Spark and Pyspark:

Back in the day, there was limited data generation. Hence, storing and processing data was done with a single storage unit and a processor, respectively. In the blink of an eye, data generation increases by leaps and bounds. Not only did it increase in volume but also its variety. Therefore, a single processor was incapable of processing high volumes of different varieties of data. Speaking of varieties of data, you can have structured, semi-structured and unstructured data.

To address this issue, the storage unit is distributed amongst each of the processors. The distribution resulted in storing and accessing data efficiently and with no network overheads. As seen below, this method is called parallel processing with distributed storage.

Big data:

Big Data refers to the massive amount of data that cannot be stored, processed, and analyzed using traditional ways.

The main elements of Big Data are:

- Volume - There is a massive amount of data generated every second.
- Velocity - The speed at which data is generated, collected, and analyzed
- Variety - The different types of data: structured, semi-structured, unstructured
- Value - The ability to turn data into useful insights for your business
- Veracity - Trustworthiness in terms of quality and accuracy

The main challenges that Big Data faced and the solutions for each are listed below:

Challenges	Solution
Single central storage	Distributed storage
Serial processing - One input - One processor - One output	Parallel processing - Multiple inputs - Multiple processors - One output
Lack of ability to process unstructured data	Ability to process every type of data

Hadoop:

Hadoop is a widely used Big Data technology for storing, processing, and analyzing large datasets.

Hadoop is a framework that uses distributed storage and parallel processing to store and manage Big Data. It is the most commonly used software to handle Big Data.

There are three components of Hadoop:

1. **Hadoop HDFS** - Hadoop Distributed File System (HDFS) is the storage unit of Hadoop.
2. **Hadoop MapReduce** - Hadoop MapReduce is the processing unit of Hadoop.
3. **Hadoop YARN** - Hadoop YARN is a resource management unit of Hadoop.

Spark:

Spark is an open-source cluster computing framework that mainly focuses on fast computation, i.e., improving the application's speed. Apache Spark minimizes the read/write operations to disk-based systems and speeds up in-memory processing.

The Apache Foundation introduced it as an extension to Hadoop to speed up its computational processes. Spark supports exclusive cluster management and uses Hadoop for storage.

Spark is a big data solution that has been proven to be easier and faster than Hadoop MapReduce. Spark is an open source software developed by UC Berkeley RAD lab in 2009. Since it was released to the public in 2010, Spark has grown in popularity and is used through the industry with an unprecedented scale.

In the era of Big Data, practitioners need more than ever fast and reliable tools to process streaming of data. Earlier tools like MapReduce were favorite but were slow. To overcome this issue, Spark offers a solution that is both fast and general-purpose. The main difference between Spark and MapReduce is that Spark runs computations in memory during the later on the hard disk. It allows high-speed access and data processing, reducing times from hours to minutes.

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

Apache Spark is the technology, and you can interact with it in different ways, for example with Java, Scala, or Python.

The interface that allows you to interact with Spark by using Python is called PySpark.

PySpark is the Python API written in python to support Apache Spark. Apache Spark is a distributed framework that can handle Big Data analysis. Apache Spark is written in Scala and can be integrated with Python, Scala, Java, R, SQL languages. Spark is basically a computational engine, that works with huge sets of data by processing them in parallel and batch systems.

Note: Spark is lightning-fast and is more favorable than the Hadoop framework. It runs 100 times faster in-memory and ten times faster on disk. Moreover, it sorts 100 TB of data 3 times faster than Hadoop using 10X fewer machines.

Note: However, Spark makes use of huge amounts of RAM to run everything in memory. And RAM has a higher price associated with it than hard-disks.

On the other hand, Hadoop is disk-based. Thus, your cost of buying an expensive RAM gets saved. However, Hadoop needs more systems to distribute the disk I/O over multiple systems.

Hadoop costs less, and Spark is comparatively expensive due to its in-memory solution.

Apache Spark supports three most powerful programming languages:

1. Scala
2. Java
3. Python
4. R

Pyspark:

Introduction:

PySpark is a Spark library written in Python to run Python application using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes).

In other words, PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine learning applications.

Spark basically written in Scala and later on due to its industry adaptation it's API PySpark released for Python using Py4J. Py4J is a Java library that is integrated within PySpark and allows python to dynamically interface with JVM objects, hence to run PySpark you also need Java to be installed along with Python, and Apache Spark.

Additionally, For the development, you can use Anaconda distribution (widely used in the Machine Learning community) which comes with a lot of useful tools like Spyder IDE, Jupyter notebook to run PySpark applications.

In real-time, PySpark has used a lot in the machine learning & Data scientists community; thanks to vast python machine learning libraries. Spark runs operations on billions and trillions of data on distributed clusters 100 times faster than the traditional python applications.

PySpark is very well used in Data Science and Machine Learning community as there are many widely used data science libraries written in Python including NumPy, TensorFlow. Also used due to its efficient processing of large datasets. PySpark has been used by many organizations like Walmart, Trivago, Sanofi, Runtastic, and many more.

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.

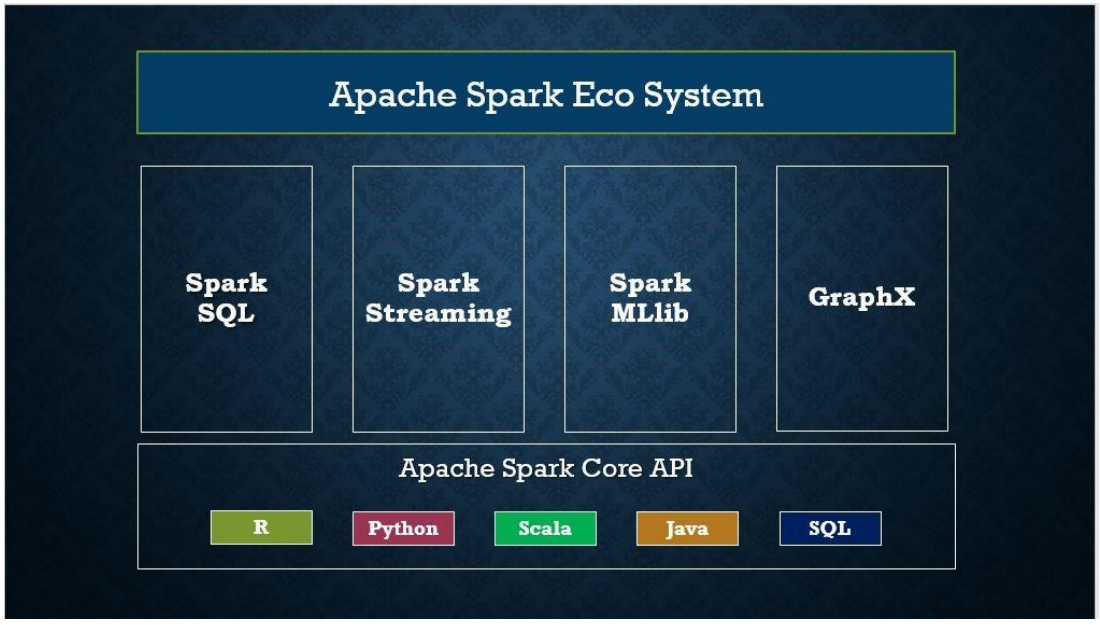
features:

- In-memory computation
- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
- Fault-tolerant
- Immutable
- Lazy evaluation
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

Advantages of PySpark:

- PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on PySpark are 100x faster than traditional systems.
- You will get great benefits using PySpark for data ingestion pipelines.
- Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems.
- PySpark also is used to process real-time data using Streaming and Kafka.
- Using PySpark streaming you can also stream files from the file system and also stream from the socket.
- PySpark natively has machine learning and graph libraries

PySpark Components :



Spark consists of one of the bigger components called Spark core which contains the majority of the libraries. On top of this Spark core, there are four different components. They are

1. Spark SQL
2. Spark Streaming
3. MLLIB
4. GraphX
5. Spark SQL
6. Spark core

It provides a SQL like interface to do the data processing with Spark as a processing engine. It can process both structured and semi-structured data. Using Spark SQL, you can query in SQL and HQL too. It can process data from multiple sources.

Spark Streaming

It provides support for processing a large stream of data. The real-time streaming is supported by using micro-batching. The incoming live data is converted into small batches and processed by the processing engine.

MLLIB

This component provides libraries for machine learning towards the statistic and dynamic analysis of the data. It also includes clustering, classification, regression and many other machine learning algorithms. It delivers high-quality algorithms with high speed.

GraphX

It provides a distributed graph computation on top of the Spark core. It consists of several Spark RDD API which helps in creating directed graphs whose vertices and edges are linked with arbitrary properties. Using GraphX, traversal, searching and pathfinding can be done.

Spark Core

All the functionalities being provided by Apache Spark are built on the highest of the Spark Core. It delivers speed by providing in-memory computation capability. Spark Core is the foundation of parallel and distributed processing of giant dataset. It is the main backbone of the essential I/O functionalities and significant in programming and observing the role of the spark cluster. It holds all the components related to scheduling, distributing and monitoring jobs on a cluster, Task dispatching, Fault recovery.

The functionalities of this component are:
It contains the basic functionality of spark. (Task scheduling, memory management, fault recovery, interacting with storage systems).

Cluster Managers:

Spark also consists of three pluggable cluster managers.

Standalone

It is a simple cluster manager which is easier to set up and execute the tasks on the cluster. It is a reliable cluster manager which can handle failures successfully. It can manage the resources based on the application requirements.

Apache Mesos

It is a general cluster manager from the Apache group that can also run Hadoop MapReduce along with Spark and other service applications. It consists of API for most of the programming languages.

Hadoop YARN

It is a resource manager which was provided in Hadoop 2. It stands for Yet Another Resource Negotiator. It is also a general-purpose cluster manager and can work in both Hadoop and Spark.

These are the three cluster managers supported by the Spark ecosystem.

Now, let me give a glimpse of the most essential part of Apache Spark. i.e., Spark RDD.

Spark RDD:

Apache Spark has a fundamental data structure called Resilient Distributed Dataset (RDD) which forms the backbone of Apache Spark. It performs two main functions called transformations and actions. Transformations are the operations that we perform on the input data. Examples are Map(), Filter(), sortBy() which we can perform on the required data. Transformations create new RDD based on the operations we perform. Actions are the processes that we perform on the newly created RDD to get the desired results. Examples of actions are collect(), countByKey(). Action returns the result to the driver.

Installation and setup pyspark:

Using PyPI

PySpark installation using PyPI is as follows:

pip install pyspark

Using Conda:

create a new conda environment from your terminal and activate it, proceed as shown below:

conda create -n pyspark_env
conda activate pyspark_env

After activating the environment, use the following command to install pyspark

conda install -c conda-forge pyspark

Using .exe on windows:

1: Go to the Apache Spark website (link)

a)Choose a Spark release and package type

b) Choose a download type: (Direct Download)

c) Download Spark. Keep in mind if you download a newer version, you will need to modify the remaining commands for the file you downloaded.

2. Move the file to where you want to unzip it.

mkdir C:\opt\spark

3. Move and unzip “spark-3.3.0-bin-hadoop3.tgz” file to above location or path

4. Download winutils.exe into your **spark-3.3.0-bin-hadoop3\bin**

curl -k -L -o winutils.exe https://github.com/steveloughran/winutils/blob/master/hadoop-2.6.0/bin/winutils.exe?raw=true

5. Make sure you have Java 7+ installed on your machine.

6. Next, we will edit our environmental variables so we can open a spark notebook in any directory.

a> setx SPARK_HOME C:\opt\spark\spark-3.3.0-bin-hadoop3

b> setx HADOOP_HOME C:\opt\spark\spark-3.3.0-bin-hadoop3

C> setx PYSPARK_DRIVER_PYTHON ipython

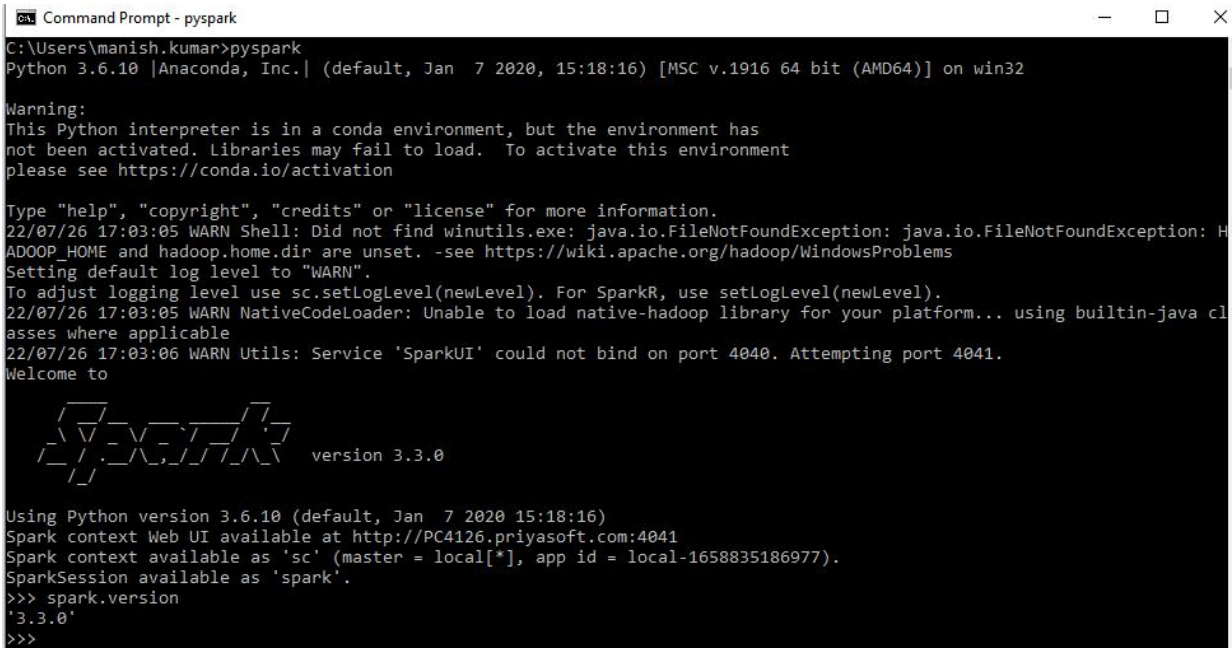
D> setx PYSPARK_DRIVER_PYTHON_OPTS notebook

E> SET PATH=C:\opt\spark\spark-3.3.0-bin-hadoop3\bin

Explore pyspark:

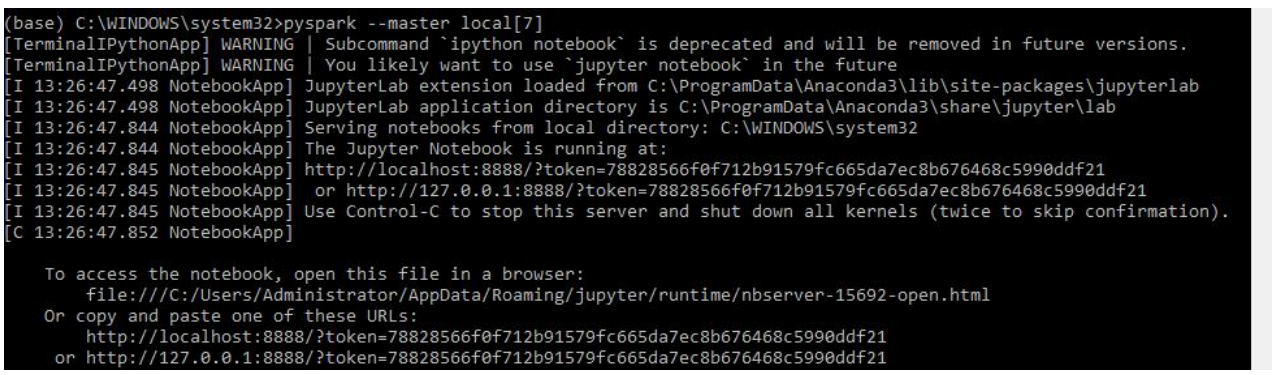
Pyspark shell:

Now open the command prompt and type pyspark command to run the Pyspark shell



Pyspark Jupyter notebook:

Step 1: Activate conda env
Step 2: run below command to run and open pyspark jupyter notebook
Cmd : pyspark --master local[“number od cores”]



Notes: The PYSPARK_DRIVER_PYTHON parameter and the PYSPARK_DRIVER_PYTHON_OPTS parameter are used to launch the PySpark shell in Jupyter Notebook. The — master parameter is used for setting the master node address. Here we launch Spark locally on 2 cores for local testing.

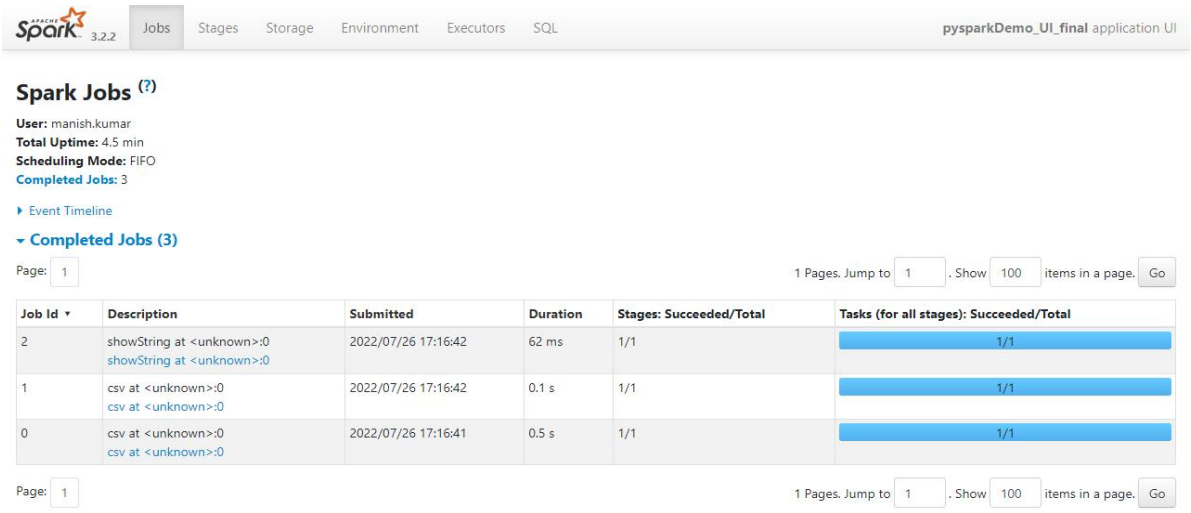
Spark Web UI:

Apache Spark provides a suite of Web UIs (Jobs, Stages, Tasks, Storage, Environment, Executors, and SQL) to monitor the status of your Spark application, resource consumption of Spark cluster, and Spark configurations. On Spark Web UI, you can see how the operations are executed.

Spark-shell also creates a Spark context web UI and by default, it can access from

http://localhost:4040

http://localhost:4041



Implementation:

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[*]").appName('PySpark_Tutorial')\
      .getOrCreate()

# where the '*' represents all the cores of the CPU.
```

Creating SparkSession:

To create a SparkSession, you need to use the builder pattern method builder()

- **getOrCreate()** — the method returns an already existing SparkSession; if not exists, it creates a new SparkSession.
- **master()** – If you are running it on the cluster you need to use your master name as an argument. usually, it would be either yarn or mesos depends on your cluster setup and also uses local[X] when running in Standalone mode. X should be an integer value and should be greater than 0 which represents how many partitions it should create when using RDD, DataFrame, and Dataset. Ideally, the value X should be the number of CPU cores.
- **appName()** the method is used to set the name of your application.
- **getOrCreate()** the method returns an existing SparkSession if it exists otherwise it creates a new SparkSession.

```
## Read The dataset
training = spark.read.csv('test1.csv',header=True,inferSchema=True)
```

In [73]:

```
training.show()
```

```
+---------+---+----------+------+
|     Name|age|Experience|Salary|
+---------+---+----------+------+
|    Krish| 31|        10| 30000|
|Sudhanshu| 30|         8| 25000|
|    Sunny| 29|         4| 20000|
```

```
|      Paul| 24|          3| 20000|
|    Harsha| 21|          1| 15000|
|  Shubham| 23|          2| 18000|
+---------+---+----------+------+
```

In [74]:

```
training.printSchema()
```

```
root
 |-- Name: string (nullable = true)
 |-- age: integer (nullable = true)
 |-- Experience: integer (nullable = true)
 |-- Salary: integer (nullable = true)
```

In [75]:

```
training.columns
```

Out[75]:

```
['Name', 'age', 'Experience', 'Salary']
```

In [ ]:

```
[Age,Experience]----> new feature--->independent feature
```

In [76]:

```
from pyspark.ml.feature import
VectorAssemblerfeatureassembler=VectorAssembler(inputCols=["age","E
xperience"],outputCol="Independent Features")
```

In [77]:

```
output=featureassembler.transform(training)
```

In [78]:

```
output.show()
```

```
+---------+---+----------+------+-------------------+
|     Name|age|Experience|Salary|Independent Features|
+---------+---+----------+------+-------------------+
|    Krish| 31|        10| 30000|          [31.0,10.0]|
|Sudhanshu| 30|         8| 25000|           [30.0,8.0]|
|    Sunny| 29|         4| 20000|           [29.0,4.0]|
|     Paul| 24|         3| 20000|           [24.0,3.0]|
|   Harsha| 21|         1| 15000|           [21.0,1.0]|
|  Shubham| 23|         2| 18000|           [23.0,2.0]|
+---------+---+----------+------+-------------------+
```

In [79]:

```
output.columns
```

Out[79]:

```
['Name', 'age', 'Experience', 'Salary', 'Independent Features']
```

In [80]:

```
finalized_data=output.select("Independent Features","Salary")
```

In [81]:

```
finalized_data.show()
```

```
+-------------------+------+
|Independent Features|Salary|
+-------------------+------+
|          [31.0,10.0]| 30000|
|           [30.0,8.0]| 25000|
|           [29.0,4.0]| 20000|
|           [24.0,3.0]| 20000|
|           [21.0,1.0]| 15000|
|           [23.0,2.0]| 18000|
+-------------------+------+
```

In [82]:

```
from pyspark.ml.regression import LinearRegression
##train test split
train_data,test_data=finalized_data.randomSplit([0.75,0.25])
regressor=LinearRegression(featuresCol='Independent Features',
labelCol='Salary')
regressor=regressor.fit(train_data)
```

In [83]:

```
### Coefficients
regressor.coefficients
```

Out[83]:

```
DenseVector([-5000.0, 7000.0])
```

In [84]:

```
### Intercepts
regressor.intercept
```

Out[84]:

```
118999.99999893687
```

In [85]:

```
### Prediction
pred_results=regressor.evaluate(test_data)
```

In [86]:

```
pred_results.predictions.show()
```

```
+-------------------+------+-----------------+
|Independent Features|Salary|       prediction|
+-------------------+------+-----------------+
|          [21.0,1.0]| 15000|20999.99999996154|
|          [29.0,4.0]| 20000|2000.000000192551|
|         [31.0,10.0]| 30000|33999.99999993094|
+-------------------+------+-----------------+
```

In [87]:

```
pred_results.meanAbsoluteError,pred_results.meanSquaredError
```

Out[87]:

```
(9333.333333233308, 125333333.3306847)
```

References:

1 https://www.netsolutions.com/insights/hadoop-vs-spark/#:~:text=Spark%20is%20lightning%2Dfast%20and,it%20processes%20everything%20in%20memory.

2> https://www.guru99.com/pyspark-tutorial.html

3> https://sparkbyexamples.com/pyspark-tutorial/

4 Download Apache spark:

https://spark.apache.org/downloads.html

4> Env setup on windows to install spark:
https://medium.com/@GalarnykMichael/install-spark-on-windows-pyspark-4498a5d8d66c

5> spark documentation:

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