**ML System Operation Validation**

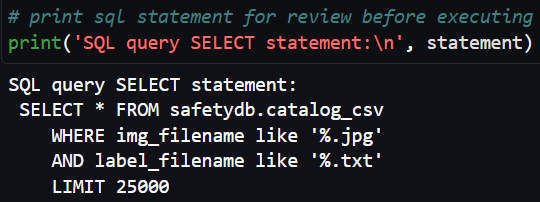
**Project: Job Site Safety**

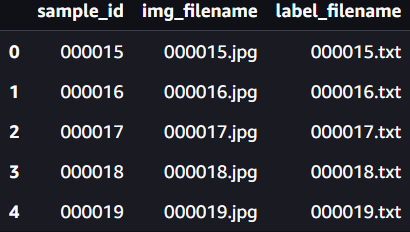
**Group Members:** [**Jason Raimondi**](mailto:jraimondi@sandiego.edu)**,** [Jeremy Cryer](mailto:jcryer@sandiego.edu)**,** [Maimuna Bashir](mailto:mbashir@sandiego.edu)

**Catalog, Feature Groups and Feature Store**

**Athena Database Catalog - Query**

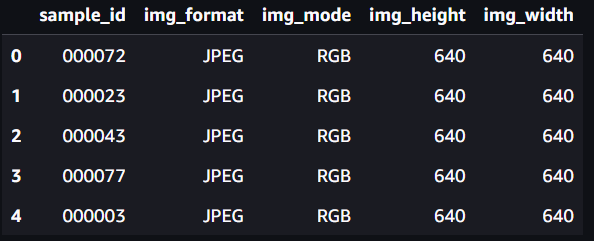
Because our computer vision dataset (i.e., images/labels) is not tabular in raw form, we decided to create a catalog in csv format that can be queried initially. This example shows querying for images in JPG format and labels in TXT format.

****

****

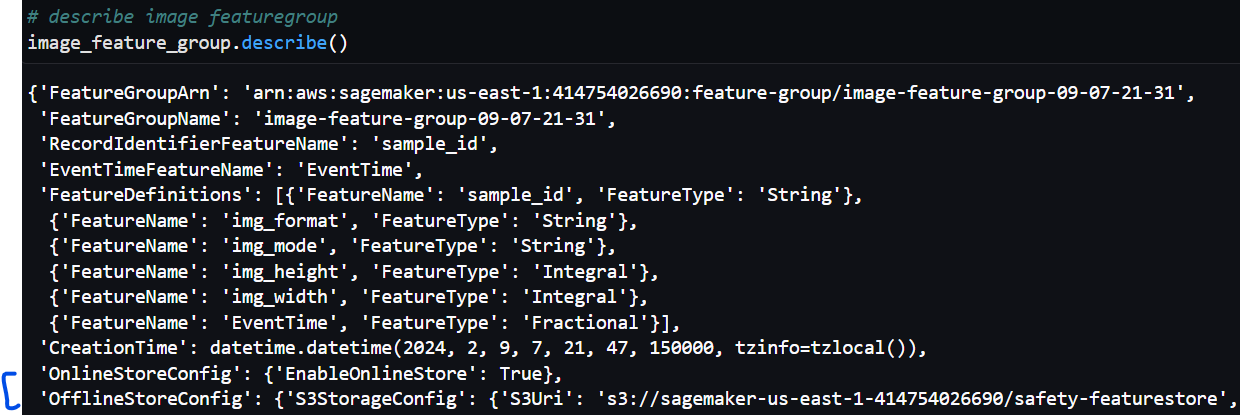
**Image Feature Group**

The image feature group contains features that were obtained during preprocessing of the raw image data.

****

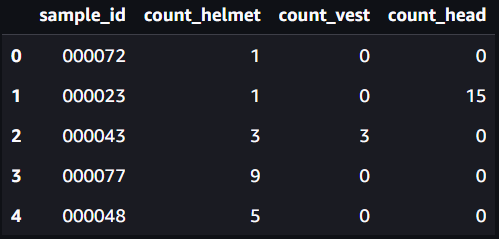
**Image Feature Store (Online and Offline)**

Describing the image feature group shows the presence of the online and offline stores.

****

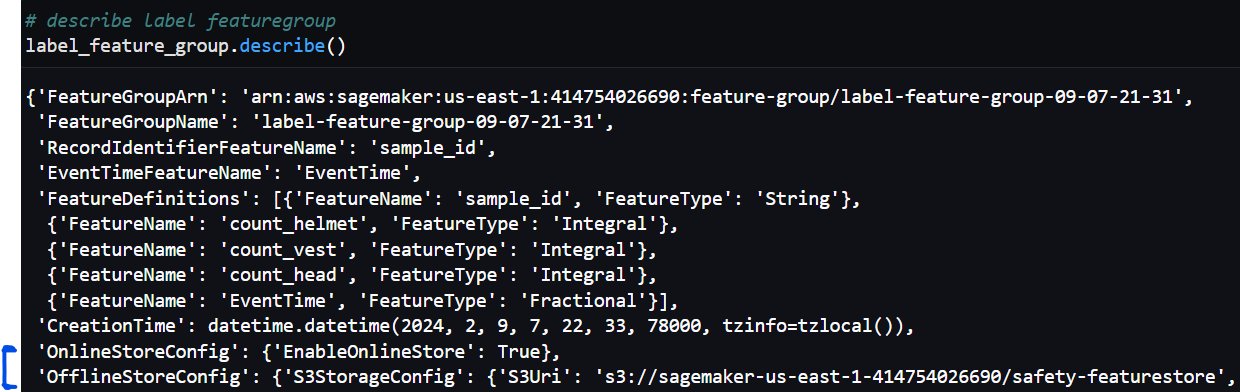
**Label Feature Group**

The label feature group contains features that were obtained during preprocessing of the raw label data (See column titles for exact features).



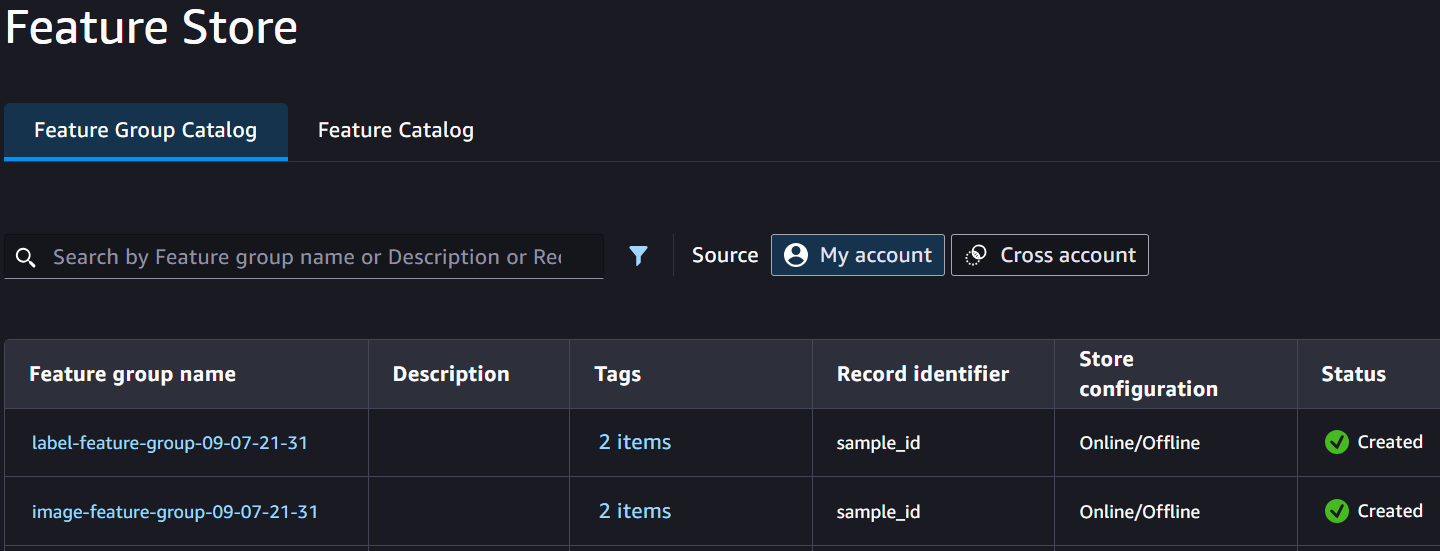
**Label Feature Store (Online and Offline)**

Describing the label feature group shows the presence of the online and offline stores.



**Feature Store (Online/Offline) viewed in Sagemaker**

In SageMaker, we can also verify the creation of the feature groups, as well as the online and offline store configurations.



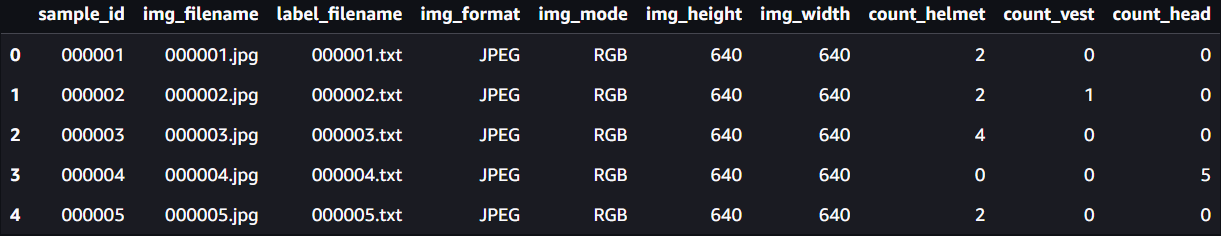
**Feature Catalog**

In SageMaker, the Feature Catalog shows all of the features within each feature group, as well as the data type.



**Joined Features by Primary Key (sample\_id)**

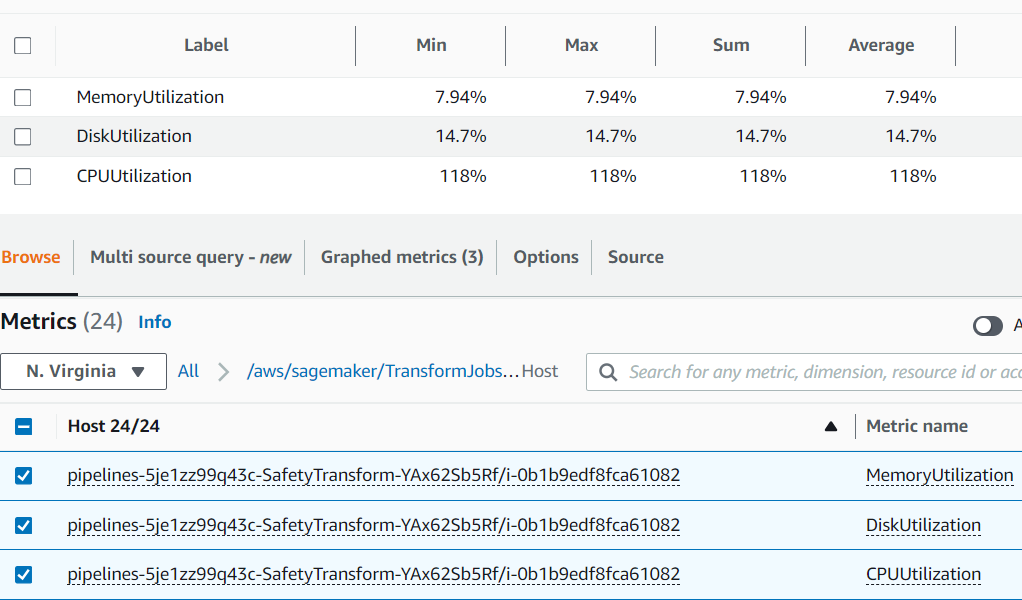
Using ‘sample\_id’ as the Primary Key, we are able to join the catalog data with the features from both the image and label feature groups into a single table. As desired, this tabular format can then be used for additional querying/filtering to select the desired samples to be used for the training dataset.



**Infrastructure Monitoring Dashboards**

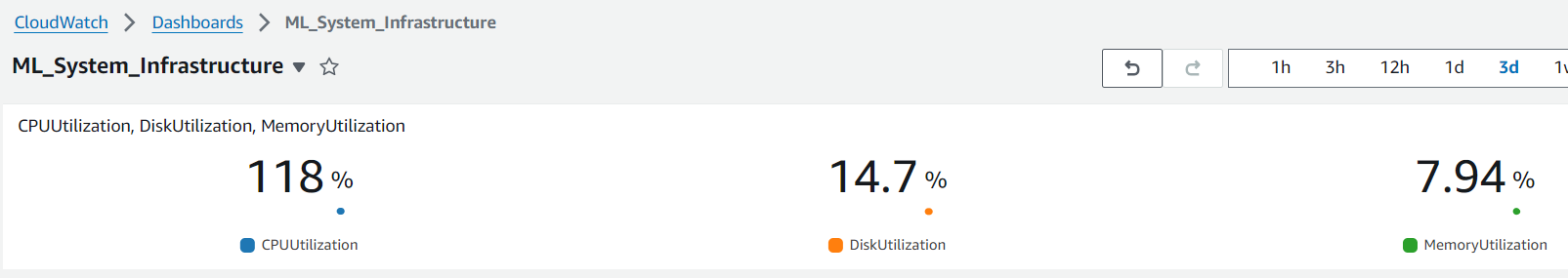
**Metrics**

We use a batch transform job in our overall pipeline. Here we can view infrastructure metrics related to resource utilization.

****

**Infrastructure Dashboard**

These infrastructure metrics were added to a CloudWatch Dashboard for easier viewing. In this example, we can see CPU Utilization is over 100%. This is a good metric to alert on, and for future work, we would be interested to incorporate auto-scaling so that we can scale up or out to avoid issues and improve performance.

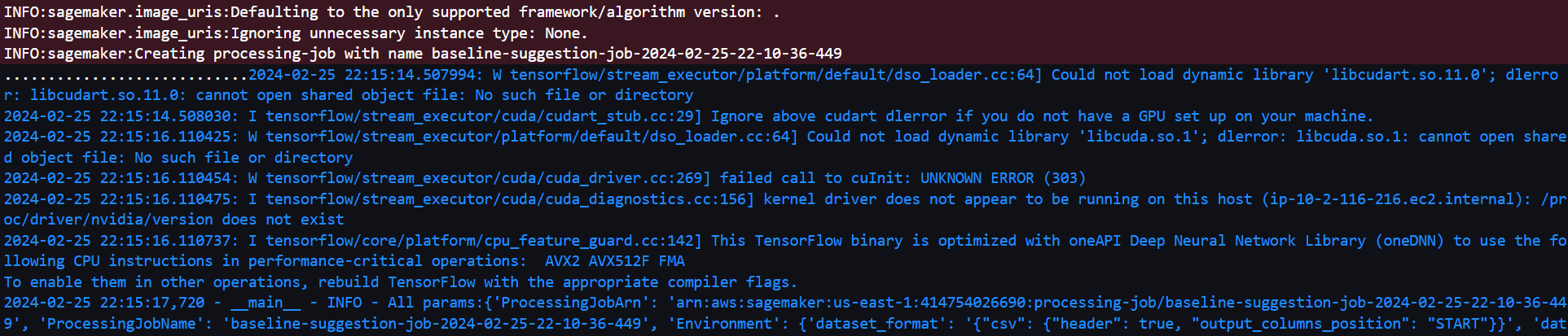
****

**Model/Data Monitoring Reports**

Due to additional complexity with our Computer Vision use case and using Script Mode with container customizations, we opted to focus on the infrastructure metrics and monitoring and save model and data metrics as future work. One of the challenges presented is that the default model monitor expects a dataset in the format of either CSV, JSON, or parquet. However, we instead have images. Considering this, we still opted to go through the steps of creating baseline results and a monitoring schedule. We did this by using our catalog data as input, which is in CSV format. However, we would desire to plan for additional engineering efforts as future work to construct this monitor in a more robust way. The catalog data on its own does not have much useful information for a real-world use case. We would want to instead consider potentially using our feature store data, exporting it to a compatible format, and using this format as input for our batch job instead of directly using our image format. Alternatively, we could research additional monitoring methods that may be able to handle image data.

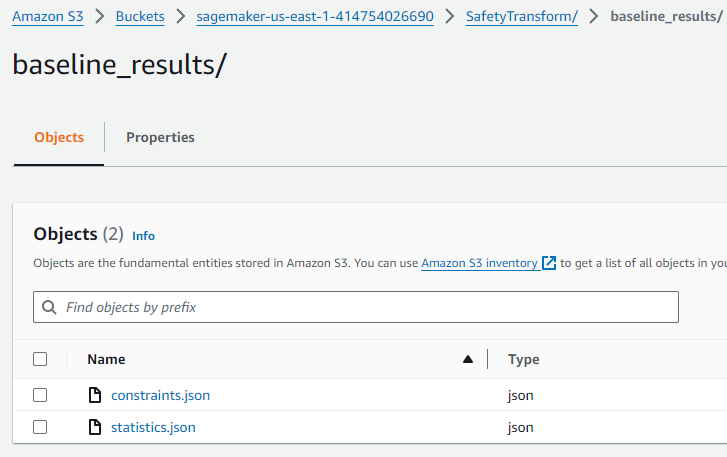
**Baseline Suggestion Job**

This shows the creation of a baseline suggestion job for the data quality monitor.

****

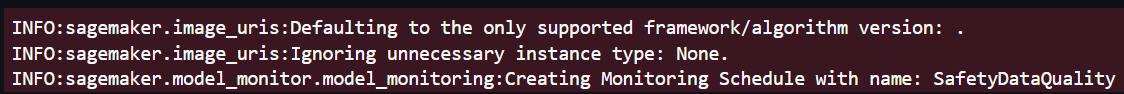
**Baseline Results**

This shows the results of the baseline suggestion job. We can see that the constraints.json and statistics.json files have been generated.

****

**Data Quality Monitoring Schedule**

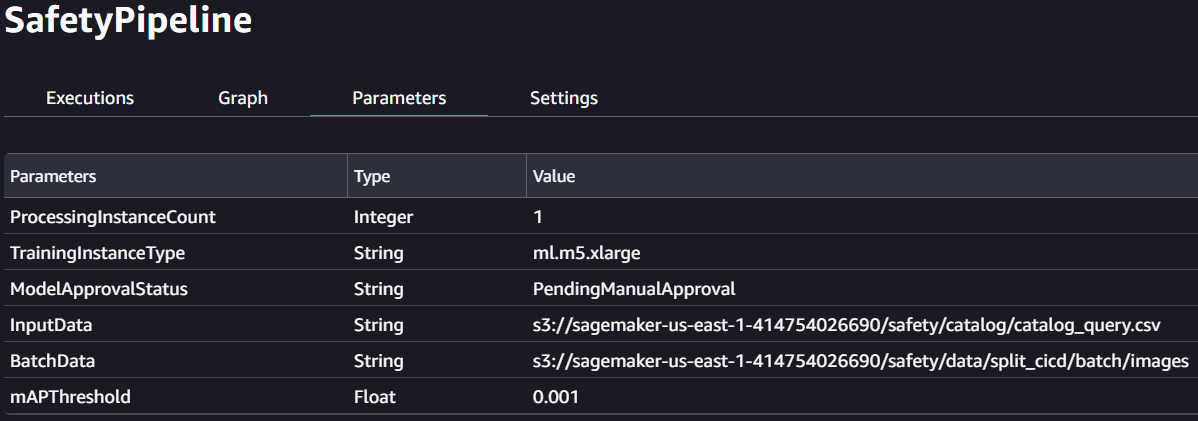
This shows the creation of the data quality monitoring schedule.

****

**CI/CD DAG in a Successful and Failed State**

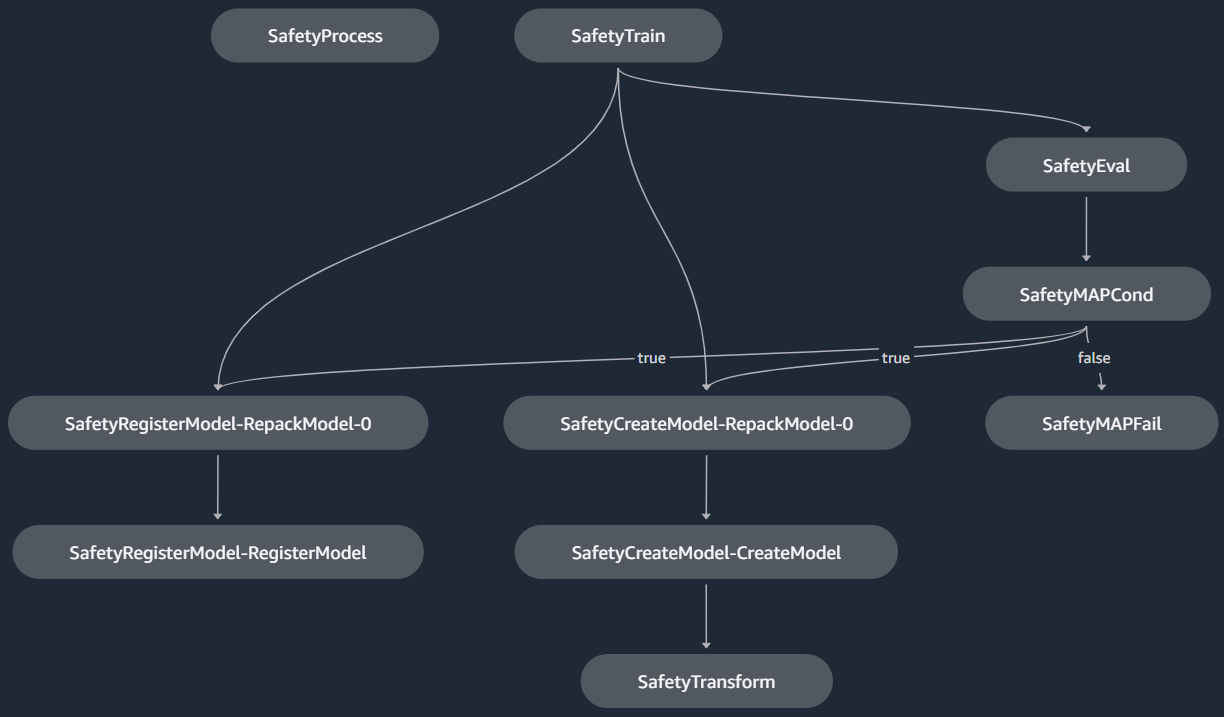
**Pipeline Parameters**

Before getting to the DAG, here are the parameters for our pipeline. We can see the instance information, model approval status, input and batch data locations, as well as our threshold metric to be used during the evaluation step. We chose to use mean Average Precision (mAP) as our metric for the Computer Vision problem.

****

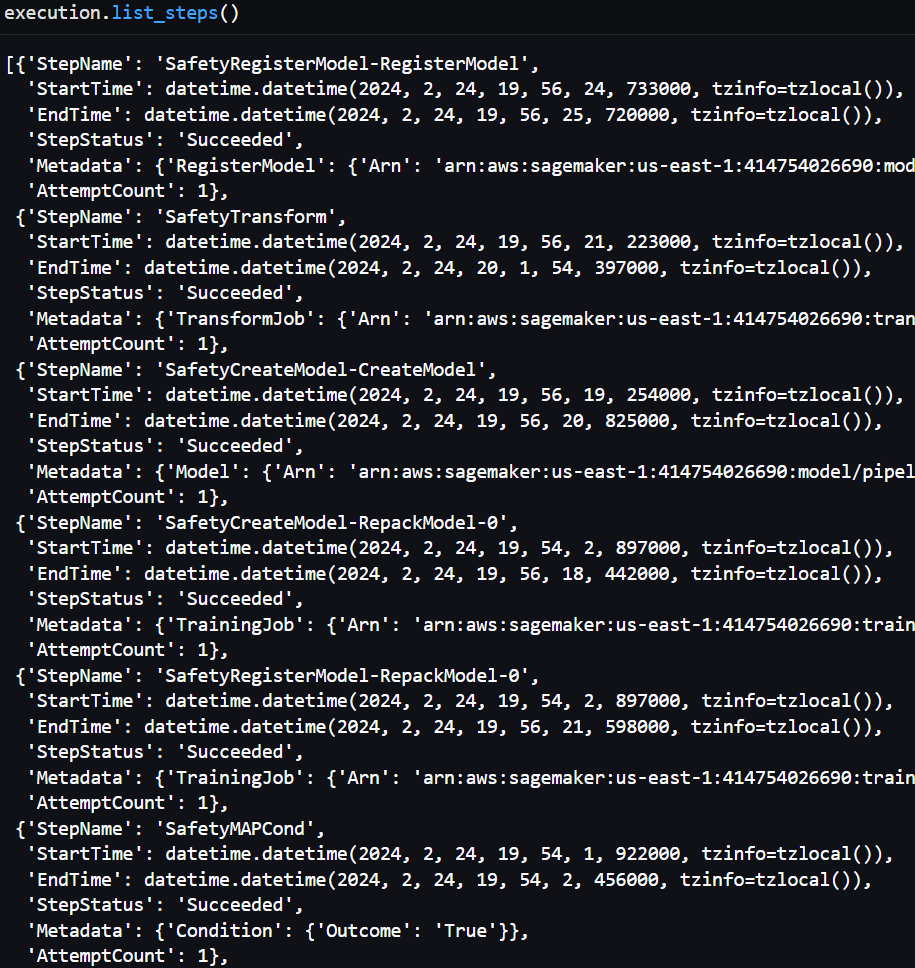
**Pipeline Directed Acyclic Graph (DAG)**

Note that the ‘SafetyProcess’ step is disconnected. For our problem, this is because we chose to not produce output from this step. This step, instead, processes and copies our image/label data to an AWS S3 location that the ‘SafetyTrain’ step is already familiar with.

****

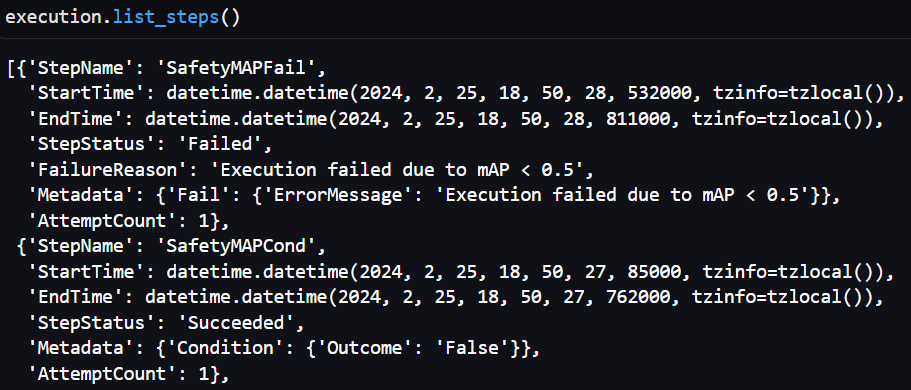
**Successful State**

This shows the ‘SafetyMAPCond’ step in a successful state with a ‘True’ outcome. This means that our model evaluation metric (i.e., mAP) exceeded our defined metric threshold value. After the successful condition step, we can see that the model is then successfully created and registered, and our transform job also runs successfully.

****

**Failed State**

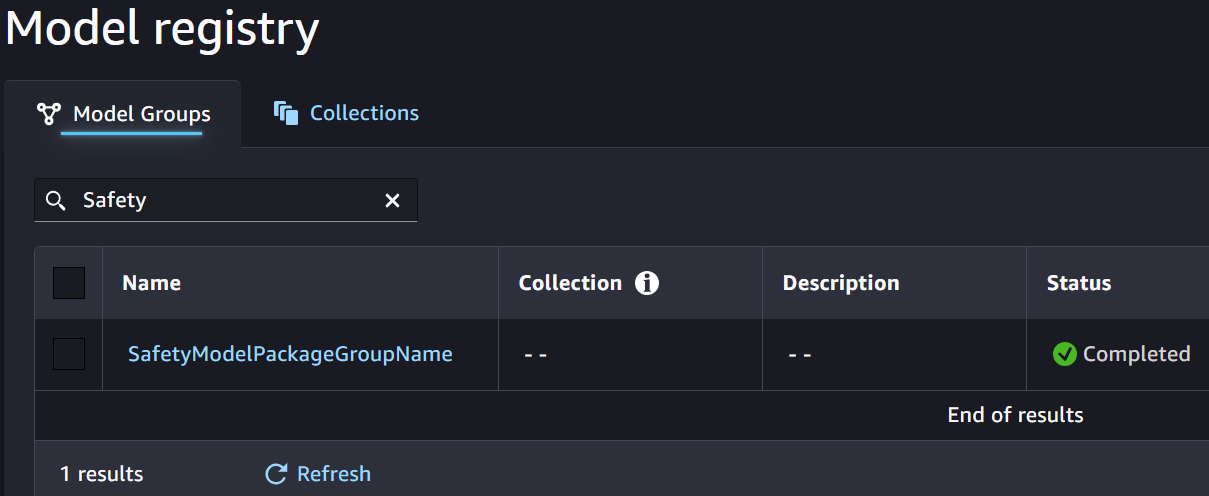
This shows the ‘SafetyMAPCond’ step in a successful state with a ‘False’ outcome. This means that our model evaluation metric (i.e., mAP) was less than our defined metric threshold value. After the condition step, we can see that the ‘SafetyMAPFail’ step runs and results in a ‘Failed’ step status. The metadata shows that the ‘Execution failed due to mAP < 0.5’.

****

**Model Registry**

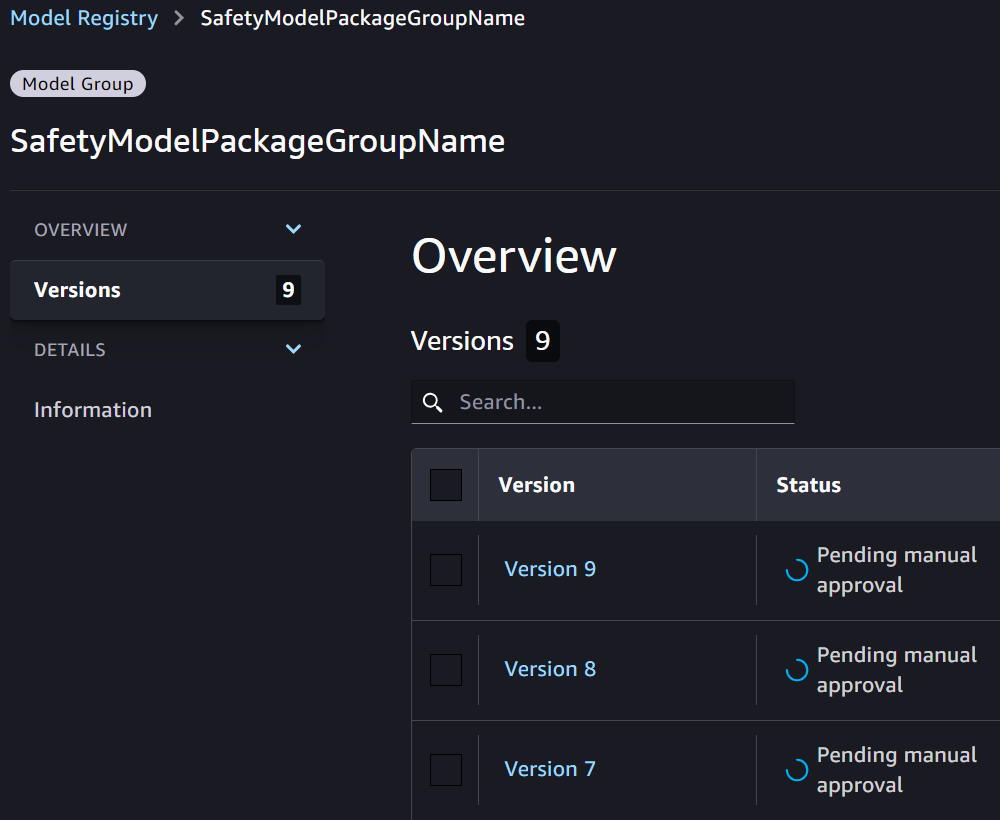
**Model Group**

In SageMaker, we can view our Model Group Package within the model registry.

****

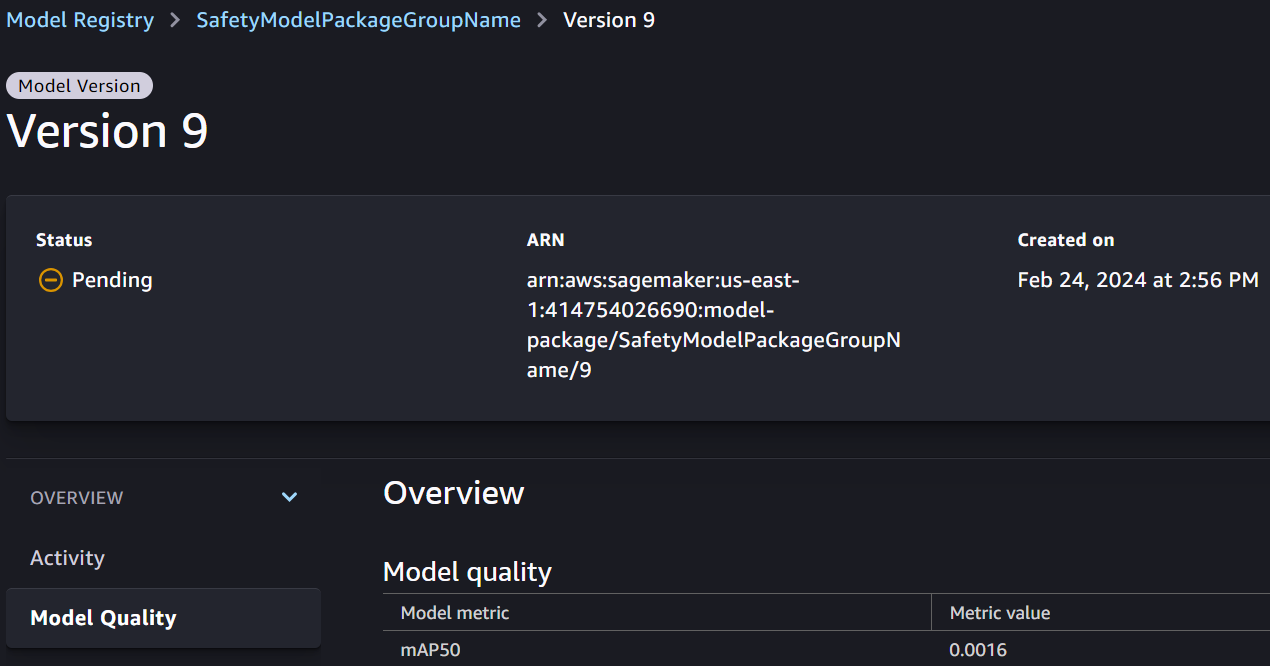
**Model Group Versions**

Within the model package group, we can view all of the different versions of the model we have created. In this example, we can see that we can choose to take manual action to approve a particular model version for production use.

****

**Model Version**

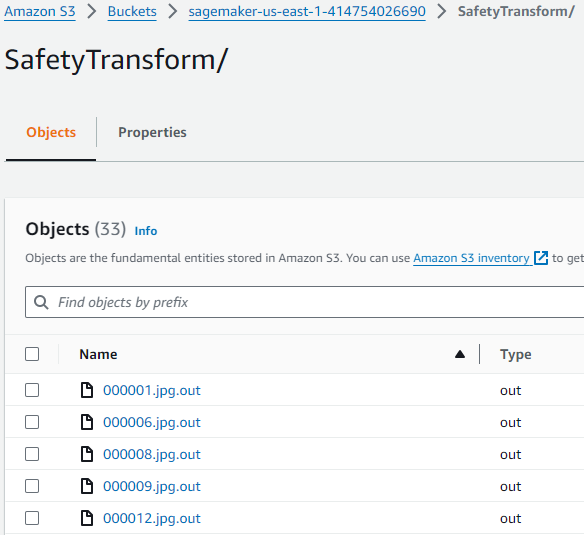
This shows details of one specific model version. Again, we can see that the approval status is pending. We can also see the value of our metric (i.e., mAP50) under the model quality section.

****

**Outputs of Batch Inference Job**

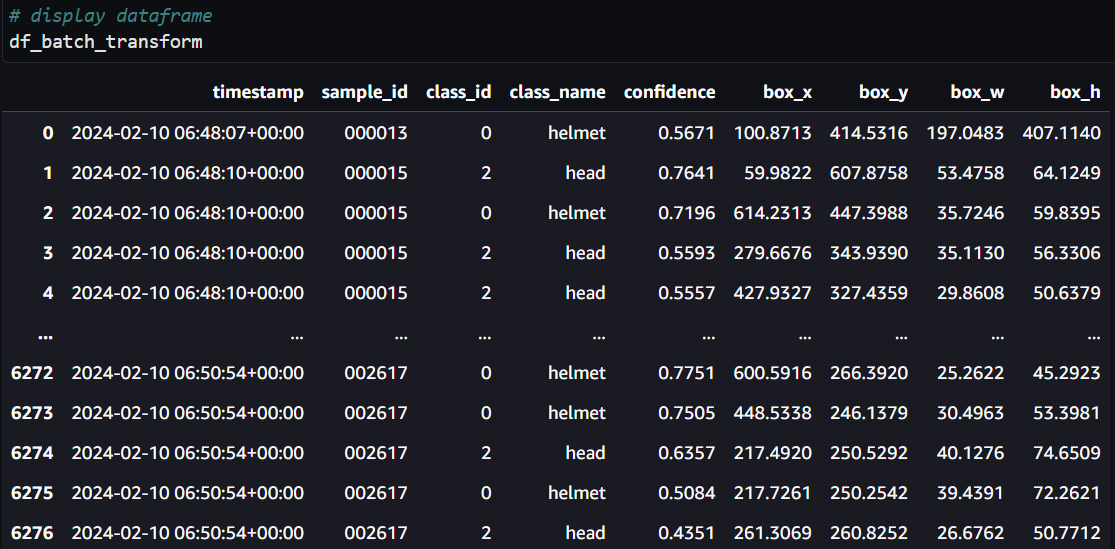
**Batch Inference S3 Output Location**

In Amazon S3, we can see the output of our batch transform job. In our case, each image processed results in a separate output file with the same name of the image and a ‘.out’ file extension appended to it.

****

**Batch Transform Combined Results**

As mentioned, our batch transform job results in individual ‘.out’ files in Amazon S3. We perform post processing to retrieve all of the individual results files and combine the results into a single dataframe. We also add a timestamp, as this is important for making the results useful and being able to know when the results were generated for each sample.

****

**Batch Transform Business Query**

Finally, we get back to our original business problem we intended to solve with this project. Our intention was to perform batch transform jobs and be able to obtain the classes (helmet and head) to help provide insights on trends regarding the usage of personal protective equipment (PPE) on the job site. In this example, we can see that we have 2,231 detections of ‘helmet’ and 3,998 detections of ‘head’. This is only for a single day, but the large number of ‘head’ detections may indicate PPE compliance issues for this particular day. Ultimately, similar transform jobs should take place on a regular basis so that this data can show trends. An upward trend in helmet detections, coupled with a downward trend in head detections, would indicate increased safety compliance.

****