DATA+AI SUMMIT EUROPE

FORMERLY SPARK+AI SUMMIT

Project Zen: Improving Apache Spark for Python Users

Hyukjin Kwon

Databricks Software Engineer

#DataTeams #DataAISummit

Hyukjin Kwon

- Apache Spark Committer / PMC
- Major Koalas contributor
- Databricks Software Engineer
- @HyukjinKwon in GitHub

Agenda

What is Project Zen?

Redesigned Documentation

PySpark Type Hints

Distribution Option for PyPI Users

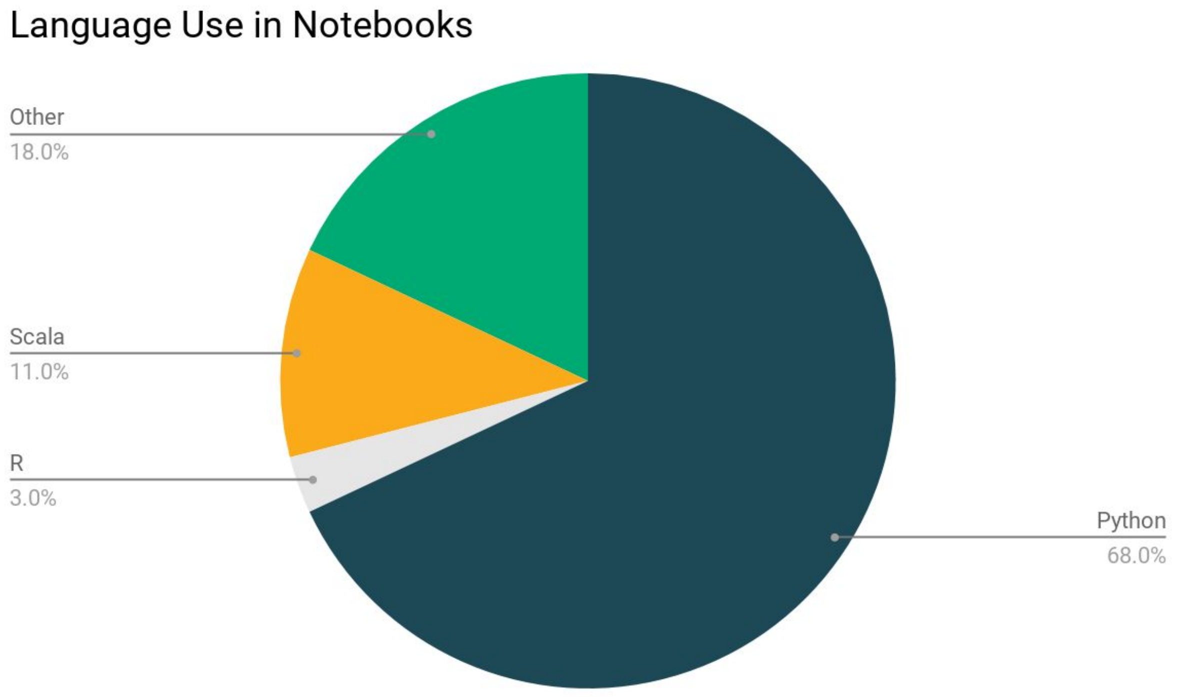
Roadmap

What is Project Zen?

Python Growth

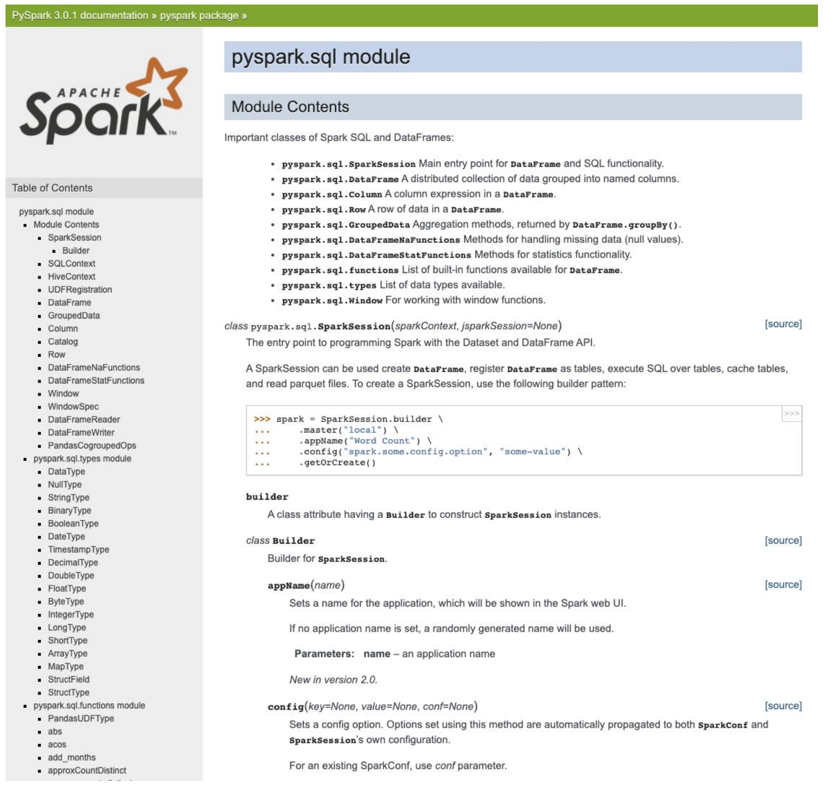
68%

of notebook commands on Databricks are in Python



PySpark Today

- Documentation difficult to navigate
  - All APIs under each module is listed in single page
  - No other information or classification
- Lack of information
  - No Quickstart page
  - No Installation page
  - No Introduction



PySpark 3.0.1 documentation » pyspark package »

Table of Contents

pyspark.sql module

Module Contents

Important classes of Spark SQL and DataFrames:

- **pyspark.sql.SparkSession** Main entry point for **DataFrame** and SQL functionality.
- **pyspark.sql.DataFrame** A distributed collection of data grouped into named columns.
- **pyspark.sql.Column** A column expression in a **DataFrame**.
- **pyspark.sql.Row** A row of data in a **DataFrame**.
- **pyspark.sql.GroupedData** Aggregation methods, returned by **DataFrame.groupBy()**.
- **pyspark.sql.DataFrameNaFunctions** Methods for handling missing data (null values).
- **pyspark.sql.DataFrameStatFunctions** Methods for statistics functionality.
- **pyspark.sql.functions** List of built-in functions available for **DataFrame**.
- **pyspark.sql.types** List of data types available.
- **pyspark.sql.Window** For working with window functions.

class pyspark.sql.**SparkSession**(*sparkContext*, *jsparkSession=None*) [source]

The entry point to programming Spark with the Dataset and DataFrame API.

A SparkSession can be used create **DataFrame**, register **DataFrame** as tables, execute SQL over tables, cache tables, and read parquet files. To create a SparkSession, use the following builder pattern:

```
>>> spark = SparkSession.builder \
...     .master("local") \
...     .appName("Word Count") \
...     .config("spark.some.config.option", "some-value") \
...     .getOrCreate()
```

builder

A class attribute having a **Builder** to construct **SparkSession** instances.

class **Builder** [source]

Builder for **SparkSession**.

appName(*name*) [source]

Sets a name for the application, which will be shown in the Spark web UI.

If no application name is set, a randomly generated name will be used.

Parameters: **name** – an application name

New in version 2.0.

config(*key=None*, *value=None*, *conf=None*) [source]

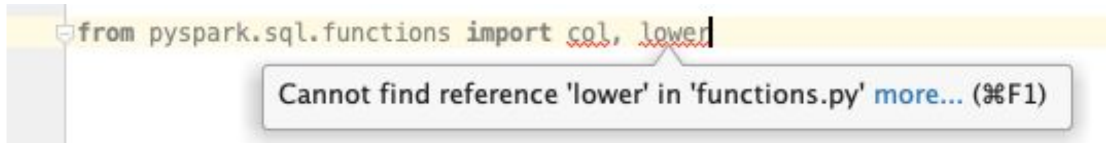
Sets a config option. Options set using this method are automatically propagated to both **SparkConf** and **SparkSession**'s own configuration.

For an existing SparkConf, use *conf* parameter.

PySpark documentation

PySpark Today

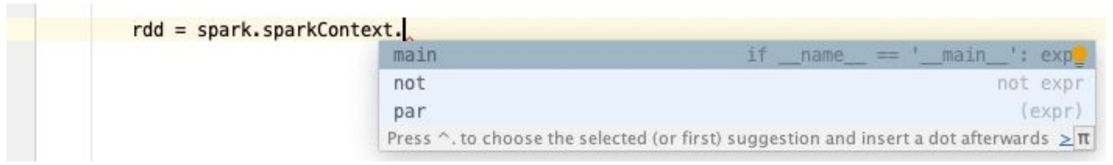
- IDE unfriendly
  - Dynamically defined functions reported as missing functions
  - Lack of autocompletion support
  - Lack of type checking support
- Notebook unfriendly
  - Lack of autocompletion support



```
from pyspark.sql.functions import col, lower
```

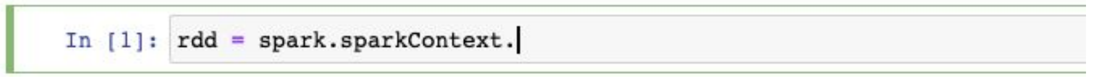
Cannot find reference 'lower' in 'functions.py' more... (⌘F1)

Missing import in IDE



```
rdd = spark.sparkContext.
main     if __name__ == '__main__': exp
not      not expr
par      (expr)
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards
```

Autocompletion in IDE



```
In [1]: rdd = spark.sparkContext.
```

Autocompletion in Jupyter

PySpark Today

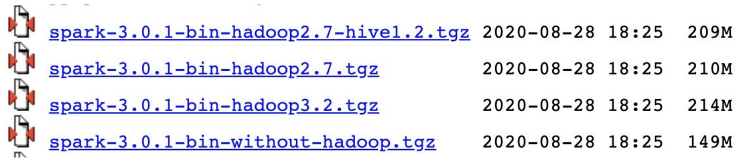
- Less Pythonic
  - Deprecated Python built-in instance support when creating a DataFrame

```
>>> spark.createDataFrame([{'a': 1}])
```

```
/.../session.py:378: UserWarning: inferring schema from dict is deprecated, please use
pyspark.sql.Row instead
```

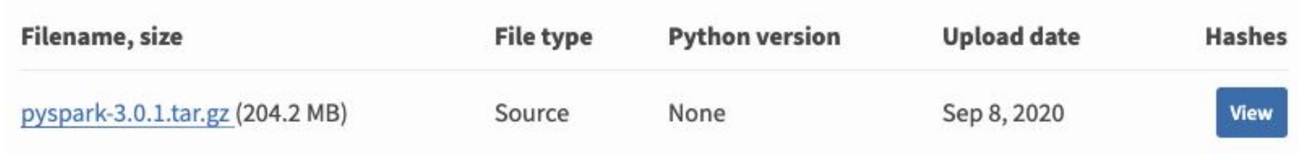
PySpark Today

- Missing distributions with other Hadoop versions in PyPI
  - Missing Hadoop 3 distribution
  - Missing Hive 1.2 distribution



```
spark-3.0.1-bin-hadoop2.7-hive1.2.tgz 2020-08-28 18:25  209M
spark-3.0.1-bin-hadoop2.7.tgz         2020-08-28 18:25  210M
spark-3.0.1-bin-hadoop3.2.tgz         2020-08-28 18:25  214M
spark-3.0.1-bin-without-hadoop.tgz    2020-08-28 18:25  149M
```

Apache Mirror



Filename, size	File type	Python version	Upload date	Hashes
pyspark-3.0.1.tar.gz (204.2 MB)	Source	None	Sep 8, 2020	View

PyPI Distribution

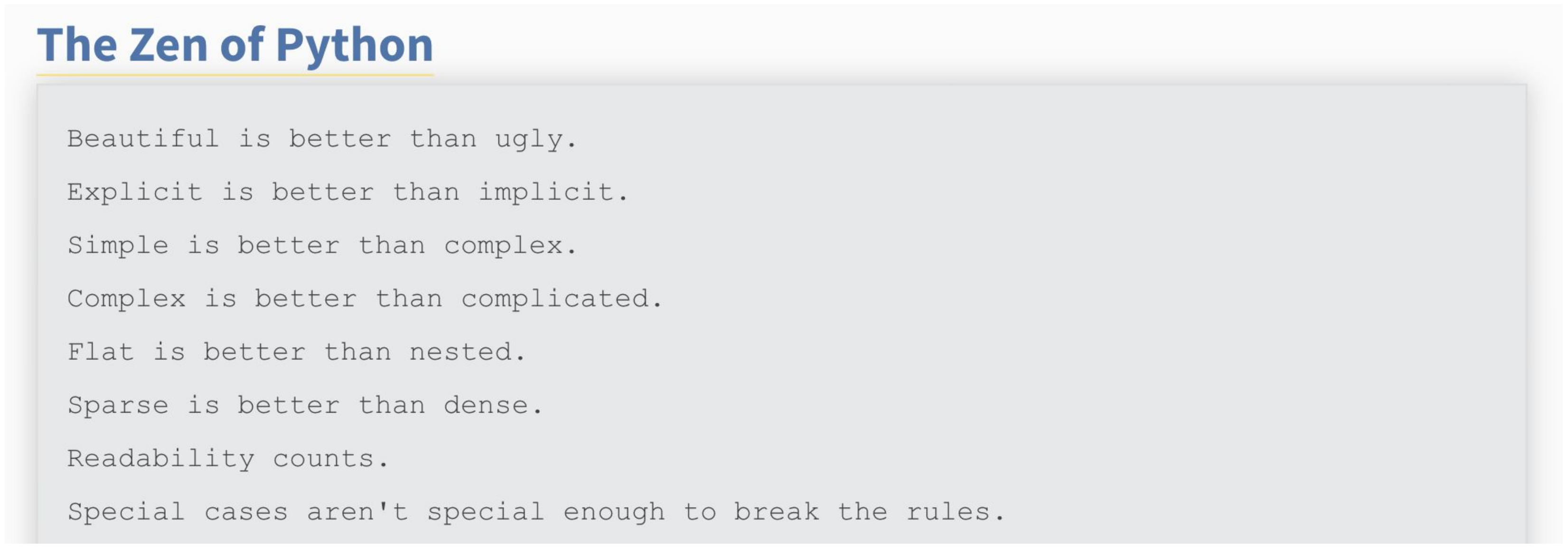
PySpark Today

- Inconsistent exceptions and warnings
  - Unclassified exceptions and warnings

```
>>> spark.range(10).explain(1, 2)
```

```
Traceback (most recent call last):
    ...
Exception: extended and mode should not be set together.
```

The Zen of Python



The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
```

PEP 20 — The Zen of Python

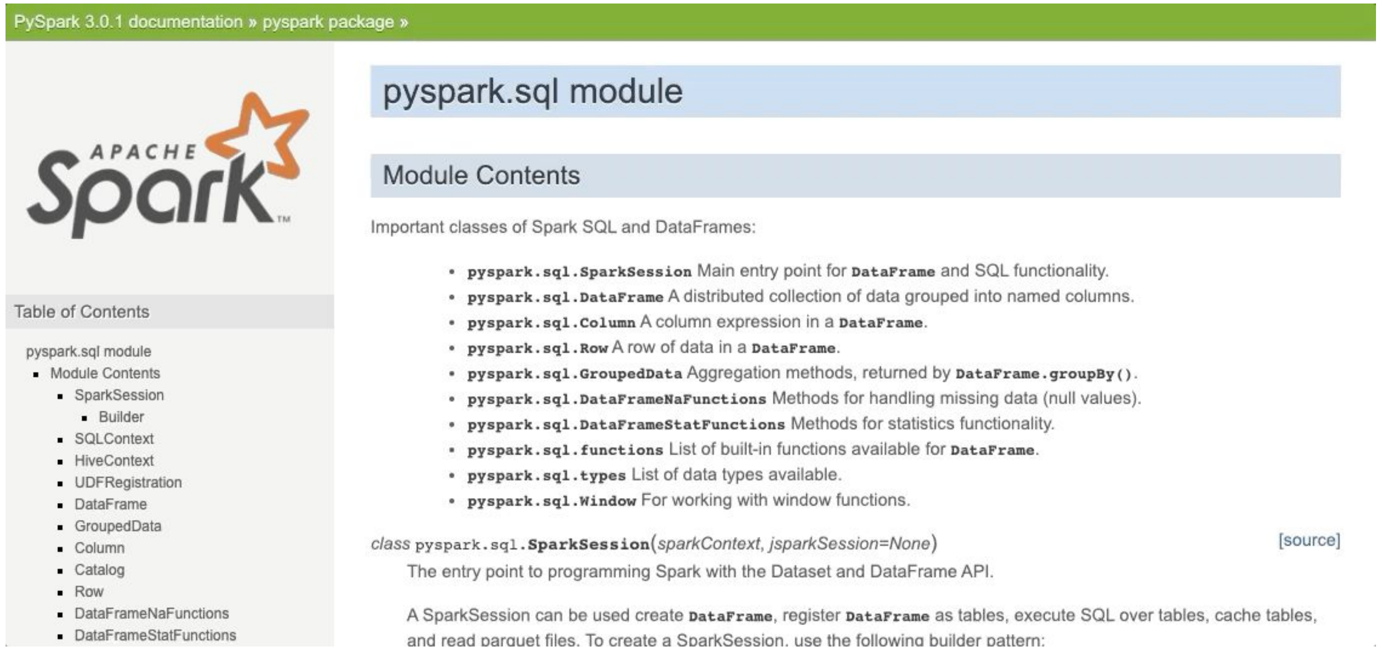
Project Zen (SPARK-32082)

- Be Pythonic
  - The Zen of Python
  - Python friendly
- Better and easier use of PySpark
  - Better documentation
  - Clear exceptions and warnings
  - Python type hints: autocompletion, static type checking and error detection
  - More options for `pip` installation
- Better interoperability with other Python libraries
  - pandas, pyarrow, NumPy, Koalas, etc.
  - Visualization

Redesigned Documentation

Problems in PySpark Documentation

- Everything in few pages
  - Whole module in single page w/o classification
- Difficult to navigate
  - Very long to stroll down
  - Virtually no structure
- No other useful pages
  - How to start?
  - How to ship 3rd party packages together?
  - How to install?
  - How to debug / setup an IDE?



PySpark 3.0.1 documentation » pyspark package »

Table of Contents

pyspark.sql module

Module Contents

Important classes of Spark SQL and DataFrames:

- `pyspark.sql.SparkSession` Main entry point for `DataFrame` and SQL functionality.
- `pyspark.sql.DataFrame` A distributed collection of data grouped into named columns.
- `pyspark.sql.Column` A column expression in a `DataFrame`.
- `pyspark.sql.Row` A row of data in a `DataFrame`.
- `pyspark.sql.GroupedData` Aggregation methods, returned by `DataFrame.groupBy()`.
- `pyspark.sql.DataFrameNaFunctions` Methods for handling missing data (null values).
- `pyspark.sql.DataFrameStatFunctions` Methods for statistics functionality.
- `pyspark.sql.functions` List of built-in functions available for `DataFrame`.
- `pyspark.sql.types` List of data types available.
- `pyspark.sql.Window` For working with window functions.

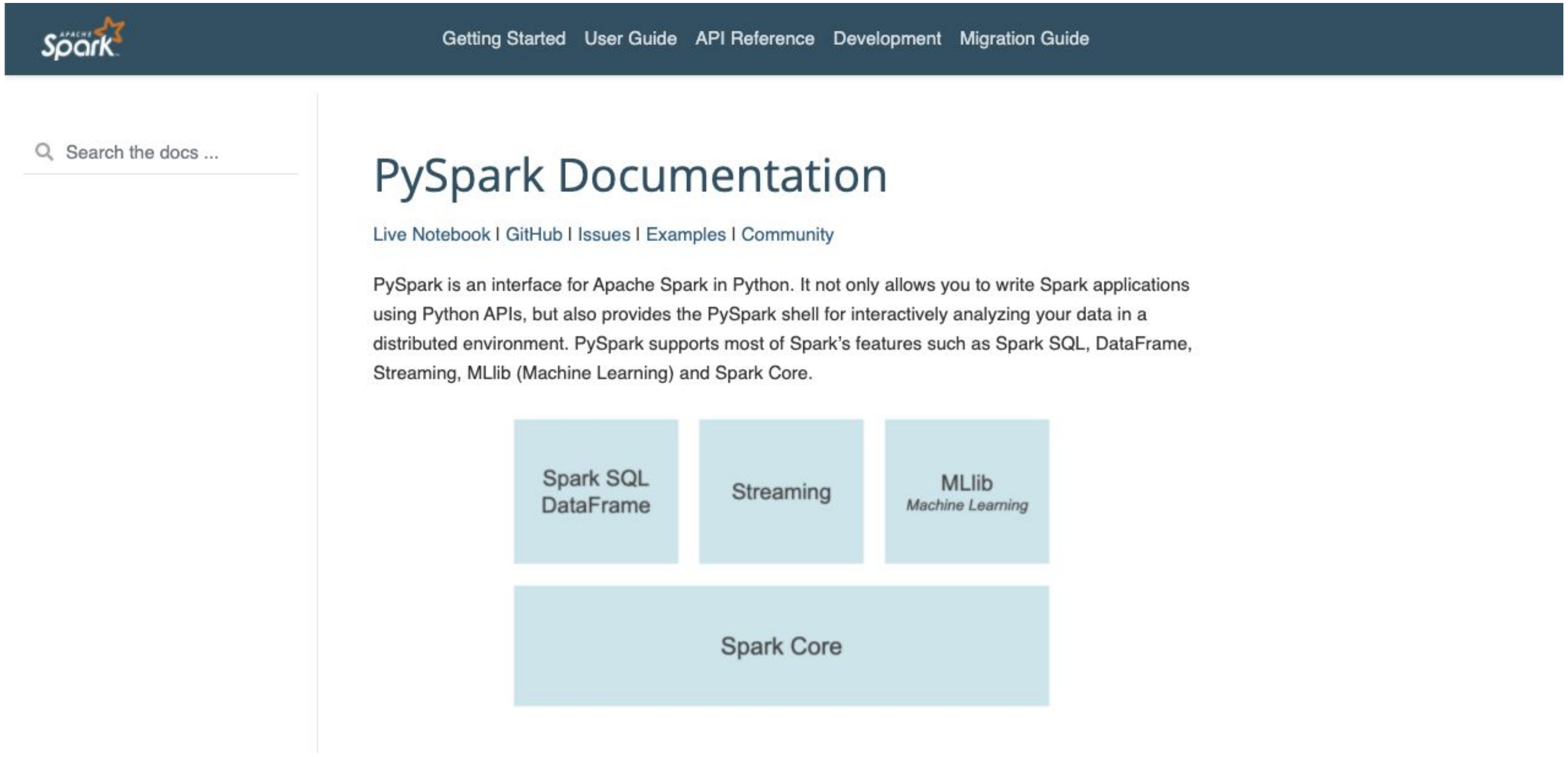
class `pyspark.sql.SparkSession`(*sparkContext*, *jsparkSession=None*) [source]

The entry point to programming Spark with the Dataset and DataFrame API.

A SparkSession can be used create `DataFrame`, register `DataFrame` as tables, execute SQL over tables, cache tables, and read parquet files. To create a SparkSession, use the following builder pattern:

(Old) PySpark documentation

New PySpark Documentation



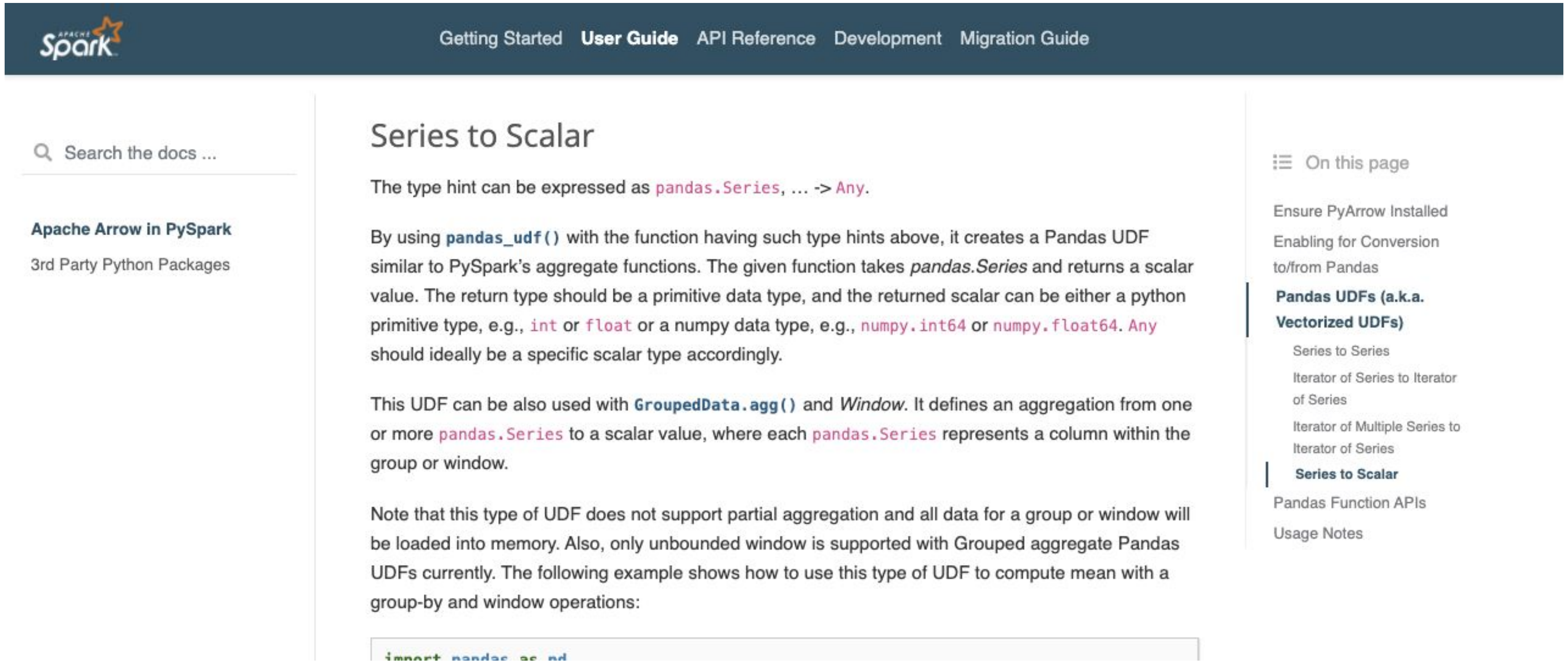
PySpark Documentation

Live Notebook | GitHub | Issues | Examples | Community

PySpark is an interface for Apache Spark in Python. It not only allows you to write Spark applications using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed environment. PySpark supports most of Spark's features such as Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) and Spark Core.

New PySpark documentation

New PySpark Documentation



Series to Scalar

The type hint can be expressed as `pandas.Series`, … -> `Any`.

By using `pandas_udf()` with the function having such type hints above, it creates a Pandas UDF similar to PySpark's aggregate functions. The given function takes *pandas.Series* and returns a scalar value. The return type should be a primitive data type, and the returned scalar can be either a python primitive type, e.g., `int` or `float` or a numpy data type, e.g., `numpy.int64` or `numpy.float64`. `Any` should ideally be a specific scalar type accordingly.

This UDF can be also used with `GroupedData.agg()` and *Window*. It defines an aggregation from one or more `pandas.Series` to a scalar value, where each `pandas.Series` represents a column within the group or window.

Note that this type of UDF does not support partial aggregation and all data for a group or window will be loaded into memory. Also, only unbounded window is supported with Grouped aggregate Pandas UDFs currently. The following example shows how to use this type of UDF to compute mean with a group-by and window operations:

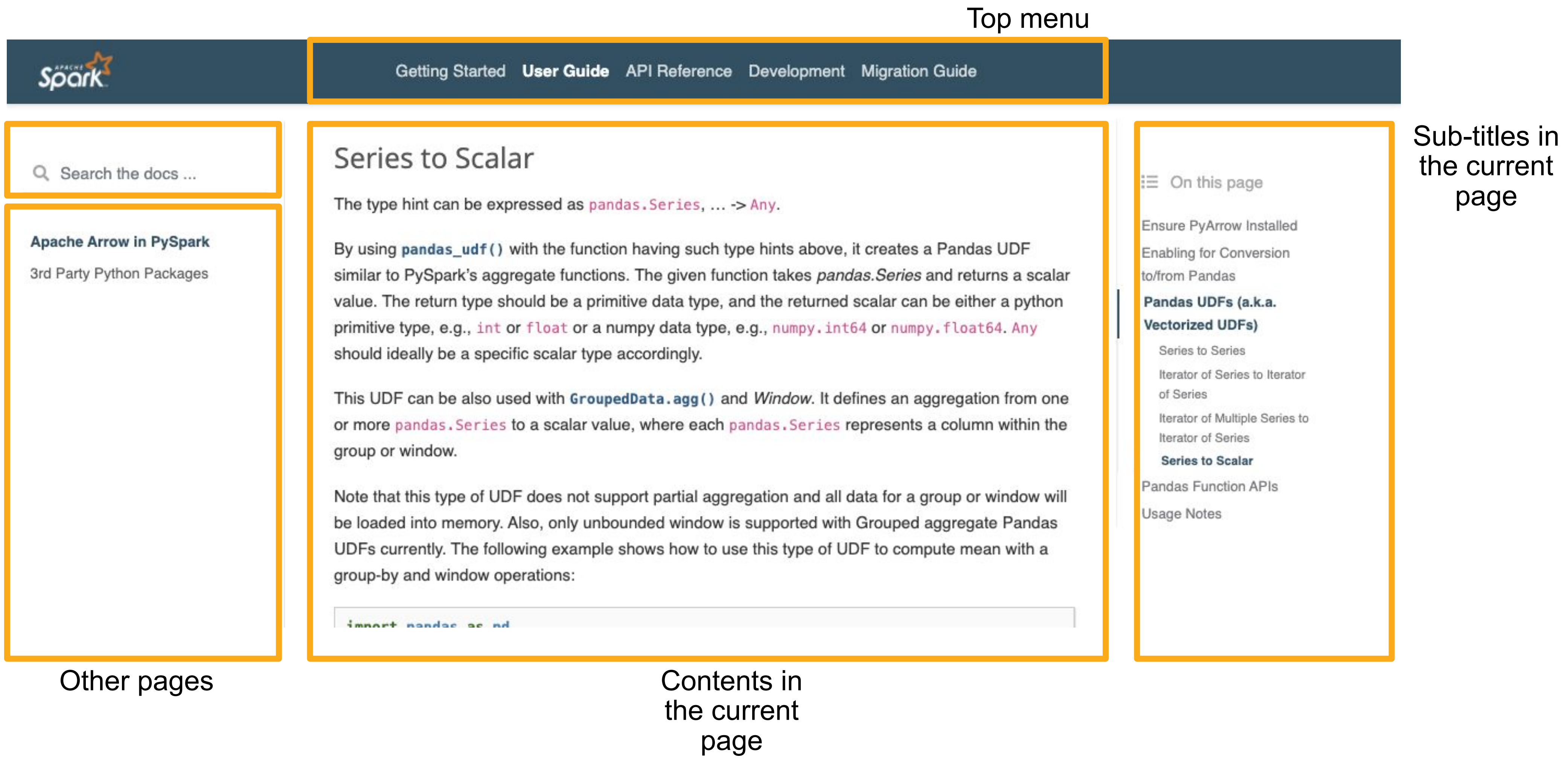
```
import pandas as pd
```

New user guide page

DATA+AI SUMMIT EUROPE

#DataTeams #DataAISummit

New PySpark Documentation


Top menu
Search
Sub-titles in the current page
Other pages
Contents in the current page
Getting Started
User Guide
API Reference
Development
Migration Guide
Search the docs ...
Apache Arrow in PySpark
3rd Party Python Packages
On this page
Ensure PyArrow Installed
Enabling for Conversion to/from Pandas
Pandas UDFs (a.k.a. Vectorized UDFs)
Series to Series
Iterator of Series to Iterator of Series
Iterator of Multiple Series to Iterator of Series
Series to Scalar
Pandas Function APIs
Usage Notes

Series to Scalar

The type hint can be expressed as `pandas.Series`, … -> `Any`.

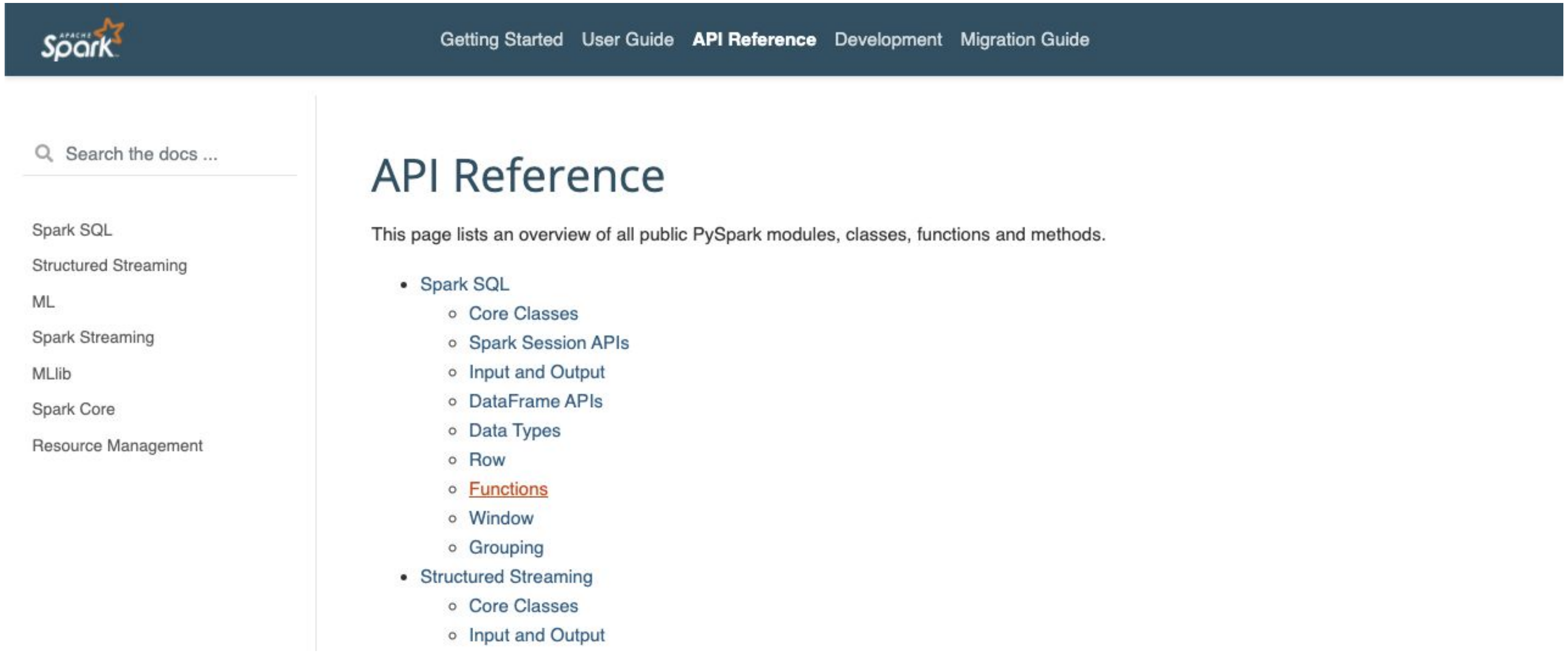
By using `pandas_udf()` with the function having such type hints above, it creates a Pandas UDF similar to PySpark's aggregate functions. The given function takes *pandas.Series* and returns a scalar value. The return type should be a primitive data type, and the returned scalar can be either a python primitive type, e.g., `int` or `float` or a numpy data type, e.g., `numpy.int64` or `numpy.float64`. `Any` should ideally be a specific scalar type accordingly.

This UDF can be also used with `GroupedData.agg()` and *Window*. It defines an aggregation from one or more `pandas.Series` to a scalar value, where each `pandas.Series` represents a column within the group or window.

Note that this type of UDF does not support partial aggregation and all data for a group or window will be loaded into memory. Also, only unbounded window is supported with Grouped aggregate Pandas UDFs currently. The following example shows how to use this type of UDF to compute mean with a group-by and window operations:

```
import pandas as pd
```

New API Reference



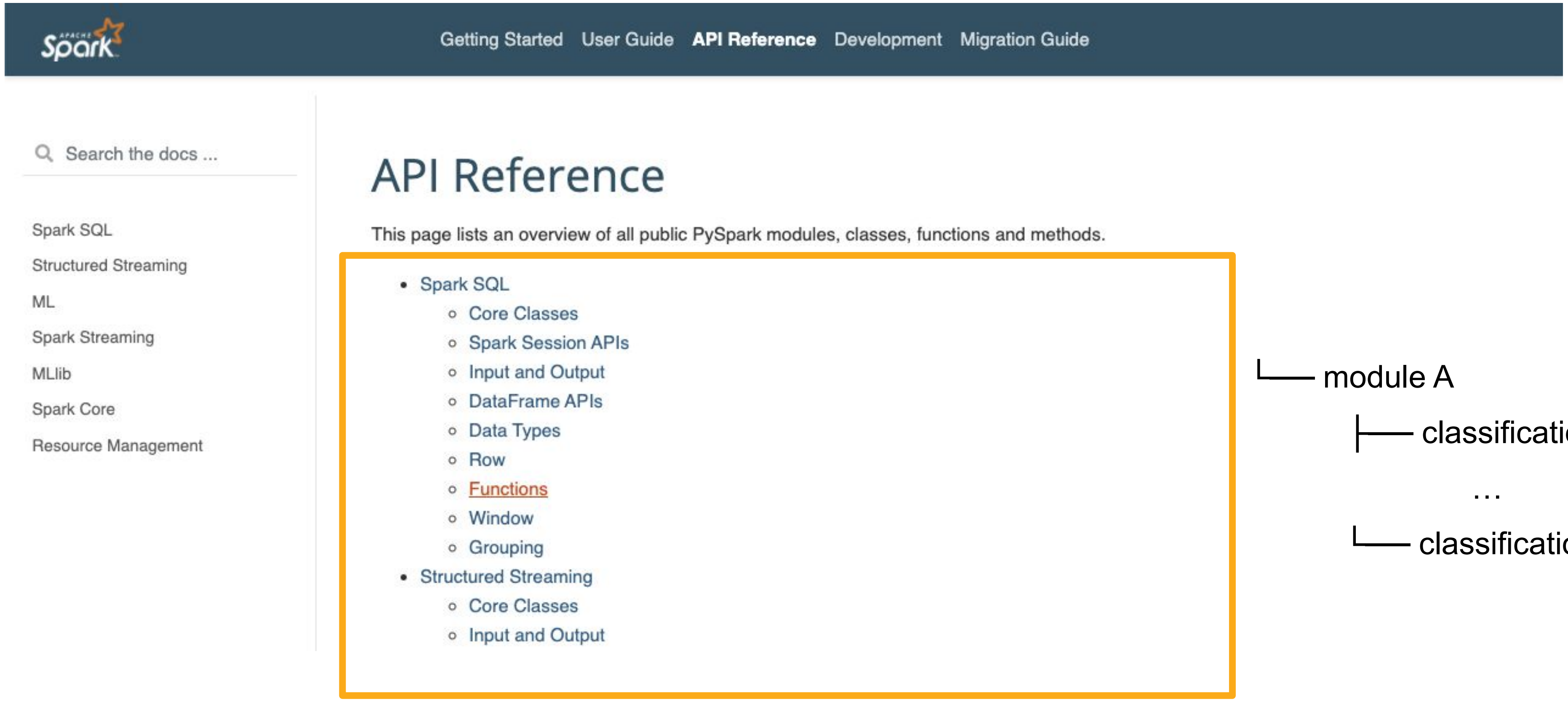
API Reference

This page lists an overview of all public PySpark modules, classes, functions and methods.

- Spark SQL
  - Core Classes
  - Spark Session APIs
  - Input and Output
  - DataFrame APIs
  - Data Types
  - Row
  - Functions
  - Window
  - Grouping
- Structured Streaming
  - Core Classes
  - Input and Output

New API reference page

New API Reference



API Reference

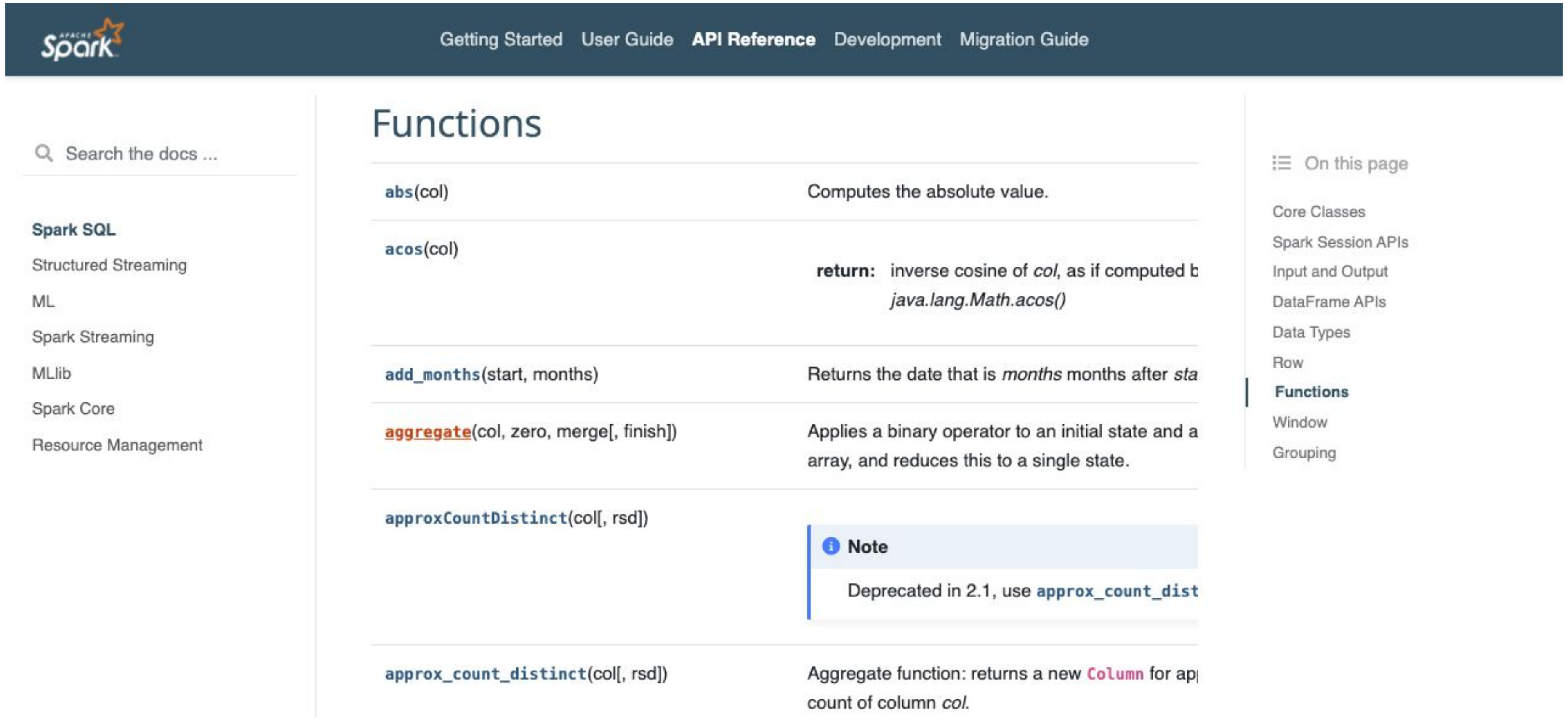
This page lists an overview of all public PySpark modules, classes, functions and methods.

- Spark SQL
  - Core Classes
  - Spark Session APIs
  - Input and Output
  - DataFrame APIs
  - Data Types
  - Row
  - Functions
  - Window
  - Grouping
- Structured Streaming
  - Core Classes
  - Input and Output

```
└── module A
      ├── classification A
             ...
      └── classification ...
```

New API reference page

New API Reference


Getting Started User Guide **API Reference** Development Migration Guide

Search the docs ...

Spark SQL
Structured Streaming
ML
Spark Streaming
MLlib
Spark Core
Resource Management

Functions

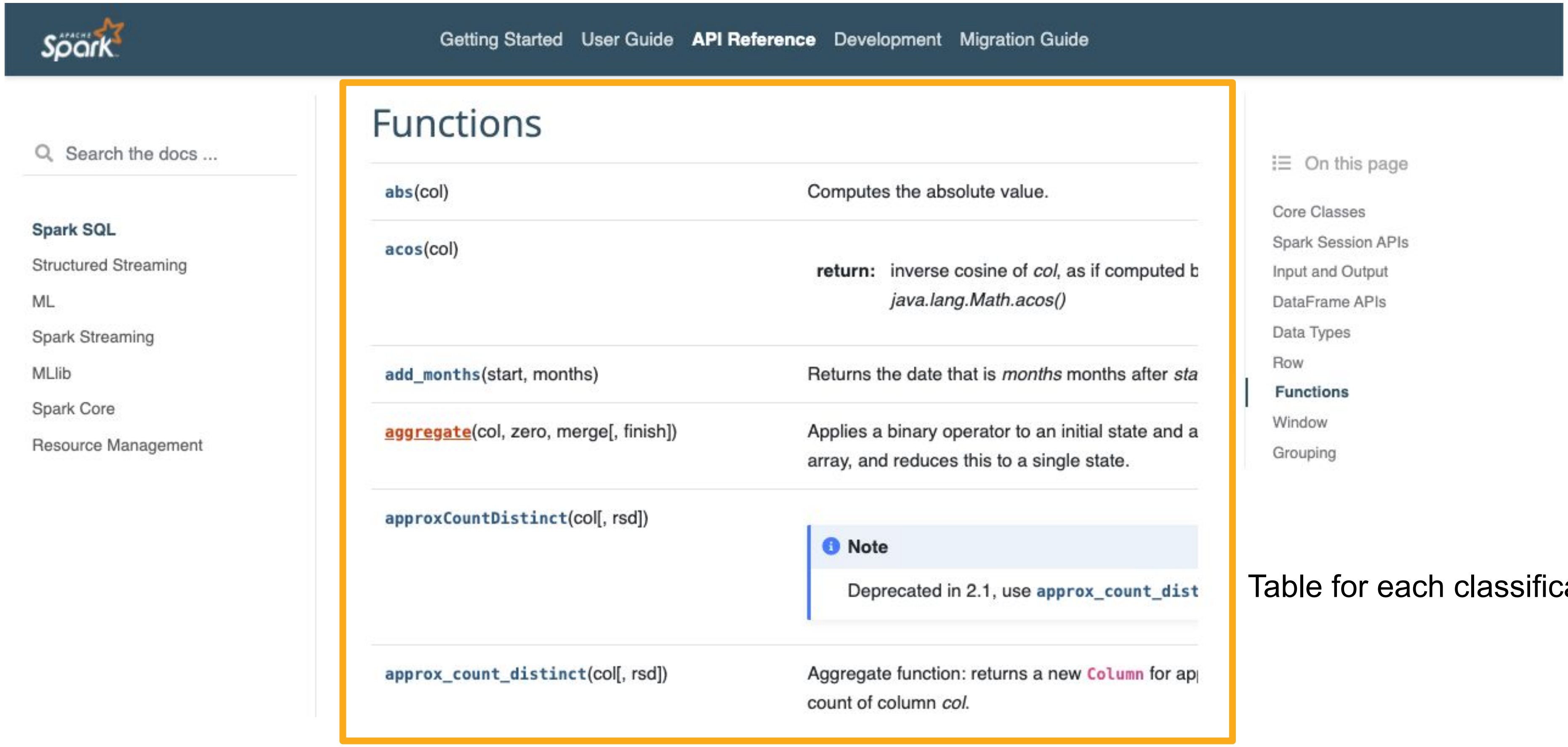
`abs`(col)	Computes the absolute value.
`acos`(col)	**return:** inverse cosine of *col*, as if computed b *java.lang.Math.acos()*
`add_months`(start, months)	Returns the date that is *months* months after *sta*
`aggregate`(col, zero, merge[, finish])	Applies a binary operator to an initial state and a array, and reduces this to a single state.
`approxCountDistinct`(col[, rsd])	**Note** Deprecated in 2.1, use `approx_count_dist`
`approx_count_distinct`(col[, rsd])	Aggregate function: returns a new `Column` for ap count of column *col*.

On this page

Core Classes
Spark Session APIs
Input and Output
DataFrame APIs
Data Types
Row
Functions
Window
Grouping

New API reference page

New API Reference



Getting Started User Guide **API Reference** Development Migration Guide

Search the docs ...

Spark SQL
Structured Streaming
ML
Spark Streaming
MLlib
Spark Core
Resource Management

Functions

`abs`(col)	Computes the absolute value.
`acos`(col)	**return:** inverse cosine of *col*, as if computed b *java.lang.Math.acos()*
`add_months`(start, months)	Returns the date that is *months* months after *sta*
`aggregate`(col, zero, merge[, finish])	Applies a binary operator to an initial state and a array, and reduces this to a single state.
`approxCountDistinct`(col[, rsd])	**Note** Deprecated in 2.1, use `approx_count_dist`
`approx_count_distinct`(col[, rsd])	Aggregate function: returns a new `Column` for ap count of column *col*.

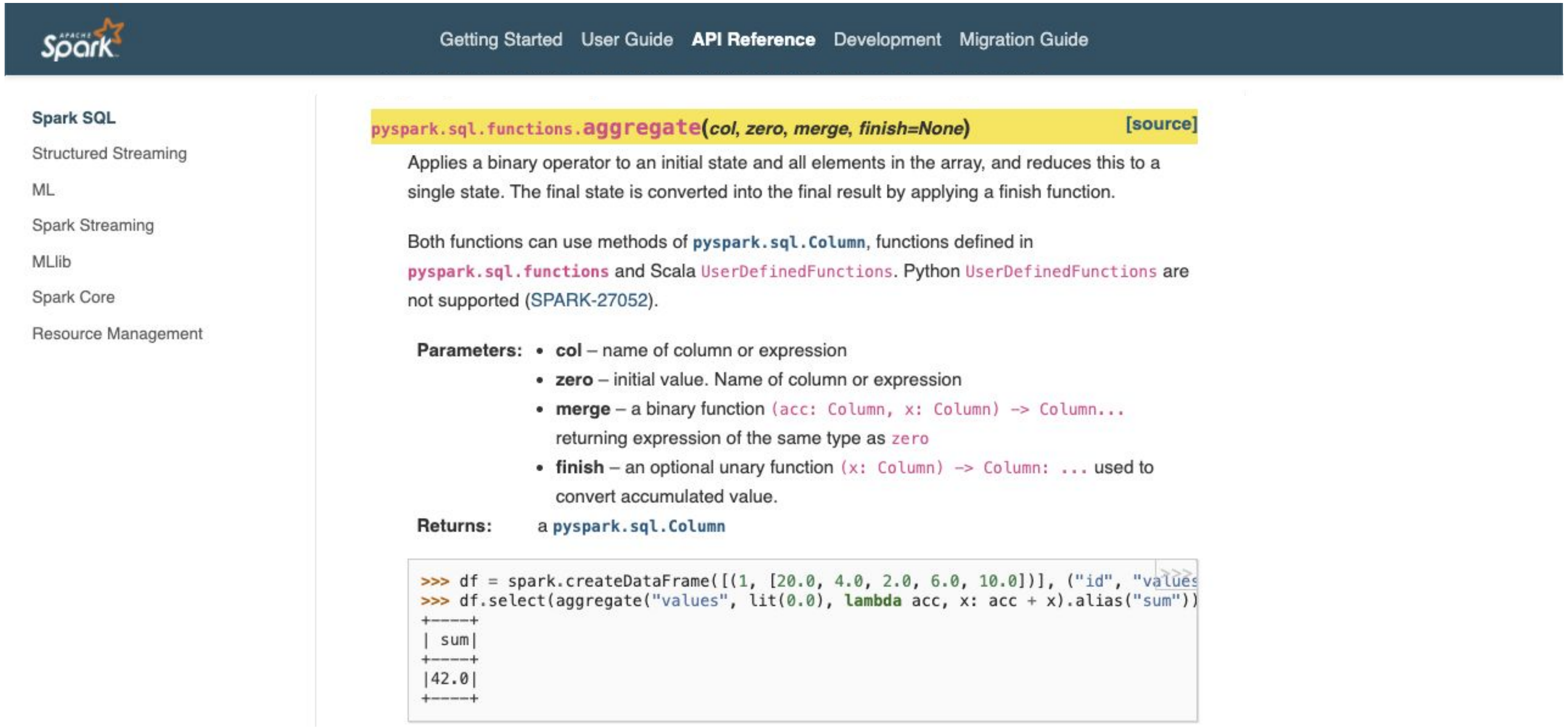
On this page

Core Classes
Spark Session APIs
Input and Output
DataFrame APIs
Data Types
Row
Functions
Window
Grouping

Table for each classification

New API reference page

New API Reference



`pyspark.sql.functions.aggregate`(*col, zero, merge, finish=None*) [source]

Applies a binary operator to an initial state and all elements in the array, and reduces this to a single state. The final state is converted into the final result by applying a finish function.

Both functions can use methods of `pyspark.sql.Column`, functions defined in `pyspark.sql.functions` and Scala `UserDefinedFunctions`. Python `UserDefinedFunctions` are not supported (SPARK-27052).

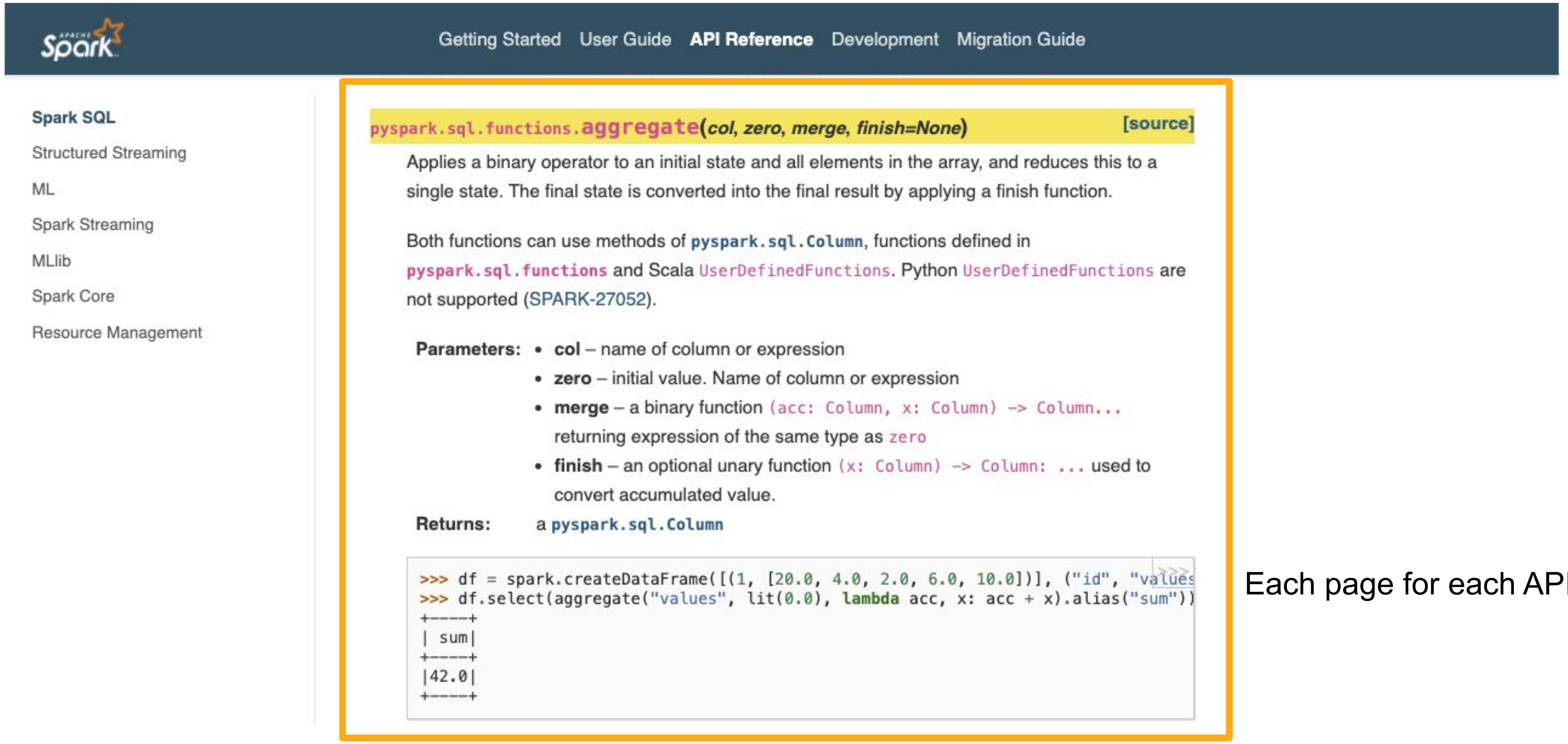
Parameters:
- **col** – name of column or expression
- **zero** – initial value. Name of column or expression
- **merge** – a binary function `(acc: Column, x: Column) -> Column...` returning expression of the same type as `zero`
- **finish** – an optional unary function `(x: Column) -> Column: ...` used to convert accumulated value.

Returns: a `pyspark.sql.Column`

```
>>> df = spark.createDataFrame([(1, [20.0, 4.0, 2.0, 6.0, 10.0])], ("id", "values
>>> df.select(aggregate("values", lit(0.0), lambda acc, x: acc + x).alias("sum"))
+----+
| sum|
+----+
|42.0|
+----+
```

New API reference page

New API Reference



Getting Started User Guide **API Reference** Development Migration Guide

Spark SQL
Structured Streaming
ML
Spark Streaming
MLlib
Spark Core
Resource Management

`pyspark.sql.functions.aggregate`(*col, zero, merge, finish=None*) [source]

Applies a binary operator to an initial state and all elements in the array, and reduces this to a single state. The final state is converted into the final result by applying a finish function.

Both functions can use methods of `pyspark.sql.Column`, functions defined in `pyspark.sql.functions` and Scala `UserDefinedFunctions`. Python `UserDefinedFunctions` are not supported (SPARK-27052).

Parameters:
- **col** – name of column or expression
- **zero** – initial value. Name of column or expression
- **merge** – a binary function `(acc: Column, x: Column) -> Column...` returning expression of the same type as `zero`
- **finish** – an optional unary function `(x: Column) -> Column: ...` used to convert accumulated value.

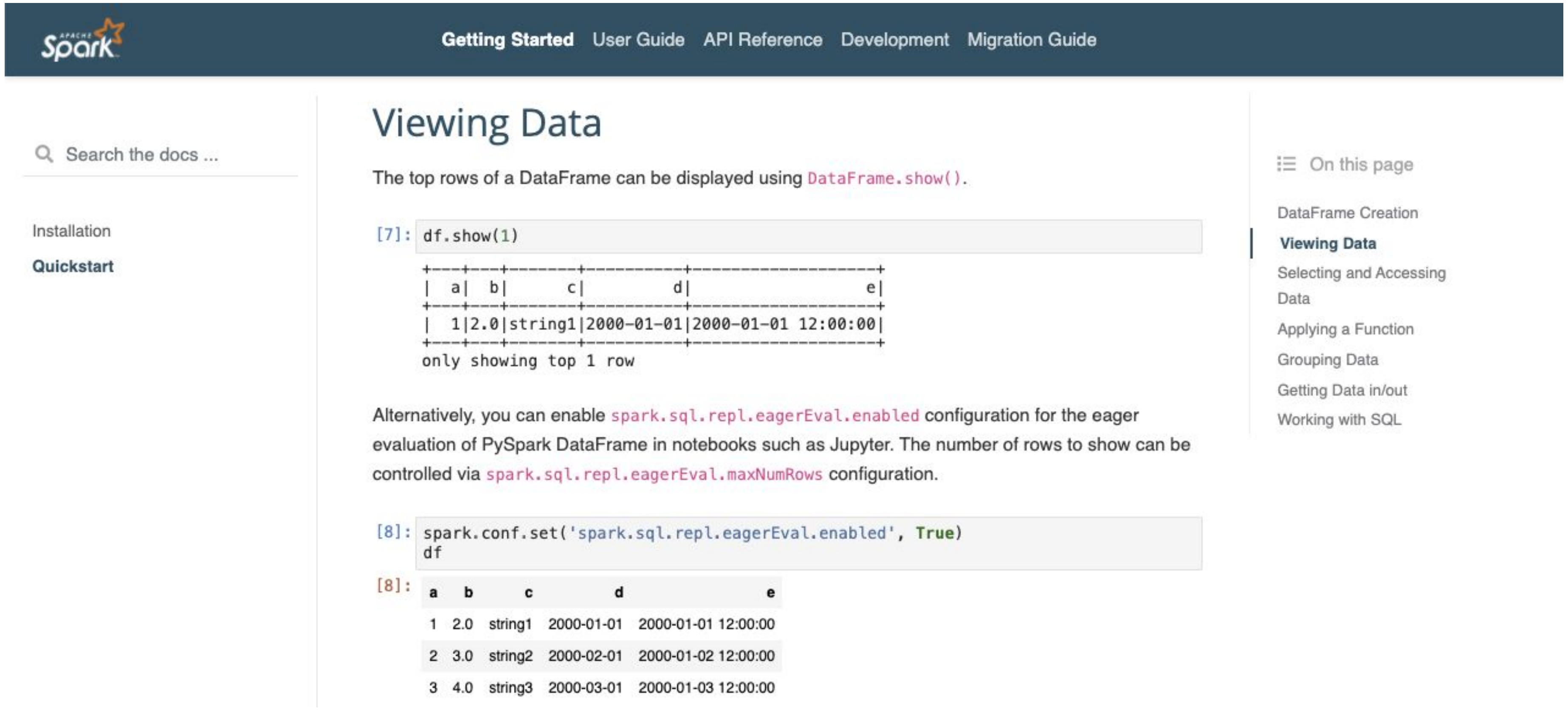
Returns: a `pyspark.sql.Column`

```
>>> df = spark.createDataFrame([(1, [20.0, 4.0, 2.0, 6.0, 10.0])], ("id", "values
>>> df.select(aggregate("values", lit(0.0), lambda acc, x: acc + x).alias("sum"))
+-----+
| sum|
+-----+
|42.0|
+-----+
```

Each page for each API

New API reference page

Quickstart



Viewing Data

The top rows of a DataFrame can be displayed using `DataFrame.show()`.

```
[7]: df.show(1)

     +---+---+-------+----------+-------------------+
     |  a|  b|      c|         d|                  e|
     +---+---+-------+----------+-------------------+
     |  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|
     +---+---+-------+----------+-------------------+
     only showing top 1 row
```

Alternatively, you can enable `spark.sql.repl.eagerEval.enabled` configuration for the eager evaluation of PySpark DataFrame in notebooks such as Jupyter. The number of rows to show can be controlled via `spark.sql.repl.eagerEval.maxNumRows` configuration.

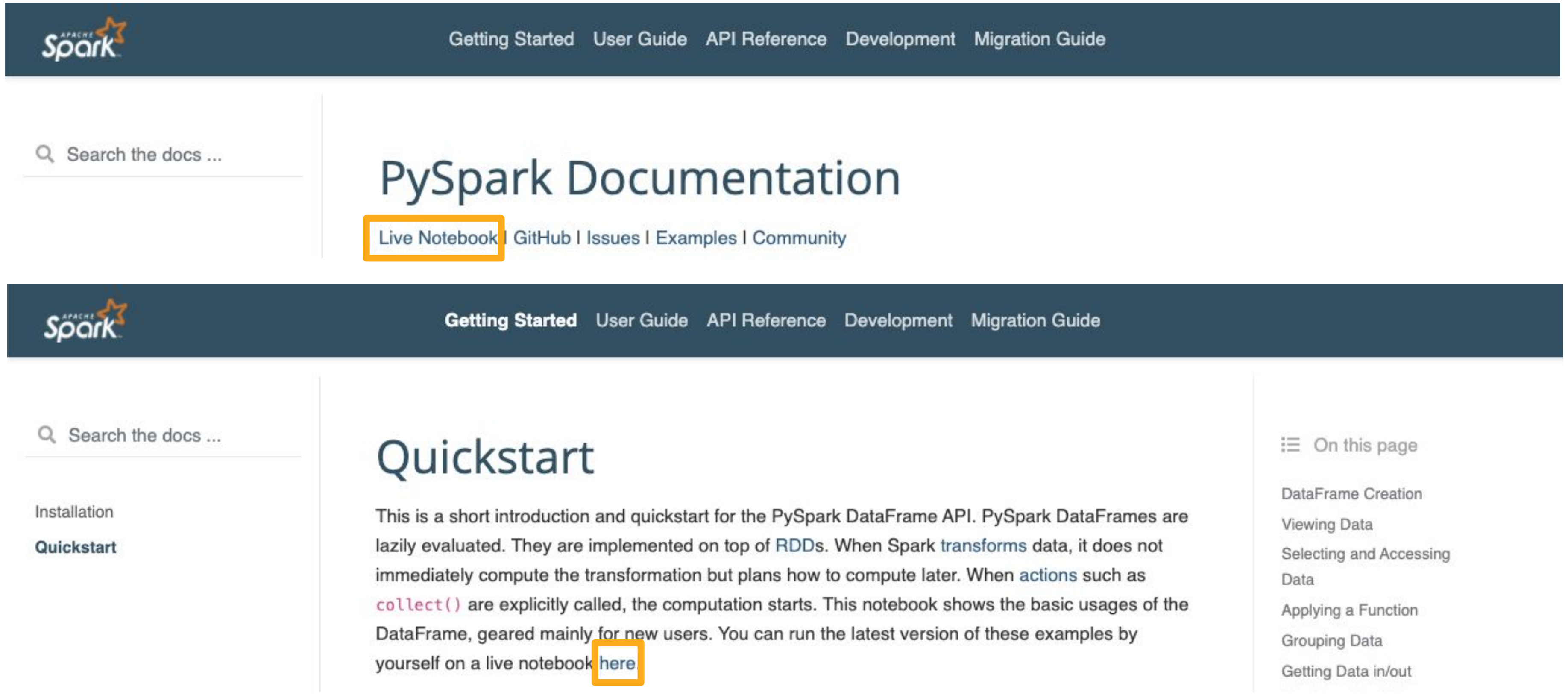
```
[8]: spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
     df
```

[8]:

a	b	c	d	e
1	2.0	string1	2000-01-01	2000-01-01 12:00:00
2	3.0	string2	2000-02-01	2000-01-02 12:00:00
3	4.0	string3	2000-03-01	2000-01-03 12:00:00

Quickstart page

Live Notebook



Getting Started User Guide API Reference Development Migration Guide

Search the docs ...

PySpark Documentation

Live Notebook | GitHub | Issues | Examples | Community

Getting Started User Guide API Reference Development Migration Guide

Search the docs ...

Installation

Quickstart

Quickstart

This is a short introduction and quickstart for the PySpark DataFrame API. PySpark DataFrames are lazily evaluated. They are implemented on top of RDDs. When Spark transforms data, it does not immediately compute the transformation but plans how to compute later. When actions such as `collect()` are explicitly called, the computation starts. This notebook shows the basic usages of the DataFrame, geared mainly for new users. You can run the latest version of these examples by yourself on a live notebook here.

On this page

DataFrame Creation

Viewing Data

Selecting and Accessing Data

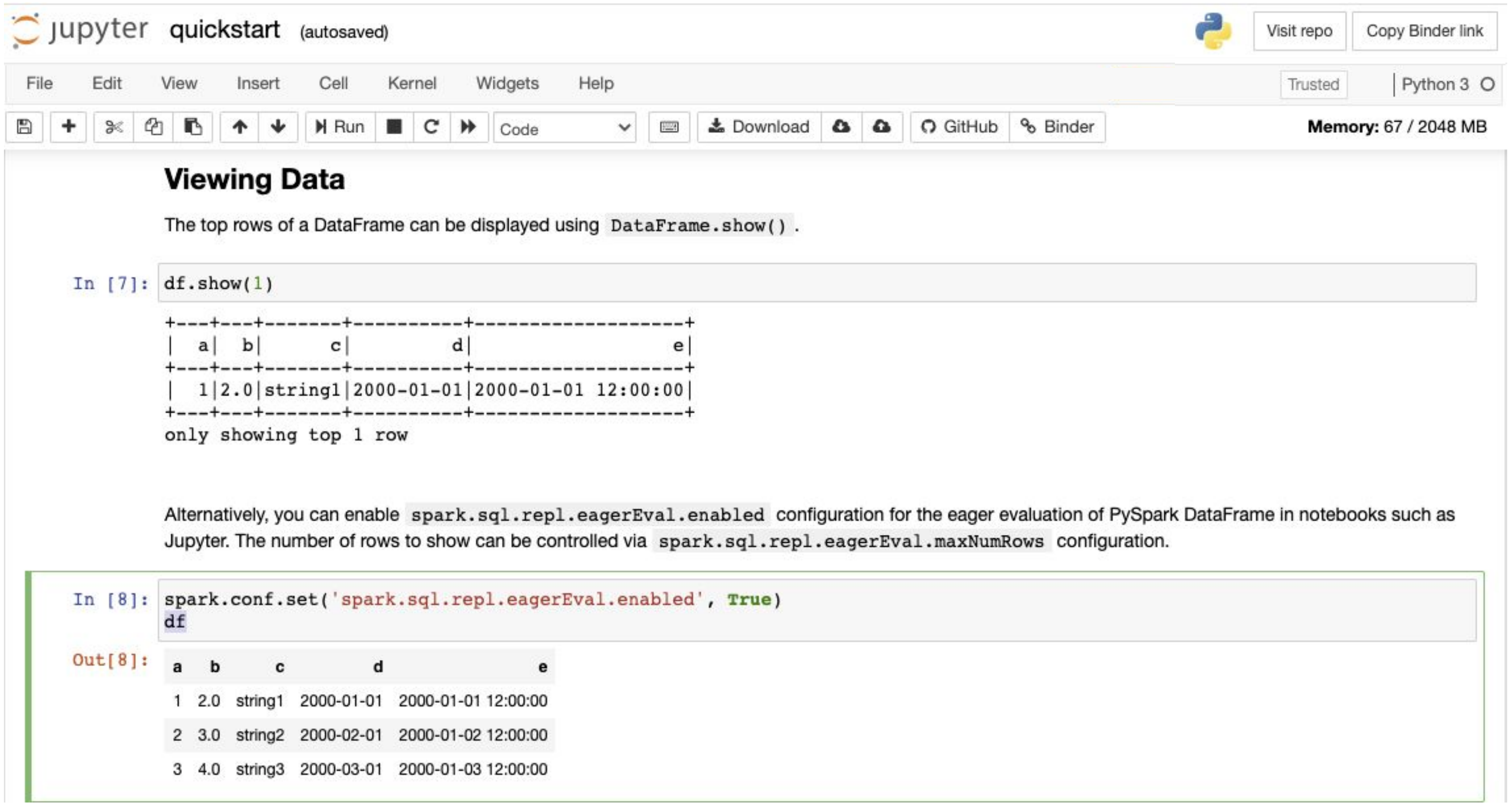
Applying a Function

Grouping Data

Getting Data in/out

Move to live notebook (Binder integration)

Live Notebook



Viewing Data

The top rows of a DataFrame can be displayed using `DataFrame.show()`.

```
In [7]: df.show(1)
```

```
+---+---+-------+----------+-------------------+
|  a|  b|      c|         d|                  e|
+---+---+-------+----------+-------------------+
|  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|
+---+---+-------+----------+-------------------+
only showing top 1 row
```

Alternatively, you can enable `spark.sql.repl.eagerEval.enabled` configuration for the eager evaluation of PySpark DataFrame in notebooks such as Jupyter. The number of rows to show can be controlled via `spark.sql.repl.eagerEval.maxNumRows` configuration.

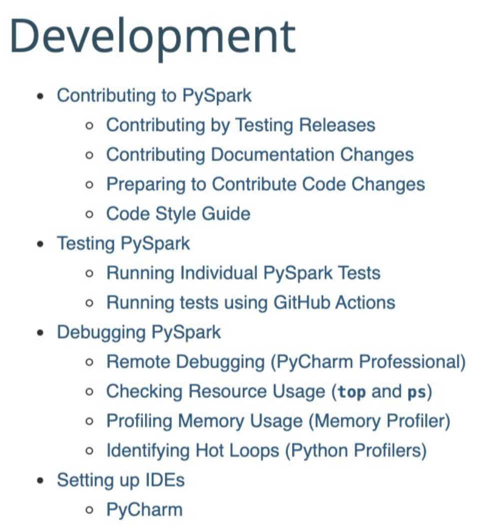
```
In [8]: spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
        df
```

Out[8]:

a	b	c	d	e
1	2.0	string1	2000-01-01	2000-01-01 12:00:00
2	3.0	string2	2000-02-01	2000-01-02 12:00:00
3	4.0	string3	2000-03-01	2000-01-03 12:00:00

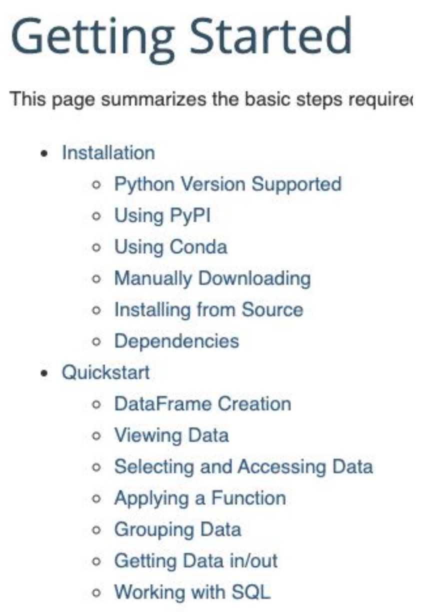
Live notebook (Binder integration)

Other New Pages



Development

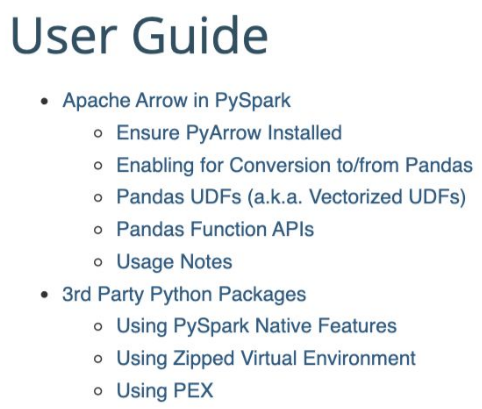
- Contributing to PySpark
  - Contributing by Testing Releases
  - Contributing Documentation Changes
  - Preparing to Contribute Code Changes
  - Code Style Guide
- Testing PySpark
  - Running Individual PySpark Tests
  - Running tests using GitHub Actions
- Debugging PySpark
  - Remote Debugging (PyCharm Professional)
  - Checking Resource Usage (`top` and `ps`)
  - Profiling Memory Usage (Memory Profiler)
  - Identifying Hot Loops (Python Profilers)
- Setting up IDEs
  - PyCharm



Getting Started

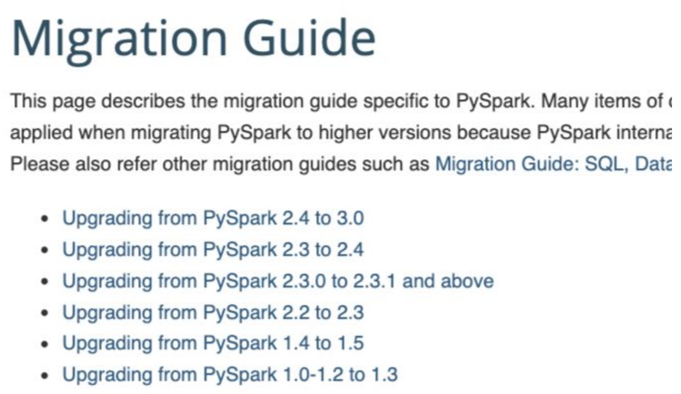
This page summarizes the basic steps require

- Installation
  - Python Version Supported
  - Using PyPI
  - Using Conda
  - Manually Downloading
  - Installing from Source
  - Dependencies
- Quickstart
  - DataFrame Creation
  - Viewing Data
  - Selecting and Accessing Data
  - Applying a Function
  - Grouping Data
  - Getting Data in/out
  - Working with SQL



User Guide

- Apache Arrow in PySpark
  - Ensure PyArrow Installed
  - Enabling for Conversion to/from Pandas
  - Pandas UDFs (a.k.a. Vectorized UDFs)
  - Pandas Function APIs
  - Usage Notes
- 3rd Party Python Packages
  - Using PySpark Native Features
  - Using Zipped Virtual Environment
  - Using PEX



Migration Guide

This page describes the migration guide specific to PySpark. Many items of
applied when migrating PySpark to higher versions because PySpark interna
Please also refer other migration guides such as Migration Guide: SQL, Data

- Upgrading from PySpark 2.4 to 3.0
- Upgrading from PySpark 2.3 to 2.4
- Upgrading from PySpark 2.3.0 to 2.3.1 and above
- Upgrading from PySpark 2.2 to 2.3
- Upgrading from PySpark 1.4 to 1.5
- Upgrading from PySpark 1.0-1.2 to 1.3

New useful pages

PySpark Type Hints

What are Python Type Hints?

```python
def greeting(name):
    return 'Hello ' + name
```

Typical Python codes

```python
def greeting(name: str) -> str:
    return 'Hello ' + name
```

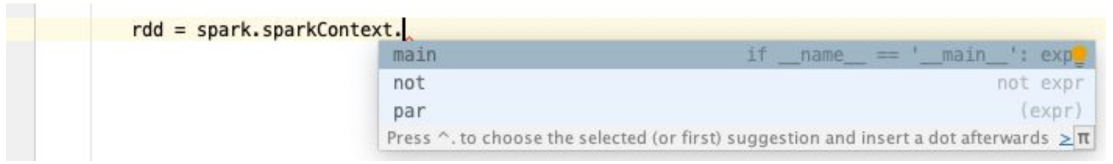
Python codes with type hints

```python
def greeting(name: str) -> str: ...
```

Stub syntax (.pyi file)

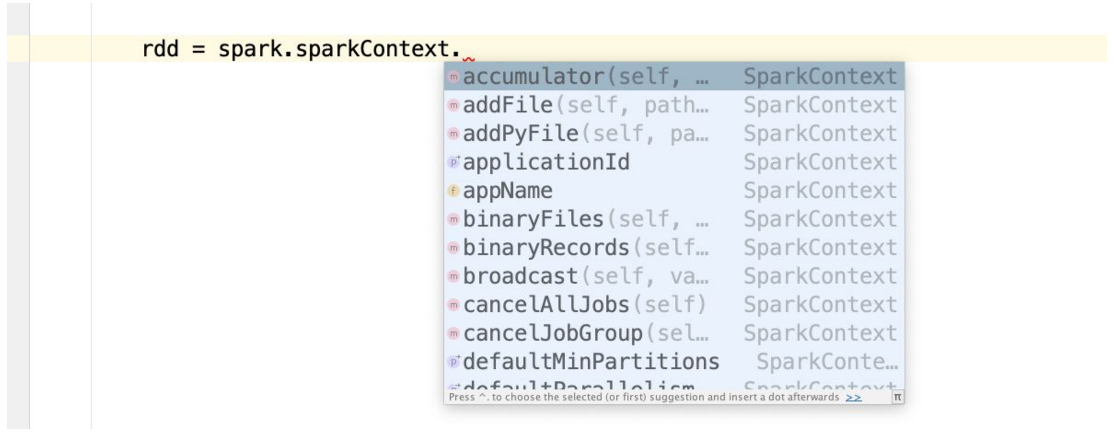
Why are Python Type Hints good?

- IDE Support
- Notebook Support
- Documentation
- Static error detection



```
rdd = spark.sparkContext.
  main                          if __name__ == '__main__': exp
  not                                                 not expr
  par                                                   (expr)
  Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

Before type hints

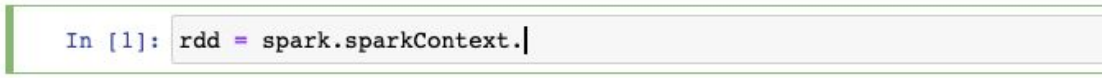


```
rdd = spark.sparkContext.
  accumulator(self, …   SparkContext
  addFile(self, path…   SparkContext
  addPyFile(self, pa…   SparkContext
  applicationId         SparkContext
  appName               SparkContext
  binaryFiles(self, …   SparkContext
  binaryRecords(self…   SparkContext
  broadcast(self, va…   SparkContext
  cancelAllJobs(self)   SparkContext
  cancelJobGroup(sel…   SparkContext
  defaultMinPartitions   SparkConte…
  Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π
```

After type hints

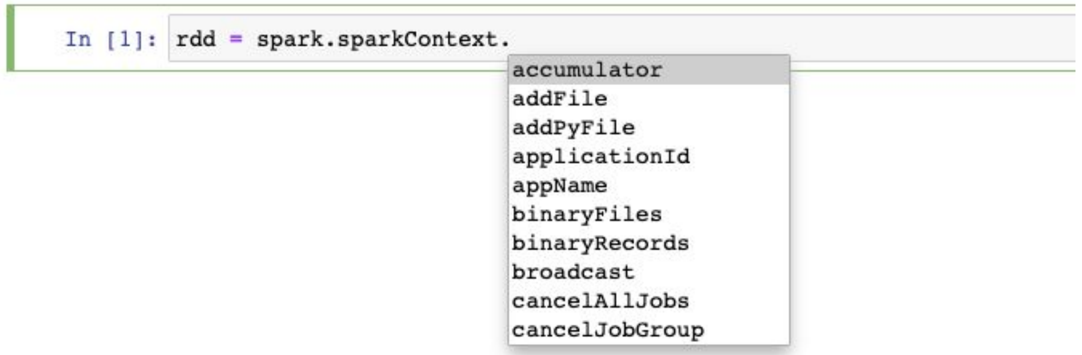
Why are Python Type Hints good?

- IDE Support
- Notebook Support
- Documentation
- Static error detection



```
In [1]: rdd = spark.sparkContext.
```

Before type hints

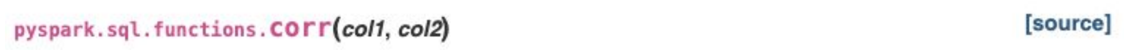


```
In [1]: rdd = spark.sparkContext.
                                 accumulator
                                 addFile
                                 addPyFile
                                 applicationId
                                 appName
                                 binaryFiles
                                 binaryRecords
                                 broadcast
                                 cancelAllJobs
                                 cancelJobGroup
```

After type hints

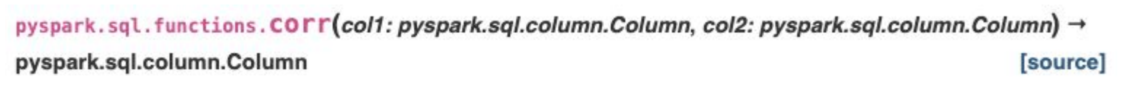
Why are Python Type Hints good?

- IDE Support
- Notebook Support
- Documentation
- Static error detection



pyspark.sql.functions.corr(*col1*, *col2*) [source]

Before type hints

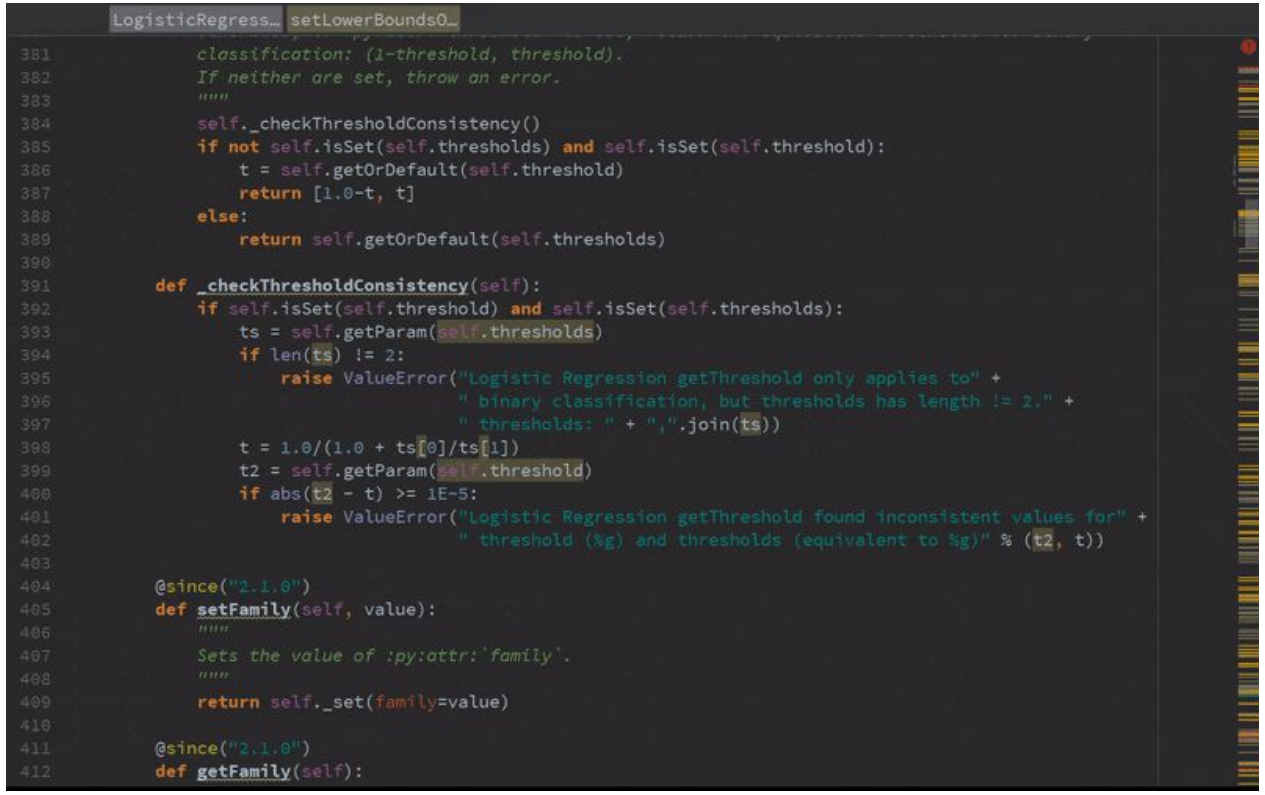


pyspark.sql.functions.corr(*col1: pyspark.sql.column.Column*, *col2: pyspark.sql.column.Column*) → pyspark.sql.column.Column [source]

After type hints

Why are Python Type Hints good?

- IDE Support
- Notebook Support
- Documentation
- Static error detection

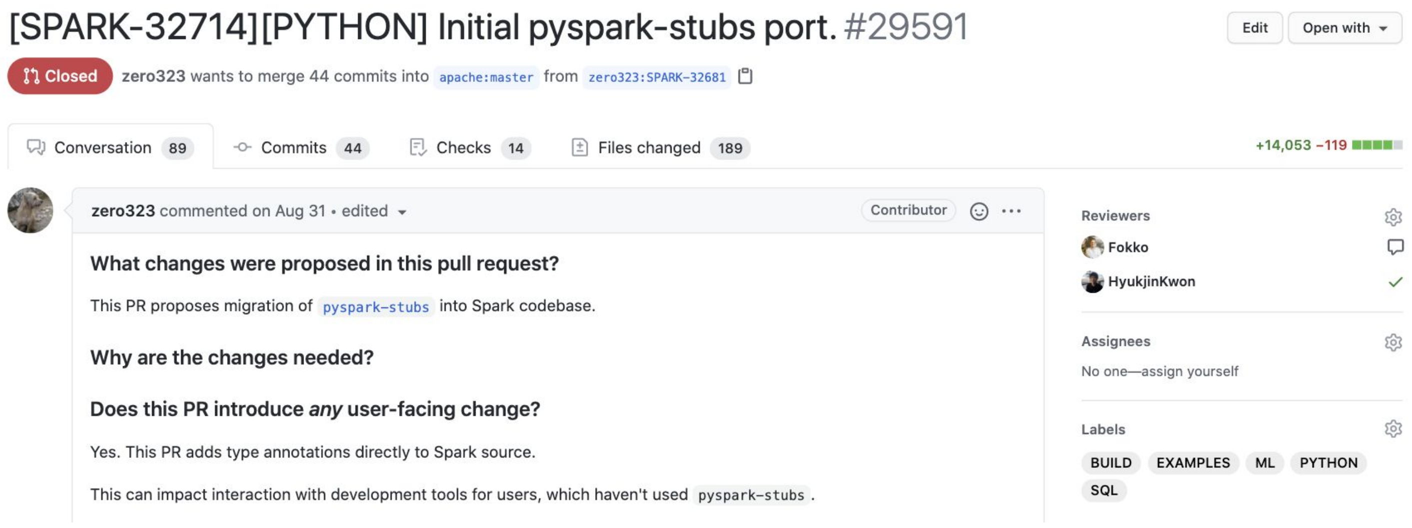


```
        classification: (1-threshold, threshold).
        If neither are set, throw an error.
        """
        self._checkThresholdConsistency()
        if not self.isSet(self.thresholds) and self.isSet(self.threshold):
            t = self.getOrDefault(self.threshold)
            return [1.0-t, t]
        else:
            return self.getOrDefault(self.thresholds)

    def _checkThresholdConsistency(self):
        if self.isSet(self.threshold) and self.isSet(self.thresholds):
            ts = self.getParam(self.thresholds)
            if len(ts) != 2:
                raise ValueError("Logistic Regression getThreshold only applies to" +
                                 " binary classification, but thresholds has length != 2." +
                                 " thresholds: " + ",".join(ts))
            t = 1.0/(1.0 + ts[0]/ts[1])
            t2 = self.getParam(self.threshold)
            if abs(t2 - t) >= 1E-5:
                raise ValueError("Logistic Regression getThreshold found inconsistent values for" +
                                 " threshold (%g) and thresholds (equivalent to %g)" % (t2, t))

    @since("2.1.0")
    def setFamily(self, value):
        """
        Sets the value of :py:attr:`family`.
        """
        return self._set(family=value)

    @since("2.1.0")
    def getFamily(self):
```

https://github.com/zero323/pyspark-stubs#motivation

Static error detection

Python Type Hints in PySpark



[SPARK-32714][PYTHON] Initial pyspark-stubs port. #29591

zero323 wants to merge 44 commits into `apache:master` from `zero323:SPARK-32681`

zero323 commented on Aug 31 • edited

What changes were proposed in this pull request?

This PR proposes migration of `pyspark-stubs` into Spark codebase.

Why are the changes needed?

Does this PR introduce *any* user-facing change?

Yes. This PR adds type annotations directly to Spark source.

This can impact interaction with development tools for users, which haven't used `pyspark-stubs` .

Built-in in the upcoming Apache Spark 3.1!

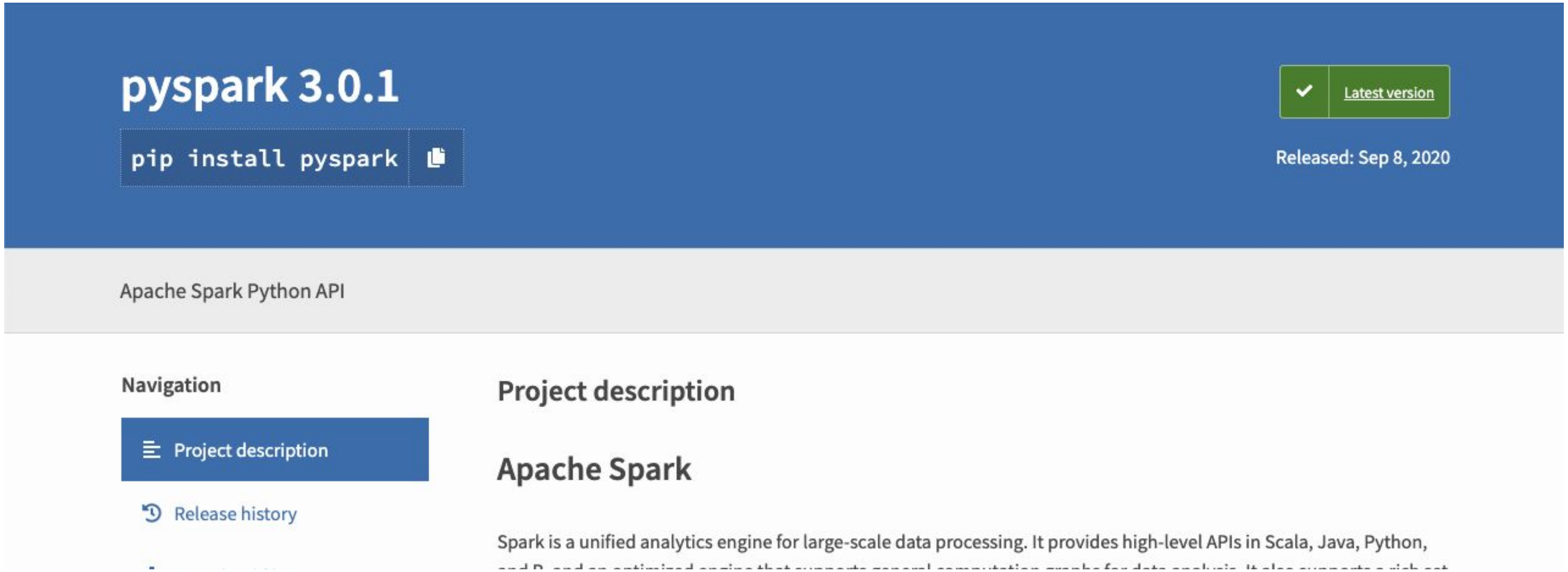
Community support: zero323/pyspark-stubs

User facing APIs only

Stub (.pyi) files

Installation Option for PyPI Users

PyPI Distribution



pyspark 3.0.1

```
pip install pyspark
```

Latest version

Released: Sep 8, 2020

Apache Spark Python API

Navigation

Project description

Release history

Project description

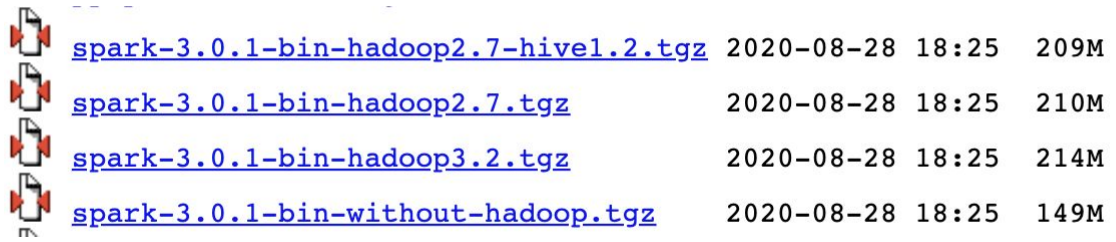
Apache Spark

Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Scala, Java, Python,

PySpark on PyPI

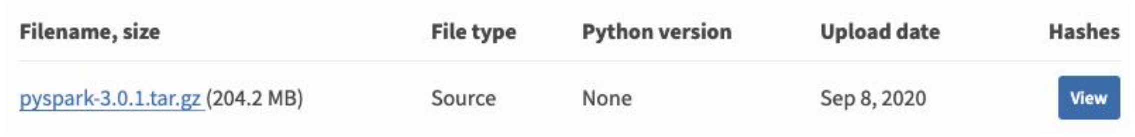
PyPI Distribution

- Multiple distributions available
  - Hadoop 2.7 and Hive 1.2
  - Hadoop 2.7 and Hive 2.3
  - Hadoop 3.2 and Hive 2.3
  - Hive 2.3 without Hadoop
- PySpark distribution in PyPI
  - Hadoop 2.7 and Hive 1.3



```
spark-3.0.1-bin-hadoop2.7-hive1.2.tgz 2020-08-28 18:25   209M
spark-3.0.1-bin-hadoop2.7.tgz         2020-08-28 18:25   210M
spark-3.0.1-bin-hadoop3.2.tgz         2020-08-28 18:25   214M
spark-3.0.1-bin-without-hadoop.tgz    2020-08-28 18:25   149M
```

Multiple distributions in Apache Mirror



Filename, size	File type	Python version	Upload date	Hashes
pyspark-3.0.1.tar.gz (204.2 MB)	Source	None	Sep 8, 2020	View

One distribution in PyPI

New Installation Options

```
HADOOP_VERSION=3.2 pip install pyspark
```

Spark with Hadoop 3.2

```
HADOOP_VERSION=2.7 pip install pyspark
```

Spark with Hadoop 2.7

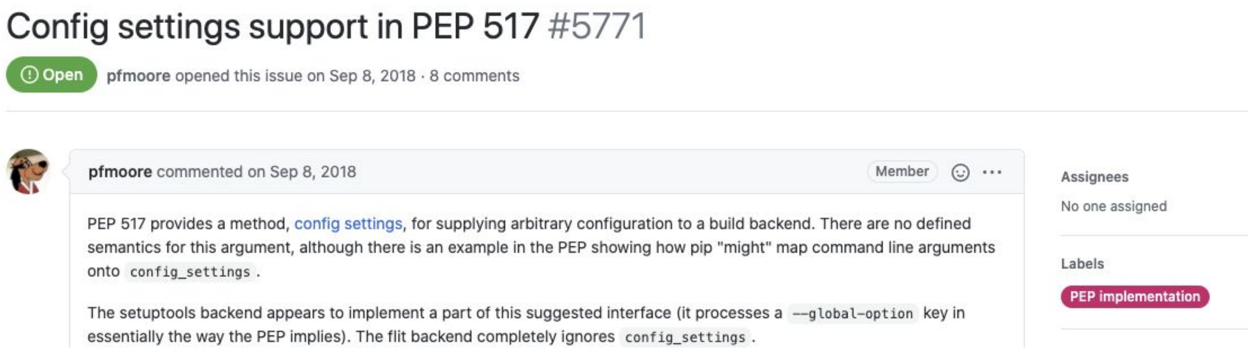
```
HADOOP_VERSION=without pip install pyspark
```

Spark without Hadoop

```
PYSPARK_RELEASE_MIRROR=http://mirror.apache-kr.org HADOOP_VERSION=2.7 pip install
```

Spark downloading from the specified mirror

Why not `pip --install-options`?



Config settings support in PEP 517 #5771

Open pfmoore opened this issue on Sep 8, 2018 · 8 comments

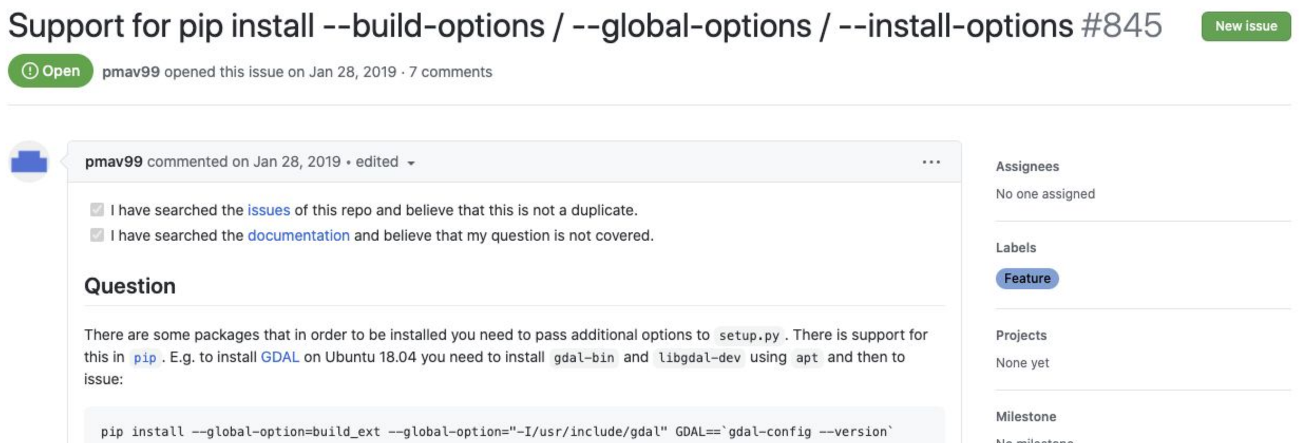
pfmoore commented on Sep 8, 2018 Member

PEP 517 provides a method, config settings, for supplying arbitrary configuration to a build backend. There are no defined semantics for this argument, although there is an example in the PEP showing how pip "might" map command line arguments onto `config_settings`.

The setuptools backend appears to implement a part of this suggested interface (it processes a `--global-option` key in essentially the way the PEP implies). The flit backend completely ignores `config_settings`.

Assignees
No one assigned

Labels
PEP implementation



Support for pip install --build-options / --global-options / --install-options #845 New issue

Open pmav99 opened this issue on Jan 28, 2019 · 7 comments

pmav99 commented on Jan 28, 2019 • edited

- I have searched the issues of this repo and believe that this is not a duplicate.
- I have searched the documentation and believe that my question is not covered.

Question

There are some packages that in order to be installed you need to pass additional options to `setup.py`. There is support for this in `pip`. E.g. to install GDAL on Ubuntu 18.04 you need to install `gdal-bin` and `libgdal-dev` using `apt` and then to issue:

```
pip install --global-option=build_ext --global-option="-I/usr/include/gdal" GDAL==`gdal-config --version`
```

Assignees
No one assigned

Labels
Feature

Projects
None yet

Milestone

Ongoing issues in pip

Roadmap

Roadmap

- Migrate to NumPy documentation style
  - Better classification
  - Better readability
  - Widely used

```
"""Specifies some hint on the current
:class:`DataFrame`.

:param name: A name of the hint.
:param parameters: Optional parameters.
:return: :class:`DataFrame`
```

reST style

```
"""Specifies some hint on the
current :class:`DataFrame`.

Parameters
----------
name : str
    A name of the hint.
parameters : dict, optional
    Optional parameters

Returns
-------
DataFrame
```

Numpydoc style

Roadmap

- Standardize warnings and exceptions
  - Classify the exception and warning types
  - Python friendly messages instead of JVM stack trace

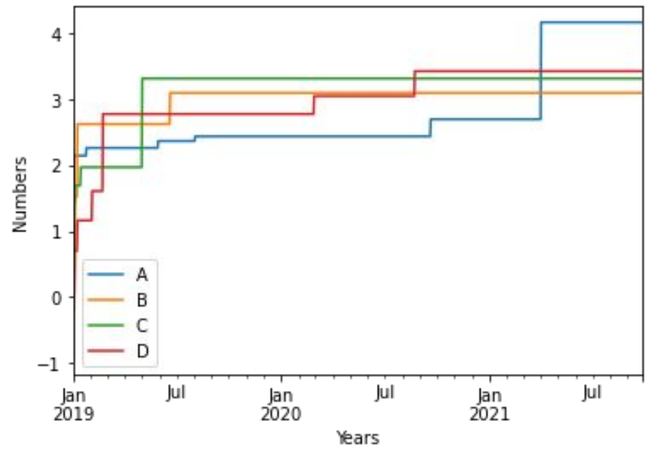
```
>>> spark.range(10).explain(1, 2)
```

```
Traceback (most recent call last):
    ...
Exception: extended and mode should not be set together.
```

Plain `Exception` being thrown

Roadmap

- Interoperability between NumPy, Koalas, other libraries
  - Common features in DataFrames
  - NumPy universe function
- Visualization and plotting
  - Make a chart from Spark DataFrame



Re-cap

Re-cap

- Python and PySpark are becoming more and more popular
- PySpark documentation is redesigned with many new pages
- Auto-completion and type checking in IDE and notebooks
- PySpark download options in PyPI

Re-cap: What's next?

- Migrate to NumPy documentation style
- Standardize warnings and exceptions
- Visualization
- Interoperability between NumPy, Koalas, other libraries

Question?