**Big Mart Sales – Random Forest Approach: One-Page Summary**

**1. Goal & Initial Analysis**

* **Objective**: Predict Item\_Outlet\_Sales (target) for items across multiple outlets.
* **Data**:
  + Train (~8.5k rows) with Item\_Outlet\_Sales.
  + Test (~5.7k rows) missing Item\_Outlet\_Sales.

**Observations**:

1. **Missing Values** in Item\_Weight and Outlet\_Size.
2. **Zero Item\_Visibility** likely means missing rather than actual zero.
3. **Inconsistent Labels** in Item\_Fat\_Content (LF, low fat, reg).
4. **Skewed** target with outliers, leading us to consider a **log transform**.

**2. Problems Faced & Code Updates**

1. **Encoding Issue**:
   * **Problem**: After label-encoding Item\_Identifier and Outlet\_Identifier, we ended up with numeric IDs in the final submission, causing “unknown ID” errors on the leaderboard.
   * **Update**: We introduced a separate test\_original to keep the original IDs intact for the final CSV. Meanwhile, we used an encoded test copy for the model predictions.
2. **Aggregator Merges for Test**:
   * **Problem**: When merging Item\_Mean\_Sales or Outlet\_Mean\_Sales from training, some item/outlet combos in test didn’t exist in train, leading to NaN.
   * **Update**: Filled aggregator NaNs with the **global** mean of Item\_Outlet\_Sales, ensuring no missing merges remain.
3. **Zero Visibility**:
   * **Problem**: “0” in Item\_Visibility was nonsensical, hurting model performance.
   * **Update**: Replaced zeros with median visibility grouped by Item\_Identifier.
4. **Skewed Sales**:
   * **Problem**: A few extremely large Item\_Outlet\_Sales values overshadowed the distribution.
   * **Update**: Used Log\_Sales = log1p(Item\_Outlet\_Sales) to reduce outlier impact and generally lower RMSE.

**3. Feature Engineering Steps**

1. **Outlet Age**
   * train['Outlet\_Age'] = 2023 - train['Outlet\_Establishment\_Year']
   * Rationale: Older outlets often differ in footfall or revenue pattern.
2. **Aggregator Features**
   * **Item\_Mean\_Sales**: mean sales of each item across outlets.
   * **Outlet\_Mean\_Sales**: mean sales of each outlet across items.
   * Hypothesis: some items are inherently popular, some outlets are known to have higher average sales.
   * Merged into train and test; missing combos in test replaced by global mean.
3. **Standard Scaling**
   * For numeric columns (Item\_Weight, Item\_Visibility, Item\_MRP, aggregator columns) to help the random forest or advanced boosting methods treat numeric features uniformly.

**4. Random Forest Model**

1. **Why Random Forest**:
   * An **ensemble** of decision trees, can capture non-linear relationships. Less prone to overfitting than a single tree.
2. **Implementation**:
   * Label-encoded categorical columns (except we retain test\_original for final IDs).
   * Trained RandomForestRegressor(n\_estimators=200, max\_depth=10, random\_state=42).
   * Used a hold-out validation approach (80/20) to measure local RMSE.
   * Typically ended around **1000–1100 RMSE** initially; aggregator features + log transform improved it further (~900–1000 range).
3. **Final Submission**:
   * **Predictions** made on the encoded test.
   * **CSV** file used IDs from test\_original to avoid the “unknown ID” error.

**5. Key Takeaways**

1. **Separate “test\_original”** vs. “test” for label encoding to preserve original IDs in submission.
2. **Advanced features** (Outlet\_Age, aggregator means, log transform) significantly boost performance.
3. Random Forest provided a **solid baseline** among ensemble methods, easily improved by further hyperparameter tuning or switching to advanced boosters (CatBoost/XGBoost).

**Summary**:  
We faced encoding & aggregator-merge issues, overcame them by retaining test\_original for submission and using aggregator placeholders for missing merges. After final improvements, our **Random Forest** approach reached a respectable RMSE on local validation and gave us a strong foundation for Big Mart Sales predictions.

I gave a try to other models such as CatBoost and XGBoost to try out if they can achieve best rmse score and run code similarly for them, But Random Forest stood out comparing with them.

**CatBoost**:

* Native **categorical handling** => no heavy manual encoding.
* Usually the best performer once aggregator features + log transform were applied, especially for data with many categorical variables.

**XGBoost**:

* Captured more complex relationships than Random Forest with **boosting** approach.
* Required parameter tuning (learning rate, depth, etc.) but generally outperformed RF with advanced features.