

This assignment does not count toward the final grade.

Assignment 0

New Attempt

- Due 9 Jan by 14:30
- Points 1
- Submitting a text entry box

This is your pre-class assignment. There are no points at stake but I want you to be able to start compiling and running Scala, as well as using the REPL before class begins.

Part 1

First, make sure you have a Java 11 JDK (or later). Load the *required* Scala tools listed in Resources. The assignments will all be based on 2.13.14 or 2.13.15 (either work fine) so that's the version I would like you to install.

Once you have done all that, run the "REPL" (type "scala" into your command line) and enter the following:

```
scala.util.Random.nextInt
```



Note the result and be ready to submit it once you have everything. The assignment is due before the first class. If you are having trouble with the REPL, you can do the same thing with the IDE, Scastie or Scalafiddle (see Resources).

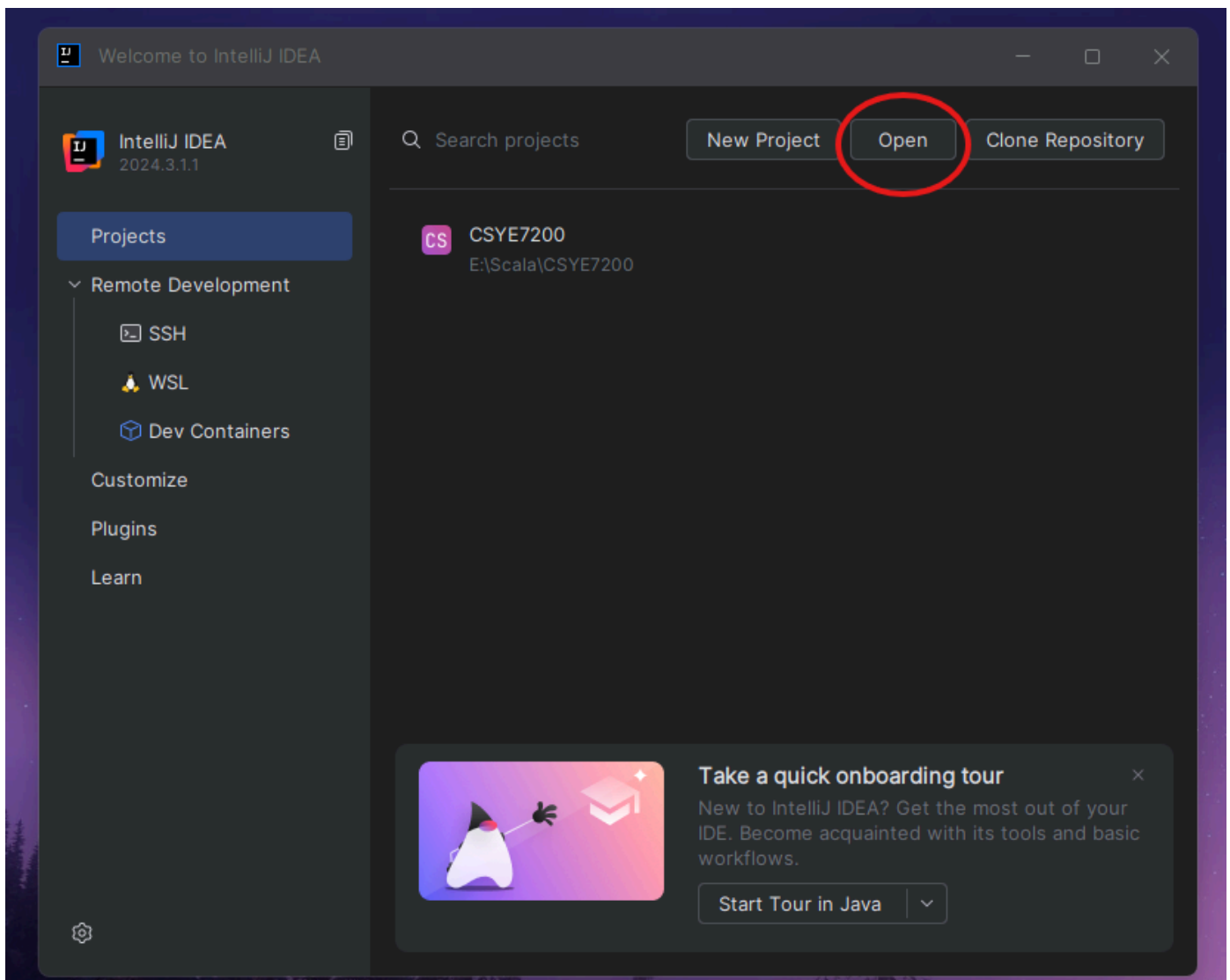
For the IDE, I strongly recommend IntelliJ/IDEA. You may be more familiar with Eclipse, but the Scala support in Eclipse has dropped off recently. "Everyone" uses IntelliJ (although feel free to look on the Scala web site--see below--to choose a different IDE such as Metals--but if you do that, you're on your own for help with building stuff).

In any case, all the information you need for installing and using Scala is at this URL which you should bookmark: <https://www.scala-lang.org>  (<https://www.scala-lang.org>).

Part 2

Now, I want you to do a little coding in your IDE. For that, follow these sequence of steps:

- Download IntelliJ (if you haven't already) using the download link: [Jetbrains IntelliJ](https://www.jetbrains.com/idea/download)  (<https://www.jetbrains.com/idea/download>).
- Clone the course repository located here: [CSYE7200 repository](https://github.com/rchillyard/CSYE7200.git)  (<https://github.com/rchillyard/CSYE7200.git>).
- Open IntelliJ and open the cloned course repository through IntelliJ



- Let IntelliJ configure Scala version and relevant JDK version. These are the recommended versions:
 - Scala: 2.13.15 (In IntelliJ, File > Project Structure > Global Libraries > Click on "+" icon > Scala SDK > Download > Version: 2.13.15 or 2.13.14 > Click OK)
 - JDK: Java 11+ (Amazon Corretto distribution preferred) ([Instructions to change Java version](https://www.baeldung.com/intellij-change-java-version)) (<https://www.baeldung.com/intellij-change-java-version>)
- Now, go and do the same thing with the IDE itself by right-clicking on the name of the HelloWorld module (in the project view), and choosing Run HelloWorld. If this Run option doesn't show up, it means that you haven't built the module yet. This normally happens automatically, but if it doesn't the IDE is quite helpful with diagnosing the problem. You may need to explicitly set up the JDK and/or the Scala SDK. (follow above)
- Next... go to the Ingest.scala (**assignment-helloworld/src/main/scala/edu/neu/coe/csye7200/assthw/Ingest.scala**) file in the same directory as HelloWorld and run it in the same way. You should see a very long listing of Movie database data.
- Open the Ingest.scala file) and update the file based on the below instructions:
 - The line 38 in the file currently looks like:


```
for (m <- ingester(source)) println(m.properties.mkString(", "))
```
 - Replace that line with the following code:


```
val kiwiMovies = for (m <- ingester(source); if (m.properties(20)=="New Zealand")) yield m
println(kiwiMovies.size)
```
- Run the Ingest.scala file, by right-clicking in the code editor and click on "Run Ingest.main()" option
- Take a screenshot of the output in the run window and upload it to the submission.

Note:

- Note that you are now using a "for comprehension" with "yield." This time, it will just give you the number of movies produced in New Zealand.
- Note that number and, together with the random number that you got in the first part, submit your answers.
(Submission can be screenshots of the outputs)

Enjoy!

Assignment 1 (Movie Database Part 1)

[New Attempt](#)

- Due 14 Jan by 11:45
- Points 25
- Submitting a text entry box or a file upload

Please note that Assignment 1 is basically the same as Assignment 0 except that you are to submit your code according to the instructions (see below). But, everything that you did in the second part of Assignment 0 should be done again here. If you already did Assignment 0, that should make it very easy.

You will find the basic source code for Assignment 1 in the class repository under the module: *assignment-helloworld*. You should just clone the whole repository and then "cd" into the *assignment-helloworld* module.

There are three separate parts to this assignment: *HelloWorld*, *Ingest*, and *Spark*. The first two of these are a "main" program that you can run either directly via your IDE (IntelliJ IDEA or Eclipse) or you can run using "sbt run." (You should be able to do that if you successfully finished assignment 0).

For the third part of the assignment, I'd like you to download the current version of Spark (see [Resources](#) (<https://northeastern.instructure.com/courses/206870/pages/resources>)). Then, using the spark-shell, run a simple Scala program with Spark, for example CountWords. You can find such examples either at the Spark site, or generally on the web. You could also use Zeppelin or the Spark Notebook to do this--anything that allows you to run Spark programs interactively.

The instructions for Assignment 1 are mostly covered by the Assignment 0 instructions. However, if you need further detail on the requirements for assignment 1 is available as Assignment1 here (except you should use Scala 2.13, not 2.12 and Java 18.0.2): Also, ignore the bit about downloading the dataset from kaggle. Instead, use the movie_metadata_5000.csv that you find in the repository.

[Assignment 1-1.pdf](#) (<https://northeastern.instructure.com/courses/206870/files/31819763?wrap=1>) 
(https://northeastern.instructure.com/courses/206870/files/31819763/download?download_frd=1)

Please follow the guidelines for submitting assignments posted under the Modules/Assignments folder.

Assignment 2 (Lazy)

[New Attempt](#)

- Due 11 Feb by 11:45
- Points 50
- Submitting a text entry box or a file upload
- Available after 4 Feb at 13:30

Assignment 2 is designed to test your understanding of the second week of CSYE7200 lectures, in particular, Scala's mechanism for dealing with lazy values. Furthermore, working through the assignment, you will understand that features like *LazyList* in Scala are not "magic." We can create our own, simply using a function for the lazily-evaluated tail (*MyLazyList*).

You can find the code under the *assignment-lazy* directory in the REPO. Make sure you do a pull first.

You are to submit evidence of all unit tests (in *MyLazyListSpec*) passing (use a screenshot). In order to get the tests to succeed, you must implement the *from* method in the companion object of *MyLazyList*. Replace the *??? //...* with your own working code. There are plenty of other examples of similar functionality in the *MyLazyList* module. Submit, along with your unit test screenshots, the expression that you used to implement the *from* method.

Additionally, I want to see answers to the following questions:

1. (a) what is the chief way by which *MyLazyList* differs from *LazyList* (the built-in Scala class that does the same thing). Don't mention the methods that *MyLazyList* does or doesn't implement--I want to know what is the *structural* difference.
(b) Why do you think there is this difference?
2. Explain what the following code actually does and why is it needed?

```
def tail = lazyTail()
```
3. List all of the recursive calls that you can find in *MyLazyList* (give line numbers).
4. List all of the mutable variables and mutable collections that you can find in *MyLazyList* (give line numbers).
5. What is the purpose of the *zip* method?
6. Why is there no *length* (or *size*) method for *MyLazyList*?


LazyList rubric

Criteria	Ratings	Pts
1.a. Structural differences between LazyList and MyLazyList		5 pts
1.b		3 pts
2		6 pts
3		6 pts
4		4 pts
5		5 pts
6		5 pts
Code		16 pts
Total points: 50		

Assignment 3 (Movie database part 2)

[New Attempt](#)

- Due 18 Feb by 11:45
- Points 50
- Submitting a text entry box or a file upload
- Available after 11 Feb at 13:30


There are **three** implementations to create in *Movie.scala*: the *parse*, *elements*, and *Rating.apply* methods, (approximately lines 101, 124, 204) and **one** in *IngestSpec.scala* (approximately line 24). [This time it is OK to edit the Spec file (but after this assignment, you should not edit any Spec files)]. But, in any case, I want to give you some familiarity with Spec files (i.e. ScalaTest with *AnyFlatSpec* and *Matchers*). The most useful documentation for this sort of thing is here: http://www.scalatest.org/user_guide/using_matchers  (http://www.scalatest.org/user_guide/using_matchers).

The purpose of this code is to read the movie database file into a sequence of *Movie* instances.

The module name for this assignment is *assignment-movie-database*. Let me clarify a little: you only need to work with *Ingest.scala*, *Movie.scala* and *IngestSpec.scala*, in addition, *MovieSpec.scala* is available for you to verify your implementation by running its unit tests. These are all in the *asstmd* package (the spec file is under test of course). You will also need *movie_metadata.csv* which is under test/resources of the project (use this version).

You don't need to create a new project, just clone/download class repo, and import assignment-movie-database, you may follow the video demo under *Canvas / Assignments / Video Demo for Assignment Import and Submit*. Remember the to follow the *Canvas / Assignments / Standard procedure for submitting assignments*.

Don't worry if some of the movie entries do not parse properly. That's to be expected. But the New Zealand productions (Kiwi movies) should all parse OK.

You will have to do some real programming to create these implementations and you may feel that you aren't quite ready for it. But, the compiler is your friend. The REPL is your friend. **You can do it!** And the principle of **Simple, Obvious, Elegant** is your friend too (I particularly commend my [blog article](http://scalaprof.blogspot.com/2015/11/simple-obvious-elegant.html)  (<http://scalaprof.blogspot.com/2015/11/simple-obvious-elegant.html>) on this).

Assignment 4 (Random State)

[New Attempt](#)

- Due 13 Mar by 11:45
- Points 51
- Submitting a text entry box or a file upload
- Available after 4 Mar at 11:00

We have talked about the importance of having a service of some kind able to create a new state rather than mutate the existing state. Here you will create a random number service that is modeled as a state. Please see my blog on this subject for background: [Working with mutable state \(http://scalaprof.blogspot.com/2016/09/working-with-mutable-state.html\)](http://scalaprof.blogspot.com/2016/09/working-with-mutable-state.html).

We need to create a trait called *RandomState* which will have two obvious methods: *next* and *get*. Of course, we don't really know what the type of the result of *get* will be, so let's make it parametric, thus: *RandomState[T]*.

But once we have a *RandomState[T]*, we will want to be able to map it into a *RandomState[U]* so we'll need to implement *map*. While we're at it, we might as well implement *flatMap* too. Technically, this will mean that it's a "monad" but we haven't talked about those yet -- but they are important.

There's one other convenience method that we should probably implement and that is *toStream* which will return a *LazyList[T]*. As usual, I have provided the basic framework and a specification for your work: *src/main/scala/edu/neu/coe/csye7200/asstrs/RandomState.scala* and the corresponding *RandomStateSpec* in the *test* directory. All you have to do is to implement the 6 *TO BE IMPLEMENTED* and run the tests. When it's all green, you're done. You can get these from the class repo (see *Course Material/Resources/Class Repository*), the module name for this assignment is *assignment-random-state*.

However, we are going to be getting a little deeper into functional programming in this assignment and some of the concepts will be new to you. You will have to stretch your minds (again) and think out of the box a little. I don't want you to do this assignment in pairs. It's sufficiently important that I want each of you to work on it alone. You will benefit from that when it comes to the mid-term exam.

I want to commend to you another one of my blogs which we talked a little about already. Read it. I do believe it will help you materially in this assignment. It is here: [Simple, obvious, elegant \(http://scalaprof.blogspot.com/2015/11/simple-obvious-elegant.html\)](http://scalaprof.blogspot.com/2015/11/simple-obvious-elegant.html).

Also, there are a couple of useful methods defined on *Function1*: *compose* and *andThen*. Keep that in mind as you work on your implementations.

One further piece of advice: try the easier implementations first (the number of points available is more or less proportional to the degree of difficulty). You will get the hang of it as you go.

You don't need to create a new project, just clone/download class repo, and import *assignment-random-state*, you may follow the video demo under *Canvas / Assignments / Video Demo for Assignment Import and Submit*. Remember to follow the *Canvas / Assignments / Standard procedure for submitting assignments*.

Assignment 5 (Functional Composition)

New Attempt

- Due 20 Mar by 14:50
- Points 85
- Submitting a website url or a file upload

This assignment builds on what we already did for the Movie database in order to test what you learned recently about functional composition, *Option*, *Try*, for comprehensions, etc.

I had a lot of fun with this assignment, and a little frustration at times. But that's to be expected any time you are doing something challenging. If you follow the mantra Simple, Obvious, Elegant, you won't have too much trouble. As noted above, I've actually done the really hard stuff for you (that's all in the *Movie.scala* file).

OK, to business...

All the work you are going to do is in the *asstfc* package. *Ingest* hasn't changed, but *Movie* (and *MovieSpec* -- also *inasstfc*) have.

There are 13 TODOs in *Function.scala* and 2 TODOs in *Movie.scala*. You can get these from the class repo (see *Course Material/Resources/Class Repository*), the module name for this assignment is *assignment-functional-composition*.

Did you find it at all bothersome (in Assignments 1 and 2) that we didn't use *Try* (or *Option*) to deal with some of the *String => Int* conversions? Some of those invocations of *toInt* might easily have thrown exceptions. Part of the problem was that it was too difficult to do, given your knowledge of Scala at the time.

But now, you know all about functional composition, for comprehensions and monads.

The basic problem that I had to solve is that some of the *apply* functions (for example in *Format*) have a mixture of strings, as well as *Int* and *Double* which require conversion from *String*. One of the elements of a *Movie* (viz. *Reviews*) was based on seven *Int* conversions from seven strings. That's easy: just use *map7* (!). Of course we didn't actually implement *map7* in class, but you're going to implement it in this assignment. But what about mixtures like in *Format*? That required a rather creative solution (if I may say so). Take a look at the code in the object *Format* and also in the object *Production*. First, we need to curry the apply method. Functions in curried form are much more tractable than functions in tupled (normal) form. Now, the rest of the code transforms that curried function into a function that can be used in a for comprehension. Don't worry too much if you don't understand what's going on here. But, I believe that if you take the time to figure it out, you can see what's happening.

In any case, this code uses some utilities that I placed in the *Function* object.

Now, here's an important hint for the one *TODO* that is in the *Movie.scala* file: Back in assignment 2, we had a *for comprehension* which utilized a filter (you can see that by taking a look at *Movie.scala* in assignment-movie-database which is unchanged). At the time, we hadn't talked about for comprehensions but hopefully you figured it out (it is kind of obvious).

But do you remember in a recent lecture that I said that the left hand side of a generator (in a *for comprehension*) is actually a pattern? Yes, it's a pattern just like in the case statement of a match expression. Neat, huh? So, you are going to implement the filter on the country value by using a pattern in the *for comprehension* instead of a filter (thus you will not be using the method *isKiwi* or anything like it). Do take a look at how it was implemented (using a filter) in our previous version.

But there's one tricky bit of syntax which we haven't covered yet. In Scala there can arise a certain ambiguity when you have identifiers that are the same as keywords (not allowed in Java) or, as here, you have a variable mentioned in a pattern matching context where you don't want just to match on anything (you would be "shadowing" the variable in such a case). You actually want to match on the value of the variable. For this (and the similar case where you want to use a keyword as a variable), just enclose the variable name inside back-ticks the ````` character.

Good luck!

Assignment 6: Web Crawler

[New Attempt](#)

- Due 3 Apr by 14:50
- Points 42
- Submitting a website url or a file upload
- Available after 27 Mar at 16:30

Implement the primitive web crawler that is partly complete in the crawler package on our class repo (you must use the Spring2025 branch).

There are three *TO BE IMPLEMENTED* to complete, with a total point value of 32. You may also earn up to 10 bonus points for suggestions on how to improve the web crawler (detailed code not required but you do need to explain in words what you would do). Two of these are in *WebCrawler.scala*. The other is in *MonadOps.scala* as follows:

Please ensure that you pull the latest versions of *WebCrawler.scala*, *HTMLParser.scala* and *MonadOps.scala*. You can get these from the class repo (see Course Material/Resources/Class Repository), the module name for this assignment is **assignment-web-crawler**.

For the processing of a Node, we will need to refer back to the way Scala processes XML documents which we covered in Serialization (that's the purpose of the tagsoup library). Hint: you can get all of the anchor, viz. the "a" nodes using `ns \ "a"`

You can get the "href" property from these nodes using `"\"` and `"@href"`.

There is also the main program but if you run that, you will need to provide Program arguments consisting of URL(s) at which to start crawling.

I strongly advise you to take advantage of the various hints. Also, make sure you know the types of any intermediate results that you derive: either by adding type annotation (in IDEA, just do option/alt + return/enter and it will give you the option of adding a type annotation) or by just selecting an identifier and displaying its type (in IDEA, that would be ctrl/shift/P). Any time you know the type you're starting with--and you know the type of result you need--now you just have to find existing methods to convert from one to the other.


Assignment 7: Analyzing Movie Rating

New Attempt

- Due 10 Apr by 14:50
- Points 50
- Submitting a website url or a file upload
- Available after 3 Apr at 16:35

You are required to analyze a movie rating dataset. The data is stored in a CSV file (either use the one in the repository or download the latest from Kaggle). You need to read this file into spark and calculate the mean rating and standard deviation for all movies. There is no test case provided for you, so you need to write your own test cases to ensure that at least your program works well.

You can refer to *WordCount.scala* file for the basic structure. Notice that you need to use specific Spark version 3.2.1 for Scala 2.12.x support. If you can run it using a version with Scala 2.13, then go ahead.

Note also that there is a module in the CSYE7200 repository called *spark-csv*. If you use that, you will have to edit the *build.sbt* file. You can use that code of course to get started with (show your results if you do this). **However**, you also need to read the CSV file using the Spark utility (see <https://spark.apache.org/docs/latest/sql-data-sources-csv.html>  (<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>)) and then create a method that accepts a DataFrame and returns the processed DataFrame.


You need to provide your code (in your own repository) together with the mean/std. dev. Don't forget to say where exactly you got the CSV file from.

Spark Assignment 1

New Attempt

- Due 27 Jan by 11:45
- Points 100
- Submitting a file upload

Create an account in Kaggle and download the Titanic Dataset.

Url: <https://www.kaggle.com/competitions/titanic/data> 
(<https://www.kaggle.com/competitions/titanic/data>)

For this assignment you will use the training dataset. training.csv

You are to load the dataset using Spark and perform the operations to answer the following questions.

1) What is the average ticket fare for each Ticket class?

(The ticket class is the pclass column. 1st = Upper; 2nd = Middle ; 3rd = Lower. Find the average cost of the ticket fare for the 1st class, the 2nd class and the 3rd class)

2) What is the survival percentage for each Ticket class? Which class has the highest survival rate?

(survival rate = number of survived per class/ total number of passengers . In the survival column, 0 = dead and 1 = survived)

3) Rose DeWitt Bukater was 17 years old when she boarded the titanic. She is traveling with her mother and fiancé(they are not married yet, so they are not related). She is traveling first class. With the information of her age, gender, class she is traveling in, and the fact that she is traveling with one parent, find the number of passengers who could possibly be Rose. (PS: if you watched the movie you will know if she survived or died)

4) Jack Dawson born in 1892 died on April 15, 1912. He is either 20 or 19 years old. He travels 3rd class and has no relatives onboard. Find the number of passengers who could possibly be Jack? (PS: Yeah he's the guy who gets Rose)

5) Split the age for every 10 years. 1-10 as one age group, 11- 20 as another etc.

What is the relation between the ages and the ticket fare? Which age group most likely survived ?

Spark Assignment 2

[New Attempt](#)

- Due 27 Mar by 14:50
- Points 100
- Submitting a website url or a file upload

Url: <https://www.kaggle.com/competitions/titanic/data> 

[\(https://www.kaggle.com/competitions/titanic/data\)](https://www.kaggle.com/competitions/titanic/data)

For this assignment you will use the training and testing datasets.(train.csv & test.csv)

You are to load the dataset using Spark and perform the operations below:

Exploratory Data Analysis- Follow up on the previous spark assignment 1 and explain a few statistics. (20 pts)

Feature Engineering - Create new attributes that may be derived from the existing attributes. This may include removing certain columns in the dataset. (30 pts)

Prediction - Use the train.csv to train a Machine Learning model of your choice & test it on the test.csv. You are required to predict if the records in test.csv survived or not. Note(1 = Survived, 0 = Dead) (50 pts)

Please note: Do not include the test.csv while training the model. Also do not use the 'Survived' column during training. Doing these would defeat the purpose of the entire model.

The classifier must have an accuracy of 70 % for 100 pts.

The submission must be on a new github repository or databricks public notebook along with the Assignment pdf. Please provide Screenshots of the results in the Assignment.pdf for submission.

Assignment - Working with APIs

[New Attempt](#)

- Due 27 Mar by 14:50
- Points 100
- Submitting a file upload
- Available after 20 Mar at 16:30

You may use Scala requests to get the Spotify Playlist data in JSON format.

The ID of the playlist is : 5Rrf7mqN8uus2AaQQQNdc1

The Spotify developer API is : <https://developer.spotify.com/console/get-playlist/>
(<https://developer.spotify.com/console/get-playlist/>)

Playlist : <https://open.spotify.com/playlist/5Rrf7mqN8uus2AaQQQNdc1>
(<https://open.spotify.com/playlist/5Rrf7mqN8uus2AaQQQNdc1>)

In the JSON response, you will have the duration_ms field. This is the duration of each song in milliseconds. You are to find the top 10 longest songs. Each song will have an Artist name and an artist ID. There may be multiple artists for the same song. You are to query the artist details of every artist in the top 10 longest songs and display them in order of their followers.

The Spotify API to query the artist is : <https://developer.spotify.com/console/get-artist/>
(<https://developer.spotify.com/console/get-artist/>)

You must call this API for every artist ID in the top 10 longest songs.

The expected output would be :

Part 1)

Songname1 , duration_ms

Songname2 , duration_ms

.....

Songname10 , duration_ms

Part 2)

Artist1 : follower_count

Artist2 : follower_count ... etc

All code must be in Scala.