A Work Project, presented as part of the requirements for the Award of a Master's degree in Business Analytics from the Nova School of Business and Economics.

Predicting demand using Autoregressive integrated moving average (ARIMA) – Application on the Iberian market for clams

In cooperation with Oceano Fresco

Nicolas Moissiadis

Work project carried out under the supervision of:

Professor Pedro M. Gardete

13/12/2022

Abstract

As the accessibility of data has grown, data-driven pricing and demand forecasting have become more widespread. However, early-stage companies are often faced with a great deal of unpredictability due to a lack of existing data before entering the market. Therefore, a publicly available database has been consulted to assess the need for clams and related elements to support the entrance of Oceano Fresco, a sustainable seafood start-up, into the Iberian Peninsula market. As a result, ARIMA has been adopted to anticipate a sales index for every month. This index can then be utilized to estimate the number of clams Oceano Fresco is capable of selling the following month based on actual demand factors, a historical equation model, and the previous year's sales.

Keywords

Clams, Demand Forecasting, ARIMA, Hyperparameter Tuning, Statistical Forecasting, Python

This work used infrastructure and resources funded by Fundação para a Ciência e a Tecnologia (UID/ECO/00124/2013, UID/ECO/00124/2019 and Social Sciences DataLab, Project 22209), POR Lisboa (LISBOA-01-0145-FEDER-007722 and Social Sciences DataLab, Project 22209) and POR Norte (Social Sciences DataLab, Project 22209).

Table of contents

1 Introduction 6
2 Background 7
2.1 The business of clam farming 7
2.2 The Oceano Fresco Approach 13
3 Factors affecting the sales price for clams in Portugal 15
3.1 Historical baseline and indices 15
3.1.1 Data foundation 15
3.1.2 Index 17
3.2 Demand Factors 19
3.2.1 Theory of demand 20
3.2.2 Portuguese demand 20
3.2.3 Off-days 21
3.2.4 School Holidays 22
3.2.5 Weather 24
4 Deliverable 2: Forecasting sales for coming month as a trend 25
4.1 Prediction and forecasting methods 25
4.1.1 Ordinary Least Squares (OLS) 26
5 Autoregressive integrated moving average (ARIMA) 30
5.1 Data Pre-processing 31
5.2 ARIMA Prediction 31
5.3 Result 34
5.4 Limitation 36
5.5 Most suitable model 36
6 Outlook 42
6.1 Project Deliverables 42
6.1.1 Automation of Data Sources 42
6.1.2 Data Collection by Oceano Fresco 43
6.2 Data analysis 45
6.2.1 Forecasting methodology 45
6.2.2 Dynamic forecasting models 46
6.2.3 Forecasting of Harvesting Restrictions 47
7 Conclusion 48
8 Citations 51
9 Appendices 54
9.1 Tables 54
9.2 Python Code 55

List of tables

Table 1 Iberian Clam Market 9

Table 2: Results of Arima split validation 35

Table 3: Explanation of terms 40

Table 4: Results of ARIMA prediction 41

Table 5: ARIMA Equations 41

Table 6: Data frame 55

Table of figures

Figure 1 Value Chain B2C Clams 12

Figure 2 Average sales quantity per month, 2014-21, indexed (average=100) 18

Figure 3 Average sales price per month, 2014-21, indexed (average=100) 19

Figure 4 Average number of off-days per month, 2014-21 21

Figure 5 Average number of school holidays per month, 2014-21 22

Figure 6 Average temperature per month, °C, 2014-21 24

Figure 7 OLS regression results 27

Figure 8 Influence Plot of observations in OLS regression 29

Figure 9 Seasonal Arima formula (Prasad, 2021) 30

Figure 10 Fitting the ARIMA model 32

Figure 11 Overview, SARIMAX results 37

Figure 12 ARIMA prediction performance and absolute errors 39

Figure 13 ARIMA prediction performance 40

1 Introduction

In recent years, data-driven pricing and demand or production modelling gained popularity and importance with increasing amounts of data available to everyone. Especially early-stage start-ups encounter high uncertainty with limited internal and no historical data available before market entry. Therefore, precision in modelling and forecasting are crucial but often relies on external and limited data.

Sales managers need to be well informed to make the best decisions for the company. In the wholesale of fresh food, prices change daily and depend on many factors. Sales managers need to be aware of the latest trends and developments that influence market prices, supply and demand, to make smart decisions when negotiating with clients and building new partnerships.

Oceano Fresco is a Portuguese sustainable food tech start-up operating on the bivalves market with a planned start of sales in January 2023. With no internal sales data, limited annual production capacity and limited shelf life of stock, accurate production and sales planning are crucial to maximizing annual profits in a niche market that faces great seasonality and limited availability of market or industry data.

The goal of this project is to provide two managerial dashboards to Oceano Fresco. The first dashboard aims to inform sales managers about the latest market trends, and changes in demand and supply factors, in Portugal. The second dashboard is a monthly production planning tool. This tool is based on an ARIMA model to forecast demand, and a Newsvendor model to maximize annual profit. Both dashboards are built in Excel and the best practices of dashboard design have been reviewed and incorporated.

In the following, the clam industry and the company Oceano Fresco are presented to provide the reader with the necessary background information. Then, the scope and limitations of this project are defined, and the company's guiding comments are discussed. Relevant concepts and

literature for the project are presented. Subsequently, the clam market with the driving demand and supply factors is analyzed. The first deliverable, designed to inform sales managers about the latest trends and developments of the relevant demand and supply factors, is presented. Next, the second deliverable, the statistical model to forecast demand and the profit maximization model, are discussed. To end, an outlook that should help the company or the following project group to improve the dashboards by automation and an extended data-driven approach to improve forecasting accuracy is discussed.

2 Background

In this chapter, clams and the business of clam farming are described, and the company Oceano Fresco is introduced. The information is based on our research and information given by the company through meetings and internal documents.

2.1 The business of clam farming

Clams

According to the Encyclopaedia Britannica, the term clam refers to any member of the invertebrate class Bivalvia - mollusks with a bivalved shell. In a stricter sense, the term clam refers to marine bivalve molluscs with shells of equal size. (Britannica 2022)

The European clam production along the Atlantic coast is dominated by the native European clam (Ruditapes decussatus), the native Palourde (Venerupis corrugata), and the invasive Manila clam (Ruditapes philippinarum). (European Parliament 2016)

Clams live in freshwater and marine environments with salty water usually buried down in sandy or muddy bottoms from close to the surface to up to 0.6 meters with a preference for shallow waters to be protected from waves and movements by the surrounding bottom

(Britannica 2022). The growth and survival of clams depend on water and weather conditions such as water salinity and temperature (Oceano Fresco 2022).

Clam Market and Consumption

In Europe, bivalve molluscs consumption has ancient origins and is part of typical diets and popular food in the Iberian Peninsula for holiday and celebratory activities. In Portugal, clams are primarily sold in local markets and consumed fresh in restaurants and seafood festivals. (Oliveira et al. 2013)

According to estimates by Oceano Fresco based on FAO Eurostat data from 2017, Iberia accounts for 44% of the total European clam consumption of 138.5t and 48% of the premium clam consumption of 24.3%. Oceano Fresco estimates the European clam market with 2 billion Euros in B2C and 1 billion in B2B. In this context, B2C refers to selling adult clams for consumption to distributors or end-customers, retail, markets, restaurants, hotels, or other food service outlets. Clams can be bought fresh, canned, or frozen. B2B refers to the sale of semi-adult clams to producers (Oceano Fresco 2022).

On the Iberian consumer market, premium clams are represented by the native European clams Ruditapes decussatus and Venerupis corrugata with wholesale prices in Portugal from around 12 to 20 Euros and from 25 to 40 Euros for end-customers like private customers or restaurants, according to Oceano Fresco (2022). In the mid-price segment, the invasive Ruditapes philippinarum can be found with wholesale prices of 5 to 10 Euros. The lowest price tier consists of imported clams, dominated by the species Meretrix Lyrata, with wholesale prices from 1 to 2 Euros. The Ibearian clam market is visualized in Table 1. The species' binomial name is followed by their name in English, Portuguese, Spanish, and Galician.

Segment	Type	Species
Premium	Native	**Ruditapes decussatus** (European clam, Amêijoa-boa, Almeja fina, Ameixa fina)
		Venerupis corrugata (Palourde, Amêijoa-macha, Almeja babosa, Ameixa babosa)
Mid	Invasive	**Ruditapes philippinarum** (Manila clam, Amêijoa-japonesa, Almeja japonesa, Ameixa xaponesa)
Low	Imported	**Meretrix Lyrata** (Asiatic hard clam, Amêijoa-vietnamita, Almeja del Pacífico, Ameixa do Indo-Pacífico)

Table 1 Iberian Clam Market

Other factors than species or supply and demand that influence pricing are the size of clams, with a price increase with size increase, and their quality, determined by the ratio of meat to shell. According to Oceano Fresco (2022), higher prices can be found in Spain compared to Portugal.

Clams are low in fat, high in protein meat, and rich in marine lipids and minerals. Clams are a low trophic species, so to say, at the bottom of the animal chain, meaning they do not consume other animals. Clams per se are a sustainable form of protein with a no or low pollution output, a favorable CO2 footprint and a reduction of eutrophication in the water. There are no or low artificial inputs to clams, meaning they do not get fed and no antibiotics are given. With an

increasing world population and thus, an increased need for protein plus the increased sense of sustainable responsibility and health, the future demand is expected to grow significantly. Today's consumption of clams in Iberia with a decreasing European production volume leads to a supply shortage and growing imports of low-tier species from Asia to Portugal and Spain. (Smaal et al. 2019)

Clam Production

Clams are harvested worldwide along the coast. Asia, especially China, is the leading producer of marine bivalves with 85% of the world's production with a continuous increase in quantity. In Europe, clam production has been declining in the last decades due to various reasons such as competing clams on space, high mortality rates, and overfishing which increases the importance of sustainable clam farming. (Smaal et al. 2019)

According to Oceano Fresco, the central production hub on Iberia of Venerupis corrugata is Galicia. The central production hub of Ruditapes decussatus is in the Algarve. Since the illegal introduction of the Manila clam to Europe in the 1970s for commercial reasons, its production volume surpasses the volume of the native European clam and Palourde due to its robustness and growth capacity (Coelho et al. 2021). Looking at the Iberian Atlantic coast, the Manila clam cultivation is prohibited in Portugal but can be found everywhere. The grey market of Manila clams is big, the annual catch volume is estimated to be around 4,000 to 17,000 tons. (Coelho et al. 2021) Cultivation is legal in Galicia, Spain, and the volume of the Manila clam has been declining only in recent years. (Pescadegalicia, 2022).

Harvesting

The harvesting of bivalves is an activity of social, economic and cultural importance in Portugal. In 2011, clams represented 39.73% of the Portuguese national capture of bivalve molluscs and an annual average of 93.89% of total bivalve production with commercialization nearly exclusively in the national market. (Oliveira et al. 2013)

Clam farming involves seeding juveniles. Farmers may get their seeds from hatcheries, natural beds or their own parks which are protected bottoms, plots, maintained in tidal flats. Clams can be harvested in sandy or muddy bottoms in littoral areas, in the sea or rivers as the Tagus, usually from small boats with 1-2 fishers on board with dredges with sieves, or by foot on intertidal areas with bottom trawls or hand shovels and rakes (Oliveira et al. 2013, FAO 2022). By European law, clams need to have a specific size to be harvested to protect the stock and prevent overfishing. Clams usually need two years from brood stock to the size to be harvested.

Harvesting activities in Portuguese littoral areas can be limited by particular weather and marine conditions e.g., rainfall or wind or waves. Additionally, biotoxins harvest closure by the government prohibits the harvesting of bivalve molluscs species.

Several barriers lack the development of harvesting and production of shellfish in Portugal such as lack of transfer of technology, training of fishers and farmers, the competitiveness of prices, and the organization of the sectors (Oliveira et al. 2013). Clam Farmers are usually individuals or small businesses. According to Oceano Fresco, there is no big player in the production and harvesting of clams.

Value chain

Figure 1 Value Chain B2C Clams

As described in the previous section, farmers harvest adult clams from natural beds' seeds bought from hatcheries or their parks. The farmers typically sell the adult clams to depurators, as the clams need to be purified. This is mandatory depending on the water quality but is typically needed for all littoral areas. The entry barrier of capital needed to purify clams hinders the small farmers from purifying themself. The clams only survive a specific time in the water tanks, a couple of weeks, which limits the possibility of increasing stock for high seasons. From the substantial water tanks from the depurators, the clams are sold to distributors or directly to the end customers. The depurators control the bargaining power. Clams are being sold alive and have a shelf life of around three days. Some depurators are vertically integrated with distributions or the production of clam seeds. End-customers are retail, markets or food service outlets e.g. hotels, or restaurants. This is the value chain for B2C, whereas B2B is the sale of seeds, semi-adult to producers which then place them in their production area and let them grow for around another year. Clams of this age are priced at only one-fifth of adult clams.

The described value chain, plus the previously discussed barriers that lack the development of harvesting and production, lead to a downward spiral. Overfishing leads to supply shortages and lets import volumes grow. Fast-growing low-quality invasive species threaten native species with high mortality rates (Oceano Fresco 2022).

2.2 The Oceano Fresco Approach

The Company

Ocean Fresco is a sustainable seafood company that started as a Portuguese food technology and biotech startup in the bivalves market founded in 2015. The company has 23 employees (July 2022). During the first seven years, the company invested in Research and Development within innovative selective breeding and cultivation processes with an expected market entry in January 2023. The company operates a Biomarine center, including a hatchery and an operations center, in Nazaré on the west coast of Portugal, an open-sea farm off the Atlantic coast of Lagos, situated in the Algarve on the south coast of Portugal and a small research hub at NOVA School of Business and Economics in Carcavelos, close to Lisbon, Portugal.

The company's mission is to disrupt the bivalve market on a European and global scale by offering consumers sustainable, tasty, healthy food with innovative aquaculture and the first open-sea clam farm worldwide. Their vision is regenerative farming, which means not taking from nature by only farming clams that were bred in their hatchery. The company’s short-term strategy is to enter the Iberian clam market in 2023 with the offer of high-quality native European clam in January for B2C.

Oceano Fresco in the value chain

Oceano Fresco is vertically integrated into the value chain. In their hatchery, they control all steps from brood stock, larvae, to seeds, and phytoplankton. After around six months in the hatchery, the clams spent 1 to 1.5 years on the open-sea farm. Then, some are directly sold at the harbor, but most are shipped back to the operations center with livestock and packing capabilities in Nazaré. By law, the clams of Oceano Fresco do not need to be purified, thanks to the excellent water quality in the open sea farm, compared to the water at the littoral where

this is a mandatory step. Therefore, there is no legal minimum time for the clams to stay here. The clams can survive a few weeks in the livestock. Only when an order needs to be shipped, the clams are taken out of the live stocking water tanks, packed and shipped alive with a shelf life of a few days (Oceano Fresco 2022).

Competitive Advantage

The vertical integration and the open-sea farm offer an all-year-round supply with almost no production seasonality or downtimes and high volumes of premium native European clams compared to traditional clam farmers. Oceano Fresco can respond fast to demand and seasonality with their livestock and without the need for depuration. Traditional farmers cannot turn around a high volume fast when demand increases which leads to supply shortages and the clams need to be depurated, which delays the response to increases in demand. Thanks to the research conducted and the advantages of the open-sea farm, the clams of Oceano Fresco are high in quality with a high meat ratio and tender meat. Additionally, their product specs are trustworthy as they control the majority of factors that influence the clam quality. When strategic partnerships have been built, Oceano Fresco is technically able to scale up faster production in both the hatchery and the open-sea farm (Oceano Fresco 2022).

Product offer and pricing strategy

Oceano Fresco plans to start with the B2C sale of adult clams for consumption with packages dedicated to retail and to out-of-home consumption in Iberia. While OF offers native European clam of high quality which can be found in the premium segment, the company plans to sell to a price above the mid-tier Manila clam but below the market prices for the premium segment, with higher prices for Spain than Portugal. While Oceano Fresco would like to build strong

relationships with their customers with a stable all-year-round supply, the company estimates to sell around 80% of the volume during high seasons when other suppliers cannot deliver high volumes in a short turnaround time which leads to supply shortages (Oceano Fresco 2022). Oceano Fresco estimates the high season to be during the Iberian summer holiday and in December, especially around Christmas and New Year holidays. As the sales price of seeds is low, Oceano Fresco does not plan to sell significant volumes via B2B.

3 Factors affecting the sales price for clams in Portugal

3.1 Historical baseline and indices

This paragraph focuses on the historical examination of a data set that can be used as a proxy for the Portuguese clam market. Based on this, a recorded performance for sales_quantity and average_price_per_kg will be calculated. The historical performance will allow insightful statements about the entire market's future development and the month-to-month fluctuations.

3.1.1 Data foundation

All the calculations are based on a publicly available database by Pescadegalicia. The Fishing Technological Platform is a project launched in 2003 by the Galician fisheries administration to make tools available to the fishing sector of Galicia, Spain, to facilitate the realization of their management. By doing so, the administration wanted to create a system of information collection to offer the statistics of fishery products sales and other information of interest to the sector itself. The platform is supported by more than 60 self-service terminals (Ticpesc terminals) found mainly in different ports. Professionals of the sector can electronically complete most of the procedures associated with their work (such as changing fishing gear, transporting documents and other activities) using the MOVTIC mobile application, which is

available 24 hours a day, 365 days a year (Pescadegalicia, 2022). In this context, the commercial transactions that can be tracked by the software and thus provide information on the extent to which various factors affect demand was mainly considered. Furthermore, due to a lack of alternative substantiated data sources and in consultation with Oceano Fresco, the described dataset was found worthy to act as a proxy for the Portuguese market.

With the goal of creating an up-to-date dataset, the Pescadegalicia variables were constrained to the years 2014 through 2021. Subsequently, the Portuguese school vacations from 2014-2021 were incorporated into the dataset. These dates were retrieved from publicholidays.pt, a website specifically for holiday information. Additionally, the existing data frame was supplemented with historical Portuguese holidays from the Holidays library in Python, which facilitates the quick and convenient identification of holidays. Lastly, to evaluate the weather factor empirically, the latitude and longitude were determined for each of the sales markets. To accomplish this task, a Python library termed "Geopy" is essential, which links the labels of the sales markets to their existing coordinates. Subsequently, these coordinates are fed into the weather API provided by Meteostat. In this step, the API maps the coordinates with a distance algorithm to the nearest weather station and thus ensures a detailed overview of the weather data in the entire region. Finally, the dataset was aggregated monthly to allow a more granular view of the available data. In this way, an average was taken for the temperature data, the sum was taken for the quantities sold in kilograms, the average was taken for the prices per kilogram sold, the number of days off, and the number of school vacations was added up. In addition, only the following Bivalves species are considered: 'Ameixa babosa,' 'Ameixa fina' and 'Ameixa xaponesa' as these are, by agreement with Oceano Fresco, the only relevant species for the company.

The completed dataset is used as a foundation for a closer examination of the three demand factors proposed by Oceano Fresco: "Off-days in a month," "Weather," and "School vacations"

and to portray a historical trend of the overall clam market about the *Galician Clams* market. In the further course, the application of indexes as a trend progression will be examined in more detail.

3.1.2 Index

In economics, an index is a numerical measure of the fluctuation of a collection of individual data points. These data points can be measured in quantity, price, or volume. An index can measure changes in the overall economy, a particular sector, or a specific market (Smriti Chand, n.d.). In the following, the historical sales data of Galicia are analyzed for a particular pattern to retrieve possible trends related to the factors: 1. sales quantity and 2. sales price.

Sales quantity

In order to closely monitor the sales quantity factor, the data from each year is aggregated and pulled across months by a specific characteristic. This factor provides standardization and minimizes the influence of the number of days in a month.

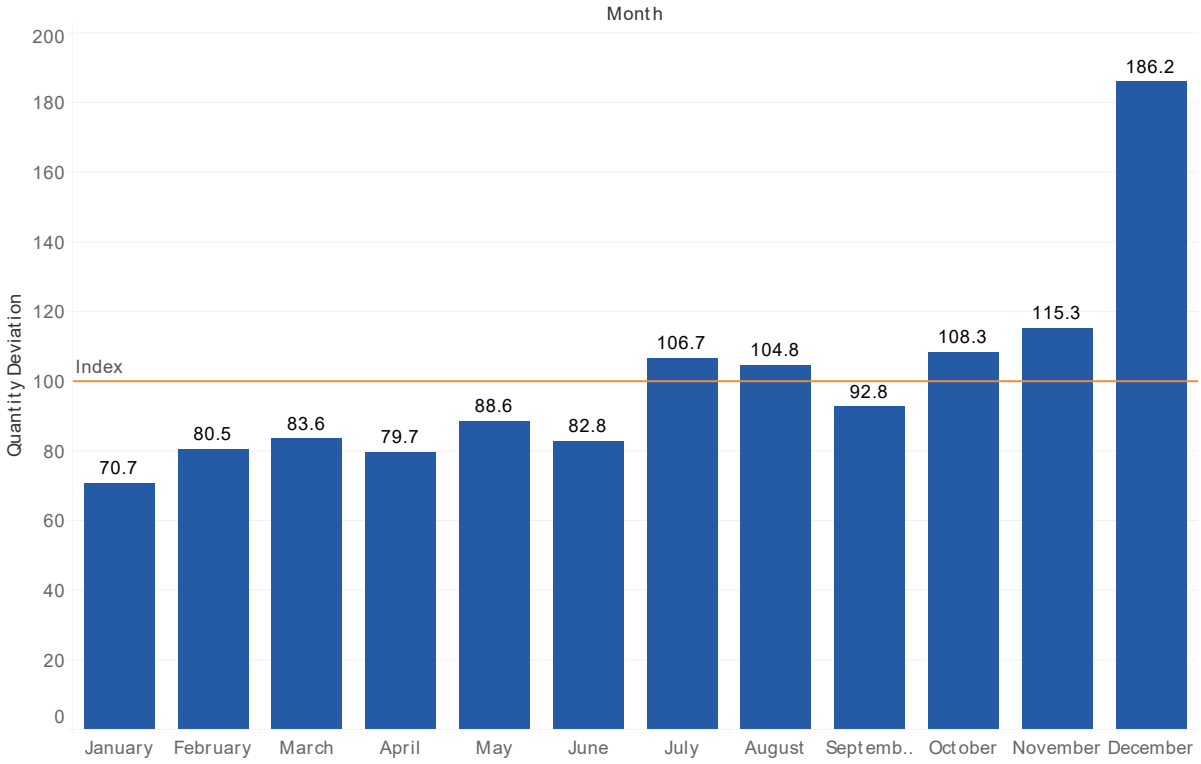


Figure 2 Average sales quantity per month, 2014-21, indexed (average=100)

The graph illustrates that in the first half of the year, sales quantity is below average, which has been computed by summing up the sales amount for each year and splitting it up into each month, comparing it to the sales quantity in that specific month. Based on these results, it can be observed in detail that the second half of the year has shown a higher sales volume in the last few years.

Sales price per kg

In a similar process, the average sales prices per kilogram are assessed. The maximum average monthly price is used as an index guide to be identifying a pattern in the data.

Figure 3 Average sales price per month, 2014-21, indexed (average=100)

The graph indicates the peak pricing occurred in December, consequently, it is a useful marker for predicting future prices. Furthermore, May to September demonstrate higher than usual prices, while January to April demonstrate much lower prices. These results are congruent with Oceano Fresco's expert opinion and are critical components in assessing fluctuations in prices on a monthly basis.

3.2 Demand Factors

The following section analyzes the demand for bivalve products. Thereby, the general theory of demand is considered and applied to the Portuguese market. Furthermore, the demand factors proposed by Oceano Fresco are evaluated in terms of their relevance by performing a T-test of

each determinant against a proxy data set of Galician seafood sellers. In this way, the variables are analyzed from a statistical point of view.

3.2.1 Theory of demand

According to the "theory of demand", demand "… is a multivariate relationship, that is, it is determined by many factors simultaneously." (Koutsoyiannis, A.,1979). By this statement, it is assumed that demand implies various determinants that can profoundly impact the latter. On the one hand, factors such as the price of the company's product and related products have a significant role. On the other hand, customer-specific attributes such as income, wealth status, and consumer behaviour determine demand. Because according to the theory of consumer behaviour, the customer is assumed to be rational and, based on the axiom of utility maximization, plans to spend his entire income to attain the highest possible satisfaction or utility (Koutsoyiannis, A.,1979).

3.2.2 Portuguese demand

Portugal has had the highest per capita consumption for fishery and aquaculture products for several years. In 2019, they had 59.9kg per capita consumption, exceeding the second place by more than 14kg. (EUMOFA, 20222) The integration of bivalves into Portuguese culture can be attributed to several determinants, such as tradition, dietary reasons, availability, and price. Due to this increasingly important role of bivalves in the Portuguese food culture, it is becoming a seafood type that is gaining more and more attention among tourists visiting Portugal. (Fonseca et al., 2006)

3.2.3 Off-days

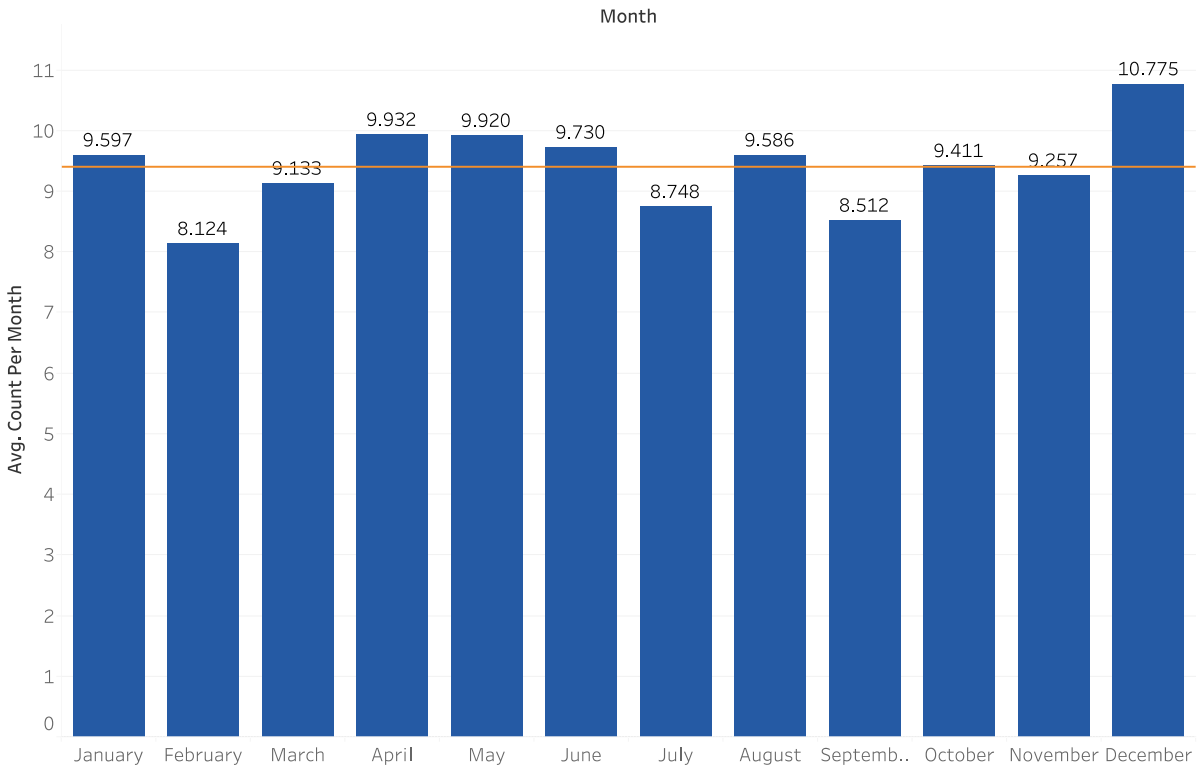


Figure 4 Average number of off-days per month, 2014-21

In the following analysis, the hypothesis of whether more days off in a month implies an increased number of clams (in kilograms) turned over is intended to be confirmed. First, a correlation is accomplished using the ".corr" method, which is part of the Python environment. It calculates the Pearson correlation coefficient between two columns of a data frame, measuring the linear relationship between two data sets. Based on the compiled data set, a correlation of 0.303 exists between the columns "quantity in kilograms" and "count_of_off_days_per_month," indicating a weak positive relationship between these variables. Consequently, as soon as the variable "quantity in kilograms" increases, the variable "count_of_off_days_per_month" also tends to increase. The obtained P-Value was significantly lower than the 0.05 threshold, being 0.0026. Thus, the null hypothesis is denied, and it is statistically established that there is a connection between the two factors. Therefore, the more days in a month, the higher the probability that the Galician distributors will sell higher amounts

of clams, implying a mutual dependence between these two factors that leads to the seasonal consumption of clams.

3.2.4 School Holidays

The variable school vacation is composed of the following school vacations: "Christmas holidays", "shrove / carnival holidays", "Springtime holidays", and "Summer holidays". School vacations imply a time when people spend considerable time with their families, including occasionally going on vacation and "indulging" themselves.

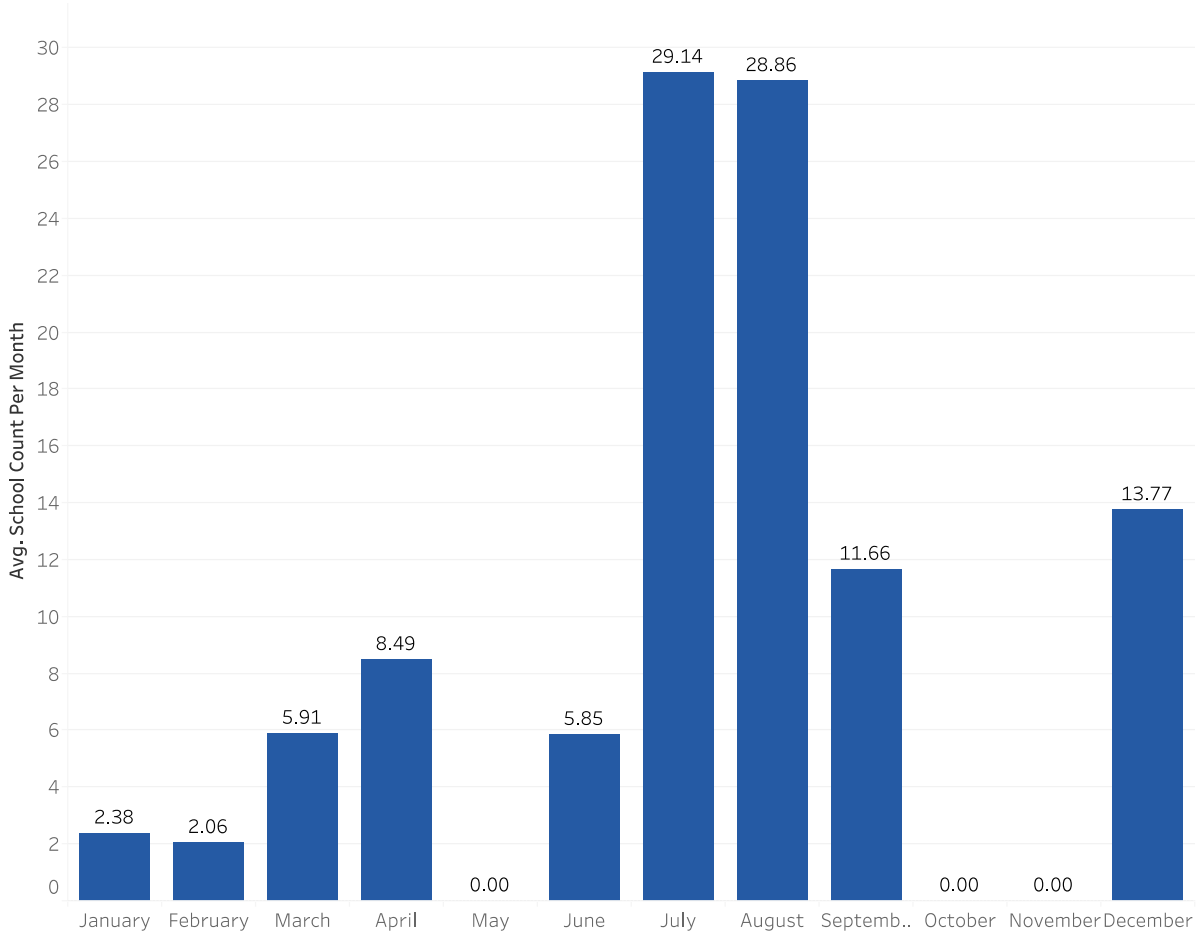


Figure 5 Average number of school holidays per month, 2014-21

The diagram depicts an overview of the school holiday's historical distribution in Portugal beginning from 2014 until 2021. From these numbers, it is apparent that the majority of school vacations coincide with the summer months.

The mentioned techniques determine the correlation between the number of school vacations in a month and the number of clams sold. The results of this calculation were as follows: A correlation of 0.203 exists between the columns "quantity in kilograms" and "school_count_per_month," indicating a weak positive relationship between these variables. Consequently, as soon as the variable "quantity in kilograms" increases, the variable "school_count_per_month" also tends to increase. In order to confer adequate validation to this initial result, the T-test described above is applied. As a result of this analysis, the P-Value is 0.05, which is equal to the designated threshold of 0.05. Thus, the null hypothesis is rejected, which is statistical proof for the correlation of these two variables.

In conclusion, the results from the second analysis of school vacations confirm the initial hypothesis of the festive consumption behavior of clams. Furthermore, based on the statistical analysis, a clear pattern that clams are not a meal for in between, but must be associated with a special occasion, can be recognized.

3.2.5 Weather

It is a known fact that Portugal has a warm, Mediterranean climate. However, it is also apparent that clams' sales are affected by weather changes. Clams have a long history of being associated with celebratory occasions, symbolizing a special holiday or a public holiday. In order to gain insight into the influence of temperature on the amount of clams sold, the average temperature from 2014-2021 for each month was studied and evaluated.

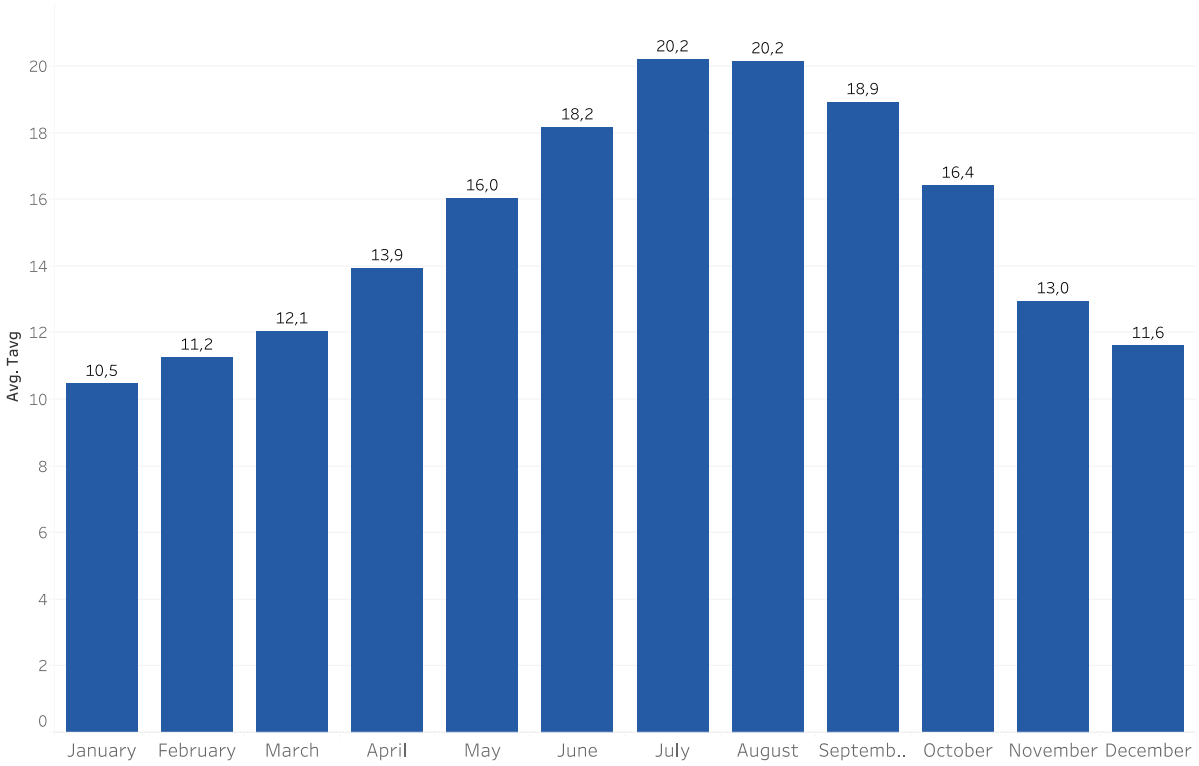


Figure 6 Average temperature per month, °C, 2014-21

As a final step of the statistical analysis of the Galician dataset, the correlation between the average monthly temperature of the years 2014-2021 with the number of clams sold is tested. Even if the weather significantly impacts clams' harvesting, as Oceano Fresco investigated, the correlation yields a negative result of -0.102. Although this result coincides with the overall survey of the number of clams sold, which peaks especially in December, according to Oceano Fresco, the primary farming season is from April to September. Therefore, it is implied by the calculated negative correlation that a higher value for the variable quantity of clams sold indicates a lower temperature.

Consequently, a comprehensive application of the T-test is employed to authenticate the existing hypothesis. In accordance with the outcome of this exploration, the hypothesis is not refused as the p-value is 0.3208, implying no relationship between these two factors after the statistical re-examination. There could be numerous explanations for why there is no connection between the weather and the amount. One potential justification could be that the type of weather does not influence the quantity of the product being purchased.

It is now proven that the above demand factors have a scientific as well as statistical relevance to the outcome of our thesis. From these results, it can be concluded that the first impression of Oceano Fresco has an impact. However, the extent to which these variables ultimately play a role in their revenue must be examined in more detail in further analyses and models.

4 Deliverable 2: Forecasting sales for coming month as a trend

The second demand from Oceano Fresco was a tool that can help management and sales representatives to optimize production and sales based on statistical forecasting. The tool should be easy to navigate, even without a data science background, yet include sophisticated statistical forecasting and prediction methods.

4.1 Prediction and forecasting methods

As mentioned in the previous chapter, an ARIMA model is used in the second deliverable. This is the result of different models fitted, compared, and fine-tuned. This process is presented in the following and the results are discussed.

4.1.1 Ordinary Least Squares (OLS)

Tool

Statsmodels is a Python library for statistical and econometric analysis and provides an Ordinary Least-Squares linear regression class that estimates a model using appropriate linear algebra on a data set (Seabold and Perktold 2010). The class returns a result table with three parts. In the following, we discuss the relevant metrics of the first two parts.

The first part of the results states data about our model. Besides general information it shows important metrics about our model. The R-squared and Adjusted R-squared represent the variance of the dependent variable explained by the regression model. The R-squared is the coefficient of determination and ranges from 0 to 1 with higher values meaning more observed variation can be explained by the model's inputs. (Lewis-Beck 1980)

The adjusted R-square is the coefficient of multiple determination and considers the number of input variables and helps to analyze multiple dependent variables' efficacy on the model (Lewis-Beck 1980).

The column coefficients give the least squares estimates of the parameters θ_n for the formula of the best-fitted linear line for each dependent variable x and the constant θ_0. The coefficient of our dependent variable x is the model's estimate of the changing size of one unit of change in the dependent variable to the change in the independent variable. The coefficient can be positive or negative.

The standard error is an estimate of the standard deviation of the coefficient. It shows the precision of the coefficient with smaller values in relation to the coefficient meaning a higher precision. The t value is obtained by a t-test and measures the size of the difference relative to

the variation in our sample data. It is a measure expressed in units of standard errors. The t-value is used to obtain the p-value. The p-value expresses the significance level of the results obtained. Most commonly, statistical significance is obtained at a significance level of 5% or less. The columns [0.025 and 0.975] represent our confidence interval at 95%.

Application

In Python, a model was fit using the statsmodels.regression.linear_model.OLS and the monthly aggregated data set based on the Galician fishery data with the dependent variables 'off_days_per_month', 'school_count_per_month', and 'tavg', and the independent variable 'Quantity sold in kg', described in the above section. A constant was added, and the default non-robust covariance type is used. The results displayed below were returned.

OLS Regression Results

Dep. Variable:	Quantity sold in KG	R-squared:	0.165
Model:	OLS	Adj. R-squared:	0.138
Method:	Least Squares	F-statistic:	6.074
Date:	Sun, 27 Nov 2022	Prob (F-statistic):	0.000809
Time:	10:18:21	Log-Likelihood:	-457.58
No. Observations:	96	AIC:	923.2
Df Residuals:	92	BIC:	933.4
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	42.6389	31.841	1.339	0.184	-20.599	105.877
off_days_per_month	7.7142	2.842	2.714	0.008	2.070	13.358
school_count_per_month	1.0045	0.332	3.030	0.003	0.346	1.663
tavg	-1.6009	0.971	-1.649	0.103	-3.529	0.327

Omnibus:	17.723	Durbin-Watson:	1.729
Prob(Omnibus):	0.000	Jarque-Bera (JB):	22.029
Skew:	0.946	Prob(JB):	1.65e-05
Kurtosis:	4.388	Cond. No.	230.

Figure 7 OLS regression results

Our linear model has an R-squared of 0.165. It can be interpreted as only 16.5% of the independent variable 'Quantity sold in kg' is explained by changes in our 3 dependent variables.

The Adjusted R-squared of 0.138 is lower than the R-squared. The model's R-squared might increase if one or more variables are eliminated. The Log-Likelihood of -457.58 is used later to compare the model to another model.

The model estimates an intercept of 42.6389 with a standard error of 31.841 and a p-value of 0.184. Therefore, the coefficient estimate for the intercept is not significant. The coefficient of off_days_per_month is estimated at 7.7142 with a standard error of 2.842 and a p-value of 0.008 which means it is statistically significant. The coefficient of the variable school_count_per_month is estimated at 1.0045 with a standard error of 0.332 and a p-value of 0.003 which means it is statistically significant. The variable tavg has a negative estimate of the coefficient of -1.6009, a standard error of 0.971 and a p-value of 0.103 which means it is not statistically significant.

The influencePlot() function of the Statsmodels library allows plotting the leverage, discrepancy, and influence of each observation in one figure. The studentized residuals on the y-axis show how unusual a data point is with values further away from 0 being more unusual. The H Leverage on the x-axis indicates the leverage on the coefficients. The size of the bubbles is based on the Cook's D value of each observation. The Cook's D value reflects the influence of each observation by summarizing how much all the values in the model change when this observation is removed.

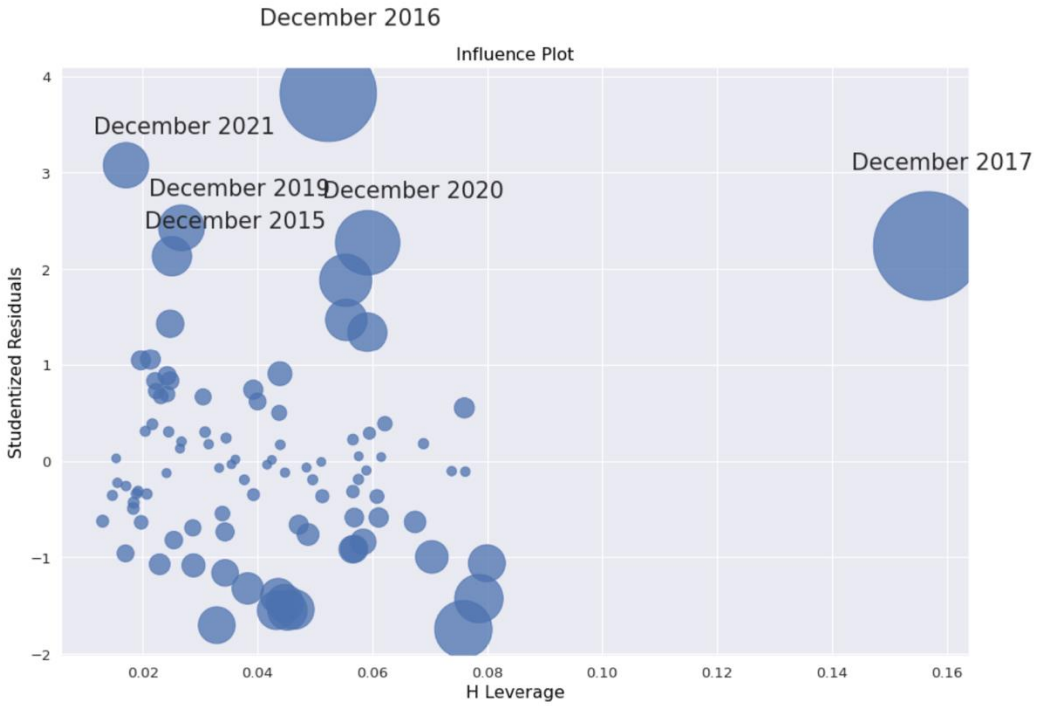


Figure 8 Influence Plot of observations in OLS regression

Based on the Cook's D value, the most influential outliers are labeled in the graph. All six observations are months of December. In some cases, it is recommended to drop the identified outliers and rebuild the model. But in this case, we can recognize a pattern of December months in the outlier information. These observed outliers have the highest studentized residuals but represent the most important month on the clam market in terms of sales volume. Therefore, it is crucial to find a statistical model that is able to predict this month more accurately. To improve the model, we could continue with feature engineering to include time variables, e.g., December as a categorical variable and a lagged dependent variable in the regression. As there exist already time series models, we continue with one of these instead.

5 Autoregressive integrated moving average (ARIMA)

In the following analysis, ARIMA's application to the Galician fisheries' proxy data is practiced. It takes advantage of the fact that time series forecasting tools are no longer limited by complexity and time but are provided by open-source applications (Khan, 2020). As the composing research paper already mentioned, the predictor of ARIMA is a composition of linear to historical values and errors. (Khan, 2020). Due to this, the application of ARIMA is essential and viable for predicting the total market demand for the near future. By testing the auto ARIMA values and building customized ARIMA (p,d,q) or seasonal ARIMA(p,d,q)(P,D,Q) m models, it is possible to attain a detailed forecasting model.

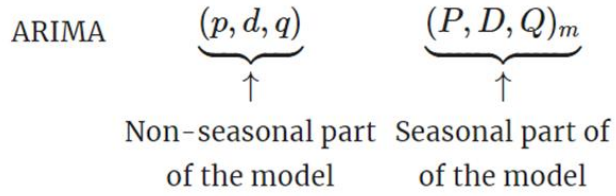


Figure 9 Seasonal Arima formula (Prasad, 2021)

Based on this, the retrieved data is prepared for the use of a seasonal ARIMA model. Afterward, the model is applied to unknown test data to check and evaluate the model's accuracy. This process aims to construct a formula that will serve as a baseline index for Oceano Fresco using manual inputs.

The model was applied to the Galician dataset, which was made available free of charge by the Galician government. Due to a time constraint caused by the availability of historical weather forecasts, only the years 2014-2021 are considered. In particular, the columns Quantity sold in kg, off_days_per_month, school_count_per_month, and column tavg (Average temperature) have already been described in more detail in section 5.1 and are intensively considered and used. The applied ARIMA model was coded and evaluated in Python language in Visual Studio Code. In the final step, the Mean Absolute Percentage error (MAPE) and Median Absolute

Percentage Error (MDAPE) were calculated to determine the model's accuracy. More precisely, the model's performance is measured by how far our predicted values deviate from the actual values.

5.1 Data Pre-processing

In order to utilize ARIMA correctly, it is essential to establish clear guidelines. Primarily, the Galician Fisheries Proxy Dataset must be amalgamated every month. This is a necessary step as ARIMA relies on the past values of the time series (its own values and lagged forecast mistakes) and produces an equation to predict the values going forward. (Duke, 2019)
Different data purification methods are used to construct a unified data set to ensure uniformity. This dataset is then presented as a data frame, which will be further utilized to compute the ARIMA model. In section 5 of this thesis, the demand factors have been visualized, making their seasonal patterns visible. Finally, a closer examination of the seasonality will be conducted to apply the model.

5.2 ARIMA Prediction

This section describes the steps taken to find the best possible ARIMA model. After cleaning and manipulating the data frame, it is possible to use the data set to forecast demand. First, a prediction period is determined, which serves as a test set in the given example. Ripley (1996) defines a test set as a set of examples used to assess a model's performance. In this specific case, the prediction period is set to 30 periods, whereas one period corresponds to one month and which is approximately a 70/30 distribution of the complete data set. Empirical evidence regarding the optimal ratio of test to train set was provided by researchers at the University of Texas, who recommended a ratio of 70/30 to avoid overfitting the data and still obtain the best possible result. (Gholamy et al., 2018) Consequently, the first 66 months, or 5.5 years, are

treated as a training set for the ARIMA model, and the remaining 30 months, or 2.5 years as a test set for our prediction.

Furthermore, the dataset is divided into y and X subsets. X, in this case, stands for exogenous factors, school vacations, days off per month, and average temperature. Y signifies the variable that needs to be predicted, the number of clams sold in kilograms. Several hyperparameters exist when using the auto_arima function, provided by the Python library pmdarima, which will now be explained in detail. Hyperparameters are predefined individual parameters within a model that, through gradual adjustment, can significantly change the result of the model. (Elgeldawi et al. 2021)

```
model_fit = pm.auto_arima(y=df_train,X=df_train_exo, test='adf',
                          max_p=5, max_d=5, max_q=5,
                          seasonal=True, m=12,
                          max_P=5, max_D=5, max_Q=5,
                          trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True,n_fits=30,method="bfgs")
```

Figure 10 Fitting the ARIMA model

The variables y and X have already been explained in detail and represent merely the predictor and the exogenous factors. Initially, creating a specific basic structure that allows the user to build an efficient model is vital. Thereby the usage of the base parameter 'error_action,' which is set equal to 'ignore,' warning the user if the model cannot fit, and controlling the error behavior of ARIMA, is needed to circumvent multiple errors or warning occurrences. By setting it to 'ignore,' the model does not interrupt when encountering an error. Because by using pmdarima, some warnings may occur, it is helpful for the user to set 'suppress_warnings' to 'True ', squelching the warnings concerning ARIMA. As a third and last point to create a basis for further work, the parameter 'stepwise' is set to True. According to Hyndman and Khandakar, testing every single hyperparameter composition is not feasible since a more extensive range

of the variables p, d, q, P, D & Q leads to an exponentially increasing number of possibilities (2008). Their approach allows a faster convergence to the minimum error coefficient and avoids an over-fitting of the model. The 'test' parameter is used to examine the data set for stationarity. George P. Box and Gwilym Jenkins published stationarity principles in their work "Time Series Analysis: Forecasting and Control," emphasizing that non-stationary data can be made stationary by "differentiating" the time series. (Box et al., 2022) This allowed them to pull apart specific trends or seasonality, as these up-and-down tendencies would be anticipated and interfere with the outcome of the forecast. This results in the definition of stationarity, expressed as "steady over time," meaning that trends such as seasonality or the like are altered so that they will not have any appreciable effect on the model's output. In this specific case, the Dickey-Fuller (adf) test is applied, with which variables (time series) can be tested for non-stationarity. It provides information about the order of integration of variables (time series), which simultaneously indicates whether a series is stationary or non-stationary (Lauridsen and Kosfeld, 2006). The three most relevant parameters in an ARIMA model are the following.

As already described in section 4.2, the variable q stands for the order of the moving average ('MA'), p for the number of time lags of the autoregressive ('AR') model, and d which signifies the order of first-differencing or the number of non-seasonal differences. Claesen described in his article "Hyperparameter tuning in Python using Optunity" that hyperparameter tuning embodies the optimization of single hyperparameters to get the best possible result of a model without significant manual effort. Thereby the same model is tested and evaluated repeatedly in a step-by-step approach formulated by Hyndman and Khandakar through a more significant number of possibilities. The equivalent to the lowercase letters p, d, and q are P, D & Q. Instead of other complicated variables, these symbolize the respective portion of the seasonal model of the lowercase variables. The model evaluates the optimal parameters for each parameter in the range of 1 to 5 based on the values of max_d, max_p, max_q, max_P, max_D, and max_Q (all

set to 5). If the model had been evaluated by randomly estimating the optimal parameters, there would have been 15625 options. Additionally, the model implies that a seasonal cycle occurs annually by setting the parameter 'seasonal' to true and m, which denotes the period of seasonal differencing, to 12. Furthermore, the number of ARIMA models to be fit is indicated by the variable n_fits. The value 30 is used for the number of prediction periods and the number of ARIMA models required. Lastly, the algorithm 'Broyden-Fletcher-Goldfarb-Shanno' (BFGS) was used as a method, which implies the type of solver from the scipy.optimize library.

5.3 Result

In the subsequent section, the implications and outcomes of the auto_arima technique that was just outlined will be examined and appraised. Initially, the ARIMA model outputs are compared with and without external variables. To demonstrate all possible combinations of variables, split validation is utilized to discover the ideal fit. In this regard, all models are closely examined, and the individual values are positioned side by side to authenticate the models. Specific attention is given to examining the residuals and the related error variables MAPE and MDAPE, which ascertain the model's accuracy.

The following three steps are considered in all cases for establishing a standardized comparison of each model. These lay the foundation for evaluating specific statistical models (Minitab, 2022).

1. Ascertain whether each term in the model is critical
2. Determine how accurately the model fits the data
3. Identify whether your model satisfies the underlying assumptions of the analysis

Split Validation

aic	bic	hqic	exogenous_variables	MAPE	MDAPE	order	seasonal_order	sum	factor
567.08	578.03	571.41	off_days_per_month	11.90%	6.93%	(0, 0, 0)	(1, 0, 1, 12)	1716.53	1.19
454.79	458.77	456.33	school_count_per_month	16.11%	13.03%	(0, 0, 0)	(0, 1, 0, 12)	1369.89	1.30
451.42	465.22	456.73	tavg	10.46%	7.76%	(1, 1, 3)	(0, 1, 1, 12)	1373.37	1.04
568.91	582.04	574.10	off_days_per_month, school_count_per_month	11.39%	7.77%	(0, 0, 0)	(1, 0, 1, 12)	1725.05	1.21
569.61	591.51	578.26	off_days_per_month, tavg	10.04%	7.39%	(2, 0, 2)	(1, 0, 1, 12)	1739.38	1.18
456.50	462.47	458.80	school_count_per_month, tavg	16.30%	13.73%	(0, 0, 0)	(0, 1, 0, 12)	1377.77	1.33
570.61	583.75	575.80	off_days_per_month, school_count_per_month, tavg	15.09%	13.75%	(0, 0, 0)	(1, 0, 0, 12)	1730.16	1.46
449.54	461.36	454.08	Without Exogenous	10.56%	8.41%	(1, 1, 3)	(0, 1, 1, 12)	1364.97	1.05

Table 2: Results of Arima split validation

Table 2 provides the findings of the split validation evaluation of the different variables. The auto_arima technique is utilized to examine all possible amalgamations of the variables. The results generated an assessment of each variable's combination performance. The performance indicators mentioned above were evaluated to obtain a comprehensive view of the individual results. Aic, bic and hqic are used to identify the best-fitting model by contrasting it with other ARIMA models. Since these performance indicators share the notion that lower values equate to a better-extracted model, each result was aggregated to create a sophisticated overview. In this instance, the 'sum' column is the result of the accumulation of the aic, bic, and hqic. The individual components were indexed and unified into one component to improve the comparison of the outcomes, allowing each equal measurement importance. In this instance, the minimal value of each performance metric was used as the index and compared to the remaining figures in the specific column. This created a percentage difference between the index value and the other values of the column, showing how distant a factor was from the minimum of the particular column. A number nearer to one reflects the most suitable combination. These results were totaled and divided by the number of variables utilized to create an optimal distinction between each variable blend. A noteworthy observation is that the fewer exogenous variables the ARIMA model obtains, the lesser the discrepancies from the

actual values. A higher quantity of variables does not directly equate to a better performance of the model. The results of split-validation demonstrate that the most effective models are either the ones that only consider the 'tavg' variable or the models that do not account for any external elements. It is thus evident that the combination of data regarding the number of clams sold combined with data on average temperature as an external factor was the most suitable combination, as it had the highest accuracy and the lowest error rate.

5.4 Limitation

Although the results presented in Table 2 provide information about the effectiveness and significance of individual variables, it was concluded that the initial model, which consists of all exogenous variables, should be kept since it adheres to the predefined objectives and the framework set by Oceano Fresco. The current demand variables are of great importance to Oceano Fresco. In addition, the chosen model provides the possibility to deeply analyze the outcomes and incorporate external variables to separate the factors that impact demand. Consequently, Oceano Fresco is endowed with a certain degree of interoperability, serving as a platform for further projections. Since until now, only a proxy set from Galicia was used as a data basis, Oceano Fresco intends to make its own statistical assumptions. In the future, it is of tremendous importance that the statistical significance of the variables is repeatedly tested and that Oceano Fresco formulates and incorporates additional variables into the model. In doing so, Oceano Fresco could tailor the ARIMA model to their company and provide even more accurate predictions.

5.5 Most suitable model

In the following section, the seasonal ARIMA model (0,0,0) (1,0,0)12 is considered more intensive to draw insights from its application. The three steps defined above are used to

examine the results to guarantee a comprehensive examination. Furthermore, the results are visualized to compare the ARIMA prediction with the predefined test set.

Result with exogenous variables

SARIMAX Results

Dep. Variable:	y	No. Observations:	66
Model:	SARIMAX(1, 0, 0, 12)	Log Likelihood	-279.305
Date:	Tue, 22 Nov 2022	AIC	570.611
Time:	17:15:35	BIC	583.748
Sample:	01-01-2014 - 06-01-2019	HQIC	575.802
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	15.4561	7.053	2.191	0.028	1.633	29.279
off_days_per_month	1.0224	2.379	0.430	0.667	-3.640	5.685
school_count_per_month	0.1557	0.482	0.323	0.747	-0.789	1.101
tavg	-0.7936	1.183	-0.671	0.502	-3.113	1.526
ar.S.L12	0.8429	0.066	12.698	0.000	0.713	0.973
sigma2	214.7413	43.884	4.893	0.000	128.729	300.753

Ljung-Box (L1) (Q):	0.06	Jarque-Bera (JB):	30.27
Prob(Q):	0.81	Prob(JB):	0.00
Heteroskedasticity (H):	1.03	Skew:	1.03
Prob(H) (two-sided):	0.95	Kurtosis:	5.60

Figure 11 Overview, SARIMAX results

Critical Terms

As seen in Figure 11, the result of the auto_arima function defined above is a single model, which will serve as a predictor for the test set that has already been defined in a subsequent step. The best model with exogenous variables is 'ARIMA (0,0,0) (1,0,0)12 ', which refers explicitly to the fact that there are 0 autoregressive terms, 0 differencing terms, 0 moving average terms, no first-order moving average terms with a lag of 12 months, and one autoregressive average term with a lag of 12 months and that the model should be run on a monthly basis.

To gauge the significance of the relationship between the response and each term in the model, one must compare the p-value to the significance level. Adopting a significance level of 0.05

implies a five percent chance of mistakenly deeming the term as not meaningfully different from 0 when it is (Di Leo and Sardanelli 2020). The null hypothesis states that the term is not significantly different from 0, indicating no association between the term and the response. According to the results in Figure 11, the variables intercept, sigma2, and ar.S.L12, which count for a pertinence of the previous year's value, appear to be significant coefficients for this model. The null hypotheses of these values are rejected because the P value is below the required significance of 0.05. Thus, these variables have a statistically significant contribution to our model.

In addition to the p-values, additional factors can influence the results of an ARIMA prediction. The first is the Ljung-Box statistic, which measures the autocorrelation of the residuals. A high Ljung-Box statistic indicates that the residuals are highly autocorrelated, which can negatively impact the accuracy of the prediction. The second factor is the heteroskedasticity statistic, which measures the variability of the residuals. A high heteroskedasticity statistic indicates that the residuals are highly variable, which can also negatively impact the accuracy of the prediction. The third factor is the Jarque-Bera statistic, which measures the skewness and kurtosis of the residuals. A high Jarque-Bera statistic indicates that the residuals are either very skewed or very kurtotic, which can negatively impact the accuracy of the prediction. In this case, the Ljung-Box statistic is relatively low, with a value of 0.06, indicating that the residuals are not highly autocorrelated. The heteroskedasticity statistic of 1.03 is also relatively low, indicating that the residuals are not highly variable. However, the Jarque-Bera statistic of 30.27 is above average, implying that the residuals are either very skewed or very kurtotic. This could negatively impact the accuracy of the prediction.

Model accuracy

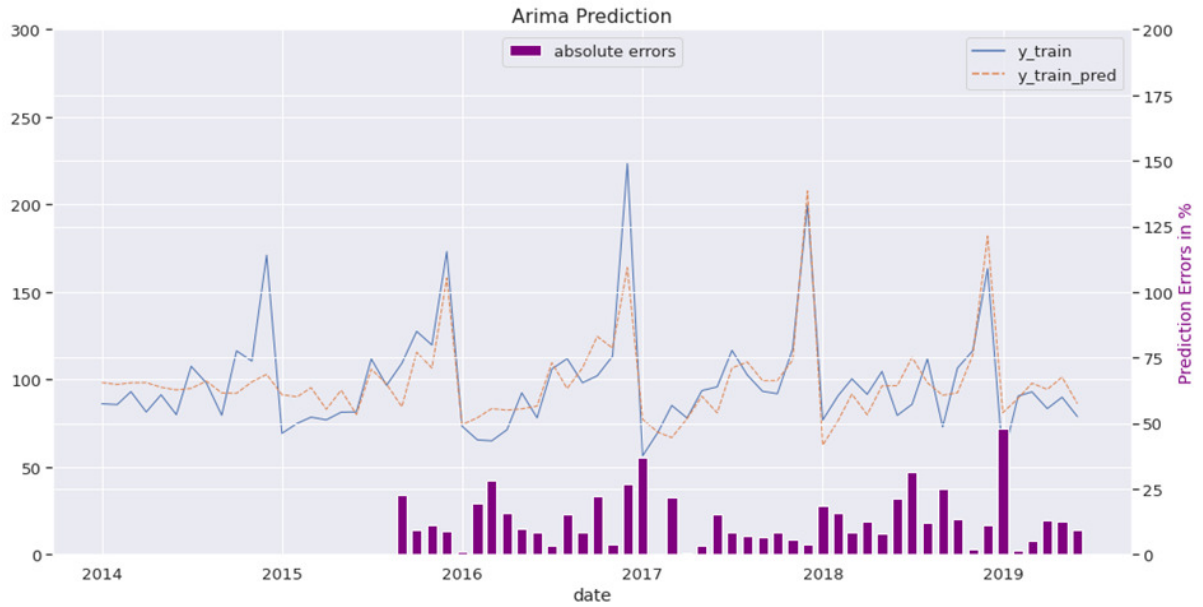


Figure 12 ARIMA prediction performance and absolute errors

The best-fitting model is applied to the training set in the first step to visualize a first performance check. The blue line inside Figure 12 indicates the actual sales values from 2014-2019, and the orange dashed line represents our predicted values. As an additional visualization of the model's performance, the purple bars denote the percentage difference between the actual and the predicted values. From this graph, the results predicted by the model differ only marginally from the actual values.

After validating the initial hypothesis with MAPE and MADP metrics, the model was tested on a previously determined test set. The model's accuracy was then evaluated against the test set by comparing the actual and predicted values. Figure 13 illustrates the assessment of the test set, which includes the last 30 months of our data frame. Table 3 provides further details of the variables. Our predicted values are slightly different from the actual values, yet the model was

successful on the set it was not trained with. As the model can forecast future months in addition to the ones used for training, it will be a valuable asset.

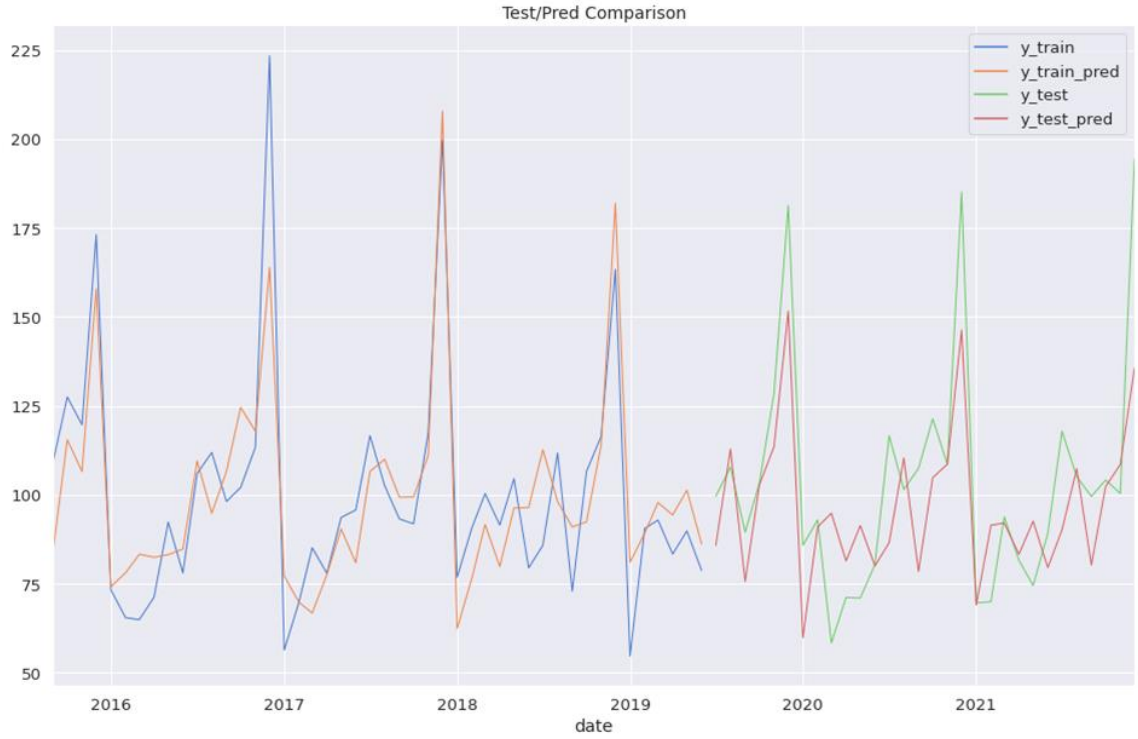


Figure 13 ARIMA prediction performance

Variable	Description
y_train (blue line)	Actual values of data frame limited from 2014 to June 2018 – used as a train set
y_train_pred (orange line)	Predicted values for data limited from 2014 to June 2018
y_test (green line)	Actual values of data frame limited from July 2018 to December 2021 – used as a test set
y_test_pred (red line)	Predicted values for data limited from July 2018 to December 2021

Table 3: Explanation of terms

The initial conclusions are strongly corroborated by the data presented in Table 4: Results of ARIMA prediction. A MAPE of 15.08% and an MDAPE of 13.75% demonstrate that the

estimated values are only off by 13.75% compared to the actual values. Overall, the preliminary findings of the ARIMA prediction are reassuring. The forecast is precise, and the error rate is low. This augurs favorably for the capability of the prediction to accurately estimate sales for the following month.

Metric	Description	Value
MAPE	Mean Absolute Percentage Error	15.087
MDAPE	Median Absolute Percentage Error	13.74

Table 4: Results of ARIMA prediction

Identify whether the model satisfies the underlying assumptions of the analysis.

With the calculated ARIMA Model, Oceano Fresco has the ability to forecast sales for the next month based on several factors. First, the forecast reflects an index to be used compared to the previous month and the previous year of the month. In this way, Oceano Fresco can base its forecast on historical data and current manual input factors such as temperature, number of days off, and number of school vacations. As a result, the following formula can be applied by the company to predict the following monthly index:

Definition	Equation
General Equation	$yt = \mu + (exogenous\ variables \cdot coefficients) + \phi_1 \cdot yt_{12}$
Applied Equation	$Value = intercept + (offday \cdot cf + schoolHol \cdot cf + tavg \cdot cf) + sar_{cf} \cdot yt_{12}$
Example for January 2022 **off_days = 10** **school_holidays = 5** **tavg = 10.5**	$January_{2022} = 15.46 + (10 \cdot 1.02 + 5 \cdot 0.156 + 10.5 \cdot (-0.79)) + 0.84 \cdot January_{2021}$

Table 5: ARIMA Equations

6 Outlook

The overall deliverables of this project contain simplifications and are subject to further improvement. Especially the waste opportunities for enhanced data analytics and further tuning of the forecasting model could be addressed to further improve this project/model/work. This section explores possibilities of future work, specifically on this project and on the Portuguese bivalve market research topic.

6.1 Project Deliverables

This section explains how critical deliverables of this project could be enhanced or extended by automation and further data collection.

6.1.1 Automation of Data Sources

The current dashboard uses various data sources, namely the historical sales data, a volume and price plan, current inventory levels, harvesting restrictions, weather data, and information about upcoming public and school holidays. Most of the values need to be updated manually, either weekly, monthly, or yearly. This section explores the possibilities of automating the data import.

- Sales data

The current sales data is manually copied each month to generate the baseline of the last month. This could be further improved if a connection to the company's ERP system were established. Additionally, the live updating could further extend the dashboard's functionality to estimate if the current demand forecast will be reached this month.

- Harvesting restrictions

The harvesting restrictions are published weekly and must be updated in the dashboard. It is the most time-intensive updating task due to the granularity of the data, which is separated by region and species. A potential automation step could involve a web crawler to collect the harvesting restriction data automatically once per week. This would also greatly benefit the idea of a *Forecast of Harvesting Restrictions* from the above section. On the other hand, the needed development and maintenance effort could constrain resources within the company, and it might be more beneficial to approach the regulatory bodies to retrieve the data over a predefined interface.

- Weather Forecast

The current weather forecast is updated once per week with a spectrum of 21 days into the future. This shows great potential to include an automated API call to one of the multiple available providers in the market. Similar to the data retrieval for the analysis in a previous section, the same data could be retrieved automatically for the upcoming weather forecast. Furthermore, more granular data would benefit the model's overall performance, as at the moment an overall weather forecast for the entire Portuguese mainland is used.

- Holiday

The Holiday data is updated once a year and could benefit from automation similar to the weather forecast. However, it must be noted that the manual workload is relatively low, and it needs to be evaluated if this automation is economically feasible.

6.1.2 Data Collection by Oceano Fresco

The current analysis used for this project is purely based on the proxy data from the Galician market. Therefore, verifying the assumptions throughout this paper with actual data collected by Oceano Fresco is crucial. Especially the sales data including the volume and price

information plays a significant role in the current forecasting model and inventory optimization. Therefore, the following data should be collected once the business is in operation:

- Customer details
- Customer segments
- Product details (clam species, size)
- Sales price
- Sales volume
- Date of sale

This could be further used to forecast individual customers and species demand volumes. The granularity of data would significantly improve the forecast quality and could be used to tailor the sales offer for each customer. For example, customers in the premium sector (e.g., gourmet restaurants) could be approached differently than retailers or distributors in terms of price as they may buy in bulk.

Furthermore, besides the primary sales data OF could increase its data collection regarding other aspects of the business. The goal would be to create an internal database for any type of data, e.g.

- Water quality and algae occurrence (in the home region and potential new areas)
- Inventory of clams by size and species including growth/death rates
- Clam phenotypes (e.g. size, color)
- Additional weather conditions (e.g., precipitation)
- Distributor information and seller feedback
- Cost of production
- Marketing efforts

This could be used in further analyses to identify correlations between exogenous variables and volume or price points. On the other hand, certain vital qualitative factors contain e.g.

- Market growth information (domestic and international)
- Cross-selling opportunities (partnership potential)
- Logistics and transport ways/modes, technological advances
- Business investments (e.g., automation)

6.2 Data analysis

The collected data can be used to improve the data analysis further and identify which additional factors impact the sales of Oceano Fresco.

6.2.1 Forecasting methodology

The current forecast prediction is purely based on the demand side with a volume forecast. Additionally, there is the possibility to extend the analysis further and create a second forecast for the price elasticity of the bivalve market. In this scenario, the correlation between the supply restrictions and price could be analyzed further and used as an exogenous factor in an ARIMA prediction for price forecasting. Llanes et al. (2020) did an overview of studies about perishable goods and their forecasting methods, comparing multiple other research papers. Their combined conclusion is structured in three categories:

- **Influencing factors**

There are several types of demand patterns including short-term fluctuations (holidays, promotions), medium-term seasonal patterns (school holidays) and long-term trends (economic situations). In addition, environmental factors such as climate, advertising campaigns and local events can also affect demand patterns. For example, a company may see an increase in sales during a holiday promotion or when prices are lowered. However, demand patterns can also be

influenced by other factors such as the number of purchases of a particular product, holidays, product prices, and variations in prices and promotional periods.

- **Forecast horizon**

The forecast horizon is the amount of time that a forecaster must make a forecast. The time series are used for short-term forecasts and the causal methods for medium- and long-term forecast horizons.

- **Forecasting Models**

Key insights include that time series models have the advantage of being able to make short-term forecasts with a relatively limited amount of data. However, they are not recommended for forecasting elements that can be influenced by changing external factors, such as climatic factors, preferences of people, and promotional events, because the error in their predictions can be high. Soft computing models can be used to improve the performance of time series models for non-linear inputs. These models can improve the relationship between input and output data, making them more effective than time series models. Additionally, the forecasting method could be extended and more sophisticated models like a machine learning model could be used. (Llanes et.al., 2020)

6.2.2 Dynamic forecasting models

The current model includes a fundamental demand forecast based on proxy data from the Galician market. Additionally, it could be analyzed how the limited availability of clams in the open-sea farm could be utilized in a more dynamic demand model. In this case the available harvestable number of clams in the open-sea farm would significantly influence the supply side beside the already identified operational harvesting restrictions. The model could incorporate the further growth development of each clam which results in greater weight, but additionally a compound effect because more giant clams generated an exponentially higher sales price. The

time-shifting mechanism of the current model could incorporate more operational limitations of the current harvesting process and precisely forecast the optimal harvesting amount each period.

6.2.3 Forecasting of Harvesting Restrictions

The main issue with the supply side is irregularity, harmful algal blooms and the short-shelf life of clams that have been harvested. As a more extended harvesting ban potentially overstretches the average shelf life of clams, no cash flow is generated in that time. The better prediction of algal blooms is already studied in multiple papers and additionally multiple warning systems have been implemented in pilot areas. As OF will be one of the most prominent players in the traditionally multi-segmented bivalve market it develops a unique advantage over time with its data collection. Therefore, the collected data could be analyzed and used for a prediction system of algal blooms in the future.

7 Conclusion

During the first meetings, the deliverables were discussed among all parties involved. While the overall scope was agreed upon quickly, it also became clear that the project was limited by several factors.

On a general note, one can say that the root cause for all the limitations was the lack of internal data. Since the company did not yet engage in any real-world sales transactions, there were no sales volumes or sales prices on which to base a potential forecast model. To mitigate this, it was decided to use proxy data from the Spanish region of Galicia. However, one must keep in mind the structural differences between Portugal and Galicia. One notable difference, for example, is the increased demand for clams during the Christmas season. According to Oceano Fresco, this is a more Spanish phenomenon, while in Portugal fish is preferred over clams for Christmas. Another example of why this must be kept in mind is that the likelihood of occurrence of government-induced harvesting bans is currently also partly based on this proxy data. This is because Decussatus Boa, the clam species that Oceano Fresco plans to grow in this region is not yet eligible for harvesting. Further, due to the lack of actual sales data, no definitive price estimate can be made in the current version of the model. Rather, informative sales and price data from the Galicia region are provided to promote a more informed sales decision by the sales representative.

Under consideration of the mentioned limitations, it was agreed to focus on two concrete deliverables.

Regarding the first deliverable, it was defined as a dashboard that should provide relevant data in a prioritized and organized way. Moreover, Oceano Fresco's demand was to automate the monthly update process as much as possible. The challenges during the implementation were manifold, mainly because the preferred program to use was defined as Microsoft Excel so that

all future company employees could use it. Compared to programming tools such as Python, extracting data directly from the web is much more challenging. Consequently, all data used for the first deliverable must be typed into the input tabs manually. However, due to a close collaboration and frequent alignments between the team and Frederico from Oceano Fresco, a dashboard emerged that will support Oceano Fresco's sales staff in its future decision-making process. All graphs and illustrations on the dashboard add value by providing relevant information to the user to plan future sales transactions and decide on sales volumes and prices.

Compared to the first deliverable, the second deliverable was agreed to emphasize analysis more strongly. Oceano Fresco's request was to compare supply and demand factors of the coming month versus a baseline. The rationale was to support the forecast process for the next month, focusing on how many kilograms will be sold, as this directly affects the harvesting activities. To come up with the best possible solution for this task and to ensure a high-quality forecast, mainly two theoretical concepts were applied.

The ARIMA (0,0,0) (1,0,0)12 model, which is composed of 0 autoregressive terms, 0 differencing terms, 0 moving average terms, no first-order moving average terms with a lag of 12 months, and one autoregressive average term with a lag of 12 months, running on a monthly basis, is the most appropriate to consider exogenous variables. The MAPE (Mean Absolute Percentage Error) and MDAPE (Mean Directional Absolute Percentage Error) of 15.08% and 13.75%, respectively, declare that the estimated values of our prediction deviate only by 13.75% from the exact values. Overall, the preliminary findings of ARIMA prediction are comforting. The forecast is precise, and the rate of inaccuracy is low. This is a positive indication that the prediction has the ability to assess sales for the subsequent month precisely. In addition to the ARIMA model, it was decided to also apply the newsvendor model. While the forecast will give the company a specific figure, it is important to note that this forecast will probably be off the actual most of the time and that the forecast represents an expected average.

The inclusion of the newsvendor model enables a comparison between the net cost of producing too few (underage) and the net cost of producing too many clams for each month. This figure differs between the months as the sales price is for example significantly higher in December. As a result, taking the risk of producing more than forecasted in such a month with high sales prices may on average result in a positive contribution to profit. By applying the newsvendor model to the clam forecast model, these effects have also been accounted for. In total, low variable cost and a high expected sales price contributed to an increased inventory level recommendation using this approach.

Even though it is the common understanding of all parties that both deliverables fulfill the company's needs, it was also agreed that the underlying work must be seen as the starting point for future projects. For example, the current model still requires manual input of external data, such as weather data, wind data, as well as data regarding government induced harvesting bans. Further, the time-shifting mechanism in the current version is still based on assumptions regarding opportunity costs and marginal production costs. These assumptions should be supplemented after the launch of the operational activities to increase the predictive power and validity of the model. Moreover, forecasting is currently done monthly, and it is not possible to simulate demand on a weekly or daily basis.

Overall, the real value, especially for dashboard two, will have to be evaluated continuously as Oceano Fresco has launched its sales activities, and the amount of actual data is increasing. Only then can the forecast data be compared to the actuals on a larger scale. It will be the task of future NOVA teams to feed these figures into the model and thereby improve the forecast outcome. However, given the current situation and information level, the NOVA team and Oceano Fresco are confident that the conducted work will already help start the business in the first quarters of 2023.

8 Citations

A Koutsoyiannis. (1975) 1979. Modern Microeconomics. Second edition. London Macmillan Education Uk. https://ucanapplym.s3.ap-south-1.amazonaws.com/RGU/notifications/E_learning/study_online/A_Koutsoyiannis_Modern_Microeconomics_Se.pdf.

Britannica, T. Editors of Encyclopaedia. "clam." Encyclopedia Britannica, October 10, 2022. https://www.britannica.com/animal/clam.

Claesen, Marc & Simm, Jaak & Popovic, Dusan & De Moor, Bart & Leuven, Ku. (2014). Hyperparameter tuning in Python using Optunity.

Coelho, Pedro, Frederico Carvalho, Thomas Goulding, Paula Chainho and José Guerreiro. "Management Models of the Manila Clam (Ruditapes philippinarum) Fisheries in Invaded European Coastal Systems." 2021. Frontiers in Marine Science (Vol 8) www.frontiersin.org/articles/10.3389/fmars.2021.685307

Elgeldawi, Enas, Awny Sayed, Ahmed R. Galal, and Alaa M. Zaki. 2021. "Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis." *Informatics* 8 (4): 79. https://doi.org/10.3390/informatics8040079.

"EUMOFA - European Market Observatory for Fisheries and Aquaculture." n.d. Www.eumofa.eu. Accessed November 11, 2022. https://www.eumofa.eu/the-eumarket.

European Parliament and Directorate General for Internal Policies of the Union, Alicia Mosteiro Cabanelas, and Giuseppe Scarcella. 2016. "The Clam Fisheries Sector in the EU: The Adriatic Sea Case." https://data.europa.eu/doi/10.2861/401646

FAO 2022. Ruditapes philippinarum. Cultured Aquatic Species Informatio Programme. Text by Goulletquer, P.. Fisheries and Aquaculture Division Rome. Updated March 02,

2006. Accessed November 29, 2022. https://www.fao.org/fishery/en/culturedspecies/ruditapes_philippinarum

Fonseca, Rosalina, Ramunas M. Vabulas, F. Ulrich Hartl, Tobias Bonhoeffer, and U. Valentin Nägerl. 2006. "A Balance of Protein Synthesis and Proteasome-Dependent 58 Degradation Determines the Maintenance of LTP." Neuron 52 (2): 239–45. https://doi.org/10.1016/j.neuron.2006.08.015.

Gholamy, Afshin; Kreinovich, Vladik; and Kosheleva, Olga, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation" (2018). Departmental Technical Reports (CS). 1209. https://scholarworks.utep.edu/cs_techrep/1209

Hyndman, Rob J., and Yeasmin Khandakar. 2008. "Automatic Time Series Forecasting: TheforecastPackage ForR." *Journal of Statistical Software* 27 (3). https://doi.org/10.18637/jss.v027.i03.

"Introduction to ARIMA Models." 2019. Duke.edu. 2019. https://people.duke.edu/~rnau/411arim.htm.

Khan, Shakir, and Hela Alghulaiakh. 2020. "ARIMA Model for Accurate Time Series Stocks Forecasting." *International Journal of Advanced Computer Science and Applications* 11 (7). https://doi.org/10.14569/ijacsa.2020.0110765.

Lauridsen, Jørgen, and Reinhold Kosfeld. 2006. "A Test Strategy for Spurious Spatial Regression, Spatial Nonstationarity, and Spatial Cointegration." *Papers in Regional Science* 85 (3): 363–77. https://doi.org/10.1111/j.1435-5957.2006.00087.x.

Llanes, Rudibel Perdigón, Hubert Viltres Sala, and Arturo Orellana García. "Models for predicting perishable products demands in food trading companies." Revista Cubana de Ciencias Informáticas 14, no. 1 (2020): 110-135.

Lewis-Beck, Michael S. 1980. "Applied regression: An introduction." Sage University Paper Series on Quantitative Applications in the Social Sciences (07-022). Newbury Park, CA: Sage.

Minitab. Accessed November 24, 2022. https://minitab.com/

Oceano Fresco, 2022; Interviews & Information based on Corporate Presentations and webpage https://oceano-fresco.pt retrieved on 12.12.2022

Oliveira, Jacinta, Fernanda Castilho, Ângela Cunha, and Mário Jorge Pereira, "Bivalve Harvesting and Production in Portugal: An Overview." 2013. Journal of Shellfish Research (Vol. 32, Issue 3, p. 911-924). National Shellfisheries Association. https://doi.org/10.2983/035.032.0334

"Pescado, Marisco, Galicia." n.d. Www.pescadegalicia.gal. Accessed November 17, 2022. https://www.pescadegalicia.gal/.

Prasad, Eswara. 2021. "What Is Auto-ARIMA?" Medium. May 20, 2021. https://medium.com/featurepreneur/what-is-auto-arima-b8025c6d732d.

Ripley, Brian D. 1996. *Pattern Recognition and Neural Networks*. Cambridge ; New York: Cambridge University Press.

Seabold, Skipper and Josef Perktold. 2010. "statsmodels: Econometric and statistical modeling with python." 9th Python in Science Conference.

Smaal, Aad C., Joao G. Ferreira, Jon Grant, Jens K. Petersen, and Øivind Strand, Goods and Services of Marine Bivalves. 2019. Springer Cham.

Wilson, Granville Tunnicliffe. 2016. "Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, Pp. 712. ISBN: 978-1-118-67502-1." *Journal of Time Series Analysis* 37 (5): 709–11. https://doi.org/10.1111/jtsa.12194.

9 Appendices

9.1 Tables

Translated column label	Original column label
Date of Sale	Data de venda
Biological group of the species	Grupo biolóxico da especie
Fao code of the species	Código Fao da especie
Galician name of species	Nome en galego da especie
Province of sale	Provincia da venda
Administrative area of sale	Zona administrativa da venda
Length of sales	Lonxa de venda
Quantity in kilograms	Cantidade en quilogramos
Amount in euros	Importe en euros
Average price in euros per kilo	Prezo medio en euros por quilo

Table 6 Explanation of the dataset variables in Spain and English

Date	Quantity	Off-days	School Holidays	Tavg
01.01.2014	86.09	9	3	12.41
01.02.2014	85.63	8	0	11.87
01.03.2014	93.00	10	3	13.88
01.04.2014	81.39	10	15	16.10
01.05.2014	91.21	10	0	16.52
01.06.2014	79.86	10	0	18.42
01.07.2014	107.50	8	27	20.05
01.08.2014	97.94	11	31	19.42

01.09.2014	79.52	8	14	20.81
01.10.2014	116.35	8	0	18.28
01.11.2014	110.49	10	0	12.65
01.12.2014	171.02	10	15	10.14
01.01.2015	69.20	10	2	9.38

Table 7: Data frame

9.2 Python Code

Import libraries

In [105…

```
import glob
import itertools
import json
import os
import pickle
import time
import warnings
from datetime import datetime

import geopy
import holidays
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import pmdarima as pm
import requests
import seaborn as sns
import statsmodels.api as sm
from geopy.geocoders import Nominatim
from google.colab import drive
from meteostat import Daily, Monthly, Point, Stations
from scipy.stats.stats import pearsonr
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

warnings.filterwarnings("ignore")
```

IMPORT DATA

In [1]:

```
frame = 0
drive.mount("/content/gdrive")

frame = 0
path = r"gdrive/MyDrive/arima_prediction"  # use your path
all_files = glob.glob(path + "/*.csv")
li = []

all_files.remove("gdrive/MyDrive/arima_prediction/harvesting_ban.csv")

for filename in all_files:
    df = pd.read_csv(
        filename, index_col=None, header=None, on_bad_lines="skip", sep=";"
    )
    li.append(df)

frame = pd.concat(li, axis=0, ignore_index=True)

frame.columns = [
    1,
    2,
```

```
    3,
    4,
    5,
    6,
    7,
    8,
    9,
    10,
]

frame.columns = [
    "Verkaufsdatum (Datum)",
    "Biologische Gruppe der Art (biologische Gruppe)",
    "Fao-Code f\xc3\xbcr die Art (fao)",
    "Galicischer Name der Art (Art)",
    "Verkaufsprovinz (Provinz)",
    "Administrativer Verkaufsbereich (Zone)",
    "Verkaufsmarkt",
    "Menge in Kilogramm (Menge)",
    "Einfuhr in Euro (Import)",
    "Durchschnittspreis in Euro pro Kilo (Preis)",
]

unique_cities = frame["Verkaufsmarkt"].unique()
```

```
Mounted at /content/gdrive
```

Data Manipulation

In [5]:

```
# Convert the 'Verkaufsdatum (Datum)' column to datetime type
frame["Verkaufsdatum (Datum)"] = pd.to_datetime(frame["Verkaufsdatum (Datum)"])

# Create a date range for the dates 01/01/2014 to 31/12/2021
dates = pd.date_range(start="01/01/2014", end="31/12/2021", freq="D")

# Store the dates in a dataframe with the header 'date'
df_date = pd.DataFrame(dates, columns=["date"])

# Convert the 'date' column to datetime type
df_date["date"] = pd.to_datetime(df_date["date"], format="%d.%m.%Y")

# Merge the original dataframe and the date dataframe on the 'date' and 'Verkaufsda
new_df = df_date.merge(
    frame, left_on="date", right_on="Verkaufsdatum (Datum)", how="left"
)

# Add year and month columns to the dataframe
new_df["year"] = new_df["date"].dt.year
new_df["month"] = new_df["date"].dt.month_name()
frame = new_df

# Group the dataframe by year and month and count the number of unique dates
df_days = frame.groupby(["year", "month"], as_index=False)["date"].nunique()

# Rename the 'date' column to 'number_of_days_per_month'
```

```
df_days.rename(columns={"date": "number_of_days_per_month"}, inplace=True)

# Merge the original dataframe with the new dataframe
frame = frame.merge(df_days, on=["year", "month"], how="left")

# Add a 'weekdays' column to the dataframe
frame["weekdays"] = frame["date"].dt.weekday

# Replace the values in the 'weekdays' column with 'Weekend' or 'Weekday' depending
frame["weekdays"] = np.where(frame["weekdays"].isin([5, 6]), "Weekend", "Weekday")

# Import the Portuguese holidays
pt_holidays = holidays.PT()

# Add an 'Is Holiday' column to the dataframe
frame["Is Holiday"] = frame["date"].apply(lambda x: x in pt_holidays)

# Read the school holidays data from an Excel file
school_holidays = pd.read_excel(
    "gdrive/MyDrive/arima_prediction/school_holidays_pt.xlsx"
)

# Convert the 'Date' and 'End' columns to datetime type
school_holidays["Date"] = pd.to_datetime(school_holidays["Date"], format="%d.%m.%Y"
school_holidays["End"] = pd.to_datetime(school_holidays["End"], format="%d.%m.%Y")

# Create an empty dataframe
school_holidays_dates = pd.DataFrame()

# Add a 'Date_test' column to the dataframe with date ranges
school_holidays_dates["Date_test"] = school_holidays.apply(
    lambda x: pd.date_range(x["Date"], x["End"]), axis=1
)

# Explode the 'Date_test' column to create a row for each date
school_holidays_dates = school_holidays_dates.explode("Date_test")

# Add a 'school_holiday_test' column to the original dataframe
frame["school_holiday_test"] = np.where(
    frame["date"].isin(school_holidays_dates["Date_test"]), "School Holiday", "Scho
)

# Create an empty dataframe
school_days_monthly = pd.DataFrame()

# Select the necessary columns from the original dataframe
school_days_monthly = frame[["year", "month", "date", "school_holiday_test"]]

# Replace the values in the 'school_holiday_test' column
school_days_monthly["school_holiday_test"] = school_days_monthly[
    "school_holiday_test"
].replace(["School", "School Holiday"], [0, 1])

# Remove duplicate rows
school_days_monthly = school_days_monthly.drop_duplicates()
```

```
# Add a 'school_count_per_month' column to the dataframe
school_days_monthly["school_count_per_month"] = school_days_monthly.groupby(
    ["year", "month"], as_index=False
)["school_holiday_test"].transform(lambda x: x.sum())

# Drop the 'school_holiday_test' column
school_days_monthly = school_days_monthly.drop(["school_holiday_test"], axis=1)

# Add an 'off_days' column to the original dataframe
frame["off_days"] = np.where(
    (frame["weekdays"] == "Weekend") | (frame["Is Holiday"] == True), 1, 0
)

# Create an empty dataframe
off_days_monthly = pd.DataFrame()

# Select the necessary columns from the original dataframe
off_days_monthly = frame[["year", "month", "date", "off_days"]]

# Remove duplicate rows
off_days_monthly = off_days_monthly.drop_duplicates()

# Add a 'count_per_month' column to the dataframe
off_days_monthly["count_per_month"] = off_days_monthly.groupby(
    ["year", "month"], as_index=False
)["off_days"].transform(lambda x: x.sum())

# Drop the 'off_days' column
off_days_monthly = off_days_monthly.drop(["off_days"], axis=1)

# Define a list of Ameixas
Ameixas = ["Ameixa babosa", "Ameixa fina", "Ameixa xaponesa"]

# Select rows from the original dataframe where the 'Galicischer Name der Art (Art)
df_2014_ameixas = frame[frame["Galicischer Name der Art (Art)"].isin(Ameixas)]

# Merge the Ameixas dataframe with the off_days_monthly dataframe on the 'year', 'm
df = df_2014_ameixas.merge(off_days_monthly, on=["year", "month", "date"], how="lef

# Merge the resulting dataframe with the school_days_monthly dataframe on the 'year
df = df.merge(school_days_monthly, on=["year", "month", "date"], how="left")

# Import the meteostat Stations class
stations = Stations()

# Create a geolocator object
geolocator = Nominatim(user_agent="myapplication")

# Create an empty list for the locations
location = []

# Create an empty dataframe
df_location = pd.DataFrame(columns=["latitude", "longitude"])

# Iterate through the list of cities
for city in unique_cities:
```

```
    # Get the location for each city
    location = geolocator.geocode(city)

    # If a location is available, store the latitude and longitude in the dataframe
    if location:
        df_location = df_location.append(
            {
                "city": city,
                "latitude": location.latitude,
                "longitude": location.longitude,
                "altitude": location.altitude,
            },
            ignore_index=True,
        )
    else:
        # If a location is not available, store NaN values
        df_location = df_location.append(
            {"city": city, "latitude": np.nan, "longitude": np.nan, "altitude": np.
            ignore_index=True,
        )

start = datetime(2014, 1, 1)
end = datetime(2021, 12, 31)

df_location.fillna(0, inplace=True)
cities = df_location["city"].unique()
# # Create a new dataframe to store the results
df_weather = pd.DataFrame()


# Append dataframes
df_complete = df.merge(df_location, left_on="Verkaufsmarkt", right_on="city")
df_complete.drop(columns=["city"], axis=1, inplace=True)
df_complete = df_complete.fillna(0)

df_complete.to_csv("df_complete.csv")
```

Exporting data frame

In [6]:
```
df_1 = df_complete.iloc[: int(len(df_complete) / 2), :]
df_2 = df_complete.iloc[int(len(df_complete) / 2) :, :]
```

In [ ]:
```
df_1.to_csv("first_half_sales_data.csv")
df_2.to_csv("second_half_sales_data.csv")
```

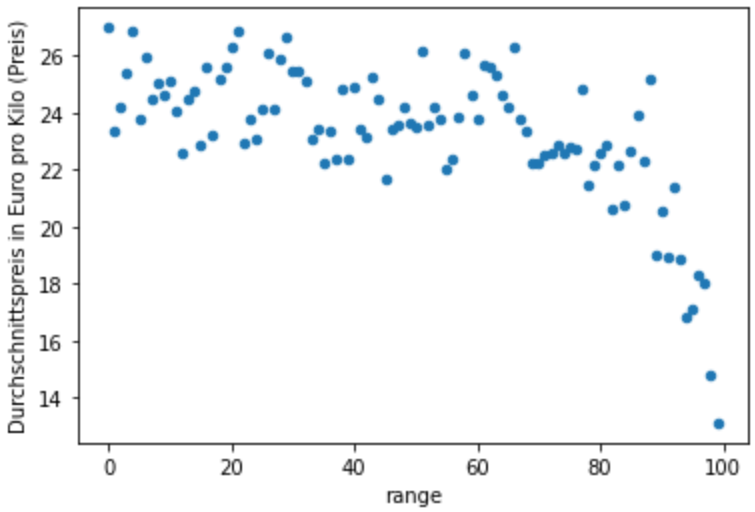
Price elasticity

In [7]:
```
df_dezember = df_complete[
    [
        "month",
        "Menge in Kilogramm (Menge)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
        "Verkaufsmarkt",
    ]
```

```
]
df_dezember = df_dezember[df_dezember["month"] == "December"]
df_dezember = df_dezember[
    ["Menge in Kilogramm (Menge)", "Durchschnittspreis in Euro pro Kilo (Preis)"]
].apply(lambda x: x.str.replace(",", "."))
df_dezember = df_dezember[
    ["Menge in Kilogramm (Menge)", "Durchschnittspreis in Euro pro Kilo (Preis)"]
].astype(float)
df_dezember = df_dezember[
    (df_dezember["Durchschnittspreis in Euro pro Kilo (Preis)"] > 10)
    & (df_dezember["Durchschnittspreis in Euro pro Kilo (Preis)"] < 150)
]
average_quantity = df_dezember["Menge in Kilogramm (Menge)"].median()
df_dezember = df_dezember.sort_values(
    by="Durchschnittspreis in Euro pro Kilo (Preis)", ascending=False
)
df_dezember = df_dezember.groupby(
    pd.qcut(df_dezember["Menge in Kilogramm (Menge)"], 100), as_index=False
)["Durchschnittspreis in Euro pro Kilo (Preis)"].mean()
df_dezember["range"] = df_dezember.index
df_dezember["range"] = df_dezember["range"].astype(int)
```

In [8]:
```
df_dezember.plot.scatter(x="range", y="Durchschnittspreis in Euro pro Kilo (Preis)"
```

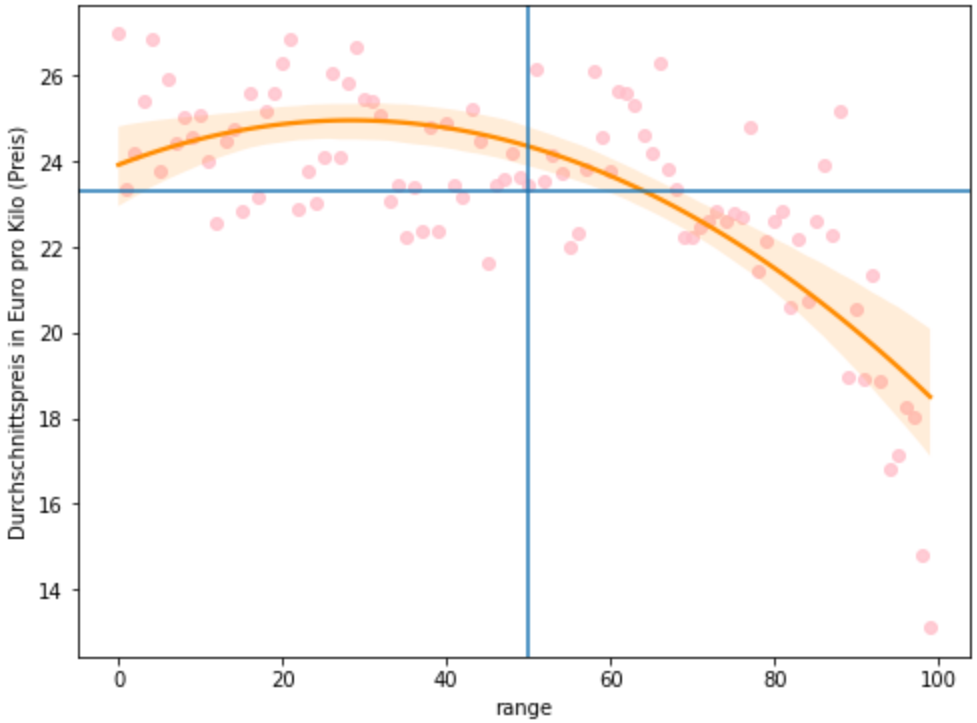
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44585713a0>
```



In [10]:
```
average_price = df_dezember["Durchschnittspreis in Euro pro Kilo (Preis)"].mean()
average_quantity = 50

fig, ax1 = plt.subplots(1, 1, figsize=(8, 6))

scatter = sns.regplot(
    x="range",
    y="Durchschnittspreis in Euro pro Kilo (Preis)",
    data=df_dezember,
    order=2,
    line_kws={"color": "darkorange"},
    scatter_kws={"color": "pink"},
)
scatter = scatter.axhline(average_price)
```

```
ax1.axvline(average_quantity)
plt.plot()
```

Out[10]: []



In [11]:
```
x_date = ax1.get_lines()[0].get_xdata()
y_date = ax1.get_lines()[0].get_ydata()

reg_line_frame = pd.DataFrame({"x": x_date, "y": y_date})
max = reg_line_frame["y"].max()
reg_line_frame.iloc[0:25, 1:2] = max
reg_line_frame
```

Out[11]:

	x	y
0	0.0	24.961901
1	1.0	24.961901
2	2.0	24.961901
3	3.0	24.961901
4	4.0	24.961901
...	...	...
95	95.0	19.211774
96	96.0	19.037947
97	97.0	18.861531
98	98.0	18.682527
99	99.0	18.500935

100 rows × 2 columns

In [12]:
```
df_dezember["regression_line"] = reg_line_frame["y"]
df_dezember["average_price"] = average_price
df_dezember["deviation"] = (
    (df_dezember["regression_line"] - average_price) / average_price
) * 100
df_excel_output = df_dezember[["deviation", "regression_line", "average_price"]]
df_excel_output.to_csv("price_elasticity.csv")
```

Korrelation

In [13]:
```
df_2014_ameixas[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
] = df_2014_ameixas[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
].apply(
    lambda x: x.str.replace(",", ".")
)

df_2014_ameixas[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
```

```
    ]
] = df_2014_ameixas[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
].astype(
    float
)
```

In [14]:
```
df_ameixas_babosa = df_2014_ameixas.loc[
    df_2014_ameixas["Galicischer Name der Art (Art)"] == "Ameixa babosa"
]
```

In [15]:
```
df_ameixas_xaponesa = df_2014_ameixas.loc[
    df_2014_ameixas["Galicischer Name der Art (Art)"] == "Ameixa xaponesa"
]
```

In [16]:
```
df_ameixas_fina = df_2014_ameixas.loc[
    df_2014_ameixas["Galicischer Name der Art (Art)"] == "Ameixa fina"
]
```

In [17]:
```
corr_matrix1 = df_ameixas_babosa.corr()
corr_matrix1["Durchschnittspreis in Euro pro Kilo (Preis)"].sort_values(ascending=F
```

Out[17]:
```
Durchschnittspreis in Euro pro Kilo (Preis)    1.000000
year                                           0.401566
Einfuhr in Euro (Import)                       0.140508
number_of_days_per_month                       0.039873
Is Holiday                                    -0.003394
off_days                                      -0.008825
Menge in Kilogramm (Menge)                    -0.018147
Name: Durchschnittspreis in Euro pro Kilo (Preis), dtype: float64
```

In [18]:
```
corr_matrix2 = df_ameixas_xaponesa.corr()
corr_matrix2["Durchschnittspreis in Euro pro Kilo (Preis)"].sort_values(ascending=F
```

Out[18]:
```
Durchschnittspreis in Euro pro Kilo (Preis)    1.000000
year                                           0.590758
Einfuhr in Euro (Import)                       0.282356
Menge in Kilogramm (Menge)                     0.144231
number_of_days_per_month                       0.071136
Is Holiday                                     0.006894
off_days                                      -0.002202
Name: Durchschnittspreis in Euro pro Kilo (Preis), dtype: float64
```

In [19]:
```
corr_matrix3 = df_ameixas_fina.corr()
corr_matrix3["Durchschnittspreis in Euro pro Kilo (Preis)"].sort_values(ascending=F
```

Out[19]:
```
Durchschnittspreis in Euro pro Kilo (Preis)    1.000000
year                                           0.317732
Einfuhr in Euro (Import)                       0.252371
Menge in Kilogramm (Menge)                     0.109788
number_of_days_per_month                       0.059846
off_days                                      -0.018075
Is Holiday                                    -0.018099
Name: Durchschnittspreis in Euro pro Kilo (Preis), dtype: float64
```

2. Korrelation

In [20]:
```
df_complete["day_of_week"] = df_complete["date"].dt.weekday
df_complete.head(5)
```

Out[20]:

	date	Verkaufsdatum (Datum)	Biologische Gruppe der Art (biologische Gruppe)	Fao-Code fÃ¼r die Art (fao)	Galicischer Name der Art (Art)	Verkaufsprovinz (Provinz)	Administrativer Verkaufsbereich (Zone)	Verka
0	2014-01-02	2014-01-02	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa	
1	2014-01-03	2014-01-03	Bivalvos	CLJ	Ameixa xaponesa	Pontevedra	Zona III - Arousa	
2	2014-01-03	2014-01-03	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa	
3	2014-01-04	2014-01-04	Bivalvos	CLJ	Ameixa xaponesa	Pontevedra	Zona III - Arousa	
4	2014-01-04	2014-01-04	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa	

5 rows × 24 columns

In [21]:
```
df_complete[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
] = df_complete[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
].apply(
    lambda x: x.str.replace(",", ".")
)

df_complete[
    [
```

```
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
] = df_complete[
    [
        "Menge in Kilogramm (Menge)",
        "Einfuhr in Euro (Import)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
    ]
].astype(
    float
)
```

In [22]:
```
df_complete["school_holiday_test_numerical"] = df_complete[
    "school_holiday_test"
].replace(["School", "School Holiday"], [0, 1])

df_complete["is_holiday_numerical"] = df_complete["Is Holiday"].astype(int)
```

In [23]:
```
df_corr = df_complete[
    [
        "Menge in Kilogramm (Menge)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
        "school_holiday_test_numerical",
        "is_holiday_numerical",
        "off_days",
        "count_per_month",
    ]
]

df_complete
```

Out[23]:

	date	Verkaufsdatum (Datum)	Biologische Gruppe der Art (biologische Gruppe)	Fao-Code fÃ¼r die Art (fao)	Galicischer Name der Art (Art)	Verkaufsprovinz (Provinz)	Administrativer Verkaufsbereich (Zone)
0	2014-01-02	2014-01-02	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa
1	2014-01-03	2014-01-03	Bivalvos	CLJ	Ameixa xaponesa	Pontevedra	Zona III - Arousa
2	2014-01-03	2014-01-03	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa
3	2014-01-04	2014-01-04	Bivalvos	CLJ	Ameixa xaponesa	Pontevedra	Zona III - Arousa
4	2014-01-04	2014-01-04	Bivalvos	CTG	Ameixa fina	Pontevedra	Zona III - Arousa
...	...	...	...	...	...	...	...
106483	2021-12-07	2021-12-07	Bivalvos	CTG	Ameixa fina	A Coruña	Zona IV - Muros
106484	2021-12-07	2021-12-07	Bivalvos	CTS	Ameixa babosa	A Coruña	Zona IV - Muros
106485	2021-12-08	2021-12-08	Bivalvos	CLJ	Ameixa xaponesa	A Coruña	Zona IV - Muros
106486	2021-12-08	2021-12-08	Bivalvos	CTG	Ameixa fina	A Coruña	Zona IV - Muros
106487	2021-12-08	2021-12-08	Bivalvos	CTS	Ameixa babosa	A Coruña	Zona IV - Muros

106488 rows × 26 columns

In [24]:
```
corr_matrix1 = df_corr.corr()
corr_matrix1["Menge in Kilogramm (Menge)"].sort_values(ascending=False)
```

Out[24]:
```
Menge in Kilogramm (Menge)                  1.000000
count_per_month                             0.046633
school_holiday_test_numerical               0.026730
is_holiday_numerical                       -0.005187
off_days                                   -0.005623
Durchschnittspreis in Euro pro Kilo (Preis)   -0.203900
Name: Menge in Kilogramm (Menge), dtype: float64
```

Importing and matching weather

In [25]:
```
stations = Stations()
cities = df_location["city"].unique()
cities = cities[0:10]
# # Create a new dataframe to store the results
```

```
df_weather = pd.DataFrame()
start = datetime(2014, 1, 1)
end = datetime(2021, 12, 31)
# Loop through each city
for city in cities:
    # Create a new dataframe for the city
    df_city = df_location[df_location["city"] == city]

    # Get the coordinates
    lat = df_city["latitude"].values[0]
    lon = df_city["longitude"].values[0]
    stations = stations.nearby(lat, lon)
    station = stations.fetch(1)
    station_id = station.wmo.values[0]
    # Create Point for the city

    # Get daily data for 2018
    data = Monthly(station_id, start, end)
    data = data.fetch()

    # Add the city name to the dataframe
    data["city"] = city

    # Concatenate the dataframes
    df_weather = pd.concat([df_weather, data])

df_weather.index = df_weather.index.strftime("%d/%m/%Y")
df_weather.reset_index(level=0, inplace=True)

# Rename the columns
df_weather["time"] = pd.to_datetime(df_weather["time"], format="%d/%m/%Y")
df_weather["year"] = df_weather["time"].dt.year
df_weather["month"] = df_weather["time"].dt.month_name()
df_weather = (
    df_weather.groupby(["year", "month"], as_index=False)[
        "tavg", "tmin", "tmax", "wspd", "pres"
    ]
    .mean()
    .round(2)
)
```

In [26]:

```
df_grouped_by_month = df_complete.groupby(["year", "month"], as_index=False).agg(
    {
        "Menge in Kilogramm (Menge)": "sum",
        "Einfuhr in Euro (Import)": "sum",
        "Durchschnittspreis in Euro pro Kilo (Preis)": "mean",
        "number_of_days_per_month": "max",
        "count_per_month": "max",
        "school_count_per_month": "max",
    }
)
df_grouped_by_month = df_grouped_by_month.merge(
    df_weather, on=["year", "month"], how="left"
)
```

In [27]:
```
df_grouped_by_year = df_grouped_by_month.groupby(["year"], as_index=True)[
    "Menge in Kilogramm (Menge)"
].sum()
df_grouped_by_year = pd.DataFrame(df_grouped_by_year)
df_grouped_by_year.rename(
    columns={"Menge in Kilogramm (Menge)": "yearly_amount"}, inplace=True
)
df_grouped_by_month = df_grouped_by_month.merge(
    df_grouped_by_year, on="year", how="left"
)
df_grouped_by_month
```

Out[27]:

	year	month	Menge in Kilogramm (Menge)	Einfuhr in Euro (Import)	Durchschnittspreis in Euro pro Kilo (Preis)	number_of_days_per_month	cou
0	2014	April	222156.750	2476421.330	13.418087	30	
1	2014	August	267323.040	2888197.210	13.268414	31	
2	2014	December	466791.750	6898154.430	16.967883	31	
3	2014	February	233735.340	2595329.090	12.771690	28	
4	2014	January	234987.770	1982688.040	9.881005	31	
...	...	...	...	...	...	...	
91	2021	March	207306.144	3187980.716	19.435464	31	
92	2021	May	164569.002	3024459.026	24.008054	31	
93	2021	November	221781.612	3804315.963	20.395409	30	
94	2021	October	230111.643	3624144.888	18.767272	31	
95	2021	September	219958.772	3926616.208	22.645230	30	

96 rows × 14 columns

price index

In [28]:
```
df_grouped_by_year_price = df_grouped_by_month.groupby(["year"], as_index=True)[
    "Durchschnittspreis in Euro pro Kilo (Preis)"
].max()
df_grouped_by_year_price = pd.DataFrame(df_grouped_by_year_price)
df_grouped_by_year_price = df_grouped_by_year_price.rename(
    columns={"Durchschnittspreis in Euro pro Kilo (Preis)": "max_price_month_year"}
)
df_grouped_by_month = df_grouped_by_month.merge(
    df_grouped_by_year_price, on="year", how="left"
)
df_grouped_by_month["price_deviation"] = (
    df_grouped_by_month["Durchschnittspreis in Euro pro Kilo (Preis)"]
    / df_grouped_by_month["max_price_month_year"]
) * 100
df_grouped_by_month.to_csv("wetter.csv")
df_price_total_aggregate = df_grouped_by_month.groupby("month", as_index=False)[
```

```
    "Durchschnittspreis in Euro pro Kilo (Preis)"
].mean()
df_price_total_aggregate["max_price_total"] = df_price_total_aggregate[
    "Durchschnittspreis in Euro pro Kilo (Preis)"
].max()
df_price_total_aggregate["price_deviation"] = (
    df_price_total_aggregate["Durchschnittspreis in Euro pro Kilo (Preis)"]
    / df_price_total_aggregate["max_price_total"]
) * 100
df_price_total_aggregate.head(5)
```

Out[28]:

	month	Durchschnittspreis in Euro pro Kilo (Preis)	max_price_total	price_deviation
0	April	16.056234	19.52712	82.225307
1	August	16.578413	19.52712	84.899426
2	December	19.527120	19.52712	100.000000
3	February	15.787169	19.52712	80.847398
4	January	14.245430	19.52712	72.952028

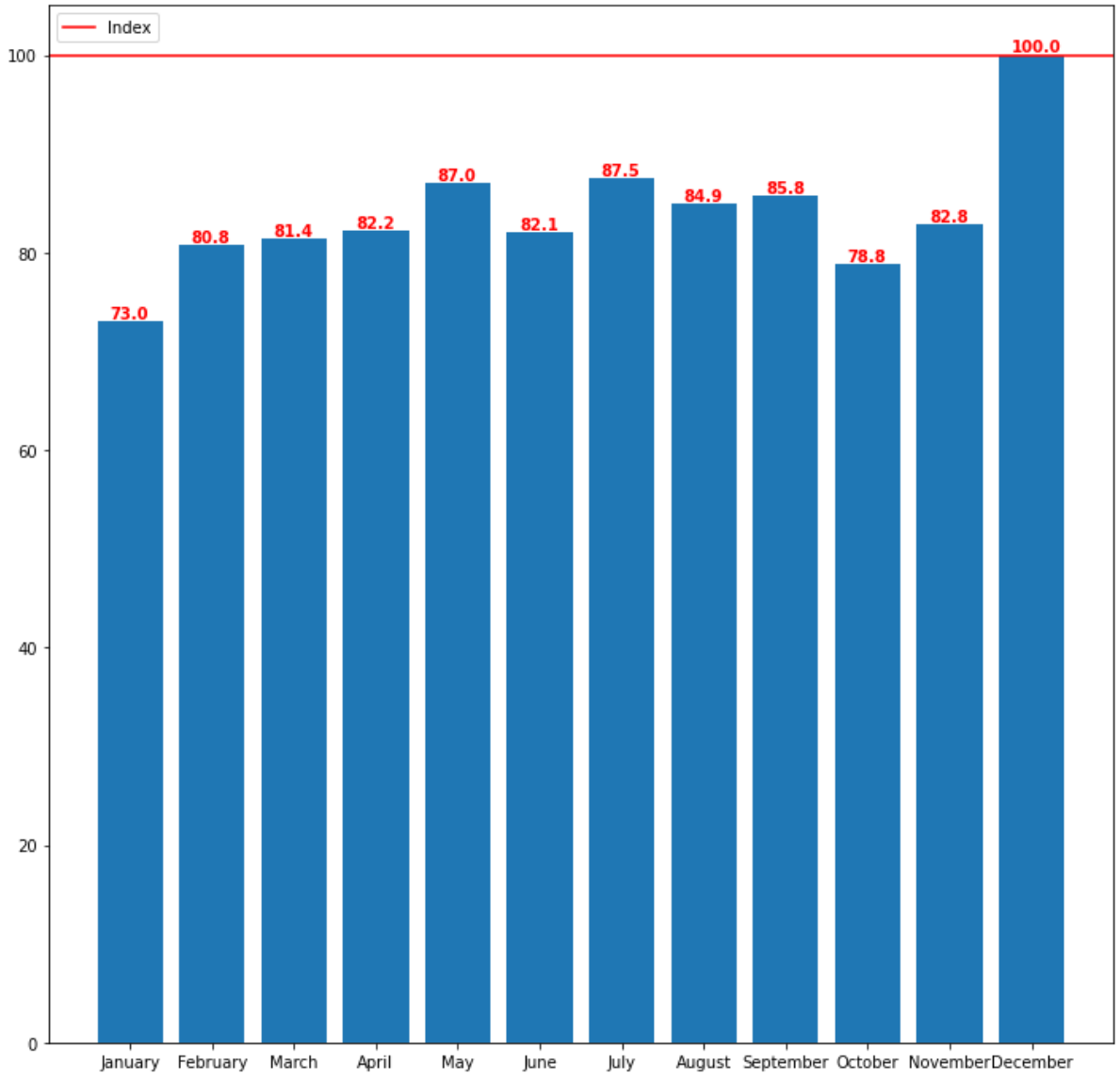
In [29]:

```
def month_string_to_number(string):
    m = {
        "jan": 1,
        "feb": 2,
        "mar": 3,
        "apr": 4,
        "may": 5,
        "jun": 6,
        "jul": 7,
        "aug": 8,
        "sep": 9,
        "oct": 10,
        "nov": 11,
        "dec": 12,
    }
    s = string.strip()[:3].lower()

    try:
        out = m[s]
        return out
    except:
        raise ValueError("Not a month")
```

In [30]:

```
total_price_deviation = df_price_total_aggregate
total_price_deviation["month_number"] = total_price_deviation["month"].apply(
    month_string_to_number
)
total_price_deviation.sort_values(by="month_number", ascending=True, inplace=True)
fig, ax = plt.subplots(figsize=(12, 12))
total_price_deviation["price_deviation"] = total_price_deviation[
    "price_deviation"
].round(1)
total_price_jonny = total_price_deviation
```

```
for i, v in enumerate(total_price_deviation["price_deviation"]):
    plt.text(i - 0.25, v + 0.3, str(v), color="red", fontweight="bold")
plt.bar(total_price_deviation["month"], total_price_deviation["price_deviation"])
plt.axhline(y=100, color="r", label="Index")
plt.legend(loc="upper left")
plt.show()
```



Wind Speed

In [31]:
```
stations = Stations()
cities = df_location["city"].unique()
cities = cities[0:10]
# # Create a new dataframe to store the results
df_wind = pd.DataFrame()
start = datetime(2014, 1, 1)
end = datetime(2022, 11, 30)
# Loop through each city
for city in cities:
    # Create a new dataframe for the city
    df_city = df_location[df_location["city"] == city]
```

```
    # Get the coordinates
    lat = df_city["latitude"].values[0]
    lon = df_city["longitude"].values[0]
    stations = stations.nearby(lat, lon)
    station = stations.fetch(1)
    station_id = station.wmo.values[0]
    # Create Point for the city

    # Get daily data for 2018
    data = Daily(station_id, start, end)
    data = data.fetch()

    # Add the city name to the dataframe
    data["city"] = city

    # Concatenate the dataframes
    df_wind = pd.concat([df_wind, data])

df_wind.index = df_wind.index.strftime("%d/%m/%Y")
df_wind.reset_index(level=0, inplace=True)

# Rename the columns
df_wind["time"] = pd.to_datetime(df_wind["time"], format="%d/%m/%Y")
df_wind["year"] = df_wind["time"].dt.year
df_wind["month"] = df_wind["time"].dt.month_name()


df_wind["count_wind"] = df_wind["wpgt"] > 37
df_wind["count"] = df_wind["wpgt"] > 0

df_wind["month_number"] = df_wind["month"].apply(month_string_to_number)
df_wind.sort_values(by="month_number", ascending=True, inplace=True)

df_wind_daily_agg = df_wind.groupby("time", as_index=False).agg(
    {"wspd": "mean", "wpgt": "max"}
)

df_wind_daily_agg
```

Out[31]:

	time	wspd	wpgt
0	2014-01-01	NaN	NaN
1	2014-01-02	NaN	NaN
2	2014-01-03	NaN	NaN
3	2014-01-04	NaN	NaN
4	2014-01-05	NaN	NaN
...	...	...	...
3251	2022-11-26	10.97	38.9
3252	2022-11-27	11.70	38.9
3253	2022-11-28	8.18	37.0
3254	2022-11-29	6.28	20.4
3255	2022-11-30	11.58	35.2

3256 rows × 3 columns

In [32]:

```
df_complete_daily = (
    df_complete[df_complete["Galicischer Name der Art (Art)"] == "Ameixa babosa"]
    .groupby("date", as_index=False)
    .agg("mean")
)
df_complete_daily
```

Out[32]:

	date	Menge in Kilogramm (Menge)	Einfuhr in Euro (Import)	Durchschnittspreis in Euro pro Kilo (Preis)	year	number_of_days_per_month	Hol
0	2014-01-04	193.306522	1890.358696	10.986522	2014.0	31.0	
1	2014-01-05	1.000000	11.350000	11.350000	2014.0	31.0	
2	2014-01-07	265.236111	3129.640000	12.683333	2014.0	31.0	
3	2014-01-08	290.670000	3499.718000	12.223000	2014.0	31.0	
4	2014-01-09	184.086000	2183.230667	12.518000	2014.0	31.0	
...	...	...	...	...	...	...	
2083	2021-12-27	56.093200	1368.923000	20.416000	2021.0	31.0	
2084	2021-12-28	64.110588	1542.217824	20.227059	2021.0	31.0	
2085	2021-12-29	89.348235	2243.371176	21.975882	2021.0	31.0	
2086	2021-12-30	114.180333	2652.653667	21.187778	2021.0	31.0	
2087	2021-12-31	36.420000	522.800000	14.872500	2021.0	31.0	

2088 rows × 16 columns

In [33]:
```
## harvesting ban

df_harvesting_bans = pd.read_csv(
    "gdrive/MyDrive/arima_prediction/harvesting_ban.csv", on_bad_lines="skip", sep=
)

df_harvesting_bans["harvest_restricted"] = df_harvesting_bans["harvesting ban "]

df_harvesting_bans["date"] = pd.to_datetime(df_harvesting_bans["date"])

df_harversting_price = df_harvesting_bans.merge(df_complete_daily, how="left")

df_harversting_price = df_harversting_price.reindex(
    columns=[
        "date",
        "harvest_restricted",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
        "Menge in Kilogramm (Menge)",
    ]
)
```

```
df_harversting_price = df_harversting_price.merge(
    df_wind_daily_agg, left_on="date", right_on="time", how="left"
)

df_harversting_price["Durchschnittspreis in Euro pro Kilo (Preis)"].interpolate(
    method="linear", inplace=True
)
df_harversting_price["Menge in Kilogramm (Menge)"].interpolate(
    method="linear", inplace=True
)

df_harversting_price["rolling_price"] = (
    df_harversting_price["Durchschnittspreis in Euro pro Kilo (Preis)"]
    .rolling(7, center=True)
    .mean()
)
df_harversting_price["rolling_volume"] = (
    df_harversting_price["Menge in Kilogramm (Menge)"].rolling(7, center=True).mean
)
df_harversting_price["rolling_wspd"] = (
    df_harversting_price["wspd"].rolling(7, center=True).mean()
)
df_harversting_price["rolling_wpgt"] = (
    df_harversting_price["wpgt"].rolling(7, center=True).mean()
)

df_harversting_price["wind_restricted"] = df_harversting_price["rolling_wpgt"] > 40

df_harversting_price["total_restricted"] = (
    df_harversting_price["wind_restricted"] | df_harvesting_bans["harvest_restricte
)

df_harversting_price.set_index("date", inplace=True)

df_harversting_price.index.sort_values()
df_harversting_price
```

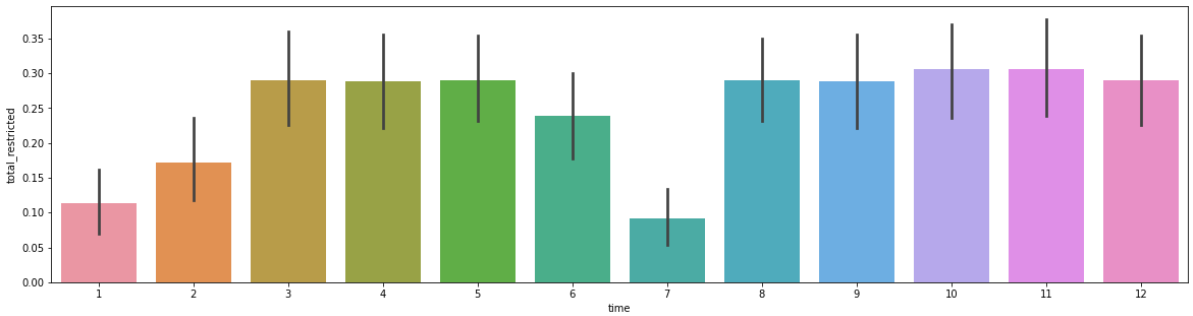
Out[33]:

date	harvest_restricted	Durchschnittspreis in Euro pro Kilo (Preis)	Menge in Kilogramm (Menge)	time	wspd	wpgt	rolling_price	rolling
2014-01-01	0	NaN	NaN	2014-01-01	NaN	NaN	NaN	
2014-02-01	0	13.773043	529.495652	2014-02-01	NaN	NaN	NaN	
2014-03-01	0	10.686818	365.631818	2014-03-01	NaN	NaN	NaN	
2014-04-01	0	10.686721	304.321185	2014-04-01	NaN	NaN	NaN	
2014-05-01	0	10.686623	243.010552	2014-05-01	NaN	NaN	11.111492	26
...	...	...	...	...	...	...	...	
2019-12-27	0	27.195909	459.853182	2019-12-27	5.750	18.5	23.721260	26
2019-12-28	0	17.105000	40.085000	2019-12-28	6.850	20.4	23.334617	26
2019-12-29	0	21.475227	195.429091	2019-12-29	6.450	18.5	NaN	
2019-12-30	0	25.845455	350.773182	2019-12-30	6.700	16.7	NaN	
2019-12-31	0	19.506000	88.756000	2019-12-31	5.825	16.7	NaN	

2191 rows × 12 columns

In [34]:
```
data = df_harversting_price.copy()
data.month = data.time.dt.month

fig, ax = plt.subplots(figsize=(20, 5))
sns.barplot(x=data.month, y=data.total_restricted, ax=ax)
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4458746310>
```



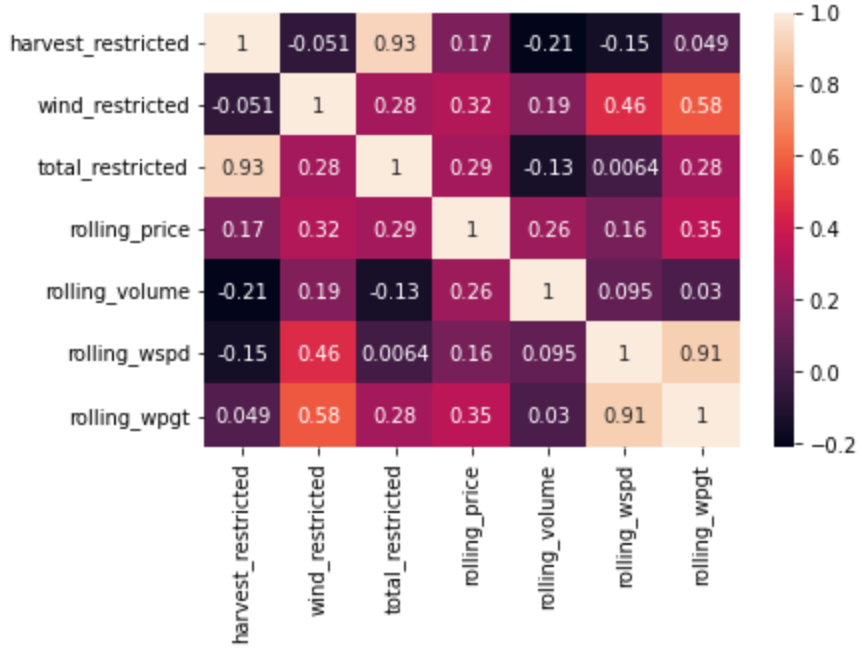
In [35]:
```
df_harversting_price = df_harversting_price.loc["2018-02-01":"2019-12-27"]
```

```
corr_matrix = df_harversting_price[
    [
        "harvest_restricted",
        "wind_restricted",
        "total_restricted",
        "rolling_price",
        "rolling_volume",
        "rolling_wspd",
        "rolling_wpgt",
    ]
].corr()
corr_matrix
```

Out[35]:

	harvest_restricted	wind_restricted	total_restricted	rolling_price	rolling_volume
harvest_restricted	1.000000	-0.050825	0.925625	0.174567	-0.209929
wind_restricted	-0.050825	1.000000	0.280968	0.323011	0.191704
total_restricted	0.925625	0.280968	1.000000	0.285030	-0.131063
rolling_price	0.174567	0.323011	0.285030	1.000000	0.262475
rolling_volume	-0.209929	0.191704	-0.131063	0.262475	1.000000
rolling_wspd	-0.148393	0.460837	0.006386	0.156900	0.095218
rolling_wpgt	0.048624	0.578819	0.284445	0.348324	0.030166

In [37]:
```
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



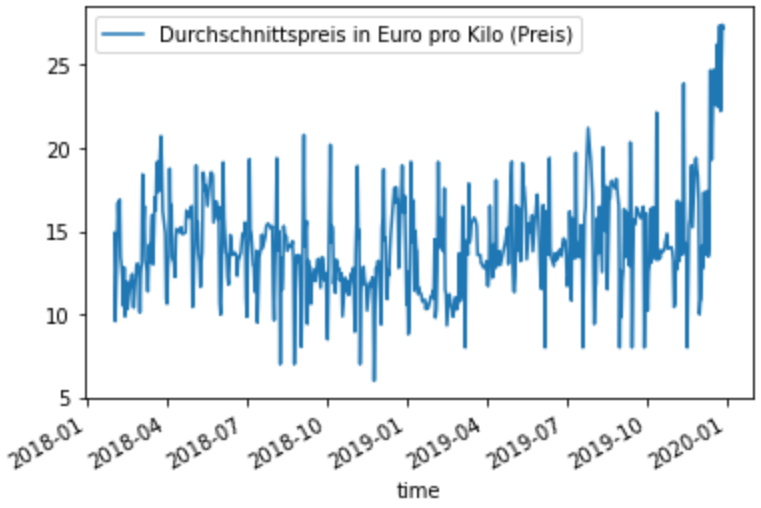
In [38]:
```
df_test = df_harversting_price[["total_restricted", "rolling_price"]]

# calculate the correlation between the columns "rolling_price" and "total_restrict
corr, p_value = pearsonr(df_test["total_restricted"], df_test["rolling_price"])
```

```
# print the results
print(
    "The correlation between the columns 'rolling_price' and 'total_restricted' is
        corr
    )
)
print("The p-value is {}".format(p_value))

# if the p-value is smaller than the significance level (0.05), then the null hypot
if p_value < 0.05:
    print(
        "The null hypothesis is rejected, there is a correlation between the column
        'rolling_price' and 'total_restricted'"
    )
else:
    print(
        "The null hypothesis is not rejected, there is no correlation between the c
        'rolling_price' and 'total_restricted'"
    )
```

```
The correlation between the columns 'rolling_price' and 'total_restricted' is 0.28
50303698554005
The p-value is 1.8572339114664087e-14
The null hypothesis is rejected, there is a correlation between the columns 'rolli
ng_price' and 'total_restricted'
```

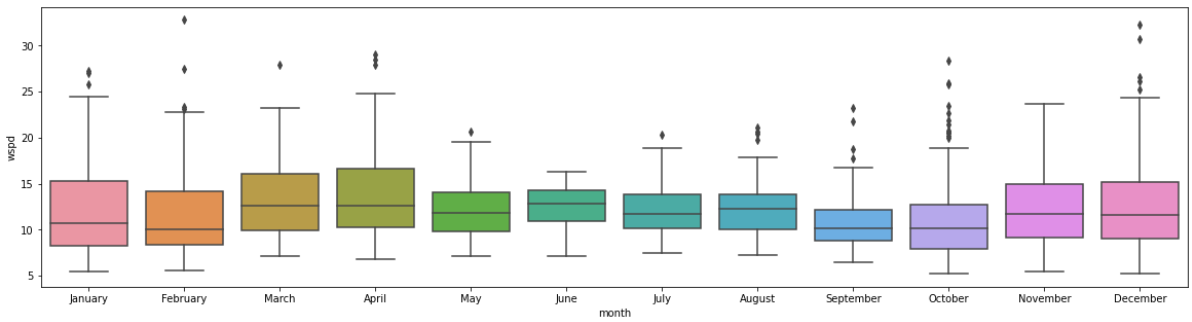
In [39]:
```
# lines = df_harversting_price.plot.line(x='time', y='Durchschnittspreis in Euro pr
lines = df_harversting_price.plot.line(
    x="time", y="Durchschnittspreis in Euro pro Kilo (Preis)"
)
lines
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4450523880>
```



In [40]:
```
# Get the coordinates
lat = 37.08027778
lon = -8.61111111
stations = stations.nearby(lat, lon)
station = stations.fetch(1)
station_id = station.wmo.values[0]
```

```
# Create Point for the city

# Get daily data for 2018

data = Daily(station_id, start, end)
data = data.fetch()

data

data.index = data.index.strftime("%d/%m/%Y")
data.reset_index(level=0, inplace=True)

# Rename the columns
data["time"] = pd.to_datetime(data["time"], format="%d/%m/%Y")
data["year"] = data["time"].dt.year
data["month"] = data["time"].dt.month_name()


data["count_wind"] = data["wpgt"] > 37
data["count"] = data["wpgt"] > 0

data["month_number"] = data["month"].apply(month_string_to_number)
data.sort_values(by="month_number", ascending=True, inplace=True)
fig, ax = plt.subplots(figsize=(20, 5))
sns.boxplot(x=data.month, y=data.wspd, ax=ax)

fig.savefig("sagres.png")
```



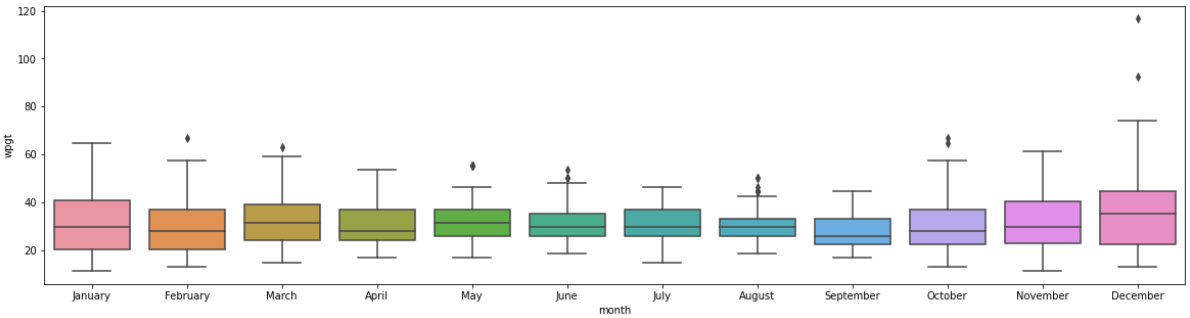
In [41]:
```
data.sort_values("time")
```

Out[41]:

	time	tavg	tmin	tmax	prcp	snow	wdir	wspd	wpgt	pres	tsun	year	month
0	2018-07-21	19.9	16.4	23.0	NaN	NaN	313.0	20.3	44.5	1016.9	NaN	2018	July
1	2018-07-22	20.0	16.8	23.0	NaN	NaN	313.0	18.9	40.8	1015.0	NaN	2018	July
2	2018-07-23	19.8	16.5	22.9	NaN	NaN	309.0	15.6	37.0	1016.6	NaN	2018	July
3	2018-07-24	19.8	16.4	22.7	NaN	NaN	300.0	14.9	37.0	1017.6	NaN	2018	July
4	2018-07-25	19.7	16.6	22.5	NaN	NaN	303.0	16.2	38.9	1016.8	NaN	2018	July
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1573	2022-11-26	16.1	13.3	19.9	NaN	NaN	58.0	7.6	20.4	1026.9	NaN	2022	November
1574	2022-11-27	16.3	14.0	19.4	NaN	NaN	4.0	6.8	25.9	1025.2	NaN	2022	November
1575	2022-11-28	15.7	13.5	18.0	NaN	NaN	353.0	11.7	31.5	1026.8	NaN	2022	November
1576	2022-11-29	14.6	12.9	17.4	NaN	NaN	3.0	10.0	25.9	1023.3	NaN	2022	November
1577	2022-11-30	15.3	12.7	17.2	NaN	NaN	286.0	10.5	29.6	1016.9	NaN	2022	November

1578 rows × 16 columns

In [42]:
```
fig, ax = plt.subplots(figsize=(20, 5))
sns.boxplot(x=df_wind.month, y=df_wind.wpgt, ax=ax)

fig.savefig("output.png")
```



In [43]:
```
df_wind = df_wind.groupby("month").agg("sum")

df_wind["availble_days"] = 1 - df_wind["count_wind"] / df_wind["count"]

df_wind.sort_values(by="month_number", ascending=True, inplace=True)
```

number of clams index

In [44]:
```
df_grouped_by_month["Menge in Kilogramm (Menge)_monthly"] = (
    df_grouped_by_month["yearly_amount"] / 12
)

df_grouped_by_month["Menge in Kilogramm (Menge)_deviation"] = (
    df_grouped_by_month["Menge in Kilogramm (Menge)"]
    / df_grouped_by_month["Menge in Kilogramm (Menge)_monthly"]
)

df_grouped_by_month["Menge in Kilogramm (Menge)_deviation"] = (
    df_grouped_by_month["Menge in Kilogramm (Menge)_deviation"] * 100
)

df_grouped_by_month["Menge in Kilogramm (Menge)_deviation"] = df_grouped_by_month[
    "Menge in Kilogramm (Menge)_deviation"
].round(2)
df_grouped_by_month
```

Out[44]:

	year	month	Menge in Kilogramm (Menge)	Einfuhr in Euro (Import)	Durchschnittspreis in Euro pro Kilo (Preis)	number_of_days_per_month	cou
0	2014	April	222156.750	2476421.330	13.418087	30	
1	2014	August	267323.040	2888197.210	13.268414	31	
2	2014	December	466791.750	6898154.430	16.967883	31	
3	2014	February	233735.340	2595329.090	12.771690	28	
4	2014	January	234987.770	1982688.040	9.881005	31	
...	...	...	...	...	...	...	
91	2021	March	207306.144	3187980.716	19.435464	31	
92	2021	May	164569.002	3024459.026	24.008054	31	
93	2021	November	221781.612	3804315.963	20.395409	30	
94	2021	October	230111.643	3624144.888	18.767272	31	
95	2021	September	219958.772	3926616.208	22.645230	30	

96 rows × 18 columns

In [45]:
```
df_grouped_by_yearly_average = df_grouped_by_month.groupby(["month"], as_index=Fals
    "Menge in Kilogramm (Menge)_deviation"
].mean()

df_grouped_by_yearly_average.rename(
    columns={"Menge in Kilogramm (Menge)_deviation": "Menge_in_Kilogramm_Index"},
    inplace=True,
)
df_grouped_by_month = df_grouped_by_month.merge(
    df_grouped_by_yearly_average, on="month", how="left"
```

```
)
df_grouped_by_month
```

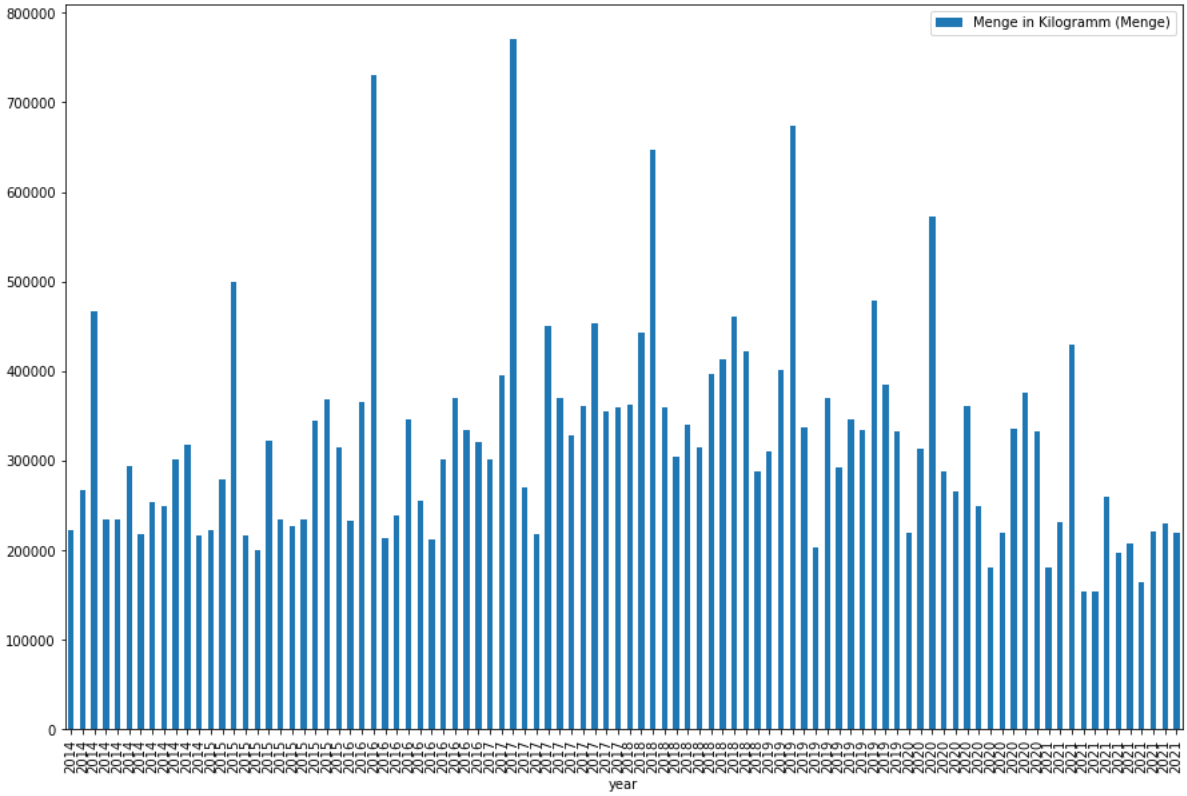
Out[45]:

	year	month	Menge in Kilogramm (Menge)	Einfuhr in Euro (Import)	Durchschnittspreis in Euro pro Kilo (Preis)	number_of_days_per_month	cou
0	2014	April	222156.750	2476421.330	13.418087	30	
1	2014	August	267323.040	2888197.210	13.268414	31	
2	2014	December	466791.750	6898154.430	16.967883	31	
3	2014	February	233735.340	2595329.090	12.771690	28	
4	2014	January	234987.770	1982688.040	9.881005	31	
...	...	...	...	...	...	...	
91	2021	March	207306.144	3187980.716	19.435464	31	
92	2021	May	164569.002	3024459.026	24.008054	31	
93	2021	November	221781.612	3804315.963	20.395409	30	
94	2021	October	230111.643	3624144.888	18.767272	31	
95	2021	September	219958.772	3926616.208	22.645230	30	

96 rows × 19 columns

In [46]:
```
df_grouped_by_month_test = df_grouped_by_month[
    ["year", "month", "Menge in Kilogramm (Menge)_deviation"]
]

df_grouped_by_month.plot(
    x="year", y="Menge in Kilogramm (Menge)", kind="bar", figsize=(15, 10)
)
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4451027340>
```



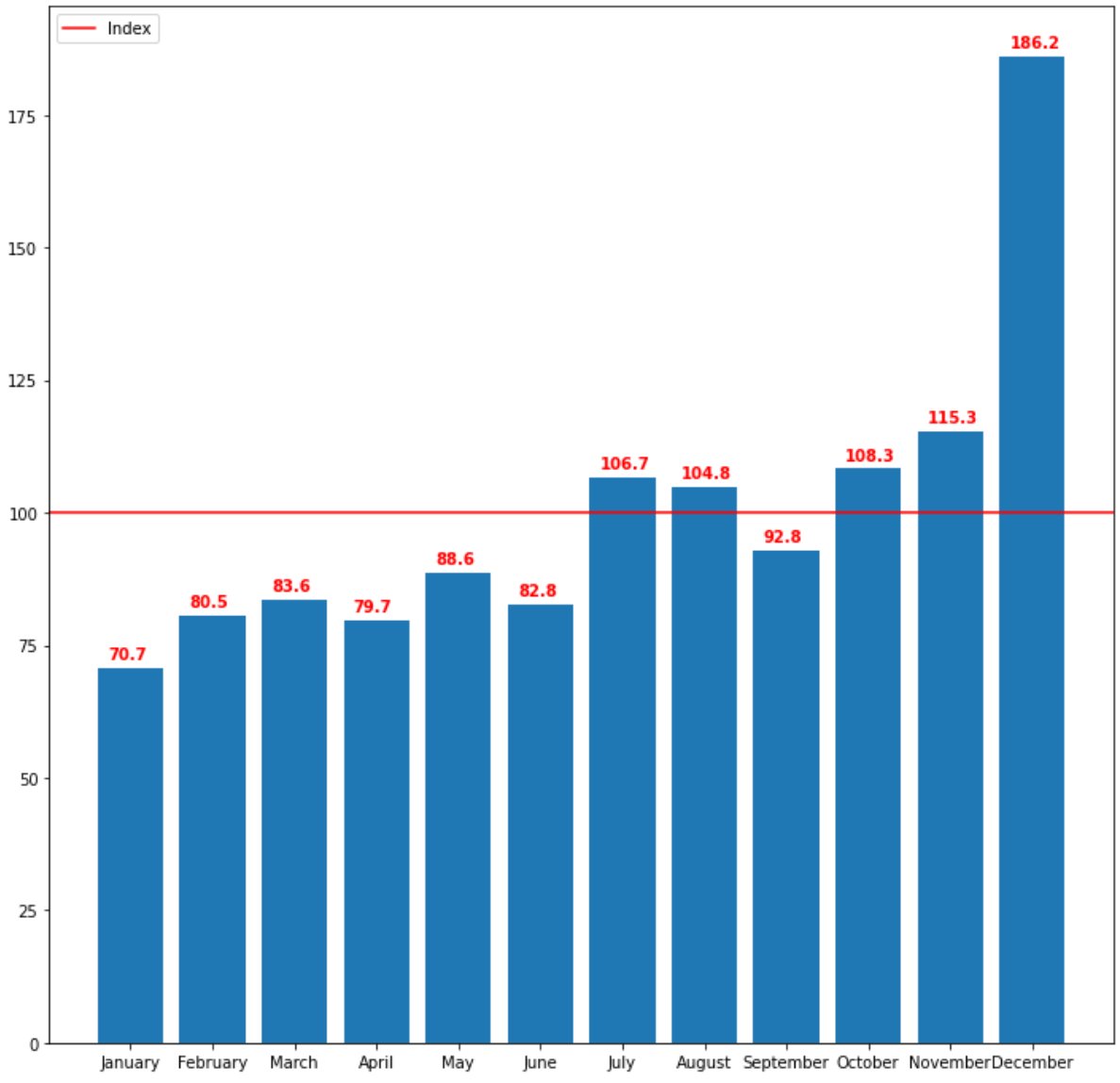
In [47]:
```
df_quantity_total_aggregate = df_grouped_by_month.groupby("month", as_index=False)[
    "Menge in Kilogramm (Menge)"
].mean()
df_quantity_total_aggregate["mean_quantity"] = df_quantity_total_aggregate[
    "Menge in Kilogramm (Menge)"
].mean()
df_quantity_total_aggregate["quantity deviation"] = (
    df_quantity_total_aggregate["Menge in Kilogramm (Menge)"]
    / df_quantity_total_aggregate["mean_quantity"]
) * 100
df_quantity_total_aggregate.head(5)
```

Out[47]:

	month	Menge in Kilogramm (Menge)	mean_quantity	quantity deviation
0	April	256290.333125	321582.743906	79.696544
1	August	337147.350375	321582.743906	104.840001
2	December	598705.798000	321582.743906	186.174728
3	February	258925.033750	321582.743906	80.515836
4	January	227316.138750	321582.743906	70.686672

In [48]:
```
year_2021_deviation = df_quantity_total_aggregate
year_2021_deviation["month_number"] = year_2021_deviation["month"].apply(
    month_string_to_number
)
year_2021_deviation.sort_values(by="month_number", ascending=True, inplace=True)
fig, ax = plt.subplots(figsize=(12, 12))
year_2021_deviation["quantity deviation"] = year_2021_deviation[
```

```
    "quantity deviation"
].round(1)

for i, v in enumerate(year_2021_deviation["quantity deviation"]):
    plt.text(i - 0.28, v + 1.7, str(v), color="red", fontweight="bold")
plt.bar(year_2021_deviation["month"], year_2021_deviation["quantity deviation"])
plt.axhline(y=100, color="r", label="Index")
plt.legend(loc="upper left")
plt.show()
```



In [49]:
```
df_grouped_by_month_test = df_grouped_by_month_test.sort_values(by=["year", "month"
```

In [50]:
```
df_grouped_by_month["month_number"] = df_grouped_by_month["month"].apply(
    month_string_to_number
)
df_grouped_by_month.sort_values(
    by=["year", "month_number"], ascending=True, inplace=True
)
```

In [51]:
```
df_arima_index_test = df_grouped_by_month[
    [
```

```
        "year",
        "month",
        "tavg",
        "count_per_month",
        "school_count_per_month",
        "Menge in Kilogramm (Menge)_deviation",
    ]
]
```

In [52]:
```
## std dev by month niklas

df_grouped_by_month[["Menge in Kilogramm (Menge)_deviation", "month"]].groupby(
    "month"
).std()
```

Out[52]:

month	Menge in Kilogramm (Menge)_deviation
April	6.701752
August	5.802891
December	19.194079
February	11.119956
January	11.767543
July	10.858839
June	6.261922
March	14.994299
May	11.025354
November	8.359072
October	11.668229
September	12.737290

T-Test Correlation

In [53]:
```
df_monthly_correlation = df_grouped_by_month[
    [
        "Menge in Kilogramm (Menge)",
        "Durchschnittspreis in Euro pro Kilo (Preis)",
        "count_per_month",
        "school_count_per_month",
        "tavg",
        "tmin",
        "tmax",
        "wspd",
        "pres",
        "Menge_in_Kilogramm_Index",
    ]
```

```
]
corr_matrix1 = df_monthly_correlation.corr()
corr_matrix1["Menge in Kilogramm (Menge)"].sort_values(ascending=False)
```

Out[53]:
```
Menge in Kilogramm (Menge)                    1.000000
Menge_in_Kilogramm_Index                      0.805064
pres                                          0.335361
count_per_month                               0.303224
Durchschnittspreis in Euro pro Kilo (Preis)   0.232016
wspd                                          0.203889
school_count_per_month                        0.200396
tmin                                         -0.089100
tavg                                         -0.102387
tmax                                         -0.121426
Name: Menge in Kilogramm (Menge), dtype: float64
```

In [54]:
```
corr_matrix1["Durchschnittspreis in Euro pro Kilo (Preis)"].sort_values(ascending=F
```

Out[54]:
```
Durchschnittspreis in Euro pro Kilo (Preis)   1.000000
Menge_in_Kilogramm_Index                      0.390387
Menge in Kilogramm (Menge)                    0.232016
school_count_per_month                        0.200250
wspd                                          0.129839
count_per_month                               0.126092
pres                                          0.111167
tavg                                          0.078963
tmin                                          0.076045
tmax                                          0.060510
Name: Durchschnittspreis in Euro pro Kilo (Preis), dtype: float64
```

In [55]:
```
df_test = df_monthly_correlation[["Menge in Kilogramm (Menge)", "count_per_month"]]

# calculate the correlation between the columns "Menge in Kilogramm (Menge)" and "c
corr, p_value = pearsonr(
    df_test["Menge in Kilogramm (Menge)"], df_test["count_per_month"]
)

# print the results
print(
    "The correlation between the columns 'Menge in Kilogramm (Menge)' and/
    'count_per_month' is {}".format(
        corr
    )
)
print("The p-value is {}".format(p_value))

# if the p-value is smaller than the significance level (0.05), then the null hypot
if p_value < 0.05:
    print(
        "The null hypothesis is rejected, there is a correlation between the column
        'Menge in Kilogramm (Menge)' and 'count_of_off_days_per_month'"
    )
else:
    print(
        "The null hypothesis is not rejected, there is no correlation between the c
```

```
        'Menge in Kilogramm (Menge)' and 'count_per_month'"
    )
```

```
The correlation between the columns 'Menge in Kilogramm (Menge)' and 'count_per_mo
nth' is 0.3032242590927613
The p-value is 0.002672434821591531
The null hypothesis is rejected, there is a correlation between the columns 'Menge
in Kilogramm (Menge)' and 'count_of_off_days_per_month'
```

In [56]:

```
df_test = df_monthly_correlation[
    ["Menge in Kilogramm (Menge)", "school_count_per_month"]
]

# calculate the correlation between the columns "Menge in Kilogramm (Menge)" and "c
corr, p_value = pearsonr(
    df_test["Menge in Kilogramm (Menge)"], df_test["school_count_per_month"]
)

# print the results
print(
    "The correlation between the columns 'Menge in Kilogramm (Menge)' and /
    'school_count_per_month' is {}".format(
        corr
    )
)
print("The p-value is {}".format(p_value))

# if the p-value is smaller than the significance level (0.05), then the null hypot
if p_value < 0.05:
    print(
        "The null hypothesis is rejected, there is a correlation between the column
        'Menge in Kilogramm (Menge)' and 'school_count_per_month'"
    )
else:
    print(
        "The null hypothesis is not rejected, there is no correlation between the c
        'Menge in Kilogramm (Menge)' and 'school_count_per_month'"
    )
```

```
The correlation between the columns 'Menge in Kilogramm (Menge)' and 'school_count
_per_month' is 0.20039616688664494
The p-value is 0.05026994700501754
The null hypothesis is not rejected, there is no correlation between the columns
'Menge in Kilogramm (Menge)' and 'school_count_per_month'
```

In [57]:

```
df_test = df_monthly_correlation[["Menge in Kilogramm (Menge)", "tavg"]]

df_test = df_test.fillna(df_test.mean())
# calculate the correlation between the columns "Menge in Kilogramm (Menge)" and "c
corr, p_value = pearsonr(df_test["Menge in Kilogramm (Menge)"], df_test["tavg"])

# print the results
print(
    "The correlation between the columns 'Menge in Kilogramm (Menge)' and 'tavg' is
        corr
    )
)
```

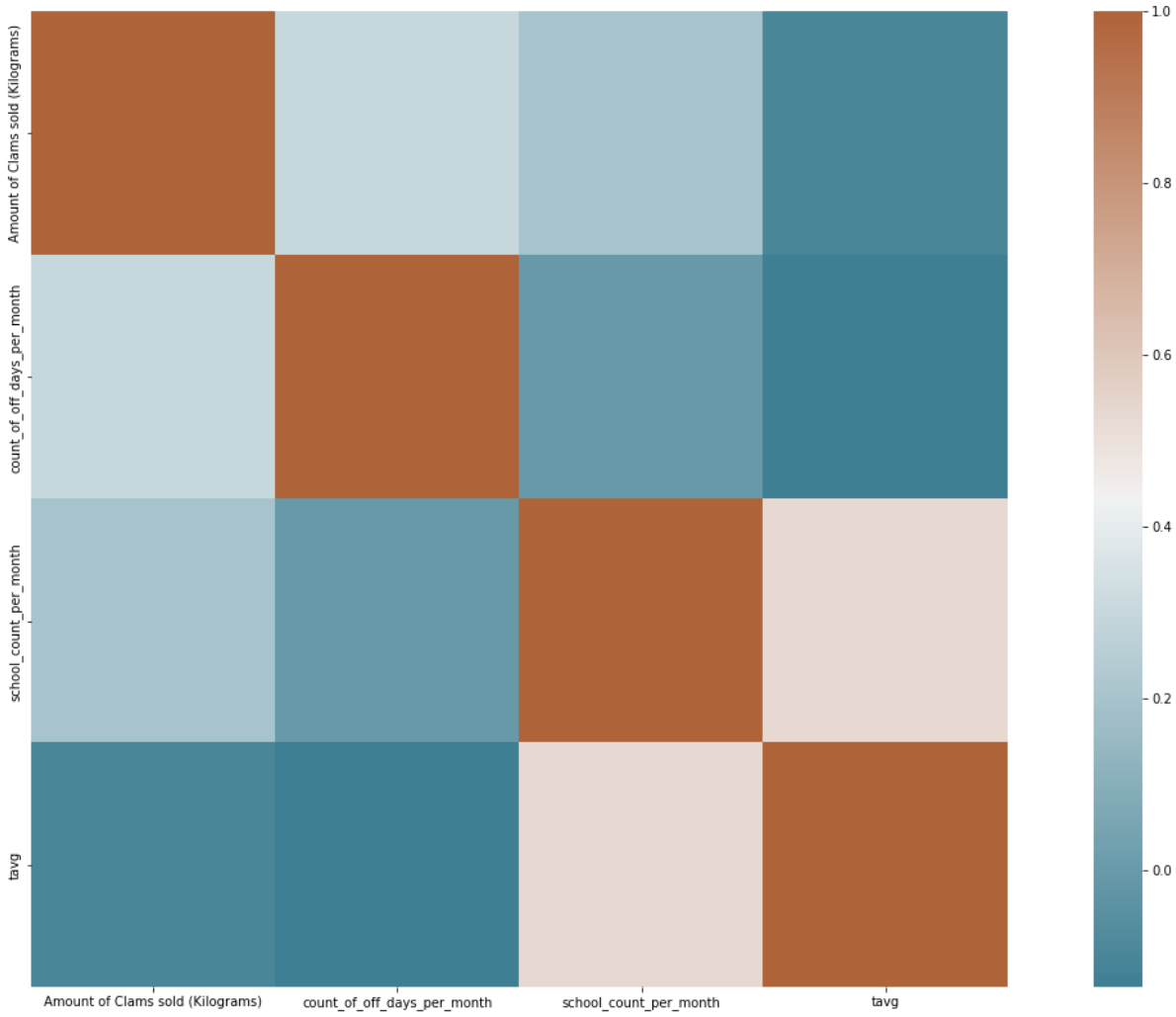
```
print("The p-value is {}".format(p_value))

# if the p-value is smaller than the significance level (0.05), then the null hypot
if p_value < 0.05:
    print(
        "The null hypothesis is rejected, there is a correlation between the column
        'Menge in Kilogramm (Menge)' and 'tavg'"
    )
else:
    print(
        "The null hypothesis is not rejected, there is no correlation between the c
        'Menge in Kilogramm (Menge)' and 'tavg'"
    )
```

```
The correlation between the columns 'Menge in Kilogramm (Menge)' and 'tavg' is -0.
10238654259032748
The p-value is 0.320880725956101
The null hypothesis is not rejected, there is no correlation between the columns
'Menge in Kilogramm (Menge)' and 'tavg'
```

In [58]:
```
df_monthly_correlation_min = df_grouped_by_month[
    ["Menge in Kilogramm (Menge)", "count_per_month", "school_count_per_month", "ta
]
```

In [59]:
```
df_monthly_correlation_min = df_monthly_correlation_min.rename(
    columns={
        "Menge in Kilogramm (Menge)": "Amount of Clams sold (Kilograms)",
        "count_per_month": "count_of_off_days_per_month",
    }
)
```

In [60]:
```
f, ax = plt.subplots(figsize=(24, 14))
corr = df_monthly_correlation_min.corr()
sns.heatmap(
    corr,
    mask=np.zeros_like(corr, dtype=np.bool),
    cmap=sns.diverging_palette(220, 30, as_cmap=True),
    square=True,
    ax=ax,
)
sns.set(font_scale=1.2)
```



Arima

chapter test_deliverable2

In [61]:
```
df = df_arima_index_test[
    [
        "year",
        "month",
        "Menge in Kilogramm (Menge)_deviation",
        "count_per_month",
        "school_count_per_month",
        "tavg",
    ]
]
df["date"] = df["year"].astype(str) + "-" + df["month"].astype(str)
df["date"] = pd.to_datetime(df["date"])
df = df.drop(["year", "month"], axis=1)
df["count_per_month"] = df["count_per_month"].astype(float)
df["school_count_per_month"] = df["school_count_per_month"].astype(float)
df["tavg"] = df["tavg"].astype(float)
df["date"] = df["date"].astype(str)

df_clam = df[
```

```
    [
        "date",
        "Menge in Kilogramm (Menge)_deviation",
        "count_per_month",
        "school_count_per_month",
        "tavg",
    ]
]
df_clam = df_clam.rename(
    columns={
        "Menge in Kilogramm (Menge)_deviation": "Quantity sold in KG",
        "count_per_month": "off_days_per_month",
    }
)
df_clam = df_clam.sort_values(by="date")
df_clam = df_clam.set_index("date")
df_clam.index = pd.DatetimeIndex(df_clam.index).to_period("M")
df_clam.index = df_clam.index.to_timestamp()
df_clam[
    ["Quantity sold in KG", "off_days_per_month", "school_count_per_month", "tavg"]
].plot(subplots=True, figsize=(15, 10))
```

Out[61]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f44503ac2b0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f44511e9940>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f44512cb040>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f44512c6580>],
      dtype=object)
```

Arima Split Validation

In [62]:
```
pred_periods = 30
# corresponds to a prediction horizion  of 2,5 years
```

```
split_number = df_clam["Quantity sold in KG"].count() - pred_periods

exo_columns = ["off_days_per_month", "school_count_per_month", "tavg"]

# define the list of possible combinations of exogenous variables to be used in the
exo_combinations = []
for i in range(1, len(exo_columns) + 1):
    exo_combinations.extend(list(itertools.combinations(exo_columns, i)))

# define the parameters to be used in the model
max_p = 5
max_d = 5
max_q = 5
seasonal = True
m = 12
max_P = 5
max_D = 5
max_Q = 5
trace = True
error_action = "ignore"
suppress_warnings = True
stepwise = True
n_fits = 30
method = "bfgs"


df_train = pd.DataFrame(df_clam["Quantity sold in KG"][:split_number]).rename(
    columns={"Quantity sold in KG": "y_train"}
)
df_test = pd.DataFrame(df_clam["Quantity sold in KG"][split_number:]).rename(
    columns={"Quantity sold in KG": "y_test"}
)
# define the dataframe to store the results
df_results = pd.DataFrame(
    columns=["aic", "bic", "hqic", "exo_variables", "MAPE", "MDAPE"]
)

# run the model for each possible combination of exogenous variables
for exo_combination in exo_combinations:
    # define the dataframes to be used in the model
    df_train_exo = pd.DataFrame(df_clam[list(exo_combination)][:split_number])
    df_test_exo = pd.DataFrame(df_clam[list(exo_combination)][split_number:])

    # run the model
    model_fit = pm.auto_arima(
        y=df_train,
        X=df_train_exo,
        test="adf",
        max_p=max_p,
        max_d=max_d,
        max_q=max_q,
        seasonal=seasonal,
        m=m,
        max_P=max_P,
        max_D=max_D,
        max_Q=max_Q,
```

```
        trace=trace,
        error_action=error_action,
        suppress_warnings=suppress_warnings,
        stepwise=stepwise,
        n_fits=n_fits,
        method=method,
    )

    # predict the values
    df_test["y_test_pred"] = model_fit.predict(
        , n_periods=pred_periods, dynamic=False
    )
    # calculate the MAPE
    MAPE = (
        np.mean(
            (
                np.abs(
                    np.subtract(df_test["y_test"], df_test["y_test_pred"])
                    / df_test["y_test"]
                )
            )
        )
        * 100
    )

    # calculate the MDAPE
    MDAPE = (
        np.median(
            (
                np.abs(
                    np.subtract(df_test["y_test"], df_test["y_test_pred"])
                    / df_test["y_test"]
                )
            )
        )
        * 100
    )

    test = model_fit.to_dict()
    test["seasonal_order"]

    # store the results in the dataframe
    df_results = df_results.append(
        {
            "aic": model_fit.aic(),
            "bic": model_fit.bic(),
            "hqic": model_fit.hqic(),
            "exo_variables": exo_combination,
            "MAPE": MAPE,
            "MDAPE": MDAPE,
            "order": test["order"],
            "seasonal_order": test["seasonal_order"],
        },
        ignore_index=True,
    )
```

```
# print the results
print(df_results)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=573.201, Time=1.84 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=636.589, Time=0.12 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=568.953, Time=0.62 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=inf, Time=0.62 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=634.777, Time=0.06 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=638.589, Time=0.11 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=612.606, Time=1.20 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=568.110, Time=0.80 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=inf, Time=0.54 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=633.466, Time=1.95 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=610.807, Time=1.20 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.51 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=613.997, Time=1.94 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=567.084, Time=0.57 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=604.190, Time=0.45 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=567.193, Time=0.48 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=631.160, Time=1.30 sec
 ARIMA(0,0,0)(1,0,2)[12] intercept   : AIC=608.391, Time=2.13 sec
 ARIMA(0,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.19 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=610.049, Time=1.01 sec
 ARIMA(0,0,0)(2,0,2)[12] intercept   : AIC=611.528, Time=1.71 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=567.976, Time=0.76 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=567.799, Time=0.91 sec
 ARIMA(0,0,0)(1,0,1)[12]             : AIC=584.152, Time=0.32 sec

Best model:  ARIMA(0,0,0)(1,0,1)[12] intercept
Total fit time: 23.413 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=2.49 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=456.576, Time=0.04 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=456.545, Time=0.66 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=1.16 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=454.792, Time=0.12 sec
 ARIMA(0,0,0)(1,1,0)[12] intercept   : AIC=455.968, Time=0.47 sec
 ARIMA(0,0,0)(0,1,1)[12] intercept   : AIC=inf, Time=0.80 sec
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.10 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=458.005, Time=0.17 sec
 ARIMA(0,0,1)(0,1,0)[12] intercept   : AIC=457.939, Time=0.23 sec
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=458.157, Time=0.55 sec

Best model:  ARIMA(0,0,0)(0,1,0)[12]
Total fit time: 7.850 seconds
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,1,1)[12]             : AIC=458.042, Time=2.12 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=477.778, Time=0.10 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=467.105, Time=1.51 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=2.04 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=455.105, Time=1.70 sec
 ARIMA(2,1,2)(0,1,0)[12]             : AIC=459.618, Time=0.55 sec
 ARIMA(2,1,2)(0,1,2)[12]             : AIC=458.752, Time=3.51 sec
 ARIMA(2,1,2)(1,1,0)[12]             : AIC=458.290, Time=2.07 sec
 ARIMA(2,1,2)(1,1,2)[12]             : AIC=458.549, Time=22.92 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=inf, Time=2.23 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=inf, Time=1.46 sec
 ARIMA(3,1,2)(0,1,1)[12]             : AIC=456.392, Time=2.75 sec
```

```
 ARIMA(2,1,3)(0,1,1)[12]             : AIC=456.234, Time=2.46 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=1.22 sec
 ARIMA(1,1,3)(0,1,1)[12]             : AIC=451.425, Time=1.94 sec
 ARIMA(1,1,3)(0,1,0)[12]             : AIC=460.512, Time=0.65 sec
 ARIMA(1,1,3)(1,1,1)[12]             : AIC=inf, Time=2.98 sec
 ARIMA(1,1,3)(0,1,2)[12]             : AIC=459.531, Time=219.65 sec
 ARIMA(1,1,3)(1,1,0)[12]             : AIC=458.772, Time=1.33 sec
 ARIMA(1,1,3)(1,1,2)[12]             : AIC=inf, Time=244.10 sec
 ARIMA(0,1,3)(0,1,1)[12]             : AIC=456.492, Time=1.73 sec
 ARIMA(1,1,4)(0,1,1)[12]             : AIC=459.556, Time=2.15 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=452.658, Time=1.43 sec
 ARIMA(0,1,4)(0,1,1)[12]             : AIC=inf, Time=1.23 sec
 ARIMA(2,1,4)(0,1,1)[12]             : AIC=459.958, Time=1.66 sec
 ARIMA(1,1,3)(0,1,1)[12] intercept   : AIC=472.570, Time=1.46 sec

Best model:  ARIMA(1,1,3)(0,1,1)[12]
Total fit time: 527.040 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=2.23 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=634.523, Time=0.14 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=570.796, Time=0.77 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=605.613, Time=0.62 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=632.718, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=636.344, Time=0.13 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=611.596, Time=2.78 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=569.968, Time=0.78 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=605.615, Time=0.59 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=632.357, Time=1.95 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=603.919, Time=1.76 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.72 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=605.866, Time=2.52 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=568.906, Time=0.65 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=603.661, Time=0.43 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=569.026, Time=0.60 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=630.183, Time=1.52 sec
 ARIMA(0,0,0)(1,0,2)[12] intercept   : AIC=595.152, Time=1.58 sec
 ARIMA(0,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.27 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=609.158, Time=3.59 sec
 ARIMA(0,0,0)(2,0,2)[12] intercept   : AIC=600.927, Time=3.05 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=569.742, Time=0.83 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=568.940, Time=1.04 sec
 ARIMA(0,0,0)(1,0,1)[12]             : AIC=585.533, Time=0.36 sec

Best model:  ARIMA(0,0,0)(1,0,1)[12] intercept
Total fit time: 31.047 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=569.610, Time=1.65 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=638.587, Time=0.22 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=570.616, Time=1.78 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=607.666, Time=1.55 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=636.718, Time=0.13 sec
 ARIMA(2,0,2)(0,0,1)[12] intercept   : AIC=611.358, Time=2.06 sec
 ARIMA(2,0,2)(1,0,0)[12] intercept   : AIC=578.521, Time=1.11 sec
 ARIMA(2,0,2)(2,0,1)[12] intercept   : AIC=645.251, Time=2.25 sec
 ARIMA(2,0,2)(1,0,2)[12] intercept   : AIC=617.186, Time=1.95 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=inf, Time=0.47 sec
```

```
 ARIMA(2,0,2)(0,0,2)[12] intercept   : AIC=inf, Time=3.35 sec
 ARIMA(2,0,2)(2,0,0)[12] intercept   : AIC=620.619, Time=2.20 sec
 ARIMA(2,0,2)(2,0,2)[12] intercept   : AIC=619.726, Time=3.60 sec
 ARIMA(1,0,2)(1,0,1)[12] intercept   : AIC=601.082, Time=1.29 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=630.268, Time=1.06 sec
 ARIMA(3,0,2)(1,0,1)[12] intercept   : AIC=593.160, Time=1.34 sec
 ARIMA(2,0,3)(1,0,1)[12] intercept   : AIC=598.132, Time=1.11 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.16 sec
 ARIMA(1,0,3)(1,0,1)[12] intercept   : AIC=628.979, Time=0.93 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=613.215, Time=1.09 sec
 ARIMA(3,0,3)(1,0,1)[12] intercept   : AIC=612.870, Time=1.48 sec
 ARIMA(2,0,2)(1,0,1)[12]             : AIC=585.725, Time=1.16 sec

Best model:  ARIMA(2,0,2)(1,0,1)[12] intercept
Total fit time: 33.006 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=3.94 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=458.202, Time=0.14 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=458.463, Time=1.00 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=1.32 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=456.501, Time=0.14 sec
 ARIMA(0,0,0)(1,1,0)[12] intercept   : AIC=457.810, Time=0.75 sec
 ARIMA(0,0,0)(0,1,1)[12] intercept   : AIC=inf, Time=1.12 sec
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.34 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=459.744, Time=0.24 sec
 ARIMA(0,0,1)(0,1,0)[12] intercept   : AIC=459.690, Time=0.32 sec
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=459.924, Time=0.64 sec

Best model:  ARIMA(0,0,0)(0,1,0)[12]
Total fit time: 11.007 seconds
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=1.37 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=635.206, Time=0.17 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=573.777, Time=0.86 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=606.471, Time=0.70 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=634.166, Time=0.08 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=637.093, Time=0.18 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=614.240, Time=2.39 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=571.841, Time=0.82 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=607.100, Time=0.69 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=637.072, Time=2.56 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=3.73 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.78 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=606.136, Time=3.03 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=570.821, Time=0.79 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=604.633, Time=0.58 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=570.611, Time=0.71 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=609.447, Time=4.29 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=634.895, Time=1.97 sec
 ARIMA(0,0,1)(1,0,0)[12] intercept   : AIC=572.958, Time=0.80 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=571.664, Time=0.93 sec
 ARIMA(0,0,0)(1,0,0)[12]             : AIC=581.440, Time=0.38 sec

Best model:  ARIMA(0,0,0)(1,0,0)[12] intercept
Total fit time: 28.891 seconds
          aic         bic        hqic  \
```

```
0  567.084281  578.032555  571.410464
1  454.792372  458.770340  456.326518
2  451.424693  465.216736  456.728447
3  568.905553  582.043481  574.096973
4  569.609550  591.506098  578.261917
5  456.501474  462.468426  458.802694
6  570.610558  583.748486  575.801978

                                      exo_variables       MAPE      MDAPE  \
0                              (off_days_per_month,)  11.903206   6.928463
1                           (school_count_per_month,)  16.108698  13.031001
2                                               (tavg,)  10.459255   7.756989
3        (off_days_per_month, school_count_per_month)  11.390210   7.767885
4                            (off_days_per_month, tavg)  10.040581   7.376541
5                         (school_count_per_month, tavg)  16.297206  13.731134
6  (off_days_per_month, school_count_per_month, t...  15.087387  13.749463

       order seasonal_order
0  (0, 0, 0)  (1, 0, 1, 12)
1  (0, 0, 0)  (0, 1, 0, 12)
2  (1, 1, 3)  (0, 1, 1, 12)
3  (0, 0, 0)  (1, 0, 1, 12)
4  (2, 0, 2)  (1, 0, 1, 12)
5  (0, 0, 0)  (0, 1, 0, 12)
6  (0, 0, 0)  (1, 0, 0, 12)
```

Adding Arima without exogenous

In [63]:

```python
pred_periods = 30
# corresponds to a prediction horizion  of 2,5 years
split_number = df_clam["Quantity sold in KG"].count() - pred_periods
df_train = pd.DataFrame(df_clam["Quantity sold in KG"][:split_number]).rename(
    columns={"Quantity sold in KG": "y_train"}
)
df_test = pd.DataFrame(df_clam["Quantity sold in KG"][split_number:]).rename(
    columns={"Quantity sold in KG": "y_test"}
)
# auto_arima
model_fit = pm.auto_arima(
    y=df_train,
    test="adf",
    max_p=5,
    max_d=5,
    max_q=5,
    seasonal=True,
    m=12,
    max_P=5,
    max_D=5,
    max_Q=5,
    trace=True,
    error_action="ignore",
    suppress_warnings=True,
    stepwise=True,
    n_fits=30,
    method="bfgs",
)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,1,1)[12]             : AIC=456.055, Time=1.95 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=475.778, Time=0.03 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=465.189, Time=0.19 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=0.46 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=453.145, Time=1.14 sec
 ARIMA(2,1,2)(0,1,0)[12]             : AIC=457.687, Time=0.39 sec
 ARIMA(2,1,2)(0,1,2)[12]             : AIC=456.764, Time=3.99 sec
 ARIMA(2,1,2)(1,1,0)[12]             : AIC=456.232, Time=1.65 sec
 ARIMA(2,1,2)(1,1,2)[12]             : AIC=456.551, Time=4.62 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=inf, Time=1.35 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=inf, Time=1.28 sec
 ARIMA(3,1,2)(0,1,1)[12]             : AIC=454.445, Time=2.87 sec
 ARIMA(2,1,3)(0,1,1)[12]             : AIC=inf, Time=4.06 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=1.04 sec
 ARIMA(1,1,3)(0,1,1)[12]             : AIC=449.535, Time=2.11 sec
 ARIMA(1,1,3)(0,1,0)[12]             : AIC=458.524, Time=0.53 sec
 ARIMA(1,1,3)(1,1,1)[12]             : AIC=inf, Time=3.05 sec
 ARIMA(1,1,3)(0,1,2)[12]             : AIC=457.536, Time=27.80 sec
 ARIMA(1,1,3)(1,1,0)[12]             : AIC=456.321, Time=1.09 sec
 ARIMA(1,1,3)(1,1,2)[12]             : AIC=inf, Time=21.89 sec
 ARIMA(0,1,3)(0,1,1)[12]             : AIC=454.469, Time=1.62 sec
 ARIMA(1,1,4)(0,1,1)[12]             : AIC=457.674, Time=1.91 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=450.611, Time=1.40 sec
 ARIMA(0,1,4)(0,1,1)[12]             : AIC=459.723, Time=1.10 sec
 ARIMA(2,1,4)(0,1,1)[12]             : AIC=457.967, Time=1.61 sec
 ARIMA(1,1,3)(0,1,1)[12] intercept   : AIC=470.659, Time=1.36 sec

Best model:  ARIMA(1,1,3)(0,1,1)[12]
Total fit time: 90.597 seconds
```

In [64]:

```python
df_test["y_test_pred"] = model_fit.predict(n_periods=pred_periods, dynamic=False)
MAPE = (
    np.mean(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
            )
        )
    )
    * 100
)

# calculate the MDAPE
MDAPE = (
    np.median(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
            )
        )
    )
    * 100
)
```

```
test = model_fit.to_dict()

# store the results in the dataframe
exo_combination = "Without Exogenous"
```

In [65]:
```
df_results = df_results.append(
    {
        "aic": model_fit.aic(),
        "bic": model_fit.bic(),
        "hqic": model_fit.hqic(),
        "exo_variables": exo_combination,
        "MAPE": MAPE,
        "MDAPE": MDAPE,
        "order": test["order"],
        "seasonal_order": test["seasonal_order"],
    },
    ignore_index=True,
)
```

In [66]:
```
df_results["sum"] = df_results.aic + df_results.bic + df_results.hqic
df_results
```

Out[66]:

	aic	bic	hqic	exo_variables	MAPE	MDAPE	order	seas
0	567.084281	578.032555	571.410464	(off_days_per_month,)	11.903206	6.928463	(0, 0, 0)	
1	454.792372	458.770340	456.326518	(school_count_per_month,)	16.108698	13.031001	(0, 0, 0)	
2	451.424693	465.216736	456.728447	(tavg,)	10.459255	7.756989	(1, 1, 3)	
3	568.905553	582.043481	574.096973	(off_days_per_month, school_count_per_month)	11.390210	7.767885	(0, 0, 0)	
4	569.609550	591.506098	578.261917	(off_days_per_month, tavg)	10.040581	7.376541	(2, 0, 2)	
5	456.501474	462.468426	458.802694	(school_count_per_month, tavg)	16.297206	13.731134	(0, 0, 0)	
6	570.610558	583.748486	575.801978	(off_days_per_month, school_count_per_month, t...	15.087387	13.749463	(0, 0, 0)	
7	449.535310	461.357061	454.081385	Without Exogenous	10.561874	8.405083	(1, 1, 3)	

In [67]:
```
df_results
```

Out[67]:

	aic	bic	hqic	exo_variables	MAPE	MDAPE	order	seas
0	567.084281	578.032555	571.410464	(off_days_per_month,)	11.903206	6.928463	(0, 0, 0)	
1	454.792372	458.770340	456.326518	(school_count_per_month,)	16.108698	13.031001	(0, 0, 0)	
2	451.424693	465.216736	456.728447	(tavg,)	10.459255	7.756989	(1, 1, 3)	
3	568.905553	582.043481	574.096973	(off_days_per_month, school_count_per_month)	11.390210	7.767885	(0, 0, 0)	
4	569.609550	591.506098	578.261917	(off_days_per_month, tavg)	10.040581	7.376541	(2, 0, 2)	
5	456.501474	462.468426	458.802694	(school_count_per_month, tavg)	16.297206	13.731134	(0, 0, 0)	
6	570.610558	583.748486	575.801978	(off_days_per_month, school_count_per_month, t...	15.087387	13.749463	(0, 0, 0)	
7	449.535310	461.357061	454.081385	Without Exogenous	10.561874	8.405083	(1, 1, 3)	

In [68]:
```
df_results_test = df_results
```

In [69]:
```
df_results_test
```

Out[69]:

	aic	bic	hqic	exo_variables	MAPE	MDAPE	order	seas
0	567.084281	578.032555	571.410464	(off_days_per_month,)	11.903206	6.928463	(0, 0, 0)	
1	454.792372	458.770340	456.326518	(school_count_per_month,)	16.108698	13.031001	(0, 0, 0)	
2	451.424693	465.216736	456.728447	(tavg,)	10.459255	7.756989	(1, 1, 3)	
3	568.905553	582.043481	574.096973	(off_days_per_month, school_count_per_month)	11.390210	7.767885	(0, 0, 0)	
4	569.609550	591.506098	578.261917	(off_days_per_month, tavg)	10.040581	7.376541	(2, 0, 2)	
5	456.501474	462.468426	458.802694	(school_count_per_month, tavg)	16.297206	13.731134	(0, 0, 0)	
6	570.610558	583.748486	575.801978	(off_days_per_month, school_count_per_month, t...	15.087387	13.749463	(0, 0, 0)	
7	449.535310	461.357061	454.081385	Without Exogenous	10.561874	8.405083	(1, 1, 3)	

In [70]:
```
df_results_test[["aic", "bic", "hqic", "MAPE", "MDAPE"]] = df_results_test[
```

```
    ["aic", "bic", "hqic", "MAPE", "MDAPE"]
].astype(float)
```

In [71]:
```
mape_min = df_results_test["MAPE"].min()
mdape_min = df_results_test["MDAPE"].min()
aic_min = df_results_test["aic"].min()
bic_min = df_results_test["bic"].min()
hqic_min = df_results_test["hqic"].min()
```

In [72]:
```
def factor(row):
    row["aic"] = row["aic"] / aic_min
    row["bic"] = row["bic"] / bic_min
    row["hqic"] = row["hqic"] / hqic_min
    row["MAPE"] = row["MAPE"] / mape_min
    row["MDAPE"] = row["MDAPE"] / mdape_min
    return (row["aic"] + row["bic"] + row["hqic"] + row["MAPE"] + row["MDAPE"]) / 5


# apply the function to each row
df_results_test["factor"] = df_results_test.apply(lambda row: factor(row), axis=1)
df_results_test
```

Out[72]:

	aic	bic	hqic	exo_variables	MAPE	MDAPE	order	seas
0	567.084281	578.032555	571.410464	(off_days_per_month,)	11.903206	6.928463	(0, 0, 0)	
1	454.792372	458.770340	456.326518	(school_count_per_month,)	16.108698	13.031001	(0, 0, 0)	
2	451.424693	465.216736	456.728447	(tavg,)	10.459255	7.756989	(1, 1, 3)	
3	568.905553	582.043481	574.096973	(off_days_per_month, school_count_per_month)	11.390210	7.767885	(0, 0, 0)	
4	569.609550	591.506098	578.261917	(off_days_per_month, tavg)	10.040581	7.376541	(2, 0, 2)	
5	456.501474	462.468426	458.802694	(school_count_per_month, tavg)	16.297206	13.731134	(0, 0, 0)	
6	570.610558	583.748486	575.801978	(off_days_per_month, school_count_per_month, t...	15.087387	13.749463	(0, 0, 0)	
7	449.535310	461.357061	454.081385	Without Exogenous	10.561874	8.405083	(1, 1, 3)	

In [73]:
```
df_results_test
```

Out[73]:

	aic	bic	hqic	exo_variables	MAPE	MDAPE	order	seas
0	567.084281	578.032555	571.410464	(off_days_per_month,)	11.903206	6.928463	(0, 0, 0)	
1	454.792372	458.770340	456.326518	(school_count_per_month,)	16.108698	13.031001	(0, 0, 0)	
2	451.424693	465.216736	456.728447	(tavg,)	10.459255	7.756989	(1, 1, 3)	
3	568.905553	582.043481	574.096973	(off_days_per_month, school_count_per_month)	11.390210	7.767885	(0, 0, 0)	
4	569.609550	591.506098	578.261917	(off_days_per_month, tavg)	10.040581	7.376541	(2, 0, 2)	
5	456.501474	462.468426	458.802694	(school_count_per_month, tavg)	16.297206	13.731134	(0, 0, 0)	
6	570.610558	583.748486	575.801978	(off_days_per_month, school_count_per_month, t...	15.087387	13.749463	(0, 0, 0)	
7	449.535310	461.357061	454.081385	Without Exogenous	10.561874	8.405083	(1, 1, 3)	

Arima with best model

In [74]:

```
pred_periods = 30
# corresponds to a prediction horizion  of 2,5 years
split_number = df_clam["Quantity sold in KG"].count() - pred_periods
df_train = pd.DataFrame(df_clam["Quantity sold in KG"][:split_number]).rename(
    columns={"Quantity sold in KG": "y_train"}
)
df_train_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][:split_number
)
df_test = pd.DataFrame(df_clam["Quantity sold in KG"][split_number:]).rename(
    columns={"Quantity sold in KG": "y_test"}
)
df_test_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][split_number:
)
# auto_arima
model_fit = pm.auto_arima(
    y=df_train,
    X=df_train_exo,
    test="adf",
    max_p=5,
    max_d=5,
    max_q=5,
    seasonal=True,
    m=12,
    max_P=5,
    max_D=5,
    max_Q=5,
```

```
    trace=True,
    error_action="ignore",
    suppress_warnings=True,
    stepwise=True,
    n_fits=30,
    method="bfgs",
)
print(model_fit.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=1.42 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=635.206, Time=0.17 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=573.777, Time=0.90 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=606.471, Time=0.69 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=634.166, Time=0.08 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=637.093, Time=0.19 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=614.240, Time=2.43 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=571.841, Time=0.87 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=607.100, Time=0.64 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=637.072, Time=2.55 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=3.71 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.69 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=606.136, Time=3.08 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=570.821, Time=0.80 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=604.633, Time=0.58 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=570.611, Time=0.73 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=609.447, Time=4.24 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=634.895, Time=1.93 sec
 ARIMA(0,0,1)(1,0,0)[12] intercept   : AIC=572.958, Time=0.74 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=571.664, Time=0.91 sec
 ARIMA(0,0,0)(1,0,0)[12]             : AIC=581.440, Time=0.35 sec

Best model:  ARIMA(0,0,0)(1,0,0)[12] intercept
Total fit time: 28.757 seconds
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                    66
Model:             SARIMAX(1, 0, 0, 12)   Log Likelihood                -279.305
Date:                  Mon, 12 Dec 2022   AIC                            570.611
Time:                          09:03:00   BIC                            583.748
Sample:                      01-01-2014   HQIC                           575.802
                           - 06-01-2019
Covariance Type:                    opg
==========================================================================================
                             coef    std err          z      P>|z|      [0.025
  0.975]
------------------------------------------------------------------------------------------
intercept                 15.4561      7.053      2.191      0.028       1.633
  29.279
off_days_per_month         1.0224      2.379      0.430      0.667      -3.640
   5.685
school_count_per_month     0.1557      0.482      0.323      0.747      -0.789
   1.101
tavg                      -0.7936      1.183     -0.671      0.502      -3.113
   1.526
ar.S.L12                   0.8429      0.066     12.698      0.000       0.713
   0.973
sigma2                   214.7413     43.884      4.893      0.000     128.729
300.753
==========================================================================================
=
Ljung-Box (L1) (Q):                   0.06   Jarque-Bera (JB):                30.2
7
```

```
Prob(Q):                              0.81   Prob(JB):                         0.0
0
Heteroskedasticity (H):               1.03   Skew:                             1.0
3
Prob(H) (two-sided):                  0.95   Kurtosis:                         5.6
0
================================================================================
=

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

Tests

In [76]:
```
df_grouped_by_month_2022_clean = pd.read_excel(
    "gdrive/MyDrive/arima_prediction/inputs_2022.xlsx"
)
df_grouped_by_month_2022_clean
```

Out[76]:

	year	month	off_days_per_month	school_count_per_month	tavg
0	2022	Januar	10	2	9.100000
1	2022	Februar	8	1	10.600000
2	2022	März	8	2	13.125000
3	2022	April	11	12	14.500000
4	2022	Mai	9	0	17.400000
5	2022	Juni	10	22	18.450000
6	2022	Juli	10	31	20.900000
7	2022	August	9	31	21.500000
8	2022	September	8	14	19.810000
9	2022	Oktober	11	0	16.436667
10	2022	November	9	0	13.853333
11	2022	Dezember	11	12	12.246667

In [77]:
```
dates = pd.date_range(start="2022-01-01", end="2022-12-31", freq="MS")
df_grouped_by_month_2022_clean["Date"] = dates
# Setting the index as the dates

df_grouped_by_month_2022_clean.set_index(
    df_grouped_by_month_2022_clean["Date"], inplace=True
)
df_grouped_by_month_2022_clean
```

Out[77]:

Date	year	month	off_days_per_month	school_count_per_month	tavg	Date
2022-01-01	2022	Januar	10	2	9.100000	2022-01-01
2022-02-01	2022	Februar	8	1	10.600000	2022-02-01
2022-03-01	2022	März	8	2	13.125000	2022-03-01
2022-04-01	2022	April	11	12	14.500000	2022-04-01
2022-05-01	2022	Mai	9	0	17.400000	2022-05-01
2022-06-01	2022	Juni	10	22	18.450000	2022-06-01
2022-07-01	2022	Juli	10	31	20.900000	2022-07-01
2022-08-01	2022	August	9	31	21.500000	2022-08-01
2022-09-01	2022	September	8	14	19.810000	2022-09-01
2022-10-01	2022	Oktober	11	0	16.436667	2022-10-01
2022-11-01	2022	November	9	0	13.853333	2022-11-01
2022-12-01	2022	Dezember	11	12	12.246667	2022-12-01

In [78]:
```
df_grouped_by_month_2022_clean = df_grouped_by_month_2022_clean.drop("Date", axis=1
df_grouped_by_month_2022_clean
```

Out[78]:

Date	year	month	off_days_per_month	school_count_per_month	tavg
2022-01-01	2022	Januar	10	2	9.100000
2022-02-01	2022	Februar	8	1	10.600000
2022-03-01	2022	März	8	2	13.125000
2022-04-01	2022	April	11	12	14.500000
2022-05-01	2022	Mai	9	0	17.400000
2022-06-01	2022	Juni	10	22	18.450000
2022-07-01	2022	Juli	10	31	20.900000
2022-08-01	2022	August	9	31	21.500000
2022-09-01	2022	September	8	14	19.810000
2022-10-01	2022	Oktober	11	0	16.436667
2022-11-01	2022	November	9	0	13.853333
2022-12-01	2022	Dezember	11	12	12.246667

In [79]:

```
exogenous_test_df = df_grouped_by_month_2022_clean[
    ["off_days_per_month", "school_count_per_month", "tavg"]
]
test_exo = pd.concat([df_test_exo, exogenous_test_df])
test_exo


filename = "ARIMA_Model.sav"
pickle.dump(model_fit, open(filename, "wb"))  ## This will create a pickle file

## Load Model
model_pickle = pickle.load(open(filename, "rb"))


## Forecast
fc, confint = model_pickle.predict(n_periods=42, X=test_exo, return_conf_int=True)
print(fc)
```

```
2019-07-01     85.807526
2019-08-01    112.834439
2019-09-01     75.632234
2019-10-01    102.753668
2019-11-01    113.539529
2019-12-01    151.604067
2020-01-01     59.866100
2020-02-01     90.976914
2020-03-01     94.827171
2020-04-01     81.416547
2020-05-01     91.348075
2020-06-01     80.010922
2020-07-01     86.569217
2020-08-01    110.316781
2020-09-01     78.472796
2020-10-01    104.791208
2020-11-01    108.631944
2020-12-01    146.239348
2021-01-01     69.065623
2021-02-01     91.439450
2021-03-01     92.061222
2021-04-01     83.278511
2021-05-01     92.580146
2021-06-01     79.532895
2021-07-01     90.163608
2021-08-01    107.319648
2021-09-01     80.281408
2021-10-01    102.309297
2021-11-01    108.504273
2021-12-01    135.596789
2022-01-01     74.271785
2022-02-01     94.232625
2022-03-01     92.724758
2022-04-01     89.332088
2022-05-01     90.852868
2022-06-01     85.130767
2022-07-01     91.302866
2022-08-01    104.700573
2022-09-01     82.564385
2022-10-01    102.265808
2022-11-01    105.952215
2022-12-01    132.276891
Freq: MS, dtype: float64
```

In [80]:

```
test_pred = model_fit.predict(X=df_test_exo, n_periods=pred_periods, dynamic=False)

df_test["y_test_pred"] = test_pred
MAPE = (
    np.mean(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
            )
        )
    )
```

```
    * 100
)
print(f"Mean Absolute Percentage Error (MAPE): {np.round(MAPE, 2)} %")

# Median Absolute Percentage Error (MDAPE)
MDAPE = (
    np.median(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
            )
        )
    )
    * 100
)
print(f"Median Absolute Percentage Error (MDAPE): {np.round(MDAPE, 2)} %")

df_metrics = pd.DataFrame(
    {
        "Metric": ["MAPE", "MDAPE"],
        "Description": [
            "Mean Absolute Percentage Error",
            "Median Absolute Percentage Error",
        ],
        "Value": [MAPE, MDAPE],
    }
)
df_metrics
```

```
Mean Absolute Percentage Error (MAPE): 15.09 %
Median Absolute Percentage Error (MDAPE): 13.75 %
```

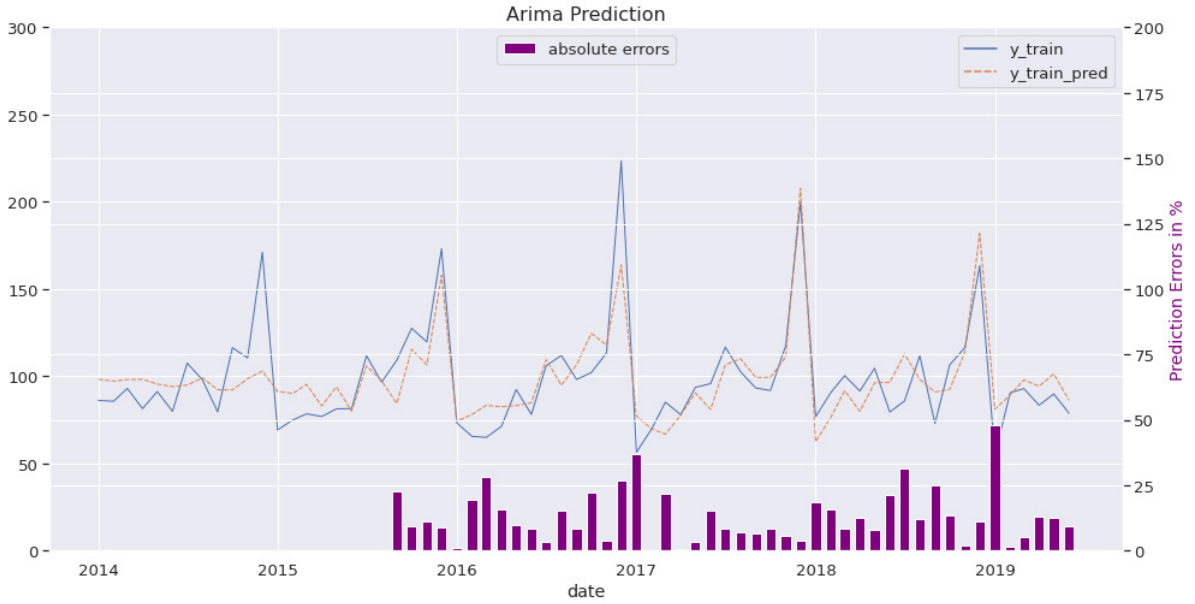
Out[80]:

	Metric	Description	Value
0	MAPE	Mean Absolute Percentage Error	15.087387
1	MDAPE	Median Absolute Percentage Error	13.749463

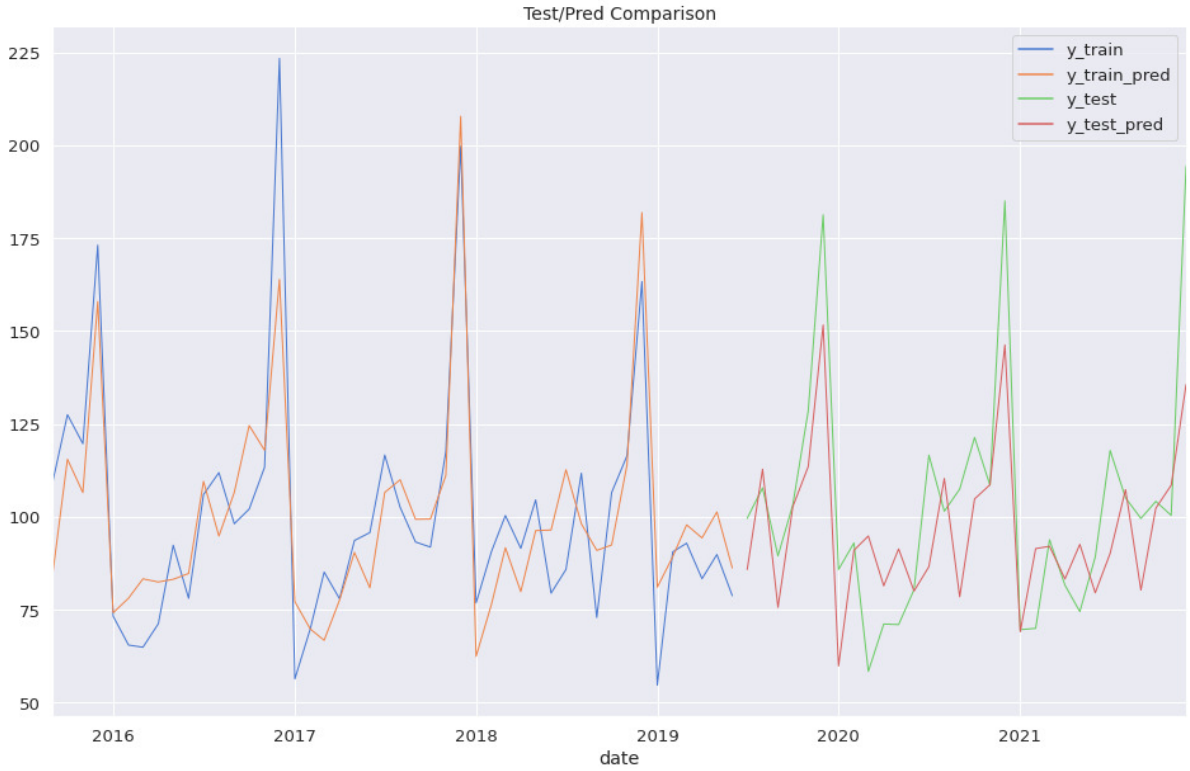
In [81]:

```
pred = model_fit.predict_in_sample(
    X=df_train_exo, dynamic=False
)  # works only with auto-arima
df_train["y_train_pred"] = pred

# Calculate the percentage difference
df_train["diff_percent"] = abs((df_train["y_train"] - pred) / df_train["y_train"])

# Print the predicted time-series
fig, ax1 = plt.subplots(figsize=(16, 8))
plt.title("Arima Prediction", fontsize=16)
sns.lineplot(data=df_train[["y_train", "y_train_pred"]], linewidth=1.0)

# Print percentage prediction errors on a separate axis (ax2)
ax2 = ax1.twinx()
ax2.set_ylabel("Prediction Errors in %", color="purple", fontsize=14)
ax2.set_ylim([0, 200])
```

```
ax1.set_ylim([0, 300])
ax2.bar(
    height=df_train["diff_percent"][20:],
    x=df_train.index[20:],
    width=20,
    color="purple",
    label="absolute errors",
)
plt.legend(loc="upper center")
plt.show()
```



In [82]:
```
test_pred = model_fit.predict(X=df_test_exo, n_periods=pred_periods, dynamic=False)


df_test["y_test_pred"] = test_pred
df_union = pd.concat([df_train, df_test])
# df_union.rename(columns={'y': 'y_test'}, inplace=True)

# Print the predicted time-series
fig, ax = plt.subplots(figsize=(16, 10))
plt.title("Test/Pred Comparison", fontsize=14)
sns.despine()
sns.lineplot(
    data=df_union[["y_train", "y_train_pred", "y_test", "y_test_pred"]],
    linewidth=1.0,
    dashes=False,
    palette="muted",
)
ax.set_xlim([df_union.index[20], df_union.index.max()])
plt.legend()
plt.show()
```



Arima with exogenous variables

In [83]:
```
pred_periods = 30
# corresponds to a prediction horizion  of 2,5 years
split_number = df_clam["Quantity sold in KG"].count() - pred_periods
df_train = pd.DataFrame(df_clam["Quantity sold in KG"][:split_number]).rename(
    columns={"Quantity sold in KG": "y_train"}
)
df_train_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][:split_number
)
df_test = pd.DataFrame(df_clam["Quantity sold in KG"][split_number:]).rename(
    columns={"Quantity sold in KG": "y_test"}
)
df_test_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][split_number:
)
# auto_arima
model_fit = pm.auto_arima(
    y=df_train,
    X=df_train_exo,
    test="adf",
    max_p=5,
    max_d=5,
    max_q=5,
    seasonal=True,
    m=12,
    max_P=5,
    max_D=5,
    max_Q=5,
    trace=True,
    error_action="ignore",
```

```
    suppress_warnings=True,
    stepwise=True,
    n_fits=30,
    method="bfgs",
)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=1.39 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=635.206, Time=0.17 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=573.777, Time=0.89 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=606.471, Time=0.73 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=634.166, Time=0.09 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=637.093, Time=0.17 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=614.240, Time=2.39 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=571.841, Time=0.90 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=607.100, Time=0.66 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=637.072, Time=2.60 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=3.72 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.82 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=606.136, Time=3.05 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=570.821, Time=0.84 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=604.633, Time=0.58 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=570.611, Time=0.72 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=609.447, Time=4.34 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=634.895, Time=1.98 sec
 ARIMA(0,0,1)(1,0,0)[12] intercept   : AIC=572.958, Time=0.79 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=571.664, Time=2.23 sec
 ARIMA(0,0,0)(1,0,0)[12]             : AIC=581.440, Time=0.91 sec

Best model:  ARIMA(0,0,0)(1,0,0)[12] intercept
Total fit time: 31.036 seconds
```

In [84]:
```
test_pred = model_fit.predict(X=df_test_exo, n_periods=pred_periods, dynamic=False)

df_test["y_test_pred"] = test_pred
```

In [85]:
```
MAPE = (
    np.mean(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
            )
        )
    )
    * 100
)
print(f"Mean Absolute Percentage Error (MAPE): {np.round(MAPE, 2)} %")

# Median Absolute Percentage Error (MDAPE)
MDAPE = (
    np.median(
        (
            np.abs(
                np.subtract(df_test["y_test"], df_test["y_test_pred"])
                / df_test["y_test"]
```

```
            )
        )
    )
    * 100
)
print(f"Median Absolute Percentage Error (MDAPE): {np.round(MDAPE, 2)} %")
```

```
Mean Absolute Percentage Error (MAPE): 15.09 %
Median Absolute Percentage Error (MDAPE): 13.75 %
```

Result Arima with exogenous: Mean Absolute Percentage Error (MAPE): 15.09 % Median Absolute Percentage Error (MDAPE): 13.75 %

Arima x Newsvendor

In [89]:
```
pred_periods = 30
# corresponds to a prediction horizion  of 2,5 years
split_number = df_clam["Quantity sold in KG"].count() - pred_periods
df_train = pd.DataFrame(df_clam["Quantity sold in KG"][:split_number]).rename(
    columns={"Quantity sold in KG": "y_train"}
)
df_train_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][:split_number
)
df_test = pd.DataFrame(df_clam["Quantity sold in KG"][split_number:]).rename(
    columns={"Quantity sold in KG": "y_test"}
)
df_test_exo = pd.DataFrame(
    df_clam[["off_days_per_month", "school_count_per_month", "tavg"]][split_number:
)
# auto_arima
model_fit = pm.auto_arima(
    y=df_train,
    X=df_train_exo,
    test="adf",
    max_p=5,
    max_d=5,
    max_q=5,
    seasonal=True,
    m=12,
    max_P=5,
    max_D=5,
    max_Q=5,
    trace=True,
    error_action="ignore",
    suppress_warnings=True,
    stepwise=True,
    n_fits=30,
    method="bfgs",
)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=3.01 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=635.206, Time=0.26 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=573.777, Time=1.85 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=606.471, Time=1.55 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=634.166, Time=0.15 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=637.093, Time=0.29 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=614.240, Time=4.11 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=571.841, Time=0.86 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=607.100, Time=0.66 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=637.072, Time=2.67 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=3.74 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=1.69 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=606.136, Time=2.96 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=570.821, Time=0.79 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=604.633, Time=0.58 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=570.611, Time=0.71 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=609.447, Time=9.74 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=634.895, Time=2.44 sec
 ARIMA(0,0,1)(1,0,0)[12] intercept   : AIC=572.958, Time=0.81 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=571.664, Time=0.95 sec
 ARIMA(0,0,0)(1,0,0)[12]             : AIC=581.440, Time=0.36 sec

Best model:  ARIMA(0,0,0)(1,0,0)[12] intercept
Total fit time: 40.247 seconds
```

In [90]:
```
model_fit.summary()
```

Out[90]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	66
Model:	SARIMAX(1, 0, 0, 12)	**Log Likelihood**	-279.305
Date:	Mon, 12 Dec 2022	**AIC**	570.611
Time:	09:08:21	**BIC**	583.748
Sample:	01-01-2014	**HQIC**	575.802
	- 06-01-2019		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	15.4561	7.053	2.191	0.028	1.633	29.279
off_days_per_month	1.0224	2.379	0.430	0.667	-3.640	5.685
school_count_per_month	0.1557	0.482	0.323	0.747	-0.789	1.101
tavg	-0.7936	1.183	-0.671	0.502	-3.113	1.526
ar.S.L12	0.8429	0.066	12.698	0.000	0.713	0.973
sigma2	214.7413	43.884	4.893	0.000	128.729	300.753

Ljung-Box (L1) (Q):	0.06	**Jarque-Bera (JB):**	30.27
Prob(Q):	0.81	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.03	**Skew:**	1.03
Prob(H) (two-sided):	0.95	**Kurtosis:**	5.60

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

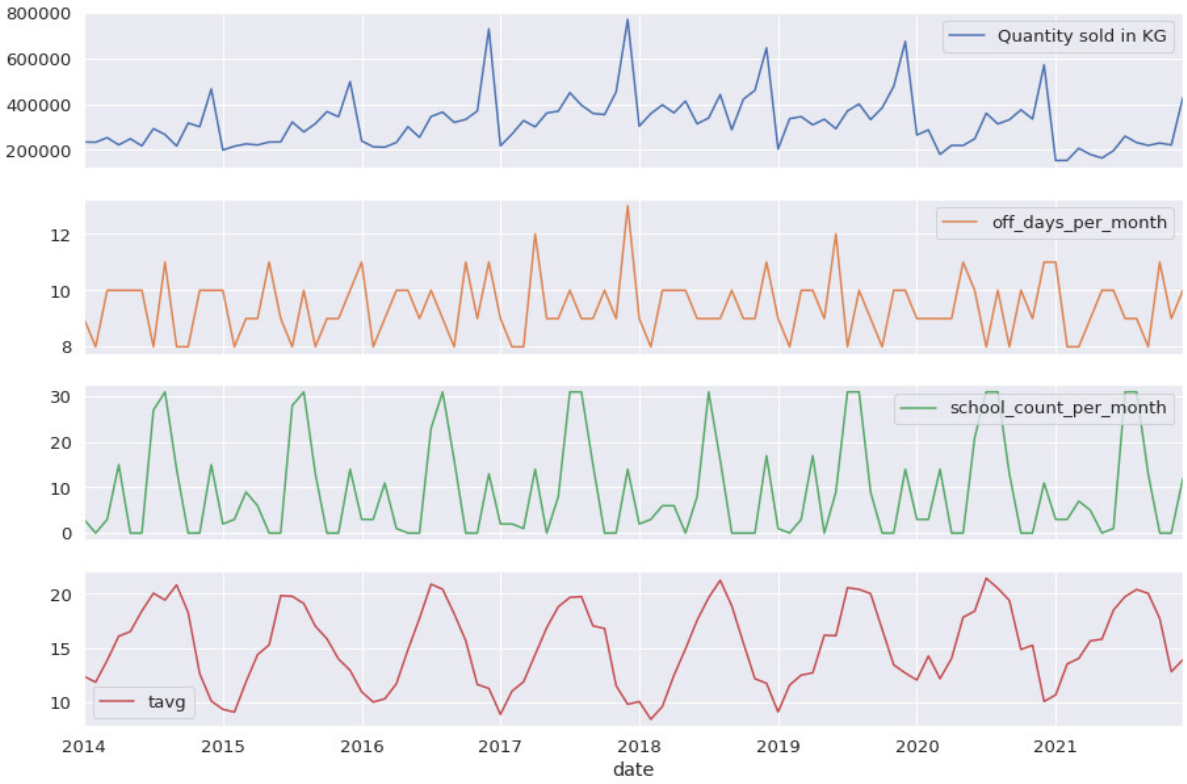
In [103…
```
n_periods = 30
fc, confint = model_fit.predict(
    X=df_test_exo, n_periods=n_periods, return_conf_int=True
)
y_forec, conf_int = model_fit.predict(
    X=df_test_exo, n_periods=n_periods, return_conf_int=True, alpha=0.05
)
```

In [104…
```
niklas_newsvendor = pd.DataFrame()
niklas_newsvendor["Date"] = y_forec.index
niklas_newsvendor["Lower"] = conf_int[:, 0]
niklas_newsvendor["Value"] = y_forec.values
niklas_newsvendor["Upper"] = conf_int[:, 1]
```

First try with pmarima

In [97]:
```
df = df_grouped_by_month[
    [
        "year",
        "month",
        "Menge in Kilogramm (Menge)",
        "count_per_month",
        "school_count_per_month",
        "tavg",
    ]
]
df["date"] = df["year"].astype(str) + "-" + df["month"].astype(str)
df["date"] = pd.to_datetime(df["date"])
df = df.drop(["year", "month"], axis=1)
df["count_per_month"] = df["count_per_month"].astype(float)
df["school_count_per_month"] = df["school_count_per_month"].astype(float)
df["tavg"] = df["tavg"].astype(float)
df["date"] = df["date"].astype(str)

df_clam = df[
    [
        "date",
        "Menge in Kilogramm (Menge)",
        "count_per_month",
        "school_count_per_month",
        "tavg",
    ]
]
df_clam = df_clam.rename(
    columns={
        "Menge in Kilogramm (Menge)": "Quantity sold in KG",
        "count_per_month": "off_days_per_month",
    }
)
df_clam = df_clam.sort_values(by="date")
df_clam = df_clam.set_index("date")
df_clam.index = pd.DatetimeIndex(df_clam.index).to_period("M")
df_clam.index = df_clam.index.to_timestamp()
df_clam[
    ["Quantity sold in KG", "off_days_per_month", "school_count_per_month", "tavg"]
].plot(subplots=True, figsize=(15, 10))
```

Out[97]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f444c595d30>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f4450340a30>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f44550eac10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f44503db430>],
      dtype=object)
```



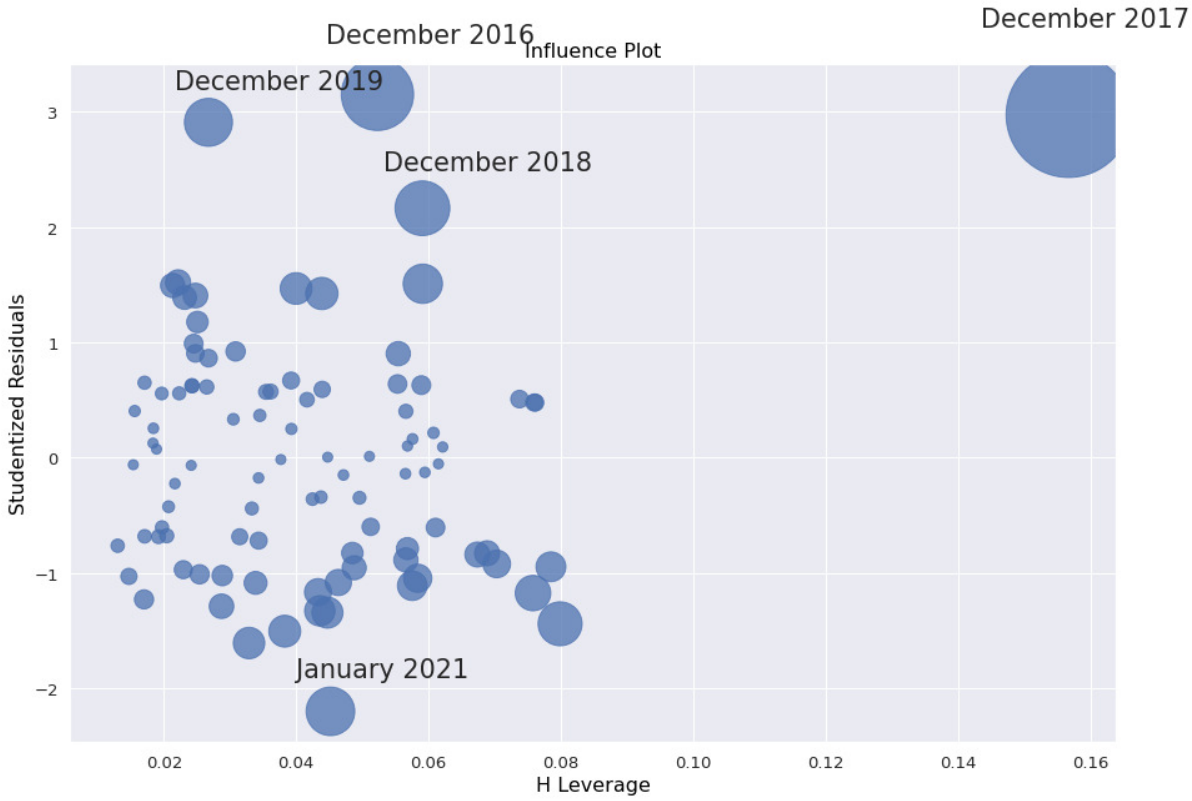
OLS

In [99]:
```
# prepare dataframe for OLS

df_OLS = df_clam
df_OLS.index = df_OLS.index.strftime("%B %Y")  # astype(str)

# define X and Y
y = df_OLS[["Quantity sold in KG"]]
x = df_OLS[["off_days_per_month", "school_count_per_month", "tavg"]]

# adding a constant, fit the OLS model and print results
x = sm.add_constant(x)
model = sm.OLS(y, x).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     Quantity sold in KG   R-squared:                       0.173
Model:                             OLS   Adj. R-squared:                  0.146
Method:                  Least Squares   F-statistic:                     6.425
Date:                 Mon, 12 Dec 2022   Prob (F-statistic):           0.000534
Time:                         09:10:20   Log-Likelihood:                -1246.9
No. Observations:                   96   AIC:                             2502.
Df Residuals:                       92   BIC:                             2512.
Df Model:                            3
Covariance Type:             nonrobust
================================================================================
========
                               coef    std err          t      P>|t|      [0.025
  0.975]
--------------------------------------------------------------------------------
--------
const                    1.243e+05   1.19e+05      1.049      0.297   -1.11e+05
3.6e+05
off_days_per_month        2.99e+04   1.06e+04      2.827      0.006    8889.877
5.09e+04
school_count_per_month   3622.3296   1233.847      2.936      0.004    1171.802
6072.857
tavg                    -7690.9272   3613.247     -2.129      0.036   -1.49e+04
-514.707
==============================================================================
Omnibus:                        6.867   Durbin-Watson:                   1.425
Prob(Omnibus):                  0.032   Jarque-Bera (JB):                6.368
Skew:                           0.606   Prob(JB):                       0.0414
Kurtosis:                       3.349   Cond. No.                         230.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
```

In [100…

```
# plot the influence of observations
fig, ax = plt.subplots(figsize=(15, 10))
fig = sm.graphics.influence_plot(model, ax=ax, criterion="cooks", size=100)
```


December 2016
December 2017
Influence Plot
December 2019
December 2018
January 2021
Studentized Residuals
H Leverage
3
2
1
0
−1
−2
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16