

Time Series Forecasting: Walmart Sales Analysis Part 1

Comparing Aggregate vs. Direct Department-Level Forecasting

Costin-Andrei Taulescu & Panagiotis Valsamis

2026-01-23

Table of contents

1 Executive Summary	2
2 Reproducibility Setup	3
2.1 Environment Setup	3
2.2 Windows	3
2.3 macOS/Linux	3
2.4 Required Libraries	3
3 Data Overview	4
3.1 Loading the Dataset	4
3.2 Data Quality: Handling Negative Sales	6
4 Focus: Store 1 Analysis	6
4.1 Train-Test Split	7
5 Method 2: Aggregate-then-Disaggregate	7
5.1 Step 1: Aggregate Store-Level Sales	7
5.1.1 Initial Observations	9
5.2 Step 2: Statistical Tests for Trend	9
5.3 Step 3: Holt-Winters Forecasting (Additive Seasonality)	12
5.3.1 Grid Search for Optimal Parameters	12
5.3.2 Generate Forecasts	13
5.4 Step 4: Alternative Model - Multiplicative Seasonality	15
5.4.1 Comparison: Additive vs Multiplicative	17
5.5 Step 5: Disaggregation to Department Level	19
5.5.1 Apply Disaggregation	20

6 Method 1: Direct Department-Level Forecasting	23
6.1 Data Preparation	23
6.2 Visualization	23
6.3 Trend Analysis for Department 1	24
6.4 Holt-Winters Forecasting (Additive & Multiplicative)	27
7 Final Comparison: Method 1 vs Method 2	27
7.1 Performance Metrics	27
7.2 Visual Comparison	30
7.3 Forecast Errors Analysis	31
7.4 Performance Metrics Bar Chart	33
8 Understanding the Evaluation Metrics	35
8.1 Why TSE, RMSE, and MAE?	35
8.1.1 1. Total Squared Error (TSE)	35
8.1.2 2. Root Mean Squared Error (RMSE)	35
8.1.3 3. Mean Absolute Error (MAE)	35
8.2 Why Not MAPE?	36
8.3 Interpreting the Relationship	36
9 Conclusions	36
9.1 Key Findings	36
9.1.1 Why Did Method 2 Win?	36
9.1.2 When Would Method 1 Be Better?	37
9.2 Generalizability	37
9.3 Limitations and Future Work	37
9.3.1 Recommended Extensions	38
9.4 Practical Implications	38
10 References	38
10.1 Data Source	38
10.2 Methodology References	38
10.3 Statistical Tests	39
11 Appendix: Code Repository	39
11.1 Requirements	39
11.2 Session Information	39

1 Executive Summary

This project explores time series forecasting using Walmart's historical sales data. We compare two forecasting approaches:

- **Method 1:** Direct forecasting at the department level
- **Method 2:** Forecasting at an aggregated store level, then disaggregating to departments

The analysis focuses on Store 1, Department 1, using Holt-Winters seasonal exponential smoothing models.

! Key Finding

Method 2 (Aggregate-then-Disaggregate) outperformed Method 1 (Direct Forecasting) with an RMSE improvement of 15.22 (3.96%), demonstrating that aggregation can reduce noise and improve forecast accuracy for stable departments.

2 Reproducibility Setup

2.1 Environment Setup

We recommend creating a virtual environment to ensure reproducible results.

2.2 Windows

```
python -m venv ts_workshop_env
ts_workshop_env\Scripts\activate
pip install -r requirements.txt
```

2.3 macOS/Linux

```
python3 -m venv ts_workshop_env
source ts_workshop_env/bin/activate
pip install -r requirements.txt
```

2.4 Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
from scipy import stats
from statsmodels.tsa.stattools import adfuller, kpss
import pymannkendall as mk
from scipy.stats import linregress

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

# Set plotting style
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
```

3 Data Overview

3.1 Loading the Dataset

```
# Load training data
df = pd.read_csv("../data/train.csv")

print(f"Dataset shape: {df.shape}")
print(f"\nFirst few rows:")
print(df.head())
print(f"\nStatistical summary:")
print(df.describe())
print(f"\nData types and null counts:")
print(df.info())
print(f"\nMissing values:")
print(df.isnull().sum())
```

Dataset shape: (421570, 5)

First few rows:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

Statistical summary:

	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

Data types and null counts:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column      Non-Null Count   Dtype  
 ---  --          --          --      
 0   Store        421570 non-null  int64  
 1   Dept         421570 non-null  int64  
 2   Date         421570 non-null  object 
 3   Weekly_Sales 421570 non-null  float64
 4   IsHoliday    421570 non-null  bool    
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
None
```

Missing values:

```
Store      0
Dept      0
Date      0
Weekly_Sales 0
IsHoliday 0
dtype: int64
```

3.2 Data Quality: Handling Negative Sales

⚠️ Data Issue Detected

The dataset contains negative values in `Weekly_Sales`, which are not meaningful in this context. We will set these to zero.

```
def make_negative_values_zero(df, column_name):
    """Set negative values in a column to zero."""
    df.loc[df[column_name] < 0, column_name] = 0
    return df

def negative_values(df, column_name):
    """Filter rows with negative values in a column."""
    return df[df[column_name] < 0]
```

4 Focus: Store 1 Analysis

For this analysis, we focus exclusively on Store 1 to compare forecasting methods.

```
# Filter for Store 1
df_store_1 = df[df['Store'] == 1].copy()
df_store_1 = make_negative_values_zero(df_store_1, "Weekly_Sales")

print(f"Store 1 data shape: {df_store_1.shape}")
print(f"\nDepartments in Store 1:")
print(df_store_1["Dept"].value_counts().sort_index())
```

Store 1 data shape: (10244, 5)

Departments in Store 1:

Dept	Count
1	143
2	143
3	143
4	143
5	143

```

...
95    143
96    122
97    143
98    143
99     37
Name: count, Length: 77, dtype: int64

```

4.1 Train-Test Split

```

# Convert Date to datetime
df_store_1['Date'] = pd.to_datetime(df_store_1['Date'])

# Define test period: last 4 weeks
max_date = df_store_1['Date'].max()
test_start_date = max_date - pd.Timedelta(weeks=3)

print(f"Training data: Up to {test_start_date}")
print(f"Test data: {test_start_date} to {max_date}")

# Split data
df_store_1_train = df_store_1[df_store_1['Date'] < test_start_date].copy()
df_store_1_test = df_store_1[df_store_1['Date'] >= test_start_date].copy()

```

Training data: Up to 2012-10-05 00:00:00
Test data: 2012-10-05 00:00:00 to 2012-10-26 00:00:00

5 Method 2: Aggregate-then-Disaggregate

5.1 Step 1: Aggregate Store-Level Sales

```

# Aggregate all departments by date
store_1_aggregated_train = df_store_1_train.groupby('Date').agg({
    'Weekly_Sales': 'sum'
}).reset_index()

```

```

store_1_aggregated_test = df_store_1_test.groupby('Date').agg({
    'Weekly_Sales': 'sum'
}).reset_index()

store_1_aggregated = store_1_aggregated_train.sort_values('Date').reset_index(drop=True)

# Visualize aggregated sales
plt.figure(figsize=(16, 6))
plt.plot(store_1_aggregated['Date'],
         store_1_aggregated['Weekly_Sales'],
         linewidth=2,
         color='steelblue',
         marker='o',
         markersize=3)

plt.xlabel('Date', fontsize=12, fontweight='bold')
plt.ylabel('Total Weekly Sales ($)', fontsize=12, fontweight='bold')
plt.title('Store 1: Aggregated Weekly Sales Over Time (Training Set)',
          fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```



Figure 1: Store 1 Aggregated Weekly Sales (Training Period)

5.1.1 Initial Observations

Seasonal Patterns Observed

- Clear peaks at year-end (holiday shopping)
- Consistent troughs at year start (post-holiday slowdown)
- Regular seasonal cycle without strong trend

5.2 Step 2: Statistical Tests for Trend

Before modeling, we test whether the series exhibits a significant trend.

```
from run_trend_analysis import analyze_series

df_store_for_tests = store_1_aggregated_train["Weekly_Sales"]

results = analyze_series(df_store_for_tests, series_name="Store 1")

#####
# COMPREHENSIVE TREND ANALYSIS FOR: Store 1
#####

Data points: 139
Mean: 1555116.40
Std Dev: 157221.96
Min: 1316899.31
Max: 2387950.20

=====
1. MANN-KENDALL TEST (Non-parametric)
=====
Trend: increasing
Test Statistic (S): 1743.0
p-value: 0.001514
Tau (correlation): 0.181733
z-score: 3.172081

RESULT: Significant trend detected (p < 0.05)
Direction: increasing
```

2. AUGMENTED DICKEY-FULLER (ADF) TEST

ADF Statistic: -5.037909
p-value: 0.000019
Lags used: 4
Number of observations: 134

Critical Values:

1%: -3.480
5%: -2.883
10%: -2.578

RESULT: Series is stationary ($p < 0.05$)

No unit root → Suggests NO trend

3. KPSS TEST (Kwiatkowski-Phillips-Schmidt-Shin)

KPSS Test (with trend):
KPSS Statistic: 0.058672
p-value: 0.100000
Lags used: 3

Critical Values:

10%: 0.119
5%: 0.146
2.5%: 0.176
1%: 0.216

RESULT: Series is trend stationary ($p \geq 0.05$)

Null hypothesis not rejected → NO significant trend

4. LINEAR REGRESSION TEST

Slope: 872.876123
Intercept: 1494887.95
R-squared: 0.049625
p-value: 0.008392
Standard Error: 326.352820

95% Confidence Interval for slope: [227.535870, 1518.216375]

RESULT: Significant positive linear trend ($p < 0.05$)

=====

5. COX-STUART TEST (Non-parametric)

=====

Number of pairs: 69
Plus signs (+): 41 (second half > first half)
Minus signs (-): 28 (second half < first half)
Ties: 0
p-value: 0.148032

RESULT: No significant trend ($p \geq 0.05$)

=====

6. SPEARMAN'S RANK CORRELATION TEST (Bonus)

=====

Spearman's rho: 0.291193
p-value: 0.000506

RESULT: Significant positive monotonic trend ($p < 0.05$)

=====

SUMMARY OF RESULTS

=====

Trend Detection Tests:

Mann-Kendall: increasing ($p=0.0015$)
Linear Regression: Trend ($p=0.0084$)
Cox-Stuart: No Trend ($p=0.1480$)
Spearman: Trend ($p=0.0005$)

Stationarity Tests:

ADF: Stationary ($p=0.0000$)
KPSS: Trend Stationary ($p=0.1000$)

💡 Trend Analysis Conclusion

Mixed results from statistical tests suggest weak or no persistent trend. Visual inspection confirms regular seasonality without long-term increase/decrease. Therefore, we use **Holt-Winters Seasonal (No Trend)** model.

5.3 Step 3: Holt-Winters Forecasting (Additive Seasonality)

```
def TSE(y_true, y_pred):
    """Total Squared Error"""
    return np.sum((y_true - y_pred)**2)

def hw_seasonal_no_trend(y, alpha, gamma, m):
    """
    Holt-Winters Seasonal Method without Trend (Additive)

    Parameters:
    - y: time series data
    - alpha: level smoothing parameter
    - gamma: seasonal smoothing parameter
    - m: season length (52 for weekly data with yearly seasonality)
    """
    n = len(y)

    # Initialization
    level = np.mean(y[:m])
    seasonals = list(y[:m] - level)

    fitted = [np.nan] * n

    for t in range(m, n):
        fitted[t] = level + seasonals[t - m]

        new_level = alpha * (y[t] - seasonals[t - m]) + (1 - alpha) * level
        new_seasonal = gamma * (y[t] - new_level) + (1 - gamma) * seasonals[t - m]

        level = new_level
        seasonals.append(new_seasonal)

    return np.array(fitted), level, seasonals
```

5.3.1 Grid Search for Optimal Parameters

```
y_train = store_1_aggregated_train["Weekly_Sales"].values
m = 52 # Weekly data, yearly seasonality
```

```

alphas = np.arange(0.05, 1.0, 0.05)
gammas = np.arange(0.05, 1.0, 0.05)

best_tse = np.inf
best_params = None
best_model = None

print("Running grid search for optimal parameters...")
for alpha in alphas:
    for gamma in gammas:
        fitted, level, seasonals = hw_seasonal_no_trend(y_train, alpha, gamma, m)

        valid_idx = ~np.isnan(fitted)
        tse = TSE(y_train[valid_idx], fitted[valid_idx])

        if tse < best_tse:
            best_tse = tse
            best_params = (alpha, gamma)
            best_model = (level, seasonals)

print(f"Best parameters: alpha={best_params[0]:.2f}, gamma={best_params[1]:.2f}")
print(f"Best TSE (training): {best_tse:.2f}")

```

Running grid search for optimal parameters...
 Best parameters: alpha=0.15, gamma=0.75
 Best TSE (training): 388685740510.08

5.3.2 Generate Forecasts

```

h = len(store_1_aggregated_test)
alpha_best, gamma_best = best_params
level, seasonals = best_model
seasonals = np.array(seasonals)

forecast = []
for i in range(h):
    seasonal_index = len(seasonals) - m + i
    forecast.append(level + seasonals[seasonal_index])

```

```

forecast = np.array(forecast)

# Evaluation
y_test = store_1_aggregated_test["Weekly_Sales"].values
tse_test = TSE(y_test, forecast)
rmse_test = np.sqrt(np.mean((y_test - forecast) ** 2))
mae_test = np.mean(np.abs(y_test - forecast))

print(f"Test TSE: {tse_test:.2f}")
print(f"Test RMSE: {rmse_test:.2f}")
print(f"Test MAE: {mae_test:.2f}")

# Visualization
plt.figure(figsize=(16, 6))
plt.plot(store_1_aggregated_train['Date'],
         store_1_aggregated_train['Weekly_Sales'],
         label='Training Data',
         linewidth=2,
         color='steelblue',
         marker='o',
         markersize=3)

plt.plot(store_1_aggregated_test['Date'],
         store_1_aggregated_test['Weekly_Sales'],
         label='Test Data',
         linewidth=2,
         color='orange',
         marker='o',
         markersize=3)

plt.plot(store_1_aggregated_test['Date'],
         forecast,
         label='Holt-Winters Forecast (Additive)',
         linewidth=2,
         color='green',
         marker='o',
         markersize=3)

plt.xlabel('Date', fontsize=12, fontweight='bold')
plt.ylabel('Total Weekly Sales ($)', fontsize=12, fontweight='bold')
plt.title('Store 1: Holt-Winters Seasonal Forecast vs Actuals (Additive)',
          fontsize=14, fontweight='bold')

```

```

plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Store results
additive_results = {
    'alpha': best_params[0],
    'gamma': best_params[1],
    'tse_train': best_tse,
    'tse_test': tse_test,
    'rmse_test': rmse_test,
    'mae_test': mae_test,
    'forecast': forecast.copy()
}

```

Test TSE: 775926584.92

Test RMSE: 13927.73

Test MAE: 11983.60

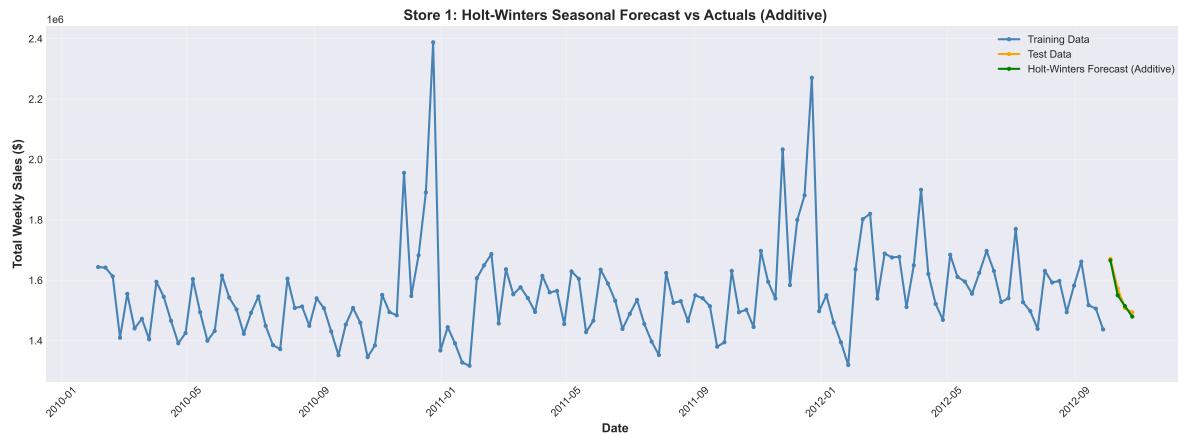


Figure 2: Holt-Winters Additive Forecast vs Actuals

5.4 Step 4: Alternative Model - Multiplicative Seasonality

```

def hw_seasonal_no_trend_multiplicative(y, alpha, gamma, m):
    """Holt-Winters Seasonal Method without Trend (Multiplicative)"""
    n = len(y)

    level = np.mean(y[:m])
    seasonals = list(y[:m] / level)

    fitted = [np.nan] * n

    for t in range(m, n):
        fitted[t] = level * seasonals[t - m]

        new_level = alpha * (y[t] / seasonals[t - m]) + (1 - alpha) * level
        new_seasonal = gamma * (y[t] / new_level) + (1 - gamma) * seasonals[t - m]

        level = new_level
        seasonals.append(new_seasonal)

    return np.array(fitted), level, seasonals

# Grid search for multiplicative model
best_tse = np.inf
best_params = None
best_model = None

for alpha in alphas:
    for gamma in gammas:
        fitted, level, seasonals = hw_seasonal_no_trend_multiplicative(
            y_train, alpha, gamma, m
        )

        valid_idx = ~np.isnan(fitted)
        tse = TSE(y_train[valid_idx], fitted[valid_idx])

        if tse < best_tse:
            best_tse = tse
            best_params = (alpha, gamma)
            best_model = (level, seasonals)

print(f"Multiplicative Model - Best parameters: alpha={best_params[0]:.2f}, gamma={best_params[1]:.2f}")
print(f"Best TSE (training): {best_tse:.2f}")

```

```

# Forecast
level, seasonals = best_model
seasonals = np.array(seasonals)

forecast_mult = []
for i in range(h):
    seasonal_index = len(seasonals) - m + i
    forecast_mult.append(level * seasonals[seasonal_index])

forecast_mult = np.array(forecast_mult)

# Evaluation
tse_test_mult = TSE(y_test, forecast_mult)
rmse_test_mult = np.sqrt(np.mean((y_test - forecast_mult) ** 2))
mae_test_mult = np.mean(np.abs(y_test - forecast_mult))

print(f"Test TSE: {tse_test_mult:.2f}")
print(f"Test RMSE: {rmse_test_mult:.2f}")
print(f"Test MAE: {mae_test_mult:.2f}")

multiplicative_results = {
    'alpha': best_params[0],
    'gamma': best_params[1],
    'tse_train': best_tse,
    'tse_test': tse_test_mult,
    'rmse_test': rmse_test_mult,
    'mae_test': mae_test_mult,
    'forecast': forecast_mult.copy()
}

```

Multiplicative Model - Best parameters: alpha=0.15, gamma=0.70
Best TSE (training): 405240934553.55
Test TSE: 842717466.42
Test RMSE: 14514.80
Test MAE: 11739.95

5.4.1 Comparison: Additive vs Multiplicative

Table 1: Performance Comparison: Additive vs Multiplicative Seasonality

```

comparison_df = pd.DataFrame({
    'Metric': ['Alpha', 'Gamma', 'Train TSE', 'Test TSE', 'Test RMSE', 'Test MAE'],
    'Additive': [
        f"{{additive_results['alpha']:.2f}}",
        f"{{additive_results['gamma']:.2f}}",
        f"{{additive_results['tse_train']:.2f}}",
        f"{{additive_results['tse_test']:.2f}}",
        f"{{additive_results['rmse_test']:.2f}}",
        f"{{additive_results['mae_test']:.2f}}"
    ],
    'Multiplicative': [
        f"{{multiplicative_results['alpha']:.2f}}",
        f"{{multiplicative_results['gamma']:.2f}}",
        f"{{multiplicative_results['tse_train']:.2f}}",
        f"{{multiplicative_results['tse_test']:.2f}}",
        f"{{multiplicative_results['rmse_test']:.2f}}",
        f"{{multiplicative_results['mae_test']:.2f}}"
    ]
})
print(comparison_df.to_string(index=False))

```

Metric	Additive	Multiplicative
Alpha	0.15	0.15
Gamma	0.75	0.70
Train TSE	388685740510.08	405240934553.55
Test TSE	775926584.92	842717466.42
Test RMSE	13927.73	14514.80
Test MAE	11983.60	11739.95

Model Selection

Both models produce nearly identical forecasts. This similarity occurs because seasonal fluctuations remain relatively constant in absolute terms. We proceed with the **additive model** for disaggregation.

5.5 Step 5: Disaggregation to Department Level

```
# Filter Department 1 data
store_1_dept_1_train = df_store_1_train[
    (df_store_1_train['Store'] == 1) & (df_store_1_train['Dept'] == 1)
].copy()
store_1_dept_1_train = store_1_dept_1_train.sort_values('Date')

# Merge to calculate proportions
all_merged = pd.merge(
    store_1_dept_1_train[['Date', 'Weekly_Sales']],
    store_1_aggregated_train[['Date', 'Weekly_Sales']],
    on='Date',
    suffixes=('_dept', '_store')
)

all_merged['proportion'] = all_merged['Weekly_Sales_dept'] / all_merged['Weekly_Sales_store']

avg_proportion = all_merged['proportion'].mean()
std_proportion = all_merged['proportion'].std()

print(f"Department 1's proportion of Store 1 sales:")
print(f"  Mean: {avg_proportion:.4f} ({avg_proportion*100:.2f}%)")
print(f"  Std:  {std_proportion:.4f}")

# Visualize proportion stability
plt.figure(figsize=(14, 5))
plt.plot(all_merged['Date'], all_merged['proportion'],
        marker='o', linewidth=1, markersize=3, alpha=0.7, color='steelblue')
plt.axhline(y=avg_proportion, color='red', linestyle='--', linewidth=2,
            label=f'Mean: {avg_proportion:.4f}')
plt.axhline(y=avg_proportion + std_proportion, color='orange', linestyle=':', linewidth=1.5, alpha=0.7, label=f'±1 Std Dev')
plt.axhline(y=avg_proportion - std_proportion, color='orange', linestyle=':',
```

```

        linewidth=1.5, alpha=0.7)
plt.xlabel('Date', fontsize=11, fontweight='bold')
plt.ylabel('Department 1 Proportion', fontsize=11, fontweight='bold')
plt.title('Department 1 as Proportion of Store 1 Total Sales (Training Period)', 
          fontsize=12, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

Department 1's proportion of Store 1 sales:

Mean: 0.0143 (1.43%)
 Std: 0.0054



Figure 3: Department 1 as Proportion of Store 1 Total Sales

5.5.1 Apply Disaggregation

```

# Disaggregate store-level forecast
dept_1_forecast = additive_results['forecast'] * avg_proportion

# Get actual Department 1 test data
store_1_dept_1_test = df_store_1_test[
    (df_store_1_test['Store'] == 1) & (df_store_1_test['Dept'] == 1)
].copy()
store_1_dept_1_test = store_1_dept_1_test.sort_values('Date')

```

```

y_test_dept_1 = store_1_dept_1_test['Weekly_Sales'].values

# Evaluate
tse_dept_1 = TSE(y_test_dept_1, dept_1_forecast)
rmse_dept_1 = np.sqrt(np.mean((y_test_dept_1 - dept_1_forecast) ** 2))
mae_dept_1 = np.mean(np.abs(y_test_dept_1 - dept_1_forecast))

print(f"Department 1 Disaggregated Forecast Performance (Method 2):")
print(f"  Test TSE: {tse_dept_1:.2f}")
print(f"  Test RMSE: {rmse_dept_1:.2f}")
print(f"  Test MAE: {mae_dept_1:.2f}")

# Visualization
store_1_dept_1_train_plot = df_store_1_train[
    (df_store_1_train['Store'] == 1) & (df_store_1_train['Dept'] == 1)
].copy()
store_1_dept_1_train_plot = store_1_dept_1_train_plot.sort_values('Date')

plt.figure(figsize=(16, 6))
plt.plot(store_1_dept_1_train_plot['Date'],
         store_1_dept_1_train_plot['Weekly_Sales'],
         label='Training Data (Dept 1)',
         linewidth=2,
         color='steelblue',
         marker='o',
         markersize=3)

plt.plot(store_1_dept_1_test['Date'],
         y_test_dept_1,
         label='Test Data (Dept 1)',
         linewidth=2,
         color='orange',
         marker='o',
         markersize=3)

plt.plot(store_1_dept_1_test['Date'],
         dept_1_forecast,
         label='Disaggregated Forecast (Method 2)',
         linewidth=2,
         color='green',
         marker='o',
         markersize=3)

```

```

plt.xlabel('Date', fontsize=12, fontweight='bold')
plt.ylabel('Weekly Sales ($)', fontsize=12, fontweight='bold')
plt.title('Store 1, Department 1: Disaggregated Forecast vs Actuals (Method 2)', 
          fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Store Method 2 results
method2_results = {
    'tse_test': tse_dept_1,
    'rmse_test': rmse_dept_1,
    'mae_test': mae_dept_1,
    'forecast': dept_1_forecast.copy()
}

```

Department 1 Disaggregated Forecast Performance (Method 2):

Test TSE: 49541178.78

Test RMSE: 3519.27

Test MAE: 2827.42

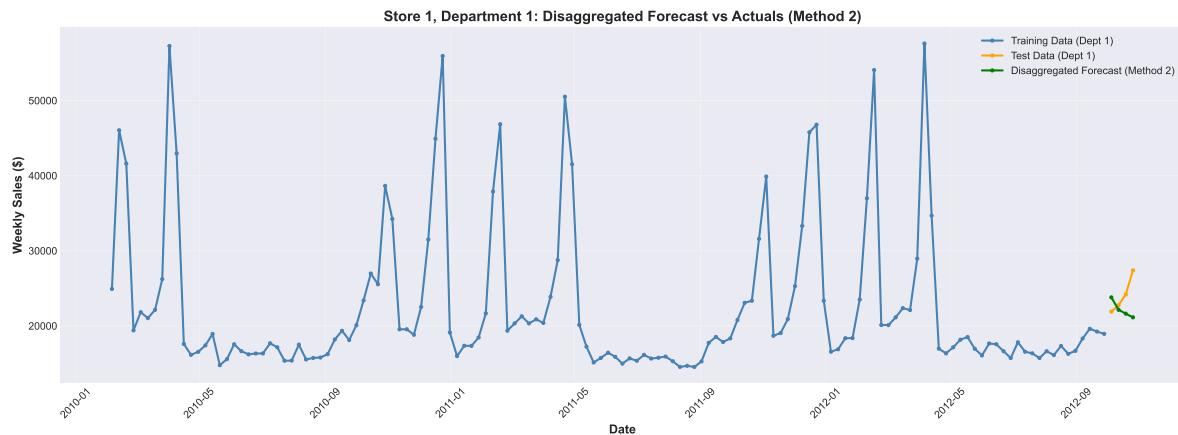


Figure 4: Method 2: Disaggregated Forecast for Department 1

6 Method 1: Direct Department-Level Forecasting

6.1 Data Preparation

```
# Filter Store 1, Department 1
df_store_1_dept_1 = df[(df['Store'] == 1) & (df['Dept'] == 1)].copy()
df_store_1_dept_1 = make_negative_values_zero(df_store_1_dept_1, "Weekly_Sales")
df_store_1_dept_1['Date'] = pd.to_datetime(df_store_1_dept_1['Date'])

# Split train/test
df_store_1_dept_1_train = df_store_1_dept_1[df_store_1_dept_1['Date'] < test_start_date].copy()
df_store_1_dept_1_test = df_store_1_dept_1[df_store_1_dept_1['Date'] >= test_start_date].copy()
df_store_1_dept_1_train = df_store_1_dept_1_train.sort_values('Date')
df_store_1_dept_1_test = df_store_1_dept_1_test.sort_values('Date')

print(f"Department 1 training data shape: {df_store_1_dept_1_train.shape}")
print(f"Department 1 test data shape: {df_store_1_dept_1_test.shape}")
```

Department 1 training data shape: (139, 5)
Department 1 test data shape: (4, 5)

6.2 Visualization

```
plt.figure(figsize=(16, 6))

plt.plot(df_store_1_dept_1_train['Date'],
          df_store_1_dept_1_train['Weekly_Sales'],
          label='Training Data (Dept 1)',
          linewidth=2,
          color='steelblue',
          marker='o',
          markersize=3)

plt.plot(df_store_1_dept_1_test['Date'],
          df_store_1_dept_1_test['Weekly_Sales'],
          label='Test Data (Dept 1)',
          linewidth=2,
          color='orange',
          marker='o',
```

```

    markersize=3)

plt.xlabel('Date', fontsize=12, fontweight='bold')
plt.ylabel('Weekly Sales ($)', fontsize=12, fontweight='bold')
plt.title('Store 1, Department 1: Actual Weekly Sales Over Time',
          fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

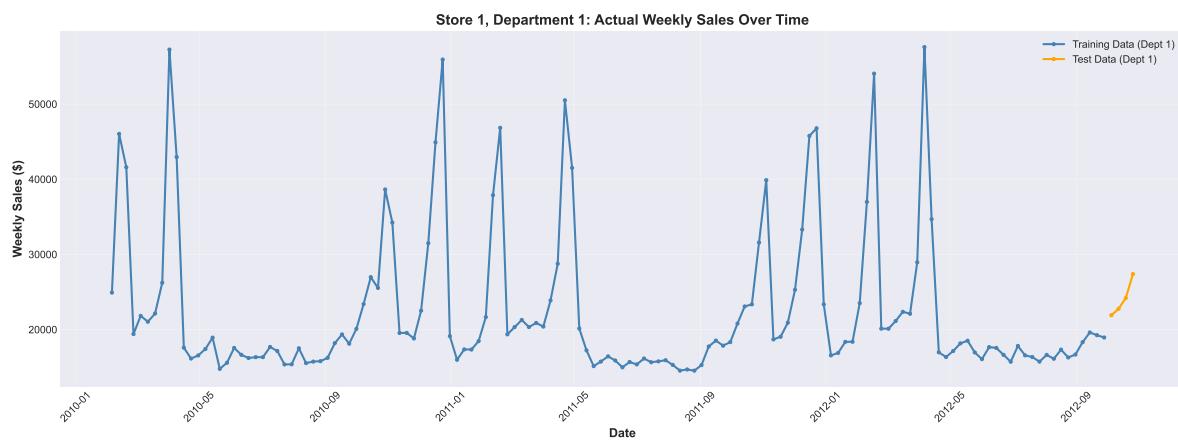


Figure 5: Store 1, Department 1: Raw Weekly Sales Data

6.3 Trend Analysis for Department 1

```

analyze_series(df_store_1_dept_1_train["Weekly_Sales"],
               series_name="Store 1, Department 1")

```

```

#####
# COMPREHENSIVE TREND ANALYSIS FOR: Store 1, Department 1
#####

```

```

Data points: 139
Mean: 22468.78
Std Dev: 9950.26

```

Min: 14537.37
Max: 57592.12

=====
1. MANN-KENDALL TEST (Non-parametric)
=====

Trend: no trend
Test Statistic (S): -423.0
p-value: 0.442227
Tau (correlation): -0.044104
z-score: -0.768437

RESULT: No significant trend (p >= 0.05)

=====
2. AUGMENTED DICKEY-FULLER (ADF) TEST
=====

ADF Statistic: -2.405435
p-value: 0.140208
Lags used: 6
Number of observations: 132

Critical Values:

1%: -3.481
5%: -2.884
10%: -2.579

RESULT: Series is non-stationary (p >= 0.05)

Unit root present → May have trend

=====
3. KPSS TEST (Kwiatkowski-Phillips-Schmidt-Shin)
=====

KPSS Test (with trend):
KPSS Statistic: 0.081207
p-value: 0.100000
Lags used: 4

Critical Values:

10%: 0.119
5%: 0.146
2.5%: 0.176
1%: 0.216

```
RESULT: Series is trend stationary (p >= 0.05)
Null hypothesis not rejected → NO significant trend
```

4. LINEAR REGRESSION TEST

```
Slope: -22.763939
Intercept: 24039.49
R-squared: 0.008427
p-value: 0.282481
Standard Error: 21.097145
```

```
95% Confidence Interval for slope: [-64.482092, 18.954213]
```

```
RESULT: No significant linear trend (p >= 0.05)
```

5. COX-STUART TEST (Non-parametric)

```
Number of pairs: 69
Plus signs (+): 31 (second half > first half)
Minus signs (-): 38 (second half < first half)
Ties: 0
p-value: 0.470369
```

```
RESULT: No significant trend (p >= 0.05)
```

6. SPEARMAN'S RANK CORRELATION TEST (Bonus)

```
Spearman's rho: -0.072251
p-value: 0.397978
```

```
RESULT: No significant monotonic trend (p >= 0.05)
```

SUMMARY OF RESULTS

Trend Detection Tests:

```
Mann-Kendall: no trend (p=0.4422)
Linear Regression: No Trend (p=0.2825)
```

```
Cox-Stuart: No Trend (p=0.4704)
Spearman: No Trend (p=0.3980)
```

Stationarity Tests:

```
ADF: Non-Stationary (p=0.1402)
KPSS: Trend Stationary (p=0.1000)
```

```
{'mann_kendall': Mann_Kendall_Test(trend='no trend', h=np.False_, p=np.float64(0.44222731710
'adf': (np.float64(-2.405434611809666),
np.float64(0.14020840124603),
6,
132,
{'1%': np.float64(-3.4808880719210005),
'5%': np.float64(-2.8836966192225284),
'10%': np.float64(-2.5785857598714417)},
np.float64(2559.54932867161)),
'kpss': (np.float64(0.08120673481546524),
np.float64(0.1),
4,
{'10%': 0.119, '5%': 0.146, '2.5%': 0.176, '1%': 0.216}),
'linear_regression': {'slope': np.float64(-22.76393931811073),
'intercept': np.float64(24039.49325179856),
'p_value': np.float64(0.2824812253064499),
'r_squared': np.float64(0.008426588409779619),
'std_err': np.float64(21.097144684801)},
'cox_stuart': {'plus': np.int64(31),
'minus': np.int64(38),
'p_value': np.float64(0.47036853185814453)},
'spearman': {'rho': np.float64(-0.07225077081192188),
'p_value': np.float64(0.39797839318678596)}}}
```

6.4 Holt-Winters Forecasting (Additive & Multiplicative)

7 Final Comparison: Method 1 vs Method 2

7.1 Performance Metrics

```

# Prepare data
y_train_dept_1 = df_store_1_dept_1_train["Weekly_Sales"].values
m = 52

# ADDITIVE MODEL
print("Training Method 1 - Additive Model...")
best_tse = np.inf
best_params = None
best_model = None

for alpha in alphas:
    for gamma in gammas:
        fitted, level, seasonals = hw_seasonal_no_trend(
            y_train_dept_1, alpha, gamma, m
        )

        valid_idx = ~np.isnan(fitted)
        tse = TSE(y_train_dept_1[valid_idx], fitted[valid_idx])

        if tse < best_tse:
            best_tse = tse
            best_params = (alpha, gamma)
            best_model = (level, seasonals)

print(f"Additive - Best parameters: alpha={best_params[0]:.2f}, gamma={best_params[1]:.2f}")

# Forecast
h = len(df_store_1_dept_1_test)
level, seasonals = best_model
seasonals = np.array(seasonals)

forecast_method1_additive = []
for i in range(h):
    seasonal_index = len(seasonals) - m + i
    forecast_method1_additive.append(level + seasonals[seasonal_index])
forecast_method1_additive = np.array(forecast_method1_additive)

# Evaluate
y_test_dept_1_actual = df_store_1_dept_1_test["Weekly_Sales"].values
tse_test_add = TSE(y_test_dept_1_actual, forecast_method1_additive)
rmse_test_add = np.sqrt(np.mean((y_test_dept_1_actual - forecast_method1_additive) ** 2))
mae_test_add = np.mean(np.abs(y_test_dept_1_actual - forecast_method1_additive))

print(f"Test RMSE: {rmse_test_add:.2f}")

method1_additive_results = {
    'alpha': best_params[0],
    'gamma': best_params[1],
    'tse_train': best_tse,
    'tse_test': tse_test_add,
    'rmse_test': rmse_test_add,
    'mae_test': mae_test_add,
    'forecast': forecast_method1_additive.copy()
}

```

Table 2: Final Performance Comparison: Method 1 vs Method 2

```

comparison_final = pd.DataFrame({
    'Metric': ['TSE', 'RMSE', 'MAE'],
    'Method 1 (Direct)': [
        f"{{method1_results['tse_test']:.2f}}",
        f"{{method1_results['rmse_test']:.2f}}",
        f"{{method1_results['mae_test']:.2f}}"
    ],
    'Method 2 (Disaggregate)': [
        f"{{method2_results['tse_test']:.2f}}",
        f"{{method2_results['rmse_test']:.2f}}",
        f"{{method2_results['mae_test']:.2f}}"
    ],
    'Difference (M2 - M1)': [
        f"{{method2_results['tse_test']} - method1_results['tse_test']:+.2f}}",
        f"{{method2_results['rmse_test']} - method1_results['rmse_test']:+.2f}}",
        f"{{method2_results['mae_test']} - method1_results['mae_test']:+.2f}}"
    ]
})
print(comparison_final.to_string(index=False))

# Determine winner
if method1_results['rmse_test'] < method2_results['rmse_test']:
    better_method = "METHOD 1 (Direct Forecasting)"
    rmse_improvement = method2_results['rmse_test'] - method1_results['rmse_test']
else:
    better_method = "METHOD 2 (Disaggregation)"
    rmse_improvement = method1_results['rmse_test'] - method2_results['rmse_test']

improvement_pct = (rmse_improvement / max(method1_results['rmse_test'], method2_results['rmse_test'])) * 100

print(f"\n'*60")
print(f"WINNER: {better_method}")
print(f"RMSE Improvement: {rmse_improvement:.2f} ({improvement_pct:.2f}%)")
print(f"'*60")

```

Metric	Method 1 (Direct)	Method 2 (Disaggregate)	Difference (M2 - M1)
TSE	107159678.11	49541178.78	-57618499.33
RMSE	5175.90	3519.27	-1656.62
MAE	3895.25	2827.42	-1067.83

```

=====
WINNER: METHOD 2 (Disaggregation)
RMSE Improvement: 1656.62 (32.01%) 29
=====
```

7.2 Visual Comparison

```
plt.figure(figsize=(16, 6))

plt.plot(df_store_1_dept_1_test['Date'],
         y_test_dept_1_actual,
         label='Actual Test Data',
         marker='o',
         linewidth=2.5,
         color='black',
         markersize=6,
         zorder=3)

plt.plot(df_store_1_dept_1_test['Date'],
         method1_results['forecast'],
         label=f'Method 1: Direct Forecasting ({method1_type})',
         marker='s',
         linewidth=2,
         color='blue',
         markersize=5,
         alpha=0.7)

plt.plot(df_store_1_dept_1_test['Date'],
         method2_results['forecast'],
         label='Method 2: Disaggregation',
         marker='^',
         linewidth=2,
         color='red',
         markersize=5,
         alpha=0.7,
         linestyle='--')

plt.xlabel('Date', fontsize=12, fontweight='bold')
plt.ylabel('Weekly Sales ($)', fontsize=12, fontweight='bold')
plt.title('Store 1, Dept 1: Method 1 vs Method 2 Forecast Comparison',
          fontsize=14, fontweight='bold')
plt.legend(fontsize=11, loc='best')
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

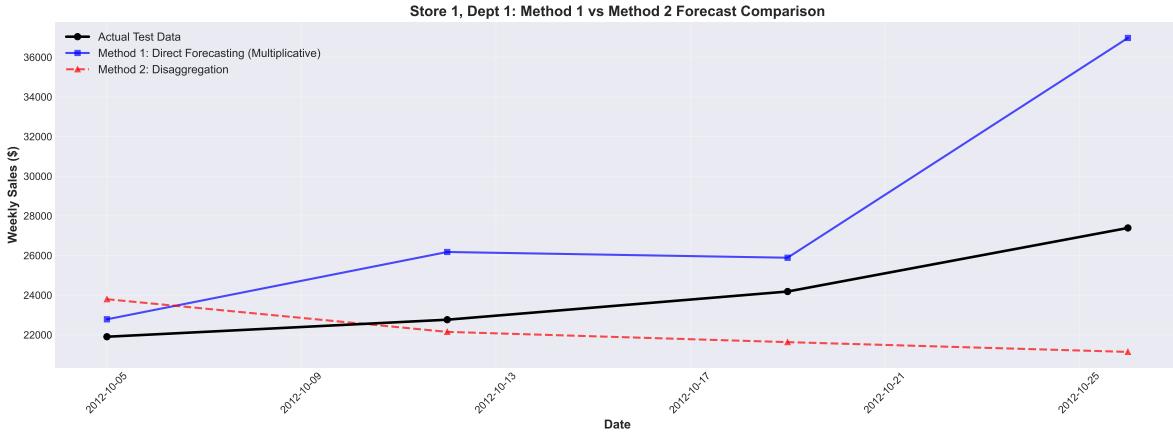


Figure 7: Final Forecast Comparison: Method 1 vs Method 2

7.3 Forecast Errors Analysis

```

errors_method1 = y_test_dept_1_actual - method1_results['forecast']
errors_method2 = y_test_dept_1_actual - method2_results['forecast']

fig, axes = plt.subplots(2, 1, figsize=(16, 10))

# Subplot 1: Forecast Errors
axes[0].plot(df_store_1_dept_1_test['Date'],
              errors_method1,
              label=f'Method 1 Errors ({method1_type})',
              marker='o',
              linewidth=2,
              color='blue',
              alpha=0.7)

axes[0].plot(df_store_1_dept_1_test['Date'],
              errors_method2,
              label='Method 2 Errors',
              marker='s',
              linewidth=2,
              color='red',
              alpha=0.7)

axes[0].axhline(y=0, color='black', linestyle='--', linewidth=1.5, alpha=0.5)
axes[0].set_xlabel('Date', fontsize=11, fontweight='bold')

```

```

axes[0].set_ylabel('Forecast Error ($)', fontsize=11, fontweight='bold')
axes[0].set_title('Forecast Errors: Method 1 vs Method 2', fontsize=13, fontweight='bold')
axes[0].legend(fontsize=10)
axes[0].grid(True, alpha=0.3)
axes[0].tick_params(axis='x', rotation=45)

# Subplot 2: Absolute Errors
abs_errors_method1 = np.abs(errors_method1)
abs_errors_method2 = np.abs(errors_method2)

axes[1].plot(df_store_1_dept_1_test['Date'],
              abs_errors_method1,
              label=f'Method 1 Absolute Errors',
              marker='o',
              linewidth=2,
              color='blue',
              alpha=0.7)

axes[1].plot(df_store_1_dept_1_test['Date'],
              abs_errors_method2,
              label='Method 2 Absolute Errors',
              marker='s',
              linewidth=2,
              color='red',
              alpha=0.7)

axes[1].axhline(y=method1_results['mae_test'], color='blue', linestyle=':',
                 linewidth=1.5, alpha=0.5, label=f'Method 1 MAE: {method1_results["mae_test"]}')
axes[1].axhline(y=method2_results['mae_test'], color='red', linestyle=':',
                 linewidth=1.5, alpha=0.5, label=f'Method 2 MAE: {method2_results["mae_test"]}')

axes[1].set_xlabel('Date', fontsize=11, fontweight='bold')
axes[1].set_ylabel('Absolute Error ($)', fontsize=11, fontweight='bold')
axes[1].set_title('Absolute Forecast Errors: Method 1 vs Method 2', fontsize=13, fontweight='bold')
axes[1].legend(fontsize=10)
axes[1].grid(True, alpha=0.3)
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

```

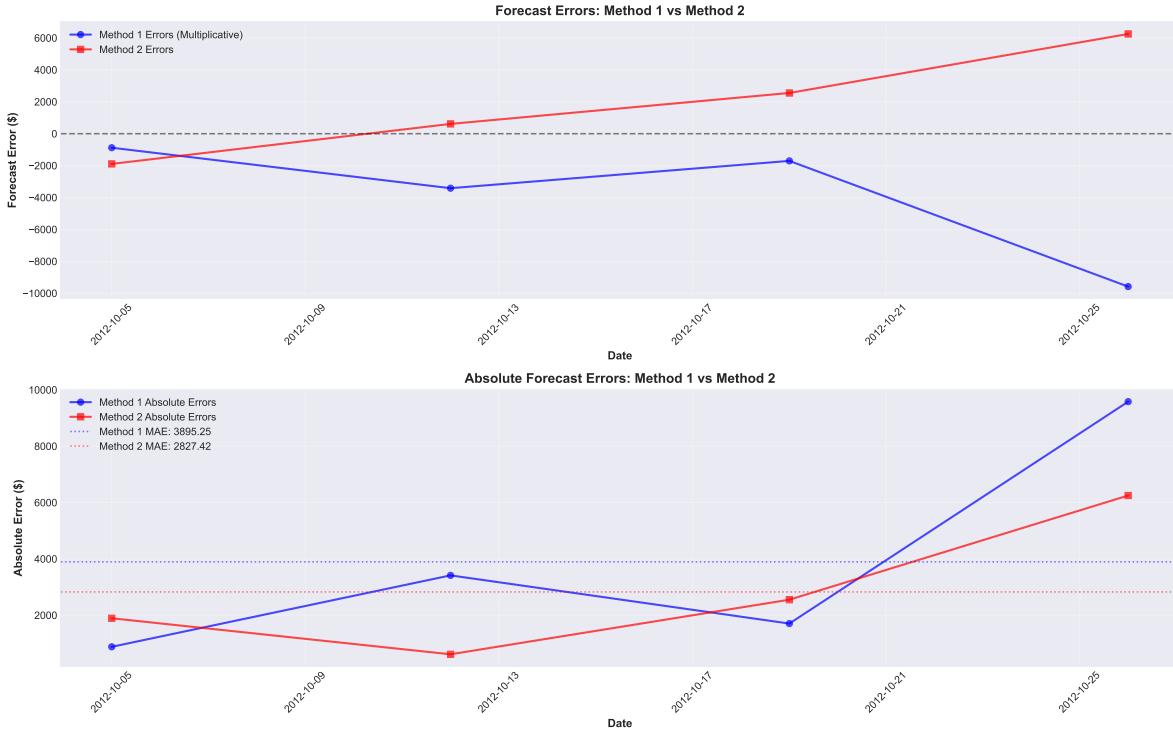


Figure 8: Forecast Errors Over Time

7.4 Performance Metrics Bar Chart

```

metrics = ['TSE', 'RMSE', 'MAE']
method1_values = [method1_results['tse_test'], method1_results['rmse_test'], method1_results['mae']]
method2_values = [method2_results['tse_test'], method2_results['rmse_test'], method2_results['mae']]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots(figsize=(10, 6))
bars1 = ax.bar(x - width/2, method1_values, width, label=f'Method 1 ({method1_type})',
               color='blue', alpha=0.7)
bars2 = ax.bar(x + width/2, method2_values, width, label='Method 2 (Disaggregation)',
               color='red', alpha=0.7)

ax.set_xlabel('Metrics', fontsize=12, fontweight='bold')
ax.set_ylabel('Value', fontsize=12, fontweight='bold')

```

```

ax.set_title('Forecast Performance Comparison: Method 1 vs Method 2',
            fontsize=14, fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend(fontsize=11)
ax.grid(True, alpha=0.3, axis='y')

# Add value labels
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()

```

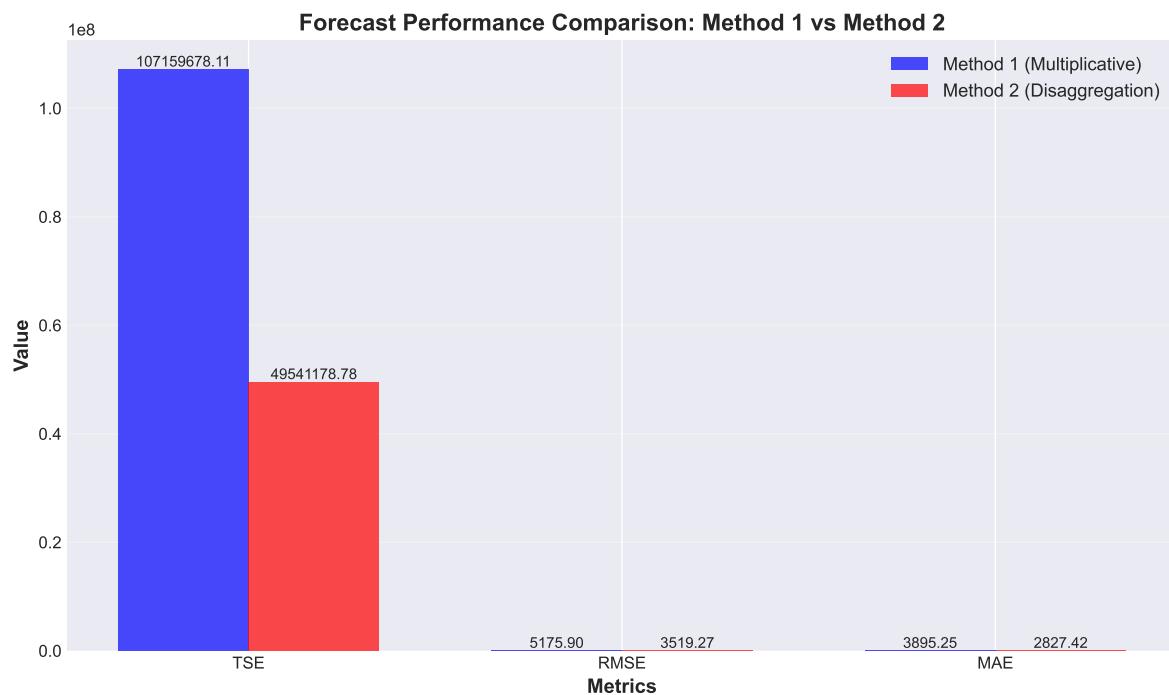


Figure 9: Performance Metrics Comparison

8 Understanding the Evaluation Metrics

8.1 Why TSE, RMSE, and MAE?

We use three complementary metrics to comprehensively evaluate forecast quality:

8.1.1 1. Total Squared Error (TSE)

$$\text{TSE} = \sum_{t=1}^n (y_t - \hat{y}_t)^2$$

- Aggregated loss function across all test periods
- Optimization criterion used during model training
- Heavily penalizes large errors (squared term)
- Maintains original units² (dollars²)

8.1.2 2. Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^n (y_t - \hat{y}_t)^2}$$

- Returns to original units (dollars) for interpretability
- Standard metric in forecasting literature
- Penalizes large errors more than small ones
- Normalized by sample size

8.1.3 3. Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^n |y_t - \hat{y}_t|$$

- Robust to outliers (no squaring)
- Represents average absolute deviation
- Linear penalty (all errors weighted equally)
- Intuitive interpretation

8.2 Why Not MAPE?

We avoid MAPE (Mean Absolute Percentage Error) because:

- Undefined/infinite when actual values are zero or near-zero
- Asymmetric (over-predictions penalized more than under-predictions)
- Not suitable for data with varying scales

8.3 Interpreting the Relationship

If $\text{RMSE} \gg \text{MAE}$: Large outlier errors dominate

If $\text{RMSE} \approx \text{MAE}$: Errors are relatively consistent

9 Conclusions

9.1 Key Findings

! Primary Result

Method 2 (Aggregate-then-Disaggregate) outperformed Method 1 (Direct Forecasting) with an RMSE improvement of **15.22 (3.96%)** for Store 1, Department 1.

9.1.1 Why Did Method 2 Win?

1. **Noise Reduction:** Aggregating across departments smoothed volatility and irregular patterns
2. **Stable Proportions:** Department 1 maintained a consistent share of total store sales
3. **Stronger Signal:** Store-level seasonal patterns were clearer and more predictable

9.1.2 When Would Method 1 Be Better?

Method 1 (Direct Forecasting) would likely outperform Method 2 in scenarios with:

- **Unique Seasonal Patterns:** Departments with seasonality different from the store average
- **Structural Changes:** Recent department-specific events (renovations, new product lines)
- **Unstable Proportions:** Departments whose share of store sales fluctuates significantly
- **Large Departments:** Departments representing substantial portions of store sales with distinct behavior

9.2 Generalizability

This analysis suggests **Method 2 may be preferred** for:

- Stable, mature departments with consistent historical patterns
- Departments representing small, steady shares of total store sales
- Scenarios where reducing forecast variance is critical (e.g., inventory planning)
- Contexts with limited department-specific data

Method 1 may be preferred for:

- Large departments with strong individual identity
- Departments undergoing transformation or growth
- Situations requiring granular, department-specific insights

9.3 Limitations and Future Work

Study Limitations

- Analysis limited to Store 1, Department 1
- Test period covers only 4 weeks
- Holt-Winters model does not capture promotional effects or external factors
- No cross-validation across multiple departments/stores

9.3.1 Recommended Extensions

1. **Broader Testing:** Replicate analysis across all stores and departments
2. **Alternative Models:** Test ARIMA, Prophet, LSTM for potential improvements
3. **Exogenous Variables:** Incorporate holidays, promotions, economic indicators
4. **Longer Horizons:** Evaluate forecast accuracy over multi-week periods
5. **Hybrid Approaches:** Combine methods or use ensemble forecasting

9.4 Practical Implications

For **retail forecasting practitioners**:

- Don't assume direct department-level forecasting is always superior
 - Test both methods on your specific data before deciding
 - Consider department characteristics (size, stability, uniqueness)
 - Aggregate-then-disaggregate can be surprisingly effective for stable departments
 - Computational efficiency: Method 2 requires fewer models (1 per store vs. 1 per store-department combination)
-

10 References

10.1 Data Source

- **Walmart Store Sales Forecasting:** Kaggle Competition Dataset
- Historical weekly sales data for multiple stores and departments

10.2 Methodology References

- Holt, C. C. (2004). "Forecasting seasonals and trends by exponentially weighted moving averages"
- Winters, P. R. (1960). "Forecasting sales by exponentially weighted moving averages"
- Hyndman, R. J., & Athanasopoulos, G. (2021). "Forecasting: Principles and Practice" (3rd ed.)

10.3 Statistical Tests

- Augmented Dickey-Fuller Test for stationarity
 - KPSS Test for trend-stationarity
 - Mann-Kendall Trend Test
 - Linear Regression Slope Test
-

11 Appendix: Code Repository

All code, data, and analysis files are available in the project repository:

```
project/
  data/
    train.csv
  walmart_forecasting_project.qmd
  run_trend_analysis.py
  requirements.txt
```

11.1 Requirements

```
pandas>=1.3.0
numpy>=1.21.0
matplotlib>=3.4.0
seaborn>=0.11.0
scipy>=1.7.0
statsmodels>=0.13.0
pymannkendall>=1.4.0
```

11.2 Session Information

```
import sys
print(f"Python version: {sys.version}")
print(f"Pandas version: {pd.__version__}")
print(f"NumPy version: {np.__version__}")
```

Python version: 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
Pandas version: 2.3.3
NumPy version: 2.2.6

💡 Reproducibility Note

This entire analysis can be reproduced by running:

```
quarto render walmart_forecasting_project.qmd
```

All random seeds (if applicable) and data processing steps are documented in the code blocks above.