LTIMindtree

Point of View

Snowflake Python UDF

A new world of opportunities

If you are like me, a data engineer working on the Snowflake Data Cloud with a bit of a penchant for data science, you will find in Python UDF a game-changer that opens the door to a world full of possibilities.

by **Srinivasaraghavan Sundar**

Pitfalls of multiplatform pipelines

We live in a world driven by data. The key to success in any industry today is firmly in the hands of those who can best utilize and govern their data. This age of data renaissance has seen the rise of many platforms that can aid an organization through its data lifecycle. However, the process of managing, auditing, and governing these platforms is a hassle. When used together, these platforms can entangle to form complex pipelines with multiple nodes.

Moreover, it is almost impossible to obtain optimal results without operating each node in a specific configuration. Each node is piloted by separate teams who specialize in that domain. They seldom have a fundamental appreciation of what goes on in any other platform. This could in turn give rise to data silos, which, in time, can delay your decision-making ability.

So, what is Snowflake's solution to this terrifying predicament? Extensibility Features. Python UDFs along with Snowpark and External Functions can help organizations do more with Snowflake, thereby minimizing the number of platforms, reducing data movement, and building streamlined pipelines. Further, clumsy pipelines can be eliminated by minimizing the number of platforms in use.

Python UDFs are functions written in Python and called like a built-in function on Snowflake. They combine large open-source libraries on Python along with the scalable compute capability provided by Snowflake. Complex data science and data engineering problems can now be solved with Snowflake.

Python UDF Data Flow

As seen in the above diagram, Python UDF brings the capability of running Python functions within SQL. Python UDF accepts 0 or more parameters. For each set of parameters passed into a UDF, a scalar value is returned. For a given Python file, a handler function is defined and called by the Python UDF.
A select statement is used to call the UDF.

Python UDF - Why it matters?

There are a number of benefits of using Python UDF. Some of these include:

1. Ability to reduce complex pipelines to a simplified form
2. Implement Machine Learning within Snowflake
3. Leverage Snowflake scalable compute to run your scheduled python jobs

At this point, I'm sure you're excited as well, hopefully, you have a few questions about what Python UDF is capable of. So, let's dive in and look at what this is like in practice with an example.

Time series analysis using Snowflake Python UDF

Figure 2: Timeseries Analysis

Dataset Description

We have a time-series dataset out here which contains the ratio of Exports to Imports of India from 1990 to January 2022.

This dataset has been obtained from FRED Economic Data. You can download the dataset from the following link (https://-fred.stlouisfed.org/series/XTE-ITT01INM156S)

An inadequate architecture

Let us assume that currently, you are making use of this dataset to perform a forecast on what the export-import ratio would be like for the next 12 months. Currently, you make use of a virtual machine to perform the forecasting, but your data is in Snowflake. Now, because this data gets appended each month, you have a job scheduled using Cron scheduler/Airflow to re-run this model once a month.

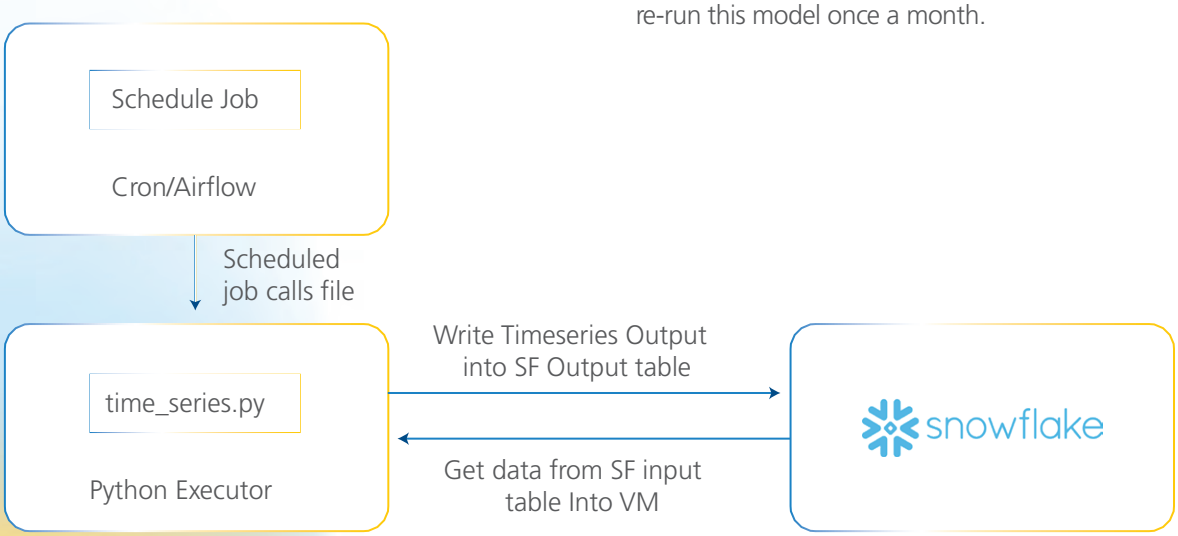

Figure 3: Current Architecture

This is a very typical example that many people would have set up at this very moment. Let us try to bring the entire pipeline within Snowflake.

Deriving Insights

Upon downloading the dataset, be sure to upload it onto your Snowflake Cloud Data Platform. Once that's done, we are ready to begin. Let us first look at the data present.

	...	DATE	RATIO
1		1990-01-01	79.867647298
2		1990-02-01	73.413283437
3		1990-03-01	83.131545751
4		1990-04-01	75.557100889
5		1990-05-01	87.765378661
6		1990-06-01	83.687921974
7		1990-07-01	86.204856764

Figure 4: Sample Data

```
select * from "COE_PRACTISE_DB"."PUBLIC"."ARIMA_TEST"
```

Figure 5 : Timeseries data visualized

Writing the UDF

We create a python UDF as follows:

```
create or replace function sarimax_main(DATE VARIANT, RATIO VARIANT)
returns variant
language python
runtime_version = '3.8'
packages = ('pandas==1.2.3','statsmodels==0.12.2')
handler = 'sarimax_func'
as
$$
import pandas as pd
import statsmodels.api as sm
from datetime import datetime
from dateutil.relativedelta import relativedelta
#define the handler function which accepts 2 parameters, date and ratio in the form of a list.
def sarimax_func(date,ratio):
#convert the two lists into a dictionary and then to a pandas dataframe.
    dict={'DATE':date,'RATIO':ratio}
    df=pd.DataFrame(dict)
#define a seasonal arima model
    model=sm.tsa.statespace.SARIMAX(df['RATIO'],order=(1, 1, 1),seasonal_order=(1,1,1,12))
    results=model.fit()
#the first prediction must occur right after the last value in the dataframe.
    start_pred=len(df)+1
#the last prediction is 12 months after the last date
    last_date=df["DATE"].iloc[-1]
    dates=[]
    for i in range(1,13):
```

```
                add_month = datetime.strptime(last_date,"%Y-%m-%dT08:00:00.000Z").date() +
relativedelta(months=i)
        dates.append(add_month.strftime("%Y-%m-%d"))
    pred_array=results.predict(start=start_pred,end=start_pred+11,dynamic=True)
    predictions=pred_array.tolist()
    return [predictions,dates]
$$;
```

Over here, we use the SARIMAX algorithm to perform time series analysis. This UDF takes in 2 parameters, DATE and RATIO and returns a list containing forecasted values for the next 12 months.

We can run the SELECT sarimax_main (DATE VARIANT, RATIO VARIANT); to obtain the forecast. But wait a second. How do we go about passing the data from the table into the UDF? Is there a way to call SQL statements from within the UDF? Well, no, UDFs act like a scalar function in that you pass parameters and it returns a value or a list of values.

Constructing the Wrapper

In order to pass the data from the table into the UDF, we need to create a Stored Procedure that wraps around the UDF. Let us create a stored procedure using javascript:

```
CREATE OR REPLACE PROCEDURE timeseries_prediction()
RETURNS variant
LANGUAGE JAVASCRIPT
execute as caller
AS
$$
//create a table to hold the timeseries prediction
var query0="create or replace transient table SARIMAX_OUTPUT (DATE STRING,RATIO
FLOAT);"
var result=snowflake.execute({sqlText:query0});
//select * from the input table
var query1=`select * from "RATIO_OF_EXPORTS_TO_IMPORTS";`
var result_array = snowflake.execute({sqlText:query1});
//collect the output in 2 lists.
```

```
date=[]
ratio=[]
while(result_array.next())
{
var date_val = result_array.getColumnValue(1);
date.push(JSON.stringify(date_val));
var ratio_val = result_array.getColumnValue(2);
ratio.push(ratio_val);
}
//call the python udf and pass date array and ratio array
query2=`select sarimax_main (parse_json('[`+date+`]'),parse_json('[`+ratio+`]'));`
var result_array2=snowflake.execute({sqlText:query2});
function_output=[]
while(result_array2.next())
{
var returned_values = result_array2.getColumnValue(1);
function_output.push(returned_values);
}
var forecast=function_output[0][0]
var dates=function_output[0][1]
//Insert into the output table created earlier.
for (let i = 0; i < dates.length; i++) {
query3=`INSERT INTO SARIMAX_OUTPUT (DATE,RATIO) VALUES
('`+dates[i]+`','`+forecast[i]+`')`
var result_array3=snowflake.execute({sqlText:query3});
}
return "SUCCESS";
$$;
```

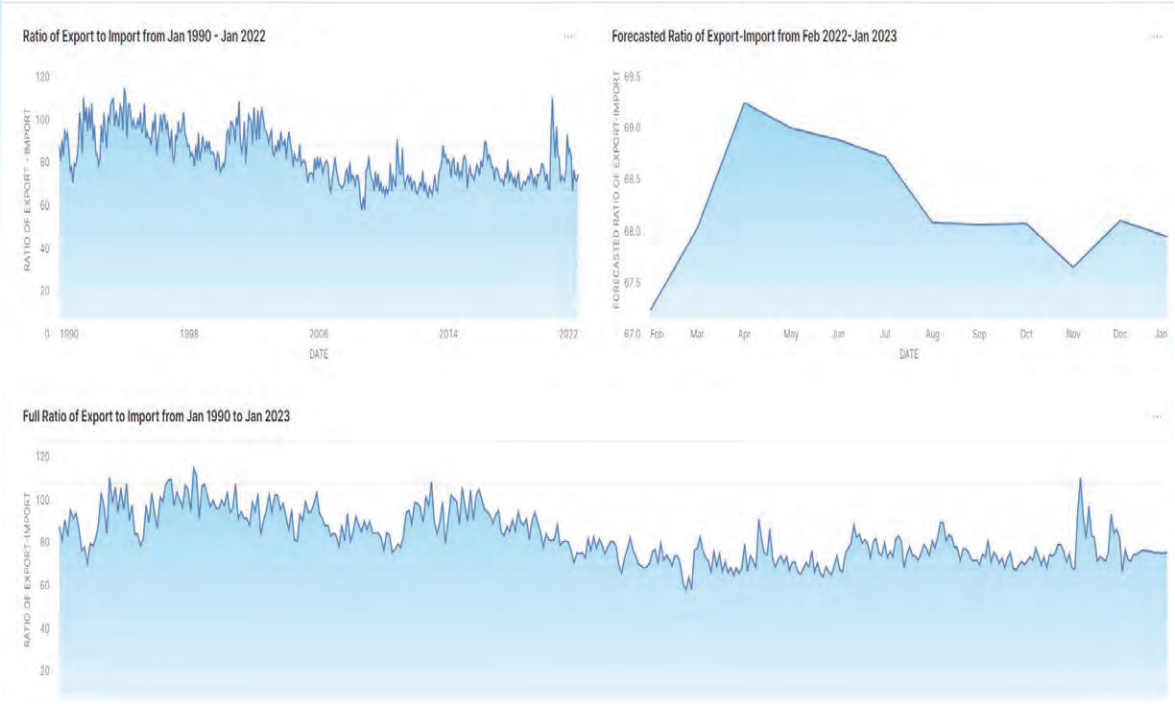

Figure 6 : Insights

In the above diagram, we make use of Snowsight to build a dashboard in order to visualize and draw insights from our data. We can see the predicted values from Feb 2022 to Jan 2023 and derive our insights from it.

Completing with a Task

And now, we can create a task on top of this stored procedure to schedule it to run as per our requirement.

```
create task timeseries_task
schedule = 'USING CRON 0 0 1 * * America/Los_Angeles'
as
  call timeseries_prediction();
```

A Simplified, Bright Future

As you can observe, we took a pipeline wherein data moved from an SF table => Spark cluster/VM=>SF table, all while being scheduled by Airflow/Linux Cron into a pipeline that only makes use of Snowflake as shown below:

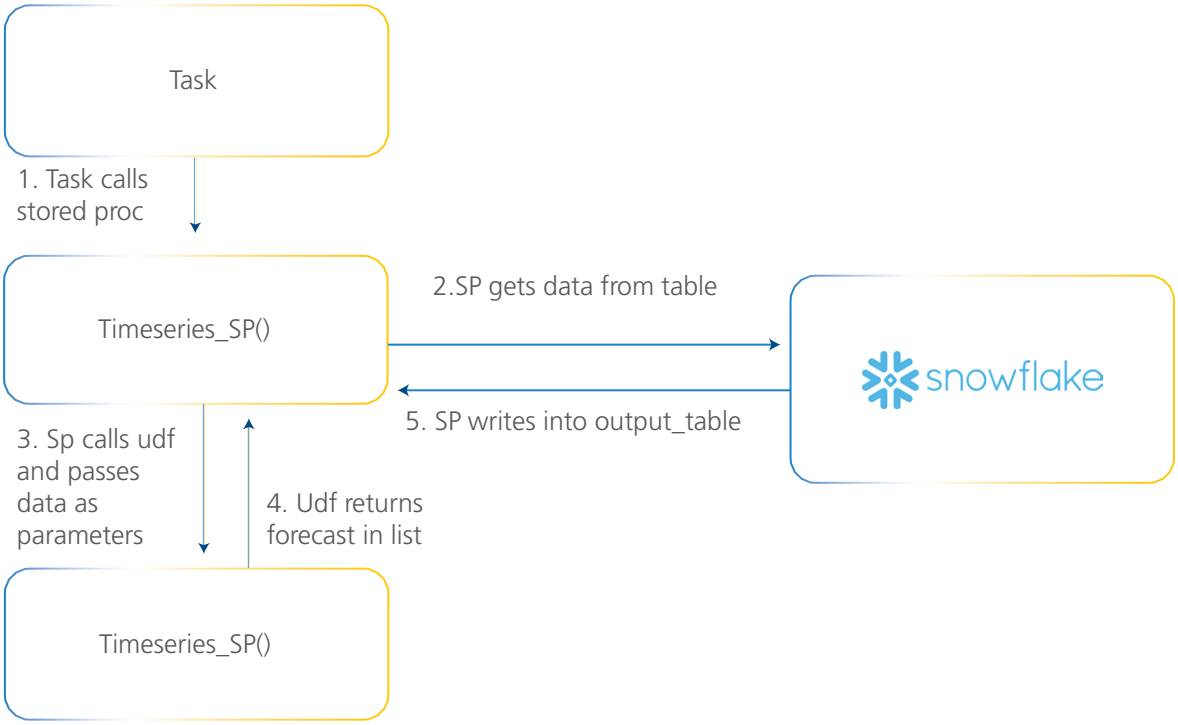

Figure 7: Improved Architecture

You can see how Python UDF straddled with data in multiple locations to achieve what was required for the user seamlessly. Multiple platforms are soon to be a relic of the past. While relics are meant to be remembered, no good can come from using them today.

We have already begun to power our products with Snowflakes extensibility features. This truly is game changing for us, and I am sure, it would be for you as well. Python UDFs are an important tool that shouldn't be overlooked.

About the Author

Srinivasaraghavan Sundar

Senior Data Engineer, Snowflake Center of Excellence, LTIMindtree

Srinivas is currently a part of LTIMindtree's Snowflake COE in the capacity of a Senior Data Engineer with a natural penchant for Data Science. Srinivas usually spends his time either picking up new skills or honing and deep diving into his areas of interest in Data Engineering & Data science. He channels his creativity into finding unique solutions for technical problems and writing content to help our readers.