write in pyspark and Oracle SQL

1. Find the percentage of drivers who completed more than 10 trips in a single month.
Table: trips
Columns: driver_id, trip_date, trip_id

```
%sql
WITH driver_monthly AS (
SELECT driver_id, TRUNC(trip_date, 'MM') AS month, COUNT(*) AS trip_count FROM trips GROUP BY driver_id, TRUNC(trip_date, 'MM')),

total_drivers_per_month AS (
SELECT month, COUNT(DISTINCT driver_id) AS total_drivers FROM driver_monthly GROUP BY month),

drivers_above_10 AS (
SELECT month, COUNT(driver_id) AS drivers_above_10 FROM driver_monthly WHERE trip_count > 10 GROUP BY month)

SELECT d.month, ROUND((a.drivers_above_10 * 100.0) / d.total_drivers, 2) AS percentage_above_10_trips FROM total_drivers_per_month d JOIN
drivers_above_10 a ON d.month = a.month;
```

```
%python
from pyspark.sql.functions import trunc, countDistinct, count, col, round

trips_monthly = trips.withColumn("month", trunc("trip_date", "MM")) \
                .groupBy("driver_id", "month") \
                .count() \
                .withColumnRenamed("count", "trip_count")

drivers_above_10 = trips_monthly.filter(col("trip_count") > 10) \
                       .groupBy("month") \
                       .agg(countDistinct("driver_id").alias("drivers_above_10"))

total_drivers = trips_monthly.groupBy("month") \
                     .agg(countDistinct("driver_id").alias("total_drivers"))

result = drivers_above_10.join(total_drivers, "month") \
               .withColumn("percentage_above_10_trips",
                       round((col("drivers_above_10") / col("total_drivers")) * 100, 2))
```

2. Identify users who gave the same rating to the same driver more than once.
Table: ratings
Columns: user_id, driver_id, rating, rating_date

```
%sql
SELECT user_id, driver_id, rating FROM ratings GROUP BY user_id, driver_id, rating HAVING COUNT(*) > 1;

%python
```

```
from pyspark.sql.functions import count

duplicate_ratings = ratings.groupBy("user_id", "driver_id", "rating") \
                          .agg(count("*").alias("cnt")) \
                          .filter("cnt > 1")
```

3. List the top 3 cities with the highest average trip fare in the last 6 months.
Table: trips
Columns: city, trip_date, fare

```
%sql
SELECT * FROM (
SELECT city, AVG(fare) AS avg_fare, DENSE_RANK() OVER (ORDER BY AVG(fare) DESC) AS rnk FROM trips WHERE trip_date >= ADD_MONTHS(SYSDATE,
-6) GROUP BY city) WHERE rnk <= 3;

%python
from pyspark.sql.functions import avg, dense_rank, current_date
from pyspark.sql.window import Window

recent_trips = trips.filter(col("trip_date") >= add_months(current_date(), -6))

city_avg = recent_trips.groupBy("city") \
                      .agg(avg("fare").alias("avg_fare"))

window_spec = Window.orderBy(col("avg_fare").desc())

top_3_cities = city_avg.withColumn("rnk", dense_rank().over(window_spec)) \
                      .filter("rnk <= 3")
```

4. Find drivers who had more than 7 days of gap between two consecutive trips.
Table: trips
Columns: driver_id, trip_date, trip_id

```
%sql
SELECT driver_id, trip_date, prev_date, trip_date - prev_date AS gap_days FROM (
SELECT driver_id, trip_date, LAG(trip_date) OVER (PARTITION BY driver_id ORDER BY trip_date) AS prev_date FROM trips) WHERE trip_date - prev_date >
7;

%python
from pyspark.sql.window import Window
from pyspark.sql.functions import lag, datediff

window_spec = Window.partitionBy("driver_id").orderBy("trip_date")

trips_with_lag = trips.withColumn("prev_trip", lag("trip_date").over(window_spec))
```

```
gaps = trips_with_lag.withColumn("gap_days", datediff("trip_date", "prev_trip")) \
                    .filter("gap_days > 7")
```

5. Flag trips where fare is more than twice the average fare of that city on that date.
Table: trips
Columns: trip_id, city, trip_date, fare

```
WITH avg_fares AS (
SELECT city, trip_date, AVG(fare) AS avg_fare FROM trips GROUP BY city, trip_date)
SELECT t.trip_id, t.city, t.trip_date, t.fare, CASE WHEN t.fare > 2 * a.avg_fare THEN 'High' ELSE 'Normal' END AS fare_flag FROM trips t JOIN avg_fares a
ON t.city = a.city AND t.trip_date = a.trip_date;
```

```
%python
from pyspark.sql.functions import avg, when

avg_fares = trips.groupBy("city", "trip_date") \
                .agg(avg("fare").alias("avg_fare"))

trips_flagged = trips.join(avg_fares, on=["city", "trip_date"]) \
                    .withColumn("fare_flag",
                               when(col("fare") > 2 * col("avg_fare"), "High").otherwise("Normal"))
```