# Hotel Booking Demand Prediction using MLOps
# Progress Report

Sampreeth Jangala Keshav Naram Pranav Srinivasan

April 9, 2025

# Table of contents

# 1 Problem Statement

Deploying and managing machine learning models in production environments presents significant challenges, including scalability, reproducibility, automation, and continuous monitoring. Traditional ML workflows often lack proper integration with deployment pipelines, leading to inefficiencies in model retraining, tracking, and deployment. Our project aims to bridge this gap by developing a fully automated, containerized MLOps pipeline that enables seamless model deployment in the cloud while ensuring scalability, reliability, and maintainability.

# 2 Dataset Description

The project utilizes the **Hotel Booking Demand** dataset from Kaggle, which contains comprehensive details on hotel reservations, cancellations, and booking behavior. This dataset is valuable for predictive modeling tasks such as demand forecasting, cancellation prediction, and revenue optimization.

## 2.1 Dataset Overview

The dataset consists of **119,390** rows and **32** columns, covering:

- **Booking Details:** Arrival dates, lead time, and stay duration.
- **Customer Information:** Guest demographics, country of origin, and repeat customer status.
- **Market Segmentation:** Booking sources and distribution channels.
- **Financial Data:** Pricing details, deposit types, and special requests.
- **Booking Modifications:** Cancellations, changes, and waiting list information.

## 2.2 Feature Classification

The dataset features three main categories:

- **Target Variable:** Whether a booking was canceled (*is_canceled*).
- **Categorical Features:** Hotel type, booking source, meal plans, customer type, reservation status, etc.
- **Numerical Features:** Timing details, stay duration, guest count, pricing, booking history, and additional service requests.

This structured dataset provides a strong foundation for machine learning models focused on hotel booking analytics.

# 3 High-Level Approach and Novelty

Our approach involves building an end-to-end MLOps pipeline that automates data ingestion, preprocessing, feature engineering, model selection, training, and deployment. The key highlights of our approach include:

- **Automation of ML Workflow:** Leveraging CI/CD pipelines for continuous integration and automated deployment.

- **Containerization and Cloud Deployment:** Ensuring reproducibility and scalability using Docker and AWS.
- **Comprehensive Monitoring and Logging:** Utilizing MLflow and TensorBoard for model tracking and evaluation.
- **Security and Code Quality Assurance:** Implementing linting, security scanning, and structured exception handling using Pylint, Flake8, Black, and Trivy.

# 4 Implementation Plan

## 4.1 Environment Setup

Setting up the project environment with Python, dependency management, logging mechanisms, and structured exception handling. This step ensures robust error tracking, debugging, and modularity for future enhancements.

### 4.1.1 Database Setup (MySQL)

Configuring and managing the relational database using **MySQL** to store structured data. This step involves defining schemas, setting up database connections, and ensuring efficient data retrieval for downstream processing.

## 4.2 Data Ingestion

Automating the collection and ingestion of raw data from various sources using **Pandas** and **NumPy**. The data is structured, stored, and made available for subsequent processing steps in a format that facilitates seamless integration with the pipeline.

## 4.3 Data Processing

Cleaning and preprocessing the raw data by handling missing values, detecting and treating outliers, and removing unnecessary columns using Scikit-learn and Pandas. The processed data is then stored in a standardized format to ensure consistency across the pipeline.

## 4.4 Feature Engineering

Enhancing the dataset by constructing new features, applying binning techniques, performing label encoding, and selecting the most relevant features using **Scikit-learn**. This step ensures that the model receives the most informative inputs to improve performance.

## 4.5 Model Selection

Experimenting with multiple machine learning models using **PyTorch** to determine the most effective one. The process includes loading data, splitting a small portion for logging insights, training models, and evaluating them using standard performance metrics such as accuracy, precision, recall, and F1-score. The results are logged using MLflow for comparison and future analysis.

### 4.6 Model Training

Selecting the best-performing model from the previous step and tuning hyperparameters using **GridSearchCV**. The trained model is evaluated, stored using **Joblib**, and logged using **MLflow** for version control and reproducibility.

### 4.7 Training Pipeline (DVC)

Integrating Data Version Control (DVC) to manage different stages of the ML pipeline, including data ingestion, processing, feature engineering, and model training. This ensures traceability and enables seamless collaboration among team members.

### 4.8 User Interface Development

Developing a web-based interface using Flask or Django, combined with HTML and CSS, to allow users to interact with the trained model. The UI will enable users to submit inputs, receive predictions, and visualize key insights related to model performance.

### 4.9 CI/CD and Deployment using Jenkins

Implementing a CI/CD pipeline using Jenkins to automate code integration, testing, and deployment. This step includes:

- Setting up Jenkins in a containerized environment.
- Integrating with GitHub for continuous version control.
- Running code quality checks using **Pylint, Flake8, Black**.
- Performing security scans for vulnerabilities using **Trivy**.
- Building and containerizing the application using Docker.
- Pushing the container image to DockerHub and deploying it to AWS using **ECS** and **CLI** commands.

## 5 Implementation Tools

- **Programming Language:** Python
- **Libraries & Frameworks:** Pandas, NumPy, Scikit-learn, TensorFlow/PyTorch, DVC, MLflow, Flask/Django
- **Database:** MySQL
- **Version Control & CI/CD:** GitHub, Jenkins
- **Containerization & Cloud Deployment:** Docker, AWS ECS, AWS CLI
- **Monitoring & Logging:** MLflow, TensorBoard
- **Security & Code Quality:** Pylint, Flake8, Black, Trivy

# 6 Individual Planned contribution

## 6.1 Sampreeth

- Initial Setup: Set up the environment, implement logging, and handle custom exceptions.
- Data Processing: Clean and preprocess data, handle outliers, missing values, and unnecessary columns, and save processed data.
- Model Training: Train the selected model using GridSearchCV, evaluate performance using metrics, save the model, and log results.

## 6.2 Keshav

- Database Setup (MySQL): Configure the database and extract relevant data for the pipeline.
- Feature Engineering: Perform feature construction, binning, label encoding, and feature selection.
- Training Pipeline (DVC): Integrate the entire data processing and model training pipeline using DVC.

## 6.3 Pranav

- Data Ingestion: Automate data ingestion and ensure proper data integrity.
- Model Selection: Split the data, train various models, evaluate them, and log performance metrics.
- User App Building: Develop the user-facing application with frontend design and backend functionality.

## 6.4 Deployment Split

- Sampreeth: Set up Jenkins container, integrate with GitHub, and implement code linting.
- Keshav: Perform file system security scan, build Docker image, and push to DockerHub.
- Pranav: Deploy the Docker image to AWS using ECS or CLI.

# 7 Progress Update

## 7.1 Design & Architecture

- **Modular Workflow:** The project follows a modular machine learning pipeline with clear stages, including data loading, cleaning, feature engineering, model training, and evaluation. This ensures that each phase can be handled efficiently and reused for future tasks.

## 7.2 Development

### 7.2.1 Data Ingestion

- The dataset is loaded from Kaggle and contains booking information for customers, including features such as customer type, market segment, hotel type, lead time, and booking status.
- We performed a preliminary check for missing values and handled data inconsistencies to ensure that the dataset is ready for further processing.

### 7.2.2 Data Processing

- Missing values were handled using a combination of techniques such as imputation and removal of rows with excessive missing data.
- Irrelevant features were dropped, and categorical features like `hotel_type`, `meal`, and `distribution_channel` were encoded using label encoding.
- Numerical features were scaled using `StandardScaler` to ensure that all features were on the same scale, which improves the performance of most machine learning models.

| Column Name | Null Values | Percentage Null Values (%) |
|---|---|---|
| country | 488 | 0.408744 |
| agent | 16340 | 13.686238 |
| company | 112593 | 94.306893 |

Table 1: Null Values and Their Percentage per column of features with null values

### 7.2.3 Exploratory Data Analysis (EDA)

- The EDA revealed the following insights:
  - `Lead Time` showed a right-skewed distribution, indicating that most customers book with short lead times.
  - `Is Canceled` is imbalanced, with cancellations being a minority class, which may affect model training.
  - There is a significant correlation between features like `number_of_adults` and `is_canceled`.
  - Hotel types and market segments vary significantly in terms of booking frequency, which was visualized using bar plots.

### 7.2.4 Visualized Plots

We explored the dataset using visualizations to extract key patterns and relationships between features and the target variable. Below are some of the significant plots generated during EDA:

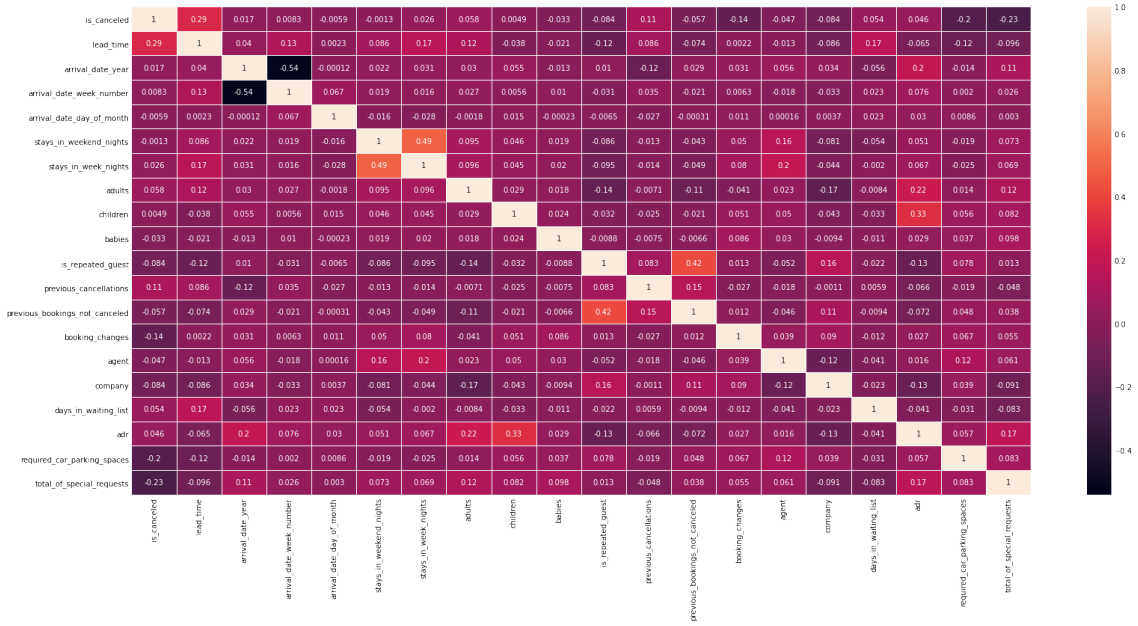
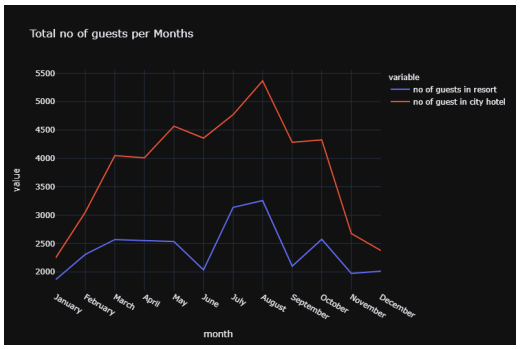
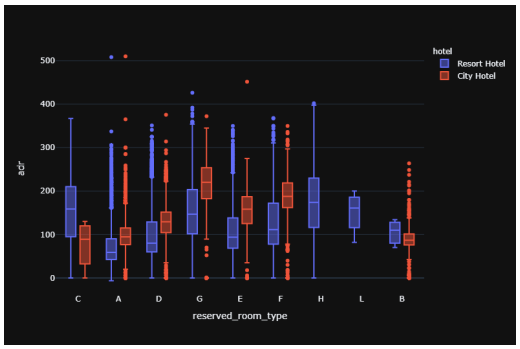


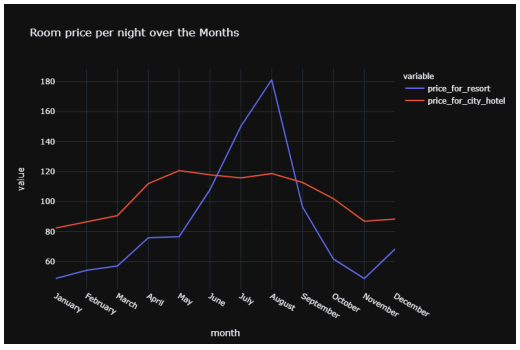

Figure 1: Correlation Heatmap of Features



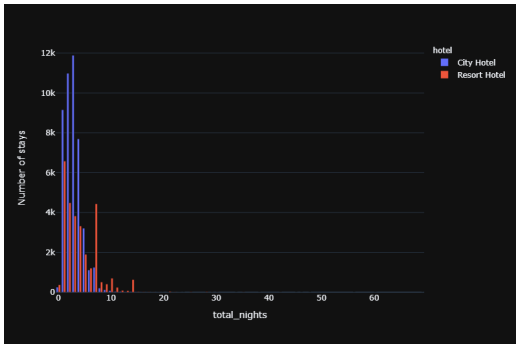

(a) Total Number of Guests Per Month




(b) Hotel Type Distribution




(c) Room Price Per Night Over the Months




(d) Number of Stays vs. Total Nights

Figure 2: Visual Analysis of Guest Trends and Booking Patterns

### 7.2.5 Feature Engineering

To improve the predictive power of our models, we performed various feature engineering techniques, which included the creation of new features and the transformation of existing ones:

- **Stay Duration:** We computed the total number of nights for each booking by summing the values of `stays_in_weekend_nights` and `stays_in_week_nights`.
- **Lead Time Categories:** The `lead_time` feature was discretized into categorical bins such as:
  - Short-term (0–30 days)
  - Medium-term (31–90 days)
  - Long-term (90+ days)
- **Encoding Categorical Features:** All non-numeric categorical features such as `hotel`, `meal`, and `customer_type` were transformed using label encoding or one-hot encoding, depending on the algorithm used.

#### 7.2.6 Model Training & Evaluation

- We trained several machine learning models, including:

| Model Name | Accuracy (%) | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 81.2 | 0.82 | 0.81 | 0.80 |
| K-Nearest Neighbors (KNN) | 89.2 | 0.90 | 0.89 | 0.89 |
| Decision Tree Classifier | 94.8 | 0.95 | 0.95 | 0.95 |
| Random Forest Classifier | 95.2 | 0.95 | 0.95 | 0.95 |
| AdaBoost Classifier | 94.6 | 0.95 | 0.95 | 0.95 |
| Gradient Boosting Classifier | 91.2 | 0.92 | 0.91 | 0.91 |
| XGBoost | 98.2 | 0.98 | 0.98 | 0.98 |
| **CatBoost Classifier** | **99.5** | **1** | **1** | **1** |
| Extra Trees Classifier | 95.2 | 0.95 | 0.95 | 0.95 |
| LightGBM (LGBM) Classifier | 96.0 | 0.96 | 0.96 | 0.96 |
| ANN (Artificial Neural Network) | 99.3 | 0.91 | 0.89 | 0.90 |
| Voting Classifier (Ensemble) | 96.4 | 0.97 | 0.96 | 0.96 |

Table 2: Model Evaluation Results

- The models were evaluated using accuracy, precision, recall, and F1-score. The CatBoost Classifier model performed the best, with an accuracy of `99.5%`, which was superior to the other models.

### 7.3 Challenges Encountered

- **Overfitting:** Despite good performance, overfitting was observed in some models. This was addressed by applying regularization techniques and tuning model complexity.
- **Class Imbalance:** The `is_canceled` feature was imbalanced, which affected model performance on the minority class. Techniques like SMOTE and adjusting class weights were explored to address this.

### 7.4 Next Steps

#### 7.4.1 CI/CD Pipeline Completion

Finalize and test the Jenkins-based CI/CD workflow for automatic:

- Code linting and security scanning
- Docker image builds and pushes to DockerHub
- Model deployment to AWS ECS using CLI integration

#### 7.4.2 Containerization and Deployment

- Ensure the model, API backend, and web app are fully containerized using Docker.
- Deploy the full stack to AWS using ECS Fargate for scalability and low maintenance.
- Set up environment variables and secrets handling securely for production deployment.

#### 7.4.3 User Interface Integration

- Connect the backend API to the frontend UI for real-time prediction capability.
- Improve the frontend to include user-friendly inputs, clear output displays, and status indicators (e.g., "Model running", "Prediction ready").

#### 7.4.4 Monitoring, Logging, and Scaling

- Integrate MLflow and TensorBoard to monitor prediction performance and usage logs.
- Set up cloud logging (e.g., CloudWatch) and alerts for failures or spikes in usage.
- Plan for horizontal scaling or load testing post-deployment.

#### 7.4.5 Documentation and Final Report

- Prepare deployment diagrams, API usage documentation, and setup guides.
- Summarize lessons learned and prepare content for the final demo.