

# Time Series Homework 2

Edward Anderson

**Honor Pledge:** On my honor as a student, I have neither given nor received any unauthorized aid on this assignment. - Edward Anderson

```
In [77]: # Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
```

## QUESTION 1: Show a data description.

1. Provide a summary of the data set to include descriptive statistics.
2. Plot the time series.
3. (optional) Provide any additional plots that may show important characteristics of your data

```
In [78]: # Load & Preprocess Data
sales = pd.read_csv('data/sales_data.csv')
sales['date'] = pd.to_datetime(sales['date'], errors='coerce')
sales['qty'] = pd.to_numeric(sales['qty'], errors='coerce')
sales['val'] = pd.to_numeric(sales['val'], errors='coerce')
# Subset and Group
sales = sales[~sales['world'].isin(['gr', 'na'])]
sales = sales.groupby('date').sum().reset_index().drop(columns=['world'])
sales.sort_values('date', inplace=True)
# Correct sales anomaly
sales.rename(columns={'qty' : 'order_quantity', 'val' : 'order_value'}, inplace=True)
sales.loc[sales['order_quantity'].idxmax(), 'order_quantity'] = sales.loc[sales['order_quantity'].idxmax(), 'order_value']
sales.loc[sales['order_value'].idxmax(), 'order_value'] = sales.loc[sales['order_value'].idxmax(), 'order_quantity']
# Group by Week
sales['date'] = sales['date'].dt.to_period('W').apply(lambda r: r.start_time)
sales = sales.groupby('date').sum().reset_index()
print(sales.head())
```

	date	order_quantity	order_value
0	2018-01-01	474.0	1147036.0
1	2018-01-08	492.0	1153592.0
2	2018-01-15	852.0	2301456.0
3	2018-01-22	756.0	1933800.0
4	2018-01-29	822.0	2020556.0

```
In [79]: # Descriptive Stats and EDA
print("Basic Descriptive Statistics:")
sales.describe().style.format("{:.0f}", subset=['order_quantity', 'order_value'])
```

Basic Descriptive Statistics:

```
Out[79]:
```

	date	order_quantity	order_value
<b>count</b>	419	419	419
<b>mean</b>	2022-01-09 04:21:11.599045120	2,914	6,472,871
<b>min</b>	2018-01-01 00:00:00	474	1,147,036
<b>25%</b>	2020-01-02 12:00:00	1,530	3,588,158
<b>50%</b>	2022-01-10 00:00:00	2,508	5,871,536
<b>75%</b>	2024-01-18 12:00:00	3,783	8,573,088
<b>max</b>	2026-01-19 00:00:00	16,116	21,200,652
<b>std</b>	nan	2,007	3,732,091

The table above shows the descriptive stats of these data. For this assignment, I will be modeling order quantity. These data are weekly, so the date represents the first day of each week. The mean order quantity is ~2900 units, the max is ~16,000, and the minimum is 474. These data are highly variable with a standard deviation of ~2,000 units. Finally, we have plenty of data points (419 weeks) which is enough to detect patterns, trends, and seasonality in the data.

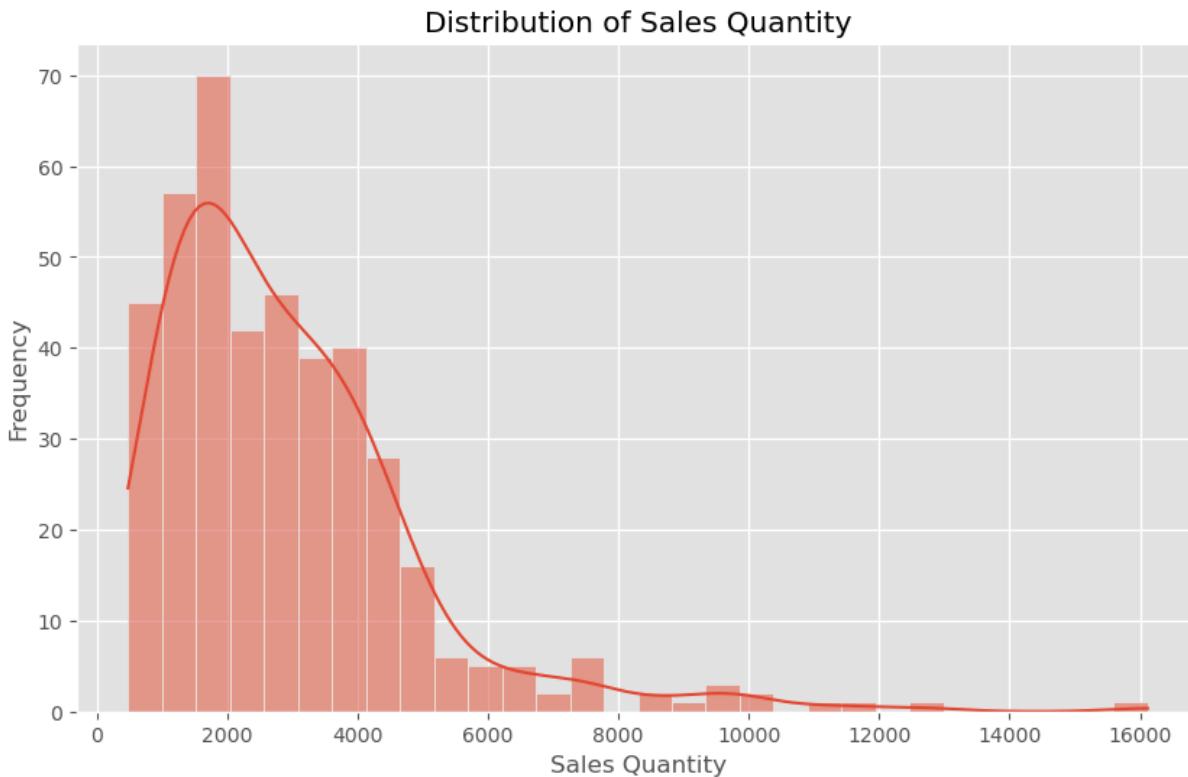
```
In [80]: # Plot Sales Qty Over Time
plot = px.line(sales, x='date', y='order_quantity',
               hover_data=['order_quantity'],
               labels={'order_quantity':'Sales Quantity',
                      'date':'Date'})
plot.update_layout(title='Sales Over Time')
plot.show()

# Plot Sales Value
plot = px.line(sales, x='date', y='order_value',
               hover_data=['order_value'],
               labels={'order_value':'Sales Value',
                      'date':'Date'})
plot.update_layout(title='Sales Value Over Time')
plot.show()
```

Based on these plots, there appears to be weak yearly seasonality, and a weak trend as well.

```
In [81]: # Distribution of Sales Quantity
plt.style.use('ggplot')
plt.figure(figsize=(10, 6))
sns.histplot(sales['order_quantity'], bins=30, kde=True)
plt.title('Distribution of Sales Quantity')
plt.xlabel('Sales Quantity')
```

```
plt.ylabel('Frequency')
plt.show()
```



The histogram above shows that sales quantity is right skewed with a long tail.

```
In [82]: # Look at Average Sales by Month
month_data = sales.copy()
month_data['month'] = month_data['date'].dt.strftime('%m')
month_data = month_data.groupby('month')['order_quantity'].mean().reset_index()
plot = px.line(month_data, x='month', y='order_quantity',
               hover_data=['order_quantity'],
               labels={'order_quantity':'Average Sales Quantity',
                       'month':'Month'})
plot.update_layout(title='Average Monthly Sales Quantity')
plot.show()
```

Based on this simple plot, we can see that on average, there is seasonality in the data with peaks in the summer months (and November -- black friday). The main dips are the winter months, which makes sense given that this is a company that sells bicycles.

```
In [83]: # FUNCTIONS
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
def diagnostic_plot(series, title="Time Series"):
    '''Plot Time Series, ACF, PACF, and Histogram for a given series.'''
    plt.style.use('ggplot')
    fig, ax = plt.subplots(2,2, figsize=(15,10))
    fig.suptitle(f'Diagnostic Plots: {title}', fontsize=16, fontweight='bold')
    # Time series plot
    ax[0,0].plot(series, linewidth=1.5, color='darkblue')
    ax[0,0].set_title('Original Time Series', fontweight='bold')
```

```

    ax[0,0].set_xlabel('Time')
    ax[0,0].set_ylabel('Value')
    ax[0,0].grid(True, alpha=0.3)
    # ACF plot
    plot_acf(series, lags=min(60, len(series)//2), ax=ax[0,1], alpha=0.05)
    ax[0,1].set_title('Autocorrelation Function (ACF)', fontweight='bold')
    # PACF plot
    plot_pacf(series, lags=min(60, len(series)//2), ax=ax[1, 0],
               alpha=0.05, method='ywm')
    ax[1, 0].set_title('Partial Autocorrelation Function (PACF)', fontweight='bold')

    # Histogram
    ax[1, 1].hist(series, bins=30, density=True, alpha=0.7,
                  color='darkblue', edgecolor='black')
    ax[1, 1].set_title('Distribution', fontweight='bold')
    ax[1, 1].set_xlabel('Value')
    ax[1, 1].set_ylabel('Density')
    ax[1, 1].grid(True, alpha=0.3, axis='y')
    plt.tight_layout()
    plt.show()

from statsmodels.tsa.stattools import adfuller, kpss
def adf_test(series):
    '''Function to perform ADF test on series and print results.'''
    result = adfuller(series)
    print("ADF TEST RESULTS:")
    print('  ADF Statistic: {:.4f}'.format(result[0]))
    print('  p-value: {:.4f}'.format(result[1]))
    if result[1] < 0.05:
        print("Conclusion: Reject Null Hypothesis -- Series is Stationary")
    else:
        print("Conclusion: Fail to Reject Null Hypothesis -- Series is Non-Stationary")
def kpss_test(series):
    '''function to perform KPSS test on series and print results...'''
    result = kpss(series)
    print("KPSS TEST RESULTS:")
    print('  KPSS Statistic: {:.4f}'.format(result[0]))
    print('  p-value: {:.4f}'.format(result[1]))
    if result[1] < 0.05:
        print("Conclusion: Reject Null Hypothesis -- Series is Non-Stationary")
    else:
        print("Conclusion: Fail to Reject Null Hypothesis -- Series is Stationary")

from statsmodels.stats.diagnostic import acorr_ljungbox
def ljung_box_test(series, lags=10):
    '''function to perform LB test on series and print results'''
    result = acorr_ljungbox(series, lags=lags, return_df=True)
    print("Ljung-Box Test Results: PVal < .05 means we reject the null and there IS")
    return result.style.format("{:.5f}", subset=['lb_stat', 'lb_pvalue'])

from statsmodels.tsa.seasonal import STL
def stl_decomposition(series, series_name, period):
    '''function to perform STL decompose and plot results'''
    stl = STL(series, period=period)
    result = stl.fit()
    fig = result.plot()

```

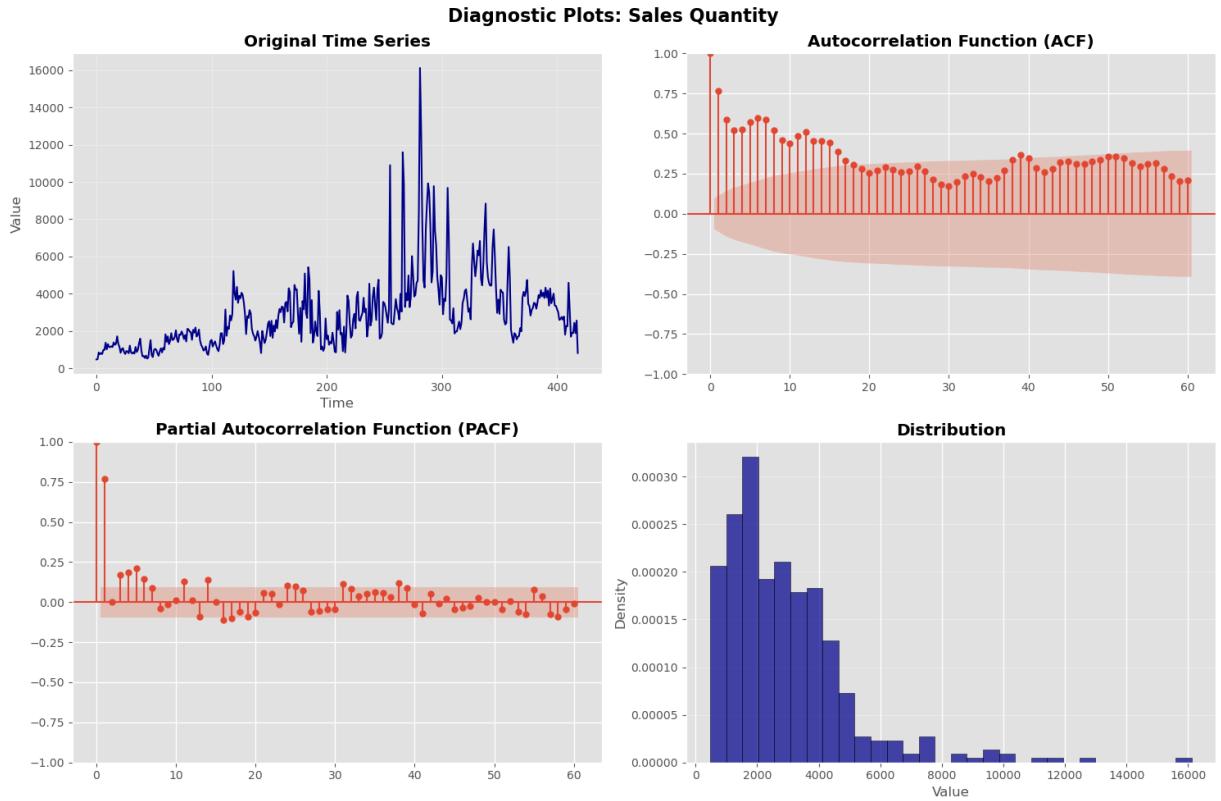
```

fig.suptitle(f'STL Decomposition of {series_name}', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()

```

In [84]: # Plot Diagnostic Plots -- SALES QUANTITY

```
diagnostic_plot(sales['order_quantity'], title='Sales Quantity')
```



The autocorrelation plot slowly decreases, which is a sign of non stationarity. The PACF shows a significant spike at lag 1 and then quickly drops off, which suggests that AR(1) could be appropriate for this data. Before we do any modeling, however, I need to ensure that the data is stationary (I don't believe that it is given the ACF plot...).

## QUESTION 2: (45 points): Apply Box Jenkins Methodology using ARIMA, SARIMA, or SARIMAX depending on your data to obtain and evaluate forecasts.

1. *Identification:* Perform all the necessary steps for identification and include automatic and manual identification. Summarize the results from your identification process.
2. *Estimation:* Perform all necessary steps for identification and summarize the results of your estimation process.
3. *Diagnostic Checking:* Perform all necessary steps for diagnostic checking and summarize the results of your diagnostic checking process.

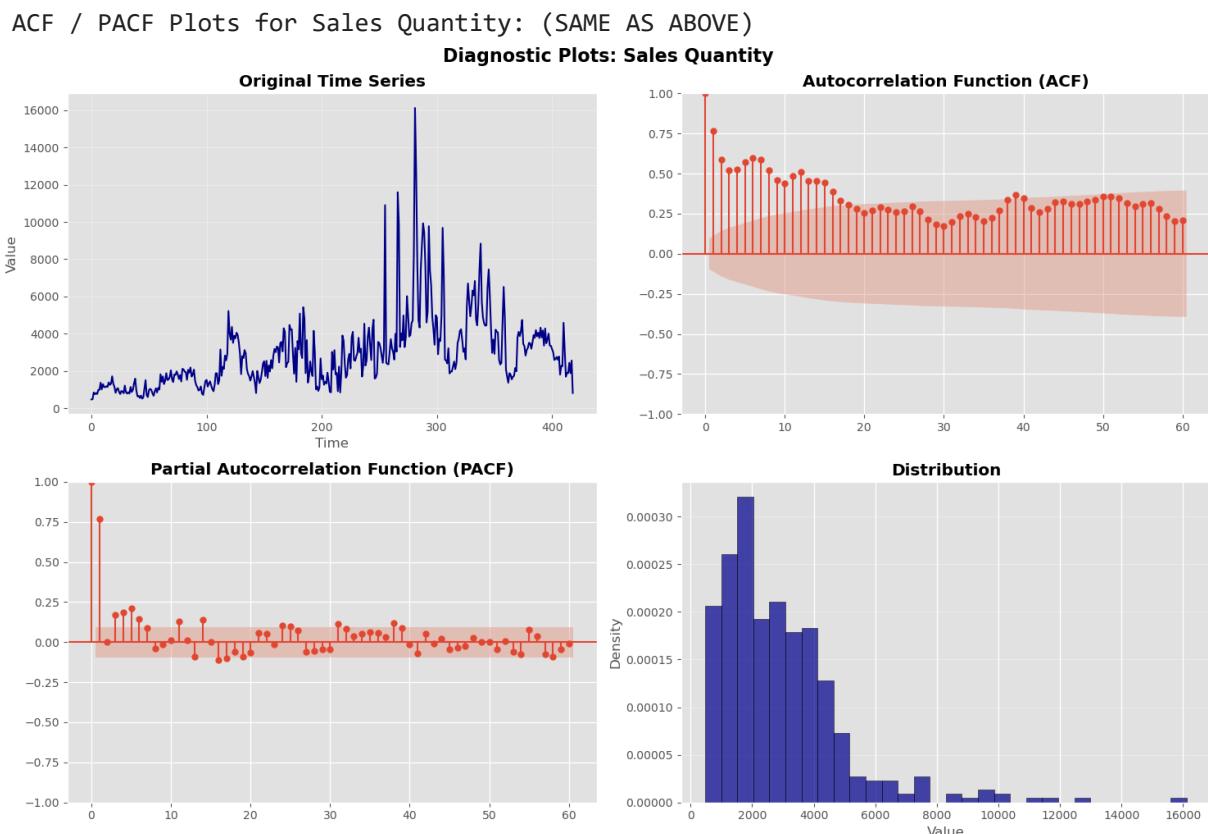
4. *Forecasting:* Compare the performance of the forecasts of your selected models using the appropriate methods on a test data set. Show graphics of the forecast performance to include confidence intervals. Describe your results and conclusions about the usefulness of the models you evaluated to forecast in the application domain of the data set.

## Part A: Identification

To identify the appropriate model(s) (and AR / MA orders), I will use ACF / PACF plots, the ADF test, the KPSS test, and the auto\_arima function.

In [85]:

```
# Plot Diagnostics
print("ACF / PACF Plots for Sales Quantity: (SAME AS ABOVE)")
diagnostic_plot(sales['order_quantity'], title='Sales Quantity')
```



Based on the ACF plot, there is a slow decay of autocorrelations at lags 1-60, which is a strong sign of non-stationarity. The PACF plot shows a significant spike at lag 1 and then quickly drops off; however, some of the lags after lag 1 are also significant. The spike at lag 1 suggests that AR(1) could be appropriate for the data. The data likely needs to be differenced, so I will assess the PACF and ACF plots after differencing to determine the appropriate AR and MA orders.

In [86]:

```
adf_test(sales['order_quantity']); print("\n")
kpss_test(sales['order_quantity'])
```

#### ADF TEST RESULTS:

ADF Statistic: -2.9045

p-value: 0.0448

Conclusion: Reject Null Hypothesis -- Series is Stationary

#### KPSS TEST RESULTS:

KPSS Statistic: 1.7403

p-value: 0.0100

Conclusion: Reject Null Hypothesis -- Series is Non-Stationary

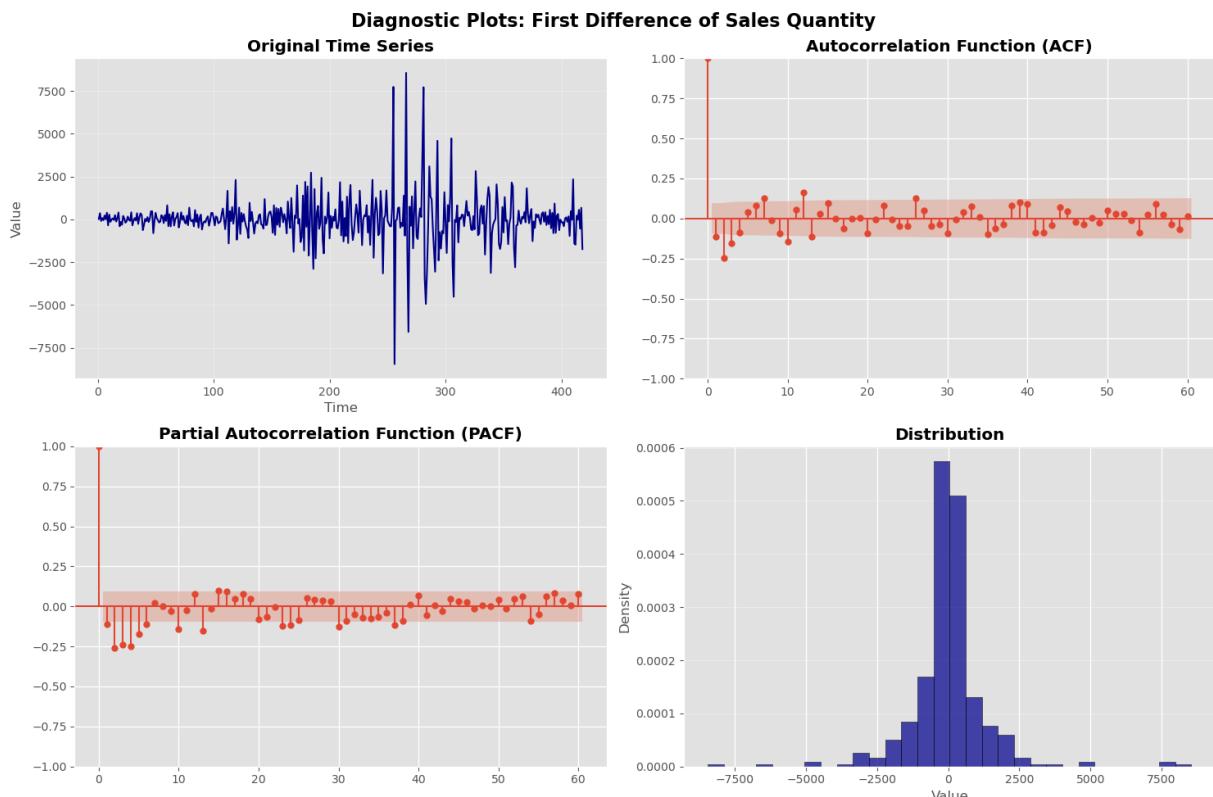
/tmp/ipykernel\_4853/1655461131.py:45: InterpolationWarning:

The test statistic is outside of the range of p-values available in the look-up table. The actual p-value is smaller than the p-value returned.

The ADF test and KPSS test conflict with each other. They both reject the null hypothesis. For ADF, this teams that the data are stationary, but for the KPSS test, this means that the data are nonstationary. The P-Value for the ADF test is close to .05, however, so it barely rejects the null hypothesis. Given the ACF plot, I do not believe these data to be stationary, so I will difference the data, and then reassess these diagnostics.

In [87]:

```
# First Difference + Diagnostics
sales['order_quantity_diff'] = sales['order_quantity'].diff()
diagnostic_plot(sales['order_quantity_diff'].dropna(), title='First Difference of Sales Quantity')
adf_test(sales['order_quantity_diff'].dropna()); print("\n")
kpss_test(sales['order_quantity_diff'].dropna())
```



```
ADF TEST RESULTS:  
ADF Statistic: -5.5045  
p-value: 0.0000  
Conclusion: Reject Null Hypothesis -- Series is Stationary
```

```
KPSS TEST RESULTS:  
KPSS Statistic: 0.2898  
p-value: 0.1000  
Conclusion: Fail to Reject Null Hypothesis -- Series is Stationary
```

```
/tmp/ipykernel_4853/1655461131.py:45: InterpolationWarning:
```

The test statistic is outside of the range of p-values available in the look-up table. The actual p-value is greater than the p-value returned.

After taking the first difference, the ACF shows small negative spikes at lag 1-3 and then quickly drops off. This suggests that an MA(3) *could* be appropriate for this data. The PACF shows significant negative spikes at lags 1-4 and then it quickly drops off. This suggests that an AR(4) model *could* be appropriate for these data. Finally, the ADF and KPSS tests both conclude that the series IS stationary after this first difference, which corroborates what I see in the ACF and PACF plots. Lets see if auto\_arima agrees with my conclusions...

Now, I am going to subset the data into a training set, and test set. I will use data before 2025 to train and data from 2025 onwards.

```
In [88]: # First, I will split the data data into training and testing set.  
sales['log_order_quantity'] = np.log(sales['order_quantity'])  
# Training vs Testing Sets  
from datetime import datetime  
CUTOFF_DATE = datetime(2025, 1, 1)  
train = sales[sales['date'] < CUTOFF_DATE].copy()  
test = sales[sales['date'] >= CUTOFF_DATE].copy()
```

```
In [89]: import pmдарима as pm  
# First try a nonseasonal model  
print("Finding best arima based on AIC:")  
auto_arima_model = pm.auto_arima(  
    train['order_quantity'],  
    seasonal=False,  
    stepwise=True,  
    trace=True,  
    suppress_warnings=True,  
    error_action='ignore',  
    information_criterion='aic'  
)  
print(auto_arima_model.summary())  
print("\n\nFinding best arima based on BIC:")  
auto_arima_model = pm.auto_arima(  
    train['order_quantity'],  
    seasonal=False,  
    stepwise=True,
```

```
    trace=True,  
    suppress_warnings=True,  
    error_action='ignore',  
    information_criterion='bic'  
)  
print(auto_arima_model.summary())
```

Finding best arima based on AIC:

Performing stepwise search to minimize aic

ARIMA(2,1,2)(0,0,0)[0]	intercept	:	AIC=6226.430, Time=0.88 sec
ARIMA(0,1,0)(0,0,0)[0]	intercept	:	AIC=6311.373, Time=0.01 sec
ARIMA(1,1,0)(0,0,0)[0]	intercept	:	AIC=6309.261, Time=0.02 sec
ARIMA(0,1,1)(0,0,0)[0]	intercept	:	AIC=6299.839, Time=0.03 sec
ARIMA(0,1,0)(0,0,0)[0]		:	AIC=6309.377, Time=0.01 sec
ARIMA(1,1,2)(0,0,0)[0]	intercept	:	AIC=6235.290, Time=0.31 sec
ARIMA(2,1,1)(0,0,0)[0]	intercept	:	AIC=6229.867, Time=0.26 sec
ARIMA(3,1,2)(0,0,0)[0]	intercept	:	AIC=inf, Time=0.27 sec
ARIMA(2,1,3)(0,0,0)[0]	intercept	:	AIC=6228.343, Time=0.28 sec
ARIMA(1,1,1)(0,0,0)[0]	intercept	:	AIC=6245.969, Time=0.20 sec
ARIMA(1,1,3)(0,0,0)[0]	intercept	:	AIC=6236.932, Time=0.22 sec
ARIMA(3,1,1)(0,0,0)[0]	intercept	:	AIC=6228.479, Time=0.21 sec
ARIMA(3,1,3)(0,0,0)[0]	intercept	:	AIC=6230.300, Time=0.61 sec
ARIMA(2,1,2)(0,0,0)[0]		:	AIC=6224.525, Time=0.14 sec
ARIMA(1,1,2)(0,0,0)[0]		:	AIC=6233.460, Time=0.06 sec
ARIMA(2,1,1)(0,0,0)[0]		:	AIC=6227.988, Time=0.14 sec
ARIMA(3,1,2)(0,0,0)[0]		:	AIC=6229.680, Time=0.41 sec
ARIMA(2,1,3)(0,0,0)[0]		:	AIC=6226.479, Time=0.17 sec
ARIMA(1,1,1)(0,0,0)[0]		:	AIC=6244.251, Time=0.14 sec
ARIMA(1,1,3)(0,0,0)[0]		:	AIC=6237.217, Time=0.16 sec
ARIMA(3,1,1)(0,0,0)[0]		:	AIC=6226.581, Time=0.18 sec
ARIMA(3,1,3)(0,0,0)[0]		:	AIC=6228.273, Time=0.18 sec

Best model: ARIMA(2,1,2)(0,0,0)[0]

Total fit time: 4.906 seconds

#### SARIMAX Results

Dep. Variable:	y	No. Observations:	364
Model:	SARIMAX(2, 1, 2)	Log Likelihood	-3107.263
Date:	Fri, 20 Feb 2026	AIC	6224.525
Time:	12:15:11	BIC	6243.997
Sample:	0	HQIC	6232.265
	- 364		

Covariance Type: opg

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.9058	0.176	5.136	0.000	0.560	1.251
ar.L2	-0.4813	0.094	-5.142	0.000	-0.665	-0.298
ma.L1	-1.2347	0.179	-6.898	0.000	-1.586	-0.884
ma.L2	0.4107	0.162	2.529	0.011	0.092	0.729
sigma2	1.628e+06	4.36e+04	37.339	0.000	1.54e+06	1.71e+06

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	4835.79
Prob(Q):	0.92	Prob(JB):	0.00
Heteroskedasticity (H):	20.74	Skew:	2.32
Prob(H) (two-sided):	0.00	Kurtosis:	20.27

Warnings:

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

Finding best arima based on BIC:

Performing stepwise search to minimize bic

```
ARIMA(2,1,2)(0,0,0)[0] intercept : BIC=6249.797, Time=0.26 sec
ARIMA(0,1,0)(0,0,0)[0] intercept : BIC=6319.162, Time=0.01 sec
ARIMA(1,1,0)(0,0,0)[0] intercept : BIC=6320.944, Time=0.02 sec
ARIMA(0,1,1)(0,0,0)[0] intercept : BIC=6311.523, Time=0.03 sec
ARIMA(0,1,0)(0,0,0)[0]          : BIC=6313.271, Time=0.01 sec
ARIMA(1,1,2)(0,0,0)[0] intercept : BIC=6254.762, Time=0.25 sec
ARIMA(2,1,1)(0,0,0)[0] intercept : BIC=6249.339, Time=0.25 sec
ARIMA(1,1,1)(0,0,0)[0] intercept : BIC=6261.546, Time=0.16 sec
ARIMA(2,1,0)(0,0,0)[0] intercept : BIC=6299.662, Time=0.03 sec
ARIMA(3,1,1)(0,0,0)[0] intercept : BIC=6251.845, Time=0.21 sec
ARIMA(3,1,0)(0,0,0)[0] intercept : BIC=6284.522, Time=0.04 sec
ARIMA(3,1,2)(0,0,0)[0] intercept : BIC=inf, Time=0.29 sec
ARIMA(2,1,1)(0,0,0)[0]          : BIC=6243.565, Time=0.10 sec
ARIMA(1,1,1)(0,0,0)[0]          : BIC=6255.934, Time=0.10 sec
ARIMA(2,1,0)(0,0,0)[0]          : BIC=6293.772, Time=0.02 sec
ARIMA(3,1,1)(0,0,0)[0]          : BIC=6246.053, Time=0.12 sec
ARIMA(2,1,2)(0,0,0)[0]          : BIC=6243.997, Time=0.08 sec
ARIMA(1,1,0)(0,0,0)[0]          : BIC=6315.053, Time=0.02 sec
ARIMA(1,1,2)(0,0,0)[0]          : BIC=6249.038, Time=0.05 sec
ARIMA(3,1,0)(0,0,0)[0]          : BIC=6278.636, Time=0.03 sec
ARIMA(3,1,2)(0,0,0)[0]          : BIC=6253.046, Time=0.32 sec
```

Best model: ARIMA(2,1,1)(0,0,0)[0]

Total fit time: 2.411 seconds

#### SARIMAX Results

```
=====
Dep. Variable:                      y      No. Observations:                  364
Model:                SARIMAX(2, 1, 1)   Log Likelihood:           -3109.994
Date:                Fri, 20 Feb 2026   AIC:                         6227.988
Time:                       12:15:14     BIC:                         6243.565
Sample:                           0      HQIC:                        6234.180
                                         - 364
Covariance Type:            opg
=====
```

```
=====
              coef    std err        z     P>|z|      [0.025      0.975]
-----
ar.L1      0.4966    0.048    10.274    0.000      0.402      0.591
ar.L2     -0.2506    0.042    -5.909    0.000     -0.334     -0.167
ma.L1     -0.8013    0.045   -17.674    0.000     -0.890     -0.712
sigma2    1.616e+06  4.02e+04   40.147    0.000    1.54e+06    1.69e+06
=====
```

```
=====
Ljung-Box (L1) (Q):                 0.15    Jarque-Bera (JB):             4328.76
Prob(Q):                            0.69    Prob(JB):                   0.00
Heteroskedasticity (H):               20.74    Skew:                      2.19
Prob(H) (two-sided):                  0.00    Kurtosis:                  19.34
=====
```

#### Warnings:

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

The auto-arima model based on AIC suggests that the best model is ARIMA(2,1,2). This is similar to what I observed in the manual diagnostics above as well. It is *slightly* different, but

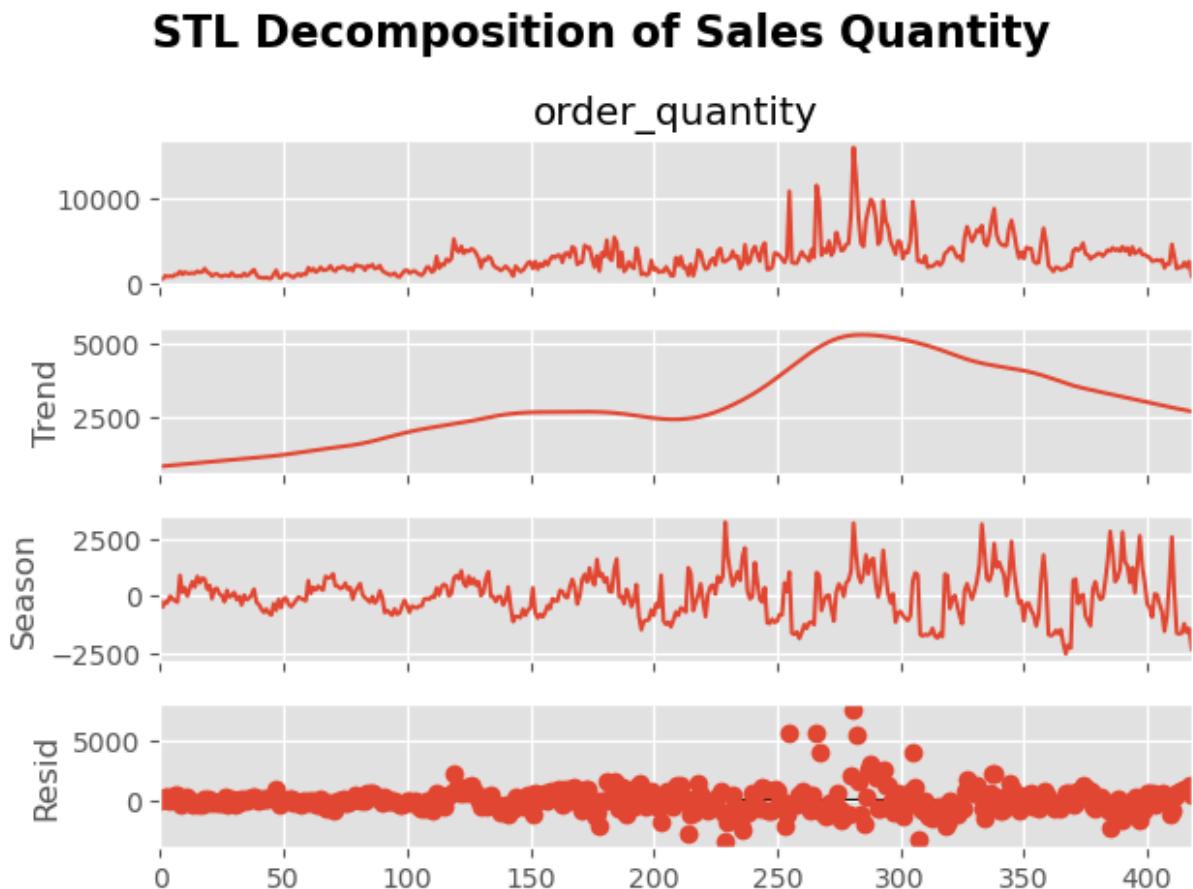
I believe that to be due to the limited data that the AutoArima is seeing. Based on BIC, the best ARIMA model is ARIMA(2,1,1). I will also try a seasonal ARIMA with the same AR and MA orders as the nonseasonal model to try to account for the weak yearly seasonality that I observed in the data.

## PART B & C: ESTIMATION + DIAGNOSTIC CHECKING

Fit the model to the data.

Perform all necessary steps for diagnostic checking and summarize the results of your diagnostic checking process.

```
In [90]: # STL Decomposition to Assess Components of Series...
stl_decomposition(sales['order_quantity'], series_name='Sales Quantity', period=52)
```



Based on the STL decomposition, there is a weak seasonal component. Based on this, I will try a seasonal ARIMA model as well as the nonseasonal ARIMA model below. Residual variance is also higher from weeks 250-300. I might be able to model this with an exogenous variable, as the heightened demand is a lagged result of the COVID pandemic (lagged because demand responded when inventory arrived).

```
In [91]: # Define Results DF
model_results = {}
```

```
In [92]: def model_fit_plot(df, date_col, actual_col, fit_col, residuals, title):
    # Diagnostic Plots... Actual vs Fitted, Residuals, ACF/PACF of Residuals, Histograms
    import scipy.stats as stats
    plt.style.use('ggplot')
    fig, ax = plt.subplots(3,2, figsize=(15,10))
    fig.suptitle(f'Diagnostic Plots: {title}', fontsize=16, fontweight='bold')
    # Actual vs Fitted
    ax[0,0].plot(df[date_col], df[actual_col], label='Actual', color='blue')
    ax[0,0].plot(df[date_col], df[fit_col], label='Fitted', color='orange')
    ax[0,0].set_title('Actual vs Fitted Values', fontweight='bold')
    ax[0,0].set_xlabel('Date')
    ax[0,0].set_ylabel('Sales Quantity')
    ax[0,0].legend()
    ax[0,0].grid(True, alpha=0.3)
    # Residuals
    ax[0,1].plot(df[date_col], residuals, label='Residuals', color='red')
    ax[0,1].set_title('Residuals Over Time', fontweight='bold')
    ax[0,1].set_xlabel('Date')
    ax[0,1].set_ylabel('Residual Value')
    ax[0,1].legend()
    ax[0,1].grid(True, alpha=0.3)
    # ACF of Residuals
    plot_acf(residuals, lags=min(60, len(residuals)//2), ax=ax[1, 0], alpha=0.05)
    ax[1, 0].set_title('ACF of Residuals', fontweight='bold')
    # PACF of Residuals
    plot_pacf(residuals, lags=min(60, len(residuals)//2), ax=ax[1, 1], alpha=0.05,
    ax[1, 1].set_title('PACF of Residuals', fontweight='bold')
    # QQ plot of residuals
    sm.qqplot(residuals, line='s', ax=ax[2, 0])
    ax[2, 0].set_title('QQ Plot of Residuals', fontweight='bold')
    # Histogram of Residuals
    ax[2, 1].hist(residuals, bins=30, density=True, alpha=0.7, color='red', edgecolor='black')
    mean = np.mean(residuals)
    sigma = np.std(residuals)
    x = np.linspace(mean - 4*sigma, mean + 4*sigma, 1000)
    normal_pdf = stats.norm.pdf(x, mean, sigma)
    ax[2, 1].plot(x, normal_pdf, color='blue', linestyle='--', label='Normal PDF')
    ax[2, 1].legend()
    ax[2, 1].set_title('Histogram of Residuals', fontweight='bold')
    ax[2, 1].set_xlabel('Residual Value')
    ax[2, 1].set_ylabel('Density')
    ax[2, 1].grid(True, alpha=0.3, axis='y')
    plt.tight_layout()
    plt.show()

def model_eval(fit_model):
    # MSE
    residuals = fit_model.resid
    mse = np.mean(residuals**2)
    # RMSE
    rmse = np.sqrt(mse)
    # AIC and BIC
    aic = fit_model.aic
    bic = fit_model.bic
    # Ljung Box Test on Residuals
```

```

ljung_box_results = acorr_ljungbox(residuals, lags=1, return_df=True)
ljung_box_pval = ljung_box_results['lb_pvalue'].iloc[0]
# Heteroskedasticity Test on Residuals
from statsmodels.stats.diagnostic import het_arch
arch_test = het_arch(residuals, nlags=12)
arch_pval = arch_test[1]
return mse, rmse, aic, bic, ljung_box_pval, arch_pval

```

```

In [93]: # Fit 212 Model
P = 2
D = 1
Q = 2
Y = train['order_quantity']

# Fit Model and Output Summary
model = ARIMA(Y, order=(P,D,Q))
model_212 = model.fit()
print(model_212.summary())

# Add fitted values to sales df for diagnostics
FIT_COL = f'ARIMA({P},{D},{Q})'
train[FIT_COL] = model_212.fittedvalues
residuals = model_212.resid

# Model Diagnostic Results
model_fit_plot(train, date_col='date', actual_col='order_quantity', fit_col=FIT_COL)
mse, rmse, aic, bic, ljung_box_pval, arch_pval = model_eval(model_212)
model_results[FIT_COL] = {
    "Fit Model" : model_212,
    'MSE' : mse,
    'RMSE' : rmse,
    'AIC' : aic,
    'BIC' : bic,
    'Ljung-Box Residual Autocorrelation P-Val' : ljung_box_pval,
    'Heteroskedasticity P-Val' : arch_pval
}

```

/home/eca4zm/miniforge3/envs/tseries2/lib/python3.10/site-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting parameters.

### SARIMAX Results

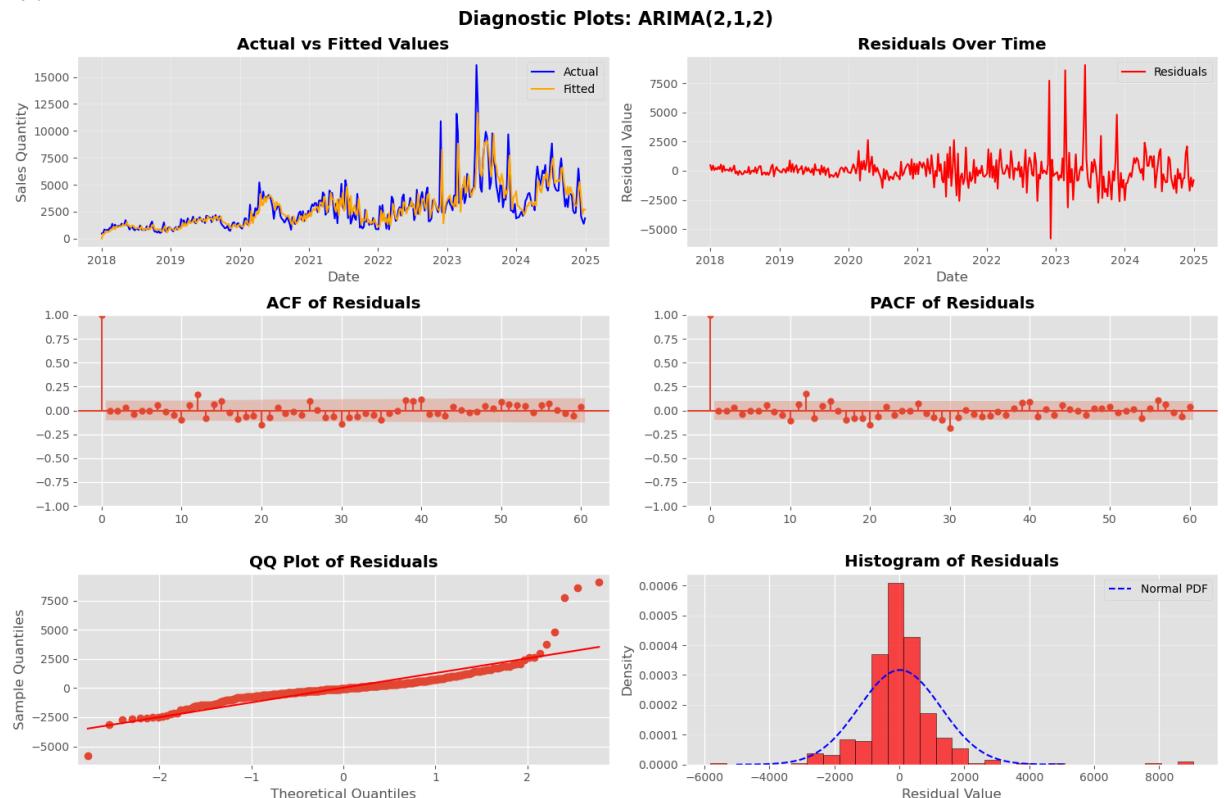
Dep. Variable:	order_quantity	No. Observations:	364
Model:	ARIMA(2, 1, 2)	Log Likelihood	-3107.263
Date:	Fri, 20 Feb 2026	AIC	6224.525
Time:	12:15:14	BIC	6243.997
Sample:	0 - 364	HQIC	6232.265
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
<hr/>						
ar.L1	0.9058	0.176	5.136	0.000	0.560	1.251
ar.L2	-0.4813	0.094	-5.142	0.000	-0.665	-0.298
ma.L1	-1.2347	0.179	-6.898	0.000	-1.586	-0.884
ma.L2	0.4107	0.162	2.529	0.011	0.092	0.729
sigma2	1.628e+06	4.36e+04	37.339	0.000	1.54e+06	1.71e+06
<hr/>						
Ljung-Box (L1) (Q):			0.01	Jarque-Bera (JB):		4835.79
Prob(Q):			0.92	Prob(JB):		0.00
Heteroskedasticity (H):			20.74	Skew:		2.32
Prob(H) (two-sided):			0.00	Kurtosis:		20.27
<hr/>						

#### Warnings:

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



Overall, this model fit well aside from the heteroskedasticity that I see above in the residuals plot. The ACF and PACF of the residuals show no spikes, however, which is a good sign. The histogram shows that the residuals are approximately normally distributed (slightly tighter

than normal), which is also a good sign. Finally, the QQ plot shows that the residuals are approximately normally distributed as well; however, there is some deviation from normality in the right tail, which is a sign that this model is not perfect.

All P Values are significant at the .05 level, which is good.

```
In [94]: # 211 Model
P = 2
D = 1
Q = 1
Y = train['order_quantity']

# Fit Model and Output Summary
model = ARIMA(Y, order=(P,D,Q))
model_211 = model.fit()
print(model_211.summary())

# Add fitted values to sales df for diagnostics
FIT_COL = f'ARIMA({P},{D},{Q})'
train[FIT_COL] = model_211.fittedvalues
residuals = model_211.resid

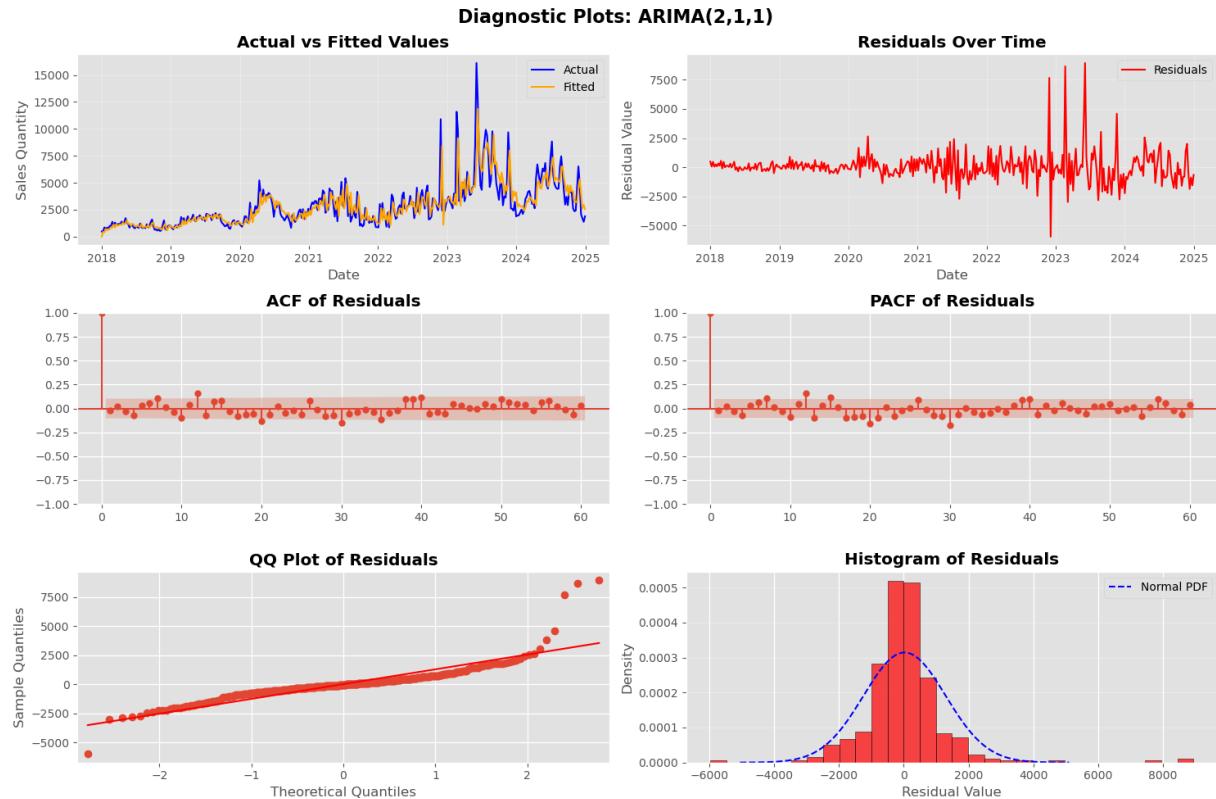
# Model Diagnostic Results
model_fit_plot(train, date_col='date', actual_col='order_quantity', fit_col=FIT_COL)
mse, rmse, aic, bic, ljung_box_pval, arch_pval = model_eval(model_211)
model_results[FIT_COL] = {
    "Fit Model" : model_211,
    'MSE' : mse,
    'RMSE' : rmse,
    'AIC' : aic,
    'BIC' : bic,
    'Ljung-Box Residual Autocorrelation P-Val' : ljung_box_pval,
    'Heteroskedasticity P-Val' : arch_pval
}
```

### SARIMAX Results

Dep. Variable:	order_quantity	No. Observations:	364
Model:	ARIMA(2, 1, 1)	Log Likelihood	-3109.994
Date:	Fri, 20 Feb 2026	AIC	6227.988
Time:	12:15:15	BIC	6243.565
Sample:	0 - 364	HQIC	6234.180
Covariance Type:	opg		

#### Warnings:

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



Similar results to the ARIMA(2,1,2) model, but slightly better BIC. There appears to be heteroskedasticity in the residuals, which is a problem. The QQ plot also has some deviation from normality in the right tail, which is another sign that this model is not perfect.

Based on the heteroskedasticity, I am going to log transform the data and refit the model to see if that helps...

The P Values are significant at the .05 level and smaller than the Pvalues in the first model.

```
In [95]: # Log Transform Data and Refit Model
auto_arima_model_log = pm.auto_arima(
    train['log_order_quantity'],
    seasonal=False,
    stepwise=True,
    trace=True,
    suppress_warnings=True,
    error_action='ignore',
    information_criterion='aic'
)
print(auto_arima_model_log.summary())
diagnostic_plot(train['log_order_quantity'].diff().dropna(), title='Log of Sales Qu
```

Performing stepwise search to minimize aic

```

ARIMA(2,1,2)(0,0,0)[0] intercept    : AIC=258.011, Time=0.73 sec
ARIMA(0,1,0)(0,0,0)[0] intercept    : AIC=308.463, Time=0.03 sec
ARIMA(1,1,0)(0,0,0)[0] intercept    : AIC=290.317, Time=0.04 sec
ARIMA(0,1,1)(0,0,0)[0] intercept    : AIC=266.249, Time=0.05 sec
ARIMA(0,1,0)(0,0,0)[0]                : AIC=306.502, Time=0.02 sec
ARIMA(1,1,2)(0,0,0)[0] intercept    : AIC=244.324, Time=0.33 sec
ARIMA(0,1,2)(0,0,0)[0] intercept    : AIC=242.653, Time=0.10 sec
ARIMA(0,1,3)(0,0,0)[0] intercept    : AIC=241.898, Time=0.15 sec
ARIMA(1,1,3)(0,0,0)[0] intercept    : AIC=242.975, Time=0.26 sec
ARIMA(0,1,4)(0,0,0)[0] intercept    : AIC=241.455, Time=0.32 sec
ARIMA(1,1,4)(0,0,0)[0] intercept    : AIC=236.657, Time=0.49 sec
ARIMA(2,1,4)(0,0,0)[0] intercept    : AIC=237.763, Time=0.46 sec
ARIMA(1,1,5)(0,0,0)[0] intercept    : AIC=237.867, Time=0.52 sec
ARIMA(0,1,5)(0,0,0)[0] intercept    : AIC=240.169, Time=0.17 sec
ARIMA(2,1,3)(0,0,0)[0] intercept    : AIC=238.370, Time=0.37 sec
ARIMA(2,1,5)(0,0,0)[0] intercept    : AIC=240.194, Time=0.66 sec
ARIMA(1,1,4)(0,0,0)[0]                : AIC=234.845, Time=0.13 sec
ARIMA(0,1,4)(0,0,0)[0]                : AIC=239.771, Time=0.06 sec
ARIMA(1,1,3)(0,0,0)[0]                : AIC=241.338, Time=0.13 sec
ARIMA(2,1,4)(0,0,0)[0]                : AIC=235.959, Time=0.19 sec
ARIMA(1,1,5)(0,0,0)[0]                : AIC=236.070, Time=0.21 sec
ARIMA(0,1,3)(0,0,0)[0]                : AIC=240.279, Time=0.06 sec
ARIMA(0,1,5)(0,0,0)[0]                : AIC=238.445, Time=0.10 sec
ARIMA(2,1,3)(0,0,0)[0]                : AIC=236.671, Time=0.16 sec
ARIMA(2,1,5)(0,0,0)[0]                : AIC=237.711, Time=0.48 sec

```

Best model: ARIMA(1,1,4)(0,0,0)[0]

Total fit time: 6.234 seconds

#### SARIMAX Results

```
=====
Dep. Variable:                      y      No. Observations:                  364
Model:                 SARIMAX(1, 1, 4)   Log Likelihood:                -111.422
Date:                 Fri, 20 Feb 2026   AIC:                            234.845
Time:                         12:15:23     BIC:                            258.211
Sample:                           0      HQIC:                           244.133
                                         - 364
Covariance Type:                    opg
=====
```

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.7348	0.128	5.753	0.000	0.484	0.985
ma.L1	-1.1612	0.134	-8.635	0.000	-1.425	-0.898
ma.L2	0.0802	0.091	0.881	0.378	-0.098	0.259
ma.L3	0.0385	0.086	0.446	0.656	-0.131	0.208
ma.L4	0.1773	0.063	2.792	0.005	0.053	0.302
sigma2	0.1079	0.006	17.363	0.000	0.096	0.120

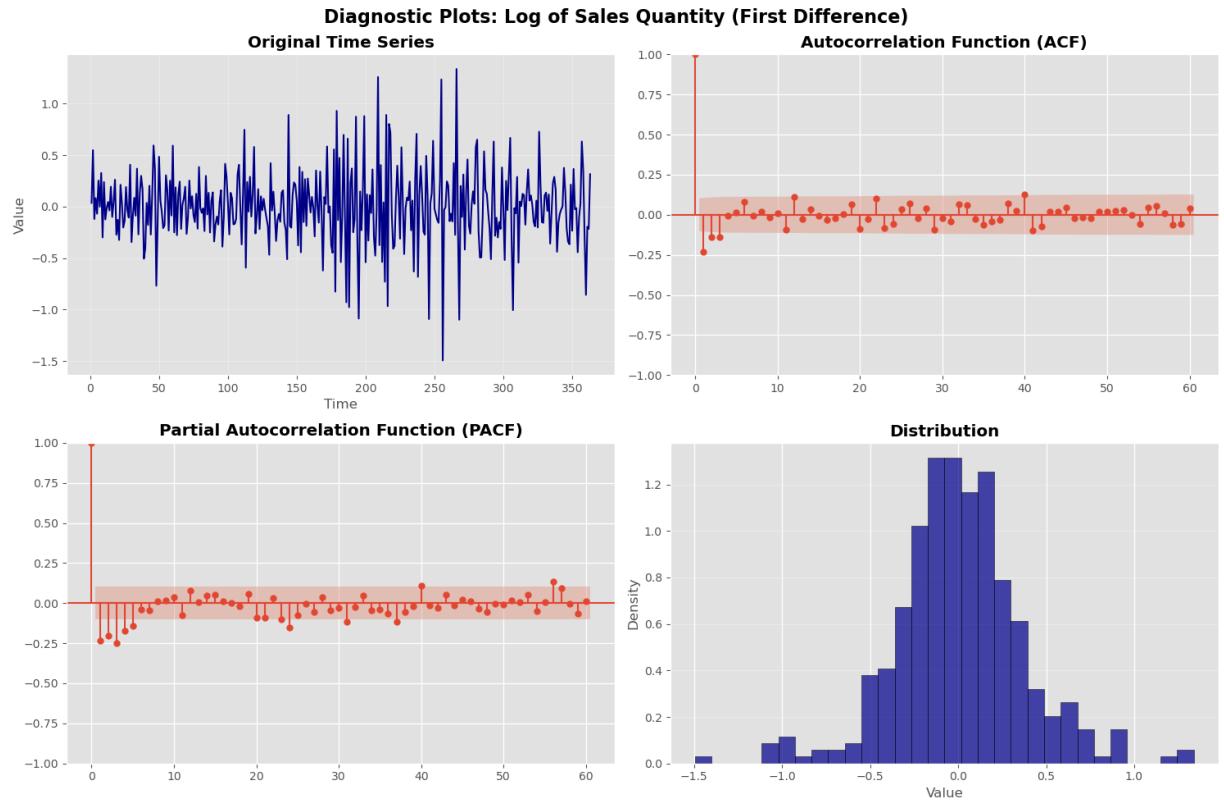
=====

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	43.03
Prob(Q):	0.92	Prob(JB):	0.00
Heteroskedasticity (H):	2.10	Skew:	0.25
Prob(H) (two-sided):	0.00	Kurtosis:	4.61

=====

Warnings:

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



```
In [96]: auto_arima_model_log = pm.auto_arima(
    train['log_order_quantity'],
    seasonal=True,
    m=52,
    d = 1,
    D = 0,
    max_p = 3,
    max_q = 3,
    max_P = 3,
    max_Q = 3,
    stepwise=True,
    trace=True,
    suppress_warnings=True,
    error_action='ignore',
    information_criterion='aic'
)
print(auto_arima_model_log.summary())
```

Performing stepwise search to minimize aic

ARIMA(2,1,2)(1,0,1)[52] intercept	:	AIC=260.392, Time=13.77 sec
ARIMA(0,1,0)(0,0,0)[52] intercept	:	AIC=308.463, Time=0.03 sec
ARIMA(1,1,0)(1,0,0)[52] intercept	:	AIC=291.760, Time=2.28 sec
ARIMA(0,1,1)(0,0,1)[52] intercept	:	AIC=266.440, Time=2.70 sec
ARIMA(0,1,0)(0,0,0)[52]	:	AIC=306.502, Time=0.02 sec
ARIMA(2,1,2)(0,0,1)[52] intercept	:	AIC=258.413, Time=8.66 sec
ARIMA(2,1,2)(0,0,0)[52] intercept	:	AIC=258.011, Time=0.35 sec
ARIMA(2,1,2)(1,0,0)[52] intercept	:	AIC=258.362, Time=15.71 sec
ARIMA(1,1,2)(0,0,0)[52] intercept	:	AIC=244.324, Time=0.38 sec
ARIMA(1,1,2)(1,0,0)[52] intercept	:	AIC=242.773, Time=9.28 sec
ARIMA(1,1,2)(2,0,0)[52] intercept	:	AIC=253.443, Time=41.10 sec
ARIMA(1,1,2)(1,0,1)[52] intercept	:	AIC=244.664, Time=11.06 sec
ARIMA(1,1,2)(0,0,1)[52] intercept	:	AIC=251.150, Time=10.91 sec
ARIMA(1,1,2)(2,0,1)[52] intercept	:	AIC=255.505, Time=49.19 sec
ARIMA(0,1,2)(1,0,0)[52] intercept	:	AIC=242.190, Time=2.56 sec
ARIMA(0,1,2)(0,0,0)[52] intercept	:	AIC=242.653, Time=0.14 sec
ARIMA(0,1,2)(2,0,0)[52] intercept	:	AIC=244.067, Time=17.50 sec
ARIMA(0,1,2)(1,0,1)[52] intercept	:	AIC=243.996, Time=7.05 sec
ARIMA(0,1,2)(0,0,1)[52] intercept	:	AIC=242.287, Time=3.76 sec
ARIMA(0,1,2)(2,0,1)[52] intercept	:	AIC=246.073, Time=18.31 sec
ARIMA(0,1,1)(1,0,0)[52] intercept	:	AIC=266.406, Time=1.59 sec
ARIMA(0,1,3)(1,0,0)[52] intercept	:	AIC=241.283, Time=5.13 sec
ARIMA(0,1,3)(0,0,0)[52] intercept	:	AIC=241.898, Time=0.17 sec
ARIMA(0,1,3)(2,0,0)[52] intercept	:	AIC=243.106, Time=22.99 sec
ARIMA(0,1,3)(1,0,1)[52] intercept	:	AIC=242.947, Time=13.01 sec
ARIMA(0,1,3)(0,0,1)[52] intercept	:	AIC=241.404, Time=5.12 sec
ARIMA(0,1,3)(2,0,1)[52] intercept	:	AIC=inf, Time=53.12 sec
ARIMA(1,1,3)(1,0,0)[52] intercept	:	AIC=242.296, Time=18.01 sec
ARIMA(0,1,3)(1,0,0)[52]	:	AIC=239.634, Time=1.42 sec
ARIMA(0,1,3)(0,0,0)[52]	:	AIC=240.279, Time=0.10 sec
ARIMA(0,1,3)(2,0,0)[52]	:	AIC=241.450, Time=19.53 sec
ARIMA(0,1,3)(1,0,1)[52]	:	AIC=241.293, Time=7.01 sec
ARIMA(0,1,3)(0,0,1)[52]	:	AIC=239.759, Time=4.60 sec
ARIMA(0,1,3)(2,0,1)[52]	:	AIC=inf, Time=89.06 sec
ARIMA(0,1,2)(1,0,0)[52]	:	AIC=240.490, Time=1.62 sec
ARIMA(1,1,3)(1,0,0)[52]	:	AIC=240.628, Time=6.93 sec
ARIMA(1,1,2)(1,0,0)[52]	:	AIC=241.105, Time=2.00 sec

Best model: ARIMA(0,1,3)(1,0,0)[52]

Total fit time: 466.256 seconds

#### SARIMAX Results

```
=====
=====
```

Dep. Variable:	y	No. Observations:
364		
Model:	SARIMAX(0, 1, 3)x(1, 0, [], 52)	Log Likelihood
114.817		-
Date:	Fri, 20 Feb 2026	AIC
239.634		
Time:	12:23:09	BIC
259.106		
Sample:	0	HQIC
247.374		
Covariance Type:	- 364	
	opg	

```

=====
             coef    std err      z   P>|z|    [0.025    0.975]
-----
ma.L1     -0.4015    0.048  -8.407   0.000   -0.495   -0.308
ma.L2     -0.1993    0.045  -4.467   0.000   -0.287   -0.112
ma.L3     -0.0946    0.052  -1.813   0.070   -0.197    0.008
ar.S.L52    0.0891    0.048   1.846   0.065   -0.006    0.184
sigma2     0.1099    0.006  17.164   0.000    0.097    0.122
=====
Ljung-Box (L1) (Q):                  0.03  Jarque-Bera (JB):        42.49
Prob(Q):                           0.87  Prob(JB):            0.00
Heteroskedasticity (H):           2.27  Skew:                 0.18
Prob(H) (two-sided):            0.00  Kurtosis:            4.64
=====
```

Warnings:

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

Based on the autoarima (nonseasonal) above, the best model to try after a log transform is a ARIMA(1,1,4).

Based on the seasonal autoarima above, the best model to try after a log transform is a SARIMA(0,1,3)x(1,0,0,52).

```
In [97]: # Fit Model 114
P = 1
D = 1
Q = 4
Y = train['log_order_quantity']

# Fit Model and Output Summary
model = ARIMA(Y, order=(P,D,Q))
model_log_114 = model.fit()
print(model_log_114.summary())

# Add fitted values to sales df for diagnostics
FIT_COL = f'LOG_ARIMA({P},{D},{Q})'
train[FIT_COL] = model_log_114.fittedvalues
residuals = model_log_114.resid

# Model Diagnostic Results
model_fit_plot(train, date_col='date', actual_col='log_order_quantity', fit_col=FIT_COL)
mse, rmse, aic, bic, ljung_box_pval, arch_pval = model_eval(model_log_114)
model_results[FIT_COL] = {
    "Fit Model" : model_log_114,
    'MSE' : mse,
    'RMSE' : rmse,
    'AIC' : aic,
    'BIC' : bic,
    'Ljung-Box Residual Autocorrelation P-Val' : lじung_box_pval,
    'Heteroskedasticity P-Val' : arch_pval
}
```

### SARIMAX Results

Dep. Variable:	log_order_quantity	No. Observations:	364
Model:	ARIMA(1, 1, 4)	Log Likelihood	-111.422
Date:	Fri, 20 Feb 2026	AIC	234.845
Time:	12:23:10	BIC	258.211
Sample:	0 - 364	HQIC	244.133
Covariance Type:	opg		

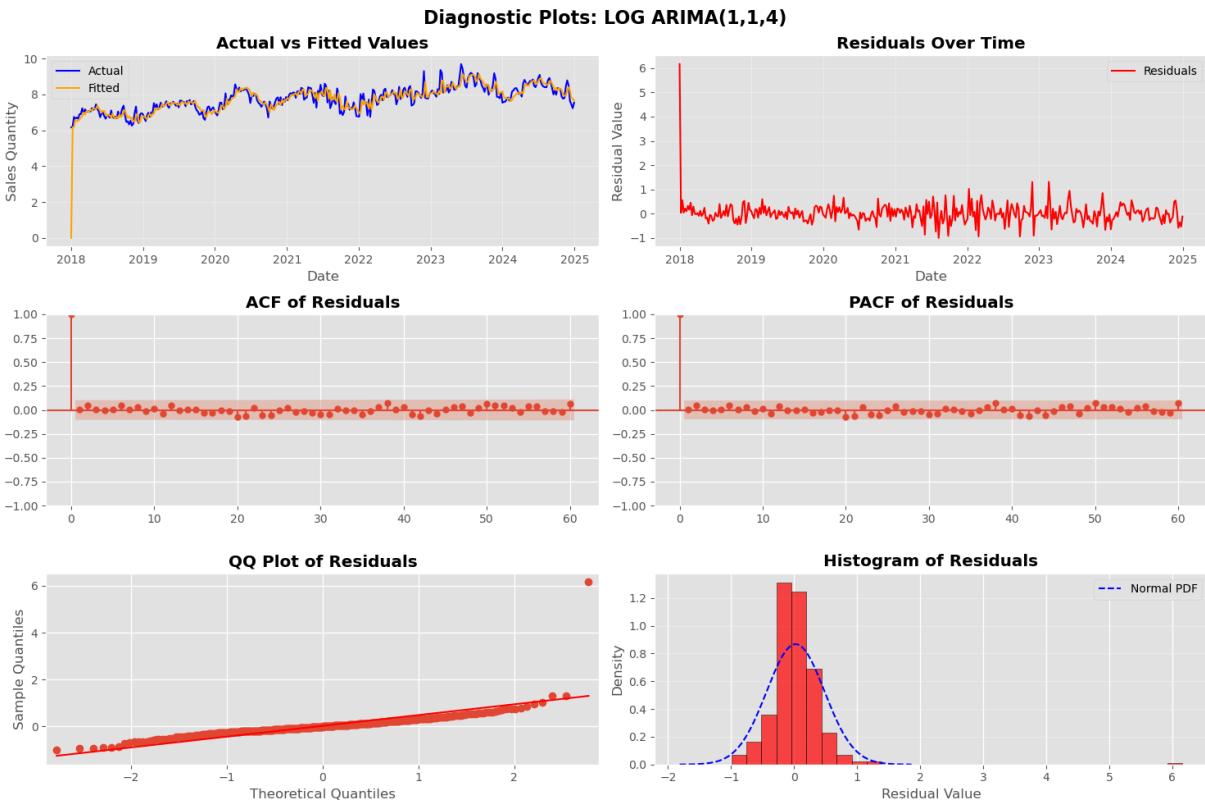
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.7348	0.128	5.753	0.000	0.484	0.985
ma.L1	-1.1612	0.134	-8.635	0.000	-1.425	-0.898
ma.L2	0.0802	0.091	0.881	0.378	-0.098	0.259
ma.L3	0.0385	0.086	0.446	0.656	-0.131	0.208
ma.L4	0.1773	0.063	2.792	0.005	0.053	0.302
sigma2	0.1079	0.006	17.363	0.000	0.096	0.120

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	43.03
Prob(Q):	0.92	Prob(JB):	0.00
Heteroskedasticity (H):	2.10	Skew:	0.25
Prob(H) (two-sided):	0.00	Kurtosis:	4.61

#### Warnings:

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



The log transform helped dramatically to reduce the heteroskedasticity and outliers in the residuals in the model above. The ACF and PACF of the residuals show no spikes and appear to be white noise, which is a good sign. The histograms are also normally distributed, and

the QQ plot shows that the residuals are approximately normally distributed as well. Overall, this model fit is better than the non-log transformed model, so I will move forward with this version.

I am unsure why the prediction is 0 at T=0.

P Values for MA L2 and L3 are not significant at the .05 level... THis indicates that these parameters may not be necessary for the model.

```
In [98]: # SARIMA LOG MODEL
P = 0
D = 1
Q = 3
P_S = 1
D_S = 0
Q_S = 0
SEASON = 52
Y = train['log_order_quantity']

# Fit Model and Output Summary
model = ARIMA(Y, order=(P,D,Q), seasonal_order=(P_S, D_S, Q_S, SEASON))
model_log_013100 = model.fit()
print(model_log_013100.summary())

# Add fitted values to sales df for diagnostics
FIT_COL = f'LOG_SARIMA({P},{D},{Q}x{P_S},{D_S},{Q_S},{SEASON})'
train[FIT_COL] = model_log_013100.fittedvalues
residuals = model_log_013100.resid

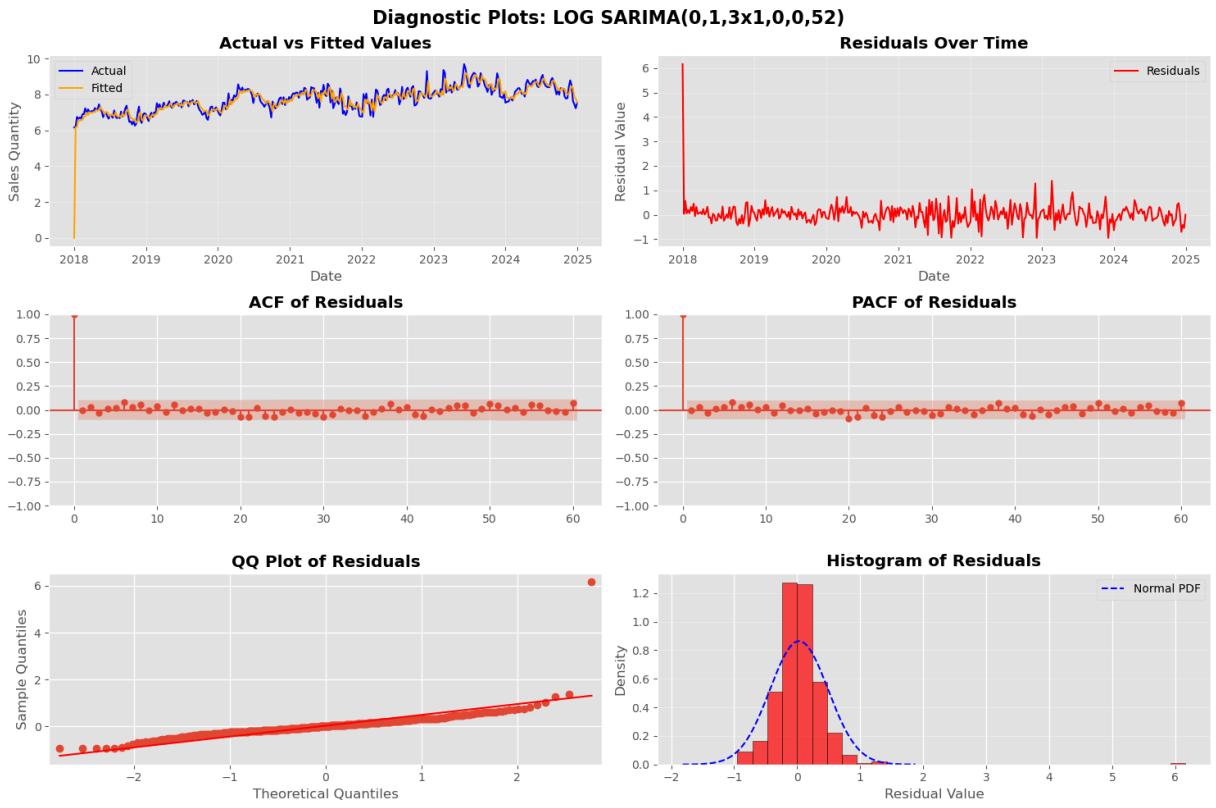
# Model Diagnostic Results
model_fit_plot(train, date_col='date', actual_col='log_order_quantity', fit_col=FIT_COL)
mse, rmse, aic, bic, ljung_box_pval, arch_pval = model_eval(model_log_013100)
model_results[FIT_COL] = {
    "Fit Model" : model_log_013100,
    'MSE' : mse,
    'RMSE' : rmse,
    'AIC' : aic,
    'BIC' : bic,
    'Ljung-Box Residual Autocorrelation P-Val' : ljung_box_pval,
    'Heteroskedasticity P-Val' : arch_pval
}
```

## SARIMAX Results

=====						
=====						
Dep. Variable:	log_order_quantity				No. Observations:	364
Model:	ARIMA(0, 1, 3)x(1, 0, [], 52)				Log Likelihood	-11 4.817
Date:	Fri, 20 Feb 2026				AIC	23 9.634
Time:	12:23:12				BIC	25 9.106
Sample:	0				HQIC	24 7.374
	- 364					
Covariance Type:	opg					
=====						
	coef	std err	z	P> z	[0.025	0.975]
-----						
ma.L1	-0.4015	0.048	-8.407	0.000	-0.495	-0.308
ma.L2	-0.1993	0.045	-4.467	0.000	-0.287	-0.112
ma.L3	-0.0946	0.052	-1.813	0.070	-0.197	0.008
ar.S.L52	0.0891	0.048	1.846	0.065	-0.006	0.184
sigma2	0.1099	0.006	17.164	0.000	0.097	0.122
=====						
Ljung-Box (L1) (Q):			0.03	Jarque-Bera (JB):		42.49
Prob(Q):			0.87	Prob(JB):		0.00
Heteroskedasticity (H):			2.27	Skew:		0.18
Prob(H) (two-sided):			0.00	Kurtosis:		4.64
=====						

## Warnings:

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



This model looks like a strong contender! Minimal residual correlation, normally distributed, and they look relatively homoskedastic.

MA L3 and AR L52 are (barely) not significant at the .05 level, which suggests that these parameters may not be necessary for the model.

```
In [99]: # SARIMA MODEL 2 with SEASONAL DIFFERENCING (Non Log)
P = 1
D = 0
Q = 1
P_S = 1
D_S = 1
Q_S = 1
SEASON = 52
Y = train['order_quantity']

# Fit Model and Output Summary
model = ARIMA(Y, order=(P,D,Q), seasonal_order=(P_S, D_S, Q_S, SEASON))
model_log_101111 = model.fit()
print(model_log_101111.summary())

# Add fitted values to sales df for diagnostics
FIT_COL = f'SARIMA({P},{D},{Q}x{P_S},{D_S},{Q_S},{SEASON})'
train[FIT_COL] = model_log_101111.fittedvalues
residuals = model_log_101111.resid

# Model Diagnostic Results
model_fit_plot(train, date_col='date', actual_col='order_quantity', fit_col=FIT_COL
mse, rmse, aic, bic, ljung_box_pval, arch_pval = model_eval(model_log_101111)
model_results[FIT_COL] = {
```

```

    "Fit Model" : model_log_101111,
    'MSE' : mse,
    'RMSE' : rmse,
    'AIC' : aic,
    'BIC' : bic,
    'Ljung-Box Residual Autocorrelation P-Val' : ljung_box_pval,
    'Heteroskedasticity P-Val' : arch_pval
}

```

### SARIMAX Results

```

=====
=====
Dep. Variable:                  order_quantity   No. Observations:      364
Model:                 ARIMA(1, 0, 1)x(1, 1, 1, 52)   Log Likelihood     -274.0686
Date:                Fri, 20 Feb 2026   AIC                 549.1373
Time:                      12:23:50   BIC                 551.088
Sample:                   0   HQIC                 549.853
                           - 364
Covariance Type:             opg
=====

```

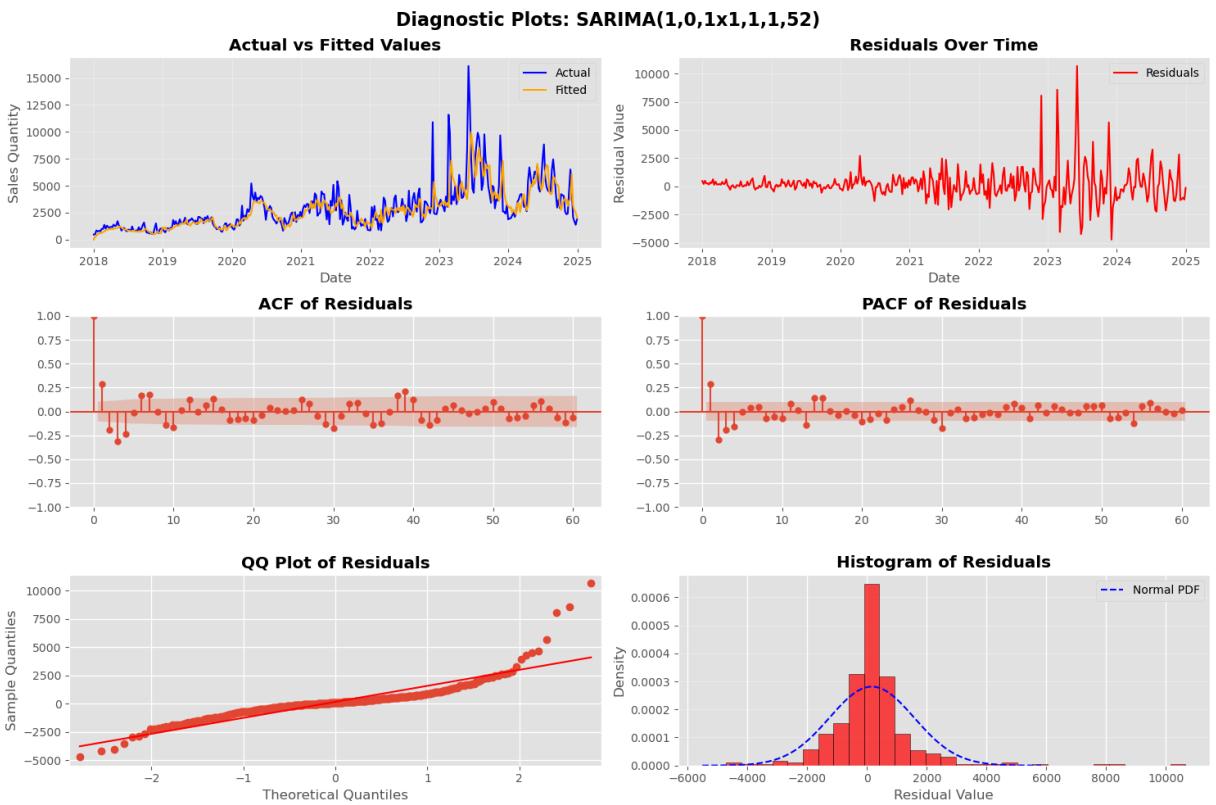
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.9704	0.016	62.529	0.000	0.940	1.001
ma.L1	-0.6700	0.033	-20.503	0.000	-0.734	-0.606
ar.S.L52	-0.1911	0.128	-1.490	0.136	-0.442	0.060
ma.S.L52	-0.6514	0.153	-4.269	0.000	-0.950	-0.352
sigma2	2.366e+06	9.39e+04	25.204	0.000	2.18e+06	2.55e+06

```

=====
Ljung-Box (L1) (Q):                  26.37   Jarque-Bera (JB):           2232.72
Prob(Q):                            0.00   Prob(JB):                     0.00
Heteroskedasticity (H):              22.07   Skew:                         2.12
Prob(H) (two-sided):                0.00   Kurtosis:                    15.40
=====
```

### Warnings:

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



Experimenting with other SARIMA modeling techniques (using seasonal differencing here). The Residual plots and QQ plots do not look great, as there appears to be signal in the data that I am not capturing in my model. That said, all of the Pvalues are meaningful aside from the AR(52) value. The residuals also show some heteroskedasticity. I will leave the model for now and see how it performs on the test set.

```
In [100...]: # Present Training Results from ALL Models
models = list(model_results.keys())
results_df = pd.DataFrame({
    'Model' : models,
    'MSE' : [model_results[model]['MSE'] for model in models],
    'RMSE' : [model_results[model]['RMSE'] for model in models],
    'AIC' : [model_results[model]['AIC'] for model in models],
    'BIC' : [model_results[model]['BIC'] for model in models],
    'Ljung-Box Residual Autocorrelation P-Val' : [model_results[model]['Ljung-Box R'],
    'Heteroskedasticity P-Val' : [model_results[model]['Heteroskedasticity P-Val']]
})
results_df.style.format({
    'MSE' : "{:.2f}",
    'RMSE' : "{:.2f}",
    'AIC' : "{:.2f}",
    'BIC' : "{:.2f}",
    'Ljung-Box Residual Autocorrelation P-Val' : "{:.5f}",
    'Heteroskedasticity P-Val' : "{:.5f}"
})
```

Out[100...]

	Model	MSE	RMSE	AIC	BIC	Ljung-Box Residual Autocorrelation	Het P-Val
<b>0</b>	ARIMA(2,1,2)	1,587,354.63	1,259.90	6,224.53	6,244.00		0.92303
<b>1</b>	ARIMA(2,1,1)	1,612,083.43	1,269.68	6,227.99	6,243.57		0.69470
<b>2</b>	LOG_ARIMA(1,1,4)		0.21	0.46	234.84	258.21	0.91856
<b>3</b>	LOG_SARIMA(0,1,3x1,0,0,52)		0.21	0.46	239.63	259.11	0.96606
<b>4</b>	SARIMA(1,0,1x1,1,1,52)	2,037,196.07	1,427.30	5,491.37	5,510.09		0.00000

In all cases, the heteroskedasticity P Value is very low, which suggests that there is still some heteroskedasticity in the residuals. This may be due to the odd spike at T=0, of which I am unsure of the cause...

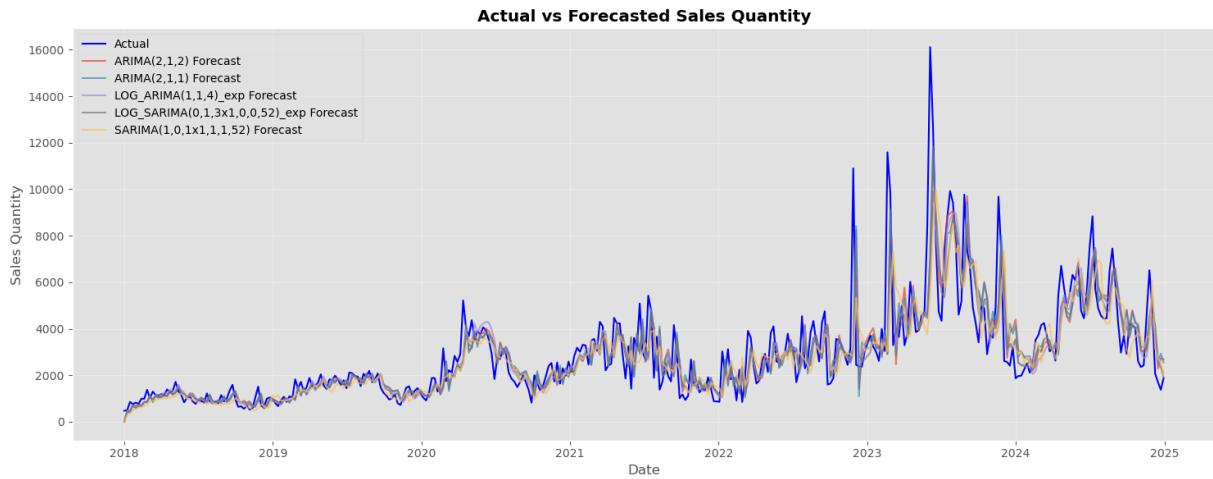
The LB test for the SARIMA with seasonal differencing is significant at the .05 level, which suggests that there is still some autocorrelation in the residuals. I noted this above as well, that we might not be capturing some of the signal in the data (due to the autocorrelation in the residual plots).

Interestingly, even though the seasonal differencing model has issues with the residuals, it has the best AIC and BIC values against the other non-log models. I will transform the predictions back to the orginal scale and plot them against the actuals to see how each tracks the data.

In [101...]

```
# Plotting Predictions vs Actual for Each Model on the Training Data
final_cols = []
for col in model_results.keys():
    if 'LOG' in col:
        train[col + '_exp'] = np.exp(train[col])
        final_cols.append(col + '_exp')
    else:
        final_cols.append(col)

plt.style.use('ggplot')
plt.figure(figsize=(15,6))
sns.lineplot(x='date', y='order_quantity', data=train, label='Actual', color='blue')
for col in final_cols:
    sns.lineplot(x='date', y=col, data=train, label=col + ' Forecast', alpha=0.7)
plt.title('Actual vs Forecasted Sales Quantity', fontweight='bold')
plt.xlabel('Date')
plt.ylabel('Sales Quantity')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



All of the models track the data well. We will see how they perform during prediction on the test set below...

## PART D: FORECASTING

Compare the performance of the forecasts of your selected models using the appropriate methods on a test data set. Show graphics of the forecast performance to include confidence intervals. Describe your results and conclusions about the usefulness of the models you evaluated to forecast in the application domain of the data set.

```
In [102...]: models
```

```
Out[102...]: ['ARIMA(2,1,2)',  
 'ARIMA(2,1,1)',  
 'LOG_ARIMA(1,1,4)',  
 'LOG_SARIMA(0,1,3x1,0,0,52)',  
 'SARIMA(1,0,1x1,1,1,52)']
```

```
# Make Predictions  
test = test[['date']]  
weeks = test.shape[0]  
for model in models:  
    fit_model = model_results[model]['Fit Model']  
    fc = fit_model.get_forecast(steps=weeks)  
    if "LOG" in model:  
        test[model + '_Forecast'] = np.exp(fc.predicted_mean.values)  
        test[model + '_Forecast_Lower'] = np.exp(fc.conf_int().iloc[:,0])  
        test[model + '_Forecast_Upper'] = np.exp(fc.conf_int().iloc[:,1])  
    else:  
        test[model + '_Forecast'] = fc.predicted_mean.values  
        test[model + '_Forecast_Lower'] = fc.conf_int().iloc[:,0]  
        test[model + '_Forecast_Upper'] = fc.conf_int().iloc[:,1]
```

```
In [104...]: final_data = pd.merge(sales[['date','order_quantity']], test, on='date', how='left')
```

```
def plot_forecast_with_ci(df, date_col, actual_col, forecast_col, lower_col, upper_  
    '''Function to plot actual vs forecast with confidence intervals'''
```

```

plt.style.use('ggplot')
plt.figure(figsize=(15,6))
sns.lineplot(x=date_col, y=actual_col, data=df, label='Actual', color='blue')
sns.lineplot(x=date_col, y=forecast_col, data=df, label=forecast_col.replace('_',
plt.fill_between(df[date_col], df[lower_col], df[upper_col], color='orange', al
plt.title(title, fontweight='bold')
plt.xlabel('Date')
plt.ylabel('Sales Quantity')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

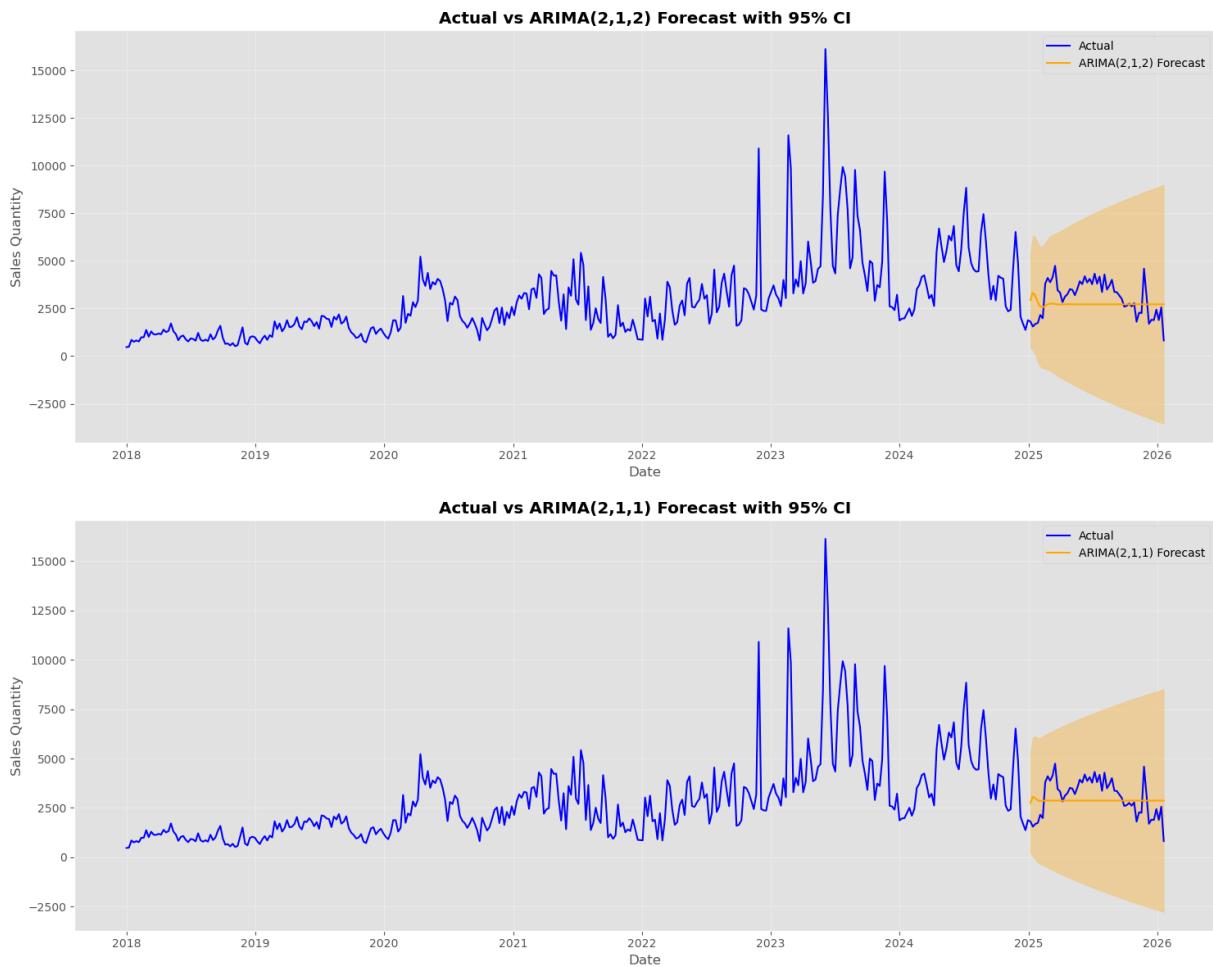
```

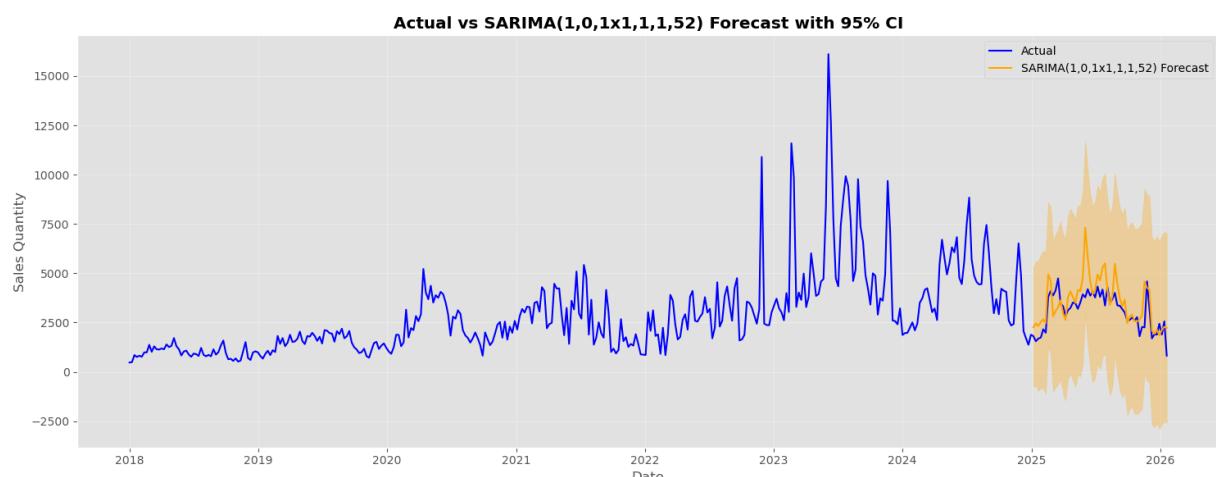
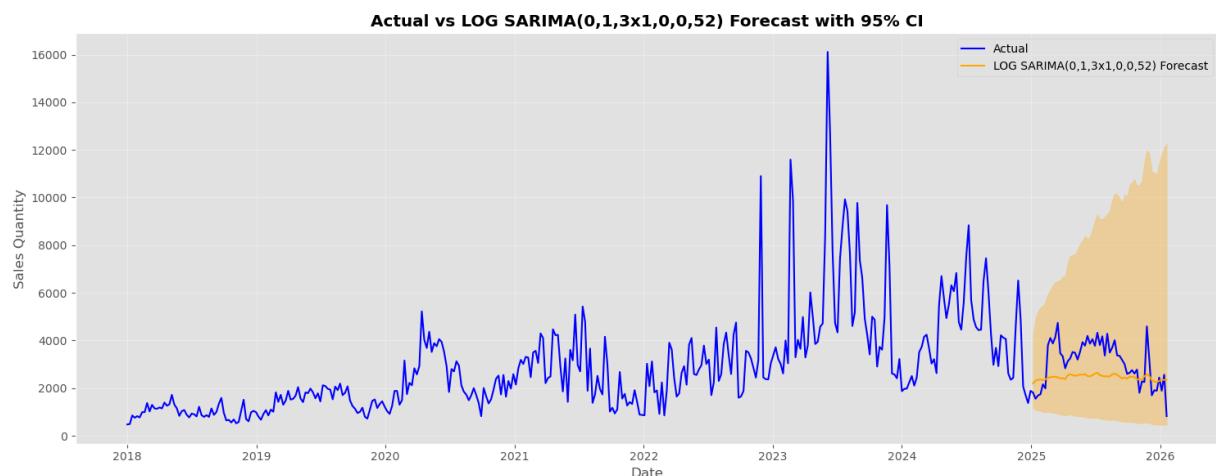
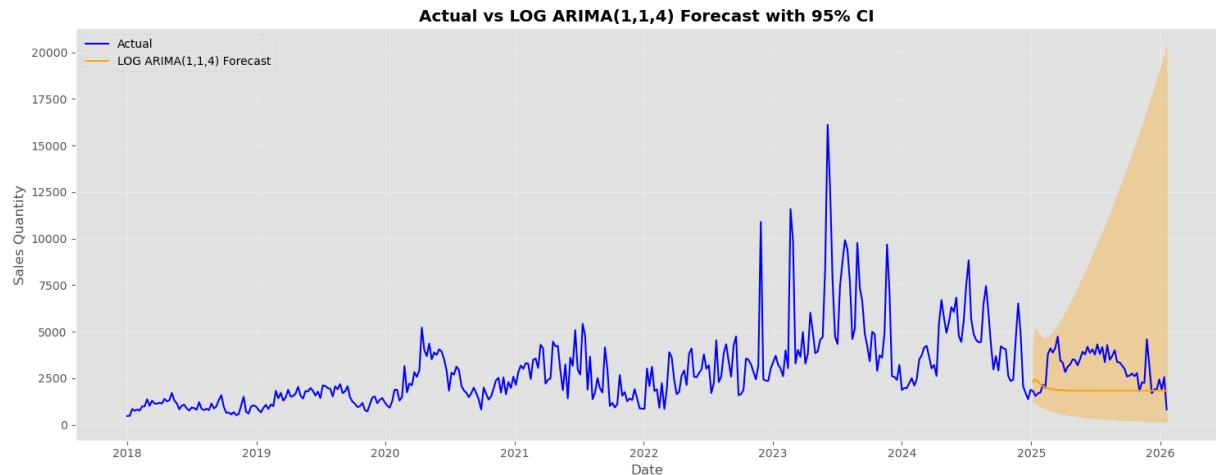
In [106...]

```

# Plot the Forecasts vs Actuals with Confidence Intervals
for model in models:
    forecast_col = model + '_Forecast'
    lower_col = model + '_Forecast_Lower'
    upper_col = model + '_Forecast_Upper'
    plot_forecast_with_ci(
        final_data,
        date_col='date',
        actual_col='order_quantity',
        forecast_col=forecast_col,
        lower_col=lower_col,
        upper_col=upper_col,
        title=f'Actual vs {model.replace("_", " ")} Forecast with 95% CI'
    )

```





Based on these forecast plots, the SARIMA model looks to have the best performance. The forecasts track the seasonal pattern much closer to the actuals than the other models. This is particularly true with the Seasonal Differenced ARIMA! It even captures the uptick in demand towards the end of the calendar year. It is not perfect, but it appears to perform much better on the test data than the other models.

I will look at MSE and RMSE to confirm these conclusions.

```
In [107...]: # Now Look at Residuals and Diagnostics for Each Model
# Calc Resids
```

```

test_results = {}
for model in models:
    forecast_col = model + '_Forecast'
    final_data[model + '_Residuals'] = final_data['order_quantity'] - final_data[forecast_col]
    test_results[model] = {
        'MSE' : np.mean(final_data[model + '_Residuals']**2),
        'RMSE' : np.sqrt(np.mean(final_data[model + '_Residuals']**2))
    }
# Convert to Frame
results_df = pd.DataFrame({
    'Model' : models,
    'Test MSE' : [test_results[model]['MSE'] for model in models],
    'Test RMSE' : [test_results[model]['RMSE'] for model in models]
})
results_df.style.format({
    'Test MSE' : "{:, .2f}",
    'Test RMSE' : "{:, .2f}"
}).background_gradient(subset=['Test MSE', 'Test RMSE'], cmap='Blues')

```

Out[107...]

	Model	Test MSE	Test RMSE
0	ARIMA(2,1,2)	1,019,003.89	1,009.46
1	ARIMA(2,1,1)	883,801.79	940.11
2	LOG_ARIMA(1,1,4)	2,375,647.02	1,541.31
3	LOG_SARIMA(0,1,3x1,0,0,52)	1,097,184.68	1,047.47
4	SARIMA(1,0,1x1,1,1,52)	889,643.34	943.21

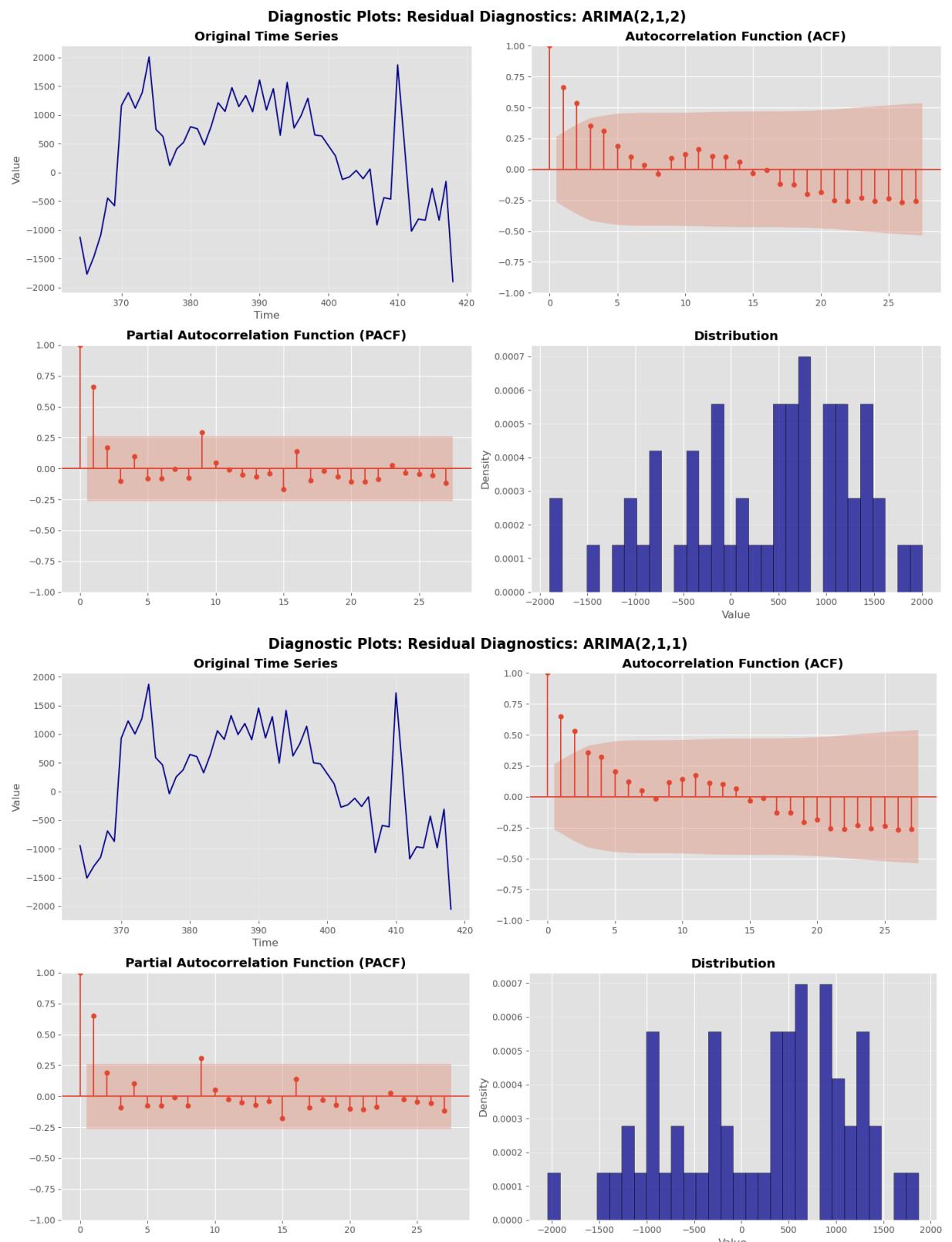
The SARIMA with seasonal differencing performs remarkably well on the test data, given that the residuals did not look great on the training data. ARIMA(2,1,1) also performs well (lowest MSE and RMSE), and barely beats the SARIMA seasonal diff model; HOWEVER, I would recommend the SARIMA model in practice because it captures the seasonality in the data. The LOG\_ARIMA(1,1,4) performs the worse on these data based on RMSE and MSE, which is surprising to me given that the residuals looked better on the training data. This just goes to show that good performance on training data does not always translate to good performance on the test data...

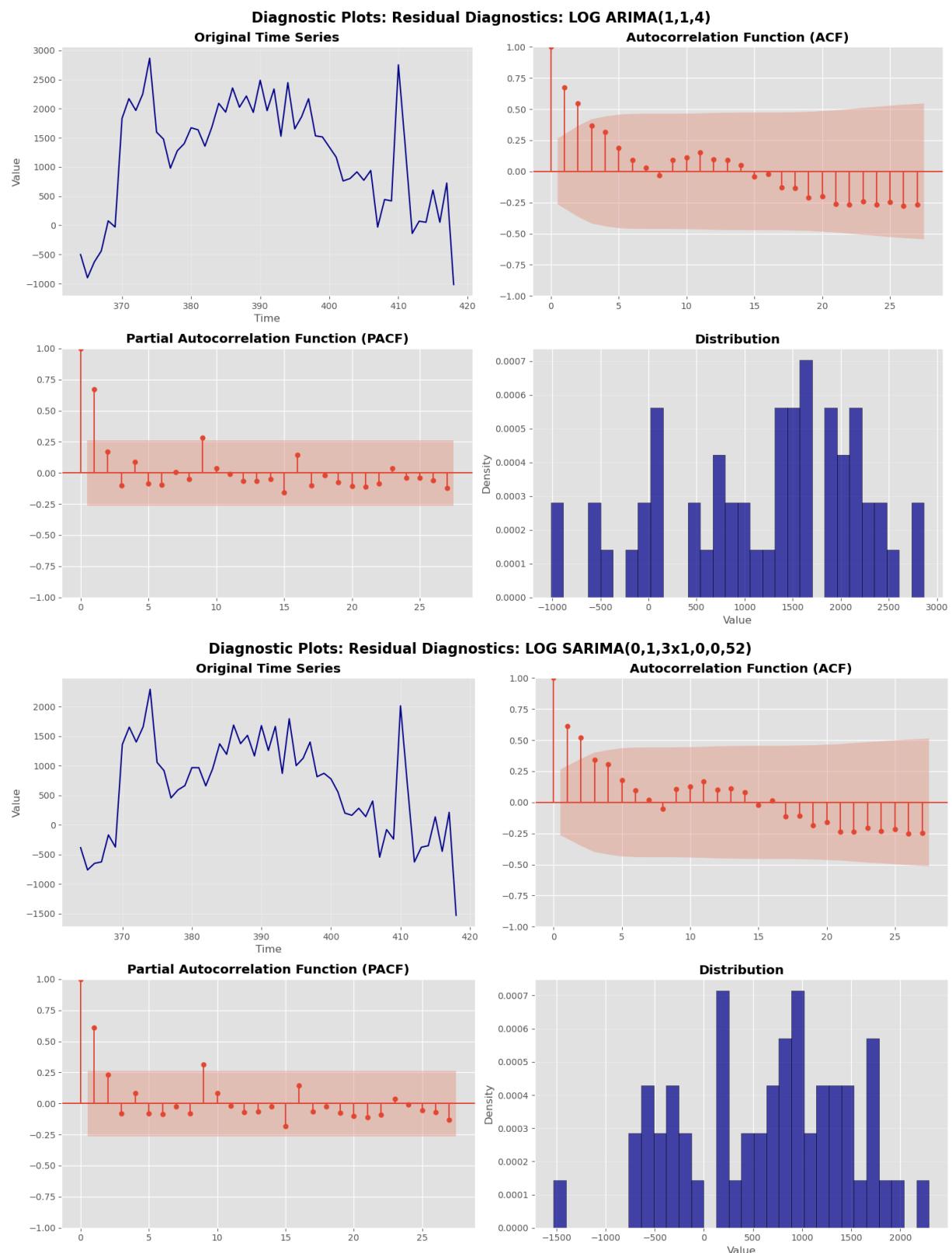
In [108...]

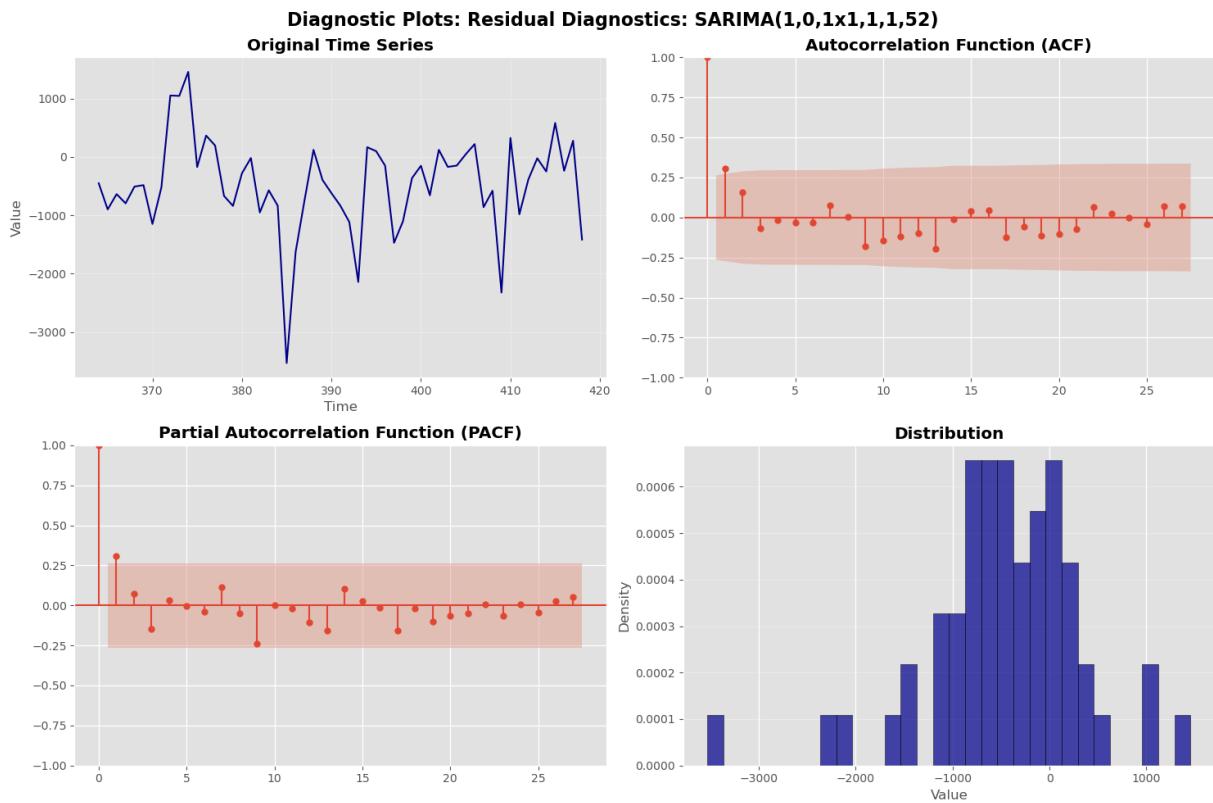
```

# Residual Diagnostics for Each Model
for model in models:
    residual_col = model + '_Residuals'
    diagnostic_plot(final_data[residual_col].dropna(), title=f'Residual Diagnostics for {model}')

```







Given the limited amount of observations in the test set (52), it is hard to get a sense of the true distribution of the residuals; however, the diagnostic plots above validate my belief that the SARIMA with seasonal differencing is the best model for these data. The residuals look more homoskedastic, fairly normally distributed, and there is minimal autocorrelation in the residuals. The other models have more issues with the residuals.

My conclusion from this homework is that, in practice, I would recommend the SARIMA model with seasonal differencing for these data. It captures the seasonality, and performs well on the test data. While some of the simpler models performed well on the test data, they did not capture the seasonal effects to the necessary extent.