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: 2023-24

Experiment 5

(Data Processing)

Name: Kresha Shah

SAP ID: 60009220080

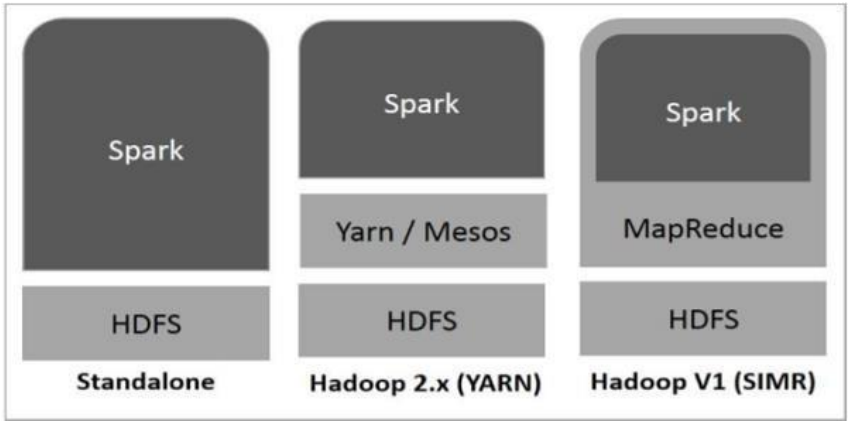
Aim: Implement data processing using SPARK.

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**



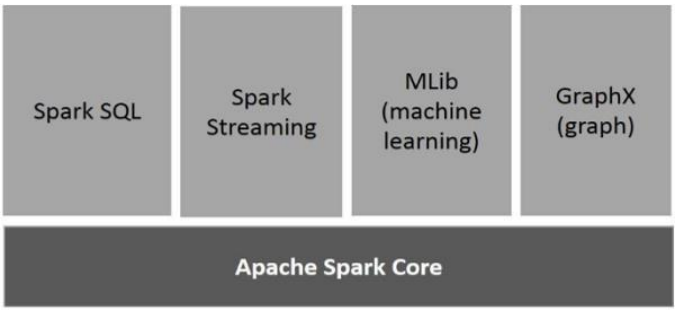
There are three ways of Spark deployment:

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 preinstallation 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.

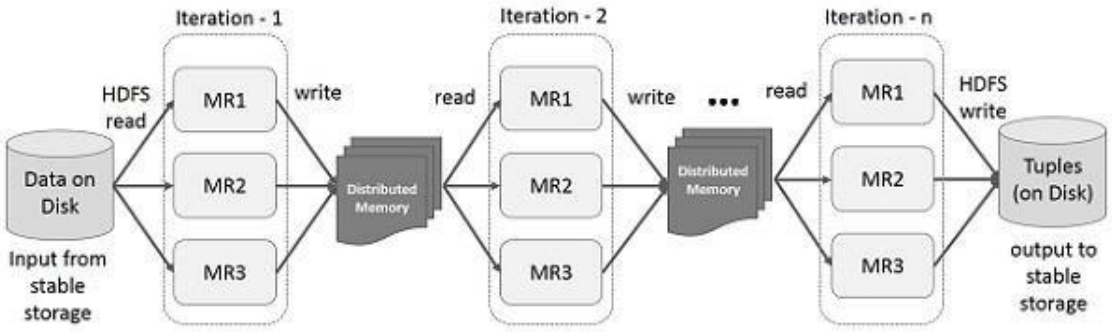
MLlib (Machine Learning Library): MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times 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

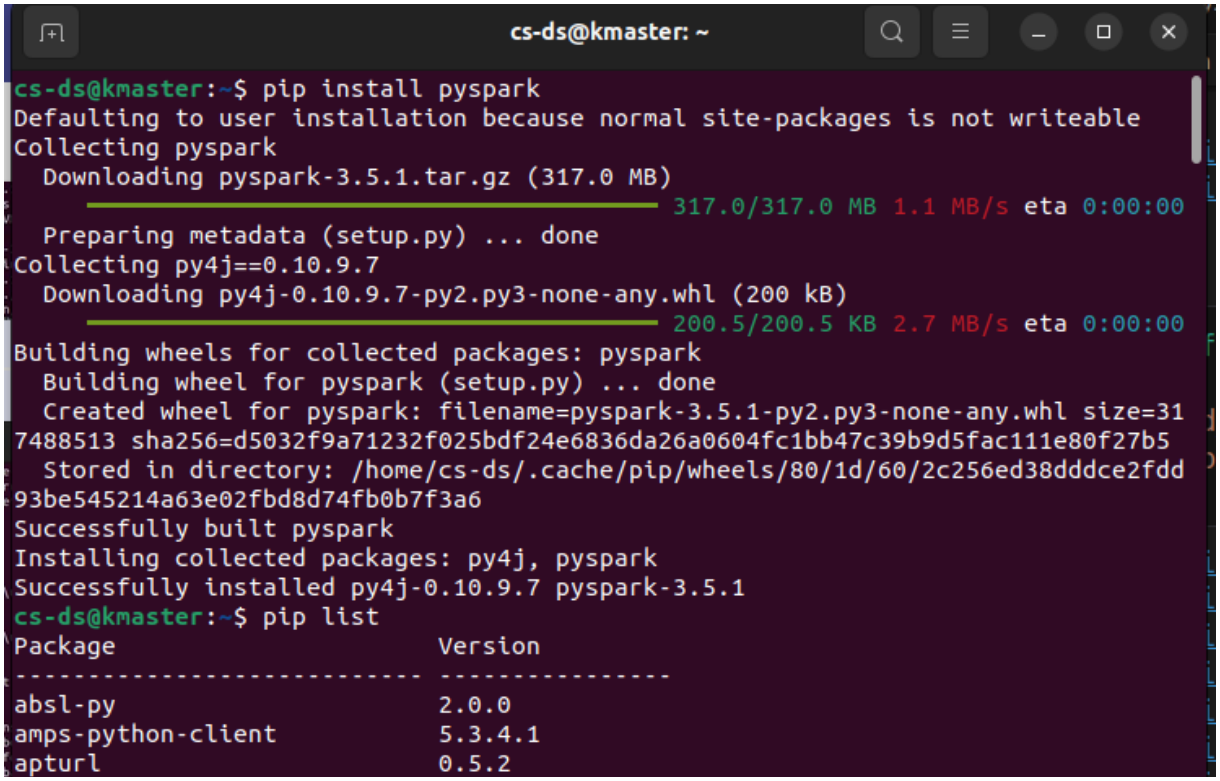
Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects,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

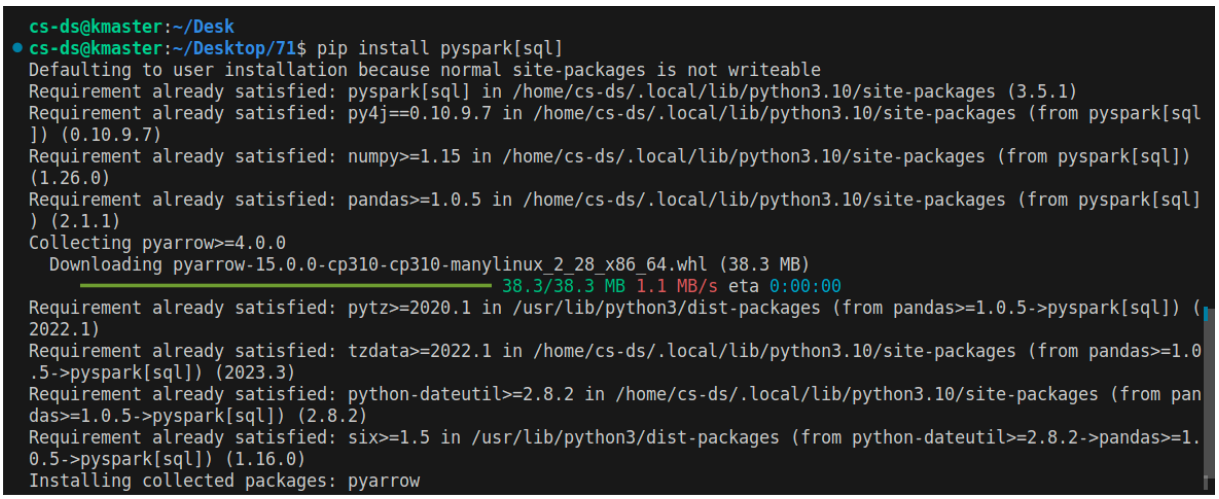
Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs − parallelizing 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

Data Frame Creation

```
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
```

```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

```
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
```

```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

```
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
```

```
DataFrame[a: bigint, b: double, c: string, d: date, e: timestamp]
```

```
    # 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)
```

Viewing Data

```
    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
```

```
spark.conf.set('spark.sql.repl.eagerEval.enabled', True)
df
```

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

```
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
```

```
df.columns
```

```
['a', 'b', 'c', 'd', 'e']
```

```
df.printSchema()
```

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

[12]
```
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|
+-------+---+---+-------+
```

[13]
```
df.collect()
```

```
[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))]
```

[14]
```
df.take(1)
```

```
[Row(a=1, b=2.0, c='string1', d=datetime.date(2000, 1, 1), e=datetime.datetime(2000, 1, 1, 12, 0))]
```

[15]
```
df.toPandas()
```

Python

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:563: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isin
  if not is_datetime64tz_dtype(pser.dtype):
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:379: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isin
  if is_datetime64tz_dtype(s.dtype):
```

	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

b. Selecting and Accessing Data

Selecting and Accessing Data

```
[16]
df.a
```

```
Column<'a'>
```

```
[18]
from pyspark.sql import Column
from pyspark.sql.functions import upper

type(df.c) == type(upper(df.c)) == type(df.c.isNull())
```

```
True
```

```
[19]
df.select(df.c).show()
```

```
+-------+
|      c|
+-------+
|string1|
|string2|
|string3|
+-------+
```

```
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|
+---+---+-------+----------+-------------------+-------+
```

```
[21]
df.filter(df.a == 1).show()
```

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

c. Applying a Function

Applying a Function

```
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()
```

Python

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pandas/serializers.py:224: FutureWarning: is_categorical_dtype is deprecated and will be remo
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pandas/serializers.py:224: FutureWarning: is_categorical_dtype is deprecated and will be remo
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pandas/serializers.py:224: FutureWarning: is_categorical_dtype is deprecated and will be remo
+-----------------+
|pandas_plus_one(a)|
+-----------------+
|                2|
|                3|
|                4|
+-----------------+
```

```
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|
+---+---+-------+----------+-------------------+
```

d. Grouping Data

Grouping Data

[26]
```
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|
+-----+------+---+---+
```

[27]
```
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|
+-----+-------+-------+
```

```
    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|
+-----+------+---+---+

/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/pyspark/sql/pa
```

```
    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 merge_ordered(l, r):
        return pd.merge_ordered(l, r)

    df1.groupby('id').cogroup(df2.groupby('id')).applyInPandas(
        merge_ordered, schema='time int, id int, v1 double, v2 string').show()
```

```
+--------+---+---+----+
|    time| id| v1|  v2|
+--------+---+---+----+
|20000101|  1|1.0|   x|
|20000102|  1|3.0|NULL|
|20000101|  2|2.0|   y|
|20000102|  2|4.0|NULL|
+--------+---+---+----+
```

e. Extracting data in various formats

Getting Data In/Out

[30]
```
df.write.csv('foo.csv', header=True)
spark.read.csv('foo.csv', header=True).show()
```

```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|black|carrot|  6| 60|
| blue|banana|  2| 20|
|  red|carrot|  3| 30|
|  red|banana|  7| 70|
|  red|carrot|  5| 50|
|  red|banana|  1| 10|
| blue| grape|  4| 40|
|  red| grape|  8| 80|
+-----+------+---+---+
```

[31]
```
df.write.parquet('bar.parquet')
spark.read.parquet('bar.parquet').show()
```

```
24/02/26 12:08:26 WARN MemoryManager: Total allocation exceeds 95.00% (906,992,014 bytes) of heap memory
Scaling row group sizes to 96.54% for 7 writers
24/02/26 12:08:26 WARN MemoryManager: Total allocation exceeds 95.00% (906,992,014 bytes) of heap memory
Scaling row group sizes to 84.47% for 8 writers
24/02/26 12:08:26 WARN MemoryManager: Total allocation exceeds 95.00% (906,992,014 bytes) of heap memory
Scaling row group sizes to 75.08% for 9 writers
24/02/26 12:08:26 WARN MemoryManager: Total allocation exceeds 95.00% (906,992,014 bytes) of heap memory
Scaling row group sizes to 84.47% for 8 writers
24/02/26 12:08:26 WARN MemoryManager: Total allocation exceeds 95.00% (906,992,014 bytes) of heap memory
Scaling row group sizes to 96.54% for 7 writers
```

```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|black|carrot|  6| 60|
| blue|banana|  2| 20|
|  red|banana|  7| 70|
|  red|carrot|  5| 50|
| blue| grape|  4| 40|
|  red|carrot|  3| 30|
|  red|banana|  1| 10|
|  red| grape|  8| 80|
+-----+------+---+---+
```

```
df.write.orc('zoo.orc')
spark.read.orc('zoo.orc').show()
```

```
+-----+------+---+---+
|color| fruit| v1| v2|
+-----+------+---+---+
|  red|banana|  7| 70|
|  red| grape|  8| 80|
|black|carrot|  6| 60|
| blue|banana|  2| 20|
|  red|banana|  1| 10|
|  red|carrot|  5| 50|
| blue| grape|  4| 40|
|  red|carrot|  3| 30|
+-----+------+---+---+
```

3. Working on the dataframe using various SQL queries for processing data.

Working with SQL

```
[5]
df.createOrReplaceTempView("tableA")
spark.sql("SELECT count(*) from tableA").show()
```

```
+--------+
|count(1)|
+--------+
|       8|
+--------+
```

```
[6]
@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()
```

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspa
+-----------+
|add_one(v1)|
+-----------+
|          2|
|          3|
|          4|
|          5|
|          6|
|          7|
|          8|
|          9|
+-----------+
```

```
from pyspark.sql.functions import expr
df.selectExpr('add_one(v1)').show()
df.select(expr('count(*)') > 0).show()
```

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/python/lib/pyspark.zip/p
+-----------+
|add_one(v1)|
+-----------+
|          2|
|          3|
|          4|
|          5|
|          6|
|          7|
|          8|
|          9|
+-----------+

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

```
import pandas as pd
import numpy as np
import pyspark.pandas as ps
from pyspark.sql import SparkSession
```

4. Working with pandas and pandas API on Spark for pre-processing the data.

Object Creation

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

```
s
```

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

```
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
```

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

```
dates = pd.date_range('20130101', periods=6)
dates
```

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

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

	A	B	C	D
2013-01-01	-0.529039	1.223985	0.603494	0.113433
2013-01-02	-1.089198	-0.359664	0.283591	-0.935272
2013-01-03	0.256029	0.415919	1.420078	-0.544752
2013-01-04	1.213683	-0.201052	-1.052002	1.236294
2013-01-05	-0.682318	0.164947	-2.043778	0.542241
2013-01-06	-0.257527	-0.066415	1.483461	-0.060607

```
psdf = ps.from_pandas(pdf)
type(psdf)
psdf
```

	A	B	C	D
2013-01-01	-0.529039	1.223985	0.603494	0.113433
2013-01-02	-1.089198	-0.359664	0.283591	-0.935272
2013-01-03	0.256029	0.415919	1.420078	-0.544752
2013-01-04	1.213683	-0.201052	-1.052002	1.236294
2013-01-05	-0.682318	0.164947	-2.043778	0.542241
2013-01-06	-0.257527	-0.066415	1.483461	-0.060607

```
spark = SparkSession.builder.getOrCreate()
sdf = spark.createDataFrame(pdf)
sdf.show()
```

47]

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/conversion.py:485: FutureWarni
  if should_localize and is_datetime64tz_dtype(s.dtype) and s.dt.tz is not None:
+------------------+-------------------+------------------+--------------------+
|                 A|                  B|                 C|                   D|
+------------------+-------------------+------------------+--------------------+
|-0.5290389155420092|  1.2239847575208347| 0.6034935745432756| 0.11343333958732911|
|-1.0891982759774186|-0.35966439557589214| 0.2835907640407051| -0.9352723011054798|
| 0.2560293452386302|  0.4159194345442462| 1.4200777502808168| -0.5447524199772743|
| 1.2136832617497266|-0.20105151981126348|-1.0520022154664581|   1.236294462529281|
| -0.682317547741866| 0.16494708044694031| -2.043778054983065|   0.542241060580985|
|-0.2575274352134124| -0.0664149075975735| 1.4834605268575678|-0.06060680383326...|
+------------------+-------------------+------------------+--------------------+
```

```
psdf = sdf.pandas_api()
psdf
```

	A	B	C	D
0	-0.529039	1.223985	0.603494	0.113433
1	-1.089198	-0.359664	0.283591	-0.935272
2	0.256029	0.415919	1.420078	-0.544752
3	1.213683	-0.201052	-1.052002	1.236294
4	-0.682318	0.164947	-2.043778	0.542241
5	-0.257527	-0.066415	1.483461	-0.060607

[49]
```
psdf.dtypes
```

```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

[50]
```
psdf.head()
```

	A	B	C	D
0	-0.529039	1.223985	0.603494	0.113433
1	-1.089198	-0.359664	0.283591	-0.935272
2	0.256029	0.415919	1.420078	-0.544752
3	1.213683	-0.201052	-1.052002	1.236294
4	-0.682318	0.164947	-2.043778	0.542241

```
psdf.index
```

```
Index([0, 1, 2, 3, 4, 5], dtype='int64')
```

```
[52]
psdf.columns
```

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

```
[ ]
psdf.to_numpy()
```

```
[54]
psdf.describe()
```

```
24/02/26 12:16:28 WARN SparkStringUtils: Truncated the string representation of a
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.181395	0.196287	0.115807	0.058556
std	0.817223	0.572949	1.404648	0.773292
min	-1.089198	-0.359664	-2.043778	-0.935272
25%	-0.682318	-0.201052	-1.052002	-0.544752
50%	-0.529039	-0.066415	0.283591	-0.060607
75%	0.256029	0.415919	1.420078	0.542241
max	1.213683	1.223985	1.483461	1.236294

```
psdf.T
```

	0	1	2	3	4	5
A	-0.529039	-1.089198	0.256029	1.213683	-0.682318	-0.257527
B	1.223985	-0.359664	0.415919	-0.201052	0.164947	-0.066415
C	0.603494	0.283591	1.420078	-1.052002	-2.043778	1.483461
D	0.113433	-0.935272	-0.544752	1.236294	0.542241	-0.060607

[56]
```
psdf.sort_index(ascending=False)
```

	A	B	C	D
5	-0.257527	-0.066415	1.483461	-0.060607
4	-0.682318	0.164947	-2.043778	0.542241
3	1.213683	-0.201052	-1.052002	1.236294
2	0.256029	0.415919	1.420078	-0.544752
1	-1.089198	-0.359664	0.283591	-0.935272
0	-0.529039	1.223985	0.603494	0.113433

[57]
```
psdf.sort_values(by='B')
```

	A	B	C	D
1	-1.089198	-0.359664	0.283591	-0.935272
3	1.213683	-0.201052	-1.052002	1.236294
5	-0.257527	-0.066415	1.483461	-0.060607
4	-0.682318	0.164947	-2.043778	0.542241
2	0.256029	0.415919	1.420078	-0.544752
0	-0.529039	1.223985	0.603494	0.113433

Missing Data

```
[8]
pdf1 = pdf.reindex(index=dates[0:4], columns=list(pdf.columns) + ['E'])
pdf1.loc[dates[0]:dates[1], 'E'] = 1
psdf1 = ps.from_pandas(pdf1)
psdf1
```

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:379: FutureWarning: is_datetime64tz_dtype i
  if is_datetime64tz_dtype(s.dtype):
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:563: FutureWarning: is_datetime64tz_dtype i
  if not is_datetime64tz_dtype(pser.dtype):
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:379: FutureWarning: is_datetime64tz_dtype i
  if is_datetime64tz_dtype(s.dtype):
```

	A	B	C	D	E
2013-01-01	-0.529039	1.223985	0.603494	0.113433	1.0
2013-01-02	-1.089198	-0.359664	0.283591	-0.935272	1.0
2013-01-03	0.256029	0.415919	1.420078	-0.544752	NaN
2013-01-04	1.213683	-0.201052	-1.052002	1.236294	NaN

```
[9]
psdf1.dropna(how='any')
```

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:563: FutureWarning: is_datetime64tz_dtype i
  if not is_datetime64tz_dtype(pser.dtype):
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:379: FutureWarning: is_datetime64tz_dtype i
  if is_datetime64tz_dtype(s.dtype):
```

	A	B	C	D	E
2013-01-01	-0.529039	1.223985	0.603494	0.113433	1.0
2013-01-02	-1.089198	-0.359664	0.283591	-0.935272	1.0

```
[60]
psdf1.fillna(value=5)
```

```
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:563: FutureWarning: is_dat
  if not is_datetime64tz_dtype(pser.dtype):
/home/cs-ds/.local/lib/python3.10/site-packages/pyspark/sql/pandas/types.py:379: FutureWarning: is_dat
  if is_datetime64tz_dtype(s.dtype):
```

	A	B	C	D	E
2013-01-01	-0.529039	1.223985	0.603494	0.113433	1.0
2013-01-02	-1.089198	-0.359664	0.283591	-0.935272	1.0
2013-01-03	0.256029	0.415919	1.420078	-0.544752	5.0
2013-01-04	1.213683	-0.201052	-1.052002	1.236294	5.0

Opeartions

```
[61]
psdf.mean()
```

```
A   -0.181395
B    0.196287
C    0.115807
D    0.058556
dtype: float64
```

```
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.
```

```
[63]
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", True)
%timeit ps.range(300000).to_pandas()
```

```
76.7 ms ± 7.86 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
[64]
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", False)
%timeit ps.range(300000).to_pandas()
```

```
618 ms ± 69 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
[70]
ps.reset_option("compute.default_index_type")
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", prev)  # Set its default value back.
```

Grouping

```
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)})

psdf
```

	A	B	C	D
0	foo	one	2.283439	-1.917823
1	bar	one	0.381945	0.380673
2	foo	two	-0.564655	-0.905398
3	bar	three	0.685973	-0.674223
4	foo	two	1.297532	1.296038
5	bar	two	-2.374164	0.931015
6	foo	one	1.423661	1.482182
7	foo	three	-0.253599	-0.284564

```
psdf.groupby('A').sum()
```

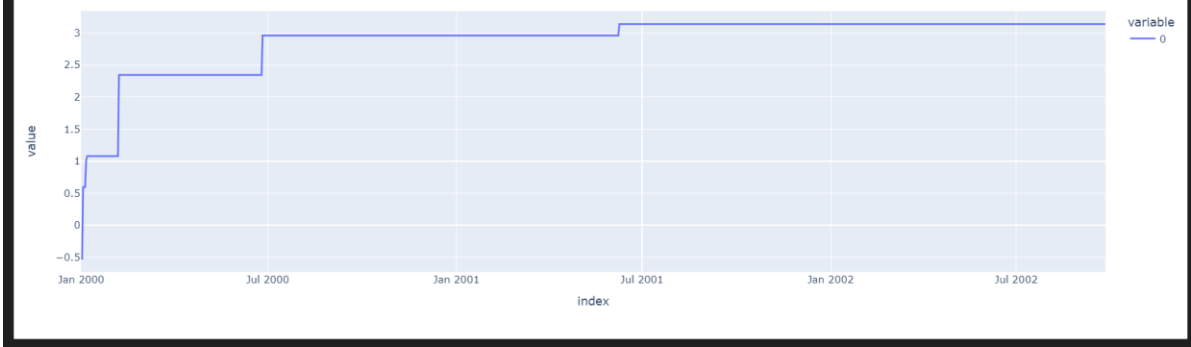
A	C	D
foo	4.186379	-0.329564
bar	-1.306246	0.637465

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

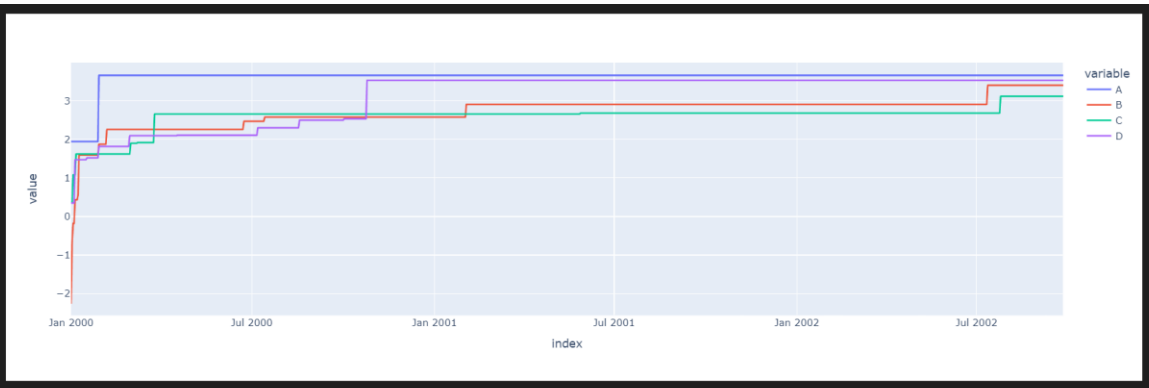
A	B	C	D
foo	one	3.707101	-0.435641
bar	one	0.381945	0.380673
foo	two	0.732877	0.390641
bar	three	0.685973	-0.674223
	two	-2.374164	0.931015
foo	three	-0.253599	-0.284564

Plotting

```
pser = pd.Series(np.random.randn(1000),
                 index=pd.date_range('1/1/2000', periods=1000))
psser = ps.Series(pser)
psser = psser.cummax()
psser.plot()
```




```
pdf = pd.DataFrame(np.random.randn(1000, 4), index=pser.index,
                   columns=['A', 'B', 'C', 'D'])
psdf = ps.from_pandas(pdf)
psdf = psdf.cummax()
psdf.plot()
```



Getting data in/out

```
psdf.to_csv('foo.csv')
ps.read_csv('foo.csv').head(10)
```

[78]

Python

	A	B	C	D
0	1.947564	-2.234966	0.359221	0.354794
1	1.947564	-0.679155	0.359221	0.354794
2	1.947564	-0.173558	1.084768	0.354794
3	1.947564	-0.173558	1.084768	0.354794
4	1.947564	0.436979	1.084768	1.477296
5	1.947564	0.436979	1.623141	1.477296
6	1.947564	0.436979	1.623141	1.477296
7	1.947564	0.556595	1.623141	1.477296
8	1.947564	1.597999	1.623141	1.477296
9	1.947564	1.597999	1.623141	1.477296

```
[79]
psdf.to_parquet('bar.parquet')
ps.read_parquet('bar.parquet').head(10)
```

```
24/02/26 12:23:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:35 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
```

	A	B	C	D
0	1.947564	-2.234966	0.359221	0.354794
1	1.947564	-0.679155	0.359221	0.354794
2	1.947564	-0.173558	1.084768	0.354794
3	1.947564	-0.173558	1.084768	0.354794
4	1.947564	0.436979	1.084768	1.477296
5	1.947564	0.436979	1.623141	1.477296
6	1.947564	0.436979	1.623141	1.477296
7	1.947564	0.556595	1.623141	1.477296

```
[80]
psdf.to_spark_io('zoo.orc', format="orc")
ps.read_spark_io('zoo.orc', format="orc").head(10)
```

```
24/02/26 12:23:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
24/02/26 12:23:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
```

	A	B	C	D
0	1.947564	-2.234966	0.359221	0.354794
1	1.947564	-0.679155	0.359221	0.354794
2	1.947564	-0.173558	1.084768	0.354794
3	1.947564	-0.173558	1.084768	0.354794
4	1.947564	0.436979	1.084768	1.477296
5	1.947564	0.436979	1.623141	1.477296
6	1.947564	0.436979	1.623141	1.477296
7	1.947564	0.556595	1.623141	1.477296
8	1.947564	1.597999	1.623141	1.477296
9	1.947564	1.597999	1.623141	1.477296

Conclusion: Performed various operations using pyspark and pyspark sql by creating a dataframe, performing functions like grouping, function, plotting, etc.