Customer Order Prediction Low-Level Design (LLD)

1. Introduction

The **Customer Order Prediction** system is designed to predict customer purchases based onhistorical transaction data. This Low-Level Design document describes the detailed implementation of the system, including data structures, algorithms, functions, and methods.

2. Functional Modules

2.1 Data Ingestion Module

This module manages the ingestion of transaction data from CSV files or databases.

2.1.1 Functions & Methods

- **Function**: `load_data(filepath: str) -> pd.DataFrame`
  - **Description**: Loads data from a CSV file or database into a Pandas DataFrame.
  - **Input**: `filepath` - Path to the data file.
  - **Output**: Pandas DataFrame containing transaction data.

```
python
Copy code
def load_data(filepath: str) -> pd.DataFrame:
    """Load data from CSV file."""
    df = pd.read_csv(filepath)
    return df
```

2.1.2 Validations

- Ensure that the file exists and is accessible.
- Check for missing or malformed data (e.g., missing `BillNo`, `Itemname`).

2.2 Data Cleaning Module

This module ensures that the data is cleaned and structured for further processing.

2.2.1 Functions & Methods

- **Function**: `clean_data(df: pd.DataFrame) -> pd.DataFrame`
  - **Description**: Cleans and validates the input DataFrame by removing null values and duplicates.
  - **Input**: Pandas DataFrame containing raw transaction data.
  - **Output**: Cleaned Pandas DataFrame.

```
python
Copy code
def clean_data(df: pd.DataFrame) -> pd.DataFrame:
    """Clean the transaction data."""
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
```

```
return df
```

2.3 Feature Engineering Module

This module derives additional features from the raw data to improve the predictive model's performance.

2.3.1 Functions & Methods

- **Function**: `add_features(df: pd.DataFrame) -> pd.DataFrame`
  - **Description**: Adds new features such as `AveragePrice`, `TotalQuantity` to the data.
  - **Input**: Cleaned Pandas DataFrame.
  - **Output**: DataFrame with additional features.

```
python
Copy code
def add_features(df: pd.DataFrame) -> pd.DataFrame:
    """Generate new features like AveragePrice and TotalQuantity."""
    df['TotalQuantity'] =
df.groupby('BillNo')['Quantity'].transform('sum')
    df['AveragePrice'] = df.groupby('BillNo')['Price'].transform('mean')
    return df
```

2.4 Model Training Module

This module handles training the machine learning model to predict future customer orders.

2.4.1 Functions & Methods

- **Function**: `train_model(X: pd.DataFrame, y: pd.Series) -> RandomForestRegressor`
  - **Description**: Trains the Random Forest Regressor on the input features `X` and target `y`.
  - **Input**:
    - `X` - Feature DataFrame.
    - `y` - Target variable (Total spent).
  - **Output**: Trained Random Forest model.

```
python
Copy code
from sklearn.ensemble import RandomForestRegressor

def train_model(X: pd.DataFrame, y: pd.Series) -> RandomForestRegressor:
    """Train the Random Forest model."""
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X, y)
    return model
```

2.5 Model Evaluation Module

This module evaluates the performance of the trained model using various metrics.

2.5.1 Functions & Methods

- **Function**: `evaluate_model(model: RandomForestRegressor, X_test: pd.DataFrame, y_test: pd.Series)`
  - **Description**: Evaluates the model using Mean Squared Error (MSE).
  - **Input**: Trained model, test data `X_test`, and actual target values `y_test`.
  - **Output**: MSE value.

```
Python Copy:
from sklearn.metrics import mean_squared_error

def evaluate_model(model: RandomForestRegressor, X_test: pd.DataFrame,
y_test: pd.Series) -> float:
    """Evaluate the model using Mean Squared Error (MSE)."""
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    return mse
```

2.6 API Module

This module creates an API using Flask/FastAPI to expose the trained model for prediction.

2.6.1 Functions & Methods

- **Function**: `predict_order(data: dict) -> float`
  - **Description**: Receives input data from the API request and returns the predicted order amount.
  - **Input**: JSON data containing features.
  - **Output**: Predicted value.

```
python code:
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict_order():
    data = request.get_json()
    # Assume `model` is the preloaded Random Forest model

    prediction =  model.predict([data['features']])
    return jsonify({'prediction': prediction[0]})
```

---

3. Class Diagram

Class Name: DataIngestion

- **Attributes**:
  - `file_path: str`
  - `data: pd.DataFrame`
- **Methods**:

- `load_data()`

Class Name: ModelTraining

- **Attributes**:
  - `X_train: pd.DataFrame`
  - `y_train: pd.Series`
- **Methods**:
  - `train_model()`

---

4. Sequence Diagram

Include a sequence diagram that outlines the flow of data and operations from data ingestion to model deployment and prediction.

---

5. Database Schema

- **Table: Transactions**
  - `BillNo`: VARCHAR
  - `Itemname`: VARCHAR
  - `Quantity`: INT
  - `Price`: FLOAT
  - `CustomerID`: VARCHAR

---

6. Error Handling

- Handle file input/output errors in the `load_data()` function.
- Implement exception handling for model prediction errors in the API.

---

7. Performance Considerations

- Optimize data loading by reading data in chunks for large datasets.
- Ensure the Random Forest model is trained using an optimal number of estimators to avoid overfitting.