

# PT/br

## Introdução

Abaixo estarei mostrando como foi realizado o estudo de previsão de preço de shampoo do dataset Shampoo Sales que está presente no Kaggle pelo link abaixo e também explicarei o código que foi feito e o resultado que foi obtido.

Kaggle Dataset: <https://www.kaggle.com/redwankarimsony/shampoo-saled-dataset>

## Séries Temporais

Em estatística, uma série temporal é uma coleção de observações feitas sequencialmente ao longo do tempo. Em modelos de regressão linear, a ordem das observações é irrelevante para a análise, em séries temporais a ordem dos dados é fundamental.

Uma característica muito importante deste tipo de dados é que as observações vizinhas são dependentes e o interesse é analisar e modelar essa dependência.

As séries temporais existem nas mais variadas áreas de aplicação, como: finanças, marketing, economia, seguros, demografia, ciências sociais, meteorologia, energia, epidemiologia, etc.

Dito de outra forma, uma série temporal é uma sequência de números coletados em intervalos regulares durante um período de tempo.

## Estudo de Caso

O estudo foi feito para fins educacionais próprios, na qual houve a necessidade de se aprender melhor sobre séries temporais.

Foi utilizado o Python como linguagem, a IDE PyCharm e a biblioteca principal ARIMA.

O dataset é composto apenas de duas colunas, sendo a primeira do preço de venda do shampoo, e a segunda a data relacionada ao preço da venda.

Abaixo estou mostrando e explicando o código além dos resultados.

### 1- Importando as bibliotecas

Utilizei as seguintes bibliotecas para realizar o estudo:

```
import warnings
import pandas as pd
import csv
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import TimeSeriesSplit
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import adfuller
```

## 2- Importando o arquivo

Há apenas um arquivo csv na competição chamado “shampoo\_sales.csv”

```
d_shampoo = "C:/Users/mcelopes/Desktop/shampoo_sales.csv"
```

```
shampoo = pd.read_csv(d_shampoo, sep=",", encoding="UTF-8", dtype=str,
error_bad_lines=False, quoting=csv.QUOTE_NONE)
```

## 3- Limpeza e Tratamento

Foi criada uma função para tratamento das colunas, a função trata\_coluna recebe como parametro o dataset e mantém o padrão de upper case em nome e conteúdo de cada coluna, além de remover espaços.

```
def trata_coluna(data):
    data.columns = [x.upper() for x in data.columns]

    for coluna in data:
        data[coluna] = data[coluna].str.strip()
        data[coluna] = data[coluna].str.replace("'", "")
        data[coluna] = data[coluna].str.upper()
```

Após a função ser chamada, seguimos com outros tratamentos como a renomeação das colunas que estão com "" a mais, conversão do tipo da coluna SALES para numérico, e também a conversão da coluna MONTH para o tipo datetime.

```
trata_coluna(shampoo)
```

```
shampoo = shampoo.rename({"MONTH": "MONTH", "SALES": "SALES"}, axis=1)
```

```
shampoo["SALES"] = pd.to_numeric(shampoo["SALES"])
```

```
shampoo["MONTH"] = "0" + shampoo["MONTH"]
```

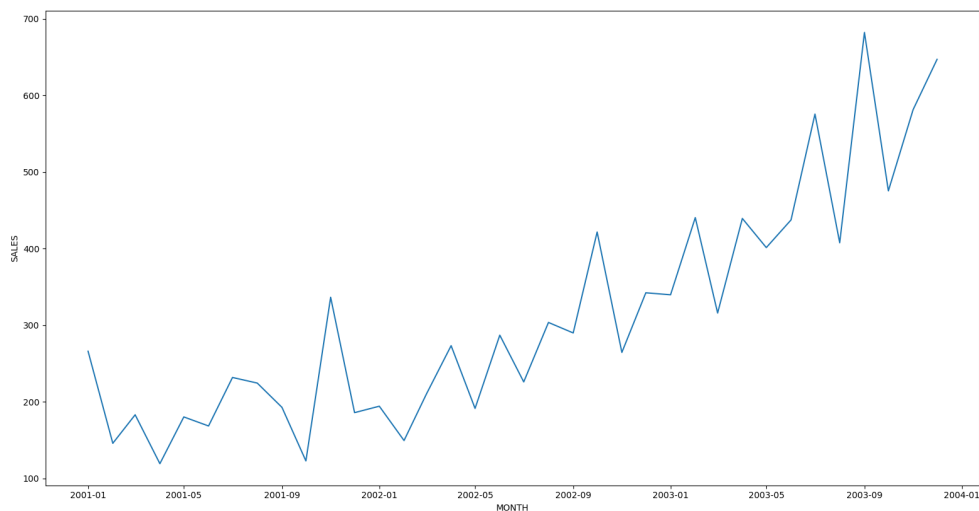
```
shampoo["MONTH"] = pd.to_datetime(shampoo["MONTH"], format="%y-%m",  
yearfirst=True)
```

## 4- Análises da base atual

### Visão Geral

Foi iniciada então a análise do dataset, primeiro temos uma visão geral sobre o dataset.

```
sns.lineplot(data=shampoo, x="MONTH", y="SALES")
```

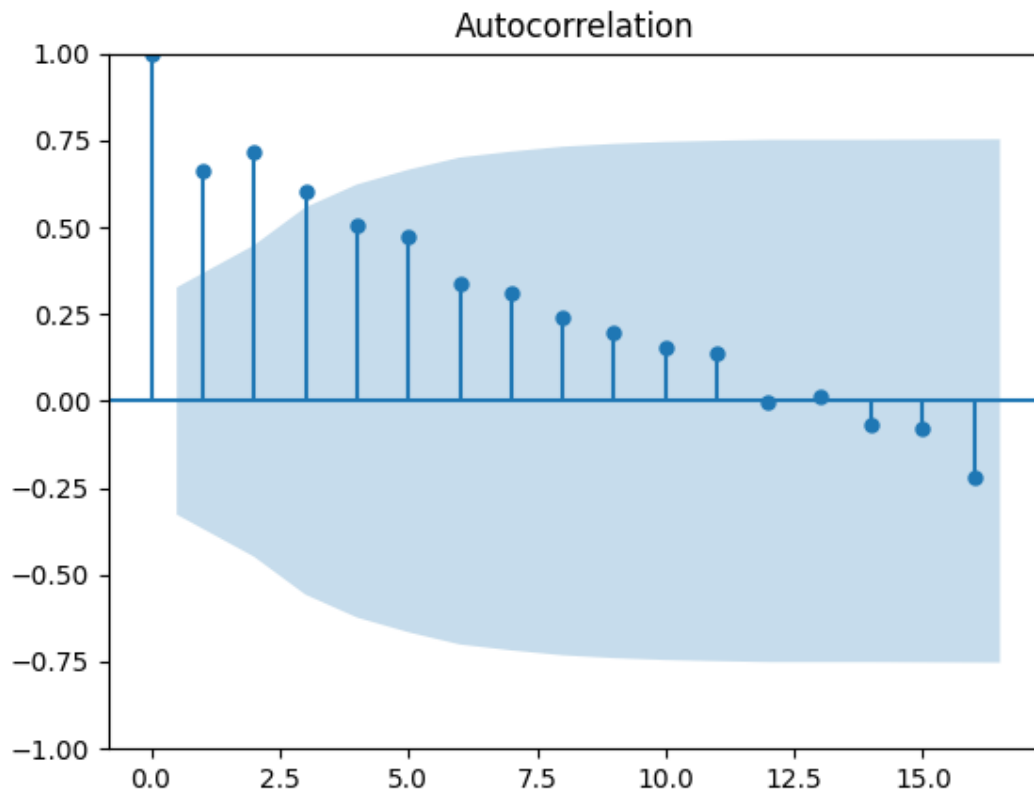


Podemos ver que com o passar do tempo, o preço do shampoo tende a aumentar, tendo algumas pequenas variações.

### Autocorrelação(FAC) e Autocorrelação Parcial(FACP)

Analizamos também a FAC e a FACP dos dados.

```
plot_acf(shampoo["SALES"].values.astype("float32"))
```

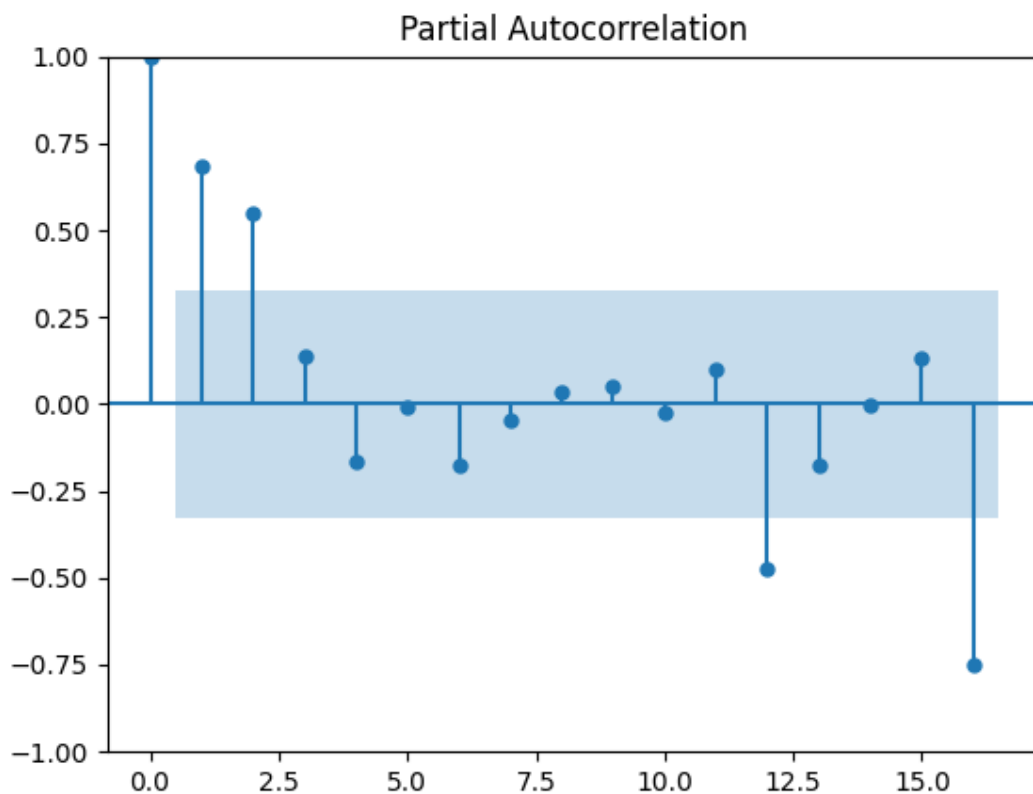


Como nos é mostrado no gráfico de FAC, a partir de 2,5 anos se torna não confiável as previsões, já que o intervalo de confiança acaba cobrindo as correlações seguintes.

O primeiro e segundo mês, podendo até enquadrar o terceiro mês também se for necessário, podem ser utilizados para testes.

Não há indícios também de sazonalidade.

Vemos também se tratar de uma série não estacionária, já que a média com o tempo não se mantém constante.



Como a FAC no lag um foi alto, então a FACP nos apresenta a variedade de auto correlações baseada no primeiro lag que tivemos, mostrando assim valores altos em outros lags como no segundo, como foi mostrado na FACP.

## Verificação de Estacionariedade

Outra forma de verificar se a série é estacionária ou não é pelo teste de **Dickey Fuller** (ADF).

Neste caso o teste de Dickey Fuller indicou que a série não é estacionária (Valor P de 100% e o valor crítico de 5% é menor que o teste estatístico)

```
X = shampoo["SALES"].values
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))
```

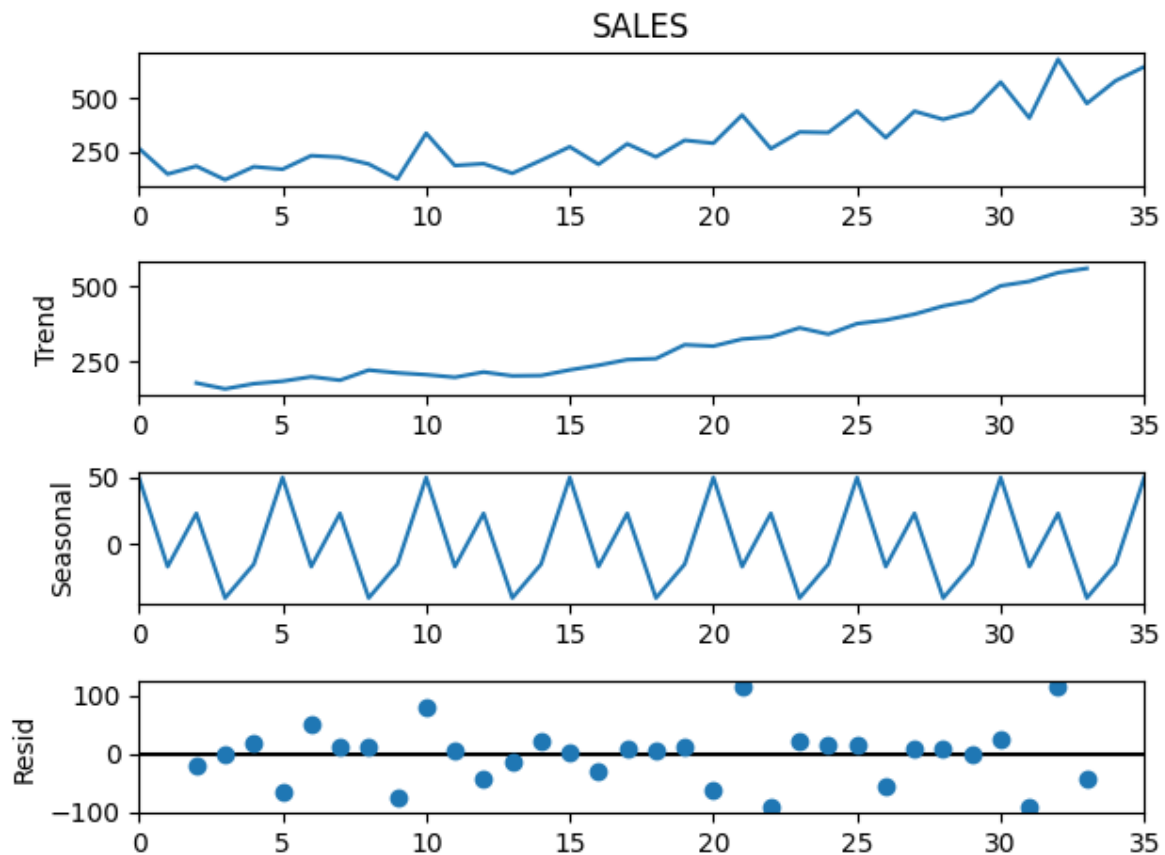
```
ADF Statistic: 3.060142
p-value: 1.000000
Critical Values:
1%: -3.724
5%: -2.986
10%: -2.633
```

## Decomposição da Sazonalidade

Realizamos também o estudo de composição sazonal do dataset, onde foi feito um estudo amplo de tendência, sazonalidade e resíduo.

Como já foi confirmado, a tendência é crescente do preço de vendas, e há uma presença de ruído em alguns pontos.

```
resultado = seasonal_decompose(shampoo["SALES"], model='additive', period=5)
resultado.plot()
```



## Estudo de Médias Móveis

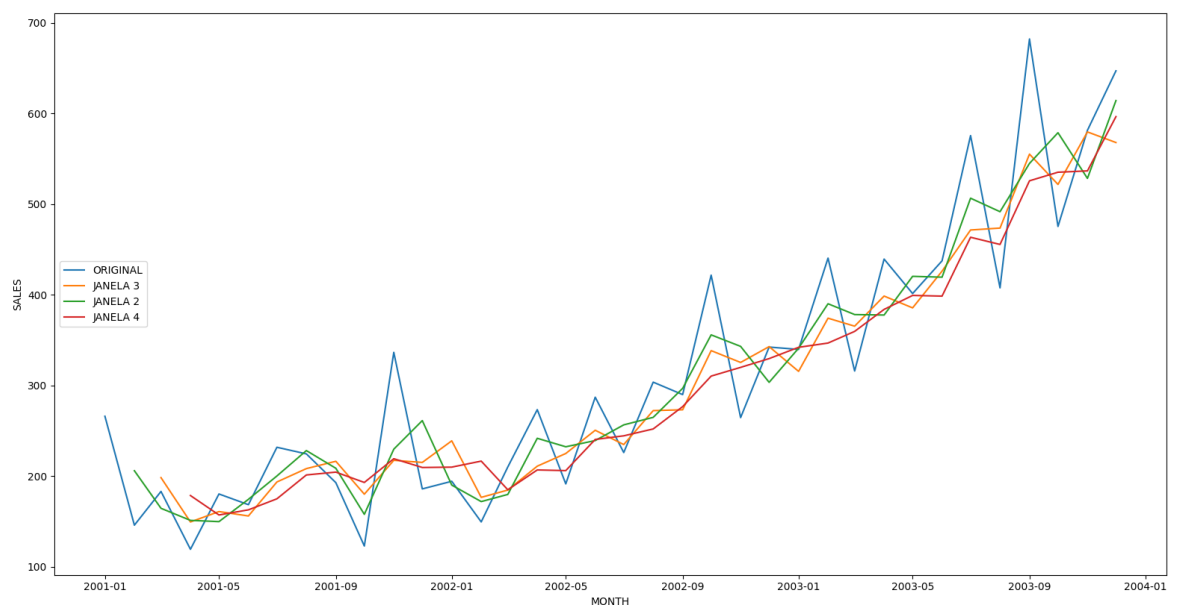
Através das médias móveis podemos obter valores para parâmetro do modelo ARIMA, além de visualizar o valor médias durante uma sequência de tempo e poder remover o ruído.

```
rolling = shampoo.rolling(window=3)
rolling_mean = rolling.mean()
shampoo["MEDIAS MOVEIS"] = rolling_mean
sns.lineplot(data=shampoo, x="MONTH", y="SALES")
sns.lineplot(data=shampoo, x="MONTH", y="MEDIAS MOVEIS")
```

Com a janela de três em três meses, podemos estar calculando a média móvel nesse período de tempo em todo o dataset, obtendo então como resultado:

```
>>> rolling_mean
SALES
0      NaN
1      NaN
2    198.333333
3    149.433333
4    160.900000
5    156.033333
6    193.533333
7    208.266667
8    216.366667
9    180.066667
10   217.400000
11   215.100000
12   238.900000
13   176.566667
14   184.633333
15   210.966667
16   224.933333
17   250.566667
18   234.800000
19   272.200000
20   273.166667
21   338.366667
22   325.333333
23   342.800000
24   315.500000
25   374.133333
26   365.333333
27   398.533333
28   385.500000
29   426.000000
30   471.400000
31   473.500000
32   555.033333
33   521.633333
34   579.533333
35   567.833333
```

Os dois primeiros meses são NaN, pois ele vai usar os dois primeiros valores mais o terceiro valor para calcular a média dos três primeiros valores. Através do gráfico abaixo analisaremos as médias móveis comparada ao dataset original:



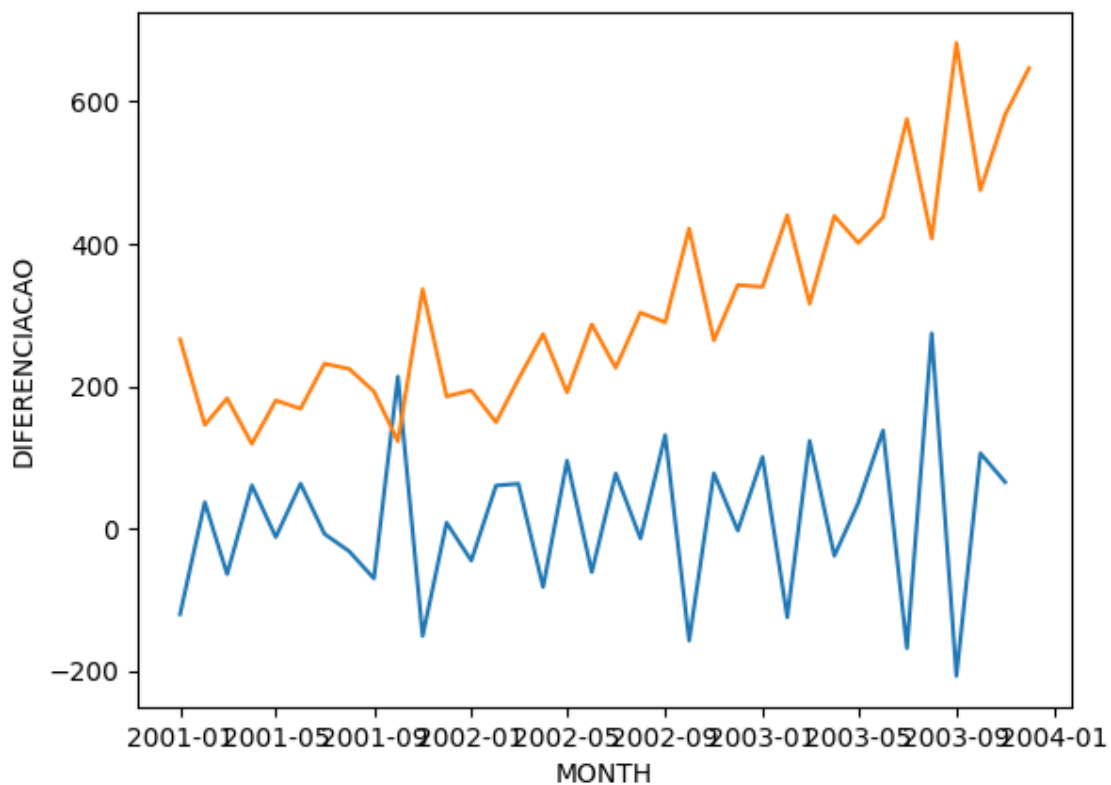
Plotamos também os valores de janela sendo dois e quatro para fazer o comparativo, e temos que a janela três obteve o melhor resultado visual.

## 5- Transformando a série para estacionária

A diferenciação é resumidamente a diferença do valor do período T com o valor do período anterior T-1. Com ela podemos diminuir tendências e variância.

Ao aplicar a função de diferenciação, podemos ver a diferença que tivemos entre os gráficos antes de diferenciar e após realizar o processo.

```
def difference(dataset, interval):  
    diff = list()  
  
    for i in range(interval, len(dataset)):  
        value = dataset[i] - dataset[i - interval]  
        diff.append(value)  
  
    return pd.Series(diff)  
  
diferenciacao = difference(shampoo["SALES"], 1)  
shampoo["DIFERENCIACAO"] = diferenciacao  
sns.lineplot(data=shampoo, x="MONTH", y="DIFERENCIACAO")
```



Podemos verificar então que temos agora uma série estacionária apenas com uma diferenciação feita.

Tendo ciência dos dados obtidos anteriormente, podemos então aplicar um tipo de modelo para a previsão dos dados futuros.

## 6- Escolha do tipo de modelo a ser aplicado



Para a escolha do melhor modelo, podemos utilizar o método Box-Jenkins, onde há a necessidade de seguir alguns critérios abaixo:

1. **Identificação:** análise do comportamento das Funções de Autocorrelação (FAC) e Autocorrelação Parcial (FACP);
2. **Estimação:** comparação dos modelos ajustados seguindo o critério da parcimônia;
3. **Checagem:** análise dos resíduos. Se os resíduos apresentarem autocorrelação, a dinâmica da série não foi completamente explicada pelos coeficientes do modelo ajustado;
4. **Previsão:** as previsões podem ser ex-ante ou ex-post. A previsão ex-ante calcula valores futuros de curto prazo e a previsão ex-post gera valores dentro do período amostral.

## Identificação

A fase de identificação pode ser visualizada abaixo com os seguintes critérios da tabela:

Modelo	Padrão FAC	Padrão FACP
AR( $p$ )	Declina exponencialmente ou com um padrão de onda senóide amortecida ou ambos	Picos significativos até $p$ defasagens
MA( $q$ )	Apresentam picos significativos até $q$ defasagens	Declina exponencialmente
ARMA( $p, q$ )	Queda exponencial	Queda exponencial

Como analisado no item 4, ambos os gráficos da PAC e da PACF tem quedas exponenciais, o que podemos ter a visão inicial de ser um modelo ARMA.

Como a análise anterior mostrou ser um gráfico não estacionário, podemos utilizar o ARIMA(1,1,1), onde podemos aplicar a diferenciação para eliminar essa não estacionaridade.

Podemos utilizar também o AUTO ARIMA, que nos apresenta já informações mais relevantes de quais dados podem ser úteis para a modelagem principal.

## Checagem

A normalidade dos resíduos pode ser analisada por meio do teste de Shapiro-Wilk, que avalia uma amostra de dados e quantifica a probabilidade de os dados terem sido extraídos de uma distribuição gaussiana (distribuição normal).

```
>>> shapiro(shampoo["SALES"])
ShapiroResult(statistic=0.9171881675720215, pvalue=0.010418307967483997)
```

```
shapiro(shampoo["SALES"])
```

Podemos ver pelo resultado que o teste estatístico deu 0.97 e o seu p-value correspondente deu 0.01. Se o p-value é menor que 0.05 nós rejeitamos a hipótese nula e temos evidências suficientes para dizer que os dados não vem de uma distribuição normal.

## Previsão

A previsão de valores através do ARIMA e AUTO ARIMA podem ser verificados na categoria seguinte com seus respectivos gráficos e amostras feitas.

## 7- Aplicação do ARIMA e AUTO ARIMA

Inicialmente utilizaremos o modelo ARIMA para poder testar como está nossa previsão.

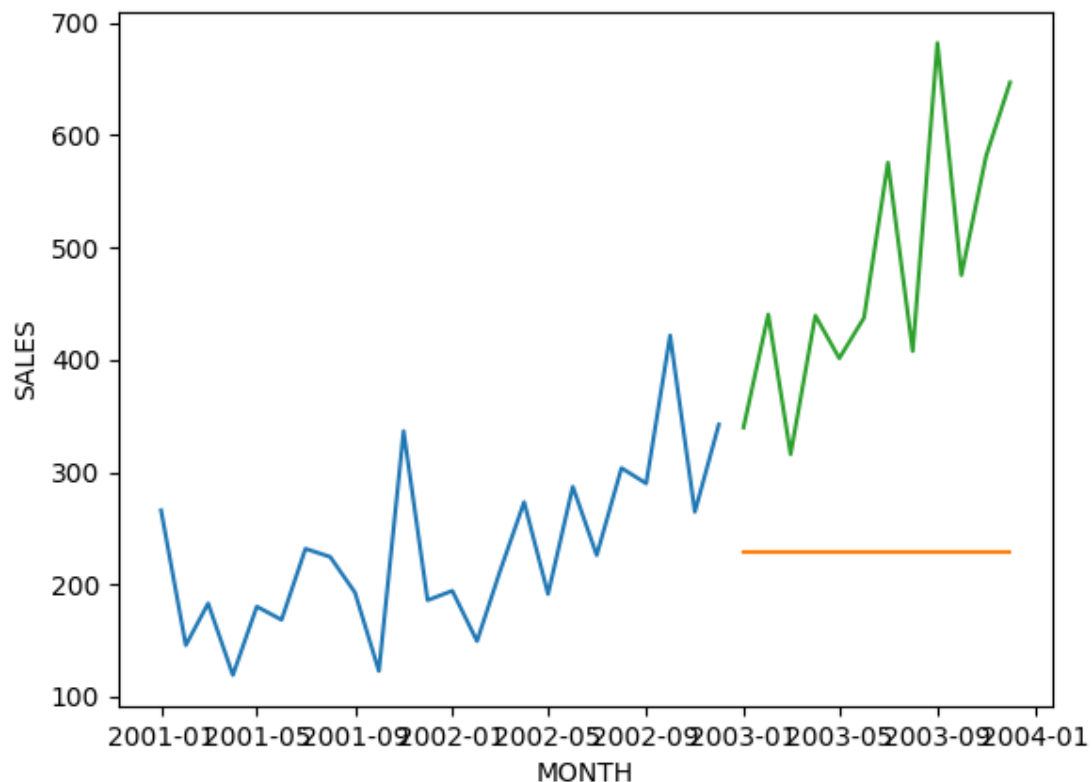
Para isso faremos a modelagem do modelo escolhido:

O dataset foi dividido em dados de treino e dados de teste, realizando a divisão por 2 anos iniciais como treino e o terceiro ano como teste.

```
shampoo_treino = shampoo[:24 ]
shampoo_teste = shampoo[24:]
```

Inicialmente aplicamos o ARIMA com os parâmetros sem valores, inicialmente como (0, 0, 0) e assim obtemos o seguinte resultado:

```
modelo = ARIMA(shampoo_treino["SALES"], order=[0, 0, 0])
```



Como vemos, a linha azul mostra a série normal do dataset com o decorrer do tempo, a linha verde é a parte que teríamos que ter previsto, porém o modelo nos retornou o resultado que aparece em forma de linha laranja.

Precisamos ajustar os valores do parâmetro de ordem para obter um melhor resultado, e com isso aplicamos o modelo ARIMA com um chute de parâmetro baseado nas evidências que encontramos.

O valor de  $p$ ,  $d$  e  $q$  respectivamente podemos chutar baseado nos seguintes aspectos:

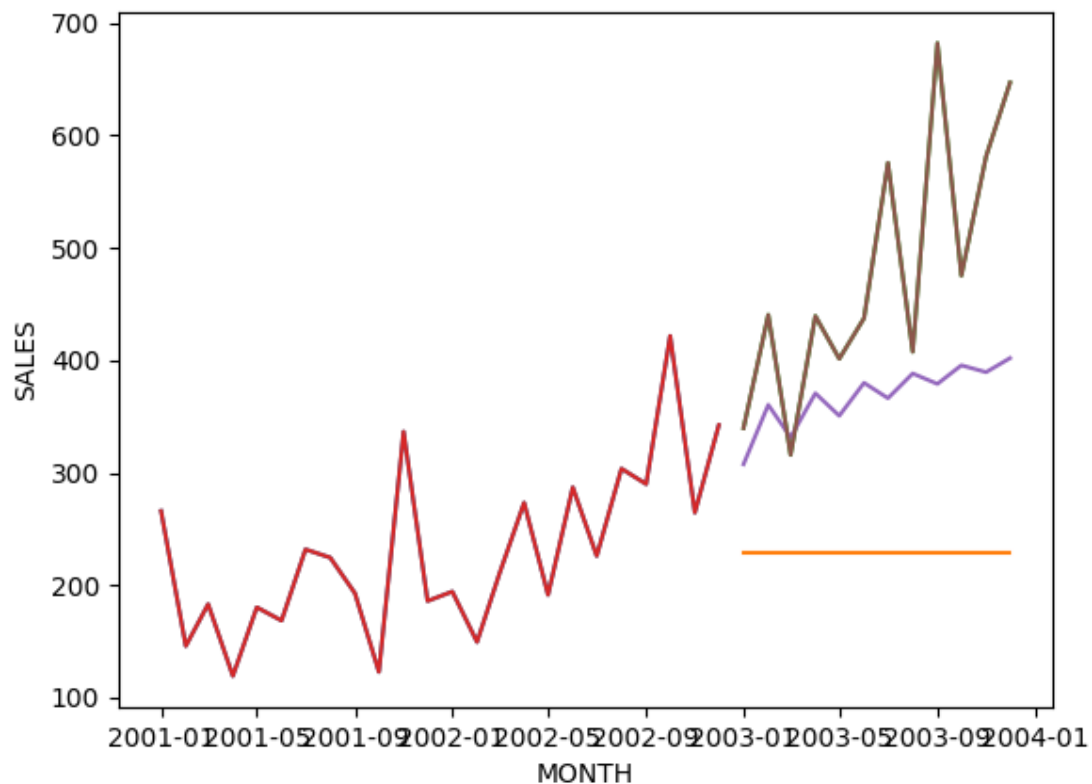
- $p$ : quantos lags estão acima do limiar no gráfico de FACP após analisar o FAC, ou seja, a ordem do modelo autoregressivo
- $d$ : o número de vezes em que os dados tiveram valores passados subtraídos, ou seja, o grau de diferenciação
- $q$ : quantos lags estão acima do limiar no gráfico de FAC após analisar o FACP, ou seja, a ordem da média móvel

Obtemos então o valor de  $p$  sendo dois, o valor de  $q$  sendo três, e o valor de  $d$  sendo um. Testando esses valores temos:

```
modelo = ARIMA(shampoo_treino["SALES"], order=[2, 1, 3])
resultado = modelo.fit().forecast(12)
shampoo_teste["PREVISAO"] = resultado
```

```
sns.lineplot(data=shampoo_treino, x="MONTH", y="SALES")
sns.lineplot(data=shampoo_teste, x="MONTH", y="PREVISAO")
sns.lineplot(data=shampoo_teste, x="MONTH", y="SALES")
```

Vemos o seguinte resultado para a previsão utilizando os parâmetros 2, 1 e 3:



A linha em roxo mostra a previsão do ARIMA, um resultado bem mais próximo do que caso não seja colocado nenhum parâmetro no modelo.

Podemos testar o modelo de AUTO ARIMA também, que nos apresenta possíveis valores de parâmetros sem qualquer análise feita e nos mostra qual seria o melhor resultado a ser utilizado.

```
modelo_auto_arima = auto_arima(shampoo_treino["SALES"].values, error_action="ignore",
                                trace=True)
```

O AUTO ARIMA detectou que os melhores valores de ordem são (1,1,0)(0,0,0)[0], onde o primeiro conjunto de parênteses representa os valores de p, d, q de ordem.

```

Performing stepwise search to minimize aic
ARIMA(2,1,2)(0,0,0)[0] intercept : AIC=inf, Time=0.23 sec
ARIMA(0,1,0)(0,0,0)[0] intercept : AIC=276.144, Time=0.01 sec
ARIMA(1,1,0)(0,0,0)[0] intercept : AIC=264.865, Time=0.04 sec
ARIMA(0,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.09 sec
ARIMA(0,1,0)(0,0,0)[0] : AIC=274.175, Time=0.01 sec
ARIMA(2,1,0)(0,0,0)[0] intercept : AIC=266.321, Time=0.09 sec
ARIMA(1,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.15 sec
ARIMA(2,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.13 sec
ARIMA(1,1,0)(0,0,0)[0] : AIC=263.115, Time=0.02 sec
ARIMA(2,1,0)(0,0,0)[0] : AIC=264.762, Time=0.03 sec
ARIMA(1,1,1)(0,0,0)[0] : AIC=263.638, Time=0.03 sec
ARIMA(0,1,1)(0,0,0)[0] : AIC=263.859, Time=0.02 sec
ARIMA(2,1,1)(0,0,0)[0] : AIC=265.433, Time=0.06 sec

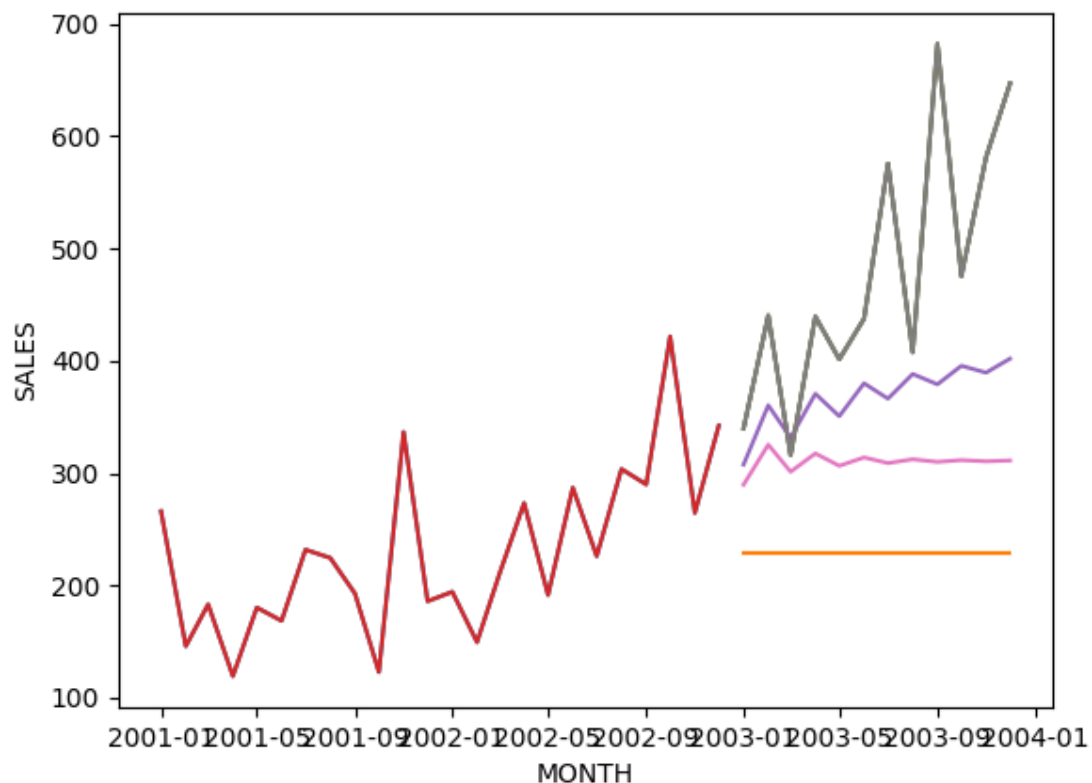
Best model: ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 0.925 seconds

```

```

previsao_auto_arima = modelo_auto_arima.predict(12)
shampoo_teste["PREVISAO_AUTO_ARIMA"] = previsao_auto_arima
sns.lineplot(data=shampoo_teste, x="MONTH", y="PREVISAO_AUTO_ARIMA")
sns.lineplot(data=shampoo_teste, x="MONTH", y="SALES")

```



Em rosa podemos ver o resultado do AUTO ARIMA, apesar de termos um bom desempenho, ele ainda fica abaixo do ARIMA convencional e dos parâmetros que decidimos pro modelo depois de todas as análises.

## 8- Metrificações

Podemos verificar qual modelo agiu melhor através do MSE, quanto menor o MSE, melhor o modelo agiu:

```
resultado_auto_arima = mean_squared_error(shampoo_teste["SALES"],
shampoo_teste["PREVISAO_AUTO_ARIMA"])
resultado_arima =
mean_squared_error(shampoo_teste["SALES"],
shampoo_teste["PREVISAO"])
```

```
AUTO ARIMA = 40544.485372020725
ARIMA = 21456.961085771847
```

```
print(f"AUTO ARIMA = {resultado_auto_arima}")
print(f"ARIMA = {resultado_arima}")
```

Como podemos verificar, o ARIMA performou melhor do que o resultado dos parâmetros do AUTO ARIMA.

## Conclusão

O estudo de séries temporais sobre o dataset de vendas de shampoo nos mostrou a variância positiva em relação ao preço das vendas. Conseguimos identificar diversos fatores para estudo, como a estacionariedade da série, estimação dos parâmetros, escolha de modelos.

O resultado obtido foi satisfatório para uma primeira análise no campo de estudo e a previsão foi a esperada para a análise feita.

# EN/en

## Introduction

Below I'll be showing how the shampoo price forecast study of the Shampoo Sales dataset that is present in Kaggle by the link below was carried out and I'll also explain the code that was made and the result that was obtained.

Kaggle Dataset: <https://www.kaggle.com/redwankarimsony/shampoo-saled-dataset>

## Time series

In statistics, a time series is a collection of observations taken sequentially over time. In linear regression models, the order of observations is irrelevant for the analysis, in time series the order of the data is fundamental.

A very important characteristic of this type of data is that neighboring observations are dependent and the interest is to analyze and model this dependence.

Time series exist in the most varied application areas, such as: finance, marketing, economics, insurance, demography, social sciences, meteorology, energy, epidemiology, etc.

In other words, a time series is a sequence of numbers collected at regular intervals over a period of time.

## Case study

The study was carried out for educational purposes, in which there was a need to learn better about time series.

Python was used as the language, the PyCharm IDE and the main ARIMA library.

The dataset is composed of only two columns, the first being the sale price of the shampoo, and the second the date related to the sale price.

Below I'm showing and explaining the code in addition to the results.

### 1- Importing libraries

I used the following libraries to carry out the study:

```
import warnings
import pandas as pd
import csv
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import TimeSeriesSplit
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import adfuller

```

## 2- Importing the file

There is only one csv file in the competition called "shampoo\_sales.csv"

```
d_shampoo = "C:/Users/mcelopes/Desktop/shampoo_sales.csv"
```

```
shampoo = pd.read_csv(d_shampoo, sep=";", encoding="UTF-8", dtype=str,
error_bad_lines=False, quoting=csv.QUOTE_NONE)
```

## 3- Cleaning and Treatment

A function for handling columns was created, the function `handle_column` receives the dataset as a parameter and maintains the upper case pattern in the name and content of each column, in addition to removing spaces.

```

def treat_column(data):
    data.columns = [x.upper() for x in data.columns]

    for column in data:
        data[column] = data[column].str.strip()
        data[column] = data[column].str.replace("'", "")
        data[column] = data[column].str.upper()

```

After the function is called, we continue with other treatments such as renaming the columns that have an extra "", converting the SALES column type to numeric, and also converting the MONTH column to the datetime type.

```
treat_column(shampoo)
```

```
shampoo = shampoo.rename({"'MONTH'": "MONTH", "'SALES'": "SALES"}, axis=1)
```

```
shampoo["SALES"] = pd.to_numeric(shampoo["SALES"])
```

```
shampoo["MONTH"] = "0" + shampoo["MONTH"]
```



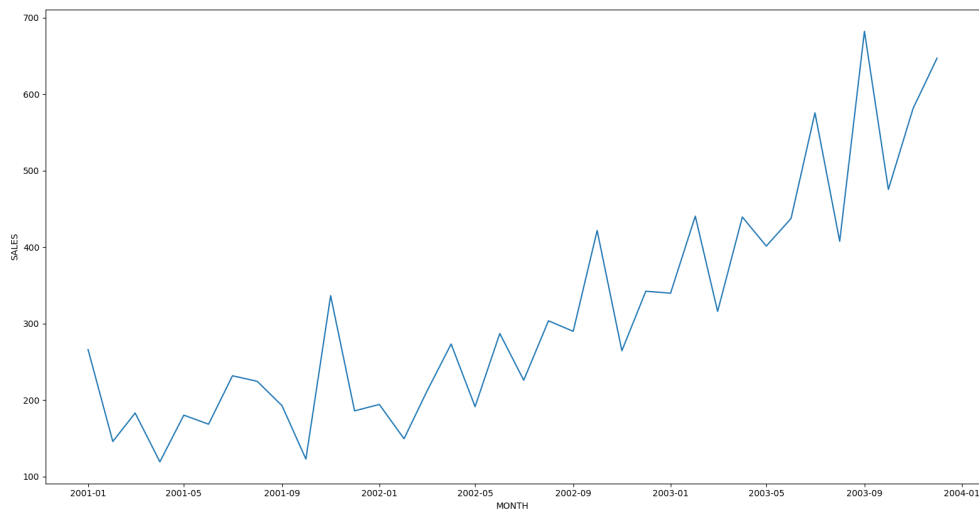
```
shampoo["MONTH"] = pd.to_datetime(shampoo["MONTH"], format="%y-%m",  
yearfirst=True)
```

## 4- Current base analysis

### Overview

The dataset analysis was then started, first we have an overview of the dataset.

```
sns.lineplot(data=shampoo, x="MONTH", y="SALES")
```

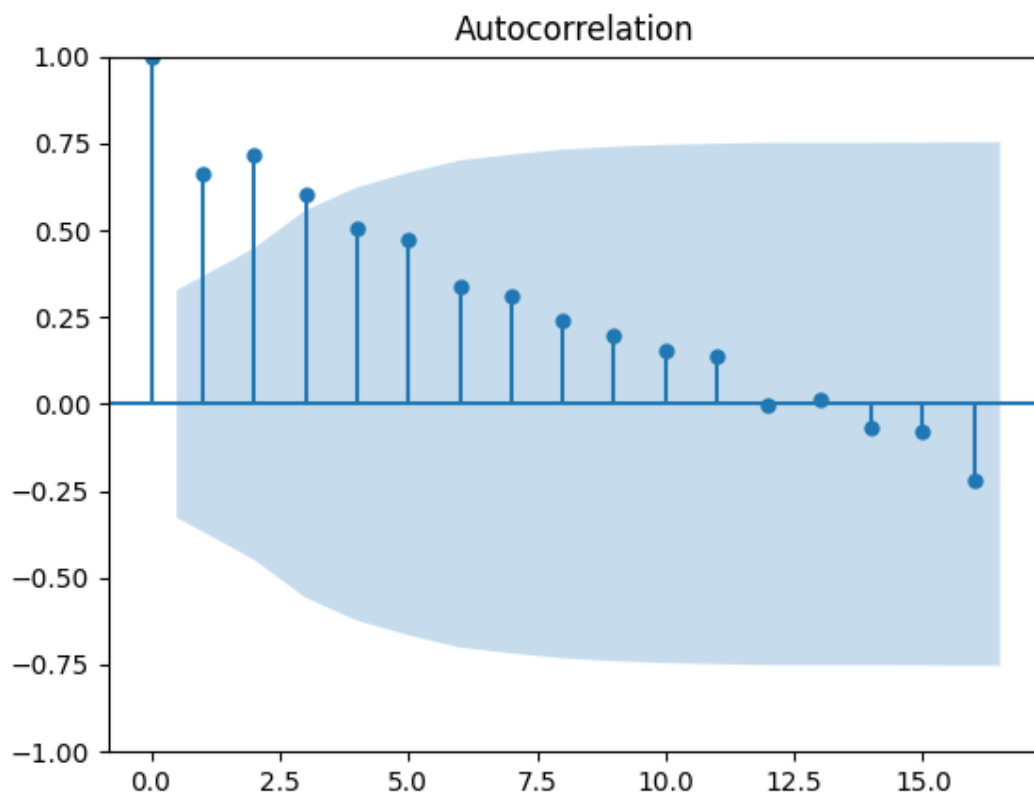


We can see that over time, the price of shampoo tends to increase, with some small variations.

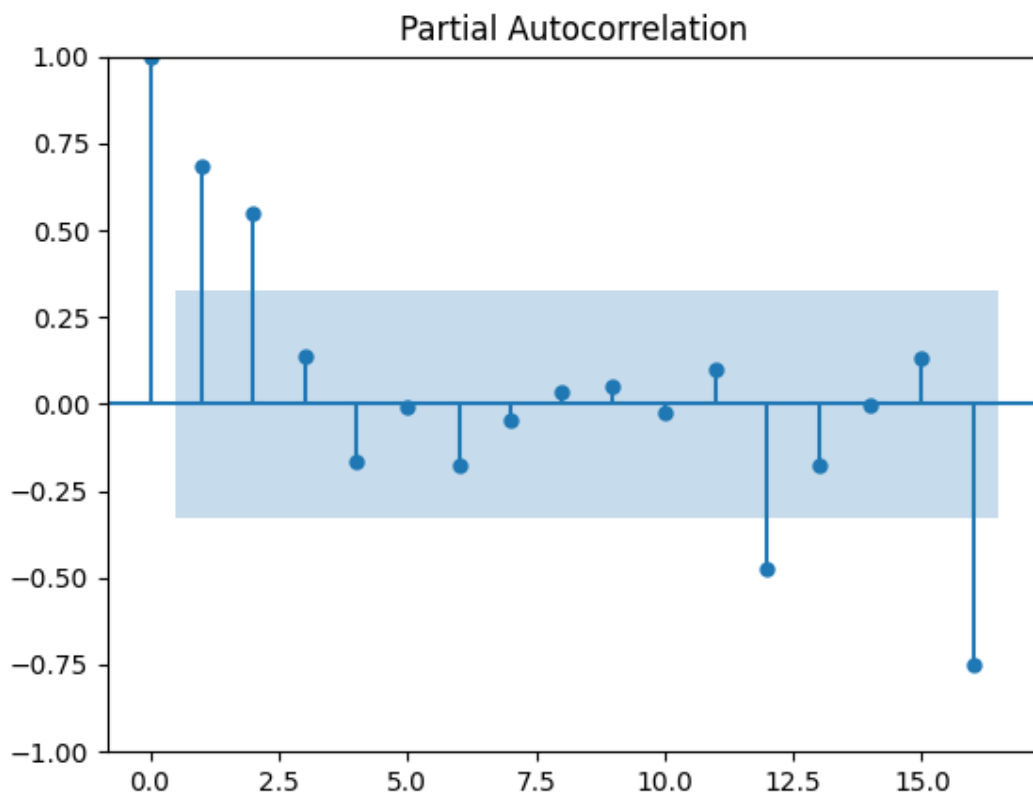
### Autocorrelation (FAC) and Partial Autocorrelation (FACP)

We also analyzed the FAC and the FACP of the data.

```
plot_acf(shampoo["SALES"].values.astype("float32"))
```



As shown in the FAC graph, after 2.5 years the predictions become unreliable, as the confidence interval ends up covering the following correlations. The first and second months, which may even include the third month if necessary, can be used for tests. There are also no signs of seasonality. We also see that it is a non-stationary series, as the average over time does not remain constant.



As the FAC in lag one was high, then the FACP presents us with the variety of autocorrelations based on the first lag we had, thus showing high values in other lags like the second, as shown in the FACP.

## Stationery Verification

Another way to check whether the series is stationary or not is the Dickey Fuller (ADF) test. In this case Dickey Fuller's test indicated that the series is not stationary (100% P value and 5% critical value is lower than the statistical test)

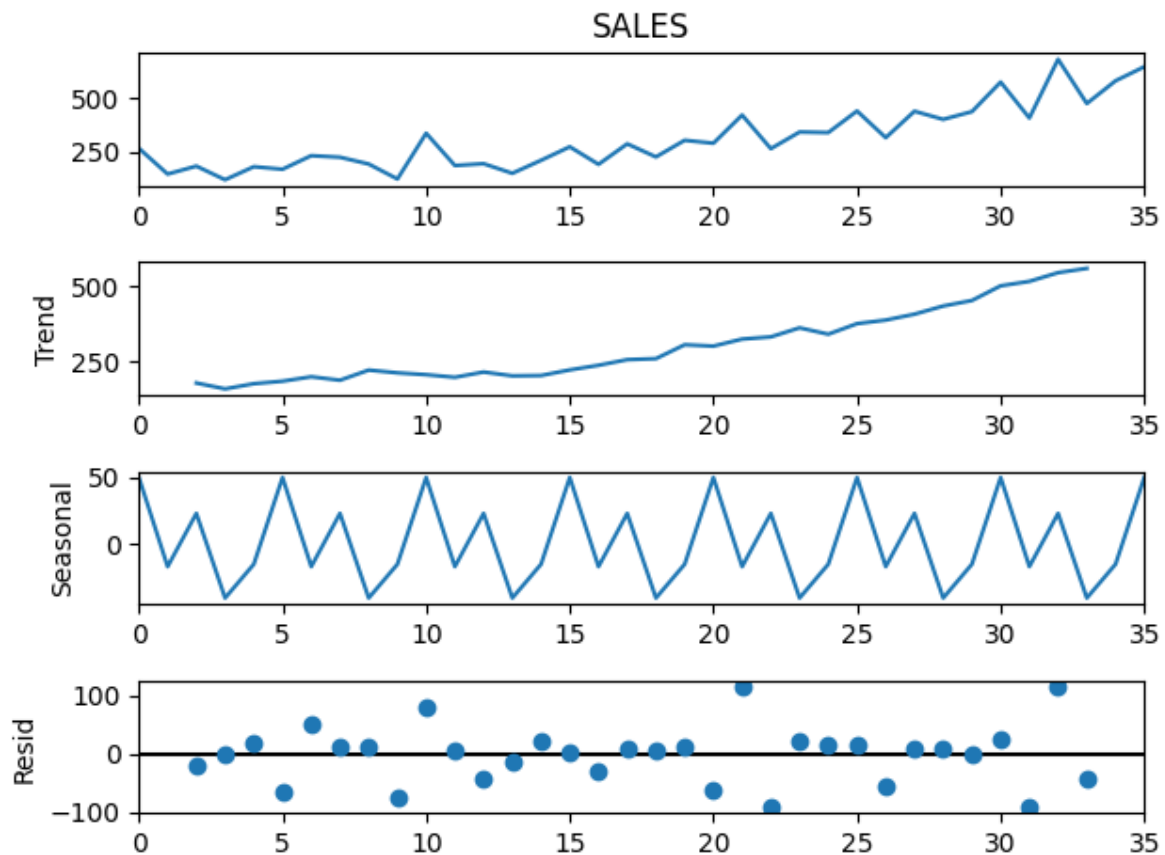
```
X = shampoo["SALES"].values
result = addler(X)
print('ADF Statistic: %f % result[0])
print('p-value: %f % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print("\t%s: %.3f % (key, value))
```

```
ADF Statistic: 3.060142
p-value: 1.000000
Critical Values:
  1%: -3.724
  5%: -2.986
 10%: -2.633
```

## Seasonality Decomposition

We also carried out the study of the seasonal composition of the dataset, where a broad study of trend, seasonality and residual was carried out.  
As already confirmed, the trend is increasing in sales prices, and there is a presence of noise in some points.

```
result = seasonal_decompose(shampoo["SALES"], model='additive', period=5)
result.plot()
```



## Study of Moving Averages

Through the moving averages we can obtain values for parameter of the ARIMA model, in addition to visualizing the average value during a time sequence and being able to remove the noise.

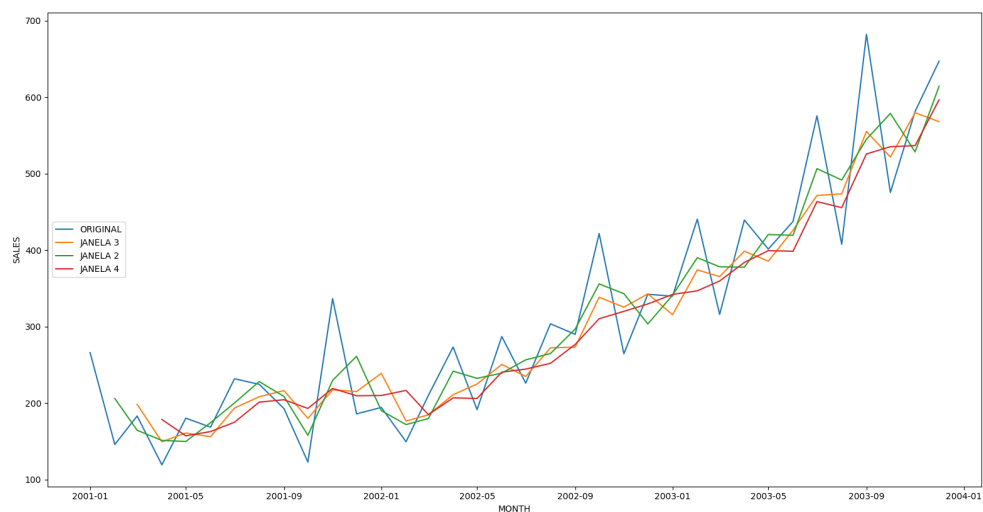
```
rolling = shampoo.rolling(window=3)
rolling_mean = rolling.mean()
shampoo["MOVABLE MEDIA"] = rolling_mean
sns.lineplot(data=shampoo, x="MONTH", y="SALES")
sns.lineplot(data=shampoo, x="MONTH", y="MOVING AVERAGES")
```

With the window of every three months, we can be calculating the moving average in this period of time across the dataset, thus obtaining the result:

```
>>> rolling_mean
      SALES
0      NaN
1      NaN
2    198.333333
3    149.433333
4    160.900000
5    156.033333
6    193.533333
7    208.266667
8    216.366667
9    180.066667
10   217.400000
11   215.100000
12   238.900000
13   176.566667
14   184.633333
15   210.966667
16   224.933333
17   250.566667
18   234.800000
19   272.200000
20   273.166667
21   338.366667
22   325.333333
23   342.800000
24   315.500000
25   374.133333
26   365.333333
27   398.533333
28   385.500000
29   426.000000
30   471.400000
31   473.500000
32   555.033333
33   521.633333
34   579.533333
35   567.833333
```

The first two months are NaN as it will use the first two values plus the third value to average the first three values.

Through the chart below, we will analyze the moving averages compared to the original dataset:



We also plotted the window values being two and four to make the comparison, and we have that window three got the best visual result.

## 5- Transforming the series to stationary

Differentiation is in short the difference of the value of period  $T$  with the value of the previous period  $T-1$ . With it we can reduce trends and variance.

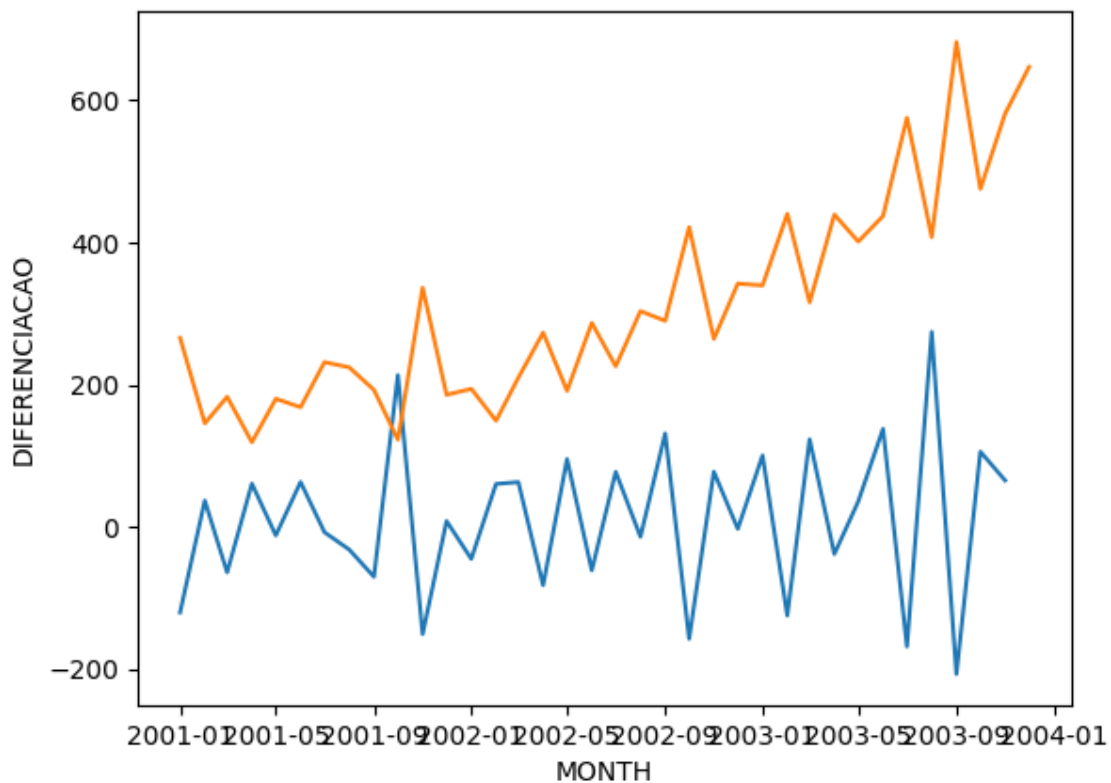
By applying the differencing function, we can see the difference we had between the graphs before differencing and after performing the process.

```
def difference(dataset, interval):  
    diff = list()
```

```
    for i in range(interval, len(dataset)):  
        value = dataset[i] - dataset[i - interval]  
        diff.append(value)
```

```
    return pd.Series(diff)
```

```
differentiation = difference(shampoo["SALES"], 1)  
shampoo["DIFFERENTIATION"] = differentiation  
sns.lineplot(data=shampoo, x="MONTH", y="DIFFERENTIATION")
```



We can then verify that we now have a stationary series with only one differentiation made.

Being aware of the data obtained previously, we can then apply a kind of model to predict the future data.

## 6- Choosing the type of template to be applied

To choose the best model, we can use the Box-Jenkins method, where it is necessary to follow some criteria below:

1. **Identification:** analysis of the behavior of the Autocorrelation Functions (FAC) and Partial Autocorrelation (FACP);
2. **Estimation:** comparison of adjusted models following the criterion of parsimony;
3. **Checking:** analysis of residues. If the residues show autocorrelation, the series dynamics was not fully explained by the adjusted model coefficients;
4. **Prediction:** Predictions can be ex-ante or ex-post. Ex-ante forecasting calculates short-term future values and ex-post forecasting generates values within the sample period.

## Identification

As analyzed in item 4, both PAC and PACF graphs have exponential drops, which we can have the initial view of being an ARMA model.

As the previous analysis showed to be a non-stationary graph, we can use ARIMA(1,1,1), where we can apply the differentiation to eliminate this non-stationarity.

We can also use AUTO ARIMA, which already presents us with more relevant information about which data can be useful for the main modeling.

## Check

The normality of the residuals can be analyzed using the Shapiro-Wilk test, which evaluates a sample of data and quantifies the probability that the data have been extracted from a Gaussian distribution (normal distribution).

```
>>> shapiro(shampoo["SALES"])  
ShapiroResult(statistic=0.9171881675720215, pvalue=0.010418307967483997)
```

```
shapiro(shampoo["SALES"])
```

We can see from the result that the statistical test gave 0.97 and its corresponding p-value gave 0.01. If the p-value is less than 0.05 we reject the null hypothesis and have enough evidence to say that the data does not come from a normal distribution.

## Forecast

The forecast values through ARIMA and AUTO ARIMA can be verified in the following category with their respective graphs and samples made.

## 7- Application of ARIMA and AUTO ARIMA

Initially, we will use the ARIMA model in order to test our forecast.

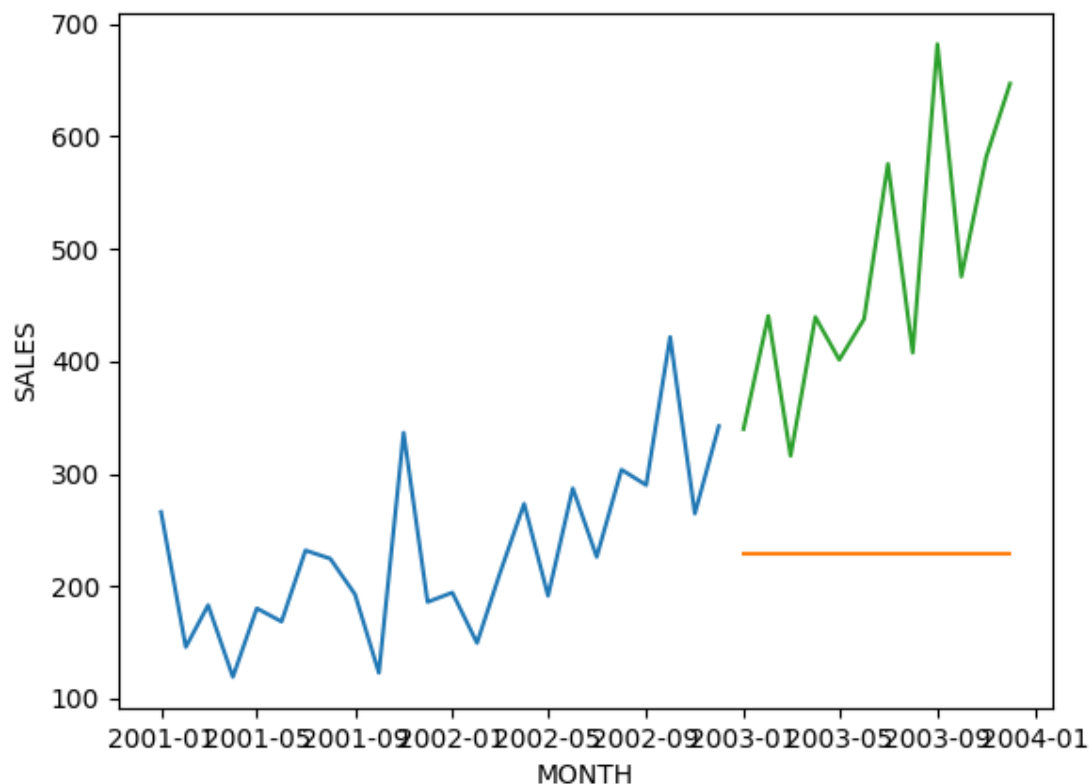
For this, we will model the chosen model:

The dataset was divided into training data and testing data, dividing by the initial 2 years as training and the third year as testing.

```
shampoo_training = shampoo[:24]
shampoo_test = shampoo[24:]
```

Initially we apply ARIMA with the parameters without values, initially as (0, 0, 0) and thus obtain the following result:

```
model = ARIMA(shampoo_traino["SALES"], order=[0, 0, 0])
```





As we can see, the blue line shows the normal series of the dataset over time, the green line is the part that we would have had to predict, but the model returned the result that appears as an orange line.

We need to adjust the order parameter values to get a better result, so we apply the ARIMA model with a parameter guess based on the evidence we found.

The value of p, d and q respectively can be guessed based on the following aspects:

- p: how many lags are above the threshold in the FACP graph after analyzing the FAC, that is, the order of the autoregressive model
- d: the number of times the data had past values subtracted, that is, the degree of differentiation
- q: how many lags are above the threshold on the FAC graph after analyzing the FACP, that is, the moving average order

We then get the value of p being two, the value of q being three, and the value of d being one.

Testing these values we have:

```
model = ARIMA(shampoo_treino["SALES"], order=[2, 1, 3])
```

```
result = model.fit().forecast(12)
```

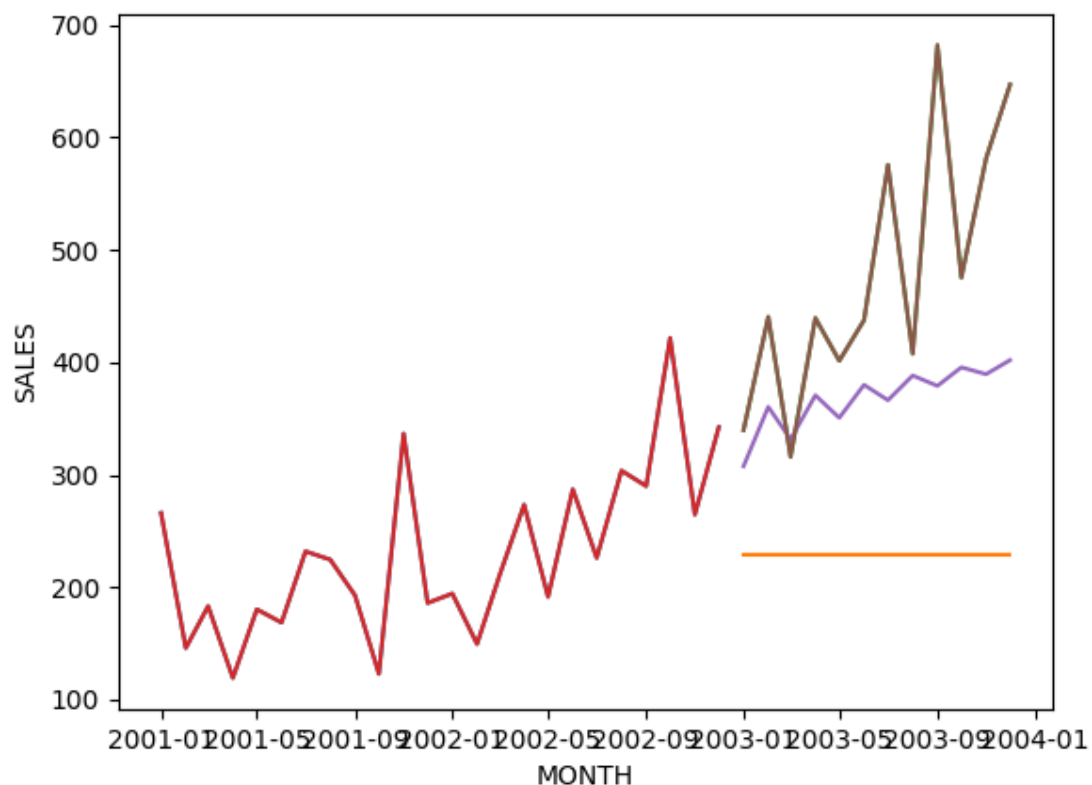
```
shampoo_test["FORCEPTION"] = result
```

```
sns.lineplot(data=shampoo_training, x="MONTH", y="SALES")
```

```
sns.lineplot(data=shampoo_test, x="MONTH", y="FORECAST")
```

```
sns.lineplot(data=shampoo_test, x="MONTH", y="SALES")
```

We see the following result for the prediction using parameters 2, 1 and 3:



The line in purple shows ARIMA's forecast, a result much closer than if no parameter is placed in the model.

We can test the AUTO ARIMA model too, which shows us possible parameter values without any analysis done and shows us what would be the best result to be used.

```
model_auto_arima = auto_arima(shampoo_training["SALES"].values, error_action="ignore", trace=True)
```

AUTO ARIMA detected that the best order values are (1,1,0)(0,0,0)[0], where the first set of parentheses represents the order p, d, q values.

```

Performing stepwise search to minimize aic
ARIMA(2,1,2)(0,0,0)[0] intercept : AIC=inf, Time=0.23 sec
ARIMA(0,1,0)(0,0,0)[0] intercept : AIC=276.144, Time=0.01 sec
ARIMA(1,1,0)(0,0,0)[0] intercept : AIC=264.865, Time=0.04 sec
ARIMA(0,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.09 sec
ARIMA(0,1,0)(0,0,0)[0] : AIC=274.175, Time=0.01 sec
ARIMA(2,1,0)(0,0,0)[0] intercept : AIC=266.321, Time=0.09 sec
ARIMA(1,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.15 sec
ARIMA(2,1,1)(0,0,0)[0] intercept : AIC=inf, Time=0.13 sec
ARIMA(1,1,0)(0,0,0)[0] : AIC=263.115, Time=0.02 sec
ARIMA(2,1,0)(0,0,0)[0] : AIC=264.762, Time=0.03 sec
ARIMA(1,1,1)(0,0,0)[0] : AIC=263.638, Time=0.03 sec
ARIMA(0,1,1)(0,0,0)[0] : AIC=263.859, Time=0.02 sec
ARIMA(2,1,1)(0,0,0)[0] : AIC=265.433, Time=0.06 sec

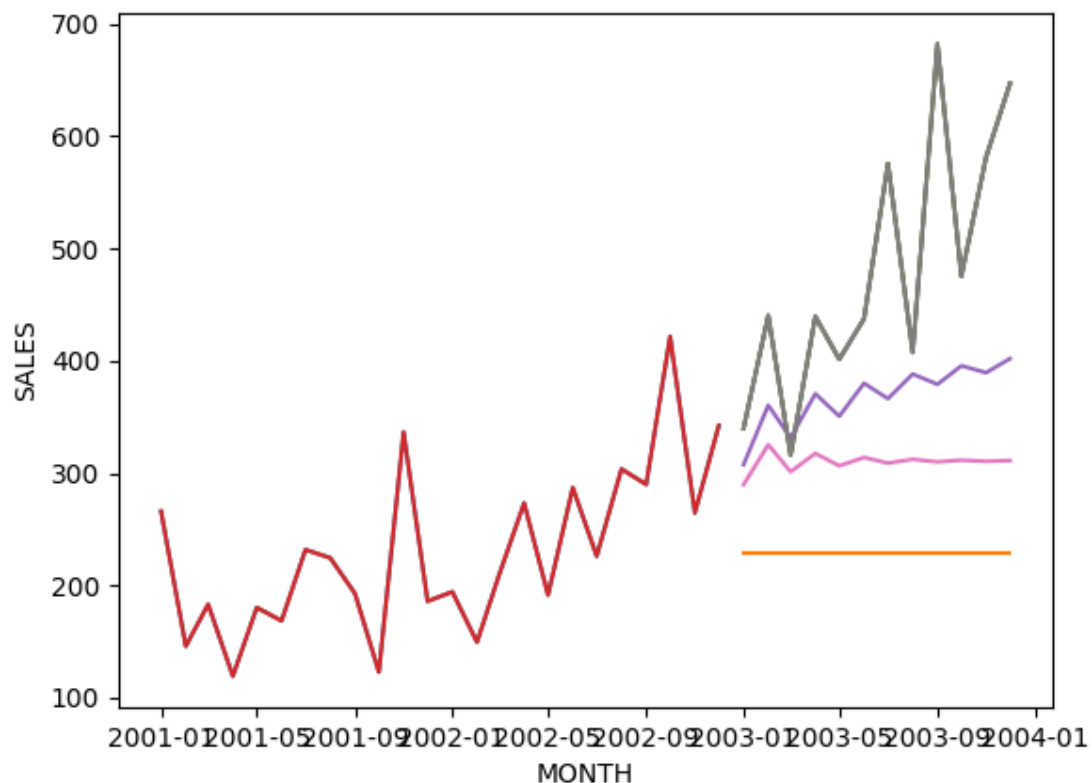
Best model: ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 0.925 seconds

```

```

previsao_auto_arima = modelo_auto_arima.predict(12)
shampoo_teste["PREVISAO_AUTO_ARIMA"] = previsao_auto_arima
sns.lineplot(data=shampoo_teste, x="MONTH", y="PREVISAO_AUTO_ARIMA")
sns.lineplot(data=shampoo_teste, x="MONTH", y="SALES")

```



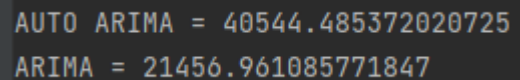
In pink we can see the result of AUTO ARIMA, despite having a good performance, it is still below the conventional ARIMA and the parameters we decided for the model after all the analyses.

## 8- Metrics

We can check which model acted better through the MSE, the smaller the MSE, the better the model acted:

```
result_auto_arima = mean_squared_error(shampoo_test["SALES"],
shampoo_test["PREVISAO_AUTO_ARIMA"])
arima_result = mean_squared_error(shampoo_test["SALES"],
shampoo_test["FORCEPTION"])
```

```
print(f"AUTO ARIMA = {result_auto_arima}")
print(f"ARIMA = {arima_result}")
```



```
AUTO ARIMA = 40544.485372020725
ARIMA = 21456.961085771847
```

As we can see, ARIMA performed better than the result of the AUTO ARIMA parameters.

## Conclusion

The time series study on the shampoo sales dataset showed us the positive variance in relation to the sales price. We were able to identify several factors for study, such as the stationarity of the series, parameter estimation, choice of models.

The result obtained was satisfactory for a first analysis in the field of study and the forecast was expected for the analysis carried out.