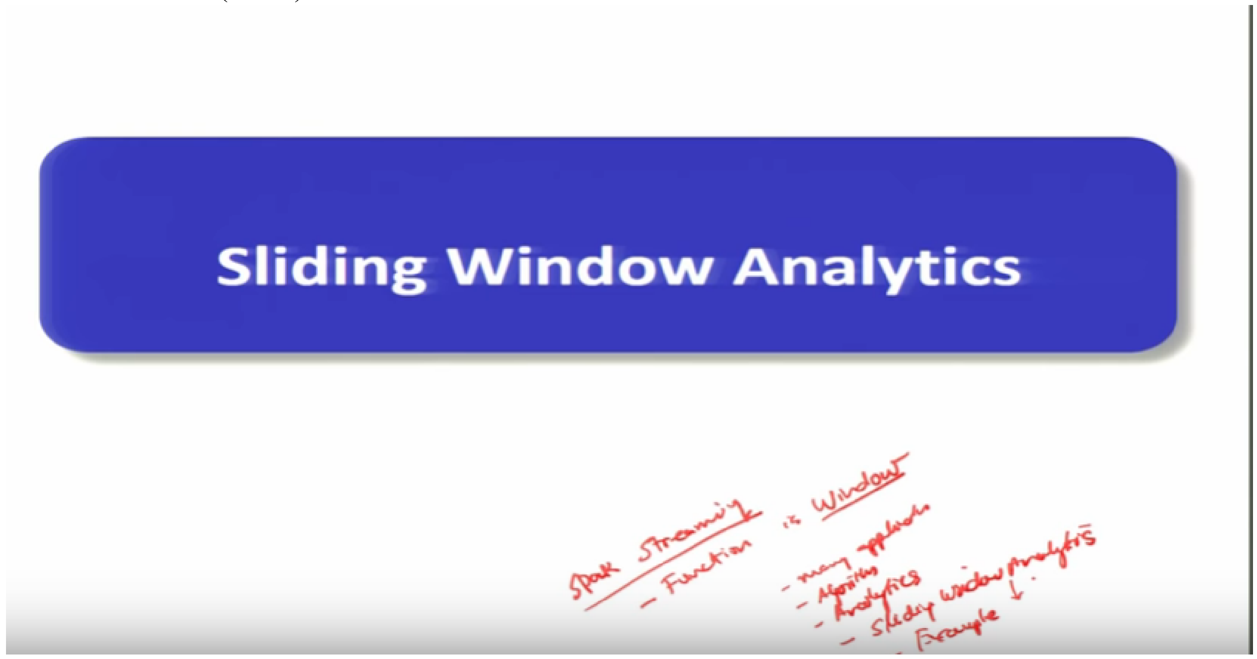
Lecture – 21

Sliding Window

Analytics

Sliding-Window Analytics.
Refer Slide Time :(0: 15)

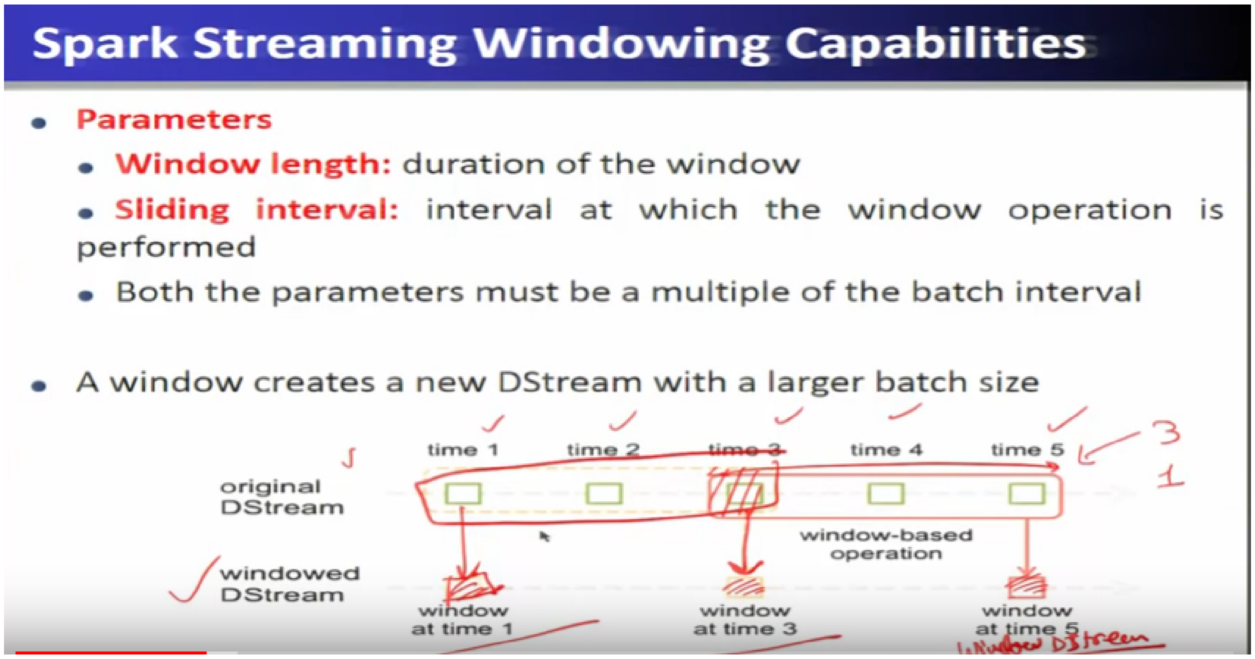


Sliding Window Analytics

Spark Streaming
- Function is Window
- many applications
- Algorithms
- Analytics
- Sliding window Analytics
- Example

So, in this, Spark Streaming, we will see that, the most important, functionality is the window based operations. So, using this window based operation, we can design many application and also, we can design many new algorithms, for doing the analytics and therefore they are called as a, 'Sliding Window Analytics'. Some of the, some examples, we are going to see, in the further slides on this, what are the different applications programs, we can build, using this feature which is called a, 'Window' and in hence, we will see in more detail about the, sliding window analytics.

Refer Slide Time :(1: 27)

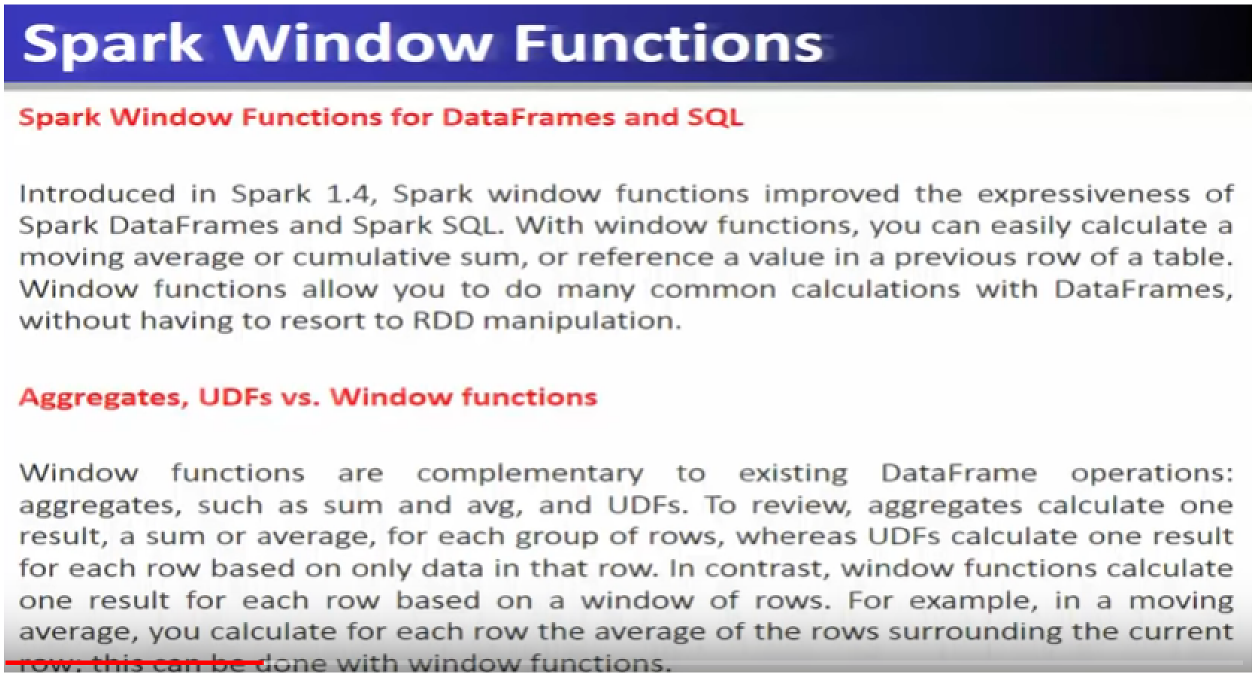


Spark Streaming Windowing Capabilities

- **Parameters**
  - **Window length:** duration of the window
  - **Sliding interval:** interval at which the window operation is performed
  - Both the parameters must be a multiple of the batch interval
- A window creates a new DStream with a larger batch size

So, Spark Streaming windowing capabilities, that is the window has, window function, has two parameters, the first one is called, 'Window Length', which is nothing but, the duration of window, the second parameter is called the, the, 'Sliding Interval' which is an interval at, which the window operation has been performed. So, both the parameters must be, in a multiple of batch intervals. So, we have to see how, these particular parameters, we can use, that is the window length and sliding interval and using this we can design many applications. So, a window creates a new, D stream with a larger batch size. So, for example, for example the if this is the input, D stream and this particular input D stream is, divided into the batches of, time one, two, time three, time four time, five and this window length, is three, of three batches, which includes, the window length is of three, size three and sliding interval is also, is basically, the two then obviously, there will be one overlap. So, that is shown over here. So, the original date D steam is shown here, which is divided into the batches, one, two, three, four, five and when the window operation is being performed, that is the window heed, windowed D stream, which will be based on per window, this particular operation will be available. So, there will be an overlap of one. So, that means, you see that, in window, at time one, there will be a window D Stream, will be available and at time three, they again there will be a window D stream at time three will be available and then window at time four, this window D Stream, will be available. So, these particular window, based sliding interval, will tell that the window operation is being performed, this is the interval. So, sliding interval, of size two, will give so, sliding interval of size one, will give, this kind of

Refer Slide Time :(4: 08)



Spark Window Functions

Spark Window Functions for DataFrames and SQL

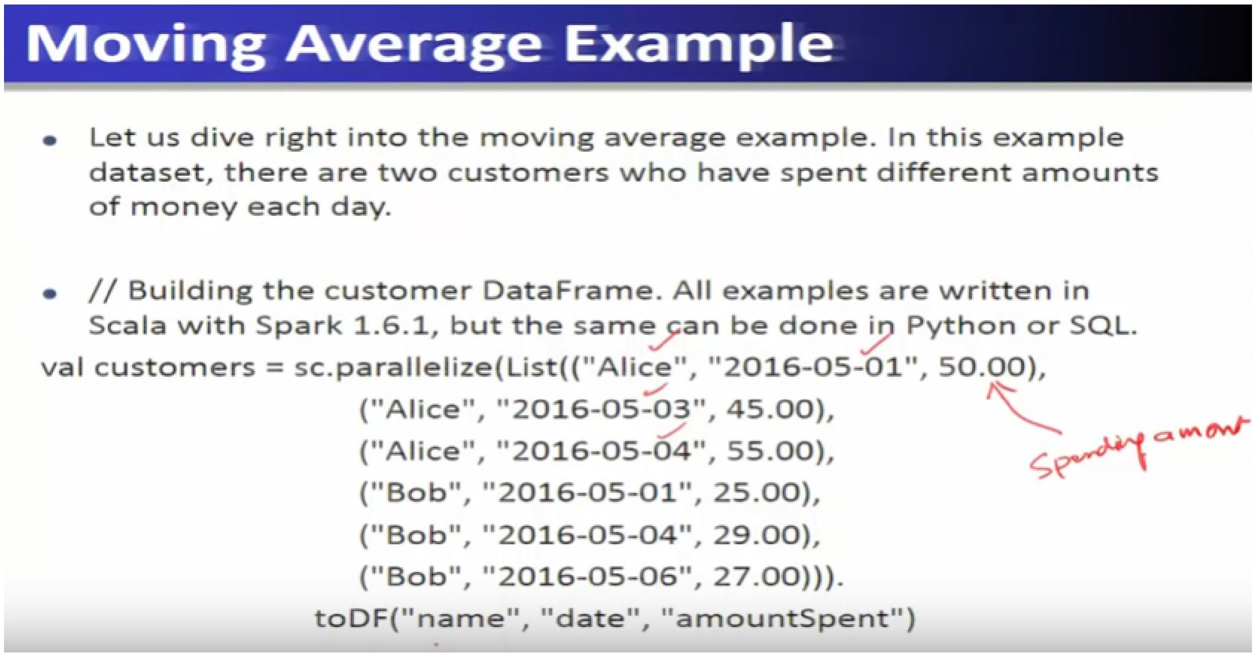
Introduced in Spark 1.4, Spark window functions improved the expressiveness of Spark DataFrames and Spark SQL. With window functions, you can easily calculate a moving average or cumulative sum, or reference a value in a previous row of a table. Window functions allow you to do many common calculations with DataFrames, without having to resort to RDD manipulation.

Aggregates, UDFs vs. Window functions

Window functions are complementary to existing DataFrame operations: aggregates, such as sum and avg, and UDFs. To review, aggregates calculate one result, a sum or average, for each group of rows, whereas UDFs calculate one result for each row based on only data in that row. In contrast, window functions calculate one result for each row based on a window of rows. For example, in a moving average, you calculate for each row the average of the rows surrounding the current row; this can be done with window functions.

Now, Spark window functions, Spark window functions for data, for data frame and SQL. So, which is introduced in a Spark one point four, Spark window functions improve the expressiveness of, Spark data frames and Spark SQL. With window functions, you can easily calculate the moving average or cumulative sum or reference value, in the previous row of a table. So, window function allows, you to do many common calculations, with it friends without having and without having to resort to RDDS, manipulations. So, aggregates UDF, versus window functions. So, window functions are complementary, to existing data frame operations, such as aggregates, aggregate such as sum and average and UDF, give you the aggregate calculations one result, one result, some are the average, for each group of rows, whereas the UDF, calculate one result for each row. So, based on, only the data, on in that row in contrast, window functions, calculate one result for each, row based on the window off rows, for example, in a moving average you can calculate each row, the average of the rows, surrounding the current row and this can be done, using the window functions.

Refer Slide Time :(5: 30)



Moving Average Example

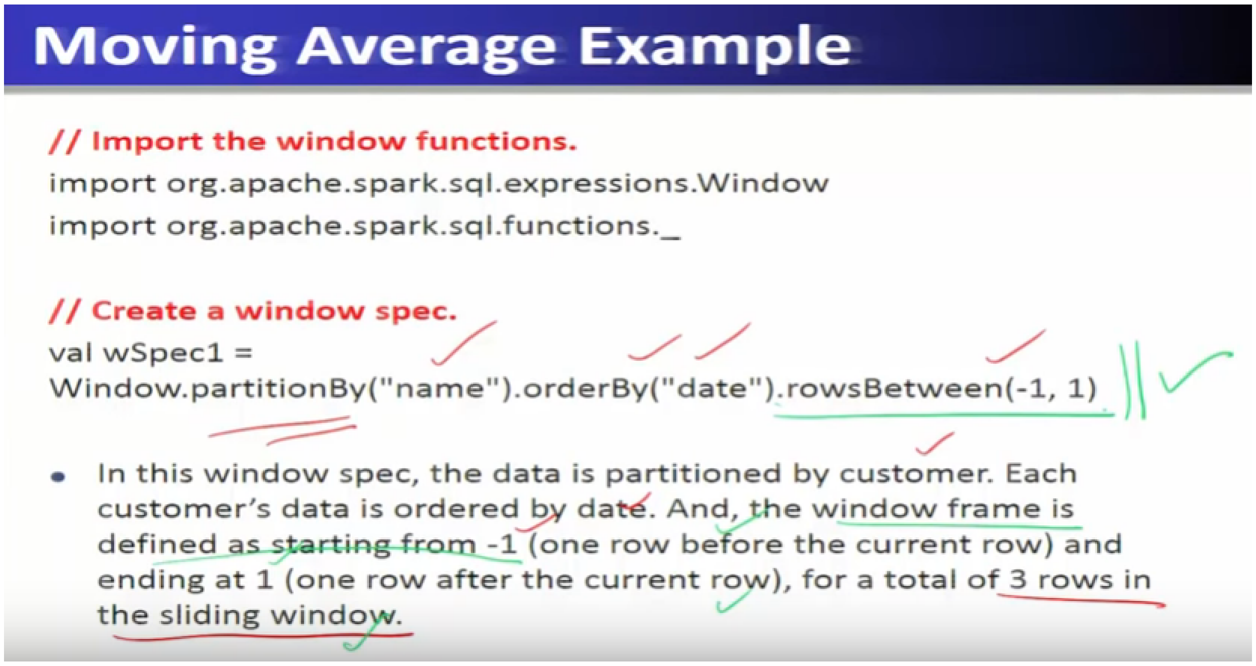
- Let us dive right into the moving average example. In this example dataset, there are two customers who have spent different amounts of money each day.

- // Building the customer DataFrame. All examples are written in Scala with Spark 1.6.1, but the same can be done in Python or SQL.

```
val customers = sc.parallelize(List(("Alice", "2016-05-01", 50.00),
                                    ("Alice", "2016-05-03", 45.00),
                                    ("Alice", "2016-05-04", 55.00),
                                    ("Bob", "2016-05-01", 25.00),
                                    ("Bob", "2016-05-04", 29.00),
                                    ("Bob", "2016-05-06", 27.00))).
                        toDF("name", "date", "amountSpent")
```

So, let us see this moving average example. So, let us dive into, the moving average example, in this example, the data set, there are two customers, who have, different spending, amounts of money each day. So, let us build the, the customer data, frame and all the examples, are written in the scalar and the same can be, visualized using Python or SQL. So, let us see the, customer and its data, is given in the form of the list, such as, the customer is Alice and 2 0 1 5, 2 0 16, 5th month and, and the date, is given over here and the time is also, B and the spending is given. So, this is the, the, the amount of spending, amount is given and this is the name of the customer and these are their dates when, the customer has, done this spending. So, given this particular detail, that this one, the name of the customer a date and the amount which is spent. So, given this particular data, of customers and.

Refer Slide Time :(6: 46)


Moving Average Example

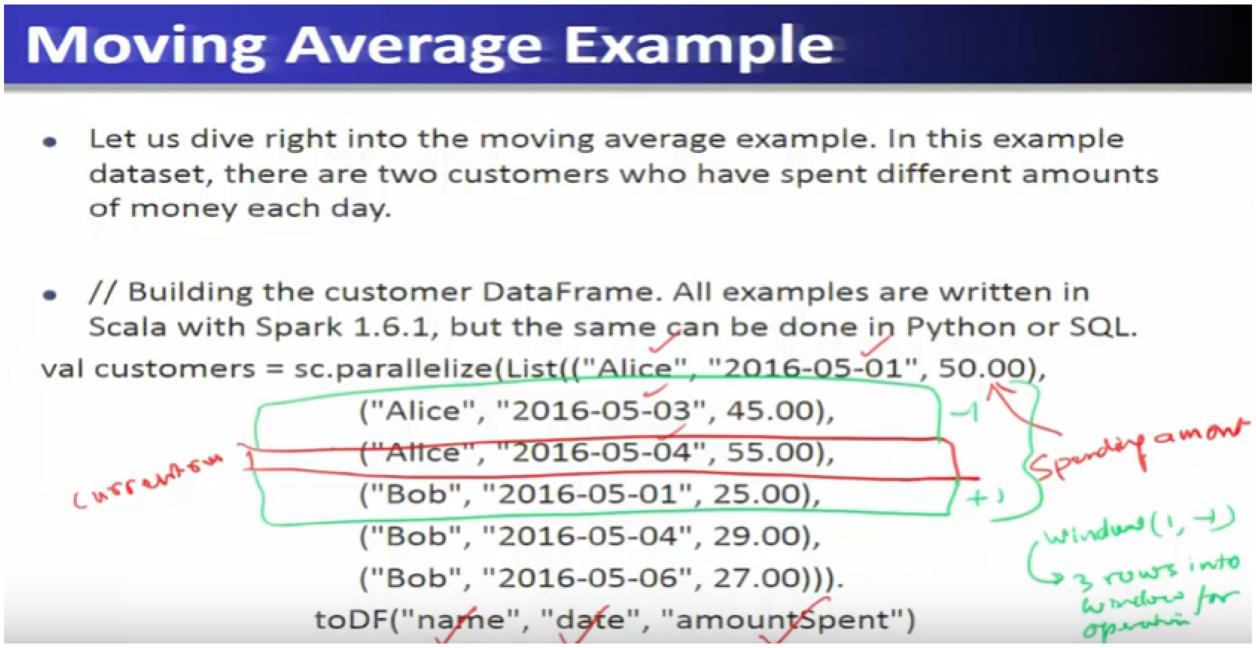
```
// Import the window functions.
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._

// Create a window spec.
val wSpec1 =
Window.partitionBy("name").orderBy("date").rowsBetween(-1, 1)
```

- In this window spec, the data is partitioned by customer. Each customer's data is ordered by date. And, the window frame is defined as starting from -1 (one row before the current row) and ending at 1 (one row after the current row), for a total of 3 rows in the sliding window.

Now, we can perform the window operations, on top of it, to calculate the, moving average, of spending. So, moving average. So, create the window specifications and so, we have to specify the window. So, so the window, will partition by the name and order by the date and the windows will calculate, we'll consider the rules, between minus 1and 1. So, minus 1 and 1 means in this, window specs, the data is partitioned by the customer and each customer's data, is ordered by the date and the window frame is defined as, starting from minus 1, that is one row before the current row and ending at one, that is one row after the current row. So, the total there are, three different rows, are considered in the sliding window.

Refer Slide Time :(8: 01)



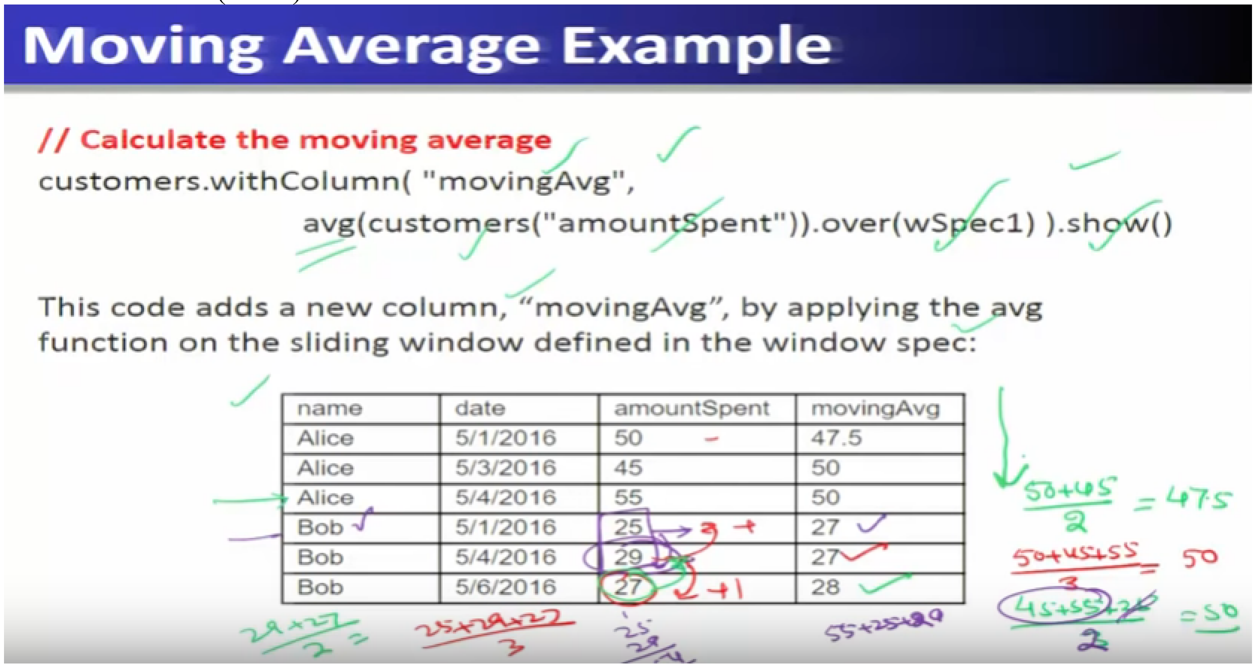
Moving Average Example

- Let us dive right into the moving average example. In this example dataset, there are two customers who have spent different amounts of money each day.
- // Building the customer DataFrame. All examples are written in Scala with Spark 1.6.1, but the same can be done in Python or SQL.

```
val customers = sc.parallelize(List(("Alice", "2016-05-01", 50.00),
                    ("Alice", "2016-05-03", 45.00),
                    ("Alice", "2016-05-04", 55.00),
                    ("Bob", "2016-05-01", 25.00),
                    ("Bob", "2016-05-04", 29.00),
                    ("Bob", "2016-05-06", 27.00))).
                toDF("name", "date", "amountSpent")
```

So, three rows means that, for example, if this is the current row, then this row, then the row above, this row, will be the minus one and the row below this, will be the plus one. So, the window of size, one and minus one, will require, three rows, into the window, for operations. So, therefore the, the window frame is defined as starting from minus one, that is one row before the current row and ending at one that is one, to after the current row. So, total there are three different rows, are there in the sliding window. So, having understood this notion, in this particular statement or written in the Scala, we have to specify using window specification. So, window partition by the name order by, the row and the, the rows between minus 1 and 1. So, that will specify the window operations.

Refer Slide Time :(9: 26)



Moving Average Example

// Calculate the moving average

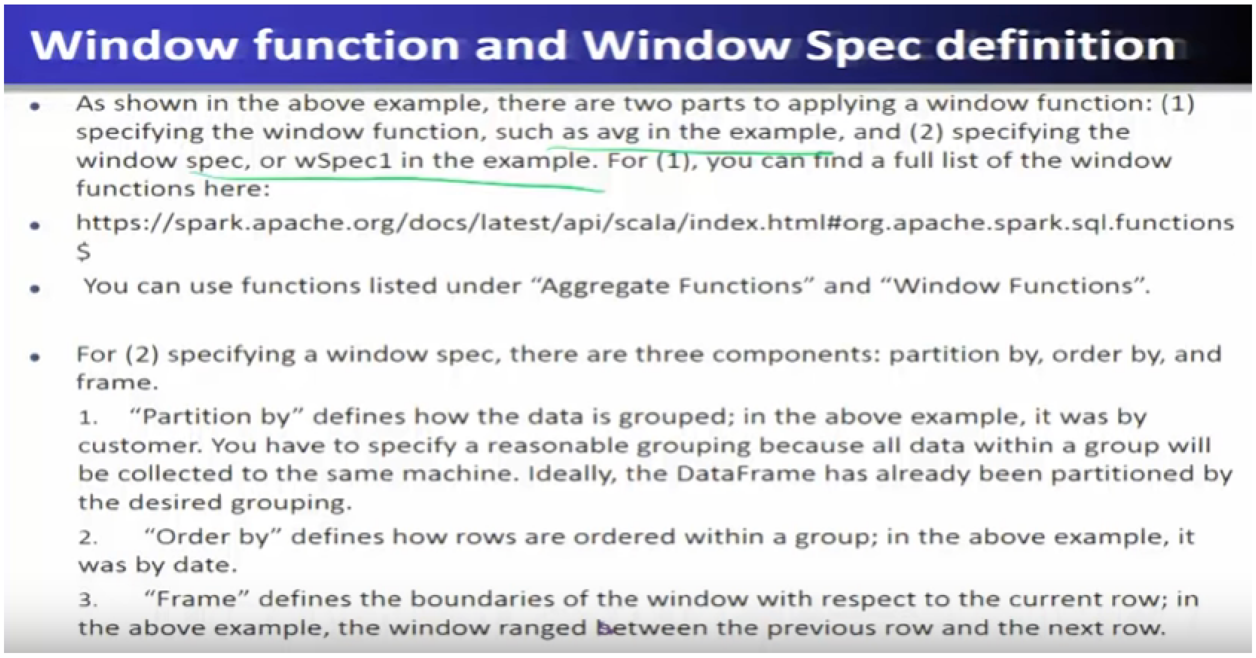
```
customers.withColumn( "movingAvg",
                avg(customers("amountSpent")).over(wSpec1) ).show()
```

This code adds a new column, "movingAvg", by applying the avg function on the sliding window defined in the window spec:

name	date	amountSpent	movingAvg
Alice	5/1/2016	50	47.5
Alice	5/3/2016	45	50
Alice	5/4/2016	55	50
Bob	5/1/2016	25	27
Bob	5/4/2016	29	27
Bob	5/6/2016	27	28

Now, let us calculate the moving average using this particular scalar command, that is the customers, with column and moving average, we have to calculate, using the average of, all of the customers, by the account, amount, amount spent or the windows specification, which we have shown and that we have to show. So, this code, will now add a new column, which is called, 'Moving Average'. So, moving average column will be now added, in the customers table and by applying the average function. So, this average function, on the sliding window, which is defined in the window specs, in the previous slide. So, that means, let us see how, 47.5 which we have obtained, is that, if this is the customer, its spending is 50. So, before that, is the row number minus 1 which is missing. But, after this there is a row, which is called, '45' which is available. So, the window of so, this window, will consider50, plus 45, divided by 2, that is nothing but, forty seven point five. Now, consider the, now consider this particular, entry that is 45, will consider this row as, minus 1 and this row as plus 1. So, this window will contain all three values and they will be 50, plus 45, plus55 and if you take this average, it comes out to be 50. So, this value will come over here again. Now, you will consider this third entry that is 55. So, this particular row will be minus one and the further row will be plus one. So, for this, to calculate the average we require 45, plus 55, plus 25divided by 3, that comes out to be 50. Similarly for 25 we can see that, for 25we can see that this becomes minus 1 and this becomes plus 1. So, for 25, these 3will, these three values will be, taken up, into the window for everything 55, plus25, plus 29, no not that, actually. So, so up to this point, 25 will not be used, why because? It belongs to another customer. So, so here 45 plus, 55 divided by 2 that will be about 24. So, let us see about the Bob. So, Bob the previous values, will be let us see this, case of Bob again. So, in Bob case, the previous value is not there and plus 1 will be there. So, only these 2values will be used, why because? Before that it is an, Alice values.

Refer Slide Time :(13: 05)

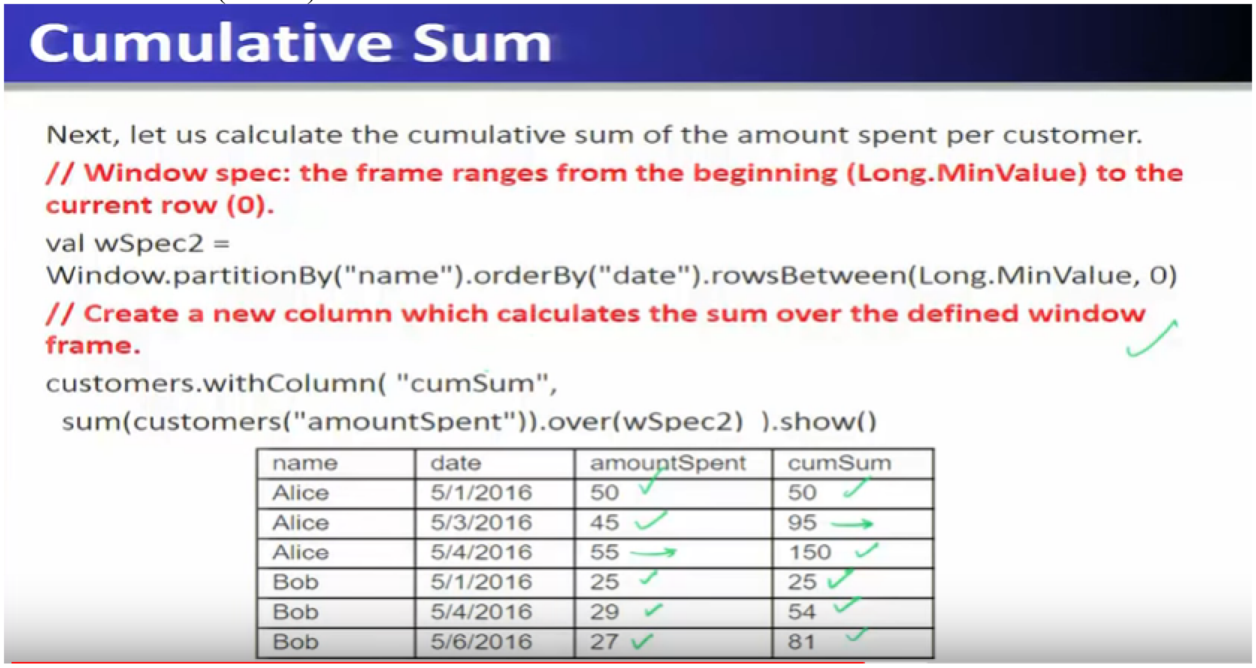


Window function and Window Spec definition

- As shown in the above example, there are two parts to applying a window function: (1) specifying the window function, such as avg in the example, and (2) specifying the window spec, or wSpec1 in the example. For (1), you can find a full list of the window functions here:
- https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$
- You can use functions listed under "Aggregate Functions" and "Window Functions".
- For (2) specifying a window spec, there are three components: partition by, order by, and frame.
  1. "Partition by" defines how the data is grouped; in the above example, it was by customer. You have to specify a reasonable grouping because all data within a group will be collected to the same machine. Ideally, the DataFrame has already been partitioned by the desired grouping.
  2. "Order by" defines how rows are ordered within a group; in the above example, it was by date.
  3. "Frame" defines the boundaries of the window with respect to the current row; in the above example, the window ranged between the previous row and the next row.

So, 25, so its, its average will be 25, plus 29 divided by 2, that is 27. And similarly for, this value we will consider it as minus 1 and this is plus 1. So, 25 plus, 29 plus, 27 divided by 3, that comes out to be 27.

And for 27 we will see that, for 27 it will only use minus 1. So, 29 plus 27, divided by 2, that comes out to be, 28. So, this way the moving average is automatically, calculated and now this is the, the small number of entries that we have to see that, that the window is sliding and is the moving average is now, calculated and filled in this particular table. So, window based function and window specification definition, as shown in the example, that there are two parts, to the two applying the window function, first is specifying the window function such as, the average, in the example and the second is specifying the window specs that is windows specification. So, for the average, you can find the full list of window functions available, on the this one Streaming, Spark Streaming functions, which you can apply, you can also, apply the aggregate functions and the window functions together, for two that is specifying the window aspects, there are three different components, partitioned by, order by and, and frame. So, partition by defines how the data, is grouped, in the above example, it was customer, you can you have to specify, reasonable grouping because, all the data within the group will be collected, on to the same machine. So, ideally the data, frame has already being partitioned, by the desired groupings. So, when you go for, order by it will define how the rows are ordered, within the group, in the above example, it was the by the date. Similarly the frames defines the boundaries of the window with as but, to the current row, in the above example the window, reached between the previous row and the next row. So, all these are, very much, needed and once it is understood then, the partition by and order by together, will perform the optimizations, in the data movement, from different cluster systems and secondly the, the frame that is, the window operations is very, very much needed or moving average and it depends upon, the application how, this particular window is used for that application.
Refer Slide Time :(16: 08)



Cumulative Sum

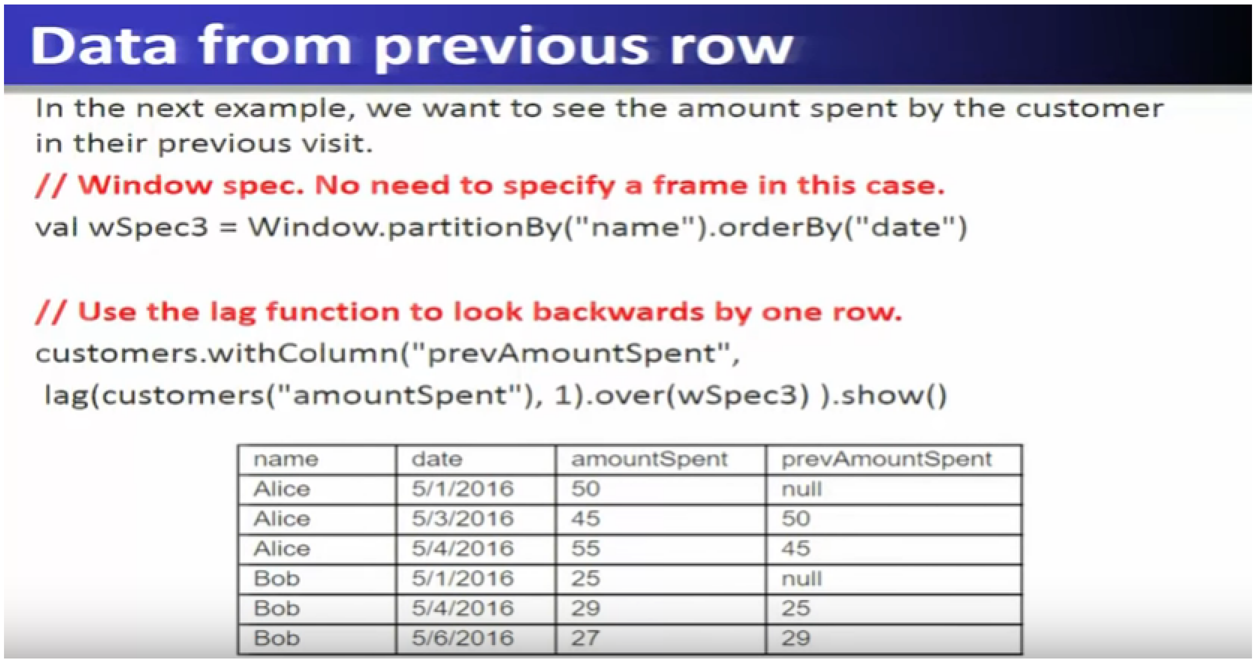
Next, let us calculate the cumulative sum of the amount spent per customer.

```
// Window spec: the frame ranges from the beginning (Long.MinValue) to the
current row (0).
val wSpec2 =
Window.partitionBy("name").orderBy("date").rowsBetween(Long.MinValue, 0)
// Create a new column which calculates the sum over the defined window
frame.
customers.withColumn( "cumSum",
  sum(customers("amountSpent")).over(wSpec2) ).show()
```

name	date	amountSpent	cumSum
Alice	5/1/2016	50	50
Alice	5/3/2016	45	95
Alice	5/4/2016	55	150
Bob	5/1/2016	25	25
Bob	5/4/2016	29	54
Bob	5/6/2016	27	81

Now, we will see another application, of window based analysis, analytics, which is called, ‘Cumulative Sum’. Let us calculate the cumulative sum of the amounts spent per, customer here the window aspects the, the frame ranges, from beginning that is, the long and minimum value, to the, current row that is zero and window specification is, aspects to, we are now saying that, you know to partition by the name, like in the previous example, order by the date also, by the previous as per previous example and the rows between, the long mean value, zero. So, let us create the new column which calculates the sum or that define window frame. So, the customer with the column, with the column that is cumulative sum, that is sum, we have to define as, function of the customer amount is spent, over the window specs, which we are now going to show. So, minimum value, along minimum value and zero, we are going to so. So, Alice has spent 50. So, the accumulated amount is 50only then, he as Alice has, spent 45. So, cumulative will be 95 automatically will be calculated, then Alice has, spent 45, 55then it becomes 150 in this case. Then Bob has spent 25. So, it becomes 25 it has this point 29 it becomes 54, then it becomes then it has expired 27, it and the cumulative amount becomes 81. So, let us see that, the new, row which is called, 'Cumulative Sum' and this will be, this will be created, using Q, using this particular command, customer with the column, a new column will beaded and will be calculated in this,

Refer Slide Time :(18: 12)



Data from previous row

In the next example, we want to see the amount spent by the customer in their previous visit.

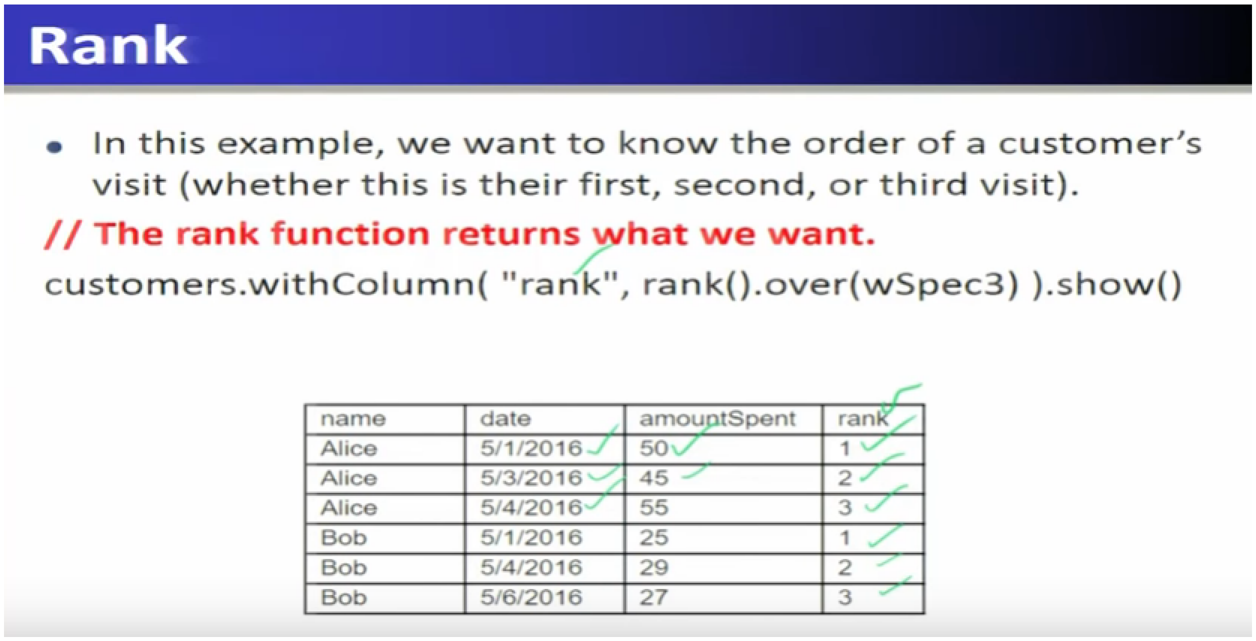
```
// Window spec. No need to specify a frame in this case.
val wSpec3 = Window.partitionBy("name").orderBy("date")

// Use the lag function to look backwards by one row.
customers.withColumn("prevAmountSpent",
 lag(customers("amountSpent"), 1).over(wSpec3) ).show()
```

name	date	amountSpent	prevAmountSpent
Alice	5/1/2016	50	null
Alice	5/3/2016	45	50
Alice	5/4/2016	55	45
Bob	5/1/2016	25	null
Bob	5/4/2016	29	25
Bob	5/6/2016	27	29

using the, the window functions.

Refer Slide Time :(18: 20)



Rank

- In this example, we want to know the order of a customer's visit (whether this is their first, second, or third visit).

```
// The rank function returns what we want.
customers.withColumn( "rank", rank().over(wSpec3) ).show()
```

name	date	amountSpent	rank
Alice	5/1/2016	50	1
Alice	5/3/2016	45	2
Alice	5/4/2016	55	3
Bob	5/1/2016	25	1
Bob	5/4/2016	29	2
Bob	5/6/2016	27	3

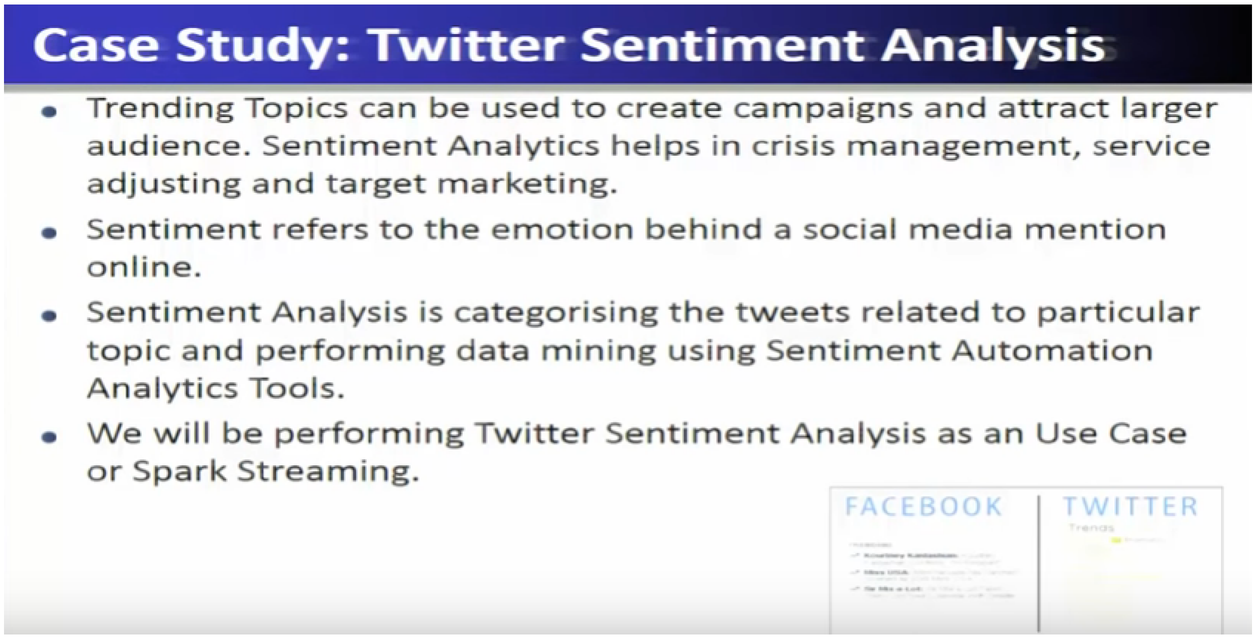
Now, we are going to see the "rank", in this example, we want to know, the order of the customers visit, to the store, whether this is the first time, second time or third visit. So, that is called the, 'Rank'. So, rank function returns, what we want? So, customer with the column and we are going to add the rank, within that particular column. So, you see the rank is headed over here and we want to rank or the window specification and, and then we have to show. **So**, window a specification, we are doing the same, window specification, which we have specified in the previous example and now, let us see that, is the Alice, when it has a span, when it has visited on this date and spend 50amount then, first time it has visited, then next day, it has spent 45, then second time it has visited and the next time it is third time. Similarly the, the Bob has be stayed, three times. So, as it is, visiting and spending, on another date, the automatically the rank of, that customer is increasing. Now, this kind of analysis or analytics, is very much, useful In many of the websites, which are doing the e-commerce or doing the business and based on for example, the first time a customer is logging into the system or doing the business, with that or taking the service, then as far as, the promotions are concerned, using this particular information it will give, some promotional discount, for the new customer to be attracted and for the old customers, these particular values are changed accordingly, based on the policies of the business.

Refer Slide Time :(20: 12)

Case Study: Twitter Sentiment Analysis with Spark Streaming

Now, let us see another case study, of this Spark Streaming, analytics, using twitter sentiment analysis, using the Spark Streaming.

Refer Slide Time :(20: 23)

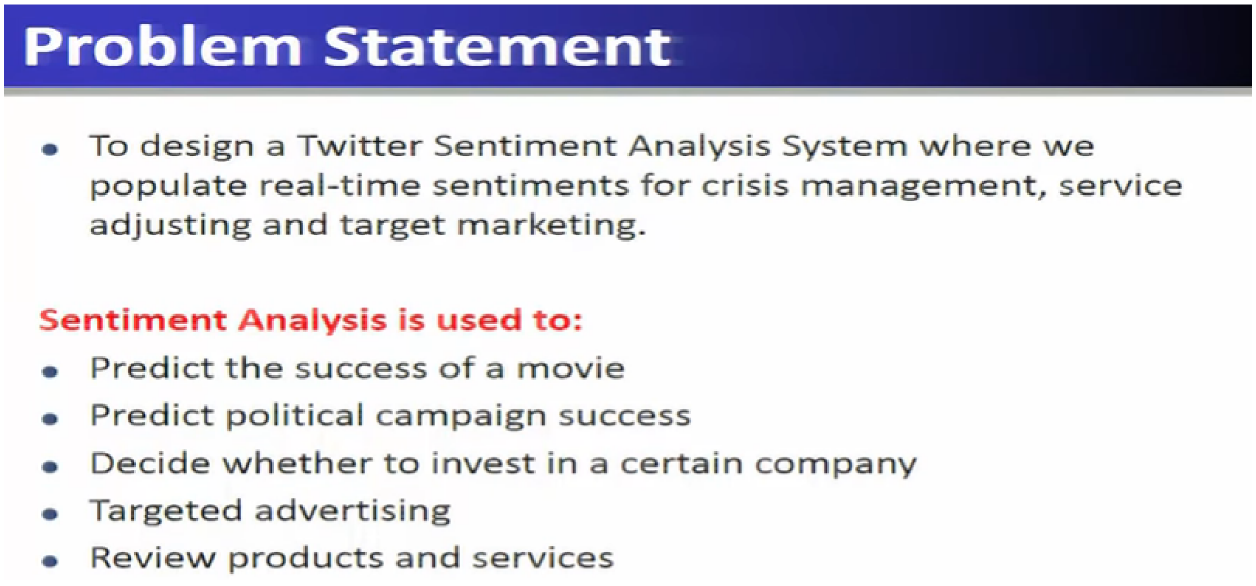


Case Study: Twitter Sentiment Analysis

- Trending Topics can be used to create campaigns and attract larger audience. Sentiment Analytics helps in crisis management, service adjusting and target marketing.
- Sentiment refers to the emotion behind a social media mention online.
- Sentiment Analysis is categorising the tweets related to particular topic and performing data mining using Sentiment Automation Analytics Tools.
- We will be performing Twitter Sentiment Analysis as an Use Case or Spark Streaming.

So, twitter sentiment analysis that means here we want to find out the trending topics, can be used to create the campaigns and attract large, larger audience, sentiment analysis, helps in the crisis management and service adjusting and, and target marketing. So, therefore the sentiment analysis is going to become very important, thing in the, the web and in all businesses and the sentiment referred to the emotion, emotion behind the social media, mentioned online, sentiment analysis is categorizing the tweets, related to the particular topic and performing data mining, using sentiment, automation, analytics, tools. We will be performing, Twitter sentiment analysis as an use case or Streaming data.

Refer Slide Time :(21: 16)


Problem Statement

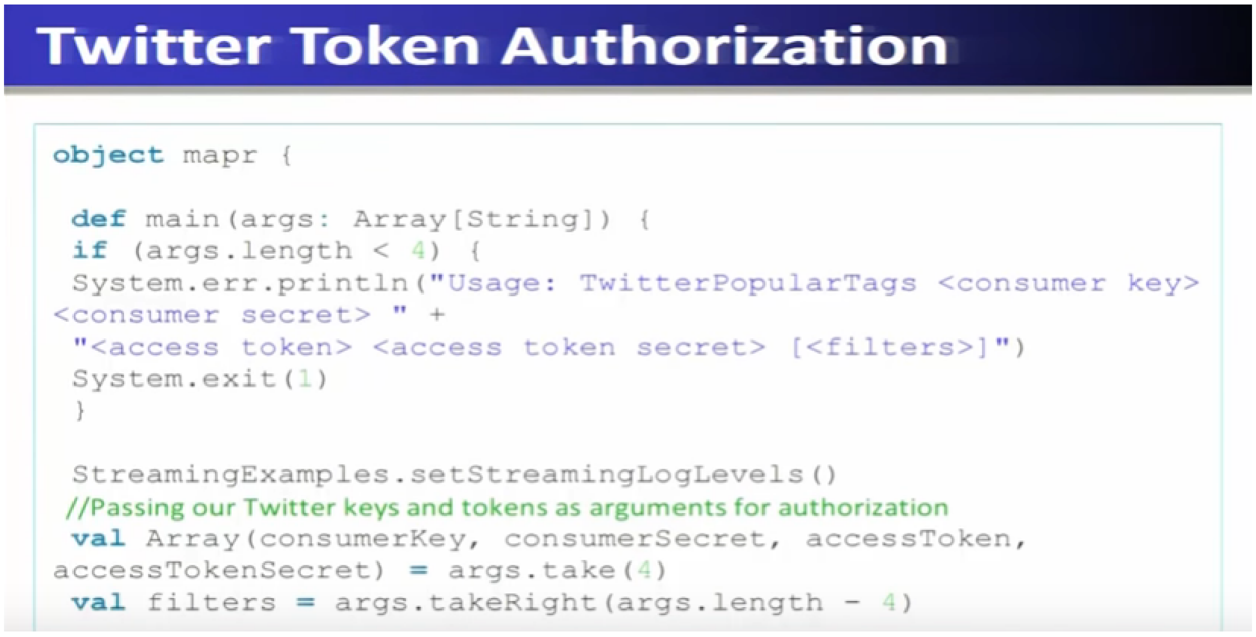
- To design a Twitter Sentiment Analysis System where we populate real-time sentiments for crisis management, service adjusting and target marketing.

Sentiment Analysis is used to:

- Predict the success of a movie
- Predict political campaign success
- Decide whether to invest in a certain company
- Targeted advertising
- Review products and services

So, let us see the what we are going to see here is to design, a sentiment, a twitter sentiment analysis, system where we populate the real-time sentiments, for crisis management, service adjusting and the target marketing. Sentiment analysis, is used to predict the success of a movie, predict the political campaign, success decide whether to invest in a certain company target, advertising, review the product and services.

Refer Slide Time :(21: 46)


Twitter Token Authorization

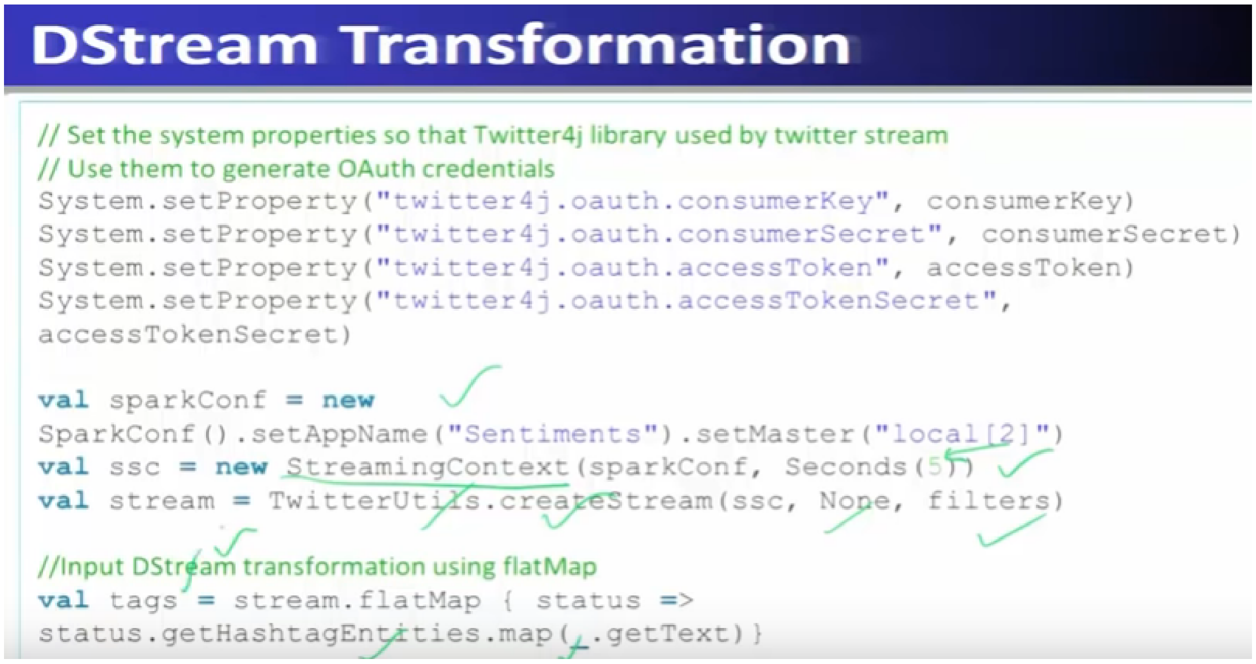
```
object mapr {

 def main(args: Array[String]) {
 if (args.length < 4) {
 System.err.println("Usage: TwitterPopularTags <consumer key>
<consumer secret> " +
 "<access token> <access token secret> [<filters>]")
 System.exit(1)
 }

 StreamingExamples.setStreamingLogLevels()
//Passing our Twitter keys and tokens as arguments for authorization
 val Array(consumerKey, consumerSecret, accessToken,
accessTokenSecret) = args.take(4)
 val filters = args.takeRight(args.length - 4)
```

So, let us go and see this particular details. So, we have to follow the pipeline, this is called token, 'Token Authorization' and here, the twitter, which are given as the input,

Refer Slide Time :(22: 05)



DStream Transformation

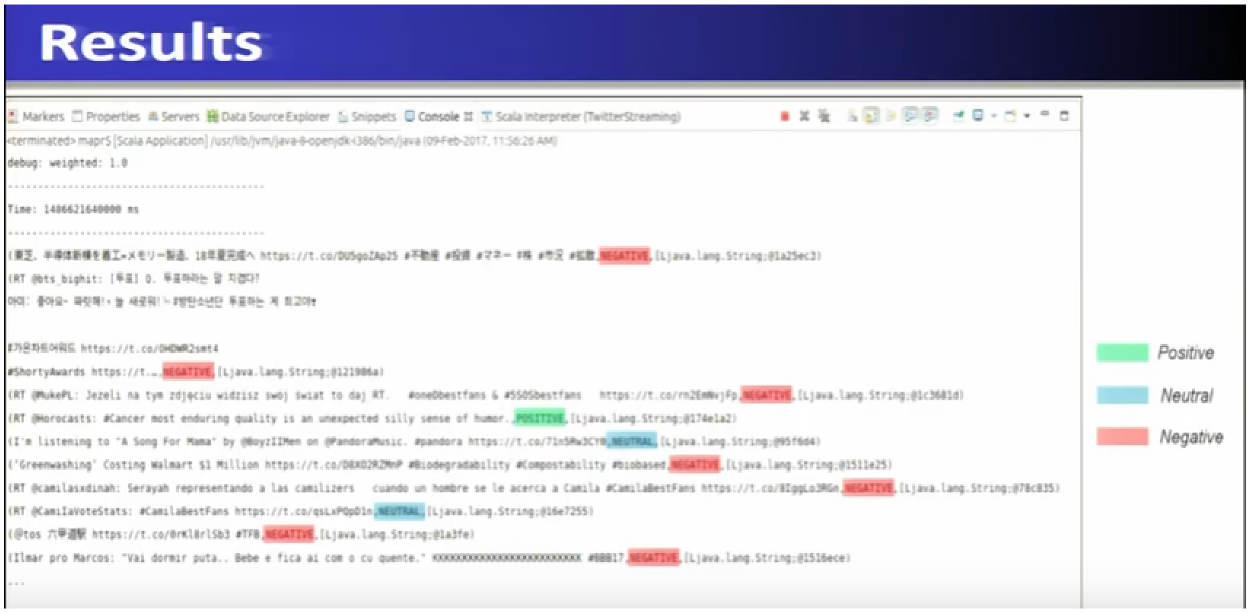
```
// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret",
accessTokenSecret)

val sparkConf = new
SparkConf().setAppName("Sentiments").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(5))
val stream = TwitterUtils.createStream(ssc, None, filters)

//Input DStream transformation using flatMap
val tags = stream.flatMap { status =>
status.getHashtagEntities.map(_.getText)}
```

we are going to apply and now, we perform the part the Streaming, D Streaming, transformations on this d stream, Twitter D stream, which is now, captured and in the, as the stream and given here, in the system. So, the first statement, is about we have to, start a new Spark configuration and we have to set the Spark Streaming or D Streaming context. So, Streaming context, is just like a Spark context, here we are using the Streaming context and this particular Streaming context, we are creating the window of 5 seconds. Now, this utility Twitter, utility will create the stream and this stream will, will use the Twitter, utility and this will be the input stream to the Spark system, is Streaming system and it will apply some filters, filters are none and so, all the stream will be taken up, into the Streaming system. Now, this particular D stream will be transformed using flat map. So, flat map will use the, will get the hash tag entries and it will get the text out of the hash tag entries and this is called the, 'Tag'. So, out of the stream, it will capture the hash tag entries.

Refer Slide Time :(23: 44)



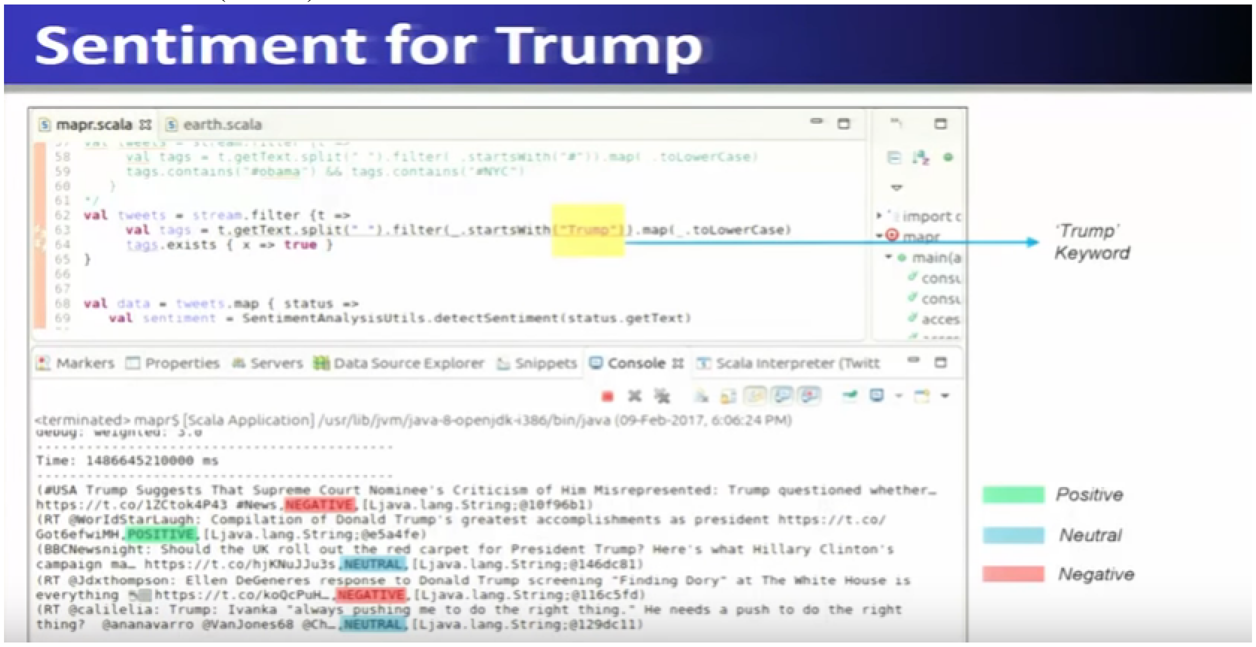
Results

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  Scala Interpreter (TwitterStreaming)
<terminated> mapr$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 11:56:26 AM)
debug: weighted: 1.0
-------------------------------------------
Time: 1486621640000 ms
-------------------------------------------
(東芝、半導体新棟を着工=メモリー製造、18年夏完成へ https://t.co/DU5goZAp25 #不動産 #投資 #マネー #株 #不況 #拡散,NEGATIVE,[Ljava.lang.String;@1a25ec3)
(RT @bts_bighit: [투표] Q. 투표하라는 말 지겹다?
아미: 좋아요- 파잇팅!• 늘 새로워! ♪ #방탄소년단 투표하는 게 최고야♥

#가온차트어워드 https://t.co/OHDWR2smt4
#ShortyAwards https://t.…,NEGATIVE,[Ljava.lang.String;@121986a)
(RT @MukePL: Jezeli na tym zdjęciu widzisz swój swiat to daj RT.   #oneDbestfans & #5SOSbestfans   https://t.co/rn2EmNvjFp,NEGATIVE,[Ljava.lang.String;@1c3681d)
(RT @Horocasts: #Cancer most enduring quality is an unexpected silly sense of humor.,POSITIVE,[Ljava.lang.String;@174e1a2)
(I'm listening to "A Song For Mama" by @BoyzIIMen on @PandoraMusic. #pandora https://t.co/71n5Rw3CY0,NEUTRAL,[Ljava.lang.String;@95f6d4)
('Greenwashing' Costing Walmart $1 Million https://t.co/D8XO2RZMnP #Biodegradability #Compostability #biobased,NEGATIVE,[Ljava.lang.String;@1511e25)
(RT @camilasxdinah: Serayah representando a las camilizers   cuando un hombre se le acerca a Camila #CamilaBestFans https://t.co/8IggLo3RGn,NEGATIVE,[Ljava.lang.String;@78c835)
(RT @CamilaVoteStats: #CamilaBestFans https://t.co/qsLxPOpD1n,NEUTRAL,[Ljava.lang.String;@16e7255)
(@tos 六甲道駅 https://t.co/0rKl8rlSb3 #TFB,NEGATIVE,[Ljava.lang.String;@1a3fe)
(Ilmar pro Marcos: "Vai dormir puta.. Bebe e fica ai com o cu quente." xxxxxxxxxxxxxxxxxxxxxxxxxxx #BBB17,NEGATIVE,[Ljava.lang.String;@1516ece)
...
```

So, this is the some results, which we have gathered, into the captured into the system.

Refer Slide Time :(23: 56)



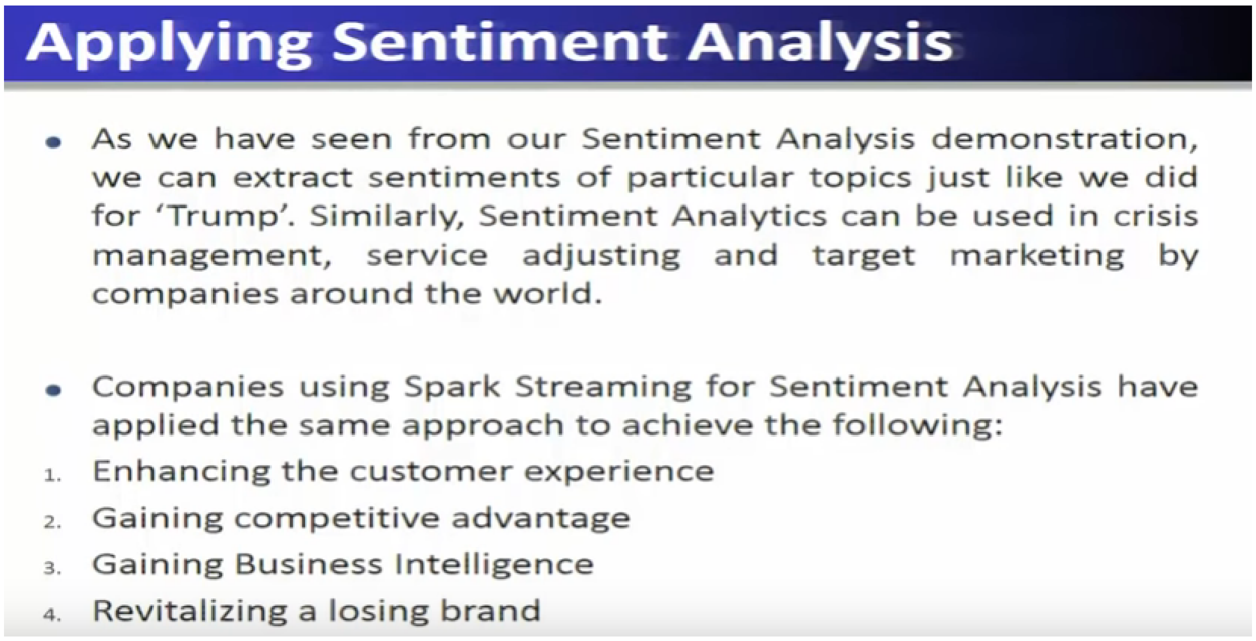
Sentiment for Trump

```
mapr.scala  earth.scala
        val tags = t.getText.split(" ").filter(_.startsWith("#")).map(_.toLowerCase)
        tags.contains("#obama") && tags.contains("#NYC")
    }
    */
    val tweets = stream.filter {t =>
        val tags = t.getText.split(" ").filter(_.startsWith("Trump")).map(_.toLowerCase)
        tags.exists { x => true }
    }


    val data = tweets.map { status =>
      val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  Scala Interpreter (Twitt
<terminated> mapr$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 6:06:24 PM)
debug: weighted: 3.0
-------------------------------------------
Time: 1486645210000 ms
-------------------------------------------
(#USA Trump Suggests That Supreme Court Nominee's Criticism of Him Misrepresented: Trump questioned whether…
https://t.co/1ZCtok4P43 #News,NEGATIVE,[Ljava.lang.String;@10f96b1)
(RT @WorIdStarLaugh: Compilation of Donald Trump's greatest accomplishments as president https://t.co/
Got6efwiMH,POSITIVE,[Ljava.lang.String;@e5a4fe)
(BBCNewsnight: Should the UK roll out the red carpet for President Trump? Here's what Hillary Clinton's
campaign ma… https://t.co/hjKNuJJu3s,NEUTRAL,[Ljava.lang.String;@146dc81)
(RT @Jdxthompson: Ellen DeGeneres response to Donald Trump screening "Finding Dory" at The White House is
everything  https://t.co/koQcPuH…,NEGATIVE,[Ljava.lang.String;@116c5fd)
(RT @calilelia: Trump: Ivanka "always pushing me to do the right thing." He needs a push to do the right
thing?  @ananavarro @VanJones68 @Ch…,NEUTRAL,[Ljava.lang.String;@129dc11)
```

And now, we are going to analyze, analyzing the sentiment, of a particular politician,
Refer Slide Time :(24: 05)

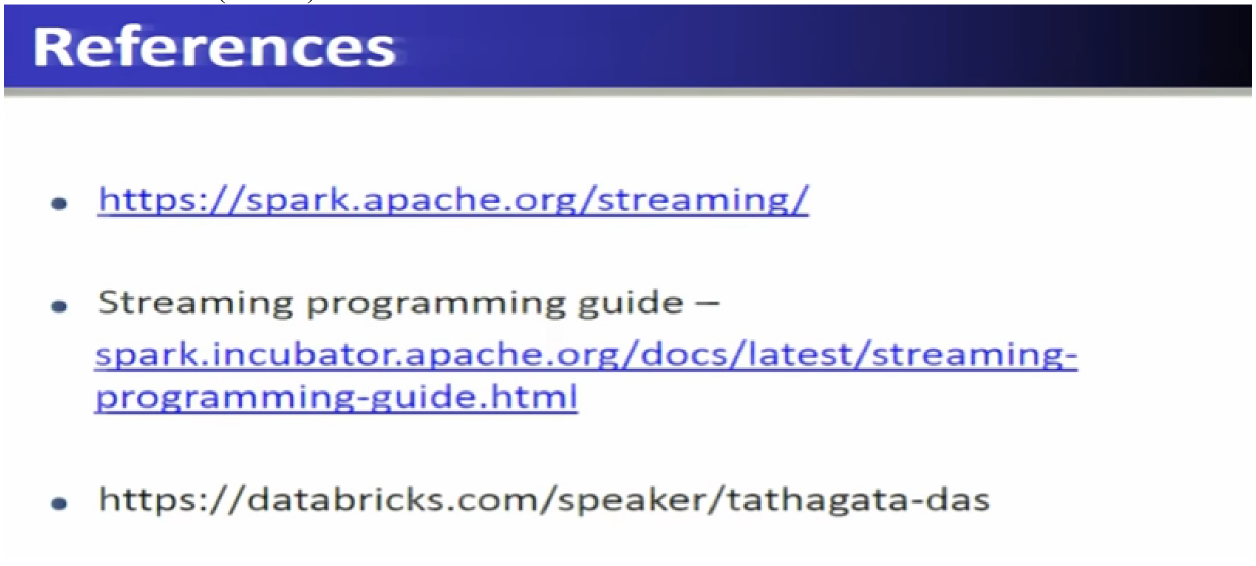


Applying Sentiment Analysis

- As we have seen from our Sentiment Analysis demonstration, we can extract sentiments of particular topics just like we did for 'Trump'. Similarly, Sentiment Analytics can be used in crisis management, service adjusting and target marketing by companies around the world.

- Companies using Spark Streaming for Sentiment Analysis have applied the same approach to achieve the following:
1. Enhancing the customer experience
2. Gaining competitive advantage
3. Gaining Business Intelligence
4. Revitalizing a losing brand

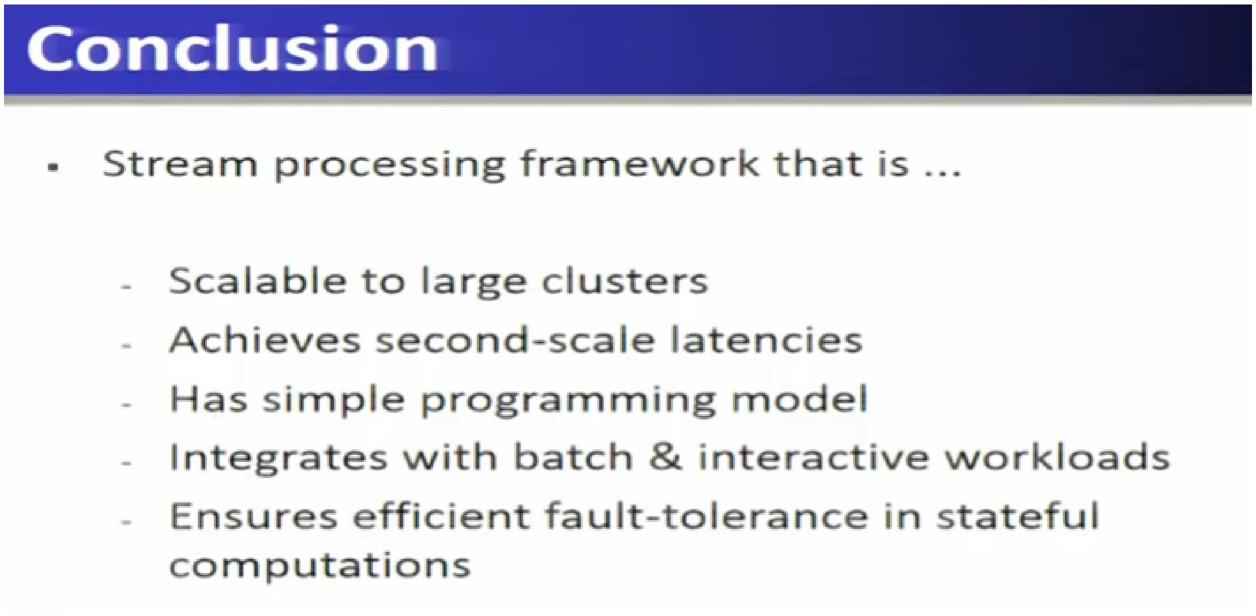
using this and now, we are applying the sentiment analysis, as we have seen, from our sentiment analysis demonstration, we can extract the sentiment of a particular topic and we did, for a politician, 'Trump'. Similarly sentiment analysis can be used in the crisis management and service adjusting and target marketing, by the companies around the world. Companies use using the Spark stream, for sentiment I have applied, the same approach to achieve the following, enhancing the customer experience, gaining the competitive advantage, gaining business intelligence, revitalizing losing brand.
Refer Slide Time :(24: 45)



References

- https://spark.apache.org/streaming/
- Streaming programming guide – spark.incubator.apache.org/docs/latest/streaming-programming-guide.html
- https://databricks.com/speaker/tathagata-das

So, these are the following references, for further details, to refer the Spark, Spark dot, Apache dot, org slash Streaming and the Streaming programming guide, is also their Spark dot, incubator dot, Apache dot, org and also, we have used some of the material from data bricks dotcom, speaker the, Tathagata Das.
Refer Slide Time :(25: 18)



Conclusion

- Stream processing framework that is ...
  - Scalable to large clusters
  - Achieves second-scale latencies
  - Has simple programming model
  - Integrates with batch & interactive workloads
  - Ensures efficient fault-tolerance in stateful computations

Conclusion; stream processing framework that is, scalable to the large clusters, achieves seconds of scaled latency, has the programming, simple programming model, will integrate into the batch and interactive workloads, ensures, efficient power tolerance, in the stateful computations. Thank you.