**Project Documentation: The WAKFLO MLOps Pipeline**

**Prepared by:** Chidiebere Odurukwe Frances

**Date:** August 16, 2025

**Audience:** E-commerce executives and marketing teams.

**1.0 Executive Summary**

In this project, I successfully operationalized the predictive model from my Jupyter Notebook, Chidiebere\_Odurukwe\_301309006...ipynb, transforming it into a complete, end-to-end MLOps (Machine Learning Operations) pipeline named **WAKFLO**.

The core objective, as I outlined in my initial analysis, was to predict repeat customer behavior to help the e-commerce business retain one-time buyers, thereby reducing acquisition costs and boosting revenue. My initial analysis identified key drivers such as high total purchases and customer lifetime value. While the notebook's primary finding was the superior performance of an XGBoost model (ROC AUC 0.92), for this project, I implemented a Neural Network model as a practical substitute to build out the full deployment and monitoring workflow.

The final solution I built includes a governed model managed by **MLflow**, an interactive web application built with **Streamlit**, and a monitoring dashboard created in **Tableau** that meets all specified assignment requirements.

**2.0 Project Workflow Overview**

I executed the project in two parallel streams that originate from the same core dataset: **Model Governance & Deployment** and **Model Monitoring**.

1. **Governance & Deployment:** A model is trained, and its lifecycle is managed by **MLflow**. The live **WAKFLO**application I developed loads the official "production" model from MLflow to make real-time predictions.
2. **Monitoring:** I created a separate Python script to analyze the training and testing data to generate key performance and drift metrics. I then saved these metrics to CSV files, which are visualized in a **Tableau** dashboard for ongoing model health checks.

**3.0 Component Breakdown**

**3.1 Data & Initial Modeling (The "Test Kitchen")**

* **What I did:** I began the project with the ecommerce\_customer\_data.csv dataset. The initial steps, detailed in my project notebook, involved data loading, cleaning, preprocessing, and training several models to find the best performer. I then encapsulated this logic into a standalone Python script named **train\_model.py** to make the process repeatable and ready for production. A key addition I made was the creation of a **SHAP explainer** object to ensure model transparency.
* **Where I did it:** The foundational analysis was performed in my **Jupyter Notebook**. The production-ready training logic was executed from my **Terminal** using the train\_model.py script.

**3.2 Model Governance & Tracking (The "Chef's Logbook")**

* **What I did:** I implemented a robust system to manage the model's lifecycle from experiment to production.
  1. **Tracking:** Every time the train\_model.py script runs, it records the model's parameters and performance metrics (accuracy, ROC AUC).
  2. **Versioning & Registration:** I saved the trained model and its SHAP explainer ("registered" them) in a central repository. Each new training run creates a new version.
  3. **Governance:** I assigned an **Alias** named production to the specific model version approved for use. This is the core of my governance strategy, replacing the deprecated "Stages" feature.
* **Where I did it:**
  1. The **MLflow Server** was run from my **first Terminal window** using a robust command to create a stable database for it.
  2. Model logging and registration were performed by the **train\_model.py script**.
  3. I assigned the production alias manually in the **MLflow Web UI**.

**3.3 Interactive Application (The "Restaurant Menu")**

* **What I did:** I built the user-friendly **"WAKFLO"** web application for real-time predictions.
  1. **UI Development:** I created a full suite of sliders and dropdowns for all 20 of the model's input features and branded the app with the WAKFLO logo and title.
  2. **Model Loading:** The app automatically connects to the MLflow server and requests the model with the production alias, ensuring it's always using the correct version.
  3. **Scenario Testing:** I added buttons to the sidebar to pre-fill the form with data for a "**High-Potential New Customer**" and an "**At-Risk Loyal Customer**."
  4. **Model Explainability:** After making a prediction, the app displays a **SHAP waterfall plot** to explain *why*the model made its decision, providing crucial transparency.
* **Where I did it:** All logic is contained in the **app.py** file. I launched the application from my **second Terminal window** using the command streamlit run app.py.

**3.4 Monitoring Dashboard (The "Health Inspector's Report")**

* **What I did:** To fulfill the assignment requirements, I created a comprehensive monitoring dashboard.
  1. **Data Generation:** I wrote the **generate\_monitoring\_data.py** script to analyze the "build" and "holdout" datasets and calculate all required statistics, including PSI for model drift and CSI for variable drift, along with their associated risk tiers.
  2. **Data Export:** I saved these statistics into model\_monitoring\_summary.csv and variable\_monitoring\_stats.csv.
  3. **Dashboard Creation:** I connected **Tableau** to these CSV files and built a three-part dashboard showing:
     + A high-level **Model Health Summary**.
     + A color-coded bar chart for **Variable Drift (CSI)**.
     + A detailed table of **Build Statistics** for each feature, including caps, floors, and imputation values.
* **Where I did it:**
  1. I generated the statistics from my **second Terminal window** on my MAC.
  2. I built and viewed the final dashboard in **Tableau Desktop**.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.