
# HANDS-ON LAB GUIDE FOR MACHINE LEARNING WITH SNOWFLAKE AND AMAZON SAGEMAKER

To be used with the Snowflake free 30-day trial at:
**https://trial.snowflake.com**

Snowflake Enterprise Edition preferred on **AWS** - US West, US East (Ohio or N. Virginia) regions recommended

AWS Account - Select region - US-WEST-2 (Oregon), US-EAST-2 (Ohio) and US-EAST-1(N. Virginia) are good choices
**Create AWS Account**

Approximate duration: 90 minutes.

# Table of Contents

# Lab Overview

In this lab, you'll learn the basics of creating an advanced analytics solution. We'll train a customer churn prediction model from data in Snowflake using SageMaker and load the churn risk scores back into Snowflake for analysis.

## Target Audience

Database and Data Warehouse Administrators and Architects. Developers looking to extend Data Apps with Machine Learning. ML Practitioners looking to incorporate Snowflake data in their ML workflow.

## What you'll learn

The exercises in this lab will walk you through the steps to:

- Create stages, databases, tables, user, role and warehouses
- Load data into a table within your Snowflake account
- Launch a SageMaker Notebook instance
- Connect to your Snowflake instance and pull data into a Pandas dataframe
- Visualize the data and perform basic feature engineering
- Unload a dataset into S3 and use it to train a machine learning model
- Run a batch of data through your model and load the results back into Snowflake

## Prerequisites

- Use of the Snowflake free 30-day trial environment
- Use of an AWS Account with the ability to launch a CloudFormation template, create a S3 bucket and SageMaker Instance
- Basic knowledge of SQL, and database concepts and objects
- Familiarity with CSV comma-delimited files
- Basic Jupyter notebook and Python knowledge

# Module 1: Prepare Your Lab Environment

## 1.1 Steps to Prepare Your Lab Environment

**1.1.1** If not yet done, register for a Snowflake free 30-day trial at https://trial.snowflake.com

- The Snowflake Enterprise Edition on AWS and US West, US East (Ohio or N. Virginia) regions recommended. Or we suggest you select the region which is physically closest to you.

- After registering, you will receive an email with an activation link and your Snowflake account URL. Bookmark this URL for easy, future access. After activation, you will create a user name and password. Write down these credentials.

**1.1.2** If you do not already have a AWS account please create a new account using this link - **Create AWS Account**. Make sure you have the permissions to use a Cloudformation Template, create a S3 bucket and launch a SageMake Notebook instance. Once logged in to your account select an AWS region closest to your Snowflake account, US-WEST-2 (Oregon), US-EAST-2 (Ohio) and US-EAST-1(N. Virginia) are good choices.

**1.1.3** Resize your browser windows so you can view this lab guide PDF and your web browser side-by-side to more easily follow the lab instructions. If possible, even better is to use a secondary display dedicated to the lab guide. It is also advisable to open a second browser window so you are able to view the Snowflake UI and AWS console side by side.

**1.1.4** Download the Snowflake and SageMaker files to your local machine.

- Go to the github repository: https://github.com/andremolenaar/SnowflakeSagemaker

- Download the files
  - sagemaker_workshop_v2.2.sql
  - workshop-snowflake-sagemaker-v2.2.ipynb
  - snowflake-sagemaker-notebook-v2.2.yaml

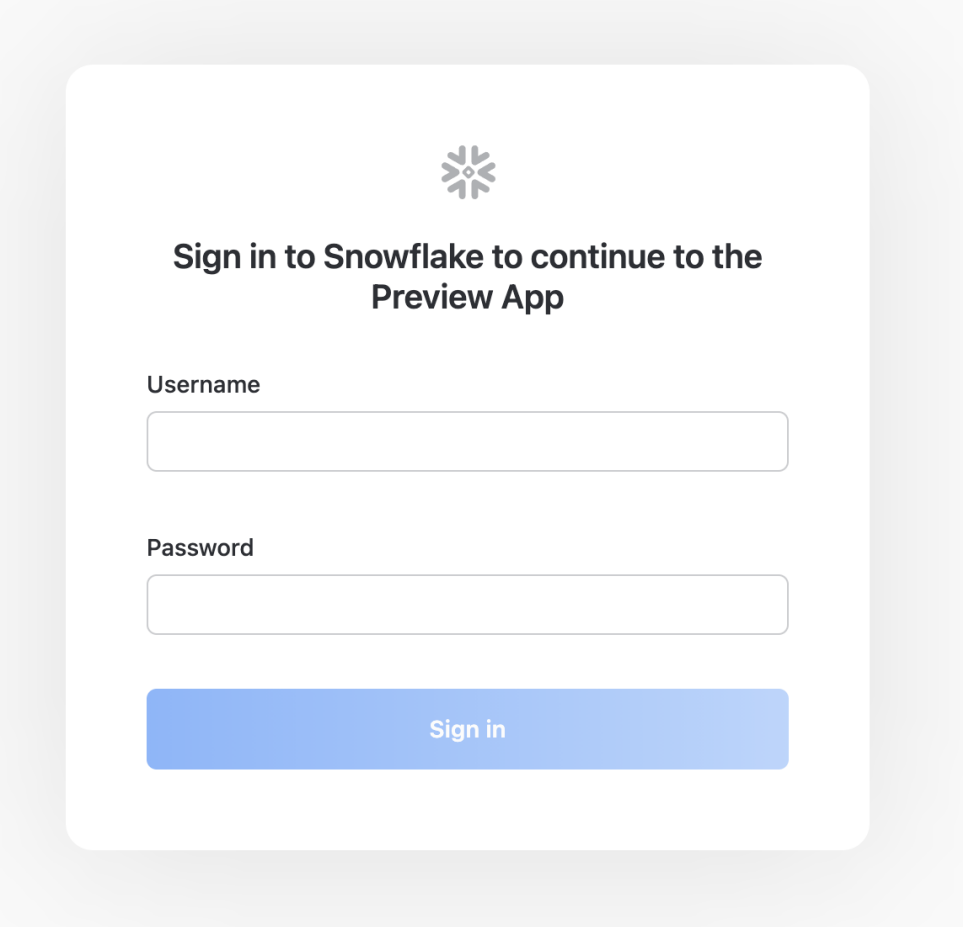
# Module 2: The Snowflake User Interface

**About the screen captures, sample code, and environment**
Screen captures in this lab depict examples and results that may slightly vary from what you may see when you complete the exercises.

## 2.1 Logging Into the Snowflake User Interface (UI)

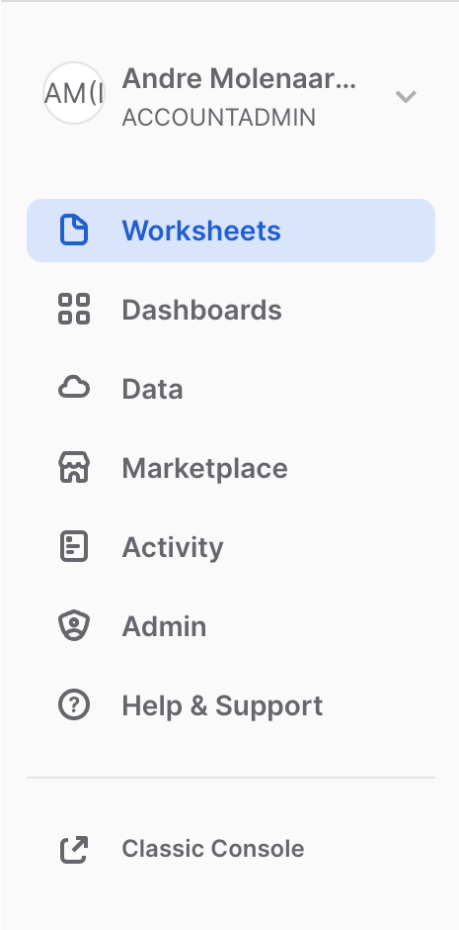
2.1.1 Open a browser window and enter the URL of your the Snowflake 30-day trial environment.

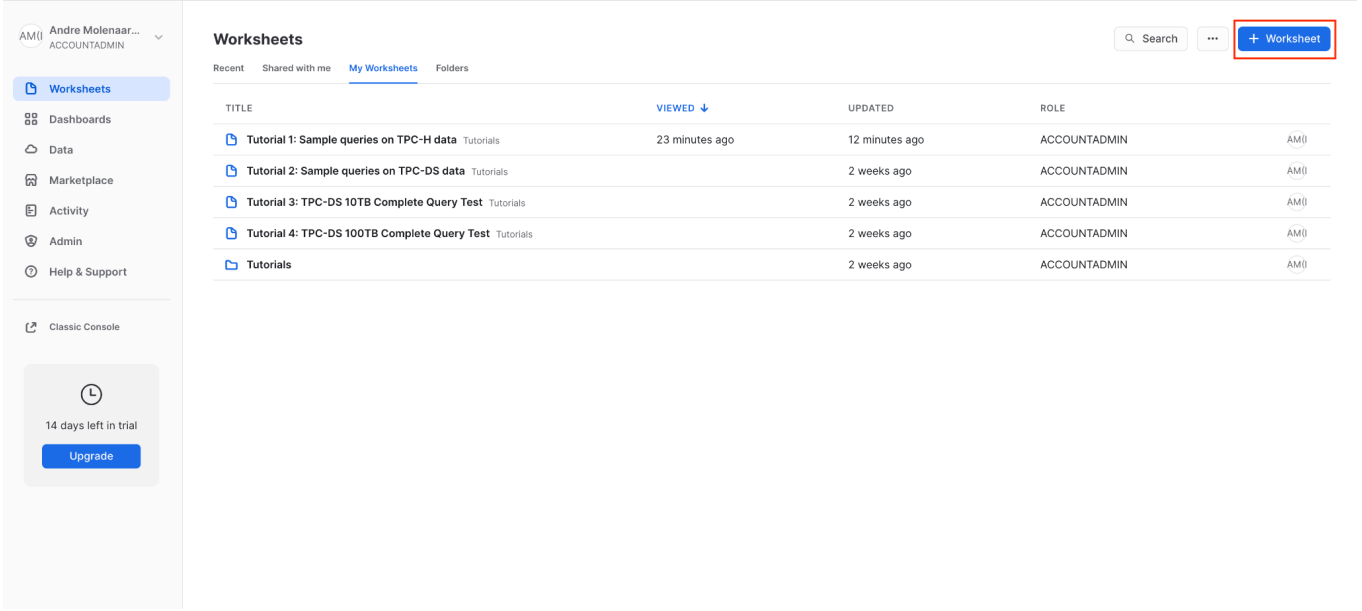
2.1.2 You should see the login screen below. Enter your unique credentials to log in.



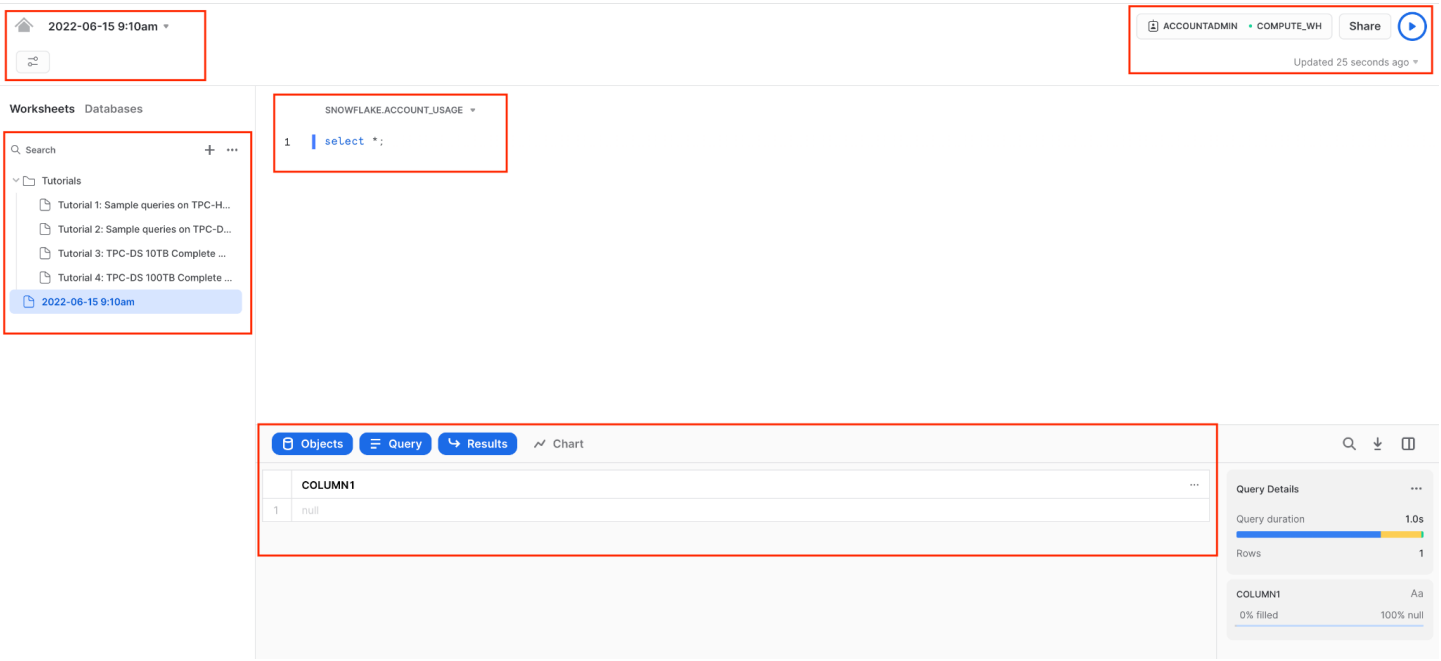
## 2.2 Navigating the Snowflake UI

2.2.1 Let's get you acquainted with Snowflake! This section covers the basic components of the user interface. We will move from top to bottom on the left-hand side margin.






2.2.2 The Worksheets tab provides an interface for submitting SQL queries, performing DDL and DML operations, and viewing results as your queries or operations complete. A new worksheet is created by clicking + Worksheet on the top right.



The top left corner contains the following:

- **Home** icon: Use this to get back to the main console/close the worksheet.
- **Worksheet_name** drop-down: The default name is the timestamp when the worksheet was created. Click the timestamp to edit the worksheet name. The drop-down also displays additional actions you can perform for the worksheet.
- **Manage filters** button: Custom filters are special keywords that resolve as a subquery or list of values.
- **+** button: This creates a new worksheet.

The top right corner contains the following:

- **Context** box: This lets Snowflake know which role and warehouse to use during this session. It can be changed via the UI or SQL commands.
- **Share** button: Open the sharing menu to share to other users or copy the link to the worksheet.
- **Play/Run** button: Run the SQL statement where the cursor currently is or multiple selected statements.

The middle pane contains the following:

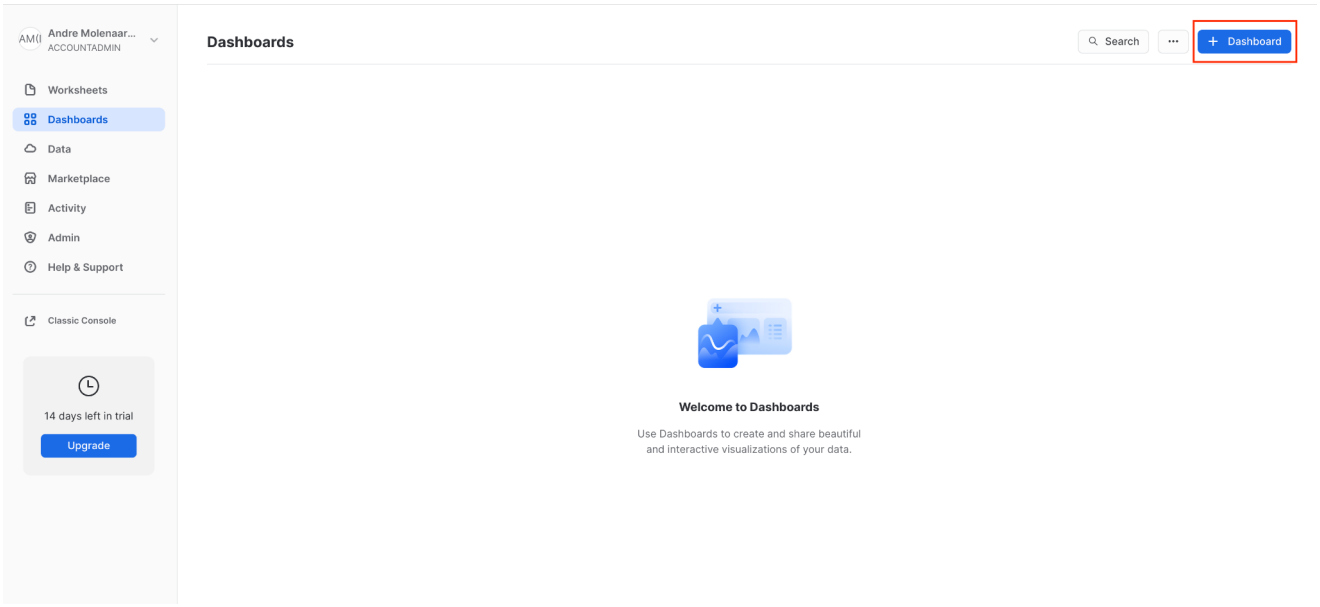
- Drop-down at the top for setting the database/schema/object context for the worksheet.
- General working area where you enter and execute queries and other SQL statements.

The middle-left panel contains the database objects browser which enables you to explore all databases, schemas, tables, and views accessible by the role currently in use for the worksheet.

The bottom pane displays the results of queries and other operations. Also includes 4 options (Object, Query, Result, Chart) that open/close their respective panels on the UI. Chart opens a visualization panel for the returned results. More on this later.

The various panes on this page can be resized by adjusting their sliders. If you need more room in the worksheet, collapse the database objects browser in the left panel. Many of the screenshots in this guide keep this panel closed.

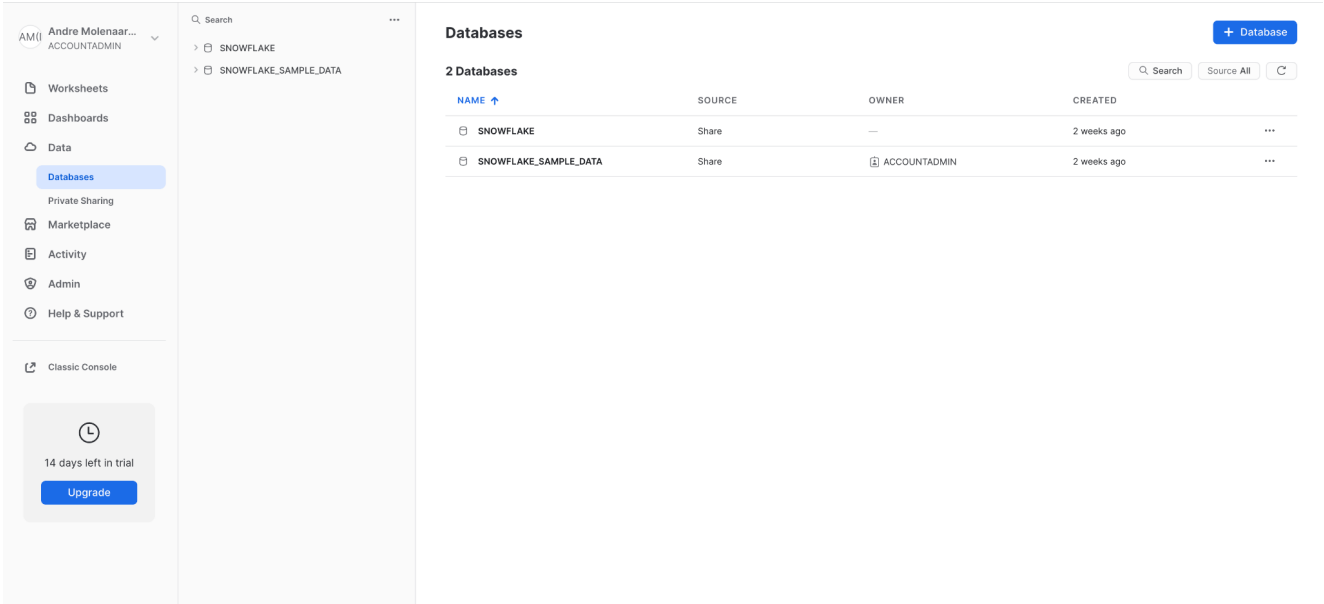
**Worksheets vs the UI** Most of the exercises in this lab are executed using pre-written SQL within this worksheet to save time. These tasks can also be done via the UI, but would require navigating back-and-forth between multiple UI tabs.




### 2.2.3 Dashboards

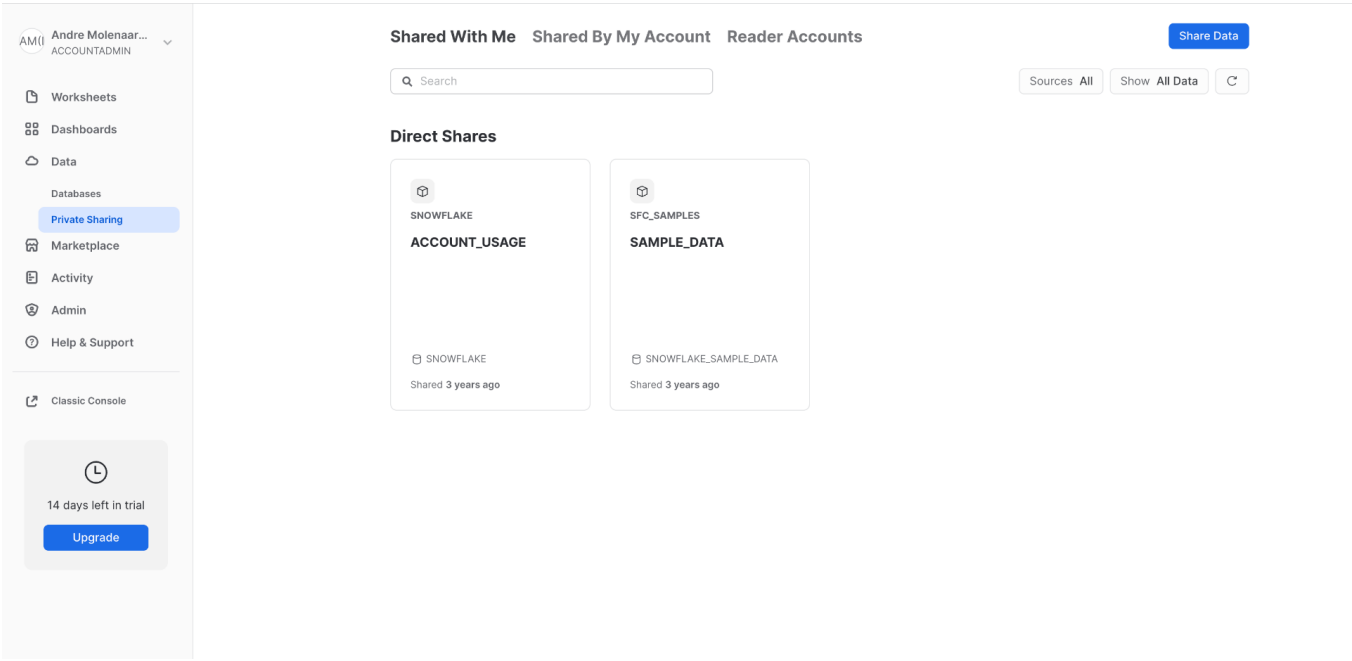
The Dashboards tab allows you to create flexible displays of one or more charts (in the form of tiles, which can be rearranged). Tiles and widgets are produced by executing SQL queries that return results in a worksheet. Dashboards work at a variety of sizes with minimal configuration.

### 2.2.4 Databases



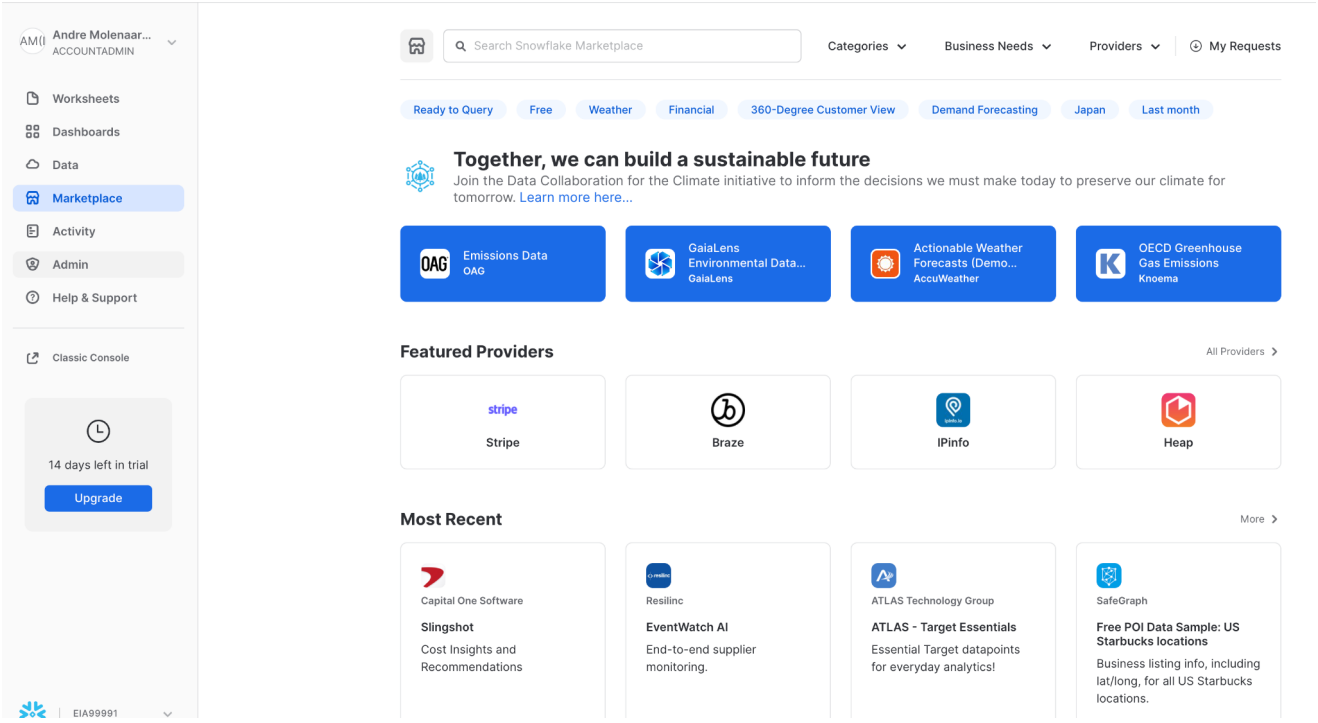
Under **Data**, the **Databases** tab shows information about the databases you have created or have permission to access. You can create, clone, drop, or transfer ownership of databases, as well as load data in the UI. Notice that a database already exists in your environment. However, we will not be using it in this lab.

### 2.2.5 Private Sharing



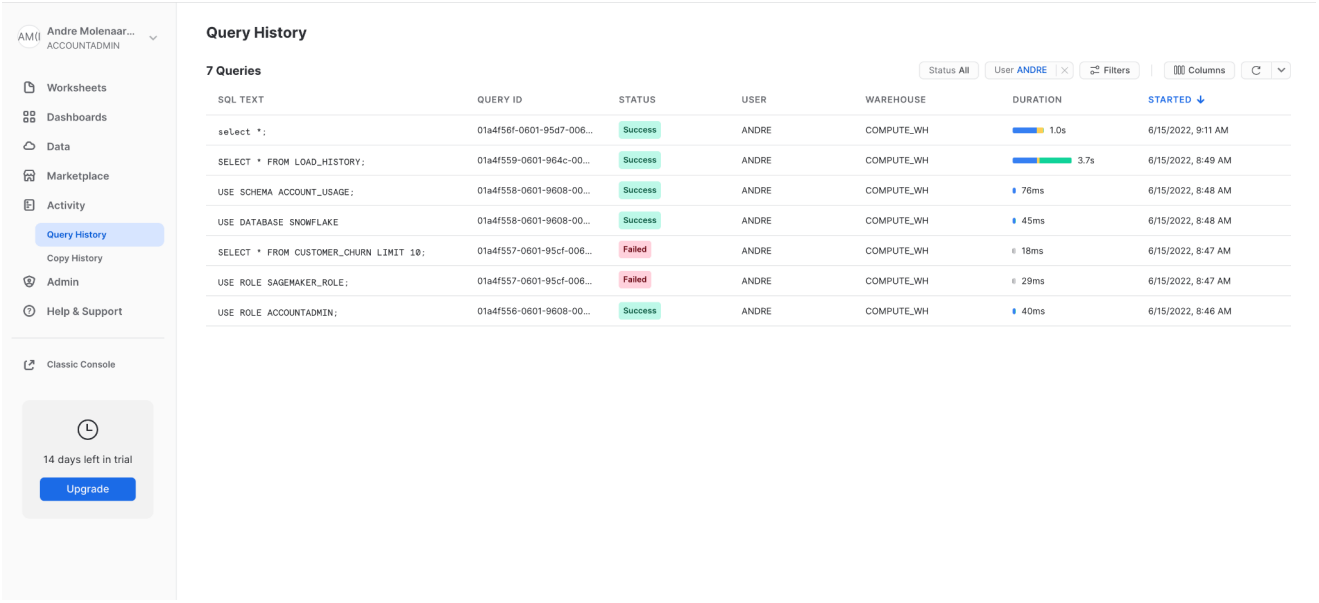

Also under **Data**, the **Private Sharing** tab is where data sharing can be configured to easily and securely share Snowflake tables among separate Snowflake accounts or external users, without having to create a copy of the data.

## 2.2.6 Marketplace




**Marketplace** is where any Snowflake customer can browse and consume data sets made available by providers. There are two types of shared data: Public and Personalized. Public data is free data sets available for querying instantaneously. Personalized data requires reaching out to the provider of data for approval of sharing data.

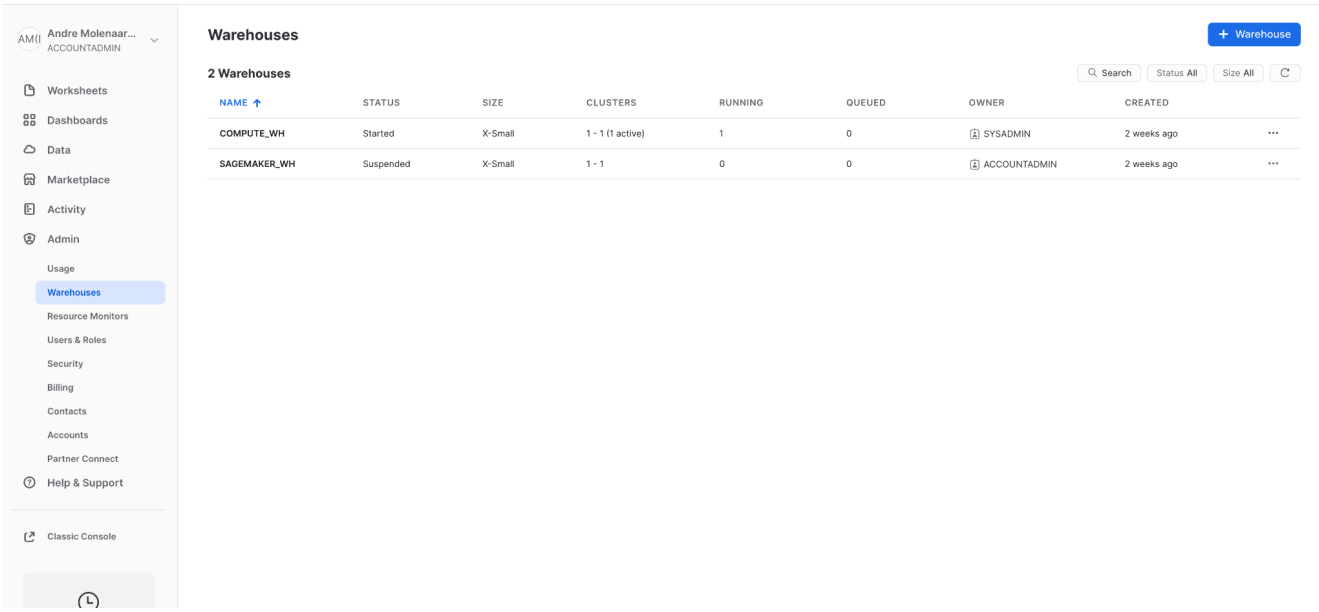
### 2.2.7 History




Under **Activity**, the **Query History** tab shows the following:

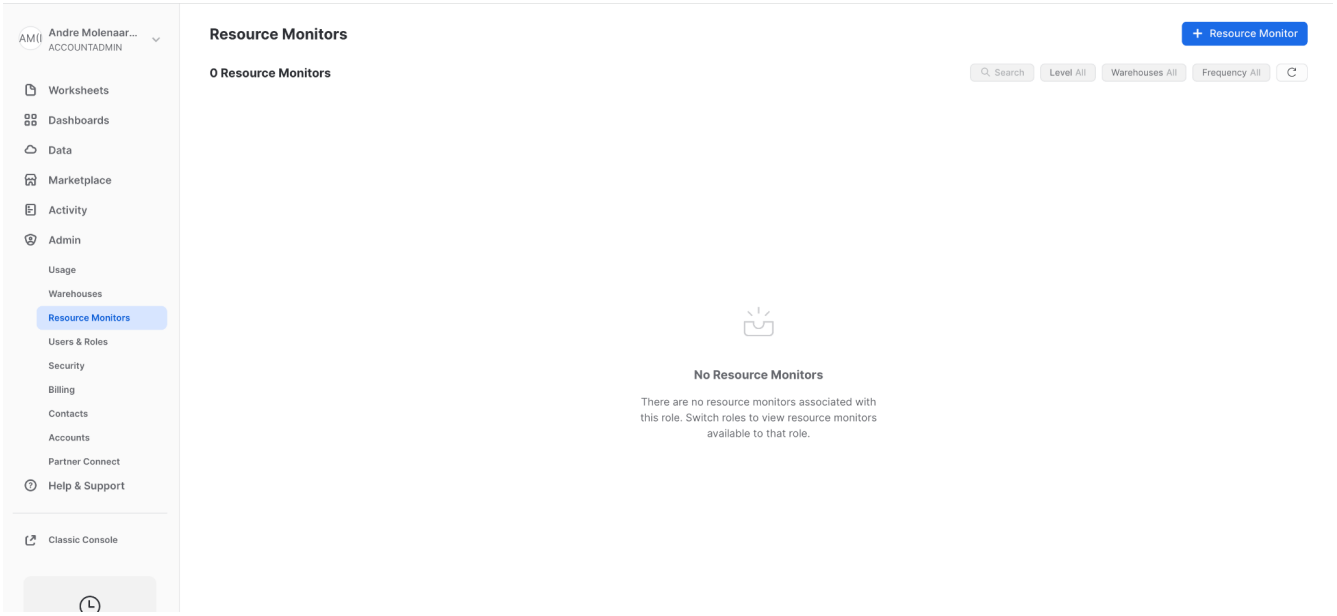
- Queries is where previous queries are shown, along with filters that can be used to hone results (user, warehouse, status, query tag, etc.). View the details of all queries executed in the last 14 days from your Snowflake account. Click a query ID to drill into it for more information.
- Copies shows the status of copy commands run to ingest data into Snowflake.

### 2.2.8 Warehouses



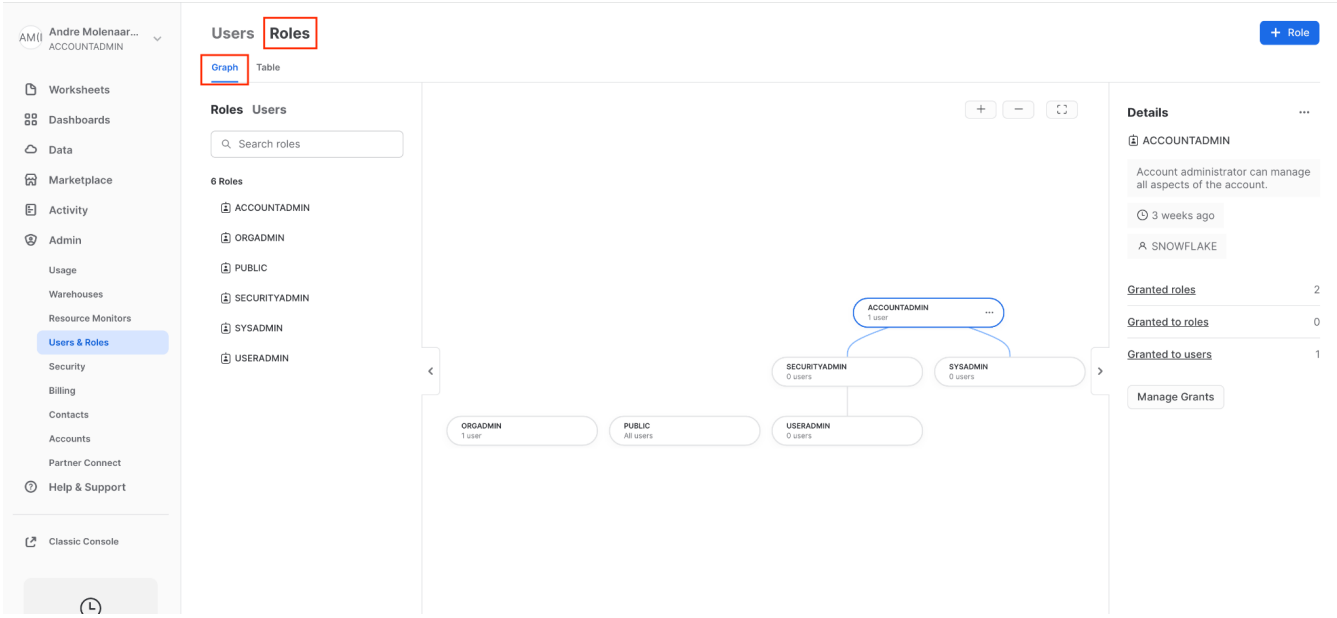
Also under **Admin**, the **Warehouses** tab is where you set up and manage compute resources known as virtual warehouses to load or query data in Snowflake. A warehouse called COMPUTE_WH (XS) already exists in your environment.

### 2.2.9 Resource Monitors



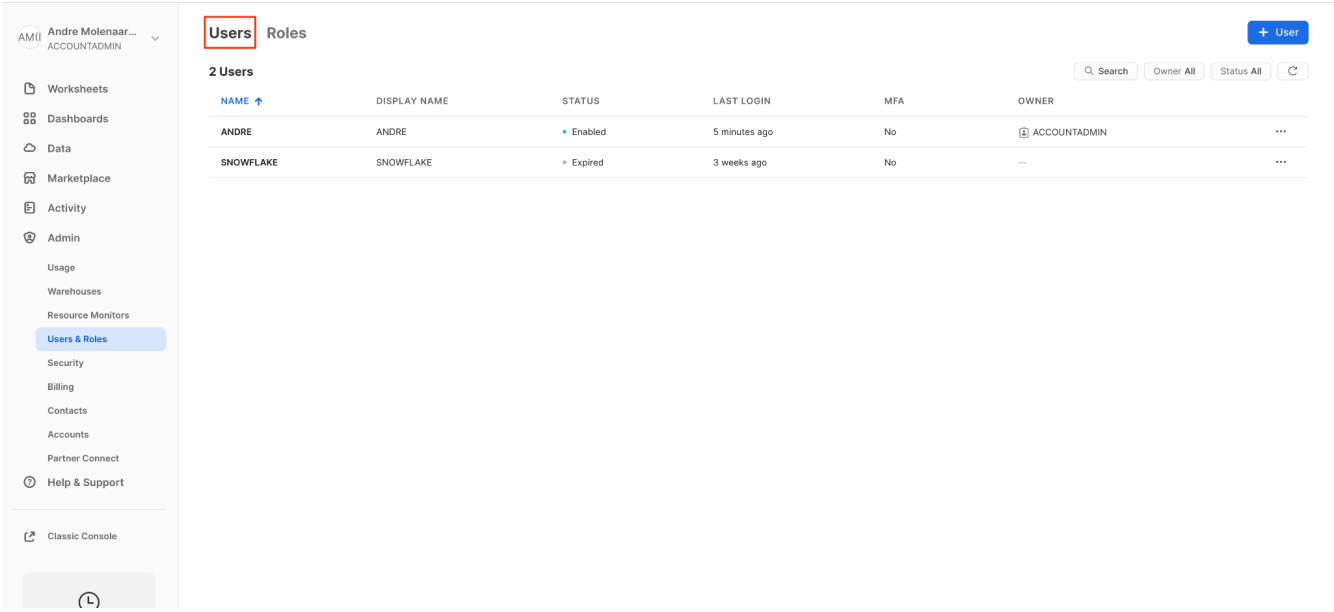

**Resource Monitors,** the second tab under **Admin**, shows all the resource monitors that have been created to control the number of credits that virtual warehouses consume. For each resource monitor, it shows the credit quota, type of monitoring, schedule, and actions performed when the virtual warehouse reaches its credit limit.

### 2.2.10 Roles

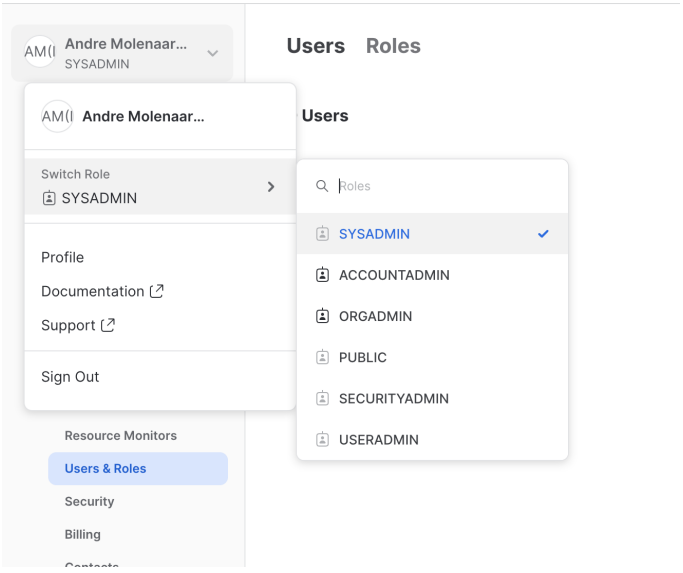


Under **Admin**, the **Users & Roles** tab shows a list of the roles and their hierarchies. Roles can be created, reorganized, and granted to users in this tab. The roles can also be displayed in tablular/list format by clicking Table at the top of the page.

### 2.2.11 Users




Also in the same place, the **Users** tab shows a list of users in the account, default roles, and owner of the users. For a new account, no records are shown because no additional roles have been created. Permissions granted through your current role determine the information shown for this tab. To see all the information available on the tab, switch your role to ACCOUNTADMIN.

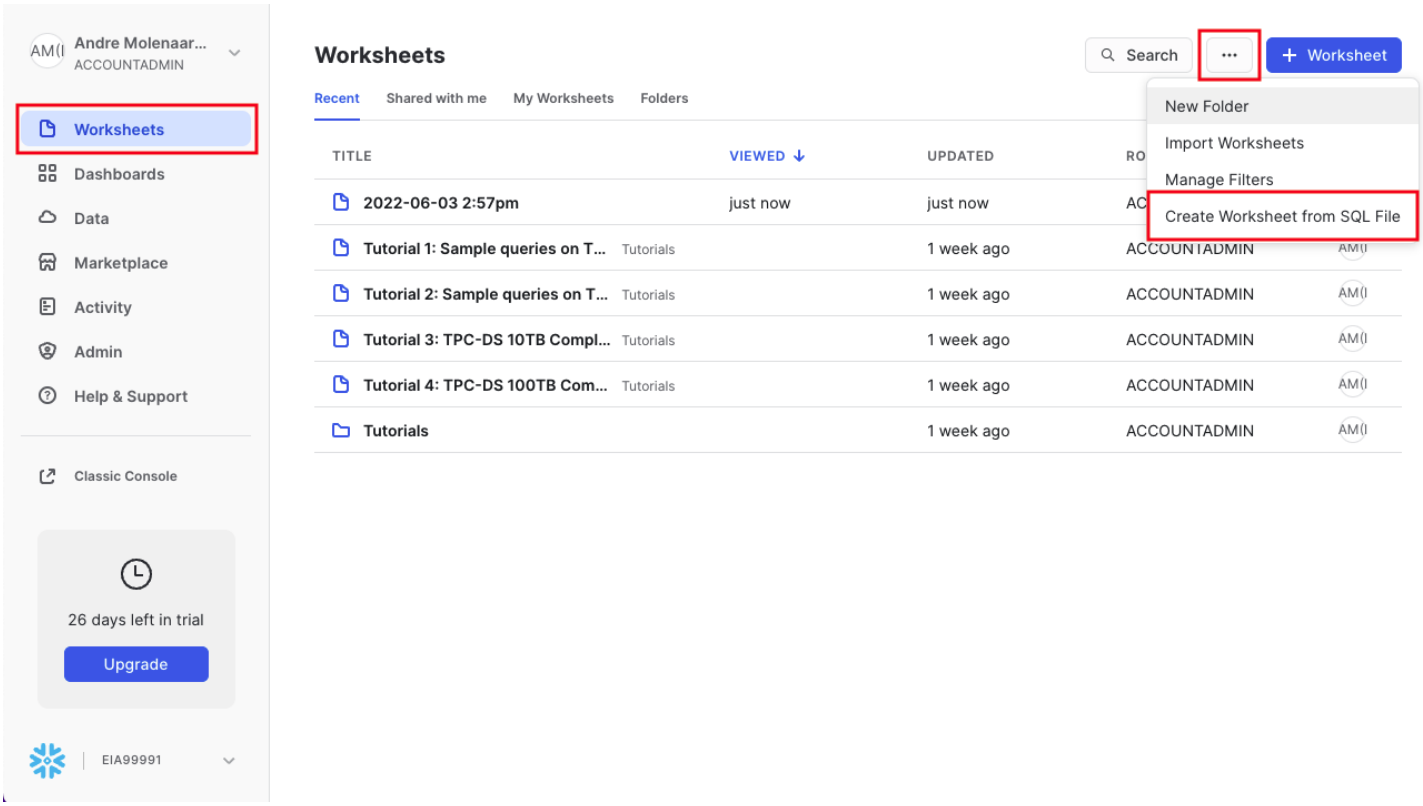



Clicking on your username in the top right of the UI allows you to change your password, roles, and preferences. Snowflake has several system defined roles. You are currently in the default role of SYSADMIN and will stay in this role for the majority of the lab.

SYSADMIN The SYSADMIN (aka System Administrator) role has privileges to create warehouses, databases, and other objects in an account. In a real-world environment, you would use different roles for the tasks in this lab, and assign roles to your users.

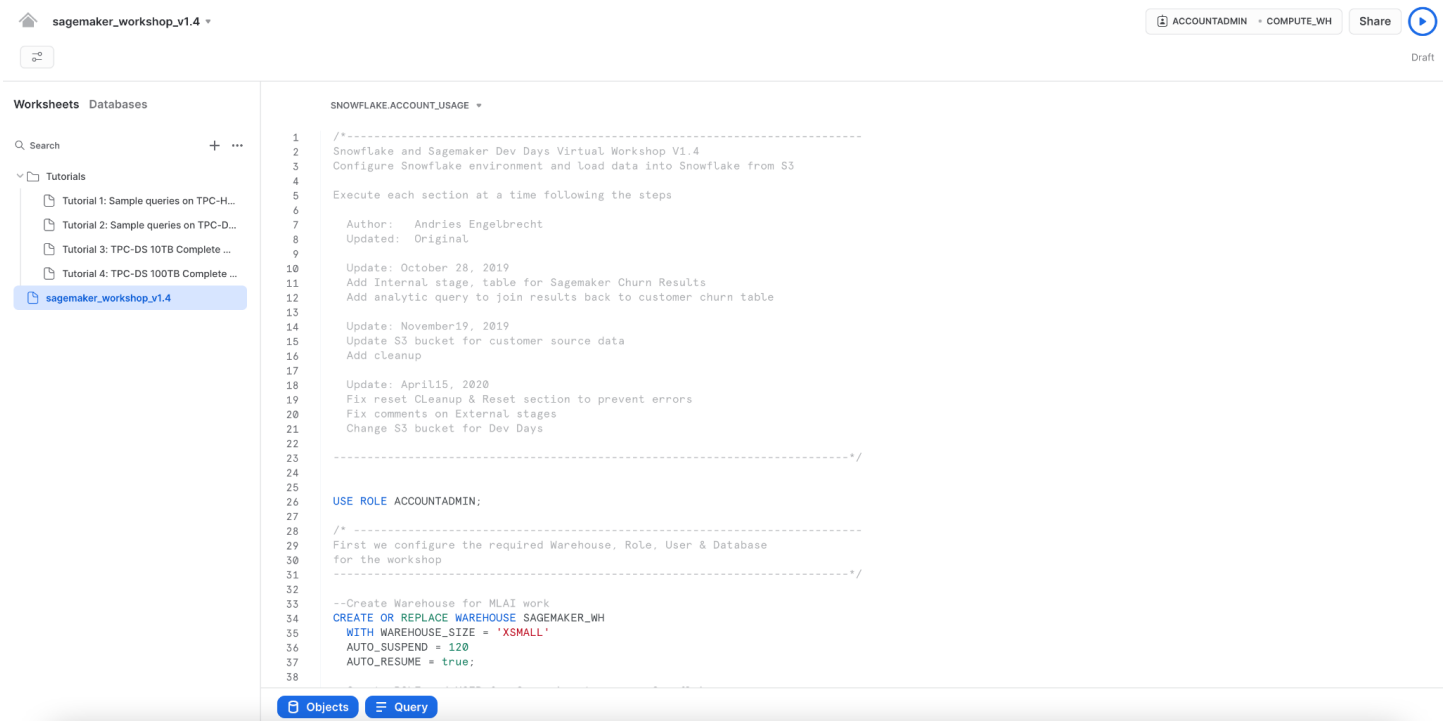
### 2.2.12 Start with the Lab

At the top left Select the workshop button, and after that, press on the 3 dots '...' button. A list of menu items will appear and select the 'Create Worksheet from SQL File' option from the menu.




Navigate to the sagemaker_workshop.sql file you downloaded earlier, and hit the 'Open' button.

You should end up with a screen looking like this:




**Warning - Do Not Copy/Paste SQL From This PDF to a Worksheet**
Copy-pasting the SQL code from this PDF into a Snowflake worksheet will result in formatting errors and the SQL will not run correctly. Make sure to use the “Load Script” method just covered.

On older or locked-down browsers, this “load script” step may not work as the browser will prevent you from opening the .sql file. If this is the case, open the .sql file with a text editor and then copy/paste all the text from the .sql file to the “Worksheet 1”

**Worksheets vs the UI**
Much of the configurations in this lab will be executed via this pre-written SQL in the Worksheet in order to save time. These configurations could also be done via the UI in a less technical manner but would take more time.

# Module 3: Preparing to Load Data & Loading Data

Let's start by preparing to load the structured data on Customer Churn into Snowflake.

This module will walk you through the steps to:

- Create a virtual warehouse
- Create a role and user
- Granting of a role to a user and privileges to a role
- Create a database and tables
- Create external and internal stages
- Create a file format for the data
- Load data into a table and querying the table

**Getting Data into Snowflake**
There are many ways to get data into Snowflake from many locations including the COPY command, Snowpipe auto-ingestion, an external connector, or a third-party ETL/ELT product. More information on getting data into Snowflake, see https://docs.snowflake.net/manuals/user-guide-data-load.html

We are using the COPY command and S3 storage for this module in a manual process so you can see and learn from the steps involved. In the real-world, a customer would likely use an automated process or ETL product to make the data loading process fully automated and much easier.

The dataset we use is publicly available and was mentioned in the book Discovering Knowledge in Data by Daniel T. Larose. It is attributed by the author to the University of California Irvine Repository of Machine Learning Datasets. The data has been exported and pre-staged for you in an Amazon AWS S3 bucket in the US-WEST region. The data consists of information from mobile operators historical records on which customers ultimately ended up churning and which continued using the service.

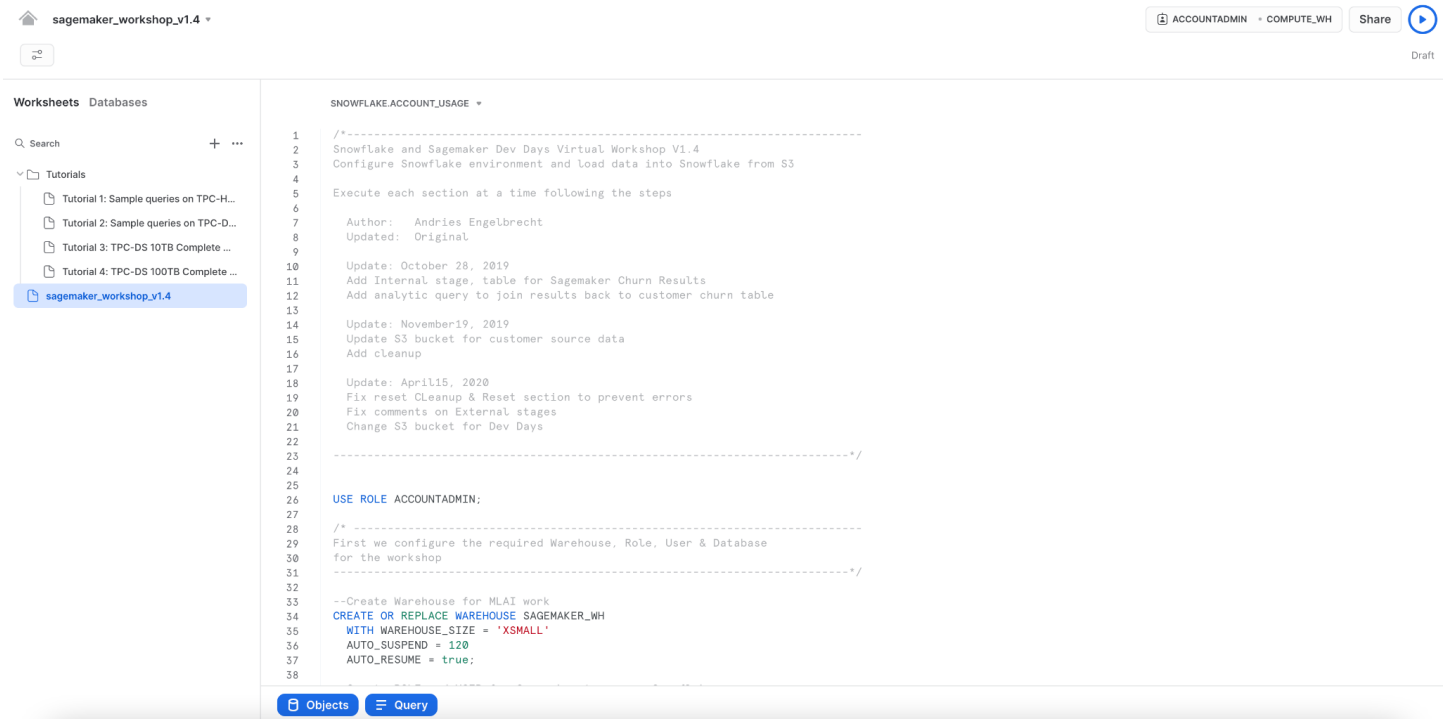
Below is a snippet from one of the Customer Churn CSV data files:

```
1,KS,128,415,382-4657,no,yes,
25,265.1,110,45.07,197.4,99,16.78,244.7,91,11.01,10,3,2.7,1,False.
2,OH,107,415,371-7191,no,yes,
26,161.6,123,27.47,195.5,103,16.62,254.4,103,11.45,13.7,3,3.7,1,False.
3,NJ,137,415,358-1921,no,no,
0,243.4,114,41.38,121.2,110,10.3,162.6,104,7.32,12.2,5,3.29,0,False.
4,OH,84,408,375-9999,yes,no,0,299.4,71,50.9,61.9,88,5.26,196.9,89,8.86,6.6,7,1.78,2,False.
5,MA,121,510,355-9993,no,yes,
24,218.2,88,37.09,348.5,108,29.62,212.6,118,9.57,7.5,7,2.03,3,False.
6,MO,147,415,329-9001,yes,no,
0,157,79,26.69,103.1,94,8.76,211.8,96,9.53,7.1,6,1.92,0,False.
```

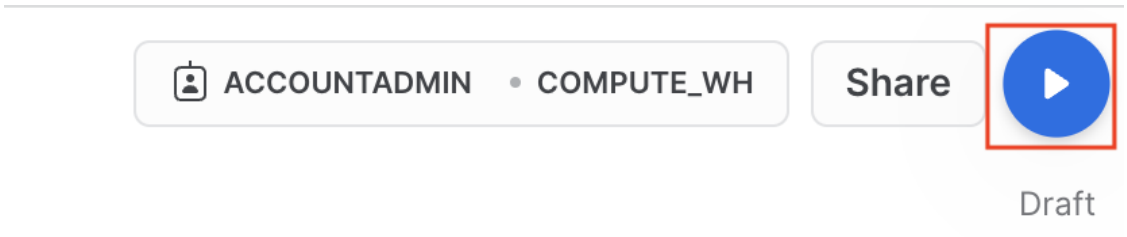
It is in comma-delimited format with no header line. This will come into play later in this module as we configure the Snowflake table which will store this data.

## 3.1 Start using Worksheets and Create a Virtual Warehouse

3.1.1 At this point, you should see the worksheet with all the SQL we loaded in a prior step.




3.1.2 To execute a SQL command or commands, click or select multiple commands with your mouse. You can now either press COMMAND & RETURN on a Mac or CONTROL & ENTER on Windows; or you can click the RUN button towards the top left hand slide of the Worksheet.




3.1.3 Next we will briefly switch roles to the ACCOUNTADMIN role primarily to allow us to create a specific role for this workshop. Execute the SQL command shown below.

```
USE ROLE ACCOUNTADMIN;
```

3.1.4 Let's create a Warehouse called SAGEMAKER_WH.
**Note:** that the Warehouse is configured to auto suspend and resume, this prevents the unnecessary use of credits if the Warehouse is not being used with the convenience that it will automatically resume when needed

```
CREATE OR REPLACE WAREHOUSE SAGEMAKER_WH
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 120
  AUTO_RESUME = true;
```

## 3.2 Create a Role and User

3.2.1 Now we will create a role named SAGEMAKER_ROLE that will be used to control access to objects in Snowflake. We will also GRANT all privileges on the SAGEMAKER_WH Warehouse to this role. We will also grant the SAGEMAKER_ROLE to the SYSADMIN role to allow SYSADMIN to have all the SAGEMAKER_ROLE privileges.

```
CREATE OR REPLACE ROLE SAGEMAKER_ROLE COMMENT='SageMaker Role';

GRANT ALL ON WAREHOUSE SAGEMAKER_WH TO ROLE SAGEMAKER_ROLE;

GRANT ROLE SAGEMAKER_ROLE TO ROLE SYSADMIN;

GRANT ROLE SAGEMAKER_ROLE TO ROLE ACCOUNTADMIN;
```

3.2.2 The next step is to create a user that will be used by external services to connect to the Snowflake account. The user SAGEMAKER will be created and assigned a default role, warehouse and database to establish the default context when connected to the Snowflake account.

Note that the SQL statement has a password “AWSSF123”, you can change the password to one that you prefer. Do note the password you use if you do change it as it will be required for the next portion of the lab when Amazon SageMaker will connect to Snowflake.

We will also grant the SAGEMAKE_ROLE to the SAGEMAKER user.

Finally we will switch to the SYSADMIN role for the next steps.

```
CREATE OR REPLACE USER SAGEMAKER PASSWORD='AWSSF123'
   DEFAULT_ROLE=SAGEMAKER_ROLE
   DEFAULT_WAREHOUSE=SAGEMAKER_WH
   DEFAULT_NAMESPACE=ML_WORKSHOP.PUBLIC
   COMMENT='SageMaker User';

GRANT ROLE SAGEMAKER_ROLE TO USER SAGEMAKER;

USE ROLE SYSADMIN;
```

## 3.3 Create a Database and Tables

3.3.1 First, let's create a database called ML_WORKSHOP that will be used for loading the customer churn data. Then assign the usage on the database to the SAGEMAKER_ROLE. We will also grant all privileges on the default schema, PUBLIC, in the database to the SAGEMAKER_ROLE.

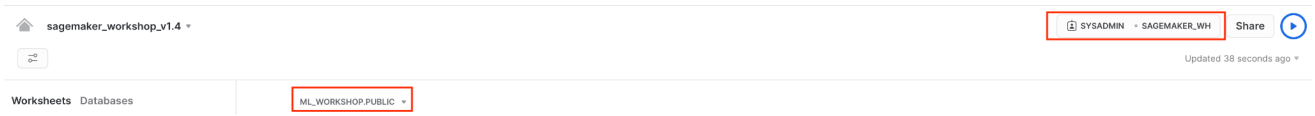
```
CREATE DATABASE IF NOT EXISTS ML_WORKSHOP;

GRANT USAGE ON DATABASE ML_WORKSHOP TO ROLE SAGEMAKER_ROLE;

GRANT ALL ON SCHEMA ML_WORKSHOP.PUBLIC TO ROLE SAGEMAKER_ROLE;
```

3.3.2 Let's now switch context to the ML_WOKRSHOP database and PUBLIC schema. As well as switch to use the SAGEMAKER_WH warehouse. The context for executing SQL commands is shown in the top right hand corner of the worksheets.

```
USE ML_WORKSHOP.PUBLIC;
USE WAREHOUSE SAGEMAKER_WH;
```



3.3.3 Next we will create the CUSTOMER_CHURN table to load customer churn data from a S3 bucket. We will also create a table, ML_RESULTS, to load the predictions from SageMaker at a later stage back into Snowflake. For both tables all privileges will be granted to the SAGEMAKER_ROLE as well.

```
CREATE OR REPLACE TABLE CUSTOMER_CHURN (
        Cust_ID INT,
   State varchar(10),
        Account_Length INT,
        Area_Code INT,
        Phone varchar(10),
        Intl_Plan varchar(10),
        VMail_Plan varchar(10),
        VMail_Message INT,
        Day_Mins FLOAT,
        Day_Calls INT,
        Day_Charge  FLOAT,
        Eve_Mins FLOAT,
        Eve_Calls INT,
        Eve_Charge FLOAT,
        Night_Mins FLOAT,
        Night_Calls INT,
        Night_Charge FLOAT,
        Intl_Mins FLOAT,
        Intl_Calls INT,
        Intl_Charge FLOAT,
        CustServ_Calls INT,
        Churn varchar(10)
);

GRANT ALL ON TABLE CUSTOMER_CHURN TO ROLE SAGEMAKER_ROLE;

CREATE OR REPLACE TABLE ML_RESULTS (
        Churn_IN INT,
   Cust_ID INT,
   Churn_Score REAL
);

GRANT ALL ON TABLE ML_RESULTS TO ROLE SAGEMAKER_ROLE;
```

**Many Options to Run Commands.**
SQL commands can be executed through the UI (limited), via the Worksheets tab, using our SnowSQL command line tool, a SQL editor of your choice via ODBC/JDBC, or through our Python or Spark connectors.

As mentioned earlier, in this lab we will run operations via pre-written SQL in the worksheet (as opposed to using the UI) to save time.

## 3.4 Create a File Format, an External Stage and Internal Stage

We are working with structured, comma-delimited data that has already been staged in a public, external S3 bucket. Before we can use this data, we first need to create an External Stage that specifies the location of our external bucket. We also need to create a File Format for the comma-delimited data.

NOTE - For this lab we are using an AWS-West bucket. In the real-world, to prevent data egress/transfer costs, you would want to select a staging location from the same region that your Snowflake environment is in.

We will also create an Internal Stage that will be used to load the SageMaker predictions back to a SNowflake table for final analysis.

3.4.1 First we will create the File Format that will be used. The CSV data does not have a header. We will also grant privileges on the File Format to the SAGEMAKER_ROLE.

```
CREATE OR REPLACE FILE FORMAT CSVHEADER
   TYPE = 'CSV'
   FIELD_DELIMITER = ','
   SKIP_HEADER = 0;

GRANT USAGE ON FILE FORMAT CSVHEADER TO ROLE SAGEMAKER_ROLE;
```

3.4.2 Next we will create an External Stage to an existing Amazon S3 Location. The data is located in a public S3 bucket and does not require credentials for ease of use with the lab. In practice you will need to configure secure access to Amazon S3. For more information see the SNowflake documentation https://docs.snowflake.com/en/user-guide/data-load-s3.html

```
CREATE OR REPLACE STAGE CHURN_DATA

url='s3://snowflake-corp-se-workshop/sagemaker-snowflake-devdays-v1.5/sourcedata/';
```

3.4.3 We will also grant the necessary privileges to the SAGEMAKER_ROLE to the external stage.

```
GRANT USAGE ON STAGE CHURN_DATA TO ROLE SAGEMAKER_ROLE;
```

3.4.4 We will also create an Internal Stage for loading the Sagemaker results.

```
CREATE OR REPLACE STAGE ML_RESULTS
FILE_FORMAT = (TYPE = CSV);

GRANT READ, WRITE ON STAGE ML_RESULTS TO ROLE SAGEMAKER_ROLE;
```

NOTE - You can look at the data in an external stage before loading it. It is not part of this lab, but you can look and run the SQL commands that are commented out in the SQL Script file.

3.4.5 Finally let's load the Customer Churn data from the S3 bucket into Snowflake.

First we switch to the SAGEMAKER_ROLE.

```
USE ROLE SAGEMAKER_ROLE;
```

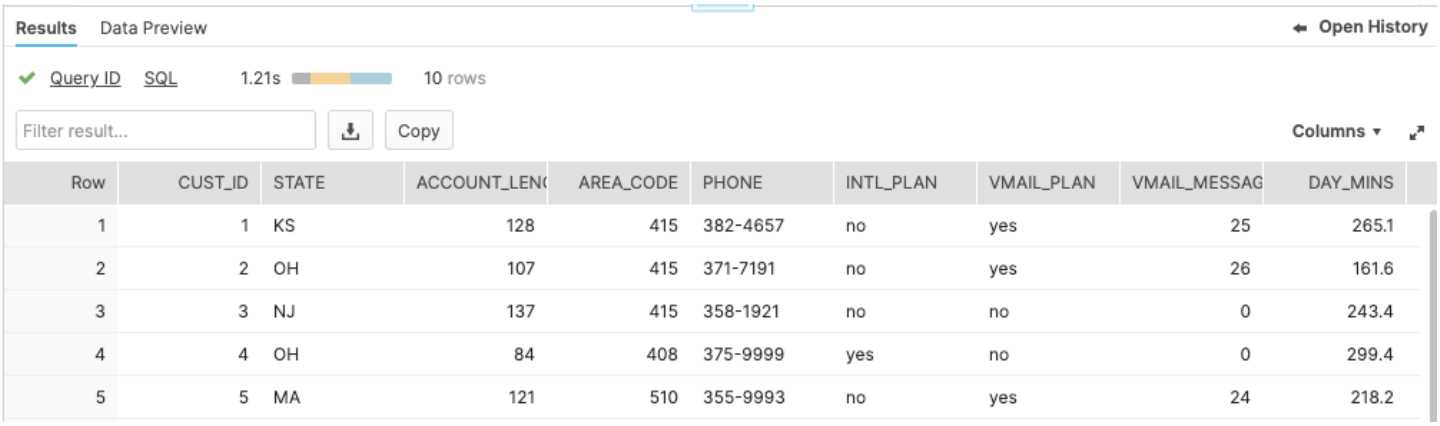
Then we will load the data using the COPY command.

```
COPY INTO CUSTOMER_CHURN FROM @CHURN_DATA/ FILE_FORMAT =
(FORMAT_NAME = CSVHEADER);
```

Let's look at the data by running a SELECT command.

```
SELECT * FROM CUSTOMER_CHURN LIMIT 10;
```

The results are displayed in the frame below the Worksheet.



Results | Data Preview | Open History

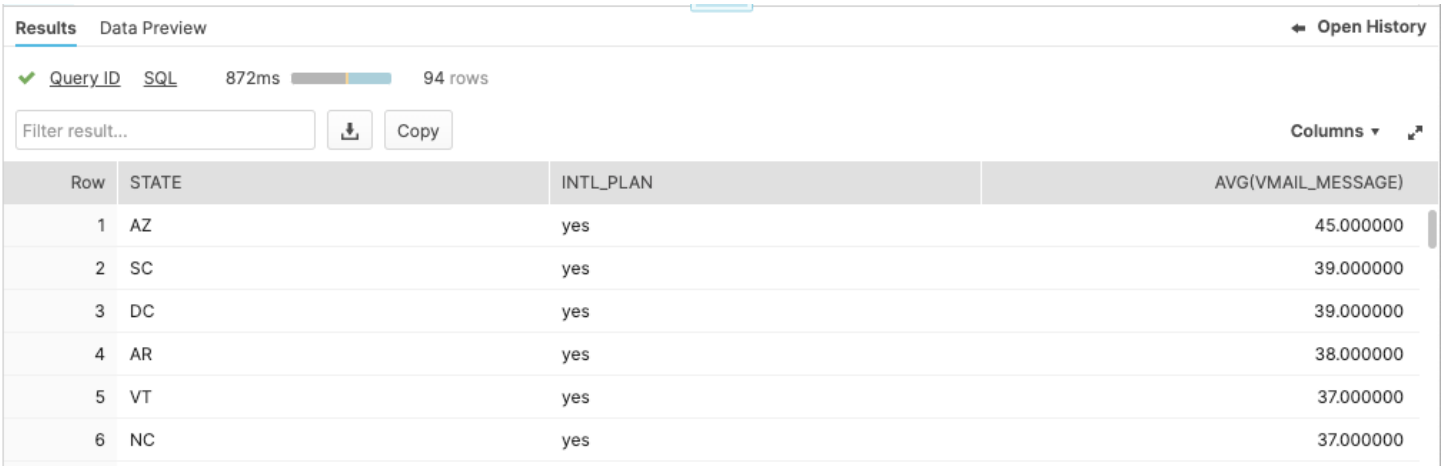
Query ID | SQL | 1.21s | 10 rows

Filter result... | Copy | Columns

| Row | CUST_ID | STATE | ACCOUNT_LEN( | AREA_CODE | PHONE | INTL_PLAN | VMAIL_PLAN | VMAIL_MESSAG | DAY_MINS |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 |
| 2 | 2 | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 |
| 3 | 3 | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 |
| 4 | 4 | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 |
| 5 | 5 | MA | 121 | 510 | 355-9993 | no | yes | 24 | 218.2 |

We can also do some quick analysis on the data in the table by querying the average voicemail messages by state for customers with a voicemail plan.

```
SELECT STATE, INTL_PLAN, AVG(VMAIL_MESSAGE)
FROM CUSTOMER_CHURN
WHERE VMAIL_PLAN = 'yes'
GROUP BY 1,2 ORDER BY AVG(VMAIL_MESSAGE) DESC;
```




| Row | STATE | INTL_PLAN | AVG(VMAIL_MESSAGE) |
|---|---|---|---|
| 1 | AZ | yes | 45.000000 |
| 2 | SC | yes | 39.000000 |
| 3 | DC | yes | 39.000000 |
| 4 | AR | yes | 38.000000 |
| 5 | VT | yes | 37.000000 |
| 6 | NC | yes | 37.000000 |

# Module 4: Deploy an Amazon SageMaker Notebook

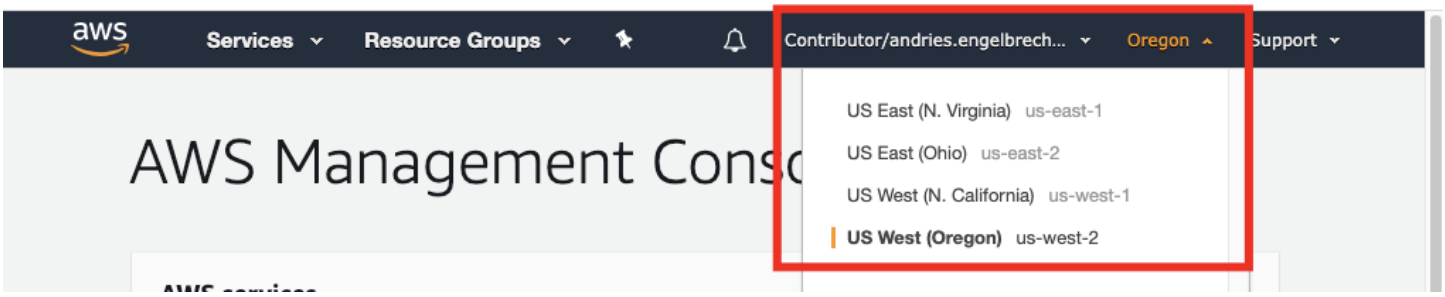
For this module, we will login to the AWS Management Console and deploy a SageMaker Notebook using a CloudFormation Template. We will then upload the workshop Notebook used for this lab.

## 4.1 Deploy the CloudFormation Template

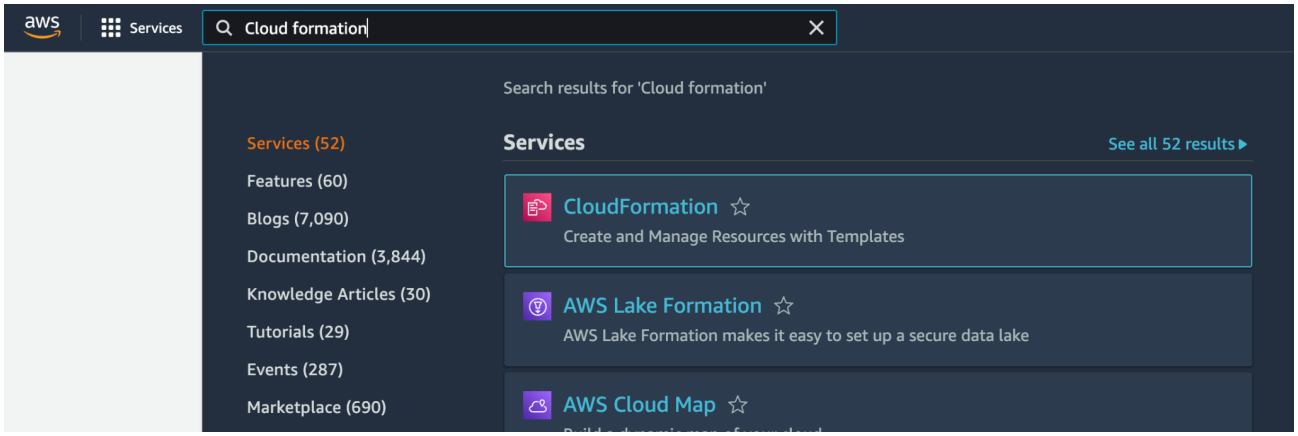
The CloudFormation Template (CFT) will create a SageMaker Notebook Instance and also create a S3 bucket that will be used for the lab. The CFT will also install the Snowflake Python Connector on the Instance for our convenience.

Open another window in your browser and log in to the AWS Management Console, if it is not already logged in.

4.1.1 In the Management Console select the region, preferably the same region as the Snowflake account you are using. In this example us-west-2 (Oregon) region is used.

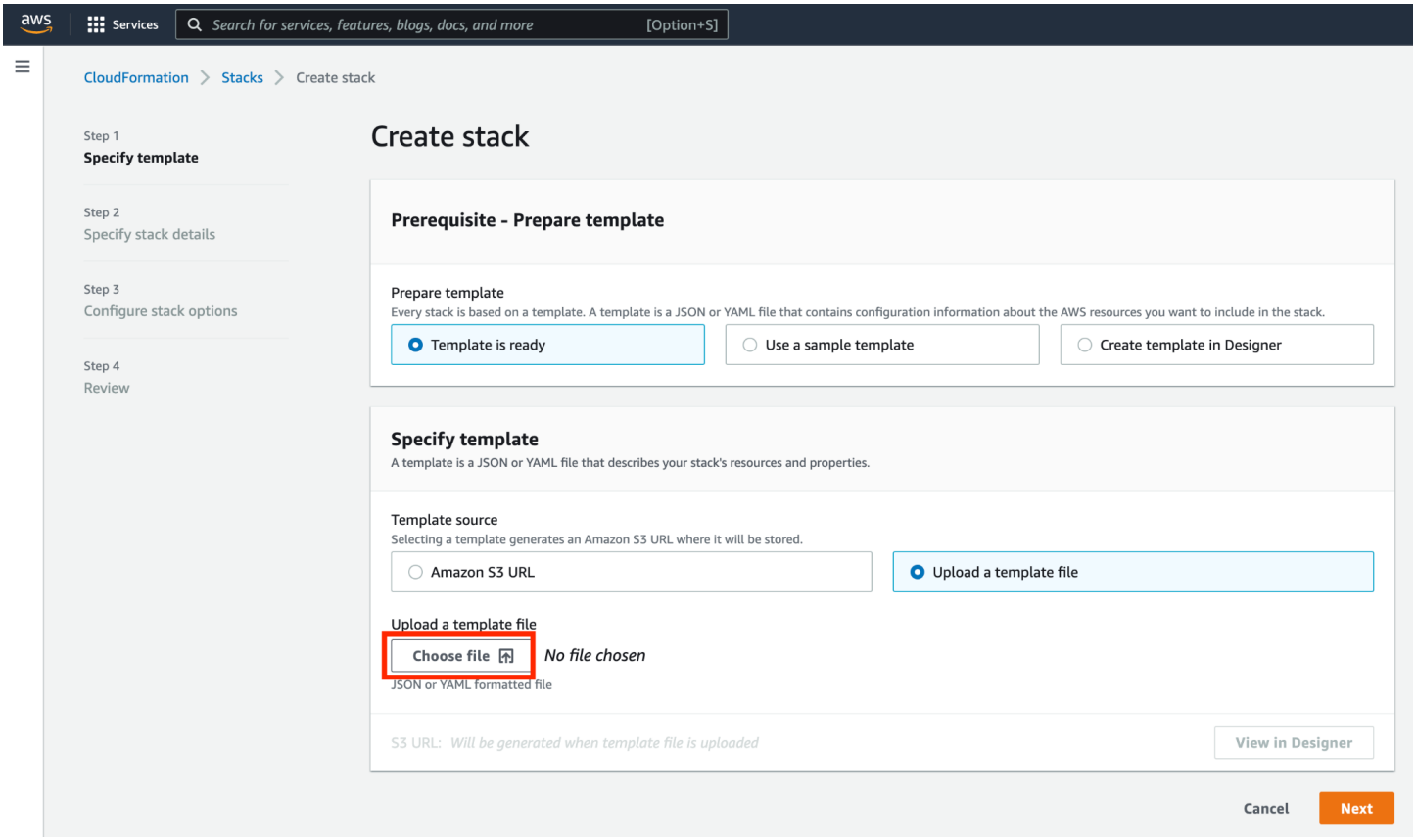



4.1.2 Navigate to the Cloud Formation Service as available in the console

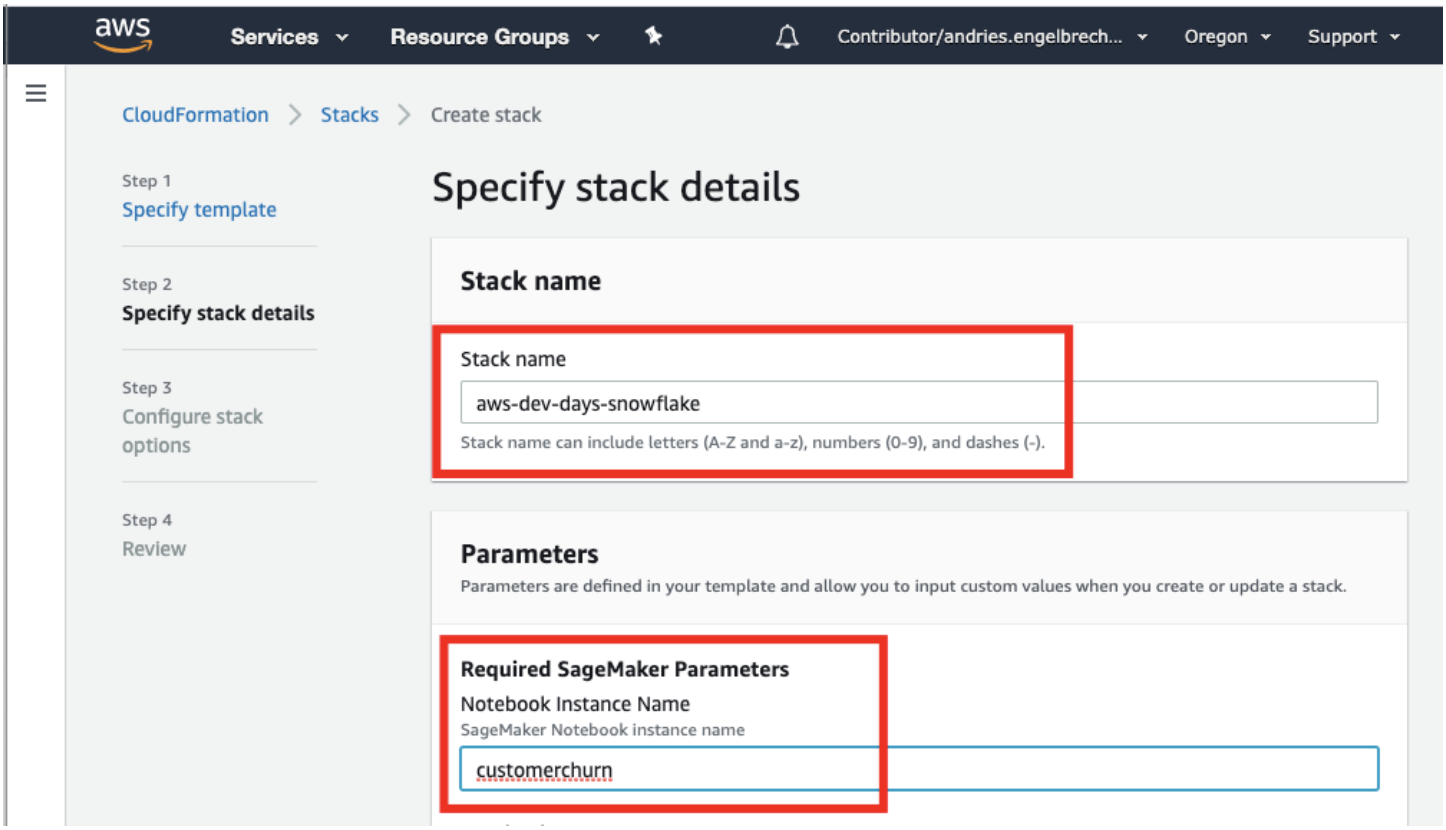


4.1.3 The Cloud Formation tool will open and you should see the Create Stack button on the right. Before proceeding, verify the AWS region by looking at the top right hand bar, similar to the step above.

4.1.4 Select the 'Template is ready' option, select the 'Upload a template file' option, and hit the 'Choose file' button to upload the provided Cloud Formation template (snowflake-sagemaker-notebook-v2.0.yaml)

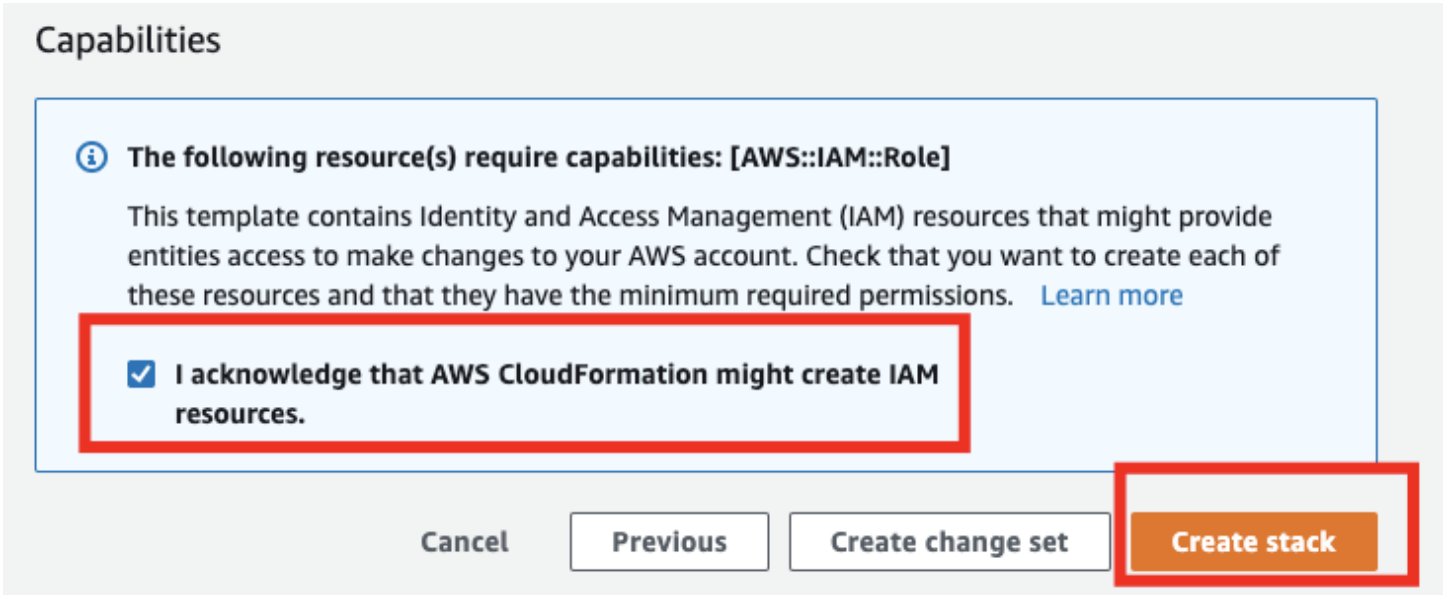



4.1.5 Click the NEXT button on the bottom right. On the following page you will need to assign a unique name for the CFT Stack Name and the Notebook Instance Name. All the other parameters can be left to the default settings for this lab. Scroll down and click the NEXT button again.

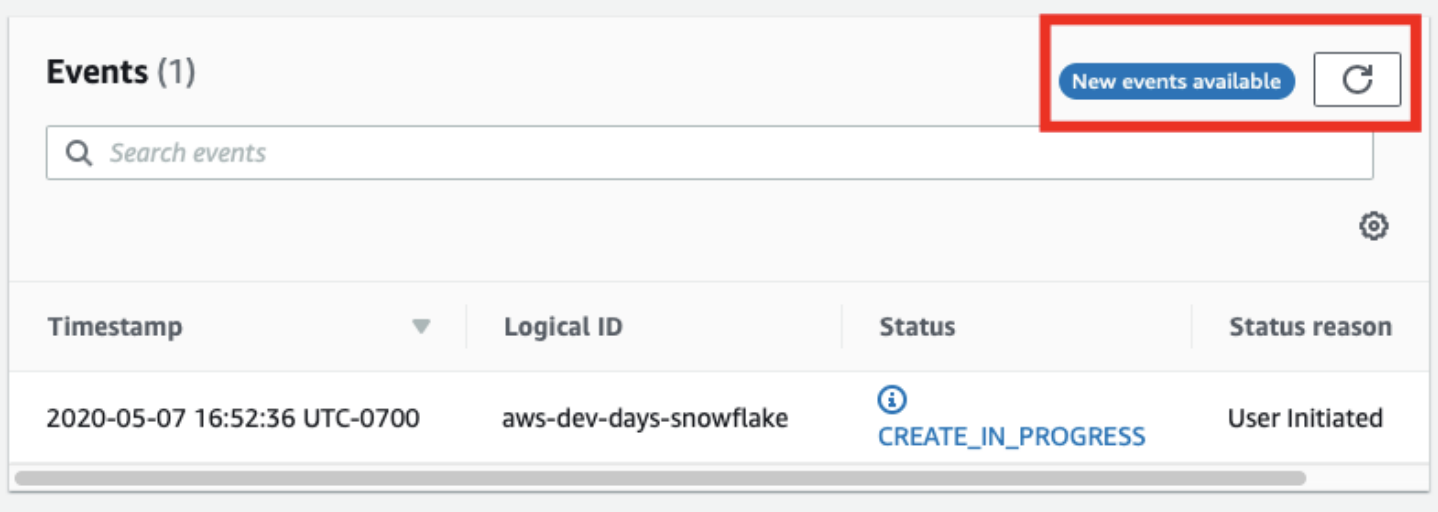



4.1.6 For steps 3 you can leave the defaults in place and just click NEXT. Step 4 will review all the settings. Scroll down and verify Notebook Instance Name. At the bottom of the page check the acknowledgement that IAM resources may be created and click on the Create stack button.

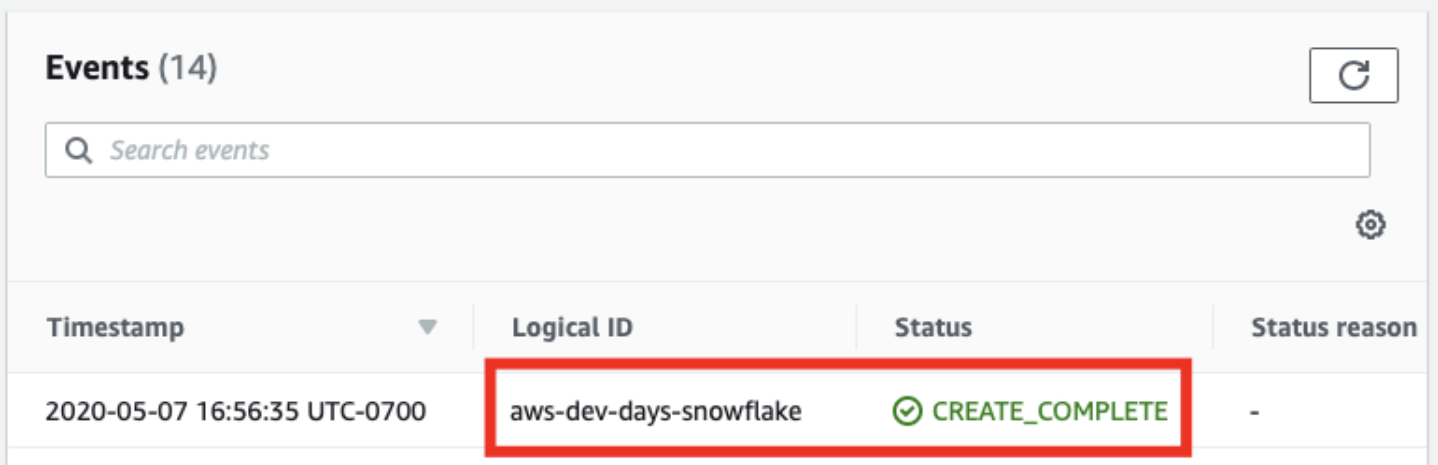
Note: It can take a few minutes to deploy the SageMaker Notebook.



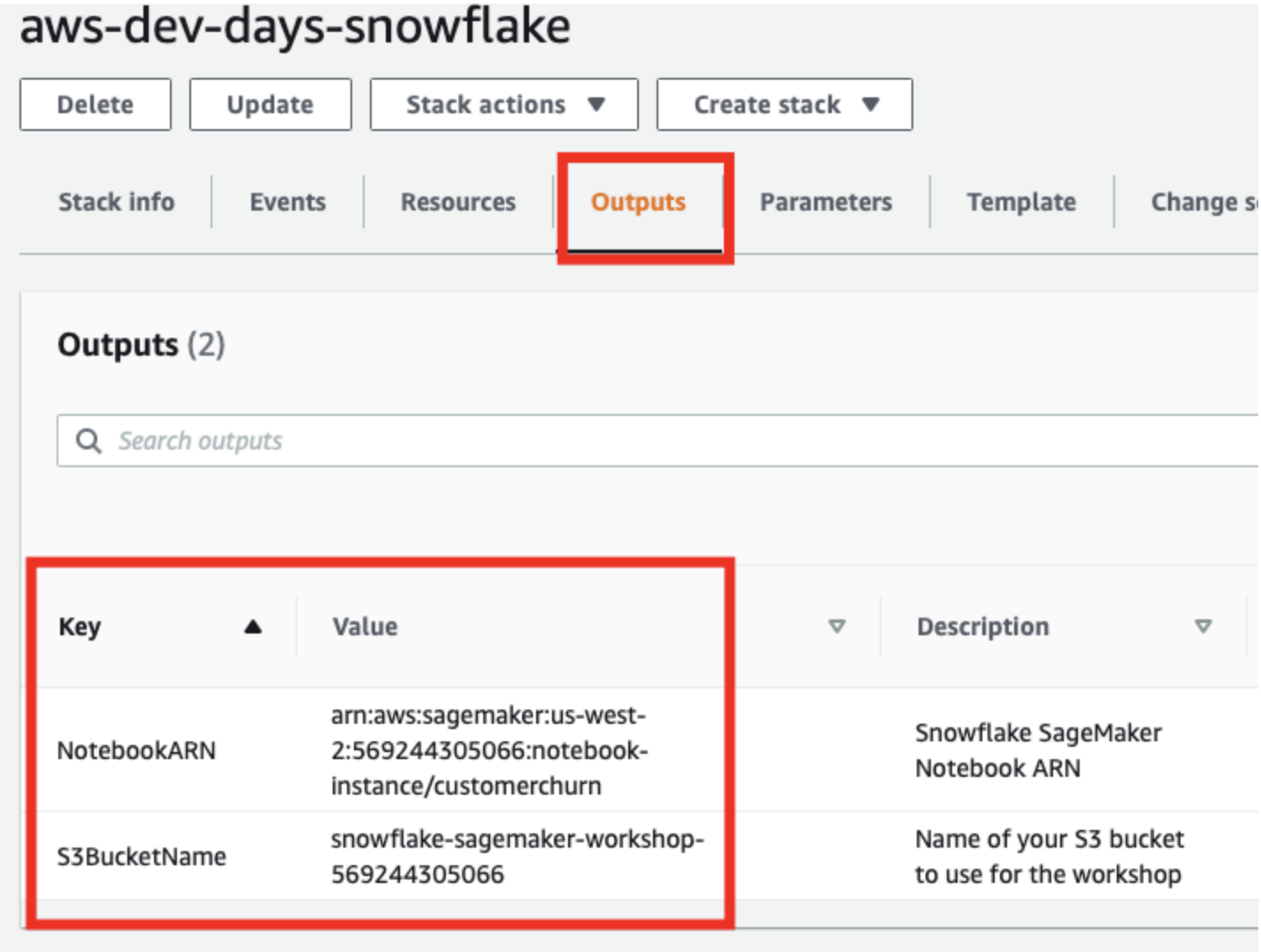
4.1.7 You can refresh the Events page by clicking the circular arrow on the top right hand side.




4.1.8 Once the Stack has completed you will see the Stack Name and a Status of CREATE_COMPLETE.



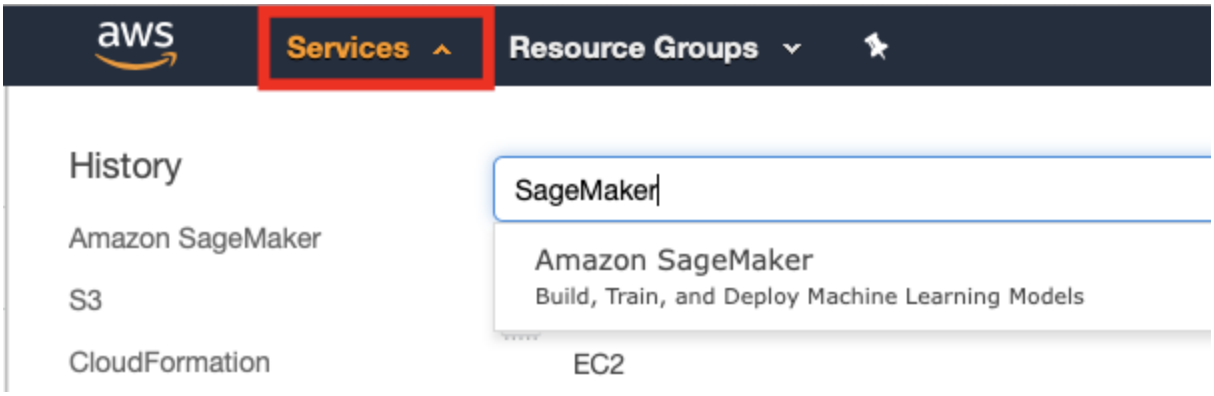

4.1.9 In the CloudFormation page click on the Outputs tab. It will list the SageMaker Notebook ARN value as well as the S3 Bucket's name that was created. Note the S3 bucket name and the SageMaker Notebook Instance name for later in the lab.



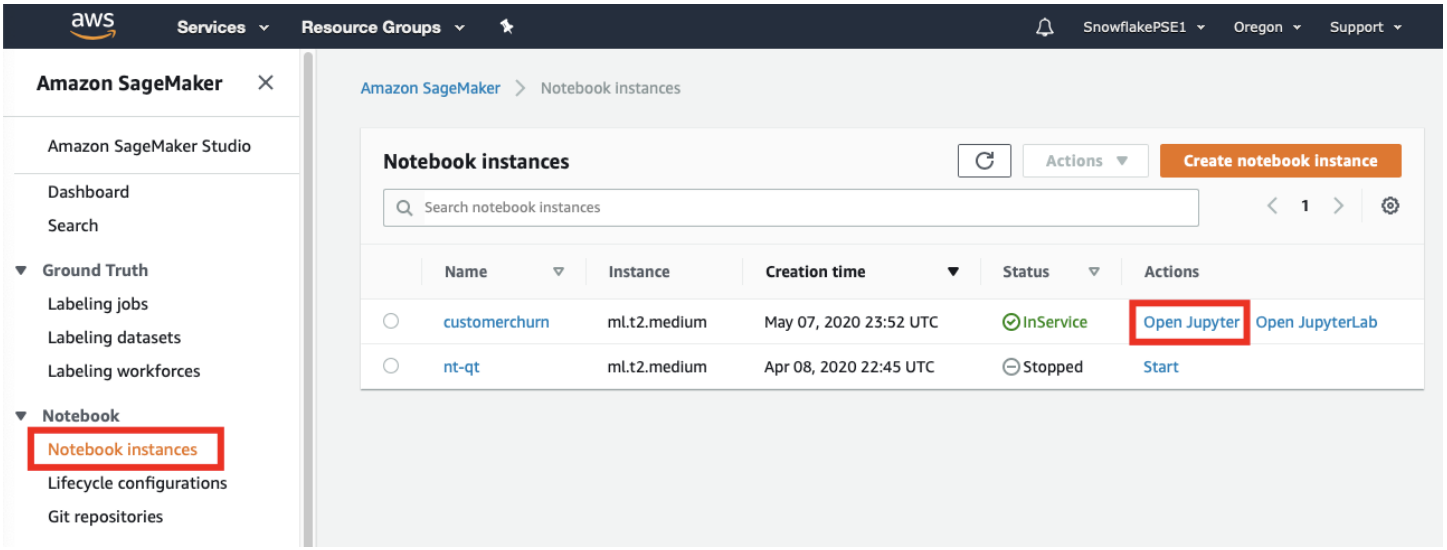

## 4.2 Upload the Notebook to the SageMaker instance

Now we can upload the notebook that was downloaded for the lab to SageMaker.

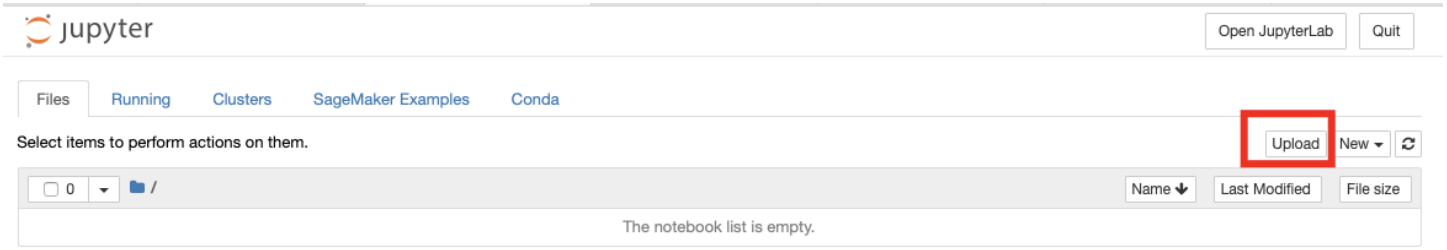
4.2.1 In the AWS console navigate to the SageMaker service. The easiest way to do it is by clicking on the Services menu at the top left and then typing SageMaker in the search bar. Click on Amazon SageMaker once it appears below the search bar.

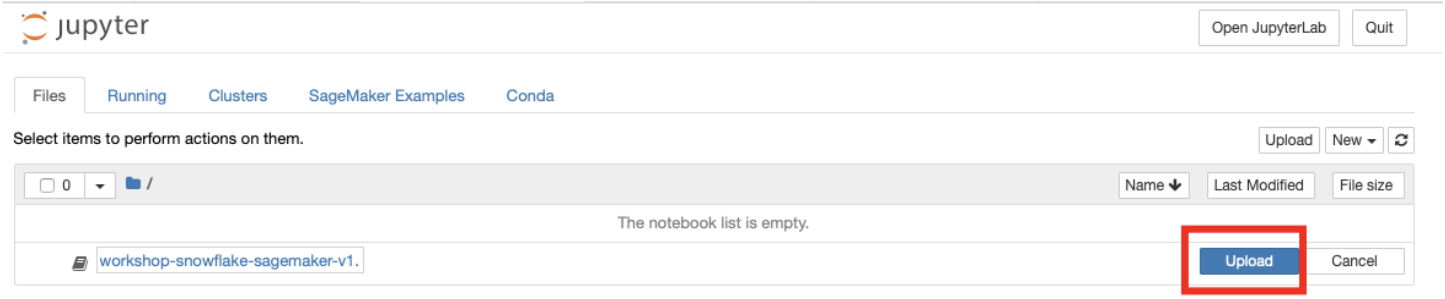



4.2.2 Once in the SageMaker console, click on the Notebook Instance menu on the left hand side. Then click on the Open Jupyter link next to the Notebook Instance you specified in the CFT.




4.2.3 This will open a new tab in your browser. On the top right hand corner you will see an Upload button. Click it and select the notebook file (.ipynb) that you downloaded to your computer at the start of the lab. To complete uploading it click the blue highlighted upload button next to the filename.






4.2.4 The notebook is now uploaded and ready to use.

# Module 5: Machine Learning Workflow in SageMaker

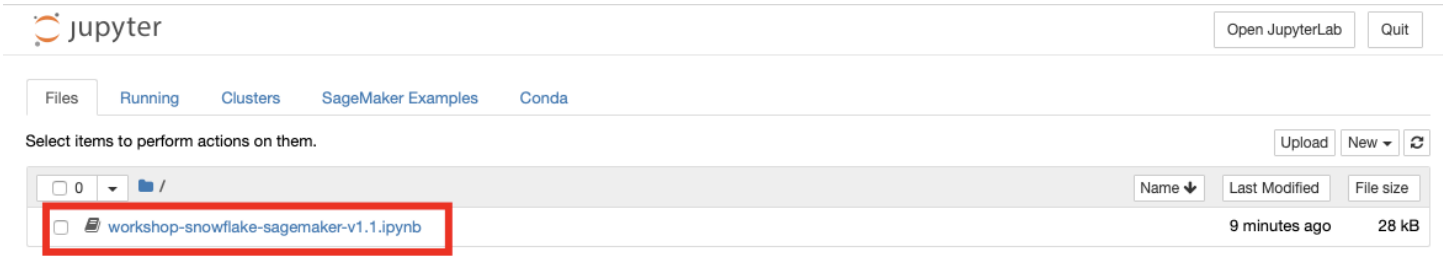
In this module we will perform a basic Machine Learning workflow in a SageMaker Notebook. We will be performing the following steps:

- Query data from Snowflake
- Explore and visualize the data
- Do feature selection
- Train a model
- Compile a model
- Run batch inference using SageMaker using the model
- Upload the model's Churn Score predictions to Snowflake

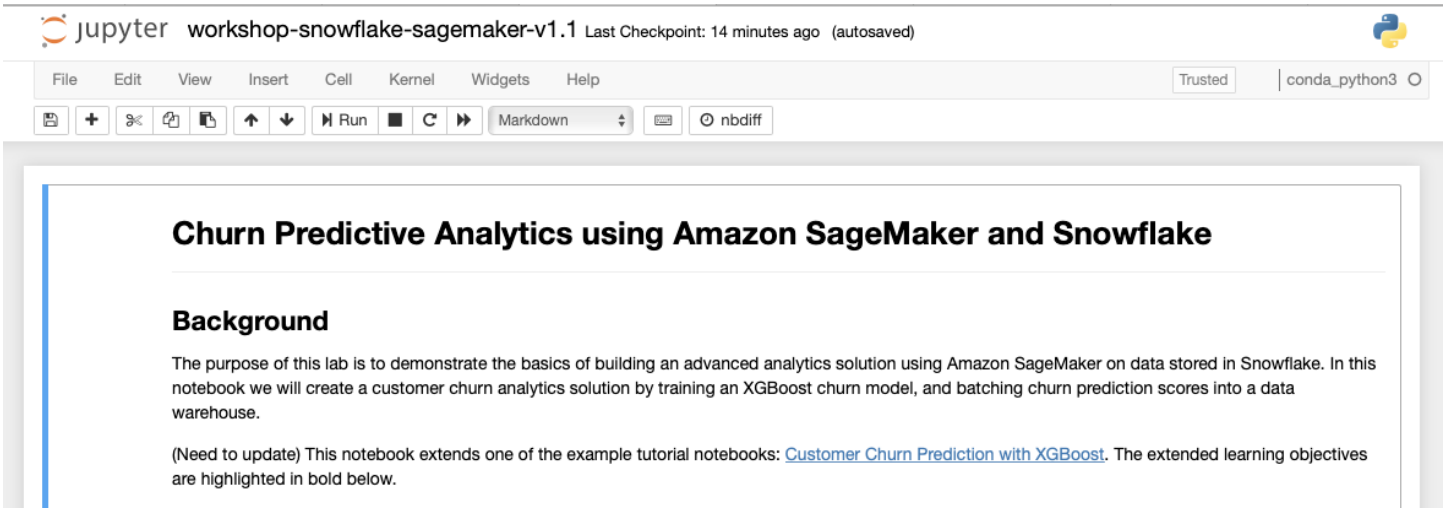
**Note:** The steps and process is very well documented in the Notebook itself with various comments. We encourage you to carefully read through each section of the Notebook.

## 5.1 Open the Notebook

5.1.1 After we uploaded the notebook file to SageMaker we can now open it by simply clicking on the file name.




5.1.2 Once you click on the notebook link it will open a new tab in your browser and provide you with the Jupyter notebook interface seen below.

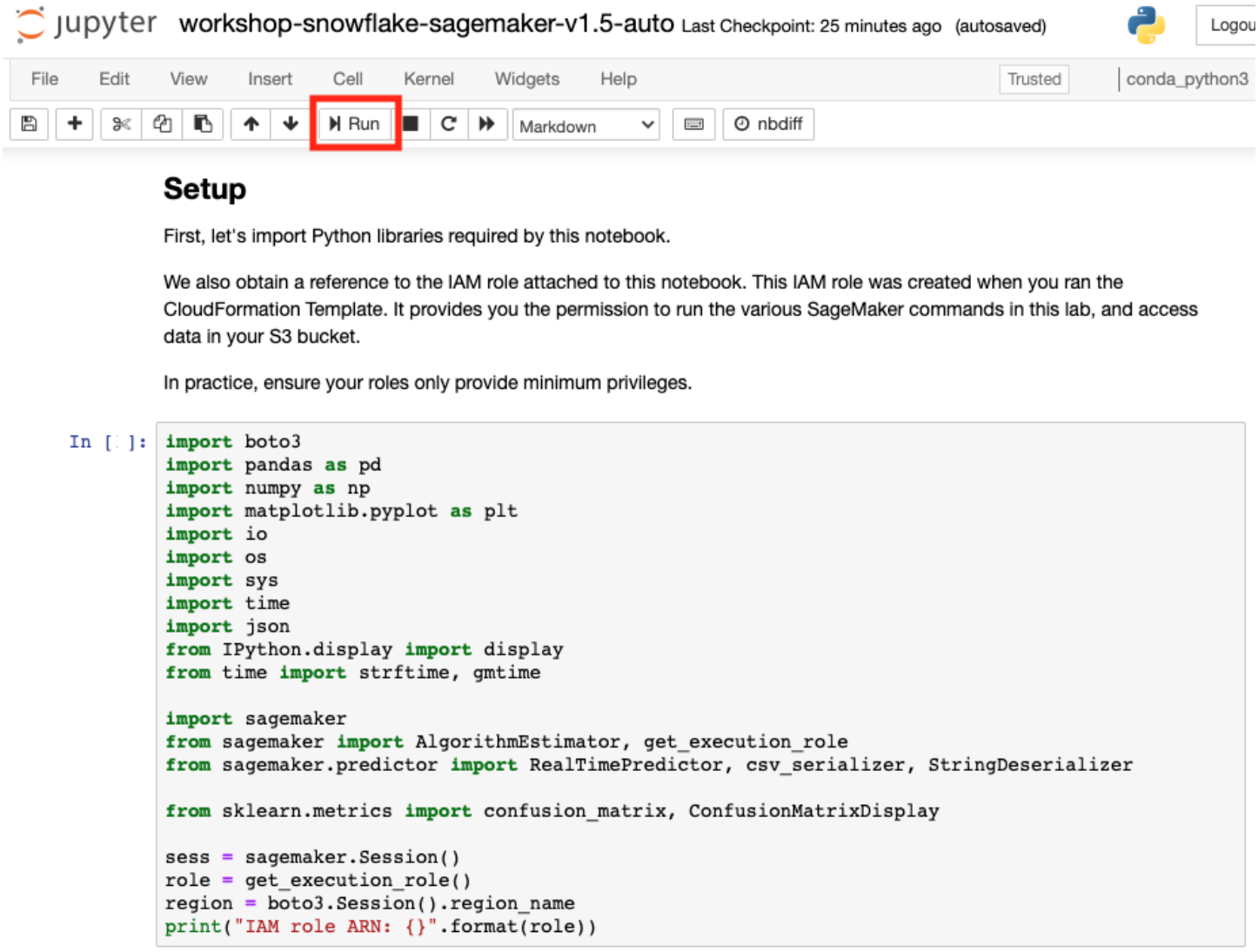



## 5.2 Execute the Machine Learning workflow in the Notebook

The notebook consists of comment sections and code sections. To execute a code section click on it to highlight it and then click on the Run button on the toolbar. *An alternative to clicking the Run button is to use Shift+Return or Enter keyboard shortcuts*.

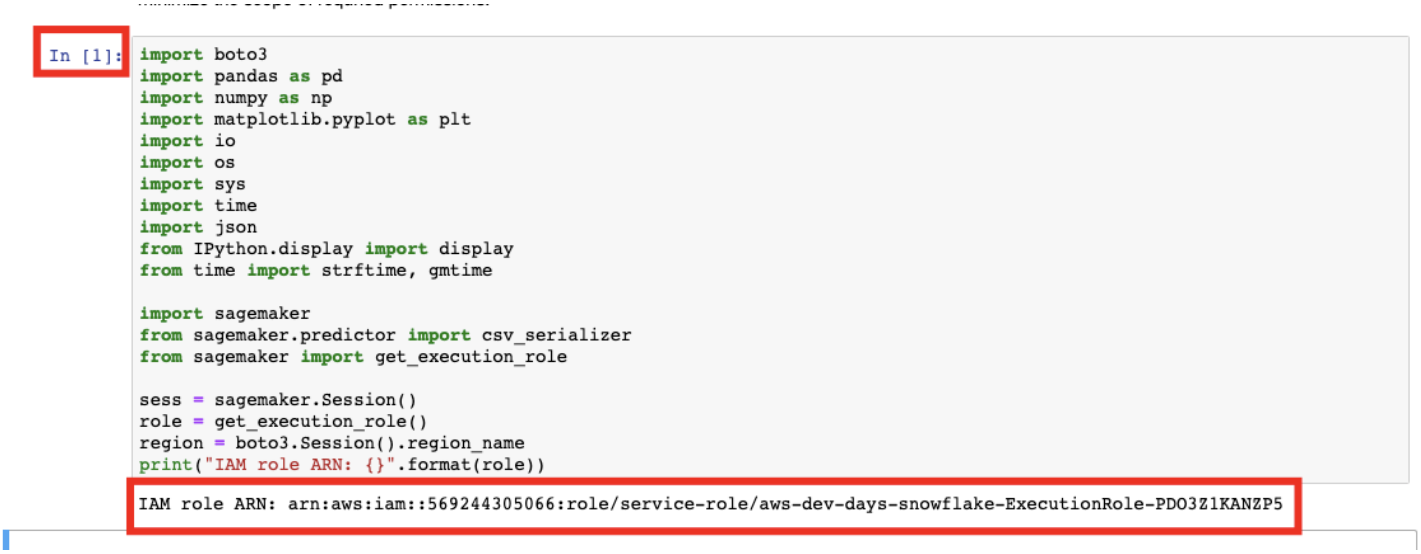
We will execute one code block at a time and make sure to work through the comments to describe what each step is doing. Do not click the run button yet, but get familiar with the interface.

5.2.1 Read through the comments and scroll down to the first code block below the **Setup** section. Click on the code block to highlight it and then click on Run.




The setup code block loads a number of libraries that will be used, as well as checks the IAM role being used.

Once a code block has been executed it will put a number in the square brackets next to it and an output below it, as seen in the example below.



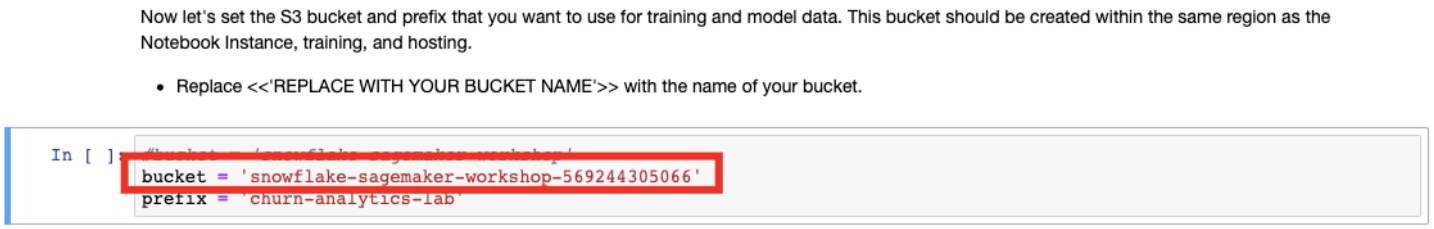
```
In [1]:
import boto3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import os
import sys
import time
import json
from IPython.display import display
from time import strftime, gmtime

import sagemaker
from sagemaker.predictor import csv_serializer
from sagemaker import get_execution_role

sess = sagemaker.Session()
role = get_execution_role()
region = boto3.Session().region_name
print("IAM role ARN: {}".format(role))

IAM role ARN: arn:aws:iam::569244305066:role/service-role/aws-dev-days-snowflake-ExecutionRole-PDO3Z1KANZP5
```

5.2.2 Next, let's configure the notebook to use the S3 bucket that was created by the CFT and the name that was captured from step 4.1.8. Replace the <REPLACE WITH YOUR BUCKET NAME> text with the name of your S3 bucket. **Do note that your bucket name will be different from the one listed in the examples.** Also remember to click the Run button after selecting the code box.



Now let's set the S3 bucket and prefix that you want to use for training and model data. This bucket should be created within the same region as the Notebook Instance, training, and hosting.

- Replace <<'REPLACE WITH YOUR BUCKET NAME'>> with the name of your bucket.

```
In [ ]:
bucket = 'snowflake-sagemaker-workshop-569244305066'
prefix = 'churn-analytics-lab'
```

5.2.3 After reading the description of the data set in the Data comment section we will configure the connection to Snowflake in the Notebook.

For the user and password you have to use the username and password you configured in your Snowflake account earlier in section 3.2.2. There is no need to change if you used the defaults provided in the SQL script.

The **Snowflake Account Name** can be derived by executing the following statement in your Snowflake environment:

```
select current_account();
```

```
206
207   select current_account();
208
```

Objects | Editor | Results | Chart

| | CURRENT_ACCOUNT() |
|---|---|
| 1 | HM53794 |

In this example, the account name is HM53794, but yours will be different.

Snowflake is deployed in various AWS regions. For the US West region you only need the account name itself. For other regions you will see a region identifier in the Snowflake URL, as an example in the US East region the URL i.e. https://app.snowflake.com/us-east-11/HM53794/w8KS9hjgPpD#query In this case you will need to identify your account with the account name and region identifier -

For this example we will use a fictitious account identifier in the US East region. Replace <ACCOUNT> with your SNowflake account name and region identifier.

Provide the connection and credentials required to connect to your Snowflake account. You'll need to modify the cell below with the appropriate **ACCOUNT** for your Snowflake trial. If you followed the lab guide instructions, the username and password below will work.

**NOTE:** For Snowflake accounts in regions other than US WEST add the Region ID after a period . i.e. XYZ123456.US-EAST-1.

In practice, security standards might prohibit you from providing credentials in clear text. As a best practice in production, you should utilize a service like AWS Secrets Manager to manage your database credentials.

In [ ]:
```
import snowflake.connector
# Connecting to Snowflake using the default authenticator
ctx = snowflake.connector.connect(
  user='sagemaker',
  password='AWSSF123',
  account='cd123456.us-east-1',
  warehouse='SAGEMAKER_WH',
  database='ML_WORKSHOP',
  schema='PUBLIC'
)
```

5.2.4 We will work through each of the comment and code blocks using the detailed comments in the Notebook. There are a number of well documented steps and not all of them will be covered in detail in the lab guide.

The following steps will be performed:

- Query the data in Snowflake and loading it into a pandas dataframe
- Exploring and visualization of the data
- Prepping the data
- Splitting the data set into training, testing and validation data sets
- The S3 bucket will be used to store the data sets

### 5.2.5 Explore

Now we can run queries against your database.

However, in practice, the data table will often contain more data than what is practical to operate on within a notebook instance, or relevant attributes are spread across multiple tables. Being able to run SQL queries and loading the data into a pandas dataframe will be helpful during the initial stages of development. Check out the Spark integration for a fully scalable solution. [Snowflake Connector for Spark]

```
In [34]: # Query Snowflake Data
cs=ctx.cursor()
allrows=cs.execute("""select Cust_ID,STATE,ACCOUNT_LENGTH,AREA_CODE,PHONE,INTL_PLAN,VMAIL_PLAN,VMAIL_MESSAGE,
                      DAY_MINS,DAY_CALLS,DAY_CHARGE,EVE_MINS,EVE_CALLS,EVE_CHARGE,NIGHT_MINS,NIGHT_CALLS,
                      NIGHT_CHARGE,INTL_MINS,INTL_CALLS,INTL_CHARGE,CUSTSERV_CALLS,
                      CHURN from CUSTOMER_CHURN """).fetchall()

churn = pd.DataFrame(allrows)
churn.columns=['Cust_id','State','Account Length','Area Code','Phone','Intl Plan', 'VMail Plan', 'VMail Message','Day M
               'Day Calls', 'Day Charge', 'Eve Mins', 'Eve Calls', 'Eve Charge', 'Night Mins', 'Night Calls','Night Charge
               'Intl Mins','Intl Calls','Intl Charge','CustServ Calls', 'Churn?']

pd.set_option('display.max_columns', 500)     # Make sure we can see all of the columns
pd.set_option('display.max_rows', 10)         # Keep the output on one page
churn
```

Out[34]:

| | Cust_id | State | Account Length | Area Code | Phone | Intl Plan | VMail Plan | VMail Message | Day Mins | Day Calls | Day Charge | Eve Mins | Eve Calls | Eve Charge | Night Mins | Night Calls | Night Charge | Intl Mins | Intl Calls | Intl Charge | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | 197.4 | 99 | 16.78 | 244.7 | 91 | 11.01 | 10.0 | 3 | 2.70 | |
| 1 | 2 | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | 195.5 | 103 | 16.62 | 254.4 | 103 | 11.45 | 13.7 | 3 | 3.70 | |
| 2 | 3 | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | 121.2 | 110 | 10.30 | 162.6 | 104 | 7.32 | 12.2 | 5 | 3.29 | |
| 3 | 4 | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | 61.9 | 88 | 5.26 | 196.9 | 89 | 8.86 | 6.6 | 7 | 1.78 | |
| 4 | 5 | MA | 121 | 510 | 355-9993 | no | yes | 24 | 218.2 | 88 | 37.09 | 348.5 | 108 | 29.62 | 212.6 | 118 | 9.57 | 7.5 | 7 | 2.03 | |

### 5.2.6 Additional exploration

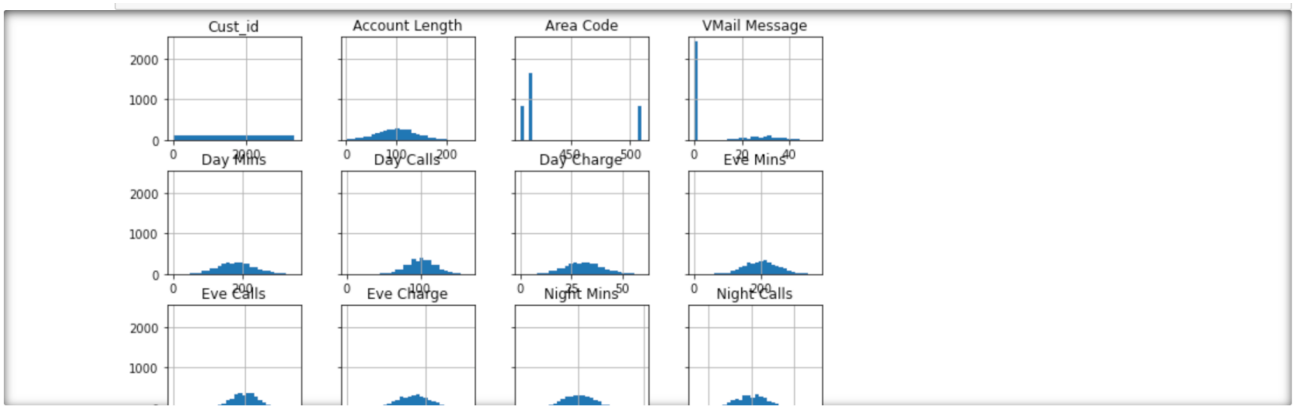
Execute the next cell to generate Frequency Tables for each column, as well as histograms for all attributes.

```
In [6]: # Frequency tables for each categorical feature
for column in churn.select_dtypes(include=['object']).columns:
    display(pd.crosstab(index=churn[column], columns='% observations', normalize='columns'))

# Histograms for each numeric features
display(churn.describe())
%matplotlib inline
hist = churn.hist(bins=30, sharey=True, figsize=(10, 10))
```

| col_0 | % observations |
|---|---|
| State | |
| AK | 0.015602 |
| AL | 0.024002 |
| AR | 0.016502 |
| AZ | 0.019202 |
| CA | 0.010201 |
| ... | ... |
| VT | 0.021902 |
| WA | 0.019802 |
| WI | 0.023402 |
| WV | 0.031803 |

Notice that the output window is scrollable, and be browsing down, you will be able to reach the histograms.




5.2.7 By analyzing the output of the exploration, we can learn the following:
- `State` appears to be quite evenly distributed
- `Phone` takes on too many unique values to be of any practical use. It's possible parsing out the prefix could have some value, but without more context on how these are allocated, we should avoid using it.
- Only 14% of customers churned, so there is some class imabalance, but nothing extreme.
- Most of the numeric features are surprisingly nicely distributed, with many showing bell-like gaussianity. `VMail Message` being a notable exception (and `Area Code` showing up as a feature we should convert to non-numeric).

Because of this, we need to make some changes to the dataset. the `Phone` attribute will be removed from the dataset, and the `Area Code` will be converted to a numeric value. Execute the next cell to do this.

In [7]:
```
churn = churn.drop('Phone', axis=1)
churn['Area Code'] = churn['Area Code'].astype(object)
```

5.2.8 Relationship between the features

Now, let's look at the relationship between each of the features in our target variable. Execute the next cell and read through the output of the statements. Note that the output window is (again) scrollable.

```
In [8]: for column in churn.select_dtypes(include=['object']).columns:
            if column != 'Churn?':
                display(pd.crosstab(index=churn[column], columns=churn['Churn?'], normalize='columns'))

        for column in churn.select_dtypes(exclude=['object']).columns:
            print(column)
            hist = churn[[column, 'Churn?']].hist(by='Churn?', bins=30)
            plt.show()
```

| Churn? | False. | True. |
|---|---|---|
| State | | |
| AK | 0.017193 | 0.006211 |
| AL | 0.025263 | 0.016563 |
| AR | 0.015439 | 0.022774 |
| AZ | 0.021053 | 0.008282 |
| CA | 0.008772 | 0.018634 |
| ... | ... | ... |
| VT | 0.022807 | 0.016563 |
| WA | 0.018246 | 0.028986 |
| WI | 0.024912 | 0.014493 |
| WV | 0.033684 | 0.020704 |

Interestingly we see that churners appear:

- Fairly evenly distributed geographically
- More likely to have an international plan
- Less likely to have a voicemail plan
- To exhibit some bimodality in daily minutes (either higher or lower than the average for non-churners)
- To have a larger number of customer service calls (which makes sense as we'd expect customers who experience lots of problems may be more likely to churn)

In addition, we see that churners take on very similar distributions for features like Day Mins and Day Charge. That's not surprising as we'd expect minutes spent talking to correlate with charges.

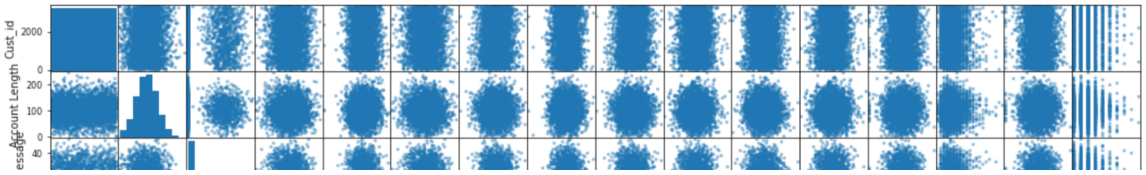
5.2.9 Let's dig deeper into the relationships between our features.

Execute the next cell to dive deeper into the relationships between the features.

In [9]:
```
display(churn.corr())
pd.plotting.scatter_matrix(churn, figsize=(18, 18))
plt.show()
```

| | Cust_id | Account Length | VMail Message | Day Mins | Day Calls | Day Charge | Eve Mins | Eve Calls | Eve Charge | Night Mins | Night Calls | Night Charge | Intl Mins | Intl Calls |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Cust_id** | 1.000000 | 0.038367 | -0.022238 | -0.022226 | -0.004176 | -0.022226 | 0.021881 | 0.013597 | 0.021889 | 0.015647 | -0.002931 | 0.015671 | 0.003058 | -0.014506 |
| **Account Length** | 0.038367 | 1.000000 | -0.004628 | 0.006216 | 0.038470 | 0.006214 | -0.006757 | 0.019260 | -0.006745 | -0.008955 | -0.013176 | -0.008960 | 0.009514 | 0.020661 |
| **VMail Message** | -0.022238 | -0.004628 | 1.000000 | 0.000778 | -0.009548 | 0.000776 | 0.017562 | -0.005864 | 0.017578 | 0.007681 | 0.007123 | 0.007663 | 0.002856 | 0.013957 |
| **Day Mins** | -0.022226 | 0.006216 | 0.000778 | 1.000000 | 0.006750 | 1.000000 | 0.007043 | 0.015769 | 0.007029 | 0.004323 | 0.022972 | 0.004300 | -0.010155 | 0.008033 |
| **Day Calls** | -0.004176 | 0.038470 | -0.009548 | 0.006750 | 1.000000 | 0.006753 | -0.021451 | 0.006462 | -0.021449 | 0.022938 | -0.019557 | 0.022927 | 0.021565 | 0.004574 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **Night Charge** | 0.015671 | -0.008960 | 0.007663 | 0.004300 | 0.022927 | 0.004301 | -0.012593 | -0.002056 | -0.012601 | 0.999999 | 0.011188 | 1.000000 | -0.015214 | -0.012329 |
| **Intl Mins** | 0.003058 | 0.009514 | 0.002856 | -0.010155 | 0.021565 | -0.010157 | -0.011035 | 0.008703 | -0.011043 | -0.015207 | -0.013605 | -0.015214 | 1.000000 | 0.032304 |
| **Intl Calls** | -0.014506 | 0.020661 | 0.013957 | 0.008033 | 0.004574 | 0.008032 | 0.002541 | 0.017434 | 0.002541 | -0.012353 | 0.000305 | -0.012329 | 0.032304 | 1.000000 |
| **Intl Charge** | 0.002994 | 0.009546 | 0.002884 | -0.010092 | 0.021666 | -0.010094 | -0.011067 | 0.008674 | -0.011074 | -0.015180 | -0.013630 | -0.015186 | 0.999993 | 0.032372 |
| **CustServ Calls** | 0.014692 | -0.003796 | -0.013263 | -0.013423 | -0.018942 | -0.013427 | -0.012985 | 0.002423 | -0.012987 | -0.009288 | -0.012802 | -0.009277 | -0.009640 | -0.017561 |

16 rows × 16 columns




We see several features that essentially have 100% correlation with one another. Including these feature pairs in some machine learning algorithms can create catastrophic problems, while in others it will only introduce minor redundancy and bias. Let's remove one feature from each of the highly correlated pairs: Day Charge from the pair with Day Mins, Night Charge from the pair with Night Mins, Intl Charge from the pair with Intl Mins:

Execute the next cell to do this.

In [ ]:
```
churn = churn.drop(['Day Charge', 'Eve Charge', 'Night Charge', 'Intl Charge'], axis=1)
```

### 5.2.10 Selecting the algorithm

Now that we've cleaned up our dataset, let's determine which algorithm to use. As mentioned above, there appear to be some variables where both high and low (but not intermediate) values are predictive of churn. In order to accommodate this in an algorithm like linear regression, we'd need to generate polynomial (or bucketed) terms. Instead, let's attempt to model this problem using gradient boosted trees. Amazon SageMaker provides an XGBoost container that we can use to train in a managed, distributed setting, and then host as a real-time prediction endpoint. XGBoost uses gradient boosted trees which naturally account for non-linear relationships between features and the target variable, as well as accommodating complex interactions between features.

Amazon SageMaker XGBoost can train on data in either a CSV or LibSVM format. For this example, we'll stick with CSV. It should:

- Have the predictor variable in the first column
- Not have a header row

But first, let's convert our categorical features into numeric features by executing the next cell.

In [11]:
```
model_data = pd.get_dummies(churn)
model_data = pd.concat([model_data['Churn?_True.'], model_data.drop(['Churn?_False.', 'Churn?_True.'], axis=1)], axis=1
to_split_data = model_data.drop(['Cust_id'], axis=1)
```

And now let's split the data into training, validation, and test sets. This will help prevent us from overfitting the model, and allow us to test the models accuracy on data it hasn't already seen.

In [12]:
```
train_data, validation_data, test_data = np.split(to_split_data.sample(frac=1, random_state=1729), [int(0.7 * len(to_sp
train_data.to_csv('train.csv', header=False, index=False)
validation_data.to_csv('validation.csv', header=False, index=False)
```

In [13]:
```
pd.set_option('display.max_columns', 100)
pd.set_option('display.width', 1000)
display(train_data)
```

| | Churn? _True. | Account Length | VMail Message | Day Mins | Day Calls | Eve Mins | Eve Calls | Night Mins | Night Calls | Intl Mins | Intl Calls | CustServ Calls | State_AK | State_AL | State_AR | State_AZ | State_CA | State_CO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1095** | 0 | 116 | 35 | 182.8 | 122 | 212.7 | 119 | 193.8 | 103 | 11.0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **608** | 0 | 144 | 33 | 251.6 | 87 | 197.6 | 118 | 209.2 | 97 | 12.2 | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2908** | 0 | 3 | 0 | 185.0 | 120 | 203.7 | 129 | 170.5 | 89 | 14.1 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| **943** | 0 | 56 | 0 | 91.1 | 90 | 179.3 | 115 | 300.7 | 89 | 11.9 | 8 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **693** | 0 | 144 | 0 | 177.5 | 93 | 287.4 | 75 | 180.5 | 118 | 11.9 | 3 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2318** | 0 | 64 | 43 | 118.4 | 100 | 144.1 | 108 | 158.1 | 91 | 8.5 | 6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1405** | 0 | 155 | 23 | 243.9 | 112 | 133.0 | 106 | 213.7 | 123 | 13.4 | 11 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **711** | 0 | 39 | 0 | 160.4 | 68 | 102.6 | 103 | 235.3 | 106 | 9.1 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1241** | 0 | 88 | 0 | 61.9 | 78 | 262.6 | 114 | 212.5 | 110 | 8.8 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1664** | 0 | 112 | 0 | 168.6 | 102 | 298.0 | 117 | 194.7 | 110 | 9.8 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

2333 rows × 70 columns

And finally, upload the modified dataset back to S3.

In [14]:
```
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('va
```

### 5.2.11 Train the model

Moving onto training, first we'll need to specify the locations of the XGBoost algorithm containers.

In [15]:
```
from sagemaker.amazon.amazon_estimator import image_uris
xgb_training_container = image_uris.retrieve(region=boto3.Session().region_name, framework='xgboost', version='0.90-1')
```

Then, because we're training with the CSV file format, we'll create s3_inputs that our training function can use as a pointer to the files in S3.

```
In [16]: s3_input_train = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/train'.format(bucket, prefix), content_type='csv')
         s3_input_validation = sagemaker.inputs.TrainingInput(s3_data='s3://{}/{}/validation/'.format(bucket, prefix), content_t
```

### 5.2.12 Specify the Hyper Parameters and start the training

Now, we can specify a few parameters like what type of training instances we'd like to use and how many, as well as our XGBoost hyperparameters. A few key hyperparameters are:

- max_depth controls how deep each tree within the algorithm can be built. Deeper trees can lead to better fit, but are more computationally expensive and can lead to overfitting. There is typically some trade-off in model performance that needs to be explored between a large number of shallow trees and a smaller number of deeper trees.
- subsample controls sampling of the training data. This technique can help reduce overfitting, but setting it too low can also starve the model of data.
- num_round controls the number of boosting rounds. This is essentially the subsequent models that are trained using the residuals of previous iterations. Again, more rounds should produce a better fit on the training data, but can be computationally expensive or lead to overfitting.
- eta controls how aggressive each round of boosting is. Larger values lead to more conservative boosting.
- gamma controls how aggressively trees are grown. Larger values lead to more conservative models.

```
In [17]: xgb = sagemaker.estimator.Estimator(xgb_training_container,
                                             role,
                                             instance_count=1,
                                             instance_type='ml.m5.xlarge',
                                             output_path='s3://{}/{}/output'.format(bucket, prefix),
                                             sagemaker_session=sess)
         xgb.set_hyperparameters(max_depth=5,
                                 eta=0.2,
                                 gamma=4,
                                 min_child_weight=6,
                                 subsample=0.8,
                                 silent=0,
                                 objective='binary:logistic',
                                 num_round=100)

         xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

```
2022-09-18 10:28:07 Starting - Starting the training job...
2022-09-18 10:28:31 Starting - Preparing the instances for trainingProfilerReport-1663496887: InProgress
......
2022-09-18 10:29:31 Downloading - Downloading input data...
2022-09-18 10:29:51 Training - Downloading the training image...
2022-09-18 10:30:31 Training - Training image download completed. Training in progress..INFO:sagemaker-containers:Imp
orted framework sagemaker_xgboost_container.training
INFO:sagemaker-containers:Failed to parse hyperparameter objective value binary:logistic to Json.
Returning the value itself
INFO:sagemaker-containers:No GPUs detected (normal if no gpus installed)
INFO:sagemaker_xgboost_container.training:Running XGBoost Sagemaker in algorithm mode
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
[10:30:35] 2333x69 matrix with 160977 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimite
r=,
INFO:root:Determined delimiter of CSV input is ','
[10:30:35] 666x69 matrix with 45954 entries loaded from /opt/ml/input/data/validation?format=csv&label_column=0&delim
iter=,
```

### 5.2.13 Batch Inference

Next we're going to evaluate our model by using a Batch Transform to generate churn scores in batch from our model_data.

First, we upload the model data to S3. SageMaker Batch Transform is designed to run asynchronously and ingest input data from S3. This differs from SageMaker's real-time inference endpoints, which receive input data from synchronous HTTP requests.

For large scale deployments the data set will be retrieved from Snwoflake using SQL and an External Stage to S3.

- Batch Transform is often the ideal option for advanced analytics use case for serveral reasons:
- Batch Transform is better optimized for throughput in comparison with real-time inference endpoints. Thus, Batch Transform is ideal for processing large volumes of data for analytics.
- Offline asynchronous processing is acceptable for most analytics use cases.
- Batch Transform is more cost efficient when real-time inference isn't necessary. You only need to pay for resources used during batch processing. There is no need to pay for ongoing resources like a hosted endpoint for real-time inference.

5.2.14 The model is now ready to be used for Batch Inference. First we upload data we plan to use as batch input to S3.

```
In [16]: batch_input = churn.iloc[:,:-1]
         batch_input.to_csv('batch.csv', header=False, index=False)
         boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'batch/in/batch.csv')

         s3uri_batch_input ='s3://{}/{}/batch/in'.format(bucket, prefix)
         print('Batch Transform input S3 uri: {}'.format(s3uri_batch_input))

         s3uri_batch_output= 's3://{}/{}/batch/out'.format(bucket, prefix)
         print('Batch Transform output S3 uri: {}'.format(s3uri_batch_output))
```

Then we run the batch transform job. Do note that the batch transform jobs run asynchronously and non blocking by default. Run the transformer wait command in the next code block "**transformer.wait()**" to notify us when the batch job is complete. This operation can take a few minutes to complete.

```
In [20]: from sagemaker.transformer import Transformer
         BATCH_INSTANCE_TYPE = 'ml.c5.xlarge'

         transformer = compiled_model.transformer(instance_count=1,
                                                  strategy='SingleRecord',
                                                  assemble_with='Line',
                                                  instance_type= BATCH_INSTANCE_TYPE,
                                                  accept = 'text/csv',
                                                  output_path=s3uri_batch_output)

         transformer.transform(s3uri_batch_input,
                               split_type= 'Line',
                               content_type= 'text/csv',
                               input_filter = "$[1:]",
                               join_source = "Input",
                               output_filter = "$[0,-1,-2]")
         .......................[2022-09-18 10:45:05 +0000] [14] [INFO] Starting gunicorn 19.10.0
```

When the batch job is complete there will be an output block below the code block and a number in the square brackets next to the code block.

```
In [18]: transformer.wait()
```

```
d response time: 223
2020-09-22 03:25:37,838 [INFO ] W-9003-model ACCESS_LOG - /169.254.255.130:57602 "POST /inv
ocations HTTP/1.1" 200 223
2020-09-22 03:25:38,062 [INFO ] W-9000-model-stdout com.amazonaws.ml.mms.wlm.WorkerLifeCycl
e - Prediction counts: Counter({'True.': 1})
2020-09-22 03:25:38,063 [INFO ] W-9000-model-stdout com.amazonaws.ml.mms.wlm.WorkerLifeCycl
e - Elapsed time: 0.223 seconds
2020-09-22 03:25:38,063 [INFO ] W-9000-model com.amazonaws.ml.mms.wlm.WorkerThread - Backen
d response time: 223
2020-09-22 03:25:38,063 [INFO ] W-9000-model ACCESS_LOG - /169.254.255.130:57602 "POST /inv
ocations HTTP/1.1" 200 223
2020-09-22 03:25:38,289 [INFO ] W-9001-model-stdout com.amazonaws.ml.mms.wlm.WorkerLifeCycl
e - Prediction counts: Counter({'False.': 1})
2020-09-22 03:25:38,289 [INFO ] W-9001-model-stdout com.amazonaws.ml.mms.wlm.WorkerLifeCycl
e - Elapsed time: 0.224 seconds
2020-09-22 03:25:38,289 [INFO ] W-9001-model com.amazonaws.ml.mms.wlm.WorkerThread - Backen
d response time: 224
2020-09-22 03:25:38,289 [INFO ] W-9001-model ACCESS_LOG - /169.254.255.130:57602 "POST /inv
ocations HTTP/1.1" 200 224
```

5.2.15 As the final step of the ML workflow we will evaluate the model's performance by comparing actual to predicted values by running the last code block.

```
In [22]: batched_churn_scores = pd.read_csv(s3uri_batch_output+'/model.csv.out', usecols=[0,1], names=['id','scores'])
gt_df = pd.DataFrame(model_data['Churn?_True.']).reset_index(drop=True)
results_df= pd.concat([gt_df,batched_churn_scores],axis=1)

pd.crosstab(index=results_df['Churn?_True.'], columns=np.round(results_df['scores']), rownames=['actual'], colnames=['p
```

Out[22]:

| predictions | 0.0 | 1.0 |
|---|---|---|
| actual | | |
| 0 | 2827 | 23 |
| 1 | 96 | 387 |

# Module 6: Upload the Churn Scores to Snowflake

To allow multiple business users to access the output of the SageMaker Batch Inference job we will upload the churn scores to Snowflake.

6.1.1 We will load the Snowflake connector pandas tools and then write the dataframe to the ML_RESULTS table in Snowflake directly.

**Do note that we renamed the dataframe column names to match the SNowflake table and they are also in all CAPS.**

In [23]:
```
from snowflake.connector.pandas_tools import write_pandas

#Set the column names of the dataframe to match the table column names
results_df.columns = ['CHURN_IN','CUST_ID','CHURN_SCORE']

# Write the predictions to the table named "ML_RESULTS".
success, nchunks, nrows, _ = write_pandas(ctx, results_df, 'ML_RESULTS')

display(nrows)
```

```
3333
```

In [24]:
```
results_df.to_csv('results.csv', header=False, index=False)
```

In [25]:
```
cs.execute('PUT file://results.csv @ml_results')
```

Out[25]:
```
<snowflake.connector.cursor.SnowflakeCursor at 0x7f7d2c1e05f8>
```

6.1.2 We can now switch back to our Snowflake Worksheet in the browser tab or window we were using earlier. Scroll down in the SQL till you see the comment lines listed below.

```
/*---------------------------------------------------------------------------
Get the ML Results returned from Sagemaker and perform some SQL Reporting
---------------------------------------------------------------------------*/
```

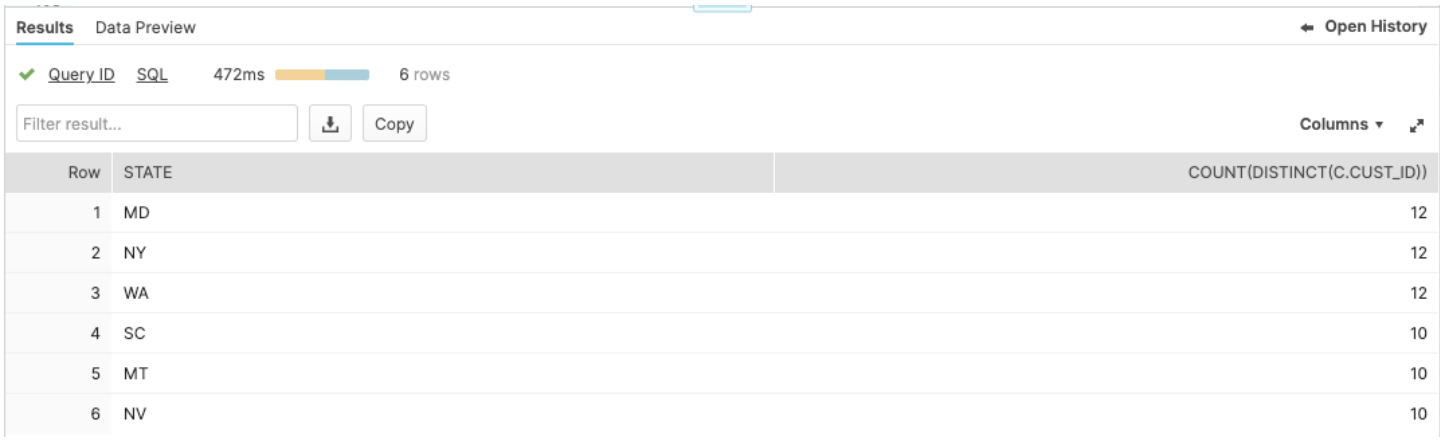
6.1.3 First we will look at the data in the ML_RESULTS table.

```
SELECT * FROM ML_RESULTS LIMIT 100;
```

6.1.4 Now we are able to join the Churn Scores output in the ML_RESULTS table from the SageMaker batch inference with the customer data.

Below is a simple analytical query that joins the ML_RESULTS table with the CUSTOMER_CHURN table that includes the customer information. This query identifies and ranks states that have 10 or more customers with a churn score above 0.75.

```
SELECT C.STATE, COUNT(DISTINCT(C.CUST_ID))
FROM ML_RESULTS M INNER JOIN  CUSTOMER_CHURN C on M.CUST_ID =
C.CUST_ID
WHERE M.CHURN_SCORE >= 0.75
GROUP BY C.STATE
HAVING COUNT(DISTINCT(C.CUST_ID)) >= 10
ORDER BY COUNT(C.CUST_ID) DESC;
```


Results Data Preview Open History

Query ID SQL 472ms 6 rows

Filter result... Copy Columns

| Row | STATE | COUNT(DISTINCT(C.CUST_ID)) |
|---|---|---|
| 1 | MD | 12 |
| 2 | NY | 12 |
| 3 | WA | 12 |
| 4 | SC | 10 |
| 5 | MT | 10 |
| 6 | NV | 10 |

# Summary & Next Steps

This lab was designed as a hands-on introduction to Snowflake and SageMaker to simultaneously teach you how to use it, while showcasing some of its key capabilities.

We encourage you to continue with your free Snowflake trial by loading in your own sample or production data and by using some of the more advanced capabilities of Snowflake not covered in this lab. There are several ways Snowflake can help you with this:

- At the very top of the UI click on the “Partner Connect” icon to get access to trial/free ETL and BI tools to help you get more data into Snowflake and then analyze it
- Read the “Definitive Guide to Maximizing Your Free Trial” document at: https://www.snowflake.com/test-driving-snowflake-the-definitive-guide-to-maximizing-your-free-trial/
- Attend a Snowflake virtual or in-person event to learn more about our capabilities and how customers use us https://www.snowflake.com/about/events/
- Contact Sales to learn more https://www.snowflake.com/free-trial-contact-sales/

We also encourage you to continue to explore the capabilities of Amazon SageMaker. For other SageMaker Machine Learning examples visit:
https://docs.aws.amazon.com/sagemaker/latest/dg/howitworks-nbexamples.html

# Resetting Your Snowflake Environment

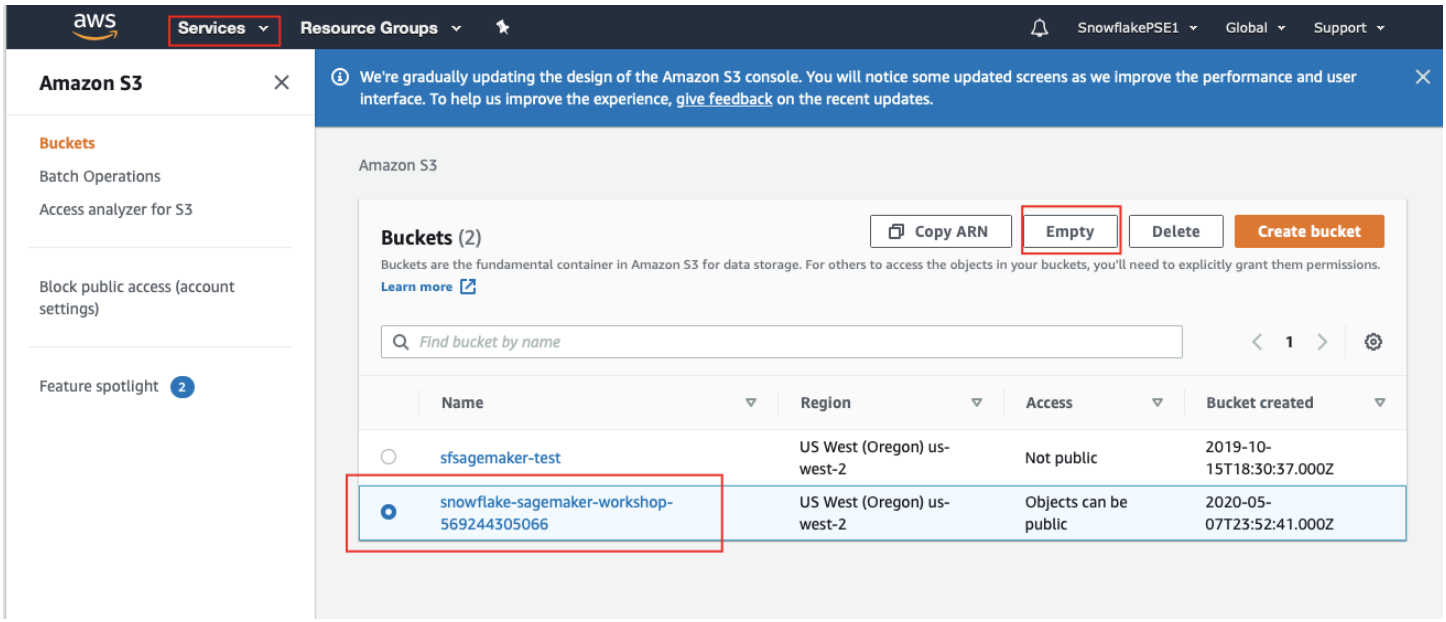
You can reset your Snowflake environment by running the SQL commands in the Worksheet:

```
USE ROLE ACCOUNTADMIN;
DROP WAREHOUSE IF EXISTS SAGEMAKER_WH;
DROP DATABASE IF EXISTS ML_WORKSHOP;
DROP ROLE IF EXISTS SAGEMAKER_ROLE;
DROP USER IF EXISTS SAGEMAKER;
```

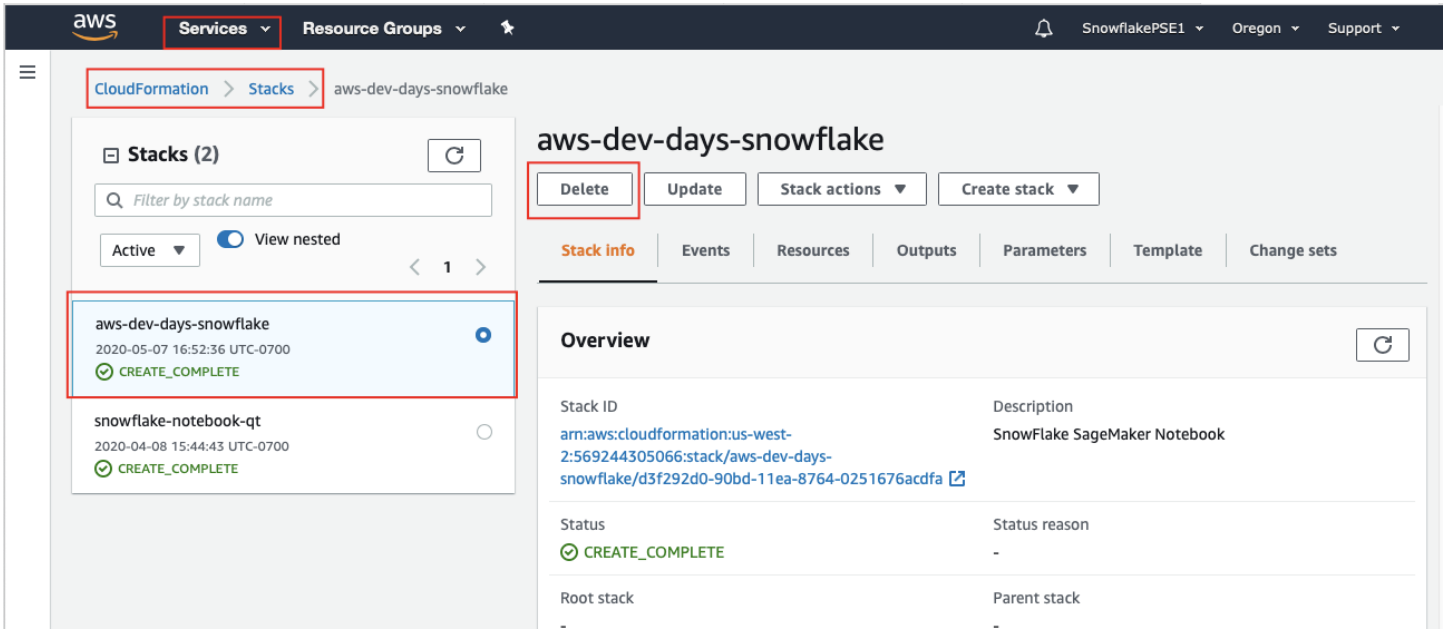
# Resetting Your AWS & SageMaker Environment

To avoid incurring charge for the AWS Services that were deployed for the lab you will need to remove the services following these steps:

- First empty the S3 bucket that was created for the lab
  - Go to the AWS Console and select the CloudFormation Service under Services.
  - Select the S3 bucket and click on Empty
  - Follow the steps to Empty the bucket




- Finally delete the AWS CloudFormation Stack that was created in Module 4
  - Go to the AWS Console and select the CloudFormation Service under Services.
  - Select the Stack name that was created for this lab
  - Click the Delete button (see picture below)
  - Follow the steps to complete the Stack deletion


aws
Services
Resource Groups
SnowflakePSE1
Oregon
Support
CloudFormation
Stacks
aws-dev-days-snowflake
Stacks (2)
Filter by stack name
Active
View nested
1
aws-dev-days-snowflake
2020-05-07 16:52:36 UTC-0700
CREATE_COMPLETE
snowflake-notebook-qt
2020-04-08 15:44:43 UTC-0700
CREATE_COMPLETE
aws-dev-days-snowflake
Delete
Update
Stack actions
Create stack
Stack info
Events
Resources
Outputs
Parameters
Template
Change sets
Overview
Stack ID
arn:aws:cloudformation:us-west-2:569244305066:stack/aws-dev-days-snowflake/d3f292d0-90bd-11ea-8764-0251676acdfa
Description
SnowFlake SageMaker Notebook
Status
CREATE_COMPLETE
Status reason
-
Root stack
-
Parent stack
-