Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Big Data Engineering (DJ19DSL604)

AY: 2022-23

Experiment 4

Name : Sarvagya Singh **SAPID : 60009200030**

BATCH : K1

(Data Processing)

Aim: Implement data processing using SPARK.

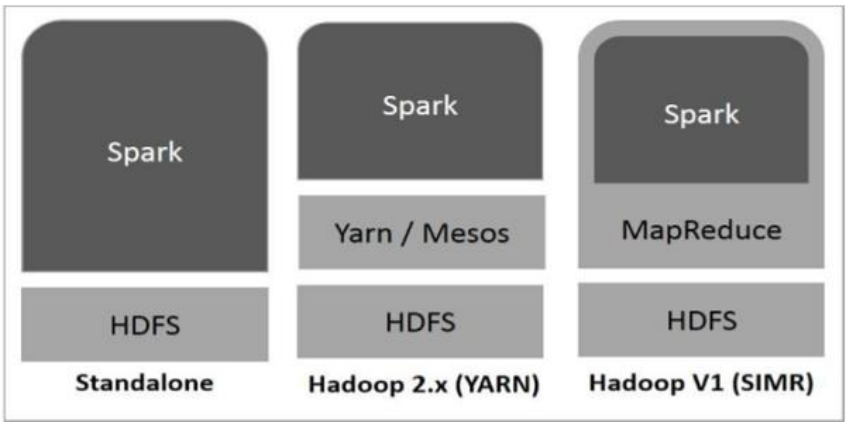
Theory:

Apache Spark

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Spark Built on Hadoop



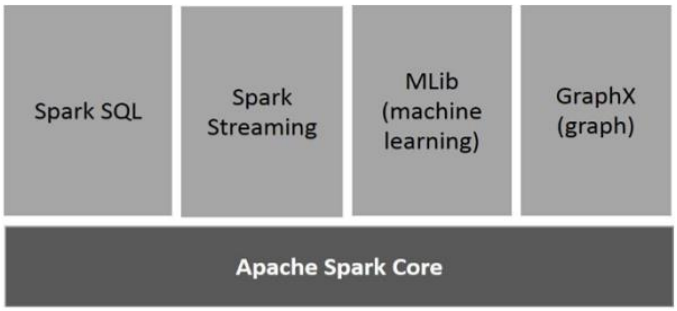
There are three ways of Spark deployment:

Standalone − Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

Hadoop Yarn − Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

Spark in MapReduce (SIMR) − Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Components of Spark



Apache Spark Core: Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL: Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

Spark Streaming: Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

MLlib (Machine Learning Library): MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout (before Mahout gained a Spark interface).

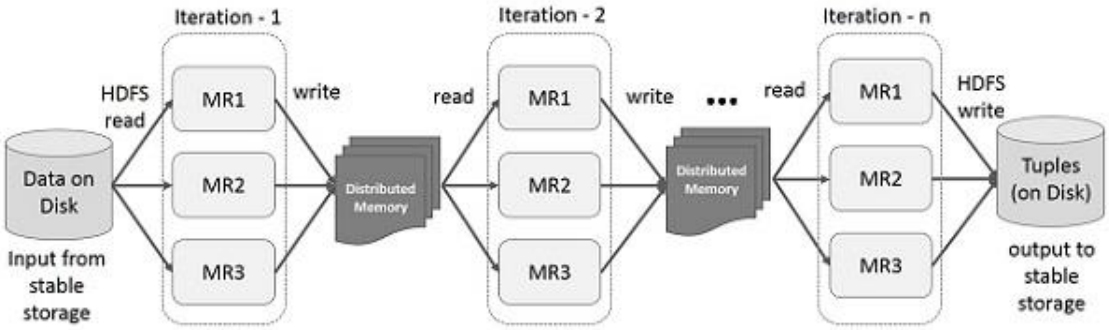
GraphX: GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant

collection of elements that can be operated on in parallel. There are two ways to create RDDs − parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.



Lab Assignment:

1. Installation of PySpark 3.3.2.
2. Create a PySpark Dataframe and implement the following on the dataframe:
    a. Viewing Data
    b. Selecting and Accessing Data
    c. Applying a Function
    d. Grouping Data
    e. Extracting data in various formats
3. Working on the dataframe using various SQL queries for processing data.
4. Working with pandas and pandas API on Spark for pre-processing the data.

Sparks

- https://spark.apache.org/docs/latest/api/python/getting_started/install.html

In [1]:
```
!pip install pyspark
```
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.3.2.tar.gz (281.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.5
  Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 kB 9.9 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.3.2-py2.py3-none-any.whl size=281824028 sha256=252803fa9a51c78734fc67ad47e9b9644603885cb8d310b40bad395e2ca6990c
  Stored in directory: /root/.cache/pip/wheels/6c/e3/9b/0525ce8a69478916513509d43693511463c6468db0de237c86
Successfully built pyspark
Installing collected packages: py4j, pyspark
  Attempting uninstall: py4j
    Found existing installation: py4j 0.10.9.7
    Uninstalling py4j-0.10.9.7:
      Successfully uninstalled py4j-0.10.9.7
Successfully installed py4j-0.10.9.5 pyspark-3.3.2
```

In [3]:
```
# Spark SQL
!pip install pyspark[sql]
# pandas API on Spark
!pip install pyspark[pandas_on_spark] plotly  # to plot your data, you can install plotly together.
```
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyspark[sql] in /usr/local/lib/python3.9/dist-packages (3.3.2)
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.9/dist-packages (from pyspark[sql]) (0.10.9.5)
Requirement already satisfied: pyarrow>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from pyspark[sql]) (9.0.0)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.9/dist-packages (from pyspark[sql]) (1.5.3)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->pyspark[sql]) (1.22.4)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->pyspark[sql]) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->pyspark[sql]) (2022.7.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.8.1->pandas>=1.0.5->pyspark[sql]) (1.16.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyspark[pandas_on_spark] in /usr/local/lib/python3.9/dist-packages (3.3.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.9/dist-packages (5.13.1)
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.9/dist-packages (from pyspark[pandas_on_spark]) (0.10.9.5)
Requirement already satisfied: pyarrow>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from pyspark[pandas_on_spark]) (9.0.0)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.9/dist-packages (from pyspark[pandas_on_spark]) (1.22.4)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.9/dist-packages (from pyspark[pandas_on_spark]) (1.5.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from plotly) (8.2.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->pyspark[pandas_on_spark]) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->pyspark[pandas_on_spark]) (2022.7.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.8.1->pandas>=1.0.5->pyspark[pandas_on_spark]) (1.16.0)
```

In [5]:
```
PYSPARK_HADOOP_VERSION=2
!pip install pyspark
```
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyspark in /usr/local/lib/python3.9/dist-packages (3.3.2)
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.9/dist-packages (from pyspark) (0.10.9.5)
```

In [7]:
```
PYSPARK_RELEASE_MIRROR='http://mirror.apache-kr.org'
PYSPARK_HADOOP_VERSION=2
# pip install
```

In [8]:
```
!pip install pyspark -v
```
```
Using pip 23.0.1 from /usr/local/lib/python3.9/dist-packages/pip (python 3.9)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyspark in /usr/local/lib/python3.9/dist-packages (3.3.2)
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.9/dist-packages (from pyspark) (0.10.9.5)
```

In [9]:
```
pip install "pyarrow>=4.0.0" --prefer-binary
```
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyarrow>=4.0.0 in /usr/local/lib/python3.9/dist-packages (9.0.0)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.9/dist-packages (from pyarrow>=4.0.0) (1.22.4)
```

In [10]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
```

In [11]:
```
from datetime import datetime, date
import pandas as pd
from pyspark.sql import Row

df = spark.createDataFrame([
    Row(a=1, b=2., c='string1', d=date(2000, 1, 1), e=datetime(2000, 1, 1, 12, 0)),
    Row(a=2, b=3., c='string2', d=date(2000, 2, 1), e=datetime(2000, 1, 2, 12, 0)),
    Row(a=4, b=5., c='string3', d=date(2000, 3, 1), e=datetime(2000, 1, 3, 12, 0))
])
df
```
Out[11]:
```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

In [12]:
```
df = spark.createDataFrame([
    (1, 2., 'string1', date(2000, 1, 1), datetime(2000, 1, 1, 12, 0)),
    (2, 3., 'string2', date(2000, 2, 1), datetime(2000, 1, 2, 12, 0)),
    (3, 4., 'string3', date(2000, 3, 1), datetime(2000, 1, 3, 12, 0))
], schema='a long, b double, c string, d date, e timestamp')
df
```
Out[12]:
```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

In [13]:
```
pandas_df = pd.DataFrame({
    'a': [1, 2, 3],
    'b': [2., 3., 4.],
    'c': ['string1', 'string2', 'string3'],
    'd': [date(2000, 1, 1), date(2000, 2, 1), date(2000, 3, 1)],
    'e': [datetime(2000, 1, 1, 12, 0), datetime(2000, 1, 2, 12, 0), datetime(2000, 1, 3, 12, 0)]
})
df = spark.createDataFrame(pandas_df)
df.head()
```
```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:474: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```
Out[13]:
```
Row(a=1, b=2.0, c='string1', d=datetime.date(2000, 1, 1), e=datetime.datetime(2000, 1, 1, 12, 0))
```

In [14]:
```
rdd = spark.sparkContext.parallelize([
    (1, 2., 'string1', date(2000, 1, 1), datetime(2000, 1, 1, 12, 0)),
    (2, 3., 'string2', date(2000, 2, 1), datetime(2000, 1, 2, 12, 0)),
    (3, 4., 'string3', date(2000, 3, 1), datetime(2000, 1, 3, 12, 0))
])
df = spark.createDataFrame(rdd, schema=['a', 'b', 'c', 'd', 'e'])
df
```
Out[14]:
```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

In [15]:
```
# All DataFrames above result same.
df.show()
df.printSchema()
```
```
+---+---+-------+----------+-------------------+
|  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|
+---+---+-------+----------+-------------------+

root
 |-- a: long (nullable = true)
 |-- b: double (nullable = true)
 |-- c: string (nullable = true)
 |-- d: date (nullable = true)
 |-- e: timestamp (nullable = true)
```

In [16]:
```
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
```

In [17]:
```
spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
df
```
Out[17]:

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

In [18]:
```
df.show(1, vertical=True)
```
```
-RECORD 0------------------
 a   | 1
 b   | 2.0
 c   | string1
 d   | 2000-01-01
 e   | 2000-01-01 12:00:00
only showing top 1 row
```

In [19]:
```
df.columns
```
Out[19]:
```
['a', 'b', 'c', 'd', 'e']
```

In [20]:
```
df.printSchema()
```
```
root
 |-- a: long (nullable = true)
 |-- b: double (nullable = true)
 |-- c: string (nullable = true)
 |-- d: date (nullable = true)
 |-- e: timestamp (nullable = true)
```

In [21]:
```
df.select("a", "b", "c").describe().show()
```
```
+-------+---+---+-------+
|summary|  a|  b|      c|
+-------+---+---+-------+
|  count|  3|  3|      3|
|   mean|2.0|3.0|   null|
| stddev|1.0|1.0|   null|
|    min|  1|2.0|string1|
|    max|  3|4.0|string3|
+-------+---+---+-------+
```

In [22]:
```
df.collect()
```
Out[22]:
```
[Row(a=1, b=2.0, c='string1', d=datetime.date(2000, 1, 1), e=datetime.datetime(2000, 1, 1, 12, 0)),
 Row(a=2, b=3.0, c='string2', d=datetime.date(2000, 2, 1), e=datetime.datetime(2000, 1, 2, 12, 0)),
 Row(a=3, b=4.0, c='string3', d=datetime.date(2000, 3, 1), e=datetime.datetime(2000, 1, 3, 12, 0))]
```

In [23]:
```
df.take(1)
```
Out[23]:
```
[Row(a=1, b=2.0, c='string1', d=datetime.date(2000, 1, 1), e=datetime.datetime(2000, 1, 1, 12, 0))]
```

In [24]:
```
df.toPandas()
```
```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning: Passing unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
```
Out[24]:

	a	b	c	d	e
0	1	2.0	string1	2000-01-01	2000-01-01 12:00:00
1	2	3.0	string2	2000-02-01	2000-01-02 12:00:00
2	3	4.0	string3	2000-03-01	2000-01-03 12:00:00

In [25]:
```
df.a
```
Out[25]:
```
Column<'a'>
```

In [26]:
```
from pyspark.sql import Column
from pyspark.sql.functions import upper

type(df.c) == type(upper(df.c)) == type(df.c.isNull())
```
Out[26]:
```
True
```

In [27]:
```
df.select(df.c).show()
```
```
+-------+
|      c|
+-------+
|string1|
|string2|
|string3|
+-------+
```

In [28]:
```
df.withColumn('upper_c', upper(df.c)).show()
```
```
+---+---+-------+----------+-------------------+-------+
|  a|  b|      c|         d|                  e|upper_c|
+---+---+-------+----------+-------------------+-------+
|  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|STRING1|
|  2|3.0|string2|2000-02-01|2000-01-02 12:00:00|STRING2|
|  3|4.0|string3|2000-03-01|2000-01-03 12:00:00|STRING3|
+---+---+-------+----------+-------------------+-------+
```

In [29]:
```
df.filter(df.a == 1).show()
```
```
+---+---+-------+----------+-------------------+
|  a|  b|      c|         d|                  e|
+---+---+-------+----------+-------------------+
|  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|
+---+---+-------+----------+-------------------+
```

In [30]:
```
import pandas as pd
from pyspark.sql.functions import pandas_udf

@pandas_udf('long')
def pandas_plus_one(series: pd.Series) -> pd.Series:
    # Simply plus one by using pandas Series.
    return series + 1

df.select(pandas_plus_one(df.a)).show()
```
```
+------------------+
|pandas_plus_one(a)|
+------------------+
|                 2|
|                 3|
|                 4|
+------------------+
```

In [31]:
```
def pandas_filter_func(iterator):
    for pandas_df in iterator:
        yield pandas_df[pandas_df.a == 1]

df.mapInPandas(pandas_filter_func, schema=df.schema).show()
```
```
+---+---+-------+----------+-------------------+
|  a|  b|      c|         d|                  e|
+---+---+-------+----------+-------------------+
|  1|2.0|string1|2000-01-01|2000-01-01 12:00:00|
+---+---+-------+----------+-------------------+
```

In [32]:
```
df = spark.createDataFrame([
    ['red', 'banana', 1, 10], ['blue', 'banana', 2, 20], ['red', 'carrot', 3, 30],
    ['blue', 'grape', 4, 40], ['red', 'carrot', 5, 50], ['black', 'carrot', 6, 60],
    ['red', 'banana', 7, 70], ['red', 'grape', 8, 80]], schema=['color', 'fruit', 'v1', 'v2'])
df.show()
```
```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|  red|banana|  1| 10|
| blue|banana|  2| 20|
|  red|carrot|  3| 30|
| blue| grape|  4| 40|
|  red|carrot|  5| 50|
|black|carrot|  6| 60|
|  red|banana|  7| 70|
|  red| grape|  8| 80|
+-----+------+---+---+
```

In [33]:
```
df.groupby('color').avg().show()
```
```
+-----+-------+-------+
|color|avg(v1)|avg(v2)|
+-----+-------+-------+
|  red|    4.8|   48.0|
| blue|    3.0|   30.0|
|black|    6.0|   60.0|
+-----+-------+-------+
```

In [34]:
```
def plus_mean(pandas_df):
    return pandas_df.assign(v1=pandas_df.v1 - pandas_df.v1.mean())

df.groupby('color').applyInPandas(plus_mean, schema=df.schema).show()
```
```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|black|carrot|  0| 60|
| blue|banana| -1| 20|
| blue| grape|  1| 40|
|  red|banana| -3| 10|
|  red|carrot| -1| 30|
|  red|carrot|  0| 50|
|  red|banana|  2| 70|
|  red| grape|  3| 80|
+-----+------+---+---+
```

In [35]:
```
df1 = spark.createDataFrame(
    [(20000101, 1, 1.0), (20000101, 2, 2.0), (20000102, 1, 3.0), (20000102, 2, 4.0)],
    ('time', 'id', 'v1'))

df2 = spark.createDataFrame(
    [(20000101, 1, 'x'), (20000101, 2, 'y')],
    ('time', 'id', 'v2'))

def asof_join(l, r):
    return pd.merge_asof(l, r, on='time', by='id')

df1.groupby('id').cogroup(df2.groupby('id')).applyInPandas(
    asof_join, schema='time int, id int, v1 double, v2 string').show()
```
```
+--------+---+---+---+
|    time| id| v1| v2|
+--------+---+---+---+
|20000101|  1|1.0|  x|
|20000102|  1|3.0|  x|
|20000101|  2|2.0|  y|
|20000102|  2|4.0|  y|
+--------+---+---+---+
```

In [36]:
```
df.write.csv('foo.csv', header=True)
spark.read.csv('foo.csv', header=True).show()
```
```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|  red|banana|  1| 10|
| blue|banana|  2| 20|
|  red|carrot|  3| 30|
| blue| grape|  4| 40|
|  red|carrot|  5| 50|
|black|carrot|  6| 60|
|  red|banana|  7| 70|
|  red| grape|  8| 80|
+-----+------+---+---+
```

In [37]:
```
df.write.parquet('bar.parquet')
spark.read.parquet('bar.parquet').show()
```
```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|  red|carrot|  5| 50|
|black|carrot|  6| 60|
|  red|banana|  7| 70|
|  red| grape|  8| 80|
|  red|banana|  1| 10|
| blue|banana|  2| 20|
|  red|carrot|  3| 30|
| blue| grape|  4| 40|
+-----+------+---+---+
```

In [38]:
```
df.write.orc('zoo.orc')
spark.read.orc('zoo.orc').show()
```
```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|  red|carrot|  5| 50|
|black|carrot|  6| 60|
|  red|banana|  7| 70|
|  red| grape|  8| 80|
|  red|banana|  1| 10|
| blue|banana|  2| 20|
|  red|carrot|  3| 30|
| blue| grape|  4| 40|
+-----+------+---+---+
```

In [39]:
```
df.createOrReplaceTempView("tableA")
spark.sql("SELECT count(*) from tableA").show()
```
```
+--------+
|count(1)|
+--------+
|       8|
+--------+
```

In [40]:
```
@pandas_udf("integer")
def add_one(s: pd.Series) -> pd.Series:
    return s + 1

spark.udf.register("add_one", add_one)
spark.sql("SELECT add_one(v1) FROM tableA").show()
```
```
+-----------+
|add_one(v1)|
+-----------+
|          2|
|          3|
|          4|
|          5|
|          6|
|          7|
|          8|
|          9|
+-----------+
```

In [41]:
```
from pyspark.sql.functions import expr

df.selectExpr('add_one(v1)').show()
df.select(expr('count(*)') > 0).show()
```
```
+-----------+
|add_one(v1)|
+-----------+
|          2|
|          3|
|          4|
|          5|
|          6|
|          7|
|          8|
|          9|
+-----------+

+--------------+
|(count(1) > 0)|
+--------------+
|          true|
+--------------+
```

In [ ]:
```
!pip install pyspark
```

In [3]:
```
import pandas as pd
import numpy as np
import pyspark.pandas as ps
from pyspark.sql import SparkSession
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/__init__.py:49: UserWarning: 'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is required to set this environment variable to '1' in both driver and exe
cutor sides if you use pyarrow>=2.0.0. pandas-on-Spark will set it for you but it does not work if there is a Spark context already launched.
  warnings.warn(
```

In [6]:
```
s = ps.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

Out[6]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

In [8]:
```
psdf = ps.DataFrame(
    {'a': [1, 2, 3, 4, 5, 6],
     'b': [100, 200, 300, 400, 500, 600],
     'c': ["one", "two", "three", "four", "five", "six"]},
    index=[10, 20, 30, 40, 50, 60]);

psdf
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

Out[8]:

	a	b	c
10	1	100	one
20	2	200	two
30	3	300	three
40	4	400	four
50	5	500	five
60	6	600	six

In [9]:
```
dates = pd.date_range('20130101', periods=6)
dates
```

Out[9]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [10]:
```
pdf = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
pdf
```

Out[10]:

	A	B	C	D
2013-01-01	1.499612	-0.754830	-0.335977	-1.611224
2013-01-02	-2.611944	0.262682	-0.841523	-0.693847
2013-01-03	-0.826292	-0.582009	-1.074521	0.770667
2013-01-04	0.129667	1.823241	0.749956	-0.070218
2013-01-05	-2.580936	0.253384	0.897712	0.190035
2013-01-06	-0.107938	0.657611	-0.074893	-1.876522

In [11]:
```
psdf = ps.from_pandas(pdf)
type(psdf)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

Out[11]:
```
pyspark.pandas.frame.DataFrame
```

In [12]:
```
psdf
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning: Passing unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
```

Out[12]:

	A	B	C	D
2013-01-01	1.499612	-0.754830	-0.335977	-1.611224
2013-01-02	-2.611944	0.262682	-0.841523	-0.693847
2013-01-03	-0.826292	-0.582009	-1.074521	0.770667
2013-01-04	0.129667	1.823241	0.749956	-0.070218
2013-01-05	-2.580936	0.253384	0.897712	0.190035
2013-01-06	-0.107938	0.657611	-0.074893	-1.876522

In [13]:
```
spark = SparkSession.builder.getOrCreate()
```

In [14]:
```
sdf = spark.createDataFrame(pdf)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:474: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

In [15]:
```
sdf.show()
```

```
+--------------------+-------------------+--------------------+--------------------+
|                   A|                  B|                   C|                   D|
+--------------------+-------------------+--------------------+--------------------+
|  1.4996124515137383|-0.7548303714764133| -0.3359774084275556| -1.6112242553226364|
| -2.6119444753574768| 0.2626815948269787| -0.8415230802463888|  -0.693847386606434|
| -0.8262916182337151|-0.5820089557862504|  -1.074520656750109|   0.770666884677625|
| 0.12966739326601223| 1.8232412954458892|  0.7499559339946512|-0.07021843199692183|
|  -2.5809355597547312|0.25338424025116557|  0.8977120418410863| 0.19003481444852047|
|-0.10793751031701546| 0.6576111463264421|-0.07489276028318853| -1.8765217169921882|
+--------------------+-------------------+--------------------+--------------------+
```

In [16]:
```
psdf = sdf.pandas_api()
```

In [17]:
```
psdf
```

Out[17]:

	A	B	C	D
0	1.499612	-0.754830	-0.335977	-1.611224
1	-2.611944	0.262682	-0.841523	-0.693847
2	-0.826292	-0.582009	-1.074521	0.770667
3	0.129667	1.823241	0.749956	-0.070218
4	-2.580936	0.253384	0.897712	0.190035
5	-0.107938	0.657611	-0.074893	-1.876522

In [18]:
```
psdf.dtypes
```

Out[18]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

In [19]:
```
psdf.head()
```

Out[19]:

	A	B	C	D
0	1.499612	-0.754830	-0.335977	-1.611224
1	-2.611944	0.262682	-0.841523	-0.693847
2	-0.826292	-0.582009	-1.074521	0.770667
3	0.129667	1.823241	0.749956	-0.070218
4	-2.580936	0.253384	0.897712	0.190035

In [20]:
```
psdf.index
```

Out[20]:
```
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
```

In [21]:
```
psdf.columns
```

Out[21]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [22]:
```
psdf.to_numpy()
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/utils.py:975: PandasAPIOnSparkAdviceWarning: `to_numpy` loads all data into the driver's memory. It should only be used if the resulting NumPy ndarray is expected to
be small.
  warnings.warn(message, PandasAPIOnSparkAdviceWarning)
```

Out[22]:
```
array([[ 1.49961245, -0.75483037, -0.33597741, -1.61122426],
       [-2.61194448,  0.26268159, -0.84152308, -0.69384739],
       [-0.82629162, -0.58200896, -1.07452066,  0.77066688],
       [ 0.12966739,  1.8232413 ,  0.74995593, -0.07021843],
       [-2.5809356 ,  0.25338424,  0.89771204,  0.19003481],
       [-0.10793751,  0.65761115, -0.07489276, -1.87652172]])
```

In [23]:
```
psdf.describe()
```

Out[23]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.749638	0.276680	-0.113208	-0.548518
std	1.616898	0.931564	0.809033	1.041973
min	-2.611944	-0.754830	-1.074521	-1.876522
25%	-2.580936	-0.582009	-0.841523	-1.611224
50%	-0.826292	0.253384	-0.335977	-0.693847
75%	0.129667	0.657611	0.749956	0.190035
max	1.499612	1.823241	0.897712	0.770667

In [24]:
```
psdf.T
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

Out[24]:

	0	1	2	3	4	5
A	1.499612	-2.611944	-0.826292	0.129667	-2.580936	-0.107938
B	-0.754830	0.262682	-0.582009	1.823241	0.253384	0.657611
C	-0.335977	-0.841523	-1.074521	0.749956	0.897712	-0.074893
D	-1.611224	-0.693847	0.770667	-0.070218	0.190035	-1.876522

In [25]:
```
psdf.sort_index(ascending=False)
```

Out[25]:

	A	B	C	D
5	-0.107938	0.657611	-0.074893	-1.876522
4	-2.580936	0.253384	0.897712	0.190035
3	0.129667	1.823241	0.749956	-0.070218
2	-0.826292	-0.582009	-1.074521	0.770667
1	-2.611944	0.262682	-0.841523	-0.693847
0	1.499612	-0.754830	-0.335977	-1.611224

In [26]:
```
psdf.sort_values(by='B')
```

Out[26]:

	A	B	C	D
0	1.499612	-0.754830	-0.335977	-1.611224
2	-0.826292	-0.582009	-1.074521	0.770667
4	-2.580936	0.253384	0.897712	0.190035
1	-2.611944	0.262682	-0.841523	-0.693847
5	-0.107938	0.657611	-0.074893	-1.876522
3	0.129667	1.823241	0.749956	-0.070218

In [27]:
```
pdf1 = pdf.reindex(index=dates[0:4], columns=list(pdf.columns) + ['E'])
```

In [28]:
```
pdf1.loc[dates[0]:dates[1], 'E'] = 1
```

In [29]:
```
psdf1 = ps.from_pandas(pdf1)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

In [29]:
```
psdf1
```

In [30]:
```
psdf1.dropna(how='any')
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning: Passing unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
```

Out[30]:

	A	B	C	D	E
2013-01-01	1.499612	-0.754830	-0.335977	-1.611224	1.0
2013-01-02	-2.611944	0.262682	-0.841523	-0.693847	1.0

In [31]:
```
psdf1.fillna(value=5)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:248: FutureWarning: Passing unit-less datetime64 dtype to .astype is deprecated and will raise in a future version. Pass 'datetime64[ns]' instead
  series = series.astype(t, copy=False)
```

Out[31]:

	A	B	C	D	E
2013-01-01	1.499612	-0.754830	-0.335977	-1.611224	1.0
2013-01-02	-2.611944	0.262682	-0.841523	-0.693847	1.0
2013-01-03	-0.826292	-0.582009	-1.074521	0.770667	5.0
2013-01-04	0.129667	1.823241	0.749956	-0.070218	5.0

In [32]:
```
psdf.mean()
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/pandas/internal.py:1573: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  fields = [
/usr/local/lib/python3.9/dist-packages/pyspark/sql/pandas/conversion.py:486: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for column, series in pdf.iteritems():
```

Out[32]:
```
A   -0.749638
B    0.276680
C   -0.113208
D   -0.548518
dtype: float64
```

In [33]:
```
prev = spark.conf.get("spark.sql.execution.arrow.pyspark.enabled")  # Keep its default value.
ps.set_option("compute.default_index_type", "distributed")  # Use default index prevent overhead.
import warnings
warnings.filterwarnings("ignore")  # Ignore warnings coming from Arrow optimizations.
```

In [34]:
```
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", True)
%timeit ps.range(300000).to_pandas()
```

```
561 ms ± 174 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [35]:
```
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", False)
%timeit ps.range(300000).to_pandas()
```

```
1.34 s ± 453 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [36]:
```
ps.reset_option("compute.default_index_type")
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", prev)  # Set its default value back.
```

In [37]:
```
psdf = ps.DataFrame({'A': ['foo', 'bar', 'foo', 'bar',
                           'foo', 'bar', 'foo', 'foo'],
                     'B': ['one', 'one', 'two', 'three',
                           'two', 'two', 'one', 'three'],
                     'C': np.random.randn(8),
                     'D': np.random.randn(8)})
```

In [38]:
```
psdf
```

Out[38]:

	A	B	C	D
0	foo	one	0.010864	1.261251
1	bar	one	0.027523	-1.594322
2	foo	two	-0.054282	0.986570
3	bar	three	-0.143679	1.506730
4	foo	two	-0.691867	-0.287832
5	bar	two	-1.066225	0.049957
6	foo	one	-0.368457	-2.584860
7	foo	three	-0.416395	-2.274655

In [39]:
```
psdf.groupby('A').sum()
```

Out[39]:

	C	D
A		
bar	-1.182381	-0.037634
foo	-1.520136	-2.899527

In [40]:
```
psdf.groupby(['A', 'B']).sum()
```

Out[40]:

		C	D
A	B		
foo	one	-0.357592	-1.323610
	two	-0.746148	0.698738
bar	three	-0.143679	1.506730
	one	0.027523	-1.594322
foo	three	-0.416395	-2.274655
bar	two	-1.066225	0.049957

In [41]:
```
pser = pd.Series(np.random.randn(1000),
                 index=pd.date_range('1/1/2000', periods=1000))
```

In [42]:
```
psser = ps.Series(pser)
```

In [43]:
```
psser = psser.cummax()
```

In [44]:
```
psser.plot()
```

In [45]:
```
pdf = pd.DataFrame(np.random.randn(1000, 4), index=pser.index,
                   columns=['A', 'B', 'C', 'D'])
```

In [46]:
```
psdf = ps.from_pandas(pdf)
```

In [47]:
```
psdf = psdf.cummax()
```

In [48]:
```
psdf.plot()
```

In [49]:
```
psdf.to_csv('foo.csv')
ps.read_csv('foo.csv').head(10)
```

Out[49]:

	A	B	C	D
0	-0.762958	0.034911	-0.918362	1.185877
1	0.147295	0.034911	-0.918362	1.185877
2	0.147295	0.249062	-0.880703	1.185877
3	0.547372	1.511899	-0.610942	1.185877
4	0.547372	1.511899	0.736316	1.185877
5	1.001431	1.511899	0.736316	1.185877
6	1.001431	1.511899	0.736316	1.185877
7	1.001431	1.511899	0.736316	1.185877
8	1.285792	1.511899	0.938824	1.185877
9	1.285792	1.511899	0.938824	1.185877

In [50]:
```
psdf.to_parquet('bar.parquet')
ps.read_parquet('bar.parquet').head(10)
```

Out[50]:

	A	B	C	D
0	-0.762958	0.034911	-0.918362	1.185877
1	0.147295	0.034911	-0.918362	1.185877
2	0.147295	0.249062	-0.880703	1.185877
3	0.547372	1.511899	-0.610942	1.185877
4	0.547372	1.511899	0.736316	1.185877
5	1.001431	1.511899	0.736316	1.185877
6	1.001431	1.511899	0.736316	1.185877
7	1.001431	1.511899	0.736316	1.185877
8	1.285792	1.511899	0.938824	1.185877
9	1.285792	1.511899	0.938824	1.185877

In [51]:
```
psdf.to_spark_io('zoo.orc', format="orc")
ps.read_spark_io('zoo.orc', format="orc").head(10)
```

Out[51]:

	A	B	C	D
0	-0.762958	0.034911	-0.918362	1.185877
1	0.147295	0.034911	-0.918362	1.185877
2	0.147295	0.249062	-0.880703	1.185877
3	0.547372	1.511899	-0.610942	1.185877
4	0.547372	1.511899	0.736316	1.185877
5	1.001431	1.511899	0.736316	1.185877
6	1.001431	1.511899	0.736316	1.185877
7	1.001431	1.511899	0.736316	1.185877
8	1.285792	1.511899	0.938824	1.185877
9	1.285792	1.511899	0.938824	1.185877

In [ ]: