Operational Analytics

Version 1.3 Last Updated: 11.17.21 Last Updated By: Sugandan B

Section A - Workshop Preparation

1. Download Student Guide and Presentation Deck
2. Decide Which AWS Account to Use in this Workshop
3. From the AWS Console, login with your root account and perform admin tasks: i.
   Create an AWS IAM user
4. Launch your own SingleStore Fully Managed Cluster
5. AWS ETL Glue with Spark Prep
6. Launch an AWS CloudShell Service

Section B - Building an Operational Analytics Solution (Build Phase)

Step 1 - Create Database and Load Data

Step 2 - Deploy a Real-Time Analytics SingleStore Stored Procedure

Step 3 - Leverage AWS Glue ETL with SingleStore Spark Connector

Step 4 - Bring it all together with Looker Dashboard

Cleanup - Terminate AWS Resources

Appendix

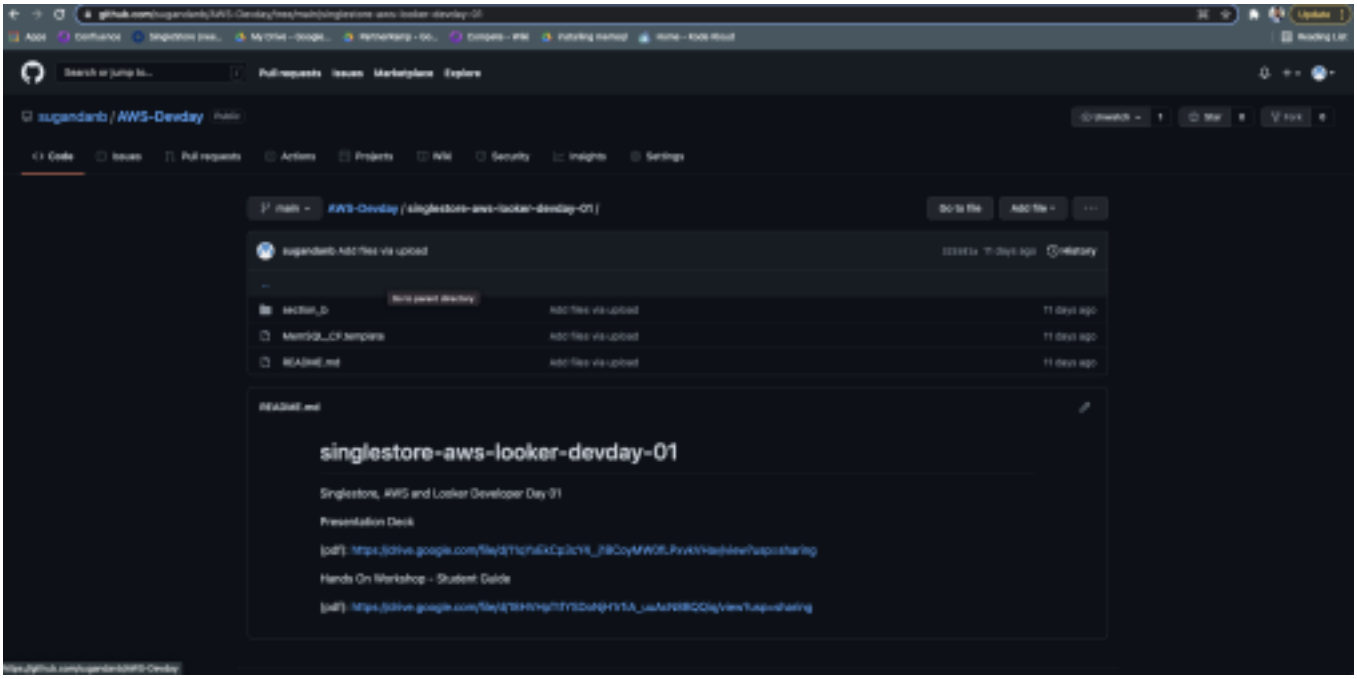
Extra Credit Homework

Sagemaker - Build, Train, Deploy anomaly detection model

Section A - Workshop Preparation

1 - Download Student Guide and Presentation Deck

[Dev Day Github Link]



Scroll down to the Readme.md section on the main page and download the following two items for this workshop. Make sure you have both of these assets available and open on your local machine.

(1) PRESENTATION DECK

This deck will be used by your instructor and it will provide you with a high level understanding of each step below.

(2) STUDENT GUIDE

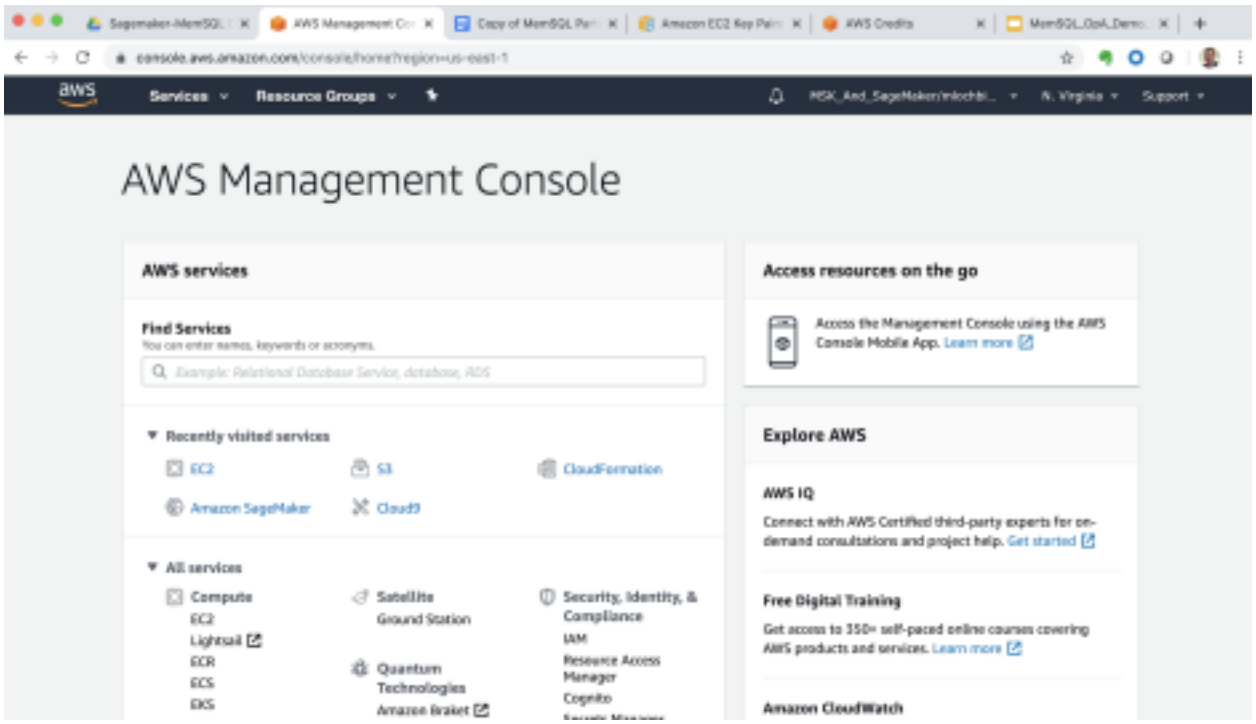
This guide will provide you with detailed instructions for each step below.

2 - Decide which AWS account to use in this workshop

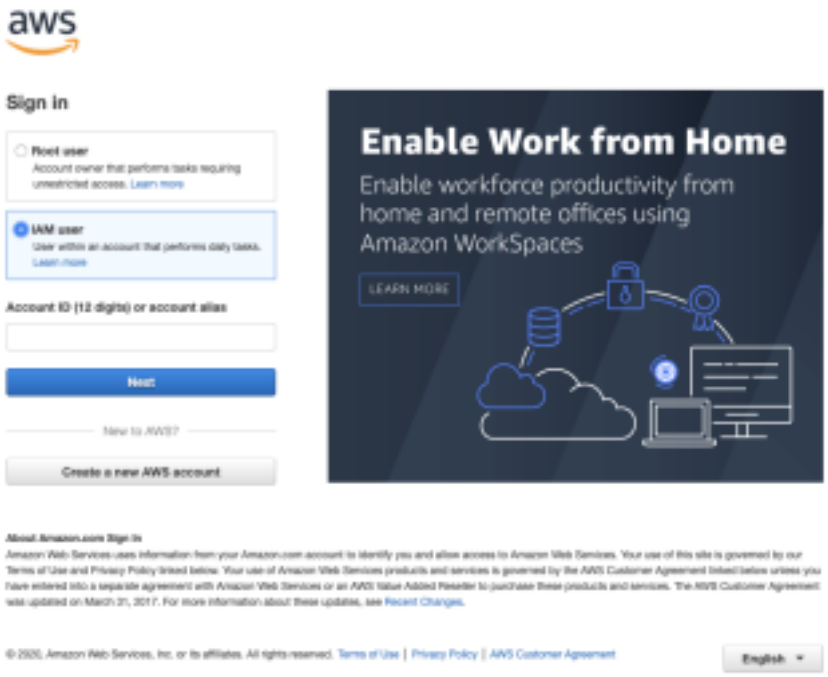
The following AWS Services need to be accessible to your AWS Account to complete this workshop.

Amazon S3, Amazon SageMaker, AWS Glue and AWS CloudShell

Here is an example of a student's AWS Console below, showing recently visited services.



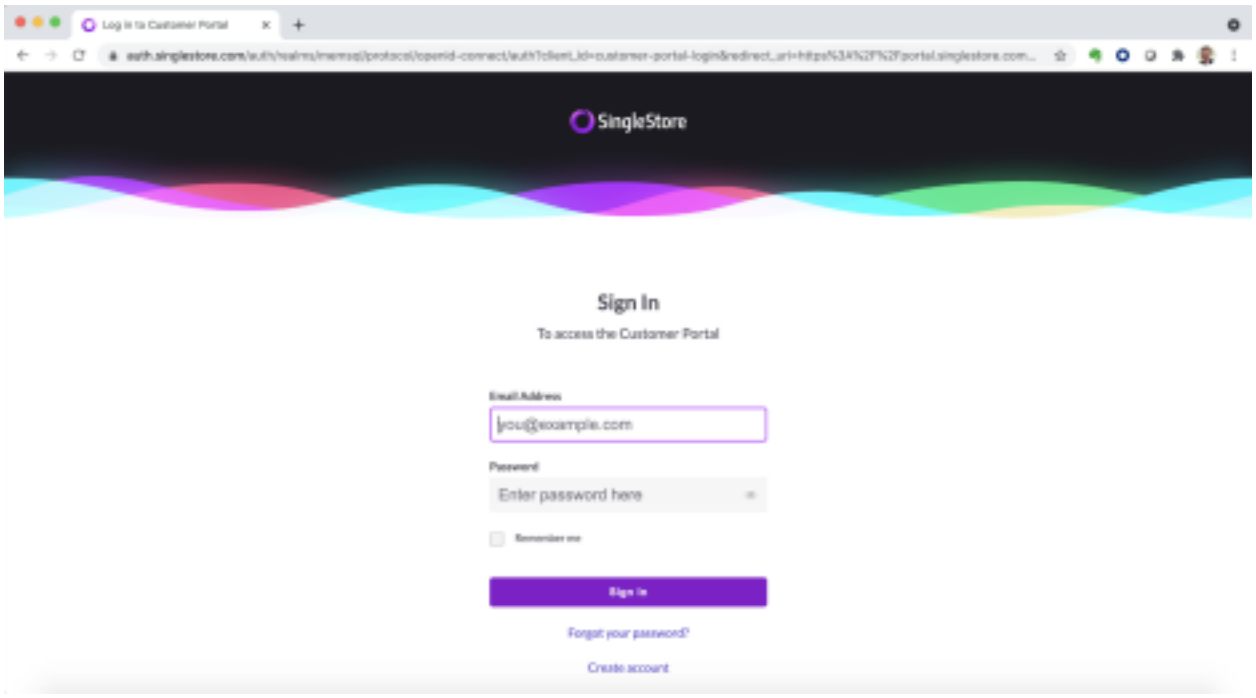
You will need to decide at this time which AWS account you will be using during the remainder of this exercise.

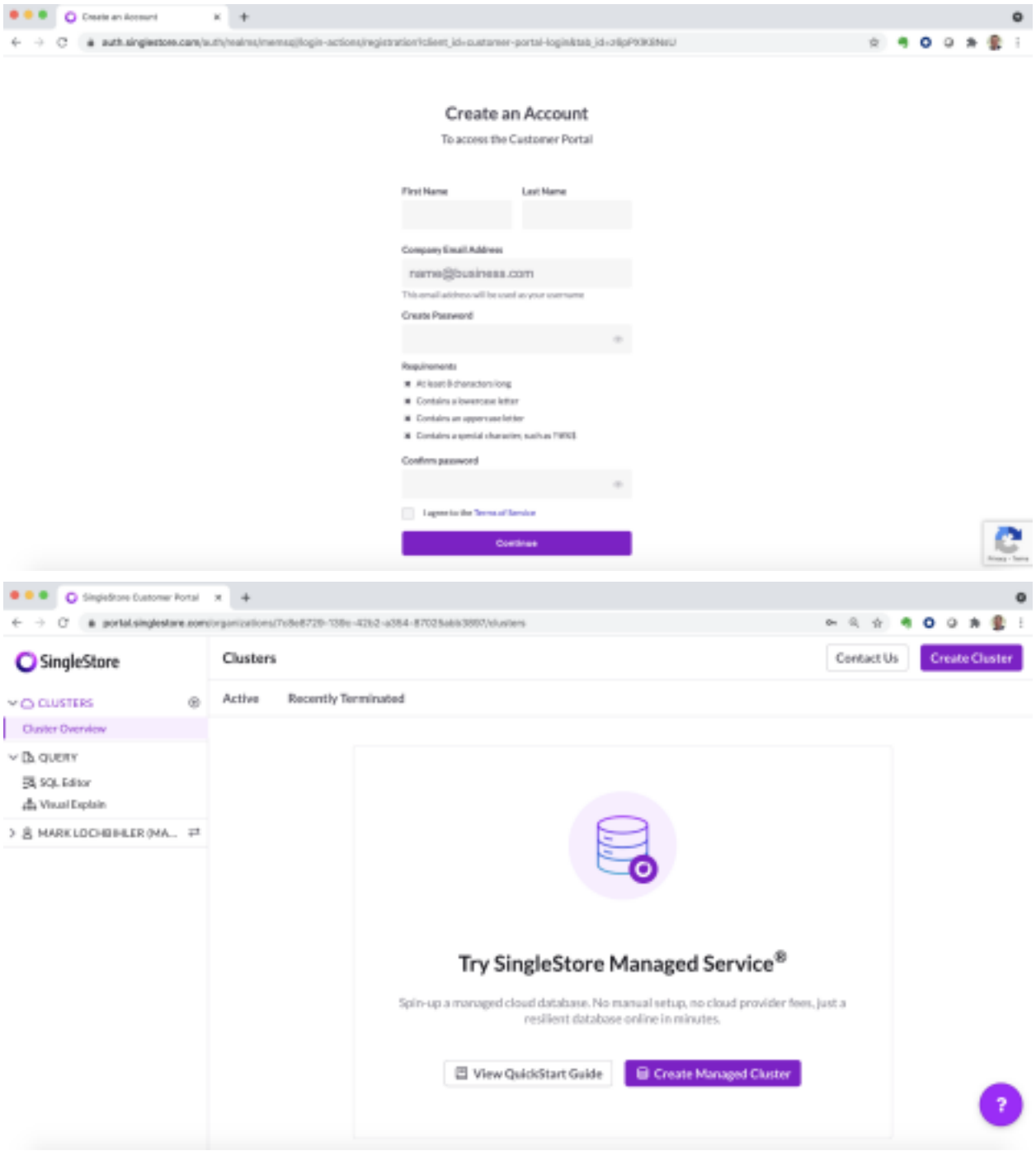


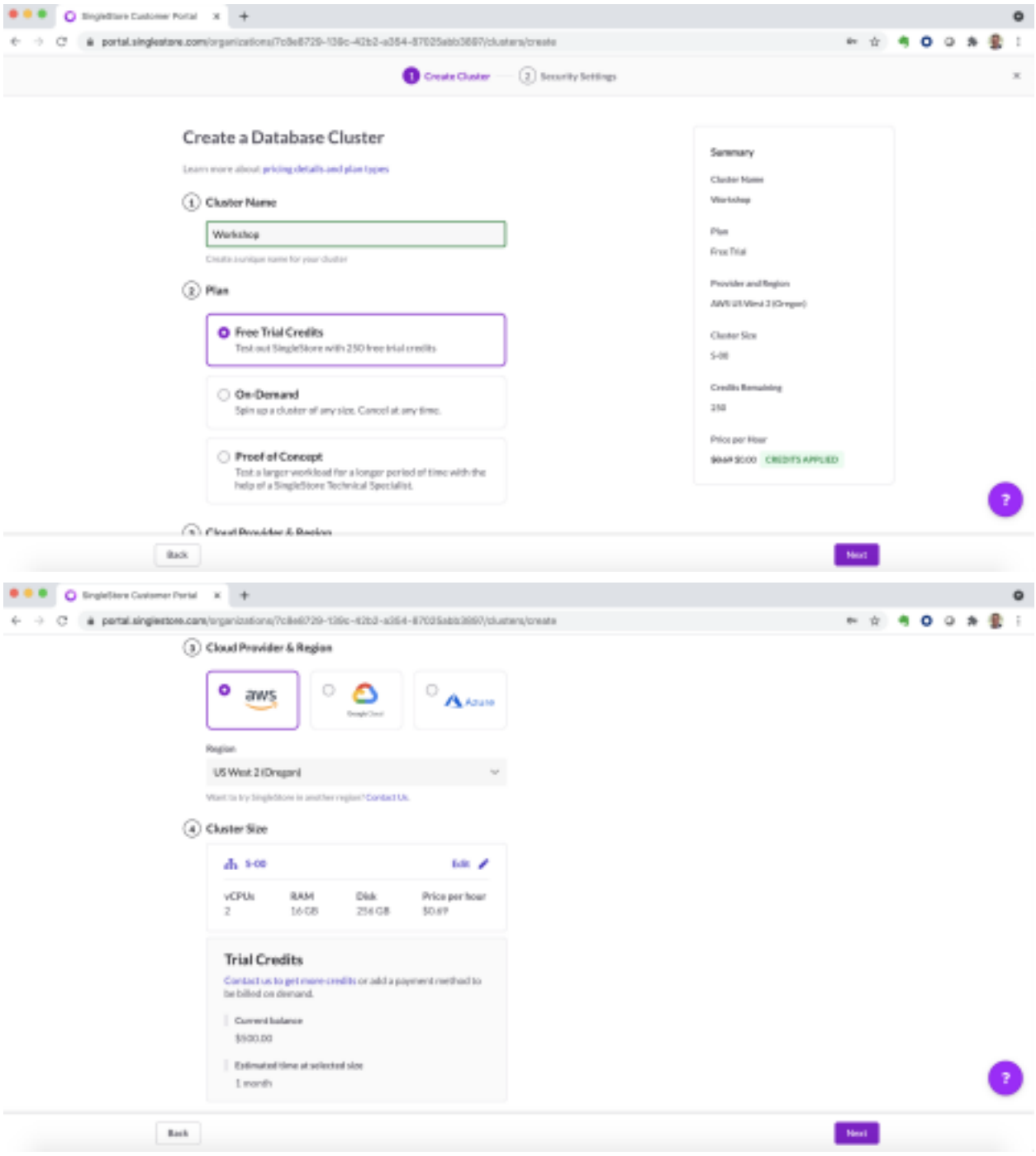
NOTE: For security reasons, we will delete the accounts shown in this guide after the workshop.

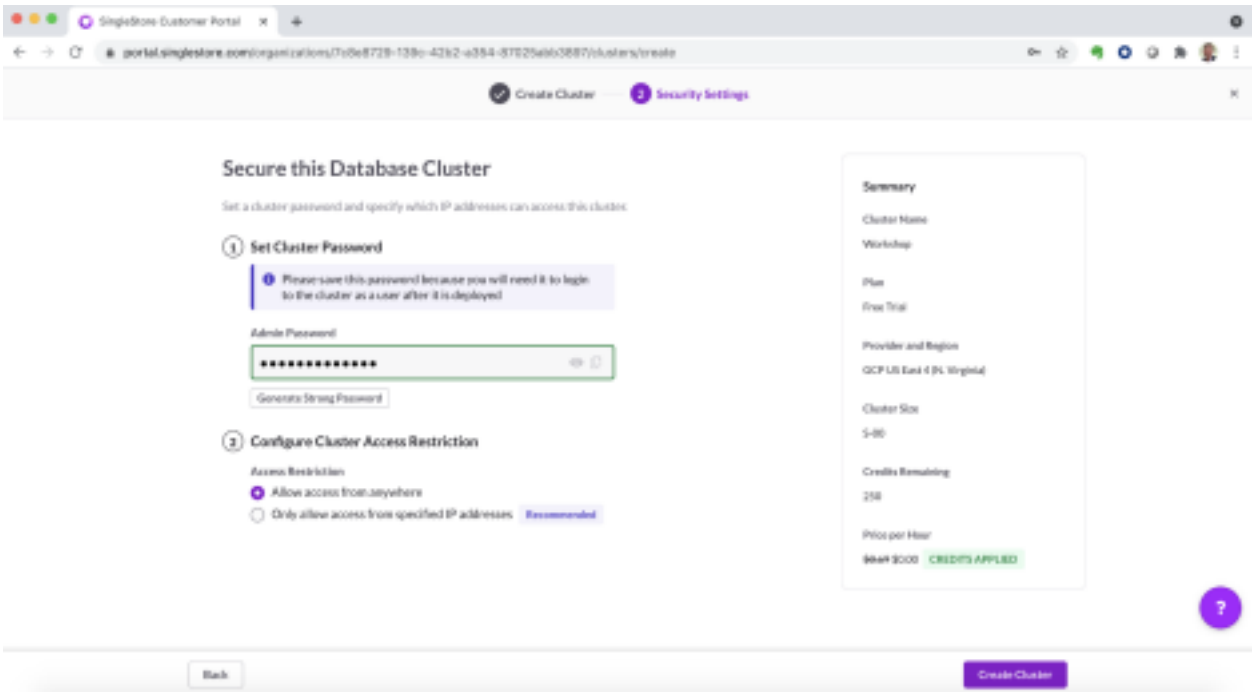
4 - Launch your own SingleStore Fully Managed cluster

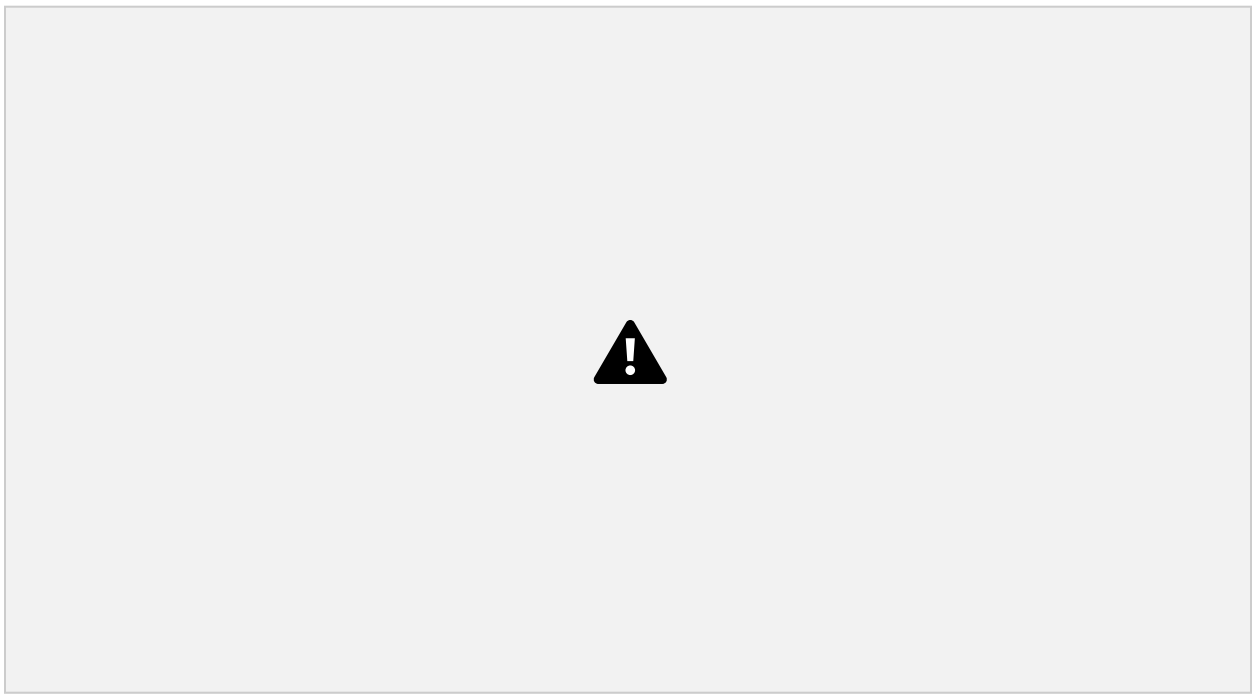
https://portal.singlestore.com

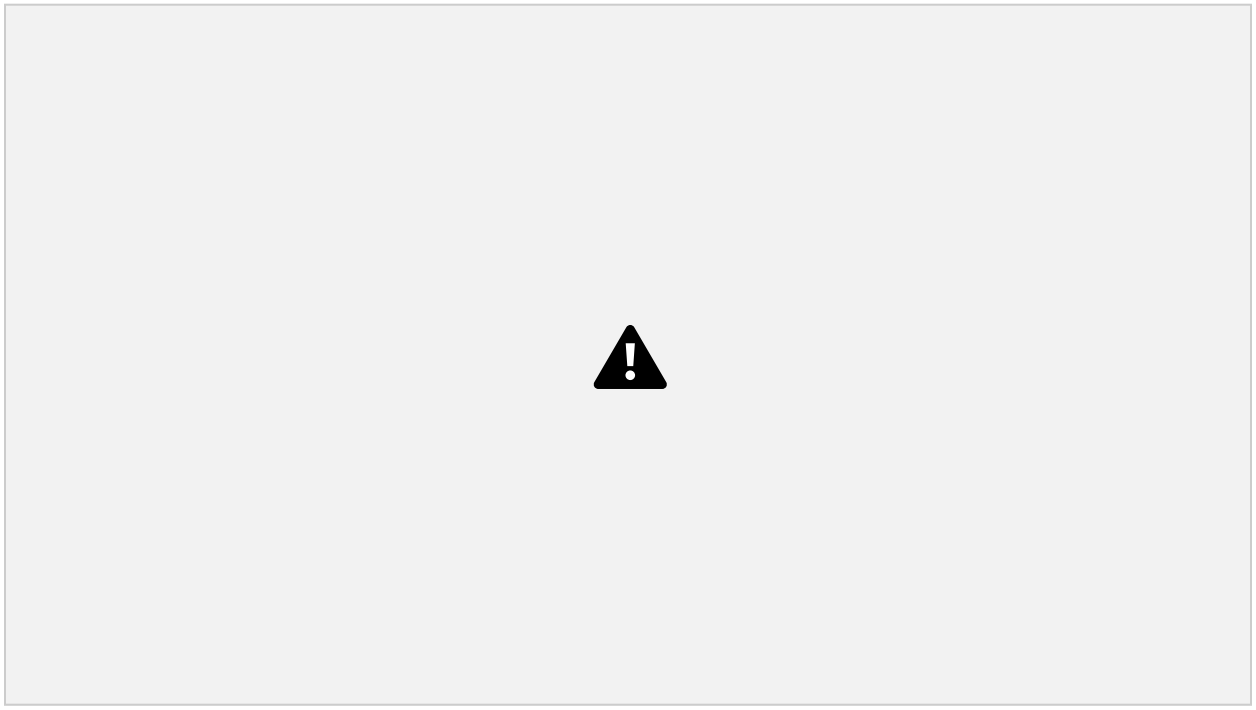


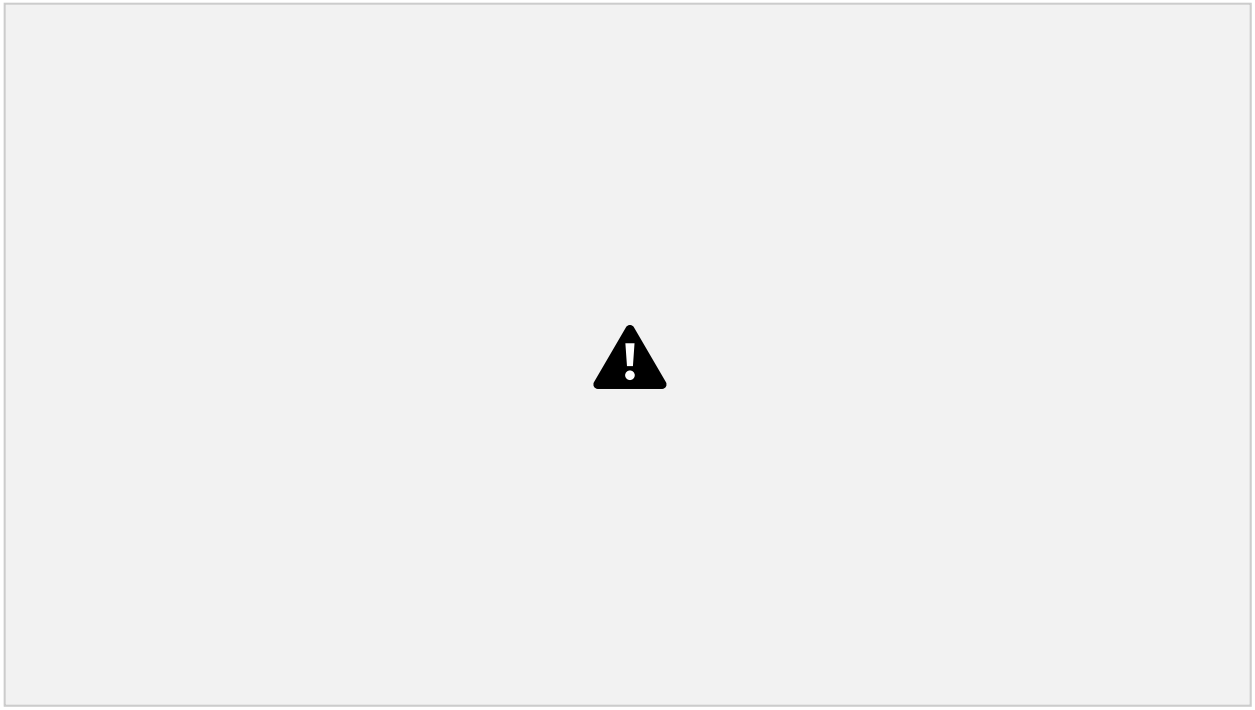

Create an Account
To access the Customer Portal
First Name
Last Name
Company Email Address
name@business.com
Create Password
Requirements
Confirm password
Continue
SingleStore
Clusters
Contact Us
Create Cluster
Active
Recently Terminated
CLUSTERS
Cluster Overview
QUERY
SQL Editor
Visual Explain
Try SingleStore Managed Service®
Spin-up a managed cloud database. No manual setup, no cloud provider fees, just a resilient database online in minutes.
View QuickStart Guide
Create Managed Cluster


Create Cluster
Security Settings
Create a Database Cluster
Cluster Name
Workshop
Plan
Free Trial Credits
Test out SingleStore with 250 free trial credits
On-Demand
Spin up a cluster of any size. Cancel at any time.
Proof of Concept
Test a larger workload for a longer period of time with the help of a SingleStore Technical Specialist.
Summary
Cluster Name
Workshop
Plan
Free Trial
Cluster Size
S-00
Credits Remaining
250
Price per Hour
$0.69 $0.00 CREDITS APPLIED
Back
Next
Cloud Provider & Region
aws
Azure
Region
US West 2 (Oregon)
Cluster Size
S-00
Edit
vCPUs
2
RAM
16 GB
Disk
256 GB
Price per hour
$0.69
Trial Credits
Contact us to get more credits or add a payment method to be billed on demand.
Current balance
$9.00.00
Estimated time at selected size
1 month
Back
Next


Create Cluster
Security Settings
Secure this Database Cluster
Set a cluster password and specify which IP addresses can access this cluster.
1 Set Cluster Password
Please save this password because you will need it to login to the cluster as a user after it is deployed
Admin Password
Generate Strong Password
2 Configure Cluster Access Restriction
Access Restriction
Allow access from anywhere
Only allow access from specified IP addresses
Recommended
Summary
Cluster Name
Workshop
Plan
Free Trial
Provider and Region
Cluster Size
S-00
Credits Remaining
250
Price per Hour
CREDITS APPLIED
Back
Create Cluster





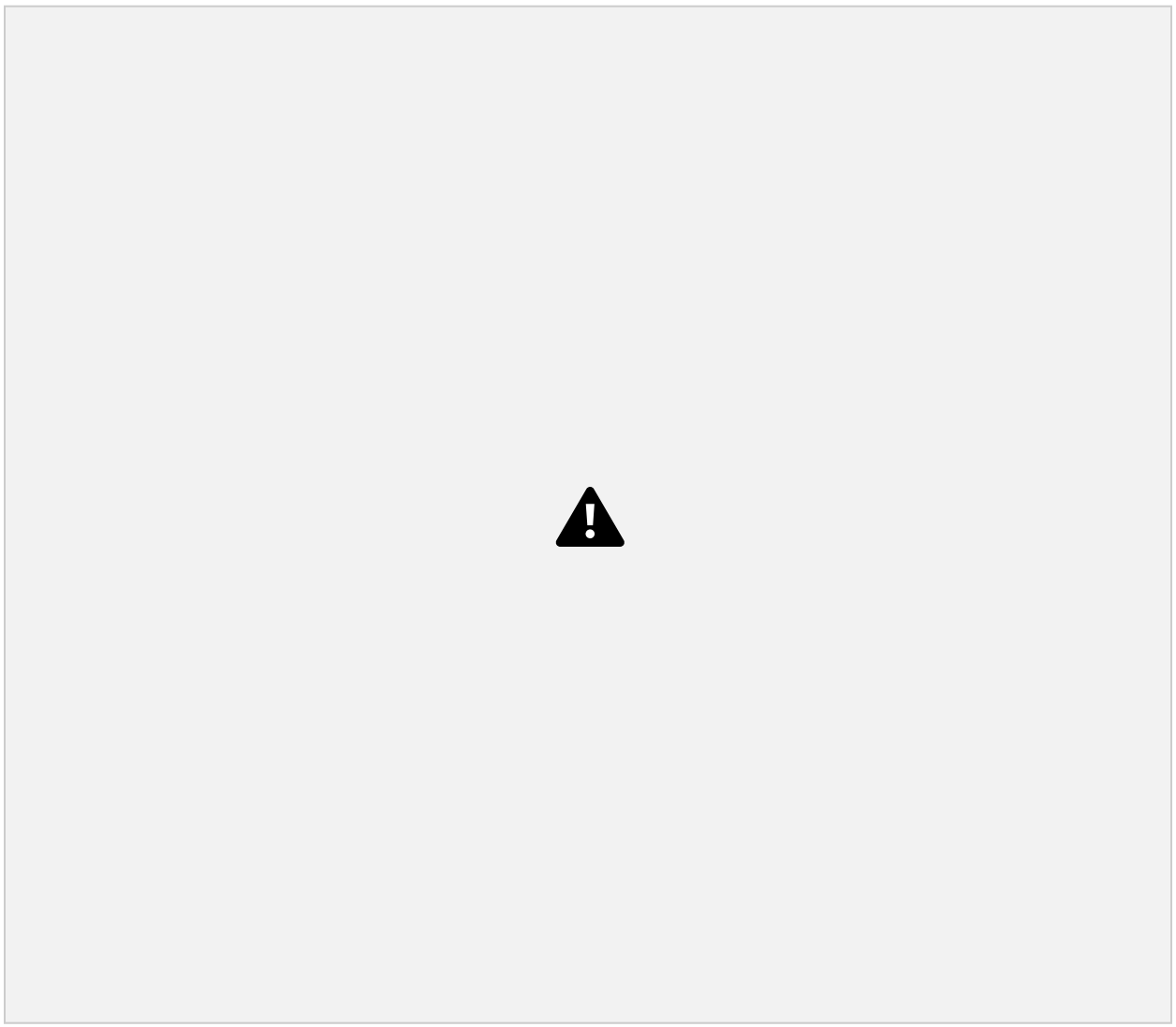


Please review the menus under the **OVERVIEW, QUERY, and INGEST** menus on the left side of the screen. We will frequently use the **Databases, SQL Editor, Active Processes and Pipeline** sub-menus throughout the remainder of this hands-on workshop.

5 - AWS Glue ETL Prep

Use the following AWS CloudFormation to store SingleStore connection credentials in AWS Secrets Manager and create an IAM role that will be used by AWS Glue job later.

Or copy paste the link in the browser:
https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=SingleStoreDevDay&templateURL=https://singlestoredevday.s3.amazonaws.com/cfn_template.yaml



6 - Launch an AWS CloudShell Service

We will use the AWS CloudShell Terminal to establish a connection with our SingleStore cluster.

From your CloudShell Terminal

Run:

```
git clone https://github.com/mlochbihler/singlestore-aws-looker-devday-01
```


Section B - Building an Operational Analytics Solution

Background

In this section, we'll illustrate rapid data ingestion with SingleStore using real-world data from the New York City Taxi and Limousine Commission (NYC TLC) yellow taxi trip dataset [1].

During this section, we will leverage SingleStore Data Ingestion Pipelines which is ingesting simulated taxi or ride-sharing data that we will then analyze in SingleStore. Two Kafka pipelines are used to stream data from their corresponding topics in a public Kafka cluster.

public-kafka.memcompute.com/drivers

public-kafka.memcompute.com/trips

If you are unfamiliar with Kafka Pipelines, see Kafka Pipelines Overview for more information.

You will also create one S3 pipeline so that you can load neighborhood data, and also, create an S3 pipeline for the NAB NYC Taxicab dataset.

Step 1 - Create Database and Load Data

From your last CloudShell Terminal

Run:

```
cd /home/ec2-user/singlestore-aws-looker-devday-01/section_b mysql -u
admin -p<password> -h <singlestore ddl endpoint service> -P 3306
```

Build Table Schemas

We will be providing you with the DDL commands to **create a new database** named **nyc_taxi with 3 tables**: **drivers**, **neighborhood**s, and **trips**.

Run:

```
mysql> source create_nyctaxi_tables.ddl;
mysql> use nyc_taxi; show tables;
```

Review Sample Payloads

For the sample stream, we will be publishing the status of 1000 drivers every second, and publishing 50 trips every thirty seconds. The “neighborhoods” stream is a static data set, and will not be updated once the initial load.

Driver Event Sample:

21,Ella,Harris,driving,POINT(40.73281182 -73.99684899),POINT(40.73950010 -74.00639998),NULL,1783796

Trip Event Sample:

30,requested,POINT(40.754100 -73.980000),POINT(40.755300 -73.962600),1483984668,1,0,0,0,1,9.299954999999471,-1

Sample Event Neighborhood:

281 Jamaica Queens POLYGON((40.70537600 -73.77582000, 40.70429051 -73.77592224, 40.70272444 -73.77547673, 40.69983099 -73.77395775, 40.69822334 -73.77734617, 40.69976700 -73.77854600, 40.69739000 -73.78446600, 40.69599500 -73.78277100, 40.69523700 -73.78211000, 40.68836400 -73.77662000, 40.68528000 -73.77240000, 40.68377800 -73.76974000, 40.68432922 -73.76963696, 40.68644322 -73.76116940, 40.68337600 -73.76042400, 40.68284600 -73.76073100, 40.67905300 -73.76183200, 40.67750200 -73.76289000, 40.67606842 -73.76346750, 40.67600300 -73.76349400, 40.67364000 -73.76383000, 40.67185300 -73.76435300, 40.66703500 -73.76696000, 40.66744600 -73.77113700, 40.66743600 -73.77295100, 40.66673000 -73.78367300, 40.66678400 -73.78511800, 40.66632163 -73.78941799, 40.66631400 -73.78948900, 40.66718129 -73.78930714, 40.67301000 -73.78808500, 40.67339100 -73.79047400, 40.67434400 -73.80121600, 40.67768100 -73.80323000, 40.68291367 -73.80583205, 40.68617474 -73.80745368, 40.68857916 -73.80870357, 40.69793100 -73.81414400, 40.70404100 -73.81702000, 40.70440200 -73.81443900, 40.70771600 -73.80278700, 40.70844600 -73.80022200, 40.70978288 -73.79553437, 40.71050718 -73.79353164, 40.71071100 -73.79304000, 40.71203000 -73.78932100, 40.71186500 -73.78771800, 40.71291943 -73.78271314, 40.71293800 -73.78262500, 40.70705100 -73.77997500, 40.70782300 -73.77678100, 40.70537600 -73.77582000))

Create SingleStore Pipelines

Run:

```
mysql> source create_nyctaxi_pipelines.dml;
mysql> show pipelines;
```

Start Pipelines

Run:

```
mysql> source start_nyctaxi_pipelines.dml;
mysql> show pipelines;
```

For a full list of SingleStore Pipeline commands, please refer to:

SingleStore-pipelines-commands

In addition, when Troubleshooting SingleStore Pipelines, the following tables will be helpful:

information_schema.pipelines_batches
information_schema.pipelines_batches_metadata
information_schema.pipelines_batches_summary
information_schema.pipelines_cursors
information_schema.pipelines_errors
information_schema.pipelines_files
information_schema.pipelines_offsets

Run Operational Analytic Queries

We inspected the "**neighborhoods**" data in previous step. Lets take a quick look at the trips and drivers data.

Run:

```
mysql> select * from trips limit 5;
mysql> select * from drivers limit 5;
mysql> select * from neighborhoods limit 5;
```

Report 1: Total number of trips for each neighborhood

Run:

```
mysql> source total_trips.sql;
```

Report 2: The average amount of time between someone requesting a ride and that person being picked up

Run:

```
mysql> source average_wait.sql;
```


Report 3: The average distance of a trip

Run:

```
mysql> source average_distance.sql;
```

Report 4: The average amount of time between someone being picked up and that person being dropped off

Run:

```
mysql> source average_ride_time.sql;
```

Report 5: The average cost of a trip

Run:

```
mysql> source average_cost.sql;
```

Report 6: The average amount of time it takes from the time a driver accepts a ride to the time they pick up the passenger

Run:

```
mysql> source average_wait2.sql;
```

Report 7: The average number of riders per trip

Run:

```
mysql> source average_num_riders.sql;
```


Step 2 - Deploy a Real-Time Analytics Stored Procedure

In this step, we will operationalize Step 1 queries to run in Real Time, in a continuous loop. This will enable for demo purposes, live statistics for use in our Real Time Dashboard. This will be done using SingleStores's extensible Procedural SQL (MPSQL) , documented here:

SingleStore Procedural SQL

Create and Start SingleStore Stored Procedure

Take a moment to inspect the MPSQL code by clicking on the code URL link and then execute the statement below:

Run:

```
mysql> source create_derived_stats.sql;
```

We will want to run the stored procedure we just created above from a separate CloudShell terminal. It is a simple procedure that will loop continuously. You can use Ctrl-C to stop it.

From a new CloudShell Terminal

Run:

```
cd /home/ec2-user/singlestore-aws-looker-devday-01/section_b mysql -u
admin -p<password> -h <singlestore ddl endpoint service> -P 3306
```

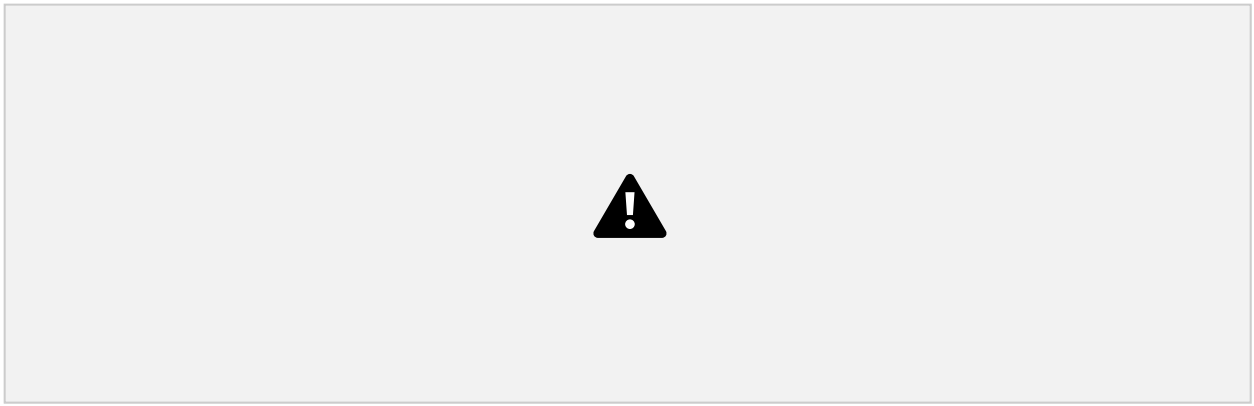
Then Run:

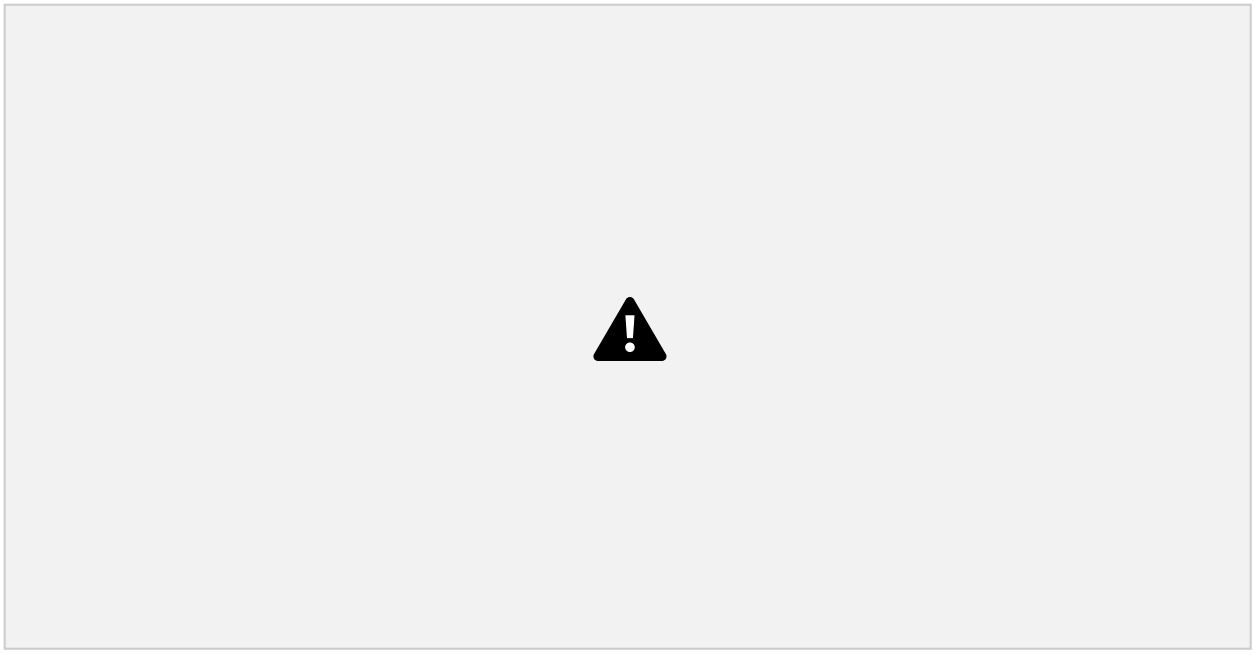
```
mysql> source start_derived_stats.dml;
```

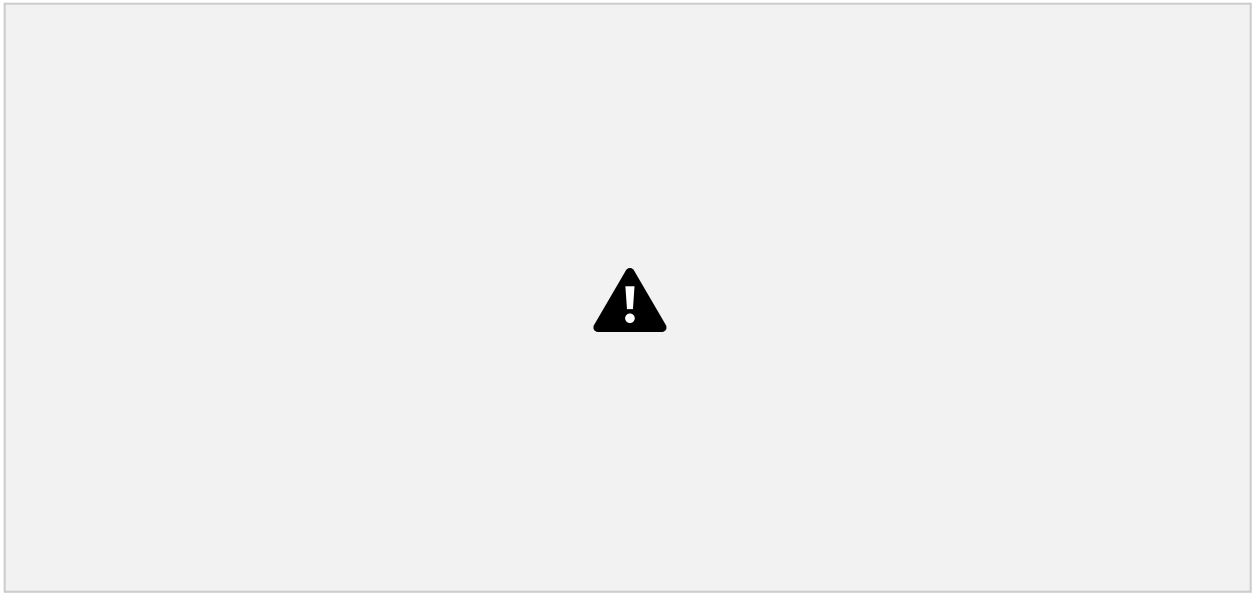

Step 3 - Leverage AWS Glue ETL with SingleStore Spark Connector

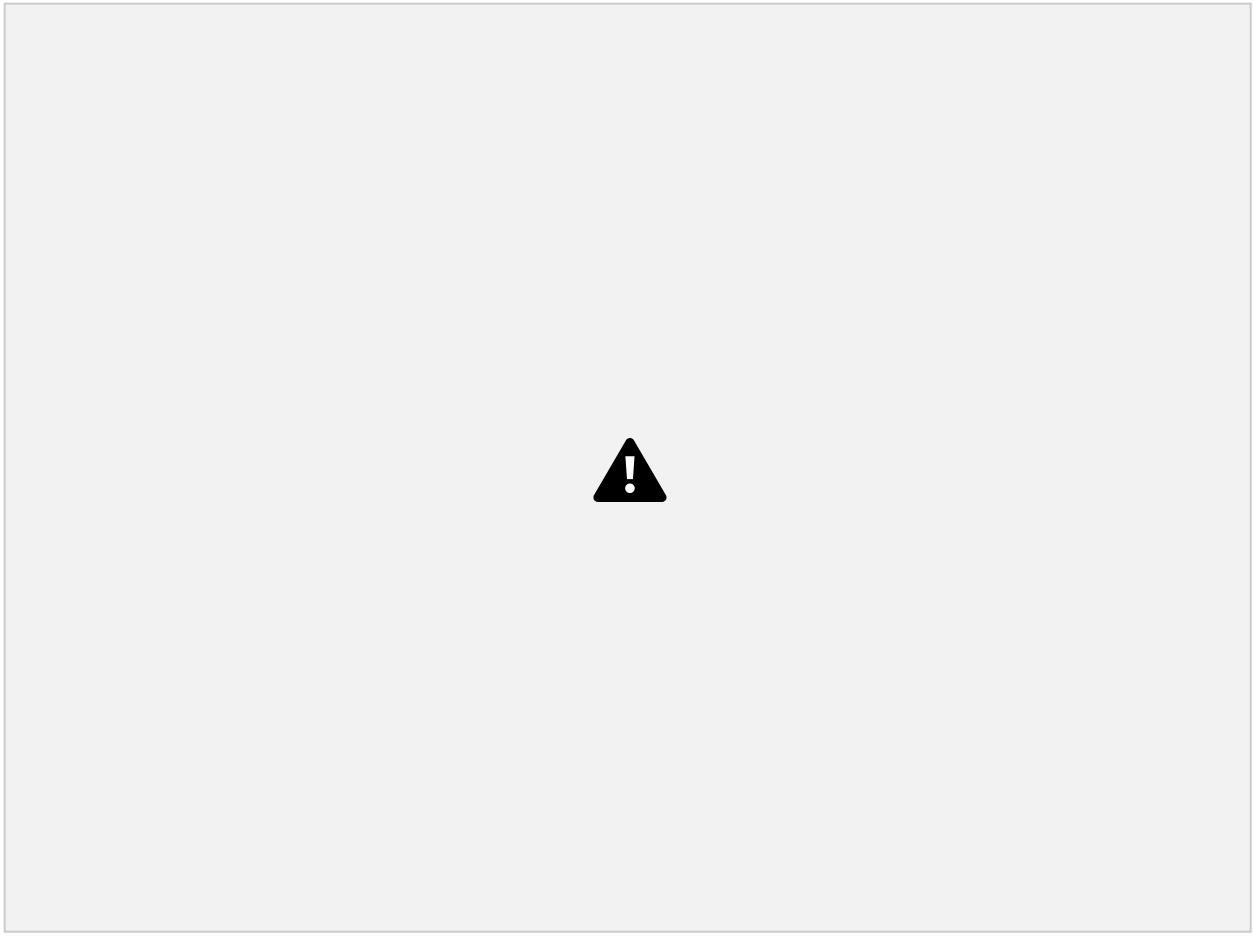
Navigate to AWS Glue Studio in AWS Management Console.

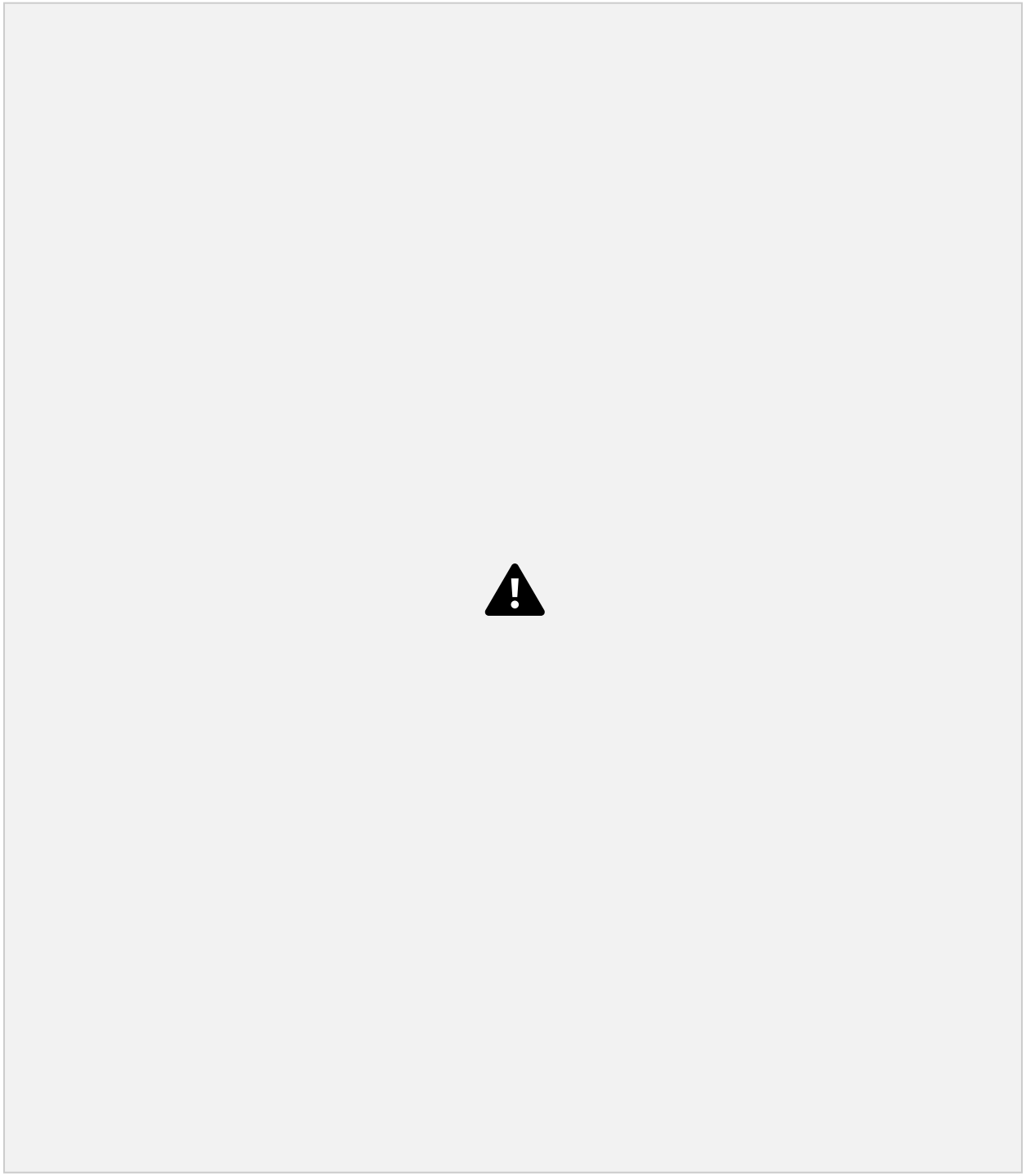
https://console.aws.amazon.com/gluestudio/home

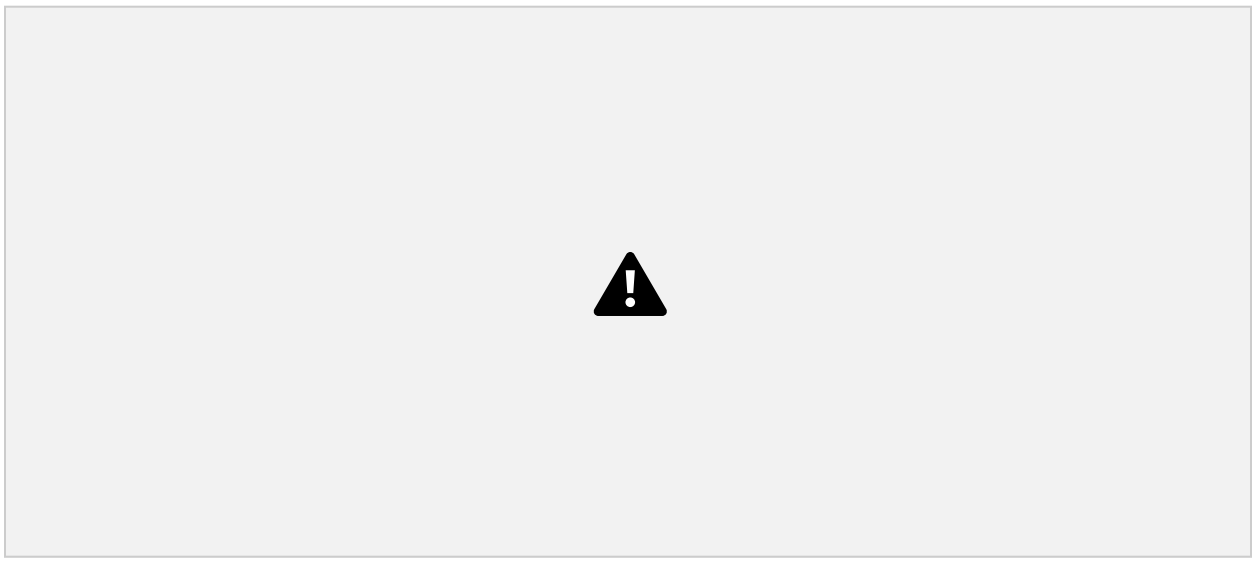


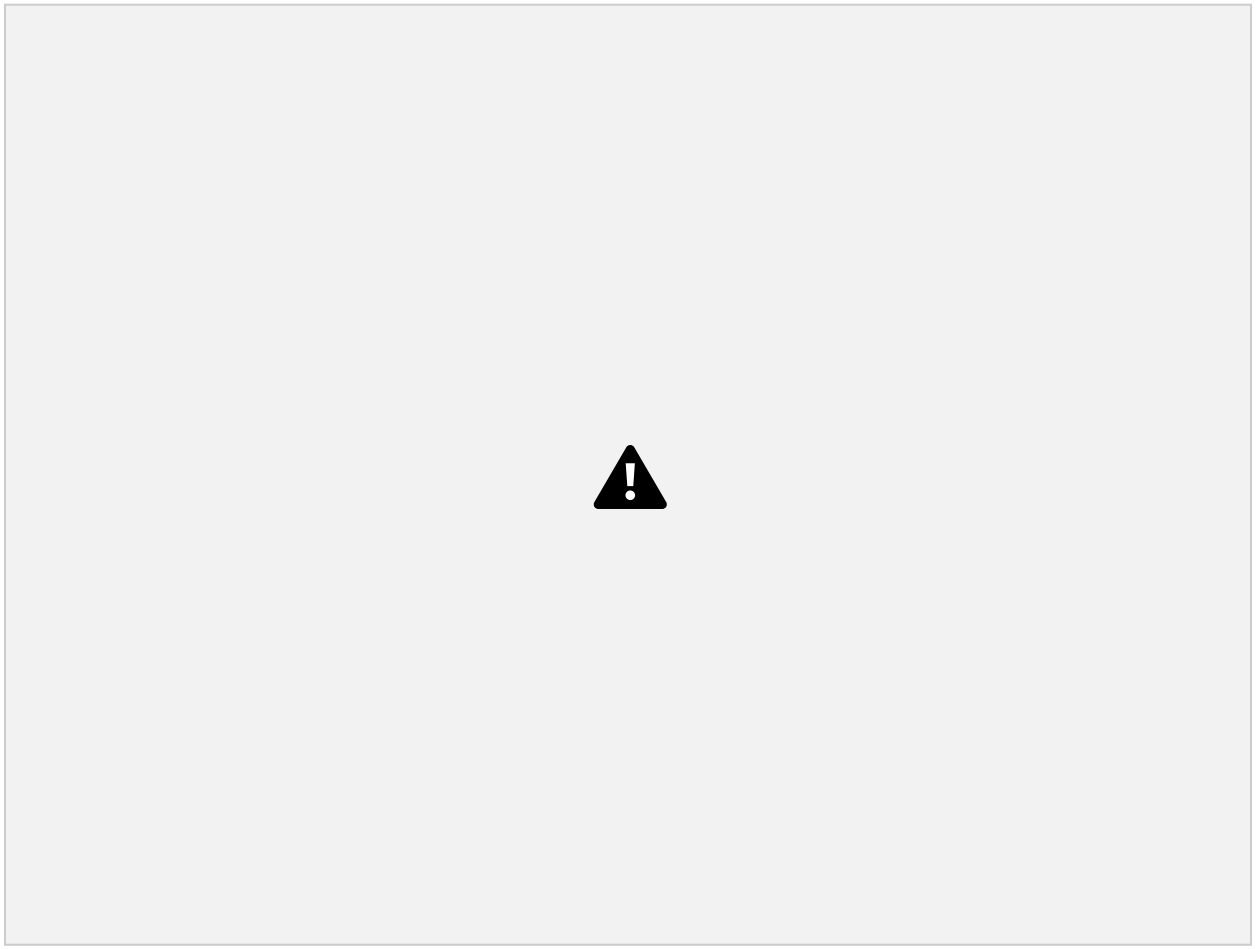












31

Update following Job details and Save

IAM Role: SingleStoreDevDay-GlueCustomETLConnectionRole

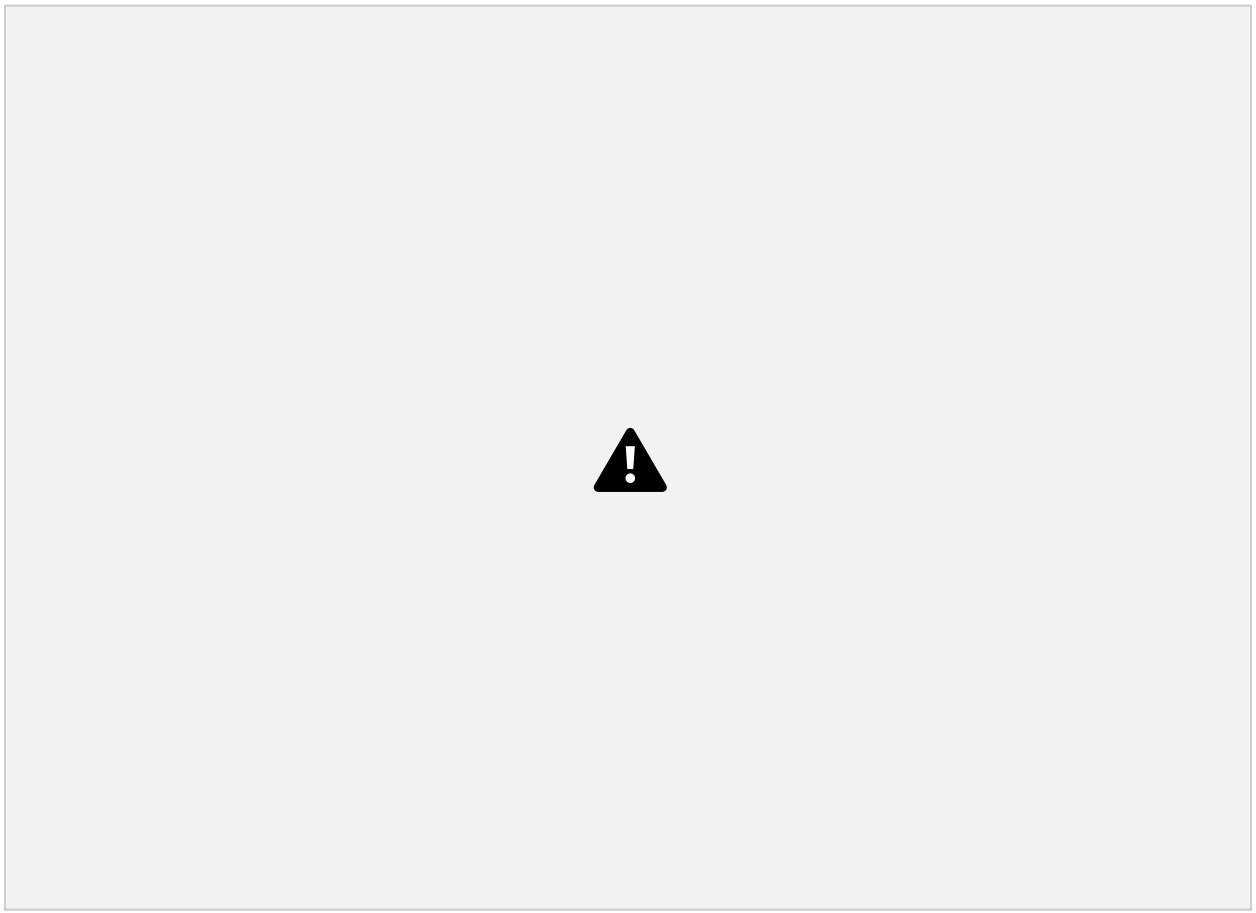
Number of workers: 3

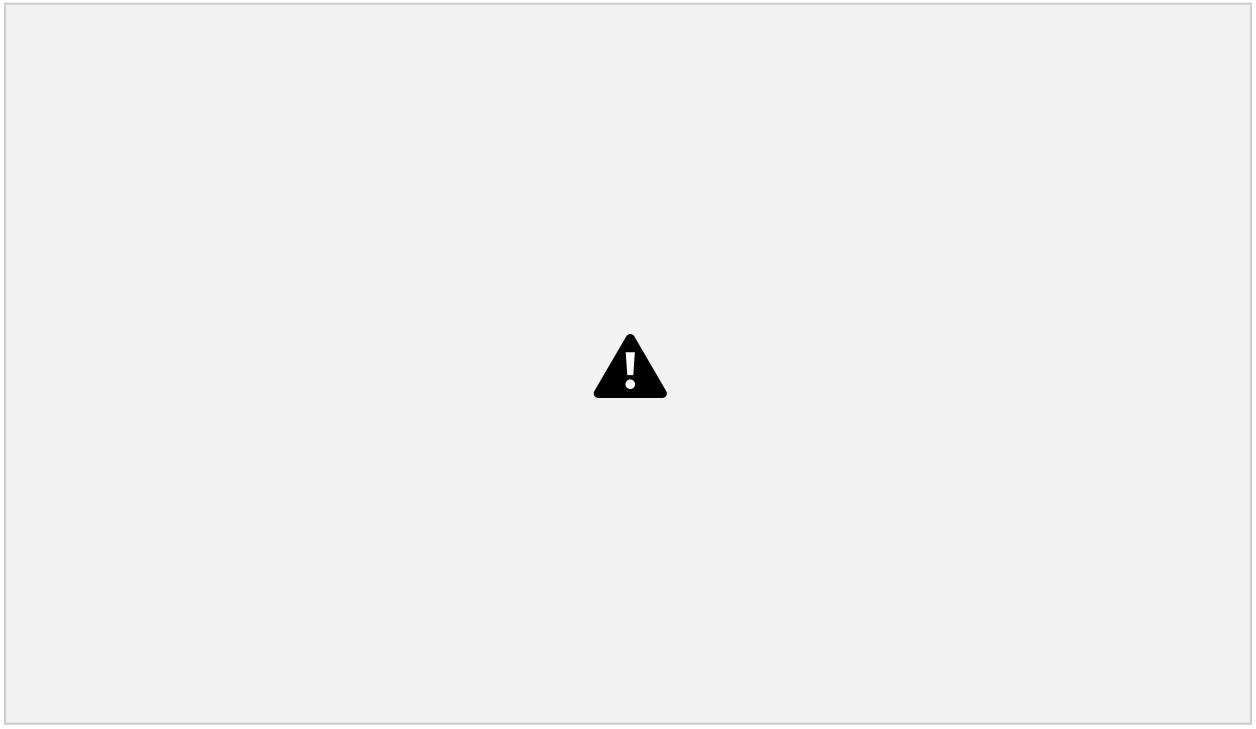
Number of retries: 0

Job bookmark: Disable

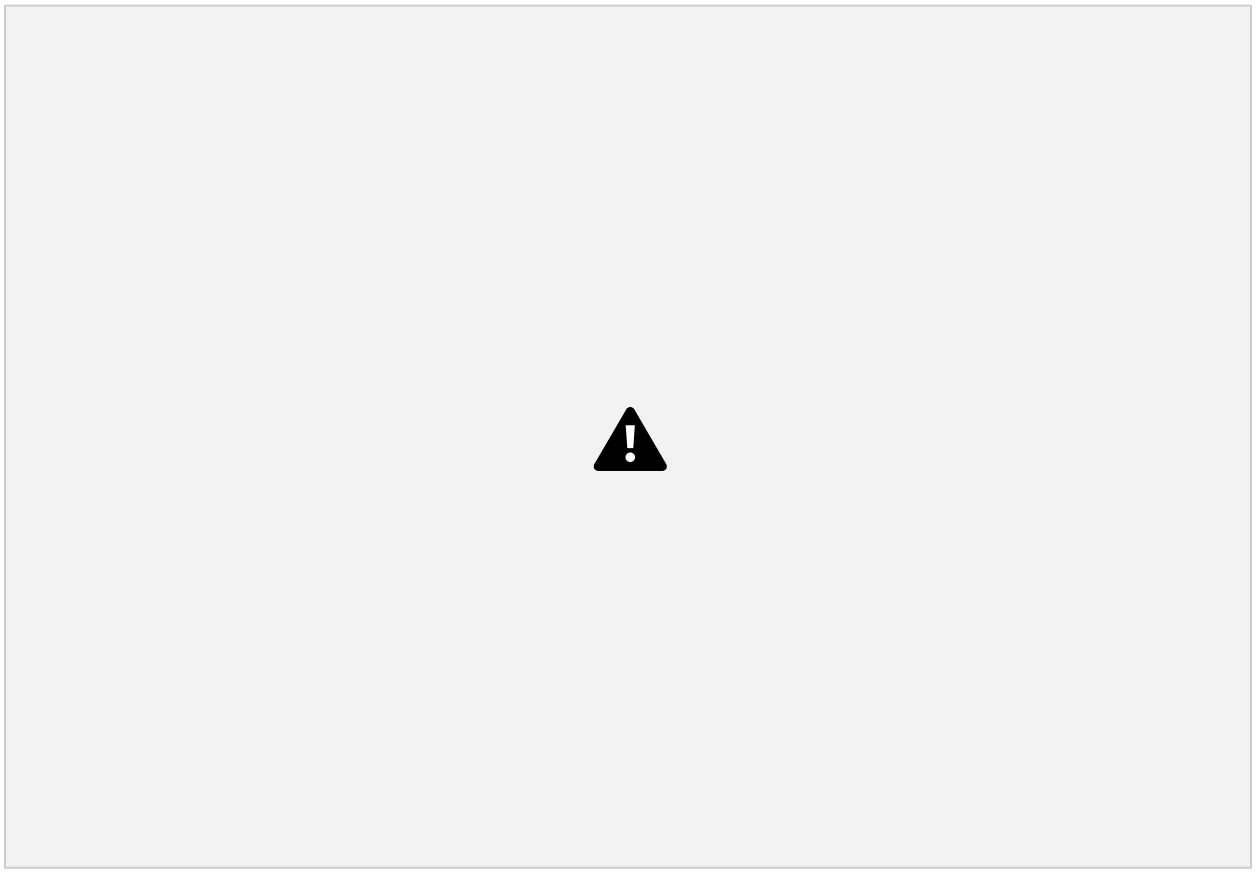
Configure the job using the Visual interface :

S3 URL for Source Node : s3://singlestoredevday/files/

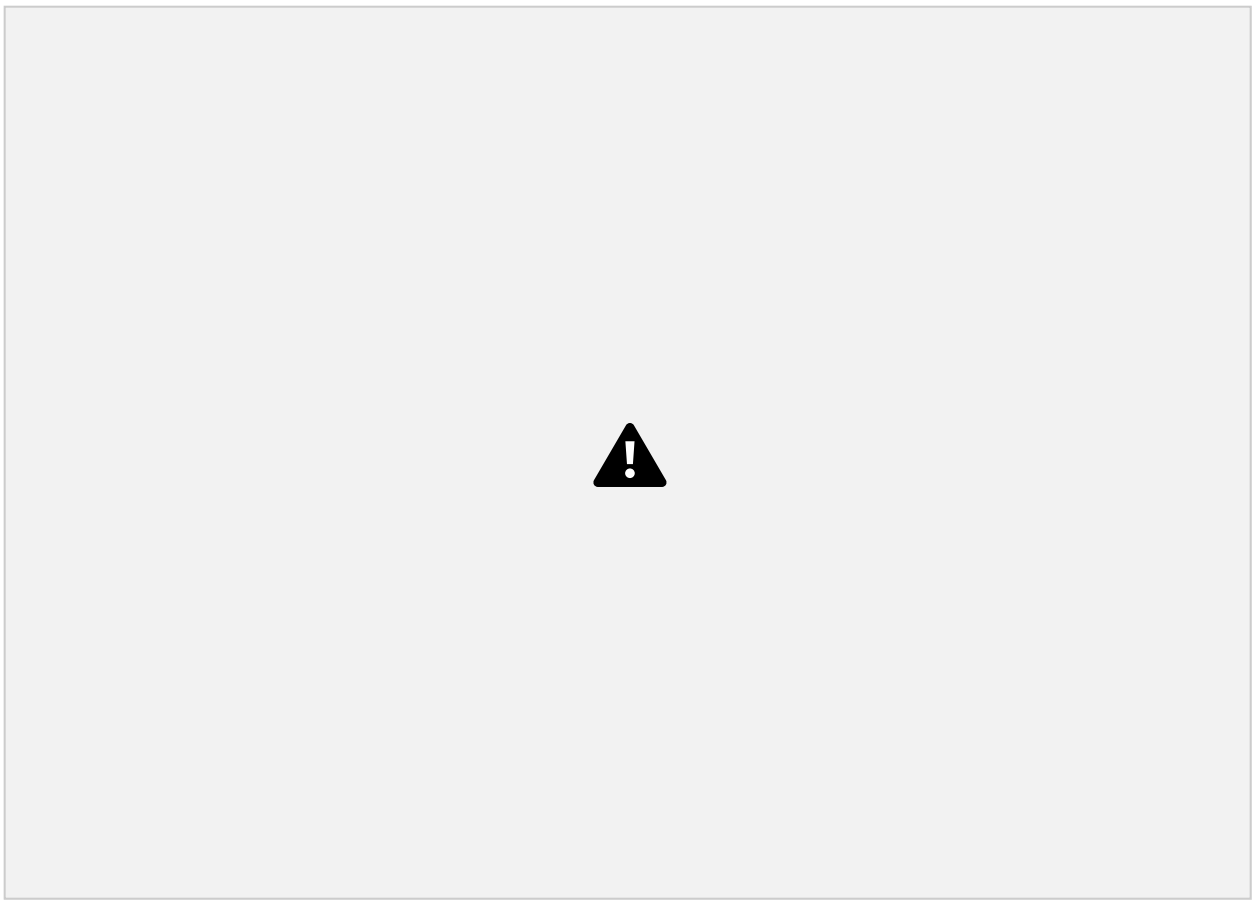




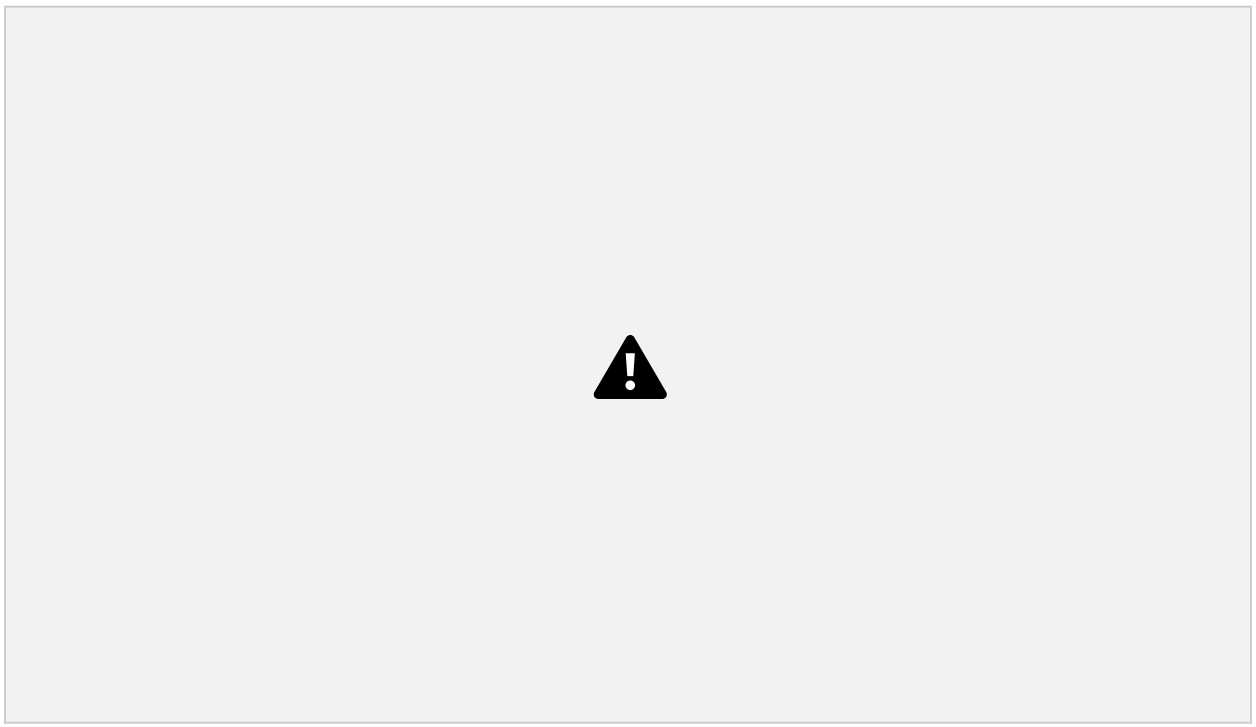
33



34



Check the values in “Data Preview”

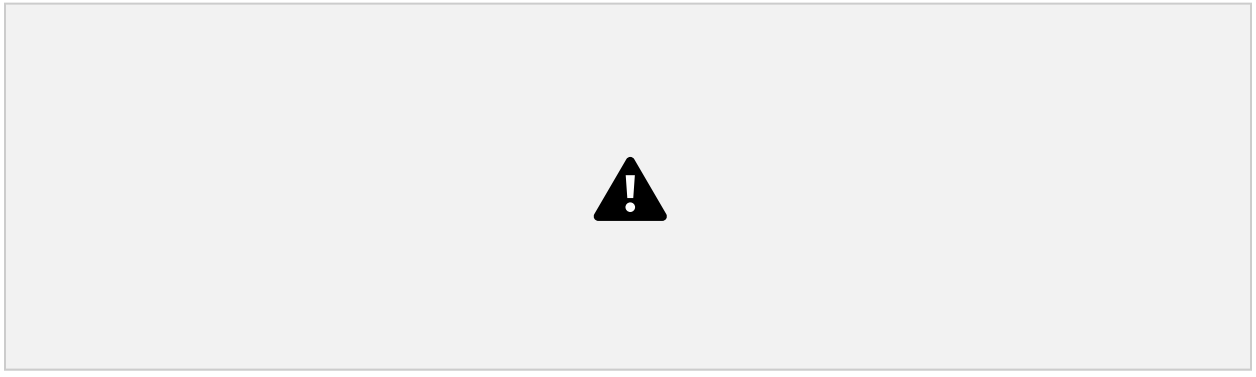


Save and run the Glue job.

Run:

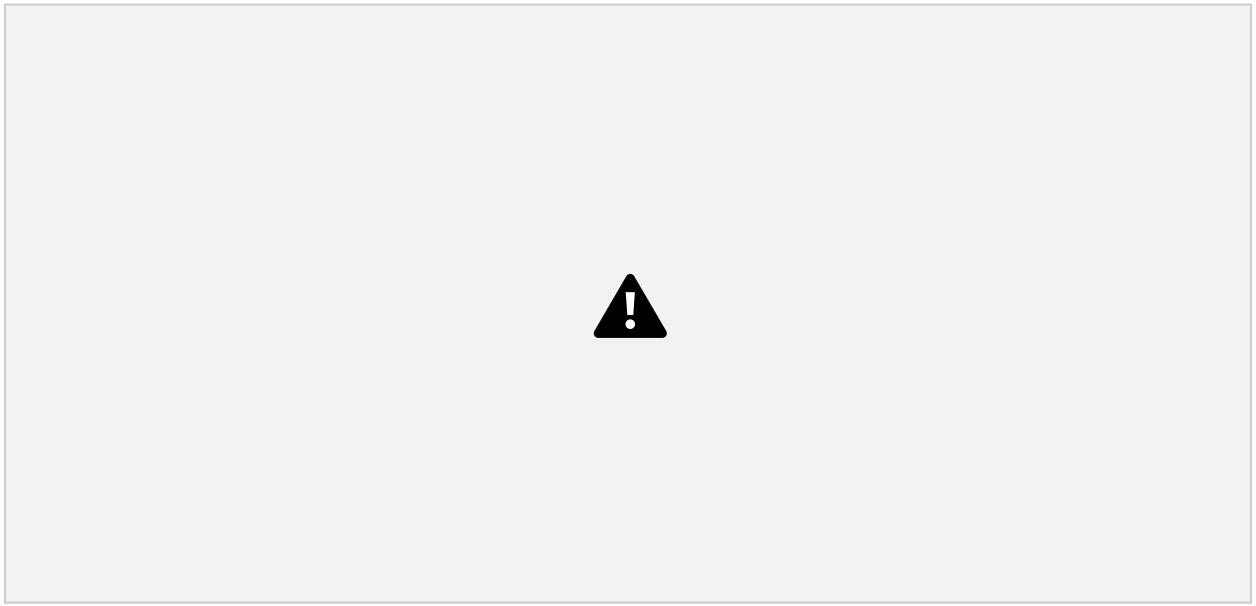
```
mysql> source show_anomaly_scores.sql;
```

Sample Output:



Step 4 - Bring it all together with an Operational Dashboard

Building Real time operational Dashboards



Dashboard Requirements

Business users require real time, self-service dashboards leveraging the data ingested by SingleStore pipelines, and output from SingleStore stored procedures.

Application shall support building of analytical model/semantic layer using provided data with business logic to support analytics.

Dashboards should be interactive, easy to use, and allow end users to do their own analysis of the data.

SingleStore will power all of the SQL compute power for the dashboards. 37

Some of the KPI’s being analyzed.

- Avg Miles Per trip
- Avg Cost per Ride
- Avg Riders Per trip
- Avg Ride Time
- Drill exploration by time and NYC borough, neighborhood & LatLong
- Drill exploration by driver
- Anomaly Score

Visualizing the Taxi Data and Creating a Dashboard

Task #1 - Log Into Looker

You'll use the following information to connect:
URL: https://memsqlhandsonlab.looker.com
User Name: lookerlabmemsql+<Your Lab User Number>@gmail.com
Password: 4Analytics_<Your Lab User Number>
Check the box next to "Stay Logged In"

For example, if you were Lab User #268, you would log in with the following credentials:
User Name: lookerlabmemsql+268@gmail.com
Password: 4Analytics_268
Check the box next to "Stay Logged In"

Task #2 - Create a Database Connection

Before we can work with the data in our database, we need to create a database connection in Looker to your SingleStore database, if one does not already exist. Looker communicates with all supported databases using JDBC. Creating and managing database connections is an Administrative function in Looker, and lab users won't have access to the Admin functionality. The instructor will demonstrate how a new connection is created. For this lab, we will all be sharing a connection named *devdayslab*, which has already been created for you.

You can find more information about creating and managing database connections in Looker here:
https://docs.looker.com/setup-and-management/connecting-to-db

Task #3 - Create a Looker Project

A project is a collection of files that describe the objects, connections, and user interface elements that will be used to carry out SQL queries for your Looker users. Lab users will not have permissions to create a new Project. The instructor will demonstrate how a new Project is created. For this lab, we will all be sharing a project named *dev_days_taxi_basic*, which has already been created for you.

You can find more information about Looker Projects here:

https://docs.looker.com/data-modeling/getting-started/how-project-works

Task #4 - Examine the Generated Files

To access the LookML files created by the generator, select “Develop” from the main menu and then select the name of the project that you’d like to work with from the drop down list. Select the project “dev_days_taxi_basic”.

Explore the types of files created by Looker Project Generator. There are two types of LookML files generated:

Model File: This file lists all of the tables that will be used in the project, and the join relationship between those tables. The tables are grouped into *explores*, which are used for logically grouping tables together that can help answer questions about a specific business concept. Designing explores so that users can get access to the data that they need to answer their questions without being overwhelmed with other data that is not relevant to their business need is an important part of the job of a Looker Data Analyst.

The Looker Project Generator will infer join relationships between tables and make a best guess at how tables should be grouped for explores. The Data Analyst can modify the defaults so that they make the most sense for the Business Users

View File: Not to be confused with a database view, a Looker View file describes the columns in a database table or database view. They can also describe a Looker Derived Table or a Looker Persistent Derived Table, but we will not be covering those in the lab. The Looker Project Generator will create a dimension for each column in the table, and will also add an additional measure named *count,* for counting the number of rows in a grouping.

The View File is also the place where the Data Analyst will add all of the custom dimensions and measures that comprise the key metrics for running your business. Defining these business specific data definitions is one of the most important jobs of the Looker Data Analyst. This ensures that business users are all using standard naming and calculations for their business

metrics.

Task #5 - Enable Development Mode

When you are looking at Project Files without enabling Development Mode, you are looking at a read-only version of the Production version of the Project files. If you want to make any modifications to the Project files, you'll need to enable Development Mode. Once you are in Development Mode, you'll be working with your own private version of the Project files (actually it's your own git branch, if you are familiar with GitHub). While in Development Mode you can make and test your modifications without affecting any other users.

To enable Development Mode, select "Develop" from the main menu and slide the Development Mode slider to the right.

You'll know that you are in Development mode when you see the purple Development Mode bar at the top of the page.

Task #6 - Enhance the Model File

Enhance the Model File. Now that we are in Development Mode, we can make changes to the Project files. The Project Generator made it's best guess about the relationship between the tables. As we saw earlier in the lab, we can use SingleStore Geospatial functions for joining the

trips data to the *neighborhoods* data. We'll add the details of that join relationship to our Model File. Once added, users will be able to use the *neighborhoods* data along with the *trips* data without having to know how to join the data themselves. Looker will take care of joining the data for them.

We will be joining the *neighborhoods* data to the *trips* data twice, once for the *pickup location* and again for the *dropoff location*. We'll alias the *neighborhoods* table, just like would if we were writing SQL.

Looker's join syntax is similar to standard SQL, but has the advantage of only needing to be defined once in the Model File, and then reused in all of your users' reports and dashboards.

Modify your *trips* explore to include the two additional joins to the *neighborhoods* table. The finished version should look like this:

```
explore: trips {
  join: drivers {
    type: left_outer
    sql_on: ${trips.driver_id} = ${drivers.id} ;;
    relationship: many_to_one
  }
  join: pickup_neighborhoods {
    from: neighborhoods
    type: left_outer
```

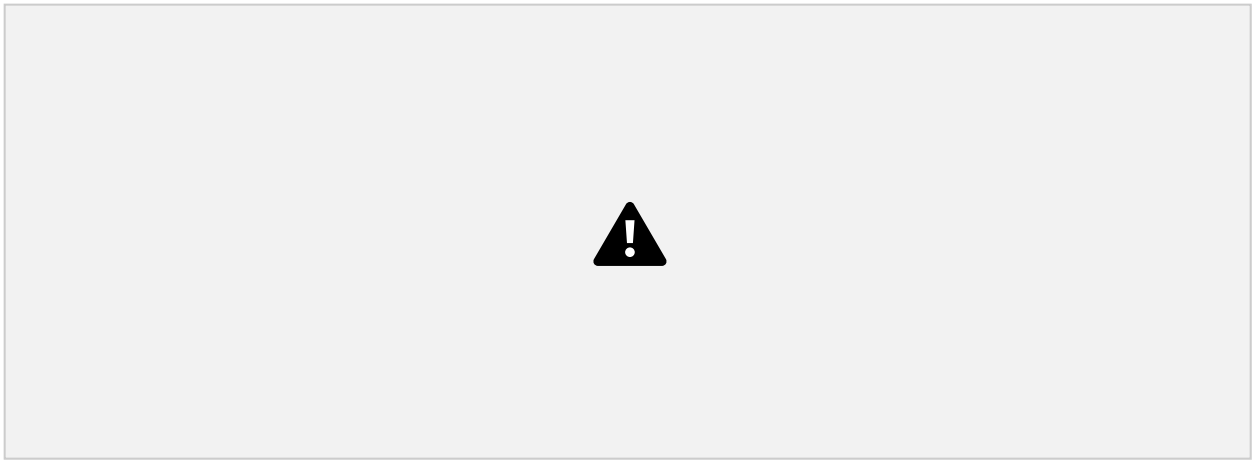


```
    relationship: many_to_one
    sql_on: GEOGRAPHY_INTERSECTS(${trips.pickup_location},
${pickup_neighborhoods.polygon}) ;;
  }
  join: dropoff_neighborhoods {
    from: neighborhoods
    type: left_outer
    relationship: many_to_one
    sql_on: GEOGRAPHY_INTERSECTS(${trips.dropoff_location},
${dropoff_neighborhoods.polygon}) ;;
  }
}
```

Notice that the syntax of the two additional joins is slightly different, using Looker's *from* identifier to associate the correct table name with the aliased table name used in the join definition.

Once your changes are made, save them by clicking the "Save Changes" button at the top of

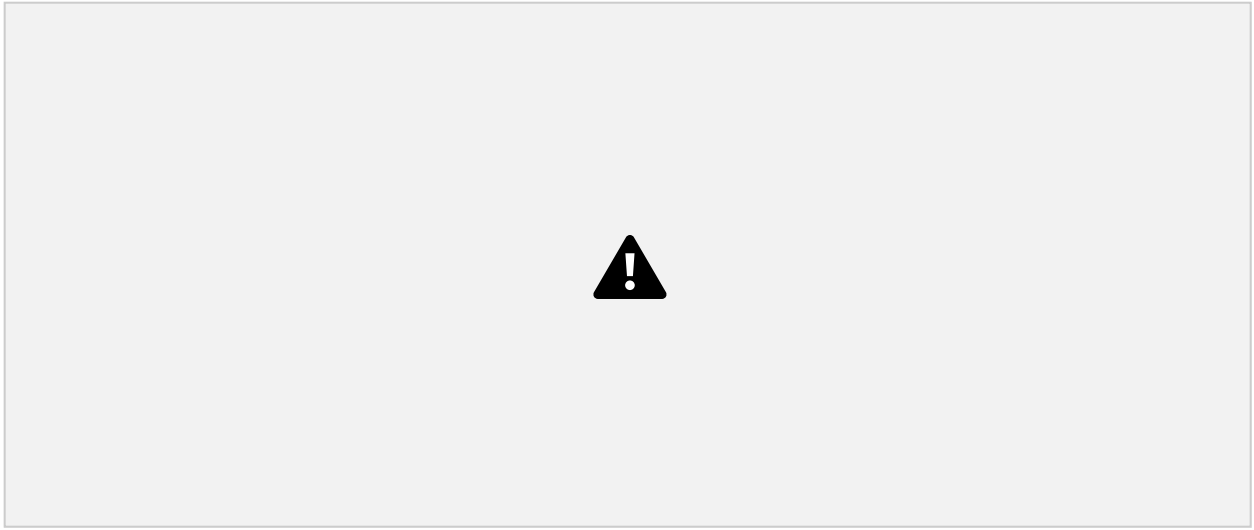
the page.



Task #7 - Enhance a View File

Let's add a new dimension to the *drivers* view file. When we are reporting on drivers it would be handy to have the drivers' names formatted as *last_name, first_name.* We'll add this as a new dimension to make it available and easy to use for all of our users.

Access the drivers View File by expanding the "views" section of the file browser and clicking on the "drivers.view" file.



Add the new dimension to the View File. You can add it anywhere you'd like. In this example I've added it above the *first_name* dimension. The finished version should look like this:

```
dimension: id {
  primary_key: yes
  type: number
```

```
    sql: ${TABLE}.id ;;
  }

  dimension: full_name {
    type: string
    sql: CONCAT(${last_name}, ', ', ${first_name}) ;;
  }

  dimension: first_name {
    type: string
    sql: ${TABLE}.first_name ;;
  }
```

Notice that we have used SingleStore-specific syntax for concatenating the first and last name along with a comma. You can use any valid database-specific syntax when defining a new dimension or measure.

Also, take note of the use of the `${field name}` syntax. The "dollar sign curly braces notation" is Lookers substitution operator and is one of the most important aspects of making your code reusable

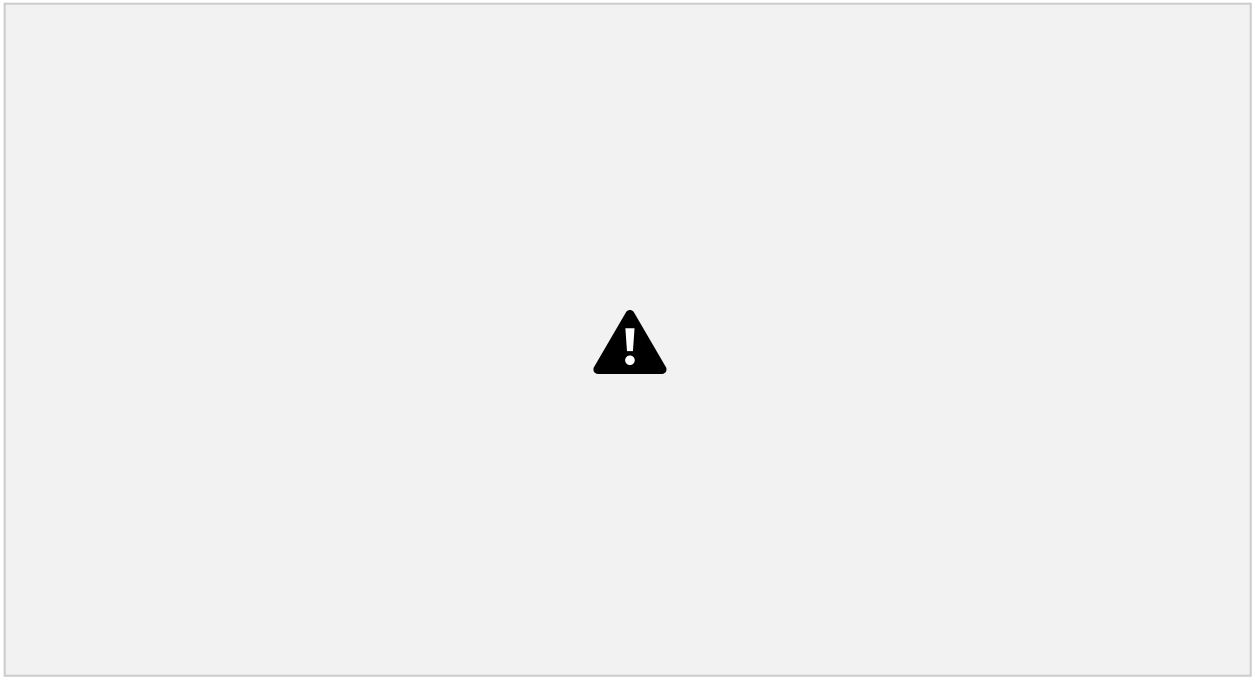
Once your changes are made, save them by clicking the "Save Changes" button at the top of the page.

Task #8 - Looker's Explore Interface

User's access Looker's Explore Interface for doing their own data exploration as well as creating their own reports and dashboards. This is also where they take advantage of the work that the Data Analyst has done enhancing the LookML Model. You'll be moving between modifying the LookML files and working with the explore interface, so you might want to open a second browser tab for the Explore Interface.

Access the Explore Interface by choosing Explore from the main menu, and then clicking on the "Trips" explore under "Dev Days Taxi Basic".

Notice that “Dev Days Taxi Basic” corresponds to the name of your Model File and “Trips” corresponds to the name of the explore within the Model File.



On the left side of the Explore Interface is the Field Picker. This is where you can see the results of the work that you have done in the LookML Model. You can see the new dimension, *Full Name*, that you added to the “driver” view file. You can also see the two joins that you added to the “trips” explore in the Model File.

Task #9 - Create a Query

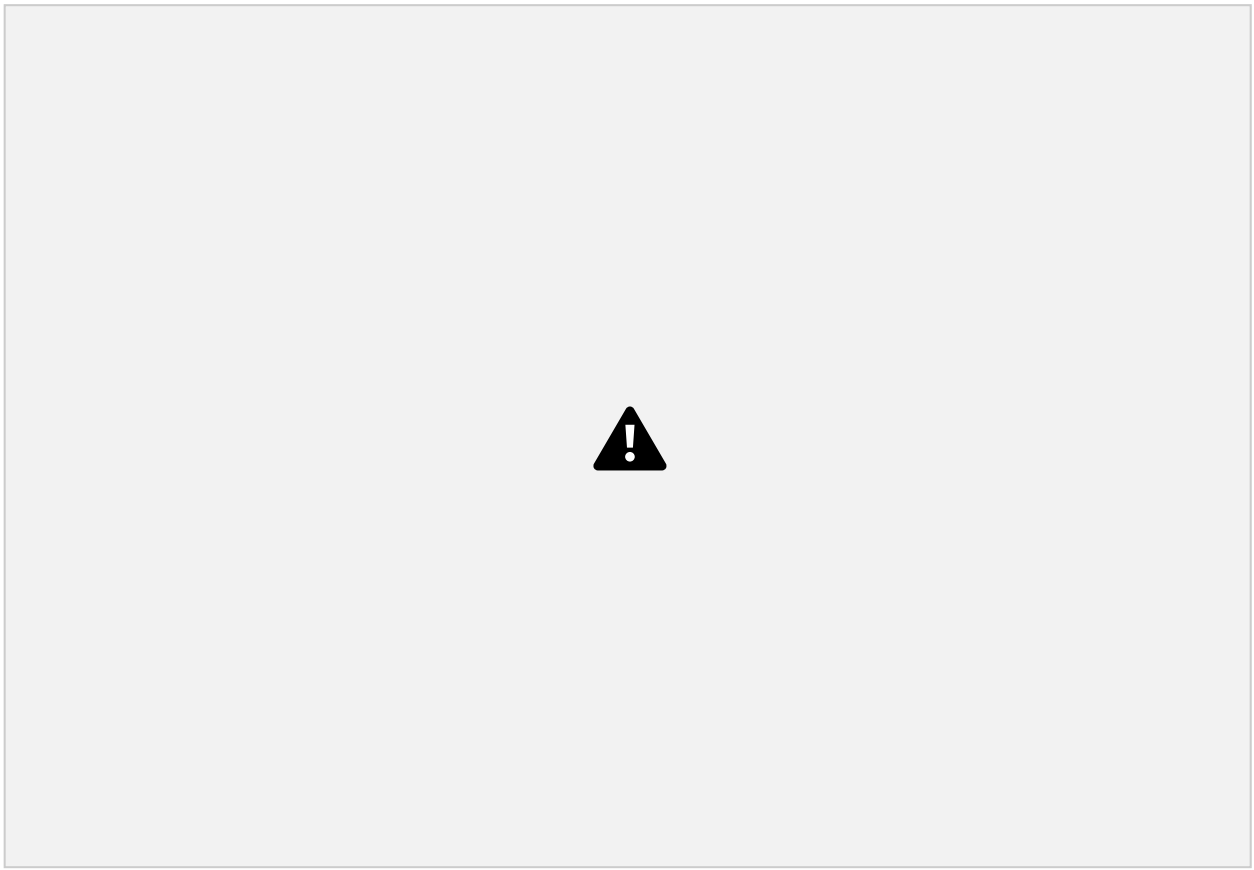
Users create their own queries by clicking on the fields in the Field Picker that they want to add to their report. This will add the fields to the data palette. To filter on a field, hover on the field name and select the “Filter” button.

We’ll create a query that shows Crosstown Traffic - the top 15 drivers who had the most rides originating in Manhattan with a destination in another borough. Follow these steps to create the query:

1. Under the “Drivers” section of the Field Picker, click on the “Full Name” dimension to add it to the Data Palette.
2. Under the “Trips” section of the Field Picker, click on the “Count” measure to add it to the Data Palette.
3. Under the “Pickup Neighborhoods” section, hover over the field “Borough” and click on the “Filter” button.

4. Under the “Dropoff Neighborhoods” section, hover over the field “Borough” and click on the “Filter” button.
5. In the filter that was added for “Pickup Neighborhood”, click in the text entry box. A list of all possible values will be displayed. Select “Manhattan”
6. In the filter that was created for “Dropoff Neighborhood”, change the “is equal to” comparison operator to “is not equal to”
7. In the filter that was created for “Dropoff Neighborhood”, click in the text entry box. A list of all possible values will be displayed. Select “Manhattan”
8. Limit the number of rows returned to 15

Your Explore Interface should now look like this:



For future queries in the lab, rather than step by step instructions, a Recipe Card format will be used instead. The Recipe Card for this query would look like this:

Title Crosstown Traffic - Top 15

Drivers

Model Name Dev Days Taxi Basic
Name
Explore Trips

Dimensions Drivers - Full Name

Measures Trips - Count

Vis Type Column Chart

Filter 1 Pickup Neighborhoods - Borough is equal to Manhattan Filter 2 Dropoff

Neighborhoods- Borough Is not equal to Manhattan Edit - Series Color Collection -

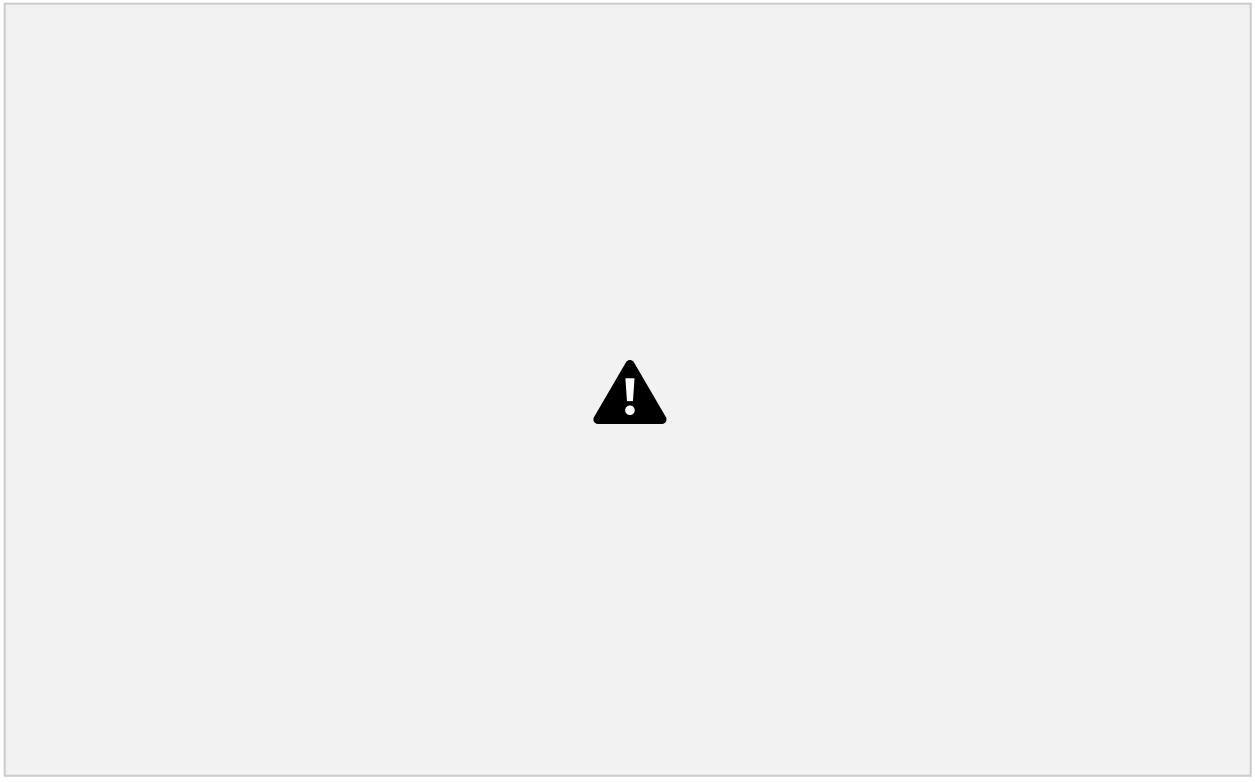
Dalton

Edit - Values Enable Value Labels

Task #10 - Review the SQL That Looker Has Generated

As you were making your selection in the Explore Interface, Looker was generating a SingleStore-specific query to answer your question.

Click on the “SQL” tab to see the SQL that Looker has generated. The SQL tab is generally only accessible by Administrators and Developers.



Notice how much of the work you have done in the LookML files is visible in this query. Looker knows how to join the tables involved in the query by using the join conditions that you added to the “trips” explore in the Model File, and the SQL that you used for defining the “full_name” field has been inserted into the query. Users are able to generate complex queries by simply clicking on the fields they are interested in.

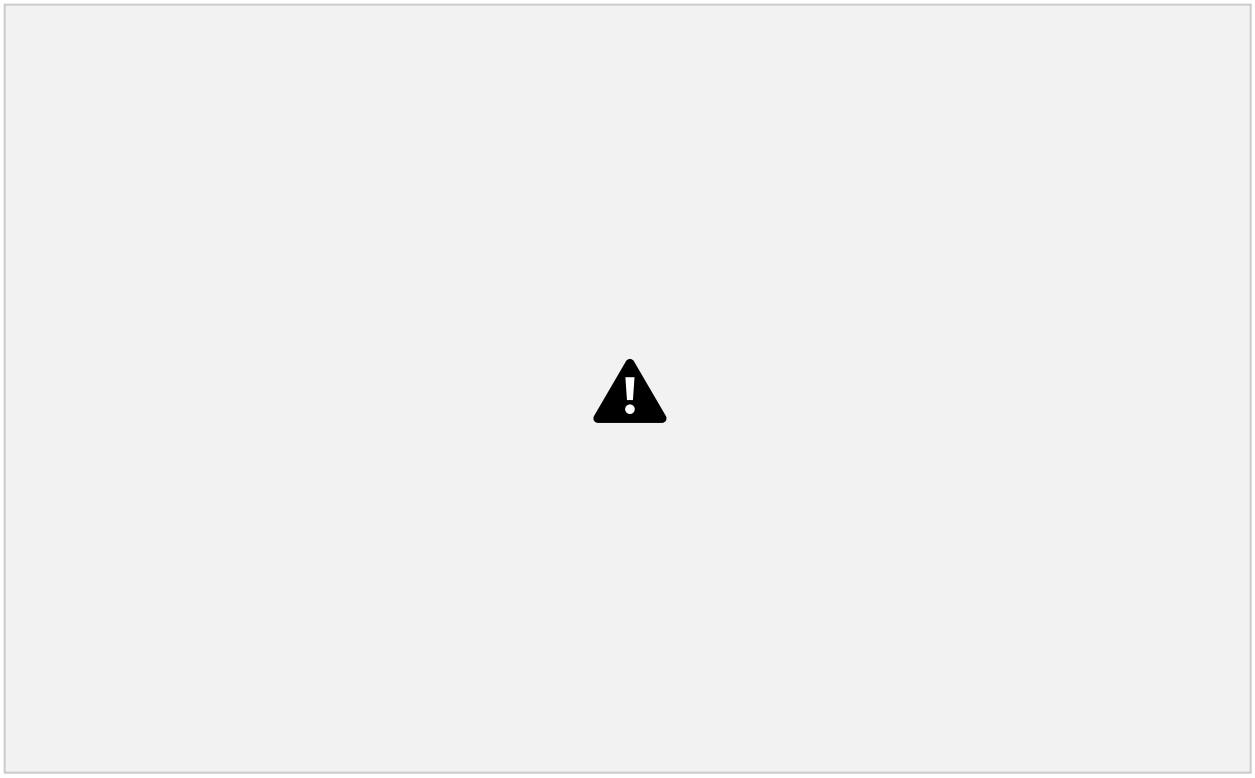
Task #11 - Visualize the Data

Now that the query has been defined, it’s time to run the query and visualize the results.

Click back to the “Results” tab:

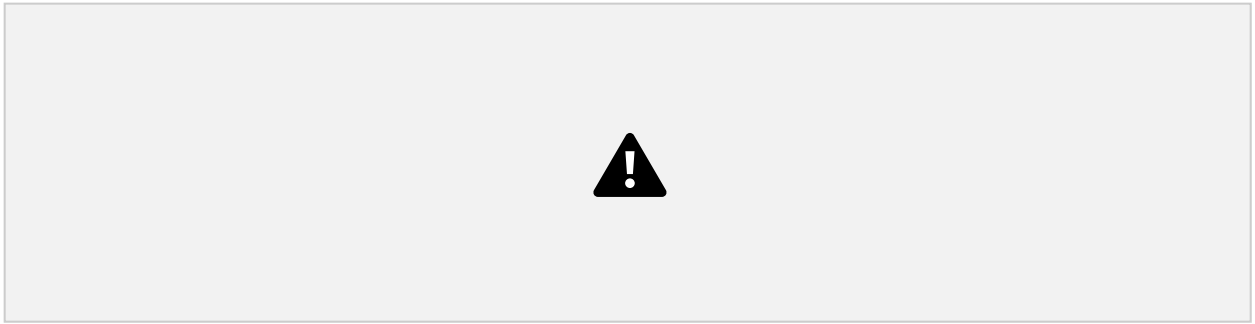
Now click the “Run” button on the top right of the page. This will send the query to SingleStore using the JDBC connection that we had previously defined and Looker will display the results.

If the “Visualization” section is not expanded, expand it now. Looker will pick a default visualization type based on the type of data that you included in your query.

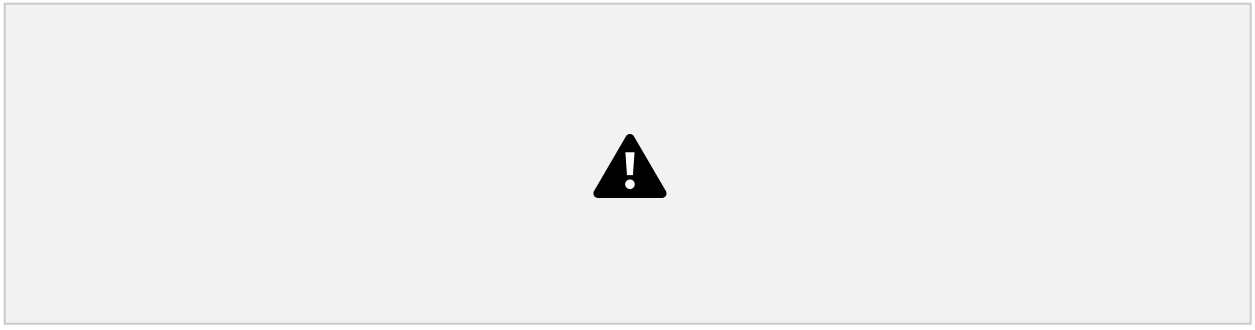


Click through some of the different visualization types available to see some different options for visualizing your data.

Select the Column Chart, the second option from the left. We’ll customize the chart a bit to make it more informative and more visually appealing. To access the chart customization options, click the gear icon on the Visualization menu bar:

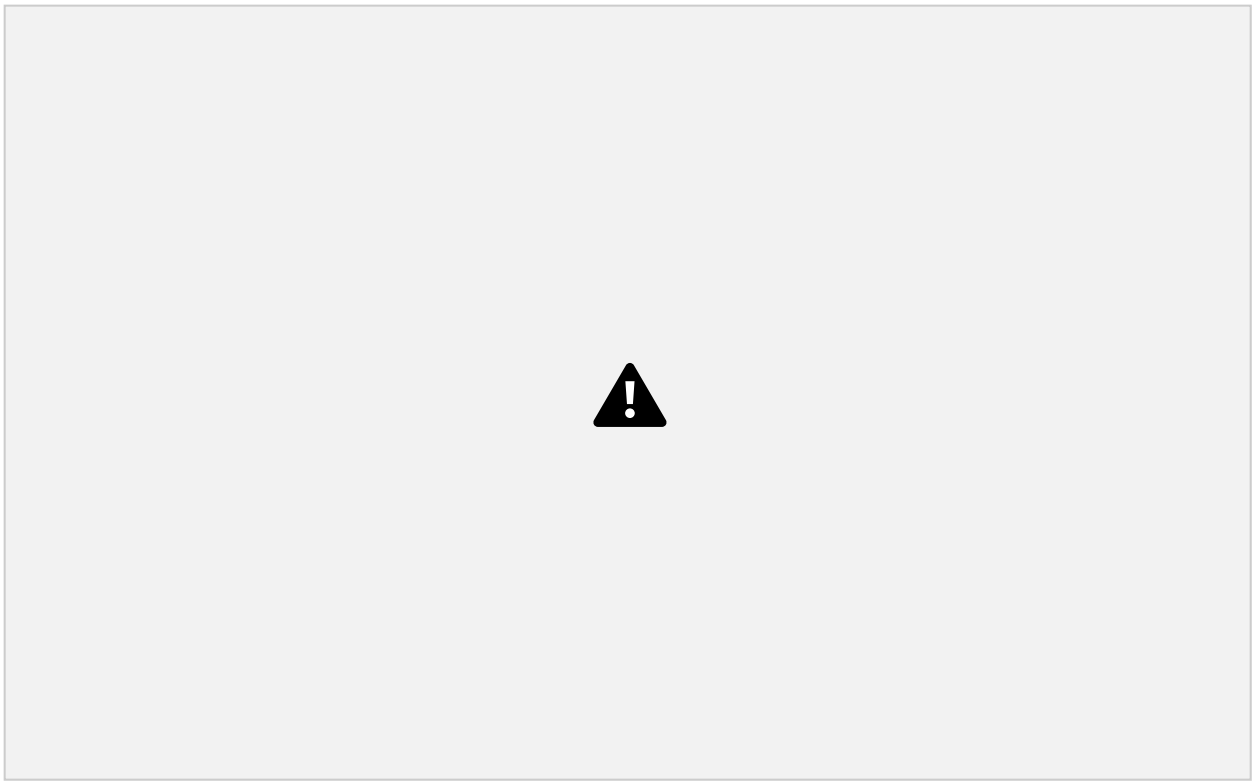


Under the “Series” tab, click through some of the Color Collections and select one that you like. You can also create custom color palettes with your company’s colors.



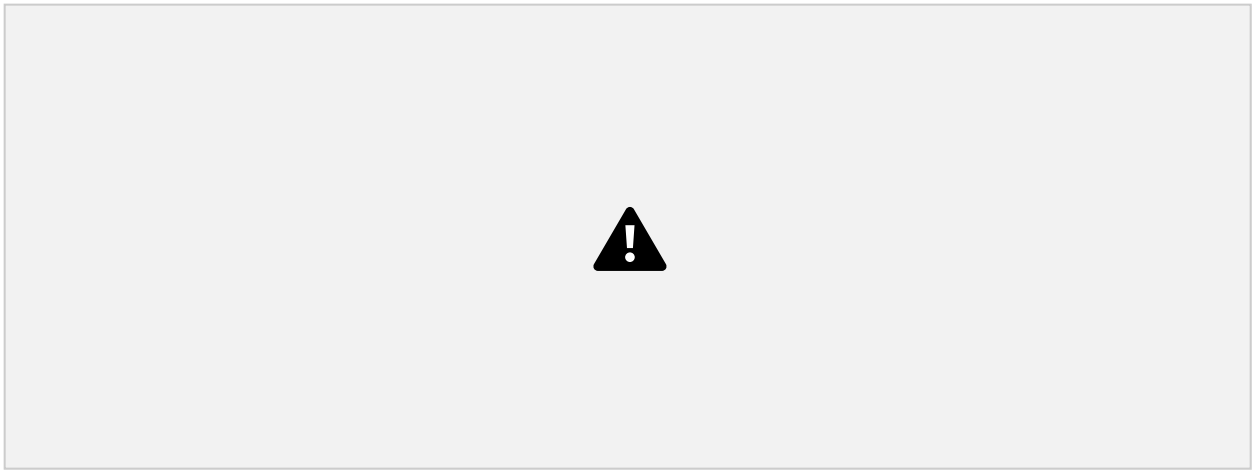
Under the “Values” tab, enable Value Labels.

To hide the chart customization options, click the gear icon again. Your completed visualization should look similar to this:

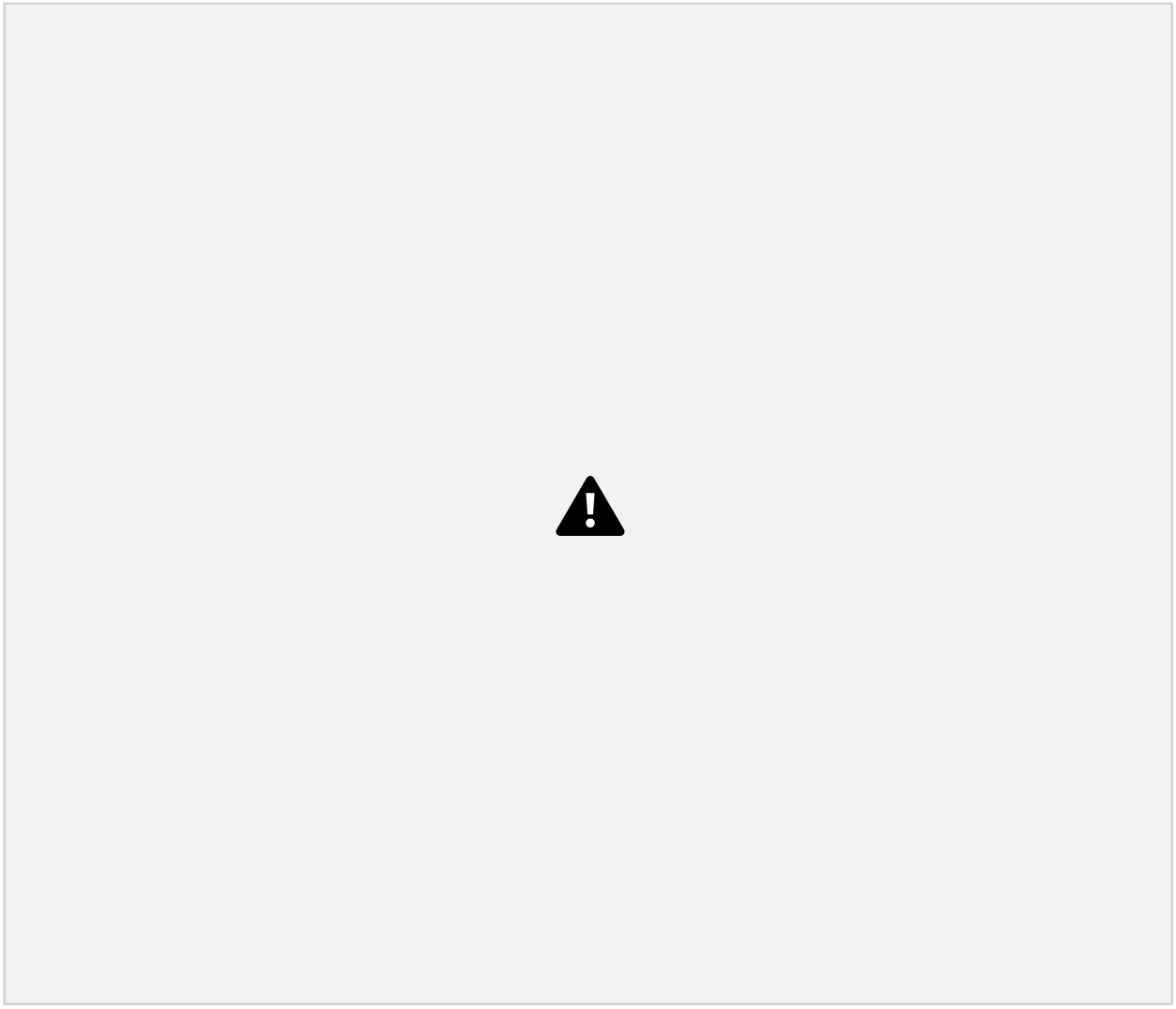


Task #12 - Add Your Visualization To a Dashboard

Save your new visualization to a dashboard. Click the gear icon in the top right corner of the page and select “Save To Dashboard”.



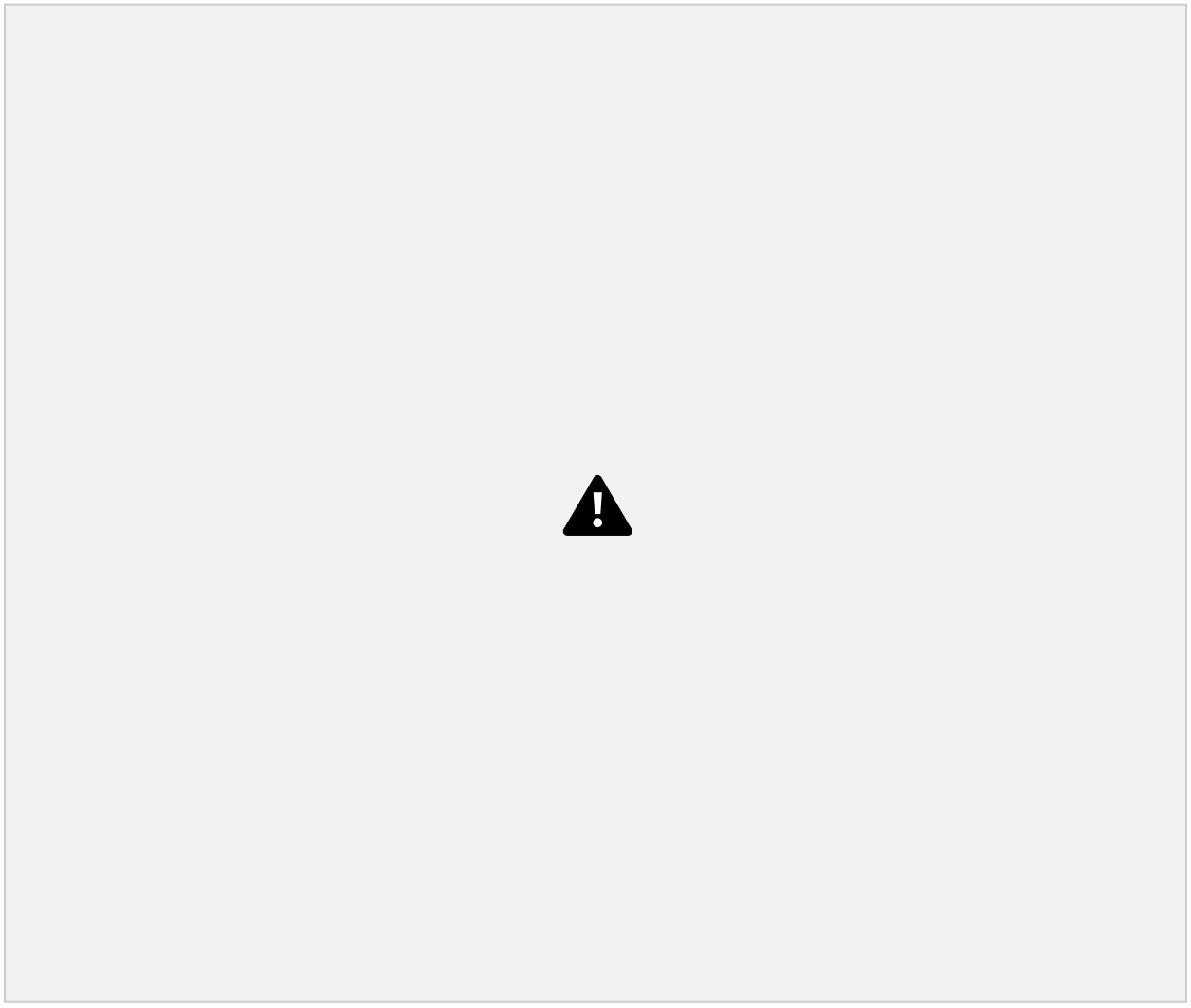
In the “Add to a Dashboard” modal, enter a title for your new dashboard tile. Select where you would like the dashboard to be stored. You should save it into your personal folder by clicking on your User Name from the choices on the left.



Click “New Dashboard” to add your visualization to a new dashboard. Add a name for your new dashboard and click the “OK” button.

Click on the name of your new dashboard if it is not already highlighted and then push the “Save

to Dashboard” button.

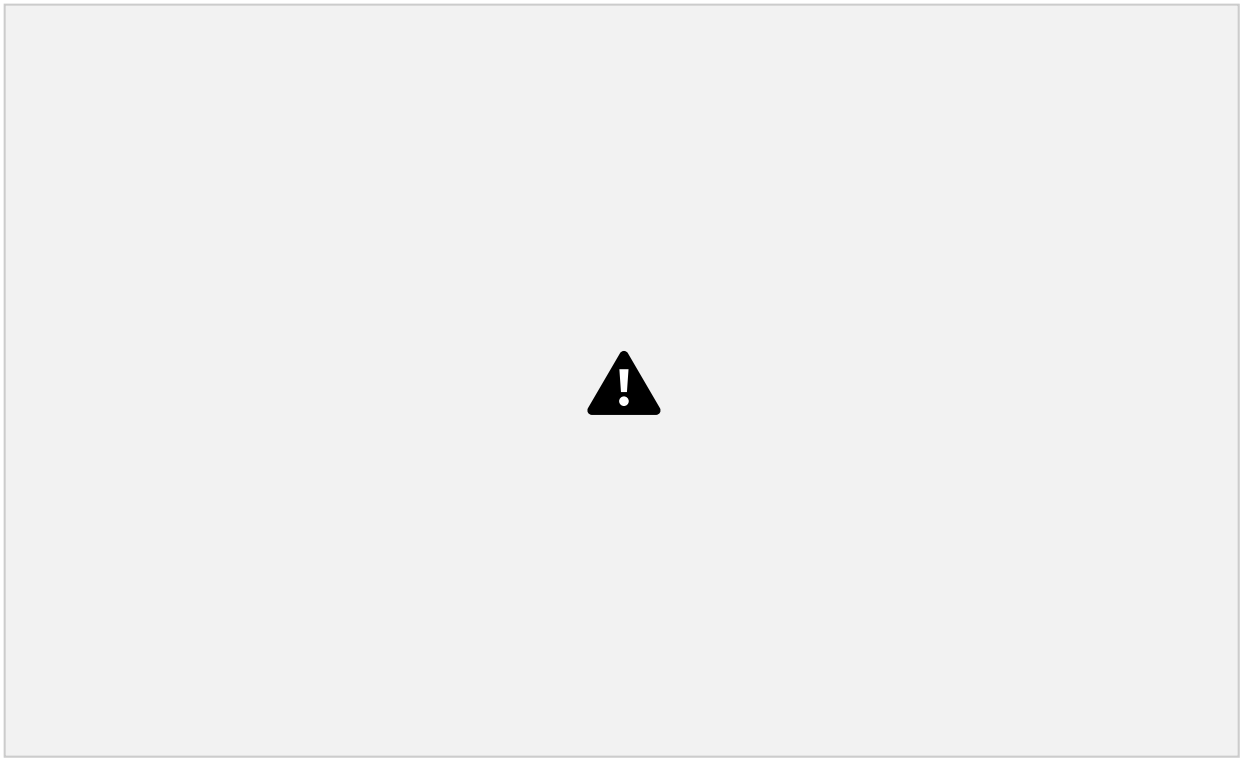


Task #13 - Edit Your New Dashboard

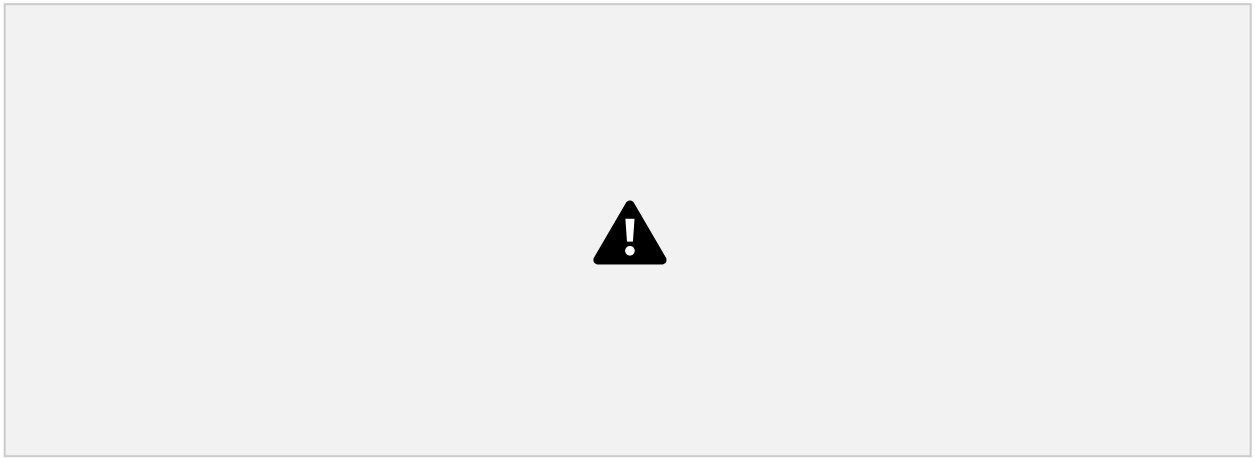
There are a few different ways to access your new dashboard.

You can click the hyperlink in the message indicating that your visualization has been saved to the dashboard:

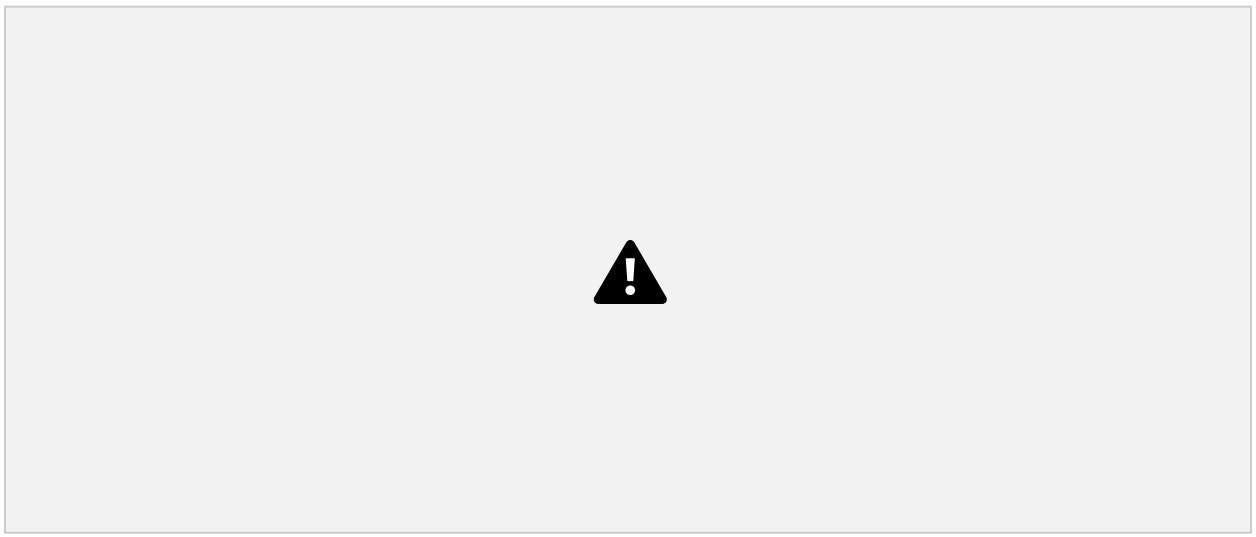
You can also select “Browse” from the main menu and then select your personal folder to see all of the content that you have created.



Your new dashboard should look similar to this:



Click the three dots in the top right to expand the menu and then select “Edit dashboard” option. This allows you to resize, reorder (assuming you have more than one tile), and make changes to an individual tile. Make your dashboard tile bigger by going into Edit Mode, clicking the Resize handle in the lower right corner of the tile, and then stretching it to the size you want. Once you have finished, click the “Save” button to exit Edit Mode.



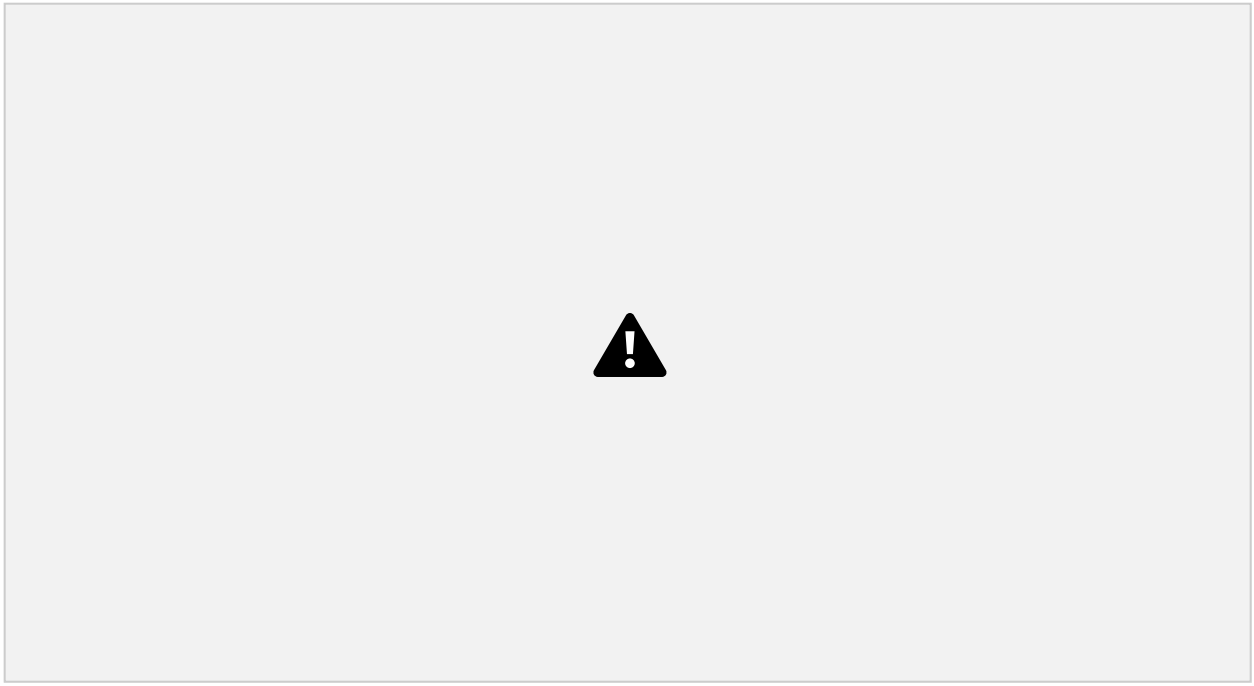
Task #14 - Build Out the LookML Model

Now that you know how to modify a LookML Model, create a visualization and add it to to a dashboard, it's time for you to apply all of the knowledge that you've gained from working with the New York City Taxi data to make your end user's self service reporting experience as smooth as possible.

We'll facilitate this by adding a number of new dimensions and measures to our LookML model. By predefing all of these metrics, we make it easy for users to simply click on the data values they are interested in without having to know how the calculations are performed. This ensures that everyone is using the same definitions for their business metrics.

Notice how you can add a *value_format_name* to fields to ensure that they are formatted appropriately when they are displayed. Also, you'll be building custom dimensions and measures from other custom dimensions and measures.

The following new measures and dimensions will be added to the "trips" view file. 55



You can copy them from this document and paste them into your *trips* view file. Be sure to save your changes after pasting them in.

```
  # The average number of riders per trip
  measure: avg_num_riders {
    type: average
    sql: ${num_riders} ;;
    value_format_name: decimal_2
   }

  # The total number of riders
  measure: total_num_riders {
    type: sum
    sql: ${num_riders} ;;
   }

  #The average fare for a trip. Only include completed rides because they are
the only ones
  #we've actually collected for
  measure: avg_price {
    type: average
    sql: ${price} ;;
    value_format_name: usd
    filters: {
      field: status
      value: "completed"
    }


   }
```

```
  # Calculate the distance using Geospatial functions. This will be straight
line distance in meters
  # so convert it to miles
  dimension: trip_distance {
    type: number
    sql: geography_distance(${pickup_location}, ${dropoff_location}) / 1610 ;;
    value_format_name: decimal_2
  }

  measure: avg_trip_distance {
    type: average
    sql: ${trip_distance} ;;
    value_format_name: decimal_2
  }

  # If the trip is not completed, use 0 as the duration
  dimension: trip_duration_mins{
    type: number
    sql: CASE
            WHEN ${status} = 'completed' THEN (${dropoff_time} -
${pickup_time}) / 60
            ELSE 0
          END;;
  }

  # Only include trips that have been completed
  # Otherwise the duration is unknown
  measure: avg_trip_duration_mins {
    type: average
    sql: ${trip_duration_mins} ;;
    value_format_name: decimal_2
    filters: {
      field: status
      value: "completed"
    }
  }

  #How long did the passenger wait from the time they requested the ride until
they were picked up?
  #If the driver is enroute, this is unknown.
  dimension: wait_time_mins {
    type: number
    sql: CASE
            WHEN ${status} = 'enroute' THEN NULL
          ELSE (${pickup_time} - ${request_time}) / 60
          END;;
```

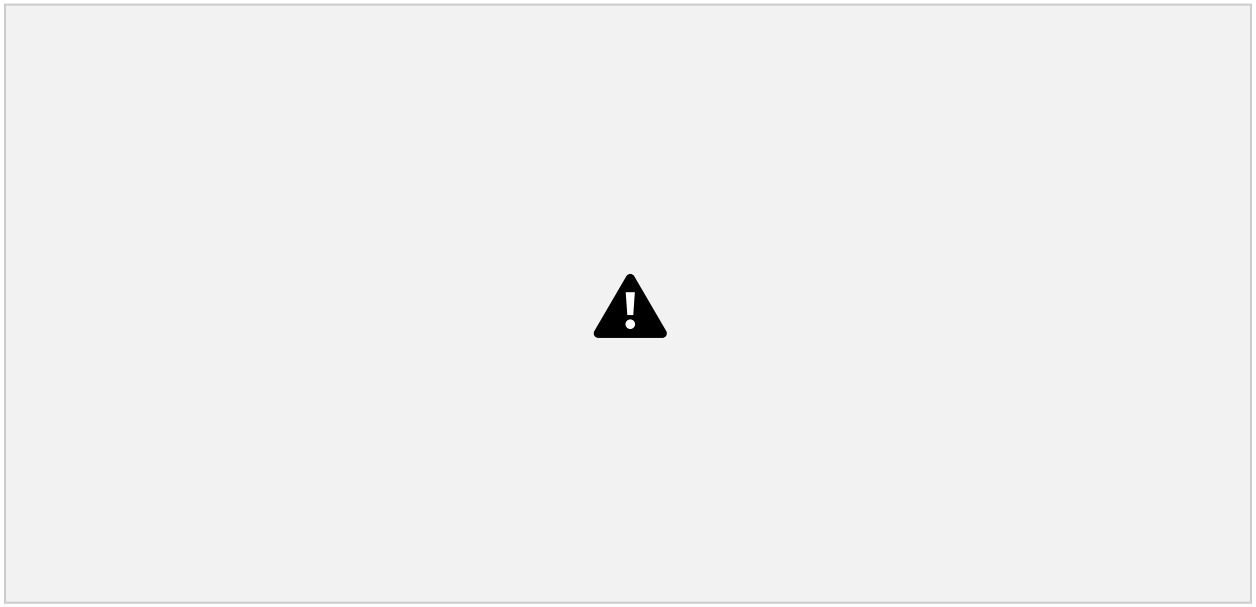
```
  }

  # Don't include enroute trips in the average wait time, since the value is
unknown
  measure: avg_wait_time_mins {
    type: average
    sql: ${wait_time_mins} ;;
    value_format_name: decimal_2
    filters: {
      field: status
      value: "-enroute"
    }
  }

  # Custom Dimensions for Looker mapping
  # Looker requires a special field type of “location” for
  # mapping longitude and latitude coordinates

  # Use SingleStore geospatial function to extract the latitude from the
dropoff location
  dimension: dropoff_latitude {
    type: number
    sql: geography_latitude(dropoff_location) ;;
  }

  # Use SingleStore geospatial function to extract the longitude from the
dropoff location
  dimension: dropoff_longitude {
    type: number
    sql: geography_longitude(dropoff_location) ;;
  }

  # The longitude and latitude values are extremely precise
  # We’ll round them to improve the clustering of the data
  dimension: dropoff_latitude_rounded {
    type: number
    sql: ROUND(${dropoff_latitude}, 3) ;;
  }

  # The longitude and latitude values are extremely precise
  # We’ll round them to improve the clustering of the data
  dimension: dropoff_longitude_rounded {
    type: number
    sql: ROUND(${dropoff_longitude}, 3) ;;
  }

  # Create the new Looker location field using the rounded
```

58

```
# longitude and latitude

dimension: dropoff_map_location {
type: location
sql_latitude: ${dropoff_longitude_rounded} ;;
sql_longitude: ${dropoff_latitude_rounded} ;;
}
```

Task #15 - Build The Rest of Your Dashboard

Use the recipe cards to build out the rest of your dashboard tiles. Feel free to experiment! Your dashboard doesn't need to match the example. If you follow the recipe cards, your finished dashboard should look like the:



59

Once you have created a visualization and saved it to your dashboard, you can simply select "Remove Fields and Filters" to clear your palette and start working on the next visualization.

Completed Trips

Title Completed Trips

Model Name	Dev Days Taxi Basic Name
Explore	Trips

Dimensions None

Measures Trips - Count

Vis Type Single Value

Filter 1 Trips - Status is equal to completed 60

Drivers Enroute

Title Drivers Enroute Model Name

Dev Days Taxi Basic

Explore Name Trips

Dimensions None

Measures Trips - Count

Vis Type Single Value

Filter 1 Trips - Status is equal to enroute

Rides In Progress

Title Rides In Progress

Model Name Dev Days Taxi Basic
Explore Name Trips

Dimensions None

Measures Trips - Count

Vis Type Single Value

Filter 1 Trips - Status is equal to driving

Current Passenger Count

Title Current Passenger Count

Model Name Dev Days Taxi Basic
Explore Name Trips

Dimensions None

Measures Trips - Total Num Riders

Vis Type Single Value

Filter 1 Trips - Status is equal to driving

Dropoff Activity

Title Dropoff Activity

Model Name Dev Days Taxi Basic
Name
Explore Trips

Dimensions Dropoff Neighborhoods - Borough

Measures Trips - Count

Vis Type Pie Chart

Filter 1 None

Edit - Plot Inner Radius = 40

Pickup Activity

Title Pickup Activity

Model Name Dev Days Taxi Basic
Name
Explore Trips

Dimensions Pickup Neighborhoods - Borough

Measures Trips - Count

Vis Type Pie Chart

Filter 1 None

Edit - Plot Inner Radius = 40

Map

Title Crosstown Traffic - Top 15 Drivers

Model Name Dev Days Taxi Basic
Name
Explore Trips

Dimensions Drivers - Full Name

Measures Trips - Count

Vis Type Column Chart

Filter 1 Pickup Neighborhoods - Borough is equal to Manhattan Filter 2 Dropoff

Neighborhoods- Borough Is not equal to Manhattan Edit - Series Color Collection -

Dalton

Edit - Values Enable Value Labels

Cleanup - Terminate AWS Resources

One you have completed the Workshop (we will leave the Looker Service and SingleStore Group Cluster up and running for one week after this session):

Run:

```
SingleStore> source stop_nyctaxi_pipelines.dml;
```

Run:

To stop the stored procedure, use Cntl-C from derive_stats CloudShell terminal or from SingleStore Studio “Process” tab

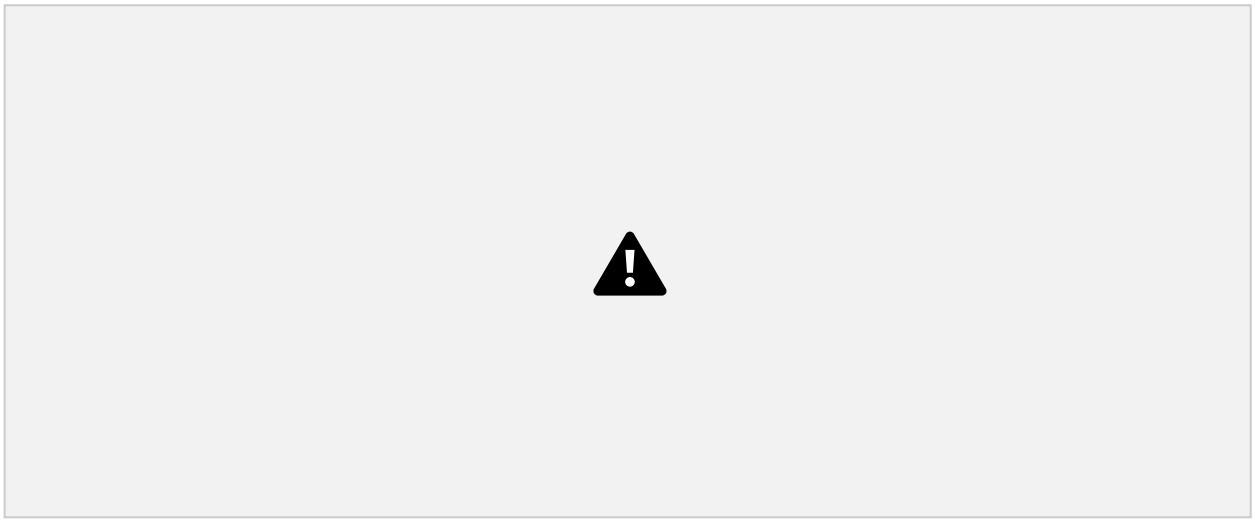
Run:

From AWS Cloud Formation services console, **Delete** your stack. Also, delete your CloudShell session and AWS Glue Resources

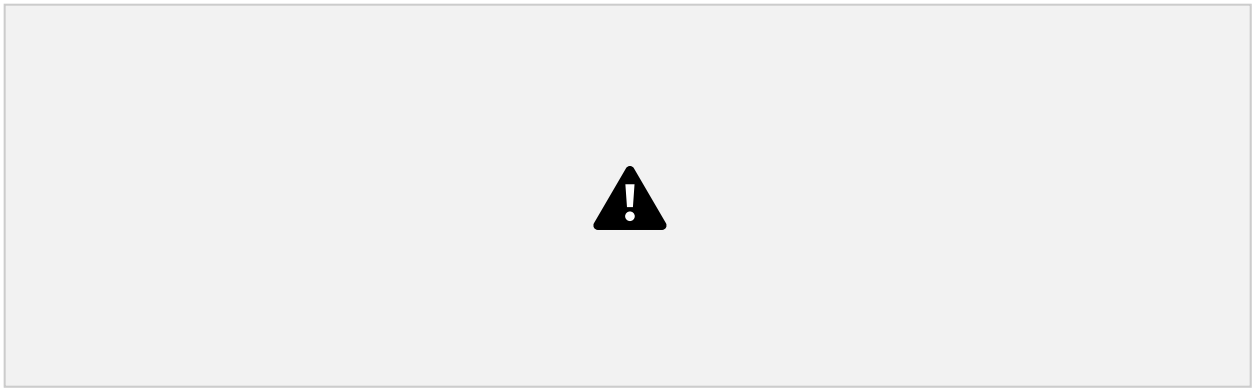
APPENDIX

Launch an AWS SageMaker Notebook session

To launch the SageMaker Service, make sure you are logged into AWS and then navigate to the AWS Services Console.



Click on "**Amazon SageMaker**". You should be taken to the following screen:



Then, click on **Create notebook Instance** in the above screen:

You need to update the **following parameters** in the **SageMaker Cloud Formation Template**:

Notebook instance settings:

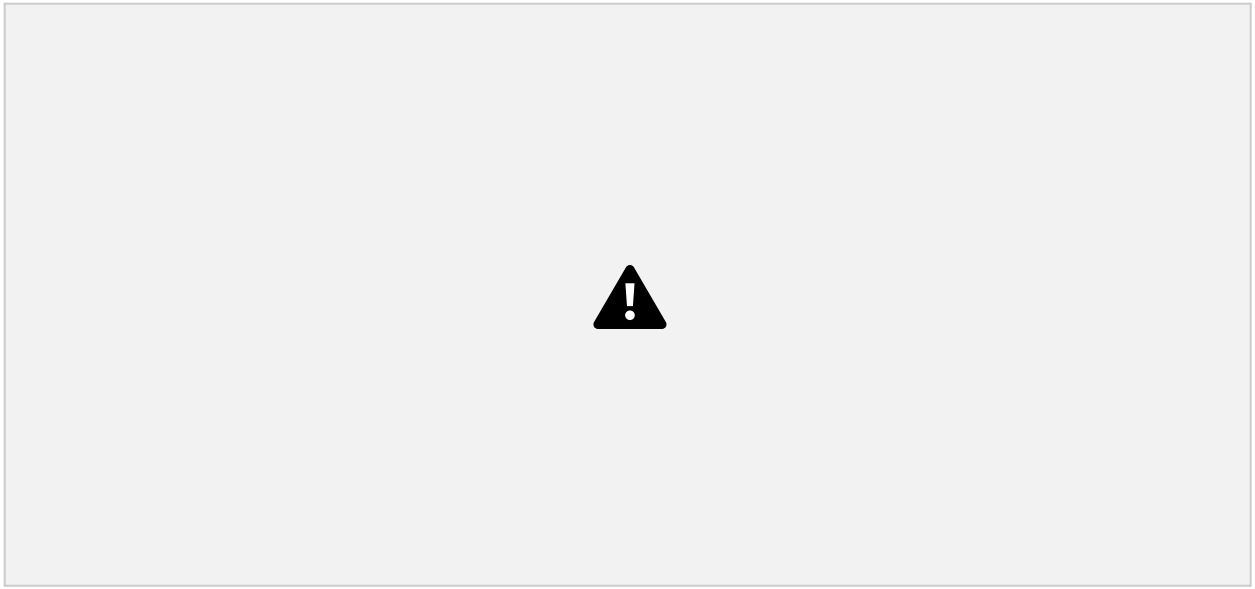
Notebook instance name = <Make up a Name for your Notebook>

Suggestion: RandomCutForestML

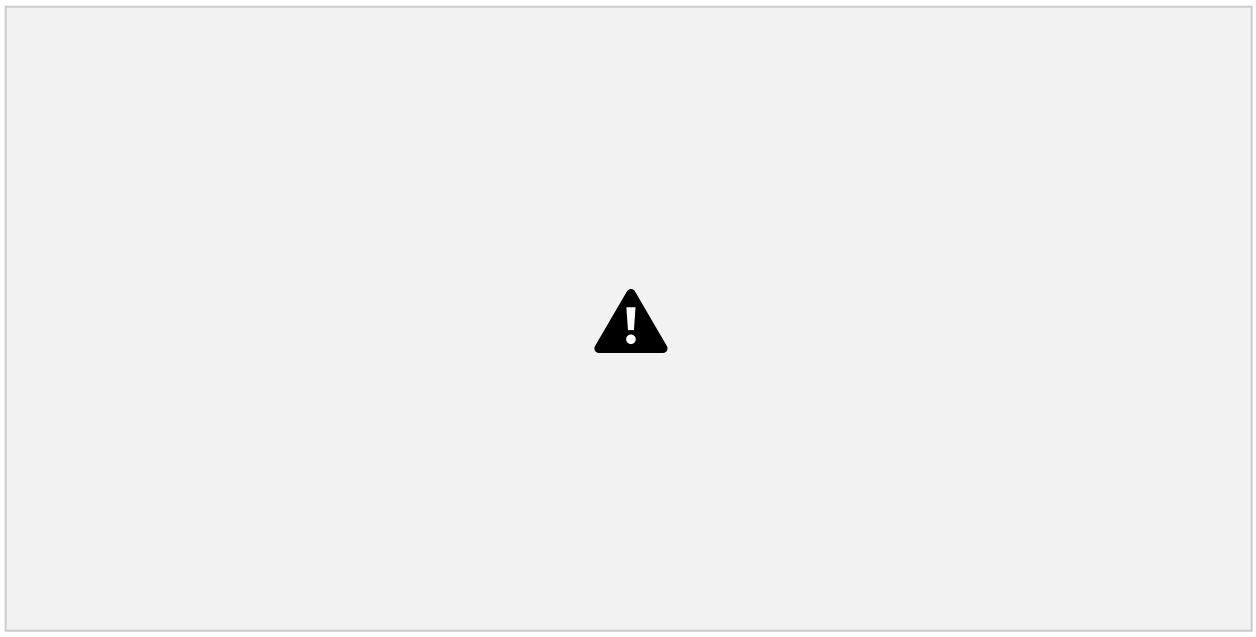
Permissions and encryptions:

IAM Role = <Create a new role>

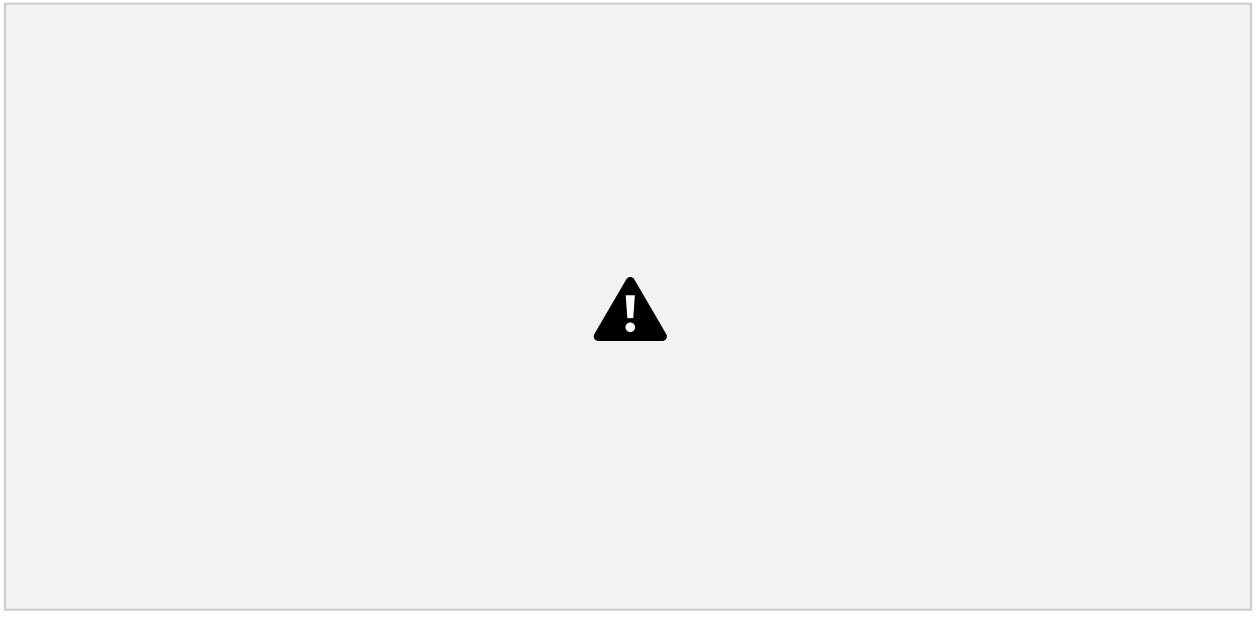
Select **"Create a new role"** using the dropdown below



Make sure to select **"Any S3 Bucket"**, and then hit the Create role button as seen below.



At this point, you should see **“Success! You created an IAM role”** (If you are using an AWS Company ID, contact your AWS Account administrator for more help if this does not work at your site)



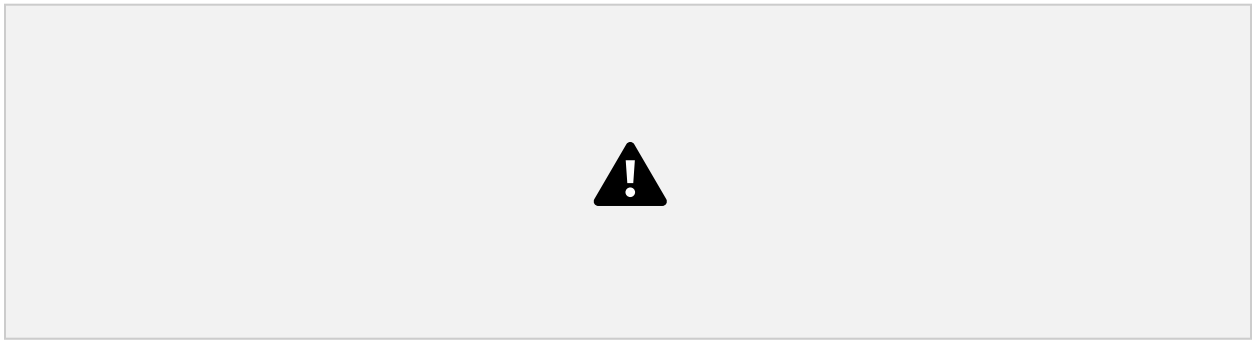
Continue filling in the following parameters:

Root Access = Choose "**Enable**"

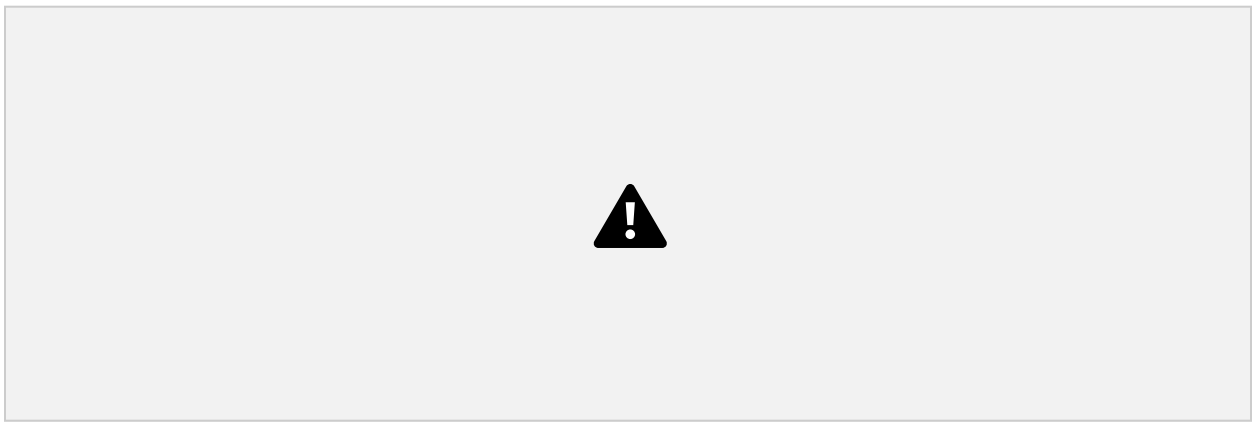
Add **Tags** to your Notebook so that you can find this instance in the EC2 Services Console. Here are some tag suggestions:

Owner = your_first_initial_last_name
Project = "AWSDevDays"

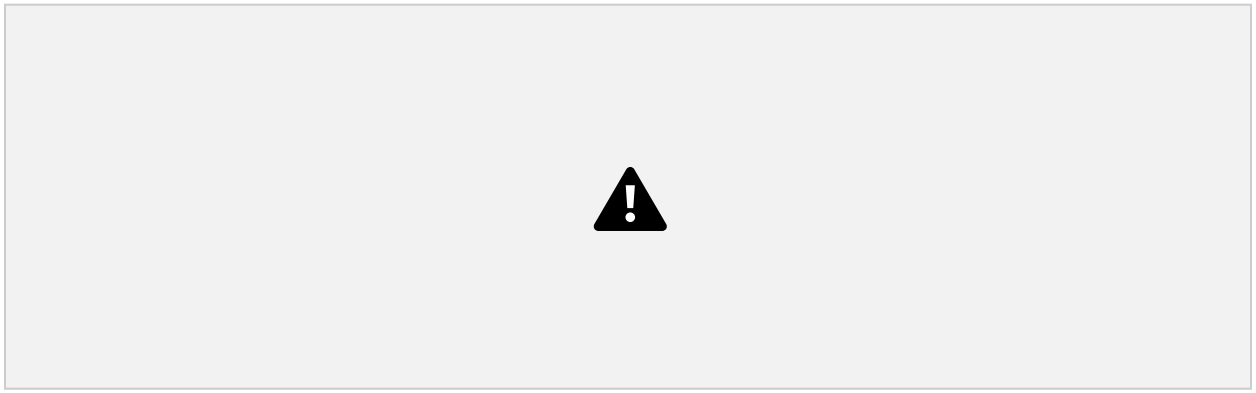
After entering the Tags, click on **Create notebook instance** as seen below:. 68



You will see the following screen this note: **Success! Your notebook is being created.**



When your notebook has been created, you will output similar to the screen below:



You should now be able to double click on the **Open Jupyter** under **Actions** as seen above.

We will use the Numenta Anomaly Benchmark (NAB) NYC Taxi dataset [2]. 69

Build, Train and Deploy a Machine Learning Model

Create a SageMaker Notebook Instance and Select a ML Project

During this step, we will be deploying a SageMaker Notebook and then following AWS sample project to train a machine learning model. This exercise will allow us to deploy and a SageMaker Restful API Inference endpoint for future consumption by SingleStore

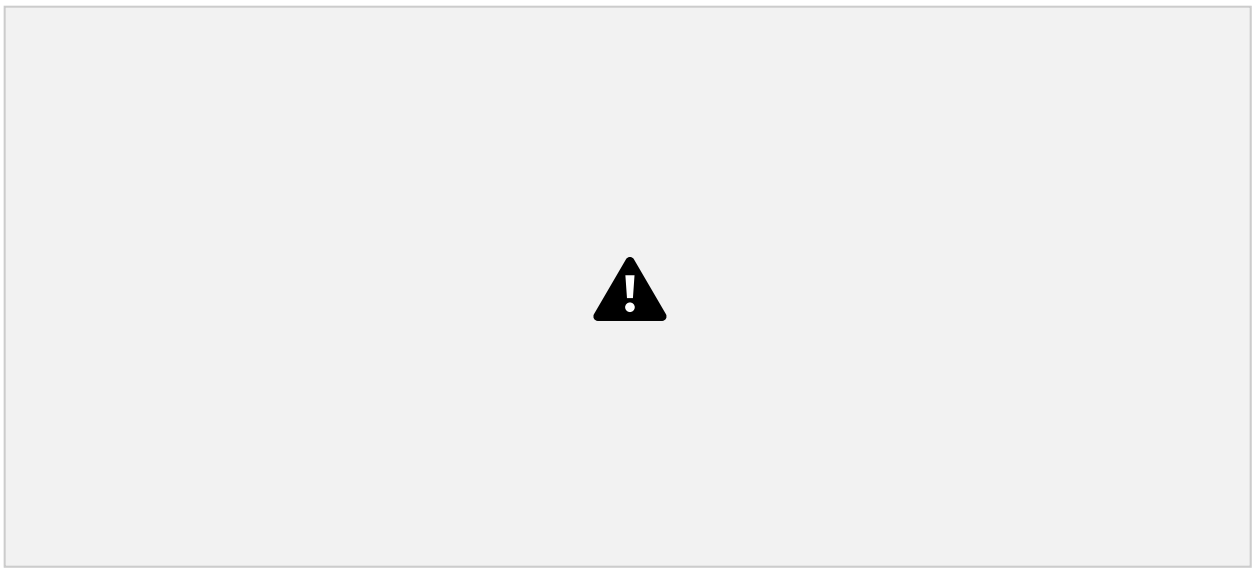
Pipelines. Please take a moment to review the goal of this Data Scientist training exercise using this link:

https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/random_cut_forest/random_cut_forest.ipynb

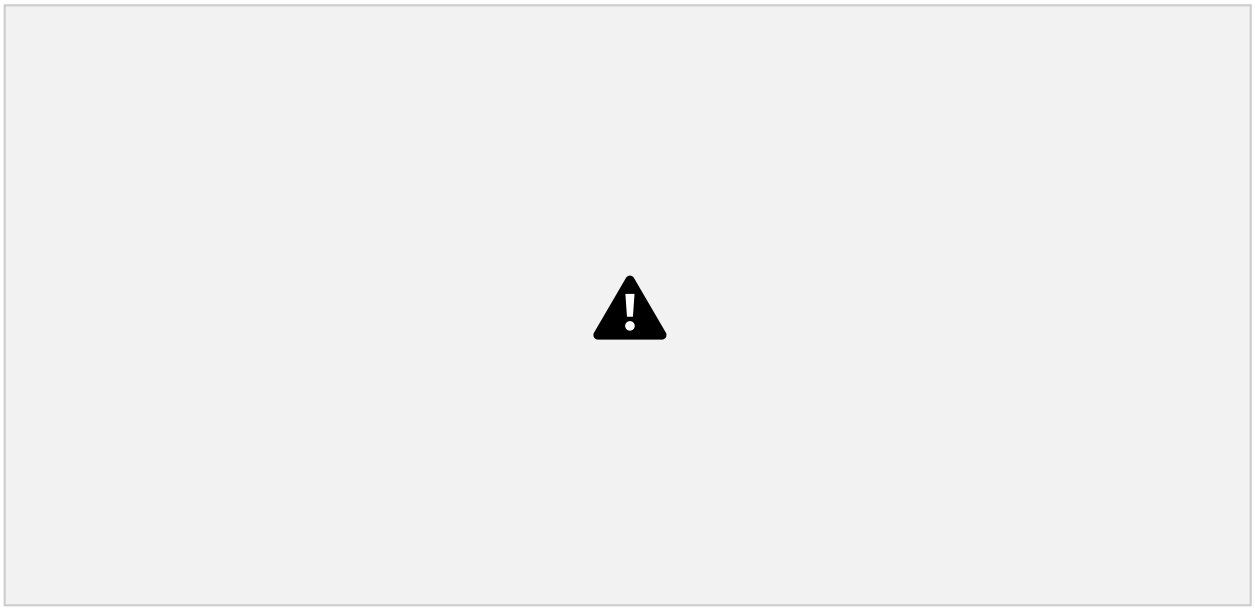
Build, Train, and Deploy a Random Cut Forest ML Model

You should be seeing the following screen at this point:

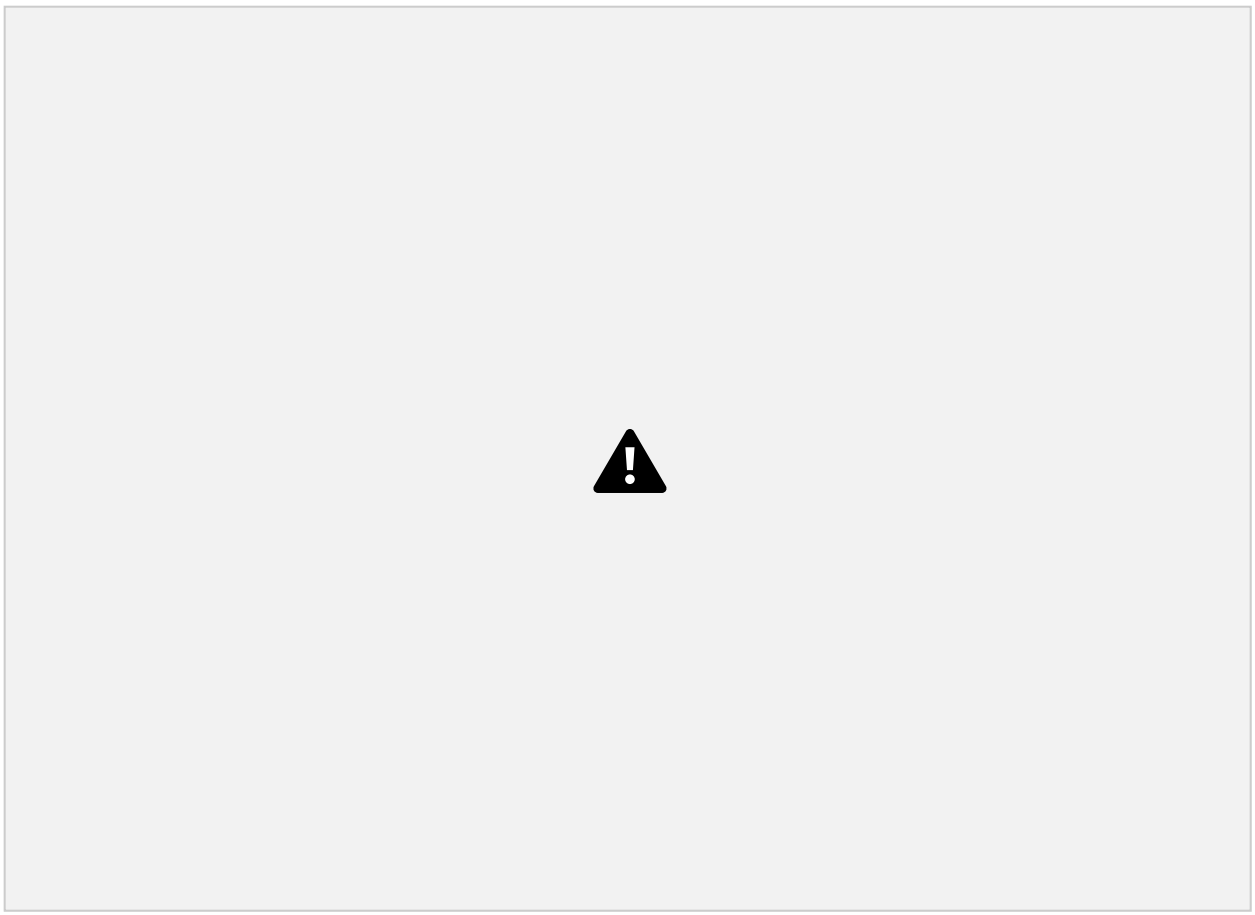
Click on SageMaker Examples and then scroll down until you can see **random_cut_forest.ipynb** similar to the screen below:



Click on the Use bottom on the right hand side of **Preview** for the **random_cut_forest.ipyb** example above and the following pop-up screen should be displayed:



From the above screen, click on Create Copy. You should now be able to **Open** the Jupyter notebook. Once it is open, you will see the following screen below. Take a moment to review the **Introduction** to understand the purpose of this training exercise:

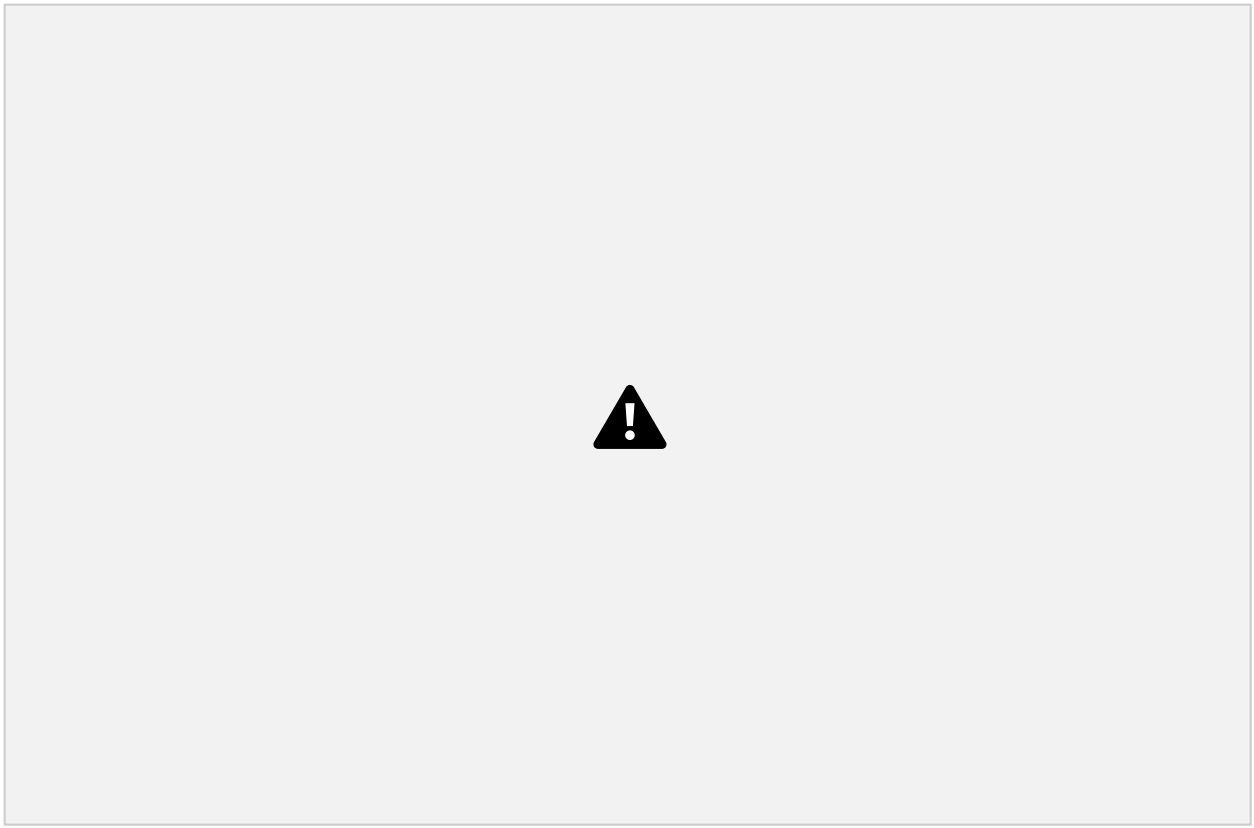


Now, scroll down until you see the section labeled **Select Amazon S3 Bucket**. We will need to edit the bucket and prefix to point to the following public S3 bucket and directory.

bucket = 'lochbihler'
prefix = 'sagemaker'

Now, make sure this section is highlighted with a green border (click on it) and use the **>Run** button at the top of the screen to execute this code snippet. When it has completed, make sure you see the output below indicating it found the public S3 bucket.

Sample Output:



Checking AWS cli and SageMaker Library Versions

We need to make sure that the AWS cli and SageMaker libraries are the right versions (at least ***AWS cli 1.16.259*** and ***SageMaker 1.42.9)***. To do this, use the following commands by adding them to the bottom of the code section above, and rerun the section.

#Check awscli version

```
!aws --version
```


Sample Output:

```
aws-cli/1.16.259 Python/3.6.5 Linux/4.14.146-93.123.amzn1.x86_64
botocore/1.12.249
```

#Check SageMaker Version

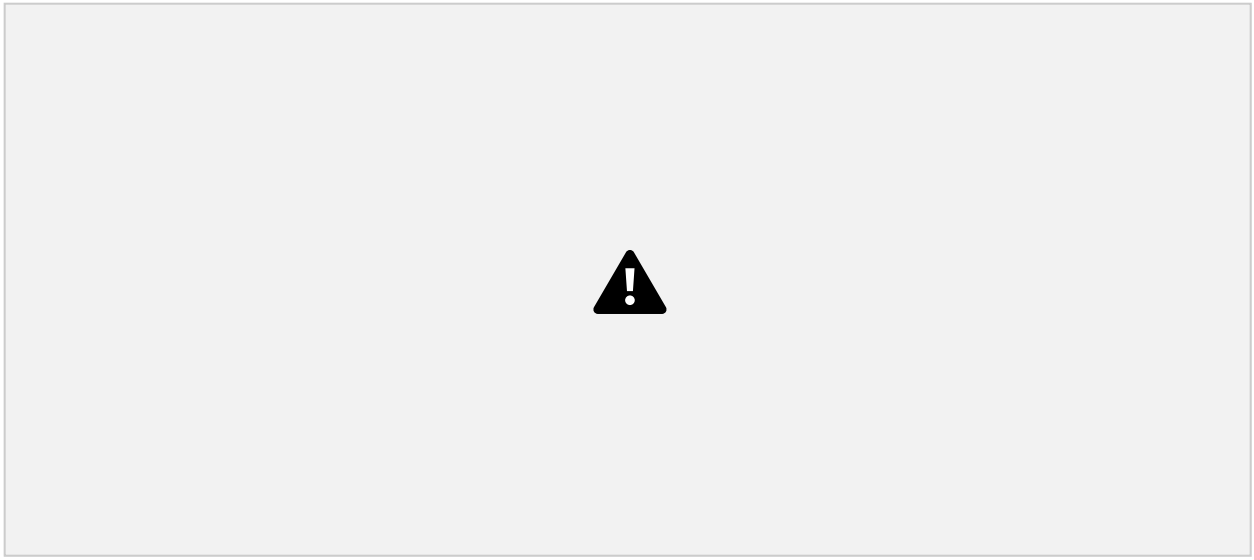
```
version = sagemaker.__version__
print ('sagemaker Version {}' .format(version))
```

Sample Output:

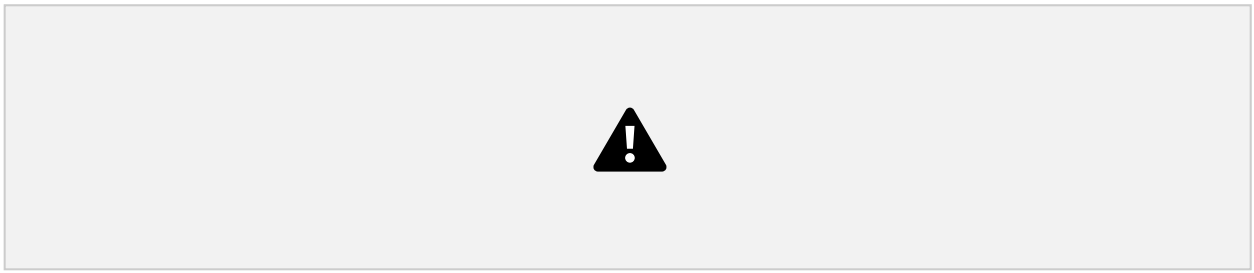
```
SageMaker Version 1.42.9
```

```
#If necessary…..
      #Upgrade awscli to latest release
      !pip3 install awscli --upgrade --user
#If necessary….
      #Upgrade SageMaker sdk to latest release
      !pip3 install SageMaker --upgrade

      #Reload SageMaker sdk
      import importlib
      importlib.reload(sagemaker)

#Optionally list all installed packages
!pip3 list -o
```

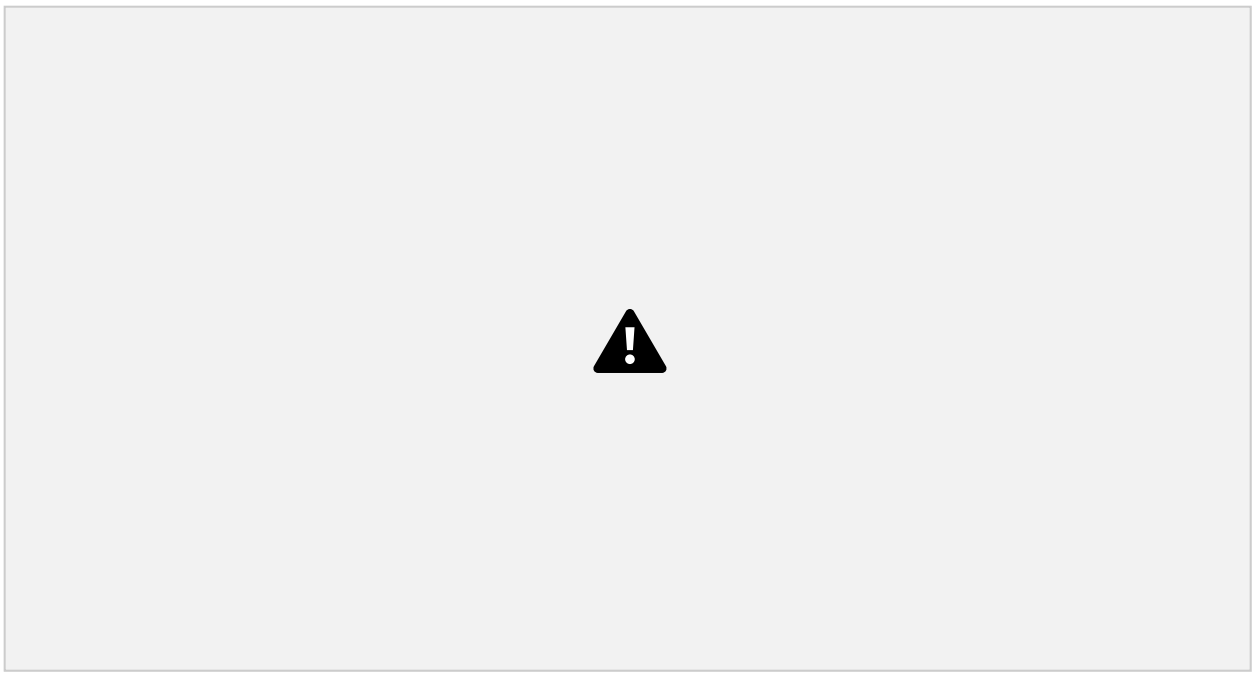

Here is an example of updating these libraries in the first section of this AWS notebook:



Once you have verified the AWS cli and SageMaker libraries are the correct versions, please continue to the next step by scrolling down until you can see this section:

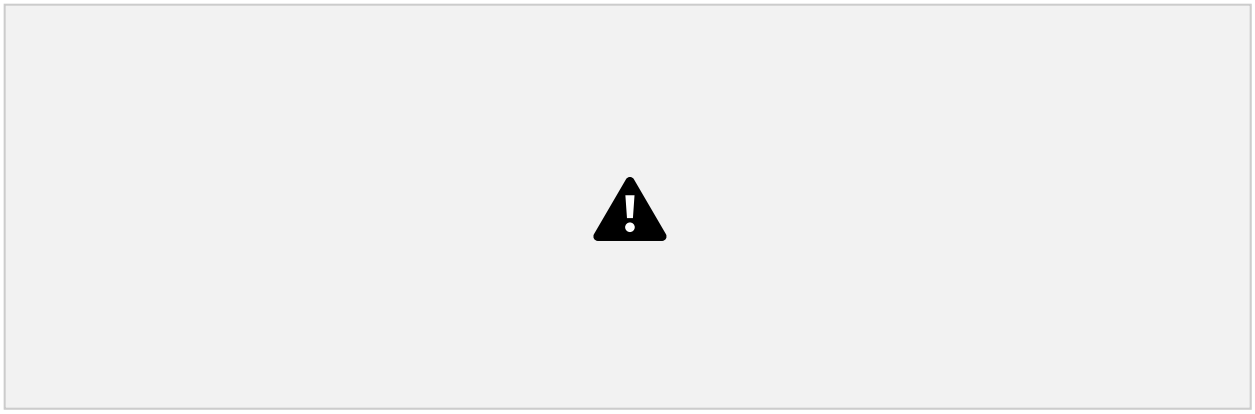


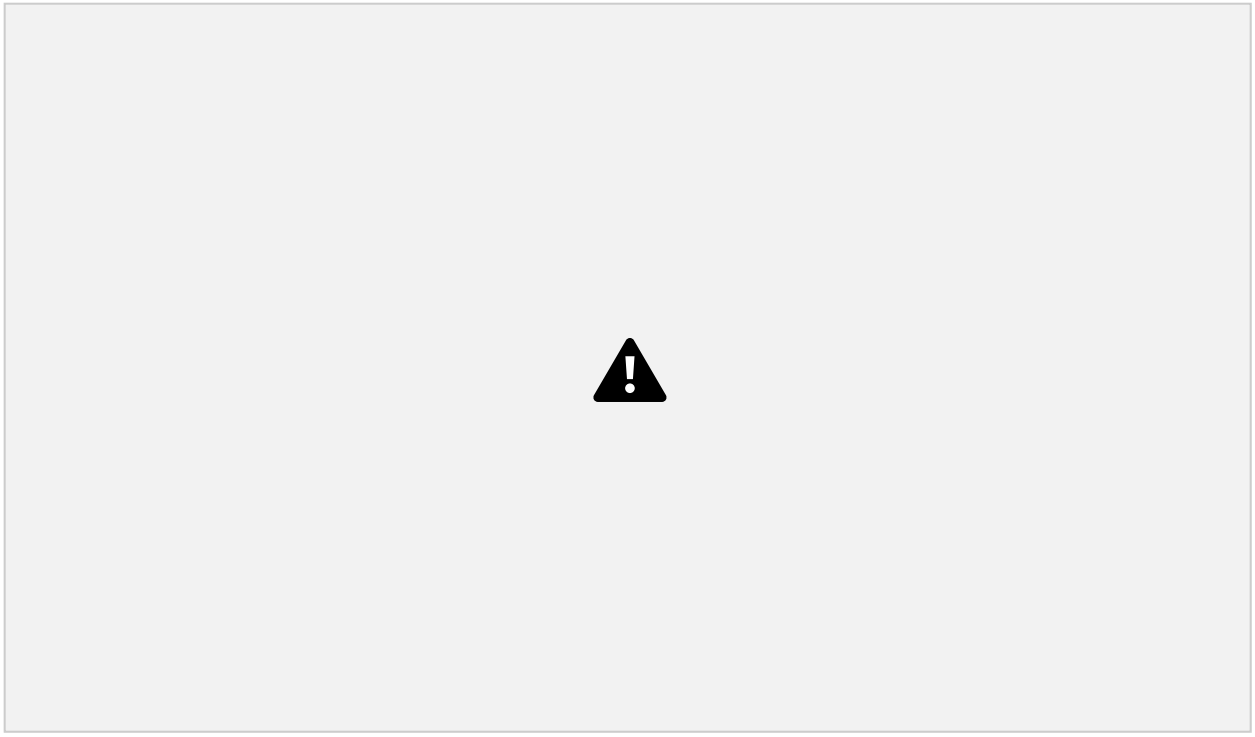
Click on the URL referenced above and inspect the nyc_taxi.csv file that we will be using to train our model. It will open up in a new tab in your browser and should appear similar to the screenshot below:

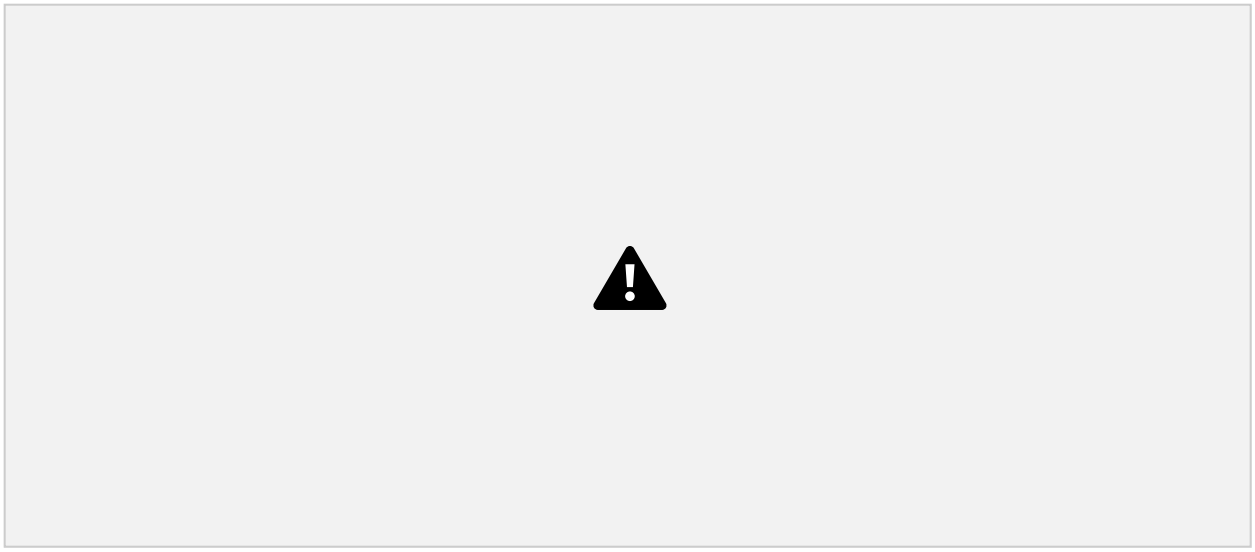


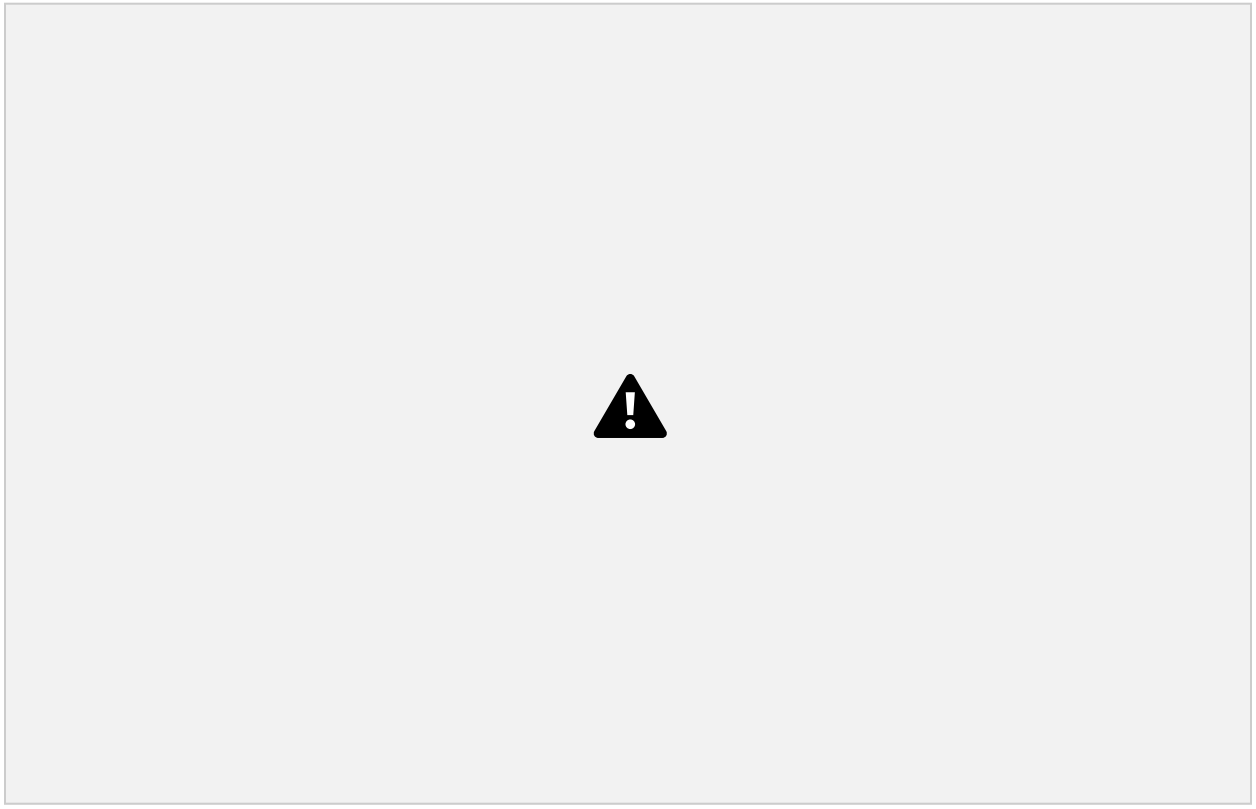
Continue to the next section. Inspect and then run it and look for similar output as below:

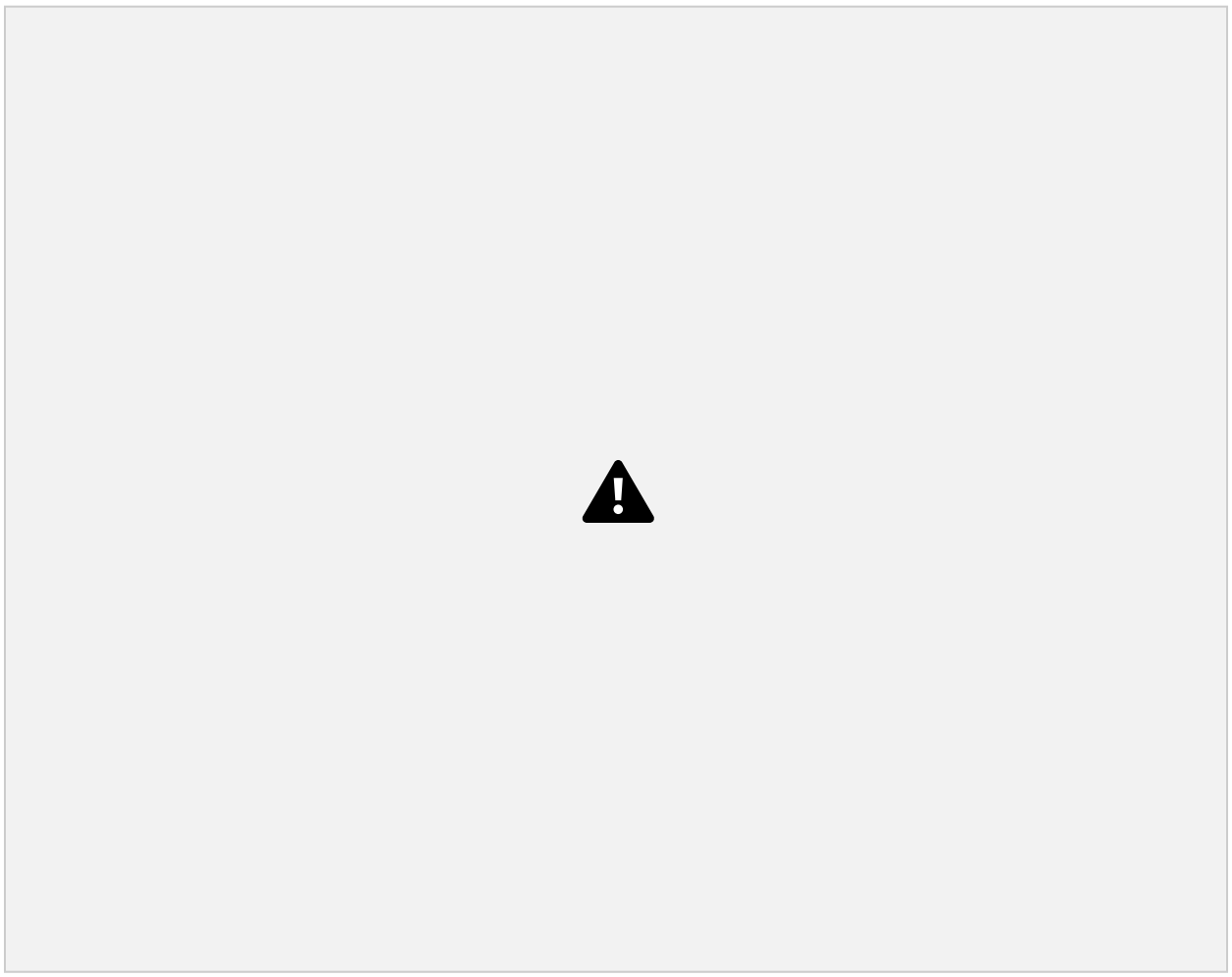
Continue to the next sections and run it as well.

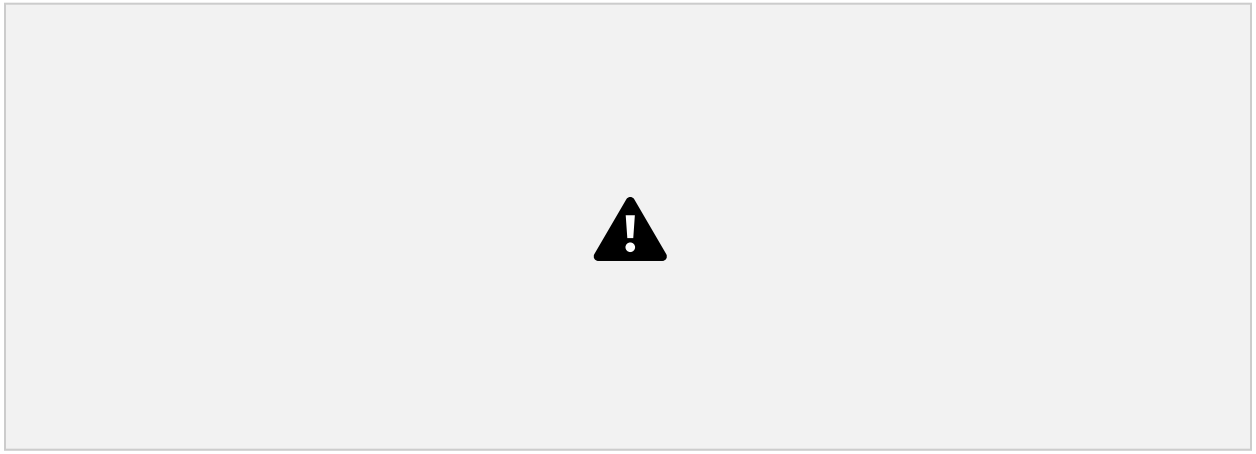












At this point, please navigate to the **Amazon SageMaker > Models** panel. You should see your model referenced similar to the screenshot below: