Name: Vaishnavi Abhimanyu Gambhire
Roll Number: 240840325065

Que 1.
1.
select * from routes r join airports s on s.iata = r.src_airport_iata join airport b on b.iata = r.dest_airport_iata
where s.iata = r.src_airport_iata
and s.iata != r.dest_airport_iata;

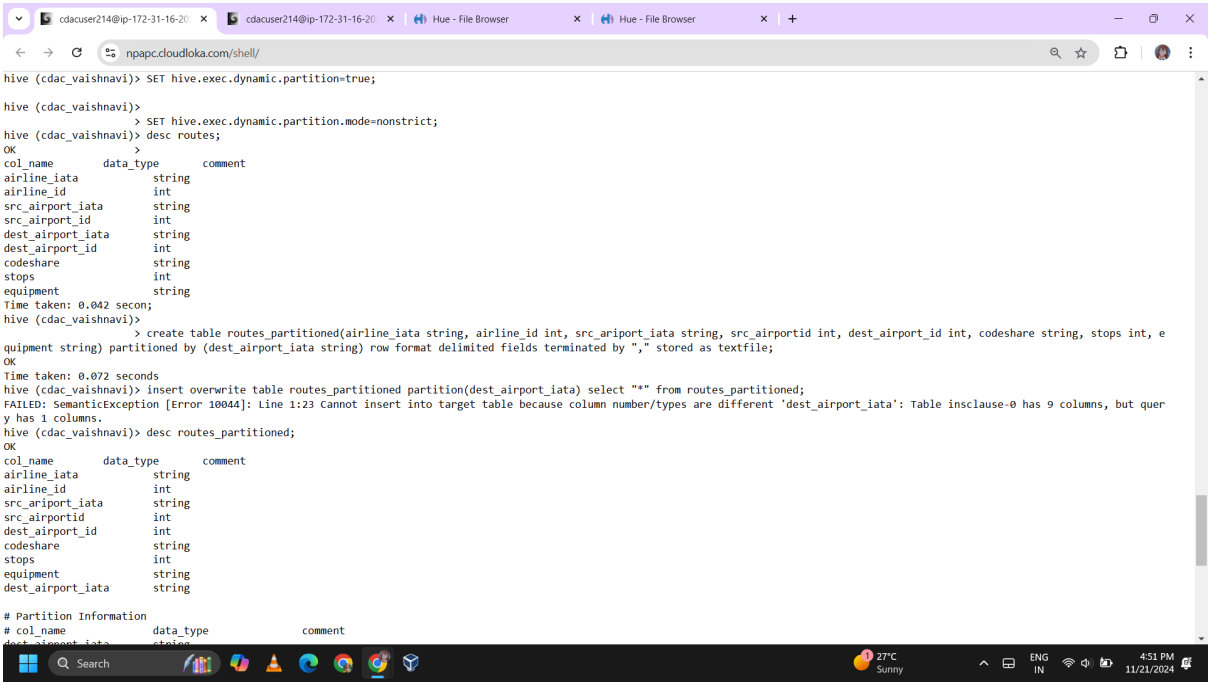
2.

3.
```
select count(equipment) from routes where equipment in (select
distinct equipment from routes);
```

Que 2:

Que 2.
Que 2.1: Answer :
```
create table routes_partitioned(airline_iata string,
airline_id int,
src_ariport_iata string,
src_airportid int,
dest_airport_id int,
codeshare string,
stops int,
equipment string)
partitioned by (dest_airport_iata string)
row format delimited fields terminated by "," stored as textfile;

insert overwrite table routes_partitioned
partition(dest_airport_iata) select * from routes_partitioned;
```



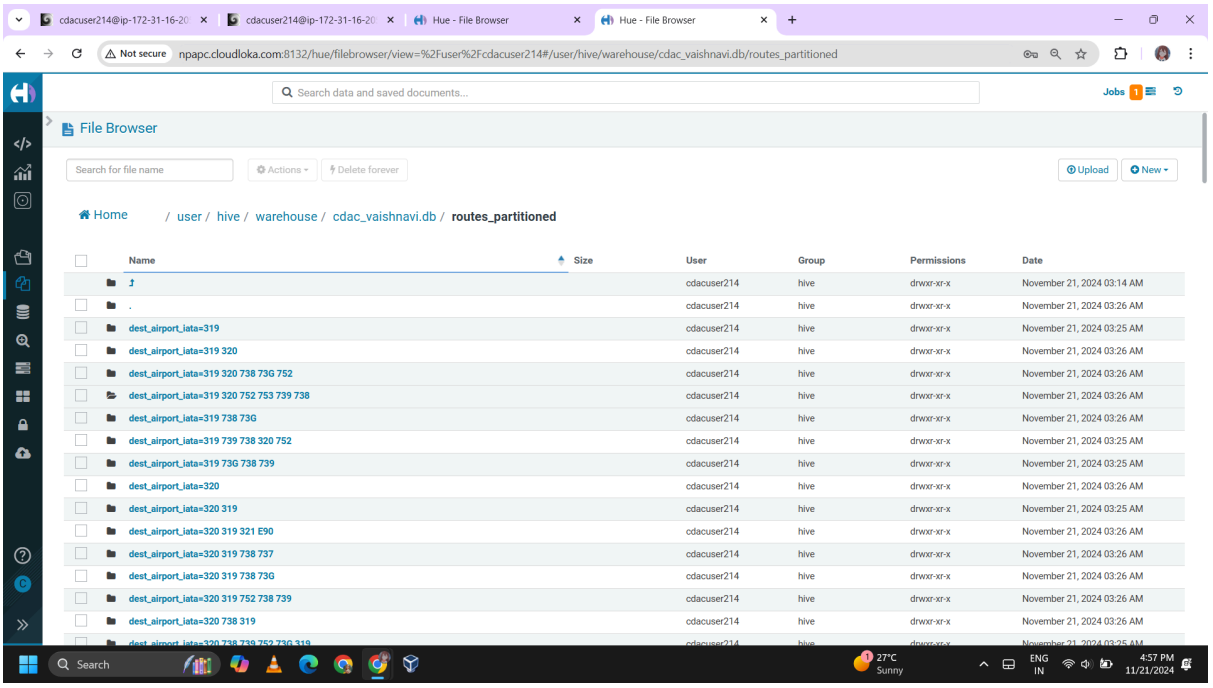
```
hive (cdac_vaishnavi)> SET hive.exec.dynamic.partition=true;

hive (cdac_vaishnavi)>
                    > SET hive.exec.dynamic.partition.mode=nonstrict;
hive (cdac_vaishnavi)> desc routes;
OK                  >
col_name        data_type       comment
airline_iata            string
airline_id              int
src_airport_iata        string
src_airport_id          int
dest_airport_iata       string
dest_airport_id         int
codeshare               string
stops                   int
equipment               string
Time taken: 0.042 secon;
hive (cdac_vaishnavi)>
                    > create table routes_partitioned(airline_iata string, airline_id int, src_ariport_iata string, src_airportid int, dest_airport_id int, codeshare string, stops int, e
quipment string) partitioned by (dest_airport_iata string) row format delimited fields terminated by "," stored as textfile;
OK
Time taken: 0.072 seconds
hive (cdac_vaishnavi)> insert overwrite table routes_partitioned partition(dest_airport_iata) select "*" from routes_partitioned;
FAILED: SemanticException [Error 10044]: Line 1:23 Cannot insert into target table because column number/types are different 'dest_airport_iata': Table insclause-0 has 9 columns, but quer
y has 1 columns.
hive (cdac_vaishnavi)> desc routes_partitioned;
OK
col_name        data_type       comment
airline_iata            string
airline_id              int
src_ariport_iata        string
src_airportid           int
dest_airport_id         int
codeshare               string
stops                   int
equipment               string
dest_airport_iata       string

# Partition Information
# col_name              data_type               comment
```

Que 2.2: Answer :

```
insert overwrite table routes_partitioned
partition(dest_airport_iata) select * from routes where
dest_airport_iata = 'ORD';
```



File Browser

Home / user / hive / warehouse / cdac_vaishnavi.db / routes_partitioned

Name	Size	User	Group	Permissions	Date
..		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:14 AM
.		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=319		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:25 AM
dest_airport_iata=319 320		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=319 320 738 73G 752		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=319 320 752 753 739 738		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=319 738 73G		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=319 739 738 320 752		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:25 AM
dest_airport_iata=319 73G 738 739		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:25 AM
dest_airport_iata=320		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=320 319		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:25 AM
dest_airport_iata=320 319 321 E90		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=320 319 738 737		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=320 319 738 73G		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=320 319 752 738 739		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM
dest_airport_iata=320 738 319		cdacuser214	hive	drwxr-xr-x	November 21, 2024 03:26 AM

SPARK:

```
print(sc)

data = spark.read.csv("airlines.csv", header=True,
inferSchema=True)
```

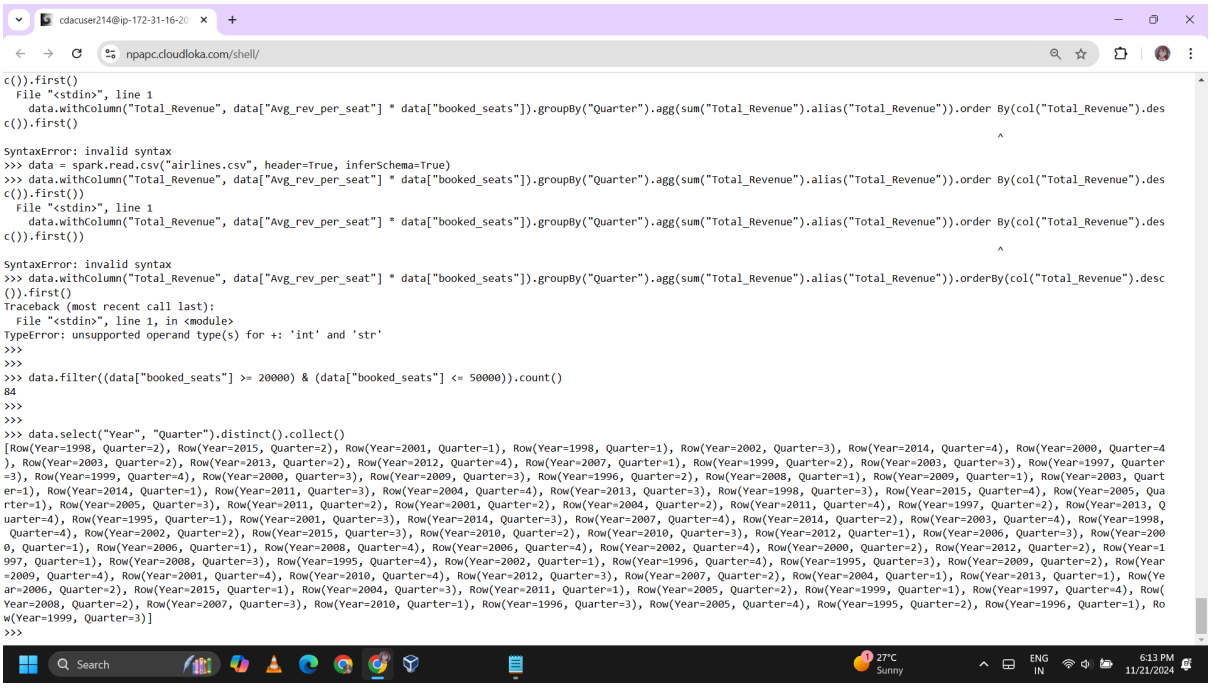
Question 1:

1.

```
data.filter((data["booked_seats"] >= 20000) &
(data["booked_seats"] <= 50000)).count()
```

2.

```
data.select("Year", "Quarter").distinct().collect()
```

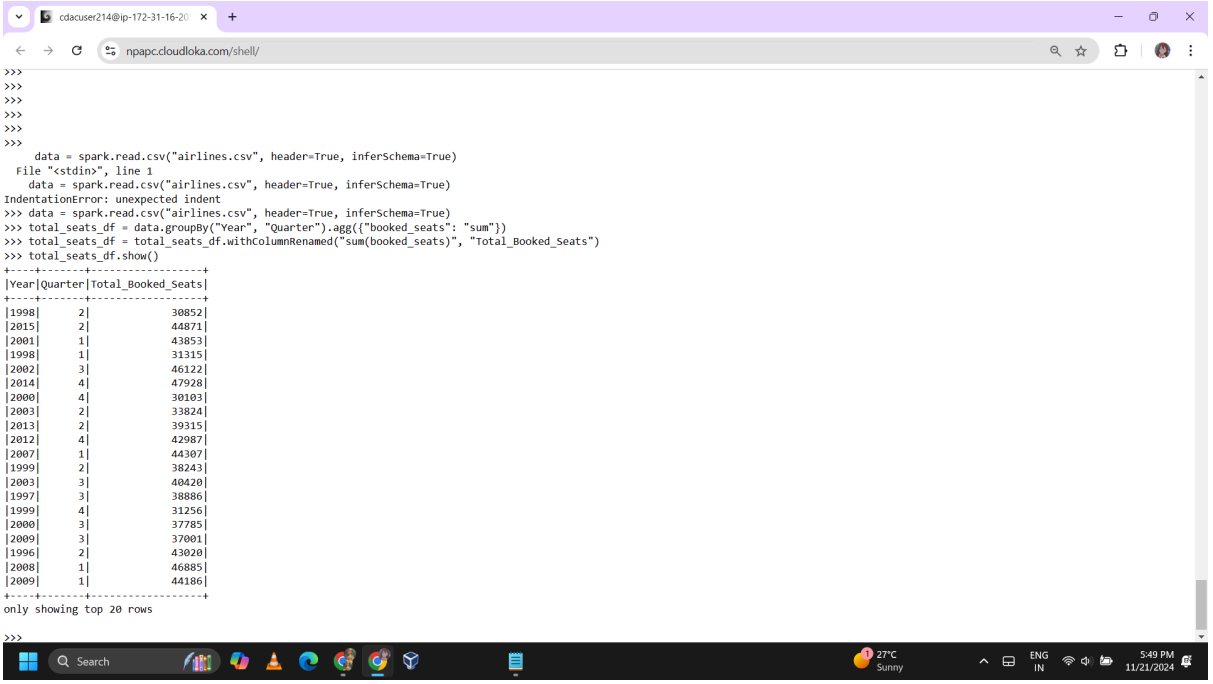


```
c()).first()
  File "<stdin>", line 1
    data.withColumn("Total_Revenue", data["Avg_rev_per_seat"] * data["booked_seats"]).groupBy("Quarter").agg(sum("Total_Revenue").alias("Total_Revenue")).order By(col("Total_Revenue").des
c()).first()
                                                                                                                                                                ^
SyntaxError: invalid syntax
>>> data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
>>> data.withColumn("Total_Revenue", data["Avg_rev_per_seat"] * data["booked_seats"]).groupBy("Quarter").agg(sum("Total_Revenue").alias("Total_Revenue")).order By(col("Total_Revenue").des
c()).first())
  File "<stdin>", line 1
    data.withColumn("Total_Revenue", data["Avg_rev_per_seat"] * data["booked_seats"]).groupBy("Quarter").agg(sum("Total_Revenue").alias("Total_Revenue")).order By(col("Total_Revenue").des
c()).first())
                                                                                                                                                                ^
SyntaxError: invalid syntax
>>> data.withColumn("Total_Revenue", data["Avg_rev_per_seat"] * data["booked_seats"]).groupBy("Quarter").agg(sum("Total_Revenue").alias("Total_Revenue")).orderBy(col("Total_Revenue").desc
()).first()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
>>>
>>> data.filter((data["booked_seats"] >= 20000) & (data["booked_seats"] <= 50000)).count()
84
>>>
>>>
>>> data.select("Year", "Quarter").distinct().collect()
[Row(Year=1998, Quarter=2), Row(Year=2015, Quarter=2), Row(Year=2001, Quarter=1), Row(Year=1998, Quarter=1), Row(Year=2002, Quarter=3), Row(Year=2014, Quarter=4), Row(Year=2000, Quarter=4
), Row(Year=2003, Quarter=2), Row(Year=2013, Quarter=2), Row(Year=2012, Quarter=4), Row(Year=2007, Quarter=1), Row(Year=1999, Quarter=2), Row(Year=2003, Quarter=3), Row(Year=1997, Quarter
=3), Row(Year=1999, Quarter=4), Row(Year=2000, Quarter=3), Row(Year=2009, Quarter=3), Row(Year=1996, Quarter=2), Row(Year=2008, Quarter=1), Row(Year=2009, Quarter=1), Row(Year=2003, Quart
er=1), Row(Year=2014, Quarter=1), Row(Year=2011, Quarter=3), Row(Year=2004, Quarter=4), Row(Year=2013, Quarter=3), Row(Year=1998, Quarter=3), Row(Year=2015, Quarter=4), Row(Year=2005, Qua
rter=1), Row(Year=2005, Quarter=3), Row(Year=2011, Quarter=2), Row(Year=2001, Quarter=2), Row(Year=2004, Quarter=2), Row(Year=2011, Quarter=4), Row(Year=1997, Quarter=2), Row(Year=2013, Q
uarter=4), Row(Year=1995, Quarter=1), Row(Year=2001, Quarter=3), Row(Year=2014, Quarter=3), Row(Year=2007, Quarter=4), Row(Year=2014, Quarter=2), Row(Year=2003, Quarter=4), Row(Year=1998,
 Quarter=4), Row(Year=2002, Quarter=2), Row(Year=2015, Quarter=3), Row(Year=2010, Quarter=2), Row(Year=2010, Quarter=3), Row(Year=2012, Quarter=1), Row(Year=2006, Quarter=3), Row(Year=200
0, Quarter=1), Row(Year=2006, Quarter=1), Row(Year=2008, Quarter=4), Row(Year=2006, Quarter=4), Row(Year=2002, Quarter=4), Row(Year=2000, Quarter=2), Row(Year=2012, Quarter=2), Row(Year=1
997, Quarter=1), Row(Year=2008, Quarter=3), Row(Year=1995, Quarter=4), Row(Year=2002, Quarter=1), Row(Year=1996, Quarter=4), Row(Year=1995, Quarter=3), Row(Year=2009, Quarter=2), Row(Year
=2009, Quarter=4), Row(Year=2001, Quarter=4), Row(Year=2010, Quarter=4), Row(Year=2012, Quarter=3), Row(Year=2007, Quarter=2), Row(Year=2004, Quarter=1), Row(Year=2013, Quarter=1), Row(Ye
ar=2006, Quarter=2), Row(Year=2015, Quarter=1), Row(Year=2004, Quarter=3), Row(Year=2011, Quarter=1), Row(Year=2005, Quarter=2), Row(Year=1999, Quarter=1), Row(Year=1997, Quarter=4), Row(
Year=2008, Quarter=2), Row(Year=2007, Quarter=3), Row(Year=2010, Quarter=1), Row(Year=1996, Quarter=3), Row(Year=2005, Quarter=4), Row(Year=1995, Quarter=2), Row(Year=1996, Quarter=1), Ro
w(Year=1999, Quarter=3)]
>>>
```

Question 2:

```
total_seats_df = data.groupBy("Year",
"Quarter").agg({"booked_seats": "sum"})
total_seats_df =
total_seats_df.withColumnRenamed("sum(booked_seats)",
"Total_Booked_Seats")
total_seats_df.show()
```

Total Number of Booked Seats:



```
>>>
>>>
>>>
>>>
>>>
>>>
    data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
  File "<stdin>", line 1
    data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
IndentationError: unexpected indent
>>> data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
>>> total_seats_df = data.groupBy("Year", "Quarter").agg({"booked_seats": "sum"})
>>> total_seats_df = total_seats_df.withColumnRenamed("sum(booked_seats)", "Total_Booked_Seats")
>>> total_seats_df.show()
+----+-------+-----------------+
|Year|Quarter|Total_Booked_Seats|
+----+-------+-----------------+
|1998|      2|            30852|
|2015|      2|            44871|
|2001|      1|            43853|
|1998|      1|            31315|
|2002|      3|            46122|
|2014|      4|            47928|
|2000|      4|            30103|
|2003|      2|            33824|
|2013|      2|            39315|
|2012|      4|            42987|
|2007|      1|            44307|
|1999|      2|            38243|
|2003|      3|            40420|
|1997|      3|            38886|
|1999|      4|            31256|
|2000|      3|            37785|
|2009|      3|            37001|
|1996|      2|            43020|
|2008|      1|            46885|
|2009|      1|            44186|
+----+-------+-----------------+
only showing top 20 rows

>>>
```

1.

Minimum Total Number of Booked Seats and MaximumTotal Number of Booked Seats

```
max_booked_seats =
total_seats_df.orderBy(col("Total_Booked_Seats").desc()).first()
print(f"Year and Quarter with highest total booked seats:
{max_booked_seats}")

min_booked_seats =
total_seats_df.orderBy(col("Total_Booked_Seats")).first()
print(f"Year and Quarter with Minimum total booked seats:
{min_booked_seats}")
```

Average Total Number of Booked Seats::

```
avg_booked_seats =
total_seats_df.groupBy("Year").agg(avg("Total_Booked_Seats").alias
("Avg_Booked_Seats"))
```

Output:

```
  File "<stdin>", line 1
    data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
IndentationError: unexpected indent
>>> data = spark.read.csv("airlines.csv", header=True, inferSchema=True)
>>> total_seats_df = data.groupBy("Year", "Quarter").agg({"booked_seats": "sum"})
>>> total_seats_df = total_seats_df.withColumnRenamed("sum(booked_seats)", "Total_Booked_Seats")
>>> total_seats_df.show()
+----+-------+------------------+
|Year|Quarter|Total_Booked_Seats|
+----+-------+------------------+
|1998|      2|             30852|
|2015|      2|             44871|
|2001|      1|             43853|
|1998|      1|             31315|
|2002|      3|             46122|
|2014|      4|             47928|
|2000|      4|             30103|
|2003|      2|             33824|
|2013|      2|             39315|
|2012|      4|             42987|
|2007|      1|             44307|
|1999|      2|             38243|
|2003|      3|             40420|
|1997|      3|             38886|
|1999|      4|             31256|
|2000|      3|             37785|
|2009|      3|             37001|
|1996|      2|             43020|
|2008|      1|             46885|
|2009|      1|             44186|
+----+-------+------------------+
only showing top 20 rows

>>> max_booked_seats = total_seats_df.orderBy(col("Total_Booked_Seats").desc()).first()
>>> print(f"Year and Quarter with highest total booked seats: {max_booked_seats}")
Year and Quarter with highest total booked seats: Row(Year=2010, Quarter=1, Total_Booked_Seats=49678)
>>>
>>> min_booked_seats = total_seats_df.orderBy(col("Total_Booked_Seats")).first()
>>> print(f"Year and Quarter with Minimum total booked seats: {min_booked_seats}")
Year and Quarter with Minimum total booked seats: Row(Year=2000, Quarter=4, Total_Booked_Seats=30103)
```

Question 2 Spark Answers of subquestions 2. 3. And 4.

```
>>> data.filter(data["Avg_rev_per_seat"] < 290).count()
9
>>>
>>>
>>>
>>> data.groupBy("Quarter").agg(avg("booked_seats").alias("Avg_Booked_Seats")).show()
+-------+------------------+
|Quarter|  Avg_Booked_Seats|
+-------+------------------+
|      1|41607.666666666664|
|      3| 39386.23809523809|
|      4| 39111.95238095238|
|      2| 38456.95238095238|
+-------+------------------+

>>> data.groupBy("Year").count().show()
+----+-----+
|Year|count|
+----+-----+
|2003|    4|
|2007|    4|
|2015|    4|
|2006|    4|
|2013|    4|
|1997|    4|
|2014|    4|
|2004|    4|
|1996|    4|
|1998|    4|
|2012|    4|
|2009|    4|
|1995|    4|
|2001|    4|
|2005|    4|
|2000|    4|
|2010|    4|
|2011|    4|
|2008|    4|
|1999|    4|
+----+-----+
```

5. While running this query Spark was not working.

```
data.withColumn("Total_Revenue", data["Avg_rev_per_seat"] *
data["booked_seats"]).groupBy("Quarter").agg(sum("Total_Revenue").
alias("Total_Revenue")).orderBy(col("Total_Revenue").desc()).first
()
```