In [1]:
```
# ======================
# AMAZON SALES PREDICTION USING LINEAR REGRESSION
# ======================

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# ---------- LOAD FILES ----------
category = pd.read_csv('category.csv')
inventory = pd.read_csv('inventory.csv')
customers = pd.read_csv('customers.csv')
order_items = pd.read_csv('order_items.csv')
orders = pd.read_csv('orders.csv')
payments = pd.read_csv('payments.csv')
products = pd.read_csv('products.csv')
sellers = pd.read_csv('sellers.csv')
shipping = pd.read_csv('shipping.csv')

# ---------- DATA PREPARATION ----------
print("Loading and merging datasets...")
df = pd.merge(order_items, orders, on='order_id', how='left')
df = pd.merge(df, products, on='product_id', how='left')
df = pd.merge(df, category, on='category_id', how='left')
df = pd.merge(df, customers, left_on='customer_id', right_on='Customer ID', how='left')
df = pd.merge(df, sellers, on='seller_id', how='left')
df = pd.merge(df, payments, on='order_id', how='left')

# Create target variable
df['total_sales'] = df['quantity'] * df['price_per_unit']
```

```
df['order_date'] = pd.to_datetime(df['order_date'])

# ---------- TIME SERIES AGGREGATION ----------
print("Aggregating data by month...")
monthly_sales = df.groupby(pd.Grouper(key='order_date', freq='M')).agg({
    'total_sales': 'sum',
    'quantity': 'sum',
    'order_id': 'nunique',
    'customer_id': 'nunique',
    'product_id': 'nunique',
    'seller_id': 'nunique'
}).reset_index()

monthly_sales.columns = ['date', 'monthly_sales', 'total_quantity', 'total_orders',
                         'unique_customers', 'unique_products', 'unique_sellers']
monthly_sales = monthly_sales.sort_values('date').reset_index(drop=True)

# Remove any months with missing sales data
monthly_sales = monthly_sales[monthly_sales['monthly_sales'].notna()]

# ---------- FEATURE ENGINEERING FOR LINEAR REGRESSION ----------
print("Creating time series features...")

# Basic time features
monthly_sales['time_index'] = range(1, len(monthly_sales) + 1)
monthly_sales['year'] = monthly_sales['date'].dt.year
monthly_sales['month'] = monthly_sales['date'].dt.month
monthly_sales['quarter'] = monthly_sales['date'].dt.quarter

# Lag features (critical for linear regression)
for lag in [1, 2, 3, 4, 5, 6, 12]:
    monthly_sales[f'sales_lag_{lag}'] = monthly_sales['monthly_sales'].shift(lag)
    monthly_sales[f'orders_lag_{lag}'] = monthly_sales['total_orders'].shift(lag)

# Rolling statistics (moving averages)
for window in [3, 6, 12]:
    monthly_sales[f'sales_ma_{window}'] = monthly_sales['monthly_sales'].rolling(window=window).mean()
    monthly_sales[f'orders_ma_{window}'] = monthly_sales['total_orders'].rolling(window=window).mean()

# Growth rates and trends
monthly_sales['sales_growth_1m'] = monthly_sales['monthly_sales'].pct_change(1)
```

```
monthly_sales['sales_growth_3m'] = monthly_sales['monthly_sales'].pct_change(3)
monthly_sales['orders_growth_1m'] = monthly_sales['total_orders'].pct_change(1)

# Seasonal features
monthly_sales['is_holiday_season'] = monthly_sales['month'].isin([11, 12]).astype(int)
monthly_sales['is_year_start'] = monthly_sales['month'].isin([1, 2]).astype(int)
monthly_sales['is_mid_year'] = monthly_sales['month'].isin([6, 7]).astype(int)

# Polynomial features for trend
monthly_sales['time_squared'] = monthly_sales['time_index'] ** 2
monthly_sales['time_cubed'] = monthly_sales['time_index'] ** 3

# Drop rows with NaN values (from lag features)
monthly_sales_clean = monthly_sales.dropna().copy()

print(f"Final dataset shape: {monthly_sales_clean.shape}")

# ---------- FEATURE SELECTION FOR LINEAR REGRESSION ----------
# Define features and target
feature_columns = [col for col in monthly_sales_clean.columns
                   if col not in ['date', 'monthly_sales'] and not col.startswith('sales_lag_')]

X = monthly_sales_clean[feature_columns]
y = monthly_sales_clean['monthly_sales']

print(f"Number of features: {len(feature_columns)}")

# Feature selection using correlation and statistical tests
correlation_with_target = X.corrwith(y).abs().sort_values(ascending=False)
high_corr_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()

print(f"Features with correlation > 0.1: {len(high_corr_features)}")

# Use only high correlation features for linear regression
X_selected = monthly_sales_clean[high_corr_features]

# ---------- DATA SCALING ----------
# Scale features for better linear regression performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_selected)
X_scaled = pd.DataFrame(X_scaled, columns=high_corr_features, index=monthly_sales_clean.index)
```

```
# ---------- TIME SERIES SPLIT ----------
# Use time-based split (more realistic than random split)
split_index = int(len(monthly_sales_clean) * 0.8)
X_train = X_scaled.iloc[:split_index]
X_test = X_scaled.iloc[split_index:]
y_train = y.iloc[:split_index]
y_test = y.iloc[split_index:]

print(f"Training set: {X_train.shape}, Test set: {X_test.shape}")

# ---------- LINEAR REGRESSION MODEL TRAINING ----------
print("\nTraining Linear Regression models...")

# Try different linear models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=1.0),
    'Lasso Regression': Lasso(alpha=0.1)
}

results = {}

for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    results[name] = {
        'model': model,
        'mae': mae,
        'rmse': rmse,
        'r2': r2,
        'predictions': y_pred
```

```
    }

    print(f"{name} - MAE: {mae:.2f}, RMSE: {rmse:.2f}, R²: {r2:.4f}")

# Select best model based on R² score
best_model_name = max(results.keys(), key=lambda x: results[x]['r2'])
best_model = results[best_model_name]['model']
print(f"\nBest model: {best_model_name}")

# ---------- FORECAST NEXT 6 MONTHS ----------
def forecast_linear_regression(model, scaler, last_data, feature_columns, n_months=6):
    """
    Forecast using linear regression model
    """
    forecasts = []
    current_data = last_data.copy()

    for i in range(n_months):
        # Prepare feature vector for next month
        future_features = {}

        # Update time-based features
        future_features['time_index'] = current_data['time_index'] + 1
        future_features['time_squared'] = future_features['time_index'] ** 2
        future_features['time_cubed'] = future_features['time_index'] ** 3

        # Update month and seasonal features
        next_month = current_data['month'] + 1
        if next_month > 12:
            next_month = 1
        future_features['month'] = next_month

        future_features['quarter'] = (next_month - 1) // 3 + 1
        future_features['year'] = current_data['year'] + (1 if next_month == 1 else 0)
        future_features['is_holiday_season'] = 1 if next_month in [11, 12] else 0
        future_features['is_year_start'] = 1 if next_month in [1, 2] else 0
        future_features['is_mid_year'] = 1 if next_month in [6, 7] else 0

        # Update lag features using recent forecasts
        for lag in [6, 5, 4, 3, 2, 1]:
            if f'sales_lag_{lag}' in feature_columns:
```

```
                if lag == 1 and forecasts:
                    future_features[f'sales_lag_{lag}'] = forecasts[-1]
                elif f'sales_lag_{lag-1}' in current_data:
                    future_features[f'sales_lag_{lag}'] = current_data[f'sales_lag_{lag-1}']

        # Update moving averages (simplified)
        for window in [3, 6, 12]:
            if f'sales_ma_{window}' in feature_columns:
                # Simplified: use recent values or forecasts
                recent_values = [current_data.get(f'sales_lag_{j}', current_data['monthly_sales'])
                               for j in range(1, min(window, 6)+1)]
                if forecasts:
                    recent_values = forecasts[-min(len(forecasts), window):] + recent_values[:(window - min(len(forecasts), wi
                future_features[f'sales_ma_{window}'] = np.mean(recent_values[:window])

        # Fill missing features with current data
        for col in feature_columns:
            if col not in future_features:
                future_features[col] = current_data[col] if col in current_data else 0

        # Create feature vector
        feature_vector = pd.DataFrame([future_features])[feature_columns]

        # Scale features
        feature_vector_scaled = scaler.transform(feature_vector)

        # Make prediction
        prediction = model.predict(feature_vector_scaled)[0]
        forecasts.append(max(prediction, 0))  # Ensure non-negative sales

        # Update current_data for next iteration
        current_data = future_features.copy()
        current_data['monthly_sales'] = prediction

    return forecasts

# Get the most recent data point
last_known_data = monthly_sales_clean.iloc[-1].copy()

# Generate forecasts
print("\nGenerating 6-month forecast...")
```

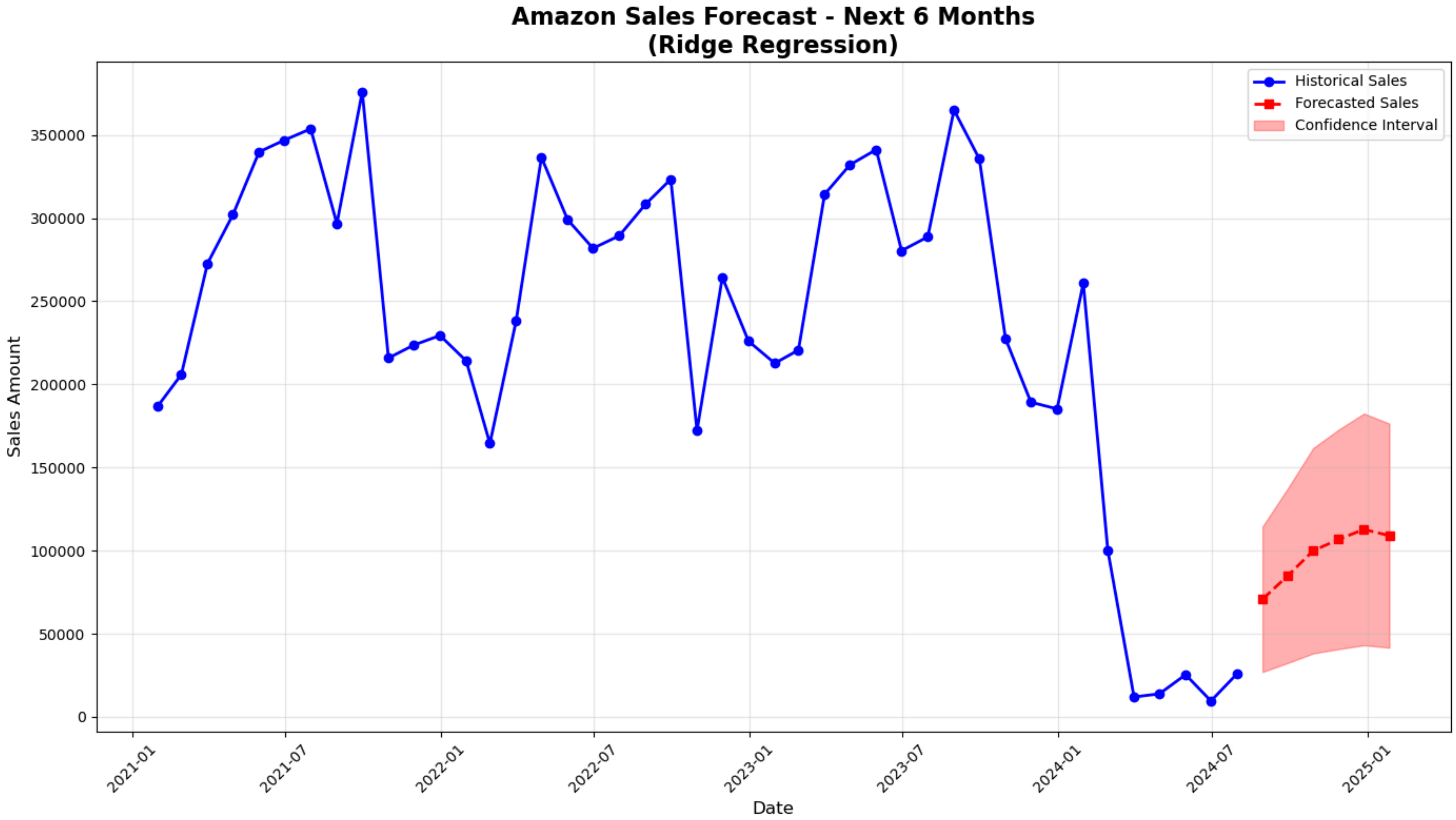
```
future_sales = forecast_linear_regression(best_model, scaler, last_known_data, high_corr_features, 6)

# ---------- CREATE FUTURE DATES ----------
last_date = monthly_sales_clean['date'].iloc[-1]
future_dates = [last_date + timedelta(days=30*i) for i in range(1, 7)]

# Create forecast dataframe
forecast_df = pd.DataFrame({
    'date': future_dates,
    'predicted_sales': future_sales,
    'type': 'forecast'
})

# Prepare historical data for plotting
historical_df = monthly_sales_clean[['date', 'monthly_sales']].copy()
historical_df['type'] = 'historical'
historical_df = historical_df.rename(columns={'monthly_sales': 'predicted_sales'})

# Combine historical and forecast data
combined_df = pd.concat([historical_df, forecast_df], ignore_index=True)

# ---------- VISUALIZE FORECAST ----------
plt.figure(figsize=(14, 8))

# Plot historical data
historical_data = combined_df[combined_df['type'] == 'historical']
forecast_data = combined_df[combined_df['type'] == 'forecast']

plt.plot(historical_data['date'], historical_data['predicted_sales'],
         label='Historical Sales', color='blue', linewidth=2, marker='o')

plt.plot(forecast_data['date'], forecast_data['predicted_sales'],
         label='Forecasted Sales', color='red', linewidth=2, linestyle='--', marker='s')

# Add confidence interval (simplified)
forecast_std = np.std(historical_data['predicted_sales'].pct_change().dropna())
confidence_upper = [x * (1 + 1.5 * forecast_std) for x in future_sales]
confidence_lower = [x * (1 - 1.5 * forecast_std) for x in future_sales]

plt.fill_between(forecast_data['date'], confidence_lower, confidence_upper,
                 alpha=0.3, color='red', label='Confidence Interval')
```

```
plt.title(f'Amazon Sales Forecast - Next 6 Months\n({best_model_name})', fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('Sales Amount', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# ---------- DISPLAY FORECAST RESULTS ----------
print("\n" + "="*60)
print("LINEAR REGRESSION - SALES FORECAST FOR NEXT 6 MONTHS")
print("="*60)

forecast_details = pd.DataFrame({
    'Month': [date.strftime('%B %Y') for date in future_dates],
    'Predicted Sales': [f"${sales:,.2f}" for sales in future_sales],
    'Monthly Growth': ['-' if i == 0 else f"{((future_sales[i] - future_sales[i-1])/future_sales[i-1]*100):+.1f}%"
                      for i in range(len(future_sales))],
    'Vs Last Year': [f"{((future_sales[i] - last_known_data.get(f'sales_lag_12', future_sales[i]))/last_known_data.get(f'sales
                    if f'sales_lag_12' in last_known_data else '-' for i in range(len(future_sales))]
})

print(forecast_details.to_string(index=False))

# Calculate summary statistics
total_forecast_sales = sum(future_sales)
avg_monthly_forecast = np.mean(future_sales)
growth_rate_6m = ((future_sales[-1] - future_sales[0]) / future_sales[0]) * 100

print(f"\nForecast Summary:")
print(f"Total Predicted Sales (6 months): ${total_forecast_sales:,.2f}")
print(f"Average Monthly Sales: ${avg_monthly_forecast:,.2f}")
print(f"6-Month Growth Rate: {growth_rate_6m:+.2f}%")

# ---------- MODEL INTERPRETATION ----------
if hasattr(best_model, 'coef_'):
    print("\n" + "="*50)
    print("MODEL INTERPRETATION (Top 10 Features)")
    print("="*50)
```

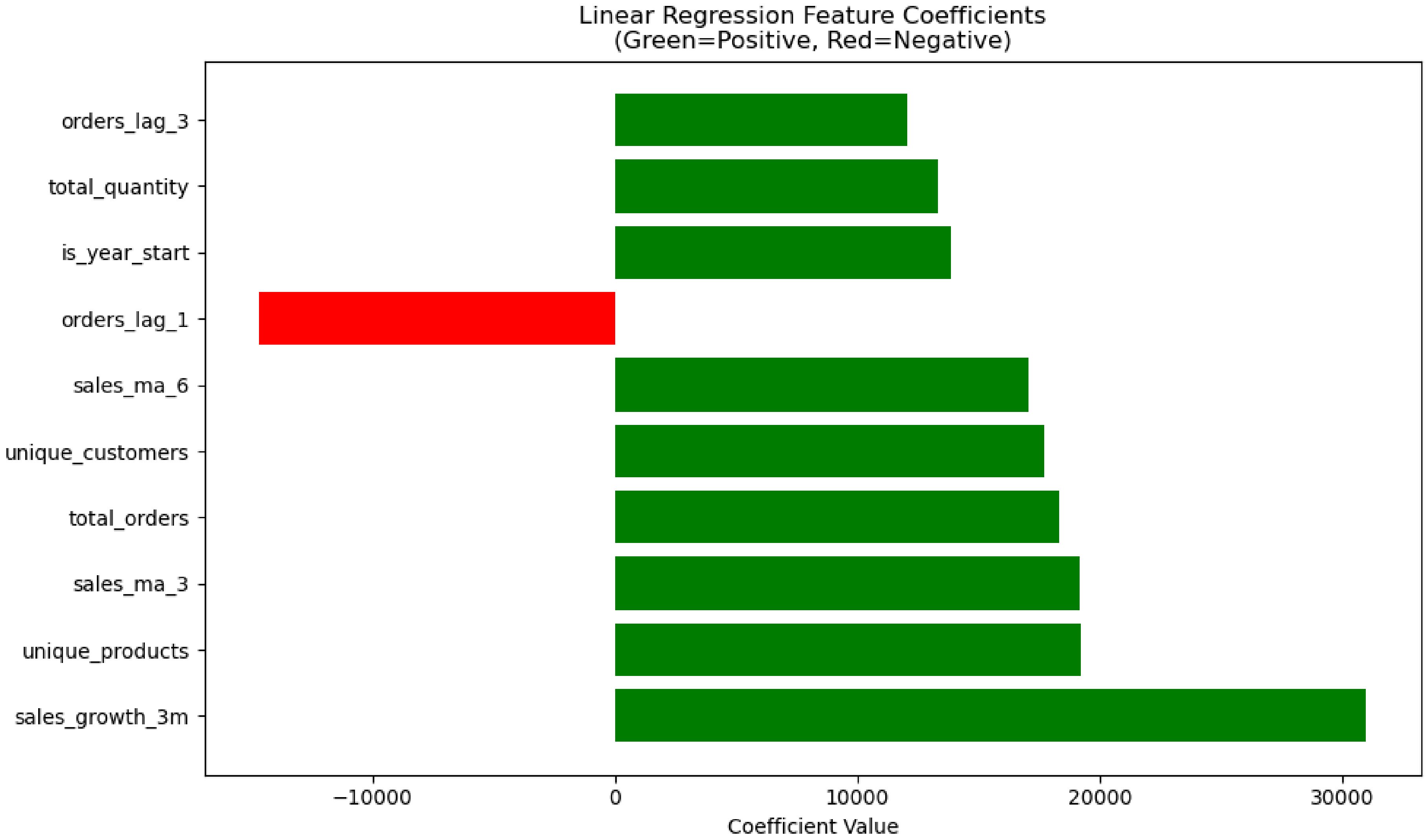
```
    feature_importance = pd.DataFrame({
        'Feature': high_corr_features,
        'Coefficient': best_model.coef_,
        'Abs_Coefficient': np.abs(best_model.coef_)
    }).sort_values('Abs_Coefficient', ascending=False)

    print(feature_importance.head(10).to_string(index=False))

    # Plot feature coefficients
    plt.figure(figsize=(10, 6))
    top_features = feature_importance.head(10)
    colors = ['green' if x > 0 else 'red' for x in top_features['Coefficient']]
    plt.barh(top_features['Feature'], top_features['Coefficient'], color=colors)
    plt.xlabel('Coefficient Value')
    plt.title('Linear Regression Feature Coefficients\n(Green=Positive, Red=Negative)')
    plt.tight_layout()
    plt.show()

# ---------- RESIDUAL ANALYSIS ----------
print("\n" + "="*50)
print("RESIDUAL ANALYSIS")
print("="*50)

y_pred_test = best_model.predict(X_test)
residuals = y_test - y_pred_test

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.scatter(y_pred_test, residuals, alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted')

plt.subplot(1, 2, 2)
plt.hist(residuals, bins=20, edgecolor='black', alpha=0.7)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals')
```

```
plt.tight_layout()
plt.show()

# Check residual statistics
print(f"Residual Mean: {residuals.mean():.2f}")
print(f"Residual Std: {residuals.std():.2f}")
print(f"Normality check: If residuals are centered around 0 with constant variance, model assumptions are reasonable")

# ---------- SAVE RESULTS ----------
forecast_output = pd.DataFrame({
    'date': future_dates,
    'predicted_sales': future_sales,
    'confidence_lower': confidence_lower,
    'confidence_upper': confidence_upper,
    'model_used': best_model_name
})

forecast_output.to_csv('amazon_sales_forecast_linear_regression.csv', index=False)
print(f"\nForecast results saved to 'amazon_sales_forecast_linear_regression.csv'")
```

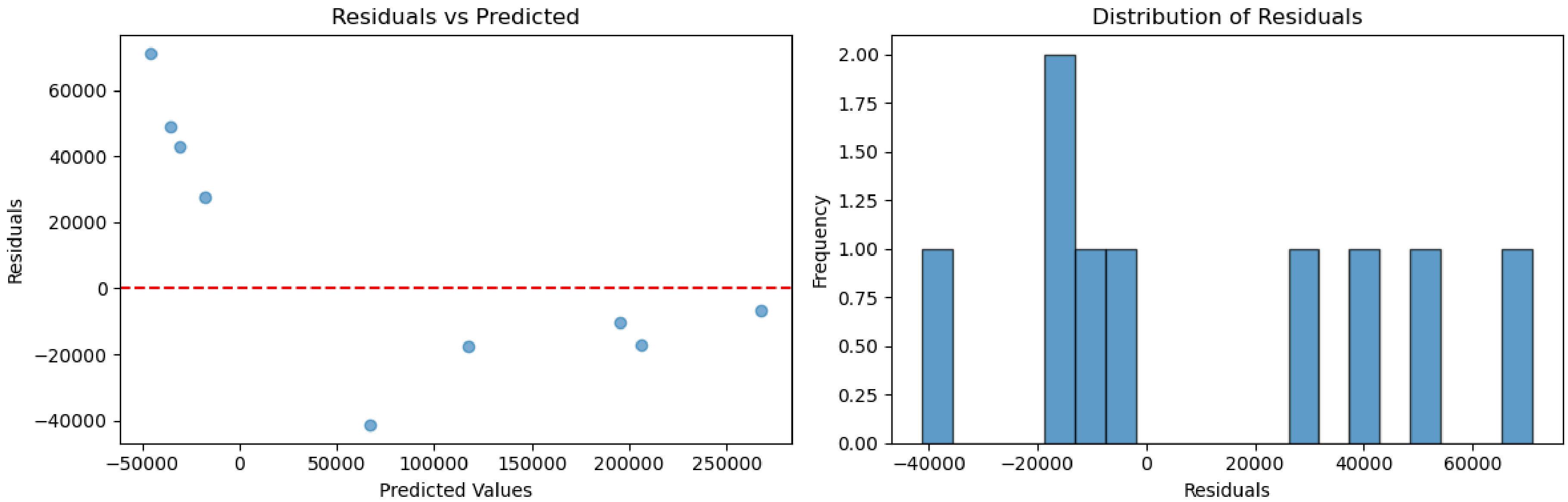
```
Loading and merging datasets...
Aggregating data by month...
Creating time series features...
Final dataset shape: (43, 39)
Number of features: 30
Features with correlation > 0.1: 24
Training set: (34, 24), Test set: (9, 24)

Training Linear Regression models...
Training Linear Regression...
Linear Regression - MAE: 108291.27, RMSE: 129529.18, R²: -1.0172
Training Ridge Regression...
Ridge Regression - MAE: 31467.35, RMSE: 37245.79, R²: 0.8332
Training Lasso Regression...
Lasso Regression - MAE: 100042.35, RMSE: 120310.16, R²: -0.7402

Best model: Ridge Regression

Generating 6-month forecast...
```


Amazon Sales Forecast - Next 6 Months
(Ridge Regression)
Historical Sales
Forecasted Sales
Confidence Interval
Sales Amount
0
50000
100000
150000
200000
250000
300000
350000
Date
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
2025-01

```
============================================================
LINEAR REGRESSION - SALES FORECAST FOR NEXT 6 MONTHS
============================================================
         Month Predicted Sales Monthly Growth Vs Last Year
   August 2024      $70,751.76              -       -75.5%
September 2024      $84,985.49         +20.1%       -70.6%
  October 2024      $99,956.28         +17.6%       -65.4%
 November 2024     $106,768.33          +6.8%       -63.0%
 December 2024     $112,772.06          +5.6%       -60.9%
  January 2025     $109,026.14          -3.3%       -62.2%

Forecast Summary:
Total Predicted Sales (6 months): $584,260.05
Average Monthly Sales: $97,376.68
6-Month Growth Rate: +54.10%

==================================================
MODEL INTERPRETATION (Top 10 Features)
==================================================
         Feature   Coefficient  Abs_Coefficient
 sales_growth_3m  30959.297676     30959.297676
 unique_products  19190.980526     19190.980526
      sales_ma_3  19164.778632     19164.778632
    total_orders  18340.616279     18340.616279
unique_customers  17696.125622     17696.125622
      sales_ma_6  17039.196286     17039.196286
    orders_lag_1 -14643.640520     14643.640520
   is_year_start  13878.956170     13878.956170
  total_quantity  13312.849355     13312.849355
    orders_lag_3  12046.343361     12046.343361
```




```
==================================================
RESIDUAL ANALYSIS
==================================================
```




```
Residual Mean: 10848.09
Residual Std: 37792.38
Normality check: If residuals are centered around 0 with constant variance, model assumptions are reasonable

Forecast results saved to 'amazon_sales_forecast_linear_regression.csv'
```

In [ ]: