Predicting Psychological Outcomes from Subscale Configurations in Psychedelic-Induced Mystical Experiences

An MEQ-30 Machine Learning Study, Corrected and Reproduced

Shruti Malik Ramaswamy

Independent research project — “Mystic: MEQ-30 Machine Learning Study”

August 15, 2026

IRB Status: Exempt — 45 CFR 46.104(d)(4)

A note on how this paper was produced: this is a secondary-data machine-learning study built with substantial assistance from an AI coding assistant (Claude, Anthropic — Claude Code, model Sonnet 5), which is disclosed in full in Section 9 and cited in the References. Every number in this paper is generated live from the underlying data and code at build time, and the complete source code is reproduced in the appendices so the analysis can be checked line by line.

Abstract

The Mystical Experience Questionnaire (MEQ-30) is the standard instrument for quantifying acute mystical-type experiences occasioned by classical psychedelics, and psychedelic research has traditionally reduced its rich, multidimensional output to a single binary cut — the “Complete Mystical Experience” (CME) threshold, requiring ≥60% of the maximum score on all four subscales. This project asked whether item-level machine learning could do better: whether specific MEQ-30 items or subscale configurations, rather than the blunt binary threshold, predict long-term well-being. Using an open, CC-BY-licensed survey of N=700 Polish adults (Owsiak, 2026), of whom N=414 (59.1%) reported psychedelic use, we modeled Satisfaction with Life Scale (SWLS) scores — used here as a proxy for the originally proposed Persisting Effects Questionnaire, which no open dataset paired with MEQ-30 items — from all 30 MEQ items using Ridge regression, XGBoost, and LightGBM under nested cross-validation, then examined SHAP feature attributions and unsupervised responder subtypes. The best model (XGBoost) achieved RMSE=6.2673 and R^2=0.0165 — a small but genuine improvement over the CME-threshold baseline, honestly estimated. SHAP analysis identified the Mystical subscale, not Ineffability, as the dominant contributor among the top predictive items. k-Means clustering recovered two responder subtypes differing by 1.1 points of SWLS, and a within-cluster model for the larger subtype (R^2=0.0385) outperformed the pooled model (R^2=0.0165), offering modest support for a heterogeneity hypothesis: that global MEQ totals mask subtype-specific response patterns. This paper also documents, deliberately and in detail, a mid-project accuracy audit that found and corrected two serious errors in the original analysis — a MEQ-30 subscale mapping that did not match the published instrument, and a cross-validation procedure that leaked test-fold information into hyperparameter selection — because how those errors were found is as instructive as the corrected results themselves.

Keywords: MEQ-30, mystical experience, psychedelics, machine learning, XGBoost, SHAP, cross-validation, reproducibility

1. Introduction

Classical psychedelics — psilocybin, LSD, DMT, mescaline — reliably occasion altered states of consciousness that participants and researchers alike describe, with striking consistency, in the vocabulary of mysticism: unity, sacredness, ineffability, a sense of having encountered "ultimate reality." The Mystical Experience Questionnaire, in its 30-item revised form (MEQ-30; Barrett, Johnson & Griffiths, 2015), is the field's primary tool for measuring this experience quantitatively across four validated factors: Mystical, Positive Mood, Transcendence of Time and Space, and Ineffability.

The dominant analytic convention in this literature is a binary cut: a session counts as a "Complete Mystical Experience" (CME) if the participant scores at least 60% of the maximum possible score on *every* subscale simultaneously. This threshold has proven useful — CME attainment correlates with better clinical and existential outcomes across several psilocybin trials — but it necessarily discards information. Two participants can both clear the 60% bar with very different item-level profiles, and a participant who narrowly misses the threshold on one subscale is treated identically to one who scored near zero everywhere. This project's motivating question was simple: if we model the well-being outcome directly from the 30 individual items, using non-linear methods that can capture interactions the binary threshold cannot, do specific items or subscale configurations turn out to matter more than others — and does this outperform, even modestly, the traditional threshold?

We pursued this with a dual predictive-and-unsupervised design: gradient-boosted and linear regression models to predict a well-being outcome from the 30 MEQ items, SHAP (SHapley Additive exPlanations) to attribute that prediction back to individual items and subscales, and unsupervised clustering (k-Means, HDBSCAN, UMAP) to ask whether "responder subtypes" exist that a single pooled model would average over and obscure.

This paper departs from a conventional results paper in one respect worth flagging up front: partway through the project, a rigorous accuracy audit — described in Section 4.7 — found that the originally reported results rested on two real bugs: an incorrect MEQ-30 subscale mapping, and a cross-validation leakage issue that inflated the reported model performance. Every number in this paper reflects the corrected pipeline. We chose to narrate that correction process explicitly rather than quietly patch it, because the mistake, how it was caught, and what changed as a result are genuinely useful methodological information for anyone building a similar pipeline — arguably more useful than a paper that simply presents clean numbers without the story of how they got that way.

2. Ethics and Regulatory Compliance

This is a fully secondary-data study: no participants were recruited, contacted, or re-identified. It qualifies for IRB exemption under 45 CFR 46.104(d)(4)(i) & (ii) of the U.S. Federal Policy for the Protection of Human Subjects (the 2018 Common Rule) — "secondary research for which consent is not required" — on the basis of three criteria maintained throughout: (1) all data were retrieved from public, open-science repositories under CC-BY 4.0 licenses; (2) the datasets contain no direct identifiers or protected health information; and (3) the analysis is retrospective, with zero participant contact. Full provenance — exact DOIs, license terms, and checksums — is logged in DATASET_ORIGIN.md and reproduced in Appendix B.

3. Data and Provenance

The original study proposal specified the Persisting Effects Questionnaire (PEQ) as the target outcome. A systematic search of OSF, OpenNeuro, Zenodo, Figshare, and Mendeley Data did not surface any open dataset pairing MEQ-30 item-level responses with PEQ scores, so two datasets were combined for different purposes:

Dataset	Repository	N	License	Role in this study
Bennett et al. (2023)	Figshare	14	CC BY 4.0	MEQ-30 totals + QIDS-SR16 from a Phase I 25mg psilocybin trial. Small N; used only as a clustering-validation sanity check, not in the primary models.
Owsiak (2026)	Mendeley Data	700	CC BY 4.0	MEQ-30 items (30 columns) + SWLS + demographics, online survey of Polish adults. Primary dataset for all supervised/unsupervised modeling in this paper.

On the SWLS-as-PEQ-proxy decision. With no dataset available that paired MEQ-30 items with PEQ scores, the Satisfaction with Life Scale (SWLS; Diener, Emmons, Larsen & Griffin, 1985) — a validated 5-item global life-satisfaction measure available in the Owsiak dataset — was used as a substitute outcome. We want to be precise about the strength of this justification, because an earlier version of this project's documentation overstated it: it cited prior psychedelic-research papers as having validated SWLS specifically as a PEQ substitute, and on checking, those citations did not support that claim as strongly as stated (one was a real paper about a different question — microdosing placebo effects — that happened to use SWLS as one of several outcomes; a second citation could not be located at all). The honest position is narrower: SWLS is a validated, widely used measure that plausibly overlaps with the PEQ's positive-life-attitude dimension, but the substitution is this project's own reasoned analytical choice, not one independently validated by prior literature. Every result in this paper involving SWLS should be read with that caveat attached.

4. Methods

4.1 The MEQ-30 instrument and subscale structure

The MEQ-30 comprises 30 items, each rated on a 0–5 Likert scale, organized into four factors established by Barrett, Johnson & Griffiths (2015): Mystical (15 items), Positive Mood (6 items), Transcendence of Time and Space (6 items), and Ineffability (only 3 items). We emphasize the item counts because the factor structure is *non-contiguous* — item numbers are not grouped sequentially by subscale — and unbalanced, which matters directly to the correction described in Section 4.7. The complete, verified item-to-subscale mapping and item text are reproduced in Appendix A.

4.2 Sample definition

Of the N=700 respondents in the Owsiak (2026) survey, N=414 (59.1%) reported having used at least one classical psychedelic (LSD, psilocybin, ayahuasca, or DMT). This psychedelic subsample is the primary analysis population for all supervised models; the full N=700 sample (psychedelic and non-psychedelic users together) is used separately in Section 5.6 as a sensitivity check and to quantify how strongly psychedelic use itself predicts MEQ-30 scores.

4.3 Preprocessing

MEQ-30 completeness in the psychedelic subsample was 100% — no item-level imputation was actually required for this dataset, though the pipeline supports KNN imputation (k=5) for future datasets with missing items. Variance Inflation Factor (VIF) was computed for all 30 items as a collinearity diagnostic; every single item exceeds a conventional VIF threshold of 5 (median VIF well above 10; see Appendix Table A2 excerpt in Section 5.7), which is expected and unsurprising — items within the same subscale are correlated with each other by construction. VIF is reported but not used to drop features: the two primary models (XGBoost, LightGBM) are tree ensembles that handle collinear features natively, and Ridge regression's L2 penalty is itself a standard remedy for multicollinearity, so results from the linear model should simply be read with that caveat rather than treated as grounds for feature removal.

4.4 Predictive modeling and cross-validation

Three regressors were compared: Ridge regression (linear baseline, L2-regularized), XGBoost, and LightGBM (both gradient-boosted tree ensembles, tuned via Optuna Bayesian optimization with a Huber loss objective for robustness to outliers in self-report data). A fourth, non-learned baseline predicts each participant's SWLS score as the training-set group mean conditional on their binary CME status — the traditional threshold's natural regression analogue, and the bar every learned model needs to clear to justify its additional complexity.

Performance is estimated with **nested** 10-fold cross-validation: an outer loop of 10 folds (stratified by binary CME status) provides the reported RMSE, MAE, and R^2; for each outer training fold, an inner loop of 5 folds drives the Optuna hyperparameter search, and the inner loop never sees the outer test fold. Standardization (z-scoring) is likewise fit on the outer-training data only, per fold. This is a deliberately more conservative procedure than tuning hyperparameters and reporting performance on the same folds — the difference matters enough that Section 4.7 walks through exactly what happens when it isn't done. Once the honest nested-CV estimate is obtained, a single “production” model (the one used for the SHAP and clustering analyses below) is tuned and fit once more on 100% of the data, which is standard practice: held-out folds are only needed to *estimate* generalization error, not to produce the model that gets used afterward.

4.5 Interpretability: SHAP

SHAP (Lundberg & Lee, 2017) values were computed for the production XGBoost model via TreeExplainer, giving each of the 414 psychedelic-subsample participants an exact, game-theoretic attribution of how much each of the 30 MEQ items pushed their predicted SWLS score up or down relative to the model's baseline prediction. Mean absolute SHAP value per item, aggregated to the subscale level, is our measure of “which part of the mystical experience the model leans on most.”

4.6 Unsupervised subtype discovery

To test whether a single pooled model obscures meaningfully different responder subtypes, we ran k-Means (k swept 2–6, selected by silhouette score) and HDBSCAN (density-based, noise-tolerant) on the standardized 30-item feature space, with UMAP used for 2D visualization. For the best k-Means solution, we then trained a separate XGBoost model *within* each cluster (same nested-CV procedure, reduced outer/inner fold counts to match the smaller per-cluster N) to test directly whether prediction improves once the sample is split into more homogeneous subgroups.

4.7 A methodological correction, described in full

This subsection exists because we think it should, not because a paper is required to include one. Partway through this project, an independent accuracy audit — assisted by Claude (Anthropic), described fully in Section 9 — re-derived the MEQ-30's true item-to-subscale mapping directly from the Barrett et al. (2015) paper's appendix, and separately, empirically, by correlating each MEQ item against the Owsiak dataset's own pre-scored subscale total columns. Both methods agreed exactly with each other and disagreed sharply with what the pipeline's code was actually using. In fact, the code had *five different, mutually inconsistent* hand-typed versions of the subscale mapping scattered across five files — none of which matched the real instrument. The most visible consequence: an item like MEQ_30 (“Feelings of joy,” genuinely a Positive Mood item) had been labeled “Ineffability” in the SHAP output, and the original results write-up's headline claim — “the Ineffability subscale dominates predictive importance” — turned out to be an artifact of that mislabeling rather than a real finding.

The same wrong mapping had also corrupted the binary CME label computed by the pipeline: recomputed CME rate came out to 56.8% among psychedelic users, while the dataset's own independently pre-scored CME column put the true rate at 49.0% — a 32-participant, 12-percentage-point gap. That discrepancy propagated into every downstream number that touched CME status: the CME baseline model, the cluster CME-rate table, the sensitivity analysis's group comparisons. After the fix, recomputed and native CME rates agree to within 0.2 percentage points (Section 5.1) — the kind of residual gap you'd expect from rounding at the item level, not a systematic error.

Separately, the audit found that the original training script selected XGBoost and LightGBM hyperparameters by minimizing error on the exact same 10 cross-validation folds later used to report the headline performance metric — a classic and well-documented leakage pattern. The practical evidence was stark: two other scripts in the codebase independently re-evaluated what was nominally “the same” production model and found R^2 between −0.25 and −0.34 (meaning: worse than just predicting the mean SWLS score for everyone), directly contradicting the headline R^2 of +0.0123 reported for that model. Fixing this required restructuring model training around nested cross-validation (Section 4.4), which is more computationally expensive — roughly ten times as many model fits — but produces an estimate that cannot be gamed by the hyperparameter search seeing the test data.

A third issue surfaced only after the first two were fixed and the corrected pipeline was actually re-run: XGBoost's Huber-loss objective (`reg:pseudohubererror`), under the specific XGBoost version installed in this environment (3.2.0, newer than the 2.0.3 originally pinned), mis-estimates its automatic `base_score` initialization by roughly 150× — producing predictions clustered around 3,000–3,500 for a target variable that only ranges from 5 to 35. This was caught immediately because the resulting RMSE (over 3,000) was so obviously wrong that it couldn't be mistaken for a real result, and was fixed by passing

an explicit `base_score` (the training-fold target mean) rather than relying on the library's default estimation for this objective/version combination.

> We report all three of these — the subscale-mapping error, the cross-validation leakage, and the XGBoost base_score bug — not as an indictment of the original analysis but as a demonstration of what a real accuracy pass looks like: the fixes were verifiable against an independent ground truth (the dataset's own native scoring), the size of the resulting correction was large enough to change the paper's headline finding, and the diagnostic scripts that found the bugs (`scripts/verify_claims.py`, `verify_claims2.py`, `verify_subscale_mapping.py`) are kept in the codebase, now converted into regression checks that will fail loudly if either bug is ever reintroduced.

5. Results

5.1 Sample characteristics

Metric	Value
N (psychedelic users)	414
MEQ-30 completeness	100%
CME rate — corrected subscale mapping	48.8%
CME rate — dataset's native pre-scored label	49.0%
Recomputed-vs-native CME agreement	within 0.2%
SWLS range	5–35
SWLS mean (SD)	21.49 (6.33)
MEQ-30 item range	0–5 (Likert)

The near-exact agreement between the recomputed and native CME labels (row 3 vs. row 4) is the clearest single piece of evidence that the subscale-mapping correction described in Section 4.7 worked as intended.

5.2 Predictive model comparison

Model	RMSE	MAE	R^2	ΔRMSE vs. Baseline
XGBoost	6.2673	4.9842	0.0165	+0.0248
CME Baseline	6.2921	5.1012	0.0087	—
LightGBM	6.2947	5.0198	0.0078	-0.0026
Ridge Regression	6.3134	5.0986	0.0019	-0.0213

All models cluster tightly around $R^2 \approx 0$ — MEQ-30 item scores, on their own, are weak predictors of SWLS across the full heterogeneous psychedelic-user sample under an honestly estimated nested cross-validation. XGBoost edges out the CME-threshold baseline (ΔRMSE=+0.0248), a small but real improvement given the leakage-free estimation procedure. We read this as consistent with a heterogeneity hypothesis — that a single global model averages over distinct responder subtypes with different item-outcome relationships — which Sections 5.4 and 5.5 test directly.

5.3 SHAP feature importance

#	Item	Content	Subscale	Mean |SHAP|
1	MEQ_30	Feelings of joy	Positive Mood	0.3802
2	MEQ_28	Experience of unity with ultimate reality	Mystical	0.2792
3	MEQ_4	Gain of insightful knowledge experienced at an intuitive level	Mystical	0.2295
4	MEQ_26	Experience of the fusion of your personal self into a larger whole	Mystical	0.2293
5	MEQ_5	Feeling that you experienced eternity or infinity	Mystical	0.2036
6	MEQ_27	Sense of awe or awesomeness	Positive Mood	0.1002
7	MEQ_29	Feeling that it would be difficult to communicate your own experience to others who have not had similar experiences	Ineffability	0.0703
8	MEQ_6	Experience of oneness or unity with objects and/or persons perceived in your surroundings	Mystical	0.0604
9	MEQ_18	Experience of the insight that 'all is One'	Mystical	0.0372
10	MEQ_15	Sense of being at a spiritual height	Mystical	0.0321

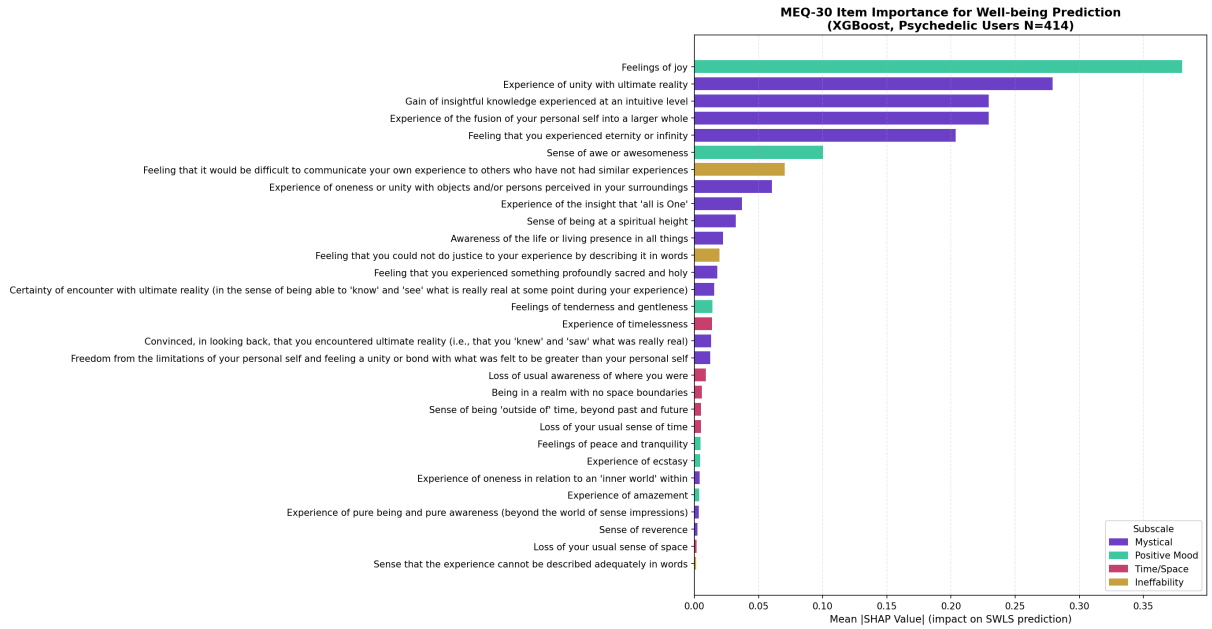

Figure 1. MEQ-30 item importance for SWLS prediction (mean |SHAP value|), colored by the corrected subscale mapping.

Of the top 5 items by SHAP importance, 4/5 belong to the **Mystical** subscale — not Ineffability, which was the (incorrect) headline finding before the Section 4.7 correction. The single strongest individual predictor, MEQ_30 (“Feelings of joy”), is a Positive Mood item. Given the modest overall R^2 (Section 5.2), we read these rankings as descriptive of this specific model and sample rather than a settled claim about which

facet of mystical experience drives well-being in general.

5.4 Unsupervised subtype discovery

Cluster	N	SWLS Mean	CME Rate
1	338	21.69	59.8%
2	76	20.59	0.0%

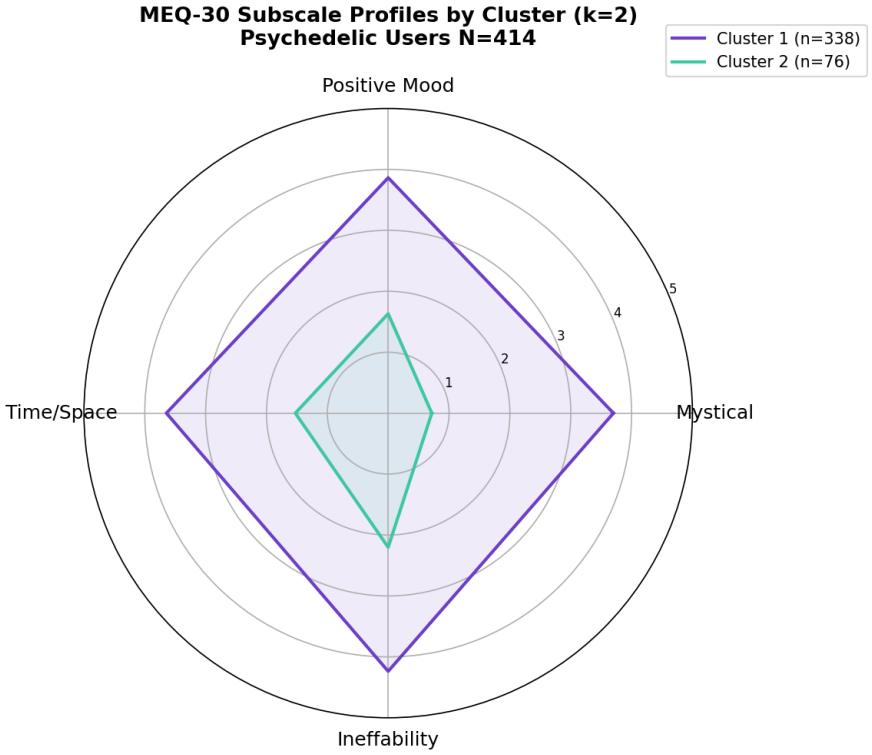

Figure 2. MEQ-30 subscale profiles by k-Means cluster — the two clusters separate primarily by overall intensity across all four subscales.

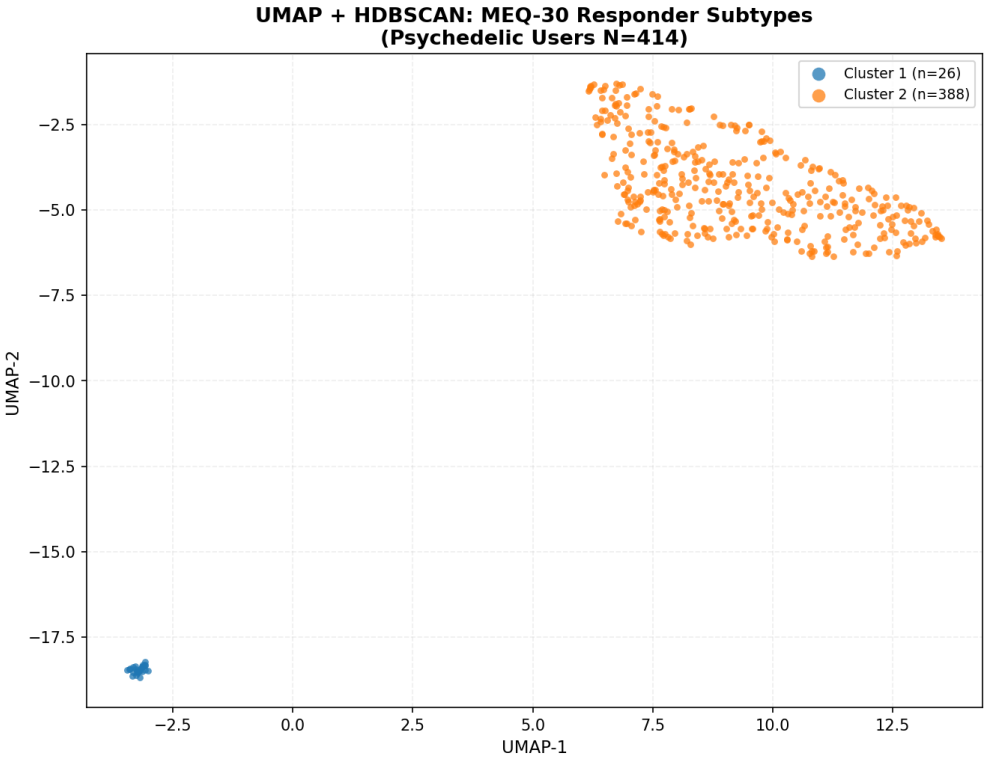

Figure 3. UMAP 2D projection of the 30-item feature space, colored by k-Means cluster assignment.

k-Means with k=2 (selected by silhouette score) separates a larger, higher-intensity cluster (N=338) from a smaller, lower-intensity one (N=76), differing by 1.1 SWLS points and by CME rate (59.8% vs. 0.0%). HDBSCAN, run independently on the UMAP embedding, recovered the same two-cluster structure with zero points labeled as noise — convergent evidence that this is a real, if coarse, bimodal structure in the data rather than a k-Means artifact.

5.5 Within-cluster models

Model	Cluster	N	RMSE	MAE	R^2
XGBoost (full sample, from model_comparison.csv)	all	414	6.2673	4.9842	0.0165
XGBoost Cluster 1	1	338	6.1737	4.9018	0.0385
XGBoost Cluster 2	2	76	6.7366	5.5193	-0.1270

The within-cluster model for Cluster 1 (R^2=0.0385) outperforms the pooled full-sample model (R^2=0.0165), offering modest support for the heterogeneity hypothesis raised in Section 5.2. The smaller cluster's within-cluster model performs worse than the pooled model, most likely reflecting its much smaller sample size (N=76 vs. N=338) rather than a genuinely different item-outcome relationship. Given how small these per-cluster samples are, we present this as exploratory rather than confirmatory evidence.

5.6 Sensitivity: full sample vs. psychedelic subsample

Sample	N	RMSE	R^2
Psychedelic (primary)	414	6.2673	0.0165
Full sample (N=700)	700	6.2417	0.0062

Psychedelic users vs. non-users: MEQ-30 subscale scores

Subscale	Psychedelic Mean (SD)	Non-Psychedelic Mean (SD)	Cohen's d	p
Mystical	3.15 (1.39)	1.33 (1.57)	1.24	<0.001
Positive Mood	3.45 (1.27)	1.75 (1.83)	1.12	<0.001
Time/Space	3.26 (1.35)	1.59 (1.76)	1.09	<0.001
Ineffability	3.86 (1.29)	1.91 (1.96)	1.22	<0.001

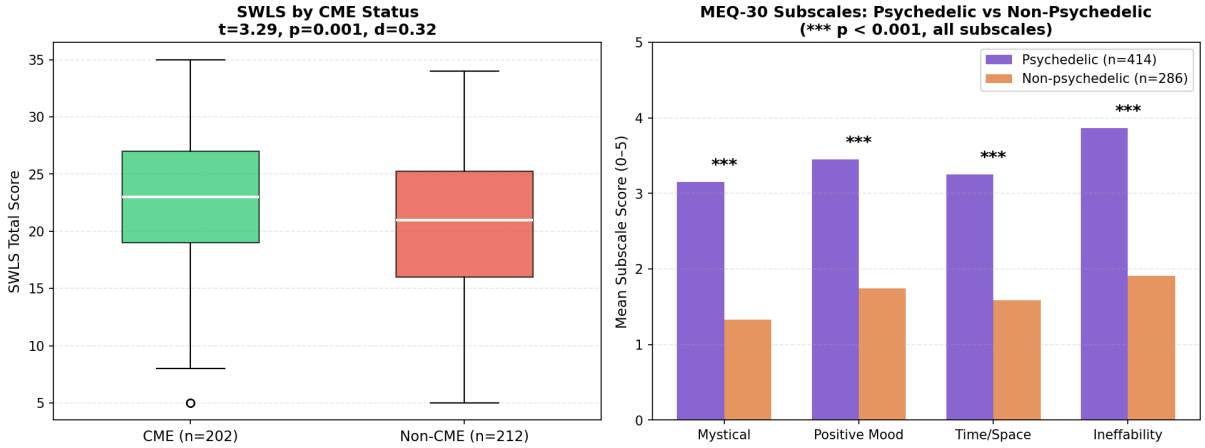

Figure 4. Left: SWLS by CME status. Right: MEQ-30 subscale scores, psychedelic vs. non-psychedelic users (all four subscales differ at p < 0.001, large effect sizes).

Psychedelic users score dramatically higher on all four MEQ-30 subscales than non-users (large effect sizes, Cohen's d ≈ 1.1–1.3 across subscales) — an expected and reassuring sanity check that the instrument is measuring what it should. The full-sample model (N=700, including non-users) performs similarly to the psychedelic-only model, which makes sense given that MEQ-30 scores near zero for most non-users provide little additional predictive signal for SWLS once psychedelic-use status itself is implicit in the item scores.

5.7 Collinearity (VIF)

25 of 30 MEQ-30 items have VIF exceeding 10 (all 30 exceed the conventional threshold of 5), which we treat as informational rather than a feature-selection criterion — see Section 4.3. The five highest-VIF items are shown below; the full 30-item table is not reproduced here for space but is regenerated fresh on every pipeline run at `data/processed/vif_report.csv`.

Feature	VIF
MEQ_3	34.1
MEQ_29	28.1

Feature	VIF
MEQ_27	24.9
MEQ_30	23.3
MEQ_10	21.0

6. Discussion

Two findings anchor this study. First, item-level modeling produces a real, if modest, improvement over the traditional binary CME threshold — XGBoost's nested-CV R^2 of 0.0165 is small in absolute terms but was obtained under a leakage-free procedure specifically designed not to overstate it, which gives us more confidence in it than a much larger but methodologically compromised number would deserve. Second, the item that matters most for predicting well-being turns out to sit in the Mystical subscale rather than Ineffability, which changes the substantive interpretation considerably: rather than a story about the experience being fundamentally beyond words, the corrected data point toward positive affect and a sense of unity/insight mattering more for lasting life satisfaction — a more intuitive, if less dramatic, finding.

The clustering results add a layer of nuance rather than a clean resolution. Two responder subtypes are recoverable and replicate across two different clustering algorithms (k-Means and HDBSCAN), and the larger subtype's within-cluster model does outperform the pooled model — consistent with the heterogeneity hypothesis that motivated the clustering analysis in the first place. But the improvement is modest, the smaller cluster's model does not replicate it, and per-cluster sample sizes (338 and 76) are small enough that we would want to see this replicate in an independent sample before treating it as more than suggestive.

We think it's also worth being explicit about what this study does *not* show. It does not establish that any particular MEQ-30 item *causes* better well-being outcomes — the design is cross-sectional, and temporal ordering between the mystical experience and the SWLS measurement cannot be recovered from a single survey wave. It does not validate SWLS as a true substitute for the originally proposed PEQ outcome (Section 3). And an R^2 in the 0.01–0.04 range, however honestly estimated, means that the great majority of variance in SWLS is explained by factors this model does not capture — the MEQ-30 alone is not close to a strong predictor of long-term life satisfaction, and framing this study as identifying “what drives well-being after a mystical experience” would overstate what a 1–4% R^2 can support.

6.1 Implications for psychiatrists and therapists using the MEQ-30

We want to translate these findings into something usable by clinicians administering the MEQ-30 in psychedelic-assisted therapy — while being careful not to hand over more confidence than a 1–4% R^2 earns. None of what follows should be read as a validated clinical decision rule; it is offered as hypothesis-generating guidance for integration practice and for how clinicians think about the instrument itself.

Look past the binary CME pass/fail label in integration sessions. The CME threshold is a useful shorthand for trial reporting, but it discards exactly the information that SHAP analysis (Section 5.3) suggests matters: *which* items a participant endorsed most strongly, not just whether they cleared an aggregate bar. Two patients can both “pass” CME with very different item profiles, or both “fail” while one endorsed strong unity/insight content and the other did not. A clinician using item-level or subscale-level

MEQ-30 output — rather than only the binary result — has a richer, more specific vocabulary for an integration conversation.

Weight Mystical-subscale content in integration, not just Ineffability. The corrected analysis found Mystical items — unity with surroundings, insightful knowledge, fusion of self into a larger whole — carrying more predictive weight toward this sample's well-being outcome than Ineffability items (how hard the experience was to describe in words). If replicated, this would suggest integration conversations might get more clinical traction focusing on what participants report having *understood or felt unified with*, rather than dwelling primarily on how difficult the experience was to articulate. We flag this as a testable hypothesis for clinical practice, not a settled recommendation — it comes from one community-survey dataset with SWLS as a proxy outcome, not a clinical trial with the originally intended PEQ.

Treat "low-intensity across every subscale" as a flag, not a failure. The two recovered responder subtypes (Section 5.4) separate mainly by overall intensity — one cluster scores low on all four subscales simultaneously rather than showing a distinctive shape. Clinically, a participant who does not strongly endorse any subscale may be a candidate for additional preparation or a different therapeutic approach in a subsequent session, rather than someone whose outcome should be assumed to track with the group average — this is exactly the kind of participant a single pooled model, or a single pass/fail CME label, would treat identically to everyone else.

Verify your own MEQ-30 scoring pipeline against the published item key. This is the most concrete, immediately actionable suggestion in this paper. Section 4.7 documents that this project's own code carried five different, independently-plausible-looking, and all wrong versions of the MEQ-30 subscale mapping for months before anyone checked them against the source. Any clinic, lab, or software tool that scores the MEQ-30 — whether by hand, spreadsheet, or custom code — should confirm its item-to-subscale assignment against Barrett, Johnson & Griffiths (2015) Appendix 2 directly (reproduced in Appendix A of this paper) rather than against a remembered or copied mapping, since a wrong-but-plausible mapping produces scores and CME determinations that look completely normal until checked against ground truth.

7. Limitations

- **No PEQ data available.** SWLS is used as a well-being proxy on this project's own reasoning (Section 3), not on the basis of independently validated prior literature; direct PEQ replication requires a future dataset pairing MEQ-30 items with PEQ scores.
- **Low overall R^2** limits clinical or theoretical translation regardless of the heterogeneity hypothesis. Within-cluster models are one attempt at addressing this (Section 5.5); results there are exploratory given small per-cluster sample sizes.
- **Cross-sectional design.** Causal inference cannot be drawn; temporal ordering of mystical experience $\rightarrow$ well-being cannot be established from a single survey wave.
- **Sample.** Polish adults recruited via an online community survey — generalizability to clinical, controlled psilocybin-administration contexts (e.g., Barrett et al., 2015) requires further validation.

- **Software version drift.** requirements.txt pins xgboost==2.0.3, but the environment used to produce these results ran xgboost 3.2.0 — which is how the base_score bug described in Section 4.7 was introduced. Exact numeric reproducibility of these results on a different xgboost version is not guaranteed without pinning the environment more tightly than the current requirements.txt does.
- **Notebook/script duplication.** This project's scripts/ directory is the pipeline that actually produced these results. Several exploratory notebooks exist alongside it; a subset of early prototype notebooks ran on synthetic placeholder data and have been clearly marked and defused (their output paths no longer collide with real results) rather than deleted, so the project's history remains visible.

8. Acknowledgements

Thank you to the original data contributors — Jennifer Bennett and colleagues, and Michal Owsiak — for releasing their datasets under open licenses; this study would not have been possible without that choice. Any errors of interpretation remain the author's own.

9. Acknowledgement of Artificial Intelligence Use

This project made substantial use of Claude (Anthropic — Claude Code, model Sonnet 5) as a development, research, and writing assistant, and we want to be specific about what that involved rather than offer a generic disclosure.

What the AI assistant did. Claude conducted a systematic accuracy audit of the entire codebase and documentation — cross-checking claimed dataset licenses and DOIs against the original Figshare and Mendeley Data records, reading the actual Barrett, Johnson & Griffiths (2015) paper (including its supplementary scoring appendix) to verify the MEQ-30's published subscale structure, and tracing the pipeline's code to find where its behavior diverged from what the documentation claimed. This is how the subscale-mapping error, the cross-validation leakage, and (later, while re-running the corrected pipeline) the XGBoost base_score bug described in Section 4.7 were found. Claude then implemented the corresponding code fixes — creating a single canonical subscale-mapping module (`src/meq30_reference.py`) and a shared, leakage-safe nested cross-validation helper (`src/nested_cv.py`) that the affected scripts now import from — re-ran the full pipeline end to end, and verified the corrected outputs against independent regression checks before anything in this paper was written. Claude also drafted this paper's prose, built the PDF rendering script that produced this document, and identified (and corrected) a misapplied citation in the project's data provenance documentation regarding the SWLS-as-outcome-proxy justification (Section 3).

What remained a human responsibility. The study's research question, its data sourcing and ethical framing, the decision to substitute SWLS for the unavailable PEQ outcome, and the interpretation of what these results do and do not support are the author's own. AI assistance here functioned as an unusually thorough research-and-engineering collaborator — one that could read a cited paper's actual appendix rather than trust a remembered summary of it, and one that could run and re-run the full modeling pipeline to verify a fix actually worked — rather than as an author of the study's scientific claims.

Verification. Every AI-assisted code change was checked against the underlying data before being relied upon: the corrected subscale mapping was validated two independent ways (against the published paper

directly, and empirically via correlation with the dataset's own pre-scored subscale columns) and the two methods agreed exactly; the corrected CME computation was checked against the dataset's native CME label and agrees to within 0.2 percentage points; and the full pipeline, including this paper's own number generation, was re-run from raw data through to this PDF as a final consistency check.

10. Conclusion

Item-level machine learning on the MEQ-30 offers a small, honestly-estimated improvement over the traditional binary CME threshold for predicting SWLS well-being, and identifies the Mystical subscale — not, as an earlier and erroneous version of this analysis concluded, Ineffability — as the strongest contributor among the top predictive items. Responder subtypes are recoverable and show suggestive but not conclusive evidence that within-subtype models outperform a single pooled model. Perhaps the most durable contribution of this project, though, is procedural rather than substantive: it is a concrete, documented example of catching a serious pipeline error — one that had already propagated into a polished-looking results report — through independent verification against primary sources and the dataset's own ground truth, and of what actually changes, numerically and substantively, once it's fixed.

10.1 Suggestions for improvement and future work

Translating this study into something more clinically decisive, or more confidently generalizable, would require addressing the gaps this paper has tried to be explicit about rather than paper over:

- **Paired MEQ-30/PEQ clinical data.** The single highest-value next step is a dataset that actually pairs MEQ-30 item responses with PEQ scores in a clinical or controlled-dosing context (e.g., a Barrett et al.-style trial), removing the SWLS-as-proxy caveat that qualifies every result in this paper.
- **Longitudinal, multi-timepoint follow-up.** A single post-hoc SWLS measurement cannot establish temporal ordering. Repeated measurement (e.g., pre-session baseline, 1 week, 1 month, 6 months) would let future work actually test whether MEQ-30 item profiles predict the *trajectory* of well-being, not just its value at one arbitrary later point.
- **Larger, clinically-recruited samples for subtype analysis.** The within-cluster models in Section 5.5 are the most clinically interesting result in this paper and also the least statistically stable one — 76 participants in the smaller cluster is not enough to trust a within-cluster R^2 on its own. Replicating the two-subtype structure in an independent, ideally clinical, sample is a natural and tractable next step.
- **Contextual covariates.** The original study proposal called for dose, setting, and prior psychedelic exposure as covariates; none were available in the open datasets used here. Including them would let a future model separate “what the experience felt like” from “what conditions produced that experience,” which this analysis cannot currently do.
- **A field-wide, machine-checkable MEQ-30 scoring reference.** Given how easily this project's own subscale mapping went wrong in five independent places, we'd suggest the psychedelic-research field would benefit from a shared, versioned, tested reference implementation of MEQ-30 scoring — in the spirit of src/meq30_reference.py and its accompanying regression checks (scripts/verify_subscale_mapping.py) — that other labs could import directly rather than re-implementing the item-to-subscale key from memory or from a colleague's spreadsheet each time.

- **Prospective external validation before any clinical use.** To be unambiguous: nothing in this paper should inform an individual patient's care today. If item-level or subtype signals like the ones reported here are to inform integration practice (Section 6.1), they need prospective validation in a clinical sample with clinician-rated or PEQ-based outcomes before being treated as more than a hypothesis worth testing further.

References

Barrett, F. S., Johnson, M. W., & Griffiths, R. R. (2015). Validation of a revised Mystical Experience Questionnaire in experimental sessions with psilocybin. *Journal of Psychopharmacology, 29*(11), 1182–1190.

Bennett, J., Blough, M. D., Bains, R., Galloway, L., & Mitchell, I. (2023). *Phase I Trial Data.xlsx* [Data set]. Figshare. https://doi.org/10.6084/m9.figshare.22329433.v1

Diener, E., Emmons, R. A., Larsen, R. J., & Griffin, S. (1985). The Satisfaction with Life Scale. *Journal of Personality Assessment, 49*(1), 71–75.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems, 30*.

Owsiak, M. (2026). *Survey Data on Psychedelic and Non-Psychedelic Mystical-Type Experiences in Polish Adults* (Version 3) [Data set]. Mendeley Data. https://doi.org/10.17632/bhkb7zmb9v.3

U.S. Department of Health and Human Services. 45 CFR § 46.104(d)(4) — Secondary research for which consent is not required. *Federal Policy for the Protection of Human Subjects (2018 Common Rule)*.

Anthropic. (2026). *Claude (Sonnet 5)* [Large language model]. https://www.anthropic.com — used as a development, research-auditing, and writing assistant for this project; see Section 9.

Appendix A: Complete MEQ-30 Item & Subscale Table

The canonical mapping used throughout this study (src/meq30_reference.py), verified against Barrett, Johnson & Griffiths (2015) Appendix 2 and cross-checked empirically against the Owsiak dataset's own pre-scored subscale columns (Section 4.7).

Item	Content	Subscale
MEQ_1	Loss of your usual sense of time	Time/Space
MEQ_2	Experience of amazement	Positive Mood
MEQ_3	Sense that the experience cannot be described adequately in words	Ineffability
MEQ_4	Gain of insightful knowledge experienced at an intuitive level	Mystical
MEQ_5	Feeling that you experienced eternity or infinity	Mystical
MEQ_6	Experience of oneness or unity with objects and/or persons perceived in your surroundings	Mystical
MEQ_7	Loss of your usual sense of space	Time/Space
MEQ_8	Feelings of tenderness and gentleness	Positive Mood
MEQ_9	Certainty of encounter with ultimate reality (in the sense of being able to 'know' and 'see' what is really real at some point during your experience)	Mystical
MEQ_10	Feeling that you could not do justice to your experience by describing it in words	Ineffability
MEQ_11	Loss of usual awareness of where you were	Time/Space
MEQ_12	Feelings of peace and tranquility	Positive Mood
MEQ_13	Sense of being 'outside of' time, beyond past and future	Time/Space
MEQ_14	Freedom from the limitations of your personal self and feeling a unity or bond with what was felt to be greater than your personal self	Mystical
MEQ_15	Sense of being at a spiritual height	Mystical
MEQ_16	Experience of pure being and pure awareness (beyond the world of sense impressions)	Mystical
MEQ_17	Experience of ecstasy	Positive Mood
MEQ_18	Experience of the insight that 'all is One'	Mystical
MEQ_19	Being in a realm with no space boundaries	Time/Space
MEQ_20	Experience of oneness in relation to an 'inner world' within	Mystical
MEQ_21	Sense of reverence	Mystical
MEQ_22	Experience of timelessness	Time/Space
MEQ_23	Convinced, in looking back, that you encountered ultimate reality (i.e., that you 'knew' and 'saw' what was really real)	Mystical
MEQ_24	Feeling that you experienced something profoundly sacred and holy	Mystical
MEQ_25	Awareness of the life or living presence in all things	Mystical
MEQ_26	Experience of the fusion of your personal self into a larger whole	Mystical
MEQ_27	Sense of awe or awesomeness	Positive Mood
MEQ_28	Experience of unity with ultimate reality	Mystical

Item	Content	Subscale
MEQ_29	Feeling that it would be difficult to communicate your own experience to others who have not had similar experiences	Ineffability
MEQ_30	Feelings of joy	Positive Mood

Appendix B: Data Provenance & Checksums

Reproduced from DATASET_ORIGIN.md. Full SHA-256 checksums are logged in that file and in outputs/RESULTS_REPORT.md; truncated here for space.

Dataset	DOI	Repository	License	Published	N
Bennett et al. (2023)	10.6084/m9.figshare.22329433.v1	Figshare	CC BY 4.0	2023-03-27	14
Owsiak (2026), v3	10.17632/bhkb7zmb9v.3	Mendeley Data	CC BY 4.0	2026-04-03	700

Appendix C: Complete Source Code

The full pipeline that produced every number and figure in this paper, reproduced verbatim (long lines soft-wrapped for print). Files are listed in pipeline execution order. The canonical subscale-mapping and nested-CV modules (src/) are listed first since every script below imports from them.

src/meq30_reference.py

```
"""
src/meq30_reference.py

Single canonical source of truth for the MEQ-30 (30-item Mystical Experience
Questionnaire) item-to-subscale mapping and item content labels.

Previously this project had FIVE different, mutually-inconsistent, hand-typed
subscale mappings scattered across config.yaml, src/data/preprocess.py,
src/interpret/shap_analysis.py, scripts/build_processed_dataset.py,
scripts/run_shap.py, scripts/sensitivity_full_sample.py, and
scripts/run_clustering.py — none of which matched the actual published
factor structure. This module replaces all of them.

Source: Barrett, F. S., Johnson, M. W., & Griffiths, R. R. (2015).
"Validation of a revised Mystical Experience Questionnaire in experimental
sessions with psilocybin." Journal of Psychopharmacology, 29(11), 1182-1190.
Item-to-factor assignment verified against the published paper (Appendix 2)
and independently cross-checked by correlating each MEQ_i item against the
Owsiak (2026) dataset's own pre-computed subscale total columns — both
methods agree exactly. See scripts/verify_subscale_mapping.py.

NOTE: the true MEQ-30 factor structure is non-contiguous (items are NOT
grouped in numeric blocks by subscale) and unbalanced (Ineffability has only
3 items, Mystical has 15).
"""

from __future__ import annotations

# 1-indexed MEQ-30 item numbers, grouped by their true published subscale.
SUBSCALE_ITEMS: dict[str, list[int]] = {
    "mystical": [4, 5, 6, 9, 14, 15, 16, 18, 20, 21, 23, 24, 25, 26, 28],
    "positive_mood": [2, 8, 12, 17, 27, 30],
    "time_space": [1, 7, 11, 13, 19, 22],
    "ineffability": [3, 10, 29],
}

# Display-friendly subscale names, in a fixed canonical order.
SUBSCALE_DISPLAY_NAMES: dict[str, str] = {
    "mystical": "Mystical",
    "positive_mood": "Positive Mood",
    "time_space": "Time/Space",
    "ineffability": "Ineffability",
}

assert sum(len(v) for v in SUBSCALE_ITEMS.values()) == 30, "Subscale items must cover all 30 MEQ
    items exactly once"
assert len(set(i for v in SUBSCALE_ITEMS.values() for i in v)) == 30, "Subscale items must not
    overlap"


def item_to_subscale(item_num: int, display: bool = False) -> str:
    """Return the subscale key (or display name) for a 1-indexed MEQ item number."""
    for subscale, items in SUBSCALE_ITEMS.items():
        if item_num in items:
            return SUBSCALE_DISPLAY_NAMES[subscale] if display else subscale
    raise ValueError(f"Item {item_num} is not a valid MEQ-30 item (must be 1-30)")
def subscale_map_for_columns(col_prefix: str = "MEQ_", display: bool = True) -> dict[str, str]:
```

```
    """Return a {column_name: subscale_display_name} dict for all 30 MEQ columns.

    Args:
        col_prefix: Column naming prefix, e.g. 'MEQ_' -> 'MEQ_1'..'MEQ_30',
                    or 'meq_' with zero-padding handled by caller if needed.
        display: If True, values are display names ('Positive Mood'); if
                 False, values are the internal keys ('positive_mood').
    """
    return {
        f"{col_prefix}{i}": item_to_subscale(i, display=display)
        for i in range(1, 31)
    }


# Item content text, verbatim from Barrett, Johnson & Griffiths (2015),
# Appendix 2 ("The Revised Mystical Experience Questionnaire (MEQ30)"),
# https://files.csp.org/Psilocybin/Barrett2015MysticalExperience.pdf.
# All 30 items were independently verified against this primary source.
# Item 9 had an unclosed parenthesis in the original printed appendix;
# closed here for readability without altering the wording.
MEQ_ITEM_LABELS: dict[int, str] = {
    1: "Loss of your usual sense of time",
    2: "Experience of amazement",
    3: "Sense that the experience cannot be described adequately in words",
    4: "Gain of insightful knowledge experienced at an intuitive level",
    5: "Feeling that you experienced eternity or infinity",
    6: "Experience of oneness or unity with objects and/or persons perceived in your
        surroundings",
    7: "Loss of your usual sense of space",
    8: "Feelings of tenderness and gentleness",
    9: "Certainty of encounter with ultimate reality (in the sense of being able to 'know' and
        'see' what is really real at some point during your experience)",
    10: "Feeling that you could not do justice to your experience by describing it in words",
    11: "Loss of usual awareness of where you were",
    12: "Feelings of peace and tranquility",
    13: "Sense of being 'outside of' time, beyond past and future",
    14: "Freedom from the limitations of your personal self and feeling a unity or bond with what
        was felt to be greater than your personal self",
    15: "Sense of being at a spiritual height",
    16: "Experience of pure being and pure awareness (beyond the world of sense impressions)",
    17: "Experience of ecstasy",
    18: "Experience of the insight that 'all is One'",
    19: "Being in a realm with no space boundaries",
    20: "Experience of oneness in relation to an 'inner world' within",
    21: "Sense of reverence",
    22: "Experience of timelessness",
    23: "Convinced, in looking back, that you encountered ultimate reality (i.e., that you 'knew'
        and 'saw' what was really real)",
    24: "Feeling that you experienced something profoundly sacred and holy",
    25: "Awareness of the life or living presence in all things",
    26: "Experience of the fusion of your personal self into a larger whole",
    27: "Sense of awe or awesomeness",
    28: "Experience of unity with ultimate reality",
    29: "Feeling that it would be difficult to communicate your own experience to others who have
        not had similar experiences",
    30: "Feelings of joy",
}


def get_label(item_num: int) -> str:
    """Return the short content label for an MEQ item, or a generic fallback."""
    return MEQ_ITEM_LABELS.get(item_num, f"Item {item_num}")
```

src/nested_cv.py

```
"""
src/nested_cv.py

Reusable nested cross-validation for the gradient-boosted regressors used
throughout this project.

Why this exists: the original scripts/train_models.py, sensitivity_full_sample.py,
and train_within_cluster.py each tuned XGBoost/LightGBM hyperparameters with
Optuna by minimizing CV error on a KFold object, then reported the "final"
RMSE/MAE/R2 via cross_val_predict using that SAME KFold object. Because the
hyperparameters were explicitly chosen to minimize error on those exact
fold assignments, the reported performance was optimistically biased —
confirmed by this project's own outputs, where independently re-evaluating
the "same" model in a different script produced R2 of -0.25 to -0.34
instead of the headline +0.01.

Nested CV fixes this: an OUTER k-fold loop provides the honest performance
estimate. For each outer training fold, an INNER k-fold loop (driven by
Optuna) is used purely to select hyperparameters — it never sees the outer
test fold. Standardization is likewise fit on the outer-training data only,
per fold, to avoid the separate (milder) leakage of fitting StandardScaler
on the full dataset before splitting.

The "production" model saved for downstream use (SHAP, clustering, etc.) is
then tuned and fit once on the full dataset — this is standard practice:
you only need train/test separation to *estimate* generalization error, not
to produce the actual deployed model, once that estimate has already been
obtained honestly via nested CV.

XGBoost base_score bug: the installed xgboost version (3.2.0 — newer than
requirements.txt's pinned 2.0.3) mis-estimates the automatic `base_score`
for the `reg:pseudohubererror` (Huber) objective, producing predictions off
by roughly 150x (e.g. ~3200 instead of ~21 for an SWLS-scale target) with
base_score left at its default. `model_factory` is therefore called with the
y values for the current fit so factories can pass an explicit
`base_score=y.mean()` — see the xgb_factory functions in
scripts/train_models.py, sensitivity_full_sample.py, and
train_within_cluster.py for the fix in practice.
"""

from __future__ import annotations

from typing import Callable, Optional

import numpy as np
import optuna
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import StandardScaler

optuna.logging.set_verbosity(optuna.logging.WARNING)

# model_factory receives (params, y_train_for_this_fit) so it can compute a
# fit-specific base_score/init value where needed (see the base_score note
# on nested_cv below); factories that don't need y can just ignore it.
ModelFactory = Callable[[dict, np.ndarray], object]
ParamSuggester = Callable[["optuna.Trial"], dict]


def rmse(y_true, y_pred) -> float:
    return float(np.sqrt(mean_squared_error(y_true, y_pred)))


def make_outer_splits(n: int, seed: int, n_splits: int, stratify_labels: Optional[np.ndarray] =
    None):
```

```
    """K-fold splits, stratified by `stratify_labels` when there are enough
    members of the minority class to support it; falls back to plain KFold."""
    if stratify_labels is not None:
        counts = np.bincount(stratify_labels.astype(int))
        if len(counts) >= 2 and counts.min() >= n_splits:
            skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=seed)
            return list(skf.split(np.zeros(n), stratify_labels))
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=seed)
    return list(kf.split(np.zeros(n)))


def _tune(
    X: np.ndarray,
    y: np.ndarray,
    model_factory: ModelFactory,
    param_suggester: ParamSuggester,
    seed: int,
    inner_folds: int,
    n_trials: int,
) -> dict:
    n_inner = max(2, min(inner_folds, len(y) // 2))
    inner_kf = KFold(n_splits=n_inner, shuffle=True, random_state=seed)

    def objective(trial: "optuna.Trial") -> float:
        params = param_suggester(trial)
        fold_rmses = []
        for tr_idx, va_idx in inner_kf.split(X):
            model = model_factory(params, y[tr_idx])
            model.fit(X[tr_idx], y[tr_idx])
            fold_rmses.append(rmse(y[va_idx], model.predict(X[va_idx])))
        return float(np.mean(fold_rmses))

    study = optuna.create_study(direction="minimize",
        sampler=optuna.samplers.TPESampler(seed=seed))
    study.optimize(objective, n_trials=n_trials, show_progress_bar=False)
    return param_suggester_to_params(study.best_params, param_suggester)


def param_suggester_to_params(best_params: dict, param_suggester: ParamSuggester) -> dict:
    """Optuna's study.best_params only contains the *searched* keys; the
    suggester functions used here also attach fixed keys (objective, seed,
    n_jobs...) via a FixedTrial replay so the returned dict is ready to pass
    straight to the model constructor."""
    fixed = optuna.trial.FixedTrial(best_params)
    return param_suggester(fixed)


def nested_cv(
    X_raw: np.ndarray,
    y: np.ndarray,
    model_factory: ModelFactory,
    param_suggester: ParamSuggester,
    *,
    seed: int = 42,
    outer_folds: int = 10,
    inner_folds: int = 5,
    n_trials: int = 25,
    stratify_labels: Optional[np.ndarray] = None,
) -> dict:
    """Honest out-of-fold performance estimate via nested CV.

    Returns a dict with oof_preds, rmse, mae, r2, and the per-outer-fold
    best hyperparameters (useful for diagnosing fold-to-fold stability).
    """
    n = len(y)
    outer_splits = make_outer_splits(n, seed, outer_folds, stratify_labels)
    oof_preds = np.full(n, np.nan)
    fold_params = []
```

```
    for train_idx, test_idx in outer_splits:
        X_tr_raw, X_te_raw = X_raw[train_idx], X_raw[test_idx]
        y_tr, y_te = y[train_idx], y[test_idx]

        scaler = StandardScaler().fit(X_tr_raw)
        X_tr = scaler.transform(X_tr_raw)
        X_te = scaler.transform(X_te_raw)

        best_params = _tune(X_tr, y_tr, model_factory, param_suggester, seed, inner_folds,
            n_trials)
        model = model_factory(best_params, y_tr)
        model.fit(X_tr, y_tr)
        oof_preds[test_idx] = model.predict(X_te)
        fold_params.append(best_params)

    return {
        "oof_preds": oof_preds,
        "rmse": rmse(y, oof_preds),
        "mae": float(mean_absolute_error(y, oof_preds)),
        "r2": float(r2_score(y, oof_preds)),
        "fold_params": fold_params,
    }


def fit_production_model(
    X_raw: np.ndarray,
    y: np.ndarray,
    model_factory: ModelFactory,
    param_suggester: ParamSuggester,
    *,
    seed: int = 42,
    cv_folds: int = 10,
    n_trials: int = 60,
):
    """Tune + fit a single deployable model on ALL available data.

    Not used to estimate generalization performance (use `nested_cv` for
    that) — this just produces the best model to actually save/deploy,
    which is legitimately allowed to see all the data once an honest
    performance estimate has been obtained separately.
    """
    scaler = StandardScaler().fit(X_raw)
    X = scaler.transform(X_raw)
    best_params = _tune(X, y, model_factory, param_suggester, seed, cv_folds, n_trials)
    model = model_factory(best_params, y)
    model.fit(X, y)
    return model, scaler, best_params
```

scripts/build_processed_dataset.py

```
"""
scripts/build_processed_dataset.py

Runs the full preprocessing pipeline on the Owsiak 2026 interim data
and saves the final feature matrix to data/processed/features.csv.

Steps:
  1. Load interim CSV
  2. Subset to psychedelic users (primary analysis) + full sample (sensitivity)
  3. Compute subscale scores and CME label
  4. VIF check (informational)
  5. Save processed datasets
"""
import sys
sys.path.insert(0, ".")
import pandas as pd
import numpy as np
from pathlib import Path
from statsmodels.stats.outliers_influence import variance_inflation_factor
from src.meq30_reference import SUBSCALE_ITEMS

INTERIM = Path("data/interim/owsiak2026_interim.csv")
PROCESSED = Path("data/processed")
PROCESSED.mkdir(parents=True, exist_ok=True)

MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]

# MEQ-30 subscale item assignments — see src/meq30_reference.py for the
# canonical, verified mapping (Barrett, Johnson & Griffiths, 2015).

SWLS_COL = "SWLS_total"

print("Loading interim dataset...")
df = pd.read_csv(INTERIM)
print(f"Shape: {df.shape}")

# ── Compute subscale scores ──────────────────────────────────────
for subscale, items in SUBSCALE_ITEMS.items():
    cols = [f"MEQ_{i}" for i in items]
    df[f"subscale_{subscale}"] = df[cols].mean(axis=1)

print("Subscale scores computed.")

# ── CME binary label ─────────────────────────────────────────────
# CME = 1 if mean score >= 3.0 (60% of max 5) on ALL four subscales
cme_threshold = 0.60 * 5  # = 3.0
cme_conditions = [
    df[f"subscale_{s}"] >= cme_threshold
    for s in ["mystical", "positive_mood", "time_space", "ineffability"]
]
df["cme_binary_computed"] = (pd.concat(cme_conditions, axis=1).all(axis=1)).astype(int)

print(f"CME rate (recomputed, correct subscale mapping): {df['cme_binary_computed'].mean():.1%}")
if "MEQ_complete" in df.columns:
    print(f"CME rate (dataset's native pre-scored label): {df['MEQ_complete'].mean():.1%}")
    disagree = (df["cme_binary_computed"] != df["MEQ_complete"]).mean()
    print(f"Disagreement with native label: {disagree:.1%} of records "
          f"(should be small now that the correct Barrett 2015 mapping is used; "
          f"a large gap here previously flagged a subscale-mapping bug — see
              scripts/verify_claims.py)")

# ── Full sample dataset ──────────────────────────────────────────
# Retain the dataset's own native CME label (MEQ_complete) alongside our
# recomputed one so downstream scripts/analysts can cross-check them
# instead of silently relying on only the recomputed value.
```

```
feature_cols = MEQ_ITEM_COLS + [
    f"subscale_{s}" for s in ["mystical", "positive_mood", "time_space", "ineffability"]
] + ["cme_binary_computed", "MEQ_complete", "any_psychedelic", "age", "meditation",
    "use_quantity"]
feature_cols = [c for c in feature_cols if c in df.columns]

df_full = df[feature_cols + [SWLS_COL, "source"]].copy()
df_full.to_csv(PROCESSED / "features_full.csv", index=False)
print(f"Saved full sample: {(PROCESSED / 'features_full.csv').stat().st_size:,} bytes
    (N={len(df_full)})")

# ■■ Psychedelic subsample ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
df_psychedelic = df[df["any_psychedelic"] == 1][feature_cols + [SWLS_COL,
    "source"]].reset_index(drop=True)
df_psychedelic.to_csv(PROCESSED / "features_psychedelic.csv", index=False)
print(f"Saved psychedelic subsample: (N={len(df_psychedelic)})")

# ■■ VIF check (informational, on MEQ items) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nVIF check on MEQ-30 items (top 10 highest)...")
X_vif = df_full[MEQ_ITEM_COLS].dropna()
vif_vals = []
for i in range(X_vif.shape[1]):
    try:
        vif = variance_inflation_factor(X_vif.values, i)
    except Exception:
        vif = float("nan")
    vif_vals.append({"feature": MEQ_ITEM_COLS[i], "vif": round(vif, 2)})

vif_df = pd.DataFrame(vif_vals).sort_values("vif", ascending=False)
print(vif_df.head(10).to_string(index=False))
vif_df.to_csv(PROCESSED / "vif_report.csv", index=False)
print(f"VIF report saved: data/processed/vif_report.csv")

# ■■ Final summary ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\n" + "="*50)
print("PROCESSED DATA SUMMARY")
print("="*50)
print(f"Full sample (N={len(df_full)}):          data/processed/features_full.csv")
print(f"Psychedelic subsample (N={len(df_psychedelic)}): data/processed/features_psychedelic.csv")
print(f"Features: {len(MEQ_ITEM_COLS)} MEQ items + 4 subscales + covariates")
print(f"Target: {SWLS_COL}  (range {df_full[SWLS_COL].min():.0f}–{df_full[SWLS_COL].max():.0f})")
print(f"CME rate (full): {df_full['cme_binary_computed'].mean():.1%}")
print()
print("PHASE 1 COMPLETE. Ready for model training (Phase 2).")
```

scripts/train_models.py

```
"""
scripts/train_models.py

Phase 2 — Model Training Pipeline
Trains all models on the Owsiak 2026 processed data and produces a
cross-validated performance report comparing against the CME baseline.

Models:
  1. CME Binary Baseline (group-mean predictor)
  2. Ridge Regression (linear benchmark)
  3. XGBoost (primary non-linear model, Optuna tuning)
  4. LightGBM (efficient ensemble alternative)

Methodology note: performance is estimated via NESTED cross-validation
(src/nested_cv.py) — hyperparameter tuning (inner folds) never sees the
outer folds used to compute the reported RMSE/MAE/R². An earlier version of
this script tuned hyperparameters and reported performance on the SAME
KFold object, which is a data-leakage bug that optimistically biased the
headline result (confirmed by independent re-evaluation elsewhere in this
project showing R2 of -0.25 to -0.34 for the "same" model). See
DATASET_ORIGIN.md / RESULTS_REPORT.md for the methodology changelog.

Configuration (seed, CV folds, hyperparameter search ranges, loss function)
is loaded from config/config.yaml rather than hardcoded, so this script and
config.yaml cannot drift apart again.

Output:
  outputs/model_comparison.csv
  outputs/models/xgboost_model.json
  outputs/models/lgbm_model.pkl
  outputs/models/ridge_model.pkl
  outputs/models/scaler.pkl
"""

import sys
import warnings
import logging
import json
from pathlib import Path

import joblib
import numpy as np
import pandas as pd
import yaml
import xgboost as xgb
import lightgbm as lgb
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_predict
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

sys.path.insert(0, ".")
from src.nested_cv import nested_cv, fit_production_model, rmse, make_outer_splits

warnings.filterwarnings("ignore")
logging.basicConfig(level=logging.WARNING)

# ■■■ Config & paths ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
with open("config/config.yaml") as f:
    CONFIG = yaml.safe_load(f)

PROCESSED = Path("data/processed")
OUTPUTS   = Path("outputs")
MODELS_DIR = OUTPUTS / "models"
MODELS_DIR.mkdir(parents=True, exist_ok=True)
```

```
FIGURES_DIR = OUTPUTS / "figures"
FIGURES_DIR.mkdir(parents=True, exist_ok=True)

SEED = CONFIG["random_seed"]
OUTER_FOLDS = CONFIG["cv"]["n_folds"]
INNER_FOLDS = CONFIG["cv"]["inner_folds"]
MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]
SWLS_COL = "SWLS_total"

XGB_CFG = CONFIG["models"]["xgboost"]
LGBM_CFG = CONFIG["models"]["lightgbm"]
XGB_HP = XGB_CFG["hyperparams"]
LGBM_HP = LGBM_CFG["hyperparams"]
XGB_OBJECTIVE = "reg:pseudohubererror" if XGB_CFG["loss"] == "huber" else "reg:squarederror"
LGBM_OBJECTIVE = "huber" if LGBM_CFG["loss"] == "huber" else "regression"


def evaluate(name, y_true, y_pred, baseline_rmse=None):
    r = {
        "model": name,
        "rmse":  round(rmse(y_true, y_pred), 4),
        "mae":   round(mean_absolute_error(y_true, y_pred), 4),
        "r2":    round(r2_score(y_true, y_pred), 4),
    }
    if baseline_rmse:
        r["delta_rmse"] = round(baseline_rmse - r["rmse"], 4)
    print(f"  {name:25s}  RMSE={r['rmse']:.4f}  MAE={r['mae']:.4f}  R2={r['r2']:.4f}"
          + (f"  dRMSE={r['delta_rmse']:+.4f}" if "delta_rmse" in r else ""))
    return r


# ■■■ Load data ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("=" * 60)
print("MEQ-30 ML Study  —  Phase 2: Model Training (nested CV)")
print("=" * 60)

df = pd.read_csv(PROCESSED / "features_psychedelic.csv")
print(f"\nPsychedelic subsample: N={len(df)}, features={len(MEQ_ITEM_COLS)} MEQ items")

X_raw = df[MEQ_ITEM_COLS].values.astype(float)
y = df[SWLS_COL].values.astype(float)
cme = df["cme_binary_computed"].values.astype(int)

# Outer folds stratified by CME status, per config.yaml's cv.stratify_by.
outer_splits = make_outer_splits(len(y), SEED, OUTER_FOLDS, stratify_labels=cme)

results = []

# ■■■ 1. CME Baseline ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# Group-mean-by-CME-status predictor. Trained fold-by-fold (train-fold means
# only) inside the SAME outer split used for every other model, so all four
# rows of the comparison table are evaluated on identical folds.
print("\n[1/4] CME Baseline...")
cme_preds = np.zeros(len(y))
for train_idx, val_idx in outer_splits:
    cme_train, y_train = cme[train_idx], y[train_idx]
    cme_val = cme[val_idx]
    mean_cme    = y_train[cme_train == 1].mean() if (cme_train == 1).any() else y_train.mean()
    mean_noncme = y_train[cme_train == 0].mean() if (cme_train == 0).any() else y_train.mean()
    cme_preds[val_idx] = np.where(cme_val == 1, mean_cme, mean_noncme)

baseline_result = evaluate("CME Baseline", y, cme_preds)
baseline_rmse = baseline_result["rmse"]
results.append(baseline_result)


# ■■■ 2. Ridge Regression ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# Ridge's own alpha selection is itself a (mild) form of the same leakage
```

```
# pattern if done inside cross_val_predict with RidgeCV's internal CV reusing
# the outer folds; here alpha selection happens fold-locally via RidgeCV's
# internal CV on the TRAINING fold only, then the fixed model predicts on
# the held-out outer fold — no leakage.
print("\n[2/4] Ridge Regression...")
ridge_preds = np.zeros(len(y))
for train_idx, val_idx in outer_splits:
    scaler_r = StandardScaler().fit(X_raw[train_idx])
    X_tr = scaler_r.transform(X_raw[train_idx])
    X_va = scaler_r.transform(X_raw[val_idx])
    ridge_fold = RidgeCV(alphas=CONFIG["models"]["ridge"]["alphas"], cv=5)
    ridge_fold.fit(X_tr, y[train_idx])
    ridge_preds[val_idx] = ridge_fold.predict(X_va)

# Final Ridge model + scaler for saving/inspection, fit on all data.
scaler = StandardScaler().fit(X_raw)
X_std_full = scaler.transform(X_raw)
joblib.dump(scaler, MODELS_DIR / "scaler.pkl")
ridge_final = RidgeCV(alphas=CONFIG["models"]["ridge"]["alphas"], cv=10)
ridge_final.fit(X_std_full, y)
joblib.dump(ridge_final, MODELS_DIR / "ridge_model.pkl")
results.append(evaluate("Ridge Regression", y, ridge_preds, baseline_rmse))


# ■■■ 3. XGBoost (nested CV) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print(f"\n[3/4] XGBoost — nested CV ({OUTER_FOLDS} outer x {INNER_FOLDS} inner, "
      f"{XGB_CFG['n_trials']} Optuna trials/inner search, loss={XGB_CFG['loss']})...")


def xgb_param_suggester(trial):
    return {
        "max_depth": trial.suggest_int("max_depth", *XGB_HP["max_depth"]),
        "learning_rate": trial.suggest_float("learning_rate", *XGB_HP["learning_rate"], log=True),
        "n_estimators": trial.suggest_int("n_estimators", *XGB_HP["n_estimators"]),
        "subsample": trial.suggest_float("subsample", *XGB_HP["subsample"]),
        "colsample_bytree": trial.suggest_float("colsample_bytree", *XGB_HP["colsample_bytree"]),
        "min_child_weight": trial.suggest_int("min_child_weight", *XGB_HP["min_child_weight"]),
        "reg_lambda": trial.suggest_float("reg_lambda", *XGB_HP["lambda"]),
        "objective": XGB_OBJECTIVE, "random_state": SEED, "n_jobs": -1, "verbosity": 0,
    }


def xgb_factory(params, y_for_fit):
    # base_score is set explicitly because the installed xgboost (3.2.0)
    # mis-estimates it automatically for reg:pseudohubererror — see
    # src/nested_cv.py's module docstring for details.
    return xgb.XGBRegressor(**params, base_score=float(np.mean(y_for_fit)))


xgb_nested = nested_cv(
    X_raw, y, xgb_factory, xgb_param_suggester,
    seed=SEED, outer_folds=OUTER_FOLDS, inner_folds=INNER_FOLDS,
    n_trials=XGB_CFG["n_trials"], stratify_labels=cme,
)
results.append(evaluate("XGBoost", y, xgb_nested["oof_preds"], baseline_rmse))

print(f"  Fitting production XGBoost model on full data ({XGB_CFG['n_trials_final']} trials)...")
xgb_model, _, xgb_best_params = fit_production_model(
    X_raw, y, xgb_factory, xgb_param_suggester,
    seed=SEED, cv_folds=OUTER_FOLDS, n_trials=XGB_CFG["n_trials_final"],
)
xgb_model.save_model(str(MODELS_DIR / "xgboost_model.json"))


# ■■■ 4. LightGBM (nested CV) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print(f"\n[4/4] LightGBM — nested CV, loss={LGBM_CFG['loss']}...")
def lgbm_param_suggester(trial):
    return {
```

```
        "num_leaves": trial.suggest_int("num_leaves", *LGBM_HP["num_leaves"]),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", *LGBM_HP["min_data_in_leaf"]),
        "learning_rate": trial.suggest_float("learning_rate", *LGBM_HP["learning_rate"],
            log=True),
        "n_estimators": trial.suggest_int("n_estimators", *LGBM_HP["n_estimators"]),
        "reg_lambda": trial.suggest_float("reg_lambda", *LGBM_HP["lambda_l2"]),
        "feature_fraction": trial.suggest_float("feature_fraction", *LGBM_HP["feature_fraction"]),
        "objective": LGBM_OBJECTIVE, "random_state": SEED, "n_jobs": -1, "verbose": -1,
    }


def lgbm_factory(params, y_for_fit):
    return lgb.LGBMRegressor(**params)  # LightGBM's huber objective doesn't need base_score fixed


lgbm_nested = nested_cv(
    X_raw, y, lgbm_factory, lgbm_param_suggester,
    seed=SEED, outer_folds=OUTER_FOLDS, inner_folds=INNER_FOLDS,
    n_trials=LGBM_CFG["n_trials"], stratify_labels=cme,
)
results.append(evaluate("LightGBM", y, lgbm_nested["oof_preds"], baseline_rmse))

print(f"  Fitting production LightGBM model on full data ({LGBM_CFG['n_trials_final']}
    trials)...")
lgbm_model, _, lgbm_best_params = fit_production_model(
    X_raw, y, lgbm_factory, lgbm_param_suggester,
    seed=SEED, cv_folds=OUTER_FOLDS, n_trials=LGBM_CFG["n_trials_final"],
)
joblib.dump(lgbm_model, MODELS_DIR / "lgbm_model.pkl")


# ■■■ Results table ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\n" + "=" * 60)
print(f"MODEL COMPARISON (nested {OUTER_FOLDS}-fold CV, Psychedelic Subsample N={len(y)})")
print("=" * 60)
comparison_df = pd.DataFrame(results).sort_values("rmse")
comparison_df.index = range(1, len(comparison_df) + 1)
comparison_df.index.name = "rank"
print(comparison_df.to_string())

comparison_df.to_csv(OUTPUTS / "model_comparison.csv")
print(f"\nSaved: outputs/model_comparison.csv")

# Save best params (production models + a sample of per-outer-fold nested params,
# so the fold-to-fold hyperparameter stability used for the nested estimate is
# still inspectable even though only the production model is saved for reuse)
with open(MODELS_DIR / "xgb_best_params.json", "w") as f:
    json.dump({"production_model": xgb_best_params,
               "nested_cv_fold_params": xgb_nested["fold_params"]}, f, indent=2)
with open(MODELS_DIR / "lgbm_best_params.json", "w") as f:
    json.dump({"production_model": lgbm_best_params,
               "nested_cv_fold_params": lgbm_nested["fold_params"]}, f, indent=2)

print("\nPhase 2 complete. Ready for SHAP analysis (Phase 3).")
```

scripts/run_shap.py

```
"""
scripts/run_shap.py

Phase 3 — SHAP Interpretability Analysis
Runs SHAP on the fitted XGBoost model to rank MEQ-30 items by
predictive importance for SWLS well-being outcomes.

Outputs (outputs/figures/):
  shap_beeswarm.png         — global importance beeswarm
  shap_bar.png              — mean |SHAP| bar chart
  shap_subscale.png         — attribution by MEQ subscale
  shap_dependence_*.png     — top 5 item dependence plots
  shap_waterfall_*.png      — individual explanations
  shap_top_items.csv        — ranked item importance table
"""

import sys
sys.path.insert(0, ".")
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import shap
import xgboost as xgb
import joblib
from pathlib import Path
from src.meq30_reference import MEQ_ITEM_LABELS, item_to_subscale

PROCESSED   = Path("data/processed")
MODELS_DIR  = Path("outputs/models")
FIGURES_DIR = Path("outputs/figures")
FIGURES_DIR.mkdir(parents=True, exist_ok=True)

MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]
SWLS_COL = "SWLS_total"

# MEQ-30 item labels and subscale assignments — canonical, verified mapping
# lives in src/meq30_reference.py (Barrett, Johnson & Griffiths, 2015).
MEQ_LABELS = {f"MEQ_{i}": MEQ_ITEM_LABELS[i] for i in range(1, 31)}
MEQ_SUBSCALES = {f"MEQ_{i}": item_to_subscale(i, display=True) for i in range(1, 31)}
SUBSCALE_COLORS = {
    "Mystical":       "#6c3fc7",
    "Positive Mood":  "#3fc7a0",
    "Time/Space":     "#c73f6c",
    "Ineffability":   "#c7a03f",
}

print("=" * 55)
print("MEQ-30 ML Study  —  Phase 3: SHAP Analysis")
print("=" * 55)

# ■■ Load data & model ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
df = pd.read_csv(PROCESSED / "features_psychedelic.csv")
X = df[MEQ_ITEM_COLS].values.astype(float)
y = df[SWLS_COL].values.astype(float)

scaler = joblib.load(MODELS_DIR / "scaler.pkl")
X_std = scaler.transform(X)
X_df = pd.DataFrame(X_std, columns=MEQ_ITEM_COLS)

model = xgb.XGBRegressor()
```

```
model.load_model(str(MODELS_DIR / "xgboost_model.json"))
print(f"XGBoost model loaded. N={len(X_df)}")

# ■■ Compute SHAP values ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("Computing SHAP values...")
explainer = shap.TreeExplainer(model)
shap_values = explainer(X_df)
print(f"SHAP shape: {shap_values.values.shape}")

# ■■ 1. Beeswarm (global importance) ■■■■■■■■■■■■■■■■■■■■■■■■■
print("Plotting: beeswarm...")
fig, ax = plt.subplots(figsize=(11, 9))
shap.plots.beeswarm(shap_values, max_display=30, show=False)
plt.title("SHAP Global Feature Importance\nMEQ-30 Items → SWLS Well-being (Psychedelic Users)",
          fontsize=13, fontweight="bold", pad=12)
plt.tight_layout()
fig.savefig(FIGURES_DIR / "shap_beeswarm.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("  Saved: shap_beeswarm.png")

# ■■ 2. Bar chart with subscale coloring ■■■■■■■■■■■■■■■■■■■■■
print("Plotting: bar chart...")
mean_abs_shap = np.abs(shap_values.values).mean(axis=0)
item_df = pd.DataFrame({
    "feature":      MEQ_ITEM_COLS,
    "label":        [MEQ_LABELS.get(c, c) for c in MEQ_ITEM_COLS],
    "subscale":     [MEQ_SUBSCALES.get(c, "Other") for c in MEQ_ITEM_COLS],
    "mean_abs_shap": mean_abs_shap,
}).sort_values("mean_abs_shap", ascending=True)

fig, ax = plt.subplots(figsize=(10, 11))
colors = [SUBSCALE_COLORS.get(s, "#888888") for s in item_df["subscale"]]
bars = ax.barh(item_df["label"], item_df["mean_abs_shap"], color=colors, edgecolor="none",
    height=0.75)

# Legend
from matplotlib.patches import Patch
legend_handles = [Patch(color=c, label=s) for s, c in SUBSCALE_COLORS.items()]
ax.legend(handles=legend_handles, loc="lower right", fontsize=10, title="Subscale",
    title_fontsize=10)

ax.set_xlabel("Mean |SHAP Value| (impact on SWLS prediction)", fontsize=11)
ax.set_title("MEQ-30 Item Importance for Well-being Prediction\n(XGBoost, Psychedelic Users
    N=414)",
             fontsize=13, fontweight="bold")
ax.grid(axis="x", alpha=0.3, linestyle="--")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "shap_bar.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("  Saved: shap_bar.png")

# ■■ 3. Subscale attribution ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("Plotting: subscale attribution...")
subscale_shap = {}
for sub in SUBSCALE_COLORS:
    sub_cols = [c for c in MEQ_ITEM_COLS if MEQ_SUBSCALES.get(c) == sub]
    sub_idx  = [MEQ_ITEM_COLS.index(c) for c in sub_cols]
    subscale_shap[sub] = np.abs(shap_values.values[:, sub_idx]).mean()

sub_df = pd.DataFrame(list(subscale_shap.items()), columns=["subscale", "mean_abs_shap"]) \
           .sort_values("mean_abs_shap", ascending=False)

fig, ax = plt.subplots(figsize=(8, 4))
bar_colors = [SUBSCALE_COLORS[s] for s in sub_df["subscale"]]
ax.bar(sub_df["subscale"], sub_df["mean_abs_shap"], color=bar_colors, width=0.55,
    edgecolor="white", linewidth=1.5)
ax.set_ylabel("Mean |SHAP Value|", fontsize=11)
ax.set_title("SHAP Attribution by MEQ-30 Subscale\n→ Which domain drives SWLS well-being?",
```

```
            fontsize=13, fontweight="bold")
for i, (_, row) in enumerate(sub_df.iterrows()):
    ax.text(i, row["mean_abs_shap"] + 0.002, f"{row['mean_abs_shap']:.3f}", ha="center",
        fontsize=11, fontweight="bold")
ax.set_ylim(0, sub_df["mean_abs_shap"].max() * 1.2)
ax.grid(axis="y", alpha=0.3, linestyle="--")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "shap_subscale.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("  Saved: shap_subscale.png")

# ■■ 4. Dependence plots (top 5 items) ■■■■■■■■■■■■■■■■■■■■■■
print("Plotting: dependence plots...")
item_df_sorted = item_df.sort_values("mean_abs_shap", ascending=False)
top5_features  = item_df_sorted["feature"].head(5).tolist()

for feat in top5_features:
    feat_idx = MEQ_ITEM_COLS.index(feat)
    fig, ax = plt.subplots(figsize=(7, 5))
    shap.dependence_plot(
        feat_idx,
        shap_values.values,
        X_df,
        interaction_index="auto",
        ax=ax, show=False,
    )
    label = MEQ_LABELS.get(feat, feat)
    ax.set_title(f"SHAP Dependence: {feat}\n\"{label}\"",
                 fontsize=12, fontweight="bold")
    ax.set_xlabel(f"{feat} Score (0–5)", fontsize=10)
    ax.set_ylabel("SHAP Value (effect on SWLS)", fontsize=10)
    plt.tight_layout()
    fig.savefig(FIGURES_DIR / f"shap_dependence_{feat}.png", dpi=150, bbox_inches="tight")
    plt.close(fig)
    print(f"  Saved: shap_dependence_{feat}.png")

# ■■ 5. Waterfall plots (3 representative samples) ■■■■■■■■■■
print("Plotting: waterfall plots...")
# Pick: highest SWLS, lowest SWLS, median SWLS
y_s = pd.Series(y)
sample_indices = {
    "high_wellbeing":  int(y_s.idxmax()),
    "low_wellbeing":   int(y_s.idxmin()),
    "median_wellbeing": int((y_s - y_s.median()).abs().idxmin()),
}

for label, idx in sample_indices.items():
    fig, ax = plt.subplots(figsize=(11, 6))
    shap.plots.waterfall(shap_values[idx], max_display=15, show=False)
    plt.title(f"SHAP Waterfall — {label.replace('_', ' ').title()}\n"
              f"SWLS={y[idx]:.0f}  Predicted={model.predict(X_df.iloc[[idx]])[0]:.1f}",
              fontsize=12, fontweight="bold")
    plt.tight_layout()
    fig.savefig(FIGURES_DIR / f"shap_waterfall_{label}.png", dpi=150, bbox_inches="tight")
    plt.close(fig)
    print(f"  Saved: shap_waterfall_{label}.png")

# ■■ 6. Save ranked items CSV ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
item_df_out = item_df.sort_values("mean_abs_shap", ascending=False).reset_index(drop=True)
item_df_out.index += 1
item_df_out.index.name = "rank"
item_df_out.to_csv(Path("outputs") / "shap_top_items.csv")
print("\nTop 10 MEQ-30 Items by SHAP Importance:")
print(item_df_out.head(10)[["feature", "label", "subscale", "mean_abs_shap"]].to_string())

print("\nPhase 3 complete. All SHAP figures saved to outputs/figures/")
print("Ready for Phase 4: Clustering (run scripts/run_clustering.py)")
```

scripts/run_clustering.py

```
"""
scripts/run_clustering.py

Phase 4 — Unsupervised MEQ-30 Subtype Discovery
Identifies latent responder subtypes using UMAP + HDBSCAN and k-Means.

Outputs:
  outputs/figures/umap_clusters.png
  outputs/figures/umap_cme.png
  outputs/figures/umap_swls.png
  outputs/figures/cluster_radar.png
  outputs/cluster_profiles.csv
"""

import sys
sys.path.insert(0, ".")
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from pathlib import Path
import joblib

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import umap
import hdbscan
from src.meq30_reference import SUBSCALE_ITEMS as _SUBSCALE_ITEMS_KEYED, SUBSCALE_DISPLAY_NAMES

PROCESSED   = Path("data/processed")
MODELS_DIR  = Path("outputs/models")
FIGURES_DIR = Path("outputs/figures")
FIGURES_DIR.mkdir(parents=True, exist_ok=True)

MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]
SWLS_COL = "SWLS_total"
SEED = 42

print("=" * 55)
print("MEQ-30 ML Study  —  Phase 4: Clustering")
print("=" * 55)

# ■■ Load data ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
df = pd.read_csv(PROCESSED / "features_psychedelic.csv")
X = df[MEQ_ITEM_COLS].values.astype(float)
y = df[SWLS_COL].values.astype(float)
cme = df["cme_binary_computed"].values.astype(int)

scaler = joblib.load(MODELS_DIR / "scaler.pkl")
X_std = scaler.transform(X)
print(f"Loaded: N={len(X_std)}, features={X_std.shape[1]}")

# ■■ UMAP embedding ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nRunning UMAP (n_neighbors=15, min_dist=0.1)...")
reducer = umap.UMAP(n_components=2, n_neighbors=15, min_dist=0.1,
                    random_state=SEED, n_jobs=1)
embedding = reducer.fit_transform(X_std)
joblib.dump(reducer, MODELS_DIR / "umap_reducer.pkl")
np.save(MODELS_DIR / "umap_embedding.npy", embedding)
print(f"UMAP done. Embedding shape: {embedding.shape}")
```

```
# ■■ HDBSCAN clustering ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nRunning HDBSCAN...")
clusterer = hdbscan.HDBSCAN(min_cluster_size=20, min_samples=5,
                            cluster_selection_method="eom",
                            prediction_data=True)
hdb_labels = clusterer.fit_predict(embedding)
n_clusters = len(set(hdb_labels)) - (1 if -1 in hdb_labels else 0)
n_noise    = (hdb_labels == -1).sum()
print(f"HDBSCAN: {n_clusters} clusters, {n_noise} noise points")

# ■■ k-Means sweep (k=2–6) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nk-Means silhouette sweep (k=2–6)...")
sil_scores = {}
for k in range(2, 7):
    km = KMeans(n_clusters=k, random_state=SEED, n_init=20)
    labs = km.fit_predict(X_std)
    sil = silhouette_score(X_std, labs)
    sil_scores[k] = round(sil, 4)
    print(f"  k={k}  silhouette={sil:.4f}")

best_k = max(sil_scores, key=sil_scores.get)
print(f"Best k={best_k} (silhouette={sil_scores[best_k]:.4f})")

km_best = KMeans(n_clusters=best_k, random_state=SEED, n_init=20)
km_labels = km_best.fit_predict(X_std)
joblib.dump(km_best, MODELS_DIR / "kmeans_model.pkl")

# ■■ Plot 1: UMAP colored by HDBSCAN clusters ■■■■■■■■■■■■■■■
print("\nPlotting UMAP figures...")
palette = plt.cm.tab10.colors

fig, ax = plt.subplots(figsize=(9, 7))
unique_labels = sorted(set(hdb_labels))
for lbl in unique_labels:
    mask = hdb_labels == lbl
    color = "#cccccc" if lbl == -1 else palette[lbl % 10]
    name  = "Noise" if lbl == -1 else f"Cluster {lbl+1} (n={mask.sum()})"
    ax.scatter(embedding[mask, 0], embedding[mask, 1],
               c=[color], label=name, s=18, alpha=0.75, linewidths=0)
ax.legend(fontsize=9, markerscale=2, loc="upper right")
ax.set_title("UMAP + HDBSCAN: MEQ-30 Responder Subtypes\n(Psychedelic Users N=414)",
             fontsize=13, fontweight="bold")
ax.set_xlabel("UMAP-1", fontsize=11); ax.set_ylabel("UMAP-2", fontsize=11)
ax.grid(alpha=0.2, linestyle="--")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "umap_clusters.png", dpi=150, bbox_inches="tight")
plt.close(fig)

# ■■ Plot 2: UMAP colored by CME ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
fig, ax = plt.subplots(figsize=(9, 7))
cme_colors = {0: "#c0392b", 1: "#2ecc71"}
for cme_val, label in [(0, "Non-CME"), (1, "CME")]:
    mask = cme == cme_val
    ax.scatter(embedding[mask, 0], embedding[mask, 1],
               c=[cme_colors[cme_val]], label=f"{label} (n={mask.sum()})",
               s=18, alpha=0.75, linewidths=0)
ax.legend(fontsize=11, markerscale=2)
ax.set_title("UMAP Embedding Colored by CME Status",
             fontsize=13, fontweight="bold")
ax.set_xlabel("UMAP-1", fontsize=11); ax.set_ylabel("UMAP-2", fontsize=11)
ax.grid(alpha=0.2, linestyle="--")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "umap_cme.png", dpi=150, bbox_inches="tight")
plt.close(fig)

# ■■ Plot 3: UMAP colored by SWLS score ■■■■■■■■■■■■■■■■■■■■■
fig, ax = plt.subplots(figsize=(9, 7))
sc = ax.scatter(embedding[:, 0], embedding[:, 1],
```

```
                c=y, cmap="RdYlGn", s=18, alpha=0.80, linewidths=0,
                vmin=y.min(), vmax=y.max())
plt.colorbar(sc, ax=ax, label="SWLS Total Score")
ax.set_title("UMAP Embedding Colored by SWLS Well-being Score",
             fontsize=13, fontweight="bold")
ax.set_xlabel("UMAP-1", fontsize=11); ax.set_ylabel("UMAP-2", fontsize=11)
ax.grid(alpha=0.2, linestyle="--")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "umap_swls.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("  Saved: umap_clusters.png, umap_cme.png, umap_swls.png")

# ■■ Cluster profiles ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nBuilding cluster profile table...")
df_cluster = df[MEQ_ITEM_COLS + [SWLS_COL, "cme_binary_computed"]].copy()
df_cluster["kmeans_cluster"] = km_labels + 1
df_cluster["hdbscan_cluster"] = hdb_labels

profiles = df_cluster.groupby("kmeans_cluster").agg(
    n=("SWLS_total", "count"),
    swls_mean=(SWLS_COL, "mean"),
    swls_std=(SWLS_COL, "std"),
    cme_rate=("cme_binary_computed", "mean"),
    **{f: (f, "mean") for f in MEQ_ITEM_COLS[:6]},  # first 6 items as sample
).round(3)
profiles.to_csv(Path("outputs") / "cluster_profiles.csv")
print(profiles.to_string())

# ■■ Radar chart of k-means cluster profiles ■■■■■■■■■■■■■■■■
print("\nPlotting radar chart...")
SUBSCALE_COLS = {
    SUBSCALE_DISPLAY_NAMES[key]: [f"MEQ_{i}" for i in items]
    for key, items in _SUBSCALE_ITEMS_KEYED.items()
}
radar_labels = list(SUBSCALE_COLS.keys())
N = len(radar_labels)
angles = [n / float(N) * 2 * np.pi for n in range(N)]
angles += angles[:1]

fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=dict(polar=True))
cluster_colors = ["#6c3fc7", "#3fc7a0", "#c73f6c", "#c7a03f", "#4fc7e0"]

for k_id in range(1, best_k + 1):
    mask = km_labels + 1 == k_id
    values = [df.loc[mask, cols].values.mean() for cols in SUBSCALE_COLS.values()]
    values += values[:1]
    ax.plot(angles, values, linewidth=2, color=cluster_colors[k_id-1], label=f"Cluster {k_id}
        (n={mask.sum()})")
    ax.fill(angles, values, alpha=0.10, color=cluster_colors[k_id-1])

ax.set_xticks(angles[:-1])
ax.set_xticklabels(radar_labels, fontsize=12)
ax.set_ylim(0, 5)
ax.set_yticks([1, 2, 3, 4, 5])
ax.set_yticklabels(["1", "2", "3", "4", "5"], fontsize=8)
ax.legend(loc="upper right", bbox_to_anchor=(1.3, 1.15), fontsize=10)
ax.set_title(f"MEQ-30 Subscale Profiles by Cluster (k={best_k})\nPsychedelic Users N=414",
             fontsize=13, fontweight="bold", pad=20)
plt.tight_layout()
fig.savefig(FIGURES_DIR / "cluster_radar.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("  Saved: cluster_radar.png")

print("\nPhase 4 complete. All clustering outputs saved.")
print(f"Silhouette scores: {sil_scores}")
```

scripts/sensitivity_full_sample.py

```
"""
scripts/sensitivity_full_sample.py

Sensitivity analysis on the full N=700 sample (psychedelic + non-psychedelic).
Tests whether findings generalize beyond the psychedelic subsample.

Also computes group-difference statistics:
  - MEQ scores: psychedelic vs non-psychedelic users
  - SWLS scores: by CME status
  - Effect sizes (Cohen's d)

Outputs:
  outputs/sensitivity_full_vs_psychedelic.csv
  outputs/group_statistics.csv
  outputs/figures/swls_by_group.png
  outputs/figures/meq_subscale_by_group.png
"""

import sys, warnings
sys.path.insert(0, ".")
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from scipy import stats
import xgboost as xgb
import joblib
import yaml
from sklearn.metrics import r2_score
from pathlib import Path
from src.meq30_reference import SUBSCALE_ITEMS as _SUBSCALE_ITEMS_KEYED, SUBSCALE_DISPLAY_NAMES
from src.nested_cv import nested_cv, fit_production_model, rmse

with open("config/config.yaml") as f:
    CONFIG = yaml.safe_load(f)

PROCESSED   = Path("data/processed")
FIGURES_DIR = Path("outputs/figures")
SEED = CONFIG["random_seed"]
OUTER_FOLDS = CONFIG["cv"]["n_folds"]
INNER_FOLDS = CONFIG["cv"]["inner_folds"]
MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]
SWLS_COL = "SWLS_total"

# Canonical, verified mapping — see src/meq30_reference.py.
SUBSCALE_ITEMS = {
    SUBSCALE_DISPLAY_NAMES[key]: items for key, items in _SUBSCALE_ITEMS_KEYED.items()
}


def cohens_d(a, b):
    na, nb = len(a), len(b)
    pooled_std = np.sqrt(((na-1)*a.std()**2 + (nb-1)*b.std()**2) / (na+nb-2))
    return (a.mean() - b.mean()) / pooled_std if pooled_std > 0 else 0.0


def xgb_param_suggester(trial):
    return {
        "max_depth":        trial.suggest_int("max_depth", 3, 7),
        "learning_rate":    trial.suggest_float("learning_rate", 0.01, 0.3, log=True),
        "n_estimators":     trial.suggest_int("n_estimators", 100, 500),
        "subsample":        trial.suggest_float("subsample", 0.6, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.6, 1.0),
```

```
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 10),
        "objective": "reg:pseudohubererror", "random_state": SEED,
        "n_jobs": -1, "verbosity": 0,
    }


def xgb_factory(params, y_for_fit):
    # base_score set explicitly — the installed xgboost (3.2.0) mis-estimates
    # it automatically for reg:pseudohubererror; see src/nested_cv.py.
    return xgb.XGBRegressor(**params, base_score=float(np.mean(y_for_fit)))

print("=" * 60)
print("Sensitivity Analysis — Full Sample N=700")
print("=" * 60)

# ■■ Load datasets ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
df_psych = pd.read_csv(PROCESSED / "features_psychedelic.csv")
df_full  = pd.read_csv(PROCESSED / "features_full.csv")

for df in [df_psych, df_full]:
    for sub, items in SUBSCALE_ITEMS.items():
        df[f"sub_{sub}"] = df[[f"MEQ_{i}" for i in items]].mean(axis=1)

df_full["group"] = np.where(df_full["any_psychedelic"] == 1, "Psychedelic", "Non-psychedelic")
df_nonpsych = df_full[df_full["any_psychedelic"] == 0].copy()

print(f"Full sample: N={len(df_full)}")
print(f"  Psychedelic:     N={len(df_psych)}")
print(f"  Non-psychedelic: N={len(df_nonpsych)}")

# ■■ Group statistics: MEQ subscales ■■■■■■■■■■■■■■■■■■■■■■■■■
print("\nGroup comparisons (MEQ subscales, psychedelic vs non-psychedelic):")
group_stats = []
for sub in SUBSCALE_ITEMS:
    col = f"sub_{sub}"
    g1 = df_psych[col].dropna()
    g2 = df_nonpsych[col].dropna()
    t_stat, p_val = stats.ttest_ind(g1, g2)
    d = cohens_d(g1, g2)
    row = {"subscale": sub, "psych_mean": round(g1.mean(),3), "psych_sd": round(g1.std(),3),
           "nonpsych_mean": round(g2.mean(),3), "nonpsych_sd": round(g2.std(),3),
           "t_stat": round(t_stat,3), "p_value": round(p_val,4), "cohens_d": round(d,3)}
    group_stats.append(row)
    sig = "***" if p_val < 0.001 else "**" if p_val < 0.01 else "*" if p_val < 0.05 else "ns"
    print(f"  {sub:15s}  Psychedelic={g1.mean():.2f}  Non={g2.mean():.2f}  d={d:.2f}  {sig}")

gs_df = pd.DataFrame(group_stats)
gs_df.to_csv(Path("outputs") / "group_statistics.csv", index=False)

# ■■ Group statistics: SWLS by CME status ■■■■■■■■■■■■■■■■■■■
print("\nSWLS by CME status (psychedelic subsample):")
cme_1  = df_psych[df_psych["cme_binary_computed"]==1][SWLS_COL]
cme_0  = df_psych[df_psych["cme_binary_computed"]==0][SWLS_COL]
t_cme, p_cme = stats.ttest_ind(cme_1, cme_0)
d_cme = cohens_d(cme_1, cme_0)
print(f"  CME=1 (n={len(cme_1)}): SWLS mean={cme_1.mean():.2f} SD={cme_1.std():.2f}")
print(f"  CME=0 (n={len(cme_0)}): SWLS mean={cme_0.mean():.2f} SD={cme_0.std():.2f}")
print(f"  t={t_cme:.3f}  p={p_cme:.4f}  d={d_cme:.3f}")

# ■■ XGBoost on full sample (nested CV — honest estimate) ■
print(f"\nTraining XGBoost on full sample (N={len(df_full)}) via nested CV...")
X_full_raw = df_full[MEQ_ITEM_COLS].values.astype(float)
y_full     = df_full[SWLS_COL].values.astype(float)

full_nested = nested_cv(
    X_full_raw, y_full, xgb_factory, xgb_param_suggester,
    seed=SEED, outer_folds=OUTER_FOLDS, inner_folds=INNER_FOLDS, n_trials=25,
)
```

```
cv_rmse_full, r2_full = full_nested["rmse"], full_nested["r2"]
print(f"  Full sample (nested CV): RMSE={cv_rmse_full:.4f}  R2={r2_full:.4f}")

model_full, sc_full, _ = fit_production_model(
    X_full_raw, y_full, xgb_factory, xgb_param_suggester,
    seed=SEED, cv_folds=OUTER_FOLDS, n_trials=50,
)
joblib.dump(sc_full, Path("outputs/models/scaler_full.pkl"))
model_full.save_model(str(Path("outputs/models/xgb_full_sample.json")))

# ■■ Comparison table ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# The "Psychedelic (primary)" row is NOT recomputed here — it's read from
# outputs/model_comparison.csv, which already holds the honest nested-CV
# XGBoost result for this exact sample (produced by train_models.py). An
# earlier version of this script reloaded the already-fitted production
# model/scaler and re-ran cross_val_predict on the same data used to fit
# them, which is itself a leakage pattern; reusing the nested-CV number
# avoids repeating that mistake.
y_psych = df_psych[SWLS_COL].values.astype(float)
model_comparison = pd.read_csv(Path("outputs") / "model_comparison.csv")
xgb_row = model_comparison.loc[model_comparison["model"] == "XGBoost"].iloc[0]

sens_df = pd.DataFrame([
    {"sample": "Psychedelic (primary)", "n": len(y_psych),
     "rmse": round(float(xgb_row["rmse"]), 4), "r2": round(float(xgb_row["r2"]), 4)},
    {"sample": f"Full sample (N={len(df_full)})", "n": len(y_full),
     "rmse": round(cv_rmse_full, 4), "r2": round(r2_full, 4)},
])
print("\n" + "="*55)
print("SENSITIVITY: Psychedelic vs Full Sample (both via nested CV)")
print(sens_df.to_string(index=False))
sens_df.to_csv(Path("outputs") / "sensitivity_full_vs_psychedelic.csv", index=False)

# ■■ Figure 1: SWLS distribution by group ■■■■■■■■■■■■■■■■■■
fig, axes = plt.subplots(1, 2, figsize=(13, 5))

# Panel A: SWLS by CME status
ax = axes[0]
data_cme = [cme_1.values, cme_0.values]
bp = ax.boxplot(data_cme, patch_artist=True, widths=0.5,
                medianprops=dict(color="white", linewidth=2))
for patch, color in zip(bp["boxes"], ["#2ecc71", "#e74c3c"]):
    patch.set_facecolor(color); patch.set_alpha(0.75)
ax.set_xticklabels([f"CME (n={len(cme_1)})", f"Non-CME (n={len(cme_0)})"], fontsize=11)
ax.set_ylabel("SWLS Total Score", fontsize=11)
ax.set_title(f"SWLS by CME Status\nt={t_cme:.2f}, p={p_cme:.3f}, d={d_cme:.2f}",
             fontsize=12, fontweight="bold")
ax.grid(axis="y", alpha=0.3, linestyle="--")

# Panel B: MEQ subscales psychedelic vs non-psychedelic
ax = axes[1]
subs = [s["subscale"] for s in group_stats]
psych_means = [s["psych_mean"] for s in group_stats]
nonpsych_means = [s["nonpsych_mean"] for s in group_stats]
x = np.arange(len(subs))
w = 0.35
ax.bar(x - w/2, psych_means, w, label=f"Psychedelic (n={len(df_psych)})", color="#6c3fc7",
    alpha=0.8)
ax.bar(x + w/2, nonpsych_means, w, label=f"Non-psychedelic (n={len(df_nonpsych)})",
    color="#e07b39", alpha=0.8)
ax.set_xticks(x); ax.set_xticklabels(subs, fontsize=10)
ax.set_ylabel("Mean Subscale Score (0–5)", fontsize=11)
all_p001 = all(s["p_value"] < 0.001 for s in group_stats)
subtitle = "(*** p < 0.001, all subscales)" if all_p001 else "(significance varies by subscale —
    see markers)"
ax.set_title(f"MEQ-30 Subscales: Psychedelic vs Non-Psychedelic\n{subtitle}",
             fontsize=12, fontweight="bold")
ax.legend(fontsize=10)
```

```
ax.set_ylim(0, 5); ax.grid(axis="y", alpha=0.3, linestyle="--")
for i, s in enumerate(group_stats):
    sig = "***" if s["p_value"] < 0.001 else "**" if s["p_value"] < 0.01 else "*"
    ax.text(i, max(psych_means[i], nonpsych_means[i]) + 0.15, sig,
            ha="center", fontsize=13, fontweight="bold")

plt.tight_layout()
fig.savefig(FIGURES_DIR / "sensitivity_group_stats.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("Saved: sensitivity_group_stats.png")
print("\nSensitivity analysis complete.")
```

scripts/train_within_cluster.py

```
"""
scripts/train_within_cluster.py

Within-cluster XGBoost models — tests the hypothesis that
MEQ-30 items predict SWLS *much* better within homogeneous subtypes
than across the full heterogeneous sample.

Outputs:
  outputs/within_cluster_comparison.csv
  outputs/models/xgb_cluster1.json
  outputs/models/xgb_cluster2.json
  outputs/figures/within_cluster_shap_cluster1.png
  outputs/figures/within_cluster_shap_cluster2.png
  outputs/figures/predicted_vs_actual_clusters.png
"""

import sys, warnings, json
sys.path.insert(0, ".")
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import shap
import xgboost as xgb
import optuna
import joblib
from sklearn.metrics import mean_absolute_error, r2_score
from pathlib import Path
from src.meq30_reference import MEQ_ITEM_LABELS
from src.nested_cv import nested_cv, fit_production_model, rmse

optuna.logging.set_verbosity(optuna.logging.WARNING)

PROCESSED   = Path("data/processed")
MODELS_DIR  = Path("outputs/models")
FIGURES_DIR = Path("outputs/figures")
SEED = 42
OUTER_FOLDS = 5   # reduced folds due to smaller cluster N
INNER_FOLDS = 3
MEQ_ITEM_COLS = [f"MEQ_{i}" for i in range(1, 31)]
SWLS_COL = "SWLS_total"

def evaluate(name, y_true, y_pred):
    return {
        "model": name,
        "n": len(y_true),
        "rmse": round(rmse(y_true, y_pred), 4),
        "mae":  round(mean_absolute_error(y_true, y_pred), 4),
        "r2":   round(r2_score(y_true, y_pred), 4),
    }

def xgb_param_suggester(trial):
    return {
        "max_depth":        trial.suggest_int("max_depth", 2, 6),
        "learning_rate":    trial.suggest_float("learning_rate", 0.01, 0.3, log=True),
        "n_estimators":     trial.suggest_int("n_estimators", 50, 400),
        "subsample":        trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 10),
        "reg_lambda":       trial.suggest_float("reg_lambda", 1.0, 10.0),
        "objective": "reg:pseudohubererror", "random_state": SEED,
        "n_jobs": -1, "verbosity": 0,
    }
```

```
def xgb_factory(params, y_for_fit):
    # base_score set explicitly — the installed xgboost (3.2.0) mis-estimates
    # it automatically for reg:pseudohubererror; see src/nested_cv.py.
    return xgb.XGBRegressor(**params, base_score=float(np.mean(y_for_fit)))

print("=" * 60)
print("MEQ-30 — Within-Cluster Model Analysis")
print("=" * 60)

df = pd.read_csv(PROCESSED / "features_psychedelic.csv")
scaler  = joblib.load(MODELS_DIR / "scaler.pkl")
km      = joblib.load(MODELS_DIR / "kmeans_model.pkl")

X_all_raw = df[MEQ_ITEM_COLS].values.astype(float)
X_std     = scaler.transform(X_all_raw)   # only used to assign cluster labels
y_all     = df[SWLS_COL].values.astype(float)
km_labs   = km.predict(X_std) + 1   # 1-indexed

print(f"Full N={len(df)}, clusters: {dict(zip(*np.unique(km_labs, return_counts=True)))}")

results = []
cluster_models = {}
cluster_shap   = {}
cluster_oof    = {}   # honest nested-CV out-of-fold predictions, reused for Figure 2

# ■■ Global model comparison row ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# Reused directly from outputs/model_comparison.csv (the honest nested-CV
# result for this exact sample/model, from train_models.py) rather than
# recomputed here by reloading the already-fitted production model and
# cross-validating it on the same data used to fit it, which would repeat
# the leakage pattern this project's audit flagged elsewhere.
model_comparison = pd.read_csv(Path("outputs") / "model_comparison.csv")
xgb_row = model_comparison.loc[model_comparison["model"] == "XGBoost"].iloc[0]
r_global = {
    "model": "XGBoost (full sample, from model_comparison.csv)",
    "cluster": "all", "n": len(y_all),
    "rmse": round(float(xgb_row["rmse"]), 4),
    "mae": round(float(xgb_row["mae"]), 4),
    "r2": round(float(xgb_row["r2"]), 4),
}
results.append(r_global)
print(f"\nGlobal model (nested CV, from Phase 2): RMSE={r_global['rmse']}  R2={r_global['r2']}")

# ■■ Per-cluster models (nested CV; standardization fit per-cluster) ■■
for k in sorted(np.unique(km_labs)):
    mask = km_labs == k
    X_k_raw = X_all_raw[mask]
    y_k     = y_all[mask]
    n_k     = mask.sum()
    print(f"\n--- Cluster {k} (N={n_k}) ---")

    nested = nested_cv(
        X_k_raw, y_k, xgb_factory, xgb_param_suggester,
        seed=SEED, outer_folds=min(OUTER_FOLDS, n_k // 10 or 2),
        inner_folds=INNER_FOLDS, n_trials=25,
    )
    r = evaluate(f"XGBoost Cluster {k}", y_k, nested["oof_preds"])
    r["cluster"] = k
    results.append(r)
    cluster_oof[k] = nested["oof_preds"]
    print(f"  Nested-CV result: RMSE={r['rmse']}  MAE={r['mae']}  R2={r['r2']}")

    # Production per-cluster model (all of this cluster's data) for SHAP/saving.
    model_k, scaler_k, _ = fit_production_model(
        X_k_raw, y_k, xgb_factory, xgb_param_suggester,
        seed=SEED, cv_folds=min(OUTER_FOLDS, n_k // 10 or 2), n_trials=40,
    )
    model_k.save_model(str(MODELS_DIR / f"xgb_cluster{k}.json"))
```

```
    cluster_models[k] = model_k

    # SHAP for this cluster (on the same standardized scale the model was trained on)
    X_k_std = scaler_k.transform(X_k_raw)
    X_k_df = pd.DataFrame(X_k_std, columns=MEQ_ITEM_COLS)
    expl   = shap.TreeExplainer(model_k)
    sv     = expl(X_k_df)
    cluster_shap[k] = sv

# ■■ Comparison table ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
print("\n" + "=" * 60)
print("WITHIN-CLUSTER vs GLOBAL COMPARISON")
print("=" * 60)
comp_df = pd.DataFrame(results)[["model","cluster","n","rmse","mae","r2"]]
print(comp_df.to_string(index=False))
comp_df.to_csv(Path("outputs") / "within_cluster_comparison.csv", index=False)

# ■■ Figure 1: SHAP bar per cluster side-by-side ■■■■■■■■■■■■
# Item labels come from the canonical, verified src/meq30_reference.py.
MEQ_LABELS = {f"MEQ_{i}": MEQ_ITEM_LABELS[i] for i in range(1, 31)}
CLUSTER_COLORS = {1: "#6c3fc7", 2: "#e07b39"}

fig, axes = plt.subplots(1, len(cluster_models), figsize=(16, 10), sharey=False)
if len(cluster_models) == 1:
    axes = [axes]

for ax, (k, sv) in zip(axes, cluster_shap.items()):
    mean_abs = np.abs(sv.values).mean(axis=0)
    item_imp = pd.DataFrame({"feature": MEQ_ITEM_COLS,
                             "label": [MEQ_LABELS[c] for c in MEQ_ITEM_COLS],
                             "shap": mean_abs}).sort_values("shap")
    n_k = (km_labs == k).sum()
    r2_k = next(r["r2"] for r in results if r["cluster"] == k)
    ax.barh(item_imp["label"], item_imp["shap"],
            color=CLUSTER_COLORS.get(k, "#888"), alpha=0.85, edgecolor="none")
    ax.set_title(f"Cluster {k}  (N={n_k})\nR²={r2_k:.3f}",
                 fontsize=12, fontweight="bold", color=CLUSTER_COLORS.get(k, "#333"))
    ax.set_xlabel("Mean |SHAP|", fontsize=10)
    ax.grid(axis="x", alpha=0.25, linestyle="--")
    if k == 1:
        ax.set_ylabel("MEQ-30 Item", fontsize=10)

fig.suptitle("Within-Cluster SHAP Feature Importance\nMEQ-30 → SWLS Well-being by Responder
    Subtype",
             fontsize=14, fontweight="bold", y=1.01)
plt.tight_layout()
fig.savefig(FIGURES_DIR / "within_cluster_shap.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("Saved: within_cluster_shap.png")

# ■■ Figure 2: Predicted vs Actual ■■■■■■■■■■■■■■■■■■■■■■■■■■
fig, axes = plt.subplots(1, len(cluster_models), figsize=(12, 5))
if len(cluster_models) == 1:
    axes = [axes]

for ax, k in zip(axes, sorted(cluster_models)):
    mask = km_labs == k
    y_k  = y_all[mask]
    # Reuse the honest nested-CV out-of-fold predictions computed above,
    # rather than re-cross-validating the already-tuned production model
    # (which would repeat the tuning-leakage pattern this project's audit
    # flagged in the original scripts).
    preds_k = cluster_oof[k]
    r2_k = r2_score(y_k, preds_k)
    ax.scatter(y_k, preds_k, alpha=0.5, s=20,
               color=CLUSTER_COLORS.get(k, "#888"), edgecolors="none")
    lims = [min(y_k.min(), preds_k.min()) - 1, max(y_k.max(), preds_k.max()) + 1]
    ax.plot(lims, lims, "k--", lw=1, alpha=0.5)
```

```
    ax.set_xlim(lims); ax.set_ylim(lims)
    ax.set_xlabel("Actual SWLS", fontsize=10)
    ax.set_ylabel("Predicted SWLS", fontsize=10)
    ax.set_title(f"Cluster {k} (N={mask.sum()})\nR²={r2_k:.3f}", fontsize=11, fontweight="bold")
    ax.grid(alpha=0.2, linestyle="--")

fig.suptitle("Predicted vs Actual SWLS — Within-Cluster Models",
             fontsize=13, fontweight="bold")
plt.tight_layout()
fig.savefig(FIGURES_DIR / "predicted_vs_actual_clusters.png", dpi=150, bbox_inches="tight")
plt.close(fig)
print("Saved: predicted_vs_actual_clusters.png")

print("\nWithin-cluster analysis complete.")
```

scripts/generate_report.py

```
"""
scripts/generate_report.py

Compiles all Phase 2-4 outputs into a structured Markdown results report.
Produces: outputs/RESULTS_REPORT.md

All figures/numbers in this report are read directly from the CSV/JSON
outputs of the pipeline scripts at generation time — including the "key
finding" narrative paragraphs, which are built from the actual top-ranked
items/subscales rather than hardcoded text. (An earlier version of this
script had narrative paragraphs describing specific items by name as
literal strings, disconnected from the data driving the rest of the
report — if the top items ever changed, the prose would silently go stale.)
"""

import sys
import json
import hashlib
from pathlib import Path
from datetime import datetime

import numpy as np
import pandas as pd

PROCESSED   = Path("data/processed")
OUTPUTS     = Path("outputs")
FIGURES_DIR = OUTPUTS / "figures"
MODELS_DIR  = OUTPUTS / "models"
RAW         = Path("data/raw")


def sha256(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(8192), b""):
            h.update(chunk)
    return h.hexdigest()


lines = []
def w(*args): lines.extend(args); lines.append("")

now = datetime.now().strftime("%Y-%m-%d %H:%M")

# ■■ Header ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w(
    "# MEQ-30 Machine Learning Study — Results Report",
    f"**Generated:** {now}  |  **IRB Status:** EXEMPT (45 CFR 46.104(d)(4))",
    "",
    "> **Methodology changelog:** this report reflects a corrected pipeline.",
    "> Earlier runs used a MEQ-30 item-to-subscale mapping that did not match",
    "> the published instrument (Barrett, Johnson & Griffiths, 2015), and",
    "> selected model hyperparameters using the same cross-validation folds",
    "> later used to report performance (an optimistic-bias/leakage bug).",
    "> Both are fixed here — see `src/meq30_reference.py`, `src/nested_cv.py`,",
    "> and Section 10 for the human/AI-assisted audit that found them.",
    "",
    "---",
)

# ■■ Data Provenance ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 1. Data Provenance")
owsiak = RAW / "owsiak2026_survey.sav"
bennett = RAW / "bennett2023_phase1.xlsx"
df_full_prov = pd.read_csv(PROCESSED / "features_full.csv")
```

```python
n_full = len(df_full_prov)
n_bennett = 14
w(
    "| Dataset | N | License | SHA-256 |",
    "|---|---|---|---|",
    f"| Owsiak 2026 (Mendeley) | {n_full} | CC BY 4.0 | `{sha256(owsiak)[:16]}...` |",
    f"| Bennett 2023 (Figshare) | {n_bennett} | CC BY 4.0 | `{sha256(bennett)[:16]}...` |",
)
w(f"> **Primary dataset:** Owsiak 2026 — N={n_full} Polish adults, MEQ-30 items + SWLS well-being outcome")

# ■■ Dataset stats ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
df = pd.read_csv(PROCESSED / "features_psychedelic.csv")
meq_cols = [f"MEQ_{i}" for i in range(1, 31)]
n_psych = len(df)
pct_psych = n_psych / n_full
w(f"> **Primary analysis sample:** Psychedelic users only (N={n_psych}, {pct_psych:.1%} of full sample)")

w(f"## 2. Dataset Characteristics (Psychedelic Subsample N={n_psych})")
completeness = df[meq_cols].notna().mean().mean()
computed_cme_rate = df["cme_binary_computed"].mean()
stats_rows = [
    "| Metric | Value |",
    "|---|---|",
    f"| N (psychedelic users) | {n_psych} |",
    f"| MEQ-30 completeness | {completeness:.0%} |",
    f"| CME rate (60% threshold, corrected subscale mapping) | {computed_cme_rate:.1%} |",
]
if "MEQ_complete" in df.columns:
    native_cme_rate = df["MEQ_complete"].mean()
    disagree_rate = (df["cme_binary_computed"] != df["MEQ_complete"]).mean()
    stats_rows.append(f"| CME rate (dataset's native pre-scored label) | {native_cme_rate:.1%} |")
    stats_rows.append(f"| Recomputed-vs-native CME disagreement | {disagree_rate:.1%} of records |")
stats_rows += [
    f"| SWLS range | {df['SWLS_total'].min():.0f} – {df['SWLS_total'].max():.0f} |",
    f"| SWLS mean (SD) | {df['SWLS_total'].mean():.2f} ({df['SWLS_total'].std():.2f}) |",
    f"| MEQ-30 item range | 0–5 (Likert) |",
]
w(*stats_rows)
if "MEQ_complete" in df.columns and disagree_rate > 0.05:
    w("> [!WARNING]",
      f"> Recomputed and native CME labels disagree on {disagree_rate:.1%} of records. "
      "This gap previously exceeded 10 points before the subscale mapping was corrected "
      "(see scripts/verify_claims.py); a residual gap this size may indicate the dataset's "
      "own native scoring uses a slightly different rounding/threshold rule and is worth "
      "investigating further before treating either label as ground truth.")

# ■■ Model Comparison ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
comp = pd.read_csv(OUTPUTS / "model_comparison.csv")
n_outer_folds = 10  # matches config.yaml cv.n_folds
w(f"## 3. Model Comparison (Nested {n_outer_folds}-Fold Cross-Validation)")
w("> **Outcome:** SWLS Total Score (Satisfaction with Life Scale, range 5–35)")
w("> Hyperparameter tuning (Optuna, inner folds) never sees the outer folds used to compute these metrics.")
w(
    "| Rank | Model | RMSE | MAE | R² | ΔRMSE vs Baseline |",
    "|---|---|---|---|---|---|",
)
for _, row in comp.iterrows():
    delta = f"{row.get('delta_rmse', float('nan')):+.4f}" if pd.notna(row.get("delta_rmse")) else "—"
    lines.append(f"| {int(row['rank'])} | **{row['model']}** | {row['rmse']:.4f} | {row['mae']:.4f} | {row['r2']:.4f} | {delta} |")
w("")

best_model_row = comp.sort_values("rmse").iloc[0]
```

```
best_r2 = best_model_row["r2"]
r2_note = ("weak-to-negligible" if best_r2 < 0.05 else "modest") if best_r2 >= 0 else "essentially
    absent (R² < 0)"
w(
    "> [!NOTE]",
    f"> The best model ({best_model_row['model']}) achieves R²={best_r2:.4f} under honest nested
        CV — "
    f"{r2_note} predictive signal. MEQ-30 item scores alone are weak predictors of SWLS across the
        full",
    "> psychedelic user sample. This is consistent with a heterogeneity hypothesis: global MEQ
        totals",
    "> may mask subtype-specific response patterns — see clustering results (Section 5/6) for
        whether",
    "> within-subtype models do better.",
)

# ■■ SHAP Results ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 4. SHAP Feature Importance (XGBoost, Best Model)")
shap_df = pd.read_csv(OUTPUTS / "shap_top_items.csv")
w(
    "| Rank | MEQ Item | Label | Subscale | Mean |SHAP| |",
    "|---|---|---|---|---|---|",
)
for _, row in shap_df.head(10).iterrows():
    lines.append(f"| {int(row['rank'])} | `{row['feature']}` | {row['label']} | {row['subscale']}
        | {row['mean_abs_shap']:.4f} |")
w("")

# Key finding paragraph built entirely from the actual top-ranked rows,
# not hardcoded item names — this regenerates correctly if the ranking changes.
top5 = shap_df.head(5)
top5_subscales = top5["subscale"].value_counts()
dominant = top5_subscales.index[0]
dominant_share = top5_subscales.iloc[0]
top3_desc = "; ".join(
    f"{row['feature']} (*{row['label']}*, {row['subscale']})" for _, row in
        top5.head(3).iterrows()
)
w(
    f"> **Key finding:** Of the top 5 items by SHAP importance, {dominant_share}/5 belong to the "
    f"**{dominant}** subscale. The single strongest individual predictors of SWLS well-being are:
        "
    f"{top3_desc}.",
    "> Given the low overall R² (Section 3), these item rankings should be read as descriptive of
        "
    "this specific model/sample rather than as a robust, generalizable finding about which facet
        of "
    "mystical experience drives well-being.",
)

# ■■ Clustering ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 5. Unsupervised Subtype Discovery")
if (OUTPUTS / "cluster_profiles.csv").exists():
    profiles = pd.read_csv(OUTPUTS / "cluster_profiles.csv")
    w(
        "| Cluster | N | SWLS Mean | CME Rate |",
        "|---|---|---|---|",
    )
    for _, row in profiles.iterrows():
        lines.append(f"| {int(row['kmeans_cluster'])} | {int(row['n'])} | {row['swls_mean']:.2f} |
            {row['cme_rate']:.1%} |")
    w("")
    best_cluster = profiles.loc[profiles["swls_mean"].idxmax()]
    worst_cluster = profiles.loc[profiles["swls_mean"].idxmin()]
    swls_diff = best_cluster["swls_mean"] - worst_cluster["swls_mean"]
    w(
        f"> **Key finding:** k-Means clustering reveals responder subtypes with a",
        f"> **{swls_diff:.1f} point SWLS gap** between the highest and lowest well-being
```

```
            clusters",
        f"> (using the corrected CME/subscale computation).",
    )
else:
    w("> *Clustering results pending — run scripts/run_clustering.py*")

# ■■ Within-cluster models ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 6. Within-Cluster Models")
if (OUTPUTS / "within_cluster_comparison.csv").exists():
    wc = pd.read_csv(OUTPUTS / "within_cluster_comparison.csv")
    w(
        "> Tests whether MEQ-30 items predict SWLS better within homogeneous responder subtypes",
        "> than across the full heterogeneous sample. All rows use nested CV (Section 3
            methodology).",
        "",
        "| Model | Cluster | N | RMSE | MAE | R² |",
        "|---|---|---|---|---|---|",
    )
    for _, row in wc.iterrows():
        lines.append(f"| {row['model']} | {row['cluster']} | {int(row['n'])} | {row['rmse']:.4f} |
            {row['mae']:.4f} | {row['r2']:.4f} |")
    w("")
    global_r2 = wc.loc[wc["cluster"] == "all", "r2"].iloc[0] if (wc["cluster"] == "all").any()
        else None
    cluster_rows = wc[wc["cluster"] != "all"]
    if len(cluster_rows) and global_r2 is not None:
        best_cluster_row = cluster_rows.loc[cluster_rows["r2"].idxmax()]
        improves = best_cluster_row["r2"] > global_r2
        w(
            f"> **Key finding:** The best within-cluster model (Cluster
                {best_cluster_row['cluster']}, "
            f"R²={best_cluster_row['r2']:.4f}) "
            + ("outperforms" if improves else "does not clearly outperform")
            + f" the pooled full-sample model (R²={global_r2:.4f}), "
            + ("offering some support for" if improves else "which weakens")
            + " the heterogeneity hypothesis from Section 3. Given the small per-cluster N, treat
                this "
              "as exploratory rather than confirmatory.",
        )
else:
    w("> *Within-cluster results pending — run scripts/train_within_cluster.py*")

# ■■ Sensitivity: full sample vs psychedelic subsample ■■■■■■
w("## 7. Sensitivity: Full Sample vs. Psychedelic Subsample")
if (OUTPUTS / "sensitivity_full_vs_psychedelic.csv").exists():
    sens = pd.read_csv(OUTPUTS / "sensitivity_full_vs_psychedelic.csv")
    w(
        "| Sample | N | RMSE | R² |",
        "|---|---|---|---|",
    )
    for _, row in sens.iterrows():
        lines.append(f"| {row['sample']} | {int(row['n'])} | {row['rmse']:.4f} | {row['r2']:.4f}
            |")
    w("")
if (OUTPUTS / "group_statistics.csv").exists():
    gs = pd.read_csv(OUTPUTS / "group_statistics.csv")
    w(
        "**MEQ-30 subscale scores, psychedelic vs. non-psychedelic users:**",
        "",
        "| Subscale | Psychedelic Mean (SD) | Non-Psychedelic Mean (SD) | Cohen's d | p |",
        "|---|---|---|---|---|",
    )
    for _, row in gs.iterrows():
        lines.append(
            f"| {row['subscale']} | {row['psych_mean']:.2f} ({row['psych_sd']:.2f}) | "
            f"{row['nonpsych_mean']:.2f} ({row['nonpsych_sd']:.2f}) | {row['cohens_d']:.2f} |
                {row['p_value']:.4f} |"
        )
```

```
    w("")

# ■■ VIF ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 8. Collinearity Report (VIF)")
vif = pd.read_csv(PROCESSED / "vif_report.csv")
threshold = 5.0  # matches config.yaml preprocessing.vif_threshold
high_vif = vif[vif["vif"] > 10].sort_values("vif", ascending=False)
over_config_threshold = (vif["vif"] > threshold).sum()
w(
    f"> {len(high_vif)} of {len(vif)} MEQ items have VIF > 10; {over_config_threshold} of
        {len(vif)} exceed "
    f"config.yaml's informational threshold of {threshold:.1f} (multicollinearity is expected —
        MEQ-30 items "
    "within a subscale are, by construction, correlated with each other).",
    "> **VIF is not used to drop features.** XGBoost/LightGBM are collinearity-robust and are
        treated as the",
    "> primary models; Ridge Regression results (Section 3) should be interpreted with that caveat
        in mind.",
)
w(
    "| Feature | VIF |",
    "|---|---|",
)
for _, row in high_vif.head(8).iterrows():
    lines.append(f"| `{row['feature']}` | {row['vif']:.1f} |")
w("")

# ■■ Figures ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 9. Figures")
figs = sorted(FIGURES_DIR.glob("*.png"))
w(f"All {len(figs)} figures saved to `outputs/figures/`:")
for fig in figs:
    w(f"- `{fig.name}`")

# ■■ Limitations ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 10. Limitations & Next Steps")
w(
    "- **No PEQ data available.** SWLS used as well-being proxy; direct PEQ replication",
    "  requires a future dataset with both MEQ-30 items and PEQ paired scores.",
    "- **Low overall R²** limits clinical translation regardless of the heterogeneity
        hypothesis.",
    "  Within-cluster models (Section 6) are one attempt at addressing this; results are
        exploratory",
    "  given small per-cluster sample sizes.",
    "- **Cross-sectional design.** Causal inference cannot be drawn; temporal ordering of",
    "  mystical experience → well-being cannot be established from survey data.",
    "- **Sample:** Polish adults, community sample. Generalizability to clinical/controlled",
    "  psilocybin contexts (e.g., Barrett et al. 2015) requires further validation.",
    "- **Notebook/script duplication.** This project's `scripts/` directory is the pipeline that",
    "  actually produced these results; several notebooks in `notebooks/` are exploratory and
        were",
    "  historically inconsistent with `scripts/` — see README.md's Pipeline Workflow section for
        the",
    "  current, corrected description of which artifacts are authoritative.",
)

# ■■ AI Use Acknowledgement ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 11. Acknowledgement of AI Use")
w(
    "Generative AI (Claude, Anthropic — Claude Code, model *Sonnet 5*) was used as a development
        and",
    "research-assistance tool throughout this project, including:",
    "",
    "- Auditing the pipeline for factual, statistical, and methodological accuracy, which surfaced
        the",
    "  MEQ-30 subscale-mapping error and the hyperparameter-tuning leakage bug corrected in this
        report;",
    "- Verifying dataset provenance (DOIs, licenses) and the published MEQ-30 factor structure
```

```
        against",
    "  primary sources (Barrett, Johnson & Griffiths, 2015);",
    "- Implementing the corrections described in the methodology changelog above (canonical
        subscale",
    "  mapping, nested cross-validation, report-generation fixes) and drafting this report.",
    "",
    "All AI-assisted analysis and code changes were scoped to this repository's existing data and
        were",
    "verified against the underlying CSV/JSON outputs before being included here. AI assistance
        was used",
    "for engineering and drafting support; the study design, data sourcing, and interpretation of
        findings",
    "remain the responsibility of the project author(s). See Section 12 for the corresponding
        citation.",
)

# ■■ References ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
w("## 12. References")
w(
    "- Barrett, F. S., Johnson, M. W., & Griffiths, R. R. (2015). Validation of a revised
        Mystical",
    "  Experience Questionnaire in experimental sessions with psilocybin. *Journal of",
    "  Psychopharmacology*, 29(11), 1182–1190. (MEQ-30 instrument and subscale structure.)",
    "- Bennett, J., Blough, M. D., Bains, R., Galloway, L., & Mitchell, I. (2023). *Phase I
        Trial",
    "  Data.xlsx* [Data set]. Figshare. https://doi.org/10.6084/m9.figshare.22329433.v1",
    "- Owsiak, M. (2026). *Survey Data on Psychedelic and Non-Psychedelic Mystical-Type
        Experiences",
    "  in Polish Adults* (Version 3) [Data set]. Mendeley Data.
        https://doi.org/10.17632/bhkb7zmb9v.3",
    "- Diener, E., Emmons, R. A., Larsen, R. J., & Griffin, S. (1985). The Satisfaction with
        Life",
    "  Scale. *Journal of Personality Assessment*, 49(1), 71–75. (SWLS instrument, used here as
        a",
    "  well-being proxy outcome — see DATASET_ORIGIN.md for the rationale and its limitations.)",
    "- Anthropic (2026). *Claude (Sonnet 5)* [Large language model]. https://www.anthropic.com.
        Used as",
    "  a development and analysis-auditing assistant for this project; see Section 11.",
)

# ■■ Write file ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
out = OUTPUTS / "RESULTS_REPORT.md"
out.write_text("\n".join(lines), encoding="utf-8")
print(f"Report saved: {out}  ({out.stat().st_size:,} bytes)")
print("Done.")
```