DATS 6313 - Time Series Term Project

Predict Future Sales

Nayaeun Kwon

Table of Contents

**Table of Figures and Tables**

**Abstract and Introduction**

1. **Description of the Dataset**
2. **Data Preprocessing**

2.1 Checking missing values

2.2 Reshaping the dataset

1. **Exploratory Data Analysis**

3.1 Time series visualization

3.2 Autocorrelation analysis

3.3 Correlation matrix

3.4 Stationary

3.4.1 Stationary test

3.4.2 Data transformation

1. **Time Series Decomposition**
2. **Modeling**

5.1 Data splitting

5.2 Holt-Winters method

5.3 Basic forecasting method

5.4 Multiple linear regression

5.4.1 Feature selection

5.4.1.1 SVD

5.4.1.2 Condition number

5.4.1.3 PCA

5.4.1.4 VIF

5.4.1.5 Back elimination

5.4.2 Multiple linear regression modeling

5.4.3 Model forecast

**5.5 ARMA / ARIMA / SARIMA method**

5.5.1 Model order selection

5.5.1.1 ACF / PACF plots

5.5.1.2 GPAC table

5.5.2 Selecting the best ARIMA model

5.5.3 Selecting the best SARIMA model

5.5.4 ARIMA vs SARIMA

5.5.4.1 Confidence interval

5.5.4.2 Mean of model residuals

5.5.4.3 Variance of the residual error and forecast error

1. **Final Modeling**

6.1 Final model selection

6.2 100-step ahead prediction

**Conclusion and Limitation**

**Appendix**

Table of Figures

Figure 1. Sales Over Time 9

Figure 2. ACF/PACF of Dependent Variable 10

Figure 3. Correlation Matrix 11

Figure 4. Rolling Mean and Variance of Original Data 13

Figure 5. Rolling Mean and Variance of Differenced Data 14

Figure 6. Time Series Decomposition of Differenced Data 15

Figure 7. Time Series Decomposition after Adjusted Seasonality 16

Figure 8. Seasonality Adjusted vs Difference Sales Data 17

Figure 9. Multiple Linear Regression Method Forecast 24

Figure 10. MLR residuals ACF Plot 25

Figure 11. Training Data ACF/PACF Plot 26

Figure 12. GPAC Table 27

Figure 13. ARIMA(12,0,0) Model Residuals ACF/PACF Plot 31

Figure 14. ARIMA(13,0,1) Model Residuals ACF/PACF Plot 31

Figure 15. ARIMA(12,0,12) Model Residuals ACF/PACF Plot 32

Figure 16. SARIMA(0,0,0)(1,0,1,12) Model Summary 33

Figure 17. SARIMA(1,0,1)(1,0,1,12) Model Summary 33

Figure 18. SARIMA(2,0,2)(1,0,1,12) Model Summary 34

Figure 19. SARIMA(0,0,0)(1,0,1,12) Model Residuals ACF/PACF Plot 35

Figure 20. SARIMA(1,0,1)(1,0,1,12) Model Residuals ACF/PACF Plot 35

Figure 21. SARIMA(2,0,2)(1,0,1,12) Model Residuals ACF/PACF Plot 36

Figure 22. SARIMA(1,0,1)(1,0,1,12) Model Forecast 39

Table of Tables

Table 1. Description of Dataset 7

Table 2. Description of Reshaping Dataset 9

Table 3. ADF / KPSS Test of Original Data 13

Table 4. ADF / KPSS Test of Differenced Data 14

Table 5. Holt-Winters Method Forecast 19

Table 6. Basic Forecasting Method Forecast 20

Table 7. Multiple Linear Regression for All Features 23

Table 8. Multiple Linear Regression for Feature Selection Features 23

Table 9. ARIMA(12,0,0) Model Summary 29

Table 10. ARIMA(13,0,1) Model Summary 30

Table 11. ARIMA(12,0,12) Model Summary 31

Table 12. Confidence Interval ARIMA(12,0,12) vs SARIMA(1,0,1)(1,0,1,12) 38

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

A screenshot of a table

Description automatically generated with low confidence A screenshot of a computer code

Description automatically generated with low confidence

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.
5. **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.

A black text on a white background

Description automatically generated with medium confidence

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:

1. Date Formatting: The 'date' column was converted into a datetime object and formatted to represent the respective ‘month’ and ‘year’ of each sales record.
2. 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.
3. 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:

A picture containing text, screenshot, font, number

Description automatically generated A screenshot of a computer code

Description automatically generated with low confidence

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.

1. **Exploratory Data Analysis**

**3.1 Time series visualization**

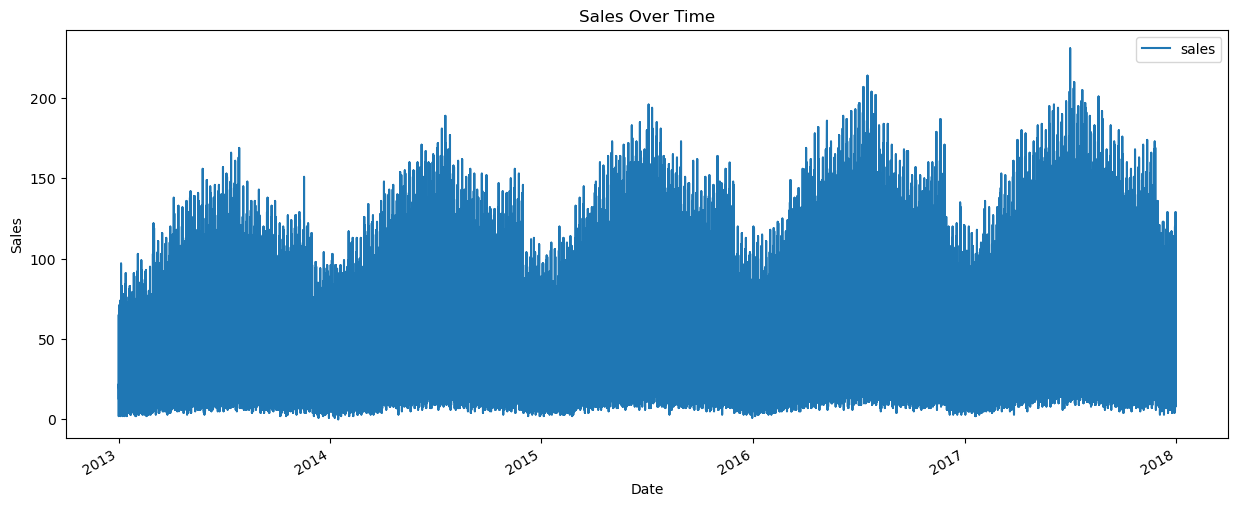


Figure 1. Sales Over Time

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**

A picture containing text, line, screenshot, plot

Description automatically generated

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.

* 1. **Correlation matrix**

A screenshot of a graph

Description automatically generated with low confidence

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.

* 1. **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.

A picture containing text, font, white, screenshot

Description automatically generatedA picture containing text, font, screenshot, white

Description automatically generated

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.

A picture containing text, line, diagram, screenshot

Description automatically generated

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.

A picture containing text, font, white, screenshot

Description automatically generatedA picture containing text, font, screenshot, white

Description automatically generated

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.

A picture containing text, line, screenshot, diagram

Description automatically generated

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.

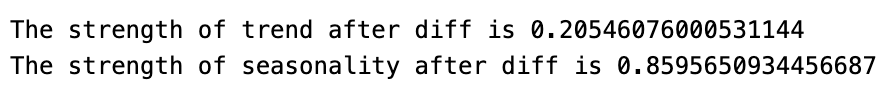
1. **Time Series Decomposition**

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

A picture containing text, screenshot, line, font

Description automatically generated

Figure 6. Time Series Decomposition of Differenced Data



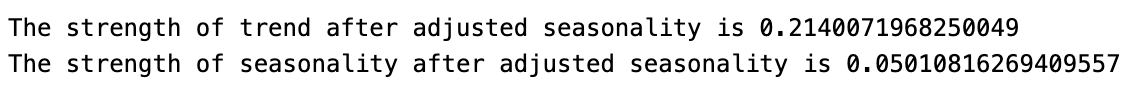
The results of the STL decomposition indicate that the series still retains significant seasonality post-differencing. The strength of the trend is approximately 0.21, suggesting a modest presence of trend in the series. However, the seasonality strength is much higher, sitting around 0.86. This high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

A picture containing screenshot, text, line, font

Description automatically generated

Figure 7. Time Series Decomposition after Adjusted Seasonality



A picture containing screenshot, text, plot, diagram

Description automatically generated

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.

1. **Modeling**

**5.1 Data splitting**



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**

A picture containing text, screenshot, plot, line

Description automatically generated

Table 5. Holt-Winters Method Forecast



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**

A picture containing screenshot, colorfulness, text, plot

Description automatically generated

Table 6. Basic Forecasting Method Forecast

A picture containing text, font, white

Description automatically generated

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.

* 1. **Multiple linear regression method**
     1. **Feature selection**

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

* + - 1. **SVD test**



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.

* + - 1. **Condition number**



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.

* + - 1. **PCA**



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.

* + - 1. **VIF**

A picture containing text, font, white

Description automatically generated

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.

* + - 1. **Back elimination**

A picture containing text, font, white

Description automatically generated A picture containing text, font, white, typography

Description automatically generated A picture containing text, font, white, screenshot

Description automatically generated

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.

A black text on a white background

Description automatically generated with low confidence

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.

* + 1. **Multiple linear regression modeling**

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

A screenshot of a table

Description automatically generated with low confidenceA black text on a white background

Description automatically generated with low confidence

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.

A picture containing text, screenshot, font, menu

Description automatically generated

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.

* + 1. **Model forecast**

A picture containing screenshot, line, plot, text

Description automatically generated

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.

A graph with blue dots

Description automatically generated with low confidence

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.



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.

* 1. **ARMA / ARIMA / SARIMA method**
     1. **Model order selection**
        1. **ACF/PACF**

A picture containing text, screenshot, line, number

Description automatically generated

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.

* + - 1. **GPAC table**

`A picture containing screenshot, colorfulness, square, diagram

Description automatically generated

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.

* + 1. **Selecting the best ARIMA model**

A screenshot of a computer

Description automatically generated with low confidence

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

A screenshot of a computer

Description automatically generated with low confidence

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

A picture containing text, screenshot, number, menu

Description automatically generated

A picture containing text, receipt, font, screenshot

Description automatically generated

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.

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, font, screenshot, white

Description automatically generated

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.

* + 1. **Selecting the best SARIMA model**

A picture containing text, receipt, font, screenshot

Description automatically generated

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

A picture containing text, screenshot, number, font

Description automatically generated

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

A screenshot of a computer

Description automatically generated with low confidence

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.

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, screenshot, line, number

Description automatically generated

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

A picture containing text, font

Description automatically generated

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.

* + 1. **ARIMA vs SARIMA**
       1. **Confidence interval**

A picture containing text, document, font, black and white

Description automatically generated vs A close-up of numbers

Description automatically generated with low confidence

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.

* + - 1. **Mean of model residual**

A picture containing text, font, white, algebra

Description automatically generated

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.

* + - 1. **Variance of the residual error versus forecast error**

A picture containing text, font, white

Description automatically generated

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.

1. **Final Model**
   1. **Final model selection**

A picture containing text, font, white, algebra

Description automatically generated

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**

**A graph with blue and orange lines

Description automatically generated with low confidence**

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()

# %%