## Assumptions:

Data & Relationships
- users.id and firms.id are stable unique keys; events.user_id and events.firm_id reference them.
- Joins from events to users and firms have no missing references.
- arr_in_thousands and firm_size are 1:1 with firms.
- title is 1:1 in users.
- created dates are 1:1 in both firms and users.

Time & Scope
- All timestamps/dates are treated as UTC.

Data Integrity
- No nulls or orphan rows in source tables.

Raw Tables

- Events
  - Each row represents one user query (no resubmissions).
  - feedback_score is required and ranges 1–5.
  - In Google Sheets, created displayed as MM/DD/YYYY but retained time.
    - I duplicated the column with TEXT(cell, "YYYY-MM-DD HH:MM:SS"), pasted values over the original, and removed the helper column to ensure accurate export.

- Firms
  - created is the firm’s account creation date (not necessarily first active date).
  - firm_size is provisioned seats (not necessarily active users).
  - arr_in_thousands is annual contract value (ACV) in thousands of USD.

- Users
  - created is the user's account creation date (not necessarily the first active date).
  - title uses a finite set: Associate, Junior Associate, Senior Associate, Partner.

Global Metrics:
- num_queries: Engagement volume. Shows how much users use the product.
- num_docs: Engagement depth. Signals deeper use than a general query.
- avg_feedback_score: Quality metric. Tells you how satisfied they are, but most subjective metric.

## Approach:

Quick Setup
- Prerequisites: python 3.10+, postgres 14 running, dbt-postgres 1.10.x
- Create venv: python3 -m venv harvey_dbt_env && source harvey_dbt_env/bin/activate
- Install dbt: pip install dbt-postgres
- Profile (~/.dbt/profiles.yml)

```
harvey_ae_takehome:
outputs:
    dev:
    dbname: harvey_db
    host: localhost
    pass: dbt_password
    port: 5432
    schema: analytics
    threads: 4
    type: postgres
    user: dbt_user
target: dev
```

- Import raw tables into harvey_db.harvey_raw schema
- Create staging, intermediate, and marts folders, as well as dbt_project.yml
- Create harvey_raw_sources.yml in staging folder

Modeling
- Staging models (stg_users, stg_firms, stg_events) rename column names and cast types.
  - materialized as views
  - converted arr_in_thousands to arr in stg_firms
- Intermediate:
  - materialized as views
  - int_user_monthly_activity: per-user per-month metrics (counts, event-type splits, avg feedback, percentile-based engagement score).
  - int_firm_usage_metrics: per-firm metrics (active_days, users, events, docs, per-day/per-user rates).
  - int_daily_event_metrics: daily events (distinct users/firms, docs, avg feedback).
- Marts:
  - materialized as tables
  - user_engagement: dependent on int_user_monthly_activity.
  - firm_usage_summary: dependent on int_firm_usage_metrics.
  - daily_event_summary: dependent on int_daily_event_metrics.

Metrics (created in intermediate models)

int_user_monthly_activity
- engagement_score:

```
        -   engagement_score: weighted mix of monthly percentiles
        -   Percentiles computed per month using percent_rank().
            -   Scores and levels reset monthly to reflect relative engagement changes over time
                -   Inputs: num_queries (60%), num_docs (30%), avg_feedback (10%).
    -   engagement_level:
        -   Based on the percentile of the monthly engagement score for a user:
            -   high: top 20%  |  medium: 30-80th percentile  |  low: bottom 30%
            -   engagement scores are weighted against

    int_firm_usage_metrics
    -   "Active" user: 1+ event in a month.
    -   arr_per_user based on number of active users, not firm_size.
```

# Analytics Questions

## 1. Based on your user_engagement model, how would you define a power user?

For an LLM/AI assistant product, a power user typically:

1. Shows high engagement intensity
   - e.g. high engagement_level in two of past three months

2. Shows consistent monthly activity (not one-time use)
   - e.g. active across consecutive months

3. Uses multiple features (not just one query type)
   - e.g. engages in 2 of the 3 types of queries: assistant, workflow, and vault

In [143…

```python
# import user_engagement table into user_engagement_df

from sqlalchemy import create_engine
import pandas as pd

# create SQLAlchemy engine (using credentials from dbt profile)
engine = create_engine(
    "postgresql+psycopg2://dbt_user:dbt_password@localhost:5432/harvey_db"
)

# query from schema.table
ue_query = "select * from analytics_final.user_engagement;"
```

```
user_engagement_df = pd.read_sql(ue_query, engine)

print(user_engagement_df.head())
```

```
  activity_month                         user_id  firm_id  num_active_days  \
0     2024-04-01  716f63ce0d669b9700ca71aa23837407     1068                1
1     2024-04-01  8256a0d1a64fa1f691480f61c43a5455     1738                1
2     2024-04-01  9a6aa2b3a5cc29c1da3f9cf18bd15a33     1286                1
3     2024-04-01  ea0a2e9d33d0c3c444049de4d38bacf0     1035                1
4     2024-04-01  fcaebc5729c112c77cbb42a244eaba9f     1738                1

   num_queries  num_workflow_queries  num_vault_queries  \
0            1                     0                  0
1            1                     1                  0
2            1                     0                  0
3            1                     1                  0
4            1                     0                  0

   num_assistant_queries  avg_queries_per_active_day  num_docs  \
0                      1                         1.0         1
1                      0                         1.0         1
2                      1                         1.0         1
3                      0                         1.0         1
4                      1                         1.0         1

   avg_docs_per_query  avg_feedback_score last_active_date  engagement_score  \
0                 1.0                 1.0       2024-04-23               0.0
1                 1.0                 1.0       2024-04-23               0.0
2                 1.0                 1.0       2024-04-20               0.0
3                 1.0                 1.0       2024-04-09               0.0
4                 1.0                 1.0       2024-04-25               0.0

   pr_engagement_score engagement_level
0                  0.0              low
1                  0.0              low
2                  0.0              low
3                  0.0              low
4                  0.0              low
```

In [142…

```
# ensure activity_month is datetime and data is sorted
user_engagement_df['activity_month'] = pd.to_datetime(user_engagement_df['activity_month'])
user_engagement_df = user_engagement_df.sort_values(['user_id', 'activity_month'])

# ------------------------------------------
# 1. High engagement
# ------------------------------------------
user_engagement_df['is_high_engagement'] = (user_engagement_df['engagement_level'] == 'high')
```

```python
# ---------------------------------------------
# 2. Active in two consecutive months
# ---------------------------------------------
user_engagement_df['prev_activity_month'] = user_engagement_df.groupby('user_id')['activity_month'].shift(1)

# compute the difference in months
user_engagement_df['month_diff'] = (
    (user_engagement_df['activity_month'].dt.year - user_engagement_df['prev_activity_month'].dt.year) * 12 +
    (user_engagement_df['activity_month'].dt.month - user_engagement_df['prev_activity_month'].dt.month)
)

# flag if active in consecutive months
user_engagement_df['active_two_consecutive_months'] = (user_engagement_df['month_diff'] == 1)


# ---------------------------------------------
# 3. Used ≥ 2 LLM interaction types
# ---------------------------------------------
user_engagement_df['num_interaction_types_used'] = (
    ((user_engagement_df['num_assistant_queries'] > 0).astype(int)) +
    ((user_engagement_df['num_workflow_queries'] > 0).astype(int)) +
    ((user_engagement_df['num_vault_queries'] > 0).astype(int))
)

user_engagement_df['used_2plus_interaction_types'] = user_engagement_df['num_interaction_types_used'] >= 2

# ---------------------------------------------
# 4. Combine into overall power user flag
# ---------------------------------------------
user_engagement_df['is_power_user'] = (
    user_engagement_df['is_high_engagement'] &
    user_engagement_df['active_two_consecutive_months'] &
    user_engagement_df['used_2plus_interaction_types']
)


# pick relevant columns (will ignore any that don't exist)
cols = [
    "user_id",
    "activity_month",
    "engagement_level",
    "num_interaction_types_used",
    "active_two_consecutive_months",
    "used_2plus_interaction_types",
    "is_power_user",
]
```

```
existing_cols = [c for c in cols if c in user_engagement_df.columns]

power_users = (
    user_engagement_df.loc[user_engagement_df["is_power_user"], existing_cols]
    .sort_values(["user_id", "activity_month"])
    .reset_index(drop=True)
)

print(power_users)
```

```
                             user_id activity_month engagement_level  \
0    0291d66d38443f5280c0953fb03525d9     2024-05-01             high
1    02ed9ca7d853d72ec4cd48332eb945c9     2024-06-01             high
2    03ef17b7c69ebaefe980d999db018883     2024-06-01             high
3    041b14a215e128bc98d1cca191eed3fa     2024-06-01             high
4    0523d4fcbe429b3e6eaa4054860bdd35     2024-05-01             high
..                                ...            ...              ...
446  fec05937cb971a78fac58924281839f2     2024-05-01             high
447  fec05937cb971a78fac58924281839f2     2024-06-01             high
448  ff04a64416c7b558cb2fcf6077853873     2024-05-01             high
449  ff04a64416c7b558cb2fcf6077853873     2024-06-01             high
450  ff9af59ee4e334a5cbf9cbc61b2ab7ad     2024-06-01             high

     num_interaction_types_used  active_two_consecutive_months  \
0                             3                           True
1                             3                           True
2                             3                           True
3                             3                           True
4                             3                           True
..                          ...                            ...
446                           3                           True
447                           2                           True
448                           3                           True
449                           3                           True
450                           3                           True

     used_2plus_interaction_types  is_power_user
0                            True           True
1                            True           True
2                            True           True
3                            True           True
4                            True           True
..                            ...            ...
446                          True           True
447                          True           True
448                          True           True
449                          True           True
450                          True           True

[451 rows x 7 columns]
```

In [136…

```python
# 451 power-user records identified across months

num_power_user_rows = len(user_engagement_df[user_engagement_df['is_power_user'] == True])
total_rows = len(user_engagement_df)

print("Number of power user records:", num_power_user_rows, "of", total_rows, "records")
```

```
Number of power user records: 451 of 4709 records
```

In [137…
```python
# 356 unique power users of 2948 total

unique_power_users = user_engagement_df.loc[user_engagement_df['is_power_user'], 'user_id'].nunique()
unique_users = user_engagement_df['user_id'].nunique()

print("Number of power users:", unique_power_users, "of", unique_users , "total users (12%)")
```

```
Number of power users: 356 of 2948 total users (12%)
```

In [138…
```python
# power users by month

power_users_by_month = (
    user_engagement_df[user_engagement_df['is_power_user']]
    .groupby('activity_month')['user_id']
    .nunique()
    .reset_index(name='num_power_users')
)

print(power_users_by_month)
```

```
  activity_month  num_power_users
0     2024-05-01              222
1     2024-06-01              229
```

## What potential issues or data quality concerns does the data surface?

(These could be anomalies, missing data, inconsistent definitions, etc.)

- One user is associated with 2 firms in the events data. Since int_user_monthly_activity groups by month and user, only one firm will be associated with this user's data.
- The events dataset spans ~3 months. Because the power-user rule requires activity in two consecutive months, only users active in adjacent months can qualify. So roughly one-third of the window doesn't qualify in that metric.
- The events table lacks a stable primary key (e.g., event_id) or a natural de-duplicate key. Without an ID or a query payload/fingerprint, distinguishing true retries from legitimate repeat actions is ambiguous and only approximable by timestamp.
- ARR is 1:1 with firm (stored as a single current value). It doesn't capture historical contract changes (upsells, downsells, renewals), so time-series or per-period ratios implicitly apply today's ARR to past periods. Similarly, firm_size can change over time but is also stored as a single value.
- Title is 1:1 with user (single current value). It doesn't capture historical role changes (e.g., promotions or transfers), so time-series analysis implicitly applies current title to past periods (or initial title to current periods).

In [139…

```
# User associated with 2 firms
# ----------------------------


# query from schema.table
stg_events_query = "select * from analytics_staging.stg_events;"
stg_events_df = pd.read_sql(stg_events_query, engine)

# ensure datetime
stg_events_df["created_at"] = pd.to_datetime(stg_events_df["created_at"])

# create month column
stg_events_df["created_month"] = stg_events_df["created_at"].dt.to_period("M").dt.to_timestamp()

out = (
    stg_events_df.groupby(["created_month", "user_id"])["firm_id"]
    .nunique()
    .reset_index(name="num_firms")
    .query("num_firms > 1")
)

print(out.head())
```

```
    created_month                           user_id  num_firms
523     2024-04-01  633c06694d118c8578aac99bfd96d5a7          2
2041    2024-05-01  633c06694d118c8578aac99bfd96d5a7          2
```

In [131…

```
# Events table spans about 3 months
# ---------------------------------

min_ts = stg_events_df["created_at"].min(skipna=True)
max_ts = stg_events_df["created_at"].max(skipna=True)

print("Event timestamps range from", min_ts ,"to", max_ts)
```

```
Event timestamps range from 2024-04-02 00:25:59 to 2024-06-26 03:31:54
```