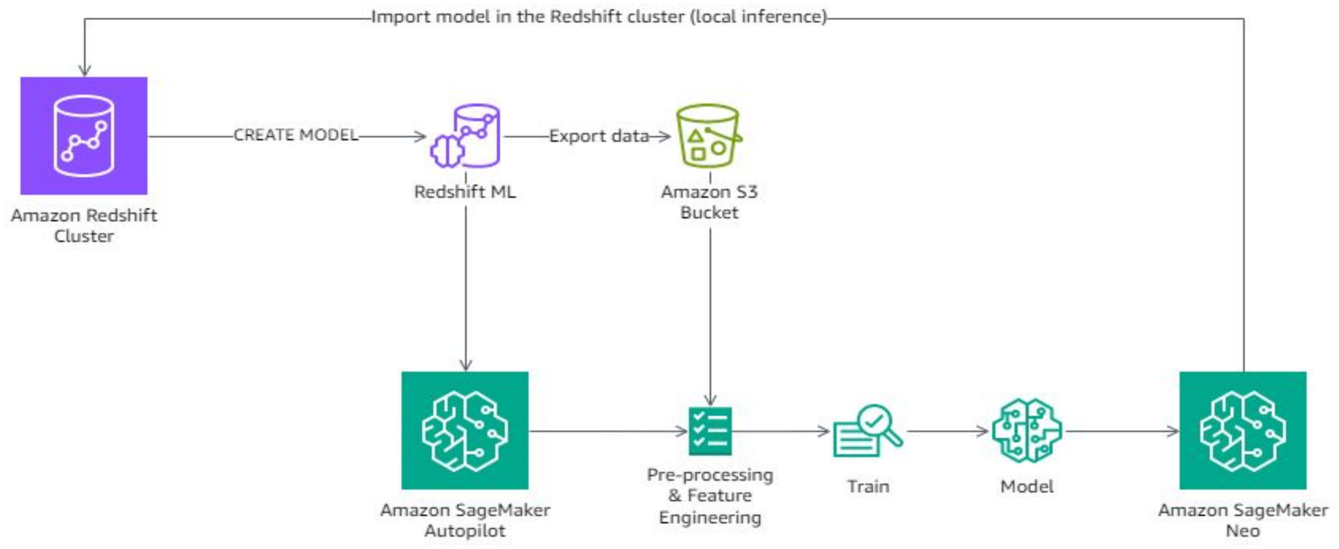
reate an ML model using Amazon Redshift ML. You use the CREATE MODEL query to create a model using Amazon SageMaker Autopilot and then perform predictions with the model.

Objectives

By the end of this lab, you should be able to do the following:

- Create a machine learning model using Amazon Redshift ML.
- Perform predictions with a model.


Import model in the Redshift cluster (local inference)
Amazon Redshift Cluster
CREATE MODEL
Redshift ML
Export data
Amazon S3 Bucket
Amazon SageMaker Autopilot
Pre-processing & Feature Engineering
Train
Model
Amazon SageMaker Neo

Create a machine learning model using Amazon Redshift ML.

With Amazon Redshift ML, you create, train, and deploy ML models using familiar SQL commands. You can use your data in Amazon Redshift with Amazon SageMaker, a fully managed ML service, without requiring ML expertise.

Redshift ML supports supervised machine learning, a commonly used technique in enterprises for advanced analytics. Supervised learning is useful when you have a dataset that includes both features - the input data - and labeled targets - the output data. In this lab, the TICKIT database contains information about ticket purchase transactions, the features. It also includes the number of tickets purchased per transaction, the target. Having both the input features and output target labels makes this an ideal dataset for supervised learning.

Connect to workgroup

At the top of the AWS Management Console, in the search bar, search for and choose `Amazon Redshift`.

On the Serverless dashboard page, choose **Query data**.

In the new browser tab, you might get the following caution in a red color banner: User information couldn’t be retrieved. You must have an account to use Redshift Query Editor V2. Choose x to close the prompt.

To create an account to use Redshift Query Editor V2, on the AWS KMS encryption page, leave all the values at the default settings and choose **Configure account**. The query editor now opens in a new browser tab.

In the Redshift query editor v2 pane, choose the vertical ellipses (three dots) at Serverless:labworkgrp.

Choose Create connection.

On the Connect to labworkgroup window prompt, choose AWS Secrets Manager.

Select Choose a secret drop-down menu and choose redshift!labnamespace-dbadmin.

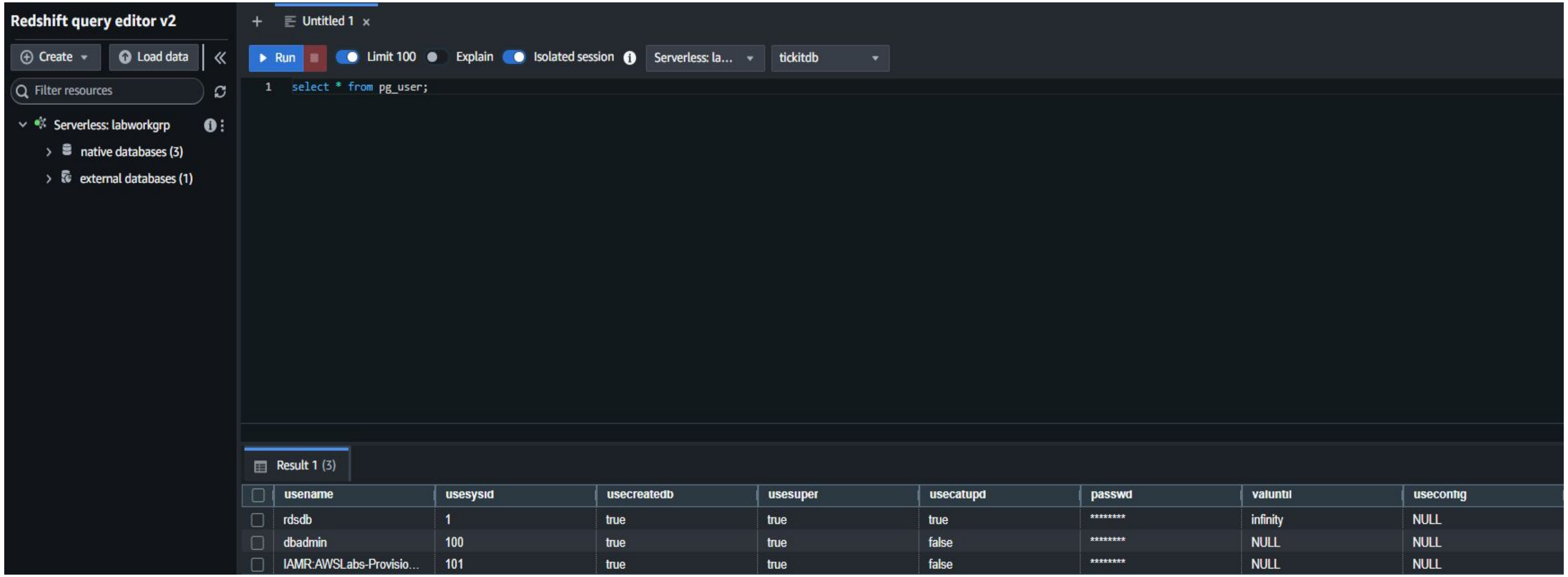
Choose **Create connection**.

On the top right, choose the newly created tickitdb database instead of the *dev* database.

To list the users

select * from pg_user;

The *dbadmin* user has already been granted SELECT on the tables, CREATE, and USAGE ON SCHEMA. These permissions have been granted by having true values for the *usecreatedb* and *usesuper* columns. The permissions allow your user to create models and query using the ML inference functions on the schemas.



```
select * from pg_user;
```

usename	usesysid	usecreatedb	usesuper	usecatupd	passwd	valuntil	useconfig
rdsdb	1	true	true	true	********	infinity	NULL
dbadmin	100	true	true	false	********	NULL	NULL
IAMR:AWSLabs-Provisio...	101	true	true	false	********	NULL	NULL

To list the pre-loaded sample tables, enter the following query:

```
SELECT distinct tablename FROM pg_table_def WHERE schemaname='public';
```

Expected output:

```
************************

**** EXAMPLE OUTPUT ****

************************



Result 1 (7)

-------------



tablename

-----------

category

date

event

listing

sales

users

   venue
```

Create a table that joins the important users sales data together. The users data, including the city, state, likes, and the categories the users ended up purchasing tickets for are important features to include. Also, include the quantity of tickets sold, the average price paid per ticket, and the average commission paid per ticket.

For the category column, perform one-hot encoding to create columns for each of the major event categories. One-hot encoding turns categorical columns like catname into several columns with numerical values (0 and 1). Converting a categorical column to numerical columns can improve an ML model’s accuracy.

To create the users sales table for the ML model, enter the following query:

CREATE TABLE users_sales_ml AS SELECT users.userid, users.city, users.state, users.likesports, users.liketheatre, users.likeconcerts, users.likejazz, users.likeclassical, users.likeopera, users.likerock, users.likevegas, users.likebroadway, users.likemusicals,

COUNT(CASE WHEN category.catname = 'Musicals' THEN 1 END) AS musicals, COUNT(CASE WHEN category.catname = 'Plays' THEN 1 END) AS plays, COUNT(CASE WHEN category.catname = 'Opera' THEN 1 END) AS opera, COUNT(CASE WHEN category.catname = 'Pop' THEN 1 END) AS pop,

ISNULL(SUM(sales.qtysold), 0) AS tickets_sold, ISNULL(AVG(sales.pricepaid / sales.qtysold), 0) AS average_price_paid_per_ticket, ISNULL(AVG(sales.commission / sales.qtysold), 0) AS average_commission_per_ticket

FROM users

LEFT OUTER JOIN sales ON sales.buyerid = users.userid

LEFT OUTER JOIN event ON sales.eventid = event.eventid

LEFT OUTER JOIN category ON event.catid = category.catid

GROUP BY users.userid, users.city, users.state, users.likesports, users.liketheatre, users.likeconcerts, users.likejazz, users.likeclassical, users.likeopera, users.likerock, users.likevegas, users.likebroadway, users.likemusicals

ORDER BY tickets_sold DESC;

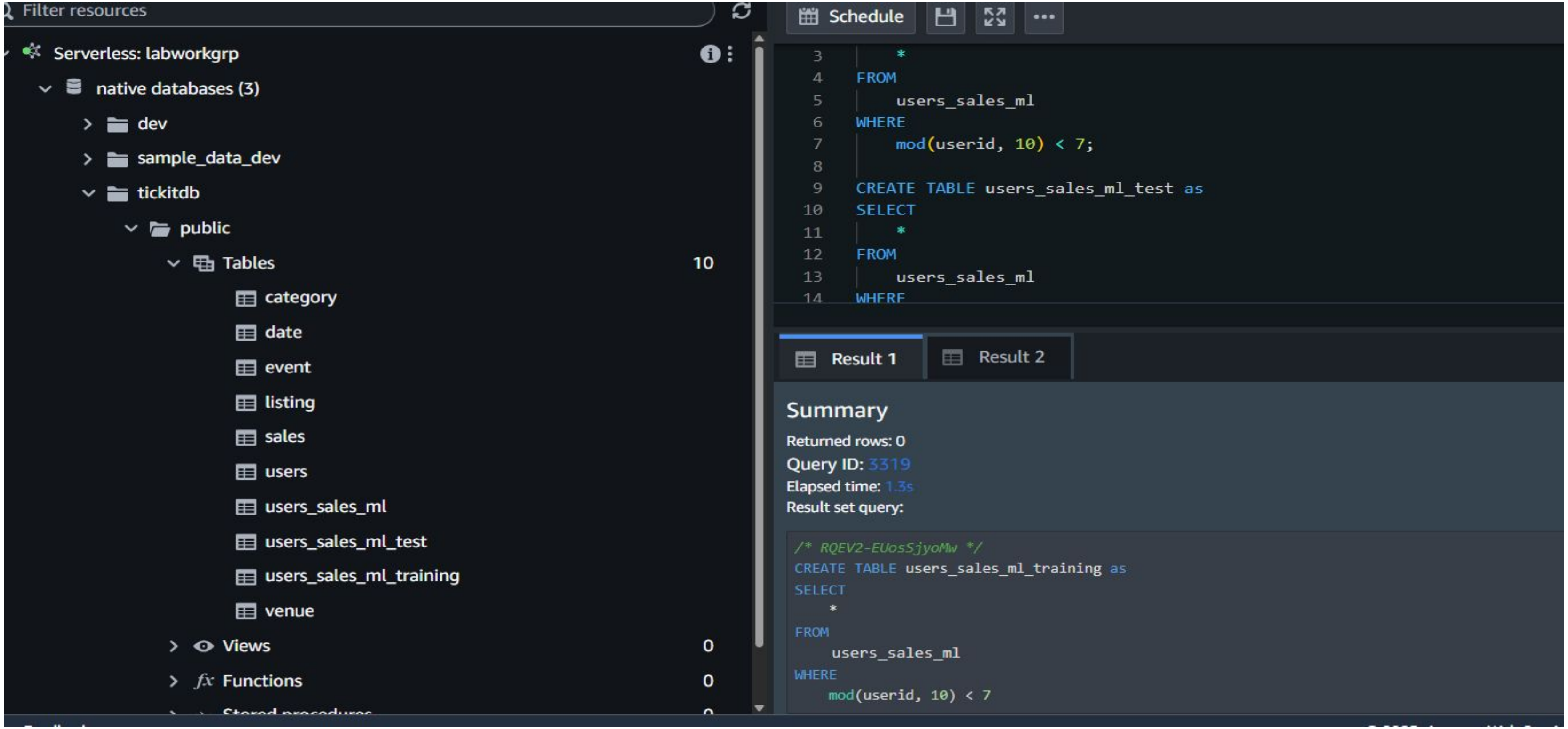
To view some data from the users sales table for the ML model, enter the following query:

```
SELECT userid, city, state, likeopera, likemusicals, opera, musicals, average_price_paid_per_ticket, tickets_sold

FROM users_sales_ml

ORDER BY tickets_sold DESC

   LIMIT 10;
```

Manually split the data so you can verify the accuracy of the model by allocating an additional prediction set. The following query splits data into two sets. The users_sales_ml_training table is for training and the users_sales_ml_test table is for test. The training table contains 70% of the data and the test table contains 30% of the data.

Command: To split the table into a training table and a test table, enter the following query:

```
CREATE TABLE users_sales_ml_trainingas

SELECT

    *

FROM

    users_sales_ml

WHERE

    mod(userid, 10) < 7;



CREATE TABLE users_sales_ml_test as

SELECT

    *

FROM

    users_sales_ml

WHERE

            mod(userid, 10) >= 7;
```



```
    *
FROM
    users_sales_ml
WHERE
    mod(userid, 10) < 7;

CREATE TABLE users_sales_ml_test as
SELECT
    *
FROM
    users_sales_ml
WHERE
```

Summary

Returned rows: 0

Query ID: 3319

Elapsed time: 1.3s

Result set query:

```
/* RQEV2-EUosSjyoMw */
CREATE TABLE users_sales_ml_training as
SELECT
    *
FROM
    users_sales_ml
WHERE
    mod(userid, 10) < 7
```

Create a model using the training data that predicts the number of tickets a customer might purchase based on the user's preferences, the categories of prior purchased tickets, their state and city of residence, and other important features.

Copy edit: To create a model, replace the S3_BUCKET placeholder value with the S3Bucket value listed to the left of these instructions. Then, run the following query:

```
CREATE MODEL predict_users_sales_ml

FROM  users_sales_ml_training TARGET tickets_sold FUNCTION predict_users_sales

    IAM_ROLE default

    PROBLEM_TYPE regression

    OBJECTIVE 'mse'

    SETTINGS ( s3_bucket 'S3_BUCKET',

        s3_garbage_collect off,

        max_runtime 1800

          );
```

The TARGET clause specifies *tickets_sold* is the label that CREATE MODEL builds a model to predict. The other columns in the table are the features, or input, used for the prediction. The number of tickets sold given a user's attributes is being predicted with this model. Since the number of tickets is a number ranging from 0 to 67 in the current dataset, a *regression* PROBLEM TYPE is applicable. The *mse* OBJECTIVE means the model is trying to optimize for the mean squared error.

After you enter the SQL command to create the model, Redshift ML securely exports the specified data from Amazon Redshift to your S3 bucket and calls Amazon SageMaker Autopilot to prepare the data (pre-processing and feature engineering), select the appropriate pre-built algorithm, and apply the algorithm for model training.

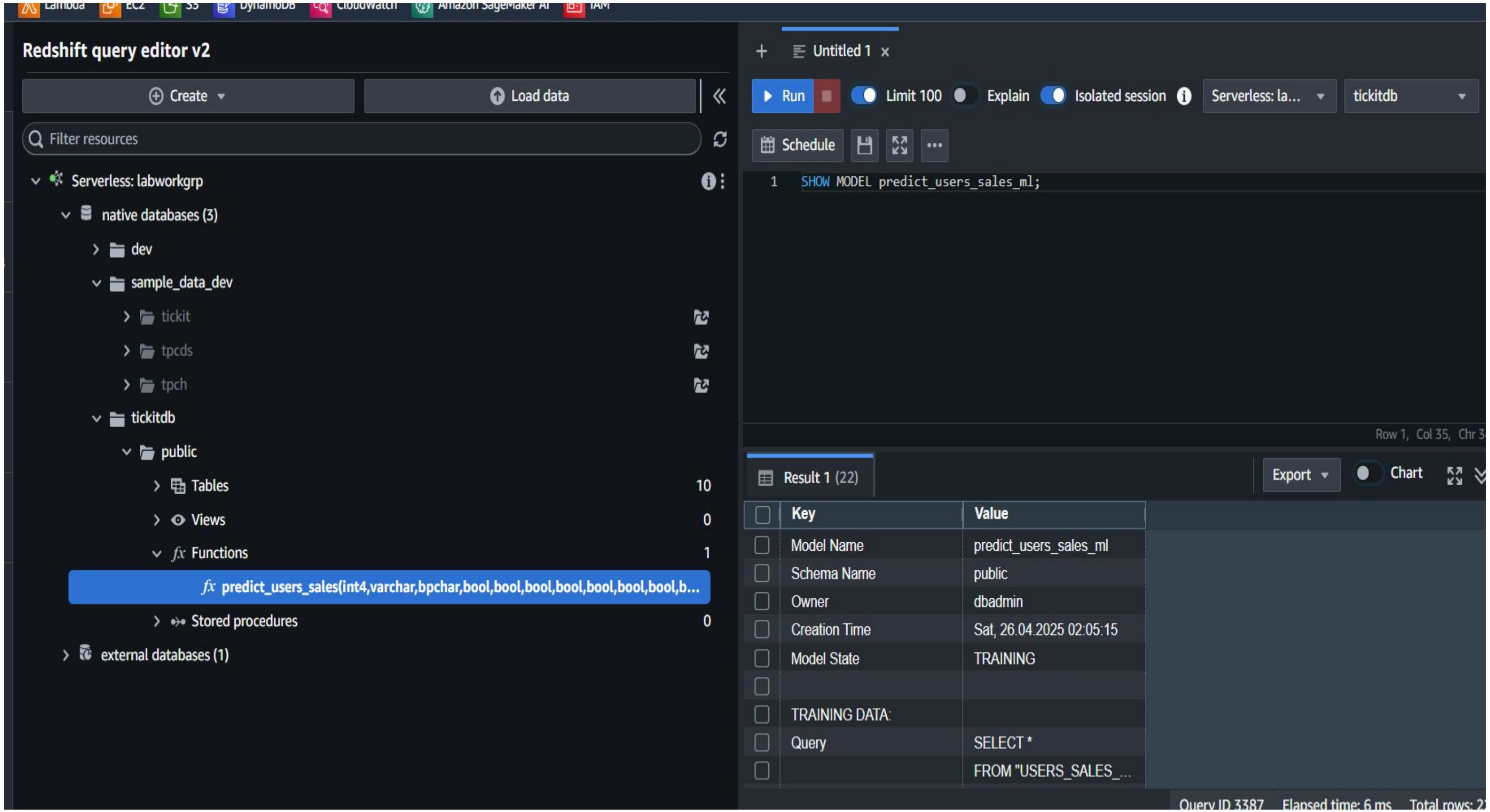
Redshift ML handles all of the interactions between Amazon Redshift, S3, and SageMaker, including all the steps involved in training and compilation. When the model has been trained, Redshift ML uses Amazon SageMaker Neo to optimize the model for deployment and makes it available as a SQL function. You can use the SQL function to apply the machine learning model to your data in queries, reports, and dashboards.

Check the status of your model by running the SHOW MODEL command from your SQL prompt.

Command: To check the status of all models, enter the following query:

```
SHOW MODEL ALL;



Result 1 (1)

------------



Schema Name | Model Name

------------+------------------------

public      | predict_users_sales_ml
```

```
SHOW MODEL predict_users_sales_ml;
```



```
SHOW MODEL predict_users_sales_ml;
```

Key	Value
Model Name	predict_users_sales_ml
Schema Name	public
Owner	dbadmin
Creation Time	Sat, 26.04.2025 02:05:15
Model State	TRAINING
TRAINING DATA:	
Query	SELECT *
	FROM "USERS_SALES_...

22. At the top of the AWS Management Console, in the search bar, search for and choose `Amazon SageMaker AI`.
23. In the left navigation pane, choose Processing and then choose Processing jobs.
24. Choose the job that contains -db- in its name.
 Consider: Which instance type is used for the processing job? Which role is the processing job using?
The first processing job is called *datasplitter*. It splits the data for training and sets up the data storage. Notice that the processing job selected an instance to run the job. In this case, an ml.m5.2xlarge is selected. The job uses the RedshiftRole you specified when running the *CREATE MODEL* query.
 Note: The datasplitter job can take 3-5 minutes to complete.
25. Once the datasplitter job is complete, in the left navigation pane, choose Processing jobs.
26. Choose the job that contains -pr- in its name.
 Consider: What is the processing image used for the processing job?
The second processing job is called *pipeline-recommender* and uses the pipeline-recommender image. The job sets up the pipeline and configures the training jobs. Once the processing job completes, the training jobs are launched.
 Note: The pipeline-recommender job can take 4-6 minutes to complete.
 Once the pipeline recommender job is complete, in the left navigation pane, choose Training and then choose Training jobs.
Ten training jobs are launched. Each training job runs in parallel.
 Note: The training jobs take 2-4 minutes to complete.
27. Choose one of the training jobs and view the details.
 Consider: Which Training image does the training job use?
The training job uses an sklearn-automl training image. SageMaker provides prebuilt Docker images that install the scikit-learn and Spark ML libraries. These libraries also include the dependencies needed to build Docker images that are compatible with SageMaker using the Amazon SageMaker Python SDK. With the SDK, you can use scikit-learn for machine learning tasks and use Spark ML to create and tune machine learning pipelines.

Evaluate the model performance

Your model has a model state of READY and can be used for inference. Review the validation:mse score and the Model Type that was selected.

You can calculate the mean square error and root mean square error based on your validation data. You use mean square error and root mean square error to measure the distance between the predicted numeric target and the actual numeric answer. A good model has a low score in both metrics.

To get the value of both model evaluation metrics, enter the following query:

```
SELECT ROUND(AVG(POWER((actual_tickets_sold - predicted_tickets_sold), 2)), 2) mse, ROUND(SQRT(AVG(POWER((actual_tickets_sold - predicted_tickets_sold), 2))), 2) rmse FROM (SELECT tickets_sold AS actual_tickets_sold, ROUND(PREDICT_USERS_SALES(userid, city, state, likesports, liketheatre, likeconcerts, likejazz, likeclassical, likeopera, likerock, likevegas, likebroadway, likemusicals, musicals, plays, opera, pop, average_price_paid_per_ticket, average_commission_per_ticket), 1) AS predicted_tickets_sold FROM users_sales_ml_test);
```

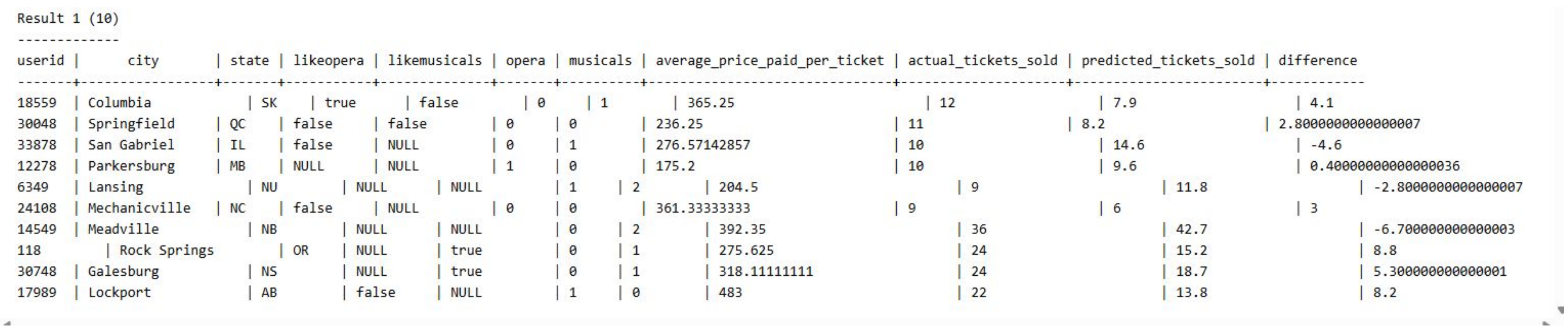
```
************************

**** EXAMPLE OUTPUT ****

************************

Result 1 (1)

------------

mse    | rmse

-------+------

11.72  | 3.42
```

Invoke the model for inference

You are ready to use the model to make predictions on new data. You can run prediction queries by including the function name in your query. For this model, you include PREDICT_USERS_SALES() in your query with the feature values from your new records to get the model predictions.

Command: To compare the predicted ticket values against the actual ticket values, enter the following query

```
SELECT userid, city, state, likeopera, likemusicals, opera, musicals, average_price_paid_per_ticket, actual_tickets_sold, predicted_tickets_sold, (actual_tickets_sold - predicted_tickets_sold) AS difference FROM (SELECT userid, city, state, likeopera, likemusicals, opera, musicals, average_price_paid_per_ticket, tickets_sold AS actual_tickets_sold, ROUND(PREDICT_USERS_SALES(userid, city, state, likesports, liketheatre, likeconcerts, likejazz, likeclassical, likeopera, likerock, likevegas, likebroadway, likemusicals, musicals, plays, opera, pop, average_price_paid_per_ticket, average_commission_per_ticket), 1) AS predicted_tickets_sold FROM users_sales_ml_test) ORDER BY actual_tickets_sold DESC LIMIT 10;
```

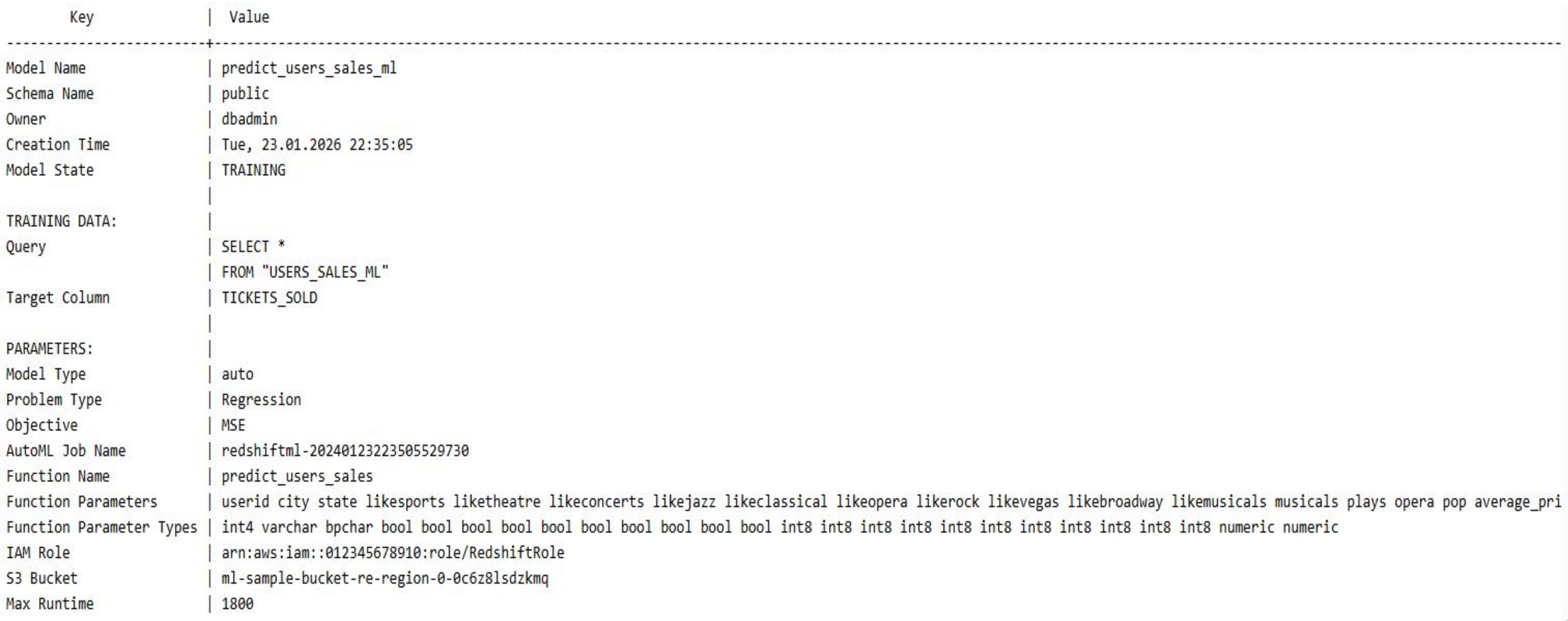


```
Result 1 (10)
-------------
userid |      city       | state | likeopera | likemusicals | opera | musicals | average_price_paid_per_ticket | actual_tickets_sold | predicted_tickets_sold | difference
-------+-----------------+-------+-----------+--------------+-------+----------+-------------------------------+---------------------+------------------------+------------
18559  | Columbia          | SK    | true      | false        | 0     | 1        | 365.25                        | 12                  | 7.9                    | 4.1
30048  | Springfield     | QC    | false     | false        | 0     | 0        | 236.25                        | 11                  | 8.2                    | 2.8000000000000007
33878  | San Gabriel     | IL    | false     | NULL         | 0     | 1        | 276.57142857                  | 10                  | 14.6                   | -4.6
12278  | Parkersburg     | MB    | NULL      | NULL         | 1     | 0        | 175.2                         | 10                  | 9.6                    | 0.40000000000000036
6349   | Lansing           | NU        | NULL      | NULL         | 1     | 2        | 204.5                         | 9                   | 11.8                   | -2.8000000000000007
24108  | Mechanicville   | NC    | false     | NULL         | 0     | 0        | 361.33333333                  | 9                   | 6                      | 3
14549  | Meadville         | NB        | NULL      | NULL         | 0     | 2        | 392.35                        | 36                  | 42.7                   | -6.700000000000003
118        | Rock Springs        | OR    | NULL      | true         | 0     | 1        | 275.625                       | 24                  | 15.2                   | 8.8
30748  | Galesburg         | NS        | NULL      | true         | 0     | 1        | 318.11111111                  | 24                  | 18.7                   | 5.300000000000001
17989  | Lockport          | AB        | false     | NULL         | 1     | 0        | 483                           | 22                  | 13.8                   | 8.2
```

Enter the SHOW MODEL command with your model name to see the status for your specific model

21. Command: To check the status of your *predict_users_sales_ml* model, enter the following query:

```
SHOW MODEL predict_users_sales_ml;
```



```
        Key              |  Value
-------------------------+----------------------------------------------------------------------------------------------------------------------------------------
Model Name               | predict_users_sales_ml
Schema Name              | public
Owner                    | dbadmin
Creation Time            | Tue, 23.01.2026 22:35:05
Model State              | TRAINING
                         |
TRAINING DATA:           |
Query                    | SELECT *
                         | FROM "USERS_SALES_ML"
Target Column            | TICKETS_SOLD
                         |
PARAMETERS:              |
Model Type               | auto
Problem Type             | Regression
Objective                | MSE
AutoML Job Name          | redshiftml-20240123223505529730
Function Name            | predict_users_sales
Function Parameters      | userid city state likesports liketheatre likeconcerts likejazz likeclassical likeopera likerock likevegas likebroadway likemusicals musicals plays opera pop average_pri
Function Parameter Types | int4 varchar bpchar bool bool bool bool bool bool bool bool bool bool int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 numeric numeric
IAM Role                 | arn:aws:iam::012345678910:role/RedshiftRole
S3 Bucket                | ml-sample-bucket-re-region-0-0c6z8lsdzkmq
Max Runtime              | 1800
```