**Sales Forecasting API Documentation**

**Introduction**

This document explains the Python Flask API that serves as an interface for the trained XGBoost sales forecasting model. The API allows users to input a future month and year, and it predicts sales for different product categories.

---

**1. Installation Requirements**

**Installing Python**

1. Download Python from the official website: Python Downloads
2. Install Python and ensure that the **Add Python to PATH** option is checked.
3. Verify installation by running:
4. python --version

**Installing Required Python Packages**

After installing Python, install the required dependencies using pip:

pip install flask pandas numpy xgboost scikit-learn

---

**2. Importing Required Libraries**

import pickle

import pandas as pd

import numpy as np

import xgboost as xgb

from flask import Flask, request, jsonify

**Explanation:**

- pickle → Loads the previously saved label encoders and sales data.
- pandas → Handles data processing.
- numpy → Used for numerical operations.
- xgboost → Loads the pre-trained XGBoost model for prediction.
- flask → Web framework to handle API requests.

---

**3. Loading the Trained Model**

model = xgb.XGBRegressor()

model.load_model("sales_forecast_model.json")

**Explanation:**

- Initializes an empty XGBRegressor model.
- Loads the pre-trained model from the sales_forecast_model.json file.

---

**4. Loading Encoders and Sales Data**

```
with open("label_encoders.pkl", "rb") as f:
  le_category, le_sub_category, sales_data = pickle.load(f)
```

**Explanation:**

- Loads the saved **LabelEncoders** and sales_data from the label_encoders.pkl file.
- le_category → Converts numerical category values back to their original text.
- le_sub_category → Converts numerical sub-category values back to their original text.
- sales_data → Previously processed sales dataset.

---

**5. Defining Feature Columns Used for Prediction**

```
FEATURES = ['Category_Encoded', 'Sub_Category_Encoded', 'Lag_1', 'Lag_2', 'Rolling_Mean_3', 'Month', 'Year']
```

**Explanation:**

- This list contains all the feature columns that were used to train the model. The new data must also have these columns to ensure compatibility with the model.

---

**6. Initializing Flask App**

```
app = Flask(__name__)
```

**Explanation:**

- Creates a Flask web application instance.

---

**7. Creating the Prediction Endpoint**

```
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    year = int(data['year'])
    month = int(data['month'])
```

**Explanation:**

- Defines an API route /predict that accepts **POST requests**.
- Extracts the **year** and **month** values from the JSON request body.

---

**8. Preparing the Data for Prediction**

```
last_month_sales = sales_data[sales_data['Year-Month'] == sales_data['Year-Month'].max()].copy()
last_month_sales['Lag_2'] = last_month_sales['Lag_1'].fillna(method='bfill')
```

**Explanation:**

- Selects sales data from the most recent month available in the dataset.
- Fills missing Lag_2 values using backward fill (bfill).

---

**9. Creating a Future Dataframe for Prediction**

future_data = last_month_sales.copy()

future_data['Year-Month'] = pd.Timestamp(year, month, 1)

future_data['Month'] = month

future_data['Year'] = year

**Explanation:**

- Creates a copy of the last available sales data.
- Updates the Year-Month, Month, and Year columns with the future values provided by the user.

---

**10. Running the Prediction**

predictions = model.predict(future_data[FEATURES])

future_data['Predicted_Sales'] = np.round(predictions).astype(int)

**Explanation:**

- Uses the XGBoost model to predict future sales based on the given month and year.
- Rounds the predictions and converts them to integers for readability.

---

**11. Mapping Encoded Values Back to Categories**

future_data['Predicted_Category'] = le_category.inverse_transform(future_data['Category_Encoded'])

future_data['Predicted_Sub_Category'] = le_sub_category.inverse_transform(future_data['Sub_Category_Encoded'])

**Explanation:**

- Converts encoded category numbers back into their original text labels.
- Converts encoded sub-category numbers back into their original text labels.

---

**12. Formatting the API Response**

result = future_data[['Month', 'Predicted_Category', 'Predicted_Sub_Category', 'Predicted_Sales']].to_dict(orient='records')

return jsonify(result)

**Explanation:**

- Extracts only the necessary columns for the response.
- Converts the DataFrame into a JSON format using .to_dict(orient='records').
- Sends the JSON response back to the user.

---

**13. Running the Flask Application**

```
if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000, debug=True)
```

**Explanation:**

- Runs the Flask application on **port 5000**.
- host='0.0.0.0' → Makes the API accessible from any device on the network.
- debug=True → Enables debugging mode to display errors in real time.

---

**How to Use the API**

1. **Start the Flask Server:**
2. python server.py
3. **Send a POST request to /predict with JSON input:**
4. {
5. "year": 2025,
6. "month": 6
7. }
8. **Expected JSON Response:**
9. [
10. {
11. "Month": 6,
12. "Predicted_Category": "Furniture",
13. "Predicted_Sub_Category": "Chairs",
14. "Predicted_Sales": 150

```
    },
    {
        "Month": 6,
        "Predicted_Category": "Office Supplies",
        "Predicted_Sub_Category": "Binders",
        "Predicted_Sales": 210
    }
]
```


---

**Conclusion**

This API allows users to predict future sales by sending a request with a **year** and **month**. The Flask server processes the request, prepares the necessary data, makes a prediction using the trained XGBoost model, and returns the results in JSON format.

This setup enables seamless integration with other applications like **dashboards, mobile apps, or other backend services**. 🚀