Build a simple search application with Cortex Search

Introduction

This tutorial describes how to get started with Cortex Search for a simple search application.

What you will learn

- Create a Cortex Search Service from on an AirBnb listings dataset.
- Create a Streamlit in Snowflake app that lets you query your Cortex Search Service.

Prerequisites

The following prerequisites are required to complete this tutorial:

- You have a Snowflake account and user with a role that grants the necessary privileges to create a database, tables, virtual warehouse objects, Cortex Search services, and Streamlit apps.

Refer to the Snowflake in 20 minutes for instructions to meet these requirements.

Step 1: Setup

Getting the sample data

You will use a sample dataset hosted on Huggingface , downloaded as a single JSON file. Download the file directly from your browser by following this link:

- AirBnB listings dataset

Note
In a non-tutorial setting, you would bring your own data, possibly already in a Snowflake table.

Creating the database, tables, and warehouse

Execute the following statements to create a database and a virtual warehouse needed for this tutorial. After you complete the tutorial, you can drop these objects.

```
CREATE DATABASE IF NOT EXISTS cortex_search_tutorial_db;
```

```
CREATE OR REPLACE WAREHOUSE cortex_search_tutorial_wh WITH
```

```
    WAREHOUSE_SIZE='X-SMALL'
    AUTO_SUSPEND = 120
    AUTO_RESUME = TRUE
    INITIALLY_SUSPENDED=TRUE;
```

Note the following:

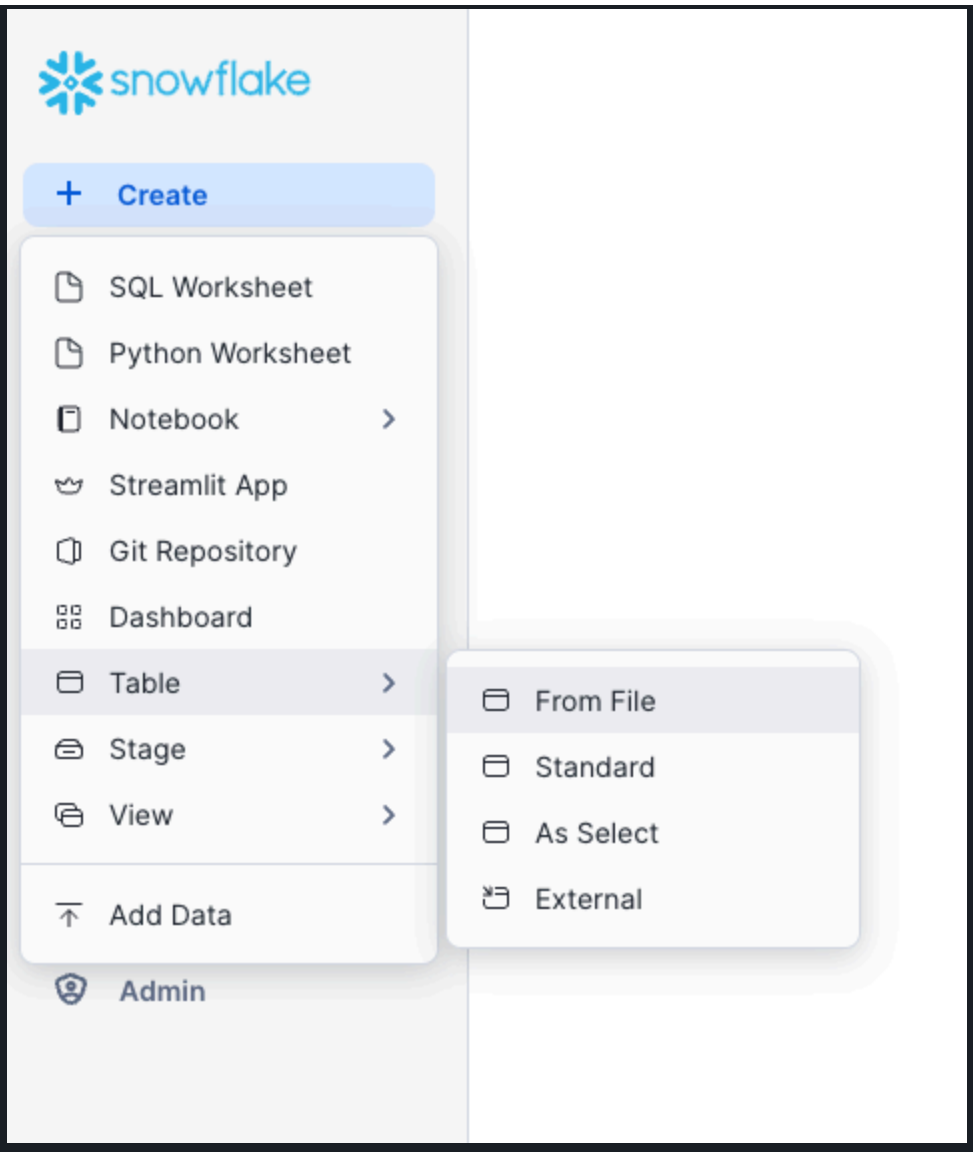
- The `CREATE DATABASE` statement creates a database. The database automatically includes a schema named ‘public’.
- The `CREATE WAREHOUSE` statement creates an initially suspended warehouse. The statement also sets `AUTO_RESUME = true`, which starts the warehouse automatically when you execute SQL statements that require compute resources.

Step 2: Load the data into Snowflake

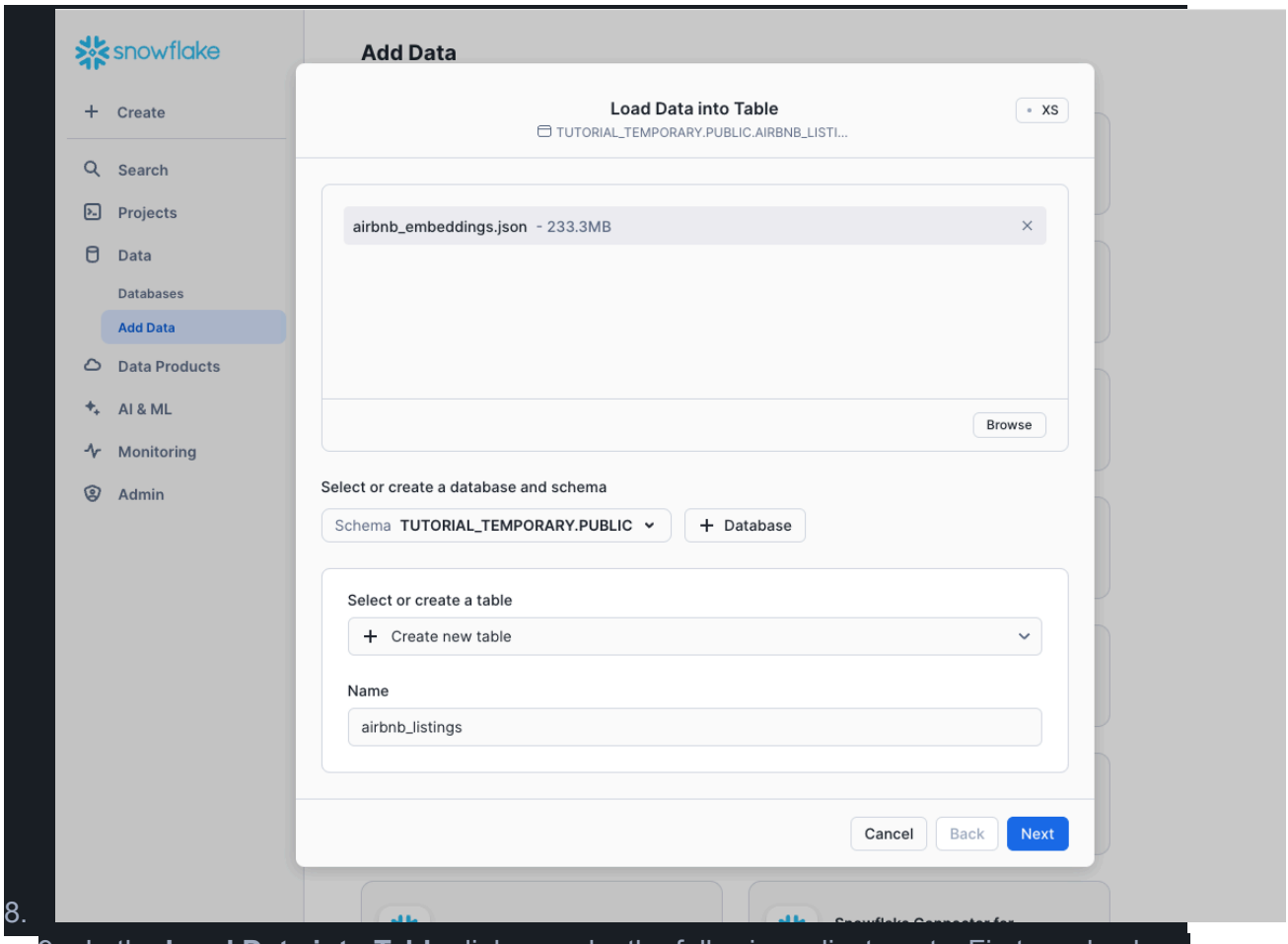
Before you can create a search service, you must load the example data into Snowflake.

You can upload the dataset in Snowsight or using SQL. To upload in Snowsight:

1. Select the **+ Create** button above the left navigation bar.
2. Then select **Table** » **From File**.

3. 

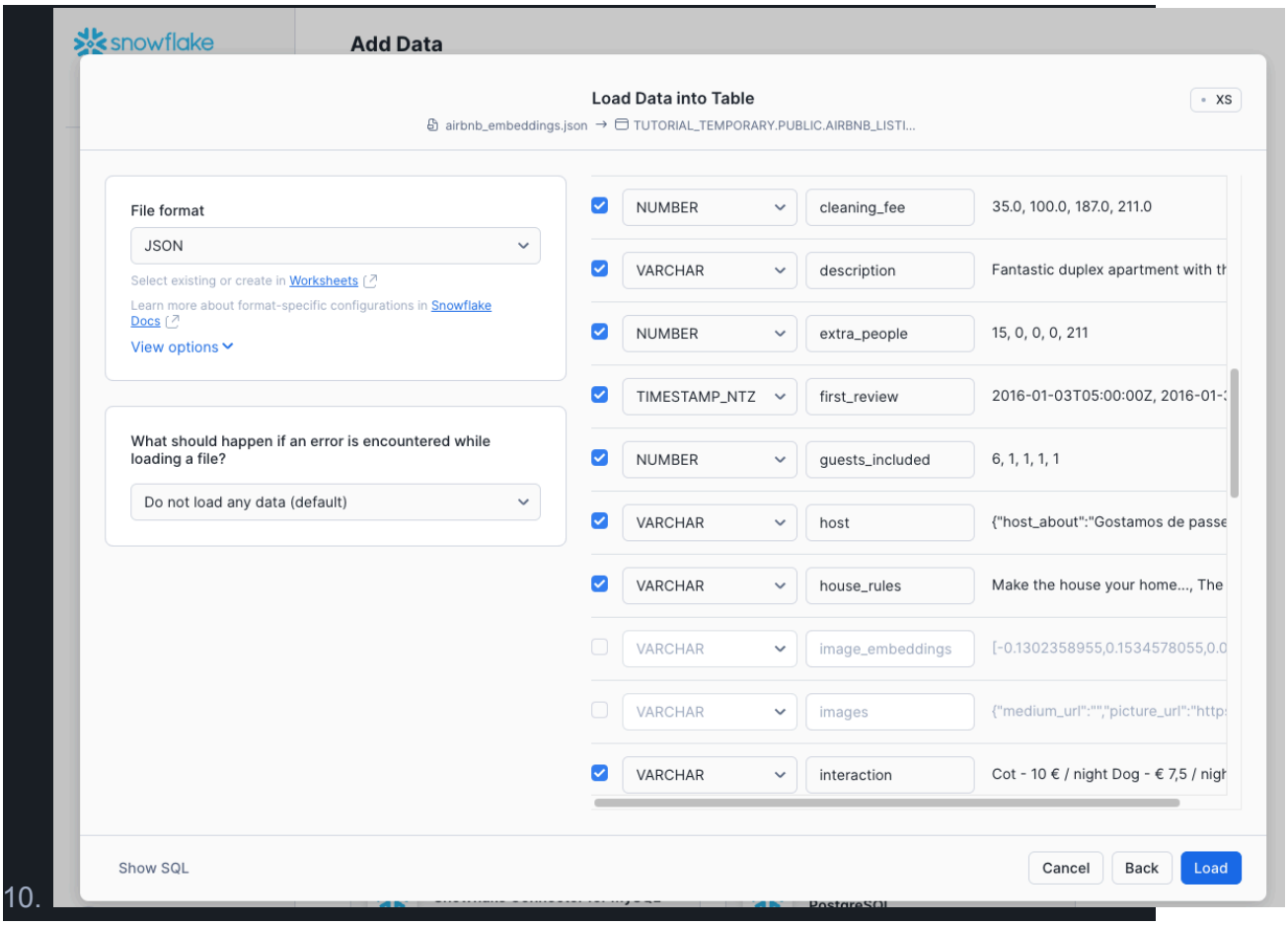
4. Select your newly-created warehouse as a warehouse for your table from the drop-down at the top right corner.
5. Drag and drop the JSON data file into the dialog.
6. Select the database you created above and specify the PUBLIC schema.
7. Finally, specify the creation of a new table called `airbnb_listings` and select **Next**.

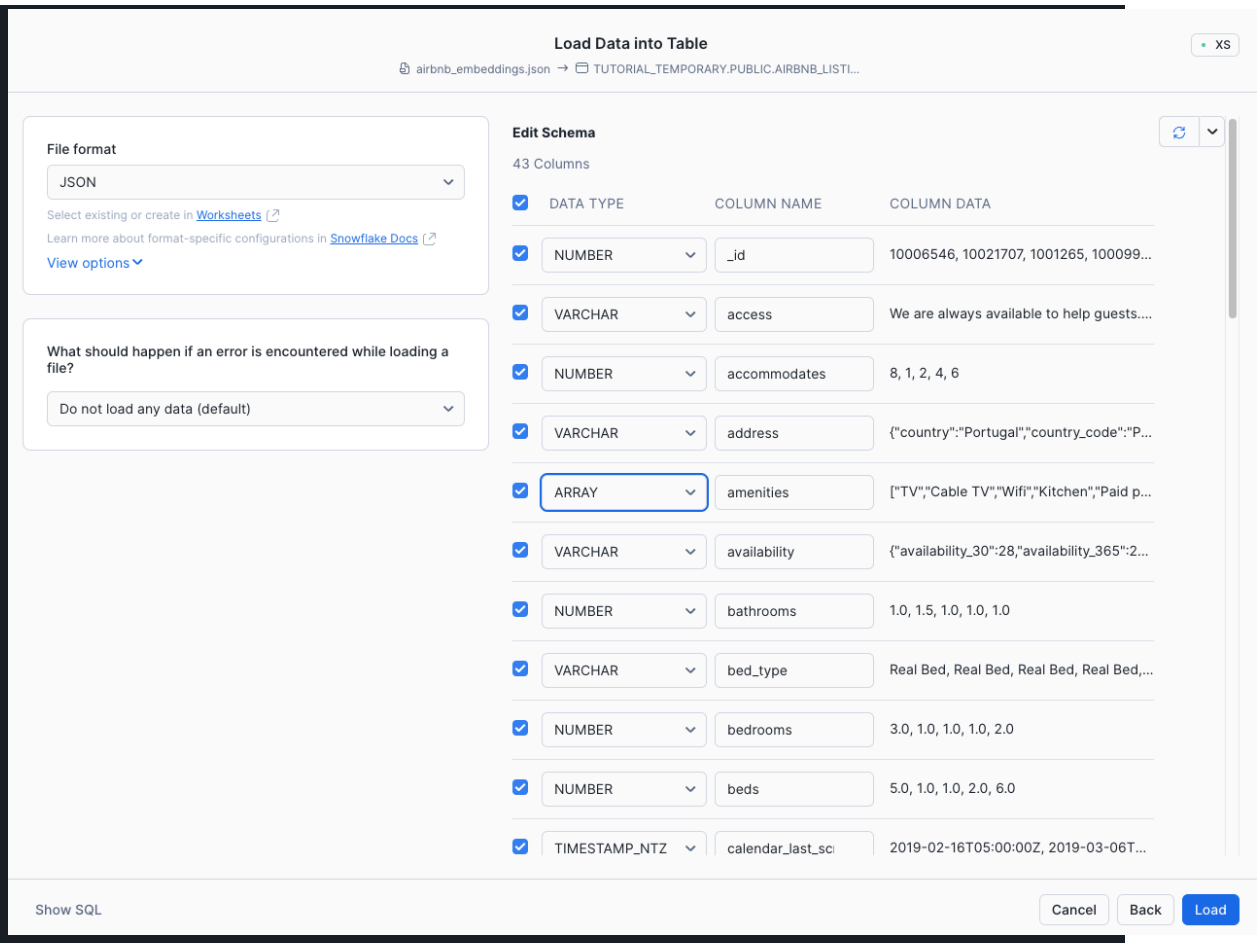


8.

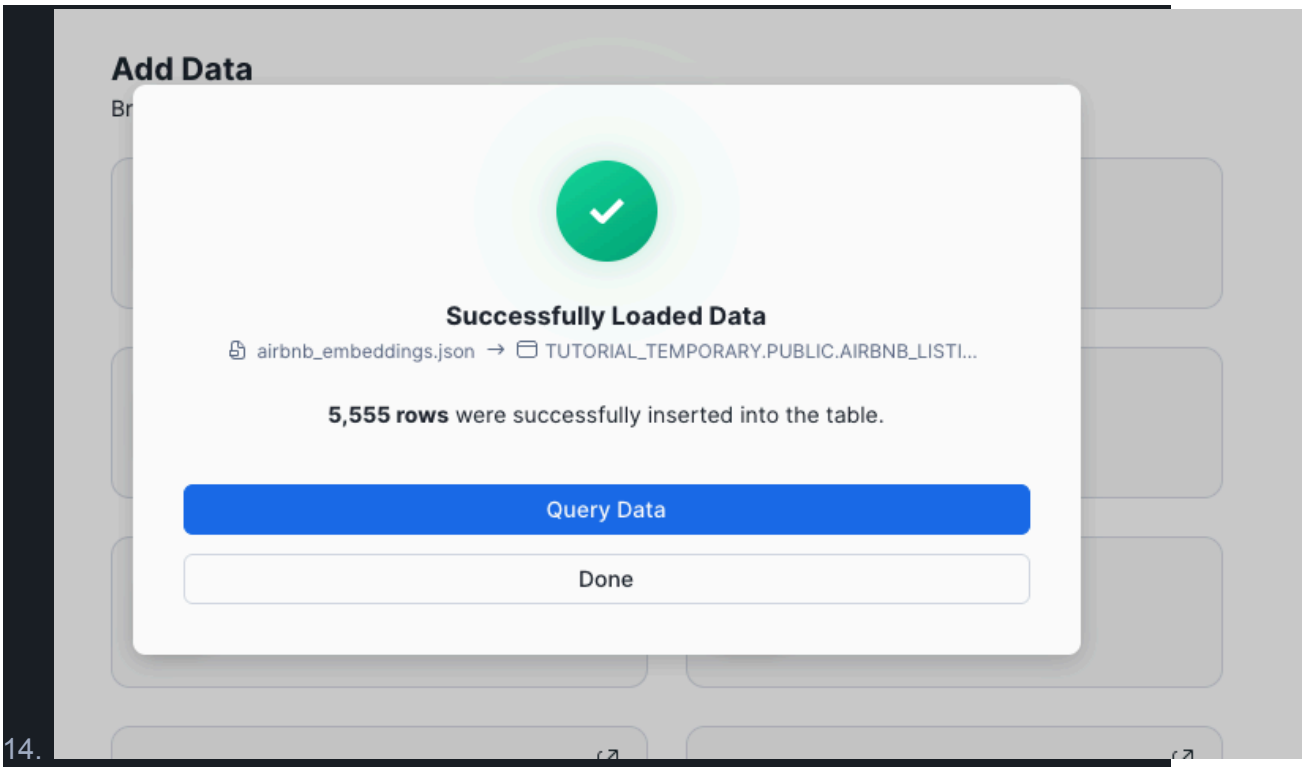
9. In the **Load Data into Table** dialog, make the following adjustments. First, uncheck the `image_embeddings`, `images`, and `text_embeddings` columns, since those do not apply to this tutorial. Second, adjust the datatype of the `amenities` field to be ARRAY type.

10.


snowflake
Add Data
Load Data into Table
airbnb_embeddings.json → TUTORIAL_TEMPORARY.PUBLIC.AIRBNB_LISTI...
XS
File format
JSON
Select existing or create in Worksheets
Learn more about format-specific configurations in Snowflake Docs
View options
What should happen if an error is encountered while loading a file?
Do not load any data (default)
NUMBER cleaning_fee 35.0, 100.0, 187.0, 211.0
VARCHAR description Fantastic duplex apartment with th
NUMBER extra_people 15, 0, 0, 0, 211
TIMESTAMP_NTZ first_review 2016-01-03T05:00:00Z, 2016-01-
NUMBER guests_included 6, 1, 1, 1, 1
VARCHAR host {"host_about":"Gostamos de passe
VARCHAR house_rules Make the house your home..., The
VARCHAR image_embeddings [-0.1302358955,0.1534578055,0.0
VARCHAR images {"medium_url":"","picture_url":"http
VARCHAR interaction Cot - 10 € / night Dog - € 7,5 / nigh
Show SQL
Cancel
Back
Load

11. 

12. Once you have made these adjustments, Select **Load** to proceed.
13. After a brief moment, you should see a confirmation page showing that the data has been loaded.

14. 

15. Select **Query Data** to open up a new Snowsight worksheet that you will use in the next step.

Step 3: Create the search service

Create a search service over our new table by running the following SQL command.

```
CREATE OR REPLACE CORTEX SEARCH SERVICE
cortex_search_tutorial_db.public.airbnb_svc
ON listing_text
ATTRIBUTES room_type, amenities
WAREHOUSE = cortex_search_tutorial_wh
TARGET_LAG = '1 hour'
AS
    SELECT
        room_type,
        amenities,
        price,
        cancellation_policy,
        ('Summary\n\n' || summary || '\n\n\nDescription\n\n' ||
description || '\n\n\nSpace\n\n' || space) as listing_text
    FROM
    cortex_search_tutorial_db.public.airbnb_listings;
```

Let's break down the arguments in this command:

- The `ON` parameter specifies the column for queries to search over. In this case, it's the `listing_text`, which is generated in the source query as a concatenation of several text columns in the base table.
- The `ATTRIBUTES` parameter specifies the columns that you will be able to filter search results on. This example filers on `room_type` and `amenities` when issuing queries to the `listing_text` column.
- The `WAREHOUSE` and `TARGET_LAG` parameters specify the user-provided warehouse and the desired freshness of the search service, respectively. This example specifies to use the `cortex_search_tutorial_wh` warehouse to create the index and perform refreshes, and to keep the service no more than `'1 hour'` behind the source table `AIRBNB_LISTINGS`.
- The `AS` field defines the source table for the service. This example concatenates several text columns in the original table into the search column `listing_text` so that queries can search over multiple fields.

Step 4: Create a Streamlit app

You can query the service with Python SDK (using the `snowflake` Python package). This tutorial demonstrates using the Python SDK in a Streamlit in Snowflake application.

First, ensure your global Snowsight UI role is the same as the role used to create the service in the service creation step.

1. Sign in to Snowsight.
2. Select **Projects** » **Streamlit** in the left-side navigation menu.
3. Select **+ Streamlit App**.
4. **Important**: Select the `cortex_search_tutorial_db` database and `public` schema for the app location.
5. In the left pane of the Streamlit in Snowflake editor, select **Packages** and add `snowflake` (version >= 0.8.0) to install the package in your application.

Replace the example application code with the following Streamlit app:

```
# Import python packages
import streamlit as st
from snowflake.core import Root
from snowflake.snowpark.context import get_active_session

# Constants
DB = "cortex_search_tutorial_db"
SCHEMA = "public"
```

```
SERVICE = "airbnb_svc"
BASE_TABLE = "cortex_search_tutorial_db.public.airbnb_listings"
ARRAY_ATTRIBUTES = {"AMENITIES"}


def get_column_specification():
    """
    Returns the name of the search column and a list of the names of
the attribute columns
    for the provided cortex search service
    """
    session = get_active_session()
    search_service_result = session.sql(f"DESC CORTEX SEARCH SERVICE
{DB}.{SCHEMA}.{SERVICE}").collect()[0]
    st.session_state.attribute_columns =
search_service_result.attribute_columns.split(",")
    st.session_state.search_column =
search_service_result.search_column
    st.session_state.columns =
search_service_result.columns.split(",")

def init_layout():
    st.title("Cortex AI Search")
    st.markdown(f"Querying service:
`{DB}.{SCHEMA}.{SERVICE}`".replace('"', ''))

def query_cortex_search_service(query, filter={}):
    """
    Queries the cortex search service in the session state and returns
a list of results
    """
    session = get_active_session()
    cortex_search_service = (
        Root(session)
        .databases[DB]
        .schemas[SCHEMA]
        .cortex_search_services[SERVICE]
    )
    context_documents = cortex_search_service.search(
        query,
        columns=st.session_state.columns,
        filter=filter,
        limit=st.session_state.limit)
    return context_documents.results
```

```
@st.cache_data
def distinct_values_for_attribute(col_name,
is_array_attribute=False):
    session = get_active_session()
    if is_array_attribute:
        values = session.sql(f'''
        SELECT DISTINCT value FROM {BASE_TABLE},
        LATERAL FLATTEN(input => {col_name})
        ''').collect()
    else:
        values = session.sql(f"SELECT DISTINCT {col_name} AS VALUE
FROM {BASE_TABLE}").collect()
    return [ x["VALUE"].replace('"', "") for x in values ]

def init_search_input():
    st.session_state.query = st.text_input("Query")

def init_limit_input():
    st.session_state.limit = st.number_input("Limit", min_value=1,
value=5)

def init_attribute_selection():
    st.session_state.attributes = {}
    for col in st.session_state.attribute_columns:
        is_multiselect = col in ARRAY_ATTRIBUTES
        st.session_state.attributes[col] = st.multiselect(
            col,
            distinct_values_for_attribute(col,
is_array_attribute=is_multiselect)
        )

def display_search_results(results):
    """
    Display the search results in the UI
    """
    st.subheader("Search results")
    for i, result in enumerate(results):
        result = dict(result)
        container = st.expander(f"[Result {i+1}]", expanded=True)

        # Add the result text.
        container.markdown(result[st.session_state.search_column])

        # Add the attributes.
        for column, column_value in sorted(result.items()):
```

```
            if column == st.session_state.search_column:
                continue
            container.markdown(f"**{column}**: {column_value}")

def create_filter_object(attributes):
    """
    Create a filter object for the search query
    """
    and_clauses = []
    for column, column_values in attributes.items():
        if len(column_values) == 0:
            continue
        if column in ARRAY_ATTRIBUTES:
            for attr_value in column_values:
                and_clauses.append({"@contains": { column: attr_value
}})
        else:
            or_clauses = [{"@eq": {column: attr_value}} for
attr_value in column_values]
            and_clauses.append({"@or": or_clauses })

    return {"@and": and_clauses} if and_clauses else {}


def main():
    init_layout()
    get_column_specification()
    init_attribute_selection()
    init_limit_input()
    init_search_input()

    if not st.session_state.query:
        return
    results = query_cortex_search_service(
        st.session_state.query,
        filter = create_filter_object(st.session_state.attributes)
    )
    display_search_results(results)


if __name__ == "__main__":
    st.set_page_config(page_title="Cortex AI Search and Summary",
layout="wide")
    main()
```

6.

Here's a brief breakdown of the major components in the Streamlit-in-Snowflake code above:

- `get_column_specification` uses a DESCRIBE SQL query to get information about the attributes available in the search service and stores them in Stremalit state.
- `init_layout` sets up the header and intro of the page.
- `query_cortex_search_service` handles querying the Cortex Search Service via the Python client library.
- `create_filter_object` processes selected filter attributes from the Streamlit form into the right objects to be used by the Python library for querying Cortex Search.
- `distinct_values_for_attribute` determines which values are possible for each filterable attribute to populate the dropdown menus.
- `init_search_input`, `init_limit_input`, `init_attribute_selection` initialize inputs for the search query, limit of number of results, and attribute filters.
- `display_search_results` formats search results into Markdown elements displayed in the results page.

Step 5: Clean up

Clean up (optional)

Execute the following DROP <object> commands to return your system to its state before you began the tutorial:

```
DROP DATABASE IF EXISTS cortex_search_tutorial_db;
```

```
DROP WAREHOUSE IF EXISTS cortex_search_tutorial_wh;
```

Dropping the database automatically removes all child database objects such as tables.

Next steps

Congratulations! You have successfully built a simple search app on text data in Snowflake. You can move on to Tutorial 2 to see how to layer on Cortex LLM Functions to build an AI chatbot with Cortex Search.

Additional resources

Additionally, you can continue learning using the following resources:

- Cortex Search overview
- Query a Cortex Search Service