SNOWPARK CODE WALKTHROUGH DOC

0. Setting Up Databse, Schema, Granting Permissions

| USE ROLE ACCOUNTADMIN;  -- Using ACCOUNTADMIN, create a new role for this exercise and grant to applicable users CREATE OR REPLACE ROLE FS\_API\_ROLE; GRANT ROLE FS\_API\_ROLE to USER kamalnayan10;  -- create our virtual warehouse CREATE OR REPLACE WAREHOUSE FS\_API\_WH AUTO\_SUSPEND = 60;  GRANT ALL ON WAREHOUSE FS\_API\_WH TO ROLE FS\_API\_ROLE;  -- Next create a new database and schema, CREATE OR REPLACE DATABASE FS\_API\_DATABASE; CREATE OR REPLACE SCHEMA FS\_API\_SCHEMA;  GRANT OWNERSHIP ON DATABASE FS\_API\_DATABASE TO ROLE FS\_API\_ROLE COPY CURRENT GRANTS; GRANT OWNERSHIP ON ALL SCHEMAS IN DATABASE FS\_API\_DATABASE TO ROLE FS\_API\_ROLE COPY CURRENT GRANTS; |
| --- |

1. Setting Up Connection

| from snowflake.snowpark.context import get\_active\_session session = get\_active\_session()  # Add a query tag to the session. This helps with debugging and performance monitoring. session.query\_tag = {"origin":"sf\_sit", "name":"overview\_of\_feature\_store\_api", "version":{"major":1, "minor":0}}  # Set session context  session.use\_role("FS\_API\_ROLE")   # Print the current role, warehouse, and database/schema print(f"role: {session.get\_current\_role()} | WH: {session.get\_current\_warehouse()} | DB.SCHEMA: {session.get\_fully\_qualified\_current\_schema()}")  # The schema where Feature Store will initialize on and test dataset stores. FS\_DEMO\_SCHEMA = session.get\_current\_schema() # the schema where the model lives. MODEL\_DEMO\_SCHEMA = session.get\_current\_schema() |
| --- |

1. Loading Sample Database

| from snowflake.ml.feature\_store.examples.example\_helper import ExampleHelper  example\_helper = ExampleHelper(session, session.get\_current\_database(), FS\_DEMO\_SCHEMA) example\_helper.list\_examples().to\_pandas() source\_tables = example\_helper.load\_example('new\_york\_taxi\_features')  for table in source\_tables:  print(f"{table}:")  snowpark\_df = session.table(table) snowpark\_df.show(5) |
| --- |

1. Checking for missing values using snowpark api

Usign the aggregator function sum as snowpark\_sum for checking if value in column is null or not

| from snowflake.snowpark.functions import col, sum as snowpark\_sum, when  null\_counts = snowpark\_df.select(  \*[snowpark\_sum(when(col(c).is\_null(), 1).otherwise(0)).alias(c) for c in snowpark\_df.columns] )  null\_counts.show() |
| --- |

1. Removing values which provide nothing to the dataset as all values are null

| df\_fe = df\_fe.drop(["CONGESTION\_SURCHARGE", "AIRPORT\_FEE"]) df\_fe.show() |
| --- |

1. Binary Encoding a feature

| from snowflake.snowpark.functions import col, when  df\_fe = df\_fe.with\_column(  "STORE\_AND\_FWD\_FLAG",  when(col("STORE\_AND\_FWD\_FLAG") == "Y", 1).otherwise(0) )  df\_fe.show() |
| --- |

1. Dealing with nominal categorical variables by one hot encoding them

| vendor\_ids = [row["VENDORID"] for row in df\_fe.select("VENDORID").distinct().collect()] payment\_types = [row["PAYMENT\_TYPE"] for row in df\_fe.select("PAYMENT\_TYPE").distinct().collect()] ratecode\_ids = [row["RATECODEID"] for row in df\_fe.select("RATECODEID").distinct().collect()]  for v\_id in vendor\_ids:  new\_col = f"VENDOR\_{v\_id}"  df\_fe = df\_fe.with\_column(  new\_col,  when(col("VENDORID") == v\_id, 1).otherwise(0)  )  for p\_type in payment\_types:  new\_col = f"PAYMENT\_{p\_type}"  df\_fe = df\_fe.with\_column(  new\_col,  when(col("PAYMENT\_TYPE") == p\_type, 1).otherwise(0)  )  for r\_id in ratecode\_ids:  new\_col = f"RATECODE\_{r\_id}"  df\_fe = df\_fe.with\_column(  new\_col,  when(col("RATECODEID") == r\_id, 1).otherwise(0)  )  df\_fe = df\_fe.drop(["VENDORID", "PAYMENT\_TYPE", "RATECODEID"]) df\_fe.show() |
| --- |

1. Extracting more features from the pickup and dropoff features like time duration of trip, hour of day, day of week and month. I also ensured that the cyclic nature of data is preserved

| from snowflake.snowpark.functions import datediff  duration\_in\_seconds = datediff("second", col("TPEP\_PICKUP\_DATETIME"), col("TPEP\_DROPOFF\_DATETIME"))  df\_fe = df\_fe.with\_column("TRIP\_DURATION\_MINUTES", duration\_in\_seconds / 60)  df\_fe = df\_fe.with\_column("TRIP\_DURATION\_MINUTES", col("TRIP\_DURATION\_MINUTES").cast("int"))  df\_fe.show() |
| --- |

| from snowflake.snowpark.functions import month, dayofweek, hour, sin, cos from numpy import pi  df\_fe = df\_fe.with\_column("PICKUP\_HOUR", hour(col("TPEP\_PICKUP\_DATETIME"))) df\_fe = df\_fe.with\_column("DAY\_OF\_WEEK", dayofweek(col("TPEP\_PICKUP\_DATETIME"))) df\_fe = df\_fe.with\_column("PICKUP\_MONTH", month(col("TPEP\_PICKUP\_DATETIME")))  df\_fe = df\_fe.with\_column("HOUR\_SIN", sin(2\*pi\*col("PICKUP\_HOUR"))/24) df\_fe = df\_fe.with\_column("HOUR\_COS", cos(2\*pi\*col("PICKUP\_HOUR"))/24)  df\_fe = df\_fe.with\_column("DAY\_SIN", sin(2\*pi\*col("DAY\_OF\_WEEK"))/7) df\_fe = df\_fe.with\_column("DAY\_COS", cos(2\*pi\*col("DAY\_OF\_WEEK"))/7)  df\_fe = df\_fe.with\_column("MONTH\_SIN", sin(2\*pi\*col("PICKUP\_MONTH"))/12) df\_fe = df\_fe.with\_column("MONTH\_COS", cos(2\*pi\*col("PICKUP\_MONTH"))/12) |
| --- |

1. Scale numerical features

| from snowflake.snowpark.functions import col, mean, stddev  columns\_to\_scale = [  "PASSENGER\_COUNT",  "TRIP\_DISTANCE",  "TRIP\_DURATION\_MINUTES" ]  for col\_name in columns\_to\_scale:  aggs = df\_fe.select(mean(col(col\_name)).alias("mean"), stddev(col(col\_name)).alias("stddev")).collect()[0]   col\_mean = aggs["MEAN"]  col\_stddev = aggs["STDDEV"]   df\_fe = df\_fe.with\_column(  col\_name,  (col(col\_name) - col\_mean) / col\_stddev  )  print("Numerical columns have been scaled.") |
| --- |

1. Create final feature dataframe to be stored on feature store

| final\_feature\_columns = [  # --- Identifiers ---  "TRIP\_ID",  "TPEP\_PICKUP\_DATETIME",   # --- Target Variable ---  "TOTAL\_AMOUNT",   # Numerical Features  "PASSENGER\_COUNT",  "TRIP\_DISTANCE",  "TRIP\_DURATION\_MINUTES",   # Cateorical Features  "PULOCATIONID",  "DOLOCATIONID",  "STORE\_AND\_FWD\_FLAG",  "VENDOR\_1", "VENDOR\_2",  "PAYMENT\_1", "PAYMENT\_2", "PAYMENT\_3", "PAYMENT\_4", "PAYMENT\_5",  "RATECODE\_1", "RATECODE\_2", "RATECODE\_3", "RATECODE\_4", "RATECODE\_5", "RATECODE\_6", "RATECODE\_99",   # Time Features(cyclical)  "HOUR\_SIN", "HOUR\_COS",  "DAY\_SIN", "DAY\_COS",  "MONTH\_SIN", "MONTH\_COS" ]  final\_features\_df = df\_fe.select(final\_feature\_columns) |
| --- |

1. Create Feature Store

| from snowflake.ml.feature\_store import (  FeatureStore,  FeatureView,  Entity,  CreationMode,  FeatureViewStatus, )  fs = FeatureStore(  session=session,   database=session.get\_current\_database(),   name=FS\_DEMO\_SCHEMA,   default\_warehouse=session.get\_current\_warehouse(),  creation\_mode=CreationMode.CREATE\_IF\_NOT\_EXIST, ) |
| --- |

1. Create Entity and corresponding FeatureView to store features in feature store

| from snowflake.ml.feature\_store import Entity  entity = Entity(  name="TRIP\_ID\_ENTITY",  join\_keys=["TRIP\_ID"],  desc="my TRIP ID Entitiy" )  fs.register\_entity(entity) entity = fs.get\_entity(name="TRIP\_ID\_ENTITY") print(entity.join\_keys)  from snowflake.ml.feature\_store import FeatureView  managed\_fv = FeatureView(  name="TAXI\_FARE\_FV",  entities=[entity],  feature\_df=final\_features\_df, # Snowpark DataFrame containing feature transformations  timestamp\_col="TPEP\_PICKUP\_DATETIME",   refresh\_freq="30 minutes",   desc="my managed feature view" ) fs.register\_feature\_view(  feature\_view=managed\_fv,  version='1.0', ) |
| --- |

1. Retrieve Feature view from feature store to pass into ML model

| retrieved\_fv = fs.get\_feature\_view(  name="TAXI\_FARE\_FV",  version="1.0" )  fs.list\_feature\_views().show() |
| --- |

1. Making a sample training dataset from feature view

| from snowflake.snowpark.functions import col  spine\_df = snowpark\_df.select(  col("TRIP\_ID"),  col("TPEP\_PICKUP\_DATETIME"),  col("TOTAL\_AMOUNT").alias("TARGET\_FARE") ).sample(n=100000)  training\_set = fs.generate\_training\_set(  spine\_df=spine\_df,  features=[retrieved\_fv],  spine\_timestamp\_col="TPEP\_PICKUP\_DATETIME", )  print("Final training data with features joined:") training\_set.show() |
| --- |

1. Training ML model using the training dataset made from querying feature store

| from snowflake.ml.modeling.xgboost import XGBRegressor from snowflake.ml.modeling.metrics import mean\_squared\_error, r2\_score  train\_sdf, test\_sdf = training\_set.random\_split([0.8, 0.2], seed=42)  feature\_cols = [  "PASSENGER\_COUNT", "TRIP\_DISTANCE", "TRIP\_DURATION\_MINUTES", "PULOCATIONID",  "DOLOCATIONID", "STORE\_AND\_FWD\_FLAG", "VENDOR\_1", "VENDOR\_2", "PAYMENT\_1",  "PAYMENT\_2", "PAYMENT\_3", "PAYMENT\_4", "PAYMENT\_5", "RATECODE\_1",  "RATECODE\_2", "RATECODE\_3", "RATECODE\_4", "RATECODE\_5", "RATECODE\_6",  "RATECODE\_99", "HOUR\_SIN", "HOUR\_COS", "DAY\_SIN", "DAY\_COS",  "MONTH\_SIN", "MONTH\_COS" ] label\_col = "TARGET\_FARE"  print("Training XGBoost model inside Snowflake...") regressor = XGBRegressor(  input\_cols=feature\_cols,  label\_cols=label\_col,  n\_estimators = 100,  max\_depth = 3,  lr = 1e-3,  output\_cols="PREDICTED\_FARE" )  regressor.fit(train\_sdf) print("Model training complete.")   result\_sdf = regressor.predict(test\_sdf)   mse = mean\_squared\_error(df=result\_sdf, y\_true\_col\_names=label\_col, y\_pred\_col\_names="PREDICTED\_FARE") r2 = r2\_score(df=result\_sdf, y\_true\_col\_name=label\_col, y\_pred\_col\_name="PREDICTED\_FARE")  print(f"Model Mean Squared Error: {mse:.2f}") print(f"Model R-squared (R²): {r2:.2f}")  result\_sdf.select(label\_col, "PREDICTED\_FARE").show() |
| --- |

THANK YOU