DATS 6313 - Time Series Term Project

Predict Future Sales

Nayaeun Kwon

Table of Contents

Table of Figures

Table of Tables

Abstract

This project aims to build a robust and reliable time series forecasting model for a given dataset. The dataset presents inherent seasonality and various challenges that need to be addressed to ensure the most accurate predictions. In the project, various forecasting models, including Holt-Winters, ARIMA, and SARIMA, were implemented and their performances were evaluated based on multiple criteria. The project also involves data preprocessing steps such as stationarity testing, differencing, and feature selection. The study's outcome reveals the SARIMA model's superiority in handling the dataset's seasonality and providing reliable forecasts. This project demonstrates the practical application of time series analysis and the complexity of selecting the most appropriate model.

Introduction

Forecasting is an essential component in various fields, including finance, sales, and logistics, among others. Accurate forecasts allow businesses to make informed decisions and plan effectively for the future. The current project focuses on the application of time series analysis for forecasting purposes.

The dataset used in this project shows a clear seasonal pattern, which adds complexity to the modeling process. Therefore, the project employs a range of forecasting models, starting with simple ones such as Holt-Winters and moving towards more complex models like ARIMA and SARIMA. The models' performances are evaluated using multiple criteria, including Mean Squared Error (MSE), Akaike Information Criterion (AIC), and Bayesian Information Criterion (BIC).

Before modeling, the data undergoes several preprocessing steps, including stationarity testing and differencing. Feature selection techniques are also employed to reduce potential multicollinearity and improve model performance.

The project's ultimate goal is to identify the model that best captures the dataset's characteristics and provides the most accurate forecasts. The results obtained can offer valuable insights into the application of time series analysis in real-world scenarios and the critical factors to consider during the model selection process.

1. Dataset Description

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10
...	...	...	...	...
912995	2017-12-27	10	50	63
912996	2017-12-28	10	50	59
912997	2017-12-29	10	50	74
912998	2017-12-30	10	50	62
912999	2017-12-31	10	50	82

913000 rows × 4 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    913000 non-null  datetime64[ns]
 1   store   913000 non-null  int64
 2   item    913000 non-null  int64
 3   sales   913000 non-null  int64
dtypes: datetime64[ns](1), int64(3)
```

Table 1. Description of Dataset

The dataset consists of 913,000 observations recorded over a period from January 1, 2013, to December 31, 2017. All variables in the dataset are non-null, implying that there are no missing observations in this dataset. Each observation represents the sales data of a particular item from a particular store on a specific date. The dataset contains the following four variables:

1. **'date'**: This variable represents the date on which the sales were recorded. It ranges from January 1, 2013, to December 31, 2017. The 'Date' variable is of the datetime64[ns] data type, indicating it is formatted as a timestamp.
2. **'store'**: This variable represents the store where the sale occurred, numbered from 1 to 10. The 'store' variable is of the int64 data type.
3. **'item'**: This variable represents the specific item that was sold, numbered from 1 to 50. The 'item' variable is of the int64 data type.
4. **'sales'**: This variable represents the number of units of the item that were sold at the store on the given date. This is an integer value that can theoretically take on any positive value, depending on the number of units sold. The 'sales' variable is of the int64 data type.

2. Data Preprocessing

2.1 Checking missing values

Before delving into the data analysis, it's critical to ensure the integrity of the dataset. This involves checking the dataset for any missing or null values that might compromise the validity of our results. At first glance, the dataset appeared to be complete, with each variable containing 913,000 non-null entries. However, to confirm the absence of any missing values, an additional check was performed.

```
Missing values: 0
```

The results of this additional check reaffirmed the initial observation: the dataset is indeed complete, with no missing values detected. This is a favorable result, as it means I won't need to employ any techniques for handling missing data.

2.2 Reshaping the dataset

To the analysis, the original dataset was reshaped to better represent the time series structure of the data. Here's a walkthrough of the steps I followed:

a. Date Formatting: The 'date' column was converted into a datetime object and formatted to represent the respective ‘month’ and ‘year’ of each sales record.
b. Grouping the Data: I grouped the data by 'store', 'item', and 'date', summing the sales for each group. This process gives me the total sales per item per store monthly.
c. Adding New Columns: The reshaped data frame now includes 'year' and 'month' columns, extracted from the 'date' column. This provides me with additional temporal information, which could prove useful in my forthcoming analysis.

The reshaped data frame looks like this:

	store	item	date	sales	year	month
0	1	1	2013-01-01	328	2013	1
1	1	1	2013-02-01	322	2013	2
2	1	1	2013-03-01	477	2013	3
3	1	1	2013-04-01	522	2013	4
4	1	1	2013-05-01	531	2013	5
...	...	...	...	...	...	...
29995	10	50	2017-08-01	2867	2017	8
29996	10	50	2017-09-01	2586	2017	9
29997	10	50	2017-10-01	2507	2017	10
29998	10	50	2017-11-01	2574	2017	11
29999	10	50	2017-12-01	1987	2017	12

30000 rows × 6 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   store   30000 non-null  int64         
 1   item    30000 non-null  int64         
 2   date    30000 non-null  datetime64[ns]
 3   sales   30000 non-null  int64         
 4   year    30000 non-null  int64         
 5   month   30000 non-null  int64         
dtypes: datetime64[ns](1), int64(5)
```

Table 2. Description of Reshaping Dataset

The reshaped dataset now contains 30,000 rows and 6 columns. This restructured data will serve as the basis for our time series analysis and modeling.

3. Exploratory Data Analysis

3.1 Time series visualization

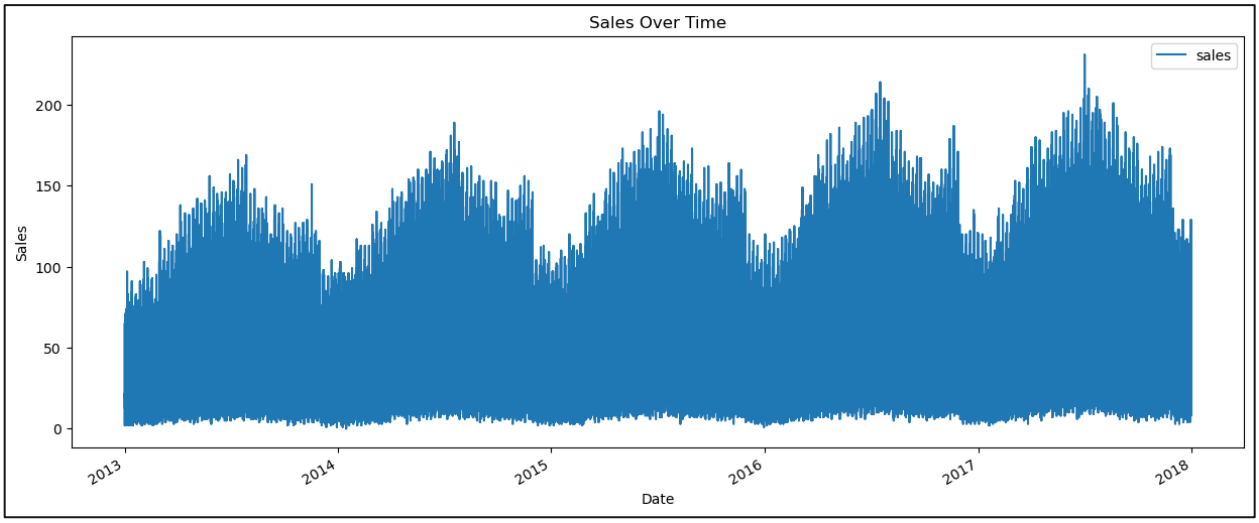


Figure 1. Sales Over Time

One of the initial steps in the analysis was to plot sales versus date. This visualization offers crucial insights into the overall trend and seasonality in the sales data. From the plot, it was observed that there is an annual seasonality pattern, with sales volumes increasing in the summer season and decreasing more in the winter. Interestingly, sales volumes in the later part of the year were higher than in the early part. Over time, there is a discernible increasing trend in sales.

In the following sections of this report, I will further explore the dataset through various visualizations and statistical analyses to better understand the sales trends over time, the relationship between the variables, and to develop a predictive model for future sales.

3.2 Autocorrelation analysis

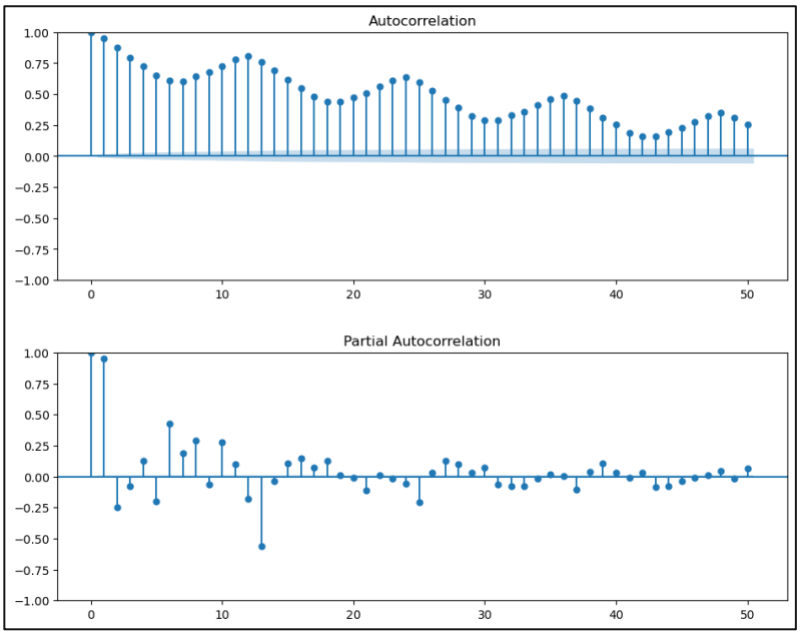


Figure 2. ACF/PACF of Dependent Variable

After preprocessing the data, I conducted an autocorrelation analysis, which includes plotting the Autocorrelation Function (ACF) and the Partial Autocorrelation Function (PACF).

The ACF plot shows the autocorrelation of sales with its own lags. From the ACF plot, I observed a strong positive autocorrelation that gradually decreases over subsequent lags but remains positive. This implies that the sales data is highly dependent on its previous values. I also detected a cyclical pattern, which could indicate seasonality in the data.

The PACF, on the other hand, represents the correlation between the sales and its lags after removing the effects of any shorter lags. The PACF plot showed a strong positive correlation at the first lag, followed by a significant drop to a negative correlation at the second lag. After some fluctuations, the correlation values approached zero. This suggests that the current sales value has a strong direct relationship with its immediate previous value, but this relationship diminishes sharply for the further past values.

Based on these observations, it appears that an Autoregressive Moving Average (ARMA) or Seasonal Autoregressive Integrated Moving Average (SARIMA) model might be suitable for this data.

Furthermore, the order of the Autoregressive (AR) part might be around 1, given the significant drop in the PACF after the first lag. However, I will need to conduct further analysis to confirm the exact order and to determine the Moving Average (MA) part.

3.3 Correlation matrix

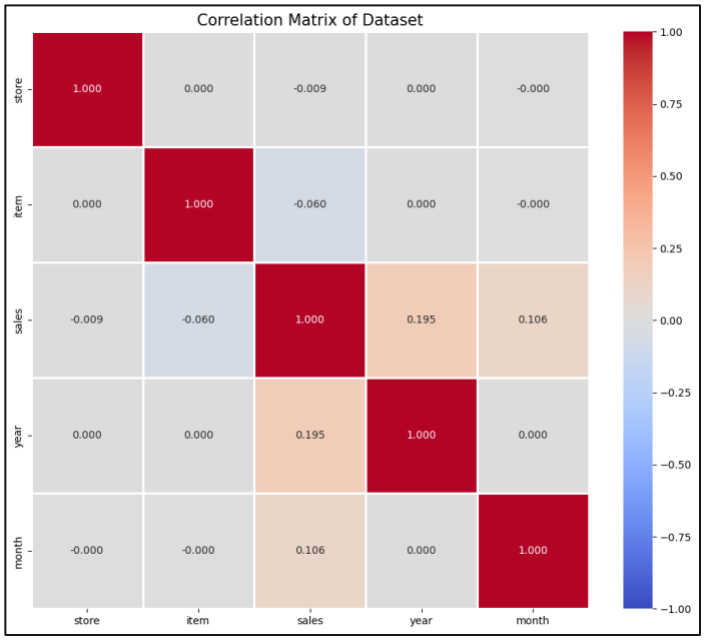


Figure 3. Correlation Matrix

Next, I examined the correlation matrix as a part of my exploratory data analysis. From the correlation matrix, I observed that 'sales' has a weak positive correlation with both 'year' and 'month'. This suggests a slight tendency for sales to increase over the years and within the year. However, the correlations are relatively weak, indicating that these trends might not be significant.

Interestingly, 'sales' has a weak negative correlation with 'item', suggesting that as the item ID increases, there might be a slight decrease in the sales volume. However, this correlation is also weak, suggesting that this relationship might not be significantly more influential than the relationships with 'year' and 'month'.

Overall, these correlations indicate that there might not be strong linear relationships between the features.

3.4 Stationary

3.4.1 Stationary test

I moved on to test for stationarity, which is a vital assumption in time series forecasting. To test for stationarity, I employed both the Augmented Dickey-Fuller (ADF) and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests.

ADF Statistic: -15.434349 p-value: 0.000000 Critical Values: 1%: -3.431 5%: -2.862 10%: -2.567	Results of KPSS Test: Test Statistic 1.284185 p-value 0.010000 Lags Used 94.000000 Critical Value (10%) 0.347000 Critical Value (5%) 0.463000 Critical Value (2.5%) 0.574000 Critical Value (1%) 0.739000

Table 3. ADF / KPSS Test of Original Data

The ADF test, which assumes that the dataset is non-stationary under the null hypothesis, returned a statistic of approximately -15, significantly below the value required for even the 1% significance level. The associated p-value, nearly zero, supports rejecting the null hypothesis. This suggests that the sales data does not have a unit root and thus is stationary.

In contrast, the KPSS test, which takes a stationary series as its null hypothesis, yielded a test statistic of roughly 1.28 and a p-value of approximately 0.01. These values indicate that the null hypothesis should be rejected at a 5% significance level, suggesting non-stationarity in the series.

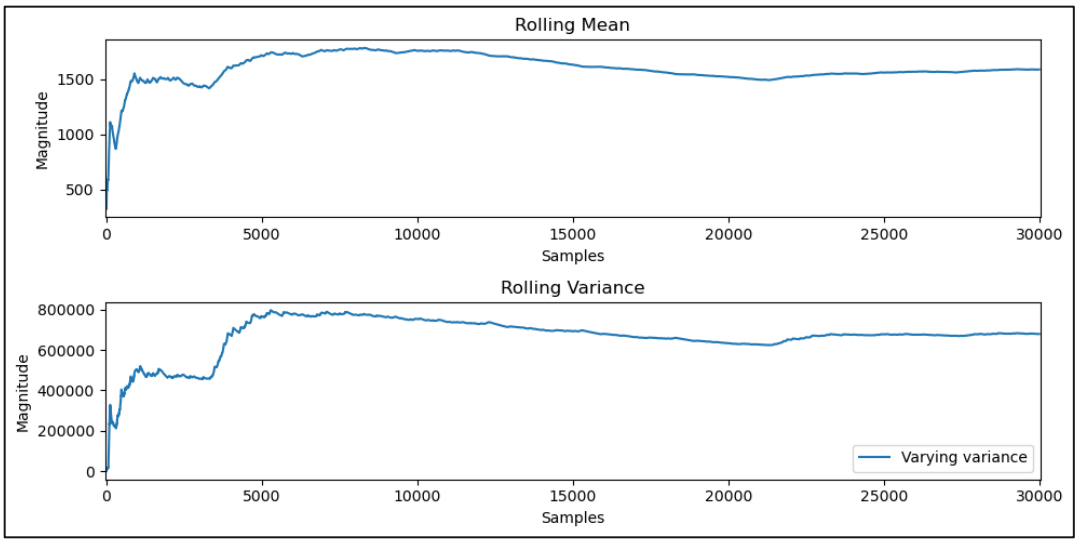


Figure 4. Rolling Mean and Variance of Original Data

In addition to these formal tests, I also scrutinized the rolling mean and variance of the sales data. While the rolling mean and variance seemed reasonably steady towards the end of the series, they were not perfectly constant. These observations suggest some degree of non-stationarity in the data, aligning with the mixed findings from the ADF and KPSS tests.

3.4.2 Data transformation

Following the stationarity tests, I decided to apply a differencing technique to enhance the stationarity of the sales series.

ADF Statistic: -30.343479	Results of KPSS Test:
p-value: 0.000000	Test Statistic 0.007733
Critical Values:	p-value 0.100000
1%: -3.431	Lags Used 217.000000
5%: -2.862	Critical Value (10%) 0.347000
10%: -2.567	Critical Value (5%) 0.463000
	Critical Value (2.5%) 0.574000
	Critical Value (1%) 0.739000

Table 4. ADF / KPSS Test of Differenced Data

After applying differencing to the sales data, I reran the ADF and KPSS tests. The ADF test on the differenced series presented a p-value close to zero, reinforcing my conclusion that the differenced series is indeed stationary.

Conversely, the KPSS test delivered a p-value of approximately 0.1, indicating that I fail to reject the null hypothesis, which suggests stationarity. This result aligns with the ADF test, adding further confidence that the differenced series is stationary.

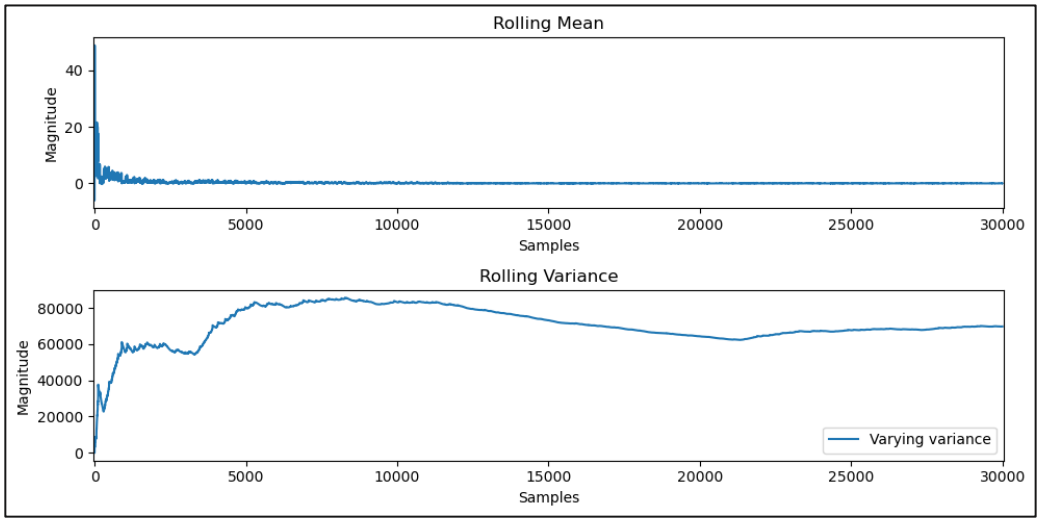


Figure 5. Rolling Mean and Variance of Differenced Data

Additionally, visual inspection of the rolling mean and variance of the differenced series offered additional evidence for stationarity. The rolling mean remained consistent, while the rolling variance appeared somewhat similar to the undifferenced data but appeared constant towards the end of the series. Given these observations, and the supporting results from the ADF and KPSS tests, I decided to retain this differenced version of the data for future modeling.

These efforts enhance the data's suitability for subsequent time series forecasting models, many of which require or assume stationarity in the input data.

4. Time Series Decomposition

Following the differencing process, I employed Seasonal and Trend decomposition using Loess (STL) on the differenced data.

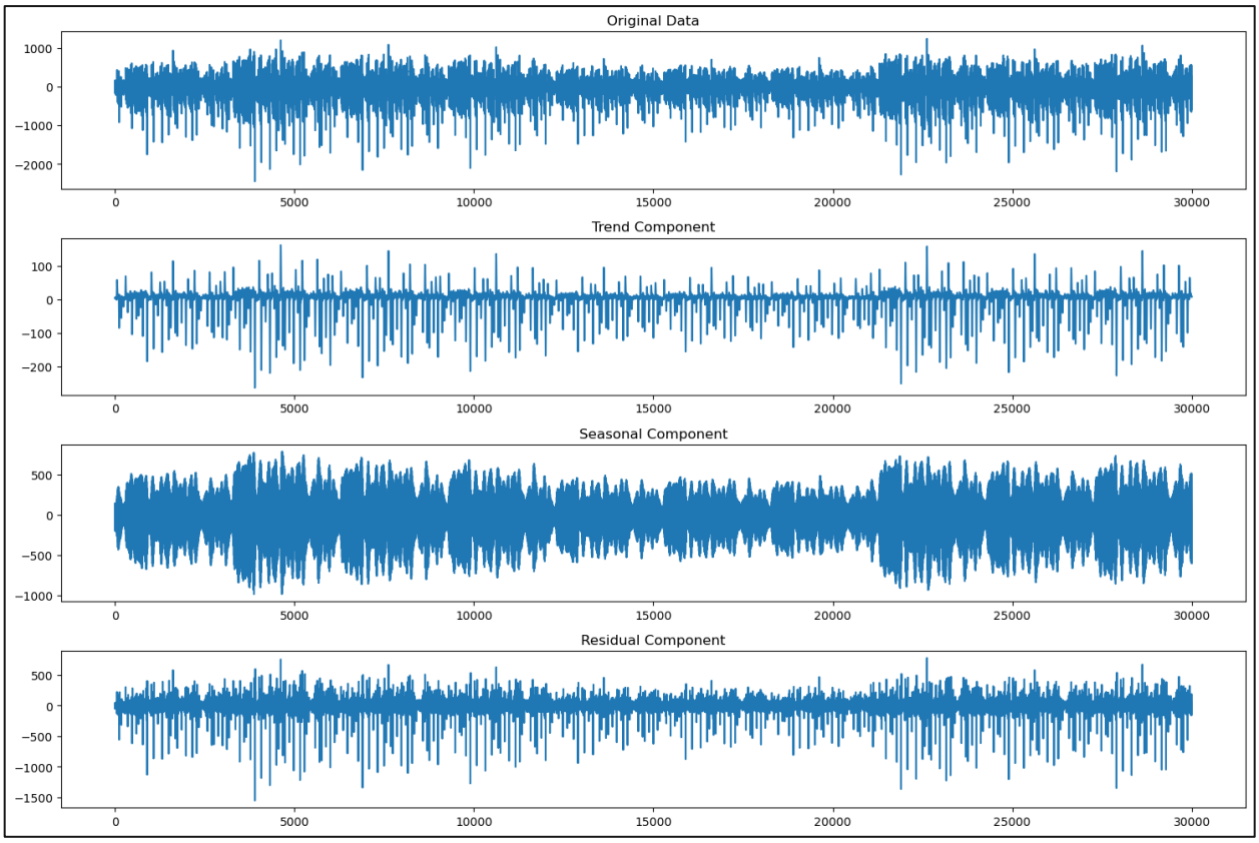


Figure 6. Time Series Decomposition of Differenced Data

```
The strength of trend after diff is 0.20546076000531144
The strength of seasonality after diff is 0.8595650934456687
```

The results of the STL decomposition indicate that the series still retains significant seasonality post-differencing. The strength of the trend is approximately 0.21, suggesting a modest presence of trend in the series. However, the seasonality strength is much higher, sitting around 0.86. This high value implies a strong seasonal component, even after the differencing process.

Having noticed the remaining seasonality in the differenced series, I decided to take an additional step to control this aspect. I subtracted the seasonal component from my differenced series, effectively creating a seasonally adjusted series.

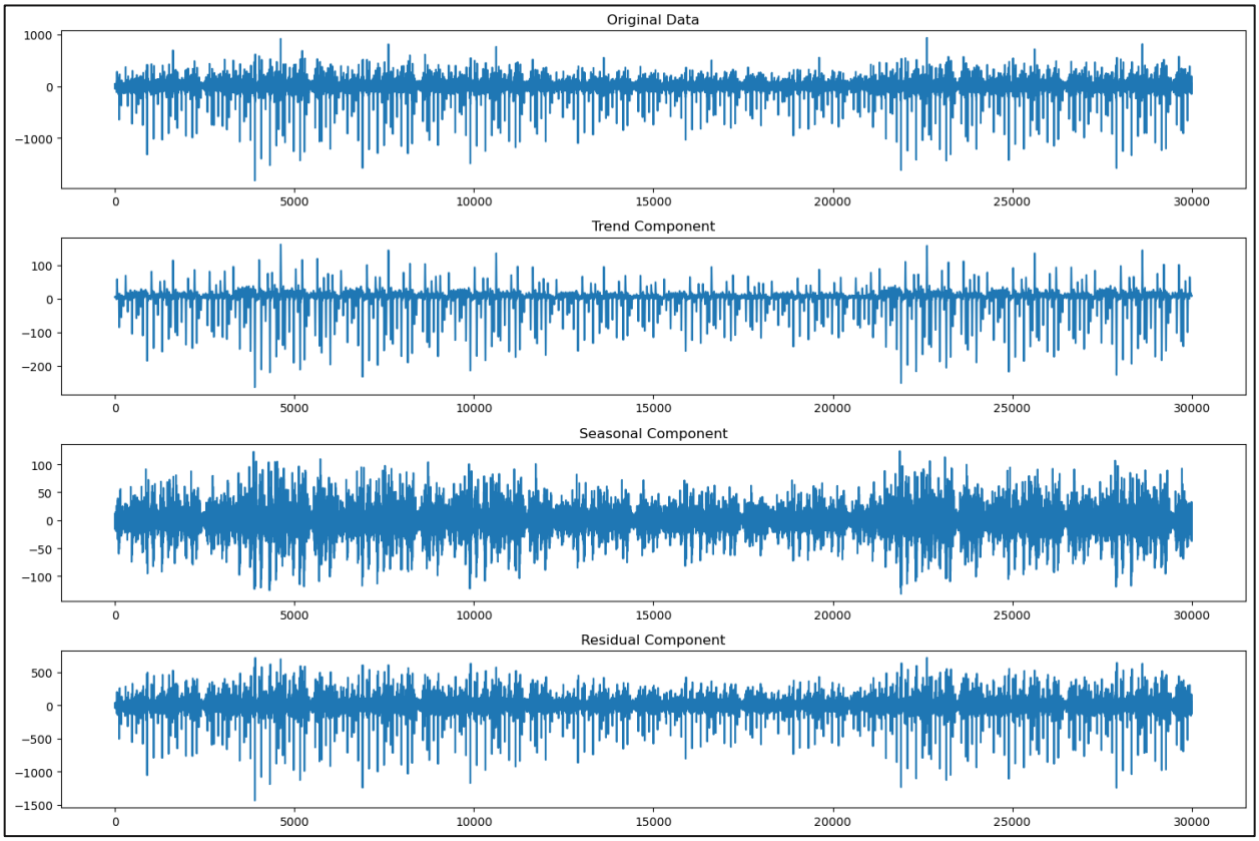


Figure 7. Time Series Decomposition after Adjusted Seasonality

```
The strength of trend after adjusted seasonality is 0.2140071968250049
The strength of seasonality after adjusted seasonality is 0.05010816269409557
```

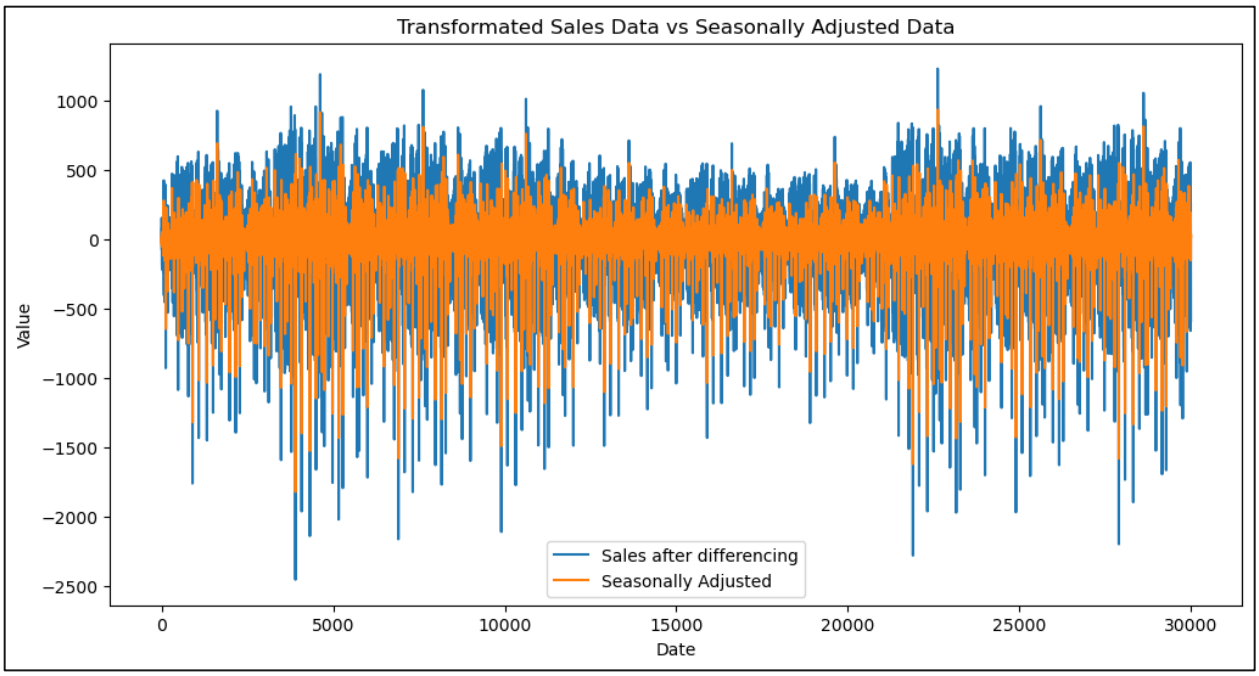


Figure 8. Seasonality Adjusted vs Difference Sales Data

To further assess the effectiveness of this operation, I performed another round of STL decomposition on the newly created seasonally adjusted series. The resulting decomposition presented a significantly reduced seasonal component, indicating that the seasonal adjustment was successful.

This step of creating a seasonally adjusted series is crucial for dealing with the strong seasonality present in the data. With the seasonal component effectively managed, it paves the way for a more robust analysis and modeling that doesn't have to heavily account for seasonality.

5. Modeling

5.1 Data splitting

```
X Train set size: 24000, X Test set size: 6000, y Train set size: 24000, y Test set size: 6000
```

Before I start modeling, I prepared the data for training and testing. I split the data, so that 80% of it was used for training and 20% for testing. In total, the training set contained 24,000 observations and the test set contained 6,000 observations.

5.2 Holt-winters method

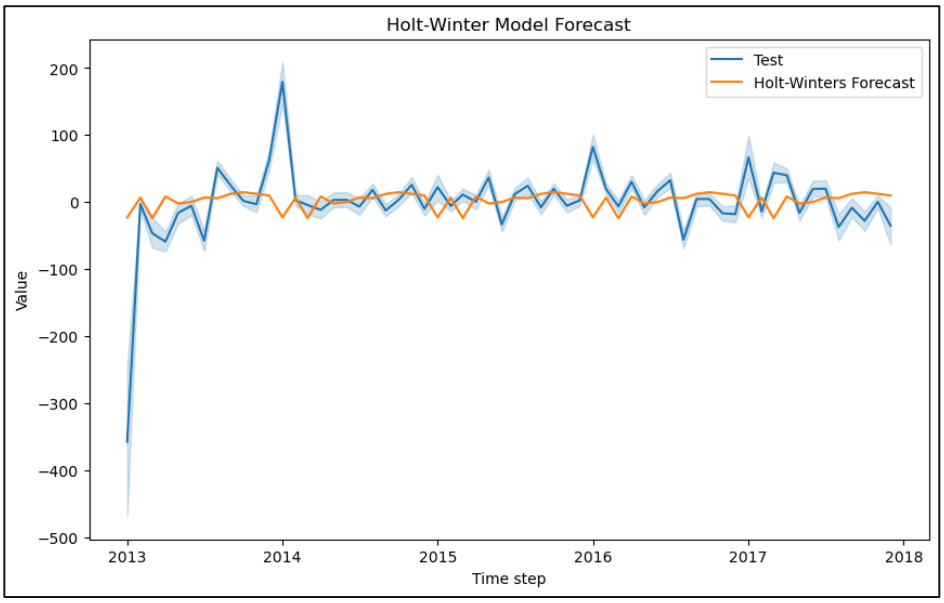


Table 5. Holt-Winters Method Forecast

```
Holt-Winter Model MSE: 13713.427362386321
```

The first model I tried was the Holt-Winters model, which is a time series forecasting method that accounts for trend and seasonality. The model was trained using the training data, and then it was used to forecast sales in the test data. The performance of the model was evaluated using MSE, a common metric for comparing the true and predicted values in regression tasks. The MSE for the Holt-Winters model turned out to be 13,713. This value serves as the benchmark for evaluating the performance of subsequent models that I will explore.

5.3 Basic forecasting method

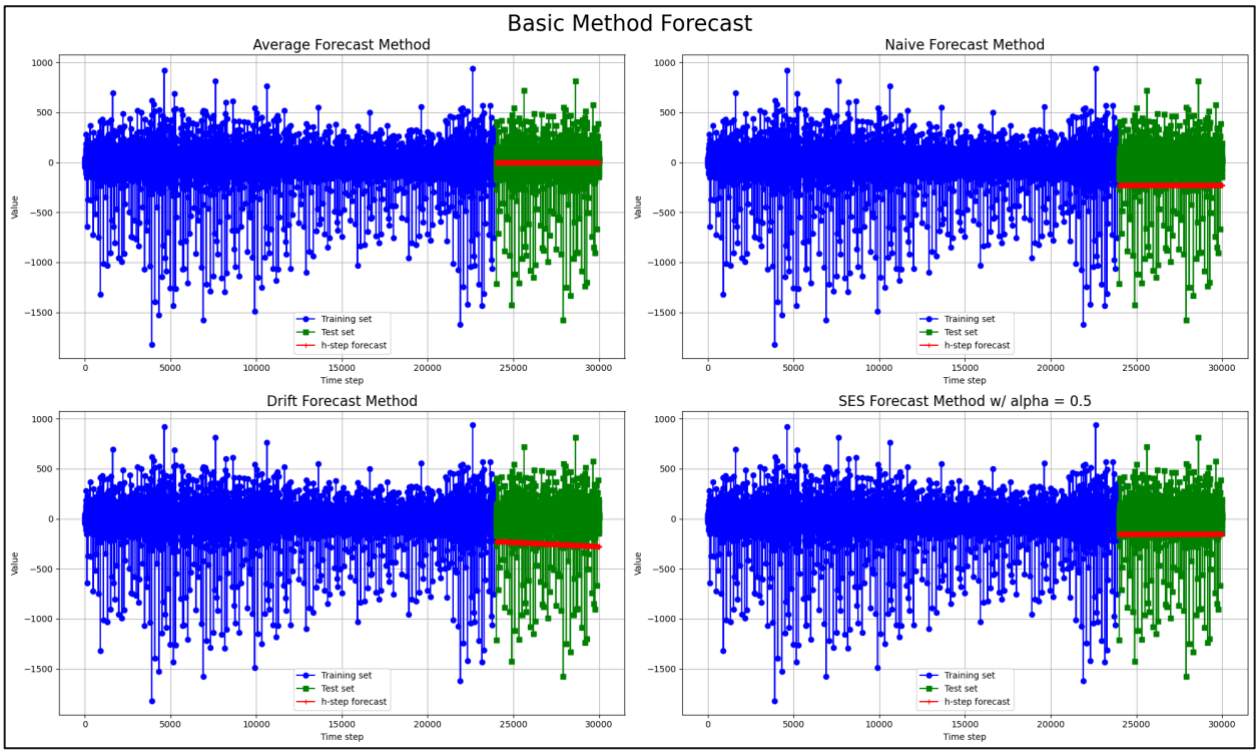


Table 6. Basic Forecasting Method Forecast

```
Average h-step ahead predction MSE is 13562.149629934996
Naive h-step ahead predction MSE is 64578.24995158316
Drift h-step ahead predction MSE is 76997.01209159679
SES h-step ahead predction MSE is 37033.847995196455
```

Next, I moved on to some basic forecasting methods, namely: Average, Naive, Drift, and Simple Exponential Smoothing (SES). Among these four methods, the Average method provided the best results for the dataset with an MSE of 13,562. This value was even lower than the MSE of 13,713 obtained from the more complex Holt-Winters model, which suggests that for this dataset, a simple averaging method might be more effective. This goes to show that more complex models are not always superior, and it's important to experiment with a range of models when dealing with time series forecasting.

5.4 Multiple linear regression method

5.4.1 Feature selection

Before proceeding with multiple linear regression, it's important to consider feature selection and check for multicollinearity.

5.4.1.1 SVD test

```
Singular values: [1.2183e+11 6.2465e+06 3.5750e+05 2.4750e+05]
```

The singular values obtained from the SVD were all quite large and far from zero, indicating that the features in my dataset are relatively independent of each other. There's no strong indication of severe multicollinearity based on these singular values.

5.4.1.2 Condition number

```
Condition number of X: {701.5975453854087}
```

A commonly used indicator of multicollinearity, the condition number for my data was 701.6. While this number indicates some degree of multicollinearity, it's below the often-used threshold of 1000 that signifies severe multicollinearity. Thus, it suggests that multicollinearity may not be a significant concern in my dataset.

5.4.1.3 PCA

```
Explained variance ratio:  [0.25 0.25 0.25 0.25]
```

The explained variance ratio for all the features was the same (25%), which suggests that each principal component contributes equally to the variance in the data. This could indicate that all the features are equally important, but further investigation would be necessary to confirm this.

5.4.1.4 VIF

	features	VIF
0	store	4.857134
1	item	4.122443
2	year	11.524987
3	month	4.545447

Following the initial multicollinearity analysis, I performed a Variance Inflation Factor (VIF) analysis to further investigate potential multicollinearity among the features. Most of the features

had a VIF less than 5, suggesting that multicollinearity is not a significant concern for them. However, the "year" feature had a VIF of around 11.5, which indicates potential multicollinearity.

5.4.1.5 Back elimination

```
AIC: 292322.986390641
BIC: 292363.41543618764
adjR2: 0.006189439818615194
***Baseline***
```

```
***Dropped store***
AIC: 292320.99120104115
BIC: 292353.33443747845
adjR2: 0.006230656310081684
```

```
***Dropped month***
AIC: 292320.9980606477
BIC: 292353.341297085
adjR2: 0.006230372273929308
['item', 'year']
```

To address these issues and refine the model, I applied the Backward Elimination process.

In the baseline model, which included all features, the AIC was 292322.99, BIC was 292363.42, and the adjusted R-squared value was 0.0062.

The 'store' feature was the first to be eliminated. The resulting model had an AIC of 292320.99, BIC of 292353.33, and the adjusted R-squared value slightly increased to 0.00623, indicating a marginal improvement in the model fit.

Next, the 'month' feature was dropped. The model with the remaining features had an AIC of 292320.99 and BIC of 292353.34, almost the same as the previous model. The adjusted R-squared value remained nearly constant at 0.00623.

Through the Back Elimination process, the 'store' and 'month' features were identified as contributing less to the model and were thus eliminated. The remaining features led to a slightly improved model fit while maintaining a simpler and more computationally efficient model.

This approach led me to two final features for the multiple linear regression model: 'item' and 'year'. I reassessed the VIF scores after the Backward Elimination process.

	features	VIF
0	item	4.122443
1	year	4.122443

All the remaining features, including 'year', now had a VIF below the threshold of 5. This suggests that any multicollinearity issues had been effectively managed, and the final features for the multiple linear regression model are 'item' and 'year'. This refined model, free from significant multicollinearity, is expected to provide more reliable results.

5.4.2 Multiple linear regression modeling

In this step, I evaluated the performance of the multiple linear regression model.

OLS Regression Results

Dep. Variable:		diff		R-squared:		0.006
Model:		OLS		Adj. R-squared:		0.006
Method:		Least Squares		F-statistic:		38.37
Date:		Wed, 10 May 2023		Prob (F-statistic):		4.67e-32
Time:		18:36:39		Log-Likelihood:		-1.4616e+05
No. Observations:		24000		AIC:		2.923e+05
Df Residuals:		23995		BIC:		2.924e+05
Df Model:		4				
Covariance Type:		nonrobust				
	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.217e+04	982.322	-12.387	0.000	-1.41e+04	-1.02e+04
store	-0.0209	0.301	-0.069	0.945	-0.611	0.569
item	0.0043	0.048	0.090	0.929	-0.089	0.098
year	6.0388	0.488	12.387	0.000	5.083	6.994
month	-0.0216	0.200	-0.108	0.914	-0.413	0.370
Omnibus:	21208.767		Durbin-Watson:		1.813	
Prob(Omnibus):	0.000		Jarque-Bera (JB):		2192881.771	
Skew:	-3.802		Prob(JB):		0.00	
Kurtosis:	49.207		Cond. No.		2.87e+06	

```
All features MSE: 13471.23
```

Table 7. Multiple Linear Regression for All Features

Initially, I included all the features in the model, which resulted in a MSE of 13471.23. This served as the benchmark for comparing the performance of the model after the feature selection process.

OLS Regression Results

Dep. Variable:		diff		R-squared:		0.006
Model:		OLS		Adj. R-squared:		0.006
Method:		Least Squares		F-statistic:		76.73
Date:		Wed, 10 May 2023		Prob (F-statistic):		6.06e-34
Time:		18:39:10		Log-Likelihood:		-1.4616e+05
No. Observations:		24000		AIC:		2.923e+05
Df Residuals:		23997		BIC:		2.923e+05
Df Model:		2				
Covariance Type:		nonrobust				
	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.217e+04	982.279	-12.388	0.000	-1.41e+04	-1.02e+04
item	0.0043	0.048	0.090	0.929	-0.089	0.098
year	6.0388	0.487	12.388	0.000	5.083	6.994
Omnibus:	21201.103		Durbin-Watson:		1.813	
Prob(Omnibus):	0.000		Jarque-Bera (JB):		2191188.035	
Skew:	-3.800		Prob(JB):		0.00	
Kurtosis:	49.189		Cond. No.		2.87e+06	

```
Feature Selection MSE: 13471.17
```

Table 8. Multiple Linear Regression for Feature Selection Features

Following the feature selection, I retained only the 'item' and 'year' features in the model and recalculated the MSE. The MSE with the selected features was 13471.17, indicating a slight improvement over the initial MSE. While this reduction in the MSE may seem insignificant, it's essential to remember that this improved performance was achieved with fewer features. This simplifies the model and makes it more computationally efficient, thereby enhancing its overall utility.

5.4.3 Model forecast

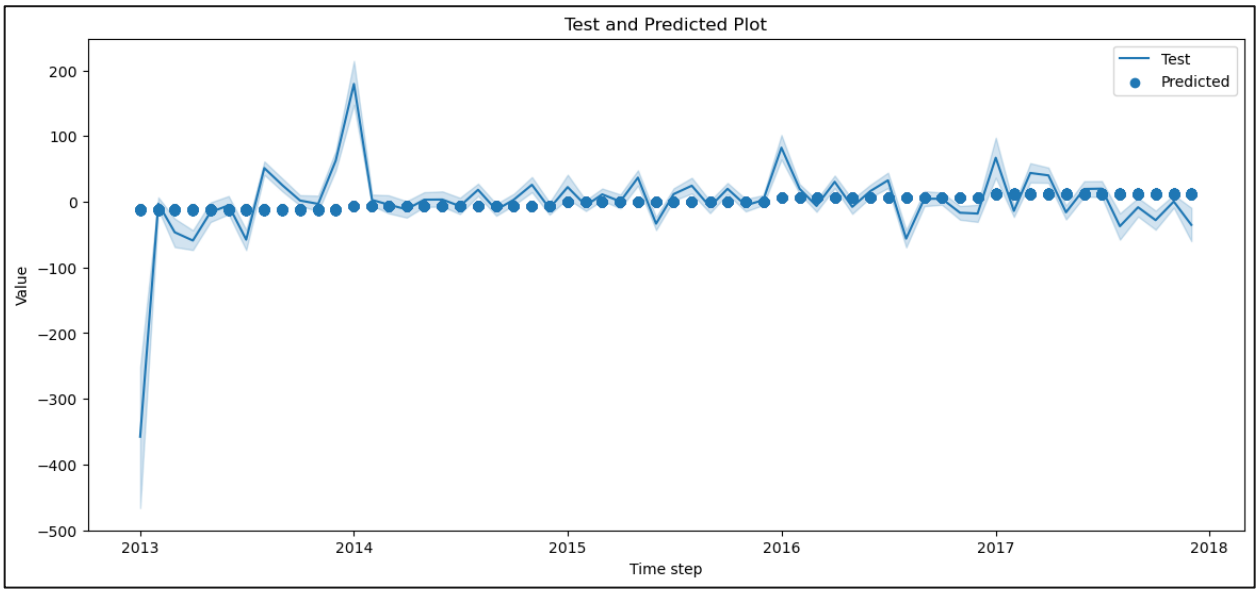


Figure 9. Multiple Linear Regression Method Forecast

To supplement the numerical evaluation of the multiple linear regression model, I performed a visual assessment of the model. First, I plotted the model's predicted values against the actual test values. This comparison shows that, although the predictions do not match the actual values perfectly, they follow the same general pattern. This deviation is expected due to the inherent variability and noise in real-world data. Despite this, the model appears to capture the overall trend effectively.

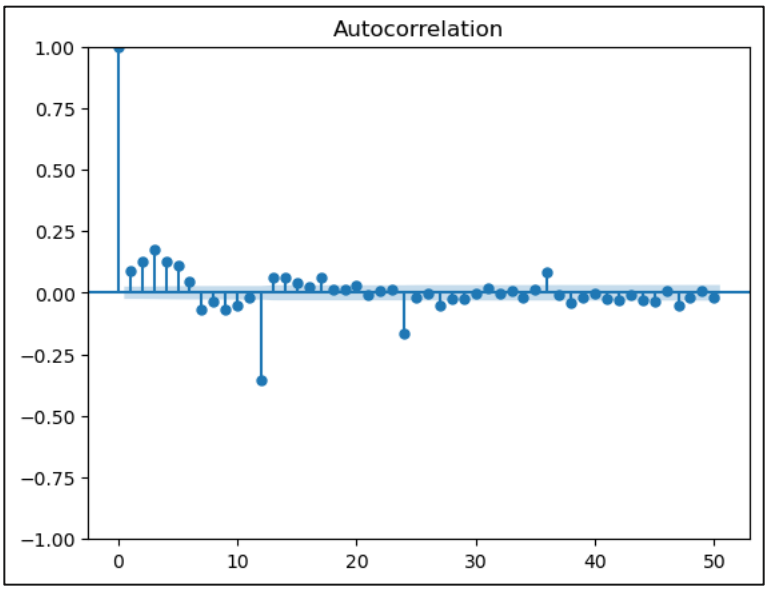


Figure 10. MLR residuals ACF Plot

Next, I examined the ACF plot of the residuals. Ideally, the residuals should present no clear pattern and decay to zero gradually. The ACF plot of the residuals from the multiple linear regression model reflects this desired characteristic.

```
Residuals Mean: -0.04
```

Additionally, the mean of the residuals is close to zero. This is an essential check because the mean of the residuals being zero is one of the key assumptions of linear regression.

Collectively, these visual checks and the residual analysis suggest that the multiple linear regression model exhibits a decent fit to the data. When compared to the Holt-Winters and Average models, the multiple linear regression model has the lowest MSE, indicating superior performance.

5.5 ARMA / ARIMA / SARIMA method

5.5.1 Model order selection

5.5.1.1 ACF/PACF

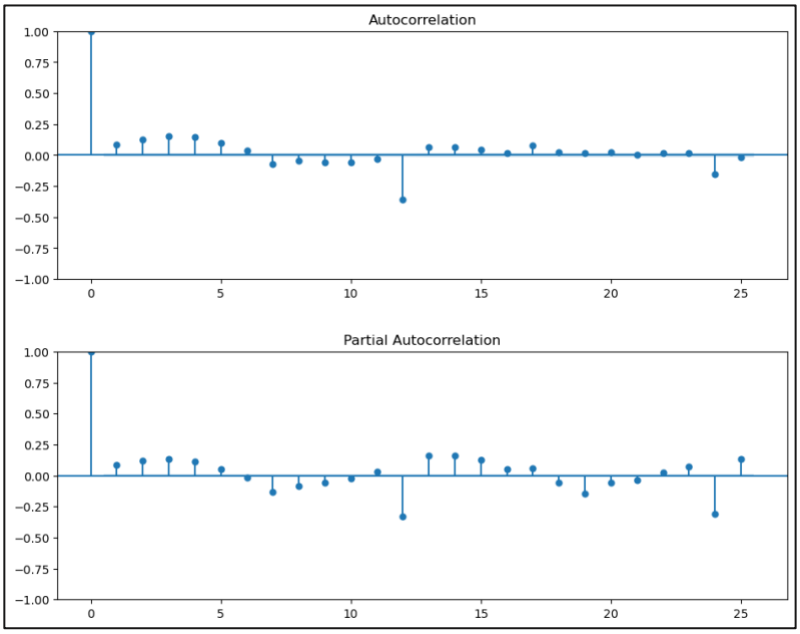


Figure 11. Training Data ACF/PACF Plot

Analyzing the ACF, PACF, and Generalized Partial Autocorrelation (GPAC) plots, I identified that the sales data might be suitably modeled using an ARMA model. This is suggested by the tail-off shapes observed in both the ACF and PACF plots. In addition, the ACF and PACF plots reveal a clear seasonality of 12 months, indicating a yearly cycle in the data.

5.5.1.2 GPAC table

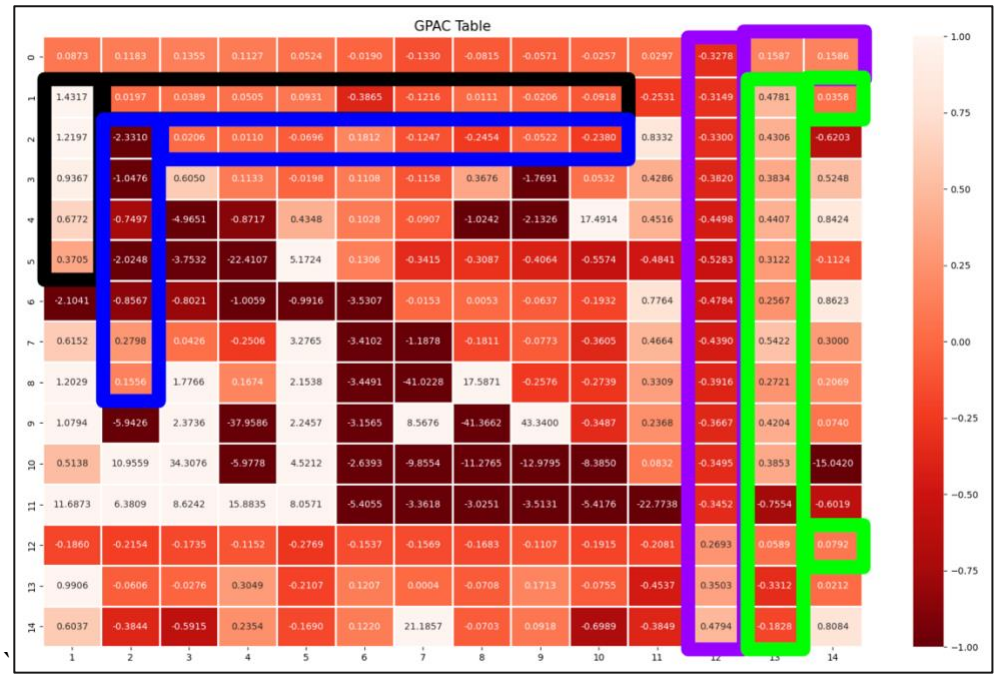


Figure 12. GPAC Table

From the GPAC analysis, I identified potential orders for the ARMA model. I am considering autoregressive orders p of 1, 12, and 13, and moving average orders q of 0 and 1. Given the observed seasonality of 12 months, I will explore various ARIMA and SARIMA models based on these potential orders to select the best fitting model.

5.5.2 Selecting the best ARIMA model

SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	ARIMA(12, 0, 0)	Log Likelihood	-143841.042
Date:	Tue, 09 May 2023	AIC	287710.083
Time:	21:32:32	BIC	287823.284
Sample:	0	HQIC	287746.795
	- 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0762	0.776	0.098	0.922	-1.445	1.597
ar.L1	0.0319	0.008	3.903	0.000	0.016	0.048
ar.L2	0.0806	0.007	11.115	0.000	0.066	0.095
ar.L3	0.1256	0.007	17.569	0.000	0.112	0.140
ar.L4	0.1205	0.007	17.163	0.000	0.107	0.134
ar.L5	0.0458	0.008	6.013	0.000	0.031	0.061
ar.L6	0.0053	0.006	0.813	0.416	-0.007	0.018
ar.L7	-0.0964	0.007	-13.229	0.000	-0.111	-0.082
ar.L8	-0.0336	0.009	-3.829	0.000	-0.051	-0.016
ar.L9	-0.0116	0.008	-1.492	0.136	-0.027	0.004
ar.L10	0.0028	0.007	0.373	0.709	-0.012	0.017
ar.L11	0.0370	0.006	5.816	0.000	0.025	0.049
ar.L12	-0.3281	0.005	-63.007	0.000	-0.338	-0.318
sigma2	9407.5185	40.703	231.127	0.000	9327.742	9487.295

Ljung-Box (L1) (Q):	65.17	Jarque-Bera (JB):	1943264.86
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.72	Skew:	-3.84
Prob(H) (two-sided):	0.00	Kurtosis:	46.41

Table 9. ARIMA(12,0,0) Model Summary

SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	ARIMA(13, 0, 1)	Log Likelihood	−143126.546
Date:	Wed, 10 May 2023	AIC	286285.091
Time:	20:57:23	BIC	286414.464
Sample:	0	HQIC	286327.047
	− 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0740	1.275	0.058	0.954	−2.425	2.573
ar.L1	0.6660	0.019	34.342	0.000	0.628	0.704
ar.L2	0.0552	0.009	6.208	0.000	0.038	0.073
ar.L3	0.0791	0.009	9.194	0.000	0.062	0.096
ar.L4	0.0511	0.009	5.703	0.000	0.034	0.069
ar.L5	−0.0163	0.008	−1.925	0.054	−0.033	0.000
ar.L6	−0.0022	0.008	−0.285	0.775	−0.017	0.013
ar.L7	−0.1004	0.007	−13.833	0.000	−0.115	−0.086
ar.L8	0.0124	0.009	1.410	0.158	−0.005	0.030
ar.L9	−0.0158	0.008	−1.981	0.048	−0.032	−0.000
ar.L10	−0.0151	0.008	−1.913	0.056	−0.031	0.000
ar.L11	0.0198	0.008	2.507	0.012	0.004	0.035
ar.L12	−0.3554	0.006	−56.804	0.000	−0.368	−0.343
ar.L13	0.3816	0.006	59.684	0.000	0.369	0.394
ma.L1	−0.6051	0.020	−30.704	0.000	−0.644	−0.567
sigma2	8859.2303	38.342	231.056	0.000	8784.081	8934.380

Ljung-Box (L1) (Q):	13.77	Jarque-Bera (JB):	1726328.18
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.72	Skew:	−3.63
Prob(H) (two-sided):	0.00	Kurtosis:	43.91

Table 10. ARIMA(13,0,1) Model Summary

SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	ARIMA(12, 0, 12)	Log Likelihood	-137606.467
Date:	Tue, 09 May 2023	AIC	275264.935
Time:	21:37:52	BIC	275475.166
Sample:	0	HQIC	275333.114
	- 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0737	0.101	0.731	0.465	-0.124	0.271
ar.L1	0.0107	0.011	0.971	0.332	-0.011	0.032
ar.L2	0.0545	0.010	5.536	0.000	0.035	0.074
ar.L3	0.0911	0.010	8.950	0.000	0.071	0.111
ar.L4	0.1314	0.010	13.819	0.000	0.113	0.150
ar.L5	0.1000	0.010	9.801	0.000	0.080	0.120
ar.L6	0.1118	0.012	9.383	0.000	0.088	0.135
ar.L7	0.0142	0.008	1.698	0.090	-0.002	0.031
ar.L8	0.1268	0.009	14.068	0.000	0.109	0.145
ar.L9	0.0827	0.011	7.274	0.000	0.060	0.105
ar.L10	0.0511	0.010	5.024	0.000	0.031	0.071
ar.L11	0.0508	0.008	5.975	0.000	0.034	0.067
ar.L12	-0.0331	0.006	-5.870	0.000	-0.044	-0.022
ma.L1	0.1090	0.004	24.570	0.000	0.100	0.118
ma.L2	0.1280	0.004	33.721	0.000	0.121	0.135
ma.L3	0.1324	0.005	29.107	0.000	0.124	0.141
ma.L4	0.1029	0.004	23.123	0.000	0.094	0.112
ma.L5	0.0517	0.005	11.149	0.000	0.043	0.061
ma.L6	-0.0232	0.004	-5.260	0.000	-0.032	-0.015
ma.L7	-0.0922	0.004	-22.377	0.000	-0.100	-0.084
ma.L8	-0.1231	0.005	-26.675	0.000	-0.132	-0.114
ma.L9	-0.1294	0.004	-30.653	0.000	-0.138	-0.121
ma.L10	-0.0978	0.005	-20.412	0.000	-0.107	-0.088
ma.L11	-0.0633	0.004	-14.700	0.000	-0.072	-0.055
ma.L12	-0.9693	0.004	-219.868	0.000	-0.978	-0.961
sigma2	7141.5502	38.558	185.217	0.000	7065.979	7217.122

Ljung-Box (L1) (Q):	0.39	Jarque-Bera (JB):	545050.62
Prob(Q):	0.53	Prob(JB):	0.00
Heteroskedasticity (H):	0.73	Skew:	-0.84
Prob(H) (two-sided):	0.00	Kurtosis:	26.29

Table 11. ARIMA(12,0,12) Model Summary

After a careful examination of the summaries of various ARIMA models, I have determined that the ARIMA(12,0,12) model is the best fit for the sales data. This decision is primarily based on the model's superior performance, which is evidenced by its comparatively lower AIC and BIC scores. To validate the selection of the ARMA(12,0,12) model, I will examine the ACF and PACF of the residuals.

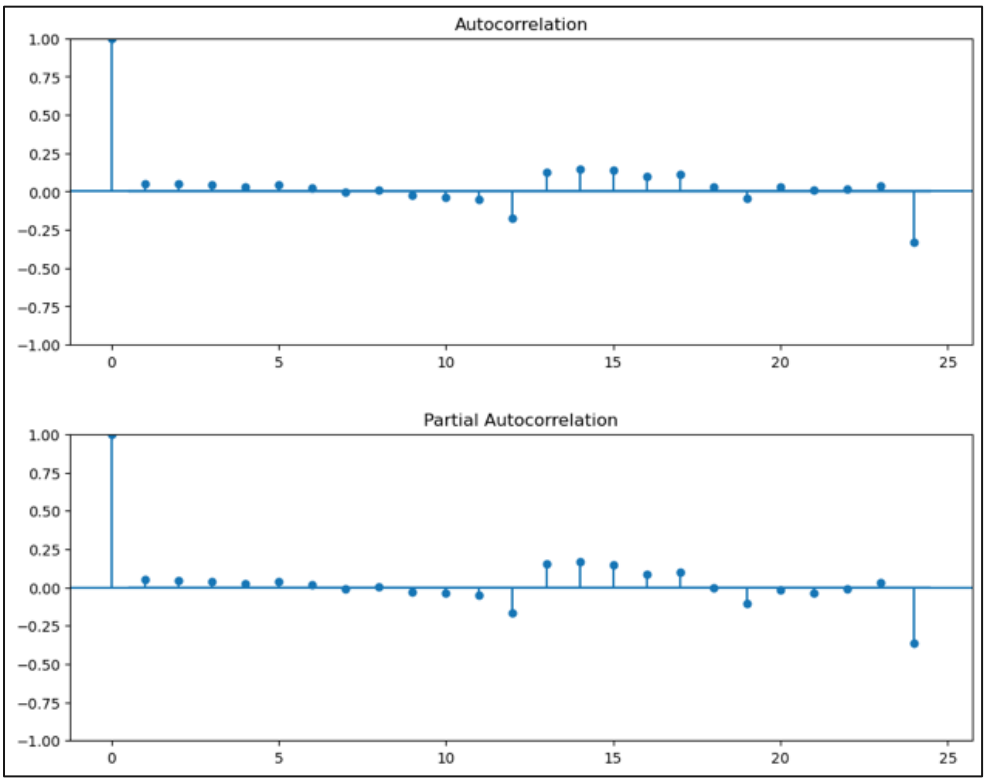


Figure 13. ARIMA(12,0,0) Model Residuals ACF/PACF Plot

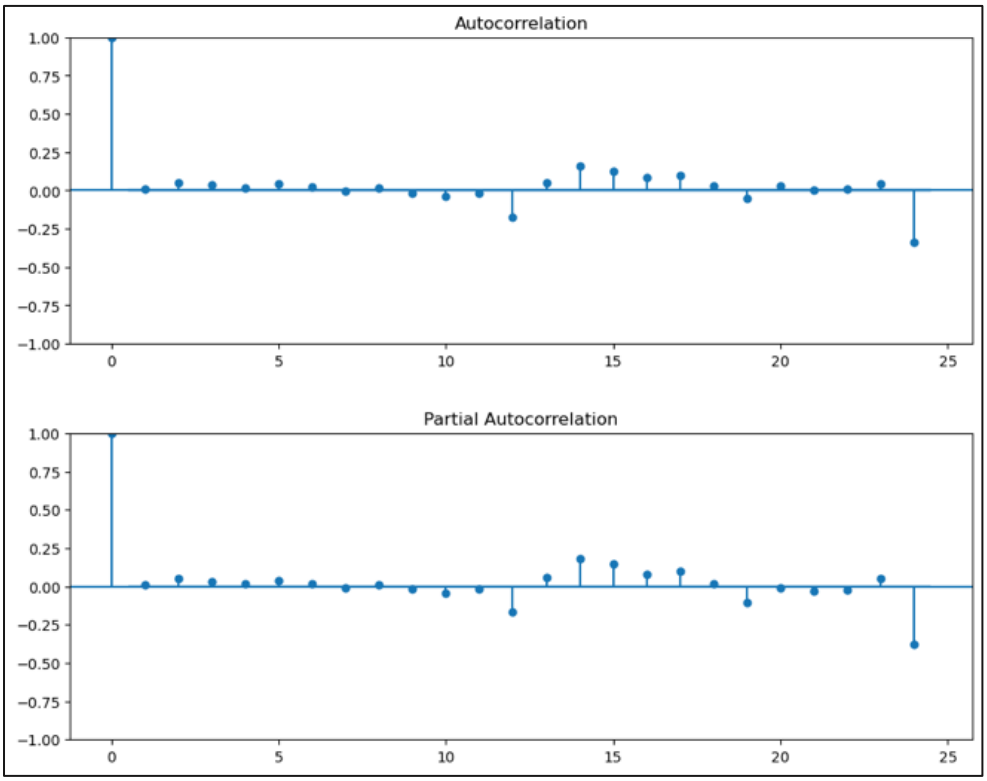


Figure 14. ARIMA(13,0,1) Model Residuals ACF/PACF Plot

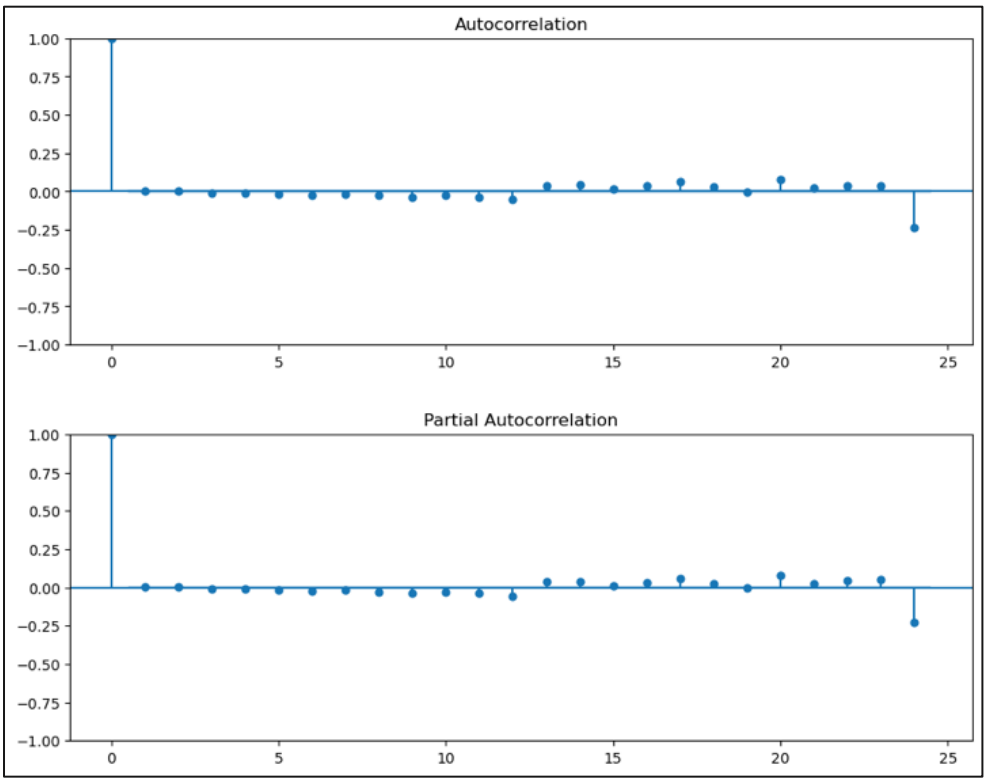


Figure 15. ARIMA(12,0,12) Model Residuals ACF/PACF Plot

```
ARIMA(12, 0, 0) MSE: 13489.630466177841
ARIMA(13, 0, 1) MSE: 13485.529520228158
ARIMA(12, 0, 12) MSE: 13361.489978925487
```

Ideally, I'd like to see no significant correlations in these plots for a well-fitted model. Upon inspection, the ARIMA(12,0,12) model showed the most satisfactory results, with no discernible patterns or significant correlations in the residuals.

Further, I evaluated the MSE for each of the considered models. The ARIMA(12,0,12) model not only produced the most satisfactory residual plots, but it also had the lowest MSE among the models. This quantitative evidence further solidifies its position as the best fitting model for the sales dataset.

5.5.3 Selecting the best SARIMA model

SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	SARIMAX(1, 0, [1], 12)	Log Likelihood	-140960.161
Date:	Tue, 09 May 2023	AIC	281926.321
Time:	21:48:48	BIC	281950.579
Sample:	0	HQIC	281934.188
	- 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.S.L12	0.2528	0.004	68.509	0.000	0.246	0.260
ma.S.L12	-0.9735	0.001	-674.548	0.000	-0.976	-0.971
sigma2	7387.7522	27.556	268.100	0.000	7333.744	7441.761

Ljung-Box (L1) (Q):	1552.24	Jarque-Bera (JB):	607787.23
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.72	Skew:	-1.74
Prob(H) (two-sided):	0.00	Kurtosis:	27.41

Figure 16. SARIMA(0,0,0)(1,0,1,12) Model Summary

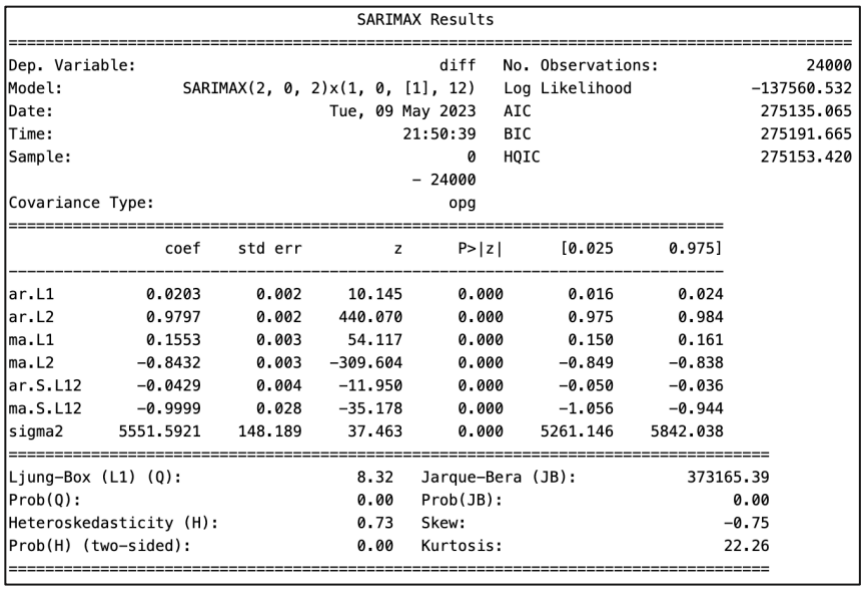
SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	SARIMAX(1, 0, 1)x(1, 0, 1, 12)	Log Likelihood	-137556.575
Date:	Tue, 09 May 2023	AIC	275123.151
Time:	21:49:33	BIC	275163.580
Sample:	0	HQIC	275136.262
	- 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9781	0.001	661.139	0.000	0.975	0.981
ma.L1	-0.8170	0.003	-247.431	0.000	-0.823	-0.811
ar.S.L12	-0.0449	0.003	-15.208	0.000	-0.051	-0.039
ma.S.L12	-0.9995	0.001	-756.938	0.000	-1.002	-0.997
sigma2	5550.7186	18.763	295.838	0.000	5513.944	5587.493

Ljung-Box (L1) (Q):	13.18	Jarque-Bera (JB):	409852.64
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.73	Skew:	-0.55
Prob(H) (two-sided):	0.00	Kurtosis:	23.21

Figure 17. SARIMA(1,0,1)(1,0,1,12) Model Summary



SARIMAX Results

Dep. Variable:	diff	No. Observations:	24000
Model:	SARIMAX(2, 0, 2)x(1, 0, [1], 12)	Log Likelihood	-137560.532
Date:	Tue, 09 May 2023	AIC	275135.065
Time:	21:50:39	BIC	275191.665
Sample:	0	HQIC	275153.420
	- 24000		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0203	0.002	10.145	0.000	0.016	0.024
ar.L2	0.9797	0.002	440.070	0.000	0.975	0.984
ma.L1	0.1553	0.003	54.117	0.000	0.150	0.161
ma.L2	-0.8432	0.003	-309.604	0.000	-0.849	-0.838
ar.S.L12	-0.0429	0.004	-11.950	0.000	-0.050	-0.036
ma.S.L12	-0.9999	0.028	-35.178	0.000	-1.056	-0.944
sigma2	5551.5921	148.189	37.463	0.000	5261.146	5842.038

Ljung-Box (L1) (Q):	8.32	Jarque-Bera (JB):	373165.39
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.73	Skew:	-0.75
Prob(H) (two-sided):	0.00	Kurtosis:	22.26

Figure 18. SARIMA(2,0,2)(1,0,1,12) Model Summary

After a thorough examination of the summaries for three different SARIMA models, I've decided that SARIMA(1,0,1)(1,0,1,12) model provides the best fit for the data. This decision is primarily driven by the model's superior performance, as demonstrated by the AIC and BIC scores. To validate my previous model selection, I will closely examine the ACF and PACF of the residuals for each model.

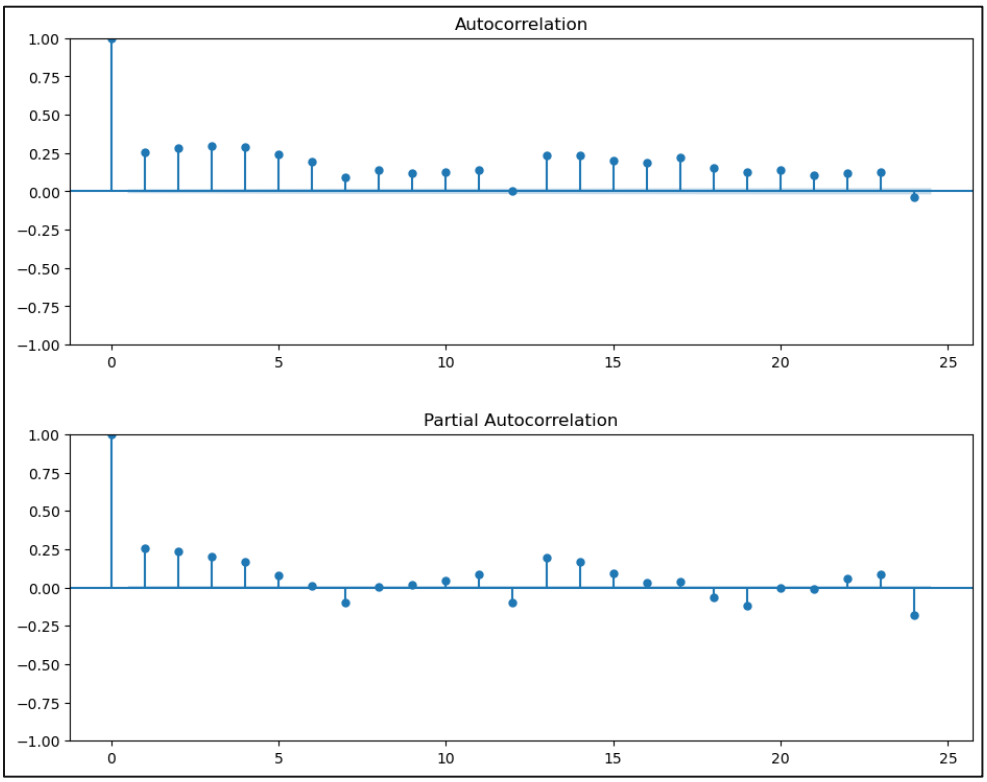


Figure 19. SARIMA(0,0,0)(1,0,1,12) Model Residuals ACF/PACF Plot

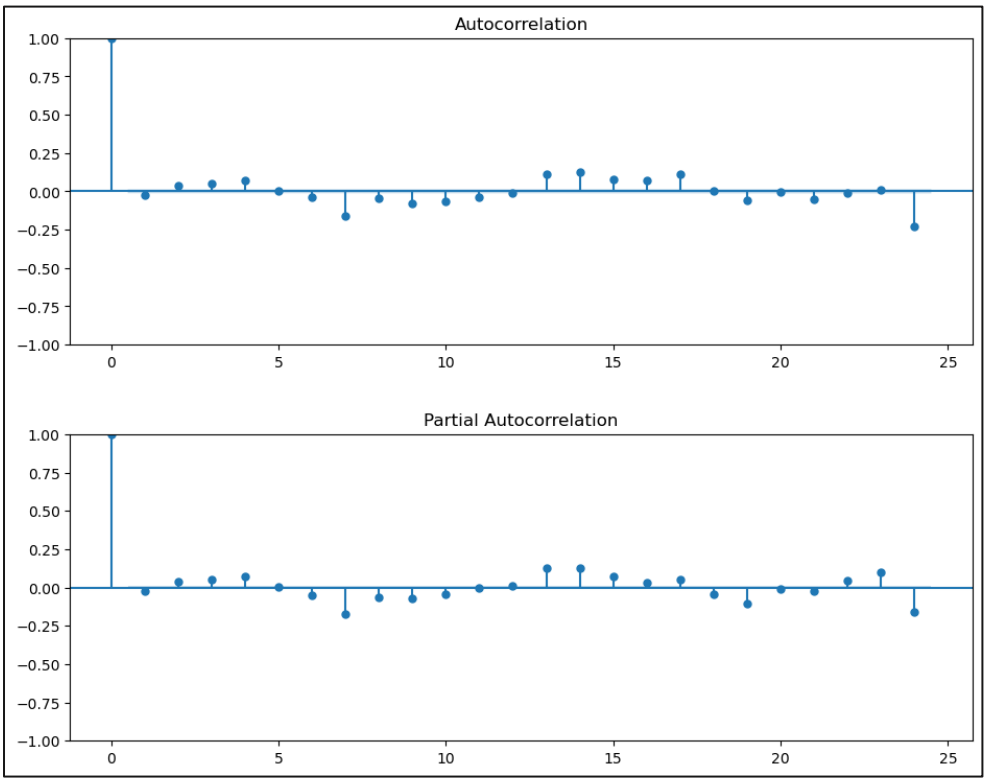


Figure 20. SARIMA(1,0,1)(1,0,1,12) Model Residuals ACF/PACF Plot

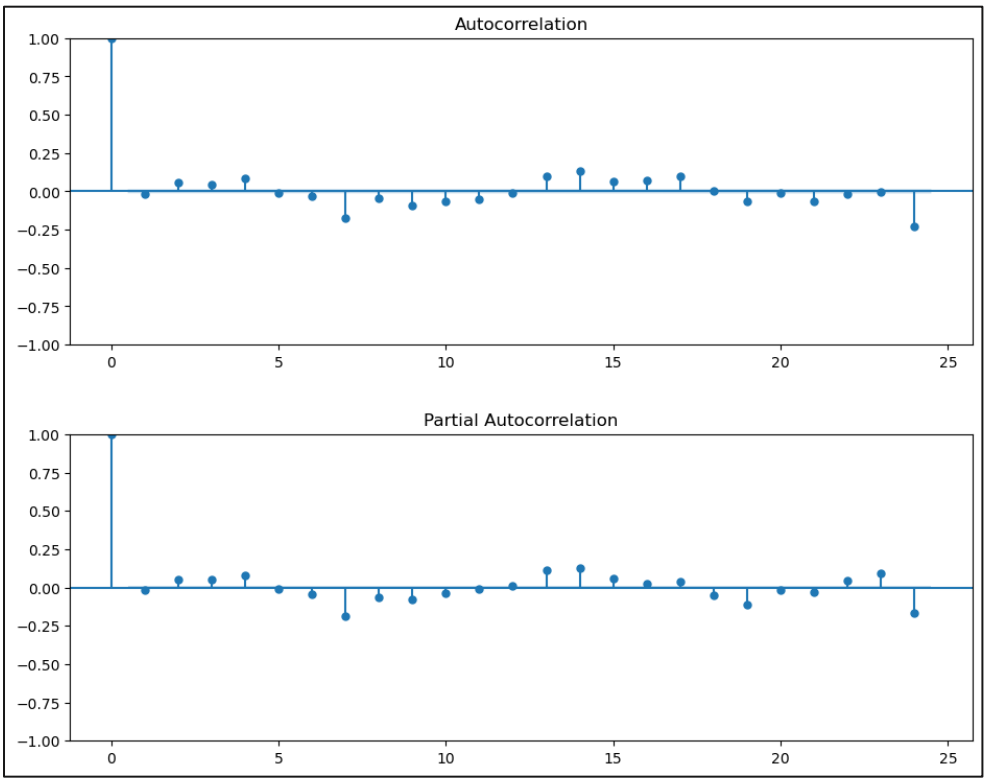


Figure 21. SARIMA(2,0,2)(1,0,1,12) Model Residuals ACF/PACF Plot

```
SARIMA(0,0,0)(1, 0, 1, 12) MSE: 13420.754837581664
SARIMA(1,0,1)(1, 0, 1, 12) MSE: 13349.477734212884
SARIMA(2,0,2)(1, 0, 1, 12) MSE: 13349.819723167504
```

Though the ACF and PACF plots, for SARIMA(1,0,1)(1,0,1,12) and SARIMA(2,0,2)(1,0,1,12) models appear somewhat similar, my chosen model, SARIMA(1,0,1)(1,0,1,12), has a significantly lower MSE. Therefore, this further analysis strengthens my decision to select the SARIMA(1,0,1)(1,0,1,12) model as the optimal choice for this dataset.

5.5.4 ARIMA vs SARIMA

5.5.4.1 Confidence interval

Confidence Intervals:

	0	1
const	−0.123842	0.271284
ar.L1	−0.010867	0.032210
ar.L2	0.035183	0.073746
ar.L3	0.071155	0.111056
ar.L4	0.112731	0.149995
ar.L5	0.079993	0.119983
ar.L6	0.088411	0.135097
ar.L7	−0.002189	0.030559
ar.L8	0.109173	0.144517
ar.L9	0.060430	0.105005
ar.L10	0.031193	0.071098
ar.L11	0.034110	0.067408
ar.L12	−0.044117	−0.022031
ma.L1	0.100335	0.117730
ma.L2	0.120604	0.135489
ma.L3	0.123521	0.141357
ma.L4	0.094166	0.111608
ma.L5	0.042576	0.060739
ma.L6	−0.031837	−0.014552
ma.L7	−0.100242	−0.084096
ma.L8	−0.132176	−0.114082
ma.L9	−0.137662	−0.121115
ma.L10	−0.107232	−0.088443
ma.L11	−0.071747	−0.054866
ma.L12	−0.977914	−0.960633
sigma2	7065.978504	7217.121824

VS

Confidence Intervals:

	0	1
ar.L1	0.975176	0.980975
ma.L1	−0.823477	−0.810534
ar.S.L12	−0.050741	−0.039155
ma.S.L12	−1.002099	−0.996923
sigma2	5513.944446	5587.492851

Table 12. Confidence Interval ARIMA(12,0,12) vs SARIMA(1,0,1)(1,0,1,12)

An examination of the confidence intervals for the ARIMA model revealed some intervals crossing zero, implying potential insignificance of certain parameters. In contrast, the SARIMA model displayed no such issue, suggesting that all its parameters were significant.

5.5.4.2 Mean of model residual

```
Mean of arima_Residuals is -0.3120557589268745

Mean of sarima_Residuals is -0.4094663135184187
```

The mean of the ARIMA model's residuals was closer to zero than that of the SARIMA model. This indicates that the ARIMA model demonstrated less bias.

5.5.4.3 Variance of the residual error versus forecast error

```
Variance of the ARIMA forecast errors: 13361.487347333305
Variance of the ARIMA residual errors: 5387.7487500989355
Variance of the SARIMA forecast errors: 13349.447781130013
Variance of the SARIMA residual errors: 5564.540467971463
```

While the SARIMA model exhibited a lower forecast error, indicating its predictions were typically closer to the future data points, the ARIMA model displayed a lower residual error, suggesting its predictions were more accurate for known data points.

After considering these factors and the inherent seasonality in the dataset, the SARIMA model was deemed the most suitable for the dataset, despite the ARIMA model's stronger performance in certain areas. The SARIMA model's superior handling of seasonal data and its better overall predictive accuracy led to its selection as the preferred model.

6. Final Model

6.1 Final model selection

```
Average h-step ahead predction MSE: 13562.149629934996
Naive h-step ahead predction MSE: 64578.24995158316
Drift h-step ahead predction MSE: 76997.01209159679
SSE 0.05 h-step ahead predction MSE: 37033.847995196455
Multiple Linear Regression MSE: 13471.227989833695
ARIMA(12, 0, 12) MSE: 13361.489978925487
SARIMA(1,0,1)(1, 0, 1, 12) MSE: 13349.477734212884
```

Upon conducting a comprehensive comparison of the MSE for all the models evaluated, the SARIMA model emerged as the most proficient, registering the lowest MSE. Consequently, the SARIMA model was selected as the final model for this study.

6.2 100-step ahead prediction

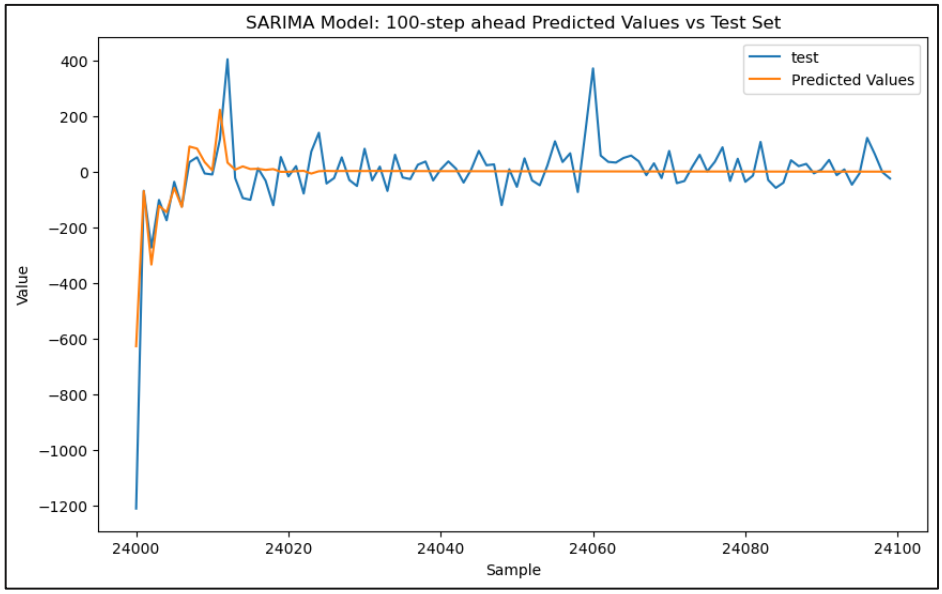


Figure 22. SARIMA(1,0,1)(1,0,1,12) Model Forecast

The forecast derived from the SARIMA model was then examined. The presented plot displays a 100-step forecast, wherein the initial part of the forecast aligns closely with the test data, thereby demonstrating the model's robust performance. However, as I venture further into the forecast, the predictions seem to flatten into a line. This observation is attributable to the inherent uncertainty and error that tend to magnify in long-term forecasts, particularly when dealing with real-world data.

Conclusion and Limitation

Throughout this study, I have performed extensive analysis and modeling, ultimately selecting the SARIMA model as the most effective for my dataset. The superior performance of the SARIMA model was evidenced by its lowest Mean Squared Error (MSE) among all models tested and its robustness was further validated through residual analysis and long-term forecasting checks.

However, acknowledging the limitations of my chosen model is crucial. One notable limitation was observed in the long-term forecast where the predictions gradually transitioned to a straight line as I moved further into the future. This simplification is a common challenge in long-term forecasting with time-series data due to the inherent increase in uncertainty and error.

Moreover, another important limitation to mention is the potential lack of exploration of non-linear relationships in my data. The correlation matrix analysis suggested that linear relationships between the features might not be strong, indicating that linear regression models, including the multiple linear regression used, might not have fully captured the potential complexities in my data. Therefore, there might be room for performance improvement by exploring models that can handle non-linear relationships.

For future work, it could be beneficial to investigate more complex models that can account for non-linear trends or sudden changes in the data. Recommendations include the use of machine learning techniques such as decision trees or neural networks, which can handle non-linear and complex relationships better than traditional linear models.

Appendix

```
#%% Import library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

import seaborn as sns
import statsmodels.api as sm
from scipy import stats
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.stattools import kpss

from statsmodels.tsa.seasonal import STL
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from statsmodels.tsa.holtwinters import SimpleExpSmoothing,
ExponentialSmoothing
from statsmodels.tsa.api import ARIMA, SARIMAX

from statsmodels.tsa.stattools import acf

from scipy import signal

from statsmodels.stats.diagnostic import acorr_ljungbox

import warnings
warnings.filterwarnings('ignore')


#%% 6-a. Pre-processing dataset: Dataset cleaning for missing observation.
You must follow the data cleaning techniques for time series dataset.

# Import train and test dataset
data = pd.read_csv('train.csv', parse_dates=['date'])
#data['date'] = pd.to_datetime(data['date'])

# Check dataset
print ("Data Head:\n", data.head())
print ("Data Statistics:\n", data.describe())

# Missing point
print ("Missing values:", data.isna().sum().sum())
# No missing point
```

```
#%%
data.plot(x='date', y='sales', figsize=(15, 6), title='Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.show()

#%%
# Reshaping train dataset
df = data.copy()

df['date'] =  pd.to_datetime(df['date'], format='%Y-%m').dt.to_period('M')

df = df.groupby(["store", "item", "date"]).sum().reset_index()

df['date'] = pd.to_datetime(df['date'].astype(str))

df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month

df


#%% 6-b. Plot of the dependent variable versus time. Write down your
observations.
#df['date'] = pd.to_datetime(df['date'].astype(str))


# Aggregating sales by date
sales_by_date = df.groupby('date')['sales'].sum()

fig, ax = plt.subplots(figsize=(15,7))

# Plotting the data
ax.plot_date(sales_by_date.index, sales_by_date.values, '-')

# Formatting the y-axis to display in real values
formatter = ticker.FuncFormatter(lambda x, p: format(int(x), ','))
ax.yaxis.set_major_formatter(formatter)

# Setting the title and labels
ax.set_title('Sales by Date')
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.show()

#%% 6-c. ACF/PACF of the dependent variable. Write down your observations.


def ACF_PACF_Plot(y, lags):
    acf = sm.tsa.stattools.acf(y, nlags=lags)
    pacf = sm.tsa.stattools.pacf(y, nlags=lags)
```

```
    fig = plt.figure(figsize=(10, 8))
    plt.subplot(211)
    plt.title('ACF/PACF of the raw data')
    plot_acf(y, ax=plt.gca(), lags=lags)
    plt.subplot(212)
    plot_pacf(y, ax=plt.gca(), lags=lags)
    fig.tight_layout(pad=3)
    plt.show()
    return acf, pacf


ACF_PACF_Plot(df['sales'], 50)

# ARMA model
# There may be seasonality

#%% 6-d. Correlation Matrix with seaborn heatmap with the Pearson,Äôs
correlation coefficient. Write down your observations.

corr_matrix = df.corr(method='pearson')

plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix,
            cmap='coolwarm',
            fmt='.3f',
            annot=True,
            vmin=-1,
            vmax=1,
            linewidth=1)
plt.title("Correlation Matrix of Dataset", size=15)
plt.show()


#%% 6-e. Split the dataset into train set (80%) and test set (20%).

# I will do this step right before modeling after checking whether dataset
is stationary

#%% 7- Stationarity:
# Check for a need to make the dependent variable stationary. If the
dependent variable is not stationary, you need to use the techniques
discussed in class to make it stationary. Perform ACF/PACF analysis for
stationarity. You need to perform ADF-test & kpss-test and plot the
rolling mean and variance for the raw data and the transformed data. Write
down your observations.
def ADF_Cal(x):
    result = adfuller(x)
    print("ADF Statistic: %f" % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))

ADF_Cal(df['sales'])
```

```
# non-stationary vs stationary
# reject the null with low p-value which means stationary


def kpss_test(timeseries):
    print('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c', nlags="auto")
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic', 'p-
value', 'Lags Used'])
    for key, value in kpsstest[3].items():
        kpss_output['Critical Value (%s)' % key] = value
    print(kpss_output)


kpss_test(df['sales'])

# stationary vs non-stationary
# reject the null with low p-value (0.01) which means non-stationary

def rolling(data):
    n = len(data)
    rolling_mean = np.zeros(n)
    rolling_var = np.zeros(n)

    for i in range(n):
        rolling_mean[i] = np.mean(data[:i + 1])
        rolling_var[i] = np.var(data[:i + 1])

    # Plotting
    fig, axs = plt.subplots(2, figsize=(10, 5))

    axs[0].plot(rolling_mean)
    axs[0].set_title('Rolling Mean')
    axs[0].set_xlabel('Samples')
    axs[0].set_ylabel('Magnitude')
    axs[0].set_xlim(-50, n + 50)

    axs[1].plot(rolling_var, label='Varying variance')
    axs[1].set_title('Rolling Variance')
    axs[1].set_xlabel('Samples')
    axs[1].set_ylabel('Magnitude')
    axs[1].legend(loc='lower right')
    axs[1].set_xlim(-50, n + 50)

    plt.tight_layout()
    plt.show()

    # return rolling_mean, rolling_var

rolling(df['sales'])


#%%
```

```
# Transformation

df['diff_'] = df['sales'].diff()

# Nan value replace
df['diff_'].fillna(method='bfill', inplace=True)

ADF_Cal(df['diff_'].dropna())
kpss_test(df['diff_'].dropna())
rolling(df['diff_'])

ACF_PACF_Plot(df['diff_'], 50)

#%% 8- Time series Decomposition:
# Approximate the trend and the seasonality and plot the detrended and the
seasonally adjusted data set using STL method. Find the out the strength
of the trend and seasonality. Refer to the lecture notes for different
type of time series decomposition techniques.

from statsmodels.tsa.seasonal import STL

STL = STL(df['diff_'], period=12)
res = STL.fit()

T = res.trend
S = res.seasonal
R = res.resid

# Plot the decomposition
fig, axes = plt.subplots(4, 1, figsize=(15, 10))
axes[0].plot(df['diff_'])
axes[0].set_title("Original Data")
T.plot(ax=axes[1], title="Trend Component")
S.plot(ax=axes[2], title="Seasonal Component")
R.plot(ax=axes[3], title="Residual Component")

plt.tight_layout()
plt.show()

# Calculate the strength of trend and seasonality
F_t = max(0, 1 - np.var(R) / np.var(T + R))
print(f'The strength of trend after diff is {F_t}')

F_s = max(0, 1 - np.var(R) / np.var(S + R))
print(f'The strength of seasonality after diff is {F_s}')




# Seasonally adjusted data and plot
df['diff'] = df['diff_'] - S

from statsmodels.tsa.seasonal import STL
```

```
STL = STL(df['diff'], period=12)
res = STL.fit()

T = res.trend
S = res.seasonal
R = res.resid

# Plot the decomposition
fig, axes = plt.subplots(4, 1, figsize=(15, 10))
axes[0].plot(df['diff'])
axes[0].set_title("Original Data")
T.plot(ax=axes[1], title="Trend Component")
S.plot(ax=axes[2], title="Seasonal Component")
R.plot(ax=axes[3], title="Residual Component")

plt.tight_layout()
plt.show()

# Calculate the strength of trend and seasonality
F_t = max(0, 1 - np.var(R) / np.var(T + R))
print(f'The strength of trend after adjusted seasonality is {F_t}')

F_s = max(0, 1 - np.var(R) / np.var(S + R))
print(f'The strength of seasonality after adjusted seasonality is {F_s}')

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(df['diff_'], label='Sales after differencing')
ax.plot(df['diff'], label='Seasonally Adjusted')
ax.set_title('Transformated Sales Data vs Seasonally Adjusted Data')
ax.set_xlabel('Sample')
ax.set_ylabel('Value')
ax.legend()
plt.show()

plot_acf(df['diff'])


#%% 6-e. Split the dataset into train set (80%) and test set (20%).
df.set_index('date', inplace=True)


split_index = int(len(df) * 0.8)
train_df = df[:split_index]
test_df = df[split_index:]

X_train = train_df.drop(columns=['diff', 'sales', 'diff_'])
X_test = test_df.drop(columns=['diff', 'sales', 'diff_'])

y_train = train_df['diff']
y_test = test_df['diff']

print(
    f'X Train set size: {len(X_train)}, X Test set size: {len(X_test)}, y
Train set size: {len(y_train)}, y Test set size: {len(y_test)}')
```

```
#%% 9- Holt-Winters method:
# Using the Holt-Winters method try to find the best fit using the train
dataset and make a prediction using the test set.

# Fit the Holt-Winters model on the training data
hw_model = ExponentialSmoothing(y_train, seasonal_periods=12, trend=None,
seasonal='add').fit()

hw_resid = hw_model.resid
# Make a prediction using the test set
hw_forecast = hw_model.forecast(steps=len(y_test))
# test_forecast = pd.Series(test_forecast, index=test_model.index)

# Evaluate the model's performance

hw_mse = mean_squared_error(y_test, hw_forecast)

print(f'Holt-Winter Model MSE: {hw_mse}')

# Plot the results
plt.figure(figsize=(10, 6))

sns.lineplot(x=y_test.index, y=y_test.values, label='Test', data=y_test)
sns.lineplot(x=y_test.index, y=hw_forecast.values, label='Holt-Winters
Forecast', data=hw_forecast)
plt.title('Holt-Winter Model Forecast')

plt.xlabel('Time step')
plt.ylabel('Value')
plt.legend()
plt.show()



#%% 10- Feature selection/elimination:
# You need to have a section in your report that explains how the feature
selection was performed and whether the collinearity exits not. Backward
stepwise regression along with SVD and condition number is needed. You
must explain that which feature(s) need to be eliminated and why. You are
welcome to use other methods like VIF, PCA or random forest for feature
elimination.

y = df['diff']
X = df.drop(['sales','diff', 'diff_'], axis=1)

# SVD analysis
H = X.T @ X
s, d, v = np.linalg.svd(H)
print ("Singular values:", d)


# Condition nu,ber
```

```
cond_num = np.linalg.cond(X)
print ("Condition number of X:", {cond_num})


def back_elimination(X, y):
    model = sm.OLS(y, sm.add_constant(X)).fit()
    aic = model.aic
    bic = model.bic
    adjstr2 = model.rsquared_adj
    features = list(X.columns)

    print(f'AIC: {aic}')
    print(f'BIC: {bic}')
    print(f'adjR2: {adjstr2}')
    print(f'***Baseline***')

    for f in features:
        model_ = sm.OLS(y, sm.add_constant(X.drop(columns=[f]))).fit()
        aic_ = model_.aic
        bic_ = model_.bic
        adjstr2_ = model_.rsquared_adj
        if aic_ < aic and bic_ < bic and adjstr2_ > adjstr2:  # good cond
            features.remove(f)
            print(f'***Dropped {f}***')
            print(f'AIC: {aic_}')
            print(f'BIC: {bic_}')
            print(f'adjR2: {adjstr2_}')
    return features


final_features = back_elimination(X_train, y_train)
print(final_features)


# VIF

def calculate_vif(X):
    vif = pd.DataFrame()
    vif["features"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
    return vif

vif_df = calculate_vif(X_train)
print(vif_df)

vif_df_after = calculate_vif(X_train.drop(columns=['store', 'month'],
axis=1))
print (vif_df_after)
# PCA


# Standardize the data
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform PCA
pca = PCA()
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Print the explained variance ratio
print("Explained variance ratio: ", pca.explained_variance_ratio_)


#%% 11- Base-models:
# average, na√Øve, drift, simple and exponential smoothing. You need to
perform an h-step prediction based on the base models and compare the
SARIMA model py_foreerformance with the base model predication.



# Average

# h-step forecast by average method
avg_y_forecast = np.zeros_like(y_test).astype(float)
for i in range(len(y_test)):
    avg_y_forecast[i] = np.mean(y_train)

# h step error
avg_error_hstep = y_test[:] - avg_y_forecast

avg_sqrerror_hstep = avg_error_hstep ** 2

# h-step MSE
avg_MSE_hstep = avg_sqrerror_hstep.sum() / len(y_test)
print(f'Average h-step ahead predction MSE is {avg_MSE_hstep}')

# # Plot
# plt.figure(figsize=(15,8))
# plt.plot(np.arange(len(y_train)), y_train, 'bo-', label='Training set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test,
'gs-', label='Test set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)),
avg_y_forecast, 'r+-', label='h-step forecast')
# plt.title('Average Forecast Method')
# plt.xlabel('Time step')
# plt.ylabel('Value')
# plt.legend()
# plt.grid()
# plt.show()

# Naive
# h-step forecast
naive_y_forecast = np.zeros_like(y_test).astype(float)
for i in range(len(y_test)):
    naive_y_forecast[i] = y_train[len(y_train)-1]
```

```
# h step error
naive_error_hstep = y_test[:] - naive_y_forecast

naive_sqrerror_hstep = naive_error_hstep ** 2

# h-step MSE
naive_MSE_hstep = naive_sqrerror_hstep.sum() / len(y_test)
print(f'Naive h-step ahead predction MSE is {naive_MSE_hstep}')

# # Plot
# plt.figure(figsize=(15,8))
# plt.plot(np.arange(len(y_train)), y_train, 'bo-', label='Training set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test,
'gs-', label='Test set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)),
naive_y_forecast, 'r+-', label='h-step forecast')
# plt.title('Naive Forecast Method')
# plt.xlabel('Time step')
# plt.ylabel('Value')
# plt.legend()
# plt.grid()
# plt.show()

# Drift

# h-step forecast
drift_y_forecast = np.zeros_like(y_test).astype(float)
for i in range(1, len(y_test) + 1):
    drift_y_forecast[i - 1] = y_train[len(y_train)-1] + i *
((y_train[len(y_train)-1] - y_train[0]) / (len(y_train) - 1))

# h step error
drift_error_hstep = y_test[:] - drift_y_forecast

drift_sqrerror_hstep = drift_error_hstep ** 2


# h-step MSE
drift_MSE_hstep = drift_sqrerror_hstep.sum() / len(y_test)
print(f'Drift h-step ahead predction MSE is {drift_MSE_hstep}')

# # Plot
# plt.figure(figsize=(15,8))
# plt.plot(np.arange(len(y_train)), y_train, 'bo-', label='Training set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test,
'gs-', label='Test set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)),
drift_y_forecast, 'r+-', label='h-step forecast')
# plt.title('Drift Forecast Method')
# plt.xlabel('Time step')
# plt.ylabel('Value')
# plt.legend()
# plt.grid()
```

```
# plt.show()

# Simple Exponential Smoothing

# 1-step prediction
alpha = 0.5
ses5_forecast = np.zeros(len(y_train))
ses5_forecast[0] = y_train[0]

for i in range(1, len(y_train)):
    ses5_forecast[i] = alpha * y_train[i - 1] + (1 - alpha) *
ses5_forecast[i - 1]

# h-step forecast
ses5_y_forecast = np.zeros_like(y_test).astype(float)
for i in range(1, len(y_test) + 1):
    ses5_y_forecast[i - 1] = alpha * y_train[len(y_train)-1] + (1 - alpha)
* ses5_forecast[-1]

# h step error
ses5_error_hstep = y_test[:] - ses5_y_forecast

ses5_sqrerror_hstep = ses5_error_hstep ** 2


# h-step MSE
ses5_MSE_hstep = ses5_sqrerror_hstep.sum() / len(y_test)
print(f'SES h-step ahead predction MSE is {ses5_MSE_hstep}')

# # Plot
# plt.figure(figsize=(15,8))
# plt.plot(np.arange(len(y_train)), y_train, 'bo-', label='Training set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)), y_test,
'gs-', label='Test set')
# plt.plot(np.arange(len(y_train), len(y_train) + len(y_test)),
ses5_y_forecast, 'r+-', label='h-step forecast')
# plt.title('SES Forecast Method w/ alpha = 0.5')
# plt.xlabel('Time step')
# plt.ylabel('Value')
# plt.legend()
# plt.grid()
# plt.show()

fig, axs = plt.subplots(2, 2, figsize=(20, 12))
fig.suptitle('Basic Method Forecast', fontsize=25)

# Average Forecast Method
axs[0, 0].plot(np.arange(len(y_train)), y_train, 'bo-', label='Training
set')
axs[0, 0].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
y_test, 'gs-', label='Test set')
axs[0, 0].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
avg_y_forecast, 'r+-', label='h-step forecast')
axs[0, 0].set_title('Average Forecast Method', fontsize=16)
```

```
axs[0, 0].set_xlabel('Time step')
axs[0, 0].set_ylabel('Value')
axs[0, 0].legend()
axs[0, 0].grid()

# Naive Forecast Method
axs[0, 1].plot(np.arange(len(y_train)), y_train, 'bo-', label='Training
set')
axs[0, 1].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
y_test, 'gs-', label='Test set')
axs[0, 1].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
naive_y_forecast, 'r+-', label='h-step forecast')
axs[0, 1].set_title('Naive Forecast Method', fontsize=16)
axs[0, 1].set_xlabel('Time step')
axs[0, 1].set_ylabel('Value')
axs[0, 1].legend()
axs[0, 1].grid()

# Drift Forecast Method
axs[1, 0].plot(np.arange(len(y_train)), y_train, 'bo-', label='Training
set')
axs[1, 0].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
y_test, 'gs-', label='Test set')
axs[1, 0].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
drift_y_forecast, 'r+-', label='h-step forecast')
axs[1, 0].set_title('Drift Forecast Method', fontsize=16)
axs[1, 0].set_xlabel('Time step')
axs[1, 0].set_ylabel('Value')
axs[1, 0].legend()
axs[1, 0].grid()

# SES Forecast Method w/ alpha = 0.5
axs[1, 1].plot(np.arange(len(y_train)), y_train, 'bo-', label='Training
set')
axs[1, 1].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
y_test, 'gs-', label='Test set')
axs[1, 1].plot(np.arange(len(y_train), len(y_train) + len(y_test)),
ses5_y_forecast, 'r+-', label='h-step forecast')
axs[1, 1].set_title('SES Forecast Method w/ alpha = 0.5', fontsize=16)
axs[1, 1].set_xlabel('Time step')
axs[1, 1].set_ylabel('Value')
axs[1, 1].legend()
axs[1, 1].grid()

plt.tight_layout()
plt.show()

# After doing SARIMA, I will compare all these models.



#%% 12- Develop the multiple linear regression model that represent the
dataset. Check the accuracy of the developed model.
```

```
# features : all

model = sm.OLS(y_train, sm.add_constant(X_train))
modelfit = model.fit()


y_pred = modelfit.predict(sm.add_constant(X_test))

# Calculate the mean squared error and R-squared score
ols_mse = mean_squared_error(y_test, y_pred)
print(f'All features MSE: {ols_mse:.2f}')


# 12-b. Hypothesis tests analysis: F-test, t-test.

modelresult = modelfit.summary()
modelresult



# features : features selection
model_store = sm.OLS(y_train, sm.add_constant(X_train[final_features]))
modelfit = model_store.fit()


# 12-a. You need to include the complete regression analysis into your
report. Perform one-step ahead prediction and compare the performance
versus the test set.

y_pred = modelfit.predict(sm.add_constant(X_test[final_features]))

# Calculate the mean squared error and R-squared score
mse = mean_squared_error(y_test, y_pred)
print(f'Feature Selection MSE: {mse:.2f}')

# 12-b. Hypothesis tests analysis: F-test, t-test.

modelresult = modelfit.summary()
modelresult


plt.figure(figsize=(14, 6))
# sns.lineplot(x=y_train.index, y=y_train.values, label='Train',
data=y_train)
sns.lineplot(x=y_test.index, y=y_test.values, label='Test', data=y_test)
plt.scatter(x=y_pred.index, y=y_pred.values, label='Predicted')
plt.xlabel('Time step')
plt.ylabel('Value')
plt.legend(loc='best')
plt.title('Test and Predicted Plot')
plt.legend()
plt.show()
```

```
f_test = modelfit.f_pvalue
t_test = modelfit.tvalues

print (f'F-test: {f_test}')
print (f't-test: \n{t_test}')

# 12-d. ACF of residuals.
residuals = y_test - y_pred

plot_acf(residuals, lags=50)

# 12-e. Q-value

q_value = acorr_ljungbox(residuals, lags=10, return_df=True)
print(f"Q value: \n{q_value}")



# p value is more than good


# 12-f. Variance and mean of the residuals.
residuals_variance = np.var(residuals)
residuals_mean = np.mean(residuals)

print(f"Residuals Variance: {residuals_variance:.2f}")
print(f"Residuals Mean: {residuals_mean:.2f}")


#%% 13- ARMA and ARIMA and SARIMA model order determination: Develop an
ARMA, ARIMA and SARIMA model that represent the dataset.


ACF_PACF_Plot(y_train, 25)

#AR(12)
#ARMA(12,0)
#ARIMA(12,0,0)
#SARIMA(1,0,0,12)

def GPAC(ry, j=7, k=7):
    c = len(ry) // 2
    gpac_table = np.zeros((j,k-1))

    for i in range(j):
        for l in range(1, k):

            den_matrix = np.zeros((l,l))
            for row in range(l):
                den_matrix[row] = ry2[c - i - row : c - i + l - row]

            num_matrix = den_matrix.copy().T
            num_matrix[-1] = ry2[c + i + 1 : c + i + 1 + l]
```

```
            num_matrix = num_matrix.T

            phi = np.linalg.det(num_matrix) / np.linalg.det(den_matrix)

            if num_matrix.shape[0] == num_matrix.shape[1] and
den_matrix.shape[0] == den_matrix.shape[1]:
                num = np.linalg.det(num_matrix)
                den = np.linalg.det(den_matrix)
                if den != 0:
                    gpac_table[i, l-1] = num / den
                else:
                    gpac_table[i, l-1] = np.nan

    plt.figure(figsize=(20, 12))
    # Create a Seaborn heatmap
    sns.heatmap(gpac_table,
                mask = np.isnan(gpac_table),
                fmt = ".4f",
                cmap = 'Reds_r',
                annot = True,
                vmin = -1,
                vmax = 1,
                linewidth = 1)
    plt.xticks(np.arange(0.5, (k-1)+0.5, 1), np.arange(1, k, 1))
    plt.title("GPAC Table", size = 15)
    plt.show()

    return gpac_table



ry = acf(y_train, nlags=50)
ry1 = ry[::-1]
ry2 = np.concatenate((ry1, ry[1:]))

GPAC(ry2,15,15)

# 13-a. Preliminary model development procedures and results. (ARMA model
order determination). Pick at least two orders using GPAC table.
# 13-b. Should include discussion of the autocorrelation function and the
GPAC. Include a plot of the autocorrelation function and the GPAC table
within this section).
# 13-c. Include the GPAC table in your report and highlight the estimated
order.

# ARIMA (12,0,0)
# ARIMA (13,0,1)
# ARIMA (12,0,12)
# SARIMA (0,0,0)(1,0,1,12)
# SARIMA (1,0,1)(1,0,1,12)
# SARIMA (2,0,2)(1,0,1,12)
```

```
#%%

# ARIMA (12,0,0)
arima120_ = ARIMA(y_train, order = (12,0,0))
arima120_fit = arima120_.fit()
print (arima120_fit.summary())

# ARIMA (13,0,1)
arima131_ = ARIMA(y_train, order = (13,0,1))
arima131_fit = arima131_.fit()
print (arima131_fit.summary())

# ARIMA (12,0,12)
arima1212_ = ARIMA(y_train, order = (12,0,12))
arima1212_fit = arima1212_.fit()
print (arima1212_fit.summary())


print("ARIMA(12, 0, 0) AIC:", arima120_fit.aic)
print("ARIMA(13, 0, 1) AIC:", arima131_fit.aic)
print("ARIMA(12, 0, 12) AIC:", arima1212_fit.aic)

print("ARIMA(12, 0, 0) BIC:", arima120_fit.bic)
print("ARIMA(13, 0, 1) BIC:", arima131_fit.bic)
print("ARIMA(12, 0, 12) BIC:", arima1212_fit.bic)

arima120_pred = arima120_fit.predict(start=len(y_train), end=len(y_train)
+ len(y_test) - 1)
arima120_mse = mean_squared_error(y_test, arima120_pred)

arima131_pred = arima131_fit.predict(start=len(y_train), end=len(y_train)
+ len(y_test) - 1)
arima131_mse = mean_squared_error(y_test, arima131_pred)

arima1212_pred = arima1212_fit.predict(start=len(y_train),
end=len(y_train) + len(y_test) - 1)
arima1212_mse = mean_squared_error(y_test, arima1212_pred)

print("ARIMA(12, 0, 0) MSE:", arima120_mse)
print("ARIMA(13, 0, 1) MSE:", arima131_mse)
print("ARIMA(12, 0, 12) MSE:", arima1212_mse)

ACF_PACF_Plot(arima120_fit.resid, lags=24)
ACF_PACF_Plot(arima131_fit.resid, lags=24)
ACF_PACF_Plot(arima1212_fit.resid, lags=24)

# Selected ARIMA (12,0,12)
arima_resid = arima1212_fit.resid


# SARIMA (0,0,0)(1,0,1,12)
sarima1_ = SARIMAX(y_train, order = (0,0,0), seasonal_order = (1,0,1,12))
sarima1_fit = sarima1_.fit()
print (sarima1_fit.summary())
```

```
# SARIMA (1,0,1)(1,0,1,12)
sarima2_ = SARIMAX(y_train, order = (1,0,1), seasonal_order = (1,0,1,12))
sarima2_fit = sarima2_.fit()
print (sarima2_fit.summary())

# SARIMA (2,0,2)(1,0,1,12)
sarima3_ = SARIMAX(y_train, order = (2,0,2), seasonal_order = (1,0,1,12))
sarima3_fit = sarima3_.fit()
print (sarima3_fit.summary())



print("SARIMA(0,0,0)(1, 0, 1, 12) AIC:", sarima1_fit.aic)
print("SARIMA(0,0,0)(1, 0, 1, 12) AIC:", sarima1_fit.bic)

sarima1_pred = sarima1_fit.predict(start=len(y_train), end=len(y_train) +
len(y_test) - 1)
sarima1_mse = mean_squared_error(y_test, sarima1_pred)

print("SARIMA(0,0,0)(1, 0, 1, 12) MSE:", sarima1_mse)


print("SARIMA(1,0,1)(1, 0, 1, 12) AIC:", sarima2_fit.aic)
print("SARIMA(1,0,1)(1, 0, 1, 12) AIC:", sarima2_fit.bic)

sarima2_pred = sarima2_fit.predict(start=len(y_train), end=len(y_train) +
len(y_test) - 1)
sarima2_mse = mean_squared_error(y_test, sarima2_pred)

print("SARIMA(1,0,1)(1, 0, 1, 12) MSE:", sarima2_mse)


print("SARIMA(2,0,2)(1, 0, 1, 12) AIC:", sarima3_fit.aic)
print("SARIMA(2,0,2)(1, 0, 1, 12) AIC:", sarima3_fit.bic)

sarima3_pred = sarima3_fit.predict(start=len(y_train), end=len(y_train) +
len(y_test) - 1)
sarima3_mse = mean_squared_error(y_test, sarima3_pred)

print("SARIMA(2,0,2)(1, 0, 1, 12) MSE:", sarima3_mse)

ACF_PACF_Plot(sarima1_fit.resid, lags=24)
ACF_PACF_Plot(sarima2_fit.resid, lags=24)
ACF_PACF_Plot(sarima3_fit.resid, lags=24)



# SARIMA (1,0,1)(1,0,1,12)
sarima_resid = sarima2_fit.resid
```

```
#%% 14- Estimate ARMA model parameters using the Levenberg Marquardt
algorithm. Display the parameter estimates, the standard deviation of the
parameter estimates and confidence intervals.

# ARIMA (12,0,12)
arima_params = arima1212_fit.params
arima_std = arima1212_fit.bse
arima_ci = arima1212_fit.conf_int()

print(f"Coefficients: \n{arima_params}")
print(f"\nStandard Errors: \n{arima_std}")
print(f"\nConfidence Intervals: \n{arima_ci}")



sarima_params = sarima2_fit.params
sarima_std = sarima2_fit.bse
sarima_ci = sarima2_fit.conf_int()

print(f"Coefficients: \n{sarima_params}")
print(f"\nStandard Errors: \n{sarima_std}")
print(f"\nConfidence Intervals: \n{sarima_ci}")

# SARIMA
#%% 15- Diagnostic Analysis: Make sure to include the followings:
# 15-a. Diagnostic tests (confidence intervals, zero/pole cancellation,
chi-square test).

# 15-b. Display the estimated variance of the error and the estimated
covariance of the estimated parameters.

# ARIMA(12,0,12)
print(f'The estimated variance of error for ARIMA(12,0,12):
\n{arima_resid.var()}')
arima_cov_theta_hat = arima1212_fit.cov_params()
print(f'The covariance for ARIMA(12,0,12): \n{arima_cov_theta_hat}')

# SARIMA(1,0,1)(1,0,1,12)
print(f'The estimated variance of error for SARIMA(1,0,1)(1,0,0,12):
\n{sarima_resid.var()}')
sarima_cov_theta_hat = sarima2_fit.cov_params()
print(f'The covariance for SARIMA(1,0,1)(1,0,0,12):
\n{sarima_cov_theta_hat}')

# ARIMA

#%%
# 15-c. Is the derived model biased or this is an unbiased estimator?

# Mean of ARIMA (12,0,12)
arima_bias = np.mean(arima_resid)
print(f"Mean of arima_Residuals is {arima_bias}\n")
```

```
# SARIMA (1,0,1)(1,0,1,12)

# Mean of SARIMA
sarima_bias = np.mean(sarima_resid)
print(f"Mean of sarima_Residuals is {sarima_bias}\n")



# ARIMA

#%%
# 15-d. Check the variance of the residual errors versus the variance of
the forecast errors.

# ARIMA(12,0,12)
arima_forecast = arima1212_fit.forecast(steps=len(y_test))

# Calculate the variance of the forecast errors and the residual errors
forecast_errors_variance = np.var(y_test.values - arima_forecast)
residual_errors_variance = np.var(arima_resid)

print("Variance of the ARIMA forecast errors:", forecast_errors_variance)
print("Variance of the ARIMA residual errors:", residual_errors_variance)


# SARIMA (1,0,1)(1,0,0,12)
sarima_forecast = sarima2_fit.forecast(steps=len(y_test))

# Calculate the variance of the forecast errors and the residual errors
forecast_errors_variance = np.var(y_test.values - sarima_forecast)
residual_errors_variance = np.var(sarima_resid)

print("Variance of the SARIMA forecast errors:", forecast_errors_variance)
print("Variance of the SARIMA residual errors:", residual_errors_variance)



# 15-e. If you find out that the ARIMA or SARIMA model may better
represents the dataset, then you can find the model accordingly. You are
not constraint only to use of ARMA model. Finding an ARMA model is a
minimum requirement and making the model better is always welcomed.


# SARIMA


#%% 17- Final Model selection:
# There should be a complete description of why your final model was
picked over base-models ARMA, ARIMA, SARIMA and LSTM. You need to compare
the performance of various models developed for your dataset and come up
with the best model that represent the dataset the best.

# MSE
```

```
print(f'Average h-step ahead predction MSE: {avg_MSE_hstep}')
print(f'Naive h-step ahead predction MSE: {naive_MSE_hstep}')
print(f'Drift h-step ahead predction MSE: {drift_MSE_hstep}')
print(f'SSE 0.05 h-step ahead predction MSE: {ses5_MSE_hstep}')

print(f'Multiple Linear Regression MSE: {ols_mse}')

print("ARIMA(12, 0, 12) MSE:", arima1212_mse)
print("SARIMA(1,0,1)(1, 0, 1, 12) MSE:", sarima2_mse)



# AIC -> SARIMA


print(f'Multiple Linear Regression AIC: {modelfit.aic}')

print("ARIMA(12, 0, 12) AIC:", arima1212_fit.aic)
print("SARIMA(1,0,1)(1, 0, 1, 12) AIC:", sarima2_fit.aic)




# BIC -> SARIMA
print(f'Multiple Linear Regression BIC: {modelfit.bic}')

print("ARIMA(12, 0, 12) BIC:", arima1212_fit.bic)
print("SARIMA(1,0,1)(1, 0, 1, 12) AIC:", sarima2_fit.bic)



# ACF PACF Plot


ACF_PACF_Plot(avg_error_hstep, lags=24)
ACF_PACF_Plot(naive_error_hstep, lags=24)
ACF_PACF_Plot(drift_error_hstep, lags=24)
ACF_PACF_Plot(ses5_error_hstep, lags=24)

ols_resid = modelfit.resid
ACF_PACF_Plot(ols_resid, lags=24)

ACF_PACF_Plot(arima_resid, lags=24)
ACF_PACF_Plot(sarima_resid, lags=24)

# SARIMA (1,0,0,12)
#%%
# 18- Forecast function:
# Once the final mode is picked (SARIMA), the forecast function needs to
be developed and included in your report.

# 19- h-step ahead Predictions:
```

```
# You need to make a multiple step ahead prediction for the duration of
the test data set. Then plot the predicted values versus the true value
(test set) and write down your observations.


# Plot the h-step predicted values versus the test set
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(y_train),len(y_train)+len(y_test))[:100],
y_test.values[:100], label='test')
plt.plot(np.arange(len(y_train),len(y_train)+len(y_test))[:100],
sarima_forecast[:100], label="Predicted Values")
plt.xlabel("Sample")
plt.ylabel("Value")
plt.title("SARIMA Model: 100-step ahead Predicted Values vs Test Set")
plt.legend()
plt.show()


# %%
```