🛠️ Week 9 Student Guide – Feature Engineering in Practice

This guide is designed to help you execute your **Week 9 Assignment**, which focuses entirely on **Feature Engineering**. Feature engineering is the process of creating **new variables** from existing data to improve the predictive power of a model and deepen analytical understanding.

💻 Part 1: Feature Engineering Fundamentals

In basketball analytics, box score statistics are often less valuable than the advanced metrics derived from them. Feature engineering uses existing metrics (like FGA, PTS, MIN) to create these advanced metrics (like TS% or USG%).

1. Creating Derived Metrics (Formulas)

This is the most common type of feature engineering. You simply use existing columns in arithmetic operations to generate a new, more meaningful metric.

Feature	Purpose	Formula (Pandas/Python)	Example Source Columns
Player Efficiency (EFF)	Basic metric for overall impact.		`PTS, REB, AST, STL, BLK, FGA, FGM, FTA, FTM, TO, GP`
True Shooting % (TS%)	Measures scoring efficiency, valuing 3-pointers and free throws.		`PTS, FGA, FTA`
Usage Rate (USG%)	Estimates the percentage of team plays used by a player.		`FGA, FTA, TO, MIN, Team_MIN`

2. Creating Contextual Features (Rolling Averages)

Rolling averages provide context by averaging a metric over a specified number of *recent* games. This creates a feature that reflects a player's **current form or trend** rather than their all-time average.

- **Key Code:** `df.groupby('Player')['Metric'].transform(lambda x: x.rolling(N, min_periods=1).mean())`
    - `groupby('Player')`: Ensures the rolling average calculation restarts for each unique player.
    - `.rolling(N)`: Specifies the window size (e.g., 5 games).
    - `min_periods=1`: Allows the calculation to start immediately, even if a player has played fewer than games.

3. Scaling and Standardization

In preparation for machine learning (ML), scaling transforms numerical data so that all features contribute equally to the model. **StandardScaler** transforms data to have a mean of 0 and a standard deviation of 1.

- **When to Use:** On the numerical features you plan to use in an ML model.
- **Key Code:** `scaler.fit_transform(df[scaled_cols])`

🏀 Part 2: Week 9 Assignment Tasks

The assignment uses a provided, cleaned, per-game box score dataset loaded as `df`.

1. Engineer Features (Step 2)

Create the new features required by the assignment using the provided box score statistics (`PTS, REB, AST, STL, BLK, TO, FGM, FGA, FTM, FTA, MIN, GP, Team_MIN`).

Python

```
# 1. Player Efficiency
df['Efficiency'] = ((df['PTS'] + df['REB'] + df['AST'] + df['STL'] +
df['BLK']) - (df['FGA'] - df['FGM']) - (df['FTA'] - df['FTM']) - df['TO']) /
df['GP']

# 2. True Shooting %
# Note: Added a small constant (epsilon) for robust division handling,
although not shown in the assignment.
df['TS%'] = df['PTS'] / (2 * (df['FGA'] + 0.44 * df['FTA']))

# 3. Usage Rate (USG%) - Ensure Team_MIN is total team minutes played (240 in
a regulation game)
df['USG%'] = 100 * ((df['FGA'] + 0.44 * df['FTA'] + df['TO']) * df['MIN']) /
df['Team_MIN']

# 4. Workload Ratio (Minutes Played / Team Minutes)
df['Workload_Ratio'] = df['MIN'] / df['Team_MIN']
```

```
# 5. Rolling 5-game Efficiency
df['Rolling_Efficiency'] =
df.groupby('Player')['Efficiency'].transform(lambda x: x.rolling(5,
min_periods=1).mean())
```

2. Visualize New Features (Step 3)

Create two plots showcasing your new features.

- **Visualization 1: Efficiency by Position (Box Plot)**
  - Compares the distribution of the calculated `Efficiency` score across different `Position` groups.

```
Python
plt.figure(figsize=(8,6))
sns.boxplot(data=df, x='Position', y='Efficiency')
plt.title('Efficiency by Position')
plt.show()
```

- **Visualization 2: True Shooting vs Usage Rate (Scatter Plot)**
  - Shows the relationship between `TS%` (efficiency) and `USG%` (volume/role). This helps identify high-volume, low-efficiency players vs. low-volume, high-efficiency players.

```
Python
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x='USG%', y='TS%')
plt.title('True Shooting vs Usage Rate')
plt.show()
```

3. Correlation Analysis (Step 4)

Generate a heatmap of the full dataset (including original and new features) to identify redundancy. Features with very **high correlation** () are often redundant, meaning they measure the same thing.

```
Python
plt.figure(figsize=(10,6))
# Select only numerical columns for correlation calculation
sns.heatmap(df.select_dtypes(include=np.number).corr(),
            cmap='coolwarm',
            annot=False) # Keep annot=False for a cleaner look
plt.title('Feature Correlation Matrix')
plt.show()
```

4. Interpretation (Step 5)

Analyze the correlation heatmap.

- Look for redundancy: Are the raw stats (`PTS`, `FGM`, `FGA`) highly correlated with your new metric, `Efficiency`? They should be, but `TS%` should offer a distinct (less correlated) view of efficiency.
- Look for predictive value: Why are `TS%` and `USG%` valuable together? They decouple *volume* from *efficiency*, which is essential for accurate player evaluation.
- Look at temporal features: How does `Rolling_Efficiency` add value? It captures form, making it more predictive of a player's *next* performance than their season-long average.
