Computing numerical and categorical statistics

You can use Cloud Data Loss Prevention (DLP) to compute numerical and categorical numerical statistics for individual columns in BigQuery tables. Cloud DLP can calculate the following:

- The column's minimum value
- The column's maximum value
- Quantile values (https://en.wikipedia.org/wiki/Quantile) for the column
- A histogram of value frequencies in the column

Compute numerical statistics

You can determine minimum, maximum, and quantile values for an individual BigQuery column. To calculate these values, you configure a DlpJob (/dlp/docs/reference/rest/v2/projects.dlpJobs#resource-dlpjob), setting the `NumericalStatsConfig` (/dlp/docs/reference/rest/v2/projects.dlpJobs#numericalstatsconfig) privacy metric to the name of the column to scan. When you run the job (/dlp/docs/reference/rest/v2/projects.dlpJobs#DlpJob), Cloud DLP computes statistics for the given column, returning its results in the `NumericalStatsResult` (/dlp/docs/reference/rest/v2/projects.dlpJobs#numericalstatsresult) object. Cloud DLP can compute statistics for the following number types:

- integer
- float
- date
- datetime
- timestamp
- time

The statistics that a scan run returns include the minimum value, the maximum value, and 99 quantile values (https://en.wikipedia.org/wiki/Quantile) that partition the set of field values into 100

equal sized buckets.

Code examples

Following is sample code in several languages that demonstrates how to use Cloud DLP to calculate numerical statistics.

ant: The code on this page requires that you first set up a Cloud DLP client. For more information about instal eating a Cloud DLP client, see Cloud DLP client libraries (/dlp/docs/reference/libraries). (Sending JSON to Clou EST endpoints does not require a client library.)

JavaNode.js (#node.js)Python (#python)Go (#go)PHP (#php)C# (#c)

pis/java-dlp/blob/master/samples/snippets/src/main/java/dlp/snippets/RiskAnalysisNumericalStats.java)

```
import com.google.api.core.SettableApiFuture;
import com.google.cloud.dlp.v2.DlpServiceClient;
import com.google.cloud.pubsub.v1.AckReplyConsumer;
import com.google.cloud.pubsub.v1.MessageReceiver;
import com.google.cloud.pubsub.v1.Subscriber;
import com.google.privacy.dlp.v2.Action;
import com.google.privacy.dlp.v2.Action.PublishToPubSub;
import com.google.privacy.dlp.v2.AnalyzeDataSourceRiskDetails.NumericalStatsResult;
import com.google.privacy.dlp.v2.BigQueryTable;
import com.google.privacy.dlp.v2.CreateDlpJobRequest;
import com.google.privacy.dlp.v2.DlpJob;
import com.google.privacy.dlp.v2.FieldId;
import com.google.privacy.dlp.v2.GetDlpJobRequest;
import com.google.privacy.dlp.v2.LocationName;
import com.google.privacy.dlp.v2.PrivacyMetric;
import com.google.privacy.dlp.v2.PrivacyMetric.NumericalStatsConfig;
import com.google.privacy.dlp.v2.RiskAnalysisJobConfig;
import com.google.privacy.dlp.v2.Value;
import com.google.pubsub.v1.ProjectSubscriptionName;
import com.google.pubsub.v1.ProjectTopicName;
import com.google.pubsub.v1.PubsubMessage;
import java.io.IOException;
import java.util.concurrent.ExecutionException;
```

```
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

class RiskAnalysisNumericalStats {

  public static void main(String[] args) throws Exception {
    // TODO(developer): Replace these variables before running the sample.
    String projectId = "your-project-id";
    String datasetId = "your-bigquery-dataset-id";
    String tableId = "your-bigquery-table-id";
    String topicId = "pub-sub-topic";
    String subscriptionId = "pub-sub-subscription";
    numericalStatsAnalysis(projectId, datasetId, tableId, topicId, subscriptionId);
  }

  public static void numericalStatsAnalysis(
      String projectId, String datasetId, String tableId, String topicId, String sub
      throws ExecutionException, InterruptedException, IOException {

    // Initialize client that will be used to send requests. This client only needs
    // once, and can be reused for multiple requests. After completing all of your r
    // the "close" method on the client to safely clean up any remaining background
    try (DlpServiceClient dlpServiceClient = DlpServiceClient.create()) {

      // Specify the BigQuery table to analyze
      BigQueryTable bigQueryTable =
          BigQueryTable.newBuilder()
              .setTableId(tableId)
              .setDatasetId(datasetId)
              .setProjectId(projectId)
              .build();

      // This represents the name of the column to analyze, which must contain numer
      String columnName = "Age";

      // Configure the privacy metric for the job
      FieldId fieldId = FieldId.newBuilder().setName(columnName).build();
      NumericalStatsConfig numericalStatsConfig =
          NumericalStatsConfig.newBuilder().setField(fieldId).build();
      PrivacyMetric privacyMetric =
          PrivacyMetric.newBuilder().setNumericalStatsConfig(numericalStatsConfig).b

      // Create action to publish job status notifications over Google Cloud Pub/Sub
      ProjectTopicName topicName = ProjectTopicName.of(projectId, topicId);
      PublishToPubSub publishToPubSub =
```

```
        PublishToPubSub.newBuilder().setTopic(topicName.toString()).build();
    Action action = Action.newBuilder().setPubSub(publishToPubSub).build();

    // Configure the risk analysis job to perform
    RiskAnalysisJobConfig riskAnalysisJobConfig =
        RiskAnalysisJobConfig.newBuilder()
            .setSourceTable(bigQueryTable)
            .setPrivacyMetric(privacyMetric)
            .addActions(action)
            .build();

    CreateDlpJobRequest createDlpJobRequest =
        CreateDlpJobRequest.newBuilder()
            .setParent(LocationName.of(projectId, "global").toString())
            .setRiskJob(riskAnalysisJobConfig)
            .build();

    // Send the request to the API using the client
    DlpJob dlpJob = dlpServiceClient.createDlpJob(createDlpJobRequest);

    // Set up a Pub/Sub subscriber to listen on the job completion status
    final SettableApiFuture<Boolean> done = SettableApiFuture.create();

    ProjectSubscriptionName subscriptionName =
        ProjectSubscriptionName.of(projectId, subscriptionId);

    MessageReceiver messageHandler =
        (PubsubMessage pubsubMessage, AckReplyConsumer ackReplyConsumer) -> {
          handleMessage(dlpJob, done, pubsubMessage, ackReplyConsumer);
        };
    Subscriber subscriber = Subscriber.newBuilder(subscriptionName, messageHandler
    subscriber.startAsync();

    // Wait for job completion semi-synchronously
    // For long jobs, consider using a truly asynchronous execution model such as
    try {
      done.get(15, TimeUnit.MINUTES);
    } catch (TimeoutException e) {
      System.out.println("Job was not completed after 15 minutes.");
      return;
    } finally {
      subscriber.stopAsync();
      subscriber.awaitTerminated();
    }
```

```
      // Build a request to get the completed job
      GetDlpJobRequest getDlpJobRequest =
          GetDlpJobRequest.newBuilder().setName(dlpJob.getName()).build();

      // Retrieve completed job status
      DlpJob completedJob = dlpServiceClient.getDlpJob(getDlpJobRequest);
      System.out.println("Job status: " + completedJob.getState());

      // Get the result and parse through and process the information
      NumericalStatsResult result = completedJob.getRiskDetails().getNumericalStatsR

      System.out.printf(
          "Value range : [%.3f, %.3f]\n",
          result.getMinValue().getFloatValue(), result.getMaxValue().getFloatValue()

      int percent = 1;
      Double lastValue = null;
      for (Value quantileValue : result.getQuantileValuesList()) {
        Double currentValue = quantileValue.getFloatValue();
        if (lastValue == null || !lastValue.equals(currentValue)) {
          System.out.printf("Value at %s %% quantile : %.3f", percent, currentValue)
        }
        lastValue = currentValue;
      }
    }
  }

  // handleMessage injects the job and settableFuture into the message reciever inte
  private static void handleMessage(
      DlpJob job,
      SettableApiFuture<Boolean> done,
      PubsubMessage pubsubMessage,
      AckReplyConsumer ackReplyConsumer) {
    String messageAttribute = pubsubMessage.getAttributesMap().get("DlpJobName");
    if (job.getName().equals(messageAttribute)) {
      done.set(true);
      ackReplyConsumer.ack();
    } else {
      ackReplyConsumer.nack();
    }
  }
}
```

Compute categorical numerical statistics

You can compute categorical numerical statistics for the individual histogram buckets within a BigQuery column, including:

- Upper bound on value frequency within a given bucket
- Lower bound on value frequency within a given bucket
- Size of a given bucket
- A sample of value frequencies within a given bucket (maximum 20)

To calculate these values, you configure a DlpJob (/dlp/docs/reference/rest/v2/projects.dlpJobs#resource-dlpjob), setting the `CategoricalStatsConfig` (/dlp/docs/reference/rest/v2/projects.dlpJobs#categoricalstatsconfig) privacy metric to the name of the column to scan. When you run the job (/dlp/docs/reference/rest/v2/projects.dlpJobs#DlpJob), Cloud DLP computes statistics for the given column, returning its results in the `CategoricalStatsResult` (/dlp/docs/reference/rest/v2/projects.dlpJobs#categoricalstatsresult) object.

You can apply the **`CategoricalStatsConfig`** to all column types, except for arrays and structs. If your colur consist solely of the types specified in "Compute numerical statistics (#compute-num-stats)," it may be more ative to scan for that metric.

Code examples

Following is sample code in several languages that demonstrates how to use Cloud DLP to calculate categorical statistics.

ant: The code on this page requires that you first set up a Cloud DLP client. For more information about instal eating a Cloud DLP client, see Cloud DLP client libraries (/dlp/docs/reference/libraries). (Sending JSON to Clou EST endpoints does not require a client library.)

JavaNode.js (#node.js)Python (#python)Go (#go)PHP (#php)C# (#c)

is/java-dlp/blob/master/samples/snippets/src/main/java/dlp/snippets/RiskAnalysisCategoricalStats.java)

```
import com.google.api.core.SettableApiFuture;
import com.google.cloud.dlp.v2.DlpServiceClient;
import com.google.cloud.pubsub.v1.AckReplyConsumer;
import com.google.cloud.pubsub.v1.MessageReceiver;
import com.google.cloud.pubsub.v1.Subscriber;
import com.google.privacy.dlp.v2.Action;
import com.google.privacy.dlp.v2.Action.PublishToPubSub;
import com.google.privacy.dlp.v2.AnalyzeDataSourceRiskDetails.CategoricalStatsResult
import com.google.privacy.dlp.v2.AnalyzeDataSourceRiskDetails.CategoricalStatsResult
import com.google.privacy.dlp.v2.BigQueryTable;
import com.google.privacy.dlp.v2.CreateDlpJobRequest;
import com.google.privacy.dlp.v2.DlpJob;
import com.google.privacy.dlp.v2.FieldId;
import com.google.privacy.dlp.v2.GetDlpJobRequest;
import com.google.privacy.dlp.v2.LocationName;
import com.google.privacy.dlp.v2.PrivacyMetric;
import com.google.privacy.dlp.v2.PrivacyMetric.CategoricalStatsConfig;
import com.google.privacy.dlp.v2.RiskAnalysisJobConfig;
import com.google.privacy.dlp.v2.ValueFrequency;
import com.google.pubsub.v1.ProjectSubscriptionName;
import com.google.pubsub.v1.ProjectTopicName;
import com.google.pubsub.v1.PubsubMessage;
import java.io.IOException;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

class RiskAnalysisCategoricalStats {

  public static void main(String[] args) throws Exception {
    // TODO(developer): Replace these variables before running the sample.
    String projectId = "your-project-id";
    String datasetId = "your-bigquery-dataset-id";
    String tableId = "your-bigquery-table-id";
    String topicId = "pub-sub-topic";
    String subscriptionId = "pub-sub-subscription";
    categoricalStatsAnalysis(projectId, datasetId, tableId, topicId, subscriptionId)
  }

  public static void categoricalStatsAnalysis(
      String projectId, String datasetId, String tableId, String topicId, String sub
      throws ExecutionException, InterruptedException, IOException {
```

```
// Initialize client that will be used to send requests. This client only needs
// once, and can be reused for multiple requests. After completing all of your r
// the "close" method on the client to safely clean up any remaining background
try (DlpServiceClient dlpServiceClient = DlpServiceClient.create()) {
  // Specify the BigQuery table to analyze
  BigQueryTable bigQueryTable =
      BigQueryTable.newBuilder()
          .setProjectId(projectId)
          .setDatasetId(datasetId)
          .setTableId(tableId)
          .build();

  // The name of the column to analyze, which doesn't need to contain numerical
  String columnName = "Mystery";

  // Configure the privacy metric for the job
  FieldId fieldId = FieldId.newBuilder().setName(columnName).build();
  CategoricalStatsConfig categoricalStatsConfig =
      CategoricalStatsConfig.newBuilder().setField(fieldId).build();
  PrivacyMetric privacyMetric =
      PrivacyMetric.newBuilder().setCategoricalStatsConfig(categoricalStatsConfi

  // Create action to publish job status notifications over Google Cloud Pub/Sub
  ProjectTopicName topicName = ProjectTopicName.of(projectId, topicId);
  PublishToPubSub publishToPubSub =
      PublishToPubSub.newBuilder().setTopic(topicName.toString()).build();
  Action action = Action.newBuilder().setPubSub(publishToPubSub).build();

  // Configure the risk analysis job to perform
  RiskAnalysisJobConfig riskAnalysisJobConfig =
      RiskAnalysisJobConfig.newBuilder()
          .setSourceTable(bigQueryTable)
          .setPrivacyMetric(privacyMetric)
          .addActions(action)
          .build();

  // Build the job creation request to be sent by the client
  CreateDlpJobRequest createDlpJobRequest =
      CreateDlpJobRequest.newBuilder()
          .setParent(LocationName.of(projectId, "global").toString())
          .setRiskJob(riskAnalysisJobConfig)
          .build();

  // Send the request to the API using the client
  DlpJob dlpJob = dlpServiceClient.createDlpJob(createDlpJobRequest);
```

```
// Set up a Pub/Sub subscriber to listen on the job completion status
final SettableApiFuture<Boolean> done = SettableApiFuture.create();

ProjectSubscriptionName subscriptionName =
    ProjectSubscriptionName.of(projectId, subscriptionId);

MessageReceiver messageHandler =
    (PubsubMessage pubsubMessage, AckReplyConsumer ackReplyConsumer) -> {
      handleMessage(dlpJob, done, pubsubMessage, ackReplyConsumer);
    };
Subscriber subscriber = Subscriber.newBuilder(subscriptionName, messageHandler
subscriber.startAsync();

// Wait for job completion semi-synchronously
// For long jobs, consider using a truly asynchronous execution model such as
try {
  done.get(15, TimeUnit.MINUTES);
} catch (TimeoutException e) {
  System.out.println("Job was not completed after 15 minutes.");
  return;
} finally {
  subscriber.stopAsync();
  subscriber.awaitTerminated();
}

// Build a request to get the completed job
GetDlpJobRequest getDlpJobRequest =
    GetDlpJobRequest.newBuilder().setName(dlpJob.getName()).build();

// Retrieve completed job status
DlpJob completedJob = dlpServiceClient.getDlpJob(getDlpJobRequest);
System.out.println("Job status: " + completedJob.getState());

// Get the result and parse through and process the information
CategoricalStatsResult result = completedJob.getRiskDetails().getCategoricalSt
List<CategoricalStatsHistogramBucket> histogramBucketList =
    result.getValueFrequencyHistogramBucketsList();

for (CategoricalStatsHistogramBucket bucket : histogramBucketList) {
  long mostCommonFrequency = bucket.getValueFrequencyUpperBound();
  System.out.printf("Most common value occurs %d time(s).\n", mostCommonFreque

  long leastCommonFrequency = bucket.getValueFrequencyLowerBound();
  System.out.printf("Least common value occurs %d time(s).\n", leastCommonFrec
```

```
      for (ValueFrequency valueFrequency : bucket.getBucketValuesList()) {
        System.out.printf(
            "Value %s occurs %d time(s).\n",
            valueFrequency.getValue().toString(), valueFrequency.getCount());
      }
    }
  }
}

// handleMessage injects the job and settableFuture into the message reciever inte
private static void handleMessage(
    DlpJob job,
    SettableApiFuture<Boolean> done,
    PubsubMessage pubsubMessage,
    AckReplyConsumer ackReplyConsumer) {
  String messageAttribute = pubsubMessage.getAttributesMap().get("DlpJobName");
  if (job.getName().equals(messageAttribute)) {
    done.set(true);
    ackReplyConsumer.ack();
  } else {
    ackReplyConsumer.nack();
  }
}
}
```


Last updated 2020-08-19 UTC.