Documentation for Dispensation and Warehouse Stock Monitoring Application

Introduction

This documentation provides a detailed overview of the Dispensation and Warehouse Stock Monitoring application, which is built using the Streamlit framework. This application allows users to monitor product dispensation trends and warehouse stock levels, detect sudden increases in product dispensation, and forecast future dispensation trends using machine learning models. Additionally, the application enables users to validate GTIN entries in the provided database.

Table of Contents

Application Overview

The Dispensation and Warehouse Stock Monitoring application is designed to:

- Detect sudden increases in product dispensation based on user-defined thresholds and rolling window sizes.
- Forecast future product dispensation using machine learning models such as ARIMA, Holt-Winters, and Moving Average.
- Monitor warehouse stock levels and identify products that fall below specified minimum stock levels.
- Validate GTIN entries in the provided database.

Data Sources

The application uses the following data sources:

1. **Product Dispensation Data**: Stored in the file **product_dispensation_data.csv**, this data includes historical product dispensation information and the corresponding dates.
2. **Warehouse Data**: Stored in the file **warehouse_data.csv**, this data includes information on current stock levels of different products in various warehouses.
3. **GTIN Data**: Stored in the file **GTIN.csv**, this data includes GTINs and associated sender information.

Application Features

The application consists of several key features:

- **Sudden Increase Detection**: Identifies sudden increases in product dispensation based on user-defined thresholds and rolling window sizes.
- **Forecasting**: Forecasts future product dispensation using various models, including ARIMA. Users can select the product and number of days to forecast.
- **Warehouse Stock Levels**: Allows users to set minimum stock levels for each warehouse and analyze stock levels to identify products that fall below the specified minimums.
- **GTIN Check**: Allows users to input a GTIN and check whether it exists in the provided database.

Functions and Modules

Detect Sudden Increase

- **Function Name**: **detect_sudden_increase**
- **Description**: Detects sudden increases in product dispensation based on a specified threshold and rolling window size.
- **Parameters**:
    - **product_to_analyze** (str): Name of the product to analyze.
    - **threshold** (float): The threshold value to identify sudden increases.
    - **rolling_window** (int): The size of the rolling window for calculating the moving average.
- **Returns**: A tuple containing two DataFrames: one with data about sudden increases and another with the original product data, including the calculated moving average.

Forecasting

- **Model**: The application uses Moving Average model to forecast future product dispensation. Other models like Holt-Winters and ARIMA can be added as needed.

Moving Average (MA) Model

The Moving Average (MA) model is a simple forecasting model that calculates the average of past values over a specified rolling window. It uses the last calculated moving average value as the forecasted value for the next selected days.

- **Function Name**: **train_ma_model**
- **Description**: Trains a Moving Average (MA) model on the given product data and returns forecasted values for the specified number of days.
- **Parameters**:
    - **data** (pd.DataFrame): Historical dispensation data with 'Date' and 'Dispensation' columns.
    - **rolling_window** (int): The size of the rolling window for calculating the moving average.
    - **forecast_days** (int): The number of days to forecast.
- **Returns**: An **np.array** containing forecasted values for the specified number of days.

The function works as follows:

1. **Calculate Moving Average**: The function calculates the moving average of the 'Dispensation' column in the provided data, using the specified rolling window size.
2. **Forecast Future Values**: Once the moving average is calculated, the function uses the last calculated value as the forecast for the next selected days.
3. **Return Forecast**: The function returns an array (**np.array**) containing the forecasted values for the specified number of days.

ARIMA Model

The ARIMA (Autoregressive Integrated Moving Average) model is a widely used statistical model for forecasting time series data. It combines autoregressive, integrated, and moving average components to make predictions.

- **Function Name**: **train_arima_model**

- **Description**: Trains an ARIMA model on the provided data and generates a forecast for the specified number of days.
- **Parameters**:
  - **data** (pd.DataFrame): Historical data with 'Date' and 'Dispensation' columns.
  - **forecast_days** (int): The number of days to forecast.
- **Returns**: A forecasted value array (**np.array**) for the next selected days.

The function works as follows:

1. **Prepare Data**: The function sets the 'Date' column as the index for the input data.
2. **Fit ARIMA Model**: The function fits an ARIMA model to the historical data using specified parameters.
3. **Generate Forecast**: The function generates a forecast for the next selected days using the fitted model.

Holt-Winters Model

The Holt-Winters model (also known as the Exponential Smoothing model) is a forecasting model that handles both trend and seasonality in time series data.

- **Function Name**: **train_holt_winters_model**
- **Description**: Trains a Holt-Winters model on the provided data and generates a forecast for the specified number of days.
- **Parameters**:
  - **data** (pd.DataFrame): Historical data with 'Date' and 'Dispensation' columns.
  - **forecast_days** (int): The number of days to forecast.
- **Returns**: A forecasted value array (**np.array**) for the next selected days.

The function works as follows:

1. **Prepare Data**: The function sets the 'Date' column as the index for the input data.
2. **Fit Holt-Winters Model**: The function fits a Holt-Winters model to the historical data using specified seasonal periods and trends.
3. **Generate Forecast**: The function generates a forecast for the next selected days using the fitted model.

Warehouse Stock Levels

- **Functionality**: Allows users to set minimum stock levels for each warehouse and analyze stock levels. Highlights products with stock levels below the specified minimums.
- **Implementation**: Reads warehouse data from the **warehouse_data.csv** file, checks stock levels against user-defined minimums, and displays the results.

These three functions are used for different forecasting models, allowing users to select the most appropriate method for their data and analysis needs.

GTIN Check

- **Function Name**: **check_gtin**
- **Description**: Validates the input GTIN by checking whether it exists in the provided GTIN data.
- **Parameters**:
    - **gtin** (int): The input GTIN to be validated.
- **Returns**: Displays the associated sender information if the GTIN exists in the database; otherwise, displays an error message.

User Interface

The application uses the Streamlit framework for a user-friendly interface. Users interact with the application through various widgets:

- **Sidebars**: Sidebars contain settings for selecting products, inputting dates, thresholds, forecasting parameters, and minimum stock levels for warehouses. They also include a GTIN input field.
- **Main Content**: The main content area displays results, data tables, and visualizations (plots) based on user inputs.

Installation and Dependencies

The application relies on the following Python packages:

- **streamlit**: A web application framework for creating interactive apps.
- **pandas**: A data analysis and manipulation library.
- **matplotlib**: A plotting library for visualizations.

WebApp Code: (app.py)

```
import streamlit as st

import pandas as pd

import matplotlib.pyplot as plt
from medications_info import display_info

from timeseries_models import train_arima_model, train_ma_model,
train_holt_winters_model


# Load data

data = pd.read_csv('product_dispensation_data_v2.csv')

data['Date'] = pd.to_datetime(data['Date'])


warehouse_df = pd.read_csv('warehouse_data.csv')


# Function to detect sudden increase

def detect_sudden_increase(product_to_analyze, threshold, rolling_window):

    product_data = data[data['Product'] == product_to_analyze]

    product_data['Moving_Avg'] =
product_data['Dispensation'].rolling(window=rolling_window, min_periods=1).mean()

    product_data['Dispensation_Diff'] = product_data['Dispensation'] -
product_data['Moving_Avg']

    sudden_increase = product_data[product_data['Dispensation_Diff'] > threshold]

    return sudden_increase, product_data


# Streamlit app

st.set_page_config(layout="wide")


# remove header

st.markdown("""

            <style>

            .st-emotion-cache-zq5wmm
            {
                visibility: hidden;
            }
            .viewerBadge_container__r5tak styles_viewerBadge__CvC9N
            {
                visibility: hidden;
            }
            </style>
            """, unsafe_allow_html = True)

# Company logo
st.image('myverimed logo transparent.png', width=300)

# Title
st.title('Dispensation and Warehouse Stock Monitoring Application')

# Sidebar
st.sidebar.title('Settings')
st.sidebar.header('Historical Analysis')
selected_product = st.sidebar.selectbox('Select Product', data['Product'].unique())
start_date = st.sidebar.date_input('Start Date', min_value=data['Date'].min(),
max_value=data['Date'].max())
end_date = st.sidebar.date_input('End Date', min_value=start_date,
max_value=data['Date'].max())
rolling_window = st.sidebar.slider('Rolling Window', min_value=0, max_value=30,
step=1, value=7)
threshold = 2


# Button to trigger detection
if st.sidebar.button('Detect Sudden Increase'):
```

```python
    sudden_increase, product_data = detect_sudden_increase(selected_product, threshold, rolling_window)

    # Convert start and end dates to pandas Timestamp objects
    start_date = pd.Timestamp(start_date)
    end_date = pd.Timestamp(end_date)

    # Filter product data based on selected date range
    filtered_data = product_data.loc[(product_data['Date'] >= start_date) & (product_data['Date'] <= end_date)]

    # Filter sudden increase data within the selected date range
    sudden_increase_filtered = sudden_increase.loc[(sudden_increase['Date'] >= start_date) & (sudden_increase['Date'] <= end_date)]

  # Display result
    if not sudden_increase_filtered.empty:
        st.warning(f"Alert: Sudden increase in dispensation of {selected_product} detected!")
        st.write(display_info(selected_product))
        st.dataframe(sudden_increase_filtered)
    else:
        st.info(f"No sudden increase in dispensation of {selected_product} detected within the specified date range.")

    # Plot
    fig, ax = plt.subplots()
    ax.plot(filtered_data['Date'], filtered_data['Dispensation'], label='Dispensation')
    ax.plot(filtered_data['Date'], filtered_data['Moving_Avg'], label='Moving Average', linestyle='--', color='red')

    for idx, row in sudden_increase_filtered.iterrows():
        ax.scatter(row['Date'], row['Dispensation'], color='red', marker='o')

    ax.set_xlabel('Date')
    ax.set_ylabel('Dispensation')
    ax.set_title('Dispensation Trend')
    ax.legend()
    # Rotate x-axis labels vertically
    plt.xticks(rotation=90)
    st.pyplot(fig)

# Forecasting using selected model

# Dropdown to select forecasting model
st.sidebar.header('Forecast Dispensation')
product_to_forecast = st.sidebar.selectbox('Select Product to forecast', data['Product'].unique())
product_data_to_forecast = data[data['Product'] == product_to_forecast]
selected_model = 'ARIMA'
#selected_model = st.sidebar.selectbox('Select Forecasting Model', ['ARIMA', 'Holt Winters', 'Moving Average'])
forecast_days = st.sidebar.slider('Forecast days', min_value=0, max_value=31, step=1, value=7)
#st.sidebar.subheader('Select rolling window only for Moving Average model')
#moving_avg_rolling_window_to_forecast = st.sidebar.slider('Moving Avg Rolling Window', min_value=0, max_value=30, step=1, value=1)

# Button to trigger analysis and forecasting
if st.sidebar.button('Forecast'):
    forecast = train_arima_model(product_data_to_forecast, forecast_days)
    #if selected_model == 'ARIMA':
    #    forecast = train_arima_model(product_data_to_forecast, forecast_days)
    #elif selected_model == 'Holt Winters':

    #    forecast = train_holt_winters_model(product_data_to_forecast,
forecast_days)

    #elif selected_model == 'Moving Average':

    #    forecast = train_ma_model(product_data_to_forecast,
moving_avg_rolling_window_to_forecast, forecast_days)



     # Round the forecast values

    forecast = [round(value) for value in forecast]



    # Display forecasted values

    st.subheader(f"Forecasted Dispensation for the next *{forecast_days}* Days",

                 divider='green')

    forecast_dates = pd.date_range(start=end_date + pd.Timedelta(days=1),
periods=forecast_days)

    forecast_df = pd.DataFrame({'Date': forecast_dates, 'Forecast': forecast})

    st.write(forecast_df)

    if selected_model == "Moving Average":

        st.subheader("Note:")

        st.write("The forecasted values are all the same for the selected days
because the Moving Average (MA) model simply uses the last computed moving average
value as the forecast for all the selected days into the future.")



    # Plot historical data and forecast

    fig, ax = plt.subplots()

    #ax.plot(product_data_to_forecast['Date'],
product_data_to_forecast['Dispensation'], label='Historical Dispensation')

    ax.plot(forecast_df['Date'], forecast_df['Forecast'], label='Forecast',
linestyle='--', color='red')

    ax.set_xlabel('Date')

    ax.set_ylabel('Dispensation')

    ax.set_title('Historical Dispensation and Forecast')

    ax.legend()
```

```python
    # Rotate x-axis labels vertically

    plt.xticks(rotation=90)

    st.pyplot(fig)



# Set minimum stock levels

st.sidebar.title('Set Minimum Stock Levels')

min_stock_levels = {}

for warehouse in warehouse_df['Warehouse'].unique():

    min_stock_levels[warehouse] = st.sidebar.number_input(f'Min Stock Level for
{warehouse}', min_value=0)



# Button to trigger analysis

if st.sidebar.button('Analyze Stock Levels'):

    warehouse_df['Below Minimum'] = warehouse_df['Stock Level'] <
warehouse_df['Warehouse'].map(min_stock_levels)



    st.subheader('Warehouse Stock Levels')

    st.dataframe(warehouse_df.style.apply(lambda x: ['background: lightcoral' if
x['Below Minimum'] else '' for _ in x], axis=1))



    st.subheader('Summary of Deficiencies')

    for warehouse in warehouse_df['Warehouse'].unique():

        deficiency_count = warehouse_df[(warehouse_df['Warehouse'] == warehouse) &
(warehouse_df['Below Minimum'])].shape[0]

        st.write(f"{warehouse}: {deficiency_count} product(s) below minimum stock
level")



# Load GTIN data

gtin_df = pd.read_csv('GTIN.csv',  encoding='ISO-8859-1')



# Sidebar title and input for GTIN

st.sidebar.title('Check GTIN')
```

```
gtin_input = st.sidebar.number_input('GTIN:', value=0)


def check_gtin(gtin):

    result = gtin_df[gtin_df['GTIN'] == gtin]

    if not result.empty:

        st.success("Sender: " + result['Sender'].values[0])

    else:

        st.error("GTIN not found in database.")


# Check if GTIN input is provided and call the function

if gtin_input:

    check_gtin(gtin_input)
```

Machine Learning models code: (timeseries_models.py)

```
import numpy as np

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from statsmodels.tsa.holtwinters import SimpleExpSmoothing



# Function to train ARIMA model and generate forecast

def train_arima_model(data, forecast_days):

    # Prepare data for ARIMA

    product_data = data.copy()

    product_data.set_index('Date', inplace=True)


    # Fit ARIMA model
```

```python
    model = ARIMA(product_data['Dispensation'], order=(5,1,0))

    model_fit = model.fit()


    # Forecast for next selected days

    forecast = model_fit.forecast(steps=forecast_days)


    return forecast


# Function to train SARIMA model and generate forecast

def train_sarima_model(data, forecast_days):

    # Prepare data for SARIMA

    product_data = data.copy()

    product_data.set_index('Date', inplace=True)


    # Fit SARIMA model

    model = SARIMAX(product_data['Dispensation'], order=(1, 1, 1),
seasonal_order=(1, 1, 1, 12))

    model_fit = model.fit()


    # Forecast for next selected days

    forecast = model_fit.forecast(steps=forecast_days)


    return forecast


# Function to train Holt Winters model and generate forecast

def train_holt_winters_model(data, forecast_days):

    # Prepare data for Holt Winters

    product_data = data.copy()

    product_data.set_index('Date', inplace=True)
    # Fit Holt Winters model
    model = ExponentialSmoothing(product_data['Dispensation'], seasonal_periods=12, trend='add', seasonal='add')
    model_fit = model.fit()

    # Forecast for next selected days
    forecast = model_fit.forecast(steps=forecast_days)

    return forecast


def train_ma_model(data, rolling_window, forecast_days):
    """
    Train Moving Average (MA) model and return forecasted values for the next selected days.

    Parameters:
        data (pd.DataFrame): Historical dispensation data with 'Date' and 'Dispensation' columns.
        rolling_window (int): Size of the rolling window for calculating the moving average.

    Returns:
        np.array: Forecasted values for the next selected days.
    """
    # Calculate moving average
    moving_avg = data['Dispensation'].rolling(window=rolling_window, min_periods=1).mean()

    # Use the last value as the forecast for the next selected days
    last_value = moving_avg.iloc[-1]
    forecast = np.full(forecast_days, last_value)
    return forecast
```

Medications code: (medications_info.py)

```
import streamlit as st


medications = {

    "Zinnat": {

        "Related Medications": ["Cefzil", "Ceclor", "Rocephin"],

        "Common Uses": "Antibiotic for bacterial infections, such as respiratory infections, skin infections"

    },

    "Beyfortus": {

        "Related Medications": ["Amoxicillin", "Ampicillin", "Augmentin"],

        "Common Uses": "Acute Bronchiolitis, affecting up to 30% of babies below 2year-old during the wither and raising to respiratory detress, with cases of death, Antibiotic for bacterial infections, such as respiratory infections, ear infections, urinary tract infections"

    },

    "Actilyse": {

        "Related Medications": ["Ciprofloxacin"],

        "Common Uses": "Thrombolytic agent used to dissolve blood clots in heart attack, stroke, pulmonary embolism"

    },

    "Ciprofloxacin": {

        "Related Medications": ["Levofloxacin", "Ofloxacin", "Moxifloxacin"],

        "Common Uses": "Acute Respiratory Infection, Antibiotic for bacterial infections, such as urinary tract infections, respiratory infections, skin infections"

    },

    "Zophren": {
        "Related Medications": ["Ondansetron", "Metoclopramide", "Domperidone"],

        "Common Uses": "Anti-nausea medication used for chemotherapy-induced
nausea, post-operative nausea"

    },

    "Ramipril": {

        "Related Medications": ["Enalapril", "Lisinopril", "Losartan"],

        "Common Uses": "Cardiac Insufficiency, Angiotensin-converting enzyme (ACE)
inhibitor used for high blood pressure, heart failure, diabetic kidney disease"

    },

    "Hydrochlorothiazid": {

        "Related Medications": ["Chlorthalidone", "Indapamide",
"Hydrochlorothiazide/Lisinopril"],

        "Common Uses": "Cardiac Insufficiency, Diuretic used for high blood
pressure, edema (fluid retention)"

    },

    "Bisoprolol": {

        "Related Medications": ["Metoprolol", "Atenolol", "Carvedilol"],

        "Common Uses": "Cardiac Insufficiency, Beta-blocker used for high blood
pressure, heart failure, angina (chest pain)"

    },

    "Digoxine": {

        "Related Medications": ["Digitoxin", "Dobutamine", "Milrinone"],

        "Common Uses": "Cardiac Insufficiency , Cardiac glycoside used for heart
failure, atrial fibrillation"

    },

    "Fluindione": {

        "Related Medications": ["Warfarin", "Dabigatran", "Rivaroxaban"],

        "Common Uses": "Cardiac Insufficiency, Anticoagulant (blood thinner) used
for preventing blood clots, stroke prevention in atrial fibrillation"

    },

    "Hycamtin": {

        "Related Medications": ["Topotecan", "Etoposide", "Cisplatin"],
        "Common Uses": "Chemotherapy medication used for small cell lung cancer, 
ovarian cancer"

    },

    "Tofranil": {

        "Related Medications": ["Amitriptyline", "Nortriptyline", "Imipramine"],

        "Common Uses": "Tricyclic antidepressant used for depression, enuresis 
(bedwetting)"

    },

    "Rimactan": {

        "Related Medications": ["Isoniazid", "Pyrazinamide", "Ethambutol"],

        "Common Uses": "Antibiotic used to treat tuberculosis and other bacterial 
infections"

    }

}



# Function to display related medications and common uses

def display_info(medication):

    if medication in medications:

        related_medications = medications[medication]["Related Medications"]

        common_uses = medications[medication]["Common Uses"]

        st.write(f"**Related Medications**: {', '.join(related_medications)}")

        st.write(f"**Common Uses**: {common_uses}")

    else:

        return "Medication not found in database."
```

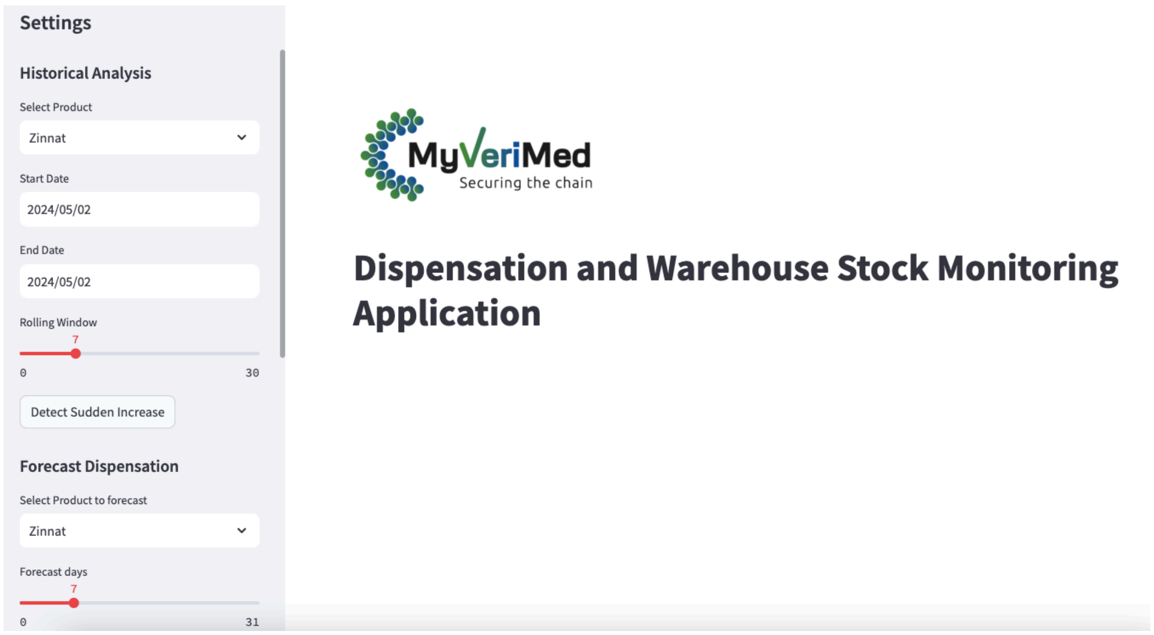
Requirements: (requirements.txt)

```
streamlit

pandas

matplotlib

seaborn

statsmodels
```

WebApp link:

https://mvmed-app.streamlit.app/

WebApp preview:



Settings

Historical Analysis

Select Product: Zinnat

Start Date: 2024/05/02

End Date: 2024/05/02

Rolling Window: 7 (0–30)

Detect Sudden Increase

Forecast Dispensation

Select Product to forecast: Zinnat

Forecast days: 7 (0–31)

MyVeriMed
Securing the chain

Dispensation and Warehouse Stock Monitoring Application