



# Google Cloud Cortex Demand Sensing

Deployment and User Guide

# Table of Contents

## [1. Solution Overview](#)

### [1.1. Solution Components](#)

### [1.2. Deployment Overview](#)

### [1.3. Deployment Steps](#)

## [2. Deployment Prerequisites](#)

### [2.1. Deploy the Data Foundation content](#)

### [2.2. Identify Data Foundation projects and datasets](#)

### [2.3. Prepare your cloud environment](#)

### [2.3. Plan for regional availability](#)

### [2.4. Create and populate external datasets \[optional\]](#)

#### [2.4.1. Promotion Calendar](#)

#### [2.4.2. Demand Plan](#)

#### [2.4.3. Time Dimension Requirements](#)

## [3. Deploy Demand Sensing](#)

### [3.1. Deploy Demand Sensing from Google Cloud Marketplace](#)

#### [3.1.5. Run Google Cloud Cortex Demand Sensing container](#)

#### [3.1.6. Complete Additional Configuration](#)

#### [3.1.7. Execute Demand Sensing Deployment](#)

### [3.2. Validate Deployment](#)

#### [3.2.1. Validate Deployment Jobs and Solution Components](#)

#### [3.2.2. Validate Demand Forecast](#)

### [3.3. Incorporating additional external datasets \[optional\]](#)

#### [3.3.1. Data Requirements.](#)

#### [3.3.2. Data Acquisition.](#)

#### [3.3.3. Customer and Product Attribution.](#)

#### [3.3.4. Demand Sensing source code.](#)

#### [3.3.5. SQL Changes.](#)

##### [Joining with historical sales.](#)

##### [Joining with the Demand Plan.](#)

#### [3.3.6. Python Code Changes.](#)

#### [3.3.7. Running deploy.sh.](#)

## [4. Re-training the model and producing Demand Forecasts](#)

[4.1. How to Train the model](#)

[4.2. Validate ML Model](#)

[4.3. Produce ML Forecast](#)

[4.4. Logging, Monitoring and Servicing](#)

[4.4.1. Vertex AI Logging and Monitoring](#)

[4.4.2. Vertex AI Managed Datasets and BigQuery Datasets servicing](#)

[5. Configure Looker](#)

[6. Feedback and updates](#)

# 1. Solution Overview

An accurate demand plan is essential for reducing business costs and maximizing profitability. Identifying near term changes in demand is mission critical to better manage and match demand with supply.

**Google Cloud Cortex Demand Sensing** provides predefined solution content to help you get started quickly with a cloud based demand sensing solution leveraging the best of our [Data Cloud](#) services like [BigQuery](#), [Vertex AI](#), and [Looker](#) together with SAP ERP data and [additional data sources](#) like Search Trends, Weather, and more.

The solution leverages [Cortex Data Foundation](#) predefined BigQuery data models as a baseline and delivers advanced use case specific analytics and machine learning models on top. Once deployed the solution helps kick start new insights and highlights potential impacts to near term demand plans.

These insights can be consumed through predefined dashboards available on the Looker Marketplace as a follow on installation. Alternatively, other preferred visualizations solutions can be used to consume the models and sample data sets provided.

By bringing together SAP and Non SAP data sets, **Cortex Demand Sensing** helps unlock new insights from diverse demand signals beyond historical sales and allows organizations to be more nimble, through more holistic integration of demand drivers.

Get started today! See our deployment guideline below.

## 1.1. Solution Components

The following Google Cloud services are used by the solution:

- [BigQuery](#)
- [Cloud Build](#)
- [Cloud Source](#)
- [Vertex AI](#)
- [Google Cloud Storage](#)

Google Cloud Cortex Framework

- [IAM - Service Accounts](#)

We recommend you or the team implementing this solution are familiar with the technical details of these platforms and tools to ensure a successful implementation.

## 1.2. Deployment Overview

This guide provides an overview of all prerequisites, setup steps and inputs required for a successful deployment of **Google Cloud Cortex Demand Sensing** solution content.

This guide is useful for and should be completed by: Google Cloud Data and Machine Learning Technical Practitioners (or similar profiles) and Software Developers/Architects (or similar profiles). Readers will benefit from having previous understanding of general cloud-native development principles and containers.

By the end of this guide, you will:

1. Have deployed Cortex Demand Sensing solution components
2. Have fulfilled the prerequisites for data availability to train Demand Sensing forecasting models
3. Have a functional Vertex AI pipeline for Demand Sensing
4. Understand the recommended design patterns and how to effectively enhance your models from the Data Foundation with external datasets
5. Be able to tweak the pipeline and redeploy or re-train with your own datasets

## 1.3. Deployment Steps

The deployment is automated through the Demand Sensing configurator and deployer you will execute from the Demand Sensing container published in the Google Cloud Marketplace. At the end of the deployment, your Google Cloud project will contain a set of sample data, BigQuery tables and a Vertex AI model. You will also have a copy of the code to adjust and retrain the model according to your source data and business needs.

The key deployment steps are summarized below:

1. Deploy [Cortex Data Foundation](#) 5.0.1 or higher
  - a. Set SAP mandt set to **900** (SAP->mandt value in *config/config.json*)
  - b. Weather and Trends must be enabled in K9 Settings (*src/k9/config/k9\_settings.yaml*)

**k9s:**

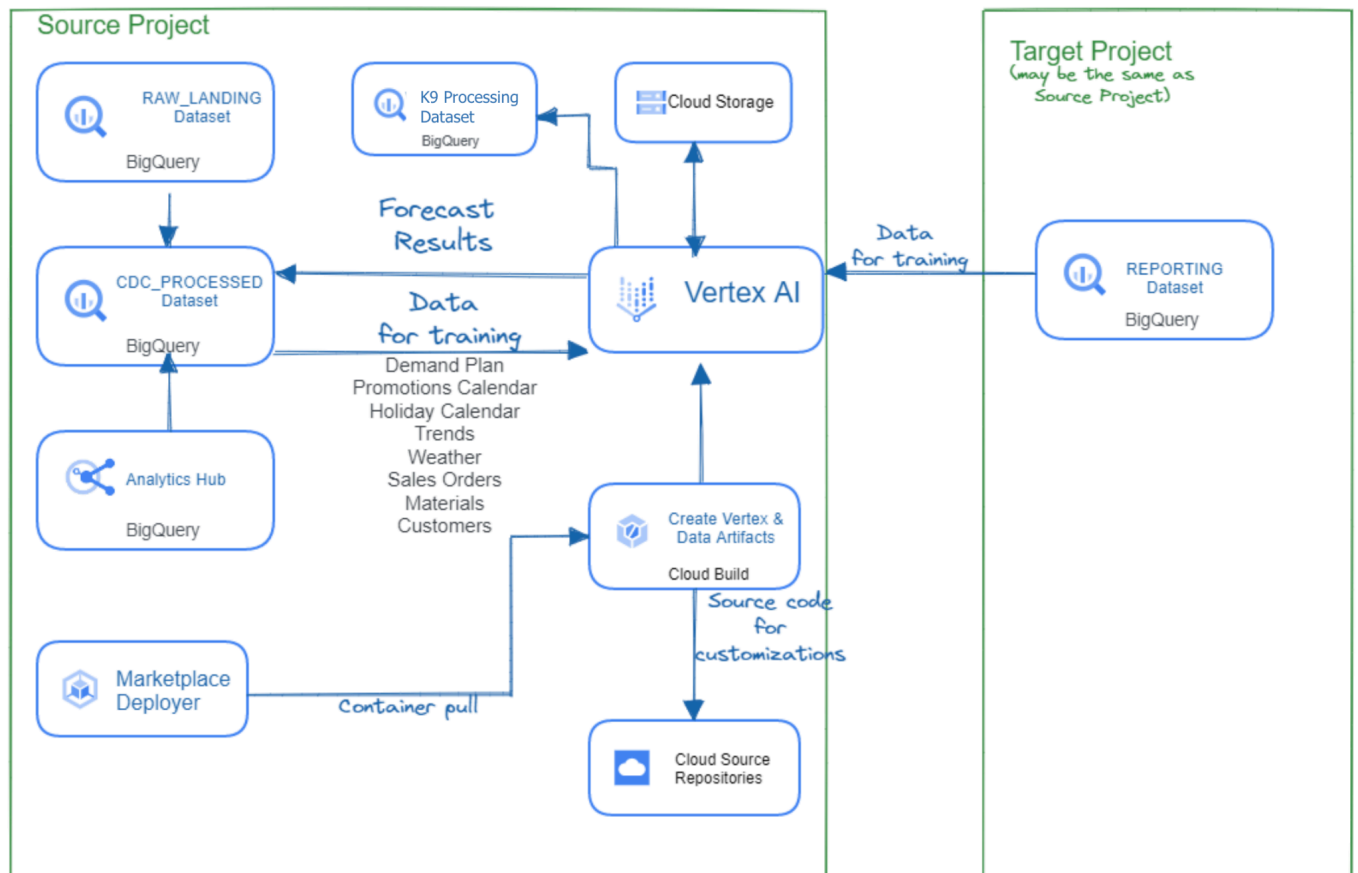
- **date\_dimension**
- **holiday\_calendar**
- **trends**
- **weather**

**Note:** Schema of *k9\_settings.yaml* in Data Foundation 5.0.1 was changed. Now you only need to list k9s to deploy ((as shown above), regardless of their stage.

2. Identify the [projects and datasets](#) used to deploy the Data Foundation (from *config/config.json*).

3. Execute Demand Sensing configuration and deployment by running its container from the Google Cloud Marketplace.
4. Validate the deployment and logs.
5. Explore the source code available in your Google Cloud Storage bucket.
6. Re-train the Vertex AI models if needed.

The following diagram illustrates the interaction between Cortex Data Foundation and Demand Sensing components:



## 2. Deployment Prerequisites

### 2.1. Deploy the Data Foundation content

Deployment requires [Cortex Data Foundation](#) to be implemented as a prerequisite. Demand Sensing will require the following minimum components from [Cortex Data Foundation](#):

- [Reporting Views](#) (and their underlying tables):
  - SalesOrders
  - CustomersMD
  - MaterialsMD

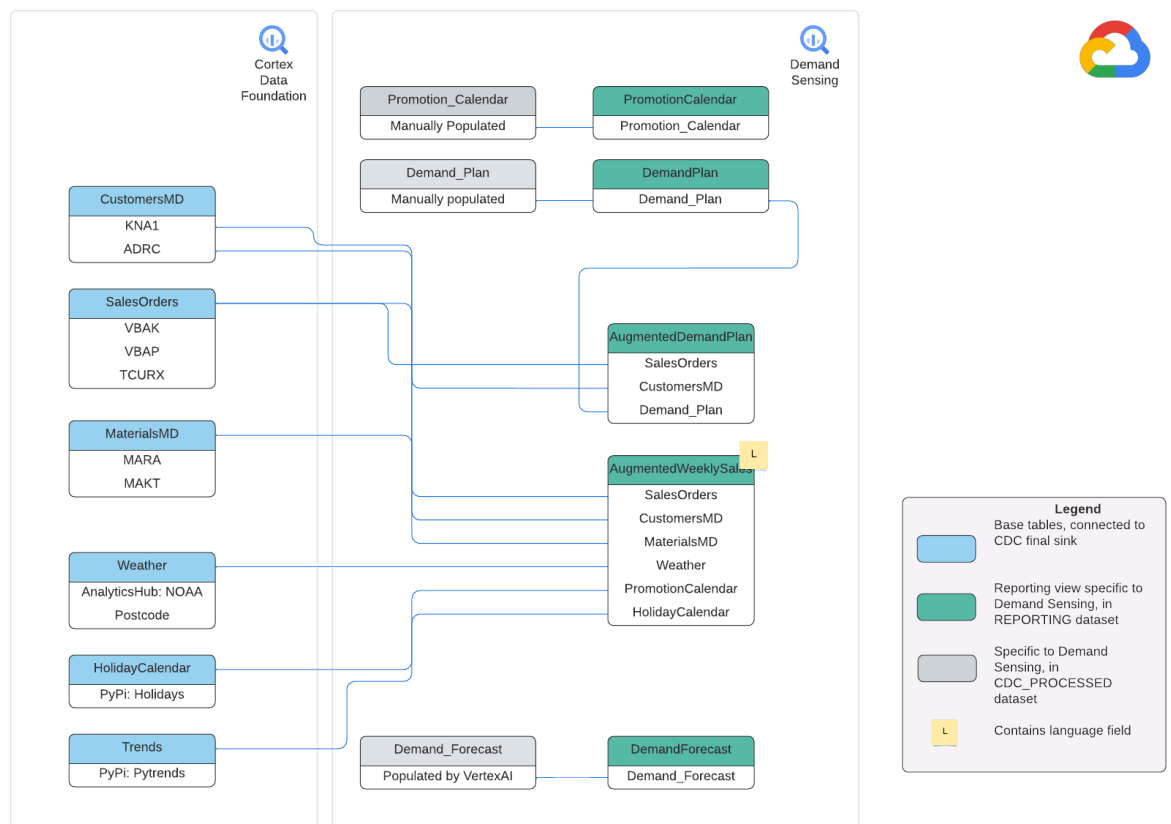
- [External Sources](#) loaded from the following DAGs
  - Trends
  - Weather
  - Holiday Calendar

If using test data, deploy the Data Foundation in full using SAP Client **900** and SQL flavor **ECC**. The test data in Mandant 900 for ECC was specifically created to feed the Vertex AI models.

All test data has been generated from an ECC source and used to tune the ML models to provide an end to end sample of a working solution. There is currently no curated demo test data for S4, so model execution and quality are not reliable if you are using the delivered test data for S4 model training.

If using your own data and not the delivered test data, configure Analytics Hub or a similar available source of your choice [as outlined in the documentation](#).

The following diagram illustrates the components required from the Data Foundation and the components added to the CDC and reporting datasets as part of the Demand Sensing requirements.



## 2.2. Identify Data Foundation projects and datasets

- Source Google Cloud Project: Project where the source data is located, from which the data models consume.
- Target Google Cloud Project: Project where the Data Foundation for SAP predefined data models was deployed. This may or may not be different from the source project depending on your needs.
- Source Raw BigQuery Dataset: BigQuery dataset where the source SAP data was replicated to or where the test data was created.
- CDC BigQuery Dataset: BigQuery dataset where the CDC processed data lands the latest available records.


**We call it below “CDC\_PROCESSED”.**

- Target BigQuery reporting dataset: BigQuery dataset where the Data Foundation for SAP predefined data models was deployed.
- K9 Processing Dataset.
- K9 Reporting Dataset.

## 2.3. Prepare your cloud environment

Make sure you can fulfill the following prerequisites before starting the deployment:

1. Setup your Cloud Shell environment

Launch the Cloud Shell by clicking in the following icon:  located in the top right corner of the console. Alternatively navigate to: <https://shell.cloud.google.com/?show=terminal>

2. Make sure you have permissions and roles:
  - a. Permissions to enable components/APIs in a [Billing-enabled project](#).
  - b. [Storage Object Admin](#).
  - c. [Service Account Creator](#).
  - d. [Cloud Build Editor](#).
  - e. [BigQuery Data Editor and BigQuery Job User](#).

If you create Service Accounts and manage their permissions during this deployment, you also should have a [Service Account Admin](#) and [Project IAM Admin](#) roles. If you need to create resources and assign permissions in advance, please follow instructions in [3.1.6. Complete Additional Configuration](#) section.

## 2.4. Plan for regional availability

[Vertex AI](#) region is one of the parameters used for deploying and running Demand Sensing.

When using [Vertex AI](#), make sure you run its jobs in the region where the [Vertex AI Forecasting feature is available](#), closest to where your source data is stored in BigQuery. If your BigQuery datasets are multi-regional, choose a Vertex AI location closest to you in the same multi-region.

You will also create a Storage Bucket in the same region you choose to run Vertex AI jobs in.

## 2.5. Create and populate external datasets [optional]

**Note:** If you used test data with Data Foundation deployment, this section can be skipped. In such cases, make sure you also use test data with Demand Sensing.

If you are **not** using test data for the initial deployment, you will need to create and populate the tables used for training of the forecasting model. These tables are additional to the [external datasets](#) populated in the Data Foundation and are outlined below with instructions on how to populate.

**IMPORTANT:** If you are not using test data with Cortex Data Foundation or Demand Sensing, you must populate the raw and CDC datasets with data using the corresponding Data Foundation capabilities. Otherwise, the Forecasting Model will not be trained at the time of Demand Sensing deployment.

If you don't populate the **Demand Plan** dataset as described below, the model will not be able to produce forecasts.

Alternatively, you can use test data for the initial deployment and replace it afterwards. The code will be available to you in a Cloud Storage Bucket to recreate tables and re-train the Vertex AI pipelines.

**Note:** All data in the BigQuery reporting views, external datasets and forecasts are aggregated on the weekly basis, with **weeks starting on Monday and ending on Sunday**. For example, the week of June 20th, 2022 goes from Monday June 20th to Sunday June 26th. You can adjust this in the DAGs and view SQL in the data foundation if required.

### 2.5.1. Promotion Calendar

If not using test data, you will need to create and populate the Promotion Calendar table. This table includes information on planned promotions for a given product and customer, on a weekly basis.

The following is the structure of the table:

Google Cloud Cortex Framework



Field Name	Data Type	Description
StartDateofWeek	DATE	Monday's date of the target week
CustomerId	STRING	Customer ID corresponding to SAP's Master Data table KNA1, field KUNNR
CatalogItemId	STRING	Material ID corresponding to SAP's Material Master table MARA, field MATNR
IsPromo	BOOLEAN	Is TRUE when there is a promotion running on the week of StartDateOfTheWeek for material CatalogItemId with customer CustomerId.
DiscountPercent	FLOAT	optional field indicating the promotion discount in %%. It is currently not used by the forecasting model.

The table is expected to exist in the **CDC\_PROCESSED** dataset before the deployment of Demand Sensing. If using test data, the deployment process will create and populate the table for you.

Here is a sample statement to create the table if not using test data.

```
CREATE TABLE IF NOT EXISTS `{{ project_id_src }}.{{ dataset_cdc_processed }}.Promotion_Calendar`
(
  StartDateOfWeek DATE,
  CustomerId STRING,
  CatalogItemId STRING,
  IsPromo BOOLEAN,
  DiscountPercent FLOAT64
)
```

The promotion data must be aggregated on the weekly basis. In other words: If there is a promotion or discount for a product and customer in a specific week, there should be a corresponding record in the Promotion\_Calendar table.

**IMPORTANT:** To train a high quality forecasting model, it is necessary to populate the Promotion Calendar both with **the historical and future promotions**. Ideally, the promotions in the table should go as far as the earliest order date. Future promotions are needed as far as the forecast horizon, up to 13 weeks from the full week with last known sales (presumably the week before the current one).

## 2.5.2. Demand Plan

If not using test data, create and populate the **Demand\_Plan** table in the CDC\_PROCESSED dataset.

The following is the expected structure of the table:

Field Name	Data Type	Description
WeekStart	DATE	Monday's date of the target week
CustomerId	STRING	Customer ID corresponding to SAP's Master Data table KNA1, field KUNNR
CatalogItemId	STRING	Material ID corresponding to SAP's Material Master table MARA, field MATNR
DemandPlan	FLOAT	Planned cumulative order quantity (as in VBAP-KWMENG field) aggregated over the week, per customer (CustomerId), per material (CatalogItemId)

Here is a sample statement to create the table if not using test data.

```
CREATE TABLE IF NOT EXISTS `{{ project_id_src }}.{{ dataset_cdc_processed }}.Demand_Plan`  
(  
  WeekStart DATE,  
  CustomerId STRING,  
  CatalogItemId STRING,  
  DemandPlan FLOAT64  
)
```

The demand plan data must be aggregated on the weekly basis, per material, per customer. When there is no sale planned on a certain week, make a row with 0 (zero) in the DemandPlan field.

**IMPORTANT:** **Demand\_Plan** table should be populated as far as the forecast horizon, up to 13 weeks from the full week with last known sales (presumably the week before the current one).

Any Demand Plan data made for periods before the last known sales data will be ignored by the forecasting model.

### 2.5.3. Time Dimension Requirements

The fields named **WeekStart** or **StartDateOfWeek** denote the time period for that record in **all** external datasets tables in BigQuery used as sources of predictive variables used in the ML model. **It stands for the full week starting on that date with Monday as the first date of the week.** A value of 2022-01-17, for instance, covers the time period from Jan 17th, 2022 through Jan 23rd, 2022, Monday to Sunday.

Values for this field must be valid parseable dates. Furthermore, the data *must not* have missing WeekStart values. To clarify, if the minimum value is 2019-01-07 and the maximum value is 2022-03-21, *every single week* in that period must be present in the datasets.

## 3. Deploy Demand Sensing

### 3.1. Deploy Demand Sensing from Google Cloud Marketplace

The instructions in this section outline how to deploy Cortex Demand Sensing from Google Cloud Marketplace. You should only proceed if you have already deployed the Data Foundation content as it is a prerequisite.

#### 3.1.5. Run Google Cloud Cortex Demand Sensing container

In the [Cloud Shell](#), execute the following command:

```
gcloud auth configure-docker gcr.io && \
docker run -it --rm \
marketplace.gcr.io/cortex-public/cortex-demand-sensing:latest
```

This command will pull and run the Demand Sensing container from Google Cloud Marketplace.

The container starts from running the Demand Sensing deployment configurator.

```
===== Google Cloud Cortex Demand Sensing deployment configuration =====
```

The configurator will guide you through the deployment configuration by asking a few questions:

- f. **Source GCP Project:** The source project where data replication and CDC processing datasets are. Refer to the [Data Foundation documentation](#) for more information.
- g. **Target GCP Project:** The target project where Reporting views are (may be the same as the source depending on your deployment of the Data Foundation). Refer to the [Data Foundation documentation](#) for more information.
- h. **Raw Landing Dataset:** BigQuery dataset where SAP data is replicated.

- i. **CDC Processed Dataset:** BigQuery dataset where the CDC processed data is. We also call it “CDC\_PROCESSED” in this Guide. The additional external tables (see [Section 2.5. Create and populate external datasets](#)) are expected to be found here if not using test data.
- j. **Reporting Dataset:** The dataset for reporting used in the Data Foundation.
- k. **K9 Processing Dataset:** Data Foundation K9 Processing Dataset.
- l. **K9 Reporting Dataset:** Data Foundation K9 Reporting Dataset.
- m. **Data Foundation BigQuery Location:** BigQuery location (multi-region) used with Data Foundation. If it was not specified, the Data Foundation used **US**. Default value is taken from the Raw Landing Dataset location.
- n. **Vertex AI Region:** Region where the Vertex AI pipelines will run. Must be a single region (e.g. us-central1), same as you used for Vertex AI bucket. Be sure to keep within the same multi-region location chosen for BigQuery.
- o. **Vertex AI service account:** Service Account to run Vertex AI jobs. If it doesn’t exist, you can create one after configuring Demand Sensing deployment, but before executing the deployment. Default value is “cortex-ds-vertexai-sa” in the source project.
- p. **Storage Bucket for Vertex AI:** GCS bucket for Vertex AI to store intermediate assets during model training and scoring. If it doesn’t exist, you can create one after configuring Demand Sensing deployment, but before executing the deployment. This bucket must be created in the region specified as “Vertex AI Region” parameter above. Default value is a combination of the source project id, “-cortex-ds-vertexai-” string and Vertex AI region.
- q. **Forecast Horizon (weeks):** How far in the future you want to make predictions, up to 13 weeks. Leave the default (13) if using Test Data. See section [4. Re-training the model and producing Demand Forecasts](#) for more details on this parameter.
- r. **Context Window (weeks):** How far back the model looks during training (and for forecasts). Leave the default (52) if using Test Data. See section [4. Re-training the model and producing Demand Forecasts](#) for more details on this parameter.
- s. **Model Training Budget (hours):** How many whole hours Vertex AI AutoML will spend actually training the model. 1 hour is the minimum time. Leave the default (1) if using Test Data. See section [4. Re-training the model and producing Demand Forecasts](#) for more details on this parameter.
- t. **Deploy Test Data:** Answer **yes** if you used test data with Demand Sensing and would like to use Test Data with Demand Sensing.

If you didn’t run Data Foundation with flags `testData: true` and would like to use Demand Sensing Test Data, please make another deployment of Data Foundation into new datasets. *Simply running Data Foundation over the same datasets will not provision test data because it’s using “--noreplace*

flag” with “*bq load*” commands. Refer to the [Data Foundation documentation](#) for more information.






- u. **Mandant/Client:** Use **900** for test data (previous parameter is **yes**). Otherwise, use the right client for your source SAP system.

### 3.1.6. Complete Additional Configuration

Demand Sensing deployment configurator will inform you about next steps required to execute Demand Sensing deployment, and will offer a command snippet you may use to perform necessary configuration steps:

*Most of the names in the examples below are going to be different for you. Vertex AI Service Account and Vertex AI Bucket names below are shown with their default values generated from the source project id “your-source-project”.*

To deploy Google Cloud Cortex Demand Sensing, please complete additional configuration steps:

1. Enable Vertex AI in the source project.
2. Create `your-source-project-cortex-ds-vertexai-us-central1` storage bucket for Vertex AI in `us-central1` region.
3. Create `cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com` service account for Vertex AI.
4. Grant `cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com` service account `objectAdmin` role on `your-source-project-cortex-ds-vertexai-us-central1` storage bucket.
5. Grant `cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com` service account the following roles in the source project:
  -  `aiplatform.user`
  -  `bigquery.dataEditor`
  -  `bigquery.jobUser`
6. Grant `cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com` service account the following roles in the target project:
  -  `bigquery.dataEditor`
  -  `bigquery.jobUser`
7. Grant source project's Cloud Build account (`12345678@cloudbuild.gserviceaccount.com`) `iam.serviceAccountUser` role for `cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com` service account.
8. Grant source project's Cloud Build account (`12345678@cloudbuild.gserviceaccount.com`) `aiplatform.admin` and `storage.admin` roles on the source project.
9. Run `./ds-deploy` command to continue deploying Cortex Demand Sensing.

Would you like to get a command snippet to perform the operations above? ☒ yes

Complete additional configuration steps as instructed by the configurator.

Once the configurator finishes, you will stay in a Bash session running inside the Demand Sensing container.

To complete additional configuration, you can use command snippets provided by the configurator (if answered **yes**) or use your own way, with the command line or Google Cloud Console.

Google Cloud Cortex Framework

```

# Set the source project as the active project
gcloud config set project your-source-project

# Enable Vertex AI and Cloud Source Repositories APIs in the source project
gcloud services enable aiplatform.googleapis.com sourcerepo.googleapis.com --project="your-source-project"

# Create a storage bucket for Vertex AI if it doesn't exist
gsutil ls -b gs://your-source-project-cortex-ds-vertexai-us-central1 2> /dev/null || \
gsutil mb -l us-central1 gs://your-source-project-cortex-ds-vertexai-us-central1

# Create a service account as Vertex AI Service Account (if it doesn't exist)
gcloud iam service-accounts describe cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com 2> /dev/null || \
gcloud iam service-accounts create cortex-ds-vertexai-sa \
--description="Vertex AI Demand Sensing" \
--display-name="Vertex AI Demand Sensing" \
--project="your-source-project"

# Grant Vertex AI Service Account permissions on the bucket
gsutil iam ch \
serviceAccount:cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com:objectAdmin \
gs://your-source-project-cortex-ds-vertexai-us-central1

# Grant BigQuery and Vertex AI permissions in the source project.
for role in 'roles/aiplatform.user' 'roles/bigquery.dataEditor' 'roles/bigquery.jobUser';
do
    gcloud projects add-iam-policy-binding your-source-project \
        --member="serviceAccount:cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com" \
        --role="$role"
done

# Grant BigQuery and Vertex AI permissions in the target project.
for role in 'roles/bigquery.dataEditor' 'roles/bigquery.jobUser';
do
    gcloud projects add-iam-policy-binding your-target-project \
        --member="serviceAccount:cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com" \
        --role="$role"
done

# Grant Cloud Build account iam.serviceAccountUser role for Vertex AI Service Account
gcloud iam service-accounts add-iam-policy-binding \
cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com \
--member="serviceAccount:12345678@cloudbuild.gserviceaccount.com" \
--role="roles/iam.serviceAccountUser"

# Grant Cloud Build account Vertex AI and Storage roles in the source project.
for role in 'roles/aiplatform.admin' 'roles/storage.admin';
do
    gcloud projects add-iam-policy-binding your-source-project \
        --member="serviceAccount:12345678@cloudbuild.gserviceaccount.com" \
        --role="$role"
done

```

**IMPORTANT:** After finishing Demand Sensing deployment configuration, your Cloud Shell will stay in the context of running Demand Sensing container. Do not close it. You can execute all necessary CLI commands from there.

If your Cloud Shell session gets suspended, you will need to run `docker run -it --rm marketplace.gcr.io/cortex-public/cortex-demand-sensing:latest` command in Cloud Shell again and use the same configuration parameters.

### 3.1.7. Execute Demand Sensing Deployment

Once you created Vertex AI BigQuery Dataset, GCS bucket for Vertex AI, Vertex AI Service account, and configured IAM roles for Cloud Build Service Account and Vertex AI Service Account, you can execute Demand Sensing deployment using `./ds-deploy` command.

Make sure you run `./ds-deploy` in the same Cloud Shell session you used with steps 1 and 2 above. You will see `/usr/src/app` as your current directory.

If everything was configured correctly, `ds-deploy` script will start Demand Sensing deployment and Vertex AI Pipeline execution.

**Note:** `ds-deploy` script checks whether Vertex AI bucket, Vertex AI BQ dataset and Vertex AI service account exist. It also makes sure Vertex AI service account and Cloud Build service account have necessary IAM roles on the projects and resources.

If you prefer configuring more precise permissions on resources, run `ds-deploy` with `--no-iam-check` flag:

```
./ds-deploy --no-iam-check
```

## 3.2. Validate Deployment

The deployment process starts with validating deployment configuration.

```
root@cbb72d09dbfd: /usr/src/app# ./ds-deploy
🐱🐱🐱 Google Cloud Cortex Demand Sensing deployment 🐱🐱🐱
Updated property [core/project].
Checking configured resources...
✅ Service Account cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com exists.
✅ Bucket your-source-project-cortex-ds-vertexai-us-centrall exists.
✅ Dataset your-source-project:VERTEX_AI_DATA exists.
✅ Service Account cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com has necessary roles in project your-source-project
✅ Service Account cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com has necessary roles in project your-target-project
✅ Service Account cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com has storage.objectAdmin on bucket your-source-project-cortex-ds-vertexai-us-centrall
```

```

✓ Service Account 12345678@cloudbuild.gserviceaccount.com has necessary roles in project your-source-project
✓ Service Account 12345678@cloudbuild.gserviceaccount.com has necessary roles in project your-target-project
✓ Service Account 12345678@cloudbuild.gserviceaccount.com has iam.serviceAccountUser role on Service Account
cortex-ds-vertexai-sa@your-source-project.iam.gserviceaccount.com

```

...there will be more output here...

### 3.2.1. Validate Deployment Jobs and Solution Components

1. The deployment process continues with a series of Cloud Build jobs. The last job will start a Vertex AI pipeline for training a Forecasting model and producing its first forecast.

If the job successfully triggers the build process, you can find the Build logs in [Cloud Build](#).

Errors will be easier to troubleshoot from there. You will find a parent build process that has two steps :

Steps	Duration	BUILD LOG	EXECUTION DETAILS	BUILD ARTIFACTS
<div> <div>✓</div> <div>Build Summary</div> <div>2 Steps</div> </div>	00:10:11	<div> <input type="checkbox"/> Wrap lines           <input type="checkbox"/> Show newest entries first           <div> <div>T</div> <div>↓</div> </div> </div>		
<div> <div>✓</div> <div>0: gcr.io/cloud-builders/gcloud</div> <div>bash -c if [ 'false' = "true" ]; then ./create_test_harness.sh 's</div> </div>	00:00:00	<pre> 1 starting build "e304916f-8cc2" 2 3 FETCHSOURCE 4 Fetching storage object: gs://cloudbuild/source/1655872276.807415-e0b3aad5df66- 5 Copying gs://cloudbuild/source/1655872276.807415-e0b3aad5df66-42b0abf4dcad079e9 6 / [0 files][ 0.0 KiB / 85.5 KiB] 7 / [1 files][ 85.5 KiB / 85.5 KiB] 8 Operation completed over 1 objects/85.5 KiB. 9 BUILD 10 Starting Step #0 11 Step #0: Already have image (with digest): gcr.io/cloud-builders/gcloud 12 Finished Step #0 13 Starting Step #1 14 Step #1: Already have image (with digest): gcr.io/cloud-builders/gcloud 15 Step #1: INFO: "context-window" missing, using default value 16 Step #1: INFO: "model-training-hours" missing, using default value. 17 Step #1: Running with the following parameters: 18 Step #1: source-project: s 19 Step #1: target-project: s 20 Step #1: storage-bucket: s-cortex-ds-vertexai 21 Step #1: cdc-processed-dataset: live_cdc 22 Step #1: raw-landing-dataset: live_raw 23 Step #1: target-reporting-dataset: liverepoi 24 Step #1: target-models-dataset: liveml 25 Step #1: vertex-ai-dataset: liveml 26 Step #1: vertex-ai-region: us-central1 27 Step #1: location: US 28 Step #1: mandt: 900 29 Step #1: forecast-horizon: 13 30 Step #1: context-window: 52 31 Step #1: forecast compute budget: 1 32 Step #1: views_dir: sql 33 Step #1: Not creating liverepoi since it already exists 34 Step #1: Creating dataset s:liveml with location: US 35 Step #1: BigQuery error in mk operation: Dataset 's:liveml' already exists. 36 Step #1: Creating dataset s:liveml with location: US </pre>		
<div> <div>✓</div> <div>1: gcr.io/cloud-builders/gcloud</div> <div>bash -c ./deploy.sh \--source-project ' ' \--target-project "</div> </div>	00:10:03			

These steps will sequentially complete the following tasks:

1. Trigger the creation of the test harness if requested,
2. Create the tables and views for training if they do not exist
3. Spawn another build process for the Vertex



```

1217 2022-06-22 15:30:47,467 | INFO | root | MetadataStore default not found.
1218 2022-06-22 15:30:57,880 | INFO | root | Resource cortex-demand-sensing-pipeline not found.
1219 2022-06-22 15:30:57,881 | INFO | root | Creating Resource cortex-demand-sensing-pipeline
1220 2022-06-22 15:30:58,272 | INFO | root | Resource cortex-demand-sensing-pipeline-cortex-demand-sensing-pipeline-2022-06-22t15-30-47-237605 not found
1221 2022-06-22 15:30:58,274 | INFO | root | Creating Resource cortex-demand-sensing-pipeline-cortex-demand-sensing-pipeline-2022-06-22t15-30-47-237605
1222 2022-06-22 15:30:58,879 | INFO | root | Resource cortex-demand-sensing-pipeline-cortex-demand-sensing-pipeline-2022-06-22t15-30-47-237605-metrics not found
1223 2022-06-22 15:30:58,880 | INFO | root | Creating Resource cortex-demand-sensing-pipeline-cortex-demand-sensing-pipeline-2022-06-22t15-30-47-237605
1224 Creating PipelineJob
1225 2022-06-22 15:31:00,330 | INFO | google.cloud.aiplatform.pipeline_jobs | Creating PipelineJob
1226 PipelineJob created. Resource name: projects/4[REDACTED]5/locations/us-central1/pipelineJobs/demand-sensing-pipeline-20220622153100
1227 2022-06-22 15:31:00,867 | INFO | google.cloud.aiplatform.pipeline_jobs | PipelineJob created. Resource name: projects, [REDACTED] 5/locations/us-cent
1228 To use this PipelineJob in another session:
1229 2022-06-22 15:31:00,868 | INFO | google.cloud.aiplatform.pipeline_jobs | To use this PipelineJob in another session:
1230 pipeline_job = aiplatform.PipelineJob.get('projects/[REDACTED]75/locations/us-central1/pipelineJobs/demand-sensing-pipeline-20220622153100')
1231 2022-06-22 15:31:00,868 | INFO | google.cloud.aiplatform.pipeline_jobs | pipeline_job = aiplatform.PipelineJob.get('projects/[REDACTED] 5/locations
1232 2022-06-22 15:31:00,868 | INFO | google.cloud.aiplatform.pipeline_jobs | View Pipeline Job:
1233 https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/demand-sensing-pipeline-20220622153100?project=\[REDACTED\]
1234 View Pipeline Job:
1235 https://console.cloud.google.com/vertex-ai/locations/us-central1/pipelines/runs/demand-sensing-pipeline-20220622153100?project=\[REDACTED\]
1236 PUSH
1237 DONE
1238 -----
1239 ID CREATE_TIME DURATION SOURCE
1240 48749c75-dffe-4d52-84dd-3f1da6d0a8f9 2022-06-22T15:30:18+00:00 41S gs://s1[REDACTED] 1817.273758-a339893be8494d8[REDACTED]
1241 Vertex AI Machine Learning Pipeline is still running, please refer to its logs and Vertex AI console for status.

```

At the end of the second step, if successful, you will also find a link to the Vertex AI pipeline:

[https://console.cloud.google.com/vertex-ai/locations/REGION/pipelines/runs/demand-sensing-pipeline-PIPELINE\\_ID?project=PROJECT\\_ID](https://console.cloud.google.com/vertex-ai/locations/REGION/pipelines/runs/demand-sensing-pipeline-PIPELINE_ID?project=PROJECT_ID)

Line “Vertex AI Machine Learning Pipeline is still running, please refer to its logs and Vertex AI console for status.” indicates that the [Vertex AI forecasting pipeline](#) has been submitted.

The training and scoring pipeline will continue to run for about an hour on test data. You will find more instructions about monitoring these pipelines in section [4.4.1. Vertex AI Logging and Monitoring](#).

**Note:** At this point, you can leave the Demand Sensing container in Cloud Shell using **exit** command.

### 3.2.2. Validate Demand Forecast

When the Vertex AI pipeline is finished, it is expected to fill the Demand\_Forecast table in the CDC\_PROCESSED dataset you specified in the source GCP project.

The following table shows the expected structure of the DemandForecast table in the CDC\_PROCESSED dataset.

<input type="checkbox"/>	Field name	Type
<input type="checkbox"/>	WeekStart	DATE
<input type="checkbox"/>	CustomerId	STRING
<input type="checkbox"/>	CatalogItemId	STRING
<input type="checkbox"/>	DateOfForecast	DATE
<input type="checkbox"/>	MLForecastQuantity	FLOAT
<input type="checkbox"/>	MLForecastQuantityLowerBound	FLOAT
<input type="checkbox"/>	MLForecastQuantityUpperBound	FLOAT

**DemandForecast** view in the reporting dataset (target project) will have the same data.

To understand the forecasting results, please follow section [4.3. Produce ML Forecast](#). Please note that the forecast is always made using the best Demand Sensing model, not the last one.

## 3.3. Incorporating additional external datasets [optional]

Every business has its own specific set of processes and respective data that can be significant for demand planning and forecasting.

You can integrate additional datasets into Demand Sensing through code modifications. You will need to join the new data into the structures consumed by Vertex AI for it to be incorporated into the machine learning model.

**Note:** Please get familiar with Demand Sensing source code by running it manually as described in [4. Re-training the model and producing Demand Forecasts](#)

### 3.3.1. Data Requirements.

Below are the requirements for any external data to make it useable by the Demand Sensing predefined forecasting model:

1. Every data point (row) must be attributable to at least one of the following:
  - a. Customer (from SAP table KNA1)
  - b. Material or Product (from SAP table MARA)
2. For historical purposes you need to have data all the way back to the date of the first sales order in SAP. You can get this from the field AUDAT from table VBAK or the existing SalesOrder view in the Data Foundation.
3. If future values can be forecasted, populate them all the way to the last week of your Demand Plan.

4. If data changes over time, it has to be aggregated on the weekly basis. For data with finer granularity, such something that changes daily, depending on what exactly the data represents, use one of the following strategies:
  - a. Addition (summarize over all week days). Useful for events that directly drive sales. More of them over the week can be attributed to higher sales numbers (traffic, clicks, subsequent retail sales).
  - b. Weekly average. Useful for slow indicators with extreme values having lesser or indirect impact on sales, peak events don't matter as much. Precipitation is an example.
  - c. Max or min. Even a single event in a week has a significant impact: heat wave, sales promotion, holidays.
  - d. Combination of a and c or b and c, similar to what Demand Sensing uses for temperature.

We use **Point-of-Sale (PoS)** data as an example of incorporating a new dataset.

### 3.3.2. Data Acquisition.

Acquire data on the same cadence as you update your Demand Plan. The easiest way to do this is by creating a Cloud Composer DAG [as in Cortex Data Foundation](#).

**Point-of-Sale** datasets can be acquired from Retailer Direct Data or syndicated data from Nielsen, IRI, or other sources. These are optional datasets that can be added to the model if you think they may be relevant.

### 3.3.3. Customer and Product Attribution.

As part of the the data acquisition step, make sure your data is attributed to Customer/Location and/or Materials so it can be joined by CustomerId (KNA1.KUNNR), Customer location (country and postal code, KNA1.LAND1 and KNA1.PSTLZ/ADRC.POST\_CODE1) and/or MaterialId (MARA.MATNR).

PoS transactions cannot always be attributed to a particular customer. Sometimes they are mapped to a DMA Location (Designated Market Area) or a group of customers. In such cases, there must be a way to attribute weekly sales per product per customer the same way, so the PoS data can be successfully joined with the sales data.

With Point-of-Sale data in this guide, for simplicity, we assume that every transaction can be mapped to a customer and a product/material.

### 3.3.4. Demand Sensing source code.

Download Demand Sensing source code. A copy in your Cloud Storage bucket is created by the deployment, after the process is successful.

It's located at `gs://VERTEX_AI_PIPELINE_BUCKET/cortex_demand_sensing_1_0`, where `VERTEX_AI_PIPELINE_BUCKET` is the bucket name you used as Vertex AI pipeline bucket when deployed Demand Sensing.

```
mkdir -p cortex_demand_sensing & cd &\ngcloud storage cp -r gs://VERTEX_AI_PIPELINE_BUCKET/cortex_demand_sensing_1_0 .
```

### 3.3.5. SQL Changes.

Create a table in `CDC_PROCESSED` dataset and fill it with data. If you choose to aggregate it, do so on a weekly basis. As you probably noticed in the source code, we refer to `CDC_PROCESSED` dataset as

```
`{{ project_id_tgt }}.{{ dataset_cdc_processed }}`.
```

Here is how it should look for **Point-of-Sale** data. Assume it's called **PoS**.

Column	Type	Description
TransactionDate	DATE	Retail sale date
CustomerId	STRING	KNA1.KUNNR
CatalogItemId	STRING	MARA.MATNR
RetailUnitsSold	FLOAT	Number of retail units sold.

General guidance on sales numbers is to stick with the number of items sold rather than money.

#### Joining with historical sales.

Next step is to add this data into the **AugmentedWeeklySales** view. Modify `sql/AugmentedWeeklySales.sql` file

1. In the last SELECT statement, add selecting **PoS.RetailUnitsSold** aggregating it over a week:

```
SUM(PoS.RetailUnitsSold) OVER (PARTITION BY DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))) AS RetailUnitsSold
```

2. Add one more JOIN to the same statement:

**LEFT JOIN**

```
`{{ project_id_tgt }}.{{ dataset_cdc_processed }}.PoS` AS PoS
```

**ON**

```
WMSPCPL.WeekStart = DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))
```

```
AND WMSPCPL.CustomerId = PoS.CustomerId
```

```
AND WMSPCPL.CatalogItemId = PoS.CatalogItemId
```

This is how that piece `sql/AugmentedWeeklySales.sql` of would look like:

```
56 -- Average interest from Google Trends a week before
57 AVG(IFNULL(Trends.InterestOverTime, 0))
58 OVER(PARTITION BY Trends.HierarchyId, Trends.CountryCode, DATE_TRUNC(Trends.WeekStart, WEEK(MONDAY))) AS AvgInterest,
59 -- Point-of-Sale data You, now * Uncommitted changes
60 SUM(PoS.RetailUnitsSold) OVER (PARTITION BY DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))) AS RetailUnitsSold
61 FROM
62 WMSPCPL
63 LEFT JOIN
64 `{{ project_id_tgt }}.{{ dataset_cdc_processed }}.PoS` AS PoS
65 ON
66 WMSPCPL.WeekStart = DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))
67 AND WMSPCPL.CustomerId = PoS.CustomerId
68 AND WMSPCPL.CatalogItemId = PoS.CatalogItemId
69 LEFT JOIN
70 `{{ project_id_tgt }}.{{ dataset_reporting_tgt }}.Weather` AS Weather
71 ON
72 -- Target week forecasts
```

## Joining with the Demand Plan.

If future data is forecasted and can be leveraged, it needs to be in `AugmentedDemandPlan` view as well.

This is not the case for Point-of-Sale data because future retail transactions are unknown, but we demonstrate respective changes in `sql/AugmentedDemandPlan.sql` file for the purpose of learning.

The approach is very similar to the modifications of the view `AugmentedWeeklySales`. Modifications to `sql/AugmentedDemandPlan.sql` file are:

1. In the last SELECT statement, add selecting `PoS.RetailUnitsSold` aggregating it over a week:

```
SUM(PoS.RetailUnitsSold) OVER (PARTITION BY DATE_TRUNC(PoS.TransactionDate,
WEEK(MONDAY))) AS RetailUnitsSold
```

2. Add one more JOIN to the same statement:

```
LEFT JOIN
`{{ project_id_tgt }}.{{ dataset_cdc_processed }}.PoS` AS PoS
ON
CWA.WeekStart = DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))
AND CWA.CustomerId = PoS.CustomerId
AND CWA.CatalogItemId = PoS.CatalogItemId
```

This is how that piece `sql/AugmentedDemandPlan.sql` of would look:

```

54 | AVG(IFNULL(Weather.MinTemp, 10)) OVER (PARTITION BY CWA.Location, CWA.WeekStart) as MinTemp,
55 | -- Point-of-Sale data | You, 6 minutes ago • Uncommitted changes
56 | SUM(PoS.RetailUnitsSold) OVER (PARTITION BY DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))) AS RetailUnitsSold
57 | FROM
58 | CWA
59 | LEFT JOIN
60 | `{{ project_id_tgt }}.{{ dataset_cdc_processed }}.PoS` AS PoS
61 | ON
62 | CWA.WeekStart = DATE_TRUNC(PoS.TransactionDate, WEEK(MONDAY))
63 | AND CWA.CustomerId = PoS.CustomerId
64 | AND CWA.CatalogItemId = PoS.CatalogItemId
65 | LEFT JOIN
66 | `{{ project_id_tgt }}.{{ dataset_reporting_tgt }}.Weather` AS Weather

```

### 3.3.6. Python Code Changes.

This part depends on whether you have future data in a forecasted or planned form.

For instance, you can have a Weather forecast, planned promotions or known holidays, but Point-of-Sale data always comes after the fact.

If future data is known, and you have it up to the last week of the Demand Plan, you don't need to make any changes in the source code in regards to the dataset. As described above, Vertex AI will expect the same column to be in **AugmentedWeeklySales** and **AugmentedDemandPlan** views.

If future data is **unknown**, just like with Point-of-Sales dataset, we need to tell Vertex AI Forecasting about it:

1. Open `src/pipeline_utils_ops.py` file, locate `make_table_spec` function.
2. In `make_table_spec` function, change `primary_table` dictionary, so that `unavailable_at_forecast_columns` list has columns from **AugmentedWeeklySales** related to your dataset joined into that view. For Point-of-Sale example, this is what `primary_table` with **RetailUnitsSold** field added would look like:

```

primary_table = {
    "bigquery_uri": f"{training_dataset_bq_url}.{training_table}",
    "table_type": "FORECASTING_PRIMARY",
    "forecasting_primary_table_metadata": {
        "time_column": "WeekStart",
        "target_column": "SalesQuantity",
        "time_series_identifier_columns": ["CatalogItemId", "CustomerId"],
        "unavailable_at_forecast_columns": [
            "SalesQuantity",
            "AvgInterest",
            "RetailUnitsSold"

```

```

    ],
    "time_granularity": {"unit": "WEEK", "quantity": 1},
  },
}

```

Every time you change model the dataset structure, with new set of columns or different columns known at training/prediction, the scoring pipeline needs to make sure it always picks the right model - one with the same set of columns it was trained on. To differentiate between model variants, use different model labels in `src/pipeline_constants.py` file. For your new version, change `model_label_value` in `PipelineConstants` class:

```

model_label_value: str = "1_1_PoS" # new label for model with PoS

```

### 3.3.7. Running deploy.sh.

Run `deploy.sh` script

```

./deploy.sh \

--source-project SOURCE_PROJECT \

--target-project TARGET_PROJECT \

--storage-bucket VERTEX_AI_PIPELINE_BUCKET \

--cdc-processed-dataset CDC_PROCESSED_DATASET_NAME \

--k9-processing-dataset K9_PROCESSING_DATASET_NAME \

--vertex-ai-sa VERTEX_AI_SERVICE_ACCOUNT \

--raw-landing-dataset RAW_LANDING_DATASET \

--target-reporting-dataset REPORTING_DATASET \

--target-models-dataset MODELS_DATASET \

--location GCP_LOCATION \

--vertex-ai-region VERTEX_AI_REGION \

--mandt MANDT \

--forecast-horizon FORECAST_HORIZON \

--context-window CONTEXT_HORIZON \

```

```
--model-training-hours MODEL_TRAINING_HOURS \
```

```
.
```

Make sure you use the same parameter values as for the initial deployment. Notice the trailing dot (.) in the command line.

## 4. Re-training the model and producing Demand Forecasts

To perform forecast model training and scoring after deploying Demand Sensing package from the Marketplace, download Demand Sensing code from your Storage Bucket used for Vertex AI (`VERTEX_AI_PIPELINE_BUCKET` parameter above).

```
mkdir -p cortex_demand_sensing & cd &\  
gcloud storage cp -r gs://VERTEX_AI_PIPELINE_BUCKET/cortex_demand_sensing_1_0 .
```

Cortex Demand Sensing uses [Vertex AI Forecasting](#) - a specialized AutoML offering based on Deep Learning models. Demand Sensing trains the forecasting model on the following historical data:

1. Weekly sales per product per customer location.
2. Weather per customer location, from NOAA\_GFS0P25 dataset.
3. Google Trends average interest per product category per customer's location region. Product category is taken from a higher level of product hierarchy (T179 and T179T).
4. Holiday Calendar.
5. Promotion Calendar with weekly promotions per product per customer.

In simple terms, when training the model, it's building a set of dependencies between weekly sales and the rest of the data. In addition to regular seasonal changes, the model will get a sense of how sales of a certain product depend on weather, Google Trends, different holidays, and promotions. The model is not only getting same week correlations, but also catching lagging indicators, such as dependencies between current Google Trends and future sales.

When building a forecast by scoring the model, the following data is used:

1. Weather forecast. It uses 16-days forecast from NOAA\_GFS0P25 dataset, plus historical weekly climate averages up to 13 weeks ahead.
2. Holiday Calendar.
3. Promotion Calendar.

Google Trends' average interest is unknown for the future, therefore it's only used as a lagging indicator for products and locations that demonstrated such dependency.

When training the model, there are 3 key variables to adjust:

Google Cloud Cortex Framework



1. **Forecast horizon** (13 weeks by default) - how far in the future you want to make predictions, up to 13 weeks. While a longer horizon gives a great opportunity to make a forecast in advance, it reduces the overall quality of the model. Depending on your supply chain, manufacturing and delivery timing, you may want to reduce the horizon.
2. **Context window** (52 weeks by default) sets how far back the model looks during training (and for forecasts). In other words, for each training datapoint, the context window determines how far back the model looks for predictive patterns. Increasing the context window has the following effects:
  - a. Increases the required training time
  - b. With a larger context window, the model uses more data points in training, causing the training time to increase
  - c. Increases the required amount of history for prediction data. Your prediction data should provide at least as many historical data points as the value of the context window.

With enough historical data, you can set the context window up to 5 times the size of the forecast horizon.

If you expect to have a lot of prediction data that does not extend into the past (cold starts), begin by setting the context window to 0. Otherwise, a context window between the size of the forecast horizon and 10 times the size of the forecast horizon should work well.

3. **Training time budget** (1 hour by default) - how much Vertex AI AutoML will spend actually training the model. 1 hour is the minimum time. If the model quality is not great ( $r^2$  metric below 0.85), you may want to reduce the context window in half, and increase the training time up to 6 hours. If the evaluation metrics show substantial improvement, train the model again, increasing the context window to 5 times the size of the forecast horizon. Consider making a proportional increase to the training budget (if you trained for 6 hours, increase the training budget to 30 hours).

**AugmentedWeeklySales** view in the Reporting dataset is used as training data for the forecasting model. It contains weekly sales per product and customer location augmented with Weather, Promotion and Trends.

Row	WeekStart	CatalogItemid	Customerid	Location	City	SalesQuantity	IsPromo	IsHoliday	AvgMaxTemp	AvgMinTemp	MaxTemp	MinTemp	AvgInterest
1	2017-01-02	C0000025	1000054	US 02108	US MA Boston	535	1	0	-0.33570818219863796	-4.3071350097656023	6.4500061035156477	-13.549993896484352	0.0
2	2017-01-02	C0000034	1000054	US 02108	US MA Boston	307	1	0	-0.33570818219863796	-4.3071350097656023	6.4500061035156477	-13.549993896484352	0.0
3	2017-01-02	C0000038	1000054	US 02108	US MA Boston	699	1	0	-0.33570818219863796	-4.3071350097656023	6.4500061035156477	-13.549993896484352	0.0
4	2017-01-02	C0000053	1000054	US 02108	US MA Boston	423	1	0	-0.33570818219863796	-4.3071350097656023	6.4500061035156477	-13.549993896484352	0.0
5	2017-01-02	C0000074	1000054	US 02108	US MA Boston	299	1	0	-0.33570818219863796	-4.3071350097656023	6.4500061035156477	-13.549993896484352	0.0
6	2017-01-02	C0000025	1000055	US 30308	US GA Atlanta	616	1	0	7.7500069754464507	1.8642996651785941	16.050012207031273	-10.849981689453102	0.0
7	2017-01-02	C0000034	1000055	US 30308	US GA Atlanta	314	1	0	7.7500069754464507	1.8642996651785941	16.050012207031273	-10.849981689453102	0.0
8	2017-01-02	C0000038	1000055	US 30308	US GA Atlanta	710	1	0	7.7500069754464507	1.8642996651785941	16.050012207031273	-10.849981689453102	0.0
9	2017-01-02	C0000053	1000055	US 30308	US GA Atlanta	441	1	0	7.7500069754464507	1.8642996651785941	16.050012207031273	-10.849981689453102	0.0
10	2017-01-02	C0000074	1000055	US 30308	US GA Atlanta	305	1	0	7.7500069754464507	1.8642996651785941	16.050012207031273	-10.849981689453102	0.0
11	2017-01-09	C0000025	1000054	US 02108	US MA Boston	473	1	0	2.0357169015067189	-4.5499938964843523	13.050012207031273	-15.649999999999977	0.0
12	2017-01-09	C0000034	1000054	US 02108	US MA Boston	262	1	0	2.0357169015067189	-4.5499938964843523	13.050012207031273	-15.649999999999977	0.0
13	2017-01-09	C0000038	1000054	US 02108	US MA Boston	582	1	0	2.0357169015067189	-4.5499938964843523	13.050012207031273	-15.649999999999977	0.0
14	2017-01-09	C0000053	1000054	US 02108	US MA Boston	349	1	0	2.0357169015067189	-4.5499938964843523	13.050012207031273	-15.649999999999977	0.0
15	2017-01-09	C0000074	1000054	US 02108	US MA Boston	272	1	0	2.0357169015067189	-4.5499938964843523	13.050012207031273	-15.649999999999977	0.0
16	2017-01-09	C0000025	1000055	US 30308	US GA Atlanta	591	1	0	14.67857840401788	9.521439034598238	19.950006103515648	-4.3499816894531023	0.0
17	2017-01-09	C0000034	1000055	US 30308	US GA Atlanta	324	1	0	14.67857840401788	9.521439034598238	19.950006103515648	-4.3499816894531023	0.0
18	2017-01-09	C0000038	1000055	US 30308	US GA Atlanta	669	1	0	14.67857840401788	9.521439034598238	19.950006103515648	-4.3499816894531023	0.0
19	2017-01-09	C0000053	1000055	US 30308	US GA Atlanta	384	1	0	14.67857840401788	9.521439034598238	19.950006103515648	-4.3499816894531023	0.0
20	2017-01-09	C0000074	1000055	US 30308	US GA Atlanta	259	1	0	14.67857840401788	9.521439034598238	19.950006103515648	-4.3499816894531023	0.0
21	2017-01-16	C0000025	1000054	US 02108	US MA Boston	499	0	0	3.9500061035156473	0.75000697544645134	9.85000000000000227	-4.3499816894531023	0.0

For more information, please [read the official documentation](#).

## 4.1. How to Train the model

There is a copy of Demand Sensing code in your Cloud Storage bucket (**Storage Bucket for Vertex AI**). The **README.md** contains the documentation on using Demand Sensing source code directly for training and scoring the forecasting model in Vertex AI.

We recommend re-training your forecasting model every few weeks, once you get more historical sales data, updated holiday calendars, refined weather forecasts, etc.

We recommend launching the pipelines from [Google Cloud Shell](#) in your source GCP project.

1. Download Demand Sensing code from Cloud Storage

```
mkdir -p cortex_demand_sensing & cd &\ngcloud storage cp -r gs://PIPELINE_BUCKET/cortex_demand_sensing_1_0 .
```

**PIPELINE\_BUCKET** - Google Cloud Storage Bucket to use with Vertex AI (**Storage Bucket for Vertex AI**).

2. Run `python3 -m pip install --upgrade -r requirements.txt`
3. Run `src/submit_pipeline.py` script as `python3 src/submit_pipeline.py` with the parameters described below.

```
python3 src/submit_pipeline.py [-h] [--debug] \  
  --model-command {train,score,train_score} \  
  --region REGION \  
  --bigquery-location LOCATION \  
  --pipeline-bucket PIPELINE_BUCKET \  
  --k9-processing-dataset K9_PROCESSING_DATASET_NAME \  
  --vertex-ai-sa VERTEX_AI_SA \  
  --source-project SOURCE_PROJECT --target-project TARGET_PROJECT \  
  --target-reporting-dataset TARGET_REPORTING_DATASET \  
  --cdc-processed-dataset CDC_PROCESSED_DATASET \  
  --forecast-horizon-weeks FORECAST_HORIZON_WEEKS \  
  [--context-window-weeks CONTEXT_WINDOW_WEEKS] \  
  [--training-node-hours TRAINING_NODE_HOURS] \  
  [--score-on-recent-model]
```

#### Parameters:

- `-h, --help` - displays command line help.
- `--debug` - Debug Logging mode if specified.
- `--model-command` - Forecasting Model command, `train`, `score` or `train_score`. For training, use `train` or `train_score` train which performs both training and scoring.
- `--region` - Vertex AI GCP region.
- `--bigquery-location` - BigQuery Location, one you used with Data Foundation,
- `--pipeline-bucket` - Google Cloud Storage Bucket to use with Vertex AI when needed.
- `--k9-processing-dataset` - Data Foundation K9 Processing Dataset.
- `--vertex-ai-sa` - Service Account for Vertex AI to use. Make sure you use the principal of the account, not its name.
- `--source-project` - Source Data GCP project (CDC Processed dataset should be there).
- `--target-project` - Target GCP project (Target Reporting dataset should be there).
- `--target-reporting-dataset` - Target Reporting dataset name.
- `--cdc-processed-dataset` - CDC Processed dataset name.
- `--forecast-horizon-weeks` - Vertex AI Forecasting Horizon.
- `--context-window-weeks` - Vertex AI Forecasting Context Window in weeks. Optional, defaults to 52.
- `--training-node-hours` - Vertex AI Forecasting Training budget in hours. Optional, defaults to 1.
- `--score-on-recent-model` - if present, makes the pipeline use the most recent model, not the best one.

## 4.2. Validate ML Model

When running the training pipeline, once **automl-forecasting-training-job** step is done, it produces a model which you can find on [Vertex AI Models page](#) of your source project. Look for the most recent one named “**Cortex Demand Sensing\_...**” with “**best\_r\_squared**” label. By clicking on the model name, then on its version number (1), you will navigate to the evaluation page with model metrics.

<a href="#">←</a> Cortex Demand Sensing_2022-08-09T04.35.41.719902 > Version 1 <a href="#">VIEW DATASET</a>					
<a href="#">EVALUATE</a> BATCH PREDICT VERSION DETAILS					
Target column SalesQuantity numeric	MAE <a href="#">?</a> 28.19	MAPE <a href="#">?</a> 6.989	RMSE <a href="#">?</a> 39.519	RMSLE <a href="#">?</a> 0.087	r <sup>2</sup> <a href="#">?</a> 0.939
Target Quantile 0.10000000149011612		Observed Quantile <a href="#">?</a> 0.16117216117216118		Scaled Pinball Loss <a href="#">?</a> 5.57	
Target Quantile 0.5		Observed Quantile <a href="#">?</a> 0.6146520146520147		Scaled Pinball Loss <a href="#">?</a> 14.095	
Target Quantile 0.8999999761581421		Observed Quantile <a href="#">?</a> 0.9029304029304029		Scaled Pinball Loss <a href="#">?</a> 7	

After the training, Vertex AI forecasts also populates an evaluations table in K9 Processing dataset in source project's BigQuery. Look for a table starting with "demand\_sensing\_evaluations\_" followed by a timestamp. Table with the most recent timestamp is the one you are looking for.

 demand_sensing_evaluations_2022_06_03T22_39_49_36...
 demand_sensing_evaluations_2022_06_05T05_12_51_99...
 demand_sensing_evaluations_2022_06_05T07_02_39_94...
 demand_sensing_evaluations_2022_06_05T08_46_12_92...

The Forecasting pipeline performs validation for the training data. If any errors are discovered, you will find them in tables starting with "errors\_validation\_" + timestamp..

**errors\_SalesQuantity.message** contains verbal explanations of errors.

## 4.3. Produce ML Forecast

Every week, or whenever your future data changes (Promotion Calendar, Weather), you can run scriping of the forecasting model, and produce a new Demand Forecast.

Do so by running **src/submit\_pipeline.py** script as described in the training section above, except with `--model-command` equal to **score** (or **train\_score** if you want to retrain the model first).

- **The forecasting model must be trained before running the scoring.**
- **The forecast is always made using the best Demand Sensing model, not the last one, unless using `--score-on-recent-model` option.**

The model performance and criteria for the best model are evaluated based on **r-squared** metric ( $R^2$ , [Coefficient of determination](#)). If you'd like to change this criteria, modify **evaluate\_model** Kubeflow component in **src/models.py** file.

Demand Plan defines what dates, products and customers the forecast will be made for. So, the **Demand\_Plan** table in **CDC\_PROCESSED** dataset must be carefully maintained for producing accurate forecasts.

**AugmentedDemandPlan** view in the Reporting dataset is used as the actual Demand Plan data for scoring the model. Similarly to AugmentedWeeklySales, it contains demand plan data augmented with Promotions, Holidays and Weather forecasts.

Row	WeekStart	CustomerId	CatalogItemId	DemandPlan	Location	Country	City	IsPromo	IsHoliday	AvgMaxTemp	AvgMinTemp	MaxTemp	MinTemp
18	2022-03-28	1000054	C0000034	350.0	US 02108	US	US MA Boston	0	0	8.8339782714843977	1.5680175781250223	14.956475830078148	-6.0829528808593523
19	2022-03-28	1000054	C0000074	303.0	US 02108	US	US MA Boston	0	0	8.8339782714843977	1.5680175781250223	14.956475830078148	-6.0829528808593523
20	2022-03-28	1000054	C0000038	896.0	US 02108	US	US MA Boston	0	0	8.8339782714843977	1.5680175781250223	14.956475830078148	-6.0829528808593523
21	2022-04-04	1000054	C0000038	761.0	US 02108	US	US MA Boston	0	0	11.923830740792432	6.2438685825893083	14.885522460937523	3.0663391113281477
22	2022-04-04	1000054	C0000053	408.0	US 02108	US	US MA Boston	0	0	11.923830740792432	6.2438685825893083	14.885522460937523	3.0663391113281477
23	2022-04-04	1000054	C0000025	558.0	US 02108	US	US MA Boston	0	0	11.923830740792432	6.2438685825893083	14.885522460937523	3.0663391113281477
24	2022-04-04	1000054	C0000034	328.0	US 02108	US	US MA Boston	0	0	11.923830740792432	6.2438685825893083	14.885522460937523	3.0663391113281477
25	2022-04-04	1000054	C0000074	330.0	US 02108	US	US MA Boston	0	0	11.923830740792432	6.2438685825893083	14.885522460937523	3.0663391113281477
26	2022-04-11	1000054	C0000034	329.0	US 02108	US	US MA Boston	0	0	15.880234200613861	8.38860037667413	18.775109863281273	5.2499938964843977
27	2022-04-11	1000054	C0000053	426.0	US 02108	US	US MA Boston	0	0	15.880234200613861	8.38860037667413	18.775109863281273	5.2499938964843977
28	2022-04-11	1000054	C0000038	770.0	US 02108	US	US MA Boston	0	0	15.880234200613861	8.38860037667413	18.775109863281273	5.2499938964843977
29	2022-04-11	1000054	C0000025	510.0	US 02108	US	US MA Boston	0	0	15.880234200613861	8.38860037667413	18.775109863281273	5.2499938964843977
30	2022-04-11	1000054	C0000074	320.0	US 02108	US	US MA Boston	0	0	15.880234200613861	8.38860037667413	18.775109863281273	5.2499938964843977
31	2022-04-18	1000054	C0000074	310.0	US 02108	US	US MA Boston	0	0	12.638238525390648	8.57034127371654	16.480493164062523	4.8803344726562727
32	2022-04-18	1000054	C0000034	358.0	US 02108	US	US MA Boston	0	0	12.638238525390648	8.57034127371654	16.480493164062523	4.8803344726562727
33	2022-04-18	1000054	C0000025	481.0	US 02108	US	US MA Boston	0	0	12.638238525390648	8.57034127371654	16.480493164062523	4.8803344726562727
34	2022-04-18	1000054	C0000053	421.0	US 02108	US	US MA Boston	0	0	12.638238525390648	8.57034127371654	16.480493164062523	4.8803344726562727
35	2022-04-18	1000054	C0000038	803.0	US 02108	US	US MA Boston	0	0	12.638238525390648	8.57034127371654	16.480493164062523	4.8803344726562727
36	2022-04-25	1000054	C0000038	801.0	US 02108	US	US MA Boston	0	0	11.929171316964307	6.7305106026785948	17.369378662109398	4.6853271484375227
37	2022-04-25	1000054	C0000025	452.0	US 02108	US	US MA Boston	0	0	11.929171316964307	6.7305106026785948	17.369378662109398	4.6853271484375227
38	2022-04-25	1000054	C0000053	406.0	US 02108	US	US MA Boston	0	0	11.929171316964307	6.7305106026785948	17.369378662109398	4.6853271484375227

When the scoring part of the pipeline is completed, it's expected to fill the Demand\_Forecast table in the CDC\_PROCESSED dataset (one you specified in deploy.cdcProcessedDataset parameter) in the source GCP project.

**DemandForecast** view in the reporting dataset (target project) will have the same data.

<input type="checkbox"/>	Field name	Type
<input type="checkbox"/>	StartDateOfWeek	DATE
<input type="checkbox"/>	CatalogItemId	STRING
<input type="checkbox"/>	CustomerId	STRING
<input type="checkbox"/>	ForecastDate	DATE
<input type="checkbox"/>	ForecastQuantity	FLOAT
<input type="checkbox"/>	ForecastQuantityLowerBound	FLOAT
<input type="checkbox"/>	ForecastQuantityUpperBound	FLOAT

DemandForecast follows the same approach of weekly aggregation of expected sales (**ForecastQuantity**), per product (**CatalogItemId**) per Customer (**CustomerId**). The table also contains statistical range with lower (**ForecastQuantityLowerBound**, under-forecasting) and higher (**ForecastQuantityUpperBound**, over-forecasting) bounds, using respective [quantiles](#) 0.1 and 0.9.

**Important notes:**

1. **Demand\_Forecast** table is truncated every time a successful forecast has been made.



2. The forecast is always made using the best model, no matter which model was trained last. The quality of the model is evaluated by  $r^2$  (r-squared) metric. This is done by the “get-best-model” step of the pipeline.

If there were errors during the forecasting process, Vertex AI will create corresponding records in K9 Processing dataset in source project’s BigQuery. Look for a table starting with “errors\_” followed by a timestamp. Table with the most recent timestamp is the one you are looking for.

 demand_sensing_evaluations_2022_06_03T22_39_49_36...
 demand_sensing_evaluations_2022_06_05T05_12_51_99...
 demand_sensing_evaluations_2022_06_05T07_02_39_94...
 demand_sensing_evaluations_2022_06_05T08_46_12_92...
 errors_2022_06_04T23_10_44_915Z_809
 errors_2022_06_04T23_30_47_323Z_570
 errors_2022_06_05T00_01_49_126Z_825

**errors\_SalesQuantity.message** contains verbal explanations of errors.

errors....message
The time series has no values to predict. The time series has been excluded from predictions.

The Forecasting pipeline performs validation for the scoring data as well. If any errors are discovered, you will find them in tables starting with “**errors\_validation\_**” + timestamp (similarly to one at the training stage).

**errors\_SalesQuantity.message** contains verbal explanations of errors.

errors_SalesQuantity	vertex_timeseries_id
There are rows with non-empty target values after this row. The time series has been excluded from predictions.	C0000034__1000054

## 4.4. Logging, Monitoring and Servicing

### 4.4.1. Vertex AI Logging and Monitoring

While running a training or forecasting pipeline, navigate to [Vertex AI Pipeline page](#) in your source project.

The most recent pipeline run with the name starting with “demand-sensing-pipeline-” is expected to be running or completed. By clicking on the run’s name you can explore the running pipeline and monitor logs of every pipeline step.

When started by the Marketplace deployment or by running Cloud Build with *cloudbuild.ml.yaml* from the source code, the pipeline will perform both model training and scoring (forecasting).

You can navigate between different pipeline steps, monitor step logs, produced assets and results.

The screenshot displays the Vertex AI Pipeline interface for a pipeline named "demand-sensing-pipeline-20220620210530". The top bar includes buttons for "CLONE", "STOP", and "DELETE". Below the bar, the "Runtime Graph" shows the pipeline's execution flow. The graph includes steps such as "time-series-dataset-create", "make-random-bq-uri", "condition-scoring-only-4", "automi-forecasting-train...", "condition-training-only-3", and "evaluate-model". Each step is represented by a box with a status icon (green checkmark for completed, yellow lightbulb for running). The "evaluate-model" step is currently running. Below the graph, the "Logs" section is visible, showing a list of log entries with timestamps, severity levels, and messages. The logs include information about exported examples, resource names, and model objects.

demand-sensing-pipeline-20220620210530

CLONE STOP DELETE

Runtime Graph 38/38 steps completed Expand Artifacts 100%

time-series-dataset-create gcr.io/\_line-components:1.0.9

make-random-bq-uri python:3.9

condition-scoring-only-4

automi-forecasting-train... gcr.io/\_line-components:1.0.9

condition-training-only-3

evaluate-model python:3.9

Logs

MAIN JOB

Logs Showing 100 log entries Severity Default Filter Filter logs

2022-06-20T21:52:03.636929058Z Exported examples available at:

2022-06-20T21:52:03.637047293Z INFO:google.cloud.aiplatform.training\_jobs:Exported examples available at:

2022-06-20T21:52:03.637336586Z bq://source-project.DS\_VERTEX\_AI.demand\_sensing\_evaluations\_2022\_06\_20T21\_07\_13\_728744

2022-06-20T21:52:03.637488872Z resource\_name: %s projects/34534534534/locations/us-central1/trainingPipelines/5835262152789721088

2022-06-20T21:52:03.637957856Z bq://source-project.DS\_VERTEX\_AI.demand\_sensing\_evaluations\_2022\_06\_20T21\_07\_13\_728744

2022-06-20T21:52:04.373235699Z <google.cloud.aiplatform.models.Model object at 0x7f5663716410>

2022-06-20T21:52:04.373300572Z resource name: projects/34534534534/locations/us-central1/models/8449865606714359808

2022-06-20T21:52:27.275561877Z Tearing down training program.

### 4.4.2. Vertex AI Managed Datasets and BigQuery Datasets servicing

While running training and scoring pipelines, Vertex AI produces a number of intermediate tables. They all reside in BigQuery, in the K9 Processing dataset.

Google Cloud Cortex Framework

Over time, the number of tables grows, and you may want to plan a periodic clean up.

Every training pipeline run produces a “**demand\_sensing\_evaluations\_...**” table and a “**preprocess\_...**” table. The evaluations table can be safely deleted unless you would like to explore the results of the model evaluation.

The “**preprocess\_...**” table is used as the data source for a [Vertex AI Managed Dataset](#) that Vertex AI needs for training the forecasting model. **Do not delete this table** while using the Demand Sensing model with a respective Vertex AI managed dataset. To discover what table is actually used by the model:

1. Go to [Vertex AI Models page](#) of your source project.
2. Look for a model named “**Cortex Demand Sensing\_...**” with the “**best\_r\_squared**” label. By clicking on the model name, then on its version number (1), you will navigate to the evaluation page with model metrics.
3. Click on “View Dataset”

 **VIEW DATASET**

4. The dataset page has the “**Dataset location(s)**” attribute which is a BigQuery table you need to keep.

<b>Dataset location(s)</b>	<a href="#">bq://source-proj..._20T21_08_51_963Z</a>  
----------------------------	--

Every scoring run also produces a “**preprocess\_...**” table. It can be safely deleted after the scoring run.

You may want to automate the process of cleaning up the tables, making sure ones utilized by models in use are preserved. In general, any Demand Sensing model that doesn’t have a “**best\_r\_squared**” label, can be deleted along with the corresponding [Vertex AI managed dataset](#) and BigQuery “**preprocess\_...**” table.

If you use different criteria for keeping models, assigning another label is a good option. In such cases, replace **best\_r\_squared** label name with one you use.

The following code snippet illustrates the same process in Python using [Vertex AI ML Metadata](#) lineage:

```
from google.cloud import aiplatform
from google.cloud import aiplatform_v1
from google.cloud import bigquery

project = "<<SOURCE PROJECT>>" # Source project
location = "<<GCP LOCATION>>" # Vertex AI location, e.g. us-central1

# BigQuery client for deleting tables
```



```

bq_client = bigquery.Client(project=project, location=location)

aiplatform.init(project=project, location=location)
client_options = aiplatform.initializer.global_config.get_client_options()

# Vertex AI Metadata Service client
metadata_client = aiplatform.gapic.MetadataServiceClient(
    client_options=client_options
)

# Using default Metadata Store
store = f"projects/{project}/locations/{location}/metadataStores/default"

# Getting Demand Sensing models using a label value
# (see src/pipeline_constants.py)
models = aiplatform.Model.list(filter=f'labels.demand-sensing-model="1_0"')

for model in models:

    # If it's a good model, skip it
    # (good models have "best_r_squared" label, see src/pipeline_constants.py)
    # If using another criteria for a good model, replace best_r_squared name here
    if "best_r_squared" in model.labels:
        continue

    model_uri = (
        f"https://{client_options.api_endpoint}/v1/{model.resource_name}"
    )

    artifact_request = aiplatform_v1.ListArtifactsRequest(
        parent=store,
        filter=f'uri="{model_uri}" AND schema_title="google.VertexModel"',
    )
    artifacts = list(metadata_client.list_artifacts(artifact_request))
    artifact = artifacts[0]

    lineage = metadata_client.query_artifact_lineage_subgraph(
        artifact=artifact.name
    )
    lineage_artifacts = list(lineage.artifacts)

```

```

dataset_artifact = list(
    filter(
        lambda artifact: artifact.schema_title == "google.VertexDataset",
        lineage_artifacts,
    )
)[0]

dataset = aiplatform.TimeSeriesDataset(
    dataset_name=dataset_artifact.metadata["resourceName"]
)
bq_source = dataset.to_dict()["metadata"]["inputConfig"]["bigquerySource"][
    "uri"
]

print(
    f"Deleting Vertex AI Model: {model.resource_name}, Vertex AI ML Dataset:"
    f" {dataset.resource_name}, BigQuery Table: {bq_source}"
)

# delete Vertex AI model
model.delete()
# delete Vertex AI managed dataset
dataset.delete()

bq_table_ref = bigquery.Table.from_string(
    bq_source.replace("bq://", "")
).reference

# delete BigQuery table
bq_client.delete_table(bq_table_ref)

```

## 5. Configure Looker

Once the content is deployed, instructions for deploying the pre-built Looker block can be found [here](#). The source for the Demand Sensing block is located in this [repository](#). Optionally, you may also customize the block by forking the GitHub repositories into your own Looker project. Instructions can be found [here](#).

## 6. Feedback and updates

Once the templates are deployed they can be customized as necessary. BigQuery views will be dependent on customer-specific change data capture mechanisms.

We have designed all BigQuery model views and VertexAI models to provide flexibility of choice when it comes to choosing the set of tools that fit best (integration or visualization tools). This means that modifications can diverge from Google's original templates to better fit existing customer investments.

We are interested in capturing feedback on additional data models and features that would be relevant to your business. Please reach out to your account team or [contact us](#) for assistance. Bugs may also be raised in the Data Foundation repository [here](#).