# Snowflake Immersion Day


## Hands on Lab

Getting Familiar with Snowflake Concepts.

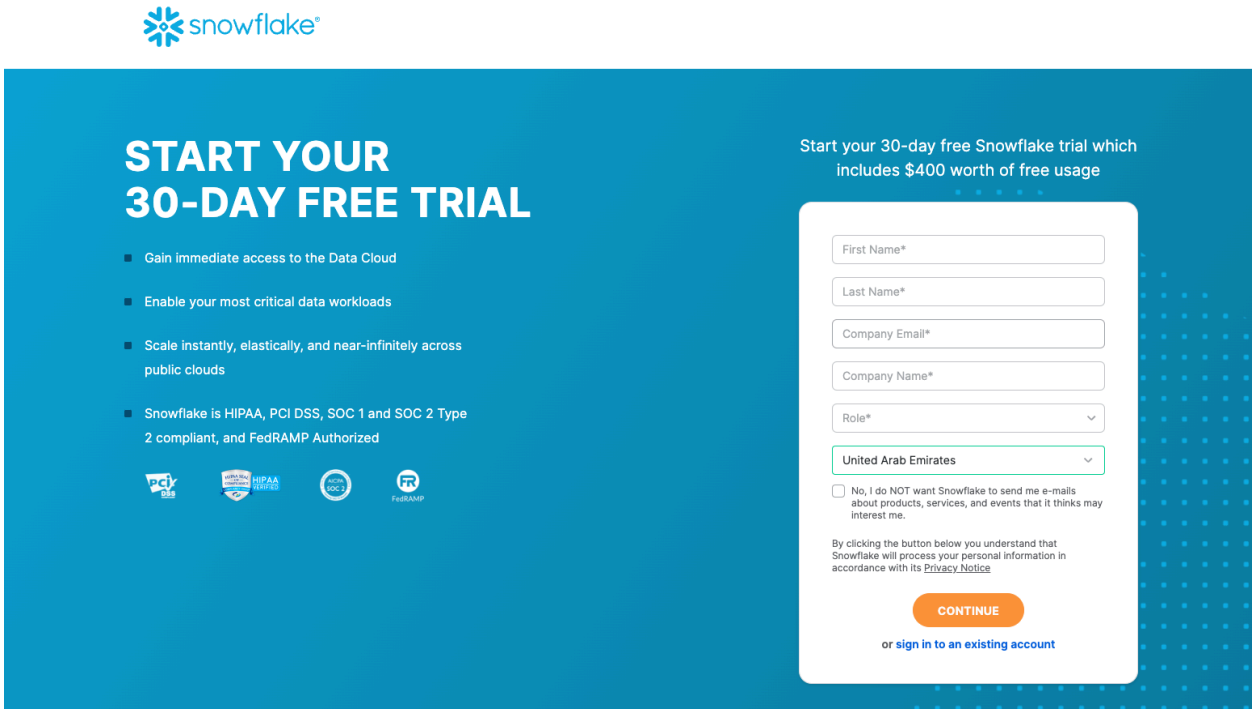
4th of November, 2024

# Setting up your Snowflake Trial Account

Navigate to:

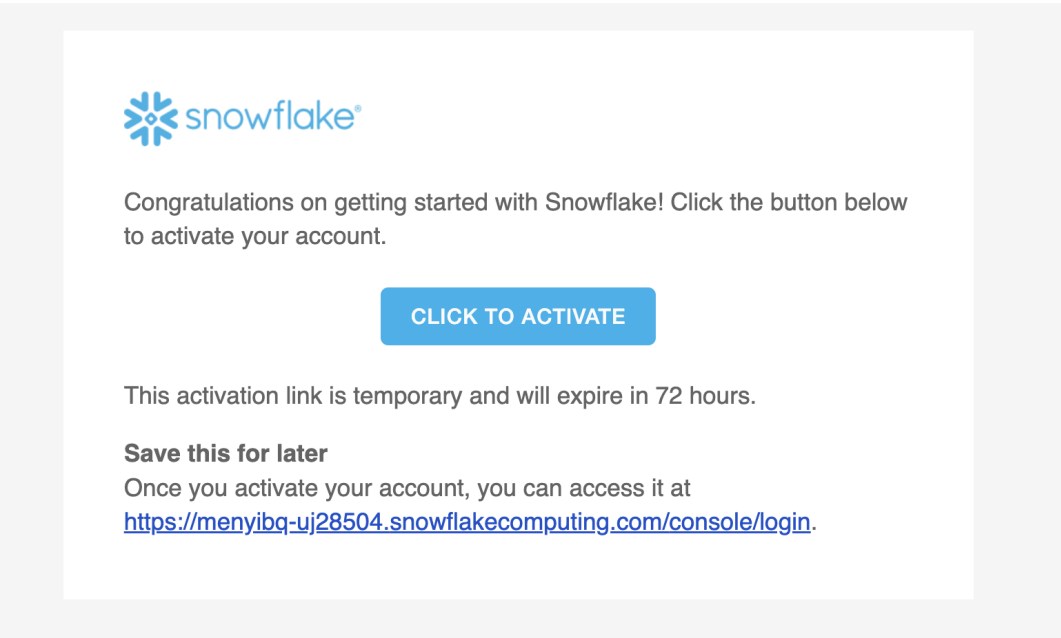
*https://signup.snowflake.com/?account=dmF1bHQ6djE6V0ZoaVJFZVVrYU1HNGF6NEQwWUtycUlsdGVPV3lqSFAyT3Rrc0dNQXh2eXI4THk1cytsTUxuRTZteEVORlE9PQ%3D%3D&owner=chris.boyd&cloud=aws&region=eu-central-1&plan=enterprise*

and fill in your details

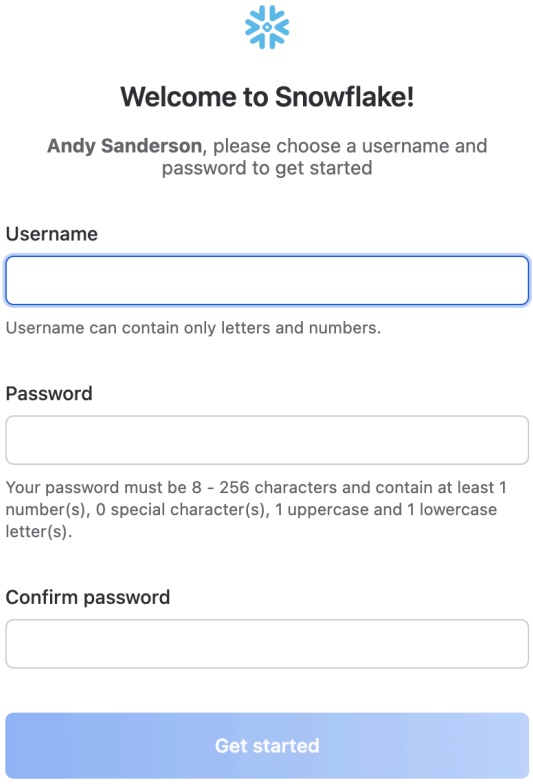



You will be asked to fill in a survey and then upon completion you  an activation email will be sent to your email address

Select **Click to Activate** on you email, which will redirect you to a snowflake signup page , Also note the url ,which will be the account url for your newly created snowflake account.

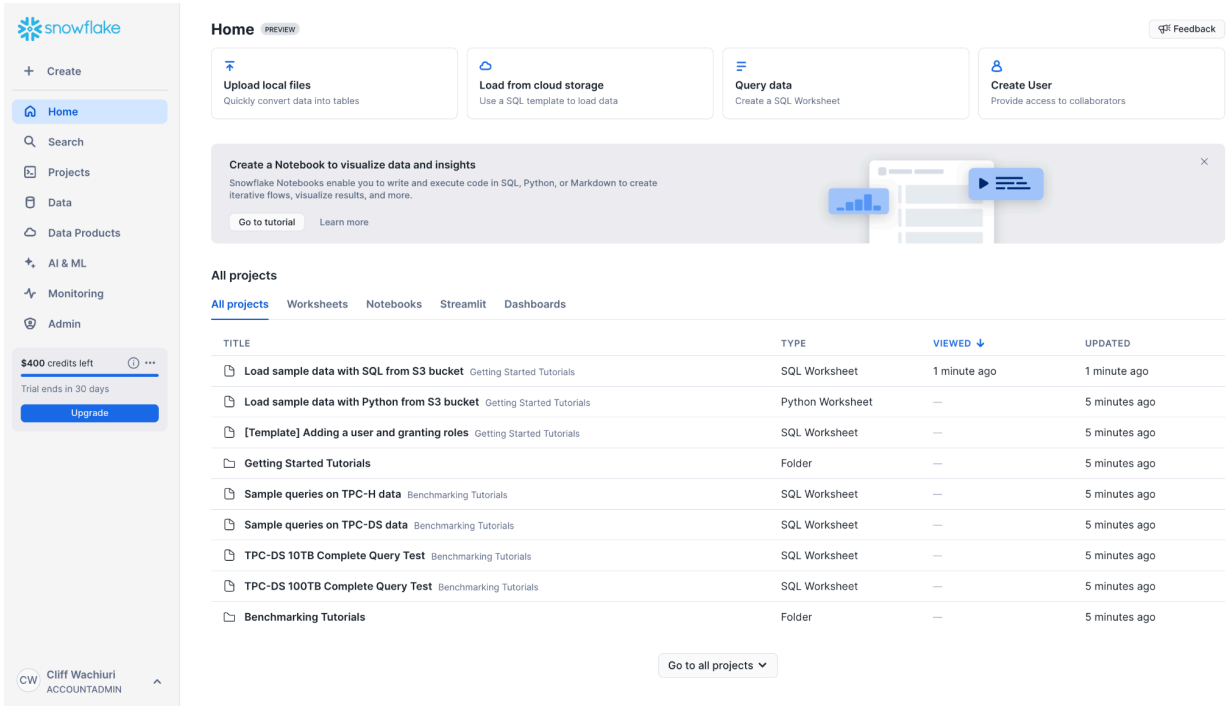



Create a username and password, record this somewhere safe - this will be the admin login for your Snowflake account.




Insert your credentials and you will then be redirected to your Snowflake Account

snowflake®


snowflake
Create
Home
Search
Projects
Data
Data Products
AI & ML
Monitoring
Admin
$400 credits left
Trial ends in 30 days
Upgrade
CW
Cliff Wachiuri
ACCOUNTADMIN
Home
PREVIEW
Feedback
Upload local files
Quickly convert data into tables
Load from cloud storage
Use a SQL template to load data
Query data
Create a SQL Worksheet
Create User
Provide access to collaborators
Create a Notebook to visualize data and insights
Snowflake Notebooks enable you to write and execute code in SQL, Python, or Markdown to create iterative flows, visualize results, and more.
Go to tutorial
Learn more
All projects
All projects
Worksheets
Notebooks
Streamlit
Dashboards
TITLE
TYPE
VIEWED
UPDATED
Load sample data with SQL from S3 bucket Getting Started Tutorials
SQL Worksheet
1 minute ago
1 minute ago
Load sample data with Python from S3 bucket Getting Started Tutorials
Python Worksheet
—
5 minutes ago
[Template] Adding a user and granting roles Getting Started Tutorials
SQL Worksheet
—
5 minutes ago
Getting Started Tutorials
Folder
—
5 minutes ago
Sample queries on TPC-H data Benchmarking Tutorials
SQL Worksheet
—
5 minutes ago
Sample queries on TPC-DS data Benchmarking Tutorials
SQL Worksheet
—
5 minutes ago
TPC-DS 10TB Complete Query Test Benchmarking Tutorials
SQL Worksheet
—
5 minutes ago
TPC-DS 100TB Complete Query Test Benchmarking Tutorials
SQL Worksheet
—
5 minutes ago
Benchmarking Tutorials
Folder
—
5 minutes ago
Go to all projects

# Creating Snowflake Objects

## What You Will Learn

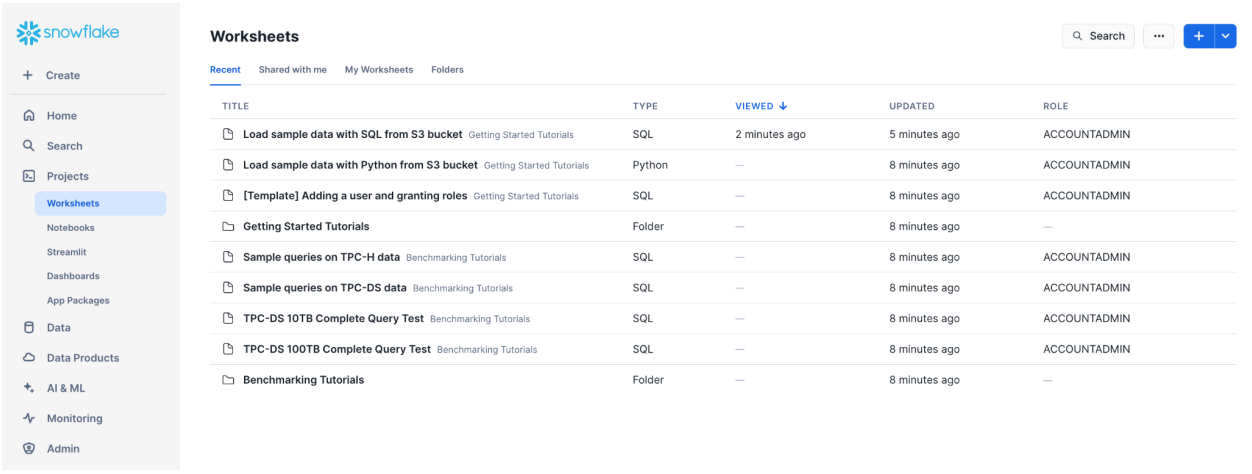
- How to Create a Snowflake Worksheet
- How to create Databases
- How to create Raw, Harmonized and Analytic Schemas complete with Tables and Views
- How to Execute All Queries within a Snowflake Worksheet Synchronously

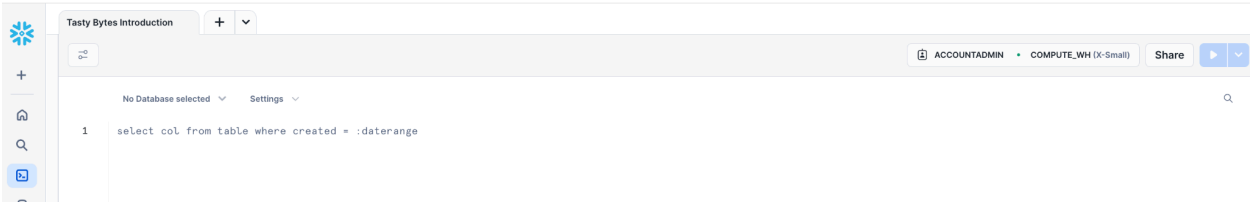
## Step 1: Create a Snowflake Worksheet

On the Navigation Bar, head on to **Projects - > Worksheets**

Within Worksheets, click the "+" button in the top-right corner of Snowsight.

Rename the Worksheet by double clicking on the auto-generated Timestamp name and inputting "Tasty Bytes - Introduction"



# Step 2: Creating a Database, Schemas and Warehouses

We will first create the snowflake objects, namely, a database, schema,  and a virtual warehouse of which we will use as our compute.

To do this, Copy and this SQL code into the newly created worksheet and Paste (*CMD + V for Mac or CTRL + V for Windows*)

Click inside the newly created Tasty Bytes - Setup Worksheet, and next to "► Run" Click "▼" and choose "Run All"

```
Unset
USE ROLE accountadmin;


-- create tb_101 database
CREATE OR REPLACE DATABASE tb_101;

-- create raw_pos schema
CREATE OR REPLACE SCHEMA tb_101.raw_pos;

-- create raw_customer schema
CREATE OR REPLACE SCHEMA tb_101.raw_customer;

-- create harmonized schema
CREATE OR REPLACE SCHEMA tb_101.harmonized;

-- create analytics schema
CREATE OR REPLACE SCHEMA tb_101.analytics;

-- create developer schema
CREATE OR REPLACE SCHEMA tb_101.sandbox;

-- create warehouses
CREATE OR REPLACE WAREHOUSE tb_de_wh
    WAREHOUSE_SIZE = xsmall
    WAREHOUSE_TYPE = 'standard'
    AUTO_SUSPEND = 60
    AUTO_RESUME = TRUE
    INITIALLY_SUSPENDED = TRUE
COMMENT = 'data engineering warehouse for tasty bytes';
```

# Step 3: Creating Tables and Views

Create the various tables and views which will store the data. We will create them using the **standard SQL Data Definition Language (DDL).** Copy and run this code into a Snowflake Worksheet similar to the step above.

Unset

```
/*--
 raw zone table build
--*/
-- country table build
CREATE OR REPLACE TABLE tb_101.raw_pos.country
(
    country_id NUMBER(18,0),
    country VARCHAR(16777216),
    iso_currency VARCHAR(3),
    iso_country VARCHAR(2),
    city_id NUMBER(19,0),
    city VARCHAR(16777216),
    city_population VARCHAR(16777216)
);

-- franchise table build
CREATE OR REPLACE TABLE tb_101.raw_pos.franchise
(
    franchise_id NUMBER(38,0),
    first_name VARCHAR(16777216),
    last_name VARCHAR(16777216),
    city VARCHAR(16777216),
    country VARCHAR(16777216),
    e_mail VARCHAR(16777216),
    phone_number VARCHAR(16777216)
);

-- location table build
CREATE OR REPLACE TABLE tb_101.raw_pos.location
(
    location_id NUMBER(19,0),
    placekey VARCHAR(16777216),
    location VARCHAR(16777216),
    city VARCHAR(16777216),
    region VARCHAR(16777216),
    iso_country_code VARCHAR(16777216),
    country VARCHAR(16777216));
```

```
-- menu table build
CREATE OR REPLACE TABLE tb_101.raw_pos.menu
(
    menu_id NUMBER(19,0),
    menu_type_id NUMBER(38,0),
    menu_type VARCHAR(16777216),
    truck_brand_name VARCHAR(16777216),
    menu_item_id NUMBER(38,0),
    menu_item_name VARCHAR(16777216),
    item_category VARCHAR(16777216),
    item_subcategory VARCHAR(16777216),
    cost_of_goods_usd NUMBER(38,4),
    sale_price_usd NUMBER(38,4),
    menu_item_health_metrics_obj VARIANT
);

-- truck table build
CREATE OR REPLACE TABLE tb_101.raw_pos.truck
(
    truck_id NUMBER(38,0),
    menu_type_id NUMBER(38,0),
    primary_city VARCHAR(16777216),
    region VARCHAR(16777216),
    iso_region VARCHAR(16777216),
    country VARCHAR(16777216),
    iso_country_code VARCHAR(16777216),
    franchise_flag NUMBER(38,0),
    year NUMBER(38,0),
    make VARCHAR(16777216),
    model VARCHAR(16777216),
    ev_flag NUMBER(38,0),
    franchise_id NUMBER(38,0),
    truck_opening_date DATE
);
```

```
-- order_header table build
CREATE OR REPLACE TABLE tb_101.raw_pos.order_header
(
    order_id NUMBER(38,0),
    truck_id NUMBER(38,0),
    location_id FLOAT,
    customer_id NUMBER(38,0),
    discount_id VARCHAR(16777216),
    shift_id NUMBER(38,0),
    shift_start_time TIME(9),
    shift_end_time TIME(9),
    order_channel VARCHAR(16777216),
    order_ts TIMESTAMP_NTZ(9),
    served_ts VARCHAR(16777216),
    order_currency VARCHAR(3),
    order_amount NUMBER(38,4),
    order_tax_amount VARCHAR(16777216),
    order_discount_amount VARCHAR(16777216),
    order_total NUMBER(38,4)
);

-- order_detail table build
CREATE OR REPLACE TABLE tb_101.raw_pos.order_detail
(
    order_detail_id NUMBER(38,0),
    order_id NUMBER(38,0),
    menu_item_id NUMBER(38,0),
    discount_id VARCHAR(16777216),
    line_number NUMBER(38,0),
    quantity NUMBER(5,0),
    unit_price NUMBER(38,4),
    price NUMBER(38,4),
    order_item_discount_amount VARCHAR(16777216)
);
```

```
-- customer loyalty table build
CREATE OR REPLACE TABLE tb_101.raw_customer.customer_loyalty
(
    customer_id NUMBER(38,0),
    first_name VARCHAR(16777216),
    last_name VARCHAR(16777216),
    city VARCHAR(16777216),
    country VARCHAR(16777216),
    postal_code VARCHAR(16777216),
    preferred_language VARCHAR(16777216),
    gender VARCHAR(16777216),
    favourite_brand VARCHAR(16777216),
    marital_status VARCHAR(16777216),
    children_count VARCHAR(16777216),
    sign_up_date DATE,
    birthday_date DATE,
    e_mail VARCHAR(16777216),
    phone_number VARCHAR(16777216));
```

```
Unset
/*--
 • harmonized view creation
--*/
-- orders_v view
CREATE OR REPLACE VIEW tb_101.harmonized.orders_v
    AS
SELECT
    oh.order_id,
    oh.truck_id,
    oh.order_ts,
    od.order_detail_id,
    od.line_number,
    m.truck_brand_name,
    m.menu_type,
    t.primary_city,
    t.region,
    t.country,
    t.franchise_flag,
    t.franchise_id,
    f.first_name AS franchisee_first_name,
    f.last_name AS franchisee_last_name,
    l.location_id,
    cl.customer_id,
    cl.first_name,
    cl.last_name,
    cl.e_mail,
    cl.phone_number,
    cl.children_count,
    cl.gender,
    cl.marital_status,
    od.menu_item_id,
    m.menu_item_name,
    od.quantity,
    od.unit_price,
    od.price,
    oh.order_amount,
    oh.order_tax_amount,
    oh.order_discount_amount,
    oh.order_total
FROM tb_101.raw_pos.order_detail od
JOIN tb_101.raw_pos.order_header oh
  ON od.order_id = oh.order_id
```

```
JOIN tb_101.raw_pos.truck t
    ON oh.truck_id = t.truck_id
JOIN tb_101.raw_pos.menu m
    ON od.menu_item_id = m.menu_item_id
JOIN tb_101.raw_pos.franchise f
    ON t.franchise_id = f.franchise_id
JOIN tb_101.raw_pos.location l
    ON oh.location_id = l.location_id
LEFT JOIN tb_101.raw_customer.customer_loyalty cl
    ON oh.customer_id = cl.customer_id;

-- loyalty_metrics_v view
CREATE OR REPLACE VIEW tb_101.harmonized.customer_loyalty_metrics_v
    AS
SELECT
    cl.customer_id,
    cl.city,
    cl.country,
    cl.first_name,
    cl.last_name,
    cl.phone_number,
    cl.e_mail,
    SUM(oh.order_total) AS total_sales,
    ARRAY_AGG(DISTINCT oh.location_id) AS visited_location_ids_array
FROM tb_101.raw_customer.customer_loyalty cl
JOIN tb_101.raw_pos.order_header oh
ON cl.customer_id = oh.customer_id
GROUP BY cl.customer_id, cl.city, cl.country, cl.first_name,
cl.last_name, cl.phone_number, cl.e_mail;

/*--
 • analytics view creation
--*/
-- orders_v view
CREATE OR REPLACE VIEW tb_101.analytics.orders_v
COMMENT = 'Tasty Bytes Order Detail View'
    AS
SELECT DATE(o.order_ts) AS date, * FROM tb_101.harmonized.orders_v o;

-- customer_loyalty_metrics_v view
CREATE OR REPLACE VIEW tb_101.analytics.customer_loyalty_metrics_v
COMMENT = 'Tasty Bytes Customer Loyalty Member Metrics View'
    AS
SELECT * FROM tb_101.harmonized.customer_loyalty_metrics_v;
```

# Step 4: Loading Data from s3 to Snowflake

As we will be loading some data from s3, we will need to create two objects

- A File format, which will define the format of how the data is stored in s3 , such as csv, json, parquet , avro etc,
- A Stage, which refers to a connection to an object storage, which could be hosted by aws, google, azure or snowflake managed object storage

Copy and run the following code to create the file format

```
Unset
CREATE OR REPLACE FILE FORMAT tb_101.public.csv_ff
type = 'csv';
```

Copy and run the following code to create a stage, to integrate with an s3 bucket

```
Unset
CREATE OR REPLACE STAGE tb_101.public.s3load
COMMENT = 'Quickstarts S3 Stage Connection'
url = 's3://sfquickstarts/frostbyte_tastybytes/'
file_format = tb_101.public.csv_ff;
```

Load the data from s3 into the snowflake tables by running the following copy commands

```
Unset
USE WAREHOUSE tb_de_wh;

-- Upsize our compute to make loading faster
ALTER WAREHOUSE tb_de_wh SET WAREHOUSE_SIZE = 'XLarge';

-- country table load
COPY INTO tb_101.raw_pos.country
FROM @tb_101.public.s3load/raw_pos/country/;

-- franchise table load
COPY INTO tb_101.raw_pos.franchise
FROM @tb_101.public.s3load/raw_pos/franchise/;

-- location table load
COPY INTO tb_101.raw_pos.location
FROM @tb_101.public.s3load/raw_pos/location/;

-- menu table load
COPY INTO tb_101.raw_pos.menu
FROM @tb_101.public.s3load/raw_pos/menu/;

-- truck table load
COPY INTO tb_101.raw_pos.truck
FROM @tb_101.public.s3load/raw_pos/truck/;


-- customer_loyalty table load
COPY INTO tb_101.raw_customer.customer_loyalty
FROM @tb_101.public.s3load/raw_customer/customer_loyalty/;

-- order_header table load
COPY INTO tb_101.raw_pos.order_header
FROM @tb_101.public.s3load/raw_pos/order_header/;

-- order_detail table load
COPY INTO tb_101.raw_pos.order_detail
FROM @tb_101.public.s3load/raw_pos/order_detail/;

-- Resize back to XSmall now that we are done loading
ALTER WAREHOUSE tb_de_wh SET WAREHOUSE_SIZE = 'XSmall';
```

We can check the number of rows loaded in our largest tables, the order_header table and order_detail table, by running the following sql statements. The total number of rows in these two tables is about **900 Million rows**

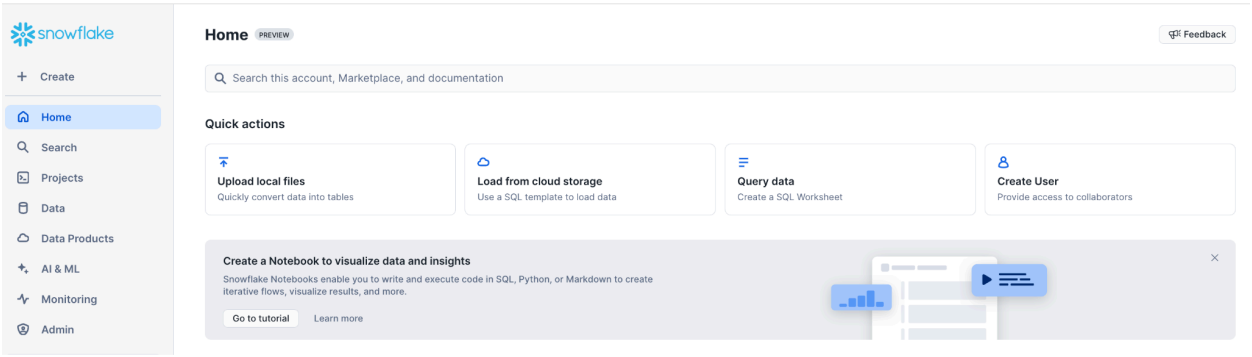
```
Unset
SELECT COUNT(*) FROM tb_101.raw_pos.order_detail;
SELECT COUNT(*) FROM tb_101.raw_pos.order_header;
```

## Step 5: Loading Data from a CSV File

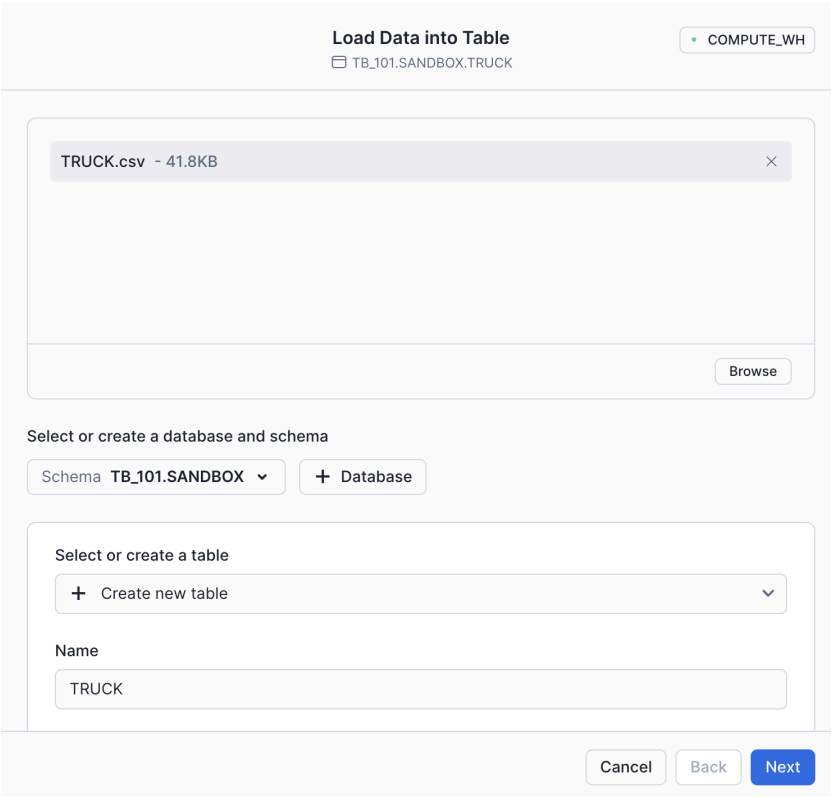
We have some data about the trucks that we would want to join with the other data that we loaded from the s3 bucket.

Download the TRUCK.csv file from this github repo

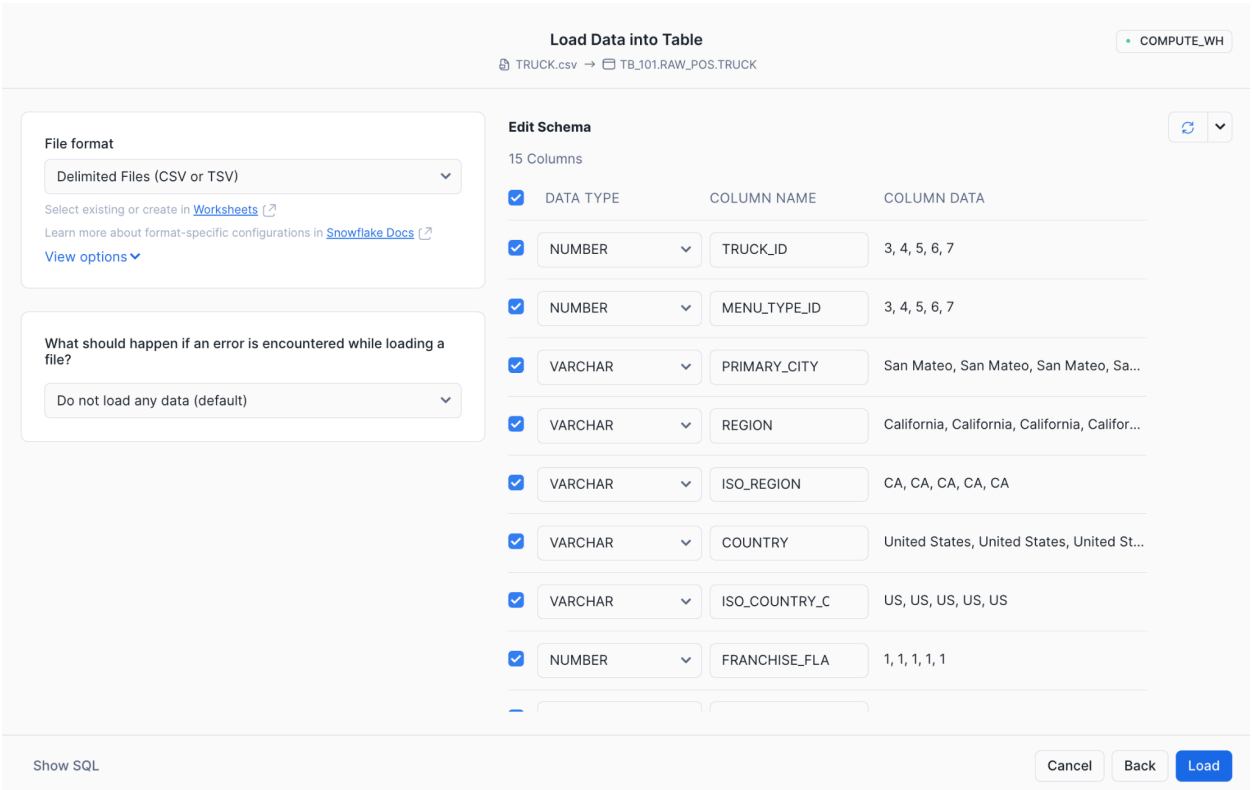
On the Navigation Bar on the left in Snowflake, Select **Home**- >, and under **Quick Actions** , select **Upload Local Files** and on the main




Upload the **TRUCK.csv** file and set the schema as **TB_101.SANDBOX** , leave the option as Create a New Table, and name the new Table as **TRUCK**




Upon selecting **Next** Snowflake will attempt to parse the csv to detect the schema and the column names.
On the left, in the **File Format**, select the **View Options** drop-down and next to the **Header section**, verify that the value is **First Line Contains Header.**




Select **Load** to finalize the creation of a table with these columns in the chosen database and schema

This data is now ready to query and join with the other data.

# Compute and Scaling

What You Will Learn

- How to Create and Configure a Snowflake Warehouse
- How to Scale a Snowflake Warehouse Up and Down
- How to Suspend a Snowflake Warehouse

## Step 1 : Create a Snowflake Warehouse

Create a new Snowflake Worksheet

In the Worksheet , run this code to create a snowflake warehouse . This will be a multi cluster warehouse of size Xsmall

```
Unset
USE ROLE accountadmin;
CREATE OR REPLACE WAREHOUSE tb_test_wh WITH
COMMENT = 'test warehouse for tasty bytes'
    WAREHOUSE_TYPE = 'standard'
    WAREHOUSE_SIZE = 'xsmall'
    MIN_CLUSTER_COUNT = 1
    MAX_CLUSTER_COUNT = 2
    SCALING_POLICY = 'standard'
    AUTO_SUSPEND = 60
    AUTO_RESUME = true
    INITIALLY_SUSPENDED = true;
```

Based on the query we ran, please see the details below on what each configuration handles within our CREATE WAREHOUSE statement.

**Warehouse Type:** Warehouses are required for queries, as well as all DML operations, including loading data into tables. Snowflake supports Standard (most-common) or Snowpark-optimized Warehouse Types.

**Warehouse Size:** Size specifies the amount of compute resources available per cluster in a warehouse. Snowflake supports X-Small through 6X-Large sizes.

**Minimum and Maximum Cluster Count:** With multi-cluster warehouses, Snowflake supports allocating, either statically or dynamically, additional clusters to make a larger pool of compute resources available.

**Scaling Policy:** Specifies the policy for automatically starting and shutting down clusters in a multi-cluster warehouse running in Auto-scale mode.

**Auto Suspend:** By default, Auto-Suspend is enabled. Snowflake automatically suspends the warehouse if it is inactive for the specified period of time, in our case 60 seconds.

**Auto Resume:** By default, auto-resume is enabled. Snowflake automatically resumes the warehouse when any statement that requires a warehouse is submitted and the warehouse is the current warehouse for the session.

**Initially Suspended:** Specifies whether the warehouse is created initially in the ‘Suspended' state.

For further information on Snowflake Warehouses please visit the Snowflake Warehouse Documentation

## Step 2: Querying Some Data with the Warehouse.

We will use the command **USE WAREHOUSE** to configure our Snowflake session to use the newly created warehouse

We can then query the table to find all food items sold at our Plant Palace branded trucks

```
Unset
USE WAREHOUSE tb_test_wh;

SELECT
    m.menu_type,
    m.truck_brand_name,
    m.menu_item_id,
    m.menu_item_name
FROM tb_101.raw_pos.menu m
WHERE truck_brand_name = 'Plant Palace';
```

## Step 3: Scaling Our Warehouse Up

After completing a basic query against one of our dimension tables, let's now get ready to query our much larger orders data set.

Let's now instantly scale our tb_test_wh up by executing our next query leveraging ALTER WAREHOUSE... SET warehouse_size.

```
Unset
ALTER WAREHOUSE tb_test_wh SET warehouse_size = 'XLarge';
```

## Step 4: Run an Aggregation Query Against a Large Data Set

With our Warehouse scaled up, let's now run our next query which uses CONCAT, COUNT and SUM to calculate orders and total sales for Tasty Bytes customer loyalty members.

```
Unset
SELECT
    o.customer_id,
    CONCAT(clm.first_name, ' ', clm.last_name) AS name,
    COUNT(DISTINCT o.order_id) AS order_count,
    SUM(o.price) AS total_sales
FROM tb_101.analytics.orders_v o
JOIN tb_101.analytics.customer_loyalty_metrics_v clm
    ON o.customer_id = clm.customer_id
GROUP BY o.customer_id, name
ORDER BY order_count DESC;
```

| | CUSTOMER_ID | NAME | ORDER_COUNT | TOTAL_SALES |
|---|---|---|---|---|
| 1 | 210161 | Josh Weiss | 115 | 4572.5000 |
| 2 | 222196 | Kaylynn Woodard | 115 | 4597.7500 |
| 3 | 203052 | Campbell Sutton | 115 | 5095.7500 |
| 4 | 209995 | Dax Mooney | 114 | 4903.0000 |
| 5 | 45123 | Gianna Benson | 112 | 5657.0000 |
| 6 | 203637 | Mullen Hood | 112 | 4801.5000 |
| 7 | 216408 | Pierce Goodwin | 111 | 4806.5000 |
| 8 | 209923 | Lilian Nguyen | 111 | 4288.0000 |
| 9 | 216587 | Zoe Key | 111 | 4270.2500 |
| 10 | 209383 | Philip Jimenez | 111 | 4882.5000 |

## Step 5:  Scale our Warehouse Down

Having seen the instant upward scalability of our Snowflake Warehouse and how it can aggregate large result sets with ease, let's now instantly scale our `tb_test_wh` back down by running the next query.

```
Unset
ALTER WAREHOUSE tb_test_wh SET warehouse_size = 'XSmall';
```

# Collaboration

Tasty Bytes Financial Analysts have let the business know that there are Trucks in the Raw Point of Sales System Data that are missing Sales for various days.

We will investigate these missing days and leverage the Snowflake Marketplace to enhance our analysis with Weather and Location data.

We will then dive into one example where Hamburg, Germany Trucks were flagged as having zero sales for a few days in February 2022.

## Step 1:  Querying Point of Sales Data for Trends

Create a Snowflake Worksheet

Let's start by kicking off these three queries to initially set our Role, Warehouse and Database context to `accountadmin`, `tb_de_wh`, `tb_101`. With the context set, we will then query our Analytics `orders_v` View to provide a result set of sales for Hamburg, Germany in 2022.

```
Unset
USE ROLE accountadmin;
USE WAREHOUSE tb_de_wh;
USE DATABASE tb_101;

WITH _feb_date_dim AS
    (
    SELECT DATEADD(DAY, SEQ4(), '2022-02-01') AS date FROM
TABLE(GENERATOR(ROWCOUNT => 28))
    )
SELECT
    fdd.date,
    ZEROIFNULL(SUM(o.price)) AS daily_sales
FROM _feb_date_dim fdd
LEFT JOIN analytics.orders_v o
    ON fdd.date = DATE(o.order_ts)
    AND o.country = 'Germany'
    AND o.primary_city = 'Hamburg'
WHERE fdd.date BETWEEN '2022-02-01' AND '2022-02-28'
GROUP BY fdd.date
ORDER BY fdd.date ASC;
```

From what we saw above, it looks like we are missing sales for February 16th through February 21st for Hamburg. Within our first party data there is not much else we can use to investigate what might have caused this.

However, with live data sets and native applications available in the Snowflake Marketplace we can immediately add Weather Metrics to our analysis and determine if severe weather may have been the root cause.

Within this step, we will retrieve a free, public listing provided by Weather Source, who is a leading provider of global weather and climate data.

## Step 2:  Acquiring the Weather Source LLC: frostbyte Snowflake Marketplace Listing

The Snowflake Marketplace is the premier location to find, try, and buy the data and applications you need to power innovative business solutions. In this step, we will be access the Weather Source LLC: frostbyte listing to help drive additional analysis on our Hamburg sales slump.

Please follow the steps below to acquire this listing in your Snowflake Account.

- In the bottom left corner, ensure you are operating as ACCOUNTADMIN
- In the left pane, navigate to 'Data Products' (Cloud Icon) and select 'Marketplace'
- In the search bar, enter: 'frostbyte'
- Select the 'Weather Source LLC: frostbyte' listing and click 'Get'
- Adjust Database name to:'TB_WEATHERSOURCE'
- Grant access to: 'PUBLIC'

Weather Source is a leading provider of global weather and climate data and its OnPoint Product Suite provides businesses with the necessary weather and climate data to quickly generate meaningful and actionable insights for a wide range of use cases across industries.

## Step 2 - Harmonizing First and Third Party Data

With the shared tb_weathersource database in place, please execute this step query to create a harmonized.daily_weather_v View joining two Weather Source tables to our country table on the Countries and Cities that Tasty Bytes Food Trucks operate within. This query will provide a View DAILY_WEATHER_V successfully created..

```
Unset
CREATE OR REPLACE VIEW tb_101.harmonized.daily_weather_v
COMMENT = 'Weather Source Daily History filtered to Tasty Bytes supported
Cities'
    AS
SELECT
    hd.*,
    TO_VARCHAR(hd.date_valid_std, 'YYYY-MM') AS yyyy_mm,
    pc.city_name AS city,
    c.country AS country_desc
FROM tb_weathersource.onpoint_id.history_day hd
JOIN tb_weathersource.onpoint_id.postal_codes pc
    ON pc.postal_code = hd.postal_code
    AND pc.country = hd.country
JOIN tb_101.raw_pos.country c
    ON c.iso_country = hd.country
    AND c.city = hd.city_name;
```

As we see in the View definition above we are joining two of the tb_weathersource Tables within the onpoint_id Schema and then Harmonizing it with our country Table from our tb_101 Database and raw_pos Schema.

This is the sort of operation we typically find in the Harmonized layer or what others may describe as the Silver zone.

## Step 3 - Visualizing Daily Temperatures

With the daily_weather_v View in our Harmonized Schema in place let's take a look at the Average Daily Weather Temperature for Hamburg in February 2022 by executing our next query.

Along the way we will leverage AVG, YEAR and MONTH functions.

```
Unset
SELECT
    dw.country_desc,
    dw.city_name,
    dw.date_valid_std,
    AVG(dw.avg_temperature_air_2m_f) AS avg_temperature_air_2m_f
FROM tb_101.harmonized.daily_weather_v dw
WHERE 1=1
    AND dw.country_desc = 'Germany'
    AND dw.city_name = 'Hamburg'
    AND YEAR(date_valid_std) = '2022'
```

```
    AND MONTH(date_valid_std) = '2' -- February
GROUP BY dw.country_desc, dw.city_name, dw.date_valid_std
ORDER BY dw.date_valid_std DESC;
```

```
115   -- using our Daily Weather History View, let's find the Average Daily Weather Temperature for
116   -- Hamburg in February 2022 and visualize it as a Line Chart
117       --> Chart Type: Line | X-Axis: DATE_VALID_STD(none) | Line: AVG_TEMPERATURE_AIR_2M_F(none)
118   SELECT
119       dw.country_desc,
120       dw.city_name,
121       dw.date_valid_std,
122       AVG(dw.avg_temperature_air_2m_f) AS avg_temperature_air_2m_f
123   FROM harmonized.daily_weather_v dw
124   WHERE 1=1
125       AND dw.country_desc = 'Germany'
126       AND dw.city_name = 'Hamburg'
127       AND YEAR(date_valid_std) = '2022'
128       AND MONTH(date_valid_std) = '2' -- February
129   GROUP BY dw.country_desc, dw.city_name, dw.date_valid_std
130   ORDER BY dw.date_valid_std DESC;
```

Results | Chart

| | COUNTRY_DESC | CITY_NAME | DATE_VALID_STD | AVG_TEMPERATURE_AIR_2M_F |
|---|---|---|---|---|
| 1 | Germany | Hamburg | 2022-02-28 | 36.0480000 |
| 2 | Germany | Hamburg | 2022-02-27 | 36.7970000 |
| 3 | Germany | Hamburg | 2022-02-26 | 38.7640000 |
| 4 | Germany | Hamburg | 2022-02-25 | 38.4970000 |
| 5 | Germany | Hamburg | 2022-02-24 | 42.9980000 |

To further investigate trends, let's utilize Snowsight Charting to create a Line Graph of the Average Temperature over time.

- Chart Type: Line
- X-Axis: DATE_VALID_STD(none)
- Line: AVG_TEMPERATURE_AIR_2M_F(none)

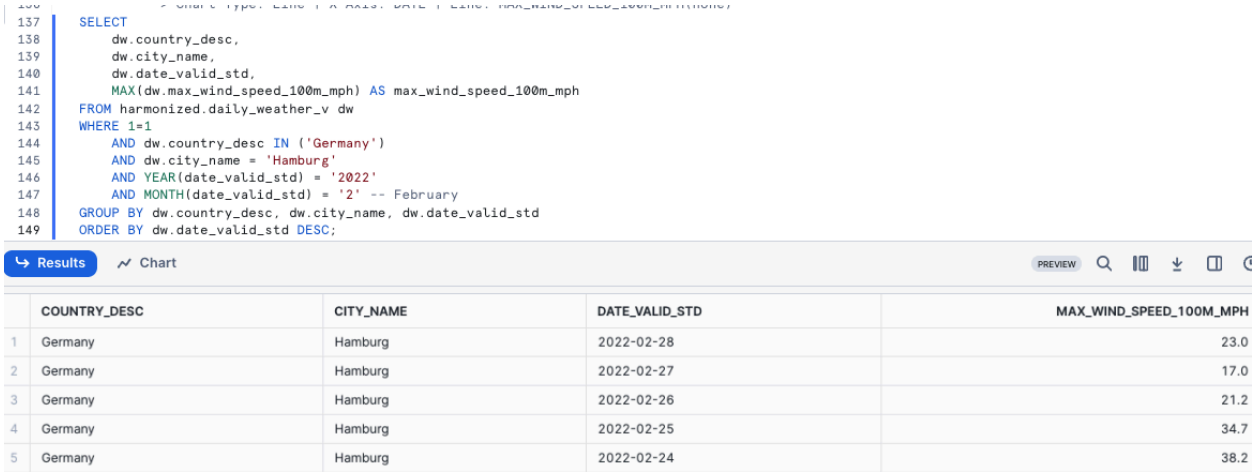

Based on what we saw above, there is nothing really standing out yet as the obvious reason for zero sales days at our trucks. Let's see what else we can find that might explain things in the next step.

# Step 4 - Bringing in Wind Data

As we saw in our previous step, it does not look like Average Daily Temperature is the reason for our zero sales days in Hamburg. Thankfully, Weather Source provides other weather metrics we can dive into as well.

Please now execute the next query where we will leverage our Harmonized View to bring in Wind metrics. In this query we will see the usage of our MAX function.

```
Unset
SELECT
    dw.country_desc,
    dw.city_name,
    dw.date_valid_std,
    MAX(dw.max_wind_speed_100m_mph) AS max_wind_speed_100m_mph
FROM tb_101.harmonized.daily_weather_v dw
WHERE 1=1
    AND dw.country_desc IN ('Germany')
    AND dw.city_name = 'Hamburg'
    AND YEAR(date_valid_std) = '2022'
    AND MONTH(date_valid_std) = '2' -- February
GROUP BY dw.country_desc, dw.city_name, dw.date_valid_std
ORDER BY dw.date_valid_std DESC;
```



```
SELECT
    dw.country_desc,
    dw.city_name,
    dw.date_valid_std,
    MAX(dw.max_wind_speed_100m_mph) AS max_wind_speed_100m_mph
FROM harmonized.daily_weather_v dw
WHERE 1=1
    AND dw.country_desc IN ('Germany')
    AND dw.city_name = 'Hamburg'
    AND YEAR(date_valid_std) = '2022'
    AND MONTH(date_valid_std) = '2' -- February
GROUP BY dw.country_desc, dw.city_name, dw.date_valid_std
ORDER BY dw.date_valid_std DESC;
```

Results | Chart

| | COUNTRY_DESC | CITY_NAME | DATE_VALID_STD | MAX_WIND_SPEED_100M_MPH |
|---|---|---|---|---|
| 1 | Germany | Hamburg | 2022-02-28 | 23.0 |
| 2 | Germany | Hamburg | 2022-02-27 | 17.0 |
| 3 | Germany | Hamburg | 2022-02-26 | 21.2 |
| 4 | Germany | Hamburg | 2022-02-25 | 34.7 |
| 5 | Germany | Hamburg | 2022-02-24 | 38.2 |

Ah ha! The wind for those zero sales days was at hurricane levels. This seems to be a better reason for why our trucks were not able to sell anything on those days.

# Using Snowflake Cortex AI

## What you will learn

How to use Snowflake Cortex for custom tasks like summarizing long-form text into JSON formatted output using prompt engineering and Snowflake Cortex task-specific LLM functions to perform operations like translate text between languages or score the sentiment of a piece of text.

You will also build an interactive Streamlit application running in Snowflake.

In this guide, we'll utilize synthetic call transcripts data, mimicking text sources commonly overlooked by organizations, including customer calls/chats, surveys, interviews, and other text data generated in marketing and sales teams.

## Step 1: Loading Call Transcripts Data

In a new SQL worksheet, run the following SQL commands to select your database, schema and warehouse.

```
Unset
USE SCHEMA tb_101.public;
USE WAREHOUSE tb_de_wh;
```

In the same SQL worksheet, run the following SQL commands to create table CALL_TRANSCRIPTS from data hosted on publicly accessible S3 bucket.

```
Unset
CREATE or REPLACE file format csvformat
  SKIP_HEADER = 1
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  type = 'CSV';

CREATE or REPLACE stage call_transcripts_data_stage
  file_format = csvformat
  url = 's3://sfquickstarts/misc/call_transcripts/';

CREATE or REPLACE table CALL_TRANSCRIPTS (
  date_created date,
```

```
  language varchar(60),
  country varchar(60),
  product varchar(60),
  category varchar(60),
  damage_type varchar(90),
  transcript varchar
) COMMENT = '{"origin":"sf_sit-is", "name":"aiml_notebooks_artic_cortex",
"version":{"major":1, "minor":0}, "attributes":{"is_quickstart":1,
"source":"sql"}}';

COPY into CALL_TRANSCRIPTS
  from @call_transcripts_data_stage;
```

## Step 2 : Using Cortex Functions

Given the data in call_transcripts table, let's see how we can use Snowflake Cortex. It offers access to industry-leading AI models, without requiring any knowledge of how the AI models work, how to deploy LLMs, or how to manage GPU infrastructure.

### Translate

Using Snowflake Cortex function snowflake.cortex.translate we can easily translate any text from one language to another. Let's see how easy it is to use this function....

```
Unset
SELECT snowflake.cortex.translate('wie geht es dir
heute?','de_DE','en_XX');
```

Executing the above SQL should generate *"How are you today?"*

Now let's see how you can translate call transcripts from German to English in batch mode using just SQL.

```
Unset
SELECT transcript,snowflake.cortex.translate(transcript,'de_DE','en_XX')
from call_transcripts where language = 'German';
```

## Sentiment Score

Now let's see how we can use snowflake.cortex.sentiment function to generate sentiment scores on call transcripts.

*Note: Score is between -1 and 1; -1 = most negative, 1 = positive, 0 = neutral*

```
Unset
SELECT transcript, snowflake.cortex.sentiment(transcript) from
call_transcripts where language = 'English';
```

## Summarize

Now that we know how to translate call transcripts in English, it would be great to have the model pull out the most important details from each transcript so we don't have to read the whole thing. Let's see how snowflake.cortex.summarize function can do this and try it on one record.

```
Unset
select transcript,snowflake.cortex.summarize(transcript) as summary from
call_transcripts where language = 'English' limit 1;
```

## Classify Text

This function takes a piece of text and a set of user-provided categories as inputs and returns a predicted category for that text. The function returns a structured JSON-formattet output.

```
Unset

SELECT
transcript,snowflake.cortex.classify_text(transcript,['Refund','Exchange']) as
classification from call_transcripts where language = 'English';
```

# Creating A Streamlit Application

## What You will Learn

- How to Create a Streamlit Application in Snowflake
- How to use Python  to transform data in Snowflake
- How to use popular standard libraries to plot graphs on streamlit

To put it all together, let's create a Streamlit application in Snowflake to display some insights about our customers.

## Step 1 : Setup

1. Click on **Home** - > **Projects** -> **Streamlit** on the left navigation menu

2. Click on **+ Streamlit App** on the top right

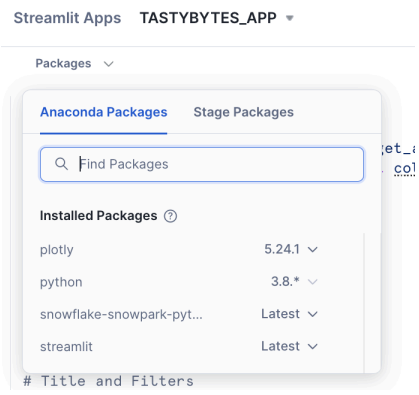
3. Enter **CUSTOMER_LOYALTY_INSIGHTS** App name

4. Select App location (TB_101 as the *database* and PUBLIC as the *schema* ) and App warehouse (COMPUTE_WH)

5. Click on Create

- At this point, you will be provided code for an example Streamlit application

6: At the top of the code editor, select **Packages** , and in the Anaconda Packages Section, select **plotly** to import the **plotly** library into the application



## Step 2 : Python  Imports

Here, we will import the various python packages and modules that we will need to use to run the application, namely:

**streamlit** : - python package to run the streamlit application

**snowpark**:- snowflake's python library for data transformation and manipulation

**pandas**:- dataframe library

**Plotly** :-  plotting library

We will also get access to the snowflake session object using the get_active_session() method. This will help us query Snowflake

To do this, replace the entire sample application code on the left with the following code

```python
# Import python packages

import streamlit as st
from snowflake.snowpark.context import get_active_session
from snowflake.snowpark.functions import col, sum as sum_, avg, count,
date_trunc
import pandas as pd
import plotly.express as px

# Get the current credentials
session = get_active_session()


# Title and Filters
st.title("Customer Loyalty Insights Dashboard")
```

# Step 3:  Plotting Customer Loyalty Insights

In order to get the data for plotting, we will use snowpark to run aggregations on the customer insights table  before returning the final result as a pandas dataframe, which we will use for plotting

Python

```
# Customer Loyalty Insights
st.header("Customer Loyalty Insights")
loyalty_df = session.table("tb_101.analytics.customer_loyalty_metrics_v")

customer_loyalty_data = (loyalty_df.group_by("CITY", "COUNTRY")
                         .agg(sum_("TOTAL_SALES").alias("TOTAL_SPENT"),
                              count("CUSTOMER_ID").alias("NUM_CUSTOMERS"))
                         .to_pandas())

st.plotly_chart(px.scatter(customer_loyalty_data, x='CITY', y='TOTAL_SPENT',
size='NUM_CUSTOMERS', color='COUNTRY', title="Customer Spending by City"))

# Customer Loyalty Distribution
st.header("Customer Loyalty Distribution")

loyalty_distribution = (
    loyalty_df.select("TOTAL_SALES")
    .to_pandas()
)
st.plotly_chart(px.histogram(loyalty_distribution, x='TOTAL_SALES', nbins=100,
title="Customer Loyalty Distribution"))
```