iPhone Sales Analysis Project

1. Installing Dependencies:

pip install pyspark

2. Configuring Hive:

Editing hive-site.xml and ensuring these settings:

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://localhost:9083</value>
</property>
<property>
  <name>hive.execution.engine</name>
  <value>mr</value>
</property>
```

3. Starting Services:

```
# Starting HDFS
start-dfs.sh
# Starting Hive
start-hive.sh
```

4. Creating Hive Tables:

4.1 Creating Partitioned Sales Table:

```
CREATE TABLE sales_partitioned (
    seller_id INT,
    product_id INT,
    buyer_id INT,
    quantity INT,
    price INT
)
PARTITIONED BY (sale_date DATE)
STORED AS PARQUET;
```

4.2 Creating Non-Partitioned Product Table:

```
CREATE TABLE product_table (
    product_id INT,
    product_name STRING,
    unit_price INT
)
STORED AS PARQUET;
```

5. Sales Data Collector API:

```
from pyspark.sql import SparkSession

def sales_data_collector_api(spark, text_file_path):
    # Reading data from the text file
    sales_df = spark.read.csv(text_file_path, header=True, sep="|", inferSchema=True)

    # Saving data into partitioned Hive table
    sales_df.write.partitionBy("sale_date").mode("overwrite").format("parquet").saveAsTable("sales_partitioned")
    return "sales_partitioned"
```

For testing:

```
spark = SparkSession.builder.appName("SalesDataCollector").enableHiveSupport().getOrCreate()
sales_data_collector_api(spark, "/path/to/sales_data.txt")
```

6. Product Data Collector API:

```
def product_data_collector_api(spark, parquet_file_path):
    # Reading data from the Parquet file
    product_df = spark.read.parquet(parquet_file_path)

    # Saving data into Hive table
    product_df.write.mode("overwrite").format("parquet").saveAsTable("product_table")
    return "product_table"
```

For testing:

```
spark = SparkSession.builder.appName("ProductDataCollector").enableHiveSupport().getOrCreate()
product_data_collector_api(spark, "/path/to/product_data.parquet")
```

7. Data Preparation API:

```python
def data_preparation_api(spark, product_hive_table, sales_hive_table, target_hive_table):
    # Loading product and sales data
    product_df = spark.sql(f"SELECT * FROM {product_hive_table}")
    sales_df = spark.sql(f"SELECT * FROM {sales_hive_table}")

    # Identifying buyers who purchased S8 but not iPhone
    s8_buyers = sales_df.join(product_df, "product_id").filter(product_df.product_name == "S8").select("buyer_id").distinct()
    iphone_buyers = sales_df.join(product_df, "product_id").filter(product_df.product_name == "iPhone").select("buyer_id").distinct()

    # Subtracting buyers
    s8_only_buyers = s8_buyers.subtract(iphone_buyers)

    # Saving result to a new Hive table
    s8_only_buyers.write.mode("overwrite").format("parquet").saveAsTable(target_hive_table)
    return target_hive_table
```

For testing:

```python
spark = SparkSession.builder.appName("DataPreparation").enableHiveSupport().getOrCreate()
data_preparation_api(spark, "product_table", "sales_partitioned", "target_hive_table")
```

```
▼ (1) Spark Jobs
  ▶ Job 0  View (Stages: 1/1)
Out[25]: 'target_hive_table'
```

8. Test all APIs:

```
# Setting up Spark session
spark = SparkSession.builder.appName("EndToEnd").enableHiveSupport().getOrCreate()

# Running Sales and Product Data Collectors
sales_data_collector_api(spark, "/path/to/sales_data.txt")
product_data_collector_api(spark, "/path/to/product_data.parquet")

# Running Data Preparation API
data_preparation_api(spark, "product_table", "sales_partitioned", "target_hive_table")

# Validating output
result_df = spark.sql("SELECT * FROM target_hive_table")
result_df.display()
```