Build a data agent using the Python SDK

This product or feature is subject to the "Pre-GA Offerings Terms" in the General Service Terms section of the Service Specific Terms (/terms/service-terms#1). Pre-GA products and features are available "as is" and might have limited support. For more information, see the launch stage descriptions (/products#product-launch-stages).

This page shows you how to use the Python SDK to make requests to the Conversational Analytics API (/gemini/docs/conversational-analytics-api/overview). The sample Python code demonstrates how to complete the following tasks:

- Authenticate and set up your environment (#authenticate_and_set_up_your_environment)
- Specify the billing project and system instructions (#specify_the_billing_project_and_system_instructions)
- Connect to a Looker, BigQuery, or Looker Studio data source (#connect_to_a_data_source)
- Set up context for stateful or stateless chat (#set_up_context_for_stateful_or_stateless_chat)
- Create a data agent (#create_a_data_agent)
- Create a conversation (#create_a_conversation)
- Manage data agents and conversations (#manage-data-agents-and-conversations)
- Use the API to ask questions (#ask-questions)
- Create a stateless multi-turn conversation (#create-a-stateless-multi-turn-conversation)
- Define helper functions (#define_helper_functions)

Tip: To run the code samples on this page in an interactive environment, see the Conversational Analytics API SDK Colaboratory notebook (https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/agents/gemini_data_analytics/intro_gemini_data_analytics_sdk.ipynb)
.

Authenticate and set up your environment

To use the Python SDK for the Conversational Analytics API, follow the instructions in the Conversational Analytics API SDK Colaboratory notebook (https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/agents/gemini_data_analytics/intro_gemini_data_analytics_sdk.ipynb) to download and install the SDK. Note that the download method and the contents of the SDK Colab are subject to change.

After you've completed the setup instructions in the notebook, you can use the following code to import the required SDK libraries, authenticate your Google Account within a Colaboratory environment, and initialize a client for making API requests:

```
from google.colab import auth
auth.authenticate_user()

from google.cloud import geminidataanalytics

data_agent_client = geminidataanalytics.DataAgentServiceClient()
data_chat_client = geminidataanalytics.DataChatServiceClient()
```

Specify the billing project and system instructions

The following sample Python code defines the billing project and system instructions that are used throughout your script:

```
# Billing project
billing_project = "my_project_name"

# System instructions
system_instruction = "Help the user analyze their data."
```

Replace the sample values as follows:

- ***my_project_name***: The ID of your billing project that has the required APIs enabled (/gemini/docs/conversational-analytics-api/enable-the-api).
- ***Help the user analyze their data.***: System instructions to guide the agent's behavior and customize it for your data needs. For example, you can use system instructions to define business terms, control response length, or set data formatting. Ideally, define system instructions by using the recommended YAML format in Write effective system instructions (/gemini/docs/conversational-analytics-api/data-agent-system-instructions) to provide detailed and structured guidance.

Connect to a data source

The following sections show how to define the connection details for your agent's data sources. Your agent can connect to data in Looker (#connect-to-looker-data), BigQuery (#connect-to-bigquery-data), or Looker Studio (#connect-to-looker-studio-data).

Connect to Looker data

The following code examples show how to define the details for a connection to a Looker Explore with either API keys or an access token. You can connect to only one Looker Explore at a time with the Conversational Analytics API.

Note: Don't include **`credentials`** in the data source during agent creation (#create_a_data_agent).

API keys (#api-keys) Access token (#access-token)

You can establish a connection with a Looker instance by using an access token, as described in Authenticate and connect to a data source with the Conversational Analytics API (/gemini/docs/conversational-analytics-api/authentication#looker-access-token).

```
looker_access_token = "my_access_token"
looker_instance_uri = "https://my_company.looker.com"
lookml_model = "my_model"
explore = "my_explore"
```

```
looker_explore_reference = geminidataanalytics.LookerExploreReference()
looker_explore_reference.looker_instance_uri = looker_instance_uri
looker_explore_reference.lookml_model = lookml_model
looker_explore_reference.explore = explore

credentials = geminidataanalytics.Credentials()
credentials.oauth.token.access_token = looker_access_token

datasource_references = geminidataanalytics.DatasourceReferences()
datasource_references.looker.explore_references = [looker_explore_referenc

# Do not include the following line during agent creation
datasource_references.credentials = credentials
```

Replace the sample values as follows:

- ***my_access_token***: The `access_token` value that you generate to authenticate to Looker.
- ***https://my_company.looker.com***: The complete URL of your Looker instance.
- ***my_model***: The name of the LookML model that includes the Explore that you want to connect to.
- ***my_explore***: The name of the Looker Explore that you want the data agent to query.

Connect to BigQuery data

With the Conversational Analytics API, there are no hard limits on the number of BigQuery tables that you can connect to. However, connecting to a large number of tables can reduce accuracy or cause you to exceed the model's input token limit.

Note: Queries that require complex joins across multiple tables might result in less accurate responses.

The following sample code defines a connection to multiple BigQuery tables.

Important: Make sure you have the necessary Identity and Access Management (IAM) permissions to query any BigQuery tables that you specify.

```
bigquery_table_reference = geminidataanalytics.BigQueryTableReference()
bigquery_table_reference.project_id = "my_project_id"
bigquery_table_reference.dataset_id = "my_dataset_id"
bigquery_table_reference.table_id = "my_table_id"

bigquery_table_reference_2 = geminidataanalytics.BigQueryTableReference()
bigquery_table_reference_2.project_id = "my_project_id_2"
bigquery_table_reference_2.dataset_id = "my_dataset_id_2"
bigquery_table_reference_2.table_id = "my_table_id_2"

bigquery_table_reference_3 = geminidataanalytics.BigQueryTableReference()
bigquery_table_reference_3.project_id = "my_project_id_3"
bigquery_table_reference_3.dataset_id = "my_dataset_id_3"
bigquery_table_reference_3.table_id = "my_table_id_3"

# Connect to your data source
datasource_references = geminidataanalytics.DatasourceReferences()
datasource_references.bq.table_references = [bigquery_table_reference, bigquery_
```

Replace the sample values as follows:

- ***my_project_id***: The ID of the Google Cloud project that contains the BigQuery dataset and table that you want to connect to. To connect to a public dataset (/bigquery/public-data), specify **`bigquery-public-data`**.
- ***my_dataset_id***: The ID of the BigQuery dataset. For example, **`san_francisco`**.
- ***my_table_id***: The ID of the BigQuery table. For example, **`street_trees`**.

Connect to Looker Studio data

The following sample code defines a connection to a Looker Studio data source.

```
studio_datasource_id = "my_datasource_id"

studio_references = geminidataanalytics.StudioDatasourceReference()
studio_references.datasource_id = studio_datasource_id

## Connect to your data source
```

```
datasource_references.studio.studio_references = [studio_references]
```

In the previous example, replace ***my_datasource_id*** with the data source ID.

Set up context for stateful or stateless chat

The Conversational Analytics API supports multi-turn conversations, which let users ask follow-up questions that build on previous context. The following sample Python code demonstrates how to set up context for either *stateful* or *stateless* chat:

- **Stateful chat**: Google Cloud stores and manages the conversation history. Stateful chat is inherently multi-turn, as the API retains context from previous messages. You only need to send the current message for each turn.
- **Stateless chat**: Your application manages the conversation history. You must include the entire conversation history with each new message. For detailed examples on how to manage multi-turn conversations in stateless mode, see Create a stateless multi-turn conversation (#create-a-stateless-multi-turn-conversation).

Note: Whether you set up **`published_context`** for stateful chat or **`inline_context`** for stateless chat, you use the **`system_instruction`** variable that you defined in Specify the billing project and system instructions (#specify_the_billing_project_and_system_instructions) to effectively guide the agent's responses. For details on the recommended YAML format for these instructions, see Write effective system instructions (/gemini/docs/conversational-analytics-api/data-agent-system-instructions).

Stateful chatStateless chat...
(#stateful-chat)

The following code sample sets up context for stateful chat, where Google Cloud stores and manages the conversation history. You can also optionally enable advanced analysis with Python by including the line **`published_context.options.analysis.python.enabled = True`** in the following sample code.

```
# Set up context for stateful chat
published_context = geminidataanalytics.Context()
published_context.system_instruction = system_instruction
published_context.datasource_references = datasource_references
# Optional: To enable advanced analysis with Python, include the following
published_context.options.analysis.python.enabled = True
```

Create a data agent

The following sample Python code makes an API request to create a data agent, which you can then use to have a conversation about your data. The data agent is configured with the specified data source, system instructions, and context.

```
data_agent_id = "data_agent_1"

data_agent = geminidataanalytics.DataAgent()
data_agent.data_analytics_agent.published_context = published_context
data_agent.name = f"projects/{billing_project}/locations/global/dataAgents/{data

request = geminidataanalytics.CreateDataAgentRequest(
    parent=f"projects/{billing_project}/locations/global",
    data_agent_id=data_agent_id, # Optional
    data_agent=data_agent,
)

try:
    data_agent_client.create_data_agent(request=request)
    print("Data Agent created")
except Exception as e:
    print(f"Error creating Data Agent: {e}")
```

In the previous example, replace the value ***data_agent_1*** with a unique identifier for the data agent.

Create a conversation

The following sample Python code makes an API request to create a conversation.

```
# Initialize request arguments
data_agent_id = "data_agent_1"
conversation_id = "conversation_1"

conversation = geminidataanalytics.Conversation()
conversation.agents = [f'projects/{billing_project}/locations/global/dataAgents/
conversation.name = f"projects/{billing_project}/locations/global/conversations/

request = geminidataanalytics.CreateConversationRequest(
    parent=f"projects/{billing_project}/locations/global",
    conversation_id=conversation_id,
    conversation=conversation,
)

# Make the request
response = data_chat_client.create_conversation(request=request)

# Handle the response
print(response)
```

Replace the sample values as follows:

- ***data_agent_1***: The ID of the data agent, as defined in the sample code block in Create a data agent (#create_a_data_agent).
- ***conversation_1***: A unique identifier for the conversation.

Manage data agents and conversations

The following code samples show how to manage your data agents and conversations by using the Conversational Analytics API. You can perform the following tasks:

- Get a data agent (#get-data-agent)

Get a data agent

The following sample Python code demonstrates how to make an API request to retrieve a data agent that you previously created.

```
# Initialize request arguments
data_agent_id = "data_agent_1"
request = geminidataanalytics.GetDataAgentRequest(
    name=f"projects/{billing_project}/locations/global/dataAgents/{data_agent_id
)

# Make the request
response = data_agent_client.get_data_agent(request=request)

# Handle the response
print(response)
```

In the previous example, replace the value ***data_agent_1*** with the unique identifier for the data agent that you want to retrieve.

List data agents

The following code demonstrates how to list all the data agents for a given project by calling the **`list_data_agents`** method. To list all agents, you must have the **`geminidataanalytics.dataAgents.list`** permission on the project. For more information on which IAM roles include this permission, see the list of predefined roles (/gemini/docs/conversational-analytics-api/access-control#predefined-roles).

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
request = geminidataanalytics.ListDataAgentsRequest(
    parent=f"projects/{billing_project}/locations/global",
)

# Make the request
page_result = data_agent_client.list_data_agents(request=request)

# Handle the response
for response in page_result:
    print(response)
```

Replace ***YOUR-BILLING-PROJECT*** with the ID of your billing project.

List accessible data agents

The following code demonstrates how to list all the accessible data agents for a given project by calling the **`list_accessible_data_agents`** method.

```
billing_project = "YOUR-BILLING-PROJECT"
creator_filter = "YOUR-CREATOR-FILTER"
location = "global"
request = geminidataanalytics.ListAccessibleDataAgentsRequest(
    parent=f"projects/{billing_project}/locations/global",
    creator_filter=creator_filter
)

# Make the request
page_result = data_agent_client.list_accessible_data_agents(request=request)

# Handle the response
```

```
for response in page_result:
    print(response)
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***YOUR-CREATOR-FILTER***: The filter to apply based on the creator of the data agent. Possible values include **`NONE`** (default), **`CREATOR_ONLY`**, and **`NOT_CREATOR_ONLY`**.

Update a data agent

The following sample code demonstrates how to update a data agent by calling the **`update_data_agent`** method on the data agent resource. The request requires a **`DataAgent`** object that includes the new values for the fields that you want to change, and an **`update_mask`** parameter that takes a **`FieldMask`** object to specify which fields to update.

To update a data agent, you must have the **`geminidataanalytics.dataAgents.update`** IAM permission on the agent. For more information on which IAM roles include this permission, see the list of predefined roles (/gemini/docs/conversational-analytics-api/access-control#predefined-roles)
.

```
data_agent_id = "data_agent_1"
billing_project = "YOUR-BILLING-PROJECT"
data_agent = geminidataanalytics.DataAgent()
data_agent.data_analytics_agent.published_context = published_context
data_agent.name = f"projects/{billing_project}/locations/global/dataAgents/{data
data_agent.description = "Updated description of the data agent."

update_mask = field_mask_pb2.FieldMask(paths=['description', 'data_analytics_age

request = geminidataanalytics.UpdateDataAgentRequest(
    data_agent=data_agent,
    update_mask=update_mask,
)

try:
    # Make the request
    data_agent_client.update_data_agent(request=request)
    print("Data Agent Updated")
```

```
except Exception as e:
    print(f"Error updating Data Agent: {e}")
```

Replace the sample values as follows:

- ***data_agent_1***: The ID of the data agent that you want to update.
- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***Updated description of the data agent.***: A description of the updated data agent.

Set the IAM policy for a data agent

To share an agent, you can use the **set_iam_policy** method to assign IAM roles to users on a specific agent. The request includes bindings that specify which roles should be assigned to which users.

Important: This API call overrides existing permissions for the resource. To preserve existing policies, call the **get_iam_policy** method (#get-iam-policy-for-data-agent) to fetch the existing policy, and then pass the existing policy along with any additional changes in the call to **set_iam_policy**.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
data_agent_id = "data_agent_1"
role = "roles/geminidataanalytics.dataAgentEditor"
users = "222larabrown@gmail.com, cloudysanfrancisco@gmail.com"

resource = f"projects/{billing_project}/locations/global/dataAgents/{data_agent_

# Construct the IAM policy
binding = policy_pb2.Binding(
    role=role,
    members= [f"user:{i.strip()}" for i in users.split(",")]
)

policy = policy_pb2.Policy(bindings=[binding])

# Create the request
request = iam_policy_pb2.SetIamPolicyRequest(
    resource=resource,
    policy=policy
)

# Send the request
try:
    response = data_agent_client.set_iam_policy(request=request)
    print("IAM Policy set successfully!")
    print(f"Response: {response}")
except Exception as e:
    print(f"Error setting IAM policy: {e}")
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***data_agent_1***: The ID of the data agent for which you want to set the IAM policy.
- ***222larabrown@gmail.com, cloudysanfrancisco@gmail.com***: A comma-separated list of user emails to which you want to grant the specified role.

Get the IAM policy for a data agent

The following sample code demonstrates how to use the **`get_iam_policy`** method to fetch the IAM policy for a data agent. The request specifies the data agent resource path.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
data_agent_id = "data_agent_1"

resource = f"projects/{billing_project}/locations/global/dataAgents/{data_agent_
request = iam_policy_pb2.GetIamPolicyRequest(
            resource=resource,
        )
try:
    response = data_agent_client.get_iam_policy(request=request)
    print("IAM Policy fetched successfully!")
    print(f"Response: {response}")
except Exception as e:
    print(f"Error setting IAM policy: {e}")
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***data_agent_1***: The ID of the data agent for which you want to get the IAM policy.

Delete a data agent

The following sample code demonstrates how to use the **`delete_data_agent`** method to soft delete a data agent. When you soft delete an agent, that agent is deleted but can still be retrieved within 30 days. The request specifies the data agent resource URL.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
data_agent_id = "data_agent_1"

request = geminidataanalytics.DeleteDataAgentRequest(
    name=f"projects/{billing_project}/locations/global/dataAgents/{data_agent_id
)

try:
    # Make the request
    data_agent_client.delete_data_agent(request=request)
    print("Data Agent Deleted")
except Exception as e:
    print(f"Error deleting Data Agent: {e}")
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***data_agent_1***: The ID of the data agent that you want to delete.

Get a conversation

The following sample code demonstrates how to use the `get_conversation` method to fetch information about an existing conversation. The request specifies the conversation resource path.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
conversation_id = "conversation_1"

request = geminidataanalytics.GetConversationRequest(
    name = f"projects/{billing_project}/locations/global/conversations/{conversa
)

# Make the request
response = data_chat_client.get_conversation(request=request)

# Handle the response
print(response)
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***conversation_1***: The ID of the conversation that you want to fetch.

List conversations

The following sample code demonstrates how to list conversations for a given project by calling the `list_conversations` method. The request specifies the parent resource URL, which is the project and location (for example, `projects/my-project/locations/global`).

By default, this method returns the conversations that you created. Admins (users with the `cloudaicompanion.topicAdmin` IAM role (/iam/docs/roles-permissions/cloudaicompanion#cloudaicompanion.topicAdmin)) can see all conversations within the project.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"
```

```
request = geminidataanalytics.ListConversationsRequest(
    parent=f"projects/{billing_project}/locations/global",
)

# Make the request
response = data_chat_client.list_conversations(request=request)

# Handle the response
print(response)
```

Replace ***YOUR-BILLING-PROJECT*** with the ID of the billing project where you've enabled the required APIs.

List messages in a conversation

The following sample code demonstrates how to use the `list_messages` method to fetch all the messages in a conversation. The request specifies the conversation resource path.

To list messages, you must have the `cloudaicompanion.topics.get` permission (/iam/docs/roles-permissions/cloudaicompanion#gemini-for-google-cloud-api-roles) on the conversation.

Note: This list operation also supports pagination.

```
billing_project = "YOUR-BILLING-PROJECT"
location = "global"

conversation_id = "conversation_1"

request = geminidataanalytics.ListMessagesRequest(
    parent=f"projects/{billing_project}/locations/global/conversations/{conversa
)

# Make the request
response = data_chat_client.list_messages(request=request)

# Handle the response
```

```
print(response)
```

Replace the sample values as follows:

- ***YOUR-BILLING-PROJECT***: The ID of your billing project.
- ***conversation_1***: The ID of the conversation for which you want to list messages.

Use the API to ask questions

After you create a data agent (#create_a_data_agent) and a conversation (#create_a_conversation), the following sample Python code sends a query to the agent. The code uses the context that you set up for stateful or stateless chat (#set_up_context_for_stateful_or_stateless_chat). The API returns a stream of messages that represent the steps that the agent takes to answer the query.

Stateful chatStateless chat...
(#stateful-chat)

Send a stateful chat request with a `Conversation` reference

You can send a stateful chat request to the data agent by referencing a `Conversation` resource that you previously created.

Note: If you're using a Looker data source, uncomment the line **`conversation_reference.data_agent_context.credentials = credentials`** in the following sample code.

```
# Create a request that contains a single user message (your question)
question = "Which species of tree is most prevalent?"
messages = [geminidataanalytics.Message()]
messages[0].user_message.text = question

data_agent_id = "data_agent_1"
conversation_id = "conversation_1"

# Create a conversation_reference
```

```
conversation_reference = geminidataanalytics.ConversationReference()
conversation_reference.conversation = f"projects/{billing_project}/locatio
conversation_reference.data_agent_context.data_agent = f"projects/{billing
# conversation_reference.data_agent_context.credentials = credentials

# Form the request
request = geminidataanalytics.ChatRequest(
    parent = f"projects/{billing_project}/locations/global",
    messages = messages,
    conversation_reference = conversation_reference
)

# Make the request
stream = data_chat_client.chat(request=request)

# Handle the response
for response in stream:
    show_message(response)
```

Replace the sample values as follows:

- ***Which species of tree is most prevalent?***: A natural language question to send to the data agent.
- ***data_agent_1***: The unique identifier for the data agent, as defined in Create a data agent (#create_a_data_agent).
- ***conversation_1***: The unique identifier for the conversation, as defined in Create a conversation (#create_a_conversation).

Create a stateless multi-turn conversation

To ask follow-up questions in a stateless conversation, your application must manage the conversation's context by sending the entire message history with each new request. The following example shows how to create a multi-turn conversation by referencing a data agent or by using inline context to provide the data source directly.

The following code sample uses data agent context. To use inline context instead, uncomment the line **`inline_context=inline_context`**.

If your agent uses a Looker data source and you're using data agent context, uncomment the line **`data_agent_context.credentials = credentials`**.

```
# List that is used to track previous turns and is reused across requests
conversation_messages = []

data_agent_id = "data_agent_1"

# Use data agent context
data_agent_context = geminidataanalytics.DataAgentContext()
data_agent_context.data_agent = f"projects/{billing_project}/locations/global/da
# data_agent_context.credentials = credentials

# Helper function for calling the API
def multi_turn_Conversation(msg):

    message = geminidataanalytics.Message()
    message.user_message.text = msg

    # Send a multi-turn request by including previous turns and the new message
    conversation_messages.append(message)

    request = geminidataanalytics.ChatRequest(
        parent=f"projects/{billing_project}/locations/global",
        messages=conversation_messages,
        # Use data agent context
        data_agent_context=data_agent_context,
        # Use inline context
        # inline_context=inline_context,
    )

    # Make the request
    stream = data_chat_client.chat(request=request)

    # Handle the response
    for response in stream:
      show_message(response)
      conversation_messages.append(response)

# Send the first turn request
multi_turn_Conversation("Which species of tree is most prevalent?")
```

```
# Send follow-up turn request
multi_turn_Conversation("Can you show me the results as a bar chart?")
```

In the previous example, replace the sample values as follows:

- ***data_agent_1***: The unique identifier for the data agent, as defined in the sample code block in Create a data agent (#create_a_data_agent).
- ***Which species of tree is most prevalent?***: A natural language question to send to the data agent.
- ***Can you show me the results as a bar chart?***: A follow-up question that builds on or refines the previous question.

Define helper functions

The following sample code contains helper function definitions that are used in the previous code samples. These functions help to parse the response from the API and display the results.

```
from pygments import highlight, lexers, formatters
import pandas as pd
import requests
import json as json_lib
import altair as alt
import IPython
from IPython.display import display, HTML

import proto
from google.protobuf.json_format import MessageToDict, MessageToJson

def handle_text_response(resp):
  parts = getattr(resp, 'parts')
  print(''.join(parts))

def display_schema(data):
  fields = getattr(data, 'fields')
  df = pd.DataFrame({
    "Column": map(lambda field: getattr(field, 'name'), fields),
    "Type": map(lambda field: getattr(field, 'type'), fields),
    "Description": map(lambda field: getattr(field, 'description', '-'), fields)
    "Mode": map(lambda field: getattr(field, 'mode'), fields)
  })
  display(df)

def display_section_title(text):
  display(HTML('<h2>{}</h2>'.format(text)))

def format_looker_table_ref(table_ref):
 return 'lookmlModel: {}, explore: {}, lookerInstanceUri: {}'.format(table_ref.l

def format_bq_table_ref(table_ref):
  return '{}.{}.{}'.format(table_ref.project_id, table_ref.dataset_id, table_ref

def display_datasource(datasource):
  source_name = ''
  if 'studio_datasource_id' in datasource:
   source_name = getattr(datasource, 'studio_datasource_id')
  elif 'looker_explore_reference' in datasource:
   source_name = format_looker_table_ref(getattr(datasource, 'looker_explore_ref
  else:
    source_name = format_bq_table_ref(getattr(datasource, 'bigquery_table_refere

  print(source_name)
  display_schema(datasource.schema)

def handle_schema_response(resp):
  if 'query' in resp:
    print(resp.query.question)
  elif 'result' in resp:
    display_section_title('Schema resolved')
    print('Data sources:')
    for datasource in resp.result.datasources:
      display_datasource(datasource)

def handle_data_response(resp):
  if 'query' in resp:
    query = resp.query
    display_section_title('Retrieval query')
    print('Query name: {}'.format(query.name))
    print('Question: {}'.format(query.question))
    print('Data sources:')
    for datasource in query.datasources:
      display_datasource(datasource)
  elif 'generated_sql' in resp:
    display_section_title('SQL generated')
    print(resp.generated_sql)
  elif 'result' in resp:
    display_section_title('Data retrieved')

    fields = [field.name for field in resp.result.schema.fields]
    d = {}
    for el in resp.result.data:
      for field in fields:
        if field in d:
          d[field].append(el[field])
        else:
          d[field] = [el[field]]

    display(pd.DataFrame(d))

def handle_chart_response(resp):
  def _value_to_dict(v):
    if isinstance(v, proto.marshal.collections.maps.MapComposite):
      return _map_to_dict(v)
    elif isinstance(v, proto.marshal.collections.RepeatedComposite):
      return [_value_to_dict(el) for el in v]
    elif isinstance(v, (int, float, str, bool)):
      return v
    else:
      return MessageToDict(v)

  def _map_to_dict(d):
    out = {}
    for k in d:
      if isinstance(d[k], proto.marshal.collections.maps.MapComposite):
        out[k] = _map_to_dict(d[k])
      else:
        out[k] = _value_to_dict(d[k])
    return out

  if 'query' in resp:
    print(resp.query.instructions)
  elif 'result' in resp:
    vegaConfig = resp.result.vega_config
    vegaConfig_dict = _map_to_dict(vegaConfig)
    alt.Chart.from_json(json_lib.dumps(vegaConfig_dict)).display();

def show_message(msg):
```

```
  m = msg.system_message
  if 'text' in m:
    handle_text_response(getattr(m, 'text'))
  elif 'schema' in m:
    handle_schema_response(getattr(m, 'schema'))
  elif 'data' in m:
    handle_data_response(getattr(m, 'data'))
  elif 'chart' in m:
    handle_chart_response(getattr(m, 'chart'))
  print('\n')
```