Apache Spark Fundamentals

Day 2 Lab

High-Performance Spark - Dataframe, Dataset, UDFs, and Caching

Transcript:

34:19
with y'all welcome to the basics and Spark Advance setup example so what we're going to be wanting to be doing
34:24
here first is make sure you have Docker desktop running and installed uh that's going to be the big thing and then you want to go to the data engineer handbook
34:30
and you go boot camp materials spark fundamentals and then in here you'll see
34:35
there's a spark fundamentals and advanced spark setup so there's two options here you can either run make up or Docker compose up so in here I'm just
34:43
going to show with Docker compose Docker compose up now this is uh setting up and uh
34:50
running all of the docker stuff that it needs to download and install and do all
34:55
of the uh stuff to like set up Docker you'll see this needs to set up four containers we need one for Iceberg we
35:02
need one for minio we need one for MC and we need one for spark Iceberg those are going to be the three big things
35:08
that are going to be part of this right and then ultimately we are going to be accessing this from Local Host
35:13
8888 that will be uh where this is going make sure you have Docker set up otherwise this is not going to work
35:20
right and this is going to take a little bit of time because these images are really big right so uh we're just going
35:26
to have to Vibe here for just a little bit while this is uh installing and then we will uh we'll go from there should be
35:33
a pretty exciting time here as we uh get to the um actual setup here CU I know y'all

35:41
are really excited to get ready to test out some spark stuff should be pretty
35:47
great so I think if we go here is it set up okay there we go so that actually set
35:53
up then we can go into notebooks and we can go here right and then this should
35:59
uh run so you'll see we have our data pulled in here this is all of the data that we have and so if you can get this
36:05
up and running uh you're good to go for the lab I want to show essentially how
36:10
the data set versus the data frame versus the um spark SQL apis work and
36:19
also know that this is supposed to work um it it works this code worked pretty
36:26
much exactly as you would expect in data bricks I don't know why there's like it's for some reason uh the tabular is
36:33
not pulling in these schemas for some reason but let's let's let's talk about
36:38
essentially how this stuff works right so and when you're in here right we're going to create um a spark session and
36:45
then we want to these are like uh essentially ways to describe
36:51
schema they look almost like a create table statement right that's what these look like they're little um SK my
36:57
definition so if you see here like for this event um uh case class you have uh
37:05
you have like user ID device ID refer host URL event time right and you'll see some of these are like string and some
37:11
of these are option integer so how it works is if you are trying to serialize
37:18
something into the data set API or like you're trying to convert um some data
37:23
into the data set API if it's nullable you have have to wrap it in option so if
37:29
you say like you can put like option string here if it's like a nullable um value and but if it's not nullable then

37:37
uh you can uh like you can just keep it as like the regular data type so this is
37:42
a great thing because now um I like just by using this case class I guarantee
37:48
that host URL and event time are never null because if they're null then the pipeline will fail and so that's like
37:56
pretty awesome actually uh it's one of those things that you can it it helps
38:01
enforce some of the assumptions around your pipeline so I like I'm I am going
38:07
to uh take some time after this like to actually get this example up and running for y'all but it actually doesn't run
38:13
right now but um I'm just going to kind of go over the different pieces so we have event we have device and then we
38:19
have event with Device info so all we're doing here is we're taking events and devices and we're going to join these
38:24
two tables together but here's the idea is we have um we read in um events right
38:32
as a CSV file and then you see this crazy as call here keeping in mind we
38:37
are in Scala here so um some of this might be very new to y'all so um but the
38:42
idea here is we read in a CSV and then we filter to where user ID is not null this is not actually necessarily needed
38:49
because this option up here lets user ID be nullable so then what we can do is we
38:54
can actually get rid of uh oh we want to keep that line but we can get rid of this wear here and then you'll see we
39:00
have uh this actually turns it into so how that works is then this is now a
39:05
data set of event that's the um the new data type of this um this is uh inferred
39:12
in Scola so you don't actually have to put the data type there but it's actually best practice to put uh to
39:17

label and type all of your um variables so that's going to be a data set of
39:22
event and then here you have a data set of device and so those are going to be
39:28
the two and then uh this is me trying to get this to work we don't need um we don't need these two
39:34
lines and so one of the things that this does is it actually gives you the
39:39
ability to work with the work with this data like in um Scala directly so for
39:46
example you see this filtered via data set so what you can do here is you can say do filter and then you can say event
39:53
and then you can say um event. userid do is defined you say and and event. device
40:00
ID do is defined and um this is uh yeah
40:05
I don't know why my spell check is pulling up here this is actually not like this is this is actually valid
40:11
syntax here but what this is doing right this is another way of oh this device ID
40:18
is defined this is essentially another way of checking to see if the user ID and the device ID are there and
40:24
filtering out those columns when they're not there um but the cool thing about this is like it gives you all those uh
40:31
it it allows you to easily pull that in because you can use the you can use that option class up here in Scala and that
40:38
allows you to easily uh kind of enforce that stuff and so what I wanted to show
40:44
here is these three um these three uh syntaxes
40:53
here they are the same these do exactly the same thing like like all three of
41:00
these data sets will be exactly the same so that's one of I just wanted to show you how like okay well it's um that's
41:09

what it is right so what I wanted to show now is I want to show the differences with how it looks with um
41:17
with a join because joins can be a little bit different I want to go down to this bottom one first because I think
41:22
this one is probably the the simplest right so with SQL right you the the way
41:28
you want to do it is you just do this line right basic every everyone in this
41:34
boot camp should know how to do that super easy um uh data frames little bit
41:40
little bit trickier right where you want to like uh you do your join and then uh you get this nice dollar syntax uh with
41:47
in in uh in scholis spark that I like I think you get it in py spark too I don't
41:53
know but like you can either do a dollar or you can do like call user ID like
41:58
that it's like the same thing like I kind of jump between the two depending on where I'm at
42:05
um uh so this is going to be the way to do it in the data frame um API uh so one
42:13
of the things that's nice about this is pieces of this can be abstracted you can actually break you see all these dots
42:19
each one of these dots could be like um in in its own function and you can kind of like cut up the data frame joins how
42:27
ever you want to cut them up so that's going to be one and then here is um and
42:33
I want to change this code a little bit this is has a little bit of my debugging in it but in here what we want is we're
42:38
going to say event. user id. getet and then here we're going to say device. device ID and then here we want to say
42:45
device. browser type and here we want to say um device. type here we say device. device
42:54
type and then here we want to say event ev. referer event. host and then event. URL
43:04
and this is event. event time so one of the things is is remember uh like in

43:09
here we have to manage null here because of some of these data are going to be
43:16
optional right so that's why um for this user ID we have doget here um but you
43:22
can also do essentially like a coals here so instead of doing doget you can do do getet orl else and then you can
43:28
put in like a minus one or like a like whatever default user ID you want to put in if you want a default one and so you
43:35
do need a manage null here because if you don't then this will map to a value
43:41
that is potentially null right we don't have to we can use we can with
43:47
confidence use doget here because oh well if we move this guy down here we
43:52
can so we can confidently use doget here because we are filtering out um the
43:58
nulles up here on this line so we know for sure user ID is never null after
44:04
this join so one of the things that's nice about this join here is you see how
44:11
uh you get access to the left and the right
44:16
um sides of the join here where you have uh event and uh device and these are
44:24
going to be from this data frame and this dat frame but then you have access
44:29
to their schemas and types here so it's very nice and uh easy to map everything
44:35
over so then that that's what this does this ends up doing the join and then mapping it to that new schema that
44:41
you're looking for and you'll see this schema actually has no options right because we are assuming that the none of
44:46
these columns are ever null so that's how you do a join with the data set API
44:53
um one of the things that I wanted to show here though is the data API is also really nice in terms of like doing a UDF
45:01

so imagine like you create a function here we're just going to create a function like uh we're going to call it
45:07
we're going to call it two uppercase because we're going to we're just going to call it that for now and we're going
45:12
to say we have a and it takes in a string right and then
45:18
uh it returns uh we'll say a string and so in this function right we just say
45:25
return um s do to upper case I know that this is uh probably
45:32
very um like repetitive and like kind of a topology here but one of the things I
45:38
wanted to show you is what you can do with this right is now with um say I
45:46
wanted to uppercase the browser type and the OS type so with um the data set API
45:52
you can call map again and then you can say uh case um and then you have row and
45:59
then event with Device info and then uh this gives you uh that
46:06
case and then you can just say um Val browser type we'll say upper browser
46:13
type upper browser type equals um two uppercase and this is going to be uh
46:20
row. browser type and then you say um uh row. browser
46:28
type equals upper browser type or we can even well actually we'll just do it that
46:34
way that and then you just want to say um uh return
46:39
row so one of the things that's really nice about doing it this way is this is
46:45
pure Scala right you see how like a deaf here is a pure Scola keyword so what you
46:51
have to do um I just want to show an example of how this works in the data frame API I'm not going to show it in
46:57
the spark SQL API because it's annoying um but um so in the data frame API how it works is you need to make another
47:03

thing here you're going to say Val 2 uppercase UDF and then you say equals UDF and then you say uh two
47:11
uppercase and then you put it's so weird you put like this like this is a great example of where Scala is just freaking
47:18
annoying but you put a weird underscore here is uh that that says that you're
47:23
passing the function as a value and uh essentially and so then then what you
47:28
can do is in here what you can do is you then wrap this uh line here and then
47:34
you'd say do as browser type and that's how you would use like a
47:40
UDF in the case of uh the data frame API so those are going to be your two ways
47:45
of doing it like uh they both like kind of have their uh benefits and their drawbacks uh I like um I personally
47:53
really like uh the doing it this way because then you don't have have like this extra UDF definition you just are
47:59
working in pure Scala and that can be a very uh powerful way to go about doing things obviously um this code right now
48:07
doesn't actually run but um I'm going to be fixing that soon um so anyways I'm going to step some of that back though
48:14
we will um I will come up with a slightly better example than that when I uh get the time but so I'm going to take
48:22
a pause here because obviously I covered a lot of Scala here and I covered a lot of um kind of information around case
48:28
classes and um spark stuff well that's this is kind of the idea behind um the
48:35
data set API keeping in mind that this is how um uh Airbnb does all of their
48:41
pipelines for the most part with this API with the data set API because you get all these nice like nullability
48:47
guarantees and not nullability guarantees um and so and you also one of

48:53
the other things that's nice is like so say I wanted to to uh create data right I can be like um um dummy data equals
49:02
and I can say like uh list and then I can be like um event I can say user ID
49:08
equals 1 device ID equals 2
49:13
referrer equals LinkedIn right host equals exactly.com
49:21
URL equals sign up event time equals
49:28
2020 30101 something like that right and then you can uh you can see how this is
49:35
really powerful at creating like your fake input data if yall remember from week um week uh three right you can have
49:44
uh you see how like we can create really good fake input data here but then this
49:50
schema is also used in the pipeline itself to actually um manage the data
49:56
right where it where it helps enforce some quality constraints on the data inside the pipeline itself so this is
50:02
another great way that you can increase your quality in your pipelines is by trying out this uh this kind of API and
50:09
Spark so that's why um Airbnb is all about it right they're all about um using this CU then you uh you get easy
50:16
access to stuff like this where then you can create fake data super easily and then from there you can uh like test
50:23
your pipelines a lot easier as well so that's um something that really cool and you don't have to make another named
50:28
Tuple or anything like that it's just already all in the class so um okay that's uh I'm going to we're going to
50:34
pause on this lab here and then we're going to shift gears to um another uh kind of topic because we have a couple
50:40

more things to talk about today uh we're going to go into uh yeah we'll go into caching and we'll check out caching next
50:48
so um you'll see um inside caching how this will work good news about this is
50:54
this is this one actually does run so so um what you can do here uh is what I
51:01
want you to do is uh we're just going to run this the first time I'm just going to hit run here um one of the things
51:08
here um for for this to run uh your kernel might need to change so there's a bunch of kernels here um we don't want
51:15
the iPie kernel because we aren't using um python here we want the Spy spon or
51:21
spyon kernel this is how you're going to get the um actual um Spark
51:27
data right and be able to get what you're looking for when in terms of spark so one of the things that we're
51:33
going to be going over here is like when uh we want to uh essentially so in this
51:41
case like we have uh what this pipeline is doing or this spark code is doing is
51:46
we have a list of users who have a bunch of devices but we also have a bunch of
51:53
devices that um so a user going to have many devices but a device can also have
51:58
many users in this data set there it's many to many both sides so what we're trying to do here is create both sides
52:05
like a user and all of their devices and then also a device and all of their users and so one of the things that we
52:12
do initially right is we have we we also want the uh like so we're we're doing
52:19
that with events aggregated right which gives us this guy here because we're going to essentially use this because we
52:25
don't want to do it at the event level we want to aggregate to the user ID and device ID level first so that then we
52:32

can uh go from there and then what we do is we take that and then we join it back
52:38
to itself and then kind of aggregate up and same thing here We join it back to itself and aggregate
52:43
up this pipeline is kind of uh trivial and one of the reasons for that is I
52:49
wanted to build a pipeline where we reuse um a data frame and in this case
52:55
you see how we have this events aggregated data frame in two spots here so what we want to do here is this gives
53:03
us a plan so what I want to do is um let's go to uh I just want to Google a
53:08
text I want to go to this text text diff thing because sometimes looking at these plans can um be uh a lot so we're going
53:17
to look at both of these plans but what I want to do is I'm just going to copy this first plan into the diff Checker
53:23
we're just going to keep him there and then what I want to do is we're going to do one small change to this code and
53:30
what we're going to do here is on events aggregated we're going to add do cache and then we're going to run this
53:40
again so now this is going to be our different uh you see this is our
53:46
different plan now for devices on device on events it's going to be this guy so
53:52
the main reason I'm doing this is to show y'all like how you can actually use something like this oh oh I missed the I
53:57
missed the first line there I need that physical plan line uh you can actually use something like this to um understand
54:06
how things are different right so let's go ahead and say find differences here so for a lot of this stuff the
54:13
differences are going to not actually be what you think they are right so um one
54:19
of the things you'll notice here right is okay um most of this is like just

54:25
numbers right it's like 228 47 see how that's all pretty much the same but then
54:31
once you get down to here this like you see this right here this hash aggregate exchange but you see it's different here
54:37
because we have in here we have um in memory table scan you see this uh this
54:43
is not um over here so what what that means is this table is actually going to
54:49
be um read from memory as opposed to being read uh from like the CSV file
54:56
right well this one does get read from the CSV file down here so but you see actually for this first
55:04
um this first uh query plan you see how like the the cached plan is actually uh
55:14
more right because which makes sense right because if you cache a data set
55:20
that is one more step and so like essentially this is what I'm trying to illustrate with with uh with using this
55:28
cach is you only get the benefit
55:33
from if you use it again right you have to use it two times and then uh then the
55:41
second time is when you're going to really see uh a lot more benefit because you only uh because the second usage is
55:48
going to be where it reads the result set and then it will uh not have to uh
55:54
it can just read the the C directly right so caching only really gives you a benefit in spark when you are reusing
56:03
stuff right so one of the things I wanted to show here as well is you'll
56:09
see up at the top here we have like this Storage level import so one of the things I wanted to show you is there's
56:15
also you can say Storage level here and then we can say um memory only that
56:21
works so this is another way that we can uh cash stuff there's two types of
56:27

there's two ways that you can cache things so if you use cache this is the default is Storage level memory only
56:33
there's persist so persist is a way where if you wanted to write stuff to
56:39
disk only this is a way to do it but remember like what I said about writing to disk is you might as well just dump
56:46
this to a table at that point so like if you're thinking about using do persist
56:52
disk the better thing to do here would be like events aggregated uh do um wr.
56:58
mode overwrite Dove as table and then we can say like event I would say like boot
57:06
camp. events aggregated aggregated staging and then
57:13
that is going to be this is the better play than using uh Dison persistence
57:18
right so uh because then you can look at the data right and like even if the job
57:24
fails or the job like uh like only gets halfway through you can see like what was written the staging so you can
57:31
understand like the Intermediate steps of the job so this is going to be a much better way to write your persistence
57:40
than um uh than using dis only right even though I've seen this I've seen this many many many times in my career
57:46
people doing it this way so really you want to just use do cach which Doh is the same as doing uh persist memory only
57:54
so this is the same thing here here and so you'll see uh there we go Oh cannot
58:02
find source for partition oh it's because it's it's upset this this uh this right line is upset we're going to
58:09
comment this line out it's like upset about that line I must have messed something up
58:14
there so this is now uh like the so this plan is going to be the same essentially
58:20
as it was uh before because we have the memory only persistence so this is ESS

58:26
enally the same way the to do the dot cach right so uh that so that's um
58:31
essentially how uh caching is going to work like the big big thing to remember
58:36
about caching is for caching you really really really want to be using the data
58:44
set more than one time because if you're not then there's no reason to cash it's
58:49
s it's similar to like caching with an API right where it's like caching really only happens after like uh a a certain
58:56
image or a certain thing has been like uh viewed many times because then caching makes the performance better but
59:03
like the first time the cold hit is never cached so and that's similar to
59:09
spark in that regard this example isn't the greatest example I'm going to be uh like working on getting getting youall a
59:16
better one but uh the whole idea here is just that right is and and and really
59:21
that's the main thing is to remember about caching is it you only want to use
59:27
it if you're G to be reusing data that's that's the main thing because otherwise there's no benefit so okay that is going
59:35
to be uh essentially the second big kind of area that I was looking at when I've
59:40
been kind of building out some of uh this this code here but let's go ahead
59:46
and go to bucket joints I think this one is going to be uh probably the most interesting and the most new for y'all
59:52
so um in this case uh we're going to be using different data sets here I have I
59:57
have matches and match details so these are going to be two um data sets that we're going to be working with today to
1:00:05
uh look at um um how to do bucket joins
1:00:10
in spark because I think that will be a really powerful thing for y'all to uh

1:00:16
understand as we kind of like go through this journey together so what we're going to be looking at is you'll see
1:00:23
here we have um a lot of commented out code but um I'm going to help yall with
1:00:29
like what to kind of look with look at first so uh initially here um uh it the
1:00:36
the stuff at the bottom doesn't really matter it might end up failing so what I would recommend to do first is just like
1:00:42
um uh comment this stuff out right because that's going to not be what we
1:00:47
want to work on but so you'll see initially here we're reading in two um
1:00:53
two data sets uh we have matches and match details and they have a shared key
1:00:58
they have a they have match ID that they can join on and so we're going to be looking at match ID to be the join key
1:01:05
for this um Iceberg table so what you can do right is um go ahead and uh we're
1:01:13
going to look at we're going to create the tables first let's just do that first um which is going to be uh
1:01:19
uncommenting this these lines and uncommenting these lines and this is going to give us uh let's go over each
1:01:25
line here very carefully to kind of understand what's going on here so we have matches bucketed and you'll see
1:01:32
here um uh we have our schema makes sense and then you have to say okay
1:01:37
we're going to use Iceberg and then we have our partition keys and in this in this case we're going to be partitioning
1:01:42
on date and we're going to partition on buckets those are going to be the two that we're going to want to partition on
1:01:49
um and then that will give us uh our ability to work with this data right and
1:01:55
like be able to uh we be able to kind of partition out this data and then how we

1:02:01
actually um write out the data matters to so in this case uh this is going to
1:02:08
be how you write out the data into the buckets so that Iceberg can do it the right way is so like you have like your
1:02:15
columns obviously and then you um have your append mode uh because I couldn't
1:02:22
get overwrite to work here that was actually something I'm I'm I I need to dig a little bit deeper into but overwrite here actually does not work a
1:02:29
pen does but overwrite does not and then uh we want to say we want to P Partition by date and then we have our bucket um
1:02:37
we want a bucket on um 16 16 buckets by match ID which is very similar to uh
1:02:44
this bucket 16 up here and then we want to save the table then down here we do a similar
1:02:50
thing but in this case uh this table actually doesn't have completion date but it does have U match ID so we can
1:02:57
bucket on just that and that we should be good to go in terms of uh matching up
1:03:03
uh the buckets to each other from there then what we want to do is we want
1:03:08
to do the same thing here where we are writing out you know various columns to
1:03:14
the we're essentially just we're moving uh the data from the CSV file into a
1:03:19
bucketed Iceberg table that's kind of the idea here and then this should give
1:03:24
us uh data that we're looking for I think we have to do like if exists oh well yeah there's oh oh there
1:03:32
we go mine just mine just mine just freaking worked right uh I I got mine to
1:03:37
just it actually just ran okay awesome so now um I want to go back into the other um uh yeah that was that was so
1:03:45
crazy so I want to go back into the other um place where I can query uh with
1:03:50
SQL so I just want to show how this works right so you'll see in when we're in the

1:03:57
match details bucket in here there should be um 16 files total right and uh
1:04:04
you'll see we have like here's and you'll see the partitioning here we have partition match ID bucket one right and
1:04:11
then it it has all sorts of uh different um uh values like because it keeps track
1:04:18
of this stuff for you right of like the highend and the low end of the um match
1:04:23
IDs so then you know like when you're uh joining things that it will be a lot
1:04:30
more performant right so that's essentially what I wanted to show with um this right so and but I wanted to
1:04:37
show the other one though so there's also matches bucketed so there's match match details bucketed and matches
1:04:43
bucketed and you'll see here though it's different because now in here we have um
1:04:49
we have our date time right uh field here which is like we
1:04:55
have our date date kind of our date time here and that's going to be how we are um and in in this case we're going to
1:05:01
have many many many more files you see how this just keeps going and keeps going there's going to be a lot more files here because it's partitioned on
1:05:08
um like how many files are here there's like a lot oh yeah there's 3,000 files right because it's like 16 times the
1:05:16
number of days and that will be the number of uh files that would be for um
1:05:21
matches because it has that second partitioning in it right and so I actually got the partitioning and Iceberg stuff to work today so this is
1:05:28
kind of the idea right is so now I'm so happy that finally actually ran so we
1:05:34
got that working and what I'm going to do now is I'm going just comment all this out for now because we I I don't this was like really slow so what I want
1:05:42
to do now is I want to show the differences between these two right so

1:05:47
what we want to look at here is oh yeah these these these explains down here we don't care about but um like you'll see
1:05:55
um in here after we write out I have these lines here it's like we have the matches bucketed create replace match
1:06:02
details bucketed Creator replace Temp View those are just the CSV files that are actually not bucketed and then down
1:06:08
here you'll see this query here is querying the iceberg this is actually going to be querying the iceberg and
1:06:15
these two are going to be a little bit different one of the things I want to add to this though at the top here is I
1:06:21
want to add this um broadcast join threshold the main reason I'm doing that is because the data here is not very big
1:06:28
and what if you put it at minus one what that does is it actually disables a broadcast join so I want to actually see
1:06:35
if this is going to use a bucket join because bucket joins really are only
1:06:40
necessary when you have very large data and so like this would default to a broadcast join if we uh if we actually
1:06:47
just let it so that's why I'm doing this to show y'all the difference we want to compare the differences in these two
1:06:54
plans to see like okay bucketing actually makes a difference right that's the whole
1:07:00
idea okay so now what you see here is we
1:07:05
have our two uh plans here right and one of the things that you'll notice is uh
1:07:11
here's that first plan and you see the first plan here is from this uh um from
1:07:17
this one right the this one's going to be a lot uh um you see how this one has
1:07:23
like uh like you see the the plan here I don't know if y'all noticed but this plan is like a lot less than this plan

1:07:32
like it's a lot less so one of the things here is you'll see that we have a
1:07:37
lot more exchanges here like we have this like exchange here right and then we have uh you have the exchange here
1:07:45
and so uh this this one is going to be this is going to this is going to shuffle right this is going to shuffle a
1:07:52
lot right whereas um you see here this one is not going to shuffle like do you
1:07:58
see exchange anywhere um so it still does the sort merge join because it has to do the sort
1:08:06
merge join that but that's within the bucket not um not uh sort merge join um
1:08:14
with from the partitioning so this will um essentially do exactly what you're
1:08:20
saying like because you see like one of the things I just hope youall notice here is just like when you have things
1:08:26
bucketed like this the um the plan is just it's it's really tidy right and it
1:08:32
has and so the the whole idea here that I'm trying to show though is if you bucket
1:08:37
things you don't have to uh um you don't
1:08:42
have to shuffle there's no Shuffle Shuffle is not in there anywhere see because you see batch scan what it means
1:08:49
by batch scan is like it's like a bucketed scan that's what it means by that right and then that's how it goes
1:08:55
up into a sort merge join so this is going to be like especially at um a very
1:09:00
very big scale this type of join is going to be dramatically more efficient
1:09:06
than this type of join down here so um that and this idea of like putting
1:09:13
things in the buckets this is exactly um one of the most important things that I
1:09:19
did to uh increase the performance of my
1:09:24

jobs right when I been working in big Tech and everything like that so uh yeah
1:09:29
um it I'm going to um going to take a pause here and I'm just going to I want to you know take any questions if anyone
1:09:36
has any thoughts or questions on things uh I'm definitely down to answer them because uh there's there's a lot going
1:09:41
on here for sure and because these but these bucket joins are very important very important and they uh like are
1:09:47
really awesome if you can get them to work the right way I hope you all have had a great time with this boot camp uh
1:09:53
and this has been a magical experience for me and uh I'm Gonna Miss This congrats on getting to the end of
1:10:00
the advanced spark lab who that was a lot right make sure to like comment and subscribe to support this channel so I
1:10:07
can keep making more educational content and make sure to check out the rest of this boot camp you know there's a lot
1:10:12
going on here

English (auto-generated)

All

From the series

From Data with Zach

Data science

Software Engineering

APIs

Application software