Sales Forecasting Script

This documentation provides a comprehensive guide to the Sales Forecasting Script, a file-agnostic solution for automating sales predictions based on historical data.

Overview

The Sales Forecasting Script is designed to:

1. **Load data from various file formats** (CSV, Excel, Text, JSON, etc.)
2. **Automatically identify relevant columns** for date and sales values
3. **Preprocess and clean the data** for forecasting
4. **Build and train an appropriate machine learning model** based on data characteristics
5. **Generate future sales predictions** and visualizations
6. **Export the results** in various formats

Installation

Prerequisites

- Python 3.7+
- Required packages: pandas, numpy, matplotlib, seaborn, scikit-learn, statsmodels, xgboost, pmdarima

Install required packages

```bash
pip install pandas numpy matplotlib seaborn scikit-learn statsmodels xgboost pmdarima
```

Download the scripts

Download the following files:

- `sales_forecast.py` - The main forecasting script
- `sales_forecasting_examples.py` - Example usage and utilities

Basic Usage

Command Line Usage

```bash
python sales_forecast.py --file your_data_file.csv --forecast_periods 12
```

Optional Arguments

- `--date_column`: Specify the date column name (auto-detected if not provided)
- `--sales_column`: Specify the sales column name (auto-detected if not provided)
- `--forecast_periods`: Number of periods to forecast (default: 12)
- `--export_format`: Format for exporting results (options: csv, excel, json; default: csv)

Examples

Basic forecasting with automatic column detection:

```bash
python sales_forecast.py --file sales_data.csv --forecast_periods 12
```

Specifying columns manually:

```bash
python sales_forecast.py --file sales_data.csv --date_column "OrderDate" --sales_column "
```

Exporting to Excel:

```bash
python sales_forecast.py --file sales_data.csv --export_format excel
```

Programmatic Usage

You can also use the `SalesForecaster` class in your own Python code:

python

```python
from sales_forecast import SalesForecaster

# Create a forecaster instance
forecaster = SalesForecaster(
    file_path="sales_data.csv",
    date_col="OrderDate",  # Optional
    sales_col="Revenue",   # Optional
    forecast_periods=12
)

# Run the complete forecasting pipeline
forecaster.run_pipeline()

# Or run individual steps for more control
forecaster.load_data()
forecaster.identify_columns()
forecaster.preprocess_data()
forecaster.split_data(test_size=0.2)
forecaster.build_model()
forecaster.visualize_results()
forecaster.export_results(output_format='csv')
```

Features

Automatic Column Detection

If you don't specify the date and sales columns, the script will automatically identify them based on:

- Column names (looking for keywords like "date", "time", "sale", "revenue", etc.)
- Data types (dates and numeric values)
- Statistical properties

Data Preprocessing

The script handles various preprocessing tasks:

- Converts date column to datetime format
- Handles missing values
- Resamples data to a regular time frequency (daily, weekly, monthly, etc.)
- Creates useful time-based features (year, month, day of week, etc.)
- Adds lag features and rolling statistics

Model Selection

The script automatically selects the appropriate model based on data characteristics:

- **SARIMA** (Seasonal AutoRegressive Integrated Moving Average) for data with seasonality
- **XGBoost** for data without strong seasonality or with complex patterns

Evaluation and Visualization

- Evaluates model performance with metrics like MAE, RMSE, and R^2
- Creates visualizations of historical data, test predictions, and future forecasts
- Includes confidence intervals for forecasts

File Format Support

The script supports various file formats:

- CSV (.csv)
- Excel (.xls, .xlsx)
- Text files (.txt) with various delimiters

- JSON (.json)
- Parquet (.parquet)

Advanced Usage

Data Preparation Utilities

The examples script includes utilities for data preparation:

```python
from sales_forecasting_examples import aggregate_data_by_date, prep_retail_data, analyze_

# Aggregate raw data by date before forecasting
aggregate_data_by_date(
    input_file="raw_sales_data.csv",
    output_file="aggregated_sales.csv",
    date_col="order_date",
    agg_cols=["sales", "quantity"],
    agg_funcs={"sales": "sum", "quantity": "sum"}
)

# Prepare retail data specifically for the UK retail dataset
daily_file, monthly_file = prep_retail_data()

# Analyze seasonality patterns in your data
analyze_seasonality("sales_data.csv", "Date", "Sales")
```

Customizing the Model

If you want more control over the forecasting process, you can run individual steps:

```python
forecaster = SalesForecaster("sales_data.csv")
forecaster.load_data()
forecaster.identify_columns()
forecaster.preprocess_data()

# Customize the train/test split
forecaster.split_data(test_size=0.3)  # Use 30% of data for testing

# Build and evaluate the model
forecaster.build_model()

# Visualize and export results
forecaster.visualize_results()
forecaster.export_results(output_format='excel')
```

Output Files

The script generates the following output files:

1. `sales_forecast_plot.png` - Visualization of historical data and forecasts
2. `<input_filename>_forecast.<format>` - Forecast results in the specified format
3. `<input_filename>_feature_importance.<format>` - Feature importance (for XGBoost model)
4. `<input_filename>_model_summary.txt` - Model summary (for SARIMA model)

Troubleshooting

Common Issues

1. **Column Detection Fails**
   - Specify the columns manually using the `--date_column` and `--sales_column` arguments

- Ensure your data has clear date and numeric sales columns

2. **Data Format Issues**
   - Ensure your dates are in a standard format
   - If using text files, check the delimiter is correctly detected
3. **Insufficient Data**
   - The model requires sufficient historical data for accurate forecasting
   - For seasonal models, at least 2-3 complete seasonal cycles are recommended

Logging

The script uses Python's logging module to provide detailed information:

- All log messages are displayed in the console
- Check the logs for warnings and errors
- Information about detected columns, data preprocessing, and model evaluation is provided

Best Practices

1. **Data Quality**
   - Clean your data before forecasting
   - Remove outliers that might skew predictions
   - Ensure your date column has a consistent frequency
2. **Time Period**
   - Match your forecast period to your business needs
   - For daily data, forecasting 30-90 days ahead is typical
   - For monthly data, 6-12 months is common
3. **Model Evaluation**

- Check the evaluation metrics (MAE, RMSE, R^2) to assess model quality
- Compare predicted vs. actual values for the test period
- Use confidence intervals to understand forecast uncertainty

Extending the Script

The script is designed to be modular and extensible. Some possible extensions:

1. **Additional Models**: Add deep learning models like LSTM or Prophet
2. **External Factors**: Incorporate external variables like holidays, weather, or economic indicators
3. **Ensemble Methods**: Combine multiple models for improved forecasting
4. **Anomaly Detection**: Identify and handle outliers automatically
5. **Web Interface**: Build a web app for interactive forecasting

License

Happy forecasting!