I need to train an XGBoost model for binary classification. I have 108 features (80 numerical, 28 categorical). My training set had 500k examples. can I use recursive feature elimination or permutation feature importance with XGBoost to select 50 top features Or should I do anova f test , mutual information , chi 2 test , correlation matrix, low variance test , high cardinality check for categorical features etc before applying wrapper methods

Short answer: with 108 features and 500k rows, you don't need heavy filter-based screening first. XGBoost handles this scale well. Do **light "data-hygiene" filters**, then use **model-based selection** (permutation or SHAP) inside cross-validation. RFE is possible but typically overkill and expensive.

What I'd do (order of operations)

1. **Hygiene (quick, safe filters)**

- Drop IDs/leakage columns, exact-duplicate/constant features, and features with extreme missingness.

- For categoricals: combine rare levels (e.g., `<1%`), fix typos, and decide encoding:

    - Prefer **XGBoost native categorical** (`enable_categorical=True`, `tree_method='hist'`/`'gpu_hist'`) if you can use a recent XGBoost.

    - Otherwise use **frequency or target encoding** (with K-fold smoothing to avoid leakage). One-hot is fine for low/medium cardinality; group the dummies as one "feature family" later.

- No scaling needed; XGB handles missing values.

  I generally **skip** ANOVA F, χ^2, and correlation pruning for tree models: they're univariate/linear, ignore interactions, and can discard useful features. Keep them only as quick sanity checks (e.g., remove quasi-constant, detect obvious non-informative columns). Mutual information can be a fast prescreen if you really want one, but don't rely on it to finalize selection.

2. **Train a strong baseline XGBoost**

- Use stratified CV + early stopping (e.g., 5-fold CV with an inner validation per fold).

- Tune only a few knobs first: `max_depth`, `eta`, `min_child_weight`, `subsample`, `colsample_bytree`, `n_estimators`, `reg_alpha`, `reg_lambda`.

- Pick a metric aligned with your problem (AUC-PR for imbalance; AUC-ROC otherwise).

3. **Model-based importance (inside CV)**

- Compute **SHAP mean |value|** per feature on the validation part of each fold **or permutation importance** on the held-out fold.

- Aggregate ranks across folds (median rank or mean importance).

- If you one-hot encoded, **sum importances across the dummies of each original categorical** so you don't accidentally "penalize" a split across many dummies.

4. **Select and verify**

- Try **Top-K = 50** by the aggregated importance; retrain the entire pipeline using only those features (with CV again).

- If metrics are statistically indistinguishable (or better), keep the smaller set; if they drop, keep more features (e.g., top-60/80). Plot performance vs K to choose a stable elbow.

5. **When to use RFE**

- RFE/Sequential Backward Selection with XGBoost works but is **computationally heavy** on 500k rows. If you must use it, drop features in **chunks** (e.g., remove 10 lowest-importance features per step until ~60, then one-by-one to 50) and do CV at each step. Otherwise, prefer the **one-shot SHAP/permutation ranking** above.

Notes & gotchas

- **Do all selection inside CV** to avoid look-ahead bias. Don't compute global importances on the full training set and then CV the reduced set.

- **Permutation importance** can underestimate the value of correlated features (importance gets spread). SHAP is usually more stable, but still check **stability across folds**.

- **High-cardinality categoricals**: native categorical or K-fold target encoding tends to beat wide one-hot. Always collapse ultra-rare levels to “Other”.

- XGBoost’s built-in “gain/cover/weight” importances are okay for a first glance, but **prefer SHAP or permutation** for selection decisions.

Minimal scikit-learn sketch (permutation route)

```
from xgboost import XGBClassifier
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import StratifiedKFold
from sklearn.inspection import permutation_importance
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
import numpy as np

num_cols = [...]          # 80 numeric
cat_cols = [...]          # 28 categorical

pre = ColumnTransformer([
    ("num", "passthrough", num_cols),
    ("cat", OneHotEncoder(handle_unknown="ignore",
min_frequency=0.01), cat_cols)  # or native categorical in XGB
])

xgb = XGBClassifier(
    tree_method="hist",    # or "gpu_hist" if available
    eval_metric="auc",
    n_estimators=1000,
    learning_rate=0.05,
    max_depth=8,
    subsample=0.8,
    colsample_bytree=0.8,
    n_jobs=-1
)
```

```
pipe = Pipeline([("pre", pre), ("clf", xgb)])

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
importances = []

for tr, va in kf.split(X, y):
    X_tr, X_va = X.iloc[tr], X.iloc[va]
    y_tr, y_va = y.iloc[tr], y.iloc[va]
    pipe.fit(X_tr, y_tr)
    r = permutation_importance(pipe, X_va, y_va, n_repeats=5,
random_state=42, n_jobs=-1)
    importances.append(r.importances_mean)

# Aggregate
mean_imp = np.mean(np.vstack(importances), axis=0)

# Map back to original features; if one-hot, sum dummy columns
belonging to the same original categorical
feature_names = pipe.named_steps["pre"].get_feature_names_out()
# ... group/sum by original categorical name here ...

# Choose top-50 and retrain/evaluate
```

Bottom line: Skip heavy univariate screening; do basic hygiene + strong XGBoost + SHAP/permutation-based ranking inside CV. Use the top-K curve to decide if 50 features is safe; only reach for RFE if you really need it and can afford the compute.