career - Churn prediction project

Help me build this project :

```
Project : Churn Prediction + Churn Reason
Segmentation (SaaS / Consumer Apps) Problem
- Predict which users will churn and categorize
why (price sensitivity, inactivity, feature
dissatisfaction, value mismatch).
- Data Preprocessing :
  - Session logs → aggregate weekly usage.
  - Create RFM-like engagement features:
     - Recency: days since active
     - Frequency: sessions/week
     - Monetary (if subscription): revenue per
user
  - Standardize logs: remove bots/spam data.
- Modeling :
  - Binary classifier (LightGBM) for churn.
  - Clustering churners into segments using
KMeans/DBSCAN.
- Insights :
  - "Silent churners" vs "value-seekers" vs
"frustrated users".
  - Recommend product-level actions:
     - Feature education for low-engagement
users
     - Pricing experiments for price-sensitive
cohort
     - UX redesign for frustrated cohort
This shows business + product mindset: not just
prediction, but actionability.
```

Nice — this is a great project: prediction **plus** actionable segmentation. Below I give you a full, runnable blueprint you can take to stakeholders, plus code snippets (Python + SQL), evaluation plans, segment definitions, and recommended product experiments. I'll make reasonable assumptions (time-based churn label, session log schema) and give defaults you can adjust.

1 — Overview / goal

Predict which users will churn in a future window, and **segment churners by likely reason** so product/marketing can take targeted actions:

- Output A: `churn_probability` per user (LightGBM).
- Output B: cluster assignment for churners with human-readable labels (KMeans/DBSCAN + manual mapping).
- Business outcomes: reduce churn, run targeted experiments, measure lift.

---

2 — Assumptions & definitions

- **Event log** (session-level): `user_id`, `event_time` (UTC timestamp), `session_id`, `event_type`, `duration_seconds`, `ua` (user agent), `device`, `country`, `revenue` (per session if any).
- **Subscription table**: `user_id`, `plan`, `price`, `start_date`, `cancel_date` (nullable).
- **Churn label**: define churn as *no session/activity in the 30 days after the observation window end*. (Use a 30/60/90 day sensitivity analysis.)
- **Observation window**: last N days (e.g., 90 days) of logs to create features; label is next 30 days.
- **Temporal split**: train on earlier time windows, validate on later time windows (no leakage).

---

3 — Data pipeline (high level)

1. Ingest raw session logs → S3 / data warehouse.
2. Clean logs: filter bots/spam, unify timezone, dedupe session_ids.
3. Aggregate weekly usage per user (or daily + aggregate to weekly).
4. Generate RFM-like and behavioral features.
5. Feature store: store feature vectors keyed by `user_id` + `snapshot_date`.
6. Train LightGBM classifier.
7. For predicted churners (or known churners in training), cluster them to get segments.
8. Produce dashboard + orchestrated experiments.

---

4 — Data cleaning & bot removal (practical rules)

- Exclude sessions with user agents matching known bot patterns (`bot`, `spider`, `curl`, `python-requests`).
- Exclude sessions with `duration_seconds = 0` combined with >1000 events in same second.
- Remove users with extremely high session counts per minute/day (top 0.01%) unless verified.
- Filter out internal IP ranges / QA accounts using allowlist.
- Remove events with impossible timestamps (future dates).

---

5 — Feature engineering (recommended set)

Basic RFM & subscription features

- `recency_days` = days since last session at snapshot.
- `frequency_week` = avg sessions per week (last 4 weeks).
- `session_count_28d` = total sessions last 28 days.
- `avg_session_duration` = mean session length (sec).
- `revenue_90d` = sum of revenue last 90 days.
- `plan_price` = current plan price, `is_trial` flag, `days_on_plan`.

Behavioral features (product- & funnel-aware)

- `pct_sessions_mobile`, `pct_sessions_desktop`.
- `feature_X_used_count` for key product features (e.g., file_uploads, reports_created).
- `time_between_sessions_median`.
- `dau_waus_ratio` or `stickiness` = DAU/WAU.
- `num_help_center_views`, `num_support_tickets`.
- `num_failed_payments`, `customer_satisfaction_score` (if exists).
- `change_in_freq` = (sessions in last 7 days) / (sessions in previous 7 days) - 1.

Cross-sectional & derived

- `trend_sessions_4w` = slope of weekly session counts (linear regression slope).
- `engagement_cluster` (precomputed embedding).
- `has_recent_feature_search` (boolean).

Encoding & scaling

- Categorical: target encoding for high-cardinality (e.g., country, referrer) with smoothing.
- Numeric: log transform skewed features (e.g., revenue), then standardize (z-score).
- Missing: treat missing revenue = 0, missing categorical = `unknown`.

---

6 — SQL: Aggregating weekly usage + RFM (example)

Adjust table/column names to your warehouse.

```sql
-- compute weekly aggregates per user, for snapshot_date = '2025-11-01'
WITH sessions AS (
  SELECT
    user_id,
    DATE_TRUNC('week', event_time) AS week,
    COUNT(DISTINCT session_id) AS sessions_in_week,
    SUM(duration_seconds) AS total_duration,
    SUM(coalesce(revenue,0)) AS revenue_week,
    COUNT_IF(event_type = 'feature_X') AS feature_x_count
  FROM events
  WHERE event_time >= DATE '2025-07-01' AND event_time < DATE '2025-11-01'
  GROUP BY user_id, week
)
, last_activity AS (
  SELECT user_id, MAX(event_time) AS last_activity FROM events
  WHERE event_time < '2025-11-01' GROUP BY user_id
)
, rfm AS (
  SELECT
    s.user_id,
    AVG(s.sessions_in_week) AS frequency_week,
    SUM(s.revenue_week) AS revenue_90d,
    AVG(s.total_duration) AS avg_session_duration,
    SUM(s.feature_x_count) AS feature_x_90d
  FROM sessions s
  GROUP BY s.user_id
)
SELECT
  r.user_id,
  DATE_PART('day', DATE '2025-11-01' - la.last_activity) AS recency_days,
  r.frequency_week,
  r.revenue_90d,
  r.avg_session_duration,
  r.feature_x_90d
FROM rfm r
LEFT JOIN last_activity la USING (user_id);
```

---

7 — Label generation (churn)

Define snapshot end `t0`. Label a user as churn = 1 if there is **no session** in `(t0, t0 + label_window_days]`. Example: label_window_days = 30.

```sql
-- snapshot t0 = '2025-11-01'
WITH last_before AS (
  SELECT DISTINCT user_id FROM events WHERE event_time <= '2025-11-01'
),
activity_after AS (
  SELECT DISTINCT user_id FROM events WHERE event_time > '2025-11-01' AND
event_time <= '2025-12-01'
)
SELECT lb.user_id,
       CASE WHEN a.user_id IS NULL THEN 1 ELSE 0 END AS churn_label
FROM last_before lb
LEFT JOIN activity_after a USING (user_id);
```

---

8 — Modeling: LightGBM (Python skeleton)

Key points:

- Use **time-based train/val/test splits** (no random split).
- Handle class imbalance: `scale_pos_weight` or use focal loss / sampling.
- Use early stopping, categorical features in LightGBM.
- Keep a calibration step if business cares about probability.

python

```python
# pip install lightgbm scikit-learn pandas shap umap-learn
import pandas as pd
import numpy as np
from sklearn.model_selection import GroupKFold, train_test_split
import lightgbm as lgb
from sklearn.metrics import roc_auc_score, precision_recall_curve, auc

# load features table (user_id, snapshot_date, features..., churn_label)
df = pd.read_parquet("features.parquet")
# assume snapshot_date is datetime; create temporal split
train = df[df.snapshot_date < '2025-09-01']
val   = df[(df.snapshot_date >= '2025-09-01') & (df.snapshot_date < '2025-10-
01')]
test  = df[df.snapshot_date >= '2025-10-01']

features = [c for c in df.columns if c not in
('user_id','snapshot_date','churn_label')]

dtrain = lgb.Dataset(train[features], label=train['churn_label'])
dval   = lgb.Dataset(val[features], label=val['churn_label'])

params = {
    'objective': 'binary',
    'metric': 'auc',
    'boosting': 'gbdt',
    'num_leaves': 64,
    'learning_rate': 0.05,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'seed': 42
}

model = lgb.train(params, dtrain, valid_sets=[dval],
early_stopping_rounds=100, num_boost_round=2000)
# predict on test
test_preds = model.predict(test[features])
print("ROC AUC:", roc_auc_score(test['churn_label'], test_preds))
```

Thresholding & business metrics

- Compute precision@k or lift@k: useful for targeting top X% of likely churners.
- Select threshold by optimizing `expected_revenue_saved - cost_of_intervention`.

---

9 — Explainability (feature-level & individual)

- Use **SHAP** for global and local explanations (average SHAP per feature across churners).
- Feature importances help cluster interpretation.
- Create rule-based labels from SHAP / feature thresholds.

```python
import shap
explainer = shap.TreeExplainer(model)
shap_vals = explainer.shap_values(test[features])
shap.summary_plot(shap_vals, test[features])
```

---

10 — Clustering churners into reason segments

Approach A: Feature-driven clustering (recommended)

1. Take churners (true churners in historic data) or predicted-high-risk users.
2. For clustering use standardized subset of features that indicate reason:
   - engagement features: `frequency_week`, `recency_days`, `avg_session_duration`, `feature_x_count`
   - monetary features: `revenue_90d`, `plan_price`, `is_trial`
   - friction features: `num_support_tickets`, `num_failed_payments`, `help_center_views`, `error_events_count`
3. Dimensionality reduction (UMAP or PCA) to 10 dims.
4. Apply `KMeans` for a fixed # clusters (e.g., 3–6) or `DBSCAN` if you want density clusters and noise.
5. Evaluate clusters with Silhouette, Davies-Bouldin, cluster stability on bootstrap.

Example:

```python
```

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import umap

churners = train[train.churn_label==1]
cluster_features =
['recency_days','frequency_week','avg_session_duration','revenue_90d',

'num_support_tickets','num_failed_payments','feature_x_90d','change_in_freq']
X = churners[cluster_features].fillna(0)
X = np.log1p(X)  # reduce skew where appropriate
scaler = StandardScaler()
Xs = scaler.fit_transform(X)

# reduce
um = umap.UMAP(n_components=5, random_state=42)
X_red = um.fit_transform(Xs)

# cluster
k = 4
km = KMeans(n_clusters=k, random_state=42)
labels = km.fit_predict(X_red)
churners['segment_id'] = labels
```

Interpreting clusters -> human labels

For each cluster compute median of key features and create human-readable mapping:

- Cluster where `recency_days` low, `frequency_week` very low, minimal support tickets => **"Silent churners"** (lost interest / low engagement).
- Cluster where `revenue_90d` low, on low-cost plan or trial, many price-related cancellations => **"Value-seekers / price-sensitive"**.
- Cluster with high `num_support_tickets`, `error_events_count`, low `avg_session_duration` => **"Frustrated / UX issues"**.
- Cluster with high usage but churn after payment issues or canceled subscription (`num_failed_payments`) => **"Payment friction"**.

Create segment profile reports (median feature values + examples).

---

11 — Validating segments

- **Characterize** each cluster with summary stats and sample user journeys.
- **Back-test**: do certain segments have different LTV, reactivation rate, or time-to-churn?
- **Qualitative validation**: sample users from each segment, review support tickets / session replay to confirm reasons.
- **A/B test readiness**: ensure enough volume per segment to power experiments.

---

12 — Actionable recommendations per segment

- **Silent churners (low engagement)**
  - Action: In-app onboarding refresh + personalized feature education emails + time-limited content -> measure reactivation (open-to-session conversion).
  - KPI: % who return and perform core action within 7 days.
- **Value-seekers / Price-sensitive**
  - Action: targeted pricing experiments (discounts, feature-limited lower tiers, trial extension). Consider price elasticity analysis.
  - KPI: conversion to paid / retention after discount ends.
- **Frustrated users (UX / bugs)**
  - Action: priority UX audit, session replays, bug fixes + proactive support outreach with voucher or personal onboarding.
  - KPI: support ticket reopen rate, NPS, retention.
- **Payment friction**
  - Action: retry logic, alternative payment flows, and clearer billing emails.
  - KPI: successful payment after attempted retry, churn due to payment problems.

---

13 — Experiment design (examples)

- Randomized controlled trials per segment. Example: For silent churners, randomly assign 10% to onboarding email + in-app tour, compare 30-day retention.
- Use holdout groups and measure lift. Compute required sample size from baseline churn rate and expected uplift.

---

14 — Monitoring & productionization

- **Model serving**: batch scoring nightly, store `churn_probability`; real-time scoring only if needed.
- **Monitoring alarms**:
  - Prediction drift: feature distribution drift (KL divergence), target drift.
  - Model performance: weekly ROC AUC, precision@k.
  - Data quality: missing rates per feature.
- **Retraining**: monthly or when performance drops beyond threshold.
- **Observability**: log predictions, actual outcomes, and model version.

---

15 — Dashboard & stakeholder outputs

Create dashboards showing:

- Top 20 features by SHAP for churn.
- Churn probability histogram.
- Segment counts, segment-specific retention curves.
- A/B experiment results (treatment vs control).
- Lists: Top N high-risk users for outreach.

Suggested visuals: UMAP 2D with clusters, SHAP summary, retention curves per segment, lift charts.

---

16 — KPIs and evaluation metrics

- Model: ROC AUC, PR AUC, Precision@k, Recall@k, F1 (for chosen threshold), Calibration (Brier score).
- Business: Retention rate uplift, ARPU, LTV, Cost per saved user vs revenue retained.

---

17 — Example end-to-end minimal checklist (MVP)

1. Collect 3 months of session logs + subscription/payment data.
2. Clean logs (bot filtering).
3. Build feature pipeline (script or Airflow job).
4. Create churn labels (30-day definition) and build train/val/test with time splits.
5. Train LightGBM; evaluate & calibrate.

6. Create clustering on historical churners (KMeans), map clusters -> reasons.
7. Prepare dashboard & deliver segment playbooks.
8. Run one pilot experiment per segment (e.g., onboarding for silent churners).
9. Monitor and iterate.

---

18 — Code snippets: end-to-end skeleton (condensed)

I already showed LightGBM and clustering snippets. Combine them into a pipeline:

- `etl.py` (SQL -> features)
- `features.py` (feature transformations)
- `train.py` (model training + SHAP)
- `segment.py` (clustering + profiling)
- `serve.py` (batch scorer)

If you want, I can scaffold these files for you with runnable code and a sample synthetic dataset.

---

19 — Risks & mitigations

- **Label noise**: users might temporarily be inactive; use different windows and survival models to reduce mislabeling.
- **Confounding**: marketing campaigns / seasonality can bias model — include campaign flags, use time-based splits.
- **Privacy**: avoid storing PII in plain text; follow your org's data retention and security rules.
- **Actionability**: segments should be interpretable by product — prioritize simple, reproducible rules.

---

20 — Next steps I can do for you (pick any)

- Generate full runnable repo scaffold (ETL + feature store + models + dashboards).
- Produce synthetic sample dataset and run the whole pipeline end-to-end (train + cluster + visualize).
- Write SQLs adapted to your warehouse (BigQuery / Redshift / Snowflake).

- Create experiment sample size calculator for your baseline churn.

Tell me which of the above you want next and I'll produce it right away (no waiting). If you want the runnable code, say which stack you prefer (pandas + scikit-learn, or Spark + PySpark) and whether to use Snowflake/BigQuery style SQL.

Now take me step-by-step through:
- Data collection : what data can I use (cite links)
- data preprocessing
- exploratory data analysis
- model building
- analysis

Nice — I'll walk you through a concrete, **step-by-step** plan for *Churn prediction + churn-reason segmentation*, with what data to collect (and example links), exact preprocessing steps, EDA recipes, modelling pipeline (LightGBM) and the analysis + interpretation steps (SHAP + clustering → human labels + experiments). I cite authoritative resources as we go so you can read deeper.

---

1) DATA COLLECTION — what to collect (and example datasets)

Collect data that lets you reconstruct user journeys, payments, product usage events, and support interactions.

Essential tables / sources

- **Event / session logs** — one row per event or session: `user_id`, `timestamp`, `session_id`, `event_type` (page_view, feature_use, error), `duration_seconds`, `properties` (e.g., feature name), `user_agent`, `ip`, `device`, `country`. Example synthetic SaaS datasets you can use to prototype: the *SaaS Subscription & Churn Analytics* dataset on Kaggle. Kaggle
- **Subscription / billing table** — `user_id`, `plan_id`, `price`, `billing_cycle`, `start_date`, `end_date` / `cancel_date`, `payment_status`, `failed_payments_count`.

- **User profile table** — `user_id`, `signup_date`, `acquisition_channel`, `plan_type`, `country`, `signup_marketing_campaign`.
- **Support / feedback** — `ticket_id`, `user_id`, `created_at`, `issue_type`, `resolution_time`, `satisfaction_score`.
- **Product metadata / features** — mapping for event_type → product feature categories (so you can count feature usage).
- **Optional**: session replays, NPS surveys, email engagement (opens/clicks), A/B experiment logs.

Why RFM-like data
Use Recency/Frequency/Monetary style engagement features — this is a standard approach to summarize user engagement and purchase power. (Good references: RFM definitions and best practices.) Investopedia +1

If you need prototyping data: Kaggle has multiple churn datasets (telco, synthetic SaaS) that are great for building the pipeline. Use them to test code and clustering. Kaggle +1

---

2) DATA PREPROCESSING — step-by-step (exact transformations & rules)

A. Ingest & unify

1. Consolidate logs into a single time zone (UTC) and canonical `user_id`.
2. Ensure schema: `event_time` (datetime), `user_id`, `session_id`, `event_type`, `duration_seconds`, `revenue`.

B. Sessionization

- If you only have events (page_view etc.), define sessions: group events by `user_id` + 30-minute inactivity gap to create `session_id` and `session_start`, `session_end`, `session_duration`. (Standard practice.)

C. Bot / noise removal (concrete rules)

- Drop events with `user_agent` matching bot patterns (`bot`, `spider`, `curl`, `python-requests`).
- Remove sessions with `duration_seconds == 0` and >1000 events per second (likely synthetic).

- Exclude internal/test accounts (filter by email domains, IP allowlist).
  These heuristics are standard for cleaning product analytics logs.

D. Dedup & time sanity

- Deduplicate identical event rows.
- Remove events with timestamps in the future or before product launch.

E. Create base aggregates (raw features)

- daily and weekly aggregates per user: `sessions_per_day`, `sessions_last_7d`, `sessions_last_28d`, `avg_session_duration`, `feature_X_count_last_28d`, `days_since_last_activity`.
- billing aggregates: `revenue_last_90d`, `failed_payments_last_90d`, `is_trial`, `days_on_plan`.

F. Imputation & transforms

- Missing revenue → 0; missing categorical → `unknown`.
- Right-skewed numeric features (revenue, counts) → `log1p` transform; then standardize (z-score) for models that require scaling for clustering.
- Encode categorical features: target/leave-one-out encoding for high-cardinality fields (referrer, campaign) *with* time-aware smoothing to avoid leakage.

G. Label creation (time-aware, to avoid leakage)

- Define `snapshot_date = t0`. Use features from `t0 - observation_window` (e.g., last 90 days) to build feature vector. Label **churn = 1** if no activity in `(t0, t0 + label_window]` (commonly 30 days). This temporal definition prevents label leakage. (Many churn papers use a no-activity window; choose 30/60/90 and test sensitivity.)

H. Train/val/test split

- Use **time-based splits** (train on older snapshots, validate on newer). Do *not* random-split users across time (avoids leakage from seasonality or campaigns).

---

3) EXPLORATORY DATA ANALYSIS — what to run and what it tells you

A. Basic diagnostics

- Churn rate over time (plot weekly/monthly churn).

- Distribution of `recency`, `frequency`, `revenue` (RFM histograms). Use log scales for skewed metrics.

B. Cohort & retention analysis

- Create weekly cohorts by `signup_week` and plot retention curves (survival / retention at 1d/7d/30d). This reveals product onboarding problems vs long-term decline.

C. Feature-target relationships

- Boxplots / violin plots of `sessions_last_28d` and `revenue_90d` grouped by churn label.
- Correlation heatmap (Spearman for counts) to find redundant features.

D. Time-series & trend checks

- For each user compute session-count slope (linear trend of weekly sessions). Negative large slope often precedes churn.

E. Segmented EDA for likely reasons

- Create candidate reason flags and validate:
  - **Price sensitivity**: low `revenue`, short `days_on_plan`, many `trial` users, or coupon use before churn.
  - **Inactivity**: very low `frequency` + high `recency`.
  - **Frustration / friction**: high `support_tickets`, many `error_events`, short `avg_session_duration`.
- Cross-check against session replays / tickets to qualitatively confirm.

F. Silent churn check
Articles and product analytics teams report a large fraction of churn is *silent* (no support ticket, no complaint) — you'll likely see many churners with low support interactions (means proactive outreach needs different tactics). Read Mixpanel's guide on silent churn.

Mixpanel

---

4) MODEL BUILDING — feature set, LightGBM setup, training & evaluation

A. Feature families to include (priority order)

1. **RFM**: `recency_days`, `sessions_last_7d`, `sessions_last_28d`, `frequency_week`.

   Investopedia

2. **Monetary/billing**: `revenue_90d`, `plan_price`, `is_trial`, `failed_payments`.
3. **Behavioral**: `feature_A_count`, `pct_sessions_mobile`, `avg_session_duration`, `time_between_sessions_median`.
4. **Friction indicators**: `support_tickets_30d`, `error_event_count`, `help_center_views`.
5. **Derived**: `trend_sessions_4w` (slope), `change_in_freq` (week-over-week %), campaign/time features.

B. Model choice & why

- Use **LightGBM**: fast, handles missing values and categorical features (or use target-encoding) and typically strong for tabular churn tasks. See LightGBM quick start and Python API for binary classification. LightGBM

C. Handling class imbalance

- If churn is rare, use `scale_pos_weight` or `is_unbalance` in LightGBM, or sample (SMOTE for non-tree methods, but less needed for tree boosting). Optimize for business metric (precision@k) instead of overall accuracy.

D. Cross-validation

- Time-based CV: sliding windows (train on months T..T+n, validate on next window). Also use grouped CV by `user_id` if you have multiple snapshots per user.

E. Hyperparameters (starting point)

- `objective='binary'`, `metric='auc'`, `learning_rate=0.03–0.1`, `num_leaves=31–128`, `max_depth=-1`, `feature_fraction=0.7–0.9`, `bagging_fraction=0.7–0.9`, `early_stopping_rounds=50`. Tune with Optuna or grid search.

F. Evaluation metrics

- **Model metrics**: ROC AUC, PR AUC, Precision@k, Recall@k, Brier score (calibration). For business decisions use **Precision@topK** or expected revenue saved.
- **Calibration**: If you act per probability (e.g., send offers to users with p>0.6), calibrate predicted probabilities (Platt scaling / isotonic).

G. Explainability

- Use SHAP (TreeExplainer) to get global feature importance and local explanations for flagged users. SHAP helps convert model signals into product actions (e.g., if `feature_x_count` is the top SHAP for many churners, that suggests feature education). (SHAP is commonly used with tree models.)

H. Reference tutorials / end-to-end examples

- There are many step-by-step LightGBM churn tutorials and end-to-end churn guides to follow for code structure / eval. (See LightGBM docs and public tutorials.) LightGBM +1

---

5) SEGMENTATION / ANALYSIS — how to map churners to *reasons*

A. Two-phase approach (recommended)

1. **Supervised prediction**: train LightGBM to get `churn_prob`.
2. **Clustering churners**: cluster only the users who churned historically (or who are predicted high-risk), using features designed to reflect reasons (monetary, engagement, friction). Use KMeans or DBSCAN depending on shape. Scikit-learn docs for KMeans/DBSCAN. Scikit-learn +1

B. Clustering recipe

- Choose features: `recency_days`, `frequency_week`, `revenue_90d`, `num_support_tickets`, `avg_session_duration`, `failed_payments`, `feature_counts`.
- Preprocess: `log1p` + `StandardScaler`, maybe dimensionality reduction (PCA / UMAP) → cluster on reduced space.
- Try **KMeans** for clear, balanced clusters (pick k by silhouette/elbow). Try **DBSCAN** if you expect dense clusters + noise/outliers (DBSCAN finds arbitrary-shaped clusters and marks noise). Scikit-learn +1

C. Label clusters into human reasons

- For each cluster compute medians of the key features and create rule-of-thumb names:
  - **Silent churners**: low support tickets, low usage, declining trend → “lost interest / low engagement”. (See Mixpanel’s silent churn discussion.) Mixpanel
  - **Value-seekers / price-sensitive**: low revenue, trial or low-plan, canceled after coupon → “price sensitivity”.
  - **Frustrated users**: high `support_tickets`, many `error_events`, short sessions → “friction/UX issues”.
  - **Payment-friction**: high `failed_payments` and churn immediately after billing cycle → “payment friction”.

D. Validate clusters

- Quantitative: compare retention curves, LTV, reactivation rates by cluster.
- Qualitative: sample users from each cluster and read support tickets/session replays to confirm reason.
- A/B test readiness: ensure each cluster has enough size for experiments.

E. Analysis outputs to produce

- Cluster profiles (median stats + top contributing features via SHAP).
- Actionable playbooks per cluster (e.g., onboarding tour for silent churners; price experiments for value-seekers; proactive support + bug fixes for frustrated users).
- Expected impact calculation: estimate revenue at risk per segment and intervention cost.

---

6) Concrete SQL / Python snippets (quick, runnable patterns)

Label generation SQL (snapshot t0):

```sql
-- users seen before t0
WITH seen AS (
  SELECT DISTINCT user_id FROM events WHERE event_time <= '2025-11-01'
),
-- users active after t0 within label window (30 days)
active_after AS (
  SELECT DISTINCT user_id FROM events WHERE event_time > '2025-11-01' AND
event_time <= '2025-12-01'
)
SELECT s.user_id,
       CASE WHEN a.user_id IS NULL THEN 1 ELSE 0 END AS churn_label
FROM seen s
LEFT JOIN active_after a USING (user_id);
```

Python (LightGBM train skeleton):

```python
import lightgbm as lgb
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import roc_auc_score

# X_train, y_train, X_val, y_val prepared with time split
train_data = lgb.Dataset(X_train, label=y_train)
val_data = lgb.Dataset(X_val, label=y_val)

params = {'objective':'binary', 'metric':'auc', 'learning_rate':0.05,
'num_leaves':64}
model = lgb.train(params, train_data, valid_sets=[val_data],
early_stopping_rounds=50)
preds = model.predict(X_test)
print("AUC:", roc_auc_score(y_test, preds))
```

Python (clustering churners with KMeans + UMAP):

```python
import umap, numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

X = churners[cluster_features].fillna(0)
X = np.log1p(X)
X = StandardScaler().fit_transform(X)
X_red = umap.UMAP(n_components=5, random_state=42).fit_transform(X)
kmeans = KMeans(n_clusters=4, random_state=42).fit(X_red)
churners['segment'] = kmeans.labels_
```

---

7) Monitoring, experiments & operationalization (short checklist)

- **Score nightly** and write `churn_prob` to a table with model version + snapshot date.
- Monitor feature drift (KL divergence) and model AUC weekly.
- For interventions, use randomized controlled trials per segment (treatment vs holdout) and compute lift on retention; use sample size calculator from baseline churn and expected uplift.
- Retrain monthly or on drift trigger.

---

8) Key references (read next)

- Kaggle — SaaS Subscription & Churn Analytics (synthetic dataset for prototyping). Kaggle
- RFM definitions & best practices (Investopedia / TechTarget articles). Investopedia +1
- Mixpanel on **silent churn** — product analytics perspective. Mixpanel
- LightGBM docs / quick start — use for model implementation & parameters. LightGBM
- scikit-learn clustering docs (KMeans, DBSCAN) — clustering algorithms and best practices. Scikit-learn

---

Quick next actions I can do right now (pick one)

1. **Scaffold a runnable repo** (ETL + feature pipeline + model + clustering) using pandas + scikit-learn + LightGBM and a synthetic Kaggle SaaS dataset.
2. **Generate a synthetic dataset** shaped like your product logs and run the full pipeline end-to-end (train + cluster + SHAP + sample segment profiles).
3. **Write a SQL feature library** adapted to BigQuery / Snowflake style for your data warehouse.
4. **Create experiment & sample size planner** for test interventions per segment.

Tell me which and I'll produce it immediately (I'll generate code / SQL / outputs in this response).

Sources