Terminologies and Concepts

Select Features for Modeling

Creates the machine learning setup:

- X (features): The input variables that will predict price
- 4 numerical features (cost, marketing, competition, demand)
- 2 encoded categorical features (category, brand tier)
- y (target): What we want to predict (actual_price)

```python
feature_columns = ['manufacturing_cost', 'marketing_spend', 'competition_level',
                   'demand_score', 'category_encoded', 'brand_encoded']
X = products[feature_columns]
y = products['actual_price']
```

Terminologies and Concepts

Preparing the model for training

- **This prepares the data for building a price prediction model. The model will learn patterns like:**
- "Electronics products with high manufacturing costs and premium brands typically have higher prices"
- "High competition and low demand usually lead to lower prices"

- **The encoded features allow the algorithm to understand that different categories and brand tiers have different pricing patterns, while keeping all data in numerical format for processing.**

Terminologies and Concepts

Step 1: Train-Test Split

Purpose: Divides your 500 products into two separate datasets:

Training Set (80% = 400 products):

- Used to train/teach the machine learning model
- Model learns pricing patterns from this data

Test Set (20% = 100 products):

- Used to evaluate how well the model performs on "unseen" data
- Simulates real-world performance on new products

Why Split?: Prevents **overfitting** - ensures the model can generalize to new data rather than just memorizing the training examples.

random_state=42: Ensures reproducible results (same split every time you run the code).

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

Terminologies and Concepts

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 2: Feature Scaling

Problem: Features have very different scales:

- `manufacturing_cost`: 10-200 range
- `marketing_spend`: 1-50 range
- `competition_level`: 0.1-1.0 range
- `demand_score`: 0.2-1.0 range

Solution: StandardScaler converts all features to have:

- **Mean = 0**
- **Standard deviation = 1**

Important Process:

1. **fit_transform()** on training data: Calculates scaling parameters AND applies them
2. **transform()** on test data: Uses the SAME scaling parameters from training

Why This Order Matters: Prevents **data leakage** - the model shouldn't "see" test data characteristics during training.

Terminologies and Concepts

Model 1: Original Features

Uses raw feature values:

- **manufacturing_cost: 10-200 range**
- **marketing_spend: 1-50 range**
- **competition_level: 0.1-1.0 range**
- **demand_score: 0.2-1.0 range**

Plus encoded categorical variables

What it learns: The relationship between original feature values and prices.

```python
model = LinearRegression()
model.fit(X_train, y_train)
```

Terminologies and Concepts

Model 2: Scaled Features

Uses standardized feature values (mean=0, std=1):

All features are now on the same scale

Makes coefficients more directly comparable

```python
model_scaled = LinearRegression()
model_scaled.fit(X_train_scaled, y_train)
```