complete solution for the transport cost evaluation system. I'll break this down into multiple parts for clarity: database setup, data processing, calculation logic, and Streamlit application.

Step 1: Database Setup (MySQL)

First, let's create the database and tables based on the structure of the provided data.

SQL Script for Database Creation

<>sql

```sql
-- Create Database
CREATE DATABASE IF NOT EXISTS Transport_database;
USE Transport_database;

-- Create Table for Transport Cost Data (from Book 3.xlsx)
CREATE TABLE TransportCosts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    lane_code VARCHAR(10),
    supplier_country VARCHAR(2),
    supplier_zip VARCHAR(2),
    dest_country VARCHAR(2),
    dest_zip VARCHAR(2),
    lead_time_days INT,
    price_50kg DECIMAL(10,2),
    price_75kg DECIMAL(10,2),
    price_100kg DECIMAL(10,2),
    price_150kg DECIMAL(10,2),
    price_200kg DECIMAL(10,2),
    -- Add more price columns for other weight brackets up to 22500kg
    price_1500kg DECIMAL(10,2),
    price_2000kg DECIMAL(10,2),
    -- ... continue for all brackets
    price_22500kg DECIMAL(10,2)
);

-- Create Table for Packaging Data (from Mappe2 1 1.xlsx)
CREATE TABLE PackagingData (
    id INT AUTO_INCREMENT PRIMARY KEY,
    packaging_type VARCHAR(50),
    pieces_per_unit INT,
    packaging_weight_kg DECIMAL(10,2),
    units_per_pallet INT,
    pallet_weight_kg DECIMAL(10,2),
    stackability_factor INT
);
```

Python Script to Import Data

```python
import pandas as pd
import mysql.connector
from mysql.connector import Error

# Database connection
def connect_db():
    try:
        connection = mysql.connector.connect(
            host='localhost',
            database='Transport_database',
            user='your_username',
            password='your_password'
        )
        return connection
    except Error as e:
        print(f"Error connecting to MySQL: {e}")
        return None

# Import data from CSV to MySQL
def import_data_to_mysql(csv_file, table_name):
    df = pd.read_csv(csv_file)
    connection = connect_db()
    cursor = connection.cursor()

    # Prepare insert query based on table structure
    columns = ', '.join(df.columns)
    placeholders = ', '.join(['%s'] * len(df.columns))
    sql = f"INSERT INTO {table_name} ({columns}) VALUES ({placeholders})"

    for _, row in df.iterrows():
        cursor.execute(sql, tuple(row))

    connection.commit()
    cursor.close()
    connection.close()

# Example usage
# import_data_to_mysql('Book_3_converted.csv', 'TransportCosts')
# import_data_to_mysql('packaging_data.csv', 'PackagingData')
```

Step 2: Calculation Logic Implementation

Implement the transport cost calculation logic based on the workflow guide.

```python
def calculate_transport_cost(inputs):
    # Step 1: Material and Packaging Calculations
    total_material_weight = inputs['material_weight'] *
inputs['pieces_per_packaging']
    total_packaging_weight = total_material_weight + inputs['packaging_weight']
    monthly_demand = inputs['daily_demand'] * 30 / inputs['deliveries_per_month']
    packaging_units_needed = monthly_demand / inputs['pieces_per_packaging']

    # Step 2: Logistics Unit (Pallet) Calculations
    pallets_needed = math.ceil(packaging_units_needed /
inputs['units_per_pallet'])
    weight_per_pallet = (inputs['units_per_pallet'] * total_packaging_weight) +
inputs['pallet_weight']
    total_shipment_weight = pallets_needed * weight_per_pallet
    loading_meters = pallets_needed / inputs['stackability_factor'] * 1.2  #
assuming 1.2m per pallet footprint

    # Step 3: Route Identification
    lane_code = (inputs['supplier_country'] + inputs['supplier_zip'] +
                 inputs['dest_country'] + inputs['dest_zip'])

    # Step 4: Price Lookup
    connection = connect_db()
    cursor = connection.cursor()
    query = "SELECT * FROM TransportCosts WHERE lane_code = %s"
    cursor.execute(query, (lane_code,))
    pricing_data = cursor.fetchone()

    if not pricing_data:
        return {"error": "Route not found"}

    # Weight brackets
    brackets = [50, 75, 100, 150, 200, 300, 400, 500, 600, 700, 800, 900, 1000,
                1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 22500]
    selected_bracket = next((b for b in brackets if total_shipment_weight <= b),
max(brackets))
```

```
    price_col = f"price_{selected_bracket}kg"
    price_per_delivery = pricing_data[pricing_data.index(price_col)]

    # Step 5: Final Price Calculations
    price_per_pallet = price_per_delivery / pallets_needed if pallets_needed > 0
else 0
    price_per_piece = price_per_delivery / monthly_demand if monthly_demand > 0
else 0

    return {
        'price_per_delivery': round(price_per_delivery, 2),
        'price_per_pallet': round(price_per_pallet, 2),
        'price_per_piece': round(price_per_piece, 3),
        'total_weight': round(total_shipment_weight, 2),
        'pallets_needed': pallets_needed,
        'lead_time_days': pricing_data[5]  # Adjust index based on column
position
    }
```

Step 3: Streamlit Application

Create the user interface and integrate with the backend logic.

<>python

```
import streamlit as st
import pandas as pd
import os

st.title("Transport Cost Evaluation System")

# Default values
DEFAULT_DESTINATION = {"country": "DE", "zip": "94"}  # Aldersbach, DE 94501

# Input Form
with st.form("input_form"):
    col1, col2 = st.columns(2)

    with col1:
        material_weight = st.number_input("Material Weight per Piece (kg)",
min_value=0.0, value=0.08, step=0.01)
        pieces_per_packaging = st.number_input("Pieces per Packaging Unit",
min_value=1, value=100)
        packaging_weight = st.number_input("Packaging Weight (kg)",
min_value=0.0, value=1.67, step=0.01)
        daily_demand = st.number_input("Daily Demand (pieces)", min_value=1,
value=800)
        deliveries_per_month = st.number_input("Deliveries per Month",
min_value=1, value=2)

    with col2:
        supplier_country = st.text_input("Supplier Country Code", value="IT")
        supplier_zip = st.text_input("Supplier Postal Code (first 2 digits)",
value="23")
        units_per_pallet = st.number_input("Packaging Units per Pallet",
min_value=1, value=48)
        pallet_weight = st.number_input("Pallet Weight (kg)", min_value=0.0,
value=25.0)
        stackability_factor = st.number_input("Stackability Factor", min_value=1,
value=2)

    submit_button = st.form_submit_button("Calculate Transport Cost")

# Process inputs and display results
if submit_button:
    inputs = {
        'material_weight': material_weight,
        'pieces_per_packaging': pieces_per_packaging,
        'packaging_weight': packaging_weight,
        'daily_demand': daily_demand,
        'deliveries_per_month': deliveries_per_month,
        'supplier_country': supplier_country.upper(),
        'supplier_zip': supplier_zip,
        'dest_country': DEFAULT_DESTINATION['country'],
        'dest_zip': DEFAULT_DESTINATION['zip'],
        'units_per_pallet': units_per_pallet,
        'pallet_weight': pallet_weight,
        'stackability_factor': stackability_factor
    }

    result = calculate_transport_cost(inputs)

    if 'error' in result:
        st.error(result['error'])
    else:
        st.subheader("Calculation Results")
        result_df = pd.DataFrame({
            'Metric': ['Price per Delivery (€)', 'Price per Pallet (€)', 'Price
per Piece (€)',
                      'Total Shipment Weight (kg)', 'Pallets Needed', 'Lead Time
(days)'],
            'Value': [result['price_per_delivery'], result['price_per_pallet'],
result['price_per_piece'],
                     result['total_weight'], result['pallets_needed'],
result['lead_time_days']]
        })
        st.table(result_df)

        # Export option
        if st.button("Export Results as CSV"):
            result_df.to_csv("transport_results.csv", index=False)
            st.success("Results exported as CSV!")
```

Step 4: Workflow Automation Features

Add functionality to save scenarios and handle edge cases.

```python
# Save scenario functionality
def save_scenario(inputs, result, scenario_name):
    scenario_data = {**inputs, **result}
    scenario_df = pd.DataFrame([scenario_data])
    if os.path.exists("scenarios.csv"):
        existing_df = pd.read_csv("scenarios.csv")
        scenario_df = pd.concat([existing_df, scenario_df], ignore_index=True)
    scenario_df.to_csv("scenarios.csv", index=False)

# Add to Streamlit app
if submit_button and 'error' not in result:
    scenario_name = st.text_input("Save Scenario (enter name)", "")
    if st.button("Save Scenario") and scenario_name:
        save_scenario(inputs, result, scenario_name)
        st.success(f"Scenario '{scenario_name}' saved!")

# Load saved scenarios
if os.path.exists("scenarios.csv"):
    scenarios = pd.read_csv("scenarios.csv")
    selected_scenario = st.selectbox("Load Saved Scenario",
scenarios['scenario_name'] if 'scenario_name' in scenarios.columns else [])
    if st.button("Load Scenario") and selected_scenario:
        # Implementation to load and populate form with saved data
        st.success(f"Scenario '{selected_scenario}' loaded!")
```

Additional Notes and Error Handling

1. **Input Validation**: Add checks for negative values, empty fields, and invalid country codes.
2. **FTL Pricing**: Implement logic for shipments >34 pallets using special pricing.
3. **National vs. International**: Add logic to differentiate pricing based on country codes.
4. **Weight vs. Loading Meter**: Compare both pricing methods and use the higher value.

This code provides a complete framework for the transport cost evaluation system. Adjust the database connection details, column mappings, and additional logic (like FTL pricing) based on specific data structures in your Excel files. Run the Streamlit app with `streamlit run your_script.py` after setting up the environment and database.