Architecture Document Design

Backorder Prediction

Table Of Contents

1. Introduction:

1.1. What is the purpose of Architecture design document?

The purpose of this document is to outline the architecture of the Backorder Prediction System, which predicts whether a product will go on backorder based on historical data. The document provides a high-level and low-level architecture overview, including system components, data flow, and deployment details.

1.1.1 Scope:

This system uses a Machine Learning model deployed via FastAPI (backend) and a Streamlit web application (frontend). Both components are hosted on an Azure Linux VM. The system processes user inputs, makes predictions, and provides insights for inventory management.

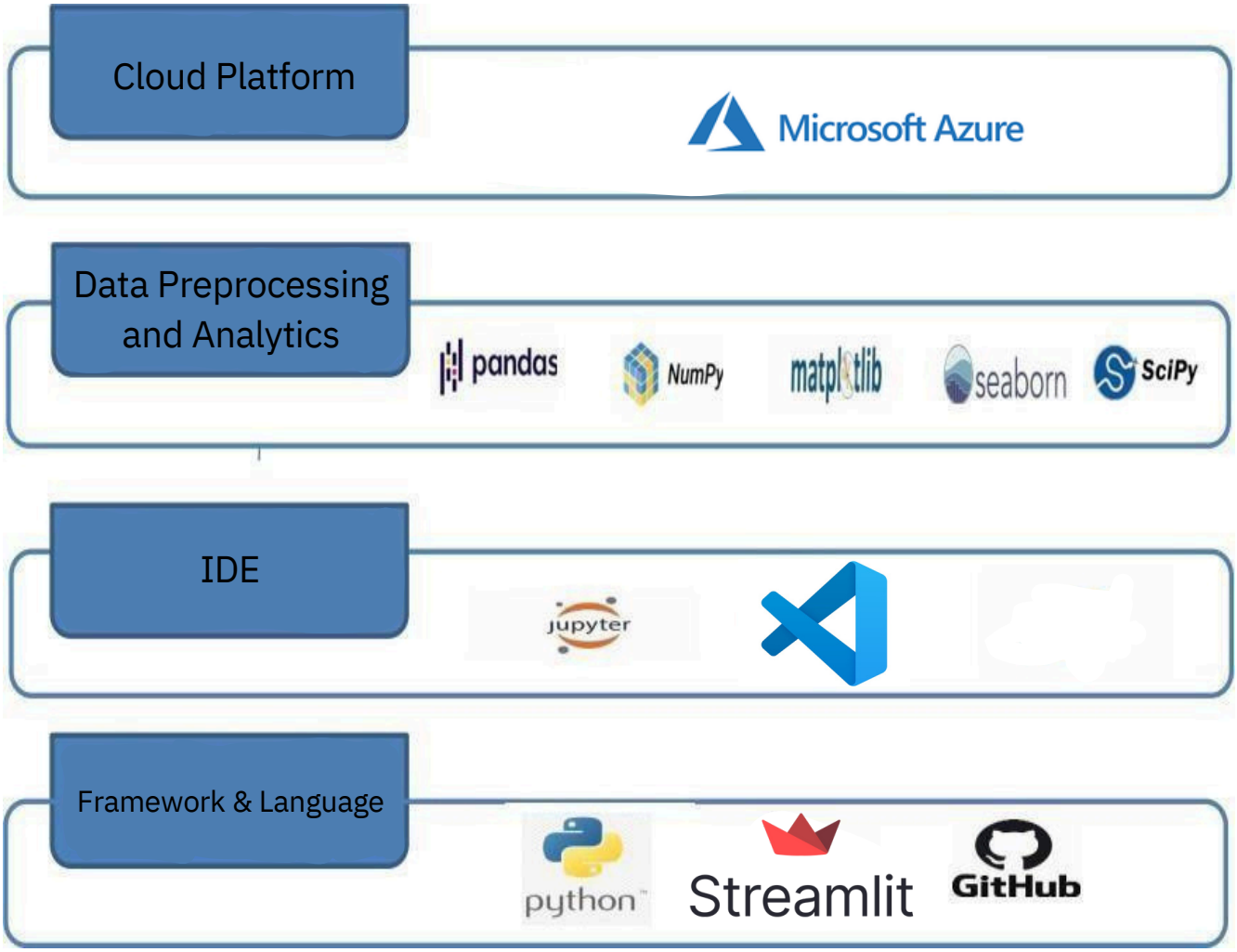
2. System Overview:

The Backorder Prediction System is designed to classify products as either going on backorder (Yes/No) based on various inventory and sales-related features.

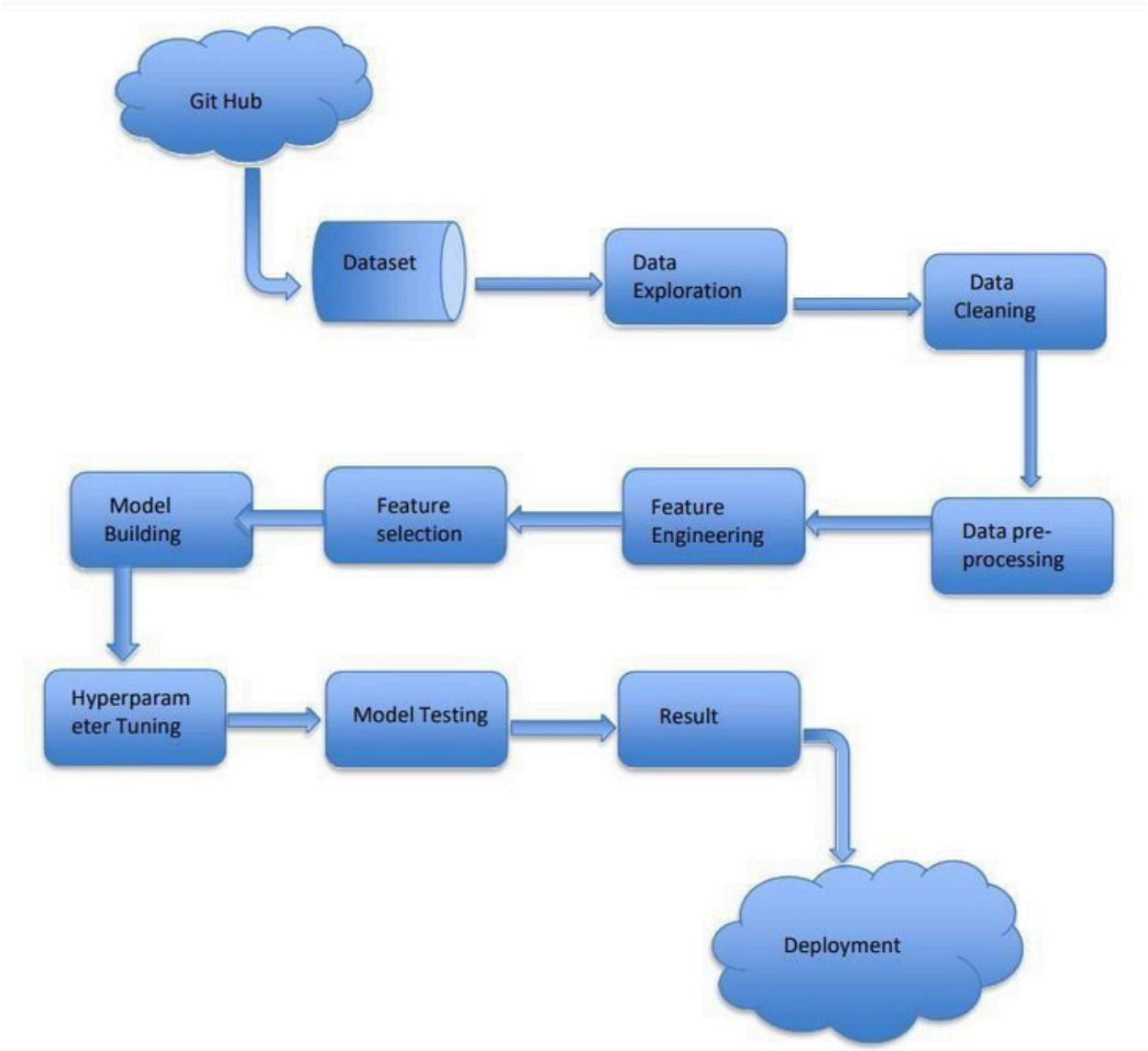
2.1 Key Features

- User-friendly Streamlit Web Application for input and prediction
- FastAPI Backend for model inference
- Machine Learning Classification Model to predict backorders
- Azure Linux VM Deployment for hosting API and UI
- Model Monitoring for performance tracking

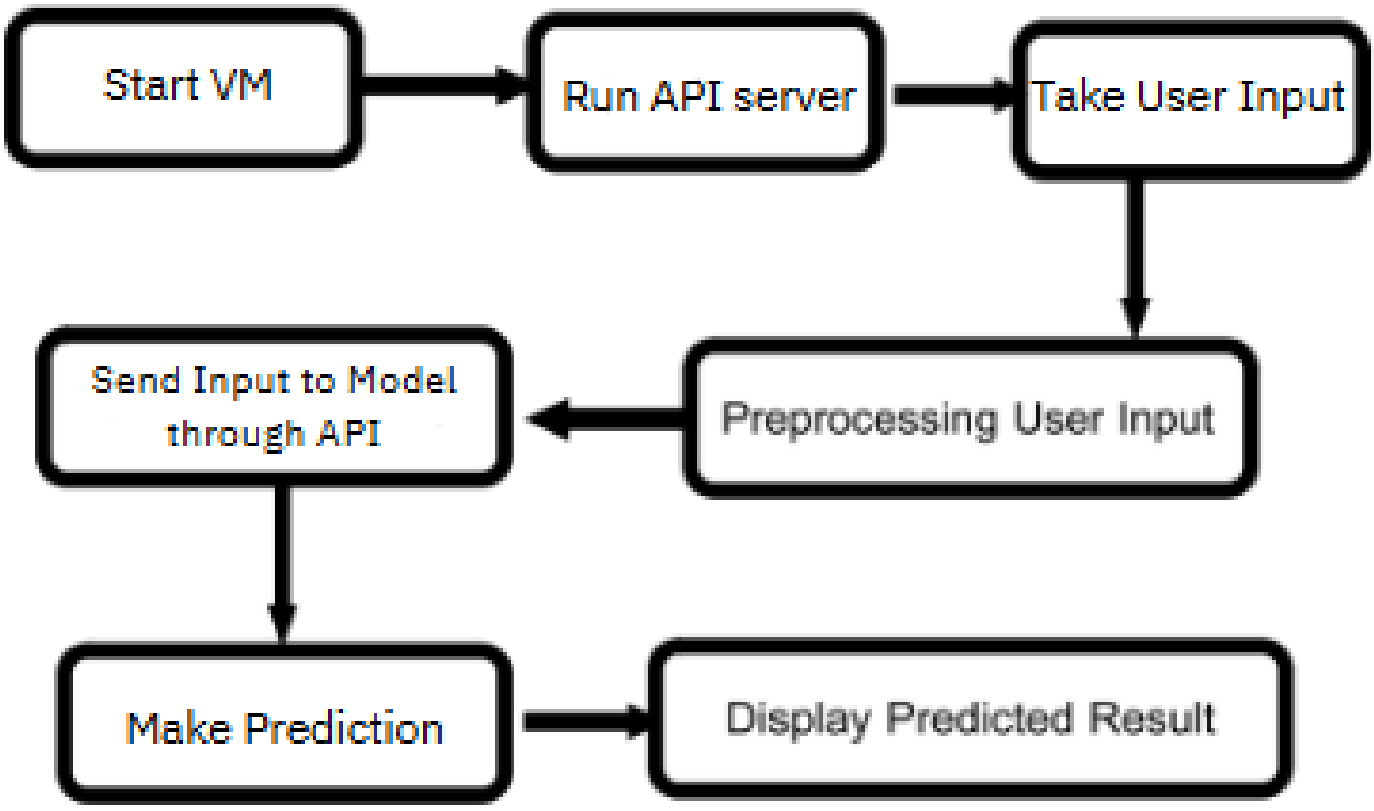
2.2 Technology Stack :



3. Architecture:



3.2 Prediction Process



4. Architecture Description:

4.1. Data Description:

Data Source:
https://github.com/rodrigosantis1/backorder_prediction/blob/master/dataset.rar
The dataset is highly imbalanced which should be addressed for accurate predictions by the model; each dataset contains 23 attributes with 19,29,937 observations after combining both training and testing sets, respectively.
The system ingests structured historical data related to inventory, supply chain, and sales performance. Data is sourced from ERP systems, containing inventory levels, lead times, past sales, forecasted demand, supplier performance, and risk indicators. The dataset is loaded into the system using Pandas, NumPy, and other relevant data handling libraries.
The dataset is split into training and testing subsets to ensure model generalization.
Stratified K-Fold Cross-Validation is applied to handle class imbalance and prevent overfitting. The split ratio ensures the distribution of the minority class (backorders) is preserved in each fold.

4.2. Exploratory Data Analysis:

Exploratory Data Analysis, or EDA, is an important step in any Data Analysis or Data Science project. EDA is the process of investigating the dataset to discover patterns, and anomalies (outliers), have a look over the distributions of the features, find correlations between them and form hypotheses based on our understanding of the dataset. Statistical and graphical analyses are performed to understand:

- Data distributions
- Correlations between features
- Presence of missing values and outliers
- Trends in sales, inventory levels, and supply chain performance
- Key insights drive feature engineering and model selection strategies.

4.3. Data Cleaning:

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

4.4. Data Pre-processing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always the case that we come across clean and formatted data.
Handling missing values using Imputer techniques (mean, median, mode, or predictive imputation).Categorical encoding for non-numeric variables using techniques like One-Hot Encoding or Label Encoding.Feature scaling & normalization (Min-Max Scaling, Standardization) for numerical features to improve model convergence and performance.

4.5. Feature Engineering:

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.

New feature creation to capture important trends in sales, inventory levels, and supply chain performance (e.g., demand-to-supply ratio, supplier reliability score). Multicollinearity analysis to remove redundant features.

4.6. Feature Selection:

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and is computationally intractable for all but the smallest of feature sets.

4.7. Model Building:

A machine learning model is built by learning and generalizing from training data, then applying that acquired knowledge to new data it has never seen before to make predictions and fulfil its purpose. Lack of data will prevent you from building the model, and access to data isn't enough. Multiple classification algorithms are trained to predict whether a product will go on backorder or not like Logistic Regression, SVM, Random Forest, XGBoost, LightGBM, Gradient-Boosting, Balanced Random Forest, Easy Ensemble Classifier. Handling class imbalance using Random Over-Sampling, Random Under-Sampling, and SMOTE (Synthetic Minority Over-Sampling Technique). Model validation using Stratified K-Fold Cross-Validation.

4.8. Hyperparameter Tuning & Optimization:

Grid Search Optimization are used to tune model hyperparameters for better performance.
Regularization techniques are applied to prevent overfitting and improve generalization.
The parameters for the best grid search model for each classification algorithm are used for fitting and prediction purposes.

4.9. Model Validation:

For machine learning systems, we should be running model evaluation and model tests in parallel. Model evaluation covers metrics and plots which summarise performance on a validation or test dataset. Models are evaluated based on classification metrics such as: Precision, Recall, PR-AUC, AUC-ROC, and F1-score.
Recall is prioritized to minimize the risk of missing potential backorders. The best-performing model is selected based on evaluation results.

4.10. Deployment:

The trained model is integrated and deployed using a combination of FastAPI, Streamlit, and Azure, ensuring scalability, accessibility, and real-time inference. The deployment architecture consists of multiple components working together:

4.10.1 FastAPI Backend:-

- Purpose: Acts as a RESTful API, serving model predictions to external applications.
- Functionality:

1. Receives HTTP POST requests with product features as input.
2. Preprocesses input data (scaling, encoding, transformation).
3. Passes the processed data to the trained model for prediction.
4. Returns the prediction in JSON format.

4.10.2 Streamlit Web UI:-

- Purpose: Provides an intuitive Graphical User Interface (GUI) for users to interact with the prediction system.
- Functionality:

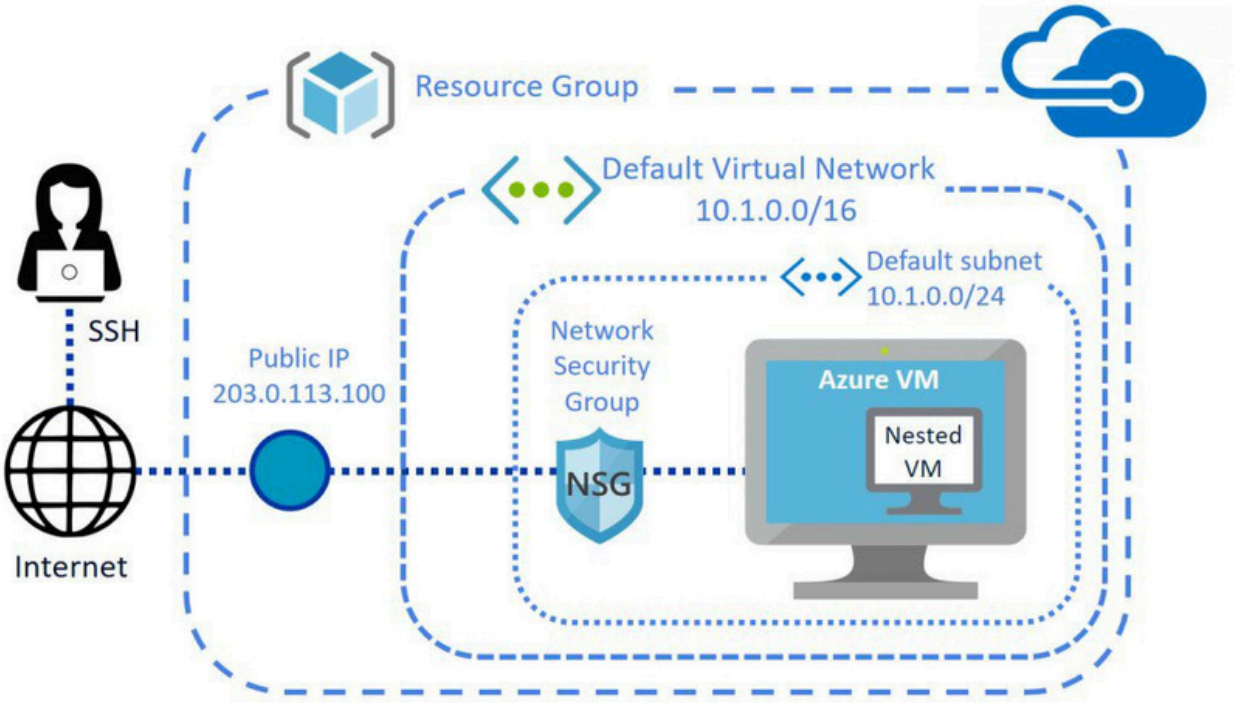
1. Allows users to input inventory details, lead time, supplier performance, and other product features.
2. Sends the input data to the FastAPI backend via API requests.
3. Displays real-time predictions with visual indicators (Backorder: Yes/No).

4.10.3 Azure Linux VM Deployment:-

- Purpose: Hosts both FastAPI (backend) and Streamlit (frontend) applications on a cloud-based infrastructure for real-time accessibility.
- Deployment Steps:

1. Setup Azure Linux VM to serve as the hosting environment.
2. Install dependencies including Python, FastAPI, Streamlit, and required libraries.
3. Run FastAPI API as a background service, exposing an endpoint for predictions.
4. Deploy Streamlit UI, linking it to FastAPI for fetching predictions.
5. Enable firewall rules and networking configurations to allow external access.

5. Deployment Architecture:



- Azure Linux VM: Hosts both the FastAPI backend and Streamlit frontend.
- Public IP Address: Allows users to access the application via a web browser.
- API & UI Deployment: Both services are run using separate background processes.