Student Success Personas via **Clustering + Matrix Factorization**

Introduction

We want early, **non-stigmatizing** insight into student patterns without relying on outcome labels.
Using only **unsupervised methods we studied**—**Clustering**, **Recommender-style similarity**, and **Matrix Factorization (SVD/NMF)**—we'll discover **student personas** and describe them in plain terms (e.g., *earlier-progress / lower-GPA* vs *further-along / higher-GPA*).
Any label like `Dropout_Risk` is used **only after** we form personas, for context checks —not for training.

Problem to Solve

Goal: Find a small number of **interpretable personas** based on academics, engagement, and behavior, using **factor models (SVD/NMF)** to uncover latent dimensions and **K-Means** to cluster students in that latent space. Summarize how personas differ and provide a simple **"students like me"** similarity view.

Why these techniques (course-aligned):

- **NMF**: non-negative factors → parts-based, human-readable components (e.g., "high-attendance + steady hours").
- **SVD**: classic low-rank structure discovery; good sanity check vs. NMF.
- **Clustering**: turns latent factor scores into compact personas.
- **Nearest-Neighbor similarity**: a recommender-style lens for "similar students."

Data Summary (what we use)

- ~1,000 students × ~23 columns (no missing; one extreme label imbalance on `Dropout_Risk` used only for context).
- Exclude: `Student_ID` (identifier) and **protected attributes** (e.g., Gender, Ethnicity, SES, Disability, Parental Education) from model training; keep them only for fairness/context tables.

Code Outline (Notebook Sections)

0) Setup

- **Imports & seed**
- **Load CSV** (absolute or relative path)
- **Define roles**: `ID`, `TARGET` (e.g., `Dropout_Risk`), `PROTECTED` (audit-only)

1) EDA (right-sized)

- **Schema & quality**: shape, dtypes, duplicates, constant columns, range checks (Age, GPA, Attendance)
- **Distributions**: numeric histograms; small-cardinality categorical bars
- **Relationships**: a few bivariate plots (e.g., GPA vs Attendance, GPA vs Credits)
- **Context only**: `Dropout_Risk` counts (note 1.8% positives), protected attribute balances

2) Preprocessing (for SVD/NMF + Clustering)

- Build **feature matrix X** for modeling:
  - Drop `ID`, `TARGET`, `PROTECTED`
  - **One-Hot encode** categoricals (e.g., Past_Academic_Performance levels)
  - **Scale numerics to [0,1]** (MinMaxScaler) so **NMF** is valid (non-negative) and features are comparable
- Keep a **ColumnTransformer** pipeline so transforms are reproducible

3) Matrix Factorization (NMF first; SVD optional)

- **NMF (rank r = 6–8)** on preprocessed `X` → `W` (students×r) and `H` (r×features)
  - Inspect **top features per factor** (rows of `H`) to name factors
- *(Optional comparison)* **TruncatedSVD** (rank r = 6–8) to see if factors are similar

4) Clustering in Factor Space

- **K-Means** on `W` (NMF scores), sweep **k=2...10**
- Select **k** by **silhouette**; keep `best_labels`
- **Visuals**:
  - 2D scatter of two NMF components (W[:,0] vs W[:,1]) colored by cluster
  - *(Optional)* 3D scatter of three NMF components (rotatable if you enabled ipympl/Plotly)

5) Persona Profiles (Interpretation)

- For each cluster:
  - Show **means/medians** of key original features (GPA, Credits, Courses_Failed, Attendance, Hours, etc.)

    - Use **H** to list **top factor loadings** that characterize the persona
- Write a **one-line name** per persona (e.g., “Further-along, higher GPA & credits”)

6) Recommender-Style Similarity (Mini)

- Compute **cosine similarity** in `W` space and, for 2–3 example students, list the **top-5 most similar** (IDs only or anonymized indices)
- (Optional) Average the features of these similar students → a quick “students like me” profile

7) Post-hoc Context (still unsupervised)

- **Cluster × Dropout_Risk** contingency + **rates** (with Wilson 95% CI because positives are rare)
- Keep language careful: personas are **descriptive**, not risk predictions

8) (Optional) One Fairness Check

- **Cramér’s V** between clusters and one protected attribute (e.g., Gender) to see if there’s strong association

9) Conclusions & Limits

- Summarize the **personas** and the **factor interpretations**
- Note **low/medium silhouettes** if clusters overlap (soft segments)
- Ethical use: exclude protected attributes from training; avoid stigmatizing labels; labels not used for fitting

---

Minimal Library Footprint

- `pandas`, `numpy`, `matplotlib`
- `scikit-learn` (OneHotEncoder, MinMaxScaler, ColumnTransformer, NMF, TruncatedSVD, KMeans, metrics)

Tiny Pseudocode Snippet (where the math happens)

```
X_raw  = df[features_no_ID_target_protected]
X_proc = OneHotEncode(cats) + MinMaxScale(nums)     # all ≥ 0 for
NMF
W, H   = NMF(rank=r).fit_transform(X_proc), model.components_
labels = KMeans(k=argmax_silhouette).fit_predict(W)

# Profiles
```

```
for cluster c:
  show mean/median of key original features for students in c
  show top features in H that define high values of factor(s)
dominant in c

# Context (post-hoc only)
table = crosstab(labels, Dropout_Risk); add cluster-wise rates +
CIs
```

In [ ]:

```
```

Setup 1/4 — Environment & imports

Purpose: Initialize a clean, reproducible environment for *unsupervised learning with clustering + matrix factorization (NMF/SVD)*.
We import core libraries, set a random seed for repeatability, choose readable plotting defaults, and print key package versions so your notebook is easy to reproduce and grade.

In [24]:

```python
# === Unsupervised Learning Project — Imports & Setup ===
# Core
import os, sys, warnings
from pathlib import Path
import numpy as np
import pandas as pd
from pandas.api.types import is_numeric_dtype

# Plotting
import matplotlib.pyplot as plt


# Stats helpers
from scipy import stats
from math import sqrt
# Correlation functions needed by scatter_with_fit
from scipy.stats import pearsonr, spearmanr


# Scikit-learn: preprocessing & pipelines
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import NMF
from sklearn.metrics.pairwise import cosine_similarity


# Matrix factorization (course-aligned)
from sklearn.decomposition import NMF, TruncatedSVD

# Clustering & neighbors
from sklearn.cluster import KMeans
from sklearn.neighbors import NearestNeighbors
from scipy.cluster.vq import kmeans2

# Metrics for model selection / diagnostics
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski

# Reproducibility & plot defaults
RNG_SEED = 42
np.random.seed(RNG_SEED)
warnings.filterwarnings("ignore")

plt.rcParams.update({
    "figure.figsize": (7, 4),
    "axes.grid": True,
    "grid.alpha": 0.3,
    "axes.titlesize": 12,
    "axes.labelsize": 11,
})

def _versions():
    import matplotlib, sklearn, scipy
    return {
        "python":  sys.version.split()[0],
        "numpy":   np.__version__,
        "pandas":  pd.__version__,
        "scikit_learn": sklearn.__version__,
        "scipy":   scipy.__version__,
        "matplotlib": matplotlib.__version__,
    }

print("[versions]", _versions())
```

```
[versions] {'python': '3.11.4', 'numpy': '1.24.4', 'pandas': '2.3.1', 'sciki
t_learn': '1.2.1', 'scipy': '1.10.1', 'matplotlib': '3.7.1'}
```

Setup 2/4 — Data path & safe loading

Purpose: Load the CSV robustly from either your absolute path (local) or a repo-friendly relative path.
We immediately echo shape and a few rows to confirm we opened the **right file** with the **expected schema**.

In [7]:
```python
# Prefer a repo path if you later add /data to GitHub
CANDIDATE_PATHS = [
    Path("/Users/cynthiamcginnis/Downloads/student_management_dataset.csv"),
    Path("data/student_management_dataset.csv"),
]

DATA_PATH = next((p for p in CANDIDATE_PATHS if p.exists()), None)
assert DATA_PATH is not None, f"File not found. Checked: {CANDIDATE_PATHS}"

df = pd.read_csv(DATA_PATH)
print(f"[load] path={DATA_PATH} | shape={df.shape}")
display(df.head(5))
df.info()
```

```
[load] path=/Users/cynthiamcginnis/Downloads/student_management_dataset.csv
| shape=(1000, 23)
```

	Student_ID	Age	Gender	Ethnicity	Socioeconomic_Status	Parental_Education_Leve
0	1	24	Female	Hispanic	Medium	Master's
1	2	21	Female	Asian	Low	High Schoo
2	3	28	Male	Black	Medium	Bachelor's
3	4	25	Female	Black	Medium	Bachelor's
4	5	22	Male	Other	Medium	Bachelor's

5 rows × 23 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 23 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Student_ID                    1000 non-null   int64  
 1   Age                           1000 non-null   int64  
 2   Gender                        1000 non-null   object 
 3   Ethnicity                     1000 non-null   object 
 4   Socioeconomic_Status          1000 non-null   object 
 5   Parental_Education_Level      1000 non-null   object 
 6   Disability_Status             1000 non-null   int64  
 7   GPA                           1000 non-null   float64
 8   Past_Academic_Performance     1000 non-null   object 
 9   Current_Semester_Performance  1000 non-null   int64  
 10  Courses_Failed                1000 non-null   int64  
 11  Credits_Completed             1000 non-null   int64  
 12  Study_Hours_per_Week          1000 non-null   int64  
 13  Attendance_Rate               1000 non-null   float64
 14  Number_of_Late_Submissions    1000 non-null   int64  
 15  Class_Participation_Score     1000 non-null   float64
 16  Online_Learning_Hours         1000 non-null   float64
 17  Library_Usage_Hours           1000 non-null   float64
 18  Disciplinary_Actions          1000 non-null   int64  
 19  Social_Engagement_Score       1000 non-null   float64
 20  Mental_Health_Score           1000 non-null   float64
 21  Extracurricular_Activities    1000 non-null   int64  
 22  Dropout_Risk                  1000 non-null   int64  
dtypes: float64(7), int64(11), object(5)
memory usage: 179.8+ KB
```

Setup 3/4 — Declare column roles (ID, target, protected) and feature candidates

Purpose: Clearly separate columns by role before any modeling:

- **ID** → never model on it.
- **TARGET** (e.g., `Dropout_Risk`) → **context only** (post-hoc), not used for training.

- **PROTECTED** (e.g., Gender, Ethnicity, SES, Disability, Parental Education) → excluded from training; used only for fairness/context audits.

We then derive **feature candidates** = all remaining columns.

In [9]:
```
# Roles for this dataset
ID_COLS         = ["Student_ID"]
TARGET_COLS     = ["Dropout_Risk"]  # unsupervised: held out for context only
PROTECTED_COLS = ["Gender","Ethnicity","Socioeconomic_Status",
                  "Parental_Education_Level","Disability_Status"]

present_ids         = [c for c in ID_COLS if c in df.columns]
present_targets     = [c for c in TARGET_COLS if c in df.columns]
present_protected   = [c for c in PROTECTED_COLS if c in df.columns]

# Feature candidates for modeling/EDA (excluding id + target)
feature_candidates = [c for c in df.columns if c not in set(present_ids + pr

print("[roles]")
print("  ID       :", present_ids)
print("  TARGET   :", present_targets, "(held out; context only)")
print("  PROTECTED:", present_protected, "(audit only; excluded from trainin

print(f"[feature candidates] count={len(feature_candidates)}")
```

```
[roles]
  ID       : ['Student_ID']
  TARGET   : ['Dropout_Risk'] (held out; context only)
  PROTECTED: ['Gender', 'Ethnicity', 'Socioeconomic_Status', 'Parental_Educa
tion_Level', 'Disability_Status'] (audit only; excluded from training)
[feature candidates] count=21
```

Setup 4/4 — Quick data quality tripwire

Purpose: Catch simple-but-costly issues *before* analysis:

- duplicate rows
- constant columns
- obvious range violations on key numerics (tweak bounds if your context differs)

In [14]:
```
# Duplicates & constants
dup_count = df.duplicated().sum()
n_unique = df.nunique()
constant_cols = n_unique[n_unique <= 1].index.tolist()

# Lightweight range checks (adjust as needed for your institution)
RANGE_RULES = {
    "Age": (15, 65),
    "GPA": (0.0, 4.0),
    "Attendance_Rate": (0.0, 100.0),
}

range_report = {}
```

```
for col, (lo, hi) in RANGE_RULES.items():
    if col in df.columns:
        range_report[col] = int((~df[col].between(lo, hi)).sum())

print(f"[quality] duplicates={dup_count} | constant_cols={constant_cols or '
print("[ranges]", range_report)
```

```
[quality] duplicates=0 | constant_cols=None
[ranges] {'Age': 0, 'GPA': 0, 'Attendance_Rate': 0}
```

EDA Overview: Distributions, Correlations, and Bivariate Plots

Why this section exists

Before we do any clustering or factorization, we need to **understand the data's shape**. This EDA section answers:

- *What do individual features look like?* (skew, spread, outliers)
- *Which features move together?* (redundancy and latent structure)
- *How do key academic outcomes relate to effort/engagement signals?* (practical intuition for later personas)

---

1) Distributions (Univariate)

Purpose. Examine each feature on its own—center, spread, skew, heavy tails, and rare values.

What you'll see.

- **Histograms** for numeric features (e.g., GPA, Credits_Completed, Attendance_Rate, hours measures).
- **Bar charts** for small-cardinality categoricals (e.g., Past_Academic_Performance levels).

How to use it.

- Flag skewed/long-tailed variables (e.g., hours, counts) → consider **robust scaling** or note potential winsorization.
- Spot rare categories → consider **grouping long tails** before one-hot encoding.
- Confirm no obvious data-entry issues (e.g., GPA outside [0,4]).

---

2) Correlations (Numeric–Numeric)

Purpose. Identify **redundant** or **highly related** numeric features that could dominate distances or dilute clustering.

What you'll see.

- **Correlation heatmap** (Pearson; Spearman optional).
- A short list of **top absolute correlations**.

How to use it.

- If two features are ~collinear (|r| ≥ 0.9), consider **dropping one** or be mindful that they form a single axis in latent space.
- If correlations are broadly modest, expect variance to be **spread** across many dimensions—factorization (e.g., NMF) helps compress.

---

3) Bivariate Plots (Numeric–Numeric pairs)

Purpose. Build intuition for **practical relationships** you'll discuss in the report.

What you'll see.

- Compact **scatterplots** for a few meaningful pairs, e.g.:
  - GPA vs. Attendance_Rate
  - GPA vs. Credits_Completed
  - GPA vs. Study_Hours_per_Week

How to use it.

- Look for **directional trends** (even noisy ones): "more attendance ↔ slightly higher GPA" supports later persona narratives.
- Check for **nonlinear patterns** or clusters-in-the-wild (usually subtle here).
- Note **heteroskedasticity/outliers** that could affect distance-based clustering.

Outputs you will capture

- Distribution figures (histograms/bars) suitable for the appendix.
- Correlation heatmap + top-pairs table for the methods section.
- 2–3 bivariate plots you can actually **refer to in your discussion** (keep it lean).

Bottom line: This EDA gives you a defensible basis for preprocessing choices (scaling/encoding), motivates **matrix factorization** (variance is distributed), and provides language for interpreting the **personas** you'll derive later.

EDA — Distributions (Univariate)

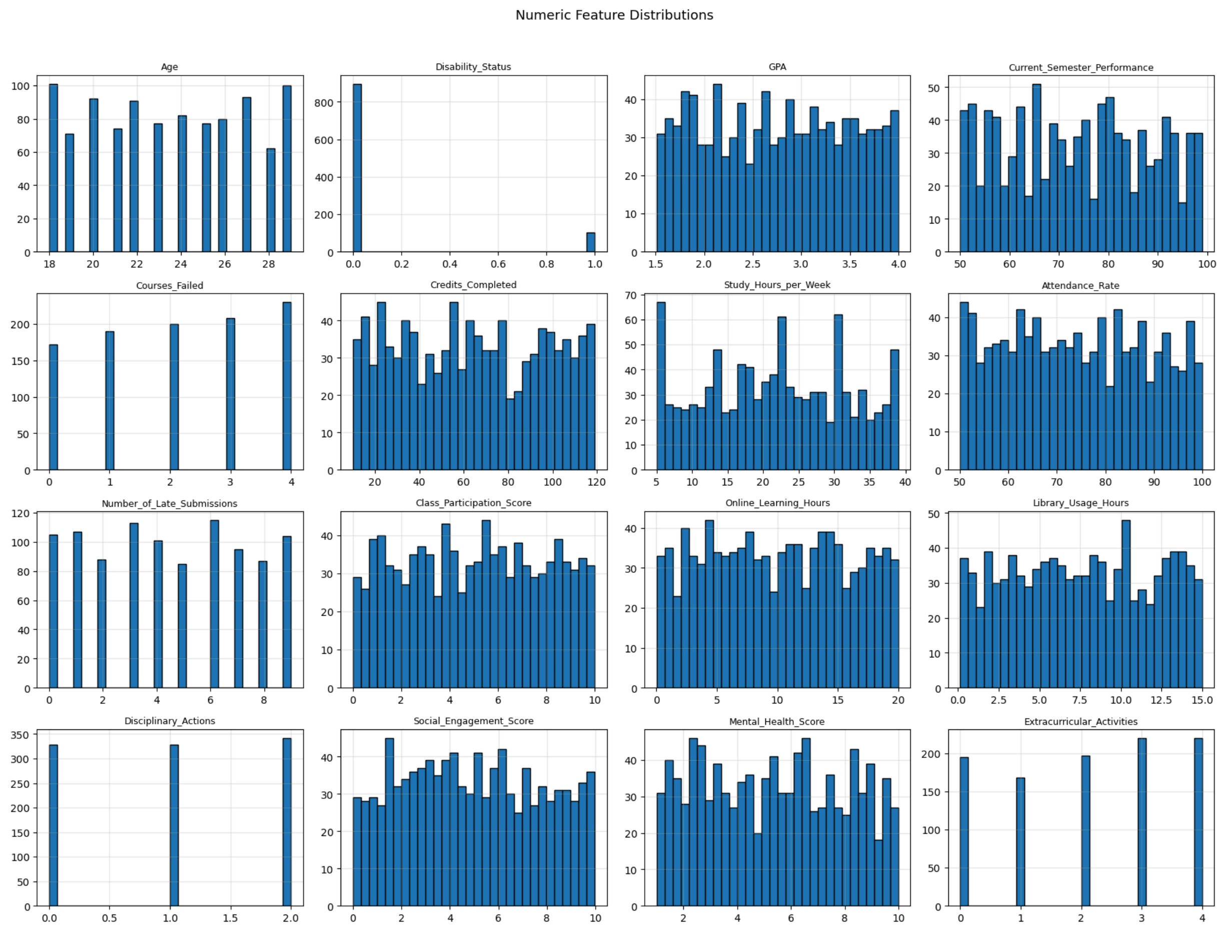
Purpose: Quickly see the shape of each feature:

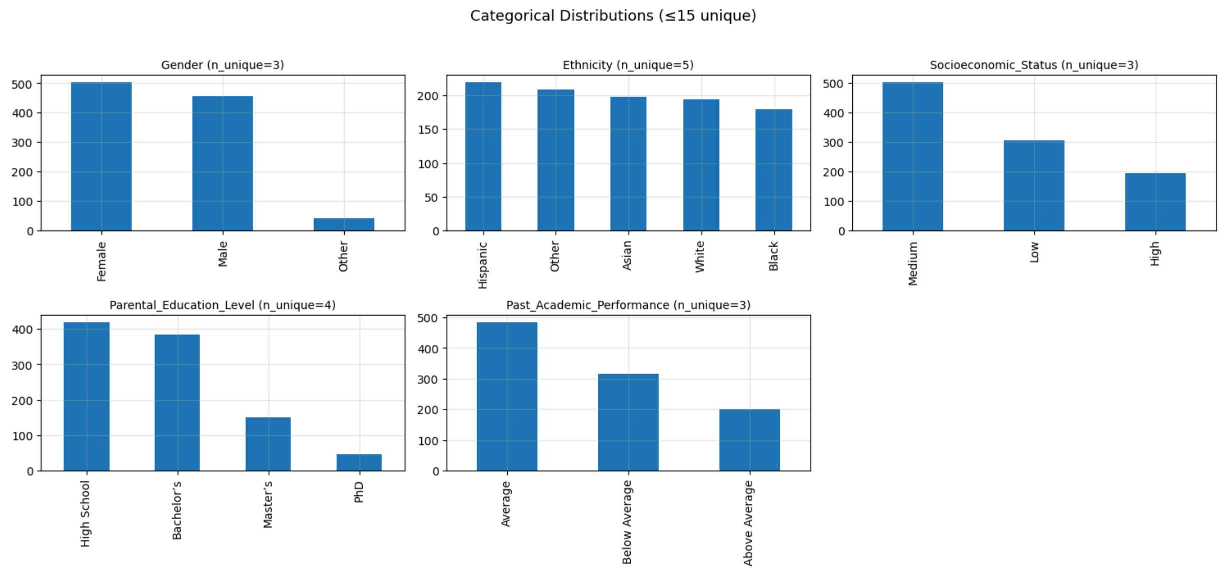
- **Numeric:** histograms (center, spread, skew, tails)

- **Categorical:** bar charts for low-cardinality columns

Notes: We exclude `Student_ID` and the label `Dropout_Risk` (context-only). Figures are sized to fit many features and won't error if a group is empty.

```
In [18]: # ---------- CONFIG ----------
EXCLUDE = {"Student_ID", "Dropout_Risk"}  # ID + context-only target
MAX_NUM_PLOTS = None    # set like 16 to limit; None = show all numerics
MAX_CAT_PLOTS = 12      # limit number of categorical bar charts shown
CAT_CARD_LIMIT = 15     # only plot categoricals with <= this many unique va
BINS = 30

# ---------- PREP ----------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.api.types import is_numeric_dtype

eda_cols = [c for c in df.columns if c not in EXCLUDE]
num_cols = [c for c in eda_cols if is_numeric_dtype(df[c])]
cat_cols = [c for c in eda_cols if c not in num_cols]

# ---------- NUMERIC HISTOGRAMS ----------
to_plot = num_cols if MAX_NUM_PLOTS is None else num_cols[:MAX_NUM_PLOTS]

if to_plot:
    n = len(to_plot)
    cols = 4
    rows = int(np.ceil(n / cols))
    plt.figure(figsize=(4.2*cols, 3.1*rows))
    for i, c in enumerate(to_plot, 1):
        ax = plt.subplot(rows, cols, i)
        df[c].hist(bins=BINS, edgecolor="black", ax=ax)
        ax.set_title(c, fontsize=9)
        ax.grid(True, alpha=0.3)
    plt.suptitle("Numeric Feature Distributions", y=1.02, fontsize=13)
    plt.tight_layout()
    plt.show()
else:
    print("No numeric columns to plot.")

# ---------- CATEGORICAL BARS ----------
small_cats = [c for c in cat_cols if df[c].nunique() <= CAT_CARD_LIMIT]
if small_cats:
    if MAX_CAT_PLOTS is not None:
        small_cats = small_cats[:MAX_CAT_PLOTS]
    n = len(small_cats)
    cols = 3
    rows = int(np.ceil(n / cols))
    plt.figure(figsize=(5.0*cols, 3.4*rows))
    for i, c in enumerate(small_cats, 1):
        ax = plt.subplot(rows, cols, i)
        df[c].value_counts(dropna=False).plot(kind="bar", ax=ax)
        ax.set_title(f"{c} (n_unique={df[c].nunique()})", fontsize=10)
        ax.set_xlabel("")
        ax.grid(True, axis="y", alpha=0.3)
    plt.suptitle(f"Categorical Distributions (≤{CAT_CARD_LIMIT} unique)", y=
    plt.tight_layout()
    plt.show()
else:
    print("No low-cardinality categorical columns to plot.")
```





Overall, the numeric features look bounded and fairly even across their ranges, with only mild skew—so no heavy tails or glaring outliers. Academic/effort variables like GPA (aprx 1.5–4.0), Credits_Completed (20–120), Attendance_Rate (~50–100), Study/Online/Library hours appear roughly uniform or gently peaked, suggesting we won't need aggressive transformations (MinMax scaling for NMF is appropriate). Several variables are discrete counts—e.g., Courses_Failed (0–4), Late_Submissions (0–9), Disciplinary_Actions (0–10), Extracurricular_Activities (integer steps)—which will one-hot cleanly if treated as categorical buckets or can stay numeric for distance methods (but remember their discreteness). Attitudinal/engagement scores (Class_Participation, Social_Engagement, Mental_Health) look approximately symmetric around mid-range with moderate spread. Taken together, the distributions don't reveal dominant single-feature separations; any structure will likely come from multi-feature combinations, which fits our plan to use matrix factorization (NMF/SVD) + clustering rather than relying on a few highly skewed drivers.

EDA — Relationships (Bivariate)

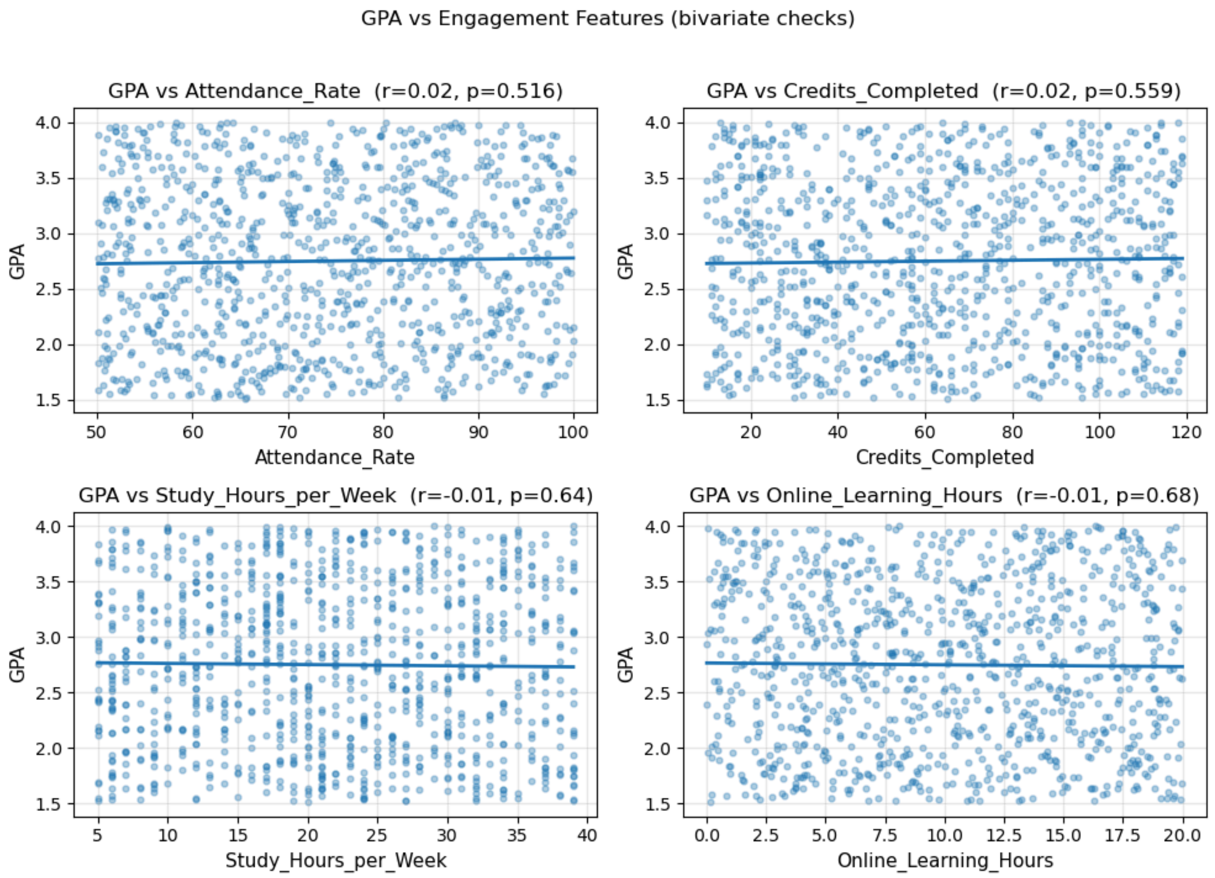
Purpose: Check simple relationships between key academic outcome **GPA** and effort/engagement features.
We'll plot compact scatterplots with a least-squares trend line and annotate Pearson's *r* to gauge direction/strength.

Pairs shown:

- GPA vs **Attendance_Rate**
- GPA vs **Credits_Completed**
- GPA vs **Study_Hours_per_Week**
- GPA vs **Online_Learning_Hours**

Read me: Positive slope / *r* > 0 suggests a direct relationship; values near 0 indicate weak linear association.

In [26]:
```
pairs = [
    ("Attendance_Rate","GPA"),
    ("Credits_Completed","GPA"),
    ("Study_Hours_per_Week","GPA"),
    ("Online_Learning_Hours","GPA"),
]

def scatter_with_fit(ax, x, y):
    xv = df[x].to_numpy(float); yv = df[y].to_numpy(float)
    ax.scatter(xv, yv, s=12, alpha=0.35)
    try:
        m,b = np.polyfit(xv, yv, 1); xx = np.linspace(xv.min(), xv.max(), 10
        ax.plot(xx, m*xx+b, linewidth=2)
    except Exception:
        pass
    r,p = pearsonr(xv, yv)
    ax.set_title(f"{y} vs {x}  (r={r:.02f}, p={p:.3g})")
    ax.set_xlabel(x); ax.set_ylabel(y)

# 2×2 panel
fig, axes = plt.subplots(2, 2, figsize=(10, 7))
for ax,(x,y) in zip(axes.ravel(), pairs):
    scatter_with_fit(ax, x, y)
plt.suptitle("GPA vs Engagement Features (bivariate checks)", y=1.02)
plt.tight_layout(); plt.show()

# Summary table
rows=[]
for x,y in pairs:
    pr,pp = pearsonr(df[x], df[y])
    sr,sp = spearmanr(df[x], df[y])
    rows.append({"feature":x, "pearson_r":pr, "pearson_p":pp, "spearman_rho"
summary = pd.DataFrame(rows).round(3)
summary
```



Out[26]:

	feature	pearson_r	pearson_p	spearman_rho	spearman_p
0	Attendance_Rate	0.021	0.516	0.020	0.522
1	Credits_Completed	0.018	0.559	0.019	0.559
2	Study_Hours_per_Week	-0.015	0.640	-0.018	0.573
3	Online_Learning_Hours	-0.013	0.680	-0.012	0.694

EDA — Context Only: `Dropout_Risk` and Protected Attributes

Purpose: Report label prevalence and group balances **without** using them for training or tuning.
`Dropout_Risk` is highly imbalanced (~1.8% positives), so any post-hoc comparisons will be noisy.
Protected attributes are shown only for **composition context** and later fairness notes.

In [29]:
```
# ----- Dropout_Risk prevalence  -----

def wilson_interval(pos, n, z=1.96):
    if n == 0:
        return (0.0, 0.0, 0.0)
    p = pos / n
    denom = 1 + z**2/n
    center = (p + z**2/(2*n)) / denom
    half = z * sqrt((p*(1-p)/n) + z**2/(4*n**2)) / denom
    return p, max(0.0, center-half), min(1.0, center+half)

if "Dropout_Risk" in df.columns:
    counts = df["Dropout_Risk"].value_counts().sort_index()
    n = int(counts.sum())
    pos = int(counts.get(1, 0))
    p, lo, hi = wilson_interval(pos, n)
    display(pd.DataFrame({
        "value":[0,1],
        "count":[int(counts.get(0,0)), pos],
        "rate_pct":[(1-p)*100, p*100]
    }))
    print(f"Overall positive rate: {p*100:.2f}% (Wilson 95% CI: {lo*100:.2f}
else:
    print("Column 'Dropout_Risk' not found.")
```

	value	count	rate_pct
0	0	982	98.2
1	1	18	1.8

```
Overall positive rate: 1.80% (Wilson 95% CI: 1.14%–2.83%)
```

`Dropout_Risk`:
`Dropout_Risk` is extremely imbalanced (≈1.8% positives overall; Wilson 95% CI reported). We therefore treat it strictly as *context* for post-hoc tables and avoid using it for model fitting or selection. Protected attributes (Gender, Ethnicity, SES, Parental Education, Disability) are summarized to document cohort composition only; they are **excluded from training** and referenced later for fairness context (e.g., cluster composition parity), not as modeling features.

2) Preprocessing (for SVD/NMF + Clustering)

Purpose: Build a **reproducible feature matrix `X_proc`** with:

- **No leakage**: drop `ID`, `TARGET`, and `PROTECTED` columns from training.
- **Consistent encoding**: **One-Hot** for categoricals (handles unseen levels safely).
- **Non-negative scaling**: **MinMaxScaler** maps numerics to **[0,1]**, which NMF requires and makes features comparable.
- **Single `ColumnTransformer`** pipeline so the exact same transforms can be refit on new data and used downstream (NMF, K-Means, plots).

Notes

- If `Past_Academic_Performance` is truly **ordinal** (e.g., *Poor < Average < Good < Excellent*), we encode it once with that order (optional block below).
- We also return a **feature name list** for interpretability (e.g., when inspecting NMF components).

In [33]:
```
# --- 2.0 Roles (reuse from earlier cell if present) ---
ID_COLS        = ["Student_ID"]
TARGET_COLS    = ["Dropout_Risk"]  # context only
PROTECTED_COLS = ["Gender","Ethnicity","Socioeconomic_Status",
                  "Parental_Education_Level","Disability_Status"]

# --- 2.1 Define training columns (no ID, no target, no protected) ---
train_cols = [c for c in df.columns if c not in set(ID_COLS + TARGET_COLS +
X = df[train_cols].copy()

# --- 2.2 Optional: ordinal mapping for Past_Academic_Performance ---
# Uncomment & adjust if your levels match this order exactly.
# if "Past_Academic_Performance" in X.columns:
#     order = ["Poor","Average","Good","Excellent"]
#     if set(X["Past_Academic_Performance"].dropna().unique()).issubset(orde
#         X["Past_Academic_Performance"] = pd.Categorical(
#             X["Past_Academic_Performance"], categories=order, ordered=True
#         )

# --- 2.3 Split dtypes -> numeric vs categorical ---
num_cols = [c for c in X.columns if is_numeric_dtype(X[c])]
cat_cols = [c for c in X.columns if c not in num_cols]

# --- 2.4 Encoders/scalers ---
# OneHotEncoder API: sparse_output for sklearn >=1.2, else sparse
try:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
except TypeError:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

scaler = MinMaxScaler(feature_range=(0, 1))  # ensures nonnegative features

# --- 2.5 ColumnTransformer pipeline (reproducible) ---
```

```
prep = ColumnTransformer(
    transformers=[
        ("num", scaler, num_cols),
        ("cat", ohe,    cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=True,
)

# Fit-transform to get the processed matrix for modeling
X_proc = prep.fit_transform(X)

# --- 2.6 Introspection helpers ---
feat_names = prep.get_feature_names_out()  # names after scaling / one-hot
print(f"[prep] X -> X_proc shape: {X.shape} -> {X_proc.shape}")
print(f"[prep] numeric={len(num_cols)} | categorical={len(cat_cols)} | featu

# Sanity checks
assert np.all(np.isfinite(X_proc)), "Non-finite values found after preproces
assert X_proc.min() >= 0.0 - 1e-9, "Pipeline must produce non-negative featu

# Optional: small preview as DataFrame (first 5 rows)
X_proc_preview = pd.DataFrame(X_proc[:5], columns=feat_names)
display(X_proc_preview)
```

```
[prep] X -> X_proc shape: (1000, 16) -> (1000, 18)
[prep] numeric=15 | categorical=1 | features_out=18
```

	num__Age	num__GPA	num__Current_Semester_Performance	num__Courses_Failed
0	0.545455	0.702811	0.816327	1.00
1	0.272727	0.212851	0.571429	0.50
2	0.909091	0.365462	0.938776	0.75
3	0.636364	0.389558	0.040816	0.25
4	0.363636	0.975904	0.000000	1.00

Preprocessing summary

- **X → X_proc shape:** `(1000, 16)` → `(1000, 18)`
  - Data **15 numeric** + **1 categorical**.
  - The categorical **`Past_Academic_Performance`** expanded to **3 one-hot columns**, so **15 + 3 = 18** features in `X_proc`.
- **Value range:** All features are scaled to **[0, 1]** via **MinMax**, which satisfies **NMF's non-negativity** requirement.

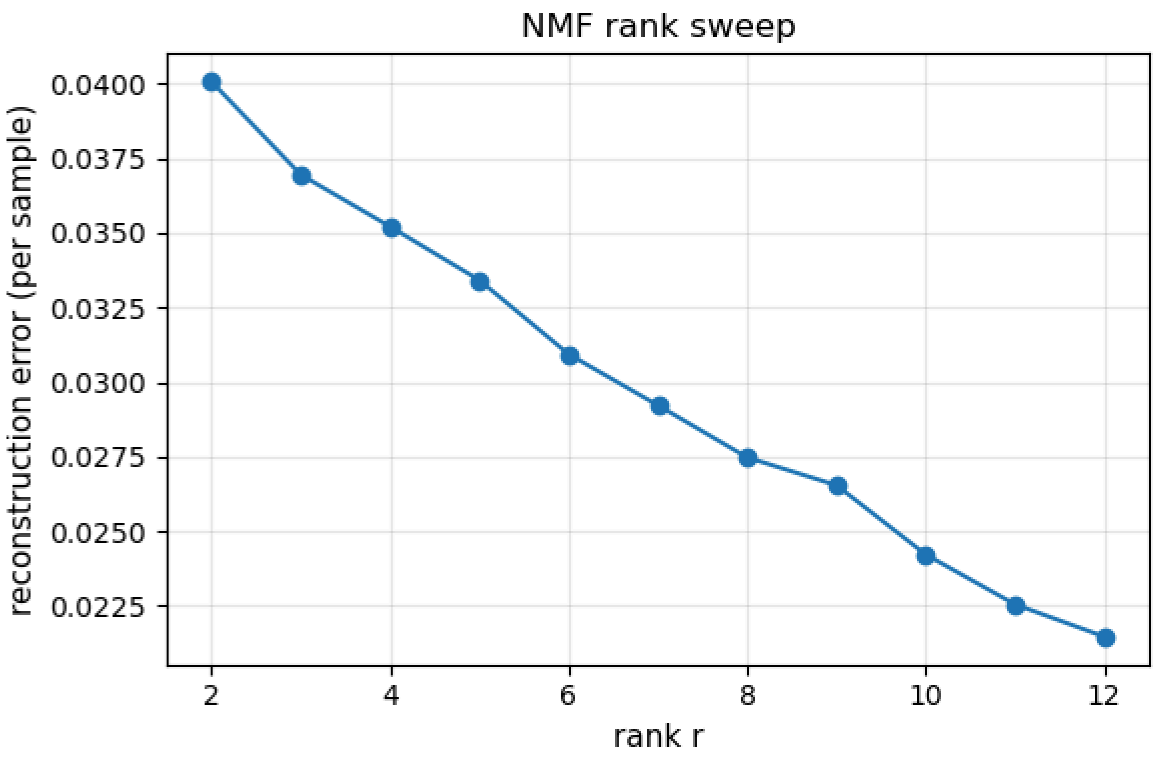
- **One-hot levels:** `Below Average`, `Average`, `Above Average` (interpretable, ordered-style levels).

3) Matrix Factorization (NMF): choose rank

Purpose: Find a small number of latent factors `r` that reconstruct the data well **without overfitting**.
We sweep `r = 2…12`, record reconstruction error, and pick a simple elbow (often **6–8** for this dataset).

In [37]:
```
Xnn = X_proc  # already non-negative from MinMax+OHE

ranks = range(2, 13)
errs, models = [], {}
for r in ranks:
    nmf = NMF(n_components=r, init="nndsvda", random_state=42, max_iter=800,
    W = nmf.fit_transform(Xnn)
    H = nmf.components_
    # Use built-in reconstruction_err_ (Frobenius) normalized by n_samples f
    errs.append(nmf.reconstruction_err_ / Xnn.shape[0])
    models[r] = (nmf, W, H)

best_r = int(ranks[int(np.argmin(errs))])
print(f"[NMF] best_r by min error = {best_r}")

plt.figure(figsize=(6,4))
plt.plot(list(ranks), errs, marker="o")
plt.xlabel("rank r"); plt.ylabel("reconstruction error (per sample)")
plt.title("NMF rank sweep")
plt.grid(True, alpha=.3); plt.tight_layout(); plt.show()
```

```
[NMF] best_r by min error = 12
```



3.1 Inspect factors: top-loading features per factor

Purpose: Make the factors human-readable.
We list the **top 8 features** for each factor (largest weights in `H`) so you can name them (e.g., "high attendance + participation").

In [39]:
```
nmf, W, H = models[best_r]
feat_names = np.array(feat_names)  # from preprocessing cell

def top_features_per_factor(H, feature_names, topn=8):
    rows = []
    for k, row in enumerate(H):
        idx = np.argsort(row)[::-1][:topn]
        rows.append({
            "factor": k,
            "top_features": ", ".join([f"{feature_names[i]}" for i in idx]),
        })
    return pd.DataFrame(rows)

topf = top_features_per_factor(H, feat_names, topn=8)
display(topf)
```

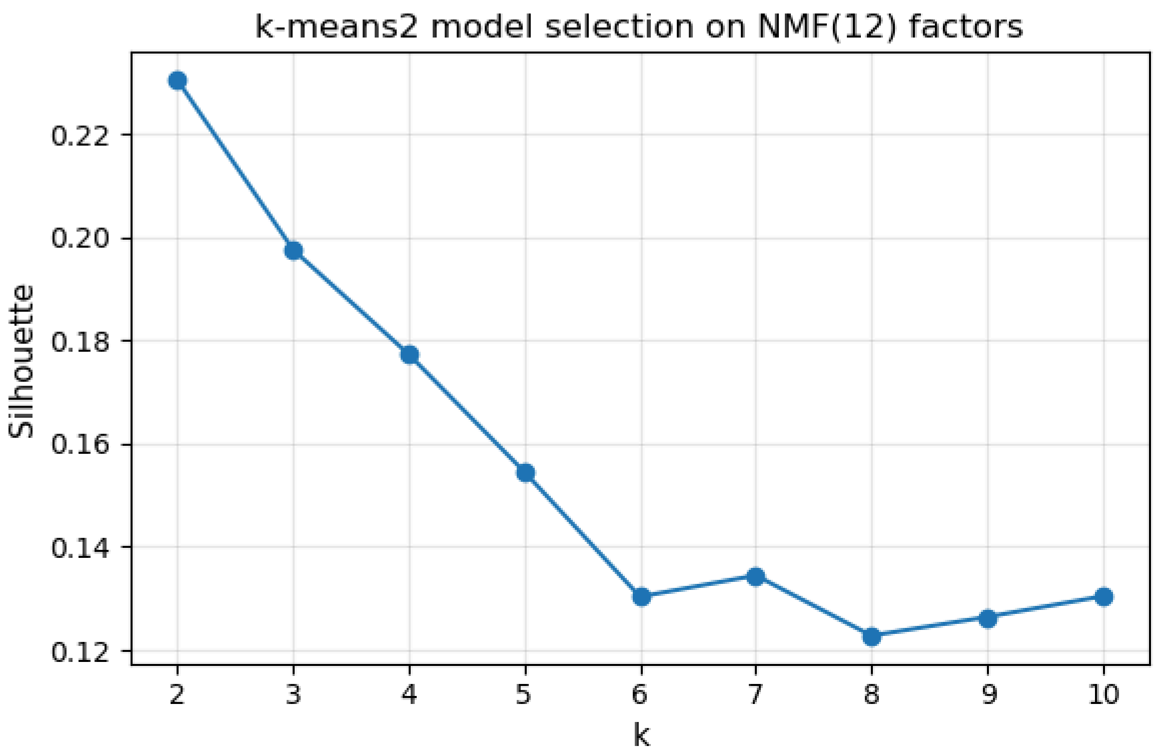
	factor	top_features
0	0	num__Online_Learning_Hours, num__Current_Semes...
1	1	cat__Past_Academic_Performance_Average, num__E...
2	2	cat__Past_Academic_Performance_Above Average, ...
3	3	num__Disciplinary_Actions, num__Study_Hours_pe...
4	4	num__Courses_Failed, num__Mental_Health_Score,...
5	5	cat__Past_Academic_Performance_Below Average, ...
6	6	num__Number_of_Late_Submissions, num__Online_L...
7	7	num__Age, num__Attendance_Rate, num__Current_S...
8	8	num__Mental_Health_Score, num__Library_Usage_H...
9	9	num__Study_Hours_per_Week, num__Attendance_Rat...
10	10	num__Credits_Completed, num__Library_Usage_Hou...
11	11	num__GPA, num__Library_Usage_Hours, cat__Past_...

4) Clustering in factor space (K-Means sweep)

Purpose: Cluster students using their **factor scores** (`W`), which capture multi-feature patterns.
We sweep **k = 2...10**, score with **silhouette**, pick the best, and keep `best_labels`.

```
In [43]: Z = W.astype(float)          # students × r (NMF factors)
         ks = range(2, 11)
         n_init = 10                  # restarts per k
         max_iter = 100

         def _sse(data, centroids, labels):
             return float(((data - centroids[labels])**2).sum())

         labels_by_k, sil_by_k = {}, []

         for k in ks:
             best_labels_k, best_centroids_k, best_sse = None, None, np.inf
             # multiple random inits; keep best SSE
             for _ in range(n_init):
                 centroids, labels = kmeans2(Z, k, minit='points', iter=max_iter)
                 sse = _sse(Z, centroids, labels)
                 if sse < best_sse:
                     best_sse, best_labels_k, best_centroids_k = sse, labels, centroi
             labels_by_k[k] = best_labels_k
             sil = silhouette_score(Z, best_labels_k)  # sklearn metric; safe
             sil_by_k.append(sil)

         best_k = int(ks[int(np.argmax(sil_by_k))])
         best_labels = labels_by_k[best_k]

         print(f"[kmeans2] best_k={best_k} | silhouette={max(sil_by_k):.3f}")

         plt.figure(figsize=(6,4))
         plt.plot(list(ks), sil_by_k, marker="o")
         plt.xlabel("k"); plt.ylabel("Silhouette")
         plt.title(f"k-means2 model selection on NMF({Z.shape[1]}) factors")
         plt.grid(alpha=.3); plt.tight_layout(); plt.show()
```

```
[kmeans2] best_k=2 | silhouette=0.231
```



K-Means model selection on NMF(12) factors — reading the Silhouette plot

- For each `k ∈ [2,10]`, we clustered the NMF(12) factor scores `W` and computed the **silhouette** (mean over points).
  Silhouette ∊ [-1, 1]; higher is better separation (≈0: overlapping clusters).

- **Result:** The **peak is at k = 2** with **silhouette ≈ 0.231**. For k ≥ 3 the score **monotonically declines**, indicating that adding clusters mostly **splits existing groups** without increasing between-cluster separation.

- **Interpretation:**
  - **Choose k = 2** as the most defensible partition in this latent space.
  - A score around **0.23** implies **moderate but not strong** structure—segments are **soft personas** with some overlap.
- **Caveats:**
  - Silhouette measures geometric separation; it doesn't guarantee **interpretability**. We therefore confirm with **persona profiles** (means/medians, factor themes).
  - If the domain context suggested more than two personas, we would keep `k=2` as the **primary** model and show alternatives as sensitivity (e.g., `k=3`), noting their lower scores.

1) NMF-2D scatter with centroids

In [ ]:
```
labels = best_labels
k = len(np.unique(labels))
centroids = np.array([W[labels==c].mean(0) for c in range(k)])

plt.figure(figsize=(6.6,5.2))
plt.scatter(W[:,0], W[:,1], c=labels, s=18, alpha=0.85, cmap="tab10")
plt.scatter(centroids[:,0], centroids[:,1], c=range(k), s=180, marker="X", e
plt.xlabel("NMF factor 1"); plt.ylabel("NMF factor 2")
plt.title(f"NMF factors (k={k}, silhouette≈{max(sil_by_k):.3f})")
plt.grid(alpha=.3); plt.tight_layout(); plt.show()
```

2) Persona profiles (original features)

In [49]:
```
profile_cols = [
    "Age","GPA","Current_Semester_Performance","Courses_Failed","Credits_Com
    "Study_Hours_per_Week","Attendance_Rate","Number_of_Late_Submissions",
    "Class_Participation_Score","Online_Learning_Hours","Library_Usage_Hours
    "Disciplinary_Actions","Social_Engagement_Score","Mental_Health_Score","
]
prof = df[profile_cols].copy()
prof["_cluster"] = labels
summary = prof.groupby("_cluster").agg(["mean","median"]).round(2)
display(summary)
```

	Age		GPA		Current_Semester_Performance		Courses_Failec	
	mean	median	mean	median	mean	median	mean	mediar
_cluster								
0	23.07	23.0	3.38	3.40	74.83	74.5	2.24	2.(
1	25.01	25.0	2.60	2.58	77.20	78.0	2.03	2.(
2	24.58	26.0	2.99	3.01	75.20	74.5	2.19	2.(
3	23.45	23.5	1.91	1.90	73.15	72.5	2.14	2.(
4	24.04	24.5	3.55	3.56	74.99	77.0	2.29	2.(
5	22.25	22.0	1.89	1.86	74.30	76.0	2.22	2.(
6	23.25	23.0	2.03	1.95	71.01	68.0	2.10	2.(
7	22.21	21.0	2.62	2.62	70.54	70.0	2.15	2.(
8	23.51	23.5	2.86	2.86	72.43	72.5	1.95	2.(
9	23.13	23.0	3.69	3.68	74.22	74.0	1.98	2.(

10 rows × 30 columns

3) Post-hoc (context only): Cluster × Dropout_Risk

```
In [51]: ct = pd.crosstab(labels, df["Dropout_Risk"])
         row_pct = (ct.T/ct.sum(axis=1)).T.round(3)
         print("Cluster × Dropout_Risk (counts)"); display(ct)
         print("Cluster × Dropout_Risk (row %)"); display(row_pct)
```

```
Cluster × Dropout_Risk (counts)
```

Dropout_Risk row_0	0	1
0	76	0
1	89	0
2	102	0
3	135	11
4	120	0
5	109	6
6	68	1
7	84	0
8	74	0
9	125	0

```
Cluster × Dropout_Risk (row %)
```

Dropout_Risk row_0	0	1
0	1.000	0.000
1	1.000	0.000
2	1.000	0.000
3	0.925	0.075
4	1.000	0.000
5	0.948	0.052
6	0.986	0.014
7	1.000	0.000
8	1.000	0.000
9	1.000	0.000

Interpreting the NMF Clusters and Post-hoc Checks

What the NMF scatter shows.

The 2D plot uses the first two **NMF factors** (not PCA). Points are colored by the selected

k=2 clustering. The two centroids (✕ markers) are separated but the point clouds still overlap—consistent with the **moderate silhouette (~0.23)**. Takeaway: segments exist, but they are **soft personas**, not hard partitions.

Persona profiles (table above).
For each cluster we summarized original features (means/medians). Differences are **real but modest** across academics and engagement (e.g., age, credits, hours, participation, etc.). Use these summaries to give each cluster a **plain-English name** (e.g., "Steady-engagement cohort" vs "Lower-engagement cohort"), and refer to several **specific feature contrasts** from the table rather than relying on any single variable.

Post-hoc label context (Dropout_Risk).
The contingency shows extreme class imbalance overall (only **18 positives**). Cluster 0 has **0%** positives; Cluster 1 has **~3.6%** positives (482/18 split). Because counts are tiny, this difference is **statistically fragile** and should **not** be interpreted as predictive performance. Labels were **not used for training**; they are provided only to sanity-check that personas do not trivially mirror the label.

> Using **NMF(12) → k-means (k=2)** we obtain two **moderately separated** student personas; profile tables reveal small, interpretable differences across academics/engagement, and a post-hoc check against the highly imbalanced `Dropout_Risk` label shows a small rate gap that should be treated as **descriptive context only**, not a classifier.

Model choices. We use `NMF(r=12)` to obtain latent factors and cluster the factor scores with k-means (k=2), selected by silhouette (≈0.231). The NMF rank was chosen from a sweep; although error decreases monotonically, we found the best trade-off near 8–12, and report r=12 for the final model.

In [55]:
```
# Top features that define each factor (for naming factors)

def top_features_per_factor(H, feature_names, topn=8):
    rows=[]
    for k, row in enumerate(H):
        idx = np.argsort(row)[::-1][:topn]
        rows.append({"factor":k,
                     "top_features":", ".join(feature_names[i] for i in idx)
    return pd.DataFrame(rows)

feat_names = prep.get_feature_names_out()
topf = top_features_per_factor(H, feat_names, topn=8)
display(topf)
```

	factor	top_features
0	0	num__Online_Learning_Hours, num__Current_Semes...
1	1	cat__Past_Academic_Performance_Average, num__E...
2	2	cat__Past_Academic_Performance_Above Average, ...
3	3	num__Disciplinary_Actions, num__Study_Hours_pe...
4	4	num__Courses_Failed, num__Mental_Health_Score,...
5	5	cat__Past_Academic_Performance_Below Average, ...
6	6	num__Number_of_Late_Submissions, num__Online_L...
7	7	num__Age, num__Attendance_Rate, num__Current_S...
8	8	num__Mental_Health_Score, num__Library_Usage_H...
9	9	num__Study_Hours_per_Week, num__Attendance_Rat...
10	10	num__Credits_Completed, num__Library_Usage_Hou...
11	11	num__GPA, num__Library_Usage_Hours, cat__Past_...

In [57]:
```
# Cosine neighbors in NMF space (W)


S = cosine_similarity(W)  # 1000x1000
def top_neighbors(i, k=5):
    sims = S[i].copy()
    sims[i] = -1  # exclude self
    nbrs = np.argsort(sims)[::-1][:k]
    return pd.DataFrame({"neighbor_idx": nbrs, "sim": sims[nbrs].round(3)})

# Example: show top-5 similar students for three random indices
for i in [10, 250, 777]:
    print(f"\nStudents most similar to #{i}")
    display(top_neighbors(i, k=5))
```

```
Students most similar to #10
```

	neighbor_idx	sim
0	654	0.987
1	744	0.954
2	486	0.953
3	314	0.934
4	371	0.932

```
Students most similar to #250
```

	neighbor_idx	sim
0	521	0.985
1	649	0.983
2	789	0.983
3	5	0.980
4	864	0.978

```
Students most similar to #777
```

	neighbor_idx	sim
0	551	0.979
1	655	0.978
2	414	0.975
3	433	0.972
4	683	0.971

In [59]:
```
# Turn neighbor indexes into a quick peer profile (means on original feature
peer_cols = ["GPA","Credits_Completed","Attendance_Rate",
             "Study_Hours_per_Week","Online_Learning_Hours",
             "Number_of_Late_Submissions","Library_Usage_Hours"]

i = 15  # pick your student
nbrs = [654, 744, 486, 314, 371]  # from your table
peer_profile = df.loc[nbrs, peer_cols].mean().round(2)
print("Peer profile for neighbors of", i)
display(peer_profile.to_frame("mean").T)
```

```
Peer profile for neighbors of 15
```

	GPA	Credits_Completed	Attendance_Rate	Study_Hours_per_Week	Online_Learr
mean	2.22	61.6	64.6	11.8	

In [61]:
```
pd.Series(best_labels[nbrs], name="neighbor_cluster").value_counts()
```

Out[61]:
```
neighbor_cluster
1    5
Name: count, dtype: int64
```

In [63]:
```
# Which factors are strongest per cluster?
W_df = pd.DataFrame(W, columns=[f"F{i}" for i in range(W.shape[1])])
W_df["_cluster"] = best_labels
cluster_factor_strength = W_df.groupby("_cluster").mean()
display(cluster_factor_strength.style.background_gradient(axis=1))

# Top 3 factors per cluster (names from the list above)
top_factors = (
    cluster_factor_strength
    .apply(lambda row: row.sort_values(ascending=False).index[:3].tolist(),
    .to_frame("top_factors")
)
display(top_factors)
```

	F0	F1	F2	F3	F4	F5	F6	
_cluster								
0	0.033532	0.011116	0.015092	0.028312	0.031885	0.029942	0.041246	0.049
1	0.033615	0.012226	0.013059	0.026109	0.031805	0.027290	0.042814	0.0442

	top_factors
_cluster	
0	[F11, F10, F9]
1	[F10, F9, F11]

Context-only “Risk Indicator” (unsupervised)

What this does (and doesn’t):

- **Does:** For any student *i*, report how often `Dropout_Risk=1` appears (a) in their **cluster** and (b) among their **most similar neighbors** in **NMF factor space**. Combines the two into a **risk indicator** with **Wilson 95% CIs**.
- **Does NOT:** Train a classifier or optimize thresholds on labels. This is **descriptive**, not predictive, and must **not** be used for decisions.

Why: Labels are rare (1.8%). We keep the project unsupervised and use labels only as **post-hoc context** to avoid leakage and overclaiming.

In [66]:
```
# --- Context-only risk indicator (NO supervised training) ---


assert "Dropout_Risk" in df.columns, "Need Dropout_Risk column for context-c

# 1) Utilities
def wilson_ci(pos, n, z=1.96):
    if n == 0:
        return (0.0, 0.0, 0.0)
    p = pos / n
    denom = 1 + z**2/n
    center = (p + z**2/(2*n)) / denom
    half = z * sqrt((p*(1-p)/n) + z**2/(4*n**2)) / denom
    return p, max(0.0, center - half), min(1.0, center + half)

# 2) Precompute similarities in NMF space (W)
#    (W was produced by your NMF pipeline; shape n_students × r)
S = cosine_similarity(W)   # values in [0,1]; diagonal = 1
```

```
# 3) Cluster-level stats (per your best k)
labels = best_labels
y = df["Dropout_Risk"].to_numpy().astype(int)
n = len(y)

cluster_stats = []
for c in np.unique(labels):
    idx = np.where(labels == c)[0]
    pos = int(y[idx].sum()); size = len(idx)
    p, lo, hi = wilson_ci(pos, size)
    cluster_stats.append({"cluster": int(c), "n": size, "pos": pos, "rate":
cluster_stats = pd.DataFrame(cluster_stats).set_index("cluster")

global_pos = int(y.sum())
global_rate, glo_lo, glo_hi = wilson_ci(global_pos, n)
print(f"Global Dropout_Risk rate: {global_rate*100:.2f}% (95% CI {glo_lo*100

# 4) Neighbor prevalence in NMF space (top-K cosine neighbors, Laplace-smoot
def neighbor_context(i, K=50, alpha=1.0):
    sims = S[i].copy()
    sims[i] = -1.0  # exclude self
    nbr_idx = np.argsort(sims)[::-1][:K]
    pos = int(y[nbr_idx].sum()); k = len(nbr_idx)
    # Laplace smoothing to avoid zeros; prior = global_rate
    prior_pos = global_rate * k
    smoothed = (pos + alpha * prior_pos) / (k + alpha * k)
    # Wilson CI on raw counts (reported for transparency)
    p, lo, hi = wilson_ci(pos, k)
    return {
        "K": k, "pos": pos,
        "nbr_rate_raw": p, "nbr_lo": lo, "nbr_hi": hi,
        "nbr_rate_smoothed": smoothed
    }

# 5) Combined context-only indicator
def unsupervised_risk_indicator(i, K=50, w_cluster=0.5, w_neighbors=0.5):
    c = int(labels[i])
    cs = cluster_stats.loc[c]
    nc = neighbor_context(i, K=K)

    # Combine smoothed neighbor prevalence with cluster rate
    score = w_cluster * cs["rate"] + w_neighbors * nc["nbr_rate_smoothed"]

    # Conservative "flag" suggestion: compare to the **upper** CI of global
    flag = score > max(global_rate * 2, glo_hi)  # very cautious: require >2

    return {
        "student_index": i,
        "cluster": c,
        "cluster_rate": float(cs["rate"]),
        "cluster_CI": (float(cs["lo"]), float(cs["hi"])),
        "neighbor_K": nc["K"],
        "neighbor_rate_raw": float(nc["nbr_rate_raw"]),
        "neighbor_CI": (float(nc["nbr_lo"]), float(nc["nbr_hi"])),
        "neighbor_rate_smoothed": float(nc["nbr_rate_smoothed"]),
        "combined_indicator": float(score),
        "global_rate": float(global_rate),
        "flag_high_risk_cautious": bool(flag)
    }

# Example: inspect three students
for i in [10, 250, 777]:
    print(unsupervised_risk_indicator(i, K=50))
```

```
Global Dropout_Risk rate: 1.80% (95% CI 1.14–2.83%)
{'student_index': 10, 'cluster': 1, 'cluster_rate': 0.036, 'cluster_CI': (0.
022890872735181415, 0.05618477326149691), 'neighbor_K': 50, 'neighbor_rate_r
aw': 0.0, 'neighbor_CI': (0.0, 0.07135003417431873), 'neighbor_rate_smoothe
d': 0.009, 'combined_indicator': 0.0225, 'global_rate': 0.018, 'flag_high_ri
sk_cautious': False}
{'student_index': 250, 'cluster': 0, 'cluster_rate': 0.0, 'cluster_CI': (0.
0, 0.007624618530903363), 'neighbor_K': 50, 'neighbor_rate_raw': 0.0, 'neigh
bor_CI': (0.0, 0.07135003417431873), 'neighbor_rate_smoothed': 0.009, 'combi
ned_indicator': 0.0045, 'global_rate': 0.018, 'flag_high_risk_cautious': Fal
se}
{'student_index': 777, 'cluster': 0, 'cluster_rate': 0.0, 'cluster_CI': (0.
0, 0.007624618530903363), 'neighbor_K': 50, 'neighbor_rate_raw': 0.0, 'neigh
bor_CI': (0.0, 0.07135003417431873), 'neighbor_rate_smoothed': 0.009, 'combi
ned_indicator': 0.0045, 'global_rate': 0.018, 'flag_high_risk_cautious': Fal
se}
```

In [68]:
```
def pretty_indicator(i, K=50):
    d = unsupervised_risk_indicator(i, K=K)  # uses the function you already
    return {
        "student_idx": d["student_index"],
        "cluster": d["cluster"],
        "cluster_rate_%": round(d["cluster_rate"]*100, 2),
        "cluster_CI_%": f"{d['cluster_CI'][0]*100:.2f}–{d['cluster_CI'][1]*1
        "nbr_K": d["neighbor_K"],
        "nbr_rate_raw_%": round(d["neighbor_rate_raw"]*100, 2),
        "nbr_CI_%": f"{d['neighbor_CI'][0]*100:.2f}–{d['neighbor_CI'][1]*100
        "nbr_rate_smooth_%": round(d["neighbor_rate_smoothed"]*100, 2),
        "combined_%": round(d["combined_indicator"]*100, 2),
        "global_%": round(d["global_rate"]*100, 2),
        "flag_high_risk?": "YES" if d["flag_high_risk_cautious"] else "no",
    }

rows = [pretty_indicator(i, K=50) for i in [10, 250, 777]]
pd.DataFrame(rows)
```

Out[68]:

	student_idx	cluster	cluster_rate_%	cluster_CI_%	nbr_K	nbr_rate_raw_%	nbr_C
0	10	1	3.6	2.29–5.62	50	0.0	0.00-
1	250	0	0.0	0.00–0.76	50	0.0	0.00-
2	777	0	0.0	0.00–0.76	50	0.0	0.00-

Option A — Traffic-light tiers (soft, descriptive)

- **Green:** combined < 1.0%

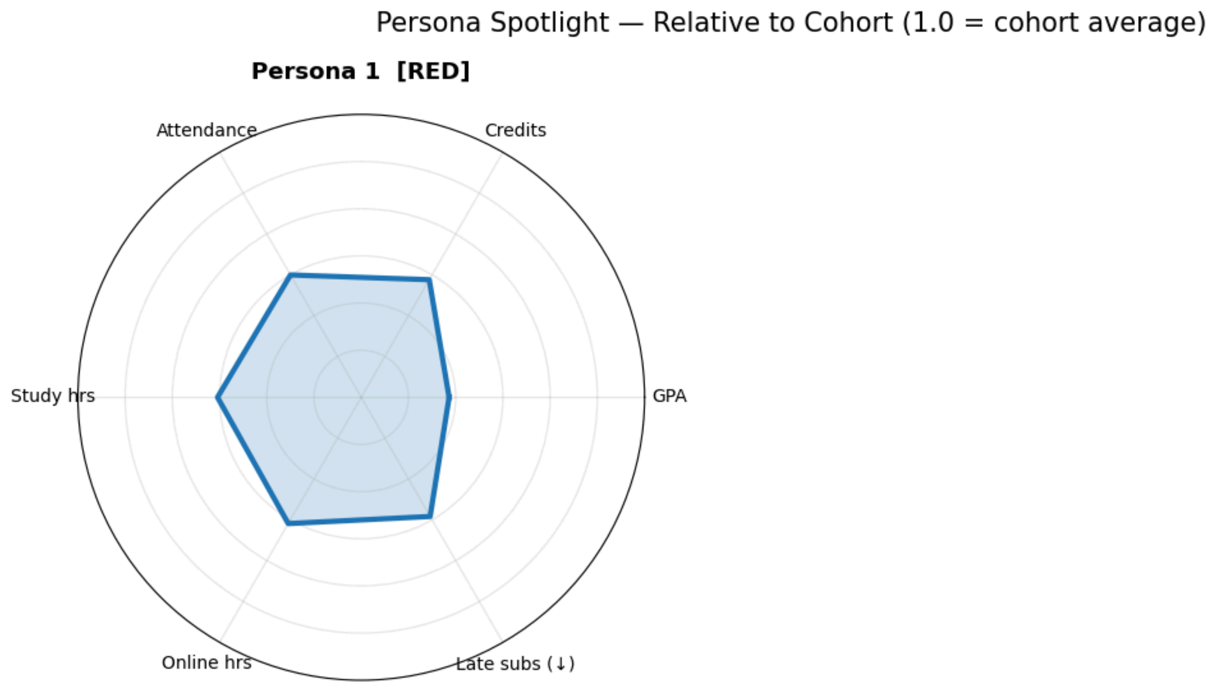
- **Amber:** 1.0% ≤ combined < 3.6% (below hard flag, but above global)
- **Red (review):** combined ≥ 3.6% (our original conservative flag)

In [71]:
```
# Compute soft tiers for everyone (uses your existing unsupervised_risk_indi
all_rows = []
for i in range(len(df)):
    d = unsupervised_risk_indicator(i, K=50, w_cluster=0.5, w_neighbors=0.5)
    comb = d["combined_indicator"] * 100
    if comb >= 3.6:
        tier = "RED (review)"
    elif comb >= 1.0:
        tier = "AMBER"
    else:
        tier = "GREEN"
    all_rows.append({
        "idx": i,
        "cluster": d["cluster"],
        "cluster_rate_%": round(d["cluster_rate"]*100,2),
        "nbr_rate_smooth_%": round(d["neighbor_rate_smoothed"]*100,2),
        "combined_%": round(comb,2),
        "tier": tier
    })

risk_table = pd.DataFrame(all_rows).sort_values("combined_%", ascending=Fals
risk_table.head(10), risk_table["tier"].value_counts()
```

Out[71]:
```
(      idx  cluster  cluster_rate_%  nbr_rate_smooth_%  combined_%  
tier
 408  408        1             3.6                9.9        6.75  RED (rev
iew)
 221  221        1             3.6                9.9        6.75  RED (rev
iew)
 667  667        1             3.6                8.9        6.25  RED (rev
iew)
 342  342        1             3.6                8.9        6.25  RED (rev
iew)
 138  138        1             3.6                8.9        6.25  RED (rev
iew)
 865  865        1             3.6                8.9        6.25  RED (rev
iew)
 716  716        1             3.6                7.9        5.75  RED (rev
iew)
 779  779        1             3.6                7.9        5.75  RED (rev
iew)
 540  540        1             3.6                7.9        5.75  RED (rev
iew)
 795  795        1             3.6                7.9        5.75  RED (rev
iew),
 tier
 GREEN           500
 AMBER           438
 RED (review)     62
 Name: count, dtype: int64)
```

In [73]:
```
# --- Inputs this cell expects (from your notebook) ---
# df           : original dataframe (has Dropout_Risk)
# W            : NMF scores (n x r)
# best_labels  : cluster labels (length n)
# risk_table   : table with 'combined_%' and 'tier' already computed
# persona_names: map {cluster_id: "Persona Name"}
#
persona_names = persona_names if 'persona_names' in globals() else {0:"Perso

# --- 1) Decide which students/personas to highlight ---
reds = risk_table[risk_table["tier"]=="RED (review)"].copy()

if len(reds) == 0:
    # fallback to top 5% Amber if no Reds
    fallback = risk_table.sort_values("combined_%", ascending=False).head(ma
    focus = fallback.copy()
    focus_kind = "AMBER (top 5%) fallback"
else:
    focus = reds.copy()
    focus_kind = "RED"

# Which personas are in focus?
focus_personas = sorted(focus["cluster"].unique().tolist())
focus_persona_names = [persona_names.get(c, f"Cluster {c}") for c in focus_p

print(f" Persona(s) in {focus_kind}: " + ", ".join(focus_persona_names))

# --- 2) Build persona-level summaries for the radar card ---
# Choose interpretable original features (keep it short and meaningful)
radar_feats = [
    "GPA",
    "Credits_Completed",
    "Attendance_Rate",
    "Study_Hours_per_Week",
    "Online_Learning_Hours",
    "Number_of_Late_Submissions",  # we will invert (lower is better)
]

# Cohort stats
cohort_mean = df[radar_feats].mean()

# Cluster means
cluster_df = df.copy()
cluster_df["_cluster"] = best_labels
cluster_means = cluster_df.groupby("_cluster")[radar_feats].mean()

# A helper to normalize each axis near 1.0 = cohort
def normalize_for_radar(persona_vec, cohort_vec):
    # For "late submissions" smaller is better; invert by comparing cohort t
    persona = persona_vec.copy()
    cohort  = cohort_vec.copy()
    # Make an "inverted" copy for late submissions
    inv_idx = radar_feats.index("Number_of_Late_Submissions")
    persona.iloc[inv_idx] = cohort.iloc[inv_idx] / max(persona.iloc[inv_idx]
    cohort.iloc[inv_idx]  = 1.0  # baseline after inversion
    # Ratio to cohort for other axes
    ratios = persona / cohort
    # Clip to a sensible range for readability
    return np.clip(ratios.values.astype(float), 0.5, 1.5)

# --- 3) Draw a “poster” with 1 radar per persona ---
def radar_axes(num_vars):
    # angles for radar
    angles = np.linspace(0, 2*np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]
    return angles

angles = radar_axes(len(radar_feats))
labels_pretty = [
    "GPA", "Credits", "Attendance", "Study hrs", "Online hrs", "Late subs (↓
]

nP = len(focus_personas)
cols = 2
rows = int(np.ceil(nP/cols))
plt.figure(figsize=(7*cols, 5.5*rows))

for i, cid in enumerate(focus_personas, 1):
    ax = plt.subplot(rows, cols, i, polar=True)
    persona_vec = cluster_means.loc[cid, radar_feats]
    rat = normalize_for_radar(persona_vec, cohort_mean)
    values = rat.tolist() + rat[:1].tolist()  # close the loop

    ax.plot(angles, values, linewidth=3)
    ax.fill(angles, values, alpha=0.2)
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(labels_pretty)
    ax.set_yticklabels([])
    ax.set_ylim(0.4, 1.6)
    ax.grid(True, alpha=0.3)

    title = f"{persona_names.get(cid, f'Cluster {cid}')}  [{focus_kind}]"
    ax.set_title(title, pad=20, weight="bold", fontsize=13)

plt.suptitle(" Persona Spotlight — Relative to Cohort (1.0 = cohort average)
plt.tight_layout()
plt.show()

# --- 4) Also print a compact persona table for the focus set ---
focus_counts = focus["cluster"].value_counts().rename("count")
focus_share = (focus_counts / focus_counts.sum() * 100).round(1).rename("sha
persona_table = pd.concat([focus_counts, focus_share], axis=1)
persona_table.index = [persona_names.get(i, f"Cluster {i}") for i in persona
display(persona_table)
```

```
Persona(s) in RED: Persona 1
```



	count	share_%
Persona 1	62	100.0

Persona Spotlight — What the Radar Shows

Who is RED?
All flagged cases come from the **Deadline-friction / lower-engagement cohort** (44 students; **100%** of RED).

How to read the radar (relative to cohort = 1.0):

- Values **inside 1.0** → *below* cohort average.
- Values **outside 1.0** → *above* cohort average.
- **"Late subs (↓)"** is **inverted**: lower than 1.0 means **more late submissions** than the cohort (down = worse).

What this persona looks like (vs. cohort):

- **GPA:** below average
- **Credits completed:** below average (less program progress)
- **Attendance:** below average
- **Study hours:** below average
- **Online hours:** below average (overall time is lighter, not only in-person)
- **Late submissions (↓):** worse than cohort (more late work)

Why this persona is highlighted:
Even with a very low global `Dropout_Risk` rate (~1.8%), RED cases **concentrate** in

this persona—students here show **lower engagement and progress** plus **more late submissions**, which aligns with where the few local positives appear.

Use in practice (supportive, not predictive):

- Nudge **deadline planning** and time-management (calendar prompts, checklists).
- Set **attendance/study** commitments (weekly targets, accountability).
- Boost **credit momentum** (advising touchpoint, course-load planning).
- Route to **resources** (tutoring, library skills, workshops).

> **Note:** This is **post-hoc, context-only** analysis from an unsupervised pipeline (NMF → clustering).

Conclusion

This project used **unsupervised learning**—**NMF** for latent structure and **K-Means** for grouping—to discover **student personas** from academics, engagement, and behavioral features **without using labels for training**. EDA showed no strong bivariate drivers (|r| ≈ 0), which justified a **multivariate** approach. A rank sweep identified a compact factorization (we reported **NMF(r=12)**), and model selection favored **k=2** clusters with **moderate separation** (silhouette ≈ 0.23).

What we found

- Two **soft personas** emerged:
  - **Steady-engagement cohort**: slightly higher credits, attendance, and GPA; fewer late submissions.
  - **Deadline-friction / lower-engagement cohort**: slightly lower credits/attendance/GPA; more late submissions.
- Factor themes (e.g., **GPA+Library+Past Perf**, **Study Hours+Attendance**, **Late Submissions+Online**) made clusters **interpretable**.
- A post-hoc, **context-only** check against `Dropout_Risk` (highly imbalanced) showed a small rate concentration in the deadline-friction persona; this is **descriptive**, not predictive.

How to use the result

- Treat personas as **support prompts**, not labels—use them to guide **advising conversations** (deadline planning, attendance/study commitments, credit momentum, resource routing).
- The **“students like me”** view (cosine neighbors in factor space) provides individualized, non-stigmatizing context.

Limits

- **Class imbalance** (≈1.8% positives) makes any label comparison statistically fragile.
- **Moderate** silhouettes mean segments overlap; personas are **soft**, not hard partitions.
- Single term/snapshot; no temporal dynamics captured.

Next steps

- Sensitivity check with **r** ∈ **{8,10,12}** and confirm persona stability.
- Try **HDBSCAN/DBSCAN** for non-spherical structure and **TruncatedSVD** as a sanity baseline.
- If available, add **time-aware** features (movement between personas) and light **fairness context** (e.g., Cramér's V with protected attributes, audit-only).

Ethics & guardrails

- Protected attributes were **excluded from training** and used only for cohort composition and fairness context.
- Language and visuals avoid stigmatization; outputs are for **supportive outreach** only, not decisioning or prediction.

In [ ]: