**Snowpark Hands-On Labs**

**Lab 1: Introduction & Setup (Python)**

**Objective**: Set up a Python environment for Snowpark.

**Steps:**

1. Open terminal or Databricks/VS Code environment.
2. Install Snowpark Python package:
3. pip install snowflake-snowpark-python
4. Import required libraries:
5. from snowflake.snowpark import Session
6. from snowflake.snowpark.functions import col
7. Create a session:
8. connection\_parameters = {
9. "account": "<your\_account>",
10. "user": "<your\_username>",
11. "password": "<your\_password>",
12. "role": "SYSADMIN",
13. "warehouse": "COMPUTE\_WH",
14. "database": "SNOWPARK\_DB",
15. "schema": "PUBLIC"
16. }
17. session = Session.builder.configs(connection\_parameters).create()
18. Validate connection:
19. session.sql("SELECT CURRENT\_DATE").show()

**Lab 2: DataFrame API**

**Objective**: Work with DataFrames inside Snowpark.

**Steps:**

1. Load a table into a DataFrame:
2. df = session.table("CUSTOMERS")
3. Select specific columns:
4. df.select("NAME", "REGION").show()
5. Filter rows:
6. df.filter(col("REGION") == "APAC").show()
7. Perform aggregation:
8. df.group\_by("REGION").count().show()
9. Join with another table:
10. orders = session.table("ORDERS")
11. joined = df.join(orders, df["ID"] == orders["CUSTOMER\_ID"])
12. joined.show()

**Lab 3: User-Defined Functions (UDFs)**

**Objective**: Create and use Python UDFs.

**Steps:**

1. Define a function:
2. def add\_discount(price: float) -> float:
3. return price \* 0.9
4. Register UDF:
5. from snowflake.snowpark.functions import udf
6. discount\_udf = udf(add\_discount, return\_type="float")
7. Apply to DataFrame:
8. df = session.table("ORDERS")
9. df = df.with\_column("discounted\_price", discount\_udf(col("PRICE")))
10. df.show()

**Lab 4: Stored Procedures**

**Objective**: Automate a workflow with stored procedures.

**Steps:**

1. Define a stored procedure:
2. def sp\_increase\_salary(session, increment: int):
3. session.sql(f"UPDATE EMPLOYEES SET SALARY = SALARY + {increment}").collect()
4. return "Salary updated"
5. session.sproc.register(func=sp\_increase\_salary, return\_type="string", input\_types=["int"], name="increase\_salary\_sp", replace=True)
6. Call stored procedure:
7. session.call("increase\_salary\_sp", 500)

**Lab 5: ML Integration with Snowpark**

**Objective**: Use Snowpark with ML workflow.

**Steps:**

1. Prepare features:
2. df = session.table("CUSTOMERS")
3. features = df.select("AGE", "INCOME", "PURCHASE\_AMOUNT").to\_pandas()
4. Train ML model in Python:
5. from sklearn.linear\_model import LinearRegression
6. model = LinearRegression()
7. model.fit(features[["AGE", "INCOME"]], features["PURCHASE\_AMOUNT"])
8. Deploy as UDF (mock example):
9. def predict\_purchase(age: int, income: float) -> float:
10. return model.predict([[age, income]])[0]
11. predict\_udf = udf(predict\_purchase, return\_type="float")
12. Use in queries:
13. df = df.with\_column("predicted\_purchase", predict\_udf(col("AGE"), col("INCOME")))
14. df.show()

**Lab 6: Performance Considerations**

**Objective**: Monitor and optimize performance.

**Steps:**

1. Use explain() to analyze query execution:
2. df = session.table("ORDERS")
3. df.filter(col("AMOUNT") > 1000).explain()
4. Enable caching for frequently accessed results:
5. df.cache\_result()
6. Create clustering for large tables (SQL in Snowflake UI):
7. ALTER TABLE ORDERS CLUSTER BY (REGION);

**Lab 7: Complex File Ingestion**

**Objective**: Ingest semi-structured data.

**Steps:**

1. Stage files in Snowflake:
2. CREATE OR REPLACE STAGE my\_stage;
3. PUT file://local\_path/customers.json @my\_stage;
4. Define file format:
5. CREATE OR REPLACE FILE FORMAT my\_json\_format TYPE = JSON;
6. Load file using Snowpark:
7. df = session.read.option("file\_format", "my\_json\_format").load("@my\_stage/customers.json")
8. df.show()
9. Flatten nested JSON data:
10. from snowflake.snowpark.functions import flatten
11. flattened = df.select(flatten(col("$")))
12. flattened.show()

**Lab 8: Wrap-Up and Troubleshooting**

* Validate all steps by querying Snowflake UI.
* Document common errors (connection issues, permissions, missing stages).
* Build a small **end-to-end pipeline**:
  + Load JSON → Transform with DataFrame → Apply UDF → Save to table.