StorePulse

User Manual

Version 1.0.0

Comprehensive Guide

Know tomorrow's visits. Act today.

Welcome to StorePulse

StorePulse is an advanced retail analytics platform designed to predict customer footfall and optimize store operations. By leveraging historical data and machine learning, StorePulse provides actionable insights that help store managers plan effectively and make data-driven decisions.

TIP: This manual is designed to get you from zero to expert. Start with the Quick Start Guide if you're new, or jump to specific sections using the headings below.

Quick Start Guide

System Requirements

Before installing StorePulse, ensure your system meets these requirements:

- • **Python**: Version 3.11 or higher
- • **Node.js**: Version 18 or higher
- • **Database**: PostgreSQL 13+ (recommended) or SQLite for development
- • **RAM**: Minimum 8GB, 16GB recommended for production
- • **Storage**: At least 10GB free space
- • **Browser**: Chrome 90+, Firefox 88+, Safari 14+, or Edge 90+

Installation

Step 1: Clone the Repository

```
git clone https://github.com/nonshenz007/StorePulse.git
cd StorePulse
```

Step 2: Quick Start with Automated Script

For the fastest setup, use our automated start script:

```
chmod +x start.sh
./start.sh
```

The script automatically:

- • Sets up Python virtual environment
- • Installs backend dependencies
- • Configures the database
- • Installs frontend dependencies
- • Starts both servers

NOTE: The first run may take 5-10 minutes depending on your internet connection.

Step 3: Access the Application

Once installation completes, open your browser and navigate to:

```
http://localhost:3000
```

You should see the StorePulse login page. Use the default credentials:

- • **Username**: `admin`
- • **Password**: `admin123`

WARNING: Change the default password immediately in production environments!

Understanding the Dashboard

The StorePulse dashboard is your command center for retail analytics. Here's what each section does:

Overview Metrics

The top row displays key performance indicators (KPIs) at a glance:

- • **Today's Visits**: Real-time count of customers who have entered the store
- • **Predicted Tomorrow**: AI-powered forecast for the next day
- • **Weekly Average**: Rolling 7-day average of daily visits
- • **Trend Indicator**: Shows whether traffic is trending up or down

TIP: Hover over any metric to see historical context and percentage changes.

Live Footfall Graph

The main graph shows:

- • **Blue Line**: Actual historical visits
- • **Green Line**: Predicted future visits
- • **Shaded Area**: Confidence interval (±15% typical range)
- • **Red Markers**: Days with significant anomalies

Reading the Graph

- • **Zoom**: Click and drag to zoom into specific time periods
- • **Pan**: Hold Shift and drag to move left/right
- • **Details**: Click any point to see exact numbers
- • **Export**: Click the download icon to export data as CSV

Quick Actions Panel

Located on the right side, this panel provides instant access to:

- • **Generate Report**: Create PDF reports for management
- • **Update Data**: Manually refresh predictions with new data
- • **Export Forecast**: Download predictions for Excel
- • **Configure Alerts**: Set up notifications for unusual patterns

---

Predictive Analytics Engine

The INGARCH Model

StorePulse uses a proprietary **Integer-Valued Generalized Autoregressive Conditional Heteroskedasticity (INGARCH)** model designed specifically for retail footfall prediction.

Why INGARCH?

Unlike traditional time-series models, INGARCH is ideal for count data (like store visits) because:

- • **Native Count Handling**: Works with whole numbers, not continuous values
- • **Volatility Modeling**: Captures busy and quiet periods accurately
- • **Seasonal Patterns**: Automatically detects weekly and monthly trends
- • **External Factors**: Incorporates weather, holidays, and events

Prediction Horizons

You can forecast across multiple time ranges:

- • **7-Day Forecast**: Best for weekly staff scheduling
- • **14-Day Forecast**: Ideal for inventory planning
- • **30-Day Forecast**: Strategic planning and trend analysis

TIP: Shorter forecasts are generally more accurate. The 7-day forecast typically achieves 90%+ accuracy.

Model Accuracy

StorePulse displays model confidence using several metrics:

- • **MAPE (Mean Absolute Percentage Error)**: Typically under 10%
- • **RMSE (Root Mean Squared Error)**: Lower is better
- • **R^2 Score**: Measures how well the model fits (0-1, closer to 1 is better)

You can view these metrics in **Dashboard** → **Analytics** → **Model Performance**.

Training the Model

The model automatically retrains:

- • **Daily**: At 2:00 AM with previous day's data
- • **Weekly**: Full retrain on Sundays
- • **Manual**: Click "Update Model" in Settings

NOTE: Manual retraining can take 15-30 minutes depending on data volume.

Advanced Features

Historical Analytics

Access detailed historical analysis through **Analytics** → **History**:

Trends View

Visualize long-term patterns:

- • **Monthly Comparisons**: See year-over-year growth
- • **Day-of-Week Analysis**: Identify which days are busiest
- • **Hour-by-Hour Patterns**: Optimize staff shifts

Anomaly Detection

StorePulse automatically flags unusual events:

- • **Traffic Spikes**: Days with unexpected high volume
- • **Drops**: Unusual quiet periods
- • **Pattern Breaks**: When regular patterns change

Click any anomaly to:

- • See potential causes
- • Compare with similar days
- • Adjust future predictions

Custom Reports

Generate professional reports for stakeholders:

Report Types

Daily Summary

- • Yesterday's performance
- • Comparison to forecast

- • Key insights and recommendations

Weekly Performance

- • 7-day overview
- • Staff efficiency metrics
- • Revenue correlation (if enabled)

Monthly Analysis

- • Comprehensive monthly review
- • Trend analysis
- • Strategic recommendations

Creating a Report

1. Navigate to **Reports** → **New Report**
2. Select report type and date range
3. Choose metrics to include
4. Click **Generate**
5. Download as PDF or send via email

TIP: Schedule reports to generate automatically every Monday morning.

Data Export

Export your data for external analysis:

Export Formats

- • **CSV**: For Excel and data analysis tools
- • **JSON**: For programmatic access
- • **Excel (.xlsx)**: With formatting and charts

- • **PDF**: Professional formatted reports

What You Can Export

- • **Raw Visit Data**: Every recorded entry/exit
- • **Hourly Aggregates**: Visits summarized by hour
- • **Daily Totals**: One row per day
- • **Predictions**: Future forecasts
- • **Custom Queries**: Use the Query Builder

Configuration Guide

Database Settings

Configure your database connection in `config.json` :

```
{
  "database": {
    "host": "localhost",
    "port": 5432,
    "name": "storepulse_db",
    "user": "postgres",
    "password": "your_password_here"
  }
}
```

PostgreSQL Setup (Recommended)

For production use, we recommend PostgreSQL:

```
# Install PostgreSQL
brew install postgresql

# Start PostgreSQL service
brew services start postgresql

# Create database
createdb storepulse_db

# Run migrations
python scripts/migrate.py
```

SQLite Setup (Development)

For testing or small deployments:

```
{
  "database": {
    "type": "sqlite",
    "path": "./data/storepulse.db"
  }
}
```

WARNING: SQLite is not recommended for production with high traffic.

Model Configuration

Tune the prediction model in `ml_config.json` :

```
{
  "model": {
    "type": "ingarch_v2",
    "lag_order": 7,
    "seasonal_periods": [7, 30],
    "confidence_interval": 0.85
  }
}
```

Parameters Explained

- • **lag_order**: How many previous days to consider (7 = one week)
- • **seasonal_periods**: Cyclical patterns to detect [weekly, monthly]
- • **confidence_interval**: Prediction band width (0.85 = ±15%)

TIP: Don't change these unless you understand time-series modeling. The defaults work well for most retail scenarios.

Alert Configuration

Set up automatic alerts for important events:

Email Alerts

```
{
  "alerts": {
    "email": {
      "enabled": true,
      "smtp_server": "smtp.gmail.com",
      "smtp_port": 587,
      "from_email": "noreply@yourdomain.com",
      "to_emails": ["manager@yourdomain.com"]
    }
  }
}
```

Alert Triggers

Configure when to send alerts:

- • **High Traffic**: When predicted visits exceed X%
- • **Low Traffic**: When predicted visits drop below Y%
- • **Anomalies**: Unusual patterns detected
- • **Model Accuracy**: When prediction errors increase

API Configuration

If integrating StorePulse with other systems:

```
{
  "api": {
    "enabled": true,
    "port": 8000,
    "allowed_origins": ["http://localhost:3000"],
    "rate_limit": "100/hour",
    "require_auth": true
  }
}
```

Security Note: Always enable authentication in production and use HTTPS.

User Management

Creating Users

Admin Panel → Users → Add New User

User Roles

Administrator

- • Full system access
- • User management
- • Configuration changes
- • Data export

Manager

- • View all analytics
- • Generate reports
- • Export data
- • Cannot modify settings

Viewer

- • View dashboard only
- • No export capabilities
- • Read-only access

TIP: Follow the principle of least privilege - give users only the access they need.

Password Policies

Configure password requirements in **Settings** → **Security**:

- • **Minimum Length**: 8-20 characters
- • **Complexity**: Require uppercase, numbers, symbols
- • **Expiration**: Force password changes every 90 days
- • **History**: Prevent reusing last 5 passwords

Two-Factor Authentication

Enable 2FA for enhanced security:

1. **Settings** → **Security** → **Enable 2FA**
2. Scan QR code with authenticator app (Google Authenticator, Authy)
3. Enter verification code
4. Save backup codes in a secure location

WARNING: If you lose access to your 2FA device and backup codes, account recovery requires administrator intervention.

API Reference

StorePulse provides a RESTful API for integration with external systems.

Authentication

All API requests require an API key:

```
    curl -H "Authorization: Bearer YOUR_API_KEY" \
      http://localhost:8000/api/v1/predictions
```

Generate API keys in **Settings** → **API** → **Create New Key**.

Core Endpoints

Get Predictions

GET `/api/v1/predictions`

Retrieve forecasted visits for upcoming days.

Parameters:

- • \`days\` (integer): Number of days to forecast (default: 7, max: 30)
- • \`confidence\` (boolean): Include confidence intervals (default: true)

Example Response:

```
{
  "predictions": [
    {"date": "2024-11-22", "visits": 1250, "confidence": [1100, 1400]},
    {"date": "2024-11-23", "visits": 980, "confidence": [850, 1110]}
  ],
  "model_accuracy": {
    "mape": 8.5,
    "r_squared": 0.92
  }
}
```

Get Historical Data

GET `/api/v1/history`

Retrieve actual visit data from the past.

Parameters:

- • `start_date` (string): ISO format date (YYYY-MM-DD)
- • `end_date` (string): ISO format date
- • `granularity` (string): 'hourly', 'daily', 'weekly'

Update Visit Data

POST `/api/v1/visits`

Add new visit records (manual or from external sensors).

Request Body:

```
{
  "timestamp": "2024-11-21T14:30:00Z",
  "visits": 45,
  "source": "door_sensor"
}
```

Trigger Model Retrain

POST `/api/v1/model/retrain`

Force immediate model retraining with latest data.

NOTE: This is resource-intensive. Use sparingly.

Rate Limiting

API requests are limited to:

- • **Free Tier**: 100 requests/hour
- • **Pro Tier**: 1000 requests/hour
- • **Enterprise**: Unlimited

Exceed these limits and you'll receive a `429 Too Many Requests` response.

Troubleshooting

Common Issues

Database Connection Failed (Error 1001)

Symptoms: Cannot start application, "Database connection refused" error

Solutions:

1. **Check PostgreSQL is running:**

`bash

brew services list

# If not running:

brew services start postgresql

`

2. **Verify credentials in config.json:**

- • Ensure username/password are correct
- • Check database name exists

3. **Test connection manually:**

`bash

psql -h localhost -U postgres -d storepulse_db

`

4. **Check firewall settings:**

Port 5432 must be accessible

Model Timeout (Error 1002)

Symptoms: Prediction requests hang or timeout

Solutions:

1. **Reduce prediction horizon:**

- • Use 7 days instead of 30 days
- • Shorter forecasts are faster

2. **Increase timeout in config:**

`json

{"model": {"timeout_seconds": 300}}

`

3. **Check data volume:**

- • Very large datasets (>5 years) may need optimization
- • Consider archiving old data

Frontend Won't Load

Symptoms: Blank page or connection error in browser

Solutions:

1. **Check backend is running:**

`bash

curl http://localhost:8000/health

# Should return: {"status": "ok"}

`

2. **Check frontend server:**

`bash

cd frontend

npm run dev

`

3. **Clear browser cache:**

- • Hard refresh: Ctrl+Shift+R (Windows) or Cmd+Shift+R (Mac)

4. **Check console for errors:**

- • Open browser DevTools (F12)
- • Look for error messages

Inaccurate Predictions

Symptoms: Forecasts don't match reality, high MAPE

Solutions:

1. **Ensure sufficient historical data:**

- • Model needs at least 60 days of data
- • More data = better accuracy

2. **Check for data quality issues:**

- • Missing days
- • Incorrect values (negative visits, extreme outliers)

3. **Retrain the model:**

- • Go to **Settings** → **Model** → **Retrain**
- • Or via API: `POST /api/v1/model/retrain`

4. **Consider external factors:**

- • Did your business change recently?
- • New competition nearby?
- • Update model configuration

Slow Dashboard Loading

Symptoms: Dashboard takes >5 seconds to load

Solutions:

1. **Optimize database queries:**

`bash

python scripts/optimize_db.py

`

2. **Enable caching:**

`json

{"cache": {"enabled": true, "ttl": 300}}

`

3. **Reduce data range displayed:**

- • Show last 90 days instead of all-time

4. **Check server resources:**

`bash

htop # Monitor CPU/RAM usage

`

Log Files

View detailed logs for debugging:

- • **Backend Logs**: `logs/backend.log`
- • **Model Logs**: `logs/ml_model.log`
- • **API Logs**: `logs/api.log`
- • **Error Logs**: `logs/errors.log`

TIP: Use `tail -f logs/backend.log` to watch logs in real-time.

Getting Help

If you can't resolve an issue:

1. **Check the FAQ**: https://docs.storepulse.com/faq

2. **Community Forum**: https://community.storepulse.com

3. **Email Support**: support@storepulse.com

4. **GitHub Issues**: For bug reports and feature requests

When contacting support, include:

- • StorePulse version (`python --version` and check `package.json`)
- • Operating system
- • Relevant log excerpts
- • Steps to reproduce the issue

Best Practices

Data Collection

Consistency is Key

- • **Regular Updates**: Ensure visit data is recorded every day
- • **No Gaps**: Missing days reduce model accuracy
- • **Validate Entries**: Check for unrealistic values (negative, extremely high)

Quality Over Quantity

- • **Accurate Sensors**: Invest in reliable door counters
- • **Manual Verification**: Spot-check automated counts weekly
- • **Document Anomalies**: Note special events (sales, closures)

Using Predictions Effectively

Staff Scheduling

- • **Week-Ahead Planning**: Use 7-day forecasts for schedules
- • **Buffer for Uncertainty**: Staff 10% above predicted peak
- • **Flexible Shifts**: Prepare on-call staff for unexpected rushes

Inventory Management

- • **Align Ordering**: Stock up before predicted busy periods
- • **Seasonal Adjustments**: Trust the model's seasonal detection

- • **Reduce Waste**: Lower orders before predicted slow periods

Marketing Decisions

- • **Campaign Timing**: Launch promotions before predicted low periods to boost traffic
- • **Measure Impact**: Compare actual vs predicted to see campaign effectiveness
- • **A/B Testing**: Run campaigns on predicted similar days for fair comparison

Model Maintenance

Regular Retraining

- • **Weekly Full Retrain**: Let the Sunday automated retrain complete
- • **After Major Events**: Manually retrain after holidays or significant changes
- • **Seasonal Updates**: Retrain when entering new seasons

Monitor Accuracy

- • **Weekly MAPE Check**: Keep it under 12%
- • **Investigate Degradation**: If accuracy drops, check for:
- • Data quality issues
- • Business changes
- • External factors (new competition)

Configuration Tuning

Start with defaults, then tune if needed:

1. **Collect 3 months of data** with default settings
2. **Analyze prediction errors** to identify patterns

3. **Adjust lag_order** if weekly patterns aren't captured
4. **Modify seasonal_periods** for monthly events
5. **Test changes** with historical data before deploying

Security Best Practices

Account Security

- • **Strong Passwords**: Use password manager, 16+ character passwords
- • **Enable 2FA**: Mandatory for admin accounts
- • **Regular Audits**: Review user access quarterly
- • **Disable Unused Accounts**: Remove accounts for former employees immediately

Data Protection

- • **Regular Backups**: Daily automated backups, test restoration monthly
- • **Encrypt Sensitive Data**: Use database encryption for PII
- • **Access Logs**: Monitor who accesses what data
- • **GDPR Compliance**: If in EU, ensure proper data handling

API Security

- • **Rotate Keys**: Change API keys every 90 days
- • **IP Whitelisting**: Restrict API access to known IPs
- • **HTTPS Only**: Never use API over unencrypted connections
- • **Rate Limiting**: Prevent abuse with strict limits

Performance Optimization

Database

- • **Regular Indexing**: Run `python scripts/optimize_db.py` monthly
- • **Archive Old Data**: Move data older than 3 years to cold storage
- • **Query Optimization**: Use the built-in slow query log

Frontend

- • **Browser Caching**: Configure proper cache headers
- • **Image Optimization**: Compress dashboard screenshots
- • **Lazy Loading**: Only load data when user scrolls to it

Backend

- • **Caching**: Enable Redis cache for frequently accessed data
- • **Async Processing**: Use background tasks for heavy computations
- • **Load Balancing**: For high-traffic stores, use multiple backend instances

---

Advanced Topics

Multi-Store Management

If you manage multiple locations, StorePulse supports multi-tenancy:

Setup

1. **Create separate databases** for each store:

`bash

createdb storepulse_store1

createdb storepulse_store2

`

2. **Configure store-specific settings** in `stores.json`:

`json

{

"stores": [

{"id": "store1", "name": "Downtown", "db": "storepulse_store1"},

{"id": "store2", "name": "Mall", "db": "storepulse_store2"}

]

}

`

3. **Access via store selector** in dashboard header

Cross-Store Analytics

Compare performance across locations:

- • **Benchmarking**: See which stores perform best
- • **Best Practices**: Identify successful strategies to replicate
- • **Resource Allocation**: Direct resources to high-potential stores

Custom Integrations

POS System Integration

Connect StorePulse to your Point-of-Sale system:

```
# Example: Send sales data to enhance predictions
import requests

def send_sales_data(date, revenue, transactions):
    response = requests.post(
        'http://localhost:8000/api/v1/sales',
        headers={'Authorization': 'Bearer YOUR_API_KEY'},
        json={
            'date': date,
            'revenue': revenue,
            'transactions': transactions
        }
    )
    return response.json()
```

Benefit: Model can correlate traffic with revenue for better insights.

Weather Data Integration

Incorporate weather forecasts for enhanced accuracy:

1. **Sign up for weather API** (OpenWeatherMap, Weather.com)

2. **Configure in weather_config.json**:

`json

{

"weather": {

"enabled": true,

"api_key": "your_api_key",

"location": "New York, NY"

}

}

`

3. **Model automatically adjusts** for rainy days, extreme heat, etc.

Staffing Software Integration

Auto-sync predictions with shift scheduling tools:

- • **When2Work**: Export forecasts to optimize schedules
- • **Deputy**: API integration for automatic staff recommendations
- • **Homebase**: Direct integration available

Machine Learning Deep Dive

Understanding INGARCH Parameters

For data scientists who want to fine-tune:

Lag Order (p):

- • Number of past observations influencing current prediction
- • Default: 7 (one week)
- • Increase for longer memory (e.g., 14 for bi-weekly patterns)
- • Decrease for faster computation

Seasonal Periods:

- • `[7]`: Weekly seasonality only
- • `[7, 30]`: Weekly + Monthly (default)
- • `[7, 30, 365]`: Add yearly patterns (requires 2+ years data)

Distribution Family:

- • Poisson (default): Standard count distribution
- • Negative Binomial: For over-dispersed data (high variance)

Model Diagnostics

Access advanced diagnostics:

```
python scripts/model_diagnostics.py
```

Outputs:

- • **Residual plots**: Check for patterns in errors
- • **ACF/PACF**: Autocorrelation functions
- • **AIC/BIC scores**: Model comparison metrics

Custom Model Development

Advanced users can create custom models:

1. **Extend base class**:

`python

from models.base import BasePredictor

class MyCustomModel(BasePredictor):

def train(self, data):

# Your training logic

pass

def predict(self, horizon):

# Your prediction logic

pass

`

2. **Register in model registry**:

`python

from models import register_model

register_model('my_custom_model', MyCustomModel)

`

3. **Configure in ml_config.json**:

`json

{"model": {"type": "my_custom_model"}}

`

Appendix

Keyboard Shortcuts

Speed up your workflow with these shortcuts:

Dashboard

- • `Ctrl/Cmd + R`: Refresh data
- • `Ctrl/Cmd + E`: Export current view
- • `Ctrl/Cmd + P`: Generate report
- • `Ctrl/Cmd + F`: Search/Filter

Analytics

- • `Ctrl/Cmd + Z`: Zoom to date range
- • `Ctrl/Cmd + D`: Compare dates
- • Arrow Keys: Navigate time periods

Glossary

MAPE (Mean Absolute Percentage Error): Average prediction error as a percentage. Lower is better. Target: <10%.

INGARCH: Integer-valued Generalized Autoregressive Conditional Heteroskedasticity. Our core prediction model optimized for count data.

Footfall: Number of customers entering a store within a time period.

Confidence Interval: Range where actual value is likely to fall. 85% CI means 85% of the time, actual visits will be within this range.

Anomaly: Unusual pattern or value that deviates significantly from expectations.

Lag Order: In time series, the number of previous time periods used to predict the current value.

Seasonality: Regular, predictable patterns that repeat over time (weekly, monthly, yearly).

File Structure Reference

```
StorePulse/
■■■ backend/
■   ■■■ app.py                 # Main backend server
■   ■■■ models/                 # ML models
■   ■■■ api/                    # API endpoints
■   ■■■ database/               # Database schemas
■■■ frontend/
■   ■■■ src/
■   ■   ■■■ components/         # React components
■   ■   ■■■ pages/              # Page views
■   ■■■ public/                 # Static assets
■■■ scripts/
■   ■■■ generate_manual_pdf.py
■   ■■■ migrate.py
■   ■■■ optimize_db.py
■■■ docs/
■   ■■■ User_Manual.md          # This file
■   ■■■ TECHNICAL_REFERENCE.md
■■■ config.json                 # Main configuration
■■■ ml_config.json              # Model configuration
■■■ start.sh                    # Quick start script
```

Version History

v1.0.0 (Current)

- • Initial release
- • INGARCH v2 model
- • Multi-store support
- • API v1

Coming in v1.1.0

- • Real-time alerts
- • Mobile app
- • Advanced weather integration
- • Custom report templates

License

Support \u0026 Contact

Documentation

- • **User Manual**: You're reading it!
- • **Technical Reference**: For developers and system admins
- • **API Docs**: https://docs.storepulse.com/api

- • **Video Tutorials**: https://learn.storepulse.com

Community

- • **Forum**: https://community.storepulse.com
- • **Discord**: https://discord.gg/storepulse
- • **GitHub**: https://github.com/nonshenz007/StorePulse

Professional Support

- • **Email**: support@storepulse.com
- • **Phone**: +1 (555) 123-4567 (M-F, 9AM-5PM EST)
- • **Enterprise Support**: 24/7 dedicated support for Pro/Enterprise customers

NOTE: The start.sh script automatically sets up the backend and frontend environments.

Thank you for choosing StorePulse. We're committed to helping you optimize your retail operations through the power of predictive analytics. Welcome to the future of retail management!