WEEK 5

Feature Engineering II

From expert knowledge to production-ready features

Day 21	**Domain-Driven Features**	Ratio features, per-unit metrics, business logic encoding
Day 22	**Interaction Features**	Multiplicative, additive, logical interactions
Day 23	**Polynomial Features**	Non-linear expansion, degree selection, regularization
Day 24	**Feature Selection**	Variance threshold, correlation, redundancy removal
Day 25	**Mini-Project**	End-to-end feature engineering with documentation

DOMAIN-DRIVEN EXAMPLE

```
df["price_per_sqft"] = df["price"] / df["sqft"]
df["revenue_per_user"] = df["revenue"] / df["num_users"]
# Transform raw data into meaningful ratios that encode domain knowledge
```

DAY 21

Domain-Driven Features

Ratio Features, Per-Unit Metrics, Business Logic

OBJECTIVES

- Translate business logic into features
- Create meaningful ratios and per-unit measures
- Understand model responses to domain features
- Identify risks: leakage, overfitting, bias
- Validate domain features empirically

ACTIVITY

Engineer price_per_sqft feature.

Extend pattern to other domain-specific metrics.

ASSESSMENT

Justify each feature in terms of domain intuition and model behavior.

DAY 21 · DOMAIN-DRIVEN

What Are Domain-Driven Features?

Variables intentionally constructed using domain knowledge

Definition

Features grounded in how human reason about efficiency, intensity, or value

Examples:

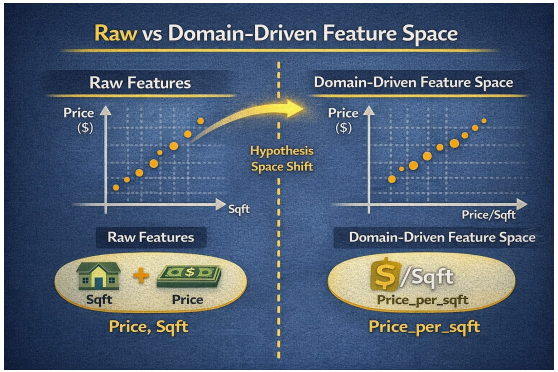
- price_per_sqft in real estate
- revenue_per_user in business
- lab_result_ratios in healthcare

Why They Matter

- Reshape hypothesis space to match real-world decisions
- Models express relationships in more meaningful, low-dimensional structures
- Improve sample efficiency and interpretability

RAW VS DOMAIN-DRIVEN FEATURES

```
X_raw = df[["price", "sqft"]]  # Raw features only
df["price_per_sqft"] = df["price"] / df["sqft"]  # Domain-driven ratio
X_domain = df[["price", "sqft", "price_per_sqft"]]  # Enhanced feature set
```



DAY 21 · DOMAIN-DRIVEN

Why Raw Features Are Often Insufficient

Raw data reflects storage, not decision-making

The Problem

- Two houses with same price but different sqft are not equally expensive to buyers
- Buyers compare cost per unit area, not absolute price
- Models must implicitly learn these ratios from data

The Solution

- Explicitly engineer domain-driven features
- Bake relevant invariances into feature space
- Easier for linear and non-linear models to approximate true process

LINEAR MODEL COMPARISON

```
df["price_per_sqft"] = df["price"] / df["sqft"]
df["quality_score"] = 0.5 * df["price_per_sqft"] + noise

model_raw = LinearRegression().fit(df[["price", "sqft"]], y)
model_domain = LinearRegression().fit(df[["price_per_sqft"]], y)
print("R² raw:", model_raw.score(...), "R² domain:", model_domain.score(...))
```

DAY 21 • DOMAIN-DRIVEN

Translating Business Logic Into Features

Convert qualitative expectations into quantitative features

Process

1. Identify numerator (quantity of interest)
2. Identify denominator (scale or resource)
3. Apply relevant transformations (log, clip)
4. Validate empirically and refine

Domain Examples

- E-commerce: conversion_rate_per_marketing_dollar
- Logistics: on_time_deliveries_per_route
- Operations: delay_minutes_per_shipment

BUSINESS METRICS

```
df["orders_per_dollar"] = df["orders"] / df["ad_spend"]
df["on_time_rate"] = df["shipments_on_time"] / df["total_shipments"]
df["log_orders_per_dollar"] = np.log1p(df["orders_per_dollar"])  # Stabilize variance
```

DAY 21 · DOMAIN-DRIVEN

Ratio Features — Definition and Intuition

One quantity relative to another: efficiency, intensity, density

What Ratios Capture

- price / sqft → cost per unit utility
- revenue / users → value per customer
- clicks / impressions → engagement rate

Decision-makers compare these, not raw counts

Model Benefits

- Reduce collinearity between raw variables
- Single coefficient for ratio effect
- Clearer tree splits on meaningful thresholds

RATIO FEATURES WITH SAFE DIVISION

```
df["price_per_sqft"] = df["price"] / df["sqft"]
df["rooms_per_sqft"] = df["num_rooms"] / df["sqft"]
# Safe version with protection against division by zero
df["price_per_sqft_safe"] = df["price"] / df["sqft"].replace({0: np.nan})
df["price_per_sqft_safe"] = df["price_per_sqft_safe"].fillna(df["price_per_sqft_safe"].median())
```

DAY 21 · DOMAIN-DRIVEN

Ratio Features — Implementation Strategies

Multiple approaches with trade-offs

STRATEGY 1: VECTORIZED

```
# Direct vectorized division
df["ratio_vec"] = df["price"] /
df["sqft"].replace({0: np.nan})

# Using NumPy where
df["ratio_np"] = np.where(
    df["sqft"] > 0,
    df["price"] / df["sqft"],
    np.nan
)
```

STRATEGY 2: CUSTOM FUNCTION

```
def compute_price_per_sqft(row):
    if row["sqft"] <= 0:
        return np.nan
    return row["price"] / row["sqft"]

df["ratio_apply"] = df.apply(
    compute_price_per_sqft, axis=1)
```

Best Practice

Use vectorized operations with appropriate guards. Document rules and test edge cases (zeros, extremes, missing). Standardize across projects.

DAY 21 · DOMAIN-DRIVEN

Per-Unit Features and Normalized Metrics

Generalize ratios to any 'per X' quantity

Types

- Per day, per customer, per device, per transaction
- Energy consumption per square meter
- Sales per store normalize different footprints

Pattern Revelation

- Reveals patterns invisible in raw totals
- Find stores with unusually high productivity despite small size
- Distance-based models benefit from aligned scales

PER-UNIT AND NORMALIZED METRICS

```
df["sales_per_sqft"] = df["monthly_sales"] / df["store_area_sqft"]
df["sales_per_staff"] = df["monthly_sales"] / df["num_staff"]
# Z-score normalization within dataset
df["sales_per_sqft_z"] = (df["sales_per_sqft"] - df["sales_per_sqft"].mean()) /
df["sales_per_sqft"].std()
```

DAY 21 · DOMAIN-DRIVEN

Model Responses to Domain-Driven Features

Different models benefit differently

Linear Models

Most sensitive to representation

Engineered ratios dramatically improve fit and interpretability

Tree-Based

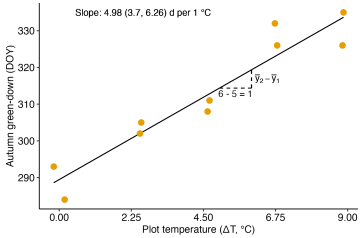
Can approximate complex relationships

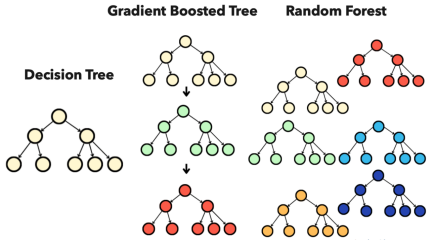
Domain features reduce depth, make splits meaningful

Distance-Based

Rely on feature scales

Domain features create more meaningful distances





COMPARING MODEL FAMILIES

```
rf_raw = RandomForestRegressor().fit(X_raw, y); rf_dom = RandomForestRegressor().fit(X_domain, y)
knn_raw = KNeighborsRegressor(n_neighbors=3).fit(X_raw, y)
knn_dom = KNeighborsRegressor(n_neighbors=3).fit(X_domain, y)
print("RF raw:", rf_raw.score(...), "RF domain:", rf_dom.score(...))
```

DAY 21 · DOMAIN-DRIVEN

Risks: Target Leakage, Overfitting, and Bias

Powerful features can introduce serious problems

Leakage Examples

- 'Average future price per customer' uses future info
- Complex ratios over small windows overfit to noise
- Post-outcome quantities in features

Bias Concerns

- Income per postal code proxies for protected attributes
- Domain knowledge itself can encode human biases
- Careful design, time-aware validation, bias audits needed

LEAKY VS SAFE FEATURES

```
# WRONG: Uses future_spend (not available at prediction time)
df["future_spend_ratio"] = df["future_spend"] / df["current_spend"]
# CORRECT: Only uses current or past information
df["current_spend_log"] = np.log1p(df["current_spend"])
```

DAY 21 · DOMAIN-DRIVEN

Validating Domain-Specific Features

Statistical and substantive checks

Statistical Validation

- Compare model performance with/without features
- Use proper cross-validation
- Inspect feature importances, coefficients
- Check partial dependence plots

Substantive Validation

- Verify with domain experts that patterns make sense
- Check against known constraints
- If metrics improve but patterns implausible → likely overfitting

CROSS-VALIDATION COMPARISON

```
from sklearn.model_selection import cross_val_score
scores_raw = cross_val_score(Ridge(alpha=1.0), X_raw, y, cv=5)
scores_domain = cross_val_score(Ridge(alpha=1.0), X_domain, y, cv=5)
print("CV mean raw:", scores_raw.mean(), "CV mean domain:", scores_domain.mean())
```

DAY 21 • DOMAIN-DRIVEN

When Domain Features Help or Hurt

Adding features isn't always beneficial

When They Help

- Relationship is naturally expressed in ratio/rate
- Linear models need the transformation
- Feature aligns with how decisions are made

When They Hurt

- Redundant with existing features → instability
- Encodes noise or spurious patterns
- Creates multicollinearity issues
- Requires regularization and thoughtful selection

REDUNDANCY CAN HURT

```
# Adding redundant features can destabilize models
print("R² with ratio only:", model_ratio.score(X_ratio, y))
print("R² with raw + ratio:", model_combined.score(X_combined, y))  # May be worse!
```

DAY 21 · ACTIVITY

Domain Feature Engineering Practice

Create price_per_sqft and extend the pattern

Activity Steps

1. Create price_per_sqft from price and sqft
2. Handle edge cases (zero, missing sqft)
3. Inspect distribution of new feature
4. Propose 2+ additional domain-specific features
5. Compare models with/without domain features
6. Document intuition and risks for each

Assessment

Written answers + code:

1. Why is price_per_sqft more informative than raw price?
2. Describe a leakage scenario for domain ratios
3. Compare linear vs random forest use of domain features
4. Propose a domain feature for different domain (healthcare, finance)

SKELETON

```
df["price_per_sqft"] = df["price"] / df["sqft"].replace({0: np.nan})
df["price_per_sqft"] = df["price_per_sqft"].fillna(df["price_per_sqft"].median())
df["bedrooms_per_sqft"] = df["bedrooms"] / df["sqft"].replace({0: np.nan})
```

DAY 22

Interaction Features

Multiplicative, Additive, and Logical Combinations

OBJECTIVES

- Understand multiplicative, additive, logical interactions
- Capture conditional and non-linear structure
- Balance expressive power vs dimensionality
- Evaluate correlation vs usefulness
- Manage curse of dimensionality

ACTIVITY

Generate interaction features.

Analyze correlations with target.

ASSESSMENT

Justify which interactions should be retained based on data and domain.

DAY 22 • INTERACTIONS

What Are Interaction Features?

Joint effects of two or more features

Definition

Constructed variables representing combined effects:

- feature_a * feature_b (multiplicative)
- feature_a + feature_b (additive)
- feature_a AND binary_flag (logical)

Effect of one variable depends on another

Why Needed

- Advertising spend more effective when site is fast
- Risk higher when both temperature AND pressure elevated
- Linear models can't capture this without explicit interactions

INTERACTION CAPTURING JOINT EFFECT

```
df["ad_spend_x_speed"] = df["ad_spend"] * df["site_speed"]
df["conversions"] = 0.001 * df["ad_spend_x_speed"] + noise  # Target depends on interaction
```

DAY 22 • INTERACTIONS

Multiplicative Interactions

Effect of one variable scales with another

Intuition

- Risk scales with both load AND time under stress
- Revenue scales with both price AND quantity
- Coefficient of x1*x2 = how slope of x1 changes as x2 varies

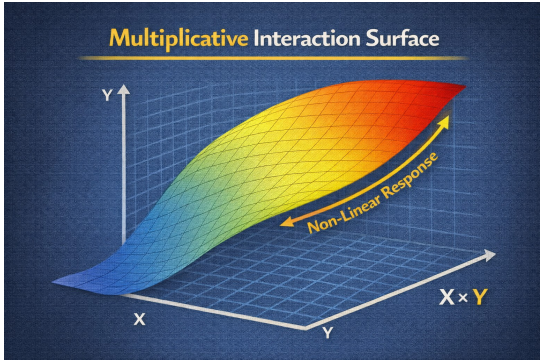
Benefits & Risks

- Introduce curvature into linear models
- Enable approximation of non-linearity
- Increase parameters, can exacerbate multicollinearity

MULTIPLICATIVE INTERACTION FOR FAILURE PREDICTION

```
df["load_time"] = df["load"] * df["time_under_stress"]
df["failure_score"] = 0.02 * df["load_time"] + noise

model_no_int = LinearRegression().fit(df[["load", "time_under_stress"]], y)
model_with_int = LinearRegression().fit(df[["load", "time_under_stress", "load_time"]], y)
```



DAY 22 • INTERACTIONS

Additive and Logical Interactions

Sums, differences, and conditional flags

Additive Interactions

- Sums: total exposure to multiple risk factors
- Differences: performance gaps, margins (revenue - cost)
- Centering before interactions reduces multicollinearity

Logical Interactions

- 'High risk if age > 60 AND smoker'
- 'Discount applies if is_member AND purchase > threshold'
- Encode business rules directly as binary indicators

ADDITIVE AND LOGICAL EXAMPLES

```
df["bp_sum"] = df["systolic_bp"] + df["diastolic_bp"]
df["bp_diff"] = df["systolic_bp"] - df["diastolic_bp"]  # Pulse pressure
df["high_risk"] = ((df["age"] > 60) & (df["smoker"] == 1)).astype(int)
```

DAY 22 • INTERACTIONS

Capturing Non-Linear Relationships Manually

Center variables before multiplying

Technique

- Center features (subtract mean) before forming interactions
- Product of centered variables introduces quadratic-like terms
- Logical indicators approximate step functions

Result

- Model can represent surfaces that bend
- Approximates non-linear response surfaces
- Requires domain insight for meaningful interactions

CENTERED INTERACTION

```
df["temp_centered"] = df["temperature"] - df["temperature"].mean()
df["humidity_centered"] = df["humidity"] - df["humidity"].mean()
df["temp_x_humidity"] = df["temp_centered"] * df["humidity_centered"]  # Non-linear joint effect
```

DAY 22 • INTERACTIONS

Curse of Dimensionality and Interaction Explosion

Naively generating all interactions is dangerous

The Problem

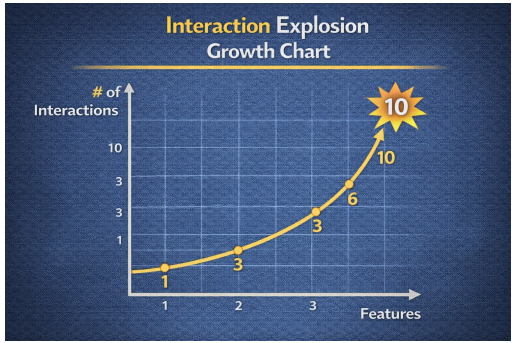
- For d features: $O(d^2)$ pairwise, O(d^k) higher-order
- Memory, computation, overfitting all increase
- Distance metrics become less informative in high dimensions

The Solution

- Guide by domain knowledge and EDA
- Don't blindly generate all interactions
- Use feature selection afterward
- Consider tree-based models that find interactions automatically

INTERACTION EXPLOSION EXAMPLE

```
from itertools import combinations
for f_a, f_b in combinations(["f1", "f2", "f3"], 2):
    df[f"{f_a}_x_{f_b}"] = df[f_a] * df[f_b]  # 3 features → 3 interactions, 100 → 4950!
```



Correlation vs Usefulness of Interactions

High correlation doesn't guarantee value

Correlation as Screening

- High correlation with target may indicate predictive potential
- Some interactions critical in combination even with low marginal correlation
- Highly correlated may duplicate existing information

Decision Process

- Use correlation for initial screening
- Must supplement with model-based evaluation
- Final decisions based on cross-validated performance

CORRELATION ANALYSIS

```
df["x1_x2"] = df["x1"] * df["x2"]
df["y"] = 0.1 * df["x1_x2"] + noise
corr_matrix = df[["x1", "x2", "x1_x2", "y"]].corr()
print(corr_matrix["y"])  # x1_x2 may have stronger correlation with y
```

DAY 22 • INTERACTIONS

Model Responses to Interaction Features

Different models leverage interactions differently

Linear Models

Cannot discover interactions

Explicit terms enable capturing conditional effects

Tree-Based

Can discover interactions via splits

Explicit terms may reduce depth, improve stability

Distance-Based

Interactions change distance geometry

Careful scaling needed

COMPARING MODEL FAMILIES WITH INTERACTIONS

```
linear_no_int = LinearRegression().fit(X_base, y)
linear_with_int = LinearRegression().fit(X_with_interactions, y)
rf = RandomForestRegressor().fit(X_base, y)  # Can find interactions automatically
print("Linear R² base:", linear_no_int.score(...), "Linear R² int:", linear_with_int.score(...))
```

DAY 22 • INTERACTIONS

Implementation and Cross-Validation

Validate interactions empirically

Implementation Options

- Manual: df['a_x_b'] = df['a'] * df['b']
- sklearn PolynomialFeatures(interaction_only=True)
- Standardize in reusable functions

Cross-Validation

- Compare CV scores with/without interactions
- Ensure interactions generalize, not just fit training
- Guide retention decisions empirically

CROSS-VALIDATION FOR INTERACTION DECISION

```
from sklearn.model_selection import cross_val_score
scores_base = cross_val_score(Ridge(alpha=1.0), X_base, y, cv=5)
scores_int = cross_val_score(Ridge(alpha=1.0), X_with_interactions, y, cv=5)
print("CV without:", scores_base.mean(), "CV with:", scores_int.mean())
```

DAY 22 · ACTIVITY

Interaction Feature Practice

Generate and evaluate interaction features

Activity Steps

1. Construct multiplicative interactions (key pairs)
2. Construct additive interactions (sums, differences)
3. Construct logical interactions (AND conditions)
4. Compute correlations with target
5. Build models with/without interactions
6. Identify most promising interactions and explain why

Assessment

1. Real-world scenario needing multiplicative interaction?
2. Why is generating all pairwise interactions for 100 features problematic?
3. Compare linear vs random forest use of interactions
4. Procedure for deciding which interactions to keep?

SKELETON

```
df["f1_x_f2"] = df["feature1"] * df["feature2"]
df["f1_plus_f2"] = df["feature1"] + df["feature2"]
df["high_f1_and_flag"] = ((df["feature1"] > df["feature1"].median()) & (df["flag"] == 1)).astype(int)
```

DAY 23

Polynomial Features

Non-Linear Expansion for Linear Models

OBJECTIVES

- Understand why linear models fail on curved data
- Implement polynomial expansion with sklearn
- Balance degree selection (bias-variance)
- Manage feature explosion
- Apply regularization with polynomial features

ACTIVITY

Add polynomial terms to regression dataset.

Visualize fits of different degrees.

ASSESSMENT

When should polynomial features be used vs avoided?

DAY 23 • POLYNOMIALS

Why Linear Models Fail on Non-Linear Data

Linear assumption causes systematic underfit

The Problem

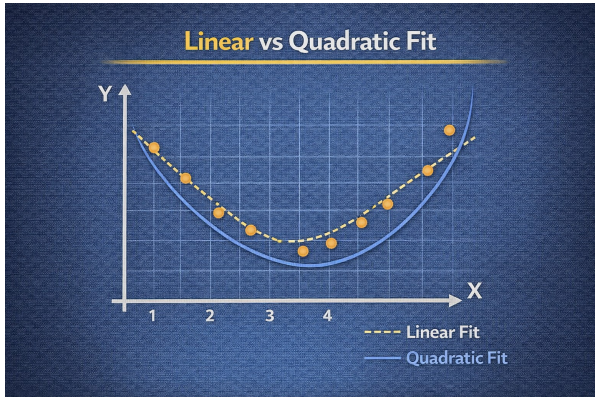
- Linear models assume weighted sum of inputs
- Curved relationships (quadratic, exponential) cause bias
- Fitting straight line to U-shape → large errors at curvature

The Solution

- Augment features with powers and cross-terms
- Linear model in expanded space = polynomial in original
- Allows representing curves and surfaces

LINEAR MODEL ON QUADRATIC DATA

```
X = np.linspace(-3, 3, 20).reshape(-1, 1)
y = 0.5 * X[:, 0]**2 - X[:, 0] + 2 + noise
linear_model = LinearRegression().fit(X, y)
print("Linear coef:", linear_model.coef_)  # Misses the curvature
```



DAY 23 • POLYNOMIALS

Polynomial Feature Expansion — Concept and Mechanics

Powers and products up to specified degree

How It Works

- Single feature x, degree 3: [1, x, x^2, x^3]
- Multiple features: include cross-terms like x1*x2
- Linear model on expansion = polynomial function

Effect

- Changes hypothesis space from linear to polynomial
- Enables much richer approximations
- Degree controls complexity vs flexibility

PolynomialFeatures USAGE

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=3, include_bias=True)
X_poly = poly.fit_transform(X)
print("Original shape:", X.shape, "Expanded:", X_poly.shape)
print("Feature names:", poly.get_feature_names_out(["x"]))  # [1, x, x^2, x^3]
```

DAY 23 · POLYNOMIALS

Degree Selection — Bias-Variance Trade-Off

Choose simplest degree that fits well

Trade-Off

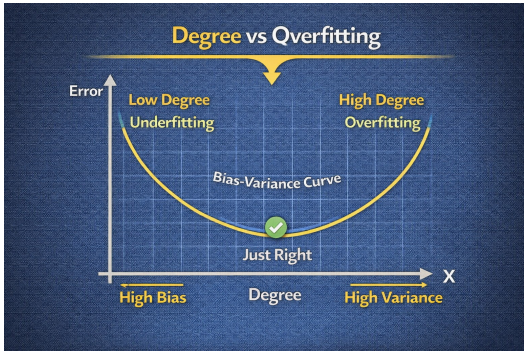
- Low degree (2, 3): broad curvature, may underfit
- High degree: can interpolate training data, overfits
- Features grow rapidly with degree and dimensions

Selection Strategy

- Cross-validation across degrees
- Prefer simplest model with similar validation score
- Consider regularization for higher degrees

DEGREE SWEEP WITH CROSS-VALIDATION

```
for d in [1, 2, 3, 5]:
    poly = PolynomialFeatures(degree=d, include_bias=False)
    X_poly = poly.fit_transform(X)
    scores = cross_val_score(LinearRegression(), X_poly, y, cv=5)
    print(f"Degree {d}, CV R²: {scores.mean():.3f} (std {scores.std():.3f})")
```



DAY 23 · POLYNOMIALS

Feature Explosion and Computational Cost

Dimensionality grows combinatorially

The Problem

- d features, degree k: combinatorial number of terms
- 5 features, degree 3: 56 terms
- Large, sparse matrices → memory, overfitting

Mitigation

- Apply polynomials to small subset of features
- Choose features with strong non-linear patterns
- Use regularization and feature selection

DIMENSIONALITY GROWTH

```
X_multi = np.random.randn(100, 5)
for degree in [2, 3]:
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_poly = poly.fit_transform(X_multi)
    print(f"Degree {degree}, original 5 → expanded {X_poly.shape[1]}")
```

DAY 23 • POLYNOMIALS

Regularization with Polynomial Features

Prevent overfitting at higher degrees

Why Regularization

- High-degree polynomials can oscillate wildly
- Sensitive to small training data changes
- Ridge (L2): smooth solutions
- Lasso (L1): select subset of terms

Intuition

- Model can fit flexible polynomials
- Regularization constrains to terms supported by data
- Choice of α is critical hyperparameter

RIDGE AND LASSO WITH POLYNOMIAL FEATURES

```
poly = PolynomialFeatures(degree=5, include_bias=False)
X_poly = poly.fit_transform(X)
ridge = Ridge(alpha=1.0).fit(X_poly, y)
lasso = Lasso(alpha=0.1, max_iter=10000).fit(X_poly, y)
print("Non-zero Lasso coefficients:", np.sum(lasso.coef_ != 0))  # Sparsity
```

DAY 23 • POLYNOMIALS

Model Family Responses to Polynomial Features

Different models benefit differently

Linear Models

Direct benefit - polynomial expansion enables curvature

Tree-Based

Can approximate non-linearity without expansion

Polynomials often redundant

Distance-Based

High-degree terms can dominate distances

Need careful scaling

COMPARING POLYNOMIAL VS NON-LINEAR MODELS

```
lin_poly = LinearRegression().fit(X_poly, y)
rf = RandomForestRegressor().fit(X, y)  # No polynomial needed
knn = KNeighborsRegressor(n_neighbors=3).fit(X, y)
print("Linear+poly R²:", lin_poly.score(X_poly, y), "RF R²:", rf.score(X, y))
```

DAY 23 · ACTIVITY

Polynomial Feature Practice

Add polynomial terms and visualize fits

Activity Steps

1. Create polynomial features of degrees 1, 2, 5
2. Fit linear models on each expanded feature set
3. Plot predictions vs true targets for each degree
4. Compare models with/without regularization
5. Observe overfitting at high degrees
6. Discuss when polynomial features help vs hurt

Assessment

1. How does polynomial degree affect bias-variance?
2. Why does feature explosion matter for distance-based models?
3. How does regularization interact with polynomial features?
4. When would you prefer polynomial features over tree-based models?

VISUALIZATION SKELETON

```
for d in [1, 2, 5]:
    poly = PolynomialFeatures(degree=d); X_poly = poly.fit_transform(X)
    model = LinearRegression().fit(X_poly, y)
    plt.plot(X_plot, model.predict(poly.transform(X_plot)), label=f"Degree {d}")
```

DAY 24

Feature Selection

Variance, Correlation, and Redundancy Removal

OBJECTIVES

- Understand why more features ≠ better models
- Implement variance thresholding
- Apply correlation-based selection
- Remove redundant features systematically
- Balance signal retention vs dimensionality

ACTIVITY

Remove low-variance and highly correlated features.

Compare model performance before/after.

ASSESSMENT

Justify selected feature set with statistical evidence and domain knowledge.

DAY 24 • SELECTION

Why More Features ≠ Better Models

Complexity, overfitting, and noise

The Problem

- More features → more complexity, overfitting risk
- Sparse data in high dimensions
- Noise features distract models into spurious patterns

The Solution

- Feature selection focuses on informative variables
- Improves interpretability
- Reduces overfitting risk
- Often speeds up training and inference

NOISE FEATURES HURT PERFORMANCE

```
X_informative = np.random.randn(100, 2)  # 2 true features
X_noise = np.random.randn(100, 20)       # 20 noise features
X_all = np.hstack([X_informative, X_noise])  # Model must sort through all
```

DAY 24 • SELECTION

Curse of Dimensionality

High dimensions make learning harder

The Phenomenon

- As features increase, data becomes sparse
- Distances between points become less meaningful
- Need exponentially more data to estimate reliably

Impact by Model

- k-NN: distances uninformative in high-D
- Trees: need more samples for reliable splits
- Linear: more parameters to estimate

KNN IN 2D VS 50D (RANDOM DATA)

```
knn_2d = KNeighborsRegressor(n_neighbors=5)
knn_50d = KNeighborsRegressor(n_neighbors=5)
score_2d = cross_val_score(knn_2d, X_2d, y, cv=5).mean()
score_50d = cross_val_score(knn_50d, X_50d, y, cv=5).mean()  # Usually much worse
```

DAY 24 • SELECTION

Variance Thresholding

Remove features with near-constant values

Concept

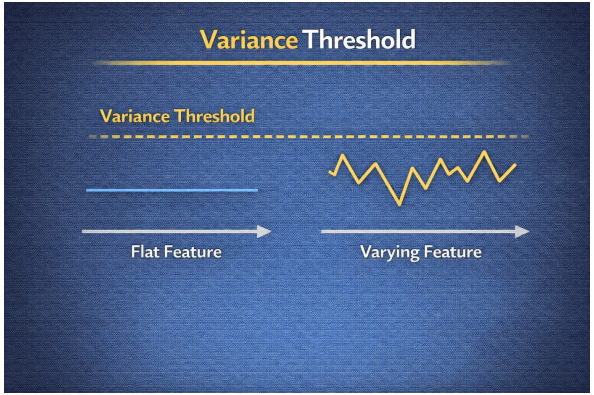
- Low-variance features carry little information
- Constant features contribute no predictive power
- Simple unsupervised technique applied first

Caveat

- Some low-variance features capture rare but critical events
- Set threshold thoughtfully (small positive value)
- Consider domain importance

VARIANCE THRESHOLD IMPLEMENTATION

```
from sklearn.feature_selection import VarianceThreshold
df = pd.DataFrame({"const": [1,1,1,1,1], "low_var": [1,1,1,2,1], "high_var": [1,2,3,4,5]})
selector = VarianceThreshold(threshold=0.0)
X_selected = selector.fit_transform(df)
print("Kept mask:", selector.get_support())  # Drops const feature
```



DAY 24 • SELECTION

Correlation-Based Feature Selection

Identify features strongly associated with target

Target Correlation

- Features highly correlated with target have predictive potential
- Low correlation may indicate irrelevance
- Supplement with model-based evaluation

Feature-Feature Correlation

- Highly correlated features are redundant
- Keeping one from each pair simplifies model
- Works best for linear relationships

CORRELATION ANALYSIS

```
df["x2"] = df["x1"] + np.random.normal(0, 0.01, size=100)  # Highly correlated
df["y"] = 2 * df["x1"] + noise
corr = df.corr()
print(corr)  # x1 and x2 redundant; either can represent signal
```

DAY 24 • SELECTION

Redundant Feature Removal in Practice

Systematically drop highly correlated features

Process

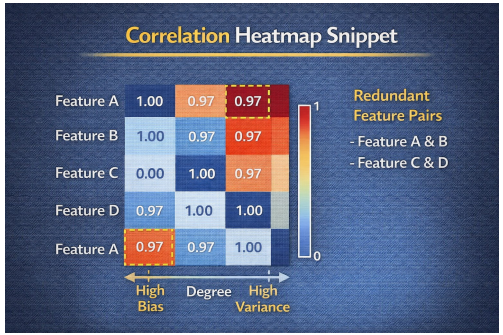
1. Compute correlation matrix
2. Identify pairs above threshold (e.g., 0.9)
3. Drop one from each pair (lower quality or target corr)
4. Reduces multicollinearity, simplifies model

Decision Criteria

- More missing values → drop
- Lower correlation with target → drop
- Less interpretable → drop

REDUNDANCY REMOVAL FUNCTION

```
def drop_correlated_features(df, threshold=0.9):
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [col for col in upper.columns if any(upper[col] > threshold)]
    return df.drop(columns=to_drop), to_drop
```



DAY 24 · SELECTION

Feature Selection vs Feature Engineering

Related but distinct processes

Feature Engineering

- Creates new features from raw data
- Captures domain-relevant structure
- Adds to feature universe

Feature Selection

- Chooses subset of features (raw or engineered)
- Acts as filter and regularizer
- Not a substitute for good engineering

ENGINEER THEN SELECT

```
df["x_sq"] = df["x"] ** 2  # Engineering: create feature
df["noise"] = np.random.randn(100)  # Noise feature

selector = SelectKBest(score_func=f_regression, k=2)
X_selected = selector.fit_transform(df[["x", "x_sq", "noise"]], y)  # Selection: keep best
```

DAY 24 • SELECTION

Model Stability and Generalization

Fewer, better features improve robustness

Benefits

- More stable coefficients across CV folds
- More consistent performance
- Focus on robust patterns, not noise idiosyncrasies

Risk of Over-Selection

- Aggressive selection may remove weak but complementary signals
- Tune selection threshold carefully
- Evaluate on held-out data

COMPARING FULL VS SELECTED FEATURES

```
model_full = Ridge(alpha=1.0); scores_full = cross_val_score(model_full, X, y, cv=5)
X_selected = X[:, :5]  # Assume first 5 are informative
model_selected = Ridge(alpha=1.0); scores_selected = cross_val_score(model_selected, X_selected, y,
cv=5)
print("Full CV R²:", scores_full.mean(), "Selected CV R²:", scores_selected.mean())
```

DAY 24 • ACTIVITY

Feature Selection Practice

Variance threshold and correlation filtering

Activity Steps

1. Apply VarianceThreshold to remove near-constant features
2. Compute correlation matrix
3. Implement drop_correlated_features() function
4. Apply correlation filter (threshold 0.9)
5. Compare Ridge performance before/after selection
6. Document which features removed and why

Assessment

1. Why are constant features rarely useful?
2. Procedure for correlation-based redundancy removal?
3. How does selection differ for linear vs tree models?
4. Explain curse of dimensionality and how selection helps

SKELETON

```
var_selector = VarianceThreshold(threshold=0.01).fit(X_train)
X_train_var = var_selector.transform(X_train)
X_reduced, dropped = drop_correlated_features(X_train_var_df, threshold=0.9)
```

DAY 25

Feature Engineering Mini-Project

End-to-End Workflow with Documentation

OBJECTIVES

- Apply full feature engineering workflow
- Integrate domain, interactions, polynomials, selection
- Guard against leakage explicitly
- Address ethical considerations
- Prepare features for production

ACTIVITY

Engineer robust feature set from cleaned dataset.

Document all decisions with rationale.

ASSESSMENT

Graded GitHub submission with code, data, and documentation.

DAY 25 · MINI-PROJECT

Designing an End-to-End Plan

Plan before coding

Plan Components

- Understand prediction task
- Identify important domain quantities
- Choose candidate transformations
- Plan feature selection strategy
- Specify leakage prevention measures

Evaluation Plan

- Cross-validation strategy
- Feature importance analysis
- Fairness and ethical review
- Document assumptions throughout

FEATURE ENGINEERING PLAN

```
plan = {"domain_features": ["per_unit_ratios"], "interactions": ["key_pairs"],
        "polynomial_terms": ["selected_degree_2"], "selection": ["variance", "correlation"],
        "leakage_checks": ["no_future_info"], "fairness": ["review_proxy_features"]}
```

DAY 25 · MINI-PROJECT

Applying Domain-Driven Features

Meaningful ratios and per-unit metrics

Implementation

- Compute price_per_sqft, revenue_per_user, etc.
- Handle division by zero safely
- Document rationale for each feature

Documentation

- What real-world quantity it represents
- How it relates to target
- Expected model impact

DOMAIN FEATURES IN PROJECT

```
df_features = df.copy()
df_features["price_per_sqft"] = df["price"] / df["sqft"].replace({0: np.nan})
df_features["price_per_sqft"] =
df_features["price_per_sqft"].fillna(df_features["price_per_sqft"].median())
df_features["revenue_per_user"] = df["revenue"] / df["num_users"].replace({0: np.nan})
```

DAY 25 · MINI-PROJECT

Engineering Interactions and Polynomials

Thoughtful, limited expansion

Interaction Strategy

- Limited set based on domain insight
- Key numeric pairs (multiplicative)
- Important conditions (logical)

Polynomial Strategy

- Selective application to few variables
- Degree 2 for variables with plausible non-linearity
- Requires regularization

INTERACTIONS AND POLYNOMIALS IN PROJECT

```
df_features["f1_x_f2"] = df_features["feature1"] * df_features["feature2"]
df_features["high_risk"] = ((df_features["age"] > 60) & (df_features["is_smoker"] == 1)).astype(int)
# Polynomial on selected variables
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(df_features[["age", "price_per_sqft"]])
```

DAY 25 · MINI-PROJECT

Guarding Against Target Leakage

Critical for valid evaluation

Leakage Sources

- Features using future information
- Aggregates including post-outcome records
- Ratios with target-derived quantities
- Scaling parameters from combined train+test

Prevention

- Review every feature definition
- Fit transformers on training only
- Use proper train/test split before any engineering
- Document time-based validity

LEAKAGE-SAFE SCALING

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Fit on train only
X_test_scaled = scaler.transform(X_test)         # Transform test
```

DAY 25 · MINI-PROJECT

Feature Selection in the Project

Pare down expanded feature space

After Engineering

- Feature space may be large and redundant
- Polynomial and interaction terms proliferate
- Selection creates compact, informative set

Process

1. Variance thresholding
2. Correlation-based redundancy removal
3. Compare model performance
4. Document selections and rationale

SELECTION PIPELINE

```
var_selector = VarianceThreshold(threshold=0.01)
X_train_var = var_selector.fit_transform(X_train_num)

X_train_reduced, dropped = drop_correlated(X_train_var_df, threshold=0.95)
print("Original:", len(numeric_cols), "After variance:", X_train_var.shape[1], "After corr:",
X_train_reduced.shape[1])
```

DAY 25 • MINI-PROJECT

Ethical Considerations in Feature Design

Fairness, privacy, and potential harms

Key Concerns

- Features acting as proxies for protected attributes
- Postal codes, income may correlate with race/gender
- Can embed and amplify biases

Mitigations

- Review each feature for fairness issues
- Consider necessity and proportionality
- Conduct fairness audits on model predictions
- Document justifications

IDENTIFYING POTENTIAL PROXY FEATURES

```
sensitive_proxies = [col for col in df_features.columns if "zip" in col.lower() or "income" in
col.lower()]
print("Potential proxy features for review:", sensitive_proxies)
```

DAY 25 • MINI-PROJECT

Preparing Features for Production

Reproducible, versioned, monitored

Requirements

- Encapsulate transformations in pipelines
- Handle missing values consistently
- Maintain column ordering
- Version external dependencies

Monitoring

- Detect data drift over time
- Log feature distributions
- Alert on unexpected values

PRODUCTION PIPELINE

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

preprocess = ColumnTransformer(transformers=[
    ("num", Pipeline([("imputer", SimpleImputer(strategy="median")), ("scaler", StandardScaler())]),
numeric_features)])
final_model = Pipeline([("preprocess", preprocess), ("model", RandomForestRegressor())])
```

DAY 25 · MINI-PROJECT

Model Training with Engineered Features

Compare with and without advanced features

Training Strategy

- Train multiple model families (linear, tree)
- Compare performance with/without engineering
- Demonstrate value of your features

Interpretation

- Analyze feature importances
- Inspect coefficients
- Relate back to domain intuition

MODEL COMPARISON

```
ridge = Ridge(alpha=1.0).fit(X_train_final, y_train)
rf = RandomForestRegressor(random_state=0).fit(X_train_final, y_train)
print("Ridge R²:", r2_score(y_test, ridge.predict(X_test_final)))
print("RF R²:", r2_score(y_test, rf.predict(X_test_final)))
```

DAY 25 · ACTIVITY

Full Feature Engineering Submission

End-to-end workflow with documentation

Deliverables

1. Cleaned input data + engineered feature dataset
2. Reproducible feature engineering pipeline
   - Domain features, interactions, polynomials
   - Feature selection
3. Documentation explaining:
   - Domain intuition for each feature type
   - Leakage prevention measures
   - Ethical considerations
4. Evaluation comparing models with/without features

Assessment Checklist

- ☐ Repository with cleaned + engineered data
- ☐ Code implements reproducible pipeline
- ☐ Documentation explains domain intuition
- ☐ Evaluation compares with/without features
- ☐ Reflection discusses leakage, selection, ethics

Submit GitHub repository URL.

Week 5 Complete!

Feature Engineering II — From Expert Knowledge to Production

Domain-Driven	Ratio features, per-unit metrics, business logic; validate empirically; avoid leakage
Interactions	Multiplicative, additive, logical; expand hypothesis space; manage dimensionality
Polynomials	Non-linear for linear models; degree selection; regularization essential
Selection	Variance threshold, correlation filter; reduce overfitting; improve stability
Mini-Project	End-to-end pipeline; documentation; ethical review; production-ready

Key Takeaway

Feature engineering transforms raw data into meaningful representations. Document decisions, validate empirically, guard against leakage, and consider ethical implications at every step.

PATTERN TO CARRY FORWARD

```
engineer_features(df) → domain_ratios() → interactions() → polynomials() → select_features() →
validate() → document()
```