

DataFrame

Cleaning Data with PySpark

Overview

1. Introduction to data cleaning with Spark
2. Immutability and lazy processing
3. Understanding Parquet

Introduction to Data Cleaning with Spark

What is Data Cleaning?

Data Cleaning: Preparing raw data for use in data processing pipelines

Possible tasks in data cleaning:

- Reformatting or replacing text
- Performing calculations
- Removing garbage or incomplete data

Why perform data cleaning with Spark?

Problems with typical data systems:

- Performance
- Organizing data flow

Advantages of Spark:

- Scalable
- Powerful framework for data handling

Data cleaning example

Raw data:

name	age (years)	city
Smith, John	37	Dallas
Wilson, A.	59	Chicago
<i>null</i>	215	

Cleaned data:

last name	first name	age (months)	state
Smith	John	444	TX
Wilson	A.	708	IL

Spark Schemas

- Define the format of a DataFrame
- May contain various data types:
 1. Strings
 2. Dates
 3. Integers
 4. arrays
- Can filter garbage data during import
- Improves read performance

Example Spark Schema

Import schema

```
import pyspark.sql.types
peopleSchema = StructType([
    # Define the name field
    StructField('name', StringType(), True),
    # Add the age field
    StructField('age', IntegerType(), True),
    # Add the city field
    StructField('city', StringType(), True)
])
```

Read CSV file containing data

```
people_df = spark.read.format('csv').load(name='rawdata.csv', schema=peopleSchema)
```


Immutability and Lazy Processing

Variable review

Python variables:

- Mutable
- Flexibility
- Potential for issues with concurrency
- Likely adds complexity

Immutability

Immutable variables are:

- A component of functional programming
- Defined once
- Unable to be directly modified
- Re-created if reassigned
- Able to be shared efficiently

Immutability Example

Define a new data frame:

```
voter_df = spark.read.csv('voterdata.csv')
```

Making changes:

```
voter_df = voter_df.withColumn('fullyear',  
    voter_df.year + 2000)  
voter_df = voter_df.drop(voter_df.year)
```

Lazy Processing

- Isn't this slow?
- Transformations
- Actions
- Allows efficient planning

```
voter_df = voter_df.withColumn('fullyear',  
                                voter_df.year + 2000)  
voter_df = voter_df.drop(voter_df.year)  
  
voter_df.count()
```

Understanding Parquet

Difficulties with CSV Files

- No defined schema
- Nested data requires special handling
- Encoding format limited

Spark and CSV files

- Slow to parse
- Files cannot be filtered (no “predicate pushdown”)
- Any intermediate use requires redefining schema

The Parquet Format

- A columnar data format
- Supported in Spark and other data processing frameworks
- Supports predicate pushdown
- Automatically stores schema information

Working with Parquet

Reading Parquet files

```
df = spark.read.format('parquet').load('filename.parquet')
```

```
df = spark.read.parquet('filename.parquet')
```

Writing Parquet files

```
df.write.format('parquet').save('filename.parquet')
```

```
df.write.parquet('filename.parquet')
```

Parquet and SQL

Parquet as backing stores for SparkSQL operations

```
flight_df = spark.read.parquet('flights.parquet')
```

```
flight_df.createOrReplaceTempView('flights')
```

```
short_flights_df = spark.sql('SELECT * FROM flights WHERE flightduration < 100')
```