Author: Parag Jain
Summary: Guide to getting started with Snowpark Python API and UDF
Tags: Getting Started, Data Science, Data Engineering, Snowpark Python, Python UDF
Date: March, 2022

Getting Started with Snowpark Python and UDF

Overview

Using the Snowpark Python API you can query and manipulate data by writing code that uses objects (like a data frame) rather than SQL statements. Snowpark is designed to make building complex data pipelines easy, allowing you to interact with Snowflake directly without moving data. When you use the Snowpark API, the library uploads and runs our code in Snowflake so that you don't need to move the data to a separate system for processing.

Currently, Snowpark is a private preview feature on selected accounts. Talk to your snowflake account representative to request access to this feature.

What You'll Build

An end to end Python Data Science use case using Snowpark python library and python user defined function (UDF) to load, clean, process data and deploy a ML model in snowflake for inferencing

What You'll Learn

§ How to create a data frame that loads data from a stage
§ How to clean and transform data for data science model training
§ How to create a user-defined function for your python code

Prerequisites

§ Familiarity with Python and ML libraries like Pandas, numpy and scikit-learn. These libraries are part of the requirements.txt file in the code
§ A Snowflake account hosted on Amazon Web Services (AWS) or Microsoft Azure.
§ Python 3.8.x
§ Jupyter Notebook or pip install notebook
§ Download the Snowpark library. The library is available for download through Google Drive. You should have received a link to this folder in email (ask administrator or snowflake account representative)

§ Enable this preview feature from UI (Ask your account admin)

To install the snowpark client library follow the below steps -

For MacOS and Linux:

```
pip install 'snowflake_snowpark_python-[x.x]-py3-none-any.whl[pandas]'
```

For Windows:

```
pip install "snowflake_snowpark_python-[x.x]--py3-none-any.whl[pandas]"
```

Snowpark can help you streamline data pipelines for data science and helps you deploy code closer to your data. So, let's get started.

Setup the example code

Download the code in this folder and unzip it to a folder name *snowpark_getting_started*

```
unzip snowpark_getting_started.zip or use any zip utility
```

Go to the folder called **customer-churn-prediction** and before we run the notebooks in this guide, you need to have a few basic machine learning libraries. Let's install those.

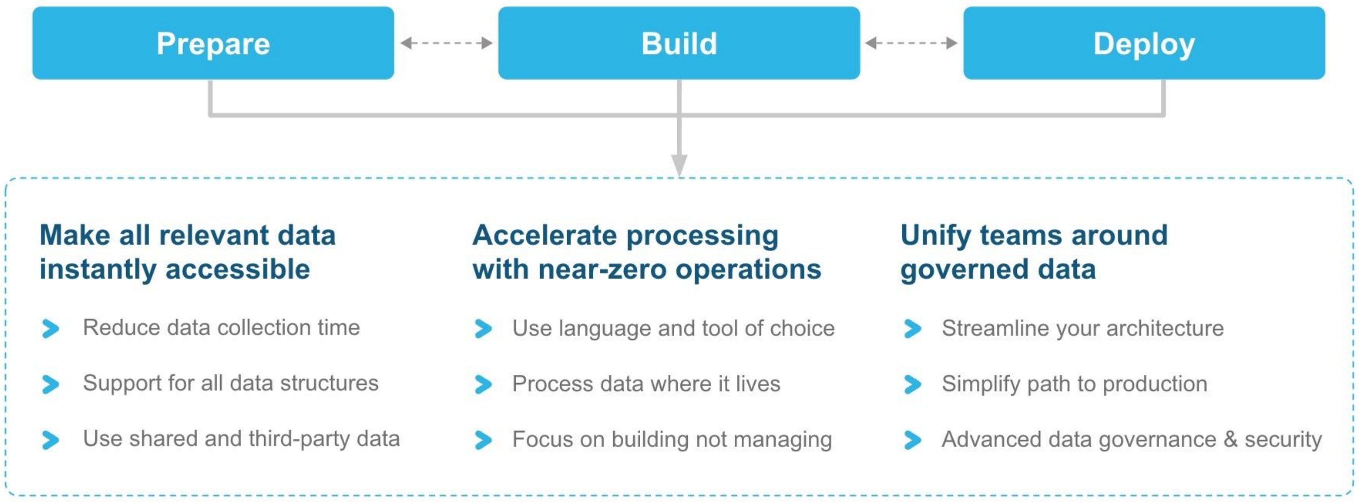
```
cd snowpark_getting_started/customer-churn-prediction

pip install -r requirements.txt
```

Problem Statement

You are a data engineering and data science team member at a Telecom company and tasked to build an end to end data pipeline and ML model in snowflake to support customer churn analysis. For this we have some customer demographic and billing data. We will ingest and analyze this data using Snowpark and will also build a ML model to deploy in snowflake.

SNOWFLAKE FOR DATA SCIENCE


Prepare
Build
Deploy
Make all relevant data instantly accessible
Reduce data collection time
Support for all data structures
Use shared and third-party data
Accelerate processing with near-zero operations
Use language and tool of choice
Process data where it lives
Focus on building not managing
Unify teams around governed data
Streamline your architecture
Simplify path to production
Advanced data governance & security

Why Snowpark?

§ Snowpark help ML and data engineers to build an end to end pipeline within snowflake and reduce the overall cost and technology footprints so that there are less point of failures
§ It helps to simplify the data pipeline by ingesting and processing the data within snowflake
§ Multiple developers like SQL and Python developers can collaborate using snowpark
§ Snowpark along with Python UDF makes data scientist's life easier by seemlessly deploying the serialized python output as a function in snowflake

Data loading and transformations

After the initial setup you can start loading data from this folder to your snowflake account. Let's set up your account details.

```
% cd cd snowpark_getting_started/customer-churn-prediction

% jupyter notebook
```

Go to config.py and update your snowflake account details.

```
snowflake_conn_prop = {
  "account": "ACCOUNT",
  "user": "USER",
  "password": "PASSWORD",
  "database": "DBNAME",
  "schema": "SCHEMANAME",
  "warehouse": "WAREHOUSE"
}
```

open the first notebook (01_customer_churn ~) and in the first cell you will see some boiler plate code to import a few libraries that you might be familiar with and snowflake.snowpark libraries.

```
from snowflake.snowpark.session import Session
from snowflake.snowpark import functions as F
from snowflake.snowpark.types import *
```

Let’s understand this, Snowpark is a standalone library that lets you work in any IDE, notebook or tools/platform to interact and build ML and data engineering processes on top of snowflake without requiring you to own a beefy virtual/on-prem server. It will let you translate the python functional programming code to SQL and pushdown to snowflake servers for execution. Your code is lazily executed and will not consume any snowflake resources unless you run execution methods like show(), collect(), write etc.

This helps you build faster and closer to your data and reduces many complexities that come with a ML engineering and data science world.

Here the Session class let you build a session with your creds to snowflake. Snowpark has many functions to interact with the data and let you convert Snowpark data frame to Pandas or vice versa for more analytical or exploratory analysis

There are many prebuilt types in snowflake and Snowpark that let you define your schema for data. Next step in this notebook is to load this data in a snowflake table and try to apply basic transformations as well like checking Nulls, defining columns etc.

We have our raw parquet data saved in raw_telco~ file, and you can load it using a pandas data frame or schema reader and snowpark data frame. But first let's put the file in the snowflake stage.

```
session.sql("create or replace stage rawdata DIRECTORY = (ENABLE = TRUE);").collect()
session.file.put("raw_telco_data.snappy.parquet","@rawdata")
```

create a table using inferred file schema and load the data

```
session.sql("CREATE FILE FORMAT if not exists MY_PARQUET_FORMAT TYPE = PARQUET;")
session.sql("CREATE \
     TABLE IF NOT EXISTS RAW_PARQUET_DATA USING TEMPLATE ( \
     SELECT \
              ARRAY_AGG(OBJECT_CONSTRUCT(*)) \
     FROM \
              TABLE( \
              INFER_SCHEMA( \
              LOCATION => '@rawdata/raw_telco_data.snappy.parquet', \
              FILE_FORMAT => 'MY_PARQUET_FORMAT' \
              ) \
              ) \
     );  ").collect()
```

```
dfRaw =
session.read.option("compression","snappy").parquet("@rawdata/raw_telco_data.snappy.parquet")
dfRaw.copy_into_table("RAW_PARQUET_DATA",MATCH_BY_COLUMN_NAME='CASE_SENSITIVE',
FORCE= True)
```

Snowpark for transformations

The Snowpark API provides programming language constructs for building SQL statements. It's a new developer experience which enables us to build code in :-

§ Language of our choice
§ Tool of our choice and
§ Lazy execution to prevent multiple network hops to server

Once the customer data is available in the RAW schema, we can use snowpark to create dimensions and fact tables. We will use the RAW_PARQUET table to create following tables -

§ DEMOGRAPHICS
§ LOCATION
§ STATUS
§ SERVICES

We will also transform and clean the data using Snowpark dataframe API, like :

```
dfDemographics = dfR.select(col("CUSTOMERID"),
                        col("COUNT").alias("COUNT"),
                        translate(col("GENDER"),lit("NULL"),lit("Male")).alias("GENDER"),
                        col("SENIOR CITIZEN").alias("SENIORCITIZEN"),
                        col("PARTNER"),
                            col("DEPENDENTS")
                        )
```

Run all the cells to create the transformed table using snowpark. Check Query history in your snowflake account and see how snowpark helps translate your dataframe code into SQL and pushes it to the snowflake account.

Your queries and operation are lazily executed on a snowflake server, meaning that you don't need a beefy machine or VM to execute the code you prepare. You can use the describe() method to check the SQL generated for your dataframe.

```
dfLoc = session.table("LOCATION")
dfServ = session.table("SERVICES")
dfJoin = dfLoc.join(dfServ,dfLoc.col("CUSTOMERID") == dfServ.col("CUSTOMERID"))
dfResult = dfJoin.select(col("CITY"), col("CONTRACT"),
col("TOTALCHARGES")).groupBy(col("CITY"), col("CONTRACT")).sum(col("TOTALCHARGES"))
```

You can then perform an action on the data frame by calling methods like show(), collect(), write etc. on the data frame to see or execute it.

```
dfResult.show()
```

All the content for the step goes here.

Snowpark for data analysis

Snowpark can help you understand various attributes about your data. You can explore target distribution, cardinality, categorical variables, numerical variables, and create functions to transform this data. Once data is prepared, using our zero-copy cloning feature, data scientists can take a snapshot of the data to get a point in time view of their production data giving you data reproducibility without data leakage or redundant copies of data. Look for this code in the 02~ notebook.

```
def analyse_rare_labels(df, var, rare_perc):
    df = df.copy()
    tmp = df.groupby(var)['CHURNVALUE'].count() / len(df)
    return tmp[tmp < rare_perc]
```

You can also save a train dataset once you are satisfied with data analysis results.

```
snowpark_train_df = session.write_pandas(data[final_cols], 'TELCO_TRAIN_SET',
auto_create_table=True)
```

ML training and deploy

Once you identified the possible variables that you want to train your model on then you can engineer specific features out of these variables using snowpark. You can pull the training dataset in a pandas dataframe and build a pipeline to build your model. This way your new data can go through that same pipeline for inferencing.

A similar technique is described in the 03~ notebook.

```
# Model Pipeline
ord_pipe = make_pipeline(
    FunctionTransformer(lambda x: x.astype(str)) ,
    OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
    )
num_pipe = make_pipeline(
    SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=0),
    MinMaxScaler()
    )

clf = make_pipeline(RandomForestClassifier(random_state=0, n_jobs=-1))
model = make_pipeline(ord_pipe, num_pipe, clf)
# fit the model
model.fit(X_train, y_train)
```

Once your model is trained you can deploy it in snowflake using Python UDF.

```
features = list(X_train.columns)
session.add_packages("scikit-learn==1.0.2", "pandas", "numpy")
@udf(name='predict_churn',is_permanent = True, stage_location = '@MODELSTAGE', replace=True)
def predict_churn(args: list) -> float:
    row = pd.DataFrame([args], columns=features)
    return model.predict(row)
```

ML inferencing using UDF

You can infer batch data or new data arriving using both SQL or python functions in snowflake.

```
new_df.select(new_df.CUSTOMERID,new_df.CHURNVALUE, F.call_udf("predict_churn",
F.array_construct(*features)).alias('PREDICTED_CHURN'))
.write.mode('overwrite').saveAsTable('churn_detection')
```

See more code in 03~ notebook

Conclusion

Snowflake simplifies your path to production with the flexibility to deploy models inside it for scalable compute and enhanced data governance or in an external serving layer to have models readily accessible as endpoints.

Snowflake can be a great solution to meet your bulk ML inference needs while the external deployment option can be used to meet any real-time inference requirements.

What we've covered

§ Setup snowpark for development
§ Connect with snowflake data cloud
§ Prepare data for data science and ML experiments
§ Analyze and transform data to make it ready for model training
§ Feature engineering and model deployment inside snowflake
§ Using Python UDF and model inferencing