Ford Motor Company Data Analyst Interview Experience

Round : 1

ctc: 15-18 lpa

Yoe: 3-5

SQL

✅ Q1. Find customers who purchased the same vehicle model more than once in the same year.

📘 **Input Table: vehicle_sales**

customer_id	model	purchase_date
101	Mustang	2023-03-15
102	Focus	2022-05-20
101	Mustang	2023-08-10
103	F-150	2023-01-11

customer_id	model	purchase_date
101	Focus	2022-04-01
101	Focus	2022-12-25

✅ **SQL Query:**

```
SELECT customer_id, model, EXTRACT(YEAR FROM purchase_date) AS year
FROM vehicle_sales
GROUP BY customer_id, model, EXTRACT(YEAR FROM purchase_date)
HAVING COUNT(*) > 1;
```

📘 **Output Table:**

customer_id	model	year
101	Mustang	2023
101	Focus	2022

📝 **Explanation:**

- We group by customer_id, model, and purchase year.
- Use EXTRACT(YEAR FROM purchase_date) to extract the year.
- Apply HAVING COUNT(*) > 1 to get combinations purchased more than once in the same year.

✅ Q2. Identify vehicles not serviced in the last 12 months.

📘 **Input Tables:**

1. vehicle_master

vehicle_id	registration_date
V001	2020-01-10
V002	2019-05-15
V003	2021-07-30
V004	2022-10-05

2. service_logs

vehicle_id	service_date
V001	2024-02-20
V002	2023-04-01
V003	2024-08-10

(Assume current date is '2025-05-21')

✅ **SQL Query:**

```
SELECT vm.vehicle_id
FROM vehicle_master vm
LEFT JOIN (
  SELECT vehicle_id, MAX(service_date) AS last_service
  FROM service_logs
  GROUP BY vehicle_id
) sl ON vm.vehicle_id = sl.vehicle_id
WHERE sl.last_service IS NULL
  OR sl.last_service < DATE '2025-05-21' - INTERVAL '12 months';
```

📘 **Output Table:**

vehicle_id
V002
V004

📝 **Explanation:**

- Use a LEFT JOIN to get the latest service date for each vehicle.
- Vehicles with NULL service dates (never serviced) or serviced before '2024-05-21' are filtered.
- Ensures only those not serviced in the last 12 months are selected.

✅ Q3. Get top 3 dealers per region based on profit (revenue - cost) for Q1 2025.

📘 **Input Table: dealer_revenue**

dealer_id	region	month	revenue	cost
D01	North	2025-01-15	100000	60000
D02	North	2025-02-10	120000	70000
D03	North	2025-03-12	110000	65000
D04	North	2025-02-25	95000	50000
D05	North	2025-01-20	98000	60000
D01	South	2025-01-10	87000	47000
D02	South	2025-03-05	93000	58000
D03	South	2025-02-18	91000	57000

✅ **SQL Query:**

```
WITH q1_data AS (
  SELECT *,
      (revenue - cost) AS profit
  FROM dealer_revenue
  WHERE month BETWEEN '2025-01-01' AND '2025-03-31'
),
ranked_data AS (
  SELECT dealer_id, region, profit,
      RANK() OVER (PARTITION BY region ORDER BY profit DESC) AS rnk
  FROM q1_data
)
SELECT dealer_id, region, profit
FROM ranked_data
WHERE rnk <= 3;
```

📘 **Output Table:**

dealer_id	region	profit
D02	North	50000
D03	North	45000
D04	North	45000
D01	South	40000
D03	South	34000
D02	South	35000

📝 **Explanation:**

- Filter only Q1 2025 (Jan to Mar) using WHERE clause.
- Calculate profit as revenue - cost.
- Use RANK() with PARTITION BY region and ORDER BY profit DESC to rank dealers within each region.
- Select top 3 per region using WHERE rnk <= 3.

✅ Q4. Calculate the month-over-month % growth in service appointments in 2024.

📘 **Input Table: service_appointments**

appointment_id	service_date	vehicle_id	cost
A001	2024-01-15	V001	200
A002	2024-01-20	V002	180
A003	2024-02-10	V003	220
A004	2024-02-28	V004	190
A005	2024-03-05	V005	250
A006	2024-03-22	V006	210
A007	2024-04-01	V007	230

✅ **SQL Query:**

```
WITH monthly_counts AS (
  SELECT
    DATE_TRUNC('month', service_date) AS month,
    COUNT(*) AS total_appointments
```

```
  FROM service_appointments
  WHERE service_date BETWEEN '2024-01-01' AND '2024-12-31'
  GROUP BY 1
),
growth_calc AS (
  SELECT
    month,
    total_appointments,
    LAG(total_appointments) OVER (ORDER BY month) AS prev_month_appointments
  FROM monthly_counts
)
SELECT
  month,
  total_appointments,
  ROUND(
    100.0 * (total_appointments - prev_month_appointments) /
NULLIF(prev_month_appointments, 0),
    2
  ) AS pct_growth
FROM growth_calc
ORDER BY month;
```

📘 **Output Table:**

month	total_appointments	pct_growth
2024-01-01	2	NULL

month	total_appointments	pct_growth
2024-02-01	2	0.00
2024-03-01	2	0.00
2024-04-01	1	-50.00

📝 **Explanation:**

- Aggregate appointments by month using DATE_TRUNC.
- Use LAG() to access the previous month's count.
- Calculate % growth: (current−previous)/previous×100(current - previous) / previous × 100(current−previous)/previous×100
- NULLIF prevents division by zero.
- ROUND(..., 2) formats the percentage to 2 decimal places.

✅ Q5. Flag older records where customer email and phone are duplicated.

📘 **Input Table: customer_contacts**

customer_id	email	phone_number	last_updated
1	a@mail.com	1111	2023-01-01
2	a@mail.com	1111	2023-03-01
3	b@mail.com	2222	2022-12-15
4	b@mail.com	2222	2023-02-10
5	c@mail.com	3333	2023-05-01

✅ **SQL Query:**

```
WITH ranked_contacts AS (
  SELECT *,
      ROW_NUMBER() OVER (PARTITION BY email, phone_number ORDER BY last_updated DESC) AS rn
  FROM customer_contacts
)
SELECT customer_id, email, phone_number, last_updated,
    CASE WHEN rn > 1 THEN 'Duplicate_Older' ELSE 'Latest' END AS status
FROM ranked_contacts;
```

📘 **Output Table:**

customer_id	email	phone_number	last_updated	status
2	a@mail.com	1111	2023-03-01	Latest
1	a@mail.com	1111	2023-01-01	Duplicate_Older
4	b@mail.com	2222	2023-02-10	Latest
3	b@mail.com	2222	2022-12-15	Duplicate_Older
5	c@mail.com	3333	2023-05-01	Latest

📝 **Explanation:**

- Use ROW_NUMBER() to rank records per (email, phone_number) pair by last_updated descending.
- Rank = 1 means the **latest**, others are flagged as **older duplicates**.
- The output includes a status column to identify them.

✅ Q6. Find customers who bought both Ford Mustang and Ford Figo.

📘 **Input Table: sales_data**

customer_id	model	purchase_date
1	Ford Mustang	2023-01-10
2	Ford Figo	2023-02-11
1	Ford Figo	2023-03-12
3	Ford Mustang	2023-04-01
4	Ford Figo	2023-04-15
3	Ford Figo	2023-05-20

✅ **SQL Query:**

```
SELECT customer_id
FROM sales_data
WHERE model IN ('Ford Mustang', 'Ford Figo')
GROUP BY customer_id
HAVING COUNT(DISTINCT model) = 2;
```

📘 **Output Table:**

customer_id
1
3

📝 **Explanation:**

- Filter rows where model is either 'Ford Mustang' or 'Ford Figo'.
- Group by customer_id and count **distinct models**.
- Customers with **both models** will have a count of 2.

✅ Q7. Get the latest service record for each vehicle along with cost.

📘 **Input Table: service_records**

vehicle_id	service_date	service_cost
V001	2023-01-10	250
V001	2023-03-15	300
V002	2023-02-05	200
V003	2023-01-20	180
V002	2023-04-12	220

✅ **SQL Query:**

```
WITH ranked_services AS (
  SELECT *,
      ROW_NUMBER() OVER (PARTITION BY vehicle_id ORDER BY service_date DESC) AS rn
  FROM service_records
)
SELECT vehicle_id, service_date, service_cost
FROM ranked_services
```

```
WHERE rn = 1;
```

📘 **Output Table:**

vehicle_id	service_date	service_cost
V001	2023-03-15	300
V002	2023-04-12	220
V003	2023-01-20	180

📝 **Explanation:**

- Use ROW_NUMBER() to assign ranks per vehicle_id based on service_date DESC.
- Filter only the **latest record** per vehicle (rn = 1).
- Returns the **most recent service date and cost** for each vehicle.

PYTHON

✅ Q1. Return vehicles not serviced in the last 12 months

📘 **Sample Input DataFrames:**

```
import pandas as pd

from datetime import datetime, timedelta


df_vehicles = pd.DataFrame({

   'vehicle_id': ['V001', 'V002', 'V003', 'V004'],
```

```
    'registration_date': ['2021-01-01', '2021-03-15', '2022-06-20', '2023-01-05']
})

df_services = pd.DataFrame({
    'vehicle_id': ['V001', 'V002', 'V003'],
    'service_date': ['2023-05-01', '2022-03-01', '2023-10-15']
})
```

✅ **Function:**

```
def vehicles_not_serviced_last_12_months(df_vehicles, df_services):
    df_services['service_date'] = pd.to_datetime(df_services['service_date'])
    cutoff_date = pd.to_datetime("today") - pd.DateOffset(months=12)

    # Get latest service per vehicle
    latest_service = df_services.sort_values('service_date').drop_duplicates('vehicle_id', keep='last')

    # Merge with vehicle list
    merged = df_vehicles.merge(latest_service, on='vehicle_id', how='left')

    # Filter vehicles never serviced or serviced before cutoff
    result = merged[(merged['service_date'].isna()) | (merged['service_date'] < cutoff_date)]

    return result[['vehicle_id']]
```

📘 **Output:**

(Assuming today's date is May 21, 2025)

vehicle_id
V002
V004

📝 **Explanation:**

- Convert service_date to datetime.
- Compute **cutoff = today - 12 months**.
- Find the **latest service per vehicle**.
- Merge with all vehicles and filter:
    - Never serviced (null),
    - Or last serviced > 12 months ago.

✅ Q2. Return duplicate rows based on email and phone

📘 **Sample Input DataFrame:**

```
df_contacts = pd.DataFrame({
  'customer_id': [1, 2, 3, 4, 5, 6],
  'email': ['a@mail.com', 'a@mail.com', 'b@mail.com', 'c@mail.com', 'b@mail.com', 'd@mail.com'],
  'phone_number': ['1111', '1111', '2222', '3333', '2222', '4444']
})
```

✅ **Function:**

```
def get_duplicate_contacts(df_contacts):
    duplicates = df_contacts[df_contacts.duplicated(subset=['email', 'phone_number'], keep=False)]
    return duplicates.sort_values(['email', 'phone_number'])
```

📘 **Output:**

customer_id	email	phone_number
1	a@mail.com	1111
2	a@mail.com	1111
3	b@mail.com	2222
5	b@mail.com	2222

📝 **Explanation:**

- Use duplicated() with keep=False to get **all instances** of duplicates.
- Sorted by email & phone for better readability.

✅ Q3. Create a function that returns a new DataFrame with average service cost by vehicle type

---

📘 **Sample Input: df_service**

```
import pandas as pd

df_service = pd.DataFrame({
    'vehicle_type': ['SUV', 'Sedan', 'SUV', 'Hatchback', 'Sedan', 'SUV'],
```

```
    'service_cost': [500, 300, 450, 200, 350, 550]
})
```

✅ **Function:**

```
def average_cost_by_vehicle_type(df_service):
    return df_service.groupby('vehicle_type', as_index=False)['service_cost'].mean()
```

📘 **Output:**

vehicle_type	service_cost
Hatchback	200.0
SUV	500.0
Sedan	325.0

📝 **Explanation:**

- Group by vehicle_type
- Aggregate using .mean() on service_cost
- Returns a new DataFrame with **average cost per type**

✅ Q4. Define a function to fill null values in mileage column with model-wise mean

---

📘 **Sample Input: df_mileage**

```
df_mileage = pd.DataFrame({
    'vehicle_id': ['V001', 'V002', 'V003', 'V004', 'V005'],
    'model': ['Ford Figo', 'Ford Mustang', 'Ford Figo', 'Ford Figo', 'Ford Mustang'],
```

```
    'mileage': [15.0, 10.0, None, 14.0, None]
})
```

✅ **Function:**

```
def fill_mileage_with_model_mean(df_mileage):
    df_mileage['mileage'] = df_mileage.groupby('model')['mileage'] \
                            .transform(lambda x: x.fillna(x.mean()))
    return df_mileage
```

📘 **Output:**

vehicle_id	model	mileage
V001	Ford Figo	15.0
V002	Ford Mustang	10.0
V003	Ford Figo	14.5
V004	Ford Figo	14.0
V005	Ford Mustang	10.0

📝 **Explanation:**

- Use groupby() on model
- Use transform() to fill NaN values with **model-wise average**
- Keeps DataFrame size unchanged while imputing missing values

---

✅ Q5. Calculate days between consecutive services for a given vehicle

---

📘 **Sample Input: df_service_logs**

```
import pandas as pd


df_service_logs = pd.DataFrame({
   'vehicle_id': ['V001', 'V001', 'V001', 'V002', 'V002'],
   'service_date': ['2022-01-10', '2022-03-10', '2022-06-15', '2023-01-01', '2023-05-01']
})
```

✅ **Function:**

```
def service_gap_days(df_service_logs):
   df = df_service_logs.copy()
   df['service_date'] = pd.to_datetime(df['service_date'])
   df = df.sort_values(['vehicle_id', 'service_date'])
   df['days_between_services'] = df.groupby('vehicle_id')['service_date'].diff().dt.days
   return df
```

📘 **Output:**

vehicle_id	service_date	days_between_services
V001	2022-01-10	NaN
V001	2022-03-10	59
V001	2022-06-15	97
V002	2023-01-01	NaN
V002	2023-05-01	120

📝 **Explanation:**

- Sort by vehicle_id and service_date
- Use groupby and diff() to get time delta in days
- First service per vehicle has NaN gap

✅ Q6. Bucket vehicles as <3 yrs, 3–7 yrs, >7 yrs from registration

---

📘 **Sample Input: df_vehicle**

```
df_vehicle = pd.DataFrame({
  'vehicle_id': ['V001', 'V002', 'V003', 'V004'],
  'registration_date': ['2023-05-10', '2020-01-01', '2015-06-30', '2018-11-20']
})
```

✅ **Function:**

```
from datetime import datetime


def bucket_vehicle_age(df_vehicle):
  df = df_vehicle.copy()
  df['registration_date'] = pd.to_datetime(df['registration_date'])
  current_date = pd.to_datetime("today")
  df['age_years'] = (current_date - df['registration_date']).dt.days / 365


  def assign_bucket(age):
    if age < 3:
      return '<3 yrs'
```

```
    elif 3 <= age <= 7:
        return '3–7 yrs'
    else:
        return '>7 yrs'

df['age_bucket'] = df['age_years'].apply(assign_bucket)
return df[['vehicle_id', 'age_bucket']]
```

📘 **Output** *(assuming today = May 21, 2025)*:

vehicle_id	age_bucket
V001	<3 yrs
V002	3–7 yrs
V003	>7 yrs
V004	3–7 yrs

📝 **Explanation:**

- Convert registration date to datetime
- Compute age in years from today
- Use .apply() with custom logic to assign buckets

✅ Q7. Return top 5 rows after sorting by service cost (descending)

📘 **Sample Input: df_service**

```
df_service = pd.DataFrame({
```

```
    'vehicle_id': ['V001', 'V002', 'V003', 'V004', 'V005', 'V006'],
    'service_cost': [250, 500, 300, 700, 400, 100]
})
```

✅ **Function:**

```
def top_5_by_service_cost(df_service):
    return df_service.sort_values(by='service_cost', ascending=False).head(5)
```

📘 **Output:**

vehicle_id	service_cost
V004	700
V002	500
V005	400
V003	300
V001	250

📝 **Explanation:**

- Sort DataFrame by service_cost in descending order
- Return **top 5 records** using .head(5)