**BUILDING A SALES PREDICTION WEB APP WITH STREAMLIT**

In today's data-driven world, businesses are increasingly relying on machine learning models to make predictions and gain insights. Sales prediction is one such application that can help businesses optimize their operations and make informed decisions. In this article, we'll explore how to create a Sales Prediction Web App using Streamlit, a popular Python framework for building interactive web applications. We'll walk through the code step by step and explain the key components of the application.

**Introduction**

Streamlit is an open-source Python library that allows developers to create web applications with minimal effort. It's well-suited for building data-driven applications, and it's beginner-friendly. In this tutorial, we'll use Streamlit to create a web app that predicts sales based on various input parameters.

**Prerequisites**

Before we dive into building the app, you'll need to have the following prerequisites:

1. **Python**: Make sure you have Python installed on your system. You can download it from the [official Python website](https://www.python.org/downloads/).
2. **Streamlit**: Install Streamlit using pip, the Python package manager:



**Getting Started**

Let's start by importing the necessary libraries and packages and loading the Machine Learning (ML) toolkit. The ML toolkit contains preprocessing components like encoders and scalers.

# ----- Load base libraries and packages

import streamlit as st

import numpy as np

import pandas as pd

import re

import os

from PIL import Image

import base64

import pickle

from sklearn.ensemble import RandomForestClassifier

# Function to load ML toolkit

def load\_ml\_toolkit(file\_path=r"streamlit\_toolkit"):

    with open(file\_path, "rb") as file:

        loaded\_toolkit = pickle.load(file)

    return loaded\_toolkit

# Importing the toolkit

loaded\_toolkit = load\_ml\_toolkit(r"streamlit\_toolkit")

encoder = loaded\_toolkit["encoder"]

scaler = loaded\_toolkit["scaler"]

# Load the saved Random Forest model from a file

with open('random\_forest\_model.pkl', 'rb') as model\_file:

    loaded\_rf\_model = pickle.load(model\_file)

In this code snippet, we import essential Python libraries, load the ML toolkit (encoder and scaler), and load a pre-trained Random Forest model for sales prediction.

**Setting up the Streamlit App**

Now, let's set up the Streamlit app and define the user interface. We'll create input fields for users to provide data and display an image to make the app visually appealing.

# Set up the Streamlit app

st.title("Sales Prediction App")

st.write("This app uses machine learning to predict sales based on certain input parameters. Enter the details below and click 'Predict' to get a sales prediction!")

# Load and display an image

image = Image.open("grocery\_shopping\_woman.png")

st.image(image, width=600)

In this section, we set the app title and provide a brief description. We also load and display an image to enhance the user experience.

**Collecting User Input**

Next, we create input fields for users to enter data. We have both categorical and numerical input fields.

# Create input fields

input\_data = {}

st.subheader("Enter the details to predict sales")

# Define categorical and numerical columns

categorical\_columns = ['products', 'state', 'store\_type', 'end\_month']

numerical\_columns = ['store\_nbr', 'onpromotion', 'cluster', 'oil\_price', 'year', 'month', 'dayofmonth', 'dayofweek']

# Create two columns for input fields

col1, col2 = st.columns(2)

# Input fields for categorical columns

with col1:

    input\_data['products'] = st.selectbox("Product Category", ['AUTOMOTIVE', 'CLEANING', 'BEAUTY', 'FOODS', 'STATIONERY', 'CELEBRATION', 'GROCERY', 'HARDWARE', 'HOME', 'LADIESWEAR', 'LAWN AND GARDEN', 'CLOTHING', 'LIQUOR,WINE,BEER', 'PET SUPPLIES'])

    input\_data['state'] = st.selectbox("State", ['Pichincha', 'Cotopaxi', 'Chimborazo', 'Imbabura', 'Santo Domingo de los Tsachilas', 'Bolivar', 'Pastaza', 'Tungurahua', 'Guayas', 'Santa Elena', 'Los Rios', 'Azuay', 'Loja', 'El Oro', 'Esmeraldas', 'Manabi'])

    input\_data['store\_type'] = st.selectbox("Store Type", ['D', 'C', 'B', 'E', 'A'])

    input\_data['end\_month'] = st.radio("Is it the end of the month?", ['True', 'False'])

# Input fields for numerical columns

with col2:

    input\_data['store\_nbr'] = st.slider("Store Number", 0, 54)

    input\_data['onpromotion'] = st.number\_input("On Promotion", step=1)

    input\_data['cluster'] = st.number\_input("Cluster", step=1)

    input\_data['oil\_price'] = st.number\_input("oil\_price", step=1)

    input\_data['year'] = st.number\_input("Year", step=1)

    input\_data['month'] = st.slider("Month", 1, 12)

    input\_data['dayofmonth'] = st.slider("Day", 1, 31)

    input\_data['dayofweek'] = st.number\_input("Day of Week (0=Sun, 1=Mon, ..., 6=Sat)", step=1)

In this section, we collect user input for various parameters such as product category, state, store type, and more. Categorical inputs are handled using select boxes and radio buttons, while numerical inputs are handled with sliders and number input fields.

**Making Predictions**

Now, it's time to use the user-provided data to make sales predictions. We apply the necessary preprocessing steps and feed the data into the pre-trained machine learning model.

# Create a button to make a prediction

if st.button("Predict", help="Click to make a prediction."):

    # Convert the input data to a pandas DataFrame

    input\_df = pd.DataFrame([input\_data])

    # Apply the imputers to categorical and numerical columns

    input\_df\_imputed\_cat = pd.DataFrame(encoder.transform(input\_df[categorical\_columns]).toarray(), columns=encoder.get\_feature\_names(categorical\_columns))

    input\_df\_imputed\_num = pd.DataFrame(scaler.transform(input\_df[numerical\_columns]), columns=numerical\_columns)

    # Make a prediction

    final\_df = pd.concat([input\_df\_imputed\_cat, input\_df\_imputed\_num], axis=1)

    prediction = loaded\_rf\_model.predict(final\_df)[0]

    # Display the prediction

    st.write(f"The predicted sales are: {prediction}.")

    # Save the input data to a CSV file

    input\_df.to\_csv("data.csv", index=False)

In this section, when the user clicks the "Predict" button, we collect the input data, apply the necessary preprocessing steps (including imputation and scaling), and make a sales prediction using the pre-trained Random Forest model. The prediction is displayed to the user, and the input data is saved to a CSV file for reference.

**Adding Custom Styling**

To enhance the presentation of the app, we can add custom CSS to style the table and create a download button for the CSV file.

# Define custom CSS for the table

css = """

table {

    background-color: #f2f2f2;

    color: #333333;

}

"""

# Set custom CSS for the table

st.write(f'<style>{css}</style>', unsafe\_allow\_html=True)

# Add a download button for the CSV file

def download\_csv():

    with open("data.csv", "r") as f:

        csv = f.read()

    b64 = base64.b64encode(csv.encode()).decode()

    button = f'<button class="download-button"><a href="data:file/csv;base64,{b64}" download="data.csv">Download Data CSV</a></button>'

    return button

st.markdown(f'<div style="text-align: center">{download\_csv()}</div>', unsafe\_allow\_html=True)

In this final section, we define custom CSS to style the table, making it visually appealing, and create a download button for the CSV file that allows users to download the input data for further analysis.

**Running the App**

To run the Streamlit app, open your command prompt or terminal, navigate to the directory where your **app.py** file is located, and execute the following command:



You will see a local web server start, and the app will open in your web browser. Users can interact with the app, input data, and receive sales predictions.

In this article, we've covered the essential steps to create a Sales Prediction Web App using Streamlit. You can further enhance the app's functionality, style, and features to meet your specific needs. Streamlit offers a wide range of customization options and extensions to make your web app even more powerful. Happy coding!

**Appreciation**

I highly recommend Azubi Africa for their comprehensive and effective programs. Read More articles about [Azubi Africa here](https://medium.com/@azubiafrica) and take a few minutes to visit this link to learn more about Azubi Africa life changing [programs](https://bit.ly/41CGCwK)

**Tags**

[Azubi Data Science](https://www.azubiafrica.org/data-analytics)

**Links**

[Jupyter Notebook](https://github.com/theeanalyst/streamlitapp/blob/main/T-Series_analysis.ipynb)

[GitHub](https://github.com/theeanalyst/streamlitapp)