D. Saul Jameson

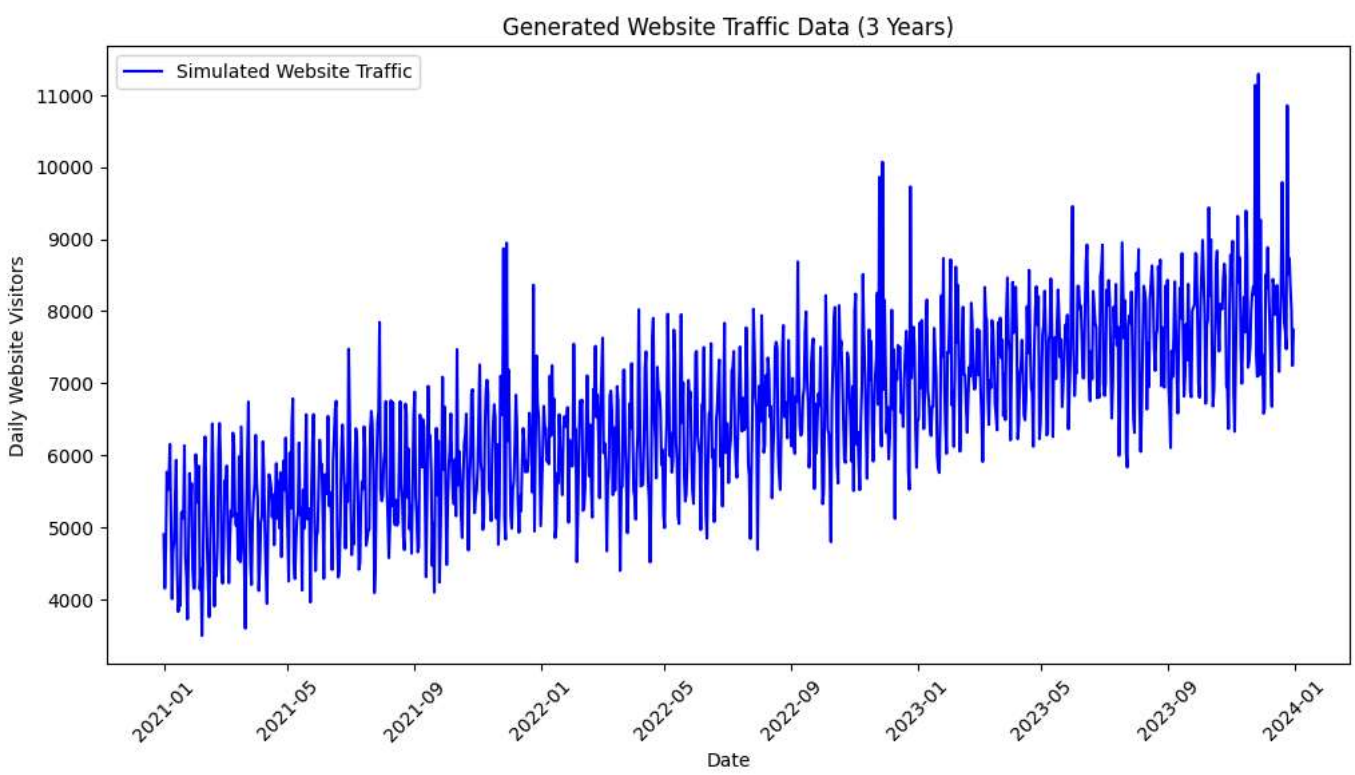
https://www.linkedin.com/in/dsauljameson/

```
# Install necessary libraries
!pip install pandas numpy matplotlib
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.23.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima import auto_arima


# Set random seed for reproducibility
np.random.seed(42)


# Function to generate 3 years of website traffic data
def generate_website_traffic(start_date="2021-01-01", days=3*365):
    date_range = pd.date_range(start=start_date, periods=days, freq='D')

    # Base traffic with gradual upward trend
    base_traffic = 5000 + np.linspace(0, 3000, days)  # Traffic increases over time

    # Weekly seasonality (higher weekdays, lower weekends)
    weekly_pattern = 800 * np.sin(2 * np.pi * date_range.dayofweek / 7)

    # Holiday spikes (e.g., Black Friday, Cyber Monday, Christmas)
    holiday_dates = [
        "2021-11-26", "2021-11-29", "2021-12-25", "2022-11-25", "2022-11-28", "2022-12-25",
        "2023-11-24", "2023-11-27", "2023-12-25"
    ]
    holiday_spikes = np.where(date_range.isin(pd.to_datetime(holiday_dates)), 3000, 0)

    # Random fluctuations (marketing campaigns, server issues, viral trends)
    random_noise = np.random.normal(scale=500, size=days)

    # Final traffic values
    traffic = base_traffic + weekly_pattern + holiday_spikes + random_noise

    # Ensure no negative values
    traffic = np.maximum(traffic, 3000).astype(int)

    # Create DataFrame
    df = pd.DataFrame({
        "date": date_range,
        "traffic": traffic
    })

    return df


# Generate the dataset
df = generate_website_traffic()
```

```
# Plot the data to check seasonality & trends
plt.figure(figsize=(12, 6))
plt.plot(df["date"], df["traffic"], label="Simulated Website Traffic", color='blue')
plt.xlabel("Date")
plt.ylabel("Daily Website Visitors")
plt.title("Generated Website Traffic Data (3 Years)")
plt.legend()
plt.xticks(rotation=45)
plt.show()
```




```
# Show first few rows
print(df.head())
```

```
        date  traffic
0 2021-01-01     4901
1 2021-01-02     4153
2 2021-01-03     4703
3 2021-01-04     5769
4 2021-01-05     5519
```

```
# Split data: 2.5 years for training, last 6 months for testing
train_size = int(len(df) * 2.5 / 3)  # 2.5 years of data
train_df = df.iloc[:train_size]
test_df = df.iloc[train_size:]


# Print dataset split information
print(f"Train set: {train_df.shape[0]} days ({train_df['date'].min()} to {train_df['date'].max()})")
print(f"Test set: {test_df.shape[0]} days ({test_df['date'].min()} to {test_df['date'].max()})")
```

```
Train set: 912 days (2021-01-01 00:00:00 to 2023-07-01 00:00:00)
Test set: 183 days (2023-07-02 00:00:00 to 2023-12-31 00:00:00)
```

```
# Automatically find the best SARIMA parameters
best_sarima = auto_arima(
    train_df["traffic"],
    seasonal=True,
    m=7,  # Weekly seasonality (7-day cycle)
    stepwise=True,
    suppress_warnings=True
)

# Print model summary
print(best_sarima.summary())
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all
  warnings.warn(
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:                  912
Model:             SARIMAX(5, 1, 5)x(1, 0, [], 7)   Log Likelihood               -7077.807
Date:                            Sun, 30 Mar 2025   AIC                          14181.614
Time:                                    13:04:57   BIC                          14244.203
Sample:                                         0   HQIC                         14205.509
                                            - 912
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     19.7838      4.647      4.257      0.000      10.676      28.891
ar.L1         -0.7647      0.059    -12.947      0.000      -0.880      -0.649
ar.L2          0.3257      0.037      8.711      0.000       0.252       0.399
ar.L3         -0.6609      0.023    -29.011      0.000      -0.706      -0.616
ar.L4         -1.0244      0.045    -22.556      0.000      -1.113      -0.935
ar.L5         -0.1259      0.049     -2.543      0.011      -0.223      -0.029
ma.L1         -0.2552      0.039     -6.534      0.000      -0.332      -0.179
ma.L2         -1.0732      0.041    -25.951      0.000      -1.154      -0.992
ma.L3          1.0772      0.029     37.218      0.000       1.020       1.134
ma.L4          0.2806      0.041      6.918      0.000       0.201       0.360
ma.L5         -0.8748      0.037    -23.949      0.000      -0.946      -0.803
ar.S.L7       -0.1386      0.049     -2.838      0.005      -0.234      -0.043
sigma2       3.87e+05   1.48e+04     26.095      0.000    3.58e+05    4.16e+05
===================================================================================
Ljung-Box (L1) (Q):                   1.13   Jarque-Bera (JB):              1000.45
Prob(Q):                              0.29   Prob(JB):                         0.00
Heteroskedasticity (H):               1.32   Skew:                             1.08
Prob(H) (two-sided):                  0.02   Kurtosis:                         7.65
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Takeaways

SARIMA has captured daily and weekly trends, but recent days matter more than seasonal weekly patterns.

The model is adjusting for noise well, meaning predictions are fairly stable.

Forecast confidence is good for 14-day predictions, but traffic spikes might not be fully captured.

Residuals (errors) are not normally distributed, so the model might struggle with sudden traffic shifts.

Full Analysis

Plain English Summary of Your SARIMA Model Output 1 Overall Model Summary You're forecasting website traffic using a SARIMA model.

The model has 5 autoregressive (AR) terms, 5 moving average (MA) terms, and 1 seasonal autoregressive (SAR) term with a 7-day seasonality (weekly cycle).

Your model is using 912 days (2.5 years) of data for training.

2 Model Fit & Selection Criteria Log Likelihood (-7077.807): How well the model explains the data (higher is better, but negative is expected in time series).

AIC (14181.614): Lower is better; used to compare models.

BIC (14244.203): Similar to AIC but penalizes complex models more.

HQIC (14205.509): Another model selection metric (lower is better).

This suggests the model is statistically well-fitted, but AIC/BIC can be used to compare against other models if needed.

3 Key Coefficients & What They Mean Autoregressive (AR) Terms → How past traffic affects future predictions. ar.L1 (-0.7647): Yesterday's traffic has a strong negative impact on today's prediction.

ar.L2 (0.3257): Two days ago's traffic has a positive impact.

ar.L3 (-0.6609), ar.L4 (-1.0244): These indicate deeper memory effects.

ar.L5 (-0.1259): The effect of 5-day lag is small but significant.

Moving Average (MA) Terms → Adjusts for noise in the data. ma.L1 (-0.2552): The model adjusts yesterday's noise downward.

ma.L2 (-1.0732), ma.L3 (1.0772), ma.L4 (0.2806), ma.L5 (-0.8748): The pattern of noise correction is strong and complex.

Seasonal Term (7-day lag) ar.S.L7 (-0.1386): Traffic seven days ago (same weekday last week) has a small negative effect on today's prediction.

This means the weekly cycle is present, but it is not the strongest factor. Recent days (1-4 days ago) have a bigger influence than weekly patterns. The moving average terms adjust for fluctuations, meaning website traffic has short-term noise that the model tries to correct.

4 Model Uncertainty (Error Terms) sigma2 (387,000): This is the estimated variance of errors (how much the forecast fluctuates). Large values mean predictions have higher uncertainty.

5 Model Diagnostics & Quality Ljung-Box Test (L1) Q = 1.13, Prob(Q) = 0.29: No significant autocorrelation left, which indicates a good fit.

Jarque-Bera (JB) Test JB = 1000.45, Prob(JB) = 0.00: Residuals (errors) are not normally distributed, meaning predictions might not be perfect.

Heteroskedasticity Test (H) H = 1.32, Prob(H) = 0.02: There is some variation in volatility, meaning traffic fluctuates unpredictably at times.

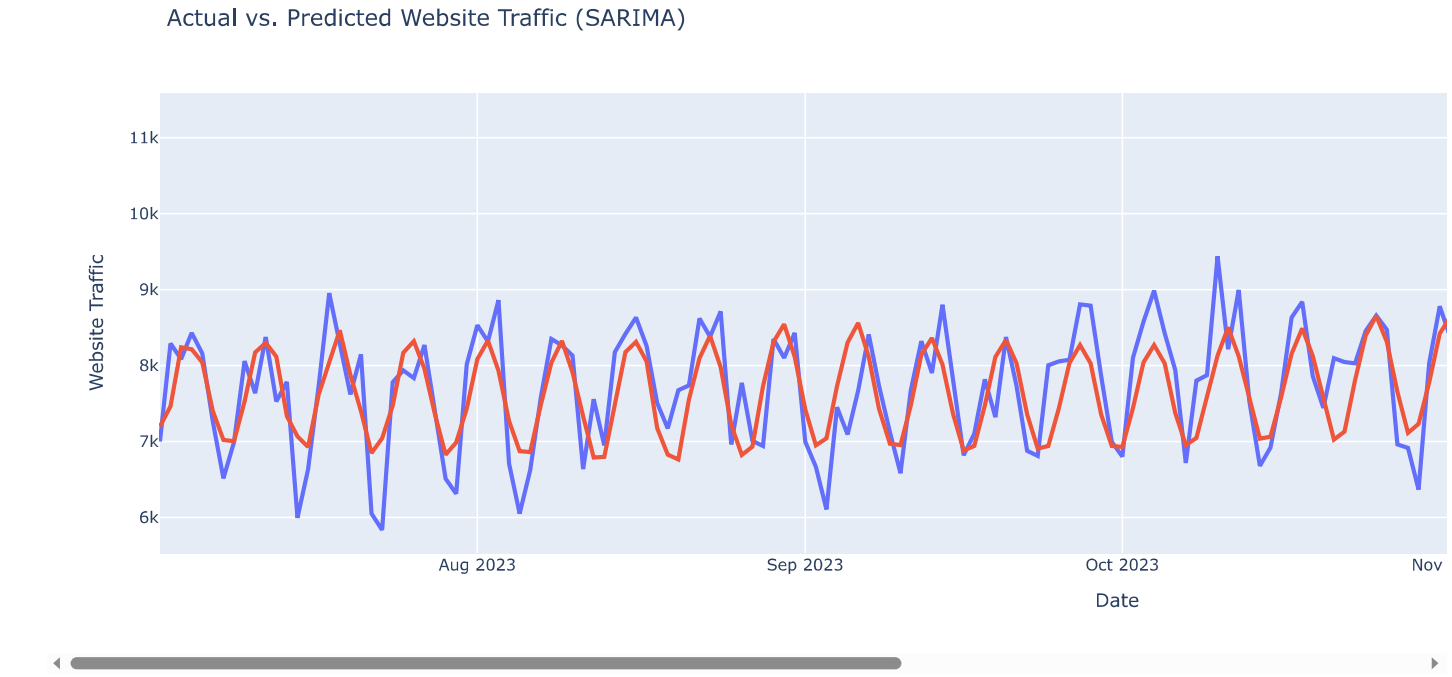
This suggests the model fits well but could struggle with big traffic spikes, such as viral trends or sudden crashes. Short-term forecasts (14 days) should be good, but long-term uncertainty is high.

```
# Initialize lists to store predictions and actual values
predictions = []
actuals = []
dates = []

# Start with the training data
history = train_df["traffic"].tolist()

# Perform rolling 14-day forecasts
for i in range(0, len(test_df) - 14, 14):
    # Train SARIMA model on the current history
    model = SARIMAX(
        history,
        order=best_sarima.order,
        seasonal_order=best_sarima.seasonal_order
    ).fit()

    # Forecast next 14 days
    forecast = model.forecast(steps=14)

    # Store results
    pred_dates = test_df.iloc[i:i+14]["date"].values
    actual_values = test_df.iloc[i:i+14]["traffic"].values

    predictions.extend(forecast)
    actuals.extend(actual_values)
    dates.extend(pred_dates)

    # Update history with actual values (simulating real-time forecasting)
    history.extend(actual_values)

# Create forecast DataFrame
forecast_df = pd.DataFrame({
    "date": dates,
    "predicted_traffic": predictions,
```

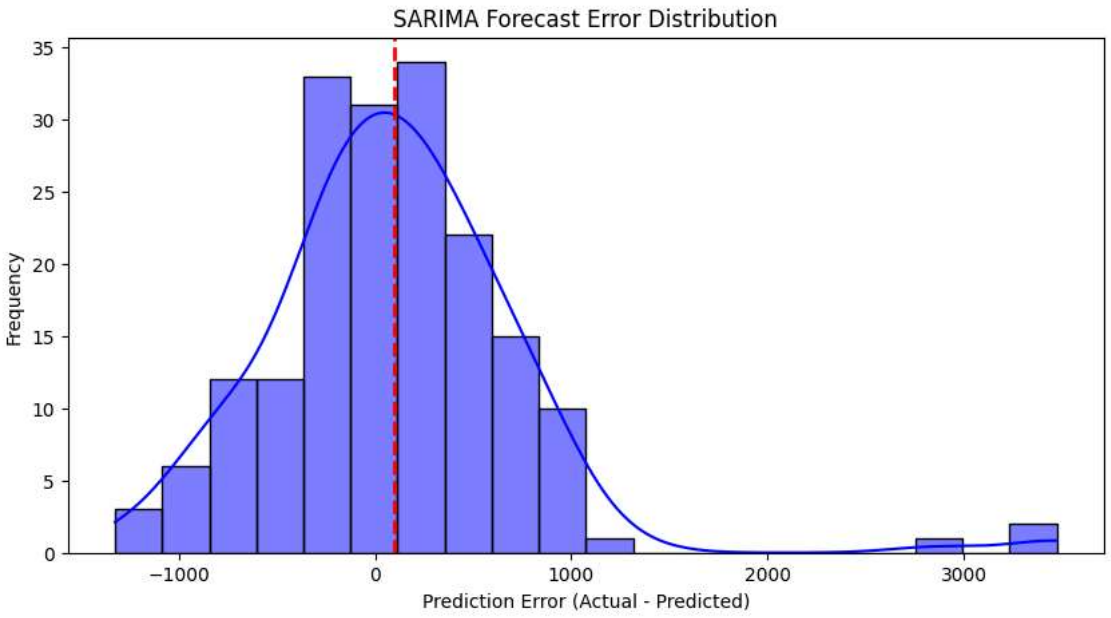
```
    "actual_traffic": actuals
})

# Print first few rows of the forecast results
print(forecast_df.head())
```

Show hidden output

```
fig = px.line(forecast_df, x="date", y=["actual_traffic", "predicted_traffic"],
              labels={"value": "Website Traffic", "date": "Date"},
              title="Actual vs. Predicted Website Traffic (SARIMA)")

fig.update_traces(line=dict(width=3))
fig.show()
```




```
# Calculate error
forecast_df["error"] = forecast_df["actual_traffic"] - forecast_df["predicted_traffic"]

# Create Seaborn distribution plot
plt.figure(figsize=(10,5))
sns.histplot(forecast_df["error"], bins=20, kde=True, color="blue")
plt.axvline(forecast_df["error"].mean(), color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Prediction Error (Actual - Predicted)")
plt.ylabel("Frequency")
plt.title("SARIMA Forecast Error Distribution")
plt.show()
```




```
# Calculate key statistics
mae = round(abs(forecast_df["error"]).mean(), 2)
mape = round((abs(forecast_df["error"]) / forecast_df["actual_traffic"]).mean() * 100, 2)
ci_low, ci_high = round(forecast_df["predicted_traffic"].quantile(0.025), 2), round(forecast_df["predicted_traffic"].quantile(0.975), 2)

# Print summary
print(f"🔷 Forecast Summary:")
print(f"   - Mean Absolute Error (MAE): {mae} visitors/day")
print(f"   - Mean Absolute Percentage Error (MAPE): {mape}%")
print(f"   - 95% Confidence Interval: {ci_low} to {ci_high} visitors/day")
```

```
🔷 Forecast Summary:
   - Mean Absolute Error (MAE): 462.96 visitors/day
   - Mean Absolute Percentage Error (MAPE): 5.83%
   - 95% Confidence Interval: 6826.18 to 8759.36 visitors/day
```

```
import plotly.io as pio

# Get the raw HTML code for embedding
html_code = pio.to_html(fig, full_html=False, include_plotlyjs='cdn')

# Print the HTML code
print(html_code)
```

```
<div>                        <script type="text/javascript">window.PlotlyConfig = {MathJaxConfig: 'local'};</script>
        <script charset="utf-8" src="https://cdn.plot.ly/plotly-2.35.2.min.js"></script>                <div id="b615c83b-f970-4574-abb6
```