

# E-COMMERCE ANALYTICS PLATFORM

Technical Documentation



**Project By:** Hemant Borana  
**Date:** December 2025



## **Technical Documentation**

### **E-Commerce Analytics Platform**

## **Contents**

1. System Architecture .....	3
2. Data Pipeline.....	4
3. Analytics Implementation.....	6
Part B: Diagnostic Analytics .....	7
Part C: Predictive Analytics.....	8
Part D: Prescriptive Analytics.....	9
4. Model Validation .....	10
5. Performance Optimization.....	11
6. Deployment Guide .....	12
7. Testing Procedures .....	13
8. Future Enhancements .....	14
9. Troubleshooting .....	15

# 1. System Architecture

## Overview

The analytics platform is built using a modular Python-based architecture with Streamlit for the front-end interface and SQLite for data storage.

## Technology Stack:

- **Backend:** Python 3.x
- **Database:** SQLite
- **Web Framework:** Streamlit
- **Data Processing:** pandas, numpy
- **Visualization:** plotly
- **Machine Learning:** scikit-learn, statsmodels
- **Deployment:** Local/Cloud-ready

## 2. Data Pipeline

### Data Generation

Synthetic e-commerce data generated with realistic patterns:

- 15,000 transactions over 24 months
- 2,000 unique customers
- Multi-dimensional attributes (region, segment, category, channel)
- Time-series data with seasonality and trends

### Data Schema

#### Transactions Table:

- transaction\_id (Primary Key)
- customer\_id (Foreign Key)
- date (DateTime)
- revenue (Float)
- category (String)
- channel (String)
- payment\_method (String)
- delivery\_days (Integer)
- satisfaction\_score (Integer 1-5)
- segment (String)
- region (String)
- age\_group (String)

#### Customers Table:

- customer\_id (Primary Key)
- segment (String: Premium/Regular/Budget)
- region (String: North/South/East/West)
- acquisition\_date (DateTime)
- age\_group (String)
- last\_purchase\_date (DateTime)
- days\_since\_purchase (Integer)
- is\_churned (Boolean)

## **Data Preprocessing**

- Date conversion to DateTime objects
- Missing value handling (none in synthetic data)
- Feature engineering for predictive models
- Encoding categorical variables for ML models

### 3. Analytics Implementation

#### Part A: Descriptive Analytics (KPI Dashboard)

##### Key Metrics:

- Total Revenue
- Total Orders
- Unique Customers
- Average Order Value (AOV)
- Average Satisfaction Score

##### Implementation:

```
def get_current_kpis(df):  
    total_revenue = df['revenue'].sum()  
    total_orders = len(df)  
    total_customers = df['customer_id'].nunique()  
    avg_order_value = total_revenue / total_orders  
    avg_satisfaction = df['satisfaction_score'].mean()  
  
    return {...}
```

##### Features:

- Real-time filtering by date, region, segment
- Interactive visualizations with Plotly
- Alert system with configurable thresholds
- Multi-dimensional trend analysis

##### Performance:

- Dashboard load time: <2 seconds
- Filter response time: <1 second
- Meets requirement: <3 seconds

## **Part B: Diagnostic Analytics**

### **Drill-Down Capabilities:**

1. Geographic: Region → Regional segments
2. Time-Based: Year → Quarter → Month
3. Product: Category → Channel → Segment
4. Customer: Segment → Demographics

### **Root Cause Analysis Tools:**

- Revenue decline investigation
- Satisfaction score analysis
- Delivery time correlation
- Churn factor identification

### **Statistical Methods:**

- Correlation analysis
- Period-over-period comparison
- Anomaly detection via threshold monitoring

# Part C: Predictive Analytics

## 1. Revenue Forecasting

- **Model:** ARIMA(5,1,2)
- **Training data:** 80% (584 days)
- **Test data:** 20% (146 days)
- **Performance:** MAE < 15%, Accuracy > 80%

```
model = ARIMA(train_data['revenue'], order=(5, 1, 2))
```

```
fitted_model = model.fit()
```

```
forecast = fitted_model.forecast(steps=forecast_days)
```

## 2. Churn Prediction

- **Model:** Random Forest Classifier
- **Features:** 7 (revenue metrics, satisfaction, delivery, segment, region)
- **Accuracy:** ~75-85%
- **Output:** Probability score (0-1)

```
model = RandomForestClassifier(n_estimators=100, max_depth=10)
```

```
model.fit(X_train, y_train)
```

## 3. Customer Lifetime Value (CLV)

- **Model:** Random Forest Regressor
- **Features:** 6 (order count, satisfaction, AOV, demographics)
- **Performance:** MAE < 20%, RMSE tracking

# **Part D: Prescriptive Analytics**

## **Optimization Engines:**

### **1. Customer Retention:**

- Input: Churn probability, customer value
- Output: Prioritized action list with ROI
- Logic: Risk-based segmentation with tailored interventions

### **2. Revenue Optimization:**

- Pricing recommendations based on satisfaction
- Channel budget allocation by performance
- Cross-sell identification

### **3. Resource Planning:**

- Demand forecasting by category
- Delivery network optimization
- Staffing recommendations by day

### **4. ROI Calculator:**

- Multiple scenario analysis
- Payback period calculation
- Net benefit quantification

## 4. Model Validation

### Revenue Forecasting

- **Validation method:** Train-test split (80-20)
- **Metrics:** MAE, RMSE, Accuracy
- **Confidence intervals:** Included in forecast visualization

### Churn Prediction

- **Validation method:** Cross-validation
- **Metrics:** Accuracy, Precision, Recall, F1-Score
- **Feature importance:** Displayed in interface

### CLV Prediction

- **Validation method:** Train-test split
- **Metrics:** MAE, RMSE,  $R^2$  score
- **Scatter plot:** Actual vs Predicted visualization

## 5. Performance Optimization

### Database Indexing:

```
CREATE INDEX idx_trans_date ON transactions(date)
```

```
CREATE INDEX idx_trans_cust ON transactions(customer_id)
```

```
CREATE INDEX idx_cust_id ON customers(customer_id)
```

### Caching Strategy:

- Streamlit @st.cache\_data for data loading
- Model predictions cached per session
- Reduces repeated computations

### Query Optimization:

- Aggregations performed in pandas
- Minimal database reads
- Efficient groupby operations

## 6. Deployment Guide

### Requirements:

python >= 3.8

streamlit >= 1.28.0

pandas >= 1.5.0

numpy >= 1.23.0

plotly >= 5.14.0

scikit-learn >= 1.2.0

statsmodels >= 0.14.0

### Installation:

pip install -r requirements.txt

python data\_generation.py

python db\_setup.py

streamlit run app.py

### Access:

- Local: <http://localhost:8501>
- Network: Configure Streamlit config.toml

## 7. Testing Procedures

### **Unit Tests:**

- Data loading functions
- KPI calculation accuracy
- Model prediction consistency

### **Integration Tests:**

- End-to-end workflow validation
- Filter functionality across pages
- Model training and prediction pipeline

### **Performance Tests:**

- Load time measurements
- Concurrent user simulation
- Database query performance

## 8. Future Enhancements

### Planned Features:

- Real-time data integration
- Advanced NLP for insight generation
- Deep learning models (LSTM for forecasting)
- Multi-user authentication
- Export functionality (PDF reports)
- Email alert system
- Mobile responsive design

### Scalability Considerations:

- Migrate to PostgreSQL for production
- Implement caching layer (Redis)
- Containerization with Docker
- Cloud deployment (AWS/Azure/GCP)

# **9. Troubleshooting**

## **Common Issues:**

- 1. Module not found errors:**
  - Solution: pip install missing package
- 2. Database locked:**
  - Solution: Close other connections, restart app
- 3. Model training slow:**
  - Solution: Reduce n\_estimators or max\_depth
- 4. Memory issues:**
  - Solution: Implement data pagination, reduce cache size

## **Support Contact:**

- Technical issues: Review error logs
- Feature requests: Document in backlog

# **THANK YOU**

FOR YOUR INTEREST

## **CONTACT US**



hemantpb123@gmail.com



9284494154

