BMW Sales Forecasting System - Modular Structure

A production-ready, maintainable Python implementation of the BMW Sales Trend Forecasting & Alert System, split into focused, reusable modules.

Project Structure

```
+-- main.py                       # Entry point - orchestrates the analysis pipeline
+-- config.py                     # Global configuration, constants, and output directory set
+-- utils.py                      # Utility functions (logging, path helpers, formatters)
+-- data.py                       # Data loading, downloading, and preprocessing
+-- analysis.py                   # Exploratory Data Analysis (EDA) and time series aggregati
+-- forecasting.py                # Time series forecasting (ARIMA with ExponentialSmoothing
+-- visualization.py              # Static (matplotlib) and interactive (plotly) visualizatio
+-- alerts.py                     # Alert system class and alert checking logic
+-- reporting.py                  # Report generation, CSV export, and summary creation
+-- aggregator.py                 # HTML aggregator page and browser automation
+-- BMW_Sales_Forecasting_Standard-one-file.py  # Original monolithic reference script
+-- outputs/                      # Generated artifacts (PNGs, HTMLs, CSVs, logs, reports) -
    +-- 01_sales_overview.png
    +-- 02_model_region_heatmap.png
    +-- 03_arima_forecast.png
    +-- 04_model_forecasts.png
    +-- 05_interactive_dashboard.html
    +-- 06_model_heatmap_interactive.html
    +-- 07_all_outputs.html
    +-- forecast_next_3_years.csv
    +-- model_forecasts_export.csv
    +-- active_alerts.csv
    +-- sales_alerts.log
    +-- ANALYSIS_SUMMARY.txt
```

Module Descriptions

`main.py` (Entry Point)

Responsibility: Orchestrate the complete analysis pipeline by importing and calling functions from all other modules in the correct sequence.

Key Functions: - `main()` — Runs the full pipeline from data loading through final aggregation - `run_alert_checks()` — Executes all alert validation rules - `inject_test_metrics()` — Injects bad metrics for testing (TEST_MODE)

Usage:

```
python main.py
```

`config.py` (Configuration & Constants)

Responsibility: Centralize all configuration, constants, and environment setup.

Key Components: - `OUTPUT_DIR` — Path to outputs directory (auto-created) - `out_path(name)` — Helper to construct paths inside outputs/ - Data URLs and file names - Forecasting parameters (ARIMA order, forecast steps, train-test split) - Alert thresholds (multipliers for overall, model, region thresholds) - Test mode flags - Matplotlib/Pandas configuration

When to Edit: - Change output directory structure - Adjust forecasting parameters (ARIMA order, steps ahead) - Modify alert threshold multipliers - Update data URLs

---

`utils.py` (Utilities)

Responsibility: Provide reusable helper functions for logging and formatting.

Key Functions: - `setup_logger(log_file)` — Configure logging to file and console - `print_section(title)` — Print formatted section headers

When to Extend: - Add new logging handlers (e.g., email, Slack) - Add formatted output helpers - Add file I/O utilities

---

`data.py` (Data Loading & Preprocessing)

Responsibility: Handle data downloads, loading, and cleaning.

Key Functions: - `download_data_file(file_name, data_url)` — Download from URL if not present - `download_required_files()` — Download all required data files - `load_and_explore_data(csv_path)` — Load CSV and display overview - `preprocess_data(df)` — Strip whitespace from column names and validate

When to Modify: - Change data sources or URLs - Add new data validation rules - Add data cleaning or transformation steps

---

`analysis.py` (EDA & Time Series Aggregation)

Responsibility: Exploratory Data Analysis and data aggregation for time series.

Key Functions: - `exploratory_data_analysis(df_clean)` — Print sales by model, region, year, and stats - `aggregate_time_series(df_clean)` — Create yearly and model/region aggregations

When to Modify: - Add new EDA plots or statistics - Change aggregation granularity (e.g., monthly instead of yearly) - Add new groupby dimensions

---

`forecasting.py` (Time Series Forecasting)

Responsibility: Implement ARIMA forecasting with automatic fallback to ExponentialSmoothing.

Key Functions: - `forecast_with_arima(ts_data, ts_years)` — Fit ARIMA, evaluate on test set, forecast future

When to Modify: - Change ARIMA order or parameters - Implement alternative forecasting methods (Prophet, SARIMA, etc.) - Adjust fallback behavior

---

`visualization.py` (Visualizations)

Responsibility: Generate static and interactive visualizations.

Key Functions: - `create_overview_visualizations(df_yearly, df_clean)` — 4-panel overview (static PNG) - `create_heatmap(df_clean)` — Model-region heatmap (static PNG) - `visualize_forecast(...)` — ARIMA forecast plot (static PNG) - `forecast_model_specific(...)` — Top 5 model forecasts (static PNG) - `create_interactive_dashboard(...)` — Multi-subplot Plotly dashboard (HTML) - `create_heatmap_interactive(...)` — Interactive heatmap (HTML)

When to Add Visualizations: - Add new chart types - Create additional dashboards - Customize colors, fonts, or layouts

---

`alerts.py` (Alert System)

Responsibility: Define and manage the SalesAlertSystem class and alert checking logic.

Key Classes & Functions: - `SalesAlertSystem` — Core alert system with methods: - `check_overall_forecast()` — Check if forecasted sales fall below threshold - `check_model_performance()` — Check if model sales are underperforming - `check_declining_trend()` — Check for sales decline - `generate_alert_report()` — Print alert summary - `setup_alert_system(...)` — Create thresholds and initialize alert system

When to Modify: - Add new alert types or severity levels - Change alert thresholds or logic - Add custom notifications (email, Slack, etc.)

`reporting.py` (Reports & Exports)

Responsibility: Generate reports, export data to CSV, and create summaries.

Key Functions: - `generate_monthly_report(...)` — Create comprehensive monthly report (text) - `export_data(...)` — Save forecasts and alerts to CSV - `generate_final_summary(...)` — Create project completion summary (text)

When to Modify: - Change report layout or content - Add new export formats (JSON, Excel, etc.) - Customize summary metrics

---

`aggregator.py` (HTML Aggregation & Browser)

Responsibility: Create an HTML page that aggregates all generated outputs and automatically open it.

Key Functions: - `create_aggregator_html()` — Generate aggregator HTML and open dashboards in browser tabs

When to Modify: - Change HTML layout or styling - Add embedded iframes or links - Modify browser automation behavior

---

Running the Analysis

Basic Run

```
python main.py
```

Outputs are created in the `outputs/` directory and automatically opened in your browser.

Enable Test Mode

Edit `config.py` and set `TEST_MODE = True` to inject bad metrics and trigger alerts for testing:

```
TEST_MODE = True
TEST_OVERALL_FORECAST_LOW = True
TEST_MODEL_UNDERPERFORMANCE = True
TEST_REGION_DECLINE = True
TEST_DECLINING_TREND = True
```

Then run:

```
python main.py
```

Output Files

- **PNGs:** Static visualizations (sales overview, heatmaps, forecasts, model-specific charts)
- **HTMLs:** Interactive dashboards (Plotly-based, embedded in aggregator)
- **CSVs:** Forecast data, alerts, model forecasts for downstream analysis
- **TXT:** Monthly reports and project summary
- **LOG:** `sales_alerts.log` containing all alert timestamps and messages

Key Features

- **Modular Design:** Each module has a single responsibility and can be tested/extended independently
- **Configurable:** Centralized config for easy parameter adjustments
- **Error Handling:** ARIMA fallback to ExponentialSmoothing, graceful error messages
- **Logging:** File and console output for auditability
- **Automated Outputs:** Browser auto-opens with aggregated results
- **Test Mode:** Inject bad data to validate alert system
- **Comprehensive Reports:** CSV exports, monthly summaries, project completion reports

---

Maintenance Tips

1. **Add a new visualization?** Create the function in `visualization.py` and call it in `main.py`
2. **Change alert logic?** Modify `alerts.py` `SalesAlertSystem` class
3. **Add a forecasting method?** Extend `forecasting.py` with new function and update `main.py` to use it
4. **Update configuration?** Edit `config.py` (no changes to business logic needed)
5. **Add a new data source?** Extend `data.py` with a new download/load function

---

Dependencies

`pandas, numpy, matplotlib, seaborn, statsmodels, sklearn, plotly, requests`

Install with:

```
pip install pandas numpy matplotlib seaborn statsmodels scikit-learn plotly requests
```

Original Reference

The original monolithic implementation is available in `BMW_Sales_Forecasting_Standard-one-file.py` for reference or comparison.

Last Updated: November 23, 2025