Given your dataset with 1,460 rows, 8 numerical features (well-correlated with the target `SalePrice`), and a list of 27 categorical features with varying numbers of unique categories, you’re planning to use linear regression and polynomial regression. Let’s break this down step-by-step to handle the categorical features and select the best ones for your models.

---

### Step 1: Encoding Categorical Features

For \*\*linear regression\*\* and \*\*polynomial regression\*\*, categorical features need to be encoded properly because these models assume numerical inputs and interpret them as having a linear (or polynomial) relationship with the target. Here’s why \*\*one-hot encoding\*\* is the best starting point for your case:

- \*\*Why One-Hot Encoding?\*\*

- Most of your categorical features (e.g., `Neighborhood`, `RoofStyle`, `Exterior1st`) are \*\*nominal\*\* (no inherent order). Using `LabelEncoder` would impose a false ordinal relationship (e.g., Neighborhood A = 0, B = 1, C = 2), which could mislead the model.

- One-hot encoding (via `pd.get\_dummies` or `sklearn.preprocessing.OneHotEncoder`) creates binary columns for each category, avoiding this issue. For example, `MSZoning` with 5 categories becomes 4 dummy variables (one dropped as the reference).

- Polynomial regression extends linear regression by adding polynomial terms, but it still relies on the same input structure, so one-hot encoding remains appropriate.

- \*\*Initial Encoding\*\*:

- Apply one-hot encoding to all 27 categorical features initially. This will generate a large number of dummy variables based on the `nunique()` counts you provided:

- Total unique categories = 5 + 2 + 4 + 4 + 2 + 5 + 3 + 25 + 9 + 8 + 5 + 8 + 6 + 8 + 15 + 16 + 4 + 5 + 6 + 6 + 5 + 2 + 4 + 7 + 3 + 9 + 6 = \*\*167\*\*.

- After one-hot encoding, you’ll get \*\*167 - 27 = 140 dummy variables\*\* (subtracting one reference category per feature).

- Combined with your 8 numerical features, this gives \*\*148 predictors\*\* before any polynomial terms.

- \*\*Problem\*\*: With 1,460 rows, 148 predictors is close to the upper limit of the heuristic of 10-20 observations per predictor (1,460 / 10 = 146, 1,460 / 20 = 73). For polynomial regression, adding squared or interaction terms will increase this further, risking overfitting and computational instability.

---

### Step 2: Reducing Dimensionality

To make this manageable and avoid overfitting, reduce the number of dummy variables:

1. \*\*Handle High-Cardinality Features\*\*:

- Features like `Neighborhood` (25), `Exterior1st` (15), `Exterior2nd` (16) contribute many dummy variables (24, 14, 15 respectively).

- Options:

- \*\*Group Rare Categories\*\*: For each feature, keep the top 5-10 most frequent categories and group the rest into an "Other" category. For `Neighborhood`, this might reduce 24 dummies to 4-9.

- \*\*Target Encoding\*\*: Replace categories with the mean `SalePrice` for that category (e.g., average price per neighborhood). This collapses each feature into one numeric column but risks leakage—use cross-validation if you try this.

- Recommendation: Start with grouping rare categories to keep one-hot encoding’s interpretability.

2. \*\*Low-Variation Features\*\*:

- Features with few categories and skewed distributions (e.g., `Street` with 2, `Utilities` with 2) might have little predictive power if one category dominates (e.g., 99% "Paved" for `Street`). Check their frequency with `value\_counts()`—if one category exceeds 90-95%, consider dropping them entirely.

3. \*\*Initial Reduction\*\*:

- Cap high-cardinality features at 5-10 categories:

- `Neighborhood`: Top 10 → 9 dummies (instead of 24).

- `Exterior1st`: Top 10 → 9 dummies (instead of 14).

- `Exterior2nd`: Top 10 → 9 dummies (instead of 15).

- Others unchanged unless similarly high.

- Recalculated total: Replace 24 + 14 + 15 = 53 with 9 + 9 + 9 = 27, reducing dummies by 26. New total ≈ 114 predictors (8 numeric + 106 dummies).

---

### Step 3: Feature Selection

With ~114 predictors after reduction, you’re still near the 1,460 / 10 = 146 limit, and polynomial regression will add more terms. Use feature selection to pick the best features:

1. \*\*Correlation with Target\*\*:

- After one-hot encoding, compute the correlation of each dummy variable with `SalePrice`. Keep the top 20-50 dummies with the strongest correlations (absolute value).

2. \*\*Regularization (Lasso)\*\*:

- Fit a \*\*Lasso regression\*\* (L1 penalty) on all 8 numeric + 106 dummy variables. Lasso shrinks less important coefficients to zero, effectively selecting features. Use cross-validation to tune the penalty parameter (`alpha`).

- This is ideal for linear regression and can guide polynomial regression too.

3. \*\*Variance Threshold\*\*:

- Remove dummy variables with very low variance (e.g., <0.01), as they indicate categories that are rare or uniform and add little signal.

4. \*\*Cross-Validation Performance\*\*:

- Test subsets of features (e.g., top 20, 50, 80 predictors) in a linear regression model using 5-fold cross-validation. Evaluate R² or RMSE on the validation set to find the sweet spot.

5. \*\*Domain Knowledge\*\*:

- Prioritize features with real estate relevance: `Neighborhood`, `BldgType`, `HouseStyle`, `KitchenQual`, `ExterQual`, `HeatingQC`, `SaleCondition`. These often drive house prices more than, say, `RoofMatl` or `Condition2`.

---

### Step 4: Polynomial Regression Considerations

- After selecting features for linear regression (e.g., 8 numeric + 20-40 dummies = 28-48 predictors), add polynomial terms (e.g., degree 2) using `sklearn.preprocessing.PolynomialFeatures`.

- This squares each feature and adds interactions (e.g., 48 predictors → ~1,176 terms), which is infeasible with 1,460 rows. Limit polynomial terms to:

- Only the 8 numeric features (well-correlated with `SalePrice`): 8 → 44 terms (8 linear + 8 squared + 28 interactions).

- Combine with the selected 20-40 dummies, keeping the total predictors < 80-100.

---

### Recommended Workflow

1. \*\*One-Hot Encode\*\*:

- Use `pd.get\_dummies` on all categorical features.

- For `Neighborhood`, `Exterior1st`, `Exterior2nd`, keep the top 10 categories each, grouping others as "Other".

2. \*\*Initial Model\*\*:

- Start with 8 numeric + all dummies (~106 after reduction).

- Fit a Lasso regression to select the top 20-40 categorical predictors.

3. \*\*Linear Regression\*\*:

- Final model: 8 numeric + 20-40 selected dummies (28-48 predictors).

- Evaluate with cross-validation.

4. \*\*Polynomial Regression\*\*:

- Take the 8 numeric features, add polynomial terms (degree 2), and combine with the 20-40 dummies.

- Total: ~60-80 predictors. Test performance.

5. \*\*Validation\*\*:

- Compare linear vs. polynomial regression using R², RMSE, and check for overfitting (train vs. test performance).

---

### Final Selection

- \*\*Keep\*\*: 8 numeric features (as they’re good), plus ~20-40 dummies from key categorical features like `Neighborhood`, `KitchenQual`, `ExterQual`, `SaleCondition`.

- \*\*Total Predictors\*\*: Aim for 30-80, well within 1,460 / 20 = 73 to 1,460 / 10 = 146.

- \*\*Why\*\*: Balances complexity, avoids overfitting, and leverages both numeric and categorical signals for `SalePrice`.

This approach ensures your models are practical and effective given your data size and goals!

Your question cuts to the heart of feature engineering and regression modeling, and it’s a sharp observation: If we’ve filtered dummies based on correlation (e.g., `|r| > 0.45`) and the original categorical grouping was somewhat arbitrary, why bother dropping additional dummies per feature? Why not treat the retained dummies (e.g., `MSZoning\_B`, `MSZoning\_C`, `MSZoning\_D`) as standalone binary features, ignoring their shared parent category (`MSZoning`)? Let’s explore this idea, its implications, and whether it’s a viable shift for your linear/polynomial regression models with 1,460 rows and `SalePrice` as the target.

---

### Your Argument: Treat Dummies as Separate Features

- \*\*Premise\*\*:

- You start with `pd.get\_dummies()` on all categorical features (e.g., 167 dummies from 27 features).

- Filter to keep only dummies with `|r| > 0.45`, say, yielding 30 dummies across various features (e.g., `MSZoning\_B`, `MSZoning\_C`, `Neighborhood\_X`, `KitchenQual\_Gd`, etc.).

- Instead of grouping them by their original feature and dropping one per group (e.g., one from `MSZoning\_B`, `C`), you propose treating each retained dummy as an independent binary predictor, regardless of its parent category.

- \*\*Reasoning\*\*:

- The original categorical feature (`MSZoning`) is just a human-imposed grouping. If `MSZoning\_B` and `MSZoning\_C` both strongly correlate with `SalePrice`, they could be seen as distinct "flags" (e.g., "is zone B" and "is zone C") rather than part of a single multi-level variable.

- Dropping one per feature assumes the retained dummies are still tied to `MSZoning`’s structure, but after filtering, that structure is broken—why enforce it?

---

### Does This Work Mathematically?

- \*\*Multicollinearity Check\*\*:

- In the full one-hot encoding (e.g., `MSZoning\_A`, `B`, `C`, `D`, `E`), the dummies are collinear because they sum to 1 per row. Dropping one resolves this.

- After filtering to `MSZoning\_B`, `MSZoning\_C`, `MSZoning\_D` (dropping `A`, `E`):

- They don’t sum to 1 across all rows anymore. When `B = C = D = 0`, the row is in `A` or `E`, not a retained category.

- These dummies are \*\*not perfectly collinear\*\* among themselves. Each is a binary indicator of its category vs. everything else (including the dropped group), so the design matrix remains full-rank.

- \*\*Conclusion\*\*: Keeping all filtered dummies without dropping one per feature doesn’t introduce multicollinearity in the traditional sense. The regression model can still estimate coefficients.

- \*\*Interpretation\*\*:

- `MSZoning\_B = 1` means "in zone B," and its coefficient is the effect of B vs. "not B" (which includes `C`, `D`, `A`, `E`).

- `MSZoning\_C = 1` means "in zone C," vs. "not C."

- When `B = C = D = 0`, the row is in `A` or `E`, absorbed into the intercept (plus numeric feature effects).

---

### Pros of Treating Dummies as Separate Features

1. \*\*Maximizes Information\*\*:

- You keep all dummies with `|r| > 0.45`, leveraging every strong signal without arbitrarily discarding one per feature. For example, if `MSZoning\_B` (0.5), `C` (-0.6), and `D` (0.47) all pass, keeping all three retains more predictive power than dropping `D`.

2. \*\*Simplifies Workflow\*\*:

- No need to group by parent feature or decide which dummy to drop. You just filter and go, treating the result as a flat set of binary predictors.

3. \*\*Arbitrary Categories Acknowledged\*\*:

- You’re right that `MSZoning` or `Neighborhood` are human constructs. If the data suggests `MSZoning\_B` and `MSZoning\_C` are independently useful, why force them into a single-feature framework? This approach respects the correlation filter’s output.

4. \*\*Works for Regression\*\*:

- Linear regression (and polynomial regression, if applied to numeric features only) can handle this. The coefficients reflect each dummy’s effect relative to its absence, adjusted for other predictors.

---

### Cons and Caveats

1. \*\*Interpretation Becomes Less Structured\*\*:

- Normally, `MSZoning\_B`, `C`, `D` are interpreted relative to a reference category (e.g., `A`). Without dropping one, their coefficients are relative to "not this category," which includes a mix of other retained and dropped categories. For example:

- `MSZoning\_B`’s coefficient is the effect of B vs. (`C`, `D`, `A`, `E`), not just vs. `A`.

- This can make it harder to compare effects within `MSZoning` directly (e.g., B vs. C), as there’s no common baseline.

2. \*\*Intercept Conflation\*\*:

- The intercept absorbs the effect of the dropped group (`A`, `E`) when all retained dummies are 0, but it also includes the baseline effect of numeric features. This muddies what the "baseline" really means compared to the standard approach.

3. \*\*Risk of Overlap\*\*:

- Retained dummies from the same feature aren’t mutually exclusive in their "off" states. For example, `MSZoning\_B = 0` includes rows in `C` and `D`, which are also modeled. This overlap might inflate variance or dilute individual effects slightly, though not enough to break the model.

4. \*\* departs from Convention\*\*:

- In regression, categorical features are typically encoded with \( k-1 \) dummies to maintain a clear reference. Your approach treats the filtered dummies as a "bag of features," which is more common in tree-based models (e.g., Random Forest) than linear regression.

5. \*\*Polynomial Regression Complexity\*\*:

- If you apply polynomial terms to all predictors (numeric + dummies), keeping more dummies increases the number of interaction terms exponentially. This might not scale well with 1,460 rows unless you limit polynomial features to numerics.

---

### Does the Original Grouping Matter?

- \*\*Your Point\*\*:

- The decision to group `MSZoning\_B`, `C`, `D` under `MSZoning` was arbitrary to begin with. If the correlation filter picks them out as strong predictors, why preserve that grouping?

- \*\*Counterpoint\*\*:

- The grouping isn’t entirely arbitrary—it reflects domain knowledge (e.g., `MSZoning` represents zoning types, `Neighborhood` reflects locations). Treating dummies as standalone loses this context, which might matter for interpretation or future analysis.

- However, if your goal is purely predictive (not explanatory), the grouping’s origin is less relevant—correlation with `SalePrice` drives the model.

---

### Can We Skip Dropping Dummies Entirely?

- \*\*Yes, You Can\*\*:

- After filtering to, say, 30 dummies with `|r| > 0.45`, you can treat them as 30 independent binary features alongside your 8 numeric features (38 predictors total). The model will:

- Estimate coefficients for each dummy relative to its absence.

- Absorb the dropped categories’ effects into the intercept.

- With 1,460 rows, 38 predictors fits well within the 10-20 observations-per-predictor heuristic (1,460 / 38 ≈ 38 to 73).

- \*\*When It’s Fine\*\*:

- Predictive focus: If you care about R² or RMSE, not coefficient interpretation, this works.

- Regularization: Lasso or Ridge can handle any subtle overlap between dummies, shrinking weak ones.

- \*\*When It’s Risky\*\*:

- Interpretability: If you want to explain "how zoning affects price," you lose the ability to compare within `MSZoning` cleanly.

- Overfitting: Without regularization, extra dummies might add noise, especially in polynomial regression.

---

### Refined Workflow (No Dropping)

```python

import pandas as pd

# Encode all dummies

df\_dummies = pd.get\_dummies(df3[categorical\_cols])

# Filter by correlation

correlations = df\_dummies.corrwith(df3['SalePrice']).abs()

selected\_dummies = correlations[correlations > 0.45].index

# Keep all filtered dummies as separate features

final\_dummies = df\_dummies[selected\_dummies]

# Combine with numeric features

final\_df = pd.concat([df3.select\_dtypes(include='number'), final\_dummies], axis=1)

# Fit model (e.g., LinearRegression or Lasso)

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(final\_df.drop('SalePrice', axis=1), final\_df['SalePrice'])

```

---

### My Take

- \*\*You’re Onto Something\*\*: Treating filtered dummies as separate features is a valid shift, especially since:

- The correlation filter already breaks the original categorical structure.

- Multicollinearity isn’t an issue post-filtering (unlike full one-hot encoding).

- It maximizes retained signal, which suits your small dataset.

- \*\*Why I Suggested Dropping\*\*:

- Habit from standard practice: \( k-1 \) dummies per feature ensures a clear reference and aligns with regression’s roots in categorical analysis.

- Interpretability: Grouping by feature preserves context (e.g., "effect of zoning type"), which might matter for real estate insights.

- \*\*Best for You\*\*:

- \*\*If Predictive Power Is Key\*\*: Skip dropping additional dummies. Keep all `|r| > 0.45` dummies as standalone features (e.g., 20-40 predictors + 8 numeric). Use Lasso to refine if needed.

- \*\*If Interpretation Matters\*\*: Drop one per feature after filtering (e.g., weakest `|r|`) to maintain some categorical coherence.

- \*\*Final Recommendation\*\*:

- Try your way: Keep all filtered dummies as separate features. It’s simpler, leverages your correlation insight, and works mathematically. Test with cross-validation (e.g., 5-fold RMSE) to ensure it doesn’t overfit vs. dropping one per feature. For polynomial regression, limit terms to numeric features to keep complexity in check.

You’ve convinced me this is a practical twist—let the data speak through correlation, not arbitrary groupings!

You’re absolutely correct to dig into this, and your question sharpens the discussion even further. Let me clarify what I meant by "unless you enforce it" and confirm your intuition about whether the model "knows" a dummy was removed to serve as a baseline. You’re right in a key sense, and I’ll explain how this ties into your strategy for linear/polynomial regression with filtered dummies.

---

### What I Meant by "Unless You Enforce It"

- \*\*Context\*\*:

- After applying `pd.get\_dummies()` to `MSZoning` (e.g., `A`, `B`, `C`, `D`, `E` → 5 dummies) and filtering to keep only `MSZoning\_B`, `C`, `D` (based on `|r| > 0.45`), the resulting columns in your dataset are just `MSZoning\_B`, `MSZoning\_C`, `MSZoning\_D`. The model sees these as three binary predictors, and their origin as part of `MSZoning` is no longer explicit in the design matrix \( X \).

- \*\*"Enforce It" Meaning\*\*:

- I was referring to a deliberate preprocessing choice where you, as the data scientist, impose the structure of the original categorical feature (`MSZoning`) back onto the filtered dummies. For example:

- \*\*Dropping one per feature\*\*: If you filter to `B`, `C`, `D` and then drop `D` (leaving `B`, `C`), you’re mimicking the standard one-hot encoding practice of using \( k-1 \) dummies (here, 2 out of 3 retained categories). This implicitly treats `A`, `D`, `E` (or at least `D`) as a reference group, preserving the idea that `B` and `C` are levels of a single variable (`MSZoning`).

- \*\*Naming or metadata\*\*: You could also group them in documentation or code comments as "from `MSZoning`," but this doesn’t affect the model—just your interpretation.

- \*\*Without Enforcement\*\*:

- If you keep `B`, `C`, `D` as is (no additional drop), the model treats them as separate binary features, and the parent category (`MSZoning`) is irrelevant to the math. The prefix `MSZoning\_` is just a label—it’s not encoded in \( X \) or used by the regression algorithm.

---

### Does the Model Know a Dummy Was Removed as a Baseline?

- \*\*Your Point\*\*:

- You’re saying that even if you drop one dummy (e.g., `D`) from the filtered set (`B`, `C`, `D`) to "play the role of a baseline," the model doesn’t inherently "know" this was done to establish a baseline for `MSZoning`. It just sees `B` and `C` as predictors, right?

- \*\*Answer\*\*:

- \*\*Yes, you’re right\*\*: The model itself has no concept of a "baseline" or why a dummy was removed. The act of dropping one (e.g., `D`) is a human decision during preprocessing, not something the model understands. After preprocessing, \( X \) contains only `MSZoning\_B` and `MSZoning\_C`, and the model estimates coefficients \( \beta\_B \) and \( \beta\_C \) based on their values (0s and 1s) and their relationship to `SalePrice`, adjusted for all other predictors.

- The "baseline" idea (e.g., `A/D/E` or `D`) exists only in \*your interpretation\* of the coefficients:

- With `B`, `C`: \( \beta\_B \) is the effect of `B = 1` vs. `B = 0` (where `B = 0` includes `C`, `D`, `A`, `E`), and similarly for \( \beta\_C \).

- The model doesn’t "know" `D` was dropped to represent a reference—it just works with what’s in \( X \).

- \*\*Mathematical View\*\*:

- The design matrix \( X \) is what it is. Whether you keep `B`, `C`, `D` or drop `D` to get `B`, `C`, the model optimizes \( y = X\beta + \epsilon \) agnostically. The "baseline" concept is a human overlay to make sense of \( \beta \) values, not a property of the algorithm.

---

### Implications for Your Strategy

- \*\*Your Approach (Keep All Filtered Dummies)\*\*:

- Filter to `MSZoning\_B`, `C`, `D` (all `|r| > 0.45`) and keep them as separate features.

- Model sees: Three columns in \( X \), each a binary predictor.

- Coefficients: \( \beta\_B \), \( \beta\_C \), \( \beta\_D \) reflect the effect of each being 1 vs. 0, with no explicit reference category enforced. When `B = C = D = 0`, the effect of `A` or `E` is absorbed into the intercept (mixed with numeric feature baselines).

- \*\*No Knowledge of Parent\*\*: The model doesn’t care that they came from `MSZoning`—it treats them like `Neighborhood\_X` or `Rooms`.

- \*\*Alternative (Drop One)\*\*:

- Filter to `B`, `C`, `D`, then drop `D`.

- Model sees: Two columns (`B`, `C`).

- Coefficients: \( \beta\_B \) and \( \beta\_C \) are now relative to `B = C = 0` (which includes `D`, `A`, `E`). You interpret this as a baseline, but the model doesn’t "know" `D` was dropped for that reason—it just fits the reduced \( X \).

- \*\*Key Insight\*\*:

- The model has no memory of `MSZoning` or the filtering process. Whether you drop one or not, it processes the columns you give it. The "baseline" is a preprocessing artifact, not a model feature.

---

### Am I Right Here?

- \*\*Yes, 100%\*\*:

- The model doesn’t "know" a dummy was removed to play the role of a baseline. That’s a human construct. If you drop `MSZoning\_D`, the model doesn’t register `D` (or `A/D/E`) as a special reference—it simply excludes `D` from \( X \) and estimates \( \beta \) for what remains. The baseline exists in your mind when you interpret \( \beta\_B \) as "B vs. not B (including D)," but the model is oblivious to this narrative.

- Similarly, if you keep `B`, `C`, `D`, there’s no baseline enforced—the model just fits all three, and their "off" states (0s) blend into the intercept.

---

### Why "Enforce It" Then?

- \*\*Human Choice\*\*:

- "Enforce it" means you \*choose\* to structure the features to reflect `MSZoning`’s categorical nature (e.g., by dropping one to mimic one-hot encoding). This isn’t for the model’s benefit—it’s for yours:

- \*\*Interpretability\*\*: "Effect of zone C vs. zone D" (if `D` is the reference) is clearer than "effect of zone C vs. not C."

- \*\*Convention\*\*: Aligns with how categorical variables are typically handled in regression.

- Without this enforcement, the model happily treats `B`, `C`, `D` as unrelated binaries, which is fine for prediction but less tied to `MSZoning` as a concept.

- \*\*Model’s Ignorance\*\*:

- Whether you enforce it or not, the model doesn’t care. It’s a black box crunching \( X \) and \( y \). Your preprocessing dictates what it sees, not its understanding.

---

### Final Confirmation for Your Strategy

- \*\*Your Way (No Dropping)\*\*:

- Keep `MSZoning\_B`, `C`, `D` as separate features after filtering.

- Model treats them as: Three independent predictors, no parent category implied.

- Correct: The model has no idea they came from `MSZoning` or that `A` and `E` were dropped. It fits them as given.

- \*\*Why It Works\*\*:

- No multicollinearity (filtered set isn’t exhaustive).

- Coefficients are interpretable as "effect of this category vs. its absence," which suits your correlation-based selection.

- \*\*No Need to Drop\*\*:

- You’re right—dropping one per feature isn’t mathematically necessary post-filtering. It’s a choice to preserve categorical structure, but since the model doesn’t know or need that structure, skipping it is valid.

---

### Conclusion

- \*\*You’re Right\*\*: The model has no idea a dummy was removed to act as a baseline, nor does it know about `MSZoning` after filtering. It treats all predictors in \( X \) equally, whether they’re filtered dummies or numeric features.

- \*\*"Unless You Enforce It"\*\*: Means you can force the model to reflect `MSZoning`’s structure by dropping one, but this is optional and for your benefit, not the model’s.

- \*\*Your Strategy\*\*: Keeping all filtered dummies (`|r| > 0.45`) as separate features is perfectly sound. The model will process them as standalone predictors, and you’re free of unnecessary constraints.

You’ve got this nailed—proceed with confidence!