A REPORT

ON

DEMAND FORECASTING IN THE E-COMMERCE INDUSTRY

BY

MUGDHA GUPTA **2021B3A72724H**

AT

COUTURE AI PVT LTD., BANGALORE

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

JUNE 2023

A REPORT

ON

Demand Forecasting in the E-Commerce Industry

BY

Name	ID Number
MUGDHA GUPTA	2021B3A72724H

Discipline

MSc Economics + BE Computer Science

Prepared in partial fulfillment of the Practice School-I
Course No. BITS C221

AT

COUTURE AI PVT LTD., BANGALORE

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

JUNE 2023

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my deepest gratitude and appreciation to the individuals who have made significant contributions to the successful completion of my PS-1 mid sem report. Their guidance, support, and encouragement have been invaluable throughout this journey.

First and foremost, I would like to extend my heartfelt thanks to Dr.Mrinal K Jagirdar, my esteemed PS faculty member. Your expertise, dedication, and unwavering support have been instrumental in shaping my PS-1
experience. I am sincerely grateful for the opportunity to learn from you and for the impact you have had on my professional growth.

I would also like to express my deepest appreciation to my mentor, Mr. Lucky Srivastava. The mentorship, patience, and extensive knowledge have been invaluable during my internship. The guidance and constant encouragement have not only provided me with valuable insights into the industry but also challenged me to exceed my potential.

I am also grateful to Couture AI where I have the privilege of interning. The opportunities, resources, and practical experiences provided have been instrumental in deepening my understanding of the industry and enhancing my professional capabilities.

Lastly, I would like to express my sincere thanks to my family, friends, and peers for their unwavering support, encouragement, and understanding throughout this internship. Their belief in my abilities and constant motivation have been a driving force behind my success.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

Practice School Division

Station: Couture AI pvt ltd.**Centre:** Bangalore

Duration: 2 Months **Date of Start:** 30.05.2023

Date of Submission: June 27, 2023

Title of the Project:
Demand Forecasting In The E-Commerce Industry

ID No./Name(s)/ Discipline(s)/of the student:
2021B3A72724H
Mugdha Gupta
MSc Economics
BE Computer Science

Name(s) and designation(s) of the expert(s): Mr. Lucky Srivastava, Deep Learning Engineer

Name(s) of the PS Faculty: Dr. Mrinal Ketan Jagirdar

Key Words: Artificial Intelligence, Machine Learning, Python, ML Libraries, Deep Learning, Demand forecasting, Time series, Autoregressive integrated moving average (ARIMA), Data Visualizations, Colab-Notebooks, Long Short-Term Memory(LSTM), Extreme Gradient Boosting(XGBoost).

Project Areas: Time Series Analysis, Demand Forecasting, Deep Learning, Gradient Boosting, Neural Networks, LSTM

Abstract:

This project focuses on utilizing the ARIMA (Autoregressive Integrated Moving Average), XGBoost(Extreme Gradient Boosting) and LSTM(Long Short-Term Memory) model for demand forecasting in the e-commerce industry. The objective is to develop a predictive model that accurately forecasts future customer demand for products or services. By analyzing historical sales data, market trends, and other relevant factors, these 3 models aim to provide insights that can optimize inventory management, prevent stockouts, minimize wastage, and improve overall supply chain efficiency. The project aims to enhance decision-making processes, facilitate effective resource allocation, and drive business growth by enabling e-commerce companies to make data-driven decisions based on reliable demand forecasts.The work presented constitutes a contribution to modeling and forecasting the demand in a fashion apparels company, by using time series approach.It demonstrates how the historical demand data could be utilized to forecast future demand and how these forecasts affect the supply chain. The historical demand information was used to develop several autoregressive integrated moving average (ARIMA) models. The selected model corresponded to the ARIMA (1, 1, 1) and it was validated by another historical demand information under the same conditions. The predictions obtained by the XGBoost model weren't very close to the actual values whereas for the LSTM model, the predicted values were very close to the actual ones and the validation loss was the least. The results obtained prove that the LSTM model could be utilized to model and forecast the future demand in the sweaters manufacturing. These results will provide to managers of this manufacturing reliable guidelines in making decisions

Signature(s) of Student(s) Signature of PS Faculty

Date : 21 July, 2023 Date

TABLE OF CONTENTS

INTRODUCTION

E-commerce demand forecasting is a dynamic process that takes into account various factors such as historical sales data, market trends, consumer behavior, seasonality, economic situation, and more to predict future demand for products sold online.

Variables that impact e-commerce customer demand :

- **Seasonal Trends**:Seasonal trends refer to the periods of high and low customer demand for products, depending on the time of year. These fluctuations, also known as seasonality, can significantly impact demand forecasting.

- **Market size and demographics:** Market size and demographics need to be looked at when creating a demand forecast since customers in different locations may have different tastes and preferences. A great example of this is Lay's potato chips. In the United States, Frito Lay sells flavors like onion and cheese. Meanwhile, in other parts of the world, particularly in Asia, they sell curry-flavored and spicy chips, which can't be easily found in the US. This is because US consumers are not fond-of spices that much.

- **Product categories**: You need to consider the product categories when creating a demand forecast. It can have a significant impact on the sales momentum of the product. For example, if your brand sells high-priced goods such as electronics items, you'll likely see low sales velocity. In this case, your forecast should be spread out for 24-48 months. On the other hand, if you sell CPG goods, you might have to forecast consistently to find accurate demand for the coming months.

- **Competition**: Another important factor to consider when forecasting demand is the impact of competitors. For example,if a new brand enters your industry, it may steal some of your customers with aggressive advertising. It can cause a drop in sales for your product. Or perhaps, one of your existing competitors ran a successful video marketing campaign that got good traction. It can win your customers over, and your product will see a decline in sales. One way to counter competition is by focusing on differentiating your brand and product. You can do this by serving customers better. For example, offering more size or color options and introducing an improved feature for your product. Alternatively, you can run PPC campaigns to boost your product sales and increase visibility.

MAIN TEXT

What does Couture AI do?

Couture.ai is a Bangalore-based company focused on delivering AI-powered solutions tailored for the retail sector. Their extensive portfolio includes various offerings and support aimed at utilizing artificial intelligence and data analysis to enhance operational efficiency, elevate customer satisfaction, and foster business expansion for retailers.

Key aspects of Couture.ai:

Retail Analytics: Couture.ai specializes in creating cutting-edge analytics solutions designed specifically for the retail sector. They harness the power of machine learning and data analytics methodologies to extract meaningful and valuable information from vast amounts of retail data. These valuable insights can be utilized to gain a deeper understanding of customer behavior, streamline inventory management, forecast demand, and improve overall operational effectiveness.

AI-powered Personalization: Couture.ai uses AI-driven recommendations and targeting to deliver personalized experiences to customers. By analyzing customer preferences, purchase history, and relevant data, their algorithms offer personalized product recommendations, offers, and promotions. This enhances customer engagement and boosts conversions for retailers.

Demand Forecasting: Within Couture.ai's range of solutions, they offer advanced capabilities for demand forecasting, empowering retailers to make accurate predictions regarding customer product demand. By analyzing historical sales data, market trends, and other relevant factors, their AI algorithms generate insights that optimize inventory planning, prevent stockouts, reduce wastage, and enhance overall supply chain management.

Pricing Optimization: Couture.ai provides retailers with AI-driven solutions for pricing optimization. These solutions help retailers determine the most effective pricing strategy for their products. By analyzing market dynamics, competitor pricing, and customer preferences, their algorithms offer recommendations on the ideal pricing levels that can maximize revenue and profitability.

Fraud Detection and Loss Prevention: Couture.ai supports retailers in addressing fraud and reducing losses by providing AI-based solutions for fraud detection and loss prevention. Through the analysis of diverse data sources, such as transaction data and behavioral patterns, their algorithms have the capability to identify suspicious activities and potential instances of fraud in real-time. This enables timely intervention and effective measures to mitigate the impact of fraudulent activities.

Customer Sentiment Analysis: The company uses NLP techniques to analyze customer sentiment from social media and other textual data. This provides insights into customer feedback and preferences, empowering retailers to make data-driven decisions and enhance customer satisfaction.

Couture.ai's solutions aim to empower retailers with data-driven decision-making capabilities, enhance customer experiences, and drive operational efficiency.

Features of Couture AI Platform

The features of Couture AI's platform include:

Seamless Integration:The platform facilitates seamless integration of applications, code, and images, enabling users to automate and manage infrastructure effortlessly. This capability simplifies the process of connecting various components and streamlines the management of the underlying infrastructure.

Training Manager:Couture AI's platform incorporates a training manager that simplifies the administration of machine learning models. Users can easily optimize hyperparameters and oversee models throughout their lifecycle, resulting in time savings and decreased compute expenses.

Project-Based Segregation:The platform provides the capability to segregate work based on projects, enabling users to concurrently handle multiple projects. Each project can have its unique set of pipelines,

datasets, notebooks, models, and other resources. This functionality allows users to effectively organize their work and maintain clear separation between different projects.

Universal Pipeline Representation:Couture AI's platform offers a cohesive configuration system for building pipelines. This unified representation seamlessly connects the development and deployment phases, streamlining the process of constructing and implementing machine learning pipelines. It fosters consistency and enhances collaboration throughout the workflow.

Accelerated End-to-End Cycle:The platform automates laborious tasks in MLOps (Machine Learning Operations) and ML engineering workflows, expediting the complete cycle of developing and deploying machine learning solutions. By automating repetitive tasks, users can dedicate more attention to their expertise and enhance productivity.

Minimized Glue Code: Couture AI's platform reduces the reliance on custom-written glue code, which is used to bridge incompatible software components. This functionality decreases the effort needed for users to manage disparate formats or incompatible schemas, simplifying the process of working with diverse data sources and systems.

Deployment Manager:The platform incorporates a deployment manager for scaling pipelines effortlessly, transitioning from local deployment to production environments for batch or online serving. This feature streamlines the deployment and management of machine learning models, ensuring a smooth transition from development to production.

Collectively, these features work towards improving the efficiency, scalability, and automation of machine learning workflows. This empowers users to concentrate on their core expertise and expedite the development and deployment of AI solutions.

Literature Review

Forecasting demand In today's organizations, which are subject to abrupt and enormous changes that affect even the most established structures and where all requirements of the business sector need accurate and practical reading into the future, the forecasts are becoming very crucial since they are the sign of survival and the language of business in the world. A forecast is a science of estimating the future level of some variables. The variable is most often demand, but it can also be something else, such as supply or price. Forecasting is the operation of making assumptions about the future values of studied variables. In manufacturing, forecasting demands is among the most crucial issues in inventory management; it can be used in various operational planning activities during the production process: capacity planning, used-product acquisition management. For both types of supply chain processes "push/pull," the demand forecasts are considered the ground of supply chain's planning. The pull processes in the supply chain are realized with reference to customer demand, while all push processes are realized in anticipation of customer demand. A company must take into consideration such factors before selecting a suitable forecasting methodology because the choice of a methodology is not as simple as it seems. Forecasting methods are categorized according to four types: qualitative, time series, causal, and simulation.

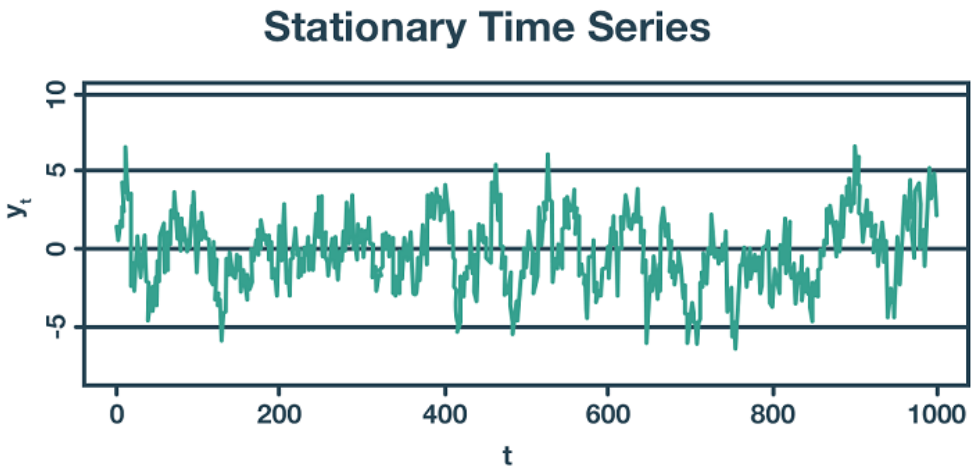
A time series is nothing but observations according to the chronological order of time. Time series forecasting models use mathematical techniques that are based on historical data to forecast demand. It is founded on the hypothesis that the future is an expansion of the past; that's why we can definitely use historical data to forecast future demand. Many studies about demand forecasting by time series analysis have been done in several domains. They encircle demand forecasting for food product sales, tourism, maintenance repair parts, electricity, automobile, and some other products and services. By time series analysis, the forecasting accuracies depend on the characteristics of the time series of demand. If the transition curves show stability and periodicity, we will reach high forecasting accuracies, whereas we can't expect high accuracies if the curves contain highly irregular patterns.

There are 4 characteristics in a time-series pattern:

- Trend: Long-term increase or decrease in the data. The trend can be any function, such as linear or exponential, and can change direction over time.
- Seasonality: Repeating cycle in the series with fixed frequencies (hour of the day, week, month, year, etc.). A seasonal pattern exists of a fixed known period.
- Cyclicity: Occurs when the data rise and fall, but without a fixed frequency and duration caused, for example, by economic conditions.
- Noise: The random variation in the series.

The time series pattern can be stationary or non-stationary.

A time series whose statistical properties, such as mean, variance, etc., remain constant over time, is called a stationary time series.



A time series whose statistical properties change over time is called a non-stationary time series. Thus a time series with a trend or seasonality is non-stationary in nature. This is because the presence of trend or seasonality will affect the mean, variance and other properties at any given point in time.

ARIMA Model
(AutoRegressive Integrated Moving Average)

ARIMA, is actually a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values. An ARIMA model is characterized by 3 terms: p, d, q where p is the order of the AR term q is the order of the MA term d is the number of differencing required to make the time series stationary.

'p' is the order of the 'Auto Regressive' (AR) term. It refers to the number of lags of Y to be used as predictors. And 'q' is the order of the 'Moving Average' (MA) term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.

The value of d, is the minimum number of differencing needed to make the series stationary. And if the time series is already stationary, then d = 0.

AR (Auto Regressive Model)

A pure Auto Regressive (AR only) model is one where Yt depends only on its own lags. That is, Yt is a function of the 'lags of Yt'.
{t-1} is the lag1 of the series, beta1 is the coefficient of lag1 that the model estimates and `alpha` is the intercept term, also estimated by the model.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

MA (Moving Average Model)

A pure Moving Average (MA only) model is one where Yt depends only on the lagged forecast errors. The error terms are the errors of the autoregressive models of the respective lags.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

An ARIMA model is one where the time series was differenced at least once to make it stationary and you combine the AR and the MA terms. So the equation becomes:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

ARIMA model in words:

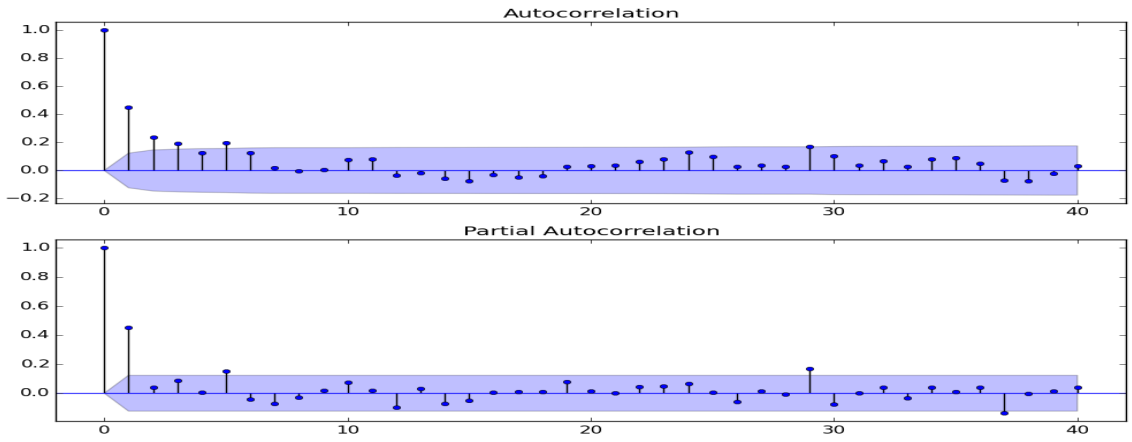
Predicted Yt = Constant + Linear combination Lags of Y (upto p lags) + Linear Combination of Lagged forecast errors (upto q lags)

How to find p?(order of AR Model)

The required number of AR terms can be found by inspecting the Partial Autocorrelation (***PACF***) plot. Partial autocorrelation can be imagined as the correlation between the series and its lag, after excluding the contributions from the intermediate lags. So, PACF sort of conveys the pure correlation between a lag and the series. That way, you will know if that lag is needed in the AR term or not.

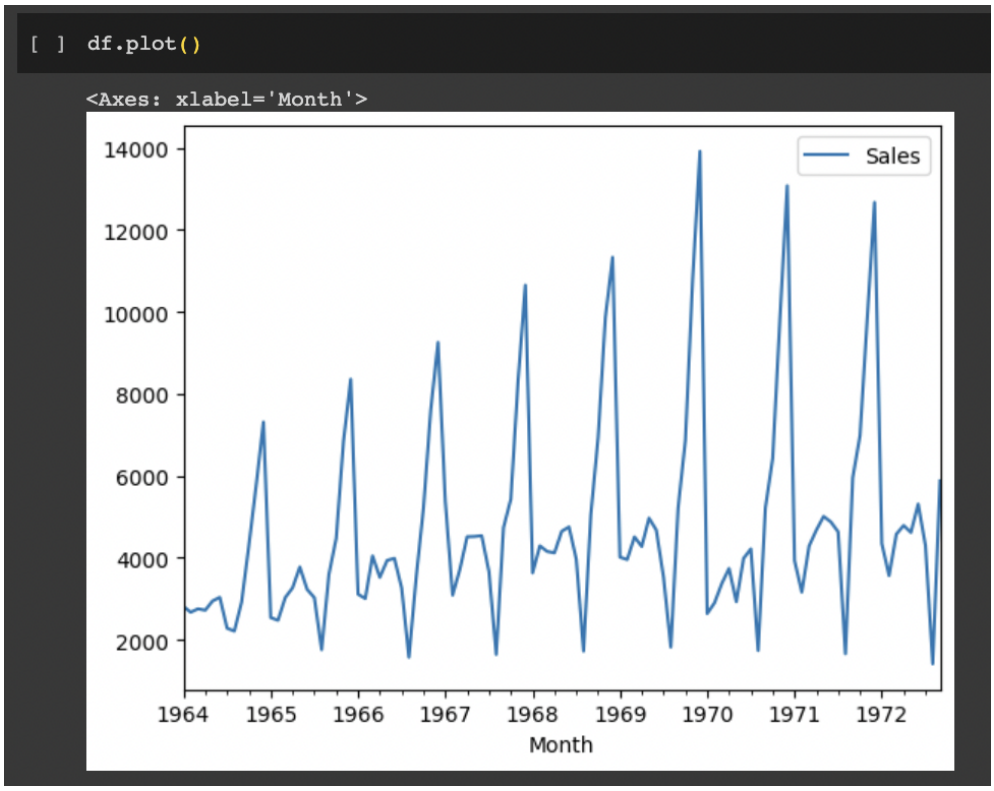
How to find q?(order of MA model)

Just like how we looked at the PACF plot for the number of AR terms, you can look at the ***ACF*** plot for the number of MA terms. An MA term is technically, the error of the lagged forecast. The ACF tells how many MA terms are required to remove any autocorrelation in the stationarised series.



Sales Forecasting of Sweaters using the ARIMA Model and Google Colab

This project was about forecasting the demand of sweaters for the next couple of years by importing a dataset from kaggle which contained the data of the number of sweaters sold from 1964-1972. The dataset was imported using a function of the pandas library. The following plot shows the sales plotted against the year.



The plot shows that the data follows a seasonal pattern within every 12 months. The sales increases every year during the winter months. Since the data is seasonal, it is non-stationary. Inferences derived from a non-stationary process lack reliability as the statistical characteristics of the data constantly change over time. When conducting analysis, one is typically interested in parameters such as the mean, variance, and so on. However, if these parameters are continuously changing, estimating them by averaging over time will yield inaccurate results. Hence, it is easier to analyze stationary data, and any forecast made using non-stationary data would be misleading and incorrect. Due to this reason, many statistical procedures employed in time series analysis assume that the underlying time series data is stationary.

This assumption is necessary because most time series forecasting methods predict that the statistical properties of the time series will remain consistent in the future, as they have been in the past.

ADF Test for Stationarity:

- The **Augmented Dickey-Fuller test** is one of the most popular tests to check for stationarity. It tests the below hypothesis.
- Null Hypothesis, H0: The time series is not stationary.
- Alternative Hypothesis, H1: The time series is stationary.

The adfuller method from the statsmodels library is used to perform this test in Python and compare the value of the test statistics or the p-value.

- If the p-value is less than or equal to 0.05 or the absolute value of the test statistics is greater than the critical value, you reject H0 and conclude that the time series is stationary.
- If the p-value is greater than 0.05 or the absolute value of the test statistics is less than the critical value, you fail to reject H0 and conclude that the time series is not stationary.

```
[ ]  ### Testing For Stationarity

     from statsmodels.tsa.stattools import adfuller
     test_result=adfuller(df['Sales'])
     #Ho: It is non stationary
     #H1: It is stationary

     def adfuller_test(sales):
         result=adfuller(sales)
         labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
         for value,label in zip(result,labels):
             print(label+' : '+str(value) )
         if result[1] <= 0.05:
             print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary")
         else:
             print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")

     adfuller_test(df['Sales'])
```

```
ADF Test Statistic : -1.8335930563276215
p-value : 0.36391577166024586
#Lags Used : 11
Number of Observations Used : 93
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

Since the p-value obtained is greater than 0.05, we do not reject the null hypothesis and therefore the data is non-stationary. We convert the data from non-stationary to stationary by differencing it.

```
[ ] df['Seasonal First Difference']=df['Sales']-df['Sales'].shift(12)
    df.head(14)
```

Month	Sales	Sales First Difference	Seasonal First Difference
1964-01-01	2815.0	NaN	NaN
1964-02-01	2672.0	-143.0	NaN
1964-03-01	2755.0	83.0	NaN
1964-04-01	2721.0	-34.0	NaN
1964-05-01	2946.0	225.0	NaN
1964-06-01	3036.0	90.0	NaN
1964-07-01	2282.0	-754.0	NaN
1964-08-01	2212.0	-70.0	NaN
1964-09-01	2922.0	710.0	NaN
1964-10-01	4301.0	1379.0	NaN
1964-11-01	5764.0	1463.0	NaN
1964-12-01	7312.0	1548.0	NaN
1965-01-01	2541.0	-4771.0	-274.0
1965-02-01	2475.0	-66.0	-197.0

```
[ ] ## Again test dickey fuller test
    adfuller_test(df['Seasonal First Difference'].dropna())
```

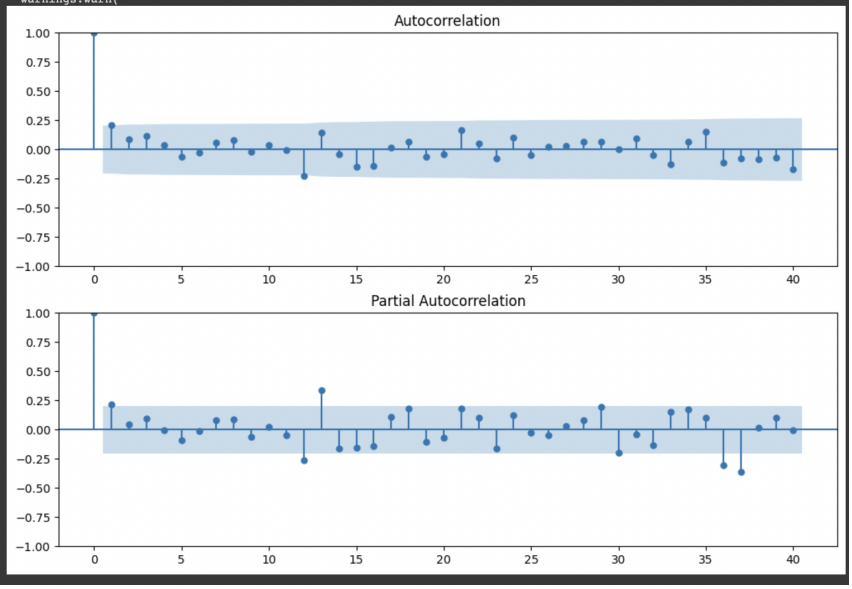
```
ADF Test Statistic : -7.626619157213166
p-value : 2.0605796968136632e-11
#Lags Used : 0
Number of Observations Used : 92
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary
```

Since the p-value obtained for the new dataset after differencing is much less than 0.05, we reject the null hypothesis and conclude that the data is stationary.
Next, the order of the ARIMA model is determined by plotting the ACF and PACF plots. These were the results obtained:

```
[ ] import statsmodels.api as sm
```

```
[ ] from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
    fig = plt.figure(figsize=(12,8))
    ax1 = fig.add_subplot(211)
    fig = sm.graphics.tsa.plot_acf(df['Seasonal First Difference'].iloc[13:],lags=40,ax=ax1)
    ax2 = fig.add_subplot(212)
    fig = sm.graphics.tsa.plot_pacf(df['Seasonal First Difference'].iloc[13:],lags=40,ax=ax2)
```

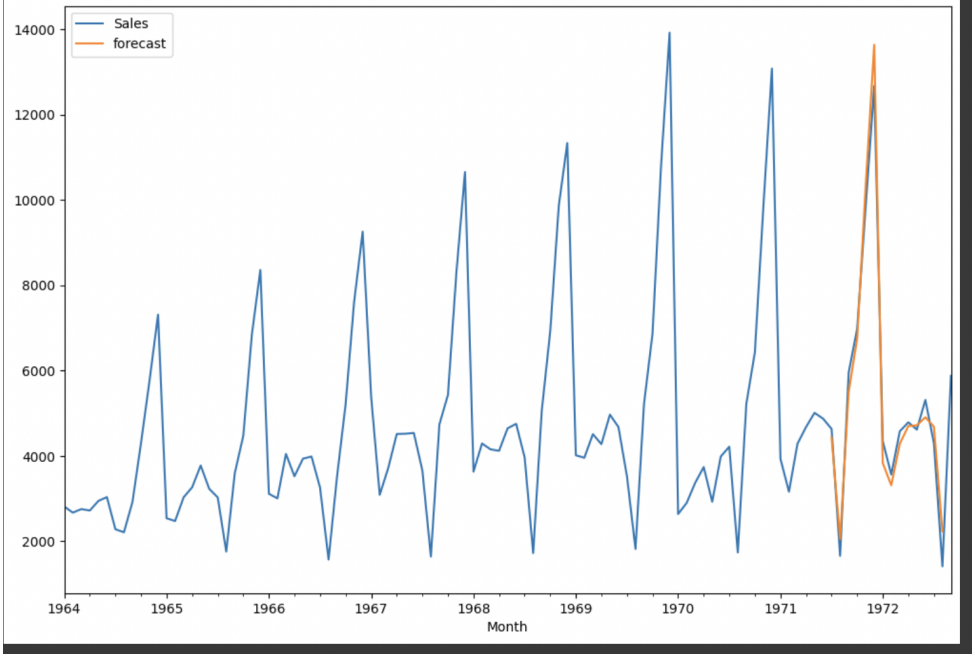
```
/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'yw' can
  warnings.warn(
```



Thus the values of p and q obtained from the above plots comes out to be 1 for both. Therefore the order of the ARIMA model that was used to train the model was (1,1,1). We divide the dataset into train and test data and try to use the model that we predicted using the train data to predict values for the test data set and check the accuracy of the model used.

```
df['forecast']=results.predict(start=90,end=103,dynamic=True)
df[['Sales','forecast']].plot(figsize=(12,8))
```

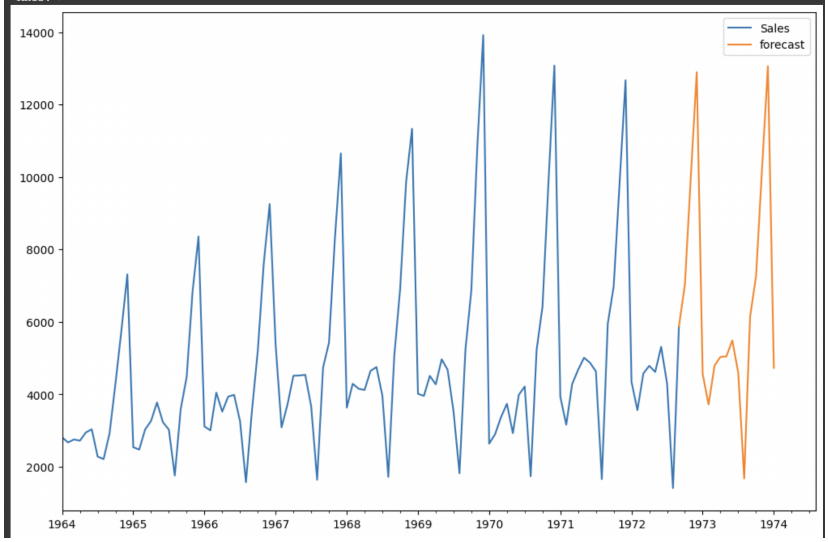
```
<Axes: xlabel='Month'>
```



The predicted data is quite close to the actual values and hence we use this model - ARIMA(1,1,1) to forecast future values. We use this value to forecast the demand of sweaters for the next couple of years. The following plot was obtained:

```
future_df=pd.concat([df,future_datest_df])
future_df['forecast'] = results.predict(start = 104, end = 120, dynamic= True)
future_df[['Sales', 'forecast']].plot(figsize=(12, 8))
```

```
<Axes: >
```



In the above code, I have used only the previous values of the sales of sweaters to determine the future values of the number of sweaters sold, but this might depend on a lot of other variables.

One of the main disadvantages of ARIMA models is that they are not suitable for multivariate time series data, meaning they have more than one variable.

ARIMA models cannot capture the interactions and dependencies between different variables, such as the effects of external factors, such as marketing, competition, or events, on the time series.

ARIMA models also require a lot of data preprocessing and tuning, as you need to check the stationarity, autocorrelation, and partial autocorrelation of the data, and find the optimal values of the parameters using trial and error.

Hence in most time series forecasting, we use some other advanced models using machine learning algorithms such as XGBoost,Seq2Seq Encoder-Decoder Neural Networks and LSTM(Long Short-Term Memory).

XGBoost is an efficient implementation of gradient boosting for classification and regression problems. XGBoost can also be used for time series forecasting, although it requires that the time series dataset be transformed into a supervised learning problem first.

The LSTM cell adds long-term memory in an even more performant way because it allows even more parameters to be learned. This makes it the most powerful [Recurrent Neural Network] to do forecasting, especially when you have a longer-term trend in your data. LSTMs are one of the state-of-the-art models for forecasting at the moment.

XGBoost
(Extreme Gradient Boosting)

XGBoost is a powerful approach for building supervised regression models. The objective function contains loss function and a regularization term. It tells about the difference between actual values and predicted values, i.e how far the model results are from the real values. The most common loss function in XGBoost for regression problems is reg:linear.

Ensemble learning involves training and combining individual models (known as base learners) to get a single prediction, and XGBoost is one of the ensemble learning methods. XGBoost expects to have the base learners which are uniformly bad at the remainder so that when all the predictions are combined, bad predictions cancel out and better one sums up to form final good predictions.

Here's how XGBoost can be used for time series forecasting:

Step 1: Make an Initial Prediction and Calculate Residuals

This prediction can be anything. But mostly we assume our initial prediction as the average value of the variables we want to predict.

We then calculate the residuals using the following formula:

Residuals = Observed values - Predicted Values

Step 2: Build an XGBoost Tree

Each tree starts with a single leaf and all the residuals go into that leaf. For example:

Next, we need to calculate something called a **Similarity Score** of this leaf.

$$\text{Similarity Score} = \frac{(Sum\ of\ Residuals)^2}{Number\ of\ Residuals + \lambda}$$

Regularization Parameter

λ (lambda) is a regularization parameter that reduces the prediction's sensitivity to individual observations and prevents the overfitting of data (this is when a model fits exactly against the training dataset). The default value of λ is 1.

Next, we see if we can do a better job clustering the residuals if we split them into two groups using thresholds based on our predictor variables. Splitting the *Residuals* basically means that we are adding branches to our tree. First, try splitting the leaf using one of the predictor variables. And then calculate the Similarity Scores for the left and right leaves of the above split.

Next we quantify how much better the leaves cluster similar *Residuals* than the root does. We do this by calculating the **Gain** of splitting the *Residuals* into two groups. If the **Gain** is positive, then it's a good idea to split, otherwise, it is not.

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

Then we compare this **Gain** to those of the splits using the other variable.

Step 3: Prune the Tree

Pruning is another way we can avoid overfitting the data. To do this we start from the bottom of our tree and work our way up to see if a split is valid or not. To establish validity, we use γ (gamma). If **Gain — γ** is positive then we keep the split, otherwise, we remove it. The default value of γ is 0.

Step 4: Calculate the Output Values of Leaves

Next, we calculate a single value in our leaf nodes because we can not have a leaf node giving us multiple outputs.

$$Output\ Value\ =\ \frac{Sum\ of\ Residuals}{Number\ of\ Residuals\ +\ \lambda}$$

Step 5: Make New Predictions

We can make new predictions using the following formula:

Initial predicted value + (eta)*output value

The XGBoost Learning Rate is ε (eta) and the default value is 0.3.

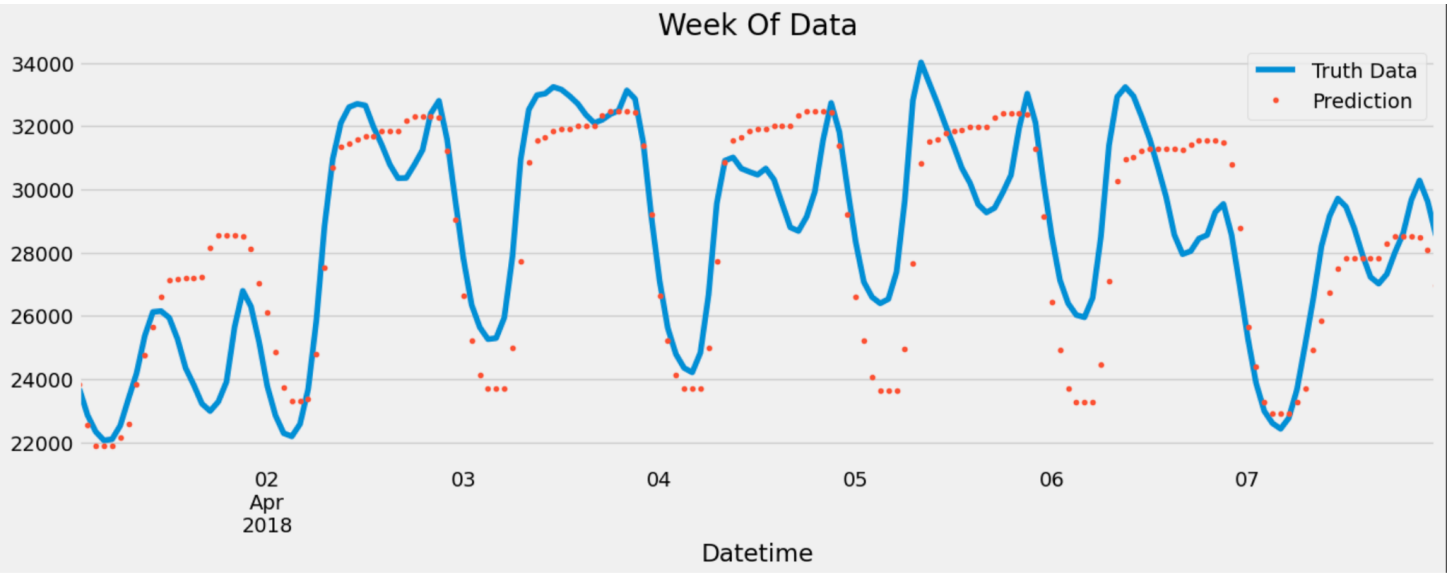
Step 6: Calculate Residuals Using the New Predictions

Step 7: Repeat Steps 2–6

If the tree we built at each iteration is indicated by T_i, where *i* is the current iteration, then the formula to calculate predictions is:

$$70 + \varepsilon x T_1 + \varepsilon x T_2 + \varepsilon x T_3 + \dots + \varepsilon x T_i$$

Where 70, is the initial prediction.
The following are the predictions I obtained with the dataset that I had used for the ARIMA model as well:

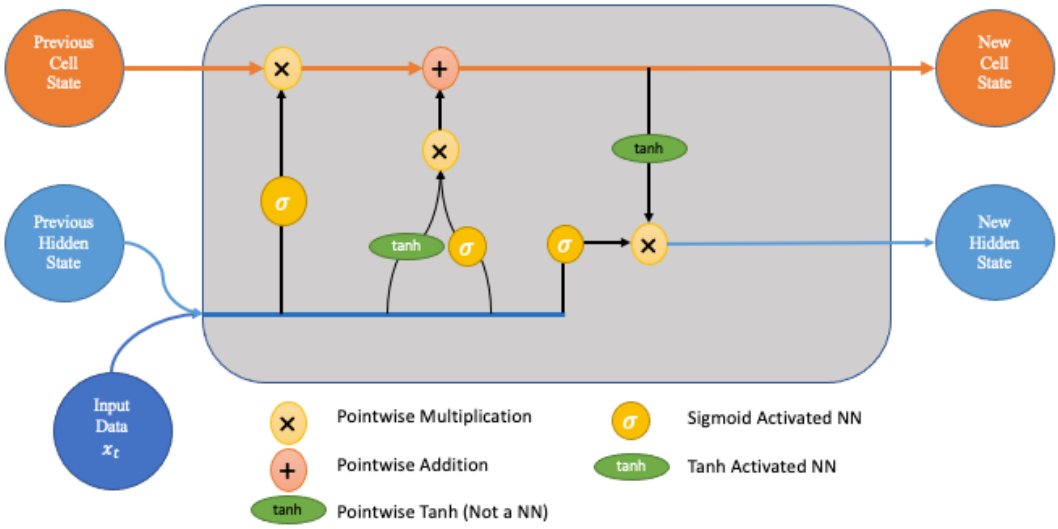


LSTM
(Long Short-Term Memory)

LSTMs are a special kind of RNN — capable of learning long-term dependencies by remembering information for long periods is the default behavior.

All RNN are in the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have a chain-like structure, but the repeating module is a bit different structure. Instead of having a single neural network layer, four interacting layers are communicating extraordinarily.



Working Of LSTMs in RNN:

Step 1: Decide How Much Past Data It Should Remember

The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state (ht-1) along with the current input xt and computes the function.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

f_t = forget gate
Decides which information to delete that is not important from previous time step

Step 2: Decide How Much This Unit Adds to the Current State

In the second layer, there are two parts. One is the sigmoid function, and the other is the tanh function. In the sigmoid function, it decides which values to let through (0 or 1). The tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t = input gate
Determines which information to let through based on its significance in the current time step

Step 3: Decide What Part of the Current Cell State Makes It to the Output

The third step is to decide what the output will be. First, we run a sigmoid layer, which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

o_t = output gate
Allows the passed in information to impact the output in the current time step

Using LSTM to predict future values:

1.Creating a data structure with 5-time steps and 1 output

```
[ ] # [[[1], [2], [3], [4], [5]]] [6] #supervised learning, if multivariate add more variables in each list
    # [[[2], [3], [4], [5], [6]]] [7] #a matrix(3D tensor) where each row is our inputs and it has a corresponding label
    # [[[3], [4], [5], [6], [7]]] [8]

    def df_to_X_y(df, window_size=5):
      df_as_np = df.to_numpy() #converting the dataframe to numpy
      X = []
      y = []
      for i in range(len(df_as_np)-window_size):
        row = [[a] for a in df_as_np[i:i+window_size]]
        X.append(row)
        label = df_as_np[i+window_size]  #actual value for 1 row of inputs(y)
        y.append(label)
      return np.array(X), np.array(y)
```

```
WINDOW_SIZE = 5
X1, y1 = df_to_X_y(temp, WINDOW_SIZE)
X1.shape, y1.shape
```

```
((70086, 5, 1), (70086,))
```

2.Initializing the RNN and Adding the LSTM layers and Output layer

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint #for saving the model that does the best on the validation
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam

model1 = Sequential()
model1.add(InputLayer((5, 1)))
model1.add(LSTM(64))
model1.add(Dense(8, 'relu'))
model1.add(Dense(1, 'linear'))

model1.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 64)                16896

 dense_10 (Dense)            (None, 8)                 520

 dense_11 (Dense)            (None, 1)                 9

=================================================================
Total params: 17,425
Trainable params: 17,425
Non-trainable params: 0
_________________________________________________________________
```

```
cp1 = ModelCheckpoint('model1/', save_best_only=True) #save_best_only will choose the model with the lowest validation loss
model1.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.0001), metrics=[RootMeanSquaredError()]) #using a very small learning rate
```

3.Fitting the RNN to the training set

```
model1.fit(X_train1, y_train1, validation_data=(X_val1, y_val1), epochs=10, callbacks=[cp1])
```

```
Epoch 1/10
1873/1875 [============================>.] - ETA: 0s - loss: 36.1392 - root_mean_squared_error: 6.0116WARNING:absl:Found untra
1875/1875 [==============================] - 19s 9ms/step - loss: 36.1074 - root_mean_squared_error: 6.0089 - val_loss: 1.4151
Epoch 2/10
1875/1875 [==============================] - ETA: 0s - loss: 2.2349 - root_mean_squared_error: 1.4950WARNING:absl:Found untrac
1875/1875 [==============================] - 17s 9ms/step - loss: 2.2349 - root_mean_squared_error: 1.4950 - val_loss: 0.6180
Epoch 3/10
1875/1875 [==============================] - ETA: 0s - loss: 0.9529 - root_mean_squared_error: 0.9761WARNING:absl:Found untrac
1875/1875 [==============================] - 16s 8ms/step - loss: 0.9529 - root_mean_squared_error: 0.9761 - val_loss: 0.5172
Epoch 4/10
1874/1875 [============================>.] - ETA: 0s - loss: 0.7399 - root_mean_squared_error: 0.8602WARNING:absl:Found untrac
1875/1875 [==============================] - 20s 11ms/step - loss: 0.7399 - root_mean_squared_error: 0.8601 - val_loss: 0.5069
Epoch 5/10
1872/1875 [============================>.] - ETA: 0s - loss: 0.6900 - root_mean_squared_error: 0.8306WARNING:absl:Found untrac
1875/1875 [==============================] - 17s 9ms/step - loss: 0.6898 - root_mean_squared_error: 0.8306 - val_loss: 0.5051
Epoch 6/10
1869/1875 [============================>.] - ETA: 0s - loss: 0.6719 - root_mean_squared_error: 0.8197WARNING:absl:Found untrac
1875/1875 [==============================] - 16s 8ms/step - loss: 0.6719 - root_mean_squared_error: 0.8197 - val_loss: 0.4979
Epoch 7/10
1875/1875 [==============================] - ETA: 0s - loss: 0.6620 - root_mean_squared_error: 0.8136WARNING:absl:Found untrac
1875/1875 [==============================] - 17s 9ms/step - loss: 0.6620 - root_mean_squared_error: 0.8136 - val_loss: 0.4949
Epoch 8/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6555 - root_mean_squared_error: 0.8096 - val_loss: 0.5238
Epoch 9/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6512 - root_mean_squared_error: 0.8069 - val_loss: 0.4958
Epoch 10/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.6484 - root_mean_squared_error: 0.8052 - val_loss: 0.5105
<keras.callbacks.History at 0x7bcf0794ada0>
```

4.Getting the predicted values

```
[ ]  train_predictions = model1.predict(X_train1).flatten()
     train_results = pd.DataFrame(data={'Train Predictions':train_predictions, 'Actuals':y_train1})
     train_results
```

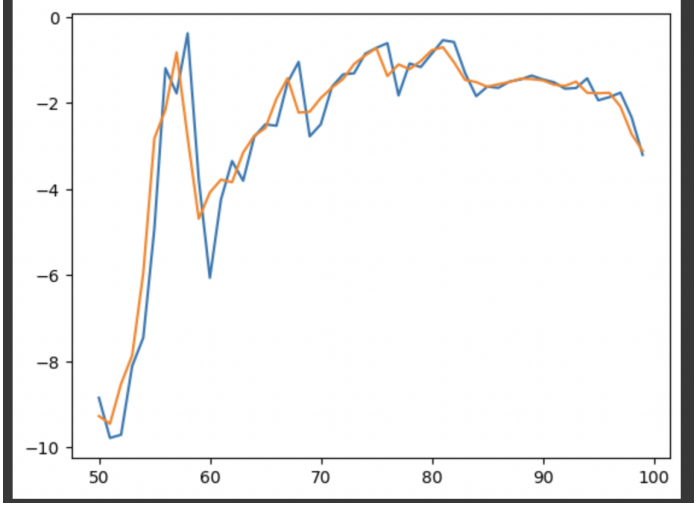
```
1875/1875 [==============================] - 6s 3ms/step
```

	Train Predictions	Actuals
0	-9.973878	-9.67
1	-9.815421	-9.17
2	-8.967499	-8.10
3	-7.517393	-7.66
4	-7.385703	-7.04
...	...	...
59995	6.059141	6.07
59996	7.359452	9.88
59997	12.202259	13.53
59998	16.090296	15.43
59999	16.265871	15.54

5.Visualizing the results of the predicted and real values

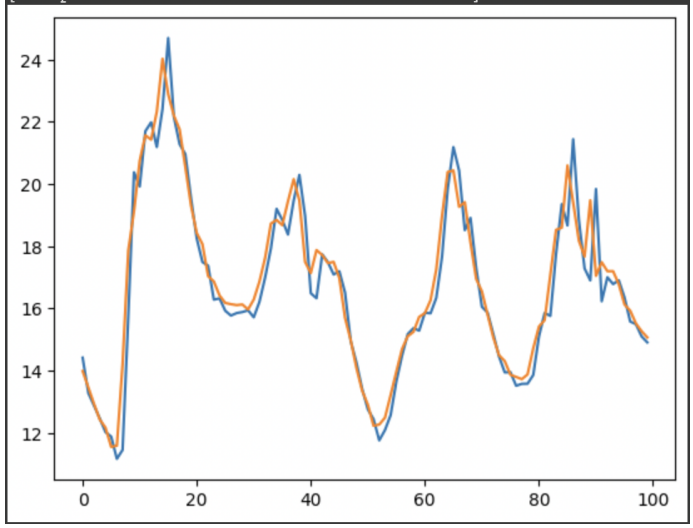
```
import matplotlib.pyplot as plt
plt.plot(train_results['Train Predictions'][50:100])
plt.plot(train_results['Actuals'][50:100])
```

```
[<matplotlib.lines.Line2D at 0x7bcf00272020>]
```




```
plt.plot(test_results['Test Predictions'][:100])
plt.plot(test_results['Actuals'][:100])
```

```
[<matplotlib.lines.Line2D at 0x7bcf10c1b790>]
```



CONCLUSION

Time series forecasting enables businesses to make informed decisions regarding production levels, procurement, and resource allocation, resulting in improved operational efficiency and cost-effectiveness. By having a clear understanding of expected demand patterns, companies can strategically plan their inventory, ensure product availability, and enhance customer satisfaction.

ARIMA offers valuable insights into the statistical properties of the time series data, allowing businesses to identify trends, seasonality, and other significant patterns that influence demand fluctuations. This information helps companies adapt their marketing strategies, promotional activities, and pricing decisions to align with customer preferences and market dynamics.

XGBoost performs exceptionally well in handling complex relationships between features and the target variable. It is particularly useful when dealing with datasets that have multiple relevant features and interactions between them. XGBoost's ability to model nonlinear relationships makes it a strong contender when the underlying patterns are intricate and not easily captured by linear models. Additionally, XGBoost is computationally efficient and can handle large datasets efficiently.

On the other hand, LSTM is a specialized type of recurrent neural network designed explicitly for sequential data like time series. Its key strength lies in capturing temporal dependencies and patterns over time. LSTM excels at detecting long-term dependencies and is well-suited for datasets with changing trends, non-stationarity, and other complex temporal patterns. It can adapt and learn from historical information to make accurate forecasts for future time periods.

In summary, by adopting time series forecasting for demand forecasting, businesses in the fashion apparel industry and beyond can gain a competitive edge by making data-driven decisions, improving customer satisfaction, optimizing resource utilization, and ultimately driving business growth. ML models serve as a valuable tool for businesses aiming to navigate the complex dynamics of demand fluctuations and achieve sustainable success in today's highly competitive marketplace.

REFERENCES

- https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/
- Andrew Ng, Machine Learning, Stanford University, Coursera. https://www.coursera.org/learn/machine-learning
- https://www.simplilearn.com/tutorials/statistics-tutorial/what-is-time-series-analysis
- https://towardsdatascience.com/machine-learning-part-19-time-series-and-autoregressive-integrated-moving-average-model-arima-c1005347b0d7
- https://statquest.org/
- http://www.holehouse.org/mlclass/
- https://towardsdatascience.com/xgboost-regression-explain-it-to-me-like-im-10-2cf324b0bbdb
- https://machinelearningmastery.com/xgboost-for-time-series-forecasting/
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

GLOSSARY

- Demand Planning: The strategic process of anticipating and managing customer demand to optimize inventory levels, production, and supply chain operations.

- Model Selection: The process of determining the optimal configuration of the ARIMA model, including the order of autoregressive, moving average, and differencing components, based on statistical criteria and diagnostic tests.

- Seasonality: Patterns that repeat at regular intervals within a time series, such as daily, weekly, or yearly. Seasonality can influence demand and is considered in ARIMA modeling to capture cyclical fluctuations

- Stationarity: A key assumption in ARIMA modeling, which implies that the statistical properties of a time series, such as mean and variance, remain constant over time. Stationarity is crucial for accurate forecasting using ARIMA.

- Time Series Analysis: The examination of data collected over a specific period, typically at regular intervals, to identify patterns, trends, and other characteristics that can be used for forecasting future values.

- Forecasting Horizon: The time period for which future demand values are predicted using the ARIMA model. It can range from short-term forecasts (e.g., days or weeks) to long-term forecasts (e.g., months or years).

- Gradient Boosting: It is an ensemble learning method that combines the predictions of multiple individual models (typically decision trees) to create a more accurate and robust final prediction.

- Recurrent Neural Network: A Recurrent Neural Network (RNN) is a type of artificial neural network designed specifically for processing sequential data. RNNs can retain and utilize information from previous time steps in their computations. This makes them well-suited for tasks involving time series data.