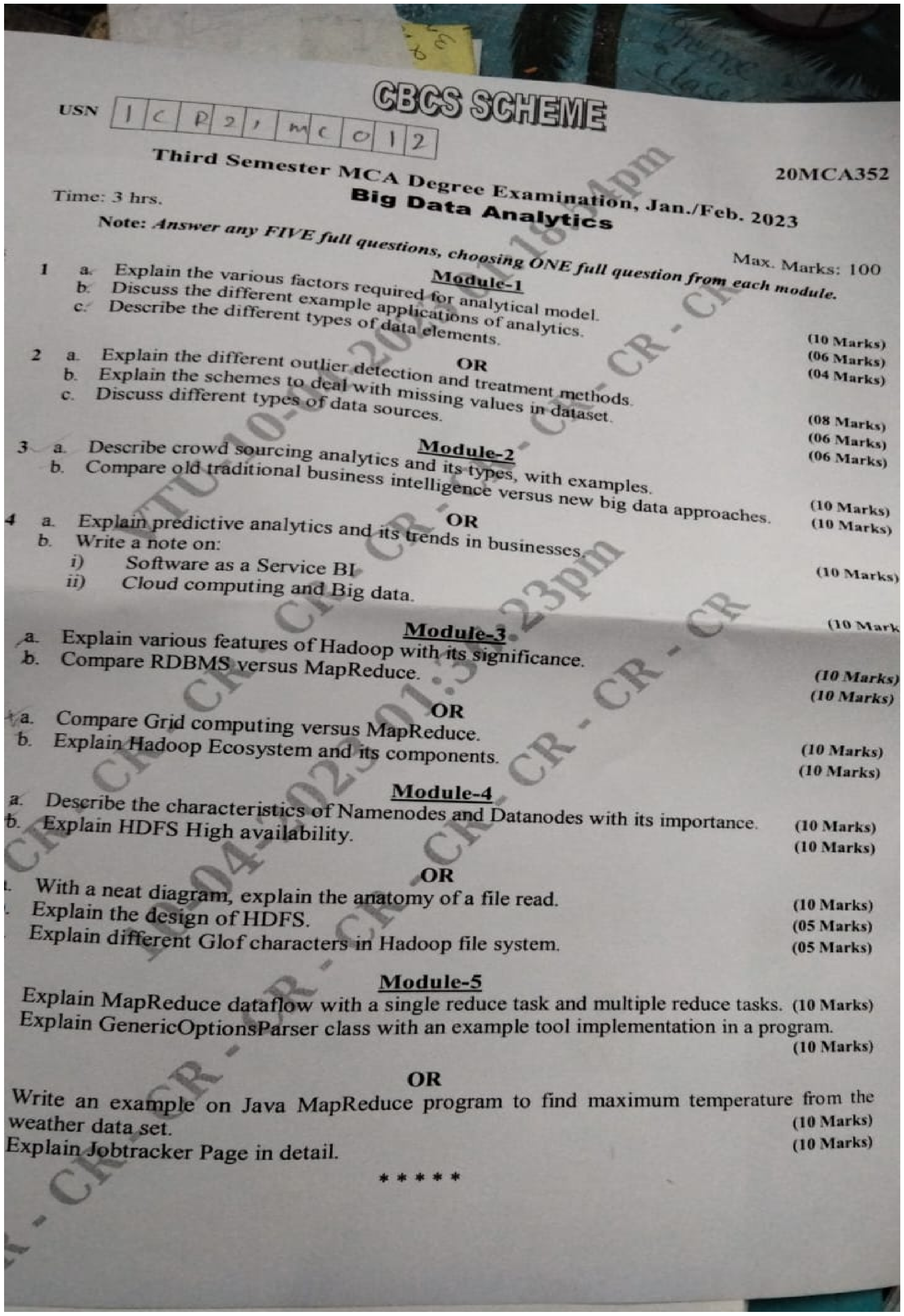


USN 1 C R 2 1 M C 0 1 2

CBCS SCHEME

20MCA352

Third Semester MCA Degree Examination, Jan./Feb. 2023

Big Data Analytics

Time: 3 hrs. Max. Marks: 100

Note: *Answer any FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. Explain the various factors required for analytical model. (10 Marks)
b. Discuss the different example applications of analytics. (06 Marks)
c. Describe the different types of data elements. (04 Marks)

OR

2 a. Explain the different outlier detection and treatment methods. (08 Marks)
b. Explain the schemes to deal with missing values in dataset. (06 Marks)
c. Discuss different types of data sources. (06 Marks)

Module-2

3 a. Describe crowd sourcing analytics and its types, with examples. (10 Marks)
b. Compare old traditional business intelligence versus new big data approaches. (10 Marks)

OR

4 a. Explain predictive analytics and its trends in businesses. (10 Marks)
b. Write a note on:
i) Software as a Service BI
ii) Cloud computing and Big data. (10 Marks)

Module-3

a. Explain various features of Hadoop with its significance. (10 Marks)
b. Compare RDBMS versus MapReduce. (10 Marks)

OR

a. Compare Grid computing versus MapReduce. (10 Marks)
b. Explain Hadoop Ecosystem and its components. (10 Marks)

Module-4

a. Describe the characteristics of Namenodes and Datanodes with its importance. (10 Marks)
b. Explain HDFS High availability. (10 Marks)

OR

. With a neat diagram, explain the anatomy of a file read. (10 Marks)
. Explain the design of HDFS. (05 Marks)
Explain different Glof characters in Hadoop file system. (05 Marks)

Module-5

Explain MapReduce dataflow with a single reduce task and multiple reduce tasks. (10 Marks)
Explain GenericOptionsParser class with an example tool implementation in a program. (10 Marks)

OR

Write an example on Java MapReduce program to find maximum temperature from the weather data set. (10 Marks)
Explain Jobtracker Page in detail. (10 Marks)

* * * * *

Odd Sem 2022 – VTU PAPER SOLUTIONS

1a. A good analytical model should satisfy several requirements, depending on the application area.

• A first critical success factor is business relevance. The analytical model should actually solve the business problem for which it was developed.

• A second criterion is statistical performance. The model should have statistical significance and predictive power. Example: in a classification setting (churn or fraud) the model should have good discriminative power.

• Depending on the application, the model should also be interpretable and justifiable. Interoperability refers to understanding the pattern and justifiability refers to the degree to which model corresponds to prior business knowledge and institution

• Analytical models should also be operationally efficient. This refers to the efforts needed to collect the data, preprocess it, evaluate the model, and feed its outputs to the business application. Example: campaign management, capital calculation etc...

• Another key attention point is the economic cost needed to set up the analytical model. This includes the cost to gather and process the data, cost to analyze the data and cost to put the resulting analytical models into production.

• Finally, analytical models should also comply with both local and international regulation and legislation.

Example: Credit risk setting.

1b. Analytics is everywhere and strongly embedded in our daily lives.

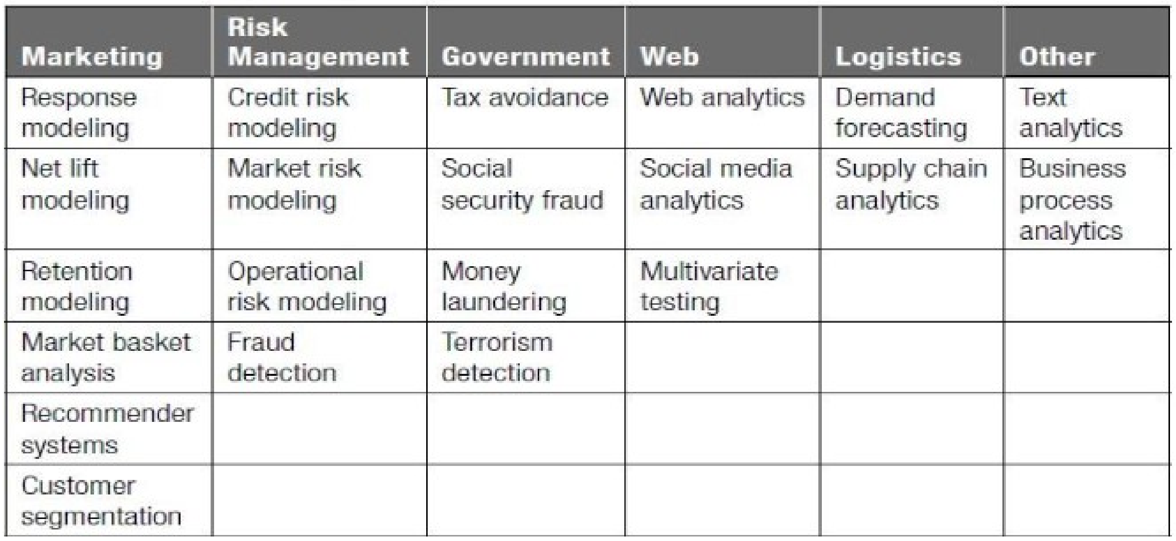
The relevance, importance and impact of analytics are now bigger than ever before and, given that more and more data are being collected and that there is strategic value in knowing what is hidden in data, analytics will continue to grow.

Physical mail box: a catalogue sent to us through mail most probably as a result of a response modeling analytical exercise that indicated, given my characteristics and previous purchase behavior, we are likely to buy one or more products from it.

Behavioral Scoring Model: Checking account balance of the customer from the past 12 months and credit payments during that period, together with other kinds of information available to the bank, to predict whether a customer will default on the loan during the next year.

Social Media: As we logged on to my Facebook page, the social ads appearing there were based on analyzing all information (posts, pictures, my friends and their behavior, etc.) available to Facebook. Twitter posts will be analyzed (possibly in real time) by social media analytics to understand both the subject tweets and the sentiment of them.

Table 1.1 presents some examples of how analytics is applied in various settings.



Marketing	Risk Management	Government	Web	Logistics	Other
Response modeling	Credit risk modeling	Tax avoidance	Web analytics	Demand forecasting	Text analytics
Net lift modeling	Market risk modeling	Social security fraud	Social media analytics	Supply chain analytics	Business process analytics
Retention modeling	Operational risk modeling	Money laundering	Multivariate testing		
Market basket analysis	Fraud detection	Terrorism detection			
Recommender systems					
Customer segmentation					

1c. t is important to appropriately consider the different types of data elements at the start of the analysis. The different types of data elements can be considered:

Continuous: These are data elements that are defined on an interval that can be limited or unlimited. Examples include income, sales, RFM (recency, frequency, monetary).

Categorical: The categorical data elements are differentiated as follows:

Nominal: These are data elements that can only take on a limited let of values with no meaningful
ordering in between. Examples: marital status, profession, purpose of loan.

Ordinal: These are data elements that can only take on a limited set of values with a meaningful
ordering in between. Examples: credit rating; age coded as young, middle aged, and old.

Binary: These are data elements that can only take on two values. Example: gender, employment
status.

Appropriately distinguishing between these different data elements is of key importance to start the analysis when importing the data into an analytics tool. For example, if marital status were to be incorrectly specified as a continuous data element, then the software would calculate its mean, standard deviation, and so on, this is obviously meaningless.

2a. Outliers are extreme observations that are very dissimilar to the rest of the population. Actually, two types of outliers are:

- Valid observations (e.g salary of boss is $1 million)
- Invalid observations (e.g age is 300 years)

Both are univariate outliers in the sense that they are outlying on one dimension, outliers can be hidden in unidimensional views of the data.

Multivariate outliers are observations that are outlying in multiple dimensions. Two important steps in dealing with outliers are detection and treatment

A first check for outlier is to calculate the minimum and maximum values for each data element

Next various graphical tools can be used to detect outliers.

Example:

- Histograms
- Box plots
- Z-scores

Figure 1.4 presents an example of a distribution for age whereby the circled areas clearly represents outlier.

Figure 1.4 Histogram representations of outliers

Another visual mechanism is **box plots**. A box plot represents three key qualifiers of data. They are:

- First Quartile (25% of the observations have a lower value)
- Median (50% of the observations have a lower value)
- Third Quartile (75% of the observations are lower value)

All three quartiles are represented as a box. The minimum and maximum values are then also added unless they are too far from the edges of the box. Too far away is then 1.5*Interquartile Range(IQR=Q3- Q1). The figure 1.5 gives an example of a box plot in which the three outliers can be seen.

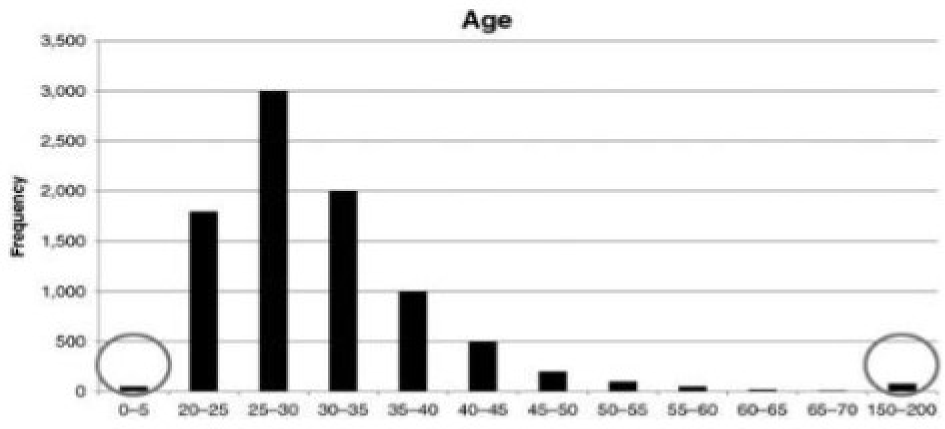


Figure 1.4 Histogram representations of outliers

Another visual mechanism is **box plots**. A box plot represents three key qualifiers of data. They are:

- First Quartile (25% of the observations have a lower value)
- Median (50% of the observations have a lower value)
- Third Quartile (75% of the observations are lower value)

All three quartiles are represented as a box. The minimum and maximum values are then also added unless they are too far from the edges of the box. Too far away is then 1.5*Interquartile Range(IQR=Q3- Q1). The figure 1.5 gives an example of a box plot in which the three outliers can be seen.

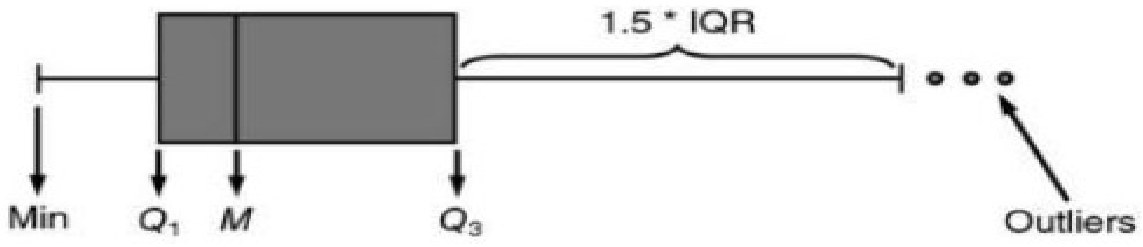
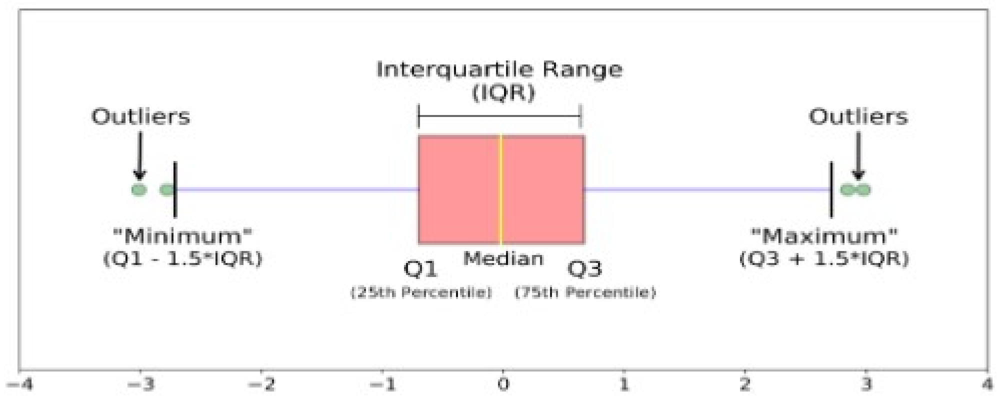


Figure 1.5: Box Plots for Outlier Detection

Box-Plot:

A box and whisker plot—also called a box plot—displays the five-number summary of a set of data. The five-number summary is the minimum, first quartile, median, third quartile, and maximum.

In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.



Z-Scores: Another way to identify outliers is to calculate **z-scores**, measuring how many standard deviations an observation lies away from the mean as follows:

Where, μ-represents the average of the

variables σ- represents standard deviation

an example is given in table 1.3. Note that by definition, the z-scores will have 0-mean and unit standard deviation.

A practical rule of thumb then defines outliers when the absolute value of the z-score |z| is bigger than
3. Note that the z-scores relies on the normal distribution.

Multivariate outliers can be detected by fitting regression lies and inspecting the observation with large errors. (e.g residual plot). Alternative methods are *clustering*. Some analytical techniques like *decision trees*, *neural networks* are fairly robust with respect to outliers. Various schemes exist to deal with outliers. It highly depends on whether the outlier represents a *valid* or invalid *observation*.

A popular scheme for is truncation/capping/winsorising. One here by imposes lower limit and upper limit on a variable and any values below/above are brought back to the limits. The limits can be calculated using the z-scores or the IQR (which is more robust than the z-scores)

Calculating upper and lower limit using z-score is:

Upper limit=μ+3σ

Lower limit= μ-3σ

This calculation is shown in the figure 1.6.

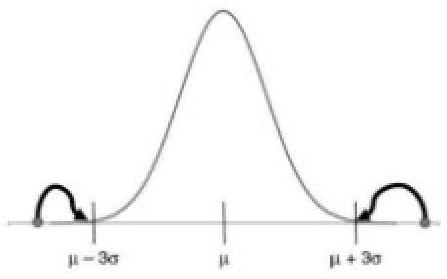


Figure 1.6: Using the z-scores for Truncation

Another way of calculating the upper and lower limit is using box plot IQR.

Upper limit=M+3s

Lower limit=M-3s

Where, M is Median and s=IQR/(2*0.6745)[3]

2b. Missing values can occur because of various reasons.

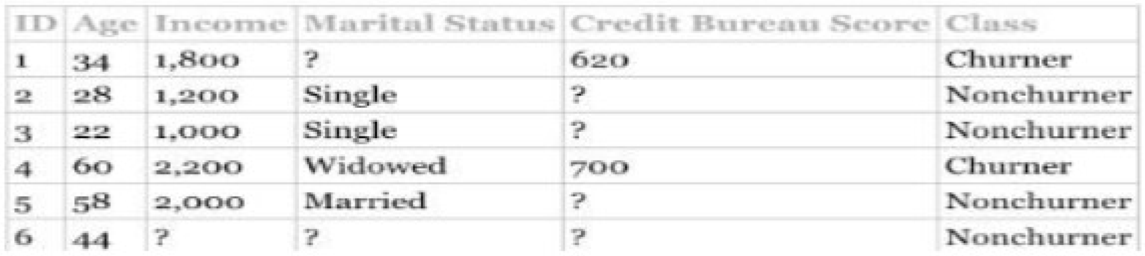
- Information can be non-applicable
- Information can also be undisclosed

Missing data can also originate an error during merging. Some analytical techniques (e.g., decision trees) can directly deal with missing values. They are:

- Replace (impute)
- Delete
- Keep

Replace: This implies replacing the missing value with a known value. Consider the table 1.2 one could calculate the missing credit bureau scores with the average or median of the known values. For marital status the mode can then be used.

Table 1.2: Dealing with missing values



ID	Age	Income	Marital Status	Credit Bureau Score	Class
1	34	1,800	?	620	Churner
2	28	1,200	Single	?	Nonchurner
3	22	1,000	Single	?	Nonchurner
4	60	2,200	Widowed	700	Churner
5	58	2,000	Married	?	Nonchurner
6	44	?	?	?	Nonchurner

Delete: This is the most straight forward option and consists of deleting observations or variables with lots of missing values. This, of course assumes that information is missing at random and has not meaningful interpretation and/or relationship to the target.

Keep: Missing values can be meaningful *(e.g a customer didn't disclose his/her income because if he/she is currently unemployed)*. Obviously, this is clearly related to the target *(e.g good/bad risk or churn)* and needs to be considered as a separate category.

2c. Data can originate from a variety of different sources. They are as follows:

- Transactional data
- Unstructured data
- Qualitative/Export based data
- Data poolers
- Publicly available data

Transactional Data: Transactions are the first important source of data. Transactional data consist of structured, low level, detailed information capturing the key characteristics of a customer transaction (e.g., purchase, claim, cash transfer, credit card payment). This type of data is usually stored in massive online transaction processing (OLTP) relational databases. It can also be summarized over longer time horizons by aggregating it into averages, absolute/relative trends, maximum/minimum values, and so on.

Unstructured data: Embedded in text documents (e.g., emails, web pages, claim forms) or multimedia content can also be interesting to analyze. However, these sources typically require extensive pre-processing before they can be successfully included in an analytical exercise.

Qualitative/Expert based data: Another important source of data is qualitative, expert based data. An expert is a person with a substantial amount of subject matter expertise within a particular setting (e.g., credit portfolio manager, brand manager). The expertise stems from both common sense and business experience, and it is important to elicit expertise as much as possible before the analytics is run. This will steer the modeling in the right direction and allow you to interpret the analytical results from the right perspective. A popular example of applying expert based validation is checking the univariate signs of a regression model. For example, one would expect a priori that higher debt has an adverse.

Data poolers: Nowadays, data poolers are becoming more and more important in the industry. Popular examples are Dun & Bradstreet, Bureau Van Dijck, and Thomson Reuters. The core business of these companies is to gather data in a particular setting (e.g., credit risk, marketing), build models with it, and sell the output of these models (e.g., scores), possibly together with the underlying raw data, to interested customers. A popular example of this in the United States is the FICO score, which is a credit score ranging between 300 and 850 that is provided by the three most important credit bureaus: Experian, Equifax, and TransUnion. Many financial institutions use these FICO scores either as their final internal model or as a benchmark against an internally developed credit scorecard.

Publicly available data: Finally, plenty of publicly available data can be included in the analytical exercise. A first important example is macroeconomic data about gross domestic product (GDP), inflation, unemployment, and so on. By including this type of data in an analytical model, it will become possible to see how the model varies with the state of the economy. This is especially
relevant in a credit risk setting, where typically all models need to be thoroughly stress tested. In addition, social media data from Facebook, Twitter, and others can be an important source of information. However, one needs to be careful here and make sure that all data gathering respects both local and international privacy regulations.

3a. **Crowdsourcing:** Crowdsourcing is a great way to capitalize on the resources that can build algorithms and predictive models

Kaggle: Kaggle describes itself as "an innovative solution for statistical/analytics outsourcing." That's a very formal way of saying that Kaggle manages competitions among the world's best data scientists. Here's how it works: Corporations, governments, and research laboratories are confronted with complex statistical challenges. They describe the problems to Kaggle and provide data sets. Kaggle converts the problems and the data into contests that are posted on its web site. The contests feature cash prizes ranging in value from $100 to $3 million. Kaggle's clients range in size from tiny start-ups to multinational corporations such as Ford Motor Company and government agencies such as NASA.

As per Anthony Goldbloom, Kaggle's founder and CEO: The idea is that someone comes to us with a problem, we put it up on our website, and then people from all over the world can compete to see who can produce the best solution."

Kaggle's approach is that it is truly a win-win scenario—contestants get access to real-world data (that has been carefully “anonymized” to eliminate privacy concerns) and prize sponsors reap the benefits of the contestants’ creativity.

Crowdsourcing is a disruptive business model whose roots are in technology but is extending beyond technology to other areas.

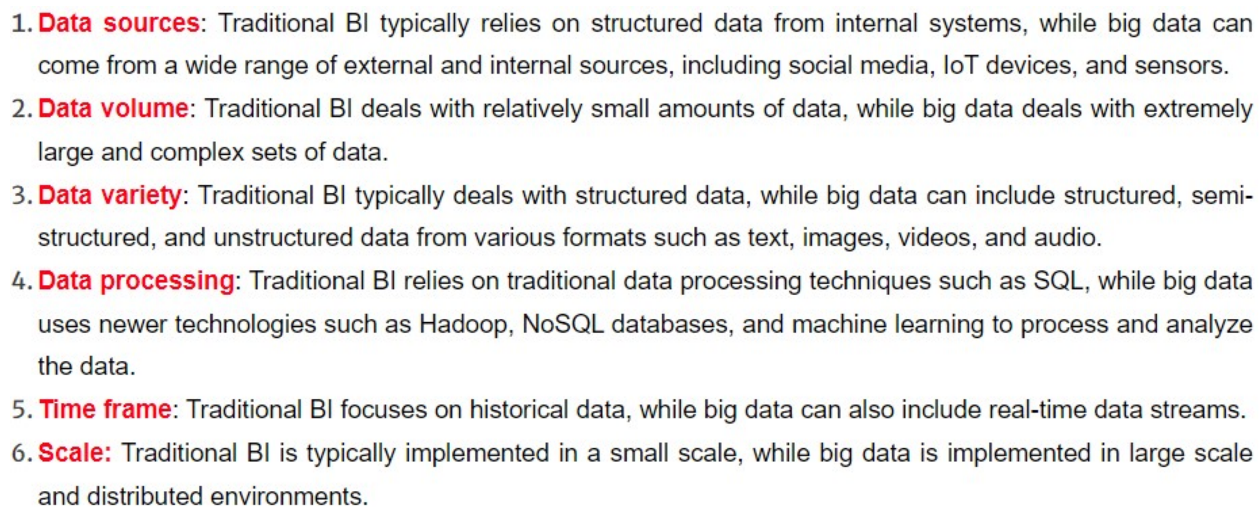
There are various types of crowd sourcing, such as crowd voting, crowd purchasing, wisdom of crowds, crowd funding, and contests.

Take for example:

- 99designs.com/, which does crowdsourcing of graphic design
- agentanything.com/, which posts “missions” where agents vie for to run errands
- 33needs.com/, which allows people to contribute to charitable programs that make a social impact

3b.

Differences between traditional business intelligence and Big Data are



1. **Data sources**: Traditional BI typically relies on structured data from internal systems, while big data can come from a wide range of external and internal sources, including social media, IoT devices, and sensors.
2. **Data volume**: Traditional BI deals with relatively small amounts of data, while big data deals with extremely large and complex sets of data.
3. **Data variety**: Traditional BI typically deals with structured data, while big data can include structured, semi-structured, and unstructured data from various formats such as text, images, videos, and audio.
4. **Data processing**: Traditional BI relies on traditional data processing techniques such as SQL, while big data uses newer technologies such as Hadoop, NoSQL databases, and machine learning to process and analyze the data.
5. **Time frame**: Traditional BI focuses on historical data, while big data can also include real-time data streams.
6. **Scale:** Traditional BI is typically implemented in a small scale, while big data is implemented in large scale and distributed environments.

4a. To master analytics, enterprises will move from being in reactive positions (business intelligence) to forward leaning positions (predictive analytics). Using all the data available-traditional internal data sources combined with new rich external data sources- will make the predictions more accurate and meaningful. Because analytics are contextual, enterprises can build confidence in the analytics and the trust will result in using analytics insight to trigger business events.

Leading trends that are making their way to the forefront of businesses today:

- Recommendation engines similar to those used in Netflix and Amazon that use past purchases and buying behavior to recommend new purchases.
- Risk engines for a wide variety of business areas, including market and credit risk, catastrophic risk, and portfolio risk.

- Innovation engines for new product innovation, drug discovery, and consumer and fashion trends to predict potential new product formulations and discoveries.
- Customer insight engines that integrate a wide variety of customer- related info, including sentiment, behavior, and even emotions. Customer insight engines will be the backbone in online and set-top box advertisement targeting, customer loyalty programs to maximize customer lifetime value, optimizing marketing campaigns for revenue lift, and targeting individuals or companies at the right time to maximize their spend.
- Optimization engines that optimize complex interrelated operations and decisions that are too overwhelming for people to systematically handle at scales, such as when, where, and how to seek natural resources to maximize output while reducing operational costs or what potential competitive strategies should be used in a global business that takes into account the various political, economic, and competitive pressures along with both internal and external operational capabilities.

4b. i. The software industry has seen some successful companies excel in the game of software as a service (SaaS) industry, such as salesforce.com. The basic principal is to make it easy for companies to gain access to solutions without the headache of building and maintaining their own onsite implementation. When you add up the costs of the people and technology, SaaS is far less expensive too. The solutions are typically sold by vendors on a subscription or pay-as-you-go basis instead of the more traditional software licensing model with annual maintenance fees.

In addition to the Omniture people, several other reasons stand out to me. They include:

- **Scaling the SaaS delivery model**. We built Omniture from the ground up to be SaaS and we understood the math better than the competition. We invented a concept called the Magic Number. The Magic Number helps you look at your SaaS business and helps you understand the value you are creating when standard GAAP accounting numbers would lead you to believe the opposite.
- **Killer sales organization**. Once we had a few well-known customers like HP, eBay, and Gannett, we stepped on the pedal from a competitive standpoint and really went to battle against the other sales organizations and we won. We focused the whole company on sales.
- **A focus on customer success.** We had 98 percent retention rate. Customer happiness and success were always first because in a SaaS business, unlike traditional enterprise software, it's too easy for customers to leave if they are not happy. James explained the three market reasons why he started Domo, knowing we had to fix three problems in traditional BI.

Here is a summary in his own words:

1. **Relieving the IT choke point.** Removing the friction for BI to become useful and enabling IT to be more strategic by enabling self-service BI.
2. **Transforming BI from cost center to a revenue generator.** Addresses a very common frustration that I've experienced as a CEO and that other

CEOs have shared with me . . . now that we've invested in capturing all this data—how do we benefit from it?

3. **The user experience.** Is where we are putting all our marbles. Today's BI is not designed for the end user. It's not intuitive, it's not accessible, it's not real time, and it doesn't meet the expectations of today's consumers of technology, who expect a much more connected experience than enterprise software delivers. We'll deliver an experience with BI that redefines BI and is unlike anything seen to date.

4bii. It is important to remember that for all kinds of reasons—technical, political, social, regulatory, and cultural—cloud computing has not been a successful business model that has been widely adopted for enterprises to store their Big Data assets. However, there are many who believe that some obvious industry verticals will soon realize that there is a huge ROI opportunity if they do embrace the cloud.

As per by Avinash Koushik, *Google's digital marketing evangelist*

"There will be Big Data platforms that companies will build, especially for the core operational system of the world. Where we continue to have an explosive amount of data come in and because the data is so proprietary that building out an infrastructure in-house seems logical. I actually think it's going to go the cloud; it's just a matter of time! It's not value add enough add enough to collect, process and store data".

Market economics are demanding that capital-intensive infrastructure costs disappear and business challenges are forcing clients to consider newer models. At the crossroads of high capital costs and rapidly changing business needs is a sea change that is driving the need for a new, compelling value proposition that is being manifested in a cloud-deployment model.

With a cloud model, you pay on a subscription basis with no upfront capi- tal expense. You don't incur the typical 30 percent maintenance fees—and all the updates on the platform are automatically available. The traditional cost of value chains is being completely disintermediated by platforms—massively scalable platforms where the marginal cost to deliver an incremental product or service is zero.

The ability to build massively scalable platforms—platforms where you have the option to keep adding new products and services for zero additional cost—is giving rise to business models that weren't possible before. Mehta calls it "the next industrial revolution, where the raw material is data and data factories replace manufacturing factories." He pointed out a few guiding principles that his firm stands by:

1. **Stop saying "cloud."** It's not about the fact that it is virtual, but the true value lies in delivering software, data, and/or analytics in an "as a service" model. Whether that is in a private hosted model or a publicly shared one does not matter. The delivery, pricing, and consumption model matters.
2. **Acknowledge the business issues.** There is no point to make light of matters around information privacy, security, access, and delivery. These issues are real, more often than not heavily regulated by multiple government agencies, and unless dealt with in a solution, will kill any platform sell.

3. **Fix some core technical gaps**. Everything from the ability to run analytics at scale in a virtual environment to ensuring information processing and analytics authenticity are issues that need solutions and have to be fixed.

5a. Features of Hadoop

1. Open Source:
Hadoop is open-source, which means it is free to use. Since it is an open-source project the source-code is available online for anyone to understand it or make some modifications as per their industry requirement.

2. Highly Scalable Cluster:
Hadoop is a highly scalable model. A large amount of data is divided into multiple inexpensive machines in a cluster which is processed parallelly. the number of these machines or nodes can be increased or decreased as per the enterprise's requirements. In traditional RDBMS (Relational DataBase Management System) the systems cannot be scaled to approach large amounts of data.
3. Fault Tolerance is Available:
Hadoop uses commodity hardware(inexpensive systems) which can be crashed at any moment. In Hadoop data is replicated on various DataNodes in a Hadoop cluster which ensures the availability of data if somehow any of your systems got crashed. You can read all of the data from a single machine if this machine faces a technical issue data can also be read from other nodes in a Hadoop cluster because the data is copied or replicated by default. By default, Hadoop makes 3 copies of each file block and stored it into different nodes. This replication factor is configurable and can be changed by changing the replication property in the *hdfs-site.xml* file.
4. High Availability is Provided:
Fault tolerance provides High Availability in the Hadoop cluster. High Availability means the availability of data on the Hadoop cluster. Due to fault tolerance in case if any of the DataNode goes down the same data can be retrieved from any other node where the data is replicated. The High available Hadoop cluster also has 2 or more than two Name Node i.e. Active NameNode and Passive NameNode also known as stand by NameNode. In case if Active NameNode fails then the Passive node will take the responsibility of Active Node and provide the same data as that of Active NameNode which can easily be utilized by the user.

5. Cost-Effective:
Hadoop is open-source and uses cost-effective commodity hardware which provides a cost-efficient model, unlike traditional Relational databases that require expensive hardware and high-end processors to deal with Big Data. The problem with traditional Relational databases is that storing the Massive volume of data is not cost-effective, so the company's started to remove the Raw data. which may not result in the correct scenario of their business. Means Hadoop provides us 2 main benefits with the cost one is it's open-source means free to use and the other is that it uses commodity hardware which is also inexpensive.

6. Hadoop Provide Flexibility:
Hadoop is designed in such a way that it can deal with any kind of dataset like structured(MySql Data), Semi-Structured(XML, JSON), Un-structured (Images and Videos) very efficiently. This means it can easily process any kind of data independent of its structure which makes it highly flexible. It is very much useful for enterprises as they can process large datasets easily, so the businesses can use Hadoop to analyze valuable insights of data from sources like social media, email, etc. With this flexibility, Hadoop can be used with log processing, Data Warehousing, Fraud detection, etc.

7. Easy to Use:
Hadoop is easy to use since the developers need not worry about any of the processing work since it is managed by the Hadoop itself. Hadoop ecosystem is also very large comes up with lots of tools like Hive, Pig, Spark, HBase, Mahout, etc.

8. Hadoop uses Data Locality:
The concept of Data Locality is used to make Hadoop processing fast. In the data locality concept, the computation logic is moved near data rather than moving the data to the computation logic. The cost of Moving data on HDFS is costliest and with the help of the data locality concept, the bandwidth utilization in the system is minimized.

9. Provides Faster Data Processing:

Hadoop uses a distributed file system to manage its storage i.e. HDFS(Hadoop Distributed File System). In DFS(Distributed File System) a large size file is broken into small size file blocks then distributed among the Nodes available in a Hadoop cluster, as this massive number of file blocks are processed parallelly which makes Hadoop faster, because of which it provides a High-level performance as compared to the traditional DataBase Management Systems.

10. Support for Multiple Data Formats:
Hadoop supports multiple data formats like CSV, JSON, Avro, and more, making it easier to work with different types of data sources. This makes it more convenient for developers and data analysts to handle large volumes of data with different formats.
11. High Processing Speed:
Hadoop's distributed processing model allows it to process large amounts of data at high speeds. This is achieved by distributing data across multiple nodes and processing it in parallel. As a result, Hadoop can process data much faster than traditional database systems.
12. Machine Learning Capabilities:
Hadoop offers machine learning capabilities through its ecosystem tools like Mahout, which is a library for creating scalable machine learning applications. With these tools, data analysts and developers can build machine learning models to analyze and process large datasets.
13. Integration with Other Tools:
Hadoop integrates with other popular tools like Apache Spark, Apache Flink, and Apache Storm, making it easier to build data processing pipelines. This integration allows developers and data analysts to use their favorite tools and frameworks for building data pipelines and processing large datasets.
14. Secure:
Hadoop provides built-in security features like authentication, authorization, and encryption. These features help to protect data and ensure that only authorized users have access to it. This makes Hadoop a more secure platform for processing sensitive data.
15. Community Support:
Hadoop has a large community of users and developers who contribute to its development and provide support to users. This means that users can access a wealth of resources and support to help them get the most out of Hadoop.

5b. The difference between RDBMS and MapReduce

Sl.No	RDBMS	MapReduce
1.	Traditional row-column based databases, basically used for data storage, manipulation and retrieval.	An open-source software used for storing data and running applications or processes concurrently.
2.	In this structured data is mostly processed.	In this both structured and unstructured data is processed.
3.	It is best suited for OLTP environment.	It is best suited for BIG data.
4.	It is less scalable than Hadoop.	It is highly scalable.
5.	Data normalization is required in RDBMS.	Data normalization is not required in Hadoop.
6.	It stores transformed and aggregated data.	It stores huge volume of data.
7.	It has no latency in response.	It has some latency in response.
8.	The data schema of RDBMS is static type.	The data schema of Hadoop is dynamic type.
9.	High data integrity available.	Low data integrity available than

		RDBMS.
10.	Cost is applicable for licensed software.	Free of cost, as it is an open source software.

6a. The difference between Grid Computing and MapReduce

Sl. No	Grid Computing MPI	MapReduce
1.	Works well for predominantly compute intensive jobs, but it becomes a problem where nodes access larger data volumes since the network bandwidth computing node becomes idle	Hadoop tries to collocate with the data and compute nodes so that the data access is fast because of data locality.
2.	MPI programs have to explicitly manage their own check point and recovery which gives more control to the programmer but makes them more difficult	In MapReduce, since the implementation detects the failed task and re-schedules replacement on machine.
3.	MPI is a shared architecture	MapReduce is a share nothing architecture
4.	Gives great control to the programmer, it also requires that they explicitly handle the mechanics of data flow	Processing in Hadoop happens only at higher level the program thinks in terms of data model since the data flow is implicit

6b. Hadoop Ecosystem is neither a programming language nor a service, it is a platform or framework which solves big data problems. You can consider it as a suite which encompasses a number of services (ingesting, storing, analyzing and maintaining) inside it.

The components of Hadoop 1.0 and Hadoop 2.0 are listed as follows:

- Common – a set of components and interfaces for filesystems and I/O.
- Avro – a serialization system for RPC and persistent data storage.
- MapReduce – a distributed data processing model.
- YARN - Yet Another Resource Negotiator
- HDFS – a distributed filesystem running on large clusters of machines.
- Pig – a data flow language and execution environment for large datasets.
- Hive – a distributed data warehouse providing SQL-like query language.
- HBase – a distributed, column-oriented database.
- Mahout, Spark MLlib - Machine Learning
- Apache Drill -SQL on Hadoop
- ZooKeeper – a distributed, highly available coordination service.
- Sqoop – a tool for efficiently moving data between relational DB and HDFS.
- Oozie - Job Scheduling
- Flume- Data Ingesting Services
- Solr & Lucene - Searching & Indexing

- Ambari - Provision, Monitor and Maintain cluster

The description of each component is as follows:

HDFS

- ***Hadoop Distributed File System*** is the core component or the backbone of Hadoop Ecosystem.
- HDFS is the one, which makes it possible to store different types of large data sets (i.e. structured, unstructured and semi structured data).
- HDFS creates a level of abstraction over the resources, from where we can see the whole HDFS as a single unit.
- It helps us in storing our data across various nodes and maintaining the log file about the stored data (metadata).
- HDFS has two core components, i.e. **NameNode and DataNode**.
    1. The **NameNode** is the main node and it doesn't store the actual data. It contains metadata, just like a log file or you can say as a table of content. Therefore, it requires less storage and high computational resources.
    2. On the other hand, all your data is stored on the **DataNodes** and hence it requires more storage resources. These DataNodes are commodity hardware (like your laptops and desktops) in the distributed environment. That's the reason, why Hadoop solutions are very cost effective.
    3. You always communicate to the NameNode while writing the data. Then, it internally sends a request to the client to store and replicate data on various DataNodes.

MapReduce

It is the core component of processing in a Hadoop Ecosystem as it provides the logic of processing. In other words, MapReduce is a software framework which helps in writing applications that processes large data sets using distributed and parallel algorithms inside Hadoop environment.

- In a MapReduce program, **Map() and Reduce()** are two functions.
    1. The **Map function** performs actions like filtering, grouping and sorting.
    2. While **Reduce function** aggregates and summarizes the result produced by map function.
    3. The result generated by the Map function is a key value pair (K, V) which acts as the input for Reduce function.

Apache PIG

- PIG has two parts: **Pig Latin**, the language and **the pig runtime,** for the execution environment. You can better understand it as Java and JVM.
- It supports *pig latin* language, which has SQL like command structure.
- The compiler internally converts pig latin to MapReduce. It produces a sequential set of MapReduce jobs, and that's an abstraction (which works like black box).
- PIG was initially developed by Yahoo.
- It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analys ing huge data sets.

How Pig works?

In PIG, first the load command, loads the data. Then we perform various functions on it like grouping, filtering, joining, sorting, etc. At last, either you can dump the data on the screen or you can store the result back in HDFS.

Apache Hive

- Facebook created HIVE for people who are fluent with SQL. Thus, HIVE makes them feel at home
- Basically, HIVE is a data warehousing component which performs reading, writing and managing large data sets in a distributed environment using SQL-like interface.

HIVE + SQL = HQL

- The query language of Hive is called Hive Query Language(HQL), which is very similar like SQL.
- It has 2 basic components: **Hive Command Line and JDBC/ODBC driver**.
- The **Hive Command line** interface is used to execute HQL commands.
- While, Java Database Connectivity (**JDBC**) and Object Database Connectivity (**ODBC**) is used to establish connection from data storage.
- Secondly, Hive is highly scalable. As, it can serve both the purposes, i.e. large data set processing (i.e. Batch query processing) and real time processing (i.e. Interactive query processing).
- It supports all primitive data types of SQL.
- You can use predefined functions, or write tailored user defined functions (UDF) also to accomplish your specific needs.

Apache Mahout

Now, let us talk about Mahout which is renowned for machine learning. Mahout provides an environment for creating machine learning applications which are scalable.

Apache Spark

- Apache Spark is a framework for real time data analytics in a distributed computing environment.
- The Spark is written in Scala and was originally developed at the University of California, Berkeley.
- It executes in-memory computations to increase speed of data processing over Map-Reduce.
- It is 100x faster than Hadoop for large scale data processing by exploiting in-memory computations and other optimizations. Therefore, it requires high processing power than Map-Reduce.

As you can see, Spark comes packed with high-level libraries, including support for R, SQL, Python, Scala, Java etc. These standard libraries increase the seamless integrations in complex workflow. Over this, it also allows various sets of services to integrate with it like MLlib, GraphX, SQL + Data Frames, Streaming services etc. to increase its capabilities.

Apache HBase

- HBase is an open source, non-relational distributed database. In other words, it is a NoSQL database.
- It supports all types of data and that is why, it's capable of handling anything and everything inside a Hadoop ecosystem.
- It is modelled after Google's BigTable, which is a distributed storage system designed to cope up with large data sets.
- The HBase was designed to run on top of HDFS and provides BigTable like capabilities.
- It gives us a fault tolerant way of storing sparse data, which is common in most Big Data use cases.
- The HBase is written in Java, whereas HBase applications can be written in REST, Avro and Thrift APIs.

For better understanding, let us take an example. You have billions of customer emails and you need to find out the number of customers who has used the word complaint in their emails. The request needs to be processed quickly (i.e. at real time). So, here we are handling a large data set while retrieving a small amount of data. For solving these kind of problems, HBase was designed.

Apache Zookeeper

- Apache Zookeeper is the coordinator of any Hadoop job which includes a combination of various services in a Hadoop Ecosystem.
- Apache Zookeeper coordinates with various services in a distributed environment.

Before Zookeeper, it was very difficult and time consuming to coordinate between different services in Hadoop Ecosystem. The services earlier had many problems with interactions like common configuration while synchronizing data. Even if the services are configured, changes in the configurations of the services make it complex and difficult to handle. The grouping and naming was also a time-consuming factor.
Due to the above problems, Zookeeper was introduced. It saves a lot of time by performing **synchronization, configuration maintenance, grouping and naming.**
Although it's a simple service, it can be used to build powerful solutions.

Big names like Rackspace, Yahoo, eBay use this service in many of their use cases and therefore, you can have an idea about the importance of Zookeeper.

Apache Oozie

Consider Apache Oozie as a clock and alarm service inside Hadoop Ecosystem. For Apache jobs, Oozie has been just like a scheduler. It schedules Hadoop jobs and binds them together as one logical work.
There are two kinds of Oozie jobs:

1. **Oozie workflow**: These are sequential set of actions to be executed**.** You can assume it as a relay race. Where each athlete waits for the last one to complete his part.
2. **Oozie Coordinator**: These are the Oozie jobs which are triggered when the data is made available to it. Think of this as the response-stimuli system in our body. In the same manner as we respond to an external stimulus, an Oozie coordinator responds to the availability of data and it rests otherwise.

Apache Flume

Ingesting data is an important part of our Hadoop Ecosystem.

- The Flume is a service which helps in ingesting unstructured and semi-structured data into HDFS.
- It gives us a solution which is reliable and distributed and helps us in **collecting, aggregating** and **moving large amount of data sets**.
- It helps us to ingest online streaming data from various sources like network traffic, social media, email messages, log files etc. in HDFS.

Apache Sqoop

Now, let us talk about another data ingesting service i.e. Sqoop. The major difference between Flume and Sqoop is that:

- Flume only ingests unstructured data or semi-structured data into HDFS.
- While Sqoop can import as well as export structured data from RDBMS or Enterprise data warehouses to HDFS or vice versa.

When we submit Sqoop command, our main task gets divided into sub tasks which is handled by individual Map Task internally. Map Task is the sub task, which imports part of data to the Hadoop Ecosystem. Collectively, all Map tasks imports the whole data.

Apache Ambari

Ambari is an Apache Software Foundation Project which aims at making Hadoop ecosystem more manageable. It includes software for **provisioning, managing and monitoring**Apache Hadoop clusters.
The Ambari provides:

1. **Hadoop cluster provisioning**:

   - It gives us step by step process for installing Hadoop services across a number of hosts.
   - It also handles configuration of Hadoop services over a cluster.

   Hadoop cluster management:

   - It provides a central management service for starting, stopping and re-configuring Hadoop services across the cluster.

   Hadoop cluster monitoring:

   - For monitoring health and status, Ambari provides us a dashboard.
   - The **Amber Alert framework** is an alerting service which notifies the user, whenever the attention is needed. For example, if a node goes down or low disk space on a node, etc.

7a. A HDFS cluster has two types of node operating in a master-worker pattern: a namenode (the master) and a number of datanodes (workers).
The **Namenode** manages the *filesystem namespace*. It maintains the *filesystem tree* and the *metadata* for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files:

- The namespace image (fsImage)
- The edit log

The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts. A *client* accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.

Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

Without the namenode, the filesystem cannot be used. In fact, if the machine running the namenode were obliterated, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes. For this reason, it is important to make the namenode resilient to failure, and Hadoop provides two mechanisms for this.

1. The first way is to back up the files that make up the persistent state of the filesystem metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

2. It is also possible to run a *secondary namenode*, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large. The secondary namenode usually runs on a separate physical

machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged namespace image, which can be used in the event of the namenode failing. However, the state of the secondary namenode lags that of the primary, so in the event of total failure of the primary, data loss is almost certain. The usual course of action in this case is to copy the namenode's metadata files that are on NFS to the secondary and run it as the new primary.

7b. The combination of replicating namenode metadata on multiple filesystems, and using the secondary namenode to create checkpoints protects against data loss, but does not provide high-availability of the filesystem. The namenode is still a *single point of failure* (SPOF), since if it did fail, all clients-including MapReduce jobs-would be unable to read, write, or list files, because the namenode is the sole repository of the metadata and the file-to block mapping. In such an event the whole Hadoop system would effectively be out of service until a new namenode could be brought online.

To recover from a failed namenode in this situation, an administrator starts a new primary namenode with one of the filesystem metadata replicas, and configures datanodes and clients to use this new namenode. The new namenode is not able to serve requests until it has

i. loaded its namespace image into memory,

ii. replayed its edit log, and

iii. received enough block reports from the datanodes to leave safe mode.

On large clusters with many files and blocks, the time it takes for a namenode to start from cold can be 30 minutes or more. The long recovery time is a problem for routine maintenance too. In fact, since unexpected failure of the namenode is so rare, the case for planned downtime is actually more important in practice.

The 0.23 release series of Hadoop remedies this situation by adding support for HDFS high-availability (HA). In this implementation there is a pair of- namenodes in an activestand by configuration. In the event of the failure of the active namenode, the standby takes over its duties to continue servicing client requests without a significant interruption.

The architectural changes are needed to replace active name node with standby name node:

i. The namenodes must use highly-available shared storage to share the edit log. When a standby namenode comes up it reads up to the end of the shared edit log to

synchronize its state with the active namenode, and then continues to read new entries as they are written by the active namenode.

ii. Datanodes must send block reports to both namenodes since the block mappings are stored in a namenode's memory, and not on disk.Clients must be configured to handle namenode failover, which uses a mechanism that is transparent to users.

iii. If the active namenode fails, then the standby can take over very quickly (in a few tens of seconds) since it has the latest state available in memory: both the latest edit log entries, and an up-to-date block mapping. The actual observed failover time will be longer in practice (around a minute or so), since the system needs to be conservative in deciding that the active namenode has failed.

iv. In the unlikely event of the standby being down when the active fails, the administrator can still start the standby from cold. This is no worse than the non- HA case, and from an operational point of view it's an improvement, since the process is a standard operational procedure built into Hadoop.

Failover and fencing

The transition from the active namenode to the standby is managed by a new entity in the system called the *failover controller*. Failover controllers are pluggable, but the first implementation uses ZooKeeper to ensure that only one namenode is active. Each namenode runs a lightweight failover controller process whose job it is to monitor its namenode for failures (using a simple heartbeating mechanism) and trigger a failover should a namenode fail.

Failover may also be initiated manually by an adminstrator, in the case of routine maintenance, for example. This is known as a *graceful failover*, since the failover controller arranges an orderly transition for both namenodes to switch roles.

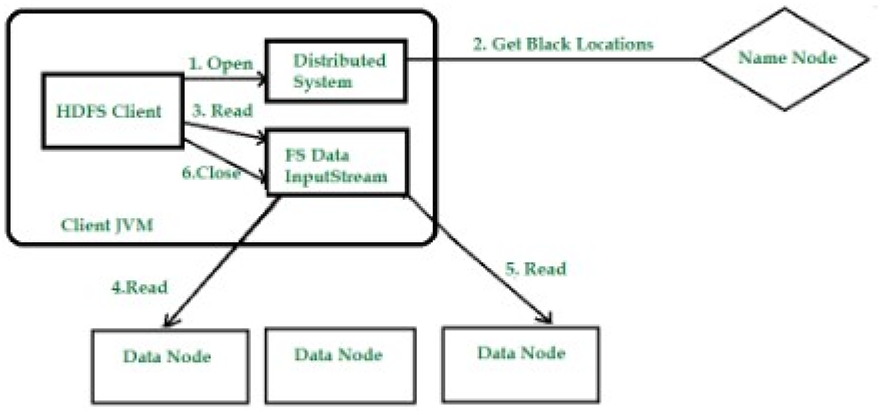
In the case of an ungraceful failover, however, it is impossible to be sure that the failed namenode has stopped running. For example, a slow network or a network partition can trigger

a failover transition, even though the previously active namenode is still running, and thinks it is still the active namenode.

The HA implementation goes to great lengths to ensure that the previously active namenode is prevented from doing any damage and causing corruption—a method known as fencing. The system employs a range of fencing mechanisms, including killing the namenode's process, revoking its access to the shared storage directory (typically by using a vendor-specific NFS command), and disabling its network port via a remote management command. As a last resort, *the previously active namenode can be fenced with a technique rather graphically known as STONITH, or "shoot the other node in the head", which uses a specialized power distribution unit to forcibly power down the host machine.*

Client failover is handled transparently by the client library. The simplest implementation uses client-side configuration to control failover. The HDFS URI uses a logical hostname which is mapped to a pair of namenode addresses (in the configuration file), and the client library tries each namenode address until the operation succeeds.

8a.



Step 1: The client opens the file it wishes to read by calling open() on the File System Object(which for HDFS is an instance of Distributed File System).

Step 2: Distributed File System(DFS) calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file. For each block, the name node returns the addresses of the data nodes that have a copy of that block. The DFS returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the data node and name node I/O.

Step 3: The client then calls read() on the stream. DFSInputStream, which has stored the info node addresses for the primary few blocks within the file, then connects to the primary (closest) data node for the primary block in the file.

Step 4: Data is streamed from the data node back to the client, which calls read() repeatedly on the stream.

Step 5: When the end of the block is reached, DFSInputStream will close the connection to the data node, then finds the best data node for the next block. This happens transparently to the client, which from its point of view is simply reading an endless stream. Blocks are read as, with the DFSInputStream opening new connections to data nodes because the client reads through the stream. It will also call the name node to retrieve the data node locations for the next batch of blocks as needed.

Step 6: When the client has finished reading the file, a function is called, close() on the FSDataInputStream.

8b. HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware.

Strengths of HDFS

1. Very large files: There are Hadoop clusters running today that store petabytes of data.

2. Streaming data access HDFS cares about the time to read the whole dataset than the latency of reading the first record.

3. Commodity hardware Hadoop is designed to run on clusters of commodity hardware (commonly available hardware available from multiple vendors) for which the chance of node failure across the cluster is high. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

Weaknesses of HDFS

1. **Low-latency data access:** Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS.
2. **Lots of small files**: Since the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, while storing millions of files is feasible, billions is beyond the capability of current hardware.

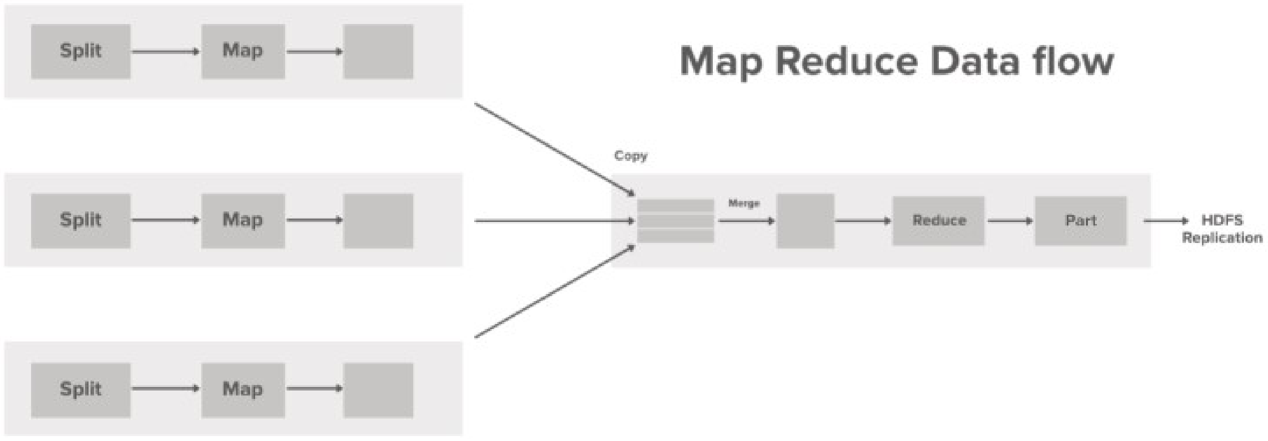
Multiple writers, arbitrary file modifications: Files in HDFS may be written to by a single writer. There is no support for multiple writers, or for modifications at arbitrary offsets in the file.

9a. Mapper is the initial line of code that initially interacts with the input dataset. suppose, If we have 100 Data-Blocks of the dataset we are analyzing then, in that case, there will be 100 Mapper program or process that runs in parallel on machines(nodes) and produce there own output known as intermediate output which is then stored on Local Disk, not on HDFS. The output of the mapper act as input for Reducer which performs some sorting and aggregation operation on data and produces the final output.

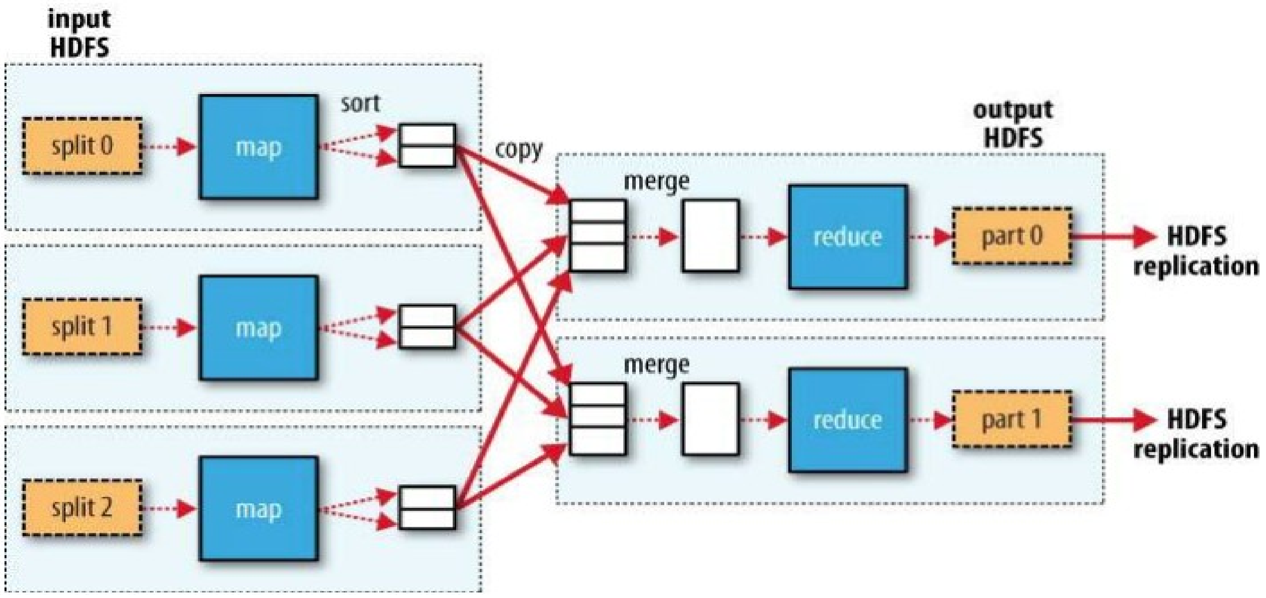
Reducer is the second part of the Map-Reduce programming model. The Mapper produces the output in the form of key-value pairs which works as input for the Reducer. But before sending this intermediate key-value pairs directly to the Reducer some process will be done which shuffle and sort the key-value pairs according to its key values. The output generated by the Reducer will be the final output which is then stored on HDFS(Hadoop Distributed File System). Reducer mainly performs some computation operation like addition, filtration, and aggregation.

Steps of Data-Flow:

- At a time single input split is processed. Mapper is overridden by the developer according to the business logic and this Mapper run in a parallel manner in all the machines in our cluster.
- The intermediate output generated by Mapper is stored on the local disk and shuffled to the reducer to reduce the task.
- Once Mapper finishes their task the output is then sorted and merged and provided to the Reducer.
- Reducer performs some reducing tasks like aggregation and other compositional operation and the final output is then stored on HDFS in part-r-00000(created by default) file.



MapReduce dataflow with a Single Reduce Task



MapReduce dataflow with a Multiple Reduce Task

9b. *GenericOptionsParser* is a utility to parse command line arguments generic to the Hadoop framework. *GenericOptionsParser* recognizes several standard command line arguments, enabling applications to easily specify a namenode, a jobtracker, additional configuration resources etc.

The supported generic options are:

```
    -conf <configuration file>     specify a configuration file
    -D <property=value>            use value for given property
    -fs <local|namenode:port>      specify a namenode
    -jt <local|jobtracker:port>    specify a job tracker
    -files <comma separated list of files>    specify comma separated
                       files to be copied to the map reduce cluster
    -libjars <comma separated list of jars>   specify comma separated
                       jar files to include in the classpath.
    -archives <comma separated list of archives>    specify comma
           separated archives to be unarchived on the compute machines.
```

The general command line syntax is:

```
bin/hadoop command [genericOptions] [commandOptions]
```

Generic command line arguments **might** modify Configuration objects, given to constructors.

The functionality is implemented using Commons CLI.

Examples:

```
$ bin/hadoop dfs -fs darwin:8020 -ls /data
list /data directory in dfs with namenode darwin:8020

$ bin/hadoop dfs -D fs.default.name=darwin:8020 -ls /data
list /data directory in dfs with namenode darwin:8020

$ bin/hadoop dfs -conf hadoop-site.xml -ls /data
list /data directory in dfs with conf specified in hadoop-site.xml

$ bin/hadoop job -D mapred.job.tracker=darwin:50020 -submit job.xml
submit a job to job tracker darwin:50020

$ bin/hadoop job -jt darwin:50020 -submit job.xml
submit a job to job tracker darwin:50020

$ bin/hadoop job -jt local -submit job.xml
submit a job to local runner

$ bin/hadoop jar -libjars testlib.jar
-archives test.tgz -files file.txt inputjar args
```

job submission with libjars, files and archives

10a.

```
import java.io.IOException;

import java.util.Iterator;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {

    // Mapper

    public static class MaxTemperatureMapper extends
            Mapper<LongWritable, Text, Text, Text> {

    public static final int MISSING = 9999;

    @Override
        public void map(LongWritable arg0, Text Value, Context context)
                throws IOException, InterruptedException {

        String line = Value.toString();

            if (!(line.length() == 0)) {

                String date = line.substring(6, 14);

                float temp_Max = Float.parseFloat(line.substring(39, 45).trim());


                float temp_Min = Float.parseFloat(line.substring(47, 53).trim());

                if (temp_Max > 30.0) {

                    context.write(new Text("The Day is Hot Day :" + date),
                                         new Text(String.valueOf(temp_Max)));
                }
                if (temp_Min < 15) {

                    context.write(new Text("The Day is Cold Day :" + date),
                            new Text(String.valueOf(temp_Min)));
                }
            }
        }

    }

// Reducer

    public static class MaxTemperatureReducer extends
            Reducer<Text, Text, Text, Text> {

        public void reduce(Text Key, Iterator<Text> Values, Context context)
                throws IOException, InterruptedException {



            String temperature = Values.next().toString();
            context.write(Key, new Text(temperature));
        }

    }

    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();

        Job job = new Job(conf, "weather example");

        job.setJarByClass(MyMaxMin.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setMapperClass(MaxTemperatureMapper.class);

        job.setReducerClass(MaxTemperatureReducer.class);

        job.setInputFormatClass(TextInputFormat.class);

        job.setOutputFormatClass(TextOutputFormat.class);

        Path OutputPath = new Path(args[1]);

        FileInputFormat.addInputPath(job, new Path(args[0]));

        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        OutputPath.getFileSystem(conf).delete(OutputPath);
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
}
```

10b. The JobTracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

1. Client applications submit jobs to the Job tracker.
2. The JobTracker talks to the NameNode to determine the location of the data
3. The JobTracker locates TaskTracker nodes with available slots at or near the data
4. The JobTracker submits the work to the chosen TaskTracker nodes.
5. The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.
6. A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable.
7. When the work is completed, the JobTracker updates its status.
8. Client applications can poll the JobTracker for information.

   The JobTracker is a point of failure for the Hadoop MapReduce service. If it goes down, all running jobs are halted.