1. **Feature Engineering Pipeline:**
2. Per‐Capita Normalization  
   * Why? Controls for household size when predicting MPCE.
   * How? Divide a total-expense column by household size to get per-person expenditure.
3. Categorical Code Abstraction (NIC & NCO)  
   * Why? Hundreds of fine‐grained industry (NIC) and occupation (NCO) codes can overfit; grouping them into a few high‐level sectors or professions improves generalization.
   * How?  
     + NIC → Section → Numeric: Extract the first two digits of each NIC code, map to a sector label (Agriculture, Manufacturing, etc.), then encode those labels as integers.
     + NCO → Major Group → Numeric: Take the leading digit of each 3-digit NCO code, map to one of ten major occupational groups, then integer-encode.
4. Latent Activity Scores via Weighted Aggregation  
   * Why? Tens of binary indicator columns (e.g. devices owned, services used) create sparsity; projecting them into a few continuous “scores” captures the same information compactly.
   * How? Pre-computed coefficient vectors (from prior PCA / regression / SHAP analysis) are dotted with groups of related indicator columns to produce:  
     + online\_activity
     + entertainment
     + vehicle
     + electronic  
        Each original block of 3–11 binary features is collapsed into a single continuous feature.
5. Person-to-Household Aggregations  
   * Why? The HCES data includes one row per household member, but the target (MPCE) is at the household level.
   * How?  
     + Head-of-Household attributes: Pull age, gender, education of the “head” member.
     + Demographic ratios: Compute male‐to‐total ratio.
     + Numerical sums & means: Sum counts (e.g., number of dependents, meals away) and average continuous attributes (e.g., years of education).
     + Flags: Aggregate any‐True binary flags (e.g. internet use) into household‐level indicators.
6. Imputation & Binarization  
   * Why? Ensure no missing values and convert key categorical/binary fields into numeric form.
   * How?  
     + Numerical fills: Group-mean or overall-mean for education; zero for counts (days away).
     + Binary conversion: Map codes (e.g. marital status == couple → 1, else 0; internet use yes/no → 1/0).
     + Mode imputation: Replace missing “meals per day” with its most common value.
7. Final Schema Alignment  
   * Why? Clean, consistent column names and ordering simplify downstream modeling.
   * How? Systematically rename columns to descriptive identifiers (e.g. head\_education\_years, Total\_Expense) and reorder so related features appear together.

### **Theory in a Nutshell**

“Shrink, aggregate, encode and normalize.”

1. Shrink large categorical spaces (NIC/NCO) into a handful of high‐level codes.
2. Aggregate person‐level records into household summaries (sums, means, flags).
3. Encode domain signals via engineered composite scores (weighted sums of related indicators).
4. Normalize & Impute to eliminate scale differences and missingness.

This approach balances interpretability (you know exactly what each feature means) with parsimony (you dramatically reduce dimensionality), setting you up for a robust MPCE prediction.

**B. Stacking Models**

2. 1. Why Demographics Matter for Consumption

Household consumption (and so MPCE) is heavily driven by the composition of the household:

* Dependency ratios (infant/adult/overall) approximate how many non-earning members rely on each earner.
* Age distribution (mean, spread, min/max) affects needs (e.g., children vs. retirees).
* Educational attainment proxies earning capacity and spending preferences.

Capturing these patterns lets a model adjust expectations of per-capita spending beyond raw income or location.

## 2. Feature Construction

### 2.1 Total Persons, Infants, Adults

* total\_persons: Household size—more mouths to feed.
* num\_infants: Ages 0–4 are high-dependency, low spending on big-ticket goods but high on essentials (food, healthcare).
* num\_adults: Potential earners (≥18 years).

### 2.2 Dependency Ratios

* infant\_dependency\_ratio = num\_infants ÷ num\_adults
* overall\_dependency\_ratio = (dependents ÷ total\_persons), where “dependents” = children (<18) + elderly (≥65).

A higher ratio means fewer earners per dependent, often reducing per-capita consumption.

### 2.3 Age Statistics

* avg\_age\_y, std\_age\_y, min\_age\_y, max\_age\_y  
  + A younger household might spend differently than an older one (e.g., school fees vs. healthcare).
  + Std. dev. captures heterogeneity within the household—mixed-age families may have broader expenditure needs.

### 2.4 Education

* avg\_education: Mean years of schooling across members.  
  + Education correlates with income and with consumption patterns (e.g., more spending on leisure, better nutrition).

## **C. Modeling Approach**

### 3.1 Pipelines with Imputation

* SimpleImputer (median)  
  + Fills any missing demographic Stats with the household’s “typical” value, robust to outliers.
* Separate Pipelines for:  
  + Linear Regression (captures *global*, smooth, additive effects of each demographic feature)
  + Random Forest (captures *non-linear* interactions, threshold effects—e.g., a household crosses a dependency ratio “tipping point”).

### 3.2 Why Two Models?

* Linear Regression gives an interpretable baseline: you can read off coefficients to see how, say, one extra infant reduces MPCE.
* Random Forest handles more complex patterns (e.g., two young children might affect spending more than twice one child).

## **D. Training & Prediction Flow**

1. compute\_household\_stats:  
   * Aggregates person-level ages and education into household summary statistics.
2. preprocess:  
   * Merges summary stats with household-level data (e.g., location, assets).
   * Imputes zeros for any truly missing household-level values.
3. fit:  
   * Trains both pipelines on the same feature set, optimizing their parameters to minimize squared error on MPCE.
4. predict / evaluate:  
   * Applies the same preprocessing to new data, then produces two sets of demographic-only “constant” predictions.
   * Correlates predictions with actual MPCE to gauge how much of the variation is explained by demographics alone.

## **D. Theoretical Takeaways**

* Demographics alone explain a substantial share of household expenditure variation—especially in contexts where age structure and dependency burdens vary widely.
* Combining a linear model (for macro-level trends) with a forest (for micro-level twists) provides both interpretability and flexibility.
* These demographic constants can then be used as features in a larger MPCE model—perhaps alongside your rich industry, asset, and behavioral scores—for a more holistic expenditure predictor.

## **1. Linear Regression: Capturing the “Global Trend”**

1. Form  
    MPCE = β0 + ∑ i =1 to p βi xi ​

where each xi​ is a demographic feature (e.g., dependency ratio, average age).

1. Role  
   * Baseline: Provides a straightforward, additive estimate of how each demographic factor “moves” MPCE.
   * Interpretability: Coefficients βi​ tell you “one more dependent reduces MPCE by βdep​” (holding other features constant).
   * Robustness: With median imputation upstream, it handles moderate missingness and outliers without drastic distortion.
2. Limitations  
   * Linearity Assumption: Assumes each feature’s effect is constant and additive—no thresholds or interactions (e.g., two infants might not cost exactly twice one).
   * Underfitting Non-Linearities: Won’t capture “tipping points” where, say, having a third dependent drastically changes spending behavior.

## **2. Random Forest: Capturing “Local Nuance”**

1. Form  
    An ensemble of decision trees, each trained on a bootstrap sample; final prediction is the average of all trees’ outputs.

1. Role  
   * Non-Linear Relationships: Automatically discovers interactions (e.g., high dependency + low education might compound into especially low MPCE).
   * Threshold Effects: Can learn that a dependency ratio above 0.6 sharply changes spending patterns.
   * Feature Importance: Provides a ranking of which demographics most reduce the error.
2. Limitations  
   * Less Interpretable: Harder to translate into simple “per-unit” effects.
   * Overfitting Risk: If unchecked (too many trees or too deep), can memorize noise—though ensemble averaging mitigates that.

## 3. Why Use Both Together?

* Complementary Strengths  
  + Linear Regression nails down the overall slope of how demographics move MPCE.
  + Random Forest refines with local adjustments, picking up on non-linear “bumps” and interactions.
* Ensemble Potential  
  + You could stack them or create a weighted blend:  
     MPCE =α MPCE(LR) + (1−α)  MPCE(RF)
  + This often improves accuracy by hedging between under- and over-fitting.
* Diagnostic Value  
  + Compare their correlations with true MPCE: if RF outperforms LR by a lot, there’s strong non-linearity to exploit; if they’re close, demographics act largely linearly.

**E. Intermediate Feature Merging(Linear Regression and Random Forest Prediction)**

They’re intermediate “constant” features that you merge back into your train/test sets so that your main model (e.g. XGBoost) can use them alongside your other engineered features.

**In a Nutshell**

* Yes, you merge both train & test predictions back into your feature tables.
* They become first‐stage model outputs that feed into your second‐stage (final) MPCE regressor.
* This two‐stage approach (demographics → constants → full model) often boosts accuracy by letting each learner focus on the patterns it handles best.

**F. Survey‐to-Survey Imputation:**

Two separate linear models (with polynomial expansions) are trained per state on two distinct feature groups—“online” indicators and “assert” (asset/household-have) indicators—and their fitted outputs are treated as state-specific constants for downstream MPCE prediction.

## **1. Why Train State-Wise Models?**

* **Heterogeneity Across States:** India’s states differ dramatically in demographics, infrastructure, income levels, and consumption norms. A single nationwide model could be swamped by these differences.
* **Local Calibration:** By fitting separate models for each State, you allow each state’s unique patterns (e.g., Internet penetration, asset ownership) to shape its own imputation function.

## **2. Two Feature Groups (“Online” vs. “Assert”)**

You split your predictors into two conceptually distinct blocks:

1. **Online Features (Is\_online\_\*)**
   * Represent household engagement with digital services (e.g., Internet usage, online purchases).
   * Proxy for connectivity, urbanization, and exposure to modern commerce.
2. **Asset/Household-Have Features (Is\_HH\_Have\_\*)**
   * Binary flags for possession of assets or facilities (e.g., electricity, water, appliances).
   * Proxy for wealth, living standard, physical capital.

Each block feeds its own polynomial‐powered linear model, yielding two separate “constants” that capture different dimensions of a household’s socioeconomic status.

## **3. Polynomial Linear Regression**

* **PolynomialFeatures(degree=2)** augments your original binaries with:  
  + **Squared terms** (xi2x\_i^2xi2​) to capture curvature (e.g., the marginal value of a second asset may differ from the first).
  + **Pairwise interactions** (xi×xjx\_i \times x\_jxi​×xj​) to capture synergies (e.g., having both a refrigerator *and* running water might multiplicatively boost consumption).
* **LinearRegression** then fits an additive model on this expanded basis.  
  + It remains convex and fast to train, yet can express simple non‐linear relationships and two-way interactions.

## **4. Training Loop & Correlation Monitoring**

* **Per‐State Fit:** For each state, you:  
  1. Subset the rows
  2. Impute missing indicators with 0
  3. Expand features (PolynomialFeatures)
  4. Fit LinearRegression
  5. Compute the **in‐sample correlation** between predictions and true MPCE to gauge how well that block alone explains spending variance in that state.
* **Overall Correlation:** By aggregating predictions across all states, you also get a bird’s-eye estimate of how much each block (online vs. assert) explains MPCE variation nationally.

## **5. Persisting & Reusing State Models**

You serialize a dictionary  
  
{ state\_int : {

'online': (poly\_online, linreg\_online),

'assert': (poly\_assert, linreg\_assert)

}

}

* Later, when you call test\_models, you reload these per-state transformers and regressors, apply them to new data, and produce two **constants** per household:  
  + online\_constant
  + having\_constant

These become **first‐stage features** in your final MPCE model.

## **6. Why Blend These State-Specific Constants?**

* **Dimension Reduction:** Millions of raw binaries → two summary statistics per household.
* **Local Expertise:** Each constant encodes the best‐fit combination of indicators for that state.
* **Ensemble Stacking:** You can feed both constants into your ultimate XGBoost/RandomForest/GBM regressor as powerful meta‐features alongside your demographic scores and industry/occupation codes.
* **Robustness:** If one block (e.g., online) is weak in a particular state, the other block still contributes predictive signal.

### **In a Nutshell**

1. **Split** your high‐dimensional indicator sets into two thematic groups.
2. **Fit** separate polynomial linear models **per state** to capture local, non‐linear patterns.
3. **Extract** out‐of‐sample constants on new data.
4. **Merge** those constants into your full feature matrix for the final MPCE predictor.

This two‐stage approach—**state‐wise polynomial imputation** followed by a **global second‐stage learner**—gives you both local calibration and overall generalization, improving your ability to impute consumption in surveys that lack direct expenditure data.

**G. Household-level “personal” expense model** using **CatBoost**

## 1. Purpose of the HouseholdExpenseModel

* Goal: Distill person-level behavioral patterns around meals and mobility into a single “personal\_level\_constant” feature per household, which captures how individual habits drive total expenditure.
* Why? Even with demographics and assets accounted for, true consumption also depends on daily routines (meals eaten, days away), so this model quantifies that.

## 2. Feature Construction & Preprocessing

1. Food Features:  
   * Six variables:  
     + meals\_per\_day (typical daily meals)
     + meals\_school, meals\_employer, meals\_others, meals\_paid, meals\_home (counts over last 30 days)
   * You compute two summary stats per feature:  
     + sum (volume over 30 days)
     + mean (rate per person)
2. Non-Food Feature:  
   * days\_away (days spent away from home in last 30 days), again aggregated as sum & mean.
3. Result: After merging with household data, you have 12 continuous features (6 sums + 6 means for food, and 2 sums + 2 means for days away—total 14).

## 3. Modeling with CatBoost

1. Why CatBoost?  
   * Handles heterogeneous numeric features and any remaining missing values smoothly.
   * Captures non-linear patterns and interactions without manual feature expansion.
   * Robust to overfitting via built-in regularization and ordered boosting.
2. Pipeline Structure:  
   * SimpleImputer(median): Fills any remaining missing aggregations with a robust central value.
   * CatBoostRegressor: Learns complex associations between daily-life behaviors and MPCE.

## 4. Training & Saving

if os.path.exists(model\_path):

expense\_model = load\_model()

else:

expense\_model = HouseholdExpenseModel()

expense\_model.fit(df\_person\_train, df\_household\_train)

expense\_model.save\_model(model\_path)

* Conditional Load/Train: Ensures you don’t re-train every run—only when no saved model exists.
* fit(…):  
  + Merges person → household aggregates
  + Identifies the fourteen aggregated features
  + Trains CatBoost on X = aggregated\_features, y = TotalExpense

## 5. Prediction & Merging

predict(…) returns a DataFrame:  
  
HH\_ID, Personal\_level\_constant

1. where the constant is CatBoost’s estimate of MPCE based solely on meal/mobility behavior.

Merge into your feature tables exactly as before:  
  
processed\_df = pd.merge(processed\_df, pred\_test, on='HH\_ID', how='left')

processed\_df\_t = pd.merge(processed\_df\_t, pred\_train, on='HH\_ID', how='left')

* + Now both train and test sets gain a "Personal\_level\_constant" column.

1. Column Reordering  
   * Finally you swap positions of "having\_constant" and "online\_constant" to maintain consistent ordering—purely cosmetic but keeps your CSV schema tidy.

## 6. Theoretical Rationale

1. Multi-Stage Modeling  
   * You already have demographic and state-wise constants; now you add a behavioral constant.
   * Each captures a distinct “dimension” of MPCE:  
     1. Demographics (household composition)
     2. Assets & Connectivity (state-wise online/assert models)
     3. Individual Habits (meal/mobility model)
2. Stacked Feature Engineering  
   * By merging these first-stage predictions as features, your final learner can combine them with raw and engineered inputs (industry codes, composite scores) in a second-stage MPCE regressor (e.g., XGBoost).
   * This yields a model that’s both interpretable (through separate constants) and powerful (through a rich ensemble of signals).

**H. XGBoost training block:**1. Outlier Removal

upper = train\_df['Total\_Expense'].quantile(0.95)

for df in [train\_df, test\_df]:

df.drop(df[df['Total\_Expense'] > upper].index, inplace=True)

* Why? The top 5% of household expenses can be extreme (e.g., data entry errors or genuinely wealthy outliers).
* Effect: Trimming these reduces variance and prevents the model from overfitting to rare, extreme values.

## 2. Filtering Invalid Categories

train\_df = train\_df[train\_df.head\_gender != 3].drop(columns=['HH\_ID'])

test\_df = test\_df[test\_df.head\_gender != 3].drop(columns=['HH\_ID'])

* Why? A head\_gender code of 3 is presumably “unknown” or an error.
* Effect: Removing these rows ensures your model only sees valid gender categories; dropping HH\_ID discards the identifier for modeling.

## 3. Defining Sample Weights

train\_weights = train\_df.household\_size

test\_weights = test\_df.household\_size

* Why? When evaluation is weighted by household size, you’re effectively measuring total error across individuals rather than per-household.
* Effect: Larger households count more in your weighted metrics, aligning with the MPCE’s per-capita nature.

## 4. Column Type Grouping

binary\_cols = [...]

high\_card\_cols = [...]

int\_cols = [...]

float\_cols = [c for c in train\_df.columns if c not in ...]

* Why? Different encoding/scaling strategies suit different feature types:  
  + Binary (0/1) → no scaling, just ensure 0/1.
  + High-cardinality categoricals → target‐encode.
  + Integers & Floats → standardize.
* Effect: Prevents inappropriate transformations (e.g., scaling a 0/1 binary into non‐binary values).

## 5. Binary Encoding

for col in binary\_cols:

mapping = {uniques[0]:0, uniques[1]:1}

train\_df[col] = train\_df[col].map(mapping).astype(int)

test\_df[col] = test\_df[col].map(mapping).astype(int)

* Why? Ensures any arbitrary labels (e.g., 'Urban'/'Rural', 'Male'/'Female') are converted to 0/1.
* Effect: Leaves these features ready for a tree‐based model without injecting spurious ordinal assumptions.

## 6. Target Encoding for High-Cardinality Categoricals

te = TargetEncoder(cols=high\_card\_cols)

train\_df = te.fit\_transform(train\_df, y\_train)

test\_df = te.transform(test\_df)

* Why? One‐hotting dozens of states or districts would explode dimensionality.
* Effect: Each category is replaced by the mean MPCE of that category in the training set, capturing its typical expenditure level while keeping feature count low.

## 7. Scaling Continuous Features

scaler = StandardScaler()

train\_df[int\_cols + float\_cols] = scaler.fit\_transform(...)

test\_df[int\_cols + float\_cols] = scaler.transform(...)

* Why? Although XGBoost is tree‐based and doesn’t strictly require scaling, standardizing can help if you later combine with linear or distance‐based algorithms.
* Effect: Places all continuous features roughly on the same scale (mean=0, σ=1), which can improve optimization stability.

## 8. XGBoost Training with Tuned Hyperparameters

model = XGBRegressor(

n\_estimators=750,

max\_depth=10,

learning\_rate=0.05,

subsample=1.0,

colsample\_bytree=0.8,

random\_state=42,

objective='reg:squarederror'

)

model.fit(X\_train, y\_train)

* Key Components:  
  + n\_estimators & learning\_rate trade off tree count vs. step size.
  + max\_depth=10 allows moderately deep trees to capture complex interactions.
  + colsample\_bytree=0.8 injects randomness, reducing overfitting.
* Effect: Builds a powerful, non‐linear ensemble that can blend all your engineered features (demographic constants, composite scores, state‐wise constants, personal constants).

## 9. Evaluation Metrics (Weighted & Unweighted)

def eval\_metrics(y\_true, y\_pred, weights=None):

if weights is not None:

y\_true, y\_pred = y\_true \* weights, y\_pred \* weights

mae = mean\_absolute\_error(y\_true, y\_pred)

mse = mean\_squared\_error(y\_true, y\_pred)

rmse = np.sqrt(mse)

mape = np.mean(np.abs((y\_true - y\_pred) / ...)) \* 100

r2 = r2\_score(y\_true, y\_pred)

return mae, mape, rmse, mse, r2

* Unweighted: Measures average error *per household*.
* Weighted: Measures average error *per person* (since weights=household\_size), aligning with MPCE’s per‐capita interpretation.
* Effect: Gives you both a household‐level and person‐level view of model accuracy.

## 10. Why This Block Matters

* Final Stage: This is where all your feature engineering pays off.
* Holistic Learning: Combines demographic, survey‐to‐survey imputation constants, behavioral constants, and raw/engineered numerical and categorical features.
* Robust Validation: Outlier removal + weighted metrics ensure your model is not just over‐tuned to extreme households but also fair across population.

### 

### 

### ✨ Theoretical Takeaway

Your XGBoost model serves as a meta‐learner that optimally weighs and interacts all those first‐stage constants and raw features. By carefully preprocessing, encoding, and scaling according to feature type—and by evaluating both household‐ and person‐level errors—you’re maximizing predictive accuracy and interpretability in the complex task of MPCE imputation.