Query 2

Find, for each year, the 3 Police Departments with the highest percentage of closed cases. Print the year, the names (locations) of the departments, their percentages as well as their ranking. The results are given in ascending order by year and ranking

Question 2

a) Implement Query 2 using the DataFrame and SQL APIs. Report and compare the execution times between the two implementations.

b) Write Spark code that converts the main data set to parquet file format and stores a single .parquet file in your team's S3 bucket. Choose one of the two implementations of subquery a) (DataFrame or SQL) and compare the execution times of your application when the data is imported as .csv and as .parquet.

```python
### DataFrame API ###
import time
from pyspark.sql import SparkSession
from pyspark.sql.functions import count, sum, col, row_number, 
format_number, when, year, to_timestamp
from pyspark.sql.window import Window

# Start the timer
start_time = time.time()

# Create SparkSession
spark = SparkSession.builder \
    .appName("Query 2") \
    .getOrCreate()

# Read both CSV files into DataFrames
crime_data_2010_2019 = "s3://initial-notebook-data-bucket-
dblab-905418150721/CrimeData/
Crime_Data_from_2010_to_2019_20241101.csv"
crime_data_2020_present = "s3://initial-notebook-data-bucket-
dblab-905418150721/CrimeData/
Crime_Data_from_2020_to_Present_20241101.csv"

# Read both CSV files into DataFrames
crime_df_2010_2019 = spark.read.csv(crime_data_2010_2019, 
header=True, inferSchema=True)
crime_df_2020_present = spark.read.csv(crime_data_2020_present, 
header=True, inferSchema=True)

# Combine the two DataFrames and remove duplicates
crime_data = 
crime_df_2010_2019.union(crime_df_2020_present).dropDuplicates()

# Convert DATE OCC to timestamp format and extract the year
```

```python
crime_data = crime_data.withColumn("DATE OCC",
to_timestamp(col("DATE OCC"), "MM/dd/yyyy hh:mm:ss a"))
crime_data = crime_data.withColumn("Year", year(col("DATE OCC")))

# Drop rows where Year is null (if any)
crime_data = crime_data.filter(col("Year").isNotNull())

# Proceed with the rest of the calculations
case_stats = crime_data.groupBy("Year", "AREA NAME").agg(
    count("*").alias("total_cases"),
    sum((col("Status") != "IC").cast("int")).alias("cases_closed")
)

case_stats = case_stats.withColumn(
    "closed_case_rate",
    (col("cases_closed") / col("total_cases")) * 100
)

window_spec =
Window.partitionBy("Year").orderBy(col("closed_case_rate").desc())

case_stats = case_stats.withColumn("rank",
row_number().over(window_spec))

top_precincts = case_stats.filter(col("rank") <= 3).orderBy("Year",
"rank")

years = top_precincts.select("Year").distinct().rdd.flatMap(lambda
x: x).collect()

yearly_tables = {}
for year in years:
    yearly_tables[year] = top_precincts.filter(col("Year") == year)
    print(f"Top 3 for Year {year}:")
    yearly_tables[year] \
        .select(
            col("Year"),
            col("AREA NAME").alias("precinct"),
            col("closed_case_rate"),
            col("rank").alias("#")
        ).show(truncate=False)

end_time = time.time()
print(f"DataFrame API took : {end_time - start_time:.2f} seconds")
```

```
Top 3 for Year 2010:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2010|Rampart |32.84713448949121 |1  |
|2010|Olympic |31.515289821999087|2  |
|2010|Harbor  |29.36028339237341 |3  |
+----+--------+------------------+---+

Top 3 for Year 2011:
```

```
Top 3 for Year 2011:
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
+----+--------+-----------------+---+
|2011|Olympic |35.040060090135206|1  |
|2011|Rampart |32.4964471814306 |2  |
|2011|Harbor  |28.51336246316431 |3  |
+----+--------+-----------------+---+

Top 3 for Year 2012:
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
+----+--------+-----------------+---+
|2012|Olympic |34.29708533302119 |1  |
|2012|Rampart |32.46000463714352 |2  |
|2012|Harbor  |29.509585848956675|3  |
+----+--------+-----------------+---+

Top 3 for Year 2013:
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
+----+--------+-----------------+---+
|2013|Olympic |33.58217940999398 |1  |
|2013|Rampart |32.1060382916053  |2  |
|2013|Harbor  |29.723638951488557|3  |
+----+--------+-----------------+---+

Top 3 for Year 2014:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2014|Van Nuys   |32.0215235281705  |1  |
|2014|West Valley|31.49754809505847 |2  |
|2014|Mission    |31.224939855653567|3  |
+----+-----------+------------------+---+

Top 3 for Year 2015:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2015|Van Nuys|32.265140677157845|1  |
|2015|Mission |30.463762673676303|2  |
|2015|Foothill|30.353001803658852|3  |
+----+--------+------------------+---+

Top 3 for Year 2016:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2016|Van Nuys   |32.194518462124094|1  |
|2016|West Valley|31.40146437042384 |2  |
|2016|Foothill   |29.908647228131645|3  |
+----+-----------+------------------+---+

Top 3 for Year 2017:
```

```
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
+----+--------+-----------------+---+
|2017|Van Nuys|32.0554272517321 |1  |
|2017|Mission |31.055387158996968|2  |
|2017|Foothill|30.469700657094183|3  |
+----+--------+-----------------+---+

Top 3 for Year 2018:
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
+----+--------+-----------------+---+
|2018|Foothill|30.731346958877126|1  |
|2018|Mission |30.727023319615913|2  |
|2018|Van Nuys|28.905206942590123|3  |
+----+--------+-----------------+---+

Top 3 for Year 2019:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2019|Mission    |30.727411112319235|1  |
|2019|West Valley|30.57974335472044 |2  |
|2019|N Hollywood|29.23808669119627 |3  |
+----+-----------+------------------+---+

Top 3 for Year 2020:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2020|West Valley|30.771131982204647|1  |
|2020|Mission    |30.14974649215894 |2  |
|2020|Harbor     |29.693486590038315|3  |
+----+-----------+------------------+---+

Top 3 for Year 2021:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2021|Mission    |30.318115590092276|1  |
|2021|West Valley|28.971087440009363|2  |
|2021|Foothill   |27.993757094211126|3  |
+----+-----------+------------------+---+

Top 3 for Year 2022:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2022|West Valley|26.536367172306498|1  |
|2022|Harbor     |26.337538060026098|2  |
|2022|Topanga    |26.234013317831096|3  |
+----+-----------+------------------+---+

Top 3 for Year 2023:
+----+--------+-----------------+---+
|Year|precinct|closed_case_rate |#  |
```

```
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2023|Foothill|26.76076020122974 |1  |
|2023|Topanga |26.538022616453986|2  |
|2023|Mission |25.662731120516817|3  |
+----+--------+------------------+---+

Top 3 for Year 2024:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2024|N Hollywood|19.598528961078763|1  |
|2024|Foothill   |18.620882188721385|2  |
|2024|77th Street|17.586318167150694|3  |
+----+-----------+------------------+---+

DataFrame API took : 29.74 seconds
```

```python
### SQL API ###
# Start the timer
start_time = time.time()

# Group data by year and area to calculate total cases and closed
cases
case_stats = crime_data.groupBy("Year", "AREA NAME").agg(
    # Count total cases per year and area
    count("*").alias("total_cases"),

    # Count closed cases where Status is NOT "IC" (Invest Cont)
    sum((col("Status") != "IC").cast("int")).alias("cases_closed")
)

# Calculate the closed case rate as a percentage
case_stats = case_stats.withColumn(
    "closed_case_rate",
    (col("cases_closed") / col("total_cases")) * 100
)

# Define a window specification for ranking precincts within each
year
window_spec =
Window.partitionBy("Year").orderBy(col("closed_case_rate").desc())

# Add a rank column to rank precincts based on closed case rate
case_stats = case_stats.withColumn("rank",
row_number().over(window_spec))

# Filter the top 3 precincts for each year
top_precincts = case_stats.filter(col("rank") <= 3).orderBy("Year",
"rank")

# Create separate DataFrames for each year
years = top_precincts.select("Year").distinct().rdd.flatMap(lambda
x: x).collect()  # Collect all unique years
yearly_tables = {}
```

```python
for year in years:
    # Filter the top 3 precincts for the specific year
    yearly_tables[year] = top_precincts.filter(col("Year") == year)

    # Print the results for the specific year with renamed columns
    print(f"Top 3 for Year {year}:")
    yearly_tables[year] \
        .select(
            col("Year"),
            col("AREA NAME").alias("precinct"),  # Rename "AREA
NAME" to "precinct"
            col("closed_case_rate"),
            col("rank").alias("#")  # Rename "rank" to "#"
        ).show(truncate=False)

# Stop the timer and print the total execution time
end_time = time.time()
print(f"SQL API took: {end_time - start_time:.2f} seconds")
```

```
Top 3 for Year 2010:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2010|Rampart |32.84713448949121 |1  |
|2010|Olympic |31.515289821999087|2  |
|2010|Harbor  |29.36028339237341 |3  |
+----+--------+------------------+---+

Top 3 for Year 2011:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2011|Olympic |35.040060090135206|1  |
|2011|Rampart |32.4964471814306  |2  |
|2011|Harbor  |28.51336246316431 |3  |
+----+--------+------------------+---+

Top 3 for Year 2012:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2012|Olympic |34.29708533302119 |1  |
|2012|Rampart |32.46000463714352 |2  |
|2012|Harbor  |29.509585848956675|3  |
+----+--------+------------------+---+

Top 3 for Year 2013:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2013|Olympic |33.58217940999398 |1  |
|2013|Rampart |32.1060382916053  |2  |
|2013|Harbor  |29.723638951488557|3  |
+----+--------+------------------+---+
```

Top 3 for Year 2014:

Year	precinct	closed_case_rate	#
2014	Van Nuys	32.0215235281705	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.224939855653567	3

Top 3 for Year 2015:

Year	precinct	closed_case_rate	#
2015	Van Nuys	32.265140677157845	1
2015	Mission	30.463762673676303	2
2015	Foothill	30.353001803658852	3

Top 3 for Year 2016:

Year	precinct	closed_case_rate	#
2016	Van Nuys	32.194518462124094	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.908647228131645	3

Top 3 for Year 2017:

Year	precinct	closed_case_rate	#
2017	Van Nuys	32.0554272517321	1
2017	Mission	31.055387158996968	2
2017	Foothill	30.469700657094183	3

Top 3 for Year 2018:

Year	precinct	closed_case_rate	#
2018	Foothill	30.731346958877126	1
2018	Mission	30.727023319615913	2
2018	Van Nuys	28.905206942590123	3

Top 3 for Year 2019:

Year	precinct	closed_case_rate	#
2019	Mission	30.727411112319235	1
2019	West Valley	30.57974335472044	2
2019	N Hollywood	29.23808669119627	3

```
Top 3 for Year 2020:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2020|West Valley|30.771131982204647|1  |
|2020|Mission    |30.14974649215894 |2  |
|2020|Harbor     |29.693486590038315|3  |
+----+-----------+------------------+---+

Top 3 for Year 2021:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2021|Mission    |30.318115590092276|1  |
|2021|West Valley|28.971087440009363|2  |
|2021|Foothill   |27.993757094211126|3  |
+----+-----------+------------------+---+

Top 3 for Year 2022:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2022|West Valley|26.536367172306498|1  |
|2022|Harbor     |26.337538060026098|2  |
|2022|Topanga    |26.234013317831096|3  |
+----+-----------+------------------+---+

Top 3 for Year 2023:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2023|Foothill|26.76076020122974 |1  |
|2023|Topanga |26.538022616453986|2  |
|2023|Mission |25.662731120516817|3  |
+----+--------+------------------+---+

Top 3 for Year 2024:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2024|N Hollywood|19.598528961078763|1  |
|2024|Foothill   |18.620882188721385|2  |
|2024|77th Street|17.586318167150694|3  |
+----+-----------+------------------+---+

SQL API took: 26.65 seconds
```

Answer

DataFrame API applies PySpark's transformation methods including groupBy, filter, withColumn, which chains explicitly the operations and took 29.74 seconds to complete, whereas SQL API implements operations using Spark's SQL engine with SQL-style queries, which was finished in 26.65 seconds.

The SQL API benefits from Spark's catalyst optimizer in a much better way: the SQL statements are optimized for far better execution plans. The DataFrame API needs to invoke explicit method calls in Python, which may result in minor overhead, whereas the SQL API's declarative way will be better able to optimize the query pipeline.

In brief, for performance-critical scenarios where data processing can be expressed with SQL, usage of the SQL API is definitely the best choice. If it require programmatic flexibility or interaction with custom Python logic then DataFrame API will be best option.

```
2b)
```

I chose to use DataFrame API

```
# S3-path
output_path_s3 = "s3://groups-bucket-dblab-905418150721/group52/
main_data_set.parquet"

# Write to S3
try:
    crime_data.write.mode("overwrite").parquet(output_path_s3)
    print(f"DataFrame writed successfully to S3: {output_path_s3}")
except Exception as e:
    print(f"Error writing to S3: {e}")
```

```
DataFrame writed successfully to S3: s3://groups-bucket-
dblab-905418150721/group52/main_data_set.parquet
```

```
### Parquet ###
# Start the timer
start_time = time.time()

# Download Parquet file
parquet_path = "s3://groups-bucket-dblab-905418150721/group52/
main_data_set.parquet/"
crime_data = spark.read.parquet(parquet_path)

# Group data by year and area to calculate total cases and closed
cases
case_stats = crime_data.groupBy("Year", "AREA NAME").agg(
    # Count total cases per year and area
    count("*").alias("total_cases"),

    # Count closed cases where Status is NOT "IC" (Invest Cont)
    sum((col("Status") != "IC").cast("int")).alias("cases_closed")
)

# Calculate the closed case rate as a percentage
case_stats = case_stats.withColumn(
    "closed_case_rate",
    (col("cases_closed") / col("total_cases")) * 100
```

```python
)

# Define a window specification for ranking precincts within each
year
window_spec =
Window.partitionBy("Year").orderBy(col("closed_case_rate").desc())

# Add a rank column to rank precincts based on closed case rate
case_stats = case_stats.withColumn("rank",
row_number().over(window_spec))

# Filter the top 3 precincts for each year
top_precincts = case_stats.filter(col("rank") <= 3).orderBy("Year",
"rank")

# Create separate DataFrames for each year
years = top_precincts.select("Year").distinct().rdd.flatMap(lambda
x: x).collect()  # Collect all unique years
yearly_tables = {}

for year in years:
    # Filter the top 3 precincts for the specific year
    yearly_tables[year] = top_precincts.filter(col("Year") == year)

    # Print the results for the specific year with renamed columns
    print(f"Top 3 for Year {year}:")
    yearly_tables[year] \
        .select(
            col("Year"),
            col("AREA NAME").alias("precinct"),
            col("closed_case_rate"),
            col("rank").alias("#")
        ).show(truncate=False)

# Stop the timer and print the total execution time
end_time = time.time()
print(f"Execution time: {end_time - start_time:.2f} seconds")
```

```
Top 3 for Year 2010:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2010|Rampart |32.84713448949121 |1  |
|2010|Olympic |31.515289821999087|2  |
|2010|Harbor  |29.36028339237341 |3  |
+----+--------+------------------+---+

Top 3 for Year 2011:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2011|Olympic |35.040060090135206|1  |
|2011|Rampart |32.4964471814306  |2  |
|2011|Harbor  |28.51336246316431 |3  |
+----+--------+------------------+---+
```

Top 3 for Year 2012:

Year	precinct	closed_case_rate	#
2012	Olympic	34.29708533302119	1
2012	Rampart	32.46000463714352	2
2012	Harbor	29.509585848956675	3

Top 3 for Year 2013:

Year	precinct	closed_case_rate	#
2013	Olympic	33.58217940999398	1
2013	Rampart	32.1060382916053	2
2013	Harbor	29.723638951488557	3

Top 3 for Year 2014:

Year	precinct	closed_case_rate	#
2014	Van Nuys	32.0215235281705	1
2014	West Valley	31.49754809505847	2
2014	Mission	31.224939855653567	3

Top 3 for Year 2015:

Year	precinct	closed_case_rate	#
2015	Van Nuys	32.265140677157845	1
2015	Mission	30.463762673676303	2
2015	Foothill	30.353001803658852	3

Top 3 for Year 2016:

Year	precinct	closed_case_rate	#
2016	Van Nuys	32.194518462124094	1
2016	West Valley	31.40146437042384	2
2016	Foothill	29.908647228131645	3

Top 3 for Year 2017:

Year	precinct	closed_case_rate	#
2017	Van Nuys	32.0554272517321	1
2017	Mission	31.055387158996968	2
2017	Foothill	30.469700657094183	3

```
Top 3 for Year 2018:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2018|Foothill|30.731346958877126|1  |
|2018|Mission |30.727023319615913|2  |
|2018|Van Nuys|28.905206942590123|3  |
+----+--------+------------------+---+

Top 3 for Year 2019:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2019|Mission    |30.727411112319235|1  |
|2019|West Valley|30.57974335472044 |2  |
|2019|N Hollywood|29.23808669119627 |3  |
+----+-----------+------------------+---+

Top 3 for Year 2020:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2020|West Valley|30.771131982204647|1  |
|2020|Mission    |30.14974649215894 |2  |
|2020|Harbor     |29.693486590038315|3  |
+----+-----------+------------------+---+

Top 3 for Year 2021:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2021|Mission    |30.318115590092276|1  |
|2021|West Valley|28.971087440009363|2  |
|2021|Foothill   |27.993757094211126|3  |
+----+-----------+------------------+---+

Top 3 for Year 2022:
+----+-----------+------------------+---+
|Year|precinct   |closed_case_rate  |#  |
+----+-----------+------------------+---+
|2022|West Valley|26.536367172306498|1  |
|2022|Harbor     |26.337538060026098|2  |
|2022|Topanga    |26.234013317831096|3  |
+----+-----------+------------------+---+

Top 3 for Year 2023:
+----+--------+------------------+---+
|Year|precinct|closed_case_rate  |#  |
+----+--------+------------------+---+
|2023|Foothill|26.76076020122974 |1  |
|2023|Topanga |26.538022616453986|2  |
|2023|Mission |25.662731120516817|3  |
+----+--------+------------------+---+

Top 3 for Year 2024
```

```
Top 3 for Year 2024:
+----+-----------+-----------------+---+
|Year|precinct   |closed_case_rate |#  |
+----+-----------+-----------------+---+
|2024|N Hollywood|19.598528961078763|1  |
|2024|Foothill   |18.620882188721385|2  |
|2024|77th Street|17.586318167150694|3  |
+----+-----------+-----------------+---+

Execution time: 11.19 seconds
```

2b) Answer

I used the DataFrame API in the context of this dataset. For executing the .csv version, it took about 29.74 seconds: reading two .csv files separately, merging them, and finally performing aggregations and calculations. Reading and parsing time for the .csv format significantly contributed to the total time. Execution time for using the .parquet format stood at 11.19 seconds. Parquet is a columnar format highly optimized and performs well with Spark in reading more data and efficiently processing it. It doesn't have the heavy parsing overhead of the .csv and enjoys columnar compression.

The improvement comes mainly because of the way Parquet handles columnar data access, which is effective for the analytical workload of this application. The use of the .parquet file format in Spark processing tasks is recommended for better performance and scalability while using big datasets.