**Snowpark Hands-On Labs**

**Lab 1: Setup Snowpark Session**

**Objective:** Connect Python to Snowflake using Snowpark.

**Steps:**

1. Install required package (run in terminal/command prompt):
2. pip install snowflake-snowpark-python
3. Import Snowpark libraries:
4. from snowflake.snowpark import Session
5. Define connection parameters (replace with your credentials):
6. connection\_parameters = {
7. "account": "<your\_account>",
8. "user": "<your\_username>",
9. "password": "<your\_password>",
10. "role": "SYSADMIN",
11. "warehouse": "COMPUTE\_WH",
12. "database": "SNOWPARK\_DB",
13. "schema": "PUBLIC"
14. }
15. Create a session:
16. session = Session.builder.configs(connection\_parameters).create()
17. Test the connection:
18. session.sql("SELECT CURRENT\_TIMESTAMP").show()

**Expected Result:** A timestamp result confirming connection.

**Lab 2: DataFrame Transformations**

**Objective:** Work with DataFrames for filtering, joins, and aggregations.

**Steps:**

1. Load tables into DataFrames:
2. customers = session.table("CUSTOMERS")
3. orders = session.table("ORDERS")
4. Apply filtering:
5. apac\_customers = customers.filter(customers["REGION"] == "APAC")
6. apac\_customers.show()
7. Perform a join:
8. joined\_df = apac\_customers.join(orders, apac\_customers["ID"] == orders["CUSTOMER\_ID"])
9. joined\_df.show()
10. Group and aggregate:
11. sales\_summary = joined\_df.group\_by("REGION").agg({"AMOUNT": "sum"})
12. sales\_summary.show()

**Expected Result:** Aggregated sales amount per region.

**Lab 3: UDF Creation**

**Objective:** Create and use a Python User-Defined Function.

**Steps:**

1. Import UDF support:
2. from snowflake.snowpark.functions import udf
3. Define a function in Python:
4. def apply\_discount(price: float) -> float:
5. return price \* 0.9
6. Register as a UDF:
7. discount\_udf = udf(apply\_discount, return\_type="float", input\_types=["float"])
8. Use UDF in a query:
9. discounted\_orders = orders.with\_column("DISCOUNTED\_AMOUNT", discount\_udf(orders["AMOUNT"]))
10. discounted\_orders.show()

**Expected Result:** A new column with discounted order amounts.

**Lab 4: Stored Procedure**

**Objective:** Automate logic with a Snowpark Stored Procedure.

**Steps:**

1. Define a stored procedure:
2. def increase\_salary(session, increment: int):
3. session.sql(f"UPDATE EMPLOYEES SET SALARY = SALARY + {increment}").collect()
4. return "Salary updated"
5. Register the stored procedure:
6. session.sproc.register(
7. func=increase\_salary,
8. return\_type="string",
9. input\_types=["int"],
10. name="INCREASE\_SALARY\_SP",
11. replace=True
12. )
13. Call the stored procedure:
14. result = session.call("INCREASE\_SALARY\_SP", 500)
15. print(result)

**Expected Result:** Employee salaries updated by +500, and message "Salary updated".

**Lab 5: ML Integration**

**Objective:** Load ML model results into a Snowflake table using Snowpark.

**Steps:**

1. Prepare features from Snowflake:
2. df = session.table("CUSTOMERS")
3. features = df.select("AGE", "INCOME", "PURCHASE\_AMOUNT").to\_pandas()
4. Train a simple ML model in Python:
5. from sklearn.linear\_model import LinearRegression
6. model = LinearRegression()
7. model.fit(features[["AGE", "INCOME"]], features["PURCHASE\_AMOUNT"])
8. predictions = model.predict(features[["AGE", "INCOME"]])
9. Add predictions back into the Pandas DataFrame:
10. features["PREDICTED\_PURCHASE"] = predictions
11. Write results back to Snowflake:
12. session.write\_pandas(features, "CUSTOMER\_PREDICTIONS", auto\_create\_table=True)
13. Verify results in Snowflake:
14. session.table("CUSTOMER\_PREDICTIONS").show()

**Expected Result:** A new table CUSTOMER\_PREDICTIONS with actual and predicted values.