**Spark and Big Data Analytics: Spark, Introduction to Data Analysis with Spark. Text, Web Content and Link Analytics: Introduction, Text Mining, Web Mining, Web Content and Web Usage Analytics, Page Rank, Structure of Web and Analyzing a Web Graph.**

## 5.1 SPARK

Apache Spark is a fast and general compute engine. Apache Spark powers the analytics applications up to 100 times faster. It supports HDFS compatible data. Spark has a simple and expressive programming model. Expressive program model implements a number of mathematical and logic operations with smaller and easier written codes which a compiler as well as programmer can understand easily. The model, therefore, gives programming ease for a wide range of applications. Applications of expressive codes are in analytics, Extract Transform Load (ETL), Machine Learning (ML), stream processing and graph computations.

Spark runs on both Windows and UNIX-like systems, such as Linux and Mac OS. Java is essential for running Spark applications. Executing a spark application on a computer system therefore requires setting a JDK path using JAVA_HOME environment variable or system variable.

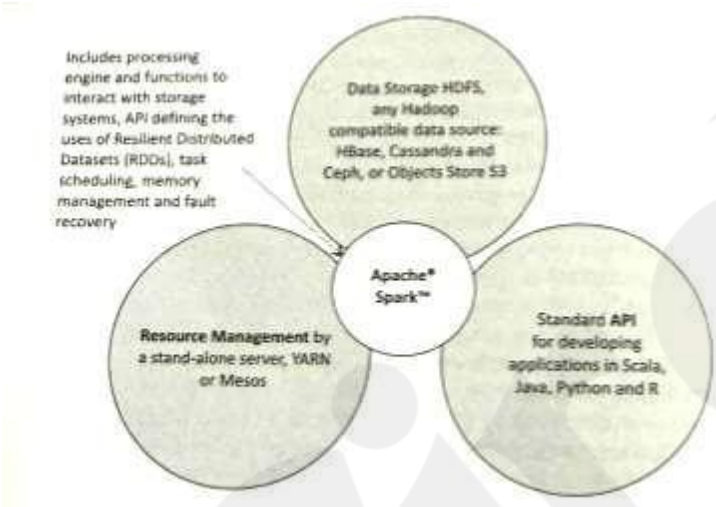
### 5.1.1 Introduction to Big Data Tool-Spark

Figure below shows the main components in the Apache software Foundation's Spark framework, which includes data storage, API's and resources management bonded with functions in Spark core.

Main components of Spark Architecture are:

1. Spark HDFS file system for data storage: Storae is at an HDFS or Hadoop compatible data source such as HDFS, HBase, Cassandra, Ceph or at the ojects tore S3.

2. Spark standard API enables the creation of applications using Scala, Java, Python and R.
3. Spark resource management can be a stand-alone server or it can be on a distributed computing framework, such as YARN or Mesos.



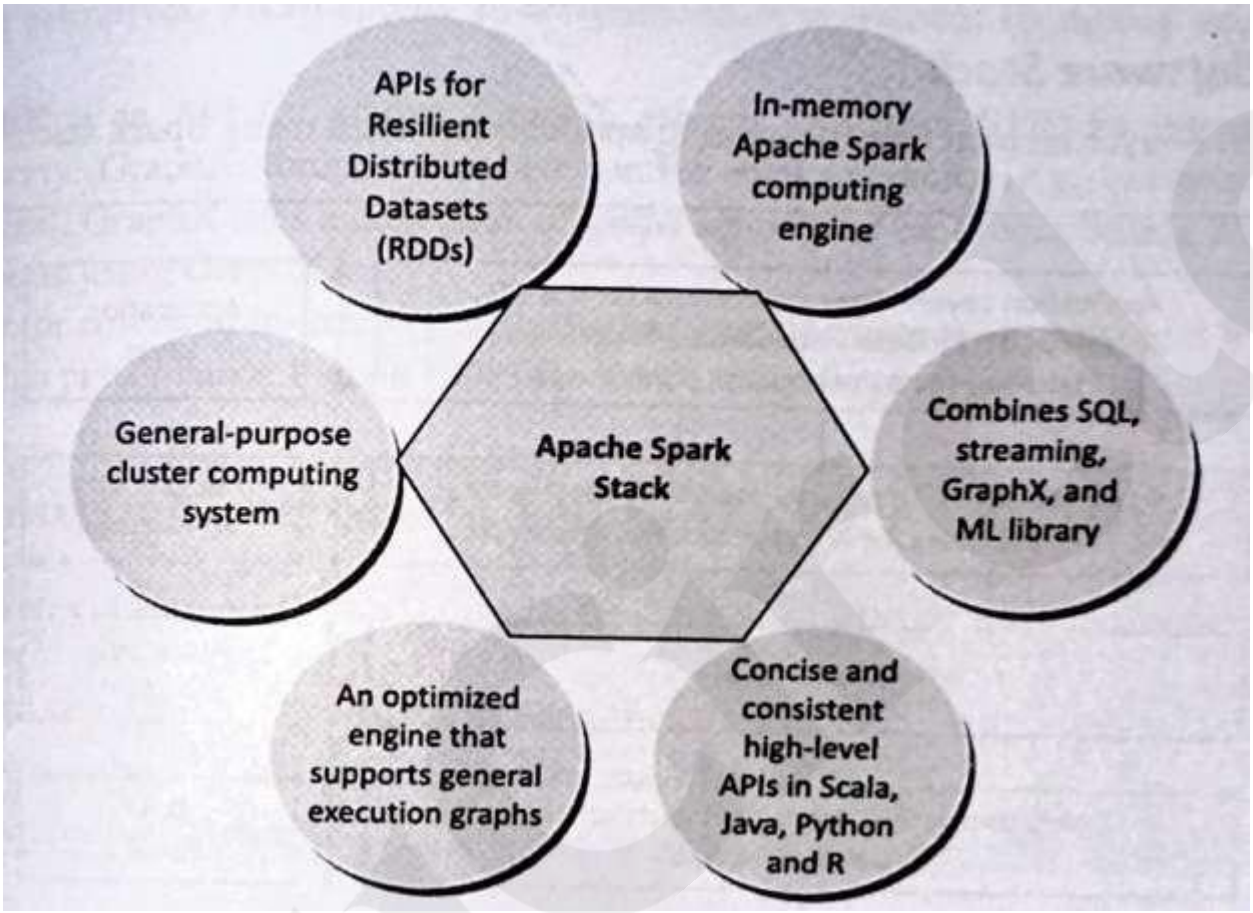

**Ceph** refers to an open source, scalable object, block, file storing unified system. Ceph provides for dynamic replication and redistribution. Ceph provides HDFS compatible mechanisms for Big data in petabytes or exabytes. An application uniquely accesses the object, block and file stores in a system using Ceph. Ceph is highly reliable.

**Apache Mesos** is an open source project developed at University of Berkeley, and now passed to Apache. It manages computing clusters. Its implementation is in C++. Apache released a stable version 1.3.0 of Mesos in 2017. Apache Mesos enables fine-grained sharing of CPU, RAM, IOs and other frameworks. Mesos offers them resource. Each resource contains a list of agents. Each agent has the Hadoop and Mapreduce executers.

**S3(Simple Storage Service)** refers to an Amazon we service on the cloud names S3 which provides Object stores for data.

### 5.1.1.1 Features of Spark

Figure below shows the main features of the Spark.



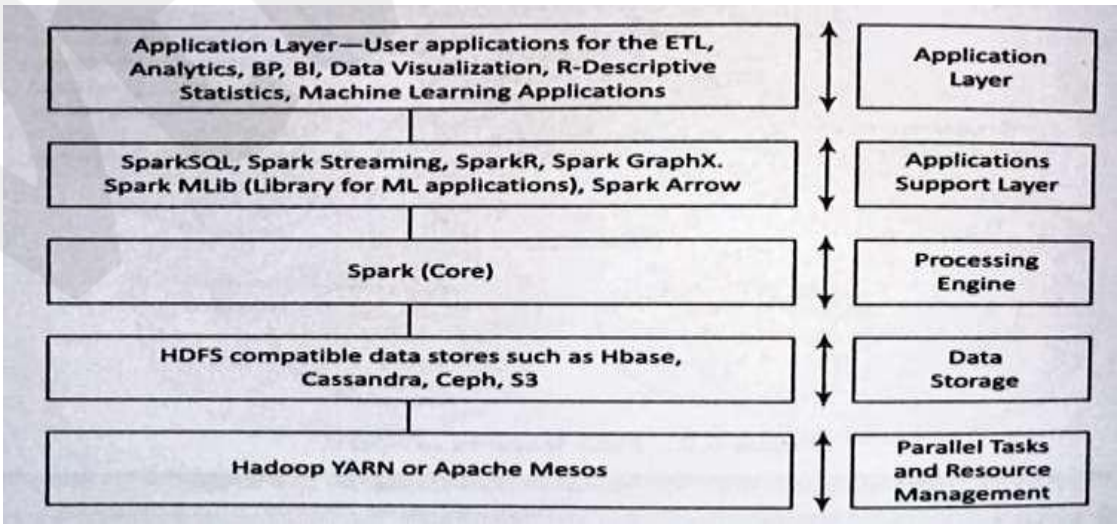

The features of Spark:

1. Spark provisions for creating applications that use the complex data. In-memory Apahe Spark computing engine enables up to 100 times performance with respect to Hadoop.
2. Execution engine uses both in-memory and on-disk computing. Intermediate results save in-memory and spill over to disk.
3. Data uploading from an Object Store for immediate use as a Spark object instance. Spark service interface sets up the Object Store.
4. Provides high performance when an application accesses memory cache from the disk.

5. Contains API to define Resilient Distributed Datasets (RDDs). RDD is a programming abstraction. RDD is the core concept in Spark framework. RDD represents a collection of Object Stores distributed across many compute nodes for parallel processing. Spark stores data in RDD on different partitions. A table has partitions into columns or rows. Similarly, an RDD can also be considered as a table in a database that can hold any type of data. RDDs are also fault tolerant.
6. Processes any kind of data, namely structured, semi-structured or unstructured data arriving from various sources.
7. Supports many new functions in addition to Map and Reduce functions.
8. Optimizes data processing performance by slowing the evaluation of Big Data queries.
9. Provides concise and consistent APIs in Scala, Java and Python. Spark codes are in Scala and run on JVM environment.
10. Provides powerful tool to analyze data interactively using shell which is available in either Scala (which runs on the Java VM and is thus a good way to use existing Java libraries) or Python. The tool also provides for learning the usages of API.

### 5.1.1.2 Spark Software Stack

Figure below shows a five-layer architecture for running applications when using Spark stack.



The main components of Spark stack are SQL, Streaming, R, GraphX, MLib and Arrow at the applications support layer. Spark core is the processing engine. Data Store provides the data to the processing engine. Hadoop, YARN or Mesos facilitates the parallel running of the tasks and the management and scheduling ofthe resources.

**Spark Stack**

Spark stack imbibes generality to Spark. Grouping of the following forms Spark stack:

**Spark SQL** for the structured data. The SQL runs the queries on Spark data in the traditional business HQL queries also run in Spark SQL. Runs UDFS for inline SQL, distributed DataFrames, Parquet, Hive and analytics and visualization applications. Spark SQL enables Spark datasets to use JDBC or ODBC APL Cassandra Data Stores.

**Spark Streaming** is for processing real-time streaming data. Processing is based on micro-batches style of computing and processing. Streaming uses the DStream which is basically a series of RDDs, to process the real-time data.

**SparkR** is an R package used as light-weight front end for Apache Spark from R. Spark API uses SparkR through the RDD class. A user can interactively run the jobs from the R shell on a cluster. An RDD API is in the distributed lists in R.

**Spark MLib** is Spark's scalable machine learning library. It consists of common learning algorithms and utilities. MLib includes classification, regression, clustering, collaborative filtering, dimensionality reduction and optimization primitives. MLib applies in recommendation systems, clustering and classification using Spark.

**Spark GraphX** is an API for graphs. GraphX extends the Spark RDD by introducing the Resilient Distributed Property. GraphX computations use fundamental operators (e.g., subgraph, join Vertices and aggregateMessages). GraphX uses a collection of graph algorithms for programming. Graph analytics tasks are created with ease using GraphX.

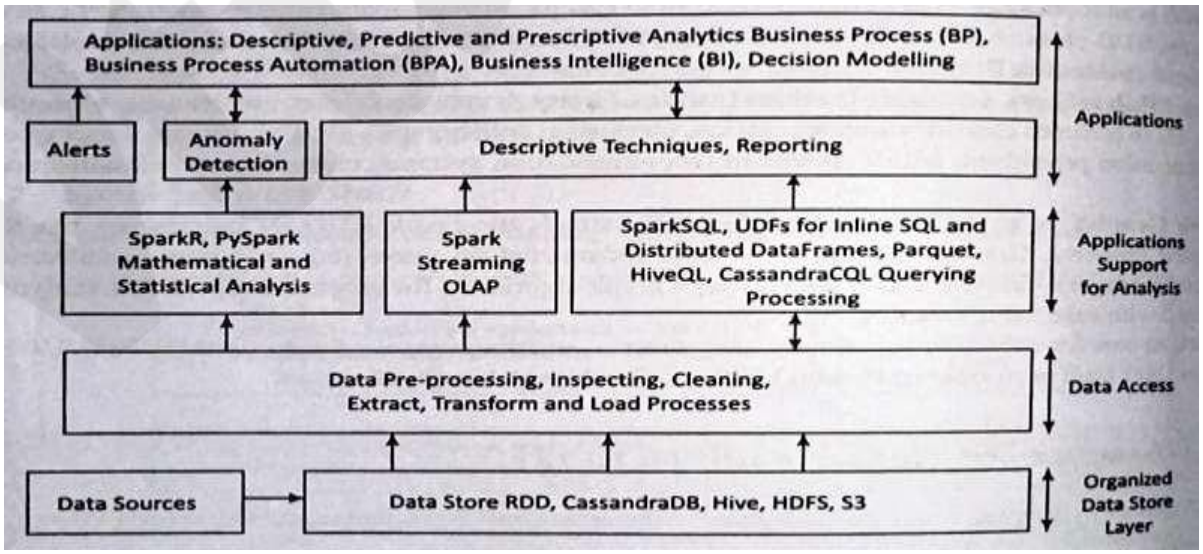
**Spark Arrow** for columnar in-memory analytics and enabling usages of vectorised UDFS (VUDFS). The Arrow enables high performance Python UDFs for SerDe and data pipelines.

## 5.2 INTRODUCTION TO DATA ANALYSIS WITH SPARK

**"Analysis of data is a process of inspecting, cleaning, transforming and modeling data with the goal of discovering useful information, suggesting conclusions and supporting decision-making."**

For examples, consider a car company. Assume that company sells five models of cars. Each model is available in 16 different colours and shades. Sales are processed through a large number of showrooms, all over the country. An analyses of annual sales and annual profits model-wise, colour-wise, region-wise and showroom-wise help in discovering useful information and making suggestive conclusions. The company does predictive analytics from the results of the analyses and decides the future strategies for manufacturing, sales and out-reaches to customers on the basis of the results of the analytics.

Figure below shows the steps between acquisition of data from different sources, applications of the analyzed data, and application support by Spark for the analyses.



Following are the steps for analyzing the data:

1. Data Storage: Store of data from the multiple sources after acquisition. The Big Data storage may be in HDFS compatible files, Cassandra, Hive, HDFS or S3.

2. Data pre-processing: This step requires:

(a) dropping out of range, inconsistent and outlier values,

(b) filtering unreliable, irrelevant and redundant information,

(c) data cleaning, editing, reduction and/or wrangling,

(d) data-validation, transformation or transcoding.

3. Extract, transform and Load (ETL)) for the analysis

4. Mathematical and statistical analysis of the data obtained after querying relevant data needing the analysis, or OLAP,

5. Applications of analyzed data, for example, descriptive, predictive and prescriptive analytics, business processes (BPS), business process automation (BPA), business intelligence (BI), decision modelling and knowledge discovery.

### 5.2.1 Spark SQL

Spark SQL is a component of Spark Big Data Stack. Spark SQL components are DataFrames (SchemaRDDs), SQLContext and JDBC server. Spark SQL at Spark does the following:

1. Runs SQL like scripts for query processing, using catalyst optimizer and tungsten execution engine

2. Processes structured data

3. Provides flexible APIs for support for many types of data sources.

4. ETL operations by creating ETL pipeline on the data from different file-formats, such as JSON, Parquet, Hive, Cassandra and then run ad-hoc querying.

Spark SQL has the following features for analysis:

1. SparkR, PySpark, Python, Java and other language support for coding for data analysis.

2. Provisioning of JDBC and ODBC APIs: Applications in Java and Microsoft programs (such as Excel) need to connect to databases using JDBC (Object Database Connectivity) and ODBC (Object Database Connectivity). Spark APIS enable that connectivity,

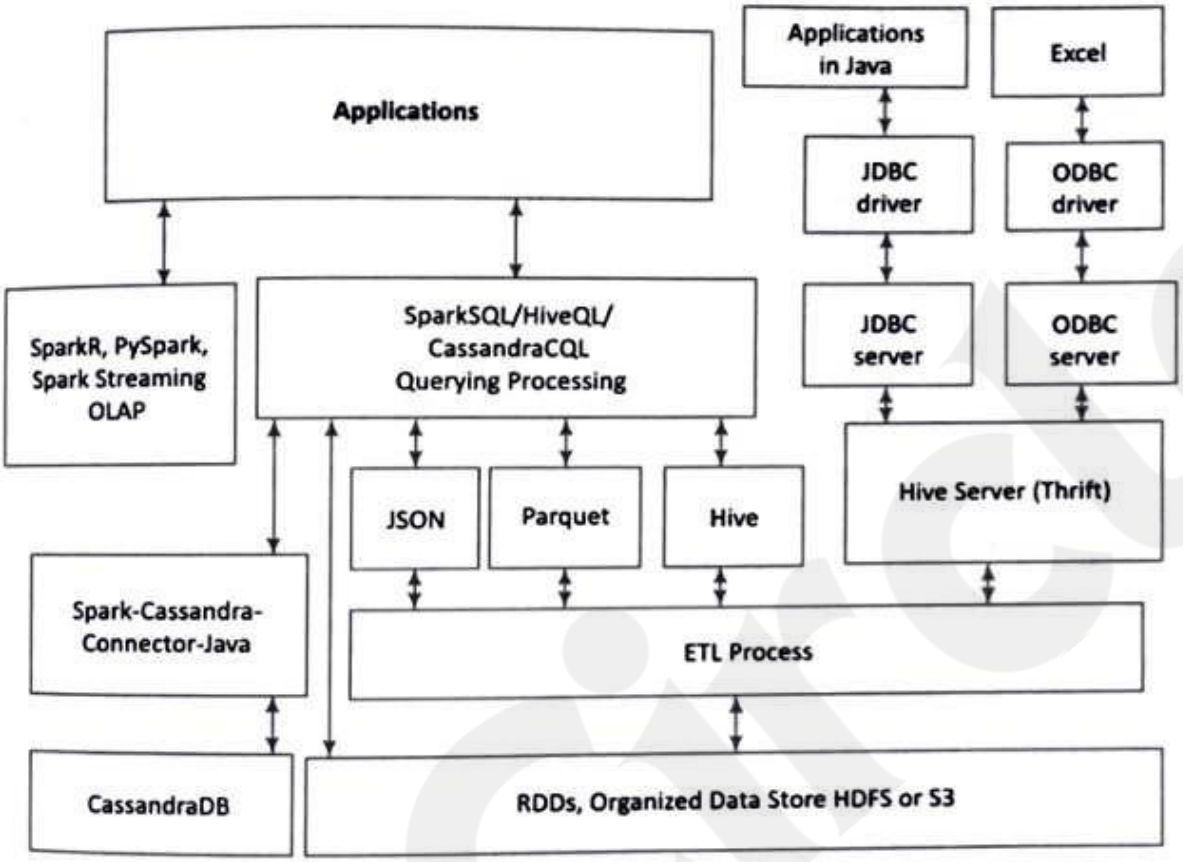
3. Spark SQL enables users to exit their data from different formats, such as Hive, JSON and Parquet, and then transform that into required formats for ad hoc querying. [Ad hoc query is a query just for this purpose' or query 'on the fly'. For example, var newToyQuery = "SELECT * FROM table WHERE ID = "+ toy_puzzleld. The result will be different each time this code executes, depending on the value of toy_puzzleld.

4. Spark SQL processing support inclusion of Hive. Hive support enables the use of Hive tables, database and data warehouse, UDFs and SerDe (serialization and deserialization).

5. Spark SQL supports HiveQL and Cassandra CQL for query processing.

6. Spark Streaming for support to OLTP and structured streaming.

Figure below shows the connectivity of applications with Spark SQL which connects to Object Stores in different formats.




**JDBC Server** An application reads the data tables in RDBMS using a JDBC client (JDBC API at the application). Many applications in Java connect to databases using JDBC driver and server. Spark SQL API provides JDBC connectivity. Command for using JDBC server is as follows:

**./sbin/start-thriftserver.sh -- master sparkMaster**

Hive Server (Thrift) enables a remote Hive client or JDBC driver to send a request to Hive and the server sends response to that. The requests can be in Scala, Java, Python or R.

**JSON, Hive, Parquet Objects**

HDFS is highly reliable for very long running queries. However, 10 operations are slow, Columnar storage is a solution for faster IOS. Columnar storage stores the data portion, presently required for the IOs. Load only columns access during processing. Also, a columnar object Data Store can be compressed or encoded according to the data type. Also, executions of different columns or column partitions can be in parallel at the data nodes.

Hive RC file records store in columns and can be partitioned into row groups. An ORC file consists of row-groups row data called stripes. ORC enables concurrent reads of the same file using separate RecordReaders.' Metadata stored using protocol buffers for addition and removal of fields.

Parquet is a nested hierarchical columnar storage concept. Apache Parquet file is a columnar storage file. The file uses an HDFS block. The block saves the file for running big long queries on Big Data. Each file compulsorily consists of metadata, though a file need not consist of data. An application retrieves the columnar data quickly from Parquet files.

Apache Parquet three projects specify the usages of files for query processing or applications. The projects are (1) parquet-format for specifying formats and Thrift definitions of metadata, (ii) parquet-mr for mplementing the sub-modules in the core components for reading and writing a nested, column-oriented data stream, and (iii) parquet-compatibility for compatibly for read-write in multiple languages.

**Spark DataFrame (SchemaRDD)** A DataFrame is a distributed collection of data organized into nane columns. DataFrame can be used for transformation using filter, join, or groupby aggregation functions.

An RDD method converts Spark DataFrames to RDDs. Each RDD consists of a number of row objects.

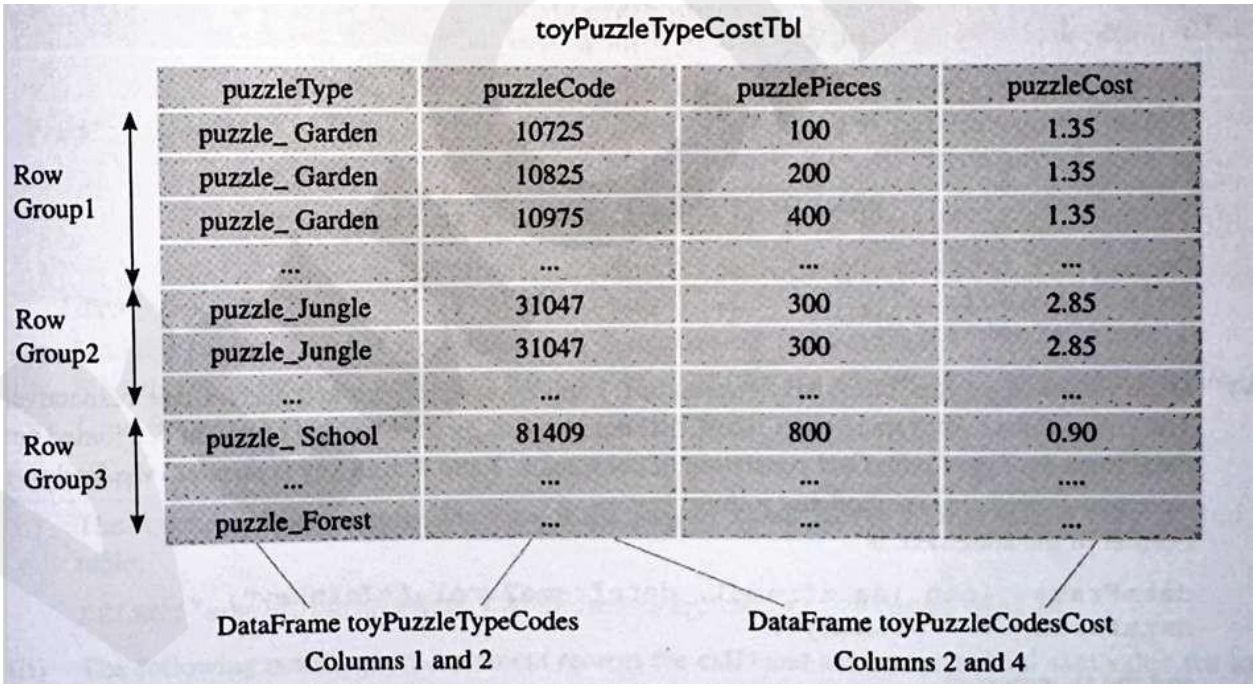
**Creating Spark Dataframe (SchemaRDD) from Parquet and JSON Objects** Data Frames can be created from different data sources. Examples of data sources are

JSON datasets, Hive tables, Parquet row Creating Spark DataFrame (SchemaRDD) from Parquet and JSON Objects DataFrames can be groups, structured data files, external databases and existing RDDs.

The following example explains the creation and usages of DataFrames from the Parquet and JSON objects:

**EXAMPLE**

Assume a table toyPuzzleTypeCostTbl with four columns. Figure below shows the sample table toyPuzzleTypeCostTbl. The columns are puzzle type, puzzle code, number of puzzle pieces and puzzle cost. DataFramel named toyPuzzleTypeCodes consists of columns 1 and 2. DataFrame2 named toyPuzzleCodesCost consists of Columns 2 and 4. The table consists of multiple rows in each row group. The dashed lines point to the columns in the two data frames.




(i) Create a sqlContext from a given SparkContext 'sc'.

(ii) Create a four columns DataFrame using the Parquet file named "toyPuzzleTypeCostTbl". (iii) How will a DataFrame create using a JSON file format file

"toyPuzzleTypeCostTbl"? Create two DataFrames using Java and SqlContext, one with columns for puzzle type and puzzle code, and the other for puzzle code and cost.

(iv) How will two DataFrames join using puzzle code as a join key to create a DataFrame of three columns, 1, 2 and 4?

**SOLUTION**

(i) The following statement creates sqlContext from Spark Context sc:

**SqlContext sqlContext = new org.apache.spark.sql.SQLContext (sc)**

(ii) The following statement creates a DataFrame named toyTypeCost:

**DataFrame toyType Cost = sqlContext.**

**parquet Pile ("toyPuzzleTypeCost Tbl")**

[DataFrame created will have four columns.]

(iii) DataFrame creates using Load() method at the sqlContext.

#To display the contents of Table: "toyPuzzleTypeCostTbl"

**spark.sql ("SELECT * FROM toyPuzzleTypeCost Tbl ")**

#To create DataFrame toyPuzzleTypeCost Tbl

**DataFrame toyPuzzleTypeCostTbl = sqlContext.Load**

**("toyPuzzleTypeCost Tbl", "json")**

The following statements create two DataFrames using DetaFrame toyPuzzleTypeCostbl. One frame of two columns, 1 and 2, puzzle type and puzzle code:

**DataFrame toyTypeCodes = toyPuzzleTypeCostTbl.**

**select (toyPuzzleTypeCostTbl ['puzzleType'], toyPuzzleTypeCostTbl**

**['puzzleCode'])**

Second frame of two columns, 2 and 4, puzzle code and puzzle cost

**DataFrame toyCodesCost = toyPuzzleTypeCostTbl.**

**select (toyPuzzleTypeCostTbl ['puzzleCode'], toyPuzzleTypeCostTbl ['puzzleCost'])**

(iv) DataFramel (toyTypeCodes) has two columns, 1 and 2, as puzzle Type and puzzle Code, respectively. The frame joins dataFrame2 (toyCodesCost) column 4 as puzzle Cost and resultant is a joined new. dataFrame toy TypeCodes Cost consisting of columns 1, 2 and 4. Join key is puzzle Code. Following statements use join() method and joining key.

Format of the statement is

**dataFrame. join (dataframel, dataframe2.col ("JoinKey") = dataframeNew ("JoinKey")**

and the statement is

**dataFrame.join(toyTypeCodes, toyCodesCost.col ("puzzleCode"). equalTo (toyTypeCodes Cost ("puzzleCode")))**

**Using HiveQL for Spark SQL** Spark SQL programming provides two contexts, SQLContext and HiveContext. While using HiveContext, then, commands access the Hive Server only and use HiveQL commands. SQLContext is a subset of Spark SQL, SQLContext does not need the HiveServer (Thrift) Therefore, when needing access to the HiveServer, specify the HiveContext. HiveQL is recommended for Spark SQL. Many resources are available in Hive readily and can directly be used.

**Use of Aggregation and Statistical Functions** Aggregation functions can be used for analysis. Hive consists of count (*), count (expr); sum (col), sum (DISTINCT col), avg (col), avg (DISTINCT col), min (col) and DOUBLE max(col) (Table 4.10). The statistical

functions stdev(), sampleStdev(), variance sample Variance() can be used for analysis with DataFrames in input.

Consider the example below to find sum of the sales of the Jaguar Land Rover model and trace the showroom that recorded the best sales.

**EXAMPLE**

Consider the annual car sales data in the following format:

CarShowroomsCumulativeYearlySales

| Car ShowroomID (csID) | Date (DT) mmddyy | *Jagaur Land Rover* Sale (JLRDS) | *Hexa* Sale (HDS) | *Zest* Sales (ZDS) | *Nexon* Sale (NDS) | *Safari Storme* Sale (SSDS) |
|---|---|---|---|---|---|---|
| 220 | .... | .. | ... | .... | ... | ... |
| 10 | 121217 | 49 | 34 | 164 | 115 | 38 |
| 122 | 121217 | 40 | 141 | 123 | 37 | 88 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |

(i) Find the Jagaur Land Rover annual sale figure over all showrooms.

(ii) Find the showroom ID and Land Rover sales figure for the showroom giving maximum Jaguar Land Rover sales.

**SOLUTION**

(i) The following query returns the sum of the Jagaur Land Rover annual sale in column 3 of the table:

**SELECT sum (JLRDS) FROM CarShowroomsCumulativeYearlySales**

(ii) The following nested query statement returns the csID and maximum annual sale value for Land Rover:

SELECT CSID, saleJL (max (map (csID, ID, JLRDS, saleJL))) FROM CarShowroomsCumulativeYearlySales

### 5.2.2 Using Python Advanced Features with Spark SQL

Python is a general purpose, interpreted, interactive, object oriented and high level programming language. Python defines the basic data types, containers, lists, dictionaries, sets, tuples, functions and classes. Python Standard Library is very extensive. The libraries for regular expressions, documentation generation, unit testing, web browsers, threading, databases, CGI, email, image manipulation and a lot of other functionalities are available in Python.

Python programming is a strong combination of performance and features in the same bundle of codes. Spark SQL binds with Python easily. Python has the expressive program statements. Spark SQL features together with Python help a programmer to build challenging applications for Big Data. The following example explains the use of PySpark, Python along with the Spark SQL:

**Example**

***Questions:***

1. **How is HiveContext and Spark SQL used?**
2. **How does PySpark use a row object?**
3. **Assume a JSON file, toyTypeProductTbl of row objects. Do a file load and query sent to the file: toyPuzzleProduct**

The table (toyPuzzleProduct) contains:

| | ProductCategory | ProductId | ProductName |
|---|---|---|---|
| Row Object1 | Toy_Airplane | 10725 | Lost Temple |
| Row Object2 | Toy_Airplane | .... | ..... |
| Row Object3 | ..... | ..... | ..... |

***Solution:***

**Import Spark SQL using:**

```
import Spark SQL
```

**Importing a row object using HiveContext:**

```
from pyspark.sql import HiveContext, toyPuzzleTypeCost
```

**Load the JSON file and query it:**

```
input = hiveCtx.jsonFile(toyTypeProductTbl)
input.registerTempTable("toyPuzzleProduct")
```

**Query to find Product_ID name:**

```
Product_ID_name = hiveCtx.sql(
    "SELECT ID, name FROM toyPuzzleProduct ORDER BY ProductCategory"
)
```

### 5.2.2.1 Python Libraries for Analysis

NumPy and SciPy are open source downloadable libraries for numerical (Num) analysis and scientific (Sci) computations in Python (Py). Python has open source library packages, NumPy, SciPy, Scikit-leam, Pandas and StatsModel, which are widely used for data analysis. Python library, matplotlib functions plot the mathematical functions.

Spark added a Python API support for UDFS. The functions take one row at a time. That requires overhead (additional codes) for SerDe. Earlier data pipelines first defined the UDFS in Java or Scala, and then invoked them from Python. Spark 2.3 provisions for vectorized UDFS (VUDFs) and Apache Arrow facilitates VUDFS which enables high performance Python UDFS for SerDe and data pipelines.

**NumPy** NumPy includes (i) N-dimensional array object, array and vector mathematics; (ii) linear algebraic functions, Fourier transform and random number functions; (iii) sophisticated (broadcasting) functions; and (iv) tools for integrating with C/C++ and Fortran codes.

NumPy provides multi-dimensional efficient containers of generic data and definitions of arbitrary data types. NumPy integrates easily with a wide variety of databases. NumPy provides import, export (load/save) elements in the arrays, indexing, sub-setting and slicing of the arrays, scalar and vector mathematics (such as files, creation of arrays, inspection of properties, copying, sorting and reshaping, addition and removal of X, power, sqr, sin, log, ceil - round up to nearest int, floor-round down up to the nearest int, round -round to nearest integer). NumPy also provides statistical functions. Table below gives the examples of NumPy functions for data analysis problems.

**Table: Examples of NumPy functions for data analysis problems**

| Function | Description | Function | Description |
|---|---|---|---|
| np.loadtxt('file.txt) | Loads a text file | np.mean(arr, axis= 0) | Description Returns mean along a specific axis |
| np.genfromtxt('file.csv', delimiter==',') | Loads a csv file with comma as the delimiter between records | np.sum(); np.min(), | Returns the sum and minimum of the array |
| np.savetxt('file.txt,' arr, delimiter=' ') | Saves a text file which is an array of strings separated by a space each. | np.max(arr, axis= 0) | Returns the maximum along a specific axis |
| np.genfromtxt('file.csv', | Saves a CSV file | np.var(arr) | Returns the variance |

| | | | |
|---|---|---|---|
| arr, delimiter=',') | which is an array of strings separated by a comma each. | | of array |
| arr.sort() | Sorts an array | np.std(arr,axis=0) | Returns the standard deviation of a specific axis |
| np.add(arr1, arr2) | Performs a vector addition of array 1 and array 2 | np.corr() | Returns the correlation coefficient |

**SciPy** SciPy adds on top of NumPy. It includes MATLAB files and special functions, such as routines for numerical integration and optimization. SciPy defines some useful functions for computing distances between a set of points.

SciPy includes (i) interactions with NumPy, (ii) creation of dense and open mesh grids, (iii) shape manipulation functions, (iv) polynomial and vectoring functions, (v) real and imaginary functions, and casting an object to a data type, and (vi) matrix creation and matrices routines and usages of spark matrices. Table below gives few examples of SciPy functions for scientific computational problems.

**Table: Examples of SciPy functions for data analysis problems**

| Function | Description | Function | Description |
|---|---|---|---|
| np.c_[b, c] | Create Stacked column- wise array | np.cast ['f'] (np.pi) | Casts an object into a data type |
| b.flatten() | Flattens the array | from numpy import polyID<br>p = polyID ([2, 3, 4]) | Creates a polynomial object p |

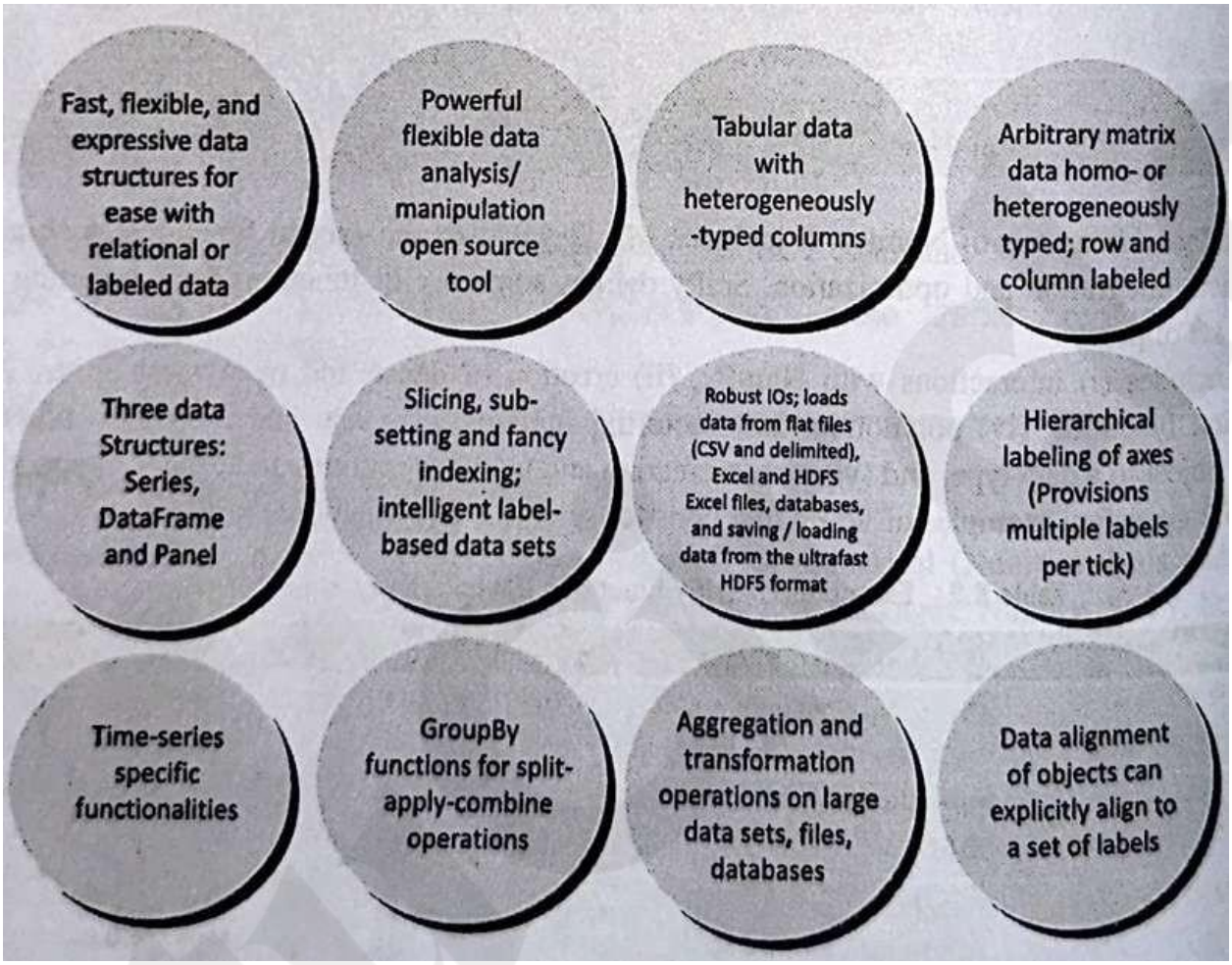
| | | | |
|---|---|---|---|
| np.vsplit (c, 2) and np.hsplit (d, 2) | Functions for vertically splitting and horizontally splitting the array at the end of the second index | A.I,<br>A.T.<br>A.H | Inverses, transposes, and conjugate transposes the matrix, A |
| linag.det(A) | Returns the determinate of A | Np.selet([c<4], [c*2]) | Returns values from a list of arrays depending on the conditions |
| np.img(c) | Returns the imaginary part of the array elements | A=np.matrix([3,4], [5,6]) | Creates a matrix |

**Panda** Panda derives its name from usages of a data structure called Panel. The first three characters Pan in Panda stand for the term 'panel'. The next two characters da in Panda stand for data. The Panda package considers three data structures: Series, DataFrame and Panel.

DataFrame is a container for Series. Panel is a container for DataFrame objects. The Panel objects can be datasets. The data need not be labeled. This means datasets, objects and DataFrames can be placed into a inserted or removed similar to as in a dictionary. DataFrame may be a DataFrame of statistical or observed Panda data structure without labels.

Panel is a container for three-dimensional data. Panel is a widely used term in econometrics. Three axes describe operations involving panel data. For example, panel data in econometric analysis. Items can be considered as along axis 0 of an inside DataFrame (set of columns). Index (rows) of each of the DataFrames correspond to axis 1 (major axis). Columns of each of the DataFrames correspond to axis 2 (minor axis)

Panel 4D consists of labels as 0 - axis, items - axis 1, major axis - axis 2 and minor axis - axis 3. Figure below shows the main features of Pandas package.




Pandas package includes the following provisions:

1. Database style DataFrames merge, join, and concatenation of objects

2. RPy interface for R functions plus additional functions

3. Panda ecosystem has statistics, machine learning, integrated development environment (IDE), API and several out of core features

4. SQL like features: SELECT, WHERE, GROUPBY, JOIN, UNION, UPDATE and DELETE

5. GroupBy feature of split-apply-combine with the steps as: (i) an object such as table, file or document splits into groups, (ii) iterate through the groups and select a group for aggregation, transformation and/or filtration. The instance method can be dispatched and applied in a manner similar to the aggregation/transformation function

6. Size mutability, which means that columns can be inserted and deleted from DataFrame and higher dimensional objects

7. Slicing and dicing a collection of DataFrame objects. The names of axes can be somewhat arbitrary in a Panel. (Arbitrary means the axes need not be named a1, a2, ..., an, and can be year, car model, sales, ...). If slicing function slices the first dimension, the lower dimension objects are obtained.

#### 5.2.2.2 User-Defined Functions (UDFs)

The functions take one row at a time. This requires overhead for SerDe. Data exchanges take place between Python and JVM. Earlier the data pipeline (between data and application) defined the UDFs in Java or Scala, and then invoked them from Python while using Python libraries for analysis or other application. SparkSQL UDFS enable registering of themselves in Python, Java and Scala.

The SQL calls the UDFS. This is a very popular way to expose advanced functionality to SQL users. User codes call the registered UDFS into the SQL statements without writing the detailed codes. The following example demonstrates how a UDF is created in PySpark.

**EXAMPLE**

Create a UDF, udfCost Plus () in pandas. The table column puzzle Cost creates using jigsaw puzzle_info.txt from an RDD. Write a UDF which increases the costs in the column, puzzle_cost USD by 10%. (The UDF takes one row at a time as input.)

**SOLUTION**

Following are the Python statements

from pyspark.sql.functions import udf

[Use udf to define a row at a time udf.]

@udfCostPlus ('float')

[Input and output costs are two values both for a single float variable, v.]

def plusTenPercent (v) ;

return v + 0.1 x v;

df.withColumn ('v4', puzzle_cost USD (df.v))

[Data Frame df has v4 as puzzleCost in the fourth column.]

Dataset at Example consists of car sales data. Column 1 represents car showroom ID. The ID key is has the Jaguar Land Rover sales figures in more than 300 rows for more than 300 dates. Sales figures of four present in all date fields on which the sales were recorded during the year. Corresponding to an ID, column 3 other models are in columns 4, 5, 6 and 7.

The product sales analysis is widely performed in many businesses. Writing a UDF for analysis of sales once and using it for different products whenever and wherever desired reduces coding efforts. This also, integrates new functions (UDFs) in higher level language with their lower level language implementations. Also, writing UDFs are

helpful when built-in functionalities in a currently used tool needs additional fucntionalities.

A UDF using aggregation function max() calculates the Land Rover sales and traces the showroom giving maximum Jaguar Land Rover sales. The UDF is of great help to perform similar analyses on different models of the car.

Java class can be created user defined method (UDF) product SalesAnalysis (). Python scripts ca also create the UDF to find that sales point ID and total yearly sale for that sales point from which the total is highest for a product. The UDF will be reusable not only for car sales analysis but also for analysis of sales of many companies, such as ACVM or Toy Company.

Python provides a register function: hiveContext (sc).registerFunction (). The command can be hiveContext (sc). registerFunction ("csIDn1", int: bestYearlySalesModell, LongType (), ("csIDn2", int: best YearlySalesModel2, LongType (), ...). csIDn1 car-showroom ID1, bestYearlySalesModel1 is best yearly sale of model 1.

#### 5.2.2.3 Vectorized User Defined Functions (VUDFS)

Python UDFS express data in detail. Therefore, Python UDFs, block-level UDFS with block-level arguments and return types, conversions or transformations are widely used in ETL or ML applications. Spark Arrow facilates columnar in-memory analytics, which results in high performance of Python UDFS, SerDe and data pipelines.

VUDFs use series data structure (meaning one-dimensional array or tuples). Spark 2.3 (2018) provisions for using vectorized UDFS (VUDFs). Apache Arrow 0.8.0 (release date December 18, 2017) facilitates usages of VUDFS. Pandas UDF, pandas_UDF uses the function to create a VUDF with (i) pandas.Series as input to the UDF, (ii) pandas.Series as output from the UDF, (iii) no grouping using GroupBy, (iv) output size same as input, and (v) returns the same data types as specified type in return pandas.Series. The following example explains the use of VUDF.

**EXAMPLE**

Create a vectorized UDF (VUDF). First define a pandas_UDFCostPlus for increasing cost puzzle_cost_USD of toys in puzzle_Costs RDD created from jigsaw puzzle_info.txt,.

**SOLUTION**

Following are the Python statements from pyspark.sql.functions import pandas_udf:

```
from pyspark.sql.functions import pandas_udf

[Use pandas_udf to define a vectorized udf.]

@pandas_udf Cost Plus ('float')

[Input/output are both a pandas.Series of elements with data type float.]

def vectorized_plusTenPercent (v):

return v4 + 0.1

df.withColumn ('v4', vectorized_plusTenPercent (df.v))

[Use udf to define a DataFrame vudf.]
```

### 5.2.2.4 Grouped Vectorized UDFS (GVUDFS)

Grouped Vectorized UDFS (GVUDFS) use Panda library split-apply-combine pattern in data analysis. The GVUDF group function operates on all the data for a group, such as operate on all the data, "for each carshowroom, compute yearly sales".

GVUDF steps are:

1. Splits a Spark DataFrame into groups based on the conditions specified in the groupby operator
2. Applies a vectorized user-defined function (pandas.DataFrame -> pandas.DataFrame) to each group
3. Combines into new group

4. Returns the results as a new Spark DataFrame

Pandas GVUDF, pandas_GVUDF, (i) uses the function similar to pandas_VUDF, (ii) pandas.DataFrame as input to the GVUDF, (ii) pandas.DataFrame as output from the GVUDF, (iii) grouping semantics defined using clause GroupBy, (iv) output size can be any and can be grouped and can be distinct from input, and (v) returns data type is a StructType. The type defines a column name and the type of the returned pandas. DataFrame.

The following example explains GVUDF for adding 10% in a cost of group of rows for toy products.

**EXAMPLE**

Add 10% cost in each value of item cost in a group of rows. Use GVUDF to define a DataFrame costTen Percet Plus GVUDF.

**SOLUTION**

Following are the Python statements from pyspark.sql.functions import pandas_udf:

**from pyspark.sql.functions import pandas_udf**

[Use pandas_udf to define a grouped vectorized udf.]

@pandas_udf (df.schema)

# Input/output are both a pandas. DataFrame

def costTenPercet Plus (pdf):

return pdf.assign (v=add (pdf.v + 0.1xpdf.v))

df.groupby('id').apply (costTenpercet Plus)

### 5.2.3 Data Analysis Operations

Examples of operations required in the above analysis are given below:

1. Filtering single and multiple columns

2. Creating a top-ten list with values or percentages

3. Setting up sub-totals

4. Creating multiple-field criteria filters

5. Creating unique lists from repeating field data

6. Finding duplicate data with specialized arrays, and using the remove duplicates command removing outliers

7. Multiple key sorting

8. Counting the number of unique items in a list

9. Using SUMIF and COUNTIF functions

10. Working with database functions, such as DSUM and DMAX

11. Converting lists to tables.

**5.2.3.1 Removing Outliers for Data Quality Improvement for Analysis**

Outliers are data which appear as they do not belong to the data set. The Outliers are generally results of human data-entry errors, programming bugs, some transition effect or phase lag in stabilizing the data vale to the true value.

The actual outliers need to be removed from the data set. For example, missing decimal in the cost of toy US$ 1,85 will make the cost 100 times more for a single toy. The result will thus be affected by a smal or large amount. When valid data is identified as an outlier, then also the results are affected.

The statistical mean is computed from the product of each observed value v of Values with probability (or weight) P and then taking the average. The variance equals the

difference of a value with respect to the mean, then square that, and then average the results. Standard deviation is just the square root of the variance

The following example explains the Python codes for removing outliers.

**EXAMPLE**

How will you remove outliers in values in a column?

**SOLUTION**

Transform RDD string or other to numeric data so that statistical methods compute and remove those who have larger distanceNumerics.

distanceNumerics = distances. map (PySpark string: float (string))

stats = distanceNumerics.stats()

[stats() means a statistical function, such as mean(), stdev()]'

statdev = std.stdev ()

mean = stats.mean ()

reasonableDistances = distance Numerics.filter (PySpark values: maths.fabs (values-mean) < 3 xstdev)

[Assume that distances that are reasonable are less than three times the standard deviation. Distance means difference with respect to mean or peak value.]

print reasonableDistances.collect ()

[Print the values within reasonable distances.]

********END********