Name: Lam Tran, Khac Minh Dai Vo

Due Date: Thursday, March 6th, 2025

Lab 1: Building a Stock Price Prediction Analytics using Snowflake & Airflow

1. Problem Statement - what application system does your team build and why; why are a database and data pipelines needed as part of the system.

Application System: Tesla Stock Price Prediction System

Our team is building an automated Tesla and Apple stock price prediction system that extracts, transforms, and loads (ETL) Tesla and Apple stock data from an external API into Snowflake. The system then trains a forecasting model using Snowflake's ML.FORECAST and generates future stock price predictions for Tesla.

This system is essential for **financial decision-making**, as investors and analysts rely on accurate stock price forecasts to make informed trading decisions for Tesla stock. By **automating data retrieval, processing, and forecasting**, it eliminates the need for manual data collection, ensuring efficiency and consistency. The integration with **Airflow DAGs** allows for seamless automation of tasks, enabling scalability with minimal human intervention. Additionally, the system provides a **structured approach** to storing, analyzing, and predicting stock movements based on historical trends, enhancing **data-driven insights** for better market analysis.

2. Solution Requirements - what are the requirements for a solution, what will the system do, what are its limitations, how will people use the system.

The system is designed to automate the extraction, processing, storage, and forecasting of Tesla and Apple stock data. It fetches stock price data daily from the Alpha Vantage API and stores it in Snowflake tables for further analysis. The extracted JSON data is transformed into a structured tabular format, including key metrics such as date, open, high, low, close, and volume, while handling missing or corrupted data to maintain integrity. The processed data is stored in dev.raw.tsla_price and dev.raw.aapl_price, with a dedicated view, dev.adhoc.tsla_price_view and dev.adhoc.aapl_price_view, created for training machine learning models. Predicted stock prices are saved in dev.analytics.tsla_price_7days_prediction to facilitate forecasting. Using Snowflake ML.FORECAST, the system trains a forecasting model to generate 7-day future stock price predictions and evaluates its performance with SHOW_EVALUATION_METRICS(). To ensure automation and scalability, Apache Airflow orchestrates ETL tasks and prediction runs at 6 AM from Monday to Friday, integrating error handling and transaction management for data integrity. However, the system has limitations—forecast accuracy depends on the quality and quantity of historical data, ML models assume stationarity in time series, which may not always be valid, and real-time stock price movements can be influenced by external events such as news and market conditions beyond historical trends.

3. Functional Analysis - discuss the functional components of the application system that you are proposing and how they collectively solve the problem. Include database & data pipeline interactions for each.

The **application system** is structured as a **data pipeline** that automates the extraction, transformation, storage, and forecasting of **Tesla and Apple stock prices** using **Snowflake ML** and **Apache Airflow**. The process begins with **data extraction**, where the **Extract Task** pulls stock price data from the **Alpha Vantage API** in JSON format and structures it into a Python dictionary. Next, the **Transform Task** converts the raw JSON into a structured list of records with key columns such as **date, open, high, low, close, and volume**, making it suitable for database insertion. The **Load Task** then stores this processed data in the **dev.raw.tsla_price** table in **Snowflake**, ensuring **data integrity through transaction management (BEGIN/COMMIT/ROLLBACK)**.

For **stock price forecasting**, the **Train Task** prepares the data by creating a **view (dev.adhoc.tsla_price_view and dev.adhoc.tsla_price_view)** and runs **Snowflake ML.FORECAST** to train a **time-series model** on Tesla's and Apple's historical stock data. The **SHOW_EVALUATION_METRICS()** function is used to assess the model's accuracy. Once trained, the **Predict Task** generates **7-day future stock price forecasts**, stores the predictions in **dev.adhoc.tsla_price_forecast**, and creates a **final table (dev.analytics.tsla(appl)_price_7days_prediction)** that merges both **historical and predicted stock prices**. The **RESULT_SCAN(LAST_QUERY_ID())** function is used to capture forecast results efficiently.

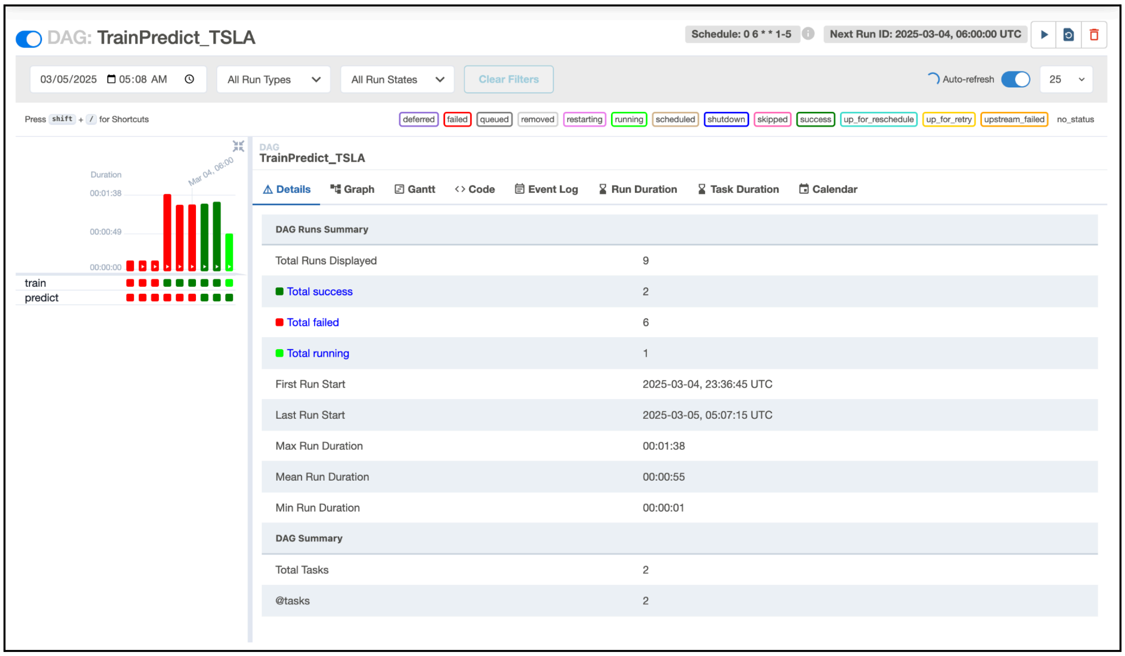
To ensure full automation, **Apache Airflow** is used to **schedule and orchestrate** the entire ETL and prediction pipeline. The **Airflow DAG** ensures a **sequential execution flow**, running the **Extract → Transform → Load → Train → Predict** tasks in order, allowing for efficient and scalable stock price forecasting with minimal manual intervention.

4. Tables structure, Screenshots, Python codes, and SQL queries

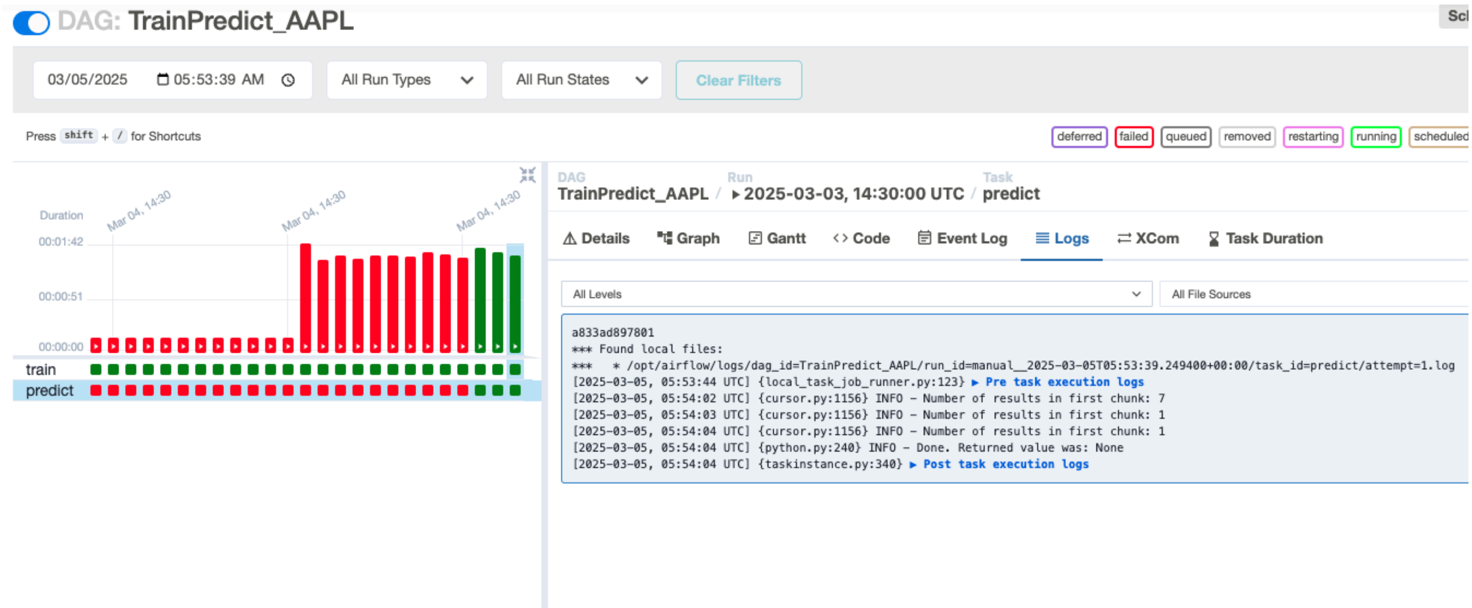
Table Name	Description
`dev.raw.tsla_price` `dev.raw.aapl_price`	Stores raw Tesla and Apple stock data extracted from Alpha Vantage API
`dev.adhoc.tsla_price_view` `dev.adhoc.aapl_price_view`	A view used for ML model training
`dev.adhoc.tsla_price_forecast` `dev.adhoc.aapl_price_forecast`	Stores forecasted Tesla and Apple stock prices
`dev.analytics.tsla_price_7days_prediction` `dev.analytics.aapl_price_7days_prediction`	Final table with historical & predicted stock prices

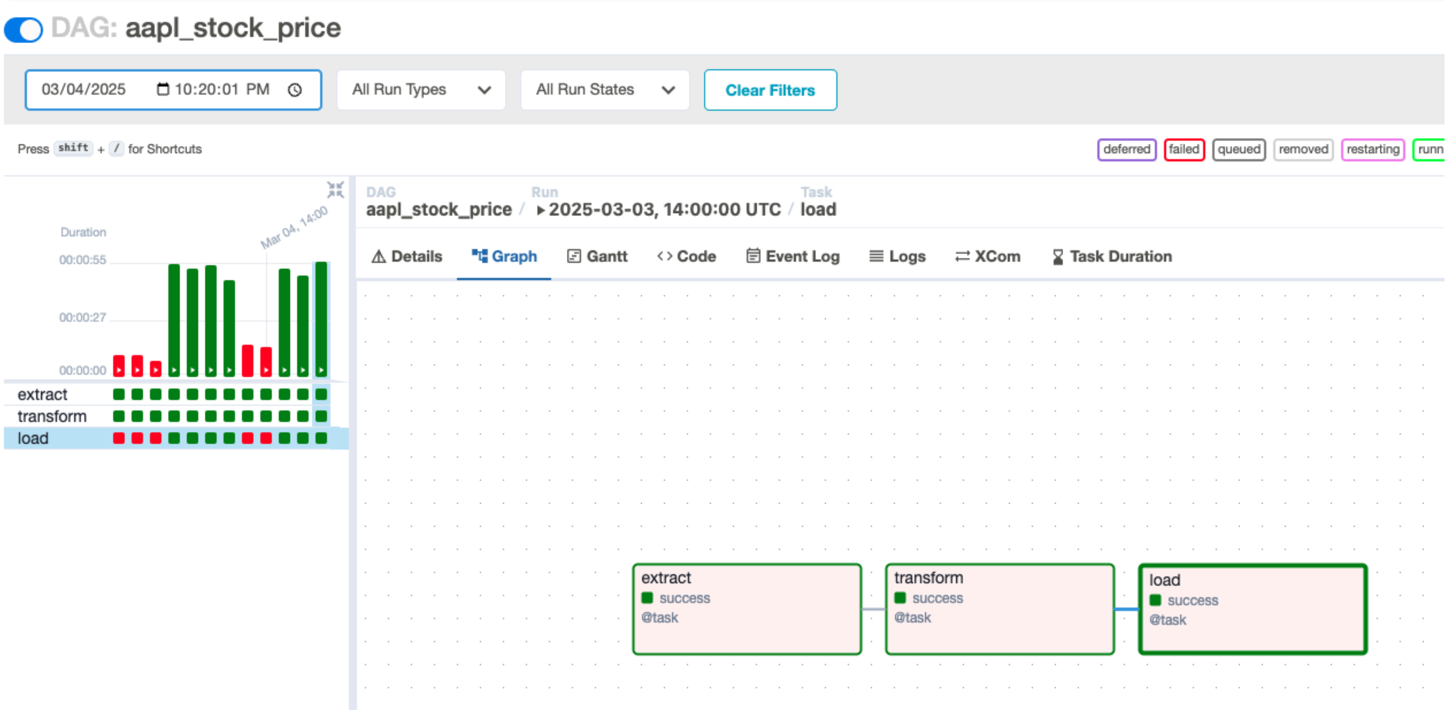
Result in airflow:

- **Tesla:**



- **Apple:**


DAG: TrainPredict_AAPL
03/05/2025 05:53:39 AM
All Run Types
All Run States
Clear Filters
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
train
predict
TrainPredict_AAPL / 2025-03-03, 14:30:00 UTC / predict
Details
Graph
Gantt
Code
Event Log
Logs
XCom
Task Duration
All Levels
All File Sources
a833ad897801
*** Found local files:
*** * /opt/airflow/logs/dag_id=TrainPredict_AAPL/run_id=manual__2025-03-05T05:53:39.249400+00:00/task_id=predict/attempt=1.log
[2025-03-05, 05:53:44 UTC] {local_task_job_runner.py:123} ▶ Pre task execution logs
[2025-03-05, 05:54:02 UTC] {cursor.py:1156} INFO - Number of results in first chunk: 7
[2025-03-05, 05:54:03 UTC] {cursor.py:1156} INFO - Number of results in first chunk: 1
[2025-03-05, 05:54:04 UTC] {cursor.py:1156} INFO - Number of results in first chunk: 1
[2025-03-05, 05:54:04 UTC] {python.py:240} INFO - Done. Returned value was: None
[2025-03-05, 05:54:04 UTC] {taskinstance.py:340} ▶ Post task execution logs


DAG: aapl_stock_price
03/04/2025 10:20:01 PM
All Run Types
All Run States
Clear Filters
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
Duration
00:00:55
00:00:27
00:00:00
extract
transform
load
aapl_stock_price / 2025-03-03, 14:00:00 UTC / load
Details
Graph
Gantt
Code
Event Log
Logs
XCom
Task Duration
extract
success
@task
transform
success
@task
load
success
@task

- **Tesla code:**

Python Code (ETL - Extract, Transform, Load)

```
@task
def extract(url):
  r = requests.get(url)
  data = r.json()
  return data

@task
def transform(data):
  results = []
  for d in data["Time Series (Daily)"]:
    stock_info = data["Time Series (Daily)"][d]
    stock_info['date'] = d
    stock_info['symbol'] = 'TSLA'
    results.append(stock_info)
  return results

@task
def load(cur, results, target_table):
  cur.execute("BEGIN;")
  cur.execute(f"""
    CREATE OR REPLACE TABLE {target_table}(
      date TIMESTAMP_NTZ PRIMARY KEY,
      symbol VARCHAR(10),
      open FLOAT, high FLOAT, low FLOAT,
      close FLOAT, volume INT
    )
  """)

  for i in results:
    sql = f"""
      INSERT INTO {target_table} VALUES
      (TO_TIMESTAMP_NTZ('{i['date']}', 'YYYY-MM-DD'), '{i['symbol']}',
      {i['1. open']}, {i['2. high']}, {i['3. low']}, {i['4. close']},
{i['5. volume']})
    """
    cur.execute(sql)

  cur.execute("COMMIT;")
```

Python Code (Training & Predicting)

```
@task
def train(cur, train_input_table, train_view, forecast_function_name):
    create_view_sql = f"CREATE OR REPLACE VIEW {train_view} AS SELECT
DATE, CLOSE, SYMBOL FROM {train_input_table};"
    create_model_sql = f"CREATE OR REPLACE SNOWFLAKE.ML.FORECAST
{forecast_function_name} (...)"

    cur.execute(create_view_sql)
    cur.execute(create_model_sql)
    cur.execute(f"CALL
{forecast_function_name}!SHOW_EVALUATION_METRICS();")

@task
def predict(cur, forecast_function_name, train_input_table,
forecast_table, final_table):
    cur.execute(f"CALL {forecast_function_name}!FORECAST(...)")
    cur.execute(f"CREATE OR REPLACE TABLE {forecast_table} AS SELECT *
FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));")
```

<u>SQL Queries in Snowflake</u>

```
Creating Training View
CREATE DATABASE IF NOT EXISTS dev;
CREATE SCHEMA IF NOT EXISTS raw;
CREATE SCHEMA IF NOT EXISTS analytics;

CREATE OR REPLACE VIEW dev.adhoc.tsla_price_view AS
SELECT DATE, CLOSE, SYMBOL FROM dev.raw.tsla_price;

-- Running ML Forecast
CALL dev.analytics.predict_tsla_price!FORECAST(FORECASTING_PERIODS => 7,
CONFIG_OBJECT => {'prediction_interval': 0.95});

-- Storing Predictions
CREATE OR REPLACE TABLE dev.adhoc.tsla_price_forecast AS
SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

Apple code:

Python Code (ETL - Extract, Transform, Load)

```python
# Extract Task
@task
def extract(url):
    """Fetches stock price data from Alpha Vantage API."""
    r = requests.get(url)
    data = r.json()
    return data

# Transform Task
@task
def transform(data):
    """Transforms raw API data into a structured format."""
    results = []
    for d in data["Time Series (Daily)"]:
        stock_info = data["Time Series (Daily)"][d]
        stock_info['date'] = d
        stock_info['symbol'] = 'AAPL'
        results.append(stock_info)
    return results
```

```python
# Load Task
@task
def load(results, target_table):
    """Loads transformed data into Snowflake."""
    cur = return_snowflake_conn()

    try:
        cur.execute("BEGIN;")

        # Ensure the table exists
        cur.execute(f"""
        CREATE OR REPLACE TABLE {target_table}(
            date DATE PRIMARY KEY,
            symbol VARCHAR(10),
            open FLOAT,
            high FLOAT,
            low FLOAT,
            close FLOAT,
            volume INT
        )""")

        for i in results:
            date = datetime.strptime(i['date'], '%Y-%m-%d').date()
            symbol = i['symbol']
            open_price = float(i['1. open'])
            high_price = float(i['2. high'])
            low_price = float(i['3. low'])
            close_price = float(i['4. close'])
            volume = int(i['5. volume'])

            sql = """INSERT INTO dev.raw.aapl_price (date, symbol, open, high, low, close, volume)
                     VALUES (%s, %s, %s, %s, %s, %s, %s)"""
            cur.execute(sql, (date, symbol, open_price, high_price, low_price, close_price, volume))

        cur.execute("COMMIT;")
        print(" Data successfully loaded into Snowflake.")

    except Exception as e:
        cur.execute("ROLLBACK;")
        print(f" Error loading data: {e}")
        raise e
```

```python
with DAG(
    dag_id='aapl_stock_price',
    start_date=datetime(2025, 3, 4),
    catchup=False,
    tags=['ETL'],
    schedule='0 14 * * 1-5'
) as dag:

    target_table = 'dev.raw.aapl_price'

    url = Variable.get('aapl_url')

    extract_task = extract(url)
    transform_task = transform(extract_task)
    load_task = load(transform_task, target_table)

    extract_task >> transform_task >> load_task
```

Python Code (Training & Predicting)

```python
@task
def train(cur, train_input_table, train_view, forecast_function_name):
    """
     - Create a view with training related columns
     - Create a model with the view above
    """

    create_view_sql = f"""CREATE OR REPLACE VIEW {train_view} AS SELECT
        DATE, CLOSE, SYMBOL
        FROM {train_input_table};"""

    create_model_sql = f"""CREATE OR REPLACE SNOWFLAKE.ML.FORECAST {forecast_function_name} (
        INPUT_DATA => SYSTEM$REFERENCE('VIEW', '{train_view}'),
        SERIES_COLNAME => 'SYMBOL',
        TIMESTAMP_COLNAME => 'DATE',
        TARGET_COLNAME => 'CLOSE',
        CONFIG_OBJECT => {{ 'ON_ERROR': 'SKIP' }}
    );"""

    try:
        cur.execute(create_view_sql)
        cur.execute(create_model_sql)
        cur.execute(f"CALL {forecast_function_name}!SHOW_EVALUATION_METRICS();")
    except Exception as e:
        print(e)
        raise
```

```python
@task
def predict(cur, forecast_function_name, train_input_table, forecast_table, final_table):
    """
    - Generate predictions and store the results in `forecast_table`.
    - Union the predictions with historical data and create `final_table`.
    """
    try:
        cur.execute(f"""
            CALL {forecast_function_name}!FORECAST(
                FORECASTING_PERIODS => 7,
                CONFIG_OBJECT => {{'prediction_interval': 0.95}}
            );
        """)

        cur.execute(f"""
            CREATE OR REPLACE TABLE {forecast_table} AS
            SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
        """)

        cur.execute(f"""
            CREATE OR REPLACE TABLE {final_table} AS
            SELECT SYMBOL, DATE, CLOSE AS actual, NULL AS forecast, NULL AS lower_bound, NULL AS upper_bound
            FROM {train_input_table}
            UNION ALL
            SELECT replace(series, '"', '') as SYMBOL, ts as DATE, NULL AS actual, forecast, lower_bound, upper_bound
            FROM {forecast_table};
        """)

    except Exception as e:
        print(f"Error in predict task: {e}")
        raise
```

```python
with DAG(
    dag_id = 'TrainPredict_AAPL',
    start_date = datetime(2025,3,4),
    catchup=False,
    tags=['ML', 'ELT'],
    schedule = '30 14 * * 1-5'
) as dag:

    train_input_table = "dev.raw.aapl_price"
    train_view = "dev.adhoc.aapl_price_view"
    forecast_table = "dev.adhoc.aapl_price_forecast"
    forecast_function_name = "dev.analytics.predict_aapl_price"
    final_table = "dev.analytics.aapl_price_7days_prediction"
    cur = return_snowflake_conn()

    train(cur, train_input_table, train_view, forecast_function_name)
    predict(cur, forecast_function_name, train_input_table, forecast_table, final_table)
```

Queries in Snowflake

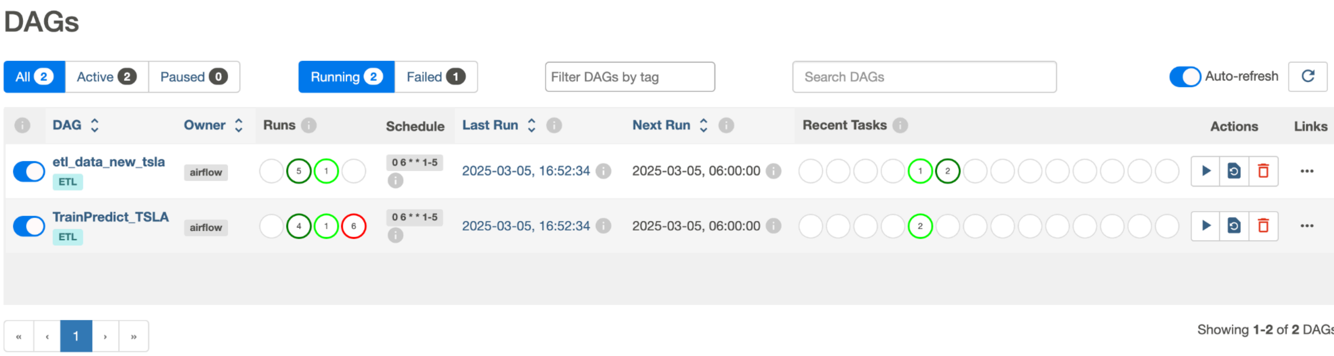
```
CREATE DATABASE IF NOT EXISTS dev;
CREATE SCHEMA IF NOT EXISTS raw;
CREATE SCHEMA IF NOT EXISTS analytics;
CREATE SCHEMA IF NOT EXISTS dev.adhoc;


SELECT * FROM DEV.RAW.AAPL_PRICE; --Check the apple_price table
SELECT CURRENT_ACCOUNT() --Check the account number


-- create the view for training
CREATE OR REPLACE VIEW dev.adhoc.aapl_price_view AS
SELECT DATE, CLOSE, SYMBOL
FROM dev.raw.aapl_price;

-- Use for Forcasting
CREATE OR REPLACE SNOWFLAKE.ML.FORECAST dev.analytics.predict_tsla_price (
    INPUT_DATA => SYSTEM$REFERENCE('VIEW', 'dev.adhoc.aapl_price_view'),
    SERIES_COLNAME => 'SYMBOL',
    TIMESTAMP_COLNAME => 'DATE',
    TARGET_COLNAME => 'CLOSE',
    CONFIG_OBJECT => { 'ON_ERROR': 'SKIP' }
);

CALL dev.analytics.predict_appl_price!FORECAST(
    FORECASTING_PERIODS => 7,
    CONFIG_OBJECT => { 'prediction_interval': 0.95 }
);
```

Results Chart

	SERIES	TS	FORECAST	LOWER_BOUND	UPPER_BOUND
1	"AAPL"	2025-03-05 00:00:00.000	235.203408897	229.629801858	241.553474625
2	"AAPL"	2025-03-06 00:00:00.000	234.912584129	225.887759189	243.563215854
3	"AAPL"	2025-03-07 00:00:00.000	234.693066034	223.518546871	244.748432452
4	"AAPL"	2025-03-10 00:00:00.000	234.969396517	222.270243197	248.328622937
5	"AAPL"	2025-03-11 00:00:00.000	236.242959181	222.31265472	249.72770327
6	"AAPL"	2025-03-12 00:00:00.000	235.481096403	221.458783136	248.877915255
7	"AAPL"	2025-03-13 00:00:00.000	235.189862807	218.842160196	251.231806607

Screenshot of the Airflow Web UI showing two pipelines

Tesla:



Apple: