IntegerType(), True)
    ,StructField("gdp", FloatType(), True)
    ,StructField("fdi", FloatType(), True)
    ,StructField("rnr", DoubleType(), True)
    ,StructField("rr", FloatType(), True)
    ,StructField("i", FloatType(), True)
    ,StructField("fr", IntegerType(), True)
    ,StructField("reg", StringType(), True)
    ,StructField("it", IntegerType(), True)
]

final_struc = StructType(fields=data_schema)

file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").schema(final_struc).option("header", True).load(file_
↳location)

#df.printSchema()

df.show()
```

```
df.limit(10).toPandas()
```

```
df = df.withColumnRenamed("reg","region")
```

```
df.limit(10).toPandas()
```

```
# df = df.toDF(*['year', 'region', 'province', 'gdp', 'fdi', 'specific', 'general',
↳'it', 'fr', 'rnr', 'rr', 'i', '_c0', 'specific_classification', 'provinceIndex',
↳'regionIndex'])
```

```
df = df.select('year','region','province','gdp', 'fdi')
```

```
df.sort("gdp").show()
```

```
from pyspark.sql import functions as F
df.sort(F.desc("gdp")).show()
```

```
from pyspark.sql.types import IntegerType, StringType, DoubleType
df = df.withColumn('gdp', F.col('gdp').cast(DoubleType()))
```

```
df = df.withColumn('province', F.col('province').cast(StringType()))
```

```
df.filter((df.gdp>10000) & (df.region=='East China')).show()
```

```
from pyspark.sql import functions as F
df.groupBy(["region", "province"]).agg(F.sum("gdp"), F.max("gdp")).show()
```

```
df.groupBy(["region", "province"]).agg(F.sum("gdp").alias("SumGDP"), F.max("gdp").alias(
    ↪ "MaxGDP")).show()
```

```
df.groupBy(["region", "province"]).agg(
    F.sum("gdp").alias("SumGDP"), \
    F.max("gdp").alias("MaxGDP") \
).show()
```

```
df.limit(10).toPandas()
```

```
casesWithNewConfirmed = cases.withColumn("NewConfirmed", 100 + F.col("confirmed"))
casesWithNewConfirmed.show()
```

```
df = df.withColumn("Exp_GDP", F.exp("gdp"))
df.show()
```

Note: Window functions

```
# Window functions
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy(F.desc('gdp'))
df.withColumn("rank", F.rank().over(windowSpec)).show()
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy('year')
```

```
dfWithLag = df.withColumn("lag_7", F.lag("gdp", 7).over(windowSpec))
```

```
df.filter(df.year>'2000').show()
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy('year').rowsBetween(-6, 0)
```

```
dfWithRoll = df.withColumn("roll_7_confirmed", F.mean("gdp").over(windowSpec))
```

```
dfWithRoll.filter(dfWithLag.year>'2001').show()
```

```
from pyspark.sql.window import Window
windowSpec = Window().partitionBy(['province']).orderBy('year').rowsBetween(Window.
↳ unboundedPreceding, Window.currentRow)
```

```
dfWithRoll = df.withColumn("cumulative_gdp", F.sum("gdp").over(windowSpec))
```

```
dfWithRoll.filter(dfWithLag.year>'1999').show()
```

Note: Pivot Dataframes

```
pivoted_df = df.groupBy('year').pivot('province') \
    .agg(F.sum('gdp').alias('gdp') , F.sum('fdi').alias('fdi'))
pivoted_df.limit(10).toPandas()
```

```
pivoted_df.columns
```

```
newColnames = [x.replace("-", "_") for x in pivoted_df.columns]
```

```
pivoted_df = pivoted_df.toDF(*newColnames)
```

```
expression = ""
cnt=0
for column in pivoted_df.columns:
    if column!='year':
        cnt +=1
        expression += f'{column}' , {column}, "
expression = f"stack({cnt}, {expression[:-1]}) as (Type,Value)"
```

```
unpivoted_df = pivoted_df.select('year', F.expr(expression))
unpivoted_df.show()
```


DATAFRAME FILTERING AND OPERATIONS WITH PYSPARK

```
from pyspark.sql import SparkSession

# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.filter("specific<10000").show()
```

```
df.filter("specific<10000").select('province').show()
```

```
df.filter("specific<10000").select(['province','year']).show()
```

```
df.filter(df["specific"] < 10000).show()
```

```
df.filter((df["specific"] < 55000) & (df['gdp'] > 200) ).show()
```

```
df.filter((df["specific"] < 55000) | (df['gdp'] > 20000) ).show()
```

```
df.filter((df["specific"] < 55000) & ~(df['gdp'] > 20000) ).show()
```

```
df.filter(df["specific"] == 8964.0).show()
```

```
df.filter(df["province"] == "Zhejiang").show()
```

```
df.filter(df["specific"] == 8964.0).collect()
```

```
result = df.filter(df["specific"] == 8964.0).collect()
```

```
type(result[0])
```

```
row = result[0]
```

```
row.asDict()
```

```
for item in result[0]:
    print(item)
```


LINEAR REGRESSION AND RANDOM FOREST/GBT CLASSIFICATION WITH PYSPARK

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField,StringType,IntegerType,StructType,
↳DoubleType, FloatType
from pyspark.sql.functions import *

data_schema = [
StructField("_c0", IntegerType(), True)
,StructField("province", StringType(), True)
,StructField("specific", DoubleType(), True)
,StructField("general", DoubleType(), True)
,StructField("year", IntegerType(), True)
,StructField("gdp", FloatType(), True)
,StructField("fdi", FloatType(), True)
,StructField("rnr", DoubleType(), True)
,StructField("rr", FloatType(), True)
,StructField("i", FloatType(), True)
,StructField("fr", IntegerType(), True)
,StructField("reg", StringType(), True)
,StructField("it", IntegerType(), True)
]

final_struct = StructType(fields=data_schema)

file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").schema(final_struct).option("header", True).load(file_
↳location)

#df.printSchema()

df.show()
```

```
df.groupBy('province').count().show()
```

```
mean_val = df.select(mean(df['general'])).collect()
mean_val[0][0]
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["general"])
```

```
mean_val = df.select(mean(df['specific'])).collect()
mean_val[0][0]
```

(continues on next page)

(continued from previous page)

```
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["specific"])
```

```
mean_val = df.select(mean(df['rr'])).collect()
mean_val[0][0]
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["rr"])
```

```
mean_val = df.select(mean(df['fr'])).collect()
mean_val[0][0]
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["fr"])
```

```
mean_val = df.select(mean(df['rnr'])).collect()
mean_val[0][0]
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["rnr"])
```

```
mean_val = df.select(mean(df['i'])).collect()
mean_val[0][0]
mean_gen = mean_val[0][0]
df = df.na.fill(mean_gen, ["i"])
```

```
from pyspark.sql.functions import *
df = df.withColumn('specific_classification', when(df.specific >= 583470.7303370787, 1) .
    → otherwise(0))
```

```
from pyspark.ml.feature import StringIndexer
```

```
indexer = StringIndexer(inputCol="province", outputCol="provinceIndex")
df = indexer.fit(df).transform(df)
```

```
indexer = StringIndexer(inputCol="reg", outputCol="regionIndex")
df = indexer.fit(df).transform(df)
```

```
df.show()
```

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

```
df.columns
```

```
assembler = VectorAssembler(
    inputCols=[
        'provinceIndex',
        # 'specific',
        'general',
        'year',
        'gdp',
        'fdi',
        # 'rnr',
        # 'rr',
```

(continues on next page)

(continued from previous page)

```
#'i',
#'fr',
'regionIndex',
'it'
],
outputCol="features")
```

```
output = assembler.transform(df)
```

```
final_data = output.select("features", "specific")
```

```
train_data, test_data = final_data.randomSplit([0.7, 0.3])
```

```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(labelCol='specific')
```

```
lrModel = lr.fit(train_data)
```

```
print("Coefficients: {} Intercept: {}".format(lrModel.coefficients, lrModel.intercept))
```

```
test_results = lrModel.evaluate(test_data)
```

```
print("RMSE: {}".format(test_results.rootMeanSquaredError))
print("MSE: {}".format(test_results.meanSquaredError))
print("R2: {}".format(test_results.r2))
```

```
from pyspark.sql.functions import corr
```

```
df.select(corr('specific', 'gdp')).show()
```

```
from pyspark.ml.classification import DecisionTreeClassifier, GBTClassifier,
↳ RandomForestClassifier
from pyspark.ml import Pipeline
```

```
dtc = DecisionTreeClassifier(labelCol='specific_classification', featuresCol='features'
↳)
rfc = RandomForestClassifier(labelCol='specific_classification', featuresCol='features'
↳)
gbt = GBTClassifier(labelCol='specific_classification', featuresCol='features')
```

```
final_data = output.select("features", "specific_classification")
train_data, test_data = final_data.randomSplit([0.7, 0.3])
```

```
rfc_model = rfc.fit(train_data)
gbt_model = gbt.fit(train_data)
dtc_model = dtc.fit(train_data)
```

```
dtc_predictions = dtc_model.transform(test_data)
rfc_predictions = rfc_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
acc_evaluator = MulticlassClassificationEvaluator(labelCol="specific_classification",  
↳ predictionCol="prediction", metricName="accuracy")
```

```
dtc_acc = acc_evaluator.evaluate(dtc_predictions)  
rfc_acc = acc_evaluator.evaluate(rfc_predictions)  
gbt_acc = acc_evaluator.evaluate(gbt_predictions)
```

```
print('-'*80)  
print('Decision tree accuracy: {0:2.2f}%'.format(dtc_acc*100))  
print('-'*80)  
print('Random forest ensemble accuracy: {0:2.2f}%'.format(rfc_acc*100))  
print('-'*80)  
print('GBT accuracy: {0:2.2f}%'.format(gbt_acc*100))  
print('-'*80)
```

```
df.select(corr('specific_classification', 'fdi')).show()
```

```
df.select(corr('specific_classification', 'gdp')).show()
```

PYSPARK REGRESSION WITH FISCAL DATA

5.1 Bring in needed imports

“A minimal example of using Pyspark for Linear Regression”

- toc: true- branch: master- badges: true
- comments: true
- author: David Kearney
- categories: [pyspark, jupyter]
- description: A minimal example of using Pyspark for Linear Regression
- title: Pyspark Regression with Fiscal Data

```
from pyspark.sql.functions import col
from pyspark.sql.types import StringType, BooleanType, DateType, IntegerType
from pyspark.sql.functions import *
```

```
## Load data from a CSV
```

```
#collapse-hide
```

```
# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.createOrReplaceTempView("fiscal_stats")

sums = spark.sql("""
select year, sum(it) as total_yearly_it, sum(fr) as total_yearly_fr
from fiscal_stats
group by 1
order by year asc
""")

sums.show()
```

5.2 Describing the Data

```
df.describe().toPandas().transpose()
```

```
df2 = df.withColumn("gdp", col("gdp").cast(IntegerType())) \
.withColumn("specific", col("specific").cast(IntegerType())) \
.withColumn("general", col("general").cast(IntegerType())) \
.withColumn("year", col("year").cast(IntegerType())) \
.withColumn("fdi", col("fdi").cast(IntegerType())) \
.withColumn("rnr", col("rnr").cast(IntegerType())) \
.withColumn("rr", col("rr").cast(IntegerType())) \
.withColumn("i", col("i").cast(IntegerType())) \
.withColumn("fr", col("fr").cast(IntegerType()))
```

```
df2.printSchema()
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

assembler = VectorAssembler(inputCols=['gdp', 'fdi'], outputCol="features")
train_df = assembler.transform(df2)
```

```
train_df.select("specific", "year").show()
```

```
lr = LinearRegression(featuresCol = 'features', labelCol='it')
lr_model = lr.fit(train_df)

trainingSummary = lr_model.summary
print("Coefficients: " + str(lr_model.coefficients))
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("R2: %f" % trainingSummary.r2)
```

```
lr_predictions = lr_model.transform(train_df)
lr_predictions.select("prediction", "it", "features").show(5)
from pyspark.ml.evaluation import RegressionEvaluator
lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                                   labelCol="it", metricName="r2")
```

```
print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(lr_predictions))
```

```
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
```

```
predictions = lr_model.transform(test_df)
predictions.select("prediction", "it", "features").show()
```

```
from pyspark.ml.regression import DecisionTreeRegressor
dt = DecisionTreeRegressor(featuresCol = 'features', labelCol = 'it')
dt_model = dt.fit(train_df)
dt_predictions = dt_model.transform(train_df)
dt_evaluator = RegressionEvaluator(
```

(continues on next page)

(continued from previous page)

```
labelCol="it", predictionCol="prediction", metricName="rmse")
rmse = dt_evaluator.evaluate(dt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
from pyspark.ml.regression import GBRegressor
gbt = GBRegressor(featuresCol = 'features', labelCol = 'it', maxIter=10)
gbt_model = gbt.fit(train_df)
gbt_predictions = gbt_model.transform(train_df)
gbt_predictions.select('prediction', 'it', 'features').show(5)

gbt_evaluator = RegressionEvaluator(
    labelCol="it", predictionCol="prediction", metricName="rmse")
rmse = gbt_evaluator.evaluate(gbt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```


DATAFRAMES, FORMATTING, CASTING DATA TYPE AND CORRELATION WITH PYSPARK

```
from pyspark.sql import SparkSession

# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.columns
```

```
df.printSchema()
```

```
# for row in df.head(5):
#     print(row)
#     print('\n')
```

```
df.describe().show()
```

```
df.describe().printSchema()
```

```
from pyspark.sql.functions import format_number
```

```
result = df.describe()
result.select(result['province']
,format_number(result['specific'].cast('float'),2).alias('specific')
,format_number(result['general'].cast('float'),2).alias('general')
,format_number(result['year'].cast('int'),2).alias('year'),format_number(result['gdp
    ↪'].cast('float'),2).alias('gdp')
,format_number(result['rnr'].cast('int'),2).alias('rnr'),format_number(result['rr']
    ↪.cast('float'),2).alias('rr')
,format_number(result['fdi'].cast('int'),2).alias('fdi'),format_number(result['it']
    ↪.cast('float'),2).alias('it')
,result['reg'].cast('string').alias('reg')
    ).show()
```

```
df2 = df.withColumn("specific_gdp_ratio",df["specific"]/(df["gdp"]*100))#.show()
```

```
df2.select('specific_gdp_ratio').show()
```

```
df.orderBy(df["specific"].asc()).head(1)[0][0]
```

```
from pyspark.sql.functions import mean
df.select(mean("specific")).show()
```

```
from pyspark.sql.functions import max,min
```

```
df.select(max("specific"),min("specific")).show()
```

```
df.filter("specific < 60000").count()
```

```
df.filter(df['specific'] < 60000).count()
```

```
from pyspark.sql.functions import count
result = df.filter(df['specific'] < 60000)
result.select(count('specific')).show()
```

```
(df.filter(df["gdp"]>8000).count()*1.0/df.count())*100
```

```
from pyspark.sql.functions import corr
df.select(corr("gdp","fdi")).show()
```

```
from pyspark.sql.functions import year
#yeardf = df.withColumn("Year",year(df["year"]))
```

```
max_df = df.groupBy('year').max()
```

```
max_df.select('year','max(gdp)').show()
```

```
from pyspark.sql.functions import month
```

```
#df.select("year","avg(gdp)").orderBy('year').show()
```


HANDLING MISSING DATA WITH PYSPARK

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import countDistinct, avg, stddev

# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.show()
```

```
# Has to have at least 2 NON-null values
df.na.drop(thresh=2).show()
```

```
# Drop any row that contains missing data
df.na.drop().show()
```

```
df.na.drop(subset=["general"]).show()
```

```
df.na.drop(how='any').show()
```

```
df.na.drop(how='all').show()
```

```
df.na.fill('example').show()
```

```
df.na.fill(0).show()
```

```
df.na.fill('example', subset=['fr']).show()
```

```
df.na.fill(0, subset=['general']).show()
```

```
# Mean Imputation
from pyspark.sql.functions import mean
mean_val = df.select(mean(df['general'])).collect()
```

```
mean_val[0][0]
```

```
mean_gen = mean_val[0][0]
```

```
df.na.fill(mean_gen, ["general"]).show()
```

```
df.na.fill(df.select(mean(df['general'])).collect()[0][0], ['general']).show()
```

GROUP BY AND AGGREGATION WITH PYSPARK

“Group By and Aggregation with Pyspark”

- toc: true- branch: master- badges: true
- comments: true
- author: David Kearney
- categories: [pyspark, jupyter]
- description: Group By and Aggregation with Pyspark
- title: Group By and Aggregation with Pyspark

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import countDistinct, avg, stddev

# Load data from a CSV
file_location = "/FileStore/tables/df_panel_fix.csv"
df = spark.read.format("CSV").option("inferSchema", True).option("header", True).
    ↪load(file_location)
display(df.take(5))
```

```
df.printSchema()
```

```
df.groupBy("province")
```

```
df.groupBy("province").mean().show()
```

```
df.groupBy("reg").mean().show()
```

```
# Count
df.groupBy("reg").count().show()
```

```
# Max
df.groupBy("reg").max().show()
```

```
# Min
df.groupBy("reg").min().show()
```

```
# Sum
df.groupBy("reg").sum().show()
```

```
# Max it across everything
df.agg({'specific': 'max'}).show()
```

```
grouped = df.groupBy("reg")
grouped.agg({"it": 'max'}).show()
```

```
df.select(countDistinct("reg")).show()
```

```
df.select(countDistinct("reg").alias("Distinct Region")).show()
```

```
df.select(avg('specific')).show()
```

```
df.select(stddev("specific")).show()
```

```
from pyspark.sql.functions import format_number
```

```
specific_std = df.select(stddev("specific").alias('std'))
specific_std.show()
```

```
specific_std.select(format_number('std', 0)).show()
```

```
df.orderBy("specific").show()
```

```
df.orderBy(df["specific"].desc()).show()
```