SVERI's College of Engineering, Pandharpur Department of Computer Science & Engineering

Big Data Analytics Lab Manual

Ms. S. S. Kadam
Ms. K. I. Chouhan
2022-23 SEM-II

EXPERIMENT LIST

VISION	
Institute Vision	**Department Vision**
To be recognized among the best institute in India for excellence in technical education.	To be nationally recognized for excellence in education augmented by research in the field of Computer Science and Engineering.

MISSION

Institute Mission

- To impart value-based technical education through innovation and excellence, empowering individuals to become leaders in their fields to create positive impact.
- To create an ambiance of academic excellence, research, and life skills by fostering a learning environment that empowers individuals to achieve their full potential.
- To foster strong relationships amongst all our stakeholders by inculcating a personal touch and mutual respect in all our interactions.

Department Mission

- To impart value-based education in Computer Science & Engineering, through effective teaching and learning approaches.
- To create ambiance for academic excellence through fruitful interaction among various stakeholders.
- To inculcate best practices for innovative research, competitive employability and sustainable entrepreneurship development.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

The Department of Computer Science and Engineering has as its PEOs to produce graduate who:

1. Apply the Computer Science domain specific knowledge and skills in the growing software and related industries.
2. Demonstrate leadership, professional ethics, project management and finance related attributes as employees or employers.
3. Engage in life-long learning for professional advancement to develop innovative solutions for individual or societal problems.
4. Demonstrate strong communication skills and ability to function effectively as an individual and part of a team.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

Engineering Graduates will be able to:

1. understand & design computer system using knowledge of Digital Techniques, Micro-Processor, Computer Organization, Advanced Computer Architecture, Operating System, System Programming, Compiler Construction, Application Softwares, etc.
2. interpret, analyze and design software system programming knowledge using Algorithmic Skills, Web Technology, Big Data Analytics, Networking Fundamentals, Machine Learning and Internet of Things.
3. adopt applications in emerging fields of Computer Science & Engineering.

PROGRAMME OUTCOMES (POs)	
Students graduating from Computer Science and Engineering will demonstrate:	
1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	**Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Experiment No. 01

Title: Basic big data operations using NumPy, SciPy & Pandas.

Theory:

NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem. Some of the important attributes of a NumPy object are:

1. **Ndim:** displays the dimension of the array
2. **Shape:** returns a tuple of integers indicating the size of the array
3. **Size:** returns the total number of elements in the NumPy array
4. **Dtype**: returns the type of elements in the array, i.e., int64, character
5. **Itemsize:** returns the size in bytes of each item
6. **Reshape**: Reshapes the NumPy array

Machine learning uses vectors. Vectors are one-dimensional arrays. It can be represented either as a row or as a column array.
Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Dataframe. It is like a spreadsheet with column names and row labels. Some commonly used data structures in pandas are:

1. **Series objects**: 1D array, similar to a column in a spreadsheet
2. **DataFrame objects:** 2D table, similar to a spreadsheet
3. **Panel objects:** Dictionary of DataFrames, similar to sheet in MS Excel

Dataframes can also be easily exported and imported from CSV, Excel, JSON, HTML and SQL database. Some other essential methods that are present in dataframes are:

1. **head():** returns the top 5 rows in the dataframe object
2. **tail():** returns the bottom 5 rows in the dataframe
3. **info():** prints the summary of the dataframe
4. **describe():** gives a nice overview of the main aggregated values over each column

1) Analytic aggregate functions

AVG() (from the example above) is an aggregate function. The OVER clause is what ensures that it's treated as an analytic (aggregate) function. Aggregate functions take all of the values within the window as input and return a single value.
MIN() (or MAX()) - Returns the minimum (or maximum) of input values
AVG() (or SUM()) - Returns the average (or sum) of input values
COUNT() - Returns the number of rows in the input

2) Analytic navigation functions

Navigation functions assign a value based on the value in a (usually) different row than the current row.
FIRST_VALUE() (or LAST_VALUE()) - Returns the first (or last) value in the input
LEAD() (and LAG()) - Returns the value on a subsequent (or preceding) row

3) Analytic numbering functions

Numbering functions assign integer values to each row based on the ordering.
ROW_NUMBER() - Returns the order in which rows appear in the input (starting with 1)
RANK() - All rows with the same value in the ordering column receive the same rank value, where the next row receives a rank value which increments by the number of rows with the previous rank value.

Conclusion:
In this, experiment we implemented analytical function using NumPy, SciPy & Pandas.

Experiment No. 02

Title: Implementation of Plotting, Filtering and Cleaning a CSV File Data Using NumPy & Pandas.

Theory:

Data Cleaning With Pandas

Data scientists spend a huge amount of time cleaning datasets and getting them in the form in which they can work. It is an essential skill of Data Scientists to be able to work with messy data, missing values, and inconsistent, noisy, or nonsensical data. To work smoothly, python provides a built-in module, Pandas. Pandas is the popular Python library that is mainly used for data processing purposes like cleaning, manipulation, and analysis. Pandas stand for "Python Data Analysis Library". It consists of classes to read, process, and write csv files. There are numerous Data cleaning tools present, but the Pandas library provides a really fast and efficient way to manage and explore data. It does that by providing us with Series and DataFrames, which help us represent data efficiently and manipulate it in various ways.

Let's get started with data cleaning step by step.

```
#importing module
import pandas as pd
```

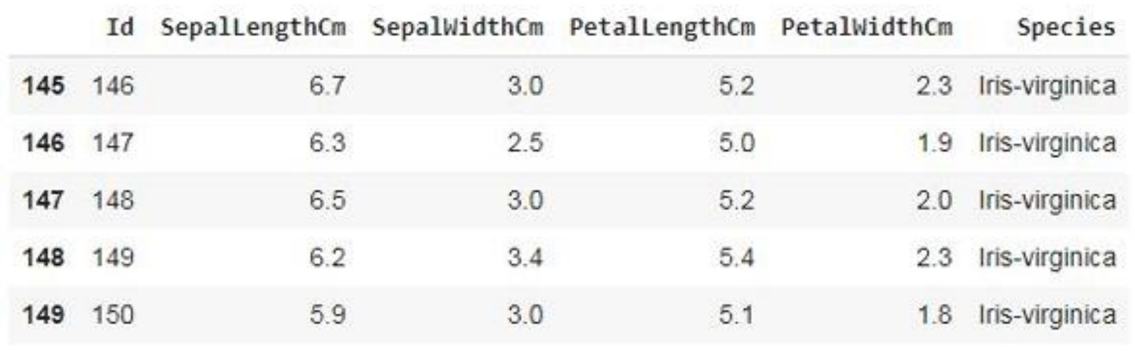
Step 1: Import Dataset

To import the dataset, we use the read_csv() function of pandas and store it in the pandas DataFrame named as data. As the dataset is in tabular format, when working with tabular data in Pandas, it will be automatically converted into a DataFrame. DataFrame is a two-dimensional, mutable data structure in Python. It is a combination of rows and columns like an excel sheet.

Python Code:

The head() function is a built-in function in pandas for the dataframe used to display the rows of the dataset. We can specify the number of rows by giving the number within the parenthesis. By default, it displays the first five rows of the dataset. If we want to see the last five rows of the dataset, we use the tail()function of the dataframe like this:

```
#displayinf last five rows of dataset
data.tail()
```



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

Step 2: Merge Dataset

Merging the dataset is the process of combining two datasets in one and lining up rows based on some particular or common property for data analysis. We can do this by using the merge() function of the dataframe. Following is the syntax of the merge function:

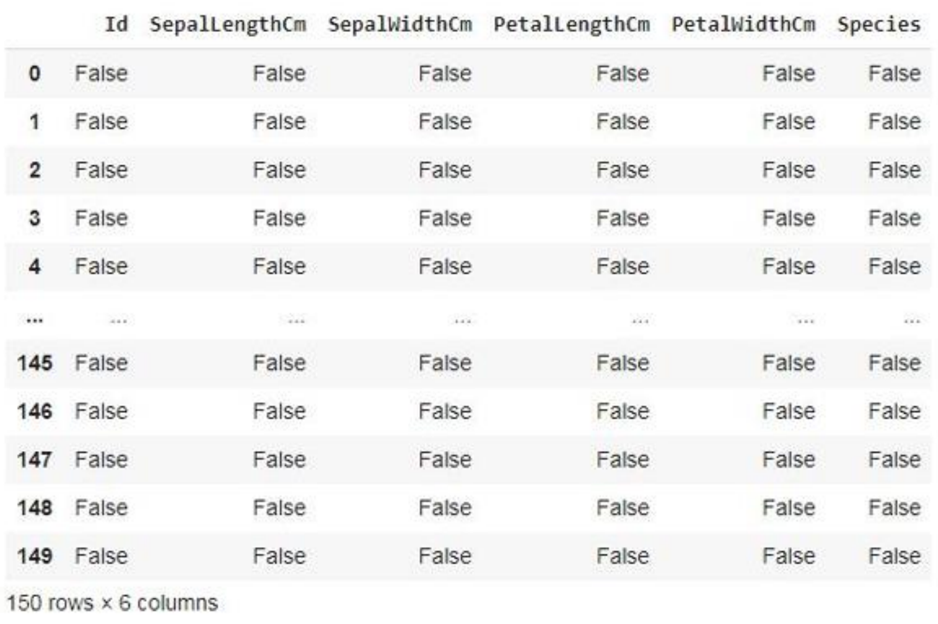
```
DataFrame_name.merge(right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'),
copy=True, indicator=False, validate=None)
```

Step 3: Rebuild Missing Data

To find and fill in the missing data in the dataset, we will use another function. There are 4 ways to find the null values if present in the dataset. Let's see them one by one:

Using isnull() function:

```
data.isnull()
```



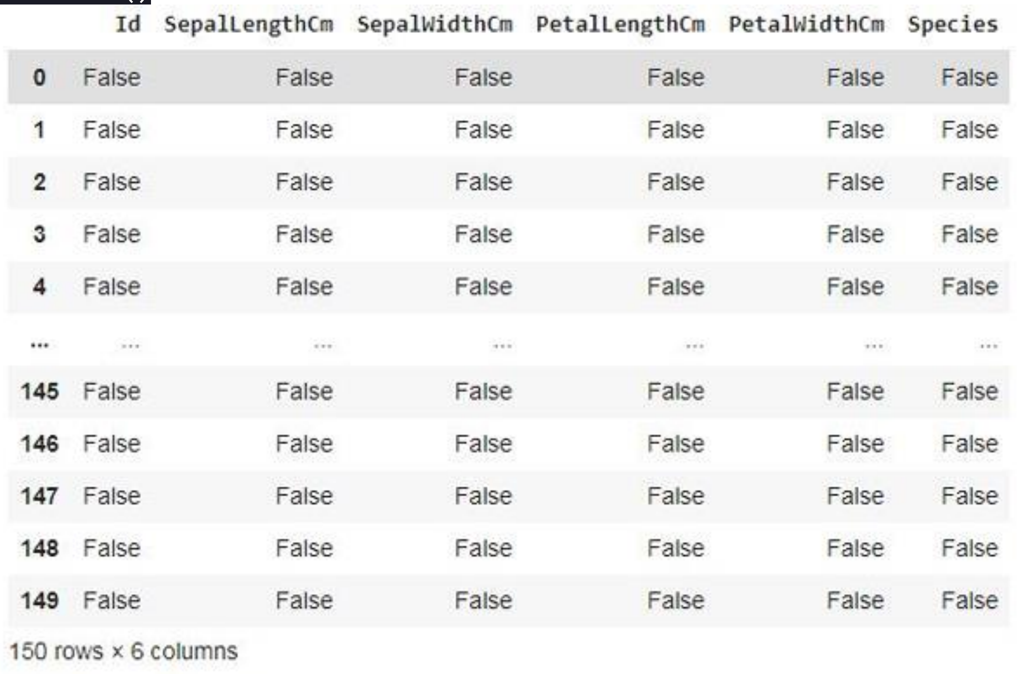
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

This function provides the Boolean value for the complete dataset to know if any null value is present or not.

Using isna() function:

```
data.isna()
```



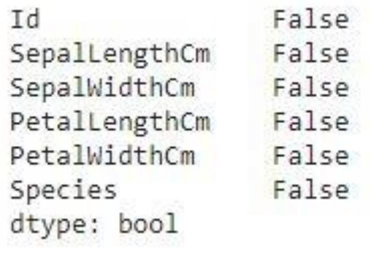
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

This is the same as the isnull() function. Ans provides the same output.

Using isna().any()

```
data.isna().any()
```

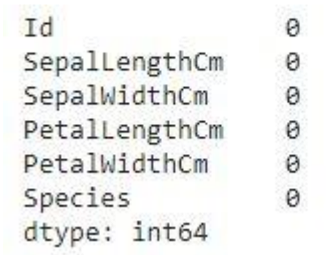


```
Id               False
SepalLengthCm    False
SepalWidthCm     False
PetalLengthCm    False
PetalWidthCm     False
Species          False
dtype: bool
```

This function also gives a boolean value if any null value is present or not, but it gives results column-wise, not in tabular format.

Using isna(). sum()

```
data.isna().sum()
```

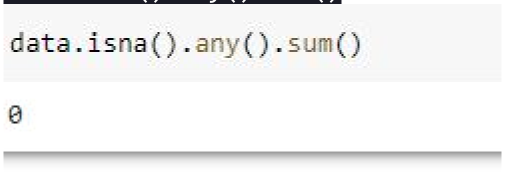


```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

This function gives the sum of the null values preset in the dataset column-wise.

Using isna().any().sum()

```
data.isna().any().sum()
```



```
data.isna().any().sum()

0
```

This function gives output in a single value if any null is present or not.

There are no null values present in our dataset. But if there are any null values preset, we can fill those places with any other value using the fillna() function of DataFrame.Following is the syntax of fillna() function:

```
DataFrame_name.fillna(value=None, method=None, axis=None, inplace=False, limit=None,
downcast=None)
```

This function will fill NA/NaN or 0 values in place of null spaces. You may also drop null values using the dropna method when the amount of missing data is relatively small and unlikely to affect the overall.

Step 4: Standardization and Normalization

Data Standardization and Normalization is a common practices in machine learning.

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

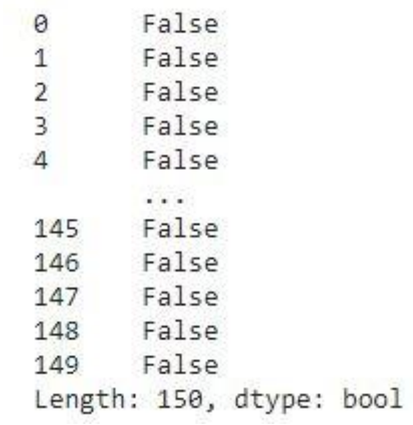
Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

This step is not needed for the dataset we are using. So, we will skip this step.

Step 5: De-Duplicate Data

De-Duplicate means removing all duplicate values. There is no need for duplicate values in data analysis. These values only affect the accuracy and efficiency of the analysis result. To find duplicate values in the dataset, we will use a simple dataframe function, i.e., duplicated(). Let's see the example:

```
data.duplicated()
```



```
0      False
1      False
2      False
3      False
4      False
       ...
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

This function also provides bool values for duplicate values in the dataset. As we can see, the dataset doesn't contain any duplicate values. If a dataset contains duplicate values, it can be removed using the drop_duplicates() function. Following is the syntax of this function:

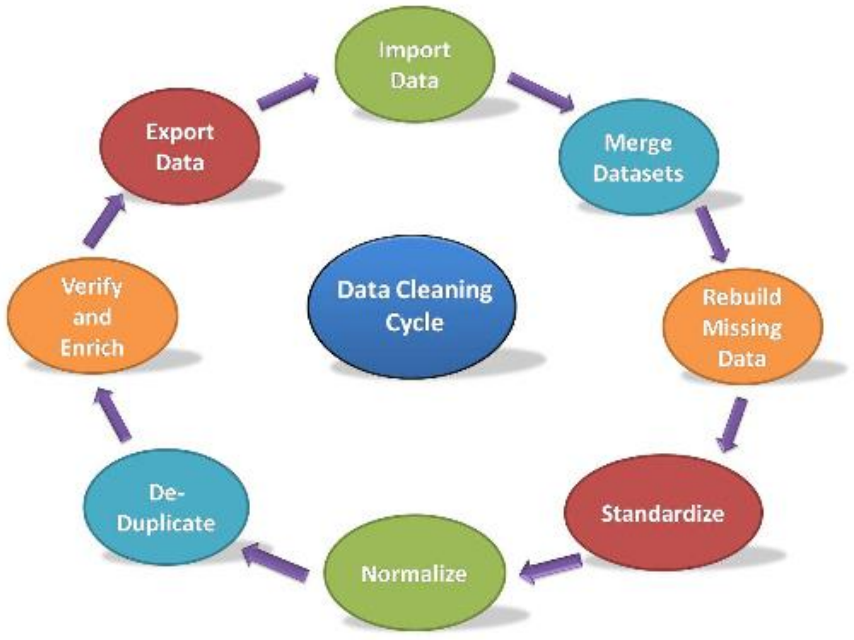
```
DataFrame_name.drop_duplicates(subset=None, keep='first', inplace=False,
ignore_index=False)
```

Step 6: Verify and Enrich the Data

After removing null, duplicate, and incorrect values, we should verify the dataset and validate its accuracy. In this step, we have to check that the data cleaned so far is making any sense. If the data is incomplete, we have to enrich the data again by data gathering activities like approaching the clients again, re-interviewing people, etc. Completeness is a little more challenging to achieve accuracy or quality in the dataset.

Step 7: Export Dataset

This is the last step of the data-cleaning process. After performing all the above operations, the data is transformed into a clean dataset, and it is ready to export for the next process in Data Science or Data Analysis.



Conclusion:

Data cleaning is a critical task in data science that helps ensure the accuracy and reliability of analysis and decision-making. Through data cleaning, errors can be removed, data quality can be improved, and the data can be made more accurate and complete. By utilizing the various techniques and tools available for data cleaning in the Python Pandas library, data scientists can gain insights from the raw data and make better informed decisions. In this experiment we, different pre-processing tasks using python libraries.

Experiment No. 03

Title: Implement multidimensional visualization by adding variables such as color, size, shape, and label by using Tableau.

Theory:
Control color, size, shape, detail, text, and tooltips for marks in the view using the Marks card. Drag fields to buttons on the Marks card to encode the mark data. Click the buttons on the Marks card to open Mark properties.

Assign colors to marks
To assign a color to marks in the view, do one of the following:

- On the Marks card, click **Color**, and then select a color from the menu.
  This updates all marks in the view to the color you choose. All marks have a default color, even when there are no fields on **Color** on the **Marks** card. For most marks, blue is the default color; for text, black is the default color.
- From the **Data** pane, drag a field to **Color** on the Marks card.
  Tableau applies different colors to marks based on the field's values and members. For example, if you drop a discrete field (a blue field), such as Category, on Color, the marks in the view are broken out by category, and each category is assigned a color.

If you drop a continuous field, such as SUM (sales), on Color, each mark in the view is colored based on its sales value.

Edit colors

- To change the color palette or customize how color is applied to your marks:
- On the Marks card, click **Color** > **Edit Colors.**

Change the size of marks

To change the size of marks in the view, do one of the following:

- On the **Marks** card, click **Size**, and then move the slider to the left or right.

  The Size slider affects different marks in different ways, as described in the following table.

MARK TYPE	DESCRIPTION
Circle, Square, Shape, Text	Makes the mark bigger or smaller.
Bar, Gantt Bar	Makes bars wider or narrower.
Line	Makes lines thicker or thinner.
Polygon	You cannot change the size of a polygon.
Pie	Makes the overall size of the pie bigger and smaller.

  The size of your data view is not modified when you change marks using the Size slider. However, if you change the view size, the mark size might change to accommodate the new formatting. For example, if you make the table bigger, the marks might become bigger as well.

- From the **Data** pane, drag a field to **Size** on the Marks card.

Edit marks sizes

To edit the size of marks, or change how size is being applied to marks in the view:

1. On the Size legend card (which appears when you add a field to Size on the Marks card), click the drop-down arrow in the right-hand corner and select **Edit Sizes**.

2. In the Edit Sizes dialog box that appears, make your changes and then click **OK**.

   The options available depend on whether the field being applied to Size is a continuous or discrete field.

   For continuous fields, you can do the following:

   - For **Sizes vary**, click the drop-down box and select one of the following:

     - **Automatically** - Selects the mapping that best fits your data. If the data is numeric and does not cross zero (all positive or all negative), the From zero mapping is used. Otherwise, the By range mapping is used.

     - **By range** - Uses the minimum and maximum values in the data to determine the distribution of sizes. For example, if a field has values from 14 to 25, the sizes are distributed across this range.

     - **From zero** - Sizes are interpolated from zero, assigning the maximum mark size to the absolute value of the data value that is farthest from zero.

   - Use the range slider to adjust the distribution of sizes. When the From zero mapping is selected from the Sizes vary drop-down menu, the lower slider is disabled because it is always set to zero.

   - Select **Reversed** to assign the largest mark to the smallest value and the smallest mark to the largest value. This option is not available if you are mapping sizes from zero because the smallest mark is always assigned to zero.

     For discrete fields, you can do the following:

     - Use the range slider to adjust the distribution of sizes.

     - Select **Reversed** to assign the largest mark to the smallest value and the smallest mark to the largest value.

Continuous axis mark sizing

For views where the mark type is **Bar** and there are continuous (green) fields on both **Rows** and **Columns**, Tableau supports additional options and defaults for sizing the bar marks on the axis where the bars are anchored.

- The bar marks in histograms are continuous by default (with no spaces between the marks), and are sized to match the size of the bins.
- When there is a field on **Size**, you can determine the width of the bar marks on the axis where the bars are anchored by using the field on **Size**. To do this, click the **Size** card and select **Fixed**.
- When there is no field on **Size**, you can specify the width of the bar marks on the axis where the bars are anchored in axis units. To do this, click the **Size** card, choose **Fixed**, and then type a number in the **Width in axis units** field.

Add labels or text for marks

To add mark labels or text to the visualization:

- From the **Data** pane, drag a field to **Label** or **Text** on the Marks card.

Conclusion:

In this experiment, we studied about different visualization controls in tableau.

Experiment No. 04

Title: Apply Filters on Dimensions and Measures for any dataset using tableau.

Theory:

When working with a text table, the Label shelf is replaced with Text, which allows you to view the numbers associated with a data view. The effect of text-encoding your data view depends on whether you use a dimension or a measure.

- Dimension – When you place a dimension on **Label** or **Text** on the Marks card, Tableau separates the marks according to the members in the dimension. The text labels are driven by the dimension member names.
- Measure – When you place a measure on **Label** or **Text** on the Marks card, the text labels are driven by the measure values. The measure can be either aggregated or disaggregated. However, dis-aggregating the measure is generally not useful because it often results in overlapping text.
- Text is the default mark type for a text table, which is also referred to as a cross-tab or a PivotTable.
- Separate marks in the view by dimension members
- To separate marks in the view (or add more granularity):
- From the **Data** pane, drag a dimension to **Detail** on the Marks card.

When you drop a dimension on **Detail** on the Marks card, the marks in a data view are separated according to the members of that dimension. Unlike dropping a dimension on the **Rows** or **Columns** shelf, dropping it on **Detail** on the Marks card is a way to show more data without changing the table structure.

Add tooltips to marks
Tooltips are details that appear when you hover over one or more marks in the view. Tooltips are also convenient for quickly filtering or removing a selection, or viewing underlying data. You can edit a tooltip to include both static and dynamic text. You can also modify which fields are included in a tooltip and whether you want to be able to use those fields to select marks in the view.

Tooltip options
After you open the Edit Tooltip dialog box, there are several options that you can choose from to format the tooltips in your view and configure their behavior.

Edit shapes

By default, ten unique shapes are used to encode dimensions. If you have more than 10 members, the shapes repeat. In addition to the default palette, you can choose from a variety of shape palettes, including filled shapes, arrows, and even weather symbols.

1. Click **Shape** on the **Mark**s card, or select **Edit Shape** on the legend's card menu.

2. In the Edit Shape dialog box, select a member on the left and then select the new shape in the palette on the right. You can also click **Assign Palette** to quickly assign the shapes to the members of the field.

Select a different shape palette using the drop-down menu in the upper right.

Conclusion:
In this experiment, we implemented Filters on Dimensions and Measures for any dataset using tableau.

Experiment No. 05

Title: Apply K-means Clustering on iris dataset in tableau.

Theory:
Cluster analysis partitions marks in the view into clusters, where the marks within each cluster are more similar to one another than they are to marks in other clusters.

Create clusters
To find clusters in a view in Tableau, follow these steps.
1. Create a view.
2. Drag **Cluster** from the **Analytics** pane into the view, and drop it on in the target area in the view:

Clustering constraints
Clustering is available in Tableau Desktop, but is not available for authoring on the web (Tableau Server, Tableau Cloud). Clustering is also not available when any of the following conditions apply:
- When you are using a cube (multidimensional) data source.
- When there is a blended dimension in the view.
- When there are no fields that can be used as variables (inputs) for clustering in the view.
- When there are no dimensions present in an aggregated view.

When any of those conditions apply, you will not be able to drag **Clusters** from the Analytics pane to the view.
In addition, the following field types cannot be used as variables (inputs) for clustering:
- Table calculations
- Blended calculations
- Ad-hoc calculations
- Generated latitude/longitude values
- Groups
- Sets
- Bins
- Parameters
- Dates
- Measure Names/Measure Values

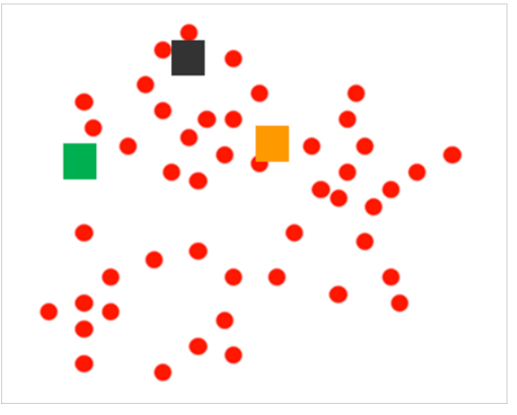
The clustering algorithm

Tableau uses the k-means algorithm for clustering. For a given number of clusters k, the algorithm partitions the data into k clusters. Each cluster has a center (centroid) that is the mean value of all the points in that cluster. K-means locates centers through an iterative procedure that minimizes distances between individual points in a cluster and the cluster center. In Tableau, you can specify a desired number of clusters, or have Tableau test different values of k and suggest an optimal number of clusters.

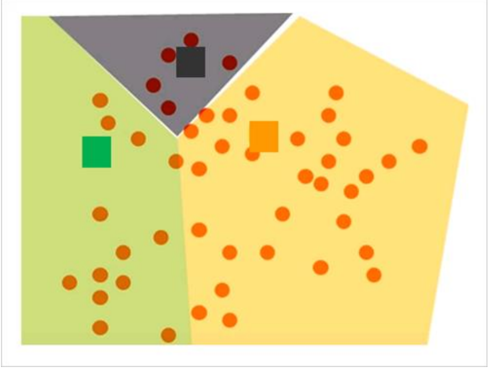
K-means requires an initial specification of cluster centers. Starting with one cluster, the method chooses a variable whose mean is used as a threshold for splitting the data in two. The centroids of these two parts are then used to initialize k-means to optimize the membership of the two clusters. Next, one of the two clusters is chosen for splitting and a variable within that cluster is chosen whose mean is used as a threshold for splitting that cluster in two. K-means is then used to partition the data into three clusters, initialized with the centroids of the two parts of the split cluster and the centroid of the remaining cluster. This process is repeated until a set number of clusters is reached.

Tableau uses Lloyd's algorithm with squared Euclidean distances to compute the k-means clustering for each k. Combined with the splitting procedure to determine the initial centers for each $k > 1$, the resulting clustering is deterministic, with the result dependent only on the number of clusters.

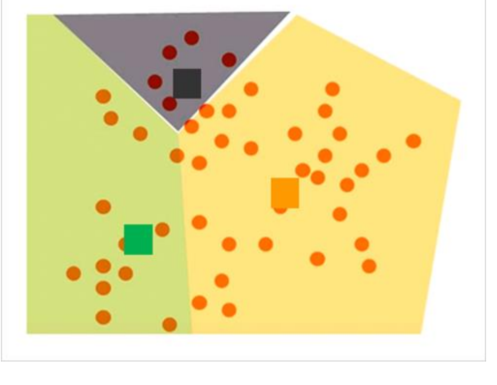
The algorithm starts by picking initial cluster centers:



It then partitions the marks by assigning each to its nearest center:



Then it refines the results by computing new centers for each partition by averaging all the points assigned to the same cluster:



It then reviews the assignment of marks to clusters and reassigns any marks that are now closer to a different center than before.

The clusters are redefined and marks are reassigned iteratively until no more changes are occurring.

Inputs for Clustering
Variables
Identifies the fields Tableau uses to compute clusters. These are the fields listed in the Variables box in the Clusters dialog box.

Level of Detail
Identifies the fields that are contributing to the view's level of detail—that is, the fields that determine the level of aggregation.

Scaling

Identifies the scaling method used for pre-processing. Normalized is currently the only scaling method Tableau uses. The formula for this method, also known as min-max normalization, is (x – min(x))/(max(x) - min(x)).

Number of Clusters: The number of individual clusters in the clustering.

Number of Points: The number of marks in the view.

Between-group sum of squares: A metric quantifying the separation between clusters as a sum of squared distances between each cluster’s center (average value), weighted by the number of data points assigned to the cluster, and the center of the data set. The larger the value, the better the separation between clusters.

Within-group sum of squares: A metric quantifying the cohesion of clusters as a sum of squared distances between the center of each cluster and the individual marks in the cluster. The smaller the value, the more cohesive the clusters.

Total sum of squares: Totals the between-group sum of squares and the within-group sum of squares. The ratio (between-group sum of squares)/(total sum of squares) gives the proportion of variance explained by the model. Values are between 0 and 1; larger values typically indicate a better model. However, you can increase this ratio just by increasing the number of clusters, so it could be misleading if you compare a five-cluster model with a three-cluster model using just this value.

Conclusion:

In this experiment, we implemented K-Clustering using tableau.

Experiment No. 06

Title: Simple MongoDB and its CRUD Operations

Theory:

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database.

Create Operations

For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.

MongoDB provides two different create operations that you can use to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

insertOne()

As the namesake, insertOne() allows you to insert one document into the collection. For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB.

```
db.RecordsDB.insertOne({     name: "Marsh",     age: "6 years",     species: "Dog",     ownerAddress: "
380 W. Fir Ave",     chipped: true })
```

If the create operation is successful, a new document is created. The function will return an object where "acknowledged" is "true" and "insertID" is the newly created "ObjectId."

```
> db.RecordsDB.insertOne({ ... name: "Marsh", ... age: "6 years", ... species: "Dog", ... ownerAddress:
"380 W. Fir Ave", ... chipped: true ... })
```

insertMany()

It's possible to insert multiple items at one time by calling the *insertMany()* method on the desired collection. In this case, we pass multiple items into our chosen collection (*RecordsDB*) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

```
db.RecordsDB.insertMany([{    name: "Marsh",    age: "6 years",    species: "Dog",    ownerAddress: "380 W. Fir Ave",    chipped: true},     {name: "Kitana",      age: "4 years",      species: "Cat", ownerAddress: "521 E. Cortland",       chipped: true}])
```

```
db.RecordsDB.insertMany([{ name: "Marsh", age: "6 years", species: "Dog", ownerAddress: "380 W. Fir Ave", chipped: true}, {name: "Kitana", age: "4 years",  species: "Cat", ownerAddress: "521 E. Cortland", chipped: true}])
```

Read Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want. The MongoDB documentation contains more information on the available query filters. Query modifiers may also be used to change how many results are returned.

MongoDB has two methods of reading documents from a collection:

- db.collection.find()
- db.collection.findOne()

find()

In order to get all the documents from a collection, we can simply use the *find()* method on our chosen collection. Executing just the *find()* method with no arguments will return all records currently in the collection.

```
db.RecordsDB.find()
```

Here we can see that every record has an assigned “ObjectId” mapped to the “_id” key.

If you want to get more specific with a read operation and find a desired subsection of the records, you can use the previously mentioned filtering criteria to choose what results should be returned. One of the most common ways of filtering the results is to search by value.

```
db.RecordsDB.find({"species":"Cat"})
```

findOne()

In order to get one document that satisfies the search criteria, we can simply use the *findOne()* method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. If no documents satisfy the search criteria, the function returns null. The function takes the following form of syntax.

```
db.{collection}.findOne({query}, {projection})
```

Update Operations

update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

You should be careful when updating documents, as updates are permanent and can’t be rolled back. This applies to delete operations as well.
For MongoDB CRUD, there are three different methods of updating documents:

- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

updateOne()
We can update a currently existing record and change a single document with an update operation. To do this, we use the *updateOne()* method on a chosen collection, which here is “RecordsDB.” To update a document, we provide the method with two arguments: an update filter and an update action.

The update filter defines which items we want to update, and the update action defines how to update those items. We first pass in the update filter. Then, we use the “$set” key and provide the fields we want to update as a value. This method will update the first record that matches the provided filter.

```
db.RecordsDB.updateOne({name: "Marsh"}, {$set:{ownerAddress: "451 W. Coffee St. A204"}})
```

updateMany()
updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.

```
db.RecordsDB.updateMany({species:"Dog"}, {$set: {age: "5"}})
```

replaceOne()

The *replaceOne()* method is used to replace a single document in the specified collection. *replaceOne()* replaces the entire document, meaning fields in the old document not contained in the new will be lost.

```
db.RecordsDB.replaceOne({name: "Kevin"}, {name: "Maki"})
```

Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.
MongoDB has two different methods of deleting records from a collection:

- db.collection.deleteOne()
- db.collection.deleteMany()

deleteOne()

deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.

```
db.RecordsDB.deleteOne({name:"Maki"})
```

deleteMany()

deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in *deleteOne()*.

```
db.RecordsDB.deleteMany({species:"Dog"})
```

Conclusion:

In this experiment we performed MongoDB CRUD operations.

Experiment No. 07

Title: Performing import, export and aggregation in MongoDB.

Theory:

Import and Export Data:

We can use MongoDB Compass to import and export data to and from collections. Compass supports import and export for both **JSON** and **CSV** files. To import or export data to or from a collection, navigate to the detailed collection view by either selecting the collection from the Databases tab or clicking the collection in the left-side navigation.

Import Data into a Collection

MongoDB Compass can import data into a collection from either a **JSON** or **CSV** file.

Limitations

- Importing data into a collection is not permitted in **MongoDB Compass Readonly Edition**.

Format Your Data

Before you can import your data into MongoDB Compass you must first ensure that it is formatted correctly.

JSON
CSV
When importing data from a **CSV** file, the first line of the file must be a comma-separated list of your document field names. Subsequent lines in the file must be comma-separated field values in the order corresponding with the field order in the first line.

EXAMPLE

The following .csv file imports three documents:

name,age,fav_color,pet

Jeff,25,green,Bongo

Alice,20,purple,Hazel

Tim,32,red,Lassie

Procedure

- To import your formatted data into a collection
- Connect to the deployment containing the collection you wish to import data into.
- Navigate to your target collection.

You can either select the collection from the Collections tab or click the collection in the left-hand pane.

- Click the ***Add Data*** dropdown and select ***Import File***.
- Compass displays the following dialog:
- Select the location of the source data file under ***Select File***.
- Choose the appropriate file type.
- Under **Select Input File Type**, select either **JSON** or **CSV**.

If you are importing a CSV file, you may specify fields to import and the types of those fields under **Specify Fields and Types**. The default data type for all fields is string.

To exclude a field from a CSV file you are importing, uncheck the checkbox next to that field name. To select a type for a field, use the dropdown menu below that field name.

Configure import options.

- Under **Options**, configure the import options for your use case.
- If you are importing a CSV file, you may select how your data is delimited.

For both JSON and CSV file imports, you can toggle **Ignore empty strings** and **Stop on errors**:

- If checked, **Ignore empty strings** drops fields with empty string values from your imported documents. The document is still imported with all other fields.
- If checked, **Stop on errors** prevents any data from being imported in the event of an error. If unchecked, data is inserted until an error is encountered and successful inserts are not rolled back. The import operation will not continue after encountering an error in either case.

Click ***Import***.

A progress bar displays the status of the import. If an error occurs during import, the progress bar turns red and an error message appears in the dialog. After successful import, the dialog closes and Compass displays the collection page containing the newly imported documents.

Export Data from a Collection

MongoDB Compass can export data from a collection as either a **JSON** or **CSV** file. If you specify a filter or aggregation pipeline for your collection, Compass only exports documents which match the specified query or pipeline results.

Procedure

Export Entire Collection
Export Filtered Subset of a Collection
Export Aggregation Results

To export an entire collection to a file:

- Connect to the deployment containing the collection you wish to export data from.
- Click ***Collection*** in the top-level menu and select ***Export Collection***.

The export dialog initially displays the query entered in the query bar prior to export, if applicable. If no query was specified, this section displays a `find` operation with no parameters, which returns all documents in the collection.

- To ignore the query filter and export your entire collection, select **Export Full Collection** and click **Select Fields**.
- Only fields that are checked are included in the exported file.
- You can add document fields to include with the **Add Field** button if the field you want to include is not automatically detected.
- Choose a file type and export location.
- Under **Select Export File Type**, select either **JSON** or **CSV**. If you select **JSON**, your data is exported to the target file as a comma-separated array.
- Then, under **Output**, choose where to export the file to.
- Click *Export*.

A progress bar displays the status of the export. If an error occurs during export, the progress bar turns red and an error message appears in the dialog. After successful export, the dialog closes.

Conclusion:

In this experiment, we performed import and export data using MondoDB Compass.

Experiment No. 08

Title: MapReduce programming

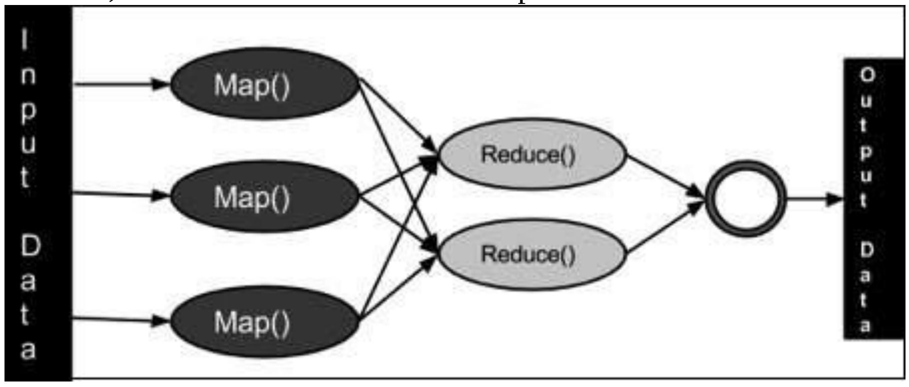
Theory:

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** − The map or mapper’s job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer’s job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Code Implementation:

1. SalesCountryReducer Class Definition-

```
public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable> {
```

Here, the first two data types, **'Text'** and **'IntWritable'** are data type of input key-value to the reducer.

Output of mapper is in the form of <CountryName1, 1>, <CountryName2, 1>. This output of mapper becomes input to the reducer. So, to align with its data type, **Text** and **IntWritable** are used as data type here.

The last two data types, 'Text' and 'IntWritable' are data type of output generated by reducer in the form of key-value pair.

Every reducer class must be extended from **MapReduceBase** class and it must implement **Reducer** interface.

2. Defining 'reduce' function-

```
public void reduce( Text t_key,
            Iterator<IntWritable> values,
            OutputCollector<Text,IntWritable> output,
            Reporter reporter) throws IOException {
```

An input to the **reduce()** method is a key with a list of multiple values.

This is given to reducer as **<United Arab Emirates, {1,1,1,1,1,1}>**

So, to accept arguments of this form, first two data types are used, viz., **Text** and **Iterator<IntWritable>**. **Text** is a data type of key and **Iterator<IntWritable>** is a data type for list of values for that key.

The next argument is of type **OutputCollector<Text,IntWritable>** which collects the output of reducer phase.

reduce() method begins by copying key value and initializing frequency count to 0.

```
Text key = t_key;
int frequencyForCountry = 0;
```

Then, using '**while'** loop, we iterate through the list of values associated with the key and calculate the final frequency by summing up all the values.

```
 while (values.hasNext()) {
           // replace type of value with the actual type of our value
           IntWritable value = (IntWritable) values.next();
           frequencyForCountry += value.get();

       }
```

Now, we push the result to the output collector in the form of **key** and obtained **frequency count**. Below code does this-

```
output.collect(key, new IntWritable(frequencyForCountry));
```

Explanation of SalesCountryDriver Class

In this section, we will understand the implementation of **SalesCountryDriver** class

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryDriver.class** will go into directory named by this package name: **SalesCountry**.

2. Define a driver class which will create a new client job, configuration object and advertise Mapper and Reducer classes.

The driver class is responsible for setting our MapReduce job to run in Hadoop. In this class, we specify **job name, data type of input/output and names of mapper and reducer classes**.

3. We set input and output directories which are used to consume input dataset and produce output, respectively.

arg[0] and **arg[1]** are the command-line arguments passed with a command given in MapReduce hands-on, i.e.,

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales

4. Trigger our job

Below code start execution of MapReduce job-

```
try {
    // Run the job
    JobClient.runJob(job_conf);
} catch (Exception e) {
    e.printStackTrace();
}
```

Conclusion:

In this experiment, we implemented MapReduce Programming using Hadoop and Java.

Experiment No. 09

Title: Partitioning and processing using Hive.

Theory:

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

The partitioning in Hive can be executed in two ways –

Static Partitioning

In static or manual partitioning, it is required to pass the values of partitioned columns manually while loading the data into the table. Hence, the data file doesn't contain the partitioned columns.

Example of Static Partitioning

- First, select the database in which we want to create a table.

1. hive> use test;

- Create the table and provide the partitioned columns by using the following command: -

1. hive> create table student (id int, name string, age int, institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';

- Let's retrieve the information associated with the table.

1. hive> describe student;

Load the data into the table and pass the values of partition columns with it by using the following command: -

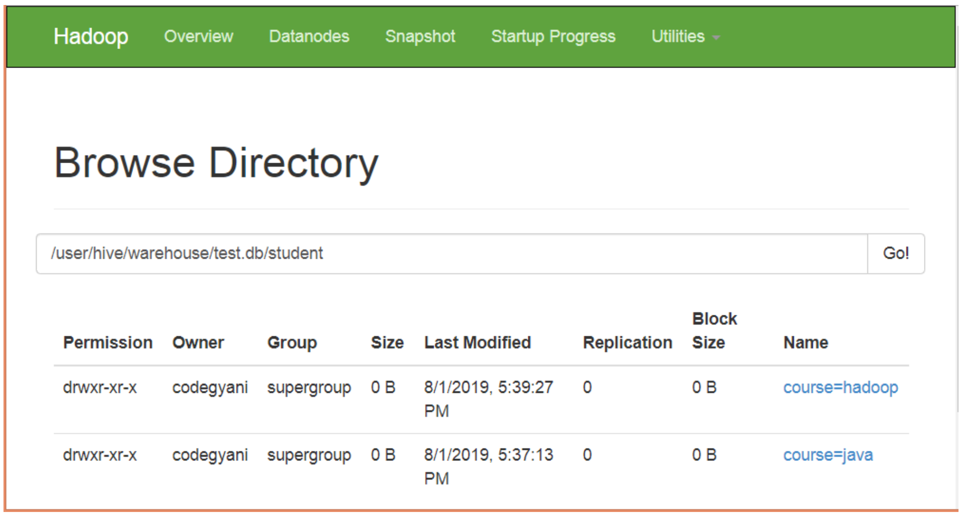
1. hive> load data local inpath '/home/codegyani/hive/student_details1' into table student
2. partition(course= "java");

Here, we are partitioning the students of an institute based on courses.

- Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

1. hive> load data local inpath '/home/codegyani/hive/student_details2' into table student
2. partition(course= "hadoop");

In the following screenshot, we can see that the table student is divided into two categories.



Browse Directory

/user/hive/warehouse/test.db/student Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 5:39:27 PM	0	0 B	course=hadoop
drwxr-xr-x	codegyani	supergroup	0 B	8/1/2019, 5:37:13 PM	0	0 B	course=java

- Let's retrieve the entire data of the able by using the following command: -

```
1. hive> select * from student;
```

Now, try to retrieve the data based on partitioned columns by using the following command: -

```
1. hive> select * from student where course="java";
```

In this case, we are not examining the entire data. Hence, this approach improves query response time.

- Let's also retrieve the data of another partitioned dataset by using the following command: -

```
1. hive> select * from student where course= "hadoop";
```

Dynamic Partitioning

In dynamic partitioning, the values of partitioned columns exist within the table. So, it is not required to pass the values of partitioned columns manually.

- First, select the database in which we want to create a table.

```
1. hive> use show;
```

- Enable the dynamic partition by using the following commands: -

```
1. hive> set hive.exec.dynamic.partition=true;
2. hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

- Create a dummy table to store the data.

```
1. hive> create table stud_demo(id int, name string, age int, institute string, course string)
2. row format delimited
3. fields terminated by ',';
```

- Now, load the data into the table.

```
1. hive> load data local inpath '/home/codegyani/hive/student_details' into table stud_demo;
```

- Create a partition table by using the following command: -

```
1. hive> create table student_part (id int, name string, age int, institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';
```

- Now, insert the data of dummy table into the partition table.

```
1. hive> insert into student_part
2. partition(course)
3. select id, name, age, institute, course
4. from stud_demo;
```

- Let's retrieve the entire data of the table by using the following command: -

```
1. hive> select * from student_part;
```

- Now, try to retrieve the data based on partitioned columns by using the following command: -

```
1. hive> select * from student_part where course= "java ";
```

In this case, we are not examining the entire data. Hence, this approach improves query response time.

- Let's also retrieve the data of another partitioned dataset by using the following command: -

```
1. hive> select * from student_part where course= "hadoop";
```

Conclusion:

In this experiment, we implemented partitioning and processing using Hive.

Experiment No. 10

Title: Perform group by, order by, sort by, cluster by, distribute by queries using Hive.

Theory:

Hive provides SQL type querying language for the ETL purpose on top of Hadoop file system.

Hive Query language (HiveQL) provides SQL type environment in Hive to work with tables, databases, queries.

We can have a different type of Clauses associated with Hive to perform different type data manipulations and querying. For better connectivity with different nodes outside the environment. HIVE provide JDBC connectivity as well.

Hive queries provides the following features:

- Data modeling such as Creation of databases, tables, etc.
- ETL functionalities such as Extraction, Transformation, and Loading data into tables
- Joins to merge different data tables
- User specific custom scripts for ease of code
- Faster querying tool on top of Hadoop

Creating Table in Hive

Here in this experiment, we are going to create table "employees_guru" with 6 columns.

create table employees_guru(Id INT, Name STRING, Age INT, Address STRING, Salary FLOAT, Department STRING);

load data local inpath '/home/hduser/Employees.txt' into TABLE employees_guru;

From the output of the query, we observe following:

We are creating table "employees_guru" with 6 column values such as Id, Name, Age, Address, Salary, Department, which belongs to the employees present in organization "guru."

1. Here in this step we are loading data into employees_guru table. The data that we are going to load will be placed under Employees.txt file

Order by query:

The ORDER BY syntax in HiveQL is similar to the syntax of ORDER BY in SQL language.

Order by is the clause we use with "SELECT" statement in Hive queries, which helps sort data. Order by clause use columns on Hive tables for sorting particular column values mentioned with Order by. For whatever the column name we are defining the order by clause the query will selects and display results by ascending or descending order the particular column values.

If the mentioned order by field is a string, then it will display the result in lexicographical order. At the back end, it has to be passed on to a single reducer.

From the output of the query, we observe following:

1. It is the query that performing on the “employees_guru” table with the ORDER BY clause with Department as defined ORDER BY column name.”Department” is String so it will display results based on lexicographical order.
2. This is actual output for the query. If we observe it properly, we can see that it get results displayed based on Department column such as ADMIN, Finance and so on in orderQuery to be perform.

Query :

SELECT * FROM employees_guru ORDER BY Department;

Group by query:

Group by clause use columns on Hive tables for grouping particular column values mentioned with the group by. For whatever the column name we are defining a “groupby” clause the query will selects and display results by grouping the particular column values.

From the output of the query, we observe following:

1. It is the query that is performed on the “employees_guru” table with the GROUP BY clause with Department as defined GROUP BY column name.
2. The output showing here is the department name, and the employees count in different departments. Here all the employees belong to the specific department is grouped by and displayed in the results. So the result is department name with the total number of employees present in each department.

Query:

SELECT Department, count(*) FROM employees_guru GROUP BY Department;

Sort by:

Sort by clause performs on column names of Hive tables to sort the output. We can mention DESC for sorting the order in descending order and mention ASC for Ascending order of the sort.

In this sort by it will sort the rows before feeding to the reducer. Always sort by depends on column types.

For instance, if column types are numeric it will sort in numeric order if the columns types are string it will sort in lexicographical order.

From the output of the query, we observe following:

1. It is the query that performing on the table “employees_guru” with the SORT BY clause with “id” as define SORT BY column name. We used keyword DESC.
2. So the output displayed will be in descending order of “id”.

Query:

SELECT * from employees_guru SORT BY Id DESC;

Cluster By:

Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.

Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

- It ensures sorting orders of values present in multiple reducers

For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end. But as front end it is an alternative clause for both Sort By and Distribute By.

This is actually back end process when we perform a query with sort by, group by, and cluster by in terms of Map reduce framework. So if we want to store results into multiple reducers, we go with Cluster By.

From the output of the query, we observe following:

1. It is the query that performs CLUSTER BY clause on Id field value. Here it's going to get a sort on Id values.
2. It displays the Id and Names present in the guru_employees sort ordered by

Query:

```
SELECT  Id, Name from employees_guru CLUSTER BY Id;
```

Distribute By:

Distribute BY clause used on tables present in Hive. Hive uses the columns in Distribute by to distribute the rows among reducers. All Distribute BY columns will go to the same reducer.

- It ensures each of N reducers gets non-overlapping ranges of column
- It doesn't sort the output of each reducer

From the output of the query, we observe following:

1. DISTRIBUTE BY Clause performing on Id of "empoloyees_guru" table
2. Output showing Id, Name. At back end, it will go to the same reducer

Query:

```
SELECT  Id, Name from employees_guru DISTRIBUTE BY Id;
```

Conclusion:

In this experiment, we implemented Hive queries.