# *Progress Report 3*

## *Intelligent Business Analytics System*

## *- for Maximizing Revenue and Efficiency*

Student Name: Seungyeol Chae

Student Number: 300362271

# Work Logs

| Date | Hours Worked | Description of Work |
|---|---|---|
| March 17 | 4 | Investigated and resolved CORS issues between the frontend (React) and backend (FastAPI). Configured CORSMiddleware correctly with allowed origins and credentials, and tested preflight behavior using curl. |
| March 18 | 5 | Focused on file upload validation. Enforced checks for required Excel columns in file_processing.py. Added error handling for missing data. Also implemented and tested a /reset-sales-data/ API to clear the database during development. |
| March 20 | 3 | Verified customer sentiment analysis and customer segmentation features. Confirmed that sentiment labels were displayed properly, and DBSCAN clusters were returned without crashing. Tested functionality with uploaded mock data. |
| March 21 | 3 | Debugged issues in demand forecasting and anomaly detection. Found problems with small datasets crashing LSTM models. Added pre-checks to avoid training when not enough data is present. Also reviewed error messages and cleaned up logging. |

# Description of Work Done

This week, I focused on incorporating better AI features into the program. Through this process, I took a step back to rethink the design of the program, considering it from the user's perspective. I reflected on which features are truly needed and what outcomes should be achieved, approaching the task as if I were rebuilding the program from scratch.

Key Features Developed:

- **Sentiment Analysis:** Automatically classifies customer reviews into POSITIVE/NEGATIVE sentiment using DistilBERT.
- **Revenue Forecasting:** Predicts 30 days of sales using Prophet and LSTM models.
- **Customer Segmentation:** Applies DBSCAN clustering to revenue and purchase data to group customers.

A mock Excel file was created containing 100+ daily records for testing. After uploading, a backend validation system checks for the required columns ('Date', 'Product', 'Revenue', 'Review'). Upon successful upload, sentiment analysis results are fetched from /sentiment-results/, customer segments from /customer-segmentation/, and forecasts from /revenue-forecast/.

## Repository Check-In of Implementation Completed

**1. Backend Services:**

- **main.py**: FastAPI application now properly initializes all API routes with CORS settings configured to support the React frontend (localhost:3000). api.py is successfully included via router.
- **api.py**: All endpoints are integrated through APIRouter. Added a global CORS preflight handler to fix persistent browser errors. Also added a reset-sales-data endpoint to clean the database for fresh testing.
- **revenue_forecasting.py**: Fixed LSTM and Prophet forecasting pipeline. Added logging and validation to handle cases where sales data is missing or insufficient. Returns 30-day forecast results as JSON.

- **demand_forecasting.py**: Implemented both LSTM and XGBoost forecasting models with proper feature engineering. Added error handling for empty datasets and normalized inputs.
- **anomaly_detection.py**: (Completed but not currently active) Contains Isolation Forest and Autoencoder models for anomaly detection using sales revenue trends.
- **customer_segmentation.py**: DBSCAN clustering works based on sales frequency and revenue. Fixed missing columns and data format issues.
- **weather_analysis.py**: Forecast model now checks for weather-related features in uploaded data and handles missing columns gracefully.

**2. Frontend Components:**

- **Dashboard.js**: Displays sentiment results, customer segments, and revenue forecasts. Integrated new charts for Prophet and LSTM.
- **DashboardForecast.js**: Added separate component for showing revenue forecast results using both Prophet and LSTM.
- **FileUpload.js**: Uploads Excel files to backend with visible upload success/failure messages. Handles CORS properly.
- **api.js**: Axios instance set with withCredentials: true and correct baseURL. Routes match backend endpoint names and structure.

**3. Database & File Processing:**

- **file_processing.py**: Processes uploaded Excel file. Validates columns like Date, Product, Revenue, and Review. Now successfully inserts uploaded data into the database.
- **db.py**: SQLAlchemy ORM configured for PostgreSQL. Added SalesData model for storing uploaded sales entries.
- **config.py**: Loads environment variables for DB settings using .env. Used to connect frontend and backend cleanly.
- **upload directory**: Automatically created when backend starts (data/uploads), storing Excel files persistently for reference.

# Next Steps & Improvements

So far, I have been in the stage of overhauling the program to ensure that the key points are functioning properly and that the AI features are working as intended. In the remaining time, I plan to refine these features to present the desired results more clearly and accurately. Specifically, I will create a more user-friendly graph and develop a Result Frontend page with more detailed content and explanations.

The following improvements are scheduled to be completed by the due date of the next progress report.

- Demand Forecasting: Currently fails when LSTM/XGBoost models don't receive enough data. Future fix will include training thresholds and fallback methods.
- Anomaly Detection: Although implemented, the feature is hidden due to instability. Plan to revise Isolation Forest and Autoencoder logic to handle edge cases.
- Market Basket Analysis: Not yet implemented in final workflow. Will incorporate Apriori algorithm and association rule mining.

After these, polishing UI and deploying to a demo platform are the final priorities.

# Issues Encountered and Fixes

Many issues were encountered during the development and integration of the new backend and frontend components:

- **CORS policy errors**: Most frontend API requests were blocked due to CORS restrictions. This was resolved by configuring CORSMiddleware in main.py to allow http://localhost:3000 explicitly and enabling allow_credentials=True. A custom preflight route (@router.options) was also added in api.py to handle browser OPTIONS requests properly.
- **Upload failure due to invalid Excel structure**: Early tests failed when the uploaded Excel files were missing one or more required columns. This was

addressed in file_processing.py by checking that all required columns (Date, Product, Revenue, Review) exist before processing, and by returning a clear error message if not.

- **Internal Server Error in Revenue Forecasting**: The LSTM model crashed when sales data was too short or missing entirely. Fixes included (1) adding a check for empty datasets in forecast_revenue() in revenue_forecasting.py, and (2) raising a descriptive ValueError if there was not enough data to train the model.
- **500 Errors in Demand Forecasting**: Similar to revenue forecasting, the LSTM and XGBoost models in demand_forecasting.py failed with empty input. Data fetch logic was revised to return clean time series data using .resample("D") and error handling was added to prevent model training with insufficient data.
- **Unexpected graph displays and stale data**: Some frontend charts displayed outdated results after new uploads. A /reset-sales-data/ endpoint was added in api.py to clear the sales table for fresh testing, ensuring only the latest upload is used.
- **Backend logging and debugging**: To support real-time troubleshooting, several print() statements were added in the backend (revenue_forecasting.py, file_processing.py) to log data length, column names, and error points.

These fixes helped stabilize data flow across the backend and frontend, and ensured consistent behavior across different modules.

# Conclusion

Following my professor's advice, I've focused on developing a more advanced program this week, incorporating more complex AI features. This process involved almost completely overhauling the previous version, but now, the key functionalities are working as expected.

By the end of the semester, I aim to refine these features further. I will also focus on building appropriate graphs and improving the UI to present essential information in a way that is easier for users to understand.