

```
In [ ]: import snowflake.snowpark
from snowflake.snowpark import functions as F
from snowflake.snowpark.session import Session
from snowflake.snowpark import version as v
import json

with open('creds.json') as f:
    data = json.load(f)
    USERNAME = data['user']
    PASSWORD = data['password']
    SF_ACCOUNT = data['account']
    SF_WH = data['warehouse']

CONNECTION_PARAMETERS = {
    "account": SF_ACCOUNT,
    "user": USERNAME,
    "password": PASSWORD,
}

session = Session.builder.configs(CONNECTION_PARAMETERS).create()
```

Environment Setup

```
In [ ]: session.sql(''create database if not exists snowflake_sample_data from share sfc_samples.sample_

Out[ ]: [Row(status='SNOWFLAKE_SAMPLE_DATA already exists, statement succeeded.')]

In [ ]: session.sql('CREATE DATABASE IF NOT EXISTS tpcds_xgboost').collect()
session.sql('CREATE SCHEMA IF NOT EXISTS tpcds_xgboost.demo').collect()
session.sql("create or replace warehouse FE_AND_INFERENCE_WH with warehouse_size='3X-LARGE'").co
session.sql("create or replace warehouse snowpark_opt_wh with warehouse_size = 'MEDIUM' warehouse
session.sql("alter warehouse snowpark_opt_wh set max_concurrency_level = 1").collect()
session.use_warehouse('FE_AND_INFERENCE_WH')
```

Select either 100 or 10 for the TPC-DS Dataset size to use below. See (<https://docs.snowflake.com/en/user-guide/sample-data-tpcds.html>)[here] for more information If you choose 100, I recommend >= 3XL warehouse.

```
In [ ]: TPCDS_SIZE_PARAM = 10
SNOWFLAKE_SAMPLE_DB = 'SNOWFLAKE_SAMPLE_DATA' # Name of Snowflake Sample Database might be differ

if TPCDS_SIZE_PARAM == 100:
    TPCDS_SCHEMA = 'TPCDS_SF100TCL'
elif TPCDS_SIZE_PARAM == 10:
    TPCDS_SCHEMA = 'TPCDS_SF10TCL'
else:
    raise ValueError("Invalid TPCDS_SIZE_PARAM selection")

store_sales = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.store_sales')
catalog_sales = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.catalog_sales')
web_sales = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.web_sales')
date = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.date_dim')
dim_stores = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.store')
customer = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.customer')
address = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.customer_address')
demo = session.table(f'{SNOWFLAKE_SAMPLE_DB}.{TPCDS_SCHEMA}.customer_demographics')
```

Feature Engineering

We will aggregate sales by customer across all channels(web, store, catalogue) and join that to customer demographic data.

```
In [ ]: store_sales_agged = store_sales.group_by('ss_customer_sk').agg(F.sum('ss_sales_price').as_('total_sales'))
web_sales_agged = web_sales.group_by('ws_bill_customer_sk').agg(F.sum('ws_sales_price').as_('total_sales'))
catalog_sales_agged = catalog_sales.group_by('cs_bill_customer_sk').agg(F.sum('cs_sales_price').as_('total_sales'))
store_sales_agged = store_sales_agged.rename('ss_customer_sk', 'customer_sk')
web_sales_agged = web_sales_agged.rename('ws_bill_customer_sk', 'customer_sk')
catalog_sales_agged = catalog_sales_agged.rename('cs_bill_customer_sk', 'customer_sk')

In [ ]: total_sales = store_sales_agged.union_all(web_sales_agged)
total_sales = total_sales.union_all(catalog_sales_agged)

In [ ]: total_sales = total_sales.group_by('customer_sk').agg(F.sum('total_sales').as_('total_sales'))

In [ ]: customer = customer.select('c_customer_sk', 'c_current_hdemo_sk', 'c_current_addr_sk', 'c_customer_demo_sk')

In [ ]: customer = customer.join(address.select('ca_address_sk', 'ca_zip'), customer['c_current_addr_sk'] == address['ca_address_sk'])
customer = customer.join(demo.select('cd_demo_sk', 'cd_gender', 'cd_marital_status', 'cd_credit_limit'),
                        customer['c_current_hdemo_sk'] == demo['cd_demo_sk'])
customer = customer.rename('c_customer_sk', 'customer_sk')
customer.show()
```

```

-----
| "CUSTOMER_SK" | "C_CURRENT_HDEMO_SK" | "C_CURRENT_ADDR_SK" | "C_CUSTOMER_ID" | "C_BIRTH_YEAR"
| "CA_ADDRESS_SK" | "CA_ZIP" | "CD_DEMO_SK" | "CD_GENDER" | "CD_MARITAL_STATUS" | "CD_CREDIT_RATI
NG" | "CD_EDUCATION_STATUS" | "CD_DEP_COUNT" |
-----

```

```

-----
| 60566488 | 6846 | 18830487 | AAAAAAAAAINLCMJDA | 1952
| 18830487 | NULL | 6846 | F | D | Good
| Advanced Degree | 1 | |
| 60566489 | 3564 | 14404870 | AAAAAAAAJNLCMJDA | 1988
| 14404870 | 50587 | 3564 | F | S | High Risk
| Unknown | 0 | |
| 60566490 | 4684 | 17765836 | AAAAAAAKNLCMJDA | 1954
| 17765836 | 30519 | 4684 | F | S | Unknown
| Unknown | 0 | |
| 60566491 | 761 | 29082657 | AAAAAAALNLCMJDA | 1932
| 29082657 | 38883 | 761 | M | M | Good
| Unknown | 0 | |
| 60566492 | 3967 | 19167218 | AAAAAAAMNLCMJDA | 1987
| 19167218 | 38048 | 3967 | M | W | High Risk
| 4 yr Degree | 0 | |
| 60566493 | 6861 | 11443476 | AAAAAAANLCMJDA | 1983
| 11443476 | 29101 | 6861 | M | M | Good
| Primary | 1 | |
| 60566494 | 4652 | 19206825 | AAAAAAANLCMJDA | 1952
| 19206825 | 26534 | 4652 | F | M | Unknown
| 2 yr Degree | 0 | |
| 60566495 | 3022 | 15555307 | AAAAAAAPNLCMJDA | 1924
| 15555307 | 70499 | 3022 | F | M | High Risk
| Secondary | 0 | |
| 60566496 | 4689 | 5762350 | AAAAAAAAOLCMJDA | 1967
| 5762350 | 67752 | 4689 | M | U | Unknown
| Unknown | 0 | |
| 60566497 | 3948 | 9041952 | AAAAAAABOLCMJDA | 1970
| 9041952 | 76867 | 3948 | F | W | High Risk
| College | 0 | |
-----

```

```
In [ ]: final_df = total_sales.join(customer, on='customer_sk')
```

```
In [ ]: session.use_database('tpcds_xgboost')
session.use_schema('demo')
final_df.write.mode('overwrite').save_as_table('feature_store')
```

```
In [ ]: session.add_packages('snowflake-snowpark-python', 'scikit-learn', 'pandas', 'numpy', 'joblib', '')
```

The version of package snowflake-snowpark-python in the local environment is 1.2.0, which does not fit the criteria for the requirement snowflake-snowpark-python. Your UDF might not work when the package version is different between the server and your local environment

package scikit-learn is not installed in the local environmentYour UDF might not work when the package is installed on the server but not on your local environment.

package joblib is not installed in the local environmentYour UDF might not work when the package is installed on the server but not on your local environment.

The version of package cachetools in the local environment is 5.3.0, which does not fit the criteria for the requirement cachetools. Your UDF might not work when the package version is different between the server and your local environment

package xgboost is not installed in the local environmentYour UDF might not work when the package is installed on the server but not on your local environment.

```
In [ ]: session.sql('CREATE OR REPLACE STAGE ml_models ').collect()
```

```
Out[ ]: [Row(status='Stage area ML_MODELS successfully created.')] 
```

```
In [ ]: from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.compose import ColumnTransformer
from xgboost import XGBRegressor
import joblib
import os

def train_model(session: snowflake.snowpark.Session) -> float:
    snowdf = session.table("feature_store")
    snowdf = snowdf.drop(['CUSTOMER_SK', 'C_CURRENT_HDEMO_SK', 'C_CURRENT_ADDR_SK', 'C_CUSTOMER_
    snowdf_train, snowdf_test = snowdf.random_split([0.8, 0.2], seed=82)

    # save the train and test sets as time stamped tables in Snowflake
    snowdf_train.write.mode("overwrite").save_as_table("tpcds_xgboost.demo.tpc_TRAIN")
    snowdf_test.write.mode("overwrite").save_as_table("tpcds_xgboost.demo.tpc_TEST")
    train_x = snowdf_train.drop("TOTAL_SALES").to_pandas() # drop labels for training set
    train_y = snowdf_train.select("TOTAL_SALES").to_pandas()
    test_x = snowdf_test.drop("TOTAL_SALES").to_pandas()
    test_y = snowdf_test.select("TOTAL_SALES").to_pandas()
    cat_cols = ['CA_ZIP', 'CD_GENDER', 'CD_MARITAL_STATUS', 'CD_CREDIT_RATING', 'CD_EDUCATION_ST
    num_cols = ['C_BIRTH_YEAR', 'CD_DEP_COUNT']

    num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('std_scaler', StandardScaler()),
    ])

    preprocessor = ColumnTransformer(
        transformers=[('num', num_pipeline, num_cols),
                      ('encoder', OneHotEncoder(handle_unknown="ignore"), cat_cols) ])

    pipe = Pipeline([('preprocessor', preprocessor),
                     ('xgboost', XGBRegressor())])
    pipe.fit(train_x, train_y)

    test_preds = pipe.predict(test_x)
    rmse = mean_squared_error(test_y, test_preds)
    model_file = os.path.join('/tmp', 'model.joblib')
    joblib.dump(pipe, model_file)
    session.file.put(model_file, "@ml_models", overwrite=True)
    return rmse
```

```
In [ ]: session.use_warehouse('snowpark_opt_wh')
train_model_sp = F.sproc(train_model, session=session, replace=True, is_permanent=True, name="xgl")
# Switch to Snowpark Optimized Warehouse for training and to run the stored proc
train_model_sp(session=session)
```

```
Out[ ]: 34196888.23949168
```

```
In [ ]: # Switch back to feature engineering/inference warehouse
session.use_warehouse('FE_AND_INFERENCE_WH')
```

```
In [ ]: import sys
import pandas as pd
import cachetools
import joblib
from snowflake.snowpark import types as T

session.add_import("@ml_models/model.joblib")

features = [ 'C_BIRTH_YEAR', 'CA_ZIP', 'CD_GENDER', 'CD_MARITAL_STATUS', 'CD_CREDIT_RATING', 'CD_

@cachetools.cached(cache={})
def read_file(filename):
    import_dir = sys._xoptions.get("snowflake_import_directory")
    if import_dir:
        with open(os.path.join(import_dir, filename), 'rb') as file:
            m = joblib.load(file)
            return m

@F.pandas_udf(session=session, max_batch_size=10000, is_permanent=True, stage_location='@ml_model
def predict(df: T.PandasDataFrame[int, str, str, str, str, str, int]) -> T.PandasSeries[float]:
    m = read_file('model.joblib')
    df.columns = features
    return m.predict(df)
```

```
In [ ]: inference_df = session.table('feature_store')
inference_df = inference_df.drop(['CUSTOMER_SK', 'C_CURRENT_HDEMO_SK', 'C_CURRENT_ADDR_SK', 'C_C
inputs = inference_df.drop("TOTAL_SALES")
snowdf_results = inference_df.select(*inputs,
                                     predict(*inputs).alias('PREDICTION'),
                                     (F.col('TOTAL_SALES')).alias('ACTUAL_SALES')
                                     )
snowdf_results.write.mode('overwrite').save_as_table('predictions')
```

```
In [ ]: inference_df.count()
```

```
Out[ ]: 62726989
```

```
In [ ]:
```