Sure! Below is a comprehensive documentation that outlines the SQL query, the relationships among the entities involved, and the machine learning process, including model selection and evaluation.

# Documentation for Product Sales Prediction Model

## 1. Overview

This documentation details the process of building a machine learning model to predict whether a product will sell out in the next week based on historical sales data. The model leverages data from multiple related tables in a SQL database.

## 2. SQL Query

The SQL query retrieves relevant data from various tables to create a comprehensive dataset for analysis. Here's the SQL query used:

```sql

WITH last\_order\_date AS (

SELECT

MAX(created\_at) AS max\_date

FROM

public.orders

)

SELECT

p.id AS product\_id,

pn.name AS product\_name,

c.name AS category\_name,

pvs.stock AS current\_stock,

pvp.price AS current\_price,

COALESCE(SUM(o.total\_amount), 0) AS total\_sales\_last\_3\_months,

COALESCE(AVG(pr.rating), 0) AS average\_rating,

COUNT(DISTINCT o.id) AS total\_orders\_last\_3\_months,

(SELECT COUNT(\*)

FROM public.orders o2

WHERE o2.created\_at >= (SELECT max\_date FROM last\_order\_date) - INTERVAL '6 months'

AND o2.id IN (SELECT pr2.order\_id FROM public.product\_ratings pr2 WHERE pr2.product\_id = p.id)) AS total\_orders\_last\_6\_months,

(SELECT COUNT(\*)

FROM public.orders o3

WHERE o3.created\_at >= (SELECT max\_date FROM last\_order\_date) - INTERVAL '3 months'

AND o3.id IN (SELECT pr3.order\_id FROM public.product\_ratings pr3 WHERE pr3.product\_id = p.id)) AS total\_orders\_last\_3\_months,

pv.weight AS product\_weight,

pv.status AS variation\_status

FROM

public.products p

JOIN

public.product\_names pn ON p.name\_id = pn.id

JOIN

public.categories c ON pn.category\_id = c.id

JOIN

public.product\_variations pv ON p.id = pv.product\_id

JOIN

public.product\_variation\_stocks pvs ON pv.id = pvs.product\_variation\_id

LEFT JOIN

public.product\_variation\_prices pvp ON pv.id = pvp.product\_variation\_id

LEFT JOIN

public.product\_ratings pr ON pr.product\_id = p.id

LEFT JOIN

public.orders o ON o.id = pr.order\_id AND o.created\_at >= (SELECT max\_date FROM last\_order\_date) - INTERVAL '3 months'

GROUP BY

p.id, pn.name, c.name, pvs.stock, pvp.price, pv.weight, pv.status;

```

### 2.1 Explanation of the SQL Query

- \*\*Common Table Expression (CTE)\*\*: The `last\_order\_date` CTE fetches the most recent order date from the `orders` table.

- \*\*Main Query\*\*:

- Joins multiple tables (`products`, `product\_names`, `categories`, `product\_variations`, etc.) to gather comprehensive product information.

- Computes aggregates such as total sales in the last three months and average product ratings.

- Retrieves relevant attributes for modeling, including current stock, price, and product weight.

### 2.2 Relationships

- \*\*Products\*\* (`products`) are linked to their names (`product\_names`) and categories (`categories`).

- Each product can have multiple variations (`product\_variations`), which in turn have stock levels (`product\_variation\_stocks`) and prices (`product\_variation\_prices`).

- Products can also be rated (`product\_ratings`), and orders (`orders`) contain multiple products.

## 3. Machine Learning Model

### 3.1 Data Preparation

After retrieving the data, a binary target variable is created to indicate whether a product will sell out in the next week based on a defined sales threshold.

```python

data['sold\_out\_next\_week'] = np.where(data['total\_sales\_last\_3\_months'] > threshold, 1, 0)

```

### 3.2 Model Selection

We chose the \*\*Random Forest Classifier\*\* for the following reasons:

- \*\*Robustness\*\*: Random Forest is less sensitive to overfitting compared to other models, especially with high-dimensional data.

- \*\*Feature Importance\*\*: It provides insights into which features are most important for making predictions.

- \*\*Handling Imbalance\*\*: It can effectively manage imbalanced datasets, especially when class weights are adjusted.

### 3.3 Model Training

The dataset is split into training and testing subsets, and the model is trained on the training data:

```python

model = RandomForestClassifier(n\_estimators=100, random\_state=42, class\_weight='balanced')

model.fit(X\_train, y\_train)

```

### 3.4 Model Evaluation

The model's performance is evaluated using:

- \*\*Confusion Matrix\*\*: To visualize true positives, true negatives, false positives, and false negatives.

- \*\*Classification Report\*\*: To provide precision, recall, and F1-score metrics.

- \*\*ROC Curve\*\*: To assess the trade-off between sensitivity and specificity.

### 3.5 Visualization

Three crucial visualizations are generated:

1. \*\*Feature Importance Bar Plot\*\*: Shows which features contribute most to the model's predictions.

2. \*\*Confusion Matrix Heatmap\*\*: Provides a visual interpretation of the confusion matrix.

3. \*\*ROC Curve\*\*: Displays the model's true positive rate against the false positive rate, with the area under the curve (AUC) indicating overall performance.

## 4. Conclusion

This documentation outlines the process of building a product sales prediction model using SQL to extract data and a Random Forest Classifier for prediction. The model is designed to help stakeholders make informed decisions regarding inventory management and sales strategies.

Future enhancements could include hyperparameter tuning, exploring additional features, or using alternative modeling techniques to improve prediction accuracy.

If you have any further questions or require additional details, feel free to ask!