InventorySync: Smart Warehouse Efficiency Dashboard

Project Description

InventorySync is a smart web-based dashboard designed to help warehouse managers and businesses keep track of their inventory, orders, and storage layout in a simple and interactive way. Instead of struggling with messy spreadsheets or scattered reports, this tool brings everything together in one place.

Users can upload their warehouse data (inventory levels, order records, and storage layout) or enter it manually. The system automatically **cleans and prepares the data**, fixing errors and filling in missing details. Then, it **integrates all information** into a single dataset so managers can see the full picture of their warehouse operations.

To make things even more powerful, InventorySync applies **feature engineering** to calculate important metrics like the **Inventory Health Score**, which shows how well stock is managed compared to customer demand.

With this prepared data, a **machine learning model** is trained to **predict future product orders**, helping managers avoid overstocking or understocking. The dashboard presents the results in a clean, interactive design where users can:

- View processed data in an easy-to-read format.
- Get insights into stock levels, order frequency, and warehouse layout.
- See the importance of different factors that affect product demand.
- Predict how many products will be ordered in the future.
- Export processed reports with one click for further use.

In short, InventorySync acts like a **smart assistant** for warehouse management—making data simple, actionable, and predictive so businesses can save time, reduce costs, and increase efficiency.

File Descriptions

1. check_file_lock.py

This file checks if any other program (like Excel or OneDrive) is keeping your data files open and blocking them. If it finds such programs, it gives you the option to close them automatically.

```python
import psutil
import os

def find_processes_using_file(file_path):
    """Find processes that are using the specified file."""
    file_path = os.path.abspath(file_path)
    print(f"Checking for processes using file: {file_path}")

    locking_processes = []
    for proc in psutil.process_iter(['pid', 'name', 'open_files']):
        try:
            for file in proc.info['open_files'] or []:
                if file.path.lower() == file_path.lower():
                    locking_processes.append(proc.info)
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            continue

    if locking_processes:
        print(f"Found {len(locking_processes)} processes using the file:")
        for proc in locking_processes:
            print(f"PID: {proc['pid']}, Name:
{proc['name']}")
    else:
        print("No processes found using the file.")

    return locking_processes
```

```
def terminate_process(pid):
    """Attempt to terminate the process with the
given PID."""
    try:
        proc = psutil.Process(pid)
        proc.terminate()
        proc.wait(timeout=3)
        print(f"Successfully terminated process
with PID {pid}")
        return True
    except (psutil.NoSuchProcess,
psutil.AccessDenied, psutil.TimeoutExpired) as e:
        print(f"Failed to terminate process with
PID {pid}: {e}")
        return False
```

```
if __name__ == "__main__":
    files_to_check = [
        "C:/Users/pc/OneDrive/Desktop/Projects/7th
sem project 2.0/cleaned_inventory_data.csv",
        "C:/Users/pc/OneDrive/Desktop/Projects/7th
sem project 2.0/cleaned_order_data.csv",
        "C:/Users/pc/OneDrive/Desktop/Projects/7th
sem project 2.0/cleaned_layout_data.csv"
    ]

    for file_path in files_to_check:
        if os.path.exists(file_path):
            locking_processes = find_processes_using_file(file_path)
            if locking_processes:
                user_input = input(f"Do you want to terminate these processes? (y/n): ").strip().lower()
                if user_input == 'y':
                    for proc in locking_processes:
                        terminate_process(proc['pid'])
                else:
                    print("Please manually terminate these processes using Task Manager.")
        else:
            print(f"File does not exist: {file_path}")
```

2. dashboard.py

This is the main dashboard of your project. It uses a web app interface to let you upload data, clean it, analyze it, and see predictions in real time. It shows warehouse insights and helps forecast product demand.

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

# Set page config as the first Streamlit command
st.set_page_config(page_title="InventorySync Dashboard", layout="wide")
```

```
print("Current working directory:", os.getcwd())
print("Module search path:", sys.path)
```

```
os.chdir('C:/Users/pc/OneDrive/Desktop/Warehouse
8/')
sys.path.append('C:/Users/pc/OneDrive/Desktop/Ware
house 8/')
```

```
st.write(f"Updated working directory:
{os.getcwd()}")
```

```
try:
    from data_cleaning import process_input_data
    from data_integration import integrate_data
    from feature_engineering import
feature_engineering
    from model_training import train_model,
predict_order_count
except ModuleNotFoundError as e:
    st.error(f"Module import error: {e}. Ensure
all scripts are in
C:/Users/pc/OneDrive/Desktop/Warehouse 8/")
    st.stop()
```

```
# Initialize session state
if 'processed' not in st.session_state:
    st.session_state.processed = False
if 'engineered_data' not in st.session_state:
    st.session_state.engineered_data = None
if 'model' not in st.session_state:
    st.session_state.model = None
```

```
st.title("InventorySync: Real-Time Inventory
Analysis Dashboard")
st.markdown("A tool for inventory management and
demand forecasting.")
```

```
base_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
```

```
# Sidebar for data input
st.sidebar.header("Data Input")
data_source = st.sidebar.radio("Choose Data
Source", ["Upload CSV Files", "Manual Input"])
```

```
if data_source == "Upload CSV Files":
    inventory_file =
st.sidebar.file_uploader("Upload Inventory Data
(CSV)", type="csv")
    order_file = st.sidebar.file_uploader("Upload
Order Data (CSV)", type="csv")
    layout_file = st.sidebar.file_uploader("Upload
Layout Data (CSV)", type="csv")
    process_button = st.sidebar.button("Process
Data")
else:
    st.sidebar.subheader("Manual Data Entry")
    product_id = st.sidebar.text_input("Product
ID", value="P1")
    stock_level = st.sidebar.number_input("Stock
Level", min_value=0, max_value=1000, value=0,
step=1)
    order_frequency =
st.sidebar.number_input("Order Frequency",
min_value=0, max_value=50, value=0, step=1)
    process_button = st.sidebar.button("Process
Data")
```

```
# Display a message while waiting for processing
if not process_button and not
st.session_state.processed:
    st.info("Please enter data and click 'Process
Data' to see the analysis.")
elif process_button:
    try:
        if data_source == "Upload CSV Files" and
inventory_file:
            inventory_data, order_data,
layout_data = process_input_data(
                inventory_file=inventory_file,
                order_file=order_file,
                layout_file=layout_file
            )
            st.write("After process_input_data:")
            st.write(f"inventory_data shape:
{inventory_data.shape},
dtypes:\n{inventory_data.dtypes}")
            st.write(f"order_data shape:
{order_data.shape}, dtypes:\n{order_data.dtypes}")
            st.write(f"layout_data shape:
{layout_data.shape},
dtypes:\n{layout_data.dtypes}")
        else:
            manual_data = {'ProductID': product_id,
'StockLevel': stock_level, 'OrderFrequency':
order_frequency}
            inventory_data, order_data,
layout_data =
process_input_data(manual_data=manual_data)
            st.write("After process_input_data
(manual):")
            st.write(f"inventory_data shape:
{inventory_data.shape},
dtypes:\n{inventory_data.dtypes}")
```

```
            st.write(f"order_data shape:
{order_data.shape}, dtypes:\n{order_data.dtypes}")
            st.write(f"layout_data shape:
{layout_data.shape},
dtypes:\n{layout_data.dtypes}")
```

```
        integrated_data =
integrate_data(inventory_data, order_data,
layout_data)
        st.write(f"After integrate_data,
integrated_data shape: {integrated_data.shape},
dtypes:\n{integrated_data.dtypes}")
```

```
        engineered_data =
feature_engineering(integrated_data)
        st.write(f"After feature_engineering,
engineered_data shape: {engineered_data.shape},
dtypes:\n{engineered_data.dtypes}")
```

```
        model = train_model(engineered_data)
        st.write("Model training completed.")
```

```
        # Store results in session state
        st.session_state.processed = True
        st.session_state.engineered_data =
engineered_data
        st.session_state.model = model
```

```
    except Exception as e:
        st.error(f"An error occurred: {e}")
        st.session_state.processed = False
```

```
# Display analysis if data has been processed
if st.session_state.processed:
    engineered_data =
st.session_state.engineered_data
    model = st.session_state.model
    st.header("Data Analysis")
    st.subheader("Processed Data Summary")
    st.write("Below is a summary of the processed
inventory data:")
    # Add option to exclude rows with 'Unknown' in
Aisle or Shelf
    exclude_unknown = st.checkbox("Exclude rows
with 'Unknown' in Aisle or Shelf", value=False)
    if exclude_unknown:
        display_data =
engineered_data[(engineered_data['Aisle'] !=
'Unknown') & (engineered_data['Shelf'] !=
'Unknown')]
        st.write(f"Displaying {len(display_data)}
rows (after excluding 'Unknown' Aisle/Shelf
values):")
    else:
        display_data = engineered_data
        st.write(f"Displaying all
{len(display_data)} rows:")
    st.write(display_data)
```

```
    st.subheader("Model Details")
    if hasattr(model, 'feature_importances_'):
        features = ['StockLevel', 'OrderFrequency',
'InventoryHealthScore']
        importances = model.feature_importances_
        for feature, importance in zip(features,
importances):
            st.write(f"- Feature: {feature},
Importance: {importance:.4f}")
    else:
        st.write("Model details not available.")
    st.subheader("Predict Products Ordered")
    stock_level_input = st.number_input("Enter
Stock Level for Prediction", min_value=0,
max_value=1000, value=0, step=1)
    order_freq_input = st.number_input("Enter
Order Frequency for Prediction", min_value=0,
max_value=50, value=0, step=1)
    if st.button("Predict"):
        input_data = pd.DataFrame({
            'StockLevel': [stock_level_input],
            'OrderFrequency': [order_freq_input],
            'InventoryHealthScore':
[stock_level_input / (1 + order_freq_input)]
        })
        prediction = predict_order_count(model,
input_data)
        st.write(f"Predicted Products Ordered:
{prediction[0]:.2f}")
```

```
    st.subheader("Export Report")
    if st.button("Download Processed Data"):
        output_file = os.path.join(base_path,
'processed_data_export.csv')
        try:
            engineered_data.to_csv(output_file,
index=False)
            st.download_button(
                "Download CSV",
                data=open(output_file, 'rb'),
                file_name='processed_data_export.c
sv'
            )
        except PermissionError as e:
            st.error(f"PermissionError while
saving export file: {e}")
            st.error("Please ensure the file is
not open in another application and try again.")
```

```
if __name__ == "__main__":
    st.write("Run this file using: streamlit run
dashboard.py")
```

3. data_cleaning.py
This file cleans the raw data. It fixes missing values, removes duplicates, and makes sure everything is in the right format. The goal is to prepare your warehouse data so it’s ready for further analysis.

```
import pandas as pd
import numpy as np
import os

# Note: Removed matplotlib import since we're not
generating images
base_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
```

```
def process_input_data(inventory_file=None,
order_file=None, layout_file=None,
manual_data=None):
    """Process and clean input data."""
    if manual_data:
        inventory_data =
pd.DataFrame([manual_data], columns=['ProductID',
'StockLevel', 'OrderFrequency'])
        order_data =
pd.DataFrame(columns=['OrderID', 'ProductsOrdered',
'ProductID'])
```

```
        layout_data =
pd.DataFrame(columns=['Aisle', 'Shelf',
'ProductID'])
    else:
        # Load datasets with error handling
        try:
            inventory_data =
pd.read_csv(inventory_file)
        except Exception as e:
            raise FileNotFoundError(f"Failed to
load inventory_file: {e}")

        try:
            order_data = pd.read_csv(order_file)
        except Exception as e:
            raise FileNotFoundError(f"Failed to
load order_file: {e}")

        try:
            layout_data = pd.read_csv(layout_file)
        except Exception as e:
            raise FileNotFoundError(f"Failed to
load layout_file: {e}")
```

```
    # Validate required columns
    required_inventory_cols = ['ProductID',
'StockLevel', 'OrderFrequency']
    for col in required_inventory_cols:
        if col not in inventory_data.columns:
            raise KeyError(f"Missing required
column '{col}' in inventory_data")
```

```
    required_order_cols = ['OrderID',
'ProductsOrdered']
    for col in required_order_cols:
        if col not in order_data.columns:
            raise KeyError(f"Missing required 
column '{col}' in order_data")
```

```
    required_layout_cols = ['Aisle', 'Shelf', 
'ProductID']
    for col in required_layout_cols:
        if col not in layout_data.columns:
            raise KeyError(f"Missing required 
column '{col}' in layout_data")
```

```
    # Clean inventory_data
    inventory_data['StockLevel'] = 
pd.to_numeric(inventory_data['StockLevel'], 
errors='coerce').fillna(inventory_data['StockLevel
'].mean()).astype(int)
    inventory_data['OrderFrequency'] = 
pd.to_numeric(inventory_data['OrderFrequency'], 
errors='coerce').fillna(inventory_data['OrderFrequ
ency'].mean()).astype(int)
    inventory_data = 
inventory_data.drop_duplicates()
```

```
    # Clean order_data
    print(f"Total rows in raw order_data: 
{len(order_data)}")
    order_data['ProductsOrdered'] = 
pd.to_numeric(order_data['ProductsOrdered'], 
errors='coerce')
    non_zero_raw = 
(order_data['ProductsOrdered'] > 0).sum()
    print(f"Number of non-zero ProductsOrdered in 
raw data: {non_zero_raw}")

    # Check if ProductsOrdered is all NaN (empty)
    if order_data['ProductsOrdered'].isna().all():
        print("ProductsOrdered column is empty.
Generating synthetic data for ProductsOrdered...")
        order_data['ProductsOrdered'] =
np.random.randint(1, 11, size=len(order_data))

    # Recalculate stats after synthetic data
generation
    non_zero_raw =
(order_data['ProductsOrdered'] > 0).sum()
    print(f"Number of non-zero ProductsOrdered
after synthetic fill (if applied): {non_zero_raw}")
    mean_products_ordered =
order_data['ProductsOrdered'].mean()
    order_data['ProductsOrdered'] =
order_data['ProductsOrdered'].fillna(mean_products
_ordered)
    non_zero_after_fill =
(order_data['ProductsOrdered'] > 0).sum()
    print(f"Mean of ProductsOrdered after cleaning:
{mean_products_ordered}")
    print(f"Number of non-zero ProductsOrdered
after filling NaNs: {non_zero_after_fill}")
```

```
    # Align ProductID with inventory_data
    print(f"Sample of ProductID in inventory_data:
{inventory_data['ProductID'].head().to_list()}")
    # Check if ProductID exists before accessing
it
    if 'ProductID' in order_data.columns:
        print(f"Sample of ProductID in order_data
before alignment:
{order_data['ProductID'].head().to_list()}")
    else:
        print("ProductID column not found in 
order_data before alignment.")
```

```
    # Add ProductID to order_data if missing
    if 'ProductID' not in order_data.columns or 
order_data['ProductID'].isna().all():
        print("ProductID missing or all NaN in 
order_data, aligning with inventory_data...")
        inventory_product_ids = 
inventory_data['ProductID'].unique()
        num_orders = len(order_data)
        order_data['ProductID'] = 
[inventory_product_ids[i % 
len(inventory_product_ids)] for i in 
range(num_orders)]
```

```
    # Print sample after alignment
    print(f"Sample of ProductID in order_data 
after alignment: 
{order_data['ProductID'].head().to_list()}")
    order_data = order_data.drop_duplicates()
```

```
    # Clean layout_data
    layout_data = 
layout_data.dropna(subset=['ProductID'])
    layout_data = layout_data.drop_duplicates()
```

```
    # Save cleaned data with error handling
    try:
        inventory_data.to_csv(os.path.join(base_pa
th, 'cleaned_inventory_data.csv'), index=False)
        order_data.to_csv(os.path.join(base_path, 
'cleaned_order_data.csv'), index=False)
        layout_data.to_csv(os.path.join(base_path, 
'cleaned_layout_data.csv'), index=False)
    except PermissionError as e:
        print(f"PermissionError while saving files: {e}")
        print("Please ensure the output files are not open in another application and try again.")
        raise
```

```
    return inventory_data, order_data, layout_data
```

```
if __name__ == "__main__":
    try:
        inventory_data, order_data, layout_data = process_input_data(
            inventory_file=os.path.join(base_path, 'inventory_data_noised.csv'),
            order_file=os.path.join(base_path, 'order_data_noised.csv'),
            layout_file=os.path.join(base_path, 'warehouse_layout_data_noised.csv')
        )
        print("Data cleaning completed.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        raise
```

4. data_integration.py

This file combines different types of data—like inventory, orders, and warehouse layout—into a single dataset. This makes it easier to analyze everything together.

```
import pandas as pd
import os

base_path = 'C:/Users/pc/OneDrive/Desktop/Projects/7th sem project 2.0/'
```

```
integrated_file = os.path.join(base_path,
'integrated_data.csv')
```

```
def integrate_data(inventory_data, order_data,
layout_data):
    """Integrate inventory, order, and layout data
into a single DataFrame."""
    # Validate required columns
    required_columns = {
        'inventory_data': ['ProductID',
'StockLevel', 'OrderFrequency'],
        'order_data': ['OrderID',
'ProductsOrdered', 'ProductID'],
        'layout_data': ['Aisle', 'Shelf',
'ProductID']
    }
    for df_name, cols in required_columns.items():
        df = locals()[df_name]
        for col in cols:
            if col not in df.columns:
                raise KeyError(f"Missing required
column '{col}' in {df_name}")
```

```
    # Ensure ProductsOrdered is numeric and fill
NaNs before merge
    order_data['ProductsOrdered'] =
pd.to_numeric(order_data['ProductsOrdered'],
errors='coerce').fillna(0)
    non_zero_before_merge =
(order_data['ProductsOrdered'] > 0).sum()
    print(f"Number of non-zero ProductsOrdered in
order_data before merge: {non_zero_before_merge}")
```

```
    # Use an inner join to keep only matched rows
    merged_data = pd.merge(inventory_data,
order_data, on='ProductID', how='inner')
    nan_count =
merged_data['ProductsOrdered'].isna().sum()
    print(f"Number of NaN ProductsOrdered after
first merge (inventory_data + order_data):
{nan_count}")
    if nan_count > 0:
        print(f"Sample of rows with NaN
ProductsOrdered:
{merged_data[merged_data['ProductsOrdered'].isna()]
.head()}")
```

```
    integrated_data = pd.merge(merged_data,
layout_data, on='ProductID', how='left')
```

```
    # Fill missing values
    integrated_data['StockLevel'] =
integrated_data['StockLevel'].fillna(0)
    integrated_data['OrderFrequency'] =
integrated_data['OrderFrequency'].fillna(0)
    integrated_data['Aisle'] =
integrated_data['Aisle'].fillna('Unknown')
    integrated_data['Shelf'] =
integrated_data['Shelf'].fillna('Unknown')
```

```
    # Debug prints
    total_rows = len(integrated_data)
    matched_rows = len(integrated_data) -
integrated_data['OrderID'].isna().sum()
    non_zero_count =
(integrated_data['ProductsOrdered'] > 0).sum()
    unmatched_rows =
integrated_data['OrderID'].isna().sum()
```

```
    unknown_aisle_count = (integrated_data['Aisle']
== 'Unknown').sum()
    unknown_shelf_count = (integrated_data['Shelf']
== 'Unknown').sum()
    print(f"Total rows after integration:
{total_rows}")
    print(f"Number of matched rows (non-null
OrderID): {matched_rows}")
    print(f"Number of unmatched rows (null
OrderID): {unmatched_rows}")
    print(f"Number of rows with non-zero
ProductsOrdered: {non_zero_count}")
    if non_zero_count > 0:
        non_zero_sample =
integrated_data[integrated_data['ProductsOrdered']
 > 0]['ProductsOrdered'].head(5).to_list()
        print(f"Sample of non-zero ProductsOrdered
values: {non_zero_sample}")
    print(f"Number of rows with Aisle as 'Unknown':
{unknown_aisle_count}")
    print(f"Number of rows with Shelf as 'Unknown':
{unknown_shelf_count}")
```

```
    # Save integrated data
    try:
        integrated_data.to_csv(integrated_file,
index=False)
    except PermissionError as e:
        print(f"PermissionError while saving
integrated_data.csv: {e}")
        print("Please ensure the file is not open
in another application and try again.")
        raise
```

```
    return integrated_data
```

```
if __name__ == "__main__":
    try:
        cleaned_inventory_file =
os.path.join(base_path,
'cleaned_inventory_data.csv')
        cleaned_order_file =
os.path.join(base_path, 'cleaned_order_data.csv')
        cleaned_layout_file =
os.path.join(base_path, 'cleaned_layout_data.csv')
```

```
        inventory_data =
pd.read_csv(cleaned_inventory_file)
        order_data =
pd.read_csv(cleaned_order_file)
        layout_data =
pd.read_csv(cleaned_layout_file)
```

```
        integrated_data =
integrate_data(inventory_data, order_data,
layout_data)
        print("Data integration completed.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        raise
```

5. feature_engineering.py

This file creates new useful information from the integrated data. For example, it calculates an "Inventory Health Score" that shows how balanced stock levels are compared to orders.

```
import pandas as pd
import os
```

```
base_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
engineered_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
os.makedirs(engineered_path, exist_ok=True)
```

```
def feature_engineering(integrated_data):
    """Perform feature engineering on the
integrated data."""
    required_columns = ['ProductID', 'StockLevel',
'OrderFrequency', 'ProductsOrdered']
    for col in required_columns:
        if col not in integrated_data.columns:
            raise KeyError(f"Missing required
column '{col}' in integrated_data")
```

```
    integrated_data = integrated_data.copy()
```

```
    # Calculate InventoryHealthScore
    integrated_data['InventoryHealthScore'] = (
        integrated_data['StockLevel'] / (1 +
integrated_data['OrderFrequency'])
    )
```

```
    # Save engineered data
    try:
        integrated_data.to_csv(os.path.join(engine
ered_path, 'engineered_data.csv'), index=False)
    except PermissionError as e:
        print(f"PermissionError while saving
engineered_data.csv: {e}")
        print("Please ensure the file is not open
in another application and try again.")
        raise
    return integrated_data
```

```
if __name__ == "__main__":
    try:
        integrated_data =
pd.read_csv(os.path.join(base_path,
'integrated_data.csv'))
        engineered_data =
feature_engineering(integrated_data)
        print("Feature engineering completed.")
    except Exception as e:
        print(f"Error in feature_engineering.py:
{e}")
        raise
```

6. model_training.py

This file trains a machine learning model. The model learns from past data to predict how many products might be ordered in the future, helping improve warehouse efficiency.

```
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error,
r2_score
import os

print("Loading model_training.py - Version: 2025-
06-01 01:00 PM IST")
```

```
base_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
```

```
def train_model(engineered_data):
    """Train a model to predict
ProductsOrdered."""
    required_columns = ['StockLevel',
'OrderFrequency', 'InventoryHealthScore',
'ProductsOrdered']
    for col in required_columns:
        if col not in engineered_data.columns:
            raise KeyError(f"Missing required
column '{col}' in engineered_data")
```

```
    # Log initial data stats
    non_zero_count =
(engineered_data['ProductsOrdered'] > 0).sum()
    print(f"Total rows in training data:
{len(engineered_data)}")
    print(f"Number of rows with non-zero
ProductsOrdered: {non_zero_count}")
```

```
    # Filter for non-zero ProductsOrdered to
improve predictions
    training_data =
engineered_data[engineered_data['ProductsOrdered']
 > 0]
```

```
    if len(training_data) == 0:
        print("Warning: No non-zero
ProductsOrdered values in training data. Using a
dummy model that predicts 0.")
        class DummyModel:
            def predict(self, X):
                return [0] * len(X)
            scaler = None
        return DummyModel()
    X = training_data[['StockLevel',
'OrderFrequency', 'InventoryHealthScore']]
    y = training_data['ProductsOrdered']
```

```
    # Scale the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```
    X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
    model = RandomForestRegressor(n_estimators=100,
random_state=42)
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print(f"Number of training samples (after
filtering): {len(X_train)}")
```

```
    # Store the scaler in the model for prediction
    model.scaler = scaler
    return model
```

```
def predict_order_count(model, input_data):
    """Predict ProductsOrdered using the trained
model."""
    required_columns = ['StockLevel',
'OrderFrequency', 'InventoryHealthScore']
    for col in required_columns:
        if col not in input_data.columns:
            raise KeyError(f"Missing required
column '{col}' in input_data for prediction")
```

```
    if getattr(model, 'scaler', None) is None:
        print("Prediction: Using dummy model, 
returning 0.")
        return [0]
```

```
    X_input = input_data[['StockLevel', 
'OrderFrequency', 'InventoryHealthScore']]
    X_input_scaled = 
model.scaler.transform(X_input)
    prediction = model.predict(X_input_scaled)
    print(f"Raw prediction before max(0, x): 
{prediction[0]}")
    prediction = max(0, prediction[0])
    return [prediction]
```

```
if __name__ == "__main__":
    try:
        engineered_data_path = 
os.path.join(base_path, 'engineered_data.csv')
        if not 
os.path.exists(engineered_data_path):
            raise FileNotFoundError(f"Engineered 
data file not found at: {engineered_data_path}")
        engineered_data = 
pd.read_csv(engineered_data_path)
        model = train_model(engineered_data)
        print("Model training completed.")
    except Exception as e:
        print(f"Error in model_training.py: {e}")
        raise
```

7. run_project.py

This is like a manager file. It runs all other files step by step (cleaning, integration, feature making, training) and finally opens the dashboard automatically.

```python
import os
import subprocess
import time

# Define paths
base_path =
'C:/Users/pc/OneDrive/Desktop/Projects/7th sem
project 2.0/'
files_to_remove = [
    os.path.join(base_path,
'cleaned_inventory_data.csv'),
    os.path.join(base_path,
'cleaned_order_data.csv'),
    os.path.join(base_path,
'cleaned_layout_data.csv'),
    os.path.join(base_path, 'integrated_data.csv'),
    os.path.join(base_path, 'engineered_data.csv')
]
```

```python
print("Starting the InventorySync pipeline...")
```

```python
# Remove old files if they exist with retry
mechanism
for file in files_to_remove:
    if os.path.exists(file):
        max_retries = 5  # Increased retries
        for attempt in range(max_retries):
            try:
                print(f"Attempting to remove old
file: {file} (Attempt {attempt +
1}/{max_retries})")
                os.remove(file)
                print(f"Successfully removed:
{file}")
                break
            except PermissionError as e:
                print(f"PermissionError: {e}.
Retrying in 5 seconds...")
                time.sleep(5)  # Increased delay
to 5 seconds
                if attempt == max_retries - 1:
                    print(f"Failed to remove {file}
after {max_retries} attempts.")
                    print("Please ensure the file
is not open in another application (e.g., Excel,
File Explorer).")
                    print("Also, consider pausing
OneDrive syncing temporarily.")
                    user_input = input("Proceed
without deletion? (y/n): ").strip().lower()
                    if user_input != 'y':
                        print("Exiting pipeline.
Please resolve the file lock and try again.")
                        exit(1)
            except Exception as e:
                print(f"Unexpected error while
removing {file}: {e}")
                break
```

```
# Run pipeline steps
try:
    print("Running data_cleaning.py...")
    subprocess.run(['python', 'data_cleaning.py'],
check=True)
    print("data_cleaning.py completed
successfully.")
except subprocess.CalledProcessError as e:
    print(f"Error: data_cleaning.py failed with
exit code {e.returncode}")
    exit(e.returncode)
```

```
try:
    print("Running data_integration.py...")
    subprocess.run(['python',
'data_integration.py'], check=True)
    print("data_integration.py completed
successfully.")
except subprocess.CalledProcessError as e:
    print(f"Error: data_integration.py failed with
exit code {e.returncode}")
    exit(e.returncode)
```

```
try:
    print("Running feature_engineering.py...")
    subprocess.run(['python',
'feature_engineering.py'], check=True)
    print("feature_engineering.py completed
successfully.")
except subprocess.CalledProcessError as e:
    print(f"Error: feature_engineering.py failed
with exit code {e.returncode}")
    exit(e.returncode)
```

```
try:
    print("Running model_training.py...")
    subprocess.run(['python', 'model_training.py'],
check=True)
```

```
    print("model_training.py completed
successfully.")
except subprocess.CalledProcessError as e:
    print(f"Error: model_training.py failed with
exit code {e.returncode}")
    exit(e.returncode)
```

```
try:
    print("Launching dashboard.py...")
    subprocess.run(['streamlit', 'run',
'dashboard.py'], check=True)
except subprocess.CalledProcessError as e:
    print(f"Error: dashboard.py failed with exit
code {e.returncode}")
    exit(e.returncode)
```

```
print("InventorySync pipeline completed
successfully.")
```

8. utils.py

This file contains extra helper functions. For example, it can calculate a different version of inventory health or save charts that show patterns in the data.

```
import pandas as pd
import matplotlib.pyplot as plt

def calculate_inventory_health(data):
    """Calculate unique Inventory Health Score."""
    data['InventoryHealthScore'] =
data['StockLevel'] / (data['OrderCount'] + 1) *
data['OrderFrequency']
    return data
```

```python
def save_custom_plot(data, column, title, filename,
color='#1ABC9C'):
    """Save a custom-styled plot."""
    plt.figure(figsize=(10, 6))
    plt.hist(data[column], bins=30, color=color,
alpha=0.7, edgecolor='black')
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)
    plt.savefig(filename)
    plt.close()
```