Apache Spark's Project ZEN

Project Zen is designed to make Python / Pyspark more friendlier to Apache Spark.
Apache Spark was designed based on Java based language but with the current massive rise in Pyspark usage, Spark needed significant changes for being more Pythonic.

In this document, I tried to draft all the efforts made and upcoming features under Project Zen.

Referred from the main Ticket

- Being Pythonic
  - Pandas UDF enhancements and type hints
  - Avoid dynamic function definitions, for example, at funcitons.py which makes IDEs unable to detect.
- Better and easier usability in PySpark
  - User-facing error message and warnings
  - Documentation
  - User guide
  - Better examples and API documentation, e.g. Koalas and pandas
- Better interoperability with other Python libraries
  - Visualization and plotting

    - Potentially better interface by leveraging Arrow
    - Compatibility with other libraries such as NumPy universal functions or pandas possibly by leveraging Koalas
- PyPI Installation
    - PySpark with Hadoop 3 support on PyPi
    - Better error handling

Done

Redesigning of Pandas UDF

There is this amazing Doc authored & Managed by Hyukjin Kwon that explains everything about this Topic.

Revisiting pandas UDF (Hyukjin Kwon)

In the past two years, the pandas UDFs are perhaps the most important changes to Spark for Python data science. However, these functionalities have evolved organically, leading to some inconsistencies and confusions among users. This document revisits UDF definition and naming.

Table of contents

SCALAR:

```
@pandas_udf(schema='...')
def func(c1: Series, c2: DataFrame) -> Series:
    pass  # DataFrame represents a struct column
```

SCALAR_ITER:

```
@pandas_udf(schema='...')
def func(iter: Iterator[Tuple[Series, DataFrame, ...]]) -> Iterator[Series]:
```

GROUPED_AGG:

```
@pandas_udf(schema='...')
def func(c1: Series, c2: DataFrame) -> int:
    pass  # DataFrame represents a struct column
```

MAP_ITER: this is not a pandas UDF anymore

```
def func(iter):
    for df in iter:
        yield df

df.mapInPandas(func, df.schema)
```

COGROUPED_MAP: this is not a pandas UDF anymore

```
def asof_join(left, right):
    return pd.merge_asof(left, right, on="...", by="...")

 df1.groupby("...").cogroup(df2.groupby("...")).applyInPandas(asof_join, schema="...")
```

Better error message for different Python and Pip Installation

A new better error message is thrown

```
Could not find valid SPARK_HOME while searching ['/Users',
'/usr/local/Cellar/python/3.7.5/Frameworks/Python.framework/Versions/3.7/bin']

Did you install PySpark via a package manager such as pip or Conda? If so,
PySpark was not found in your Python environment. It is possible your
Python environment does not properly bind with your package manager.

Please check your default 'python' and if you set PYSPARK_PYTHON and/or
PYSPARK_DRIVER_PYTHON environment variables, and see if you can import
PySpark, for example, 'python -c 'import pyspark'.

If you cannot import, you can install by using the Python executable directly,
```

for example, 'python -m pip install pyspark [--user]'. Otherwise, you can also explicitly set the Python executable, that has PySpark installed, to PYSPARK_PYTHON or PYSPARK_DRIVER_PYTHON environment variables, for example, 'PYSPARK_PYTHON=python3 pyspark'.
...

Pythonic Error Message, No more Java Error Message

Uglier Java Stack Traces are replaced by more Pythonic cleaner Error Message which a python developer can really understand.

Resources:
Check Holden's talk to know more.
https://github.com/apache/spark/pull/28661
https://github.com/apache/spark/pull/28749
https://github.com/apache/spark/pull/28762

Future

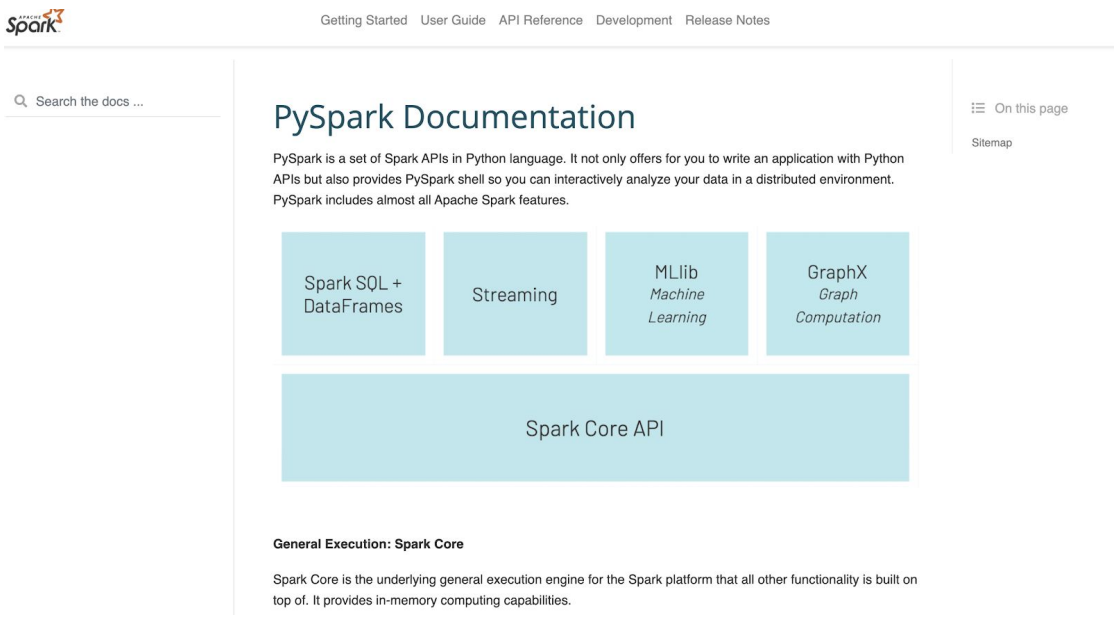
Improving Pyspark Documentation

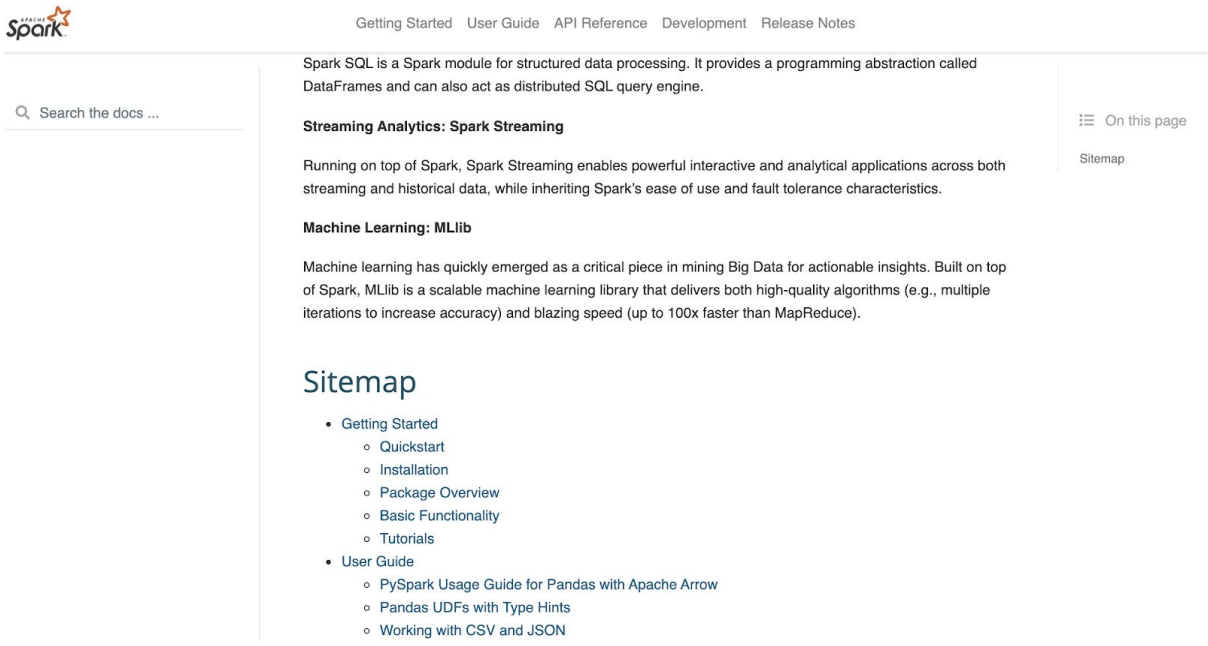
Currently the community doesnt appreciate Pyspark Documentation !
It's hard to read & understand.

Koalas Documentation is considered as reference

Current PySpark Documentation

Upcoming Pyspark Documentation





Pyspark with Hadoop 3.2

Even with Spark 3.0, Pyspark is still using Hadoop 2.7.0.
Pyspark binaries should have Hadoop 3.2 as default/choice.

This would enable the PyPI version to be compatible with session token authorisations and assist in accessing data residing in object stores with stronger encryption methods.
If not PyPI then as a tar file in the apache download archives at the least please.

Replace dictionary-based function definitions to proper functions in functions.py

Jira

Currently some functions in functions.py are defined by a dictionary. It programmatically defines the functions to the module; however, it makes some IDEs such as PyCharm don't detect.

Also, it makes hard to add proper examples into the docstrings.

Migration to Numpy Documentation Style

Current

```
def explain(self, extended=None, mode=None):
    """Prints the (logical and physical) plans to the console for debugging purpose.

    :param extended: boolean, default ``False``. If ``False``, prints only the physical plan.
        When this is a string without specifying the ``mode``, it works as the mode is
        specified.
    :param mode: specifies the expected output format of plans.

        * ``simple``: Print only a physical plan.
        * ``extended``: Print both logical and physical plans.
        * ``codegen``: Print a physical plan and generated codes if they are available.
        * ``cost``: Print a logical plan and statistics if they are available.
        * ``formatted``: Split explain output into two sections: a physical plan outline \
            and node details.

    >>> df.explain()
    == Physical Plan ==
    *(1) Scan ExistingRDD[age#0,name#1]

    >>> df.explain(True)
    == Parsed Logical Plan ==
    ...
    == Analyzed Logical Plan ==
    ...
    == Optimized Logical Plan ==
    ...
    == Physical Plan ==
    ...
```

Upcoming

```
def aggregate(self, func: Union[List[str], Dict[str, List[str]]]):
    """Aggregate using one or more operations over the specified axis.

    Parameters
    ----------
    func : dict or a list
         a dict mapping from column name (string) to
         aggregate functions (list of strings).
         If a list is given, the aggregation is performed against
         all columns.

    Returns
    -------
    DataFrame

    Notes
    -----
    `agg` is an alias for `aggregate`. Use the alias.

    See Also
    --------
    DataFrame.apply : Invoke function on DataFrame.
    DataFrame.transform : Only perform transforming type operations.
    DataFrame.groupby : Perform operations over groups.
    Series.aggregate : The equivalent function for Series.

    Examples
    --------
    >>> df = ks.DataFrame([[1, 2, 3],
    ...                    [4, 5, 6],
    ...                    [7, 8, 9],
    ...                    [np.nan, np.nan, np.nan]],
    ...                   columns=['A', 'B', 'C'])
```