Report

Introduction and data

Inbound Logistics Forecasting Benchmark

Introduction

The problem at hand refers to forecasting the inbound material volume (in tons) on monthly basis for the next 4 months for an international automotive company.

The motivation behind that was the lack of synchronization between suppliers and freight forwarders systems, causing over- or under-capacity planning whenever a plant's material demands change abruptly, leading to higher logistics transportation costs.

I published a paper last year about this system, as a result of a research on inbound forecasting systems that I started in 2018. It can be found here Forecasting System for Inbound Logistics Material Flows at an International Automotive Company. Howerver this codebase was written in R using the forecasting package forecast by Robert Hyndman and George Athanasopoulos in their book Forecasting at Scale. It included algorithms like SARIMA, Exponential Smoothing, Multilayer Neural Networks, Prophet and Vector Autoregression. At that point in time the python packages for timeseries forecasting were not as good as R's. However, the python ecosystem for forecasting has grown a lot in the recent years. There are Python packages like nitxla, lightgbm, catboost, as well as LLM forecasting models like chronos that offer many additional functionalities to leverage the use of new algorithms.

In the M5 competition results link it was shown that boosted tree models can outperform traditional statistical and deep learning forecasting methods. This is something we will be exploring in this project.

In addition, I would like to use the historical covid numbers from the European Centre for Disease Prevention and Control to evaluate the influence of this variable on the accuracy of the predictions. This is something that has not been explored in research.

Research question

Which new methods can be used to improve the forecasting accuracy for the Inbound Logistics Volume of an International Automotive Company?

The idea is to create a forecasting system which is accurate and robust to adapt for outliers and unexpected events(e.g. COVID-19). To evaluate the forecast accuracy the `MAE (Mean Absolute Error)` and `SMAPE (Symmetric Mean Absolute Error)` will be used. This will allow us to care about the fact that in some months the transportation volume could have been 0.

The test timeframes are:

- Jan 2022 - Apr 2022
- May 2022 - Aug 2022
- Jul 2022 - Oct 2022

This means that models tested in each frame can only be trained with data prior to that frame to avoid data leakage.

One of the main Business KPI's to track forecast accuracy will be how many timeseries are in a particular `SMAPE` range, for that we will use the following intervals:

- 0% to 10%
- 10 to 20%
- 20 to 30%
- 30 to 40%
- greater than 40%

The business experts are particularly interested in having a forecating systems for which most of the timeseries have a `SMAPE` of less or equal than 20%.

Hypothesis

There are new forecasting methods which can deliver better accuracy than traditional statistical methods.

Data description

There are two dataset, one containing the historical volume data, another one containing the production data. In total there are:

- 624 inbound logistics Provider-Plant connections
- 18 plants
- 38 Providers

The historical transport volume data contains data since 2014-01-01 until 2022-10-01. The historical production data contains data since 2014-01-01 until 2023-12-01. All data until October 2022 is actual produced values, the rest are planning values.

The two input data sources for this project are:

- **Inbound_Volume_Data.csv** contains the historical transported material volume since January 2014 until October 2022 on monthly basis. This data comes from the Logistics Parts Mangement System.
  - **Timestamp**: Monthly data of the format YYYY/MM.
  - **Provider**: Logistics Service Provider.
  - **Plant**: Assembly Plant.
  - **Actual Vol [Kg]**: Actual transported volume from Provider to Plant in kg.
  - **Expected Vol [Kg]**: Expected transported volume from Provider to Plant in kg. [1]
- **production_data.csv** contains the historical production levels of all the european plants in number of vehicles per month from January 2014 until October 2022. Data after October 2022 refers to planned production values.
  - **Timestamp**: Monthly data of the format YYYY/MM.
  - **Plant_X**: Column containing the production level for Plant X.

[1] *Expected in this context means the volume value which the internal ERP system would calculate. That means, given the number of units in the call-off order and using the weights of the parts, the*

total expected weight of a delivery can be calculated. However, as mentioned before, due to the sync issue, the delivered volume and expected volume would differ.

Additionally, I will use the **monthly_covid_rate_per_country.parquet** file, which is generated after pivoting the file **Covid-19_cases_age_specific.csv** to monthly values per country as columns.

- **Covid-19_cases_age_specific.csv**: This data file contains information on the 14-day notification rate of newly reported COVID-19 cases per 100 000 population by age group, week and country. Each row contains the corresponding data for a certain week and country. The file is updated weekly. source. The Covid data ranges from 2020-01-06 until 2023-11-20.
    - **country**: Country name
    - **country_code**: cuontry code
    - **year_week**: YYYY-WW data
    - **age_group**: age group
    - **new_cases**: new covid cases
    - **population**: population
    - **rate_14_day_per_100k**: covid rate per 100.000 inhabitants
    - **source**: Covid source type
- The **monthly_covid_rate_per_country.parquet**, contains the monthly 14-day notification of newly reported COVID-19 cases per 100 000 population per european contry. The Covid data ranges from 2020-01-06 until 2023-11-20. This file is a pivoted version of the original file, for which each row represents a month and each column a country with its corresponding covid cases. Columns are:
    - **Timestamp**: Monthly date of the format YYYY-MM-DD
    - **Country**: Monthly COVID-19 Rate Per 100k (14-Day Average) in the given country

The **Inbound_Volume_Data** and **production_data** were obtained from the ERP System of the company and were annonymized for research purposes. The Covid **Covid-19_cases_age_specific.csv** data is available on the website of the European Centre for Disease Prevention and Control.

Packages & Functions

In [1]:
```
import os
import json
import sys
import os
import numpy as np
import pandas as pd
import networkx as nx
from datetime import datetime
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error

import warnings
warnings.filterwarnings('ignore')

# Add the parent directory to sys.path
sys.path.insert(0, os.path.abspath('..'))

from src.data_preprocessing import (preprocessing_volume_data,
                                    preprocessing_production,
```

```
                                    preprocesing_covid,
                                    data_quality_vol_analysis,
                                    apply_data_quality_timeseries)

from src.plotting import (plot_ratio_vol_prod,
                          plot_ratio_all_ts,
                          generate_smape_err_buckets,
                          plot_err_less_20_SMAPE,
                          plot_smape_buckets)
from src.utils import (read_config, smape,
                       store_pickle, read_pickle,
                       calculate_accuracy_metrics,
                       data_dict_covid, data_dict_prod,
                       data_dict_vol)
from src.feature_eng import features_seasonal_decomposition, apply_feature_eng
from src.models import train_test_lightgbm, train_test_stats_models
```

Data Dictionary

Historical Volume Data

In [2]:
```
data_dict_vol
print(data_dict_vol.to_markdown(index=False))
```

```
| Name                | Description                                                    | R
ole      | Type    | Format   |
|:--------------------|:---------------------------------------------------------------|:-
---------|:--------|:---------|
| Timestamp           | Monthly date of the format YYYY-MM-DD                          | I
D        | ordinal | datetime |
| Provider            | Logistics Provider ID                                          | I
D        | nominal | category |
| Plant               | Assembly Plant ID                                              | I
D        | nominal | category |
| Actual_Vol_[Kg]     | Actual transported volume from Provider to Plant in kg         | r
esponse  | numeric | float    |
| Expected_Vol_[Kg]   | Expected transported volume from Provider to Plant in kg       | p
redictor | numeric | float    |
| Year                | Year in which transport took place                             | p
redictor | numeric | int      |
| Month               | Month in which transport took place                            | p
redictor | numeric | int      |
| ts_key              | Timeseries key                                                 | I
D        | numeric | category |
| Actual_Vol_[Tons]   | Actual transported volume from Provider to Plant in tons       | p
redictor | numeric | float    |
| Expected_Vol_[Tons] | Expected transported volume from Provider to Plant in tons     | p
redictor | numeric | float    |
```

Historical Production Planning

In [3]:
```
data_dict_prod
print(data_dict_prod.to_markdown(index=False))
```

Name	Description	Role	Type	Format
Timestamp	Monthly date of the format YYYY-MM-DD	ID	ordinal	datetime
Plant	Assembly Plant ID	ID	nominal	category
Production	Production Volume in Number of Units	predictor	numeric	int

Covid Data

In [4]:
```
data_dict_covid
print(data_dict_covid.to_markdown(index=False))
```

Name	Description	Role	Type	Format
Timestamp	Monthly date of the format YYYY-MM-DD	ID	ordinal	datetime
Country	Monthly COVID-19 Rate Per 100k (14-Day Average) in the given country	predictor	numeric	float

Methodology

Config File

This configuration file is used to define various paths and parameters for data preprocessing, data quality checks, and feature engineering in the data analytics project. This is best practice in software projects to use `config` files, I always use this files whenever I create any project. Below is a detailed description of each section:

Preprocessing

This section specifies the file paths for different stages of data processing:

- **Bronze Tables**: Paths to raw data files.
  - `vol_data_path` : Path to the inbound volume data CSV file.
  - `prod_data_path` : Path to the production data CSV file.
  - `covid_data_path` : Path to the COVID-19 cases CSV file.
- **Silver Tables**: Paths to intermediate processed data files.
  - `vol_silver_path` : Path to the historical volume data in Parquet format.
  - `prod_silver_path` : Path to the production planning data in Parquet format.
  - `covid_silver_path` : Path to the monthly COVID-19 rate per country in Parquet format.
- **Gold Tables**: Paths to final processed data files.
  - `vol_gold_path` : Path to the historical volume data in Parquet format.
  - `prod_gold_path` : Path to the production planning data in Parquet format.
  - `covid_gold_path` : Path to the monthly COVID-19 rate per country in Parquet format.
  - `ratio_gold_path` : Path to the ratio of volume to production data in Parquet format.
  - `seasonal_feat_gold_path` : Path to the seasonal features data in Parquet format.
  - `timeseries_gold_path` : Path to the timeseries data in Parquet format.

- **Reports**: Path to the generated PDF report.
  - `pdf_report_ratios_path`: Path to the timeseries volume to production ratio PDF report.

Data Quality

This section specifies parameters for data quality checks:

- `ts_len_threshold`: Threshold for the length of the timeseries data.

Feature Engineering

This section specifies parameters for feature engineering:

- `lag_months`: List of months to be used for lag features.
- `rolling_months`: List of months to be used for rolling features.
- `drop_cols`: List of columns to be dropped from the dataset.

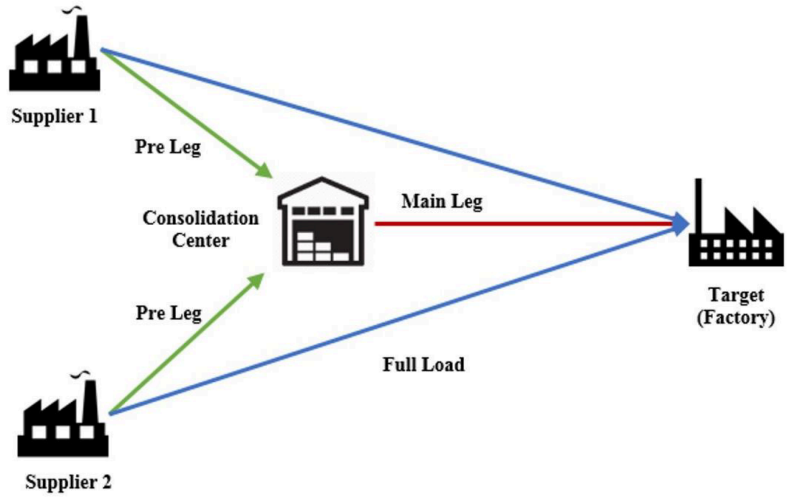
This configuration file helps in organizing and managing the paths and parameters required for different stages of the data analytics pipeline.

Import data

In [7]:
```
config = read_config(yaml_file_path="../config.yaml")
df_vol = pd.read_csv(config['preprocessing']['vol_data_path'],index_col=0)
df_prod = pd.read_csv(config['preprocessing']['prod_data_path'],index_col=0)
df_covid = pd.read_csv(config['preprocessing']['covid_data_path'])
```

Data structure

We can use the python package `networkx` to analyze the structure of the network. We can use the plotting functions to visualize it and understand how the Providers distribute material volume to the plants. First of all, we can model this problem as a bipartite graph, since the forecasting focuses only in the main legs of the Area Forwarding-based Inbound Logistics Network. As shown in the next figure:



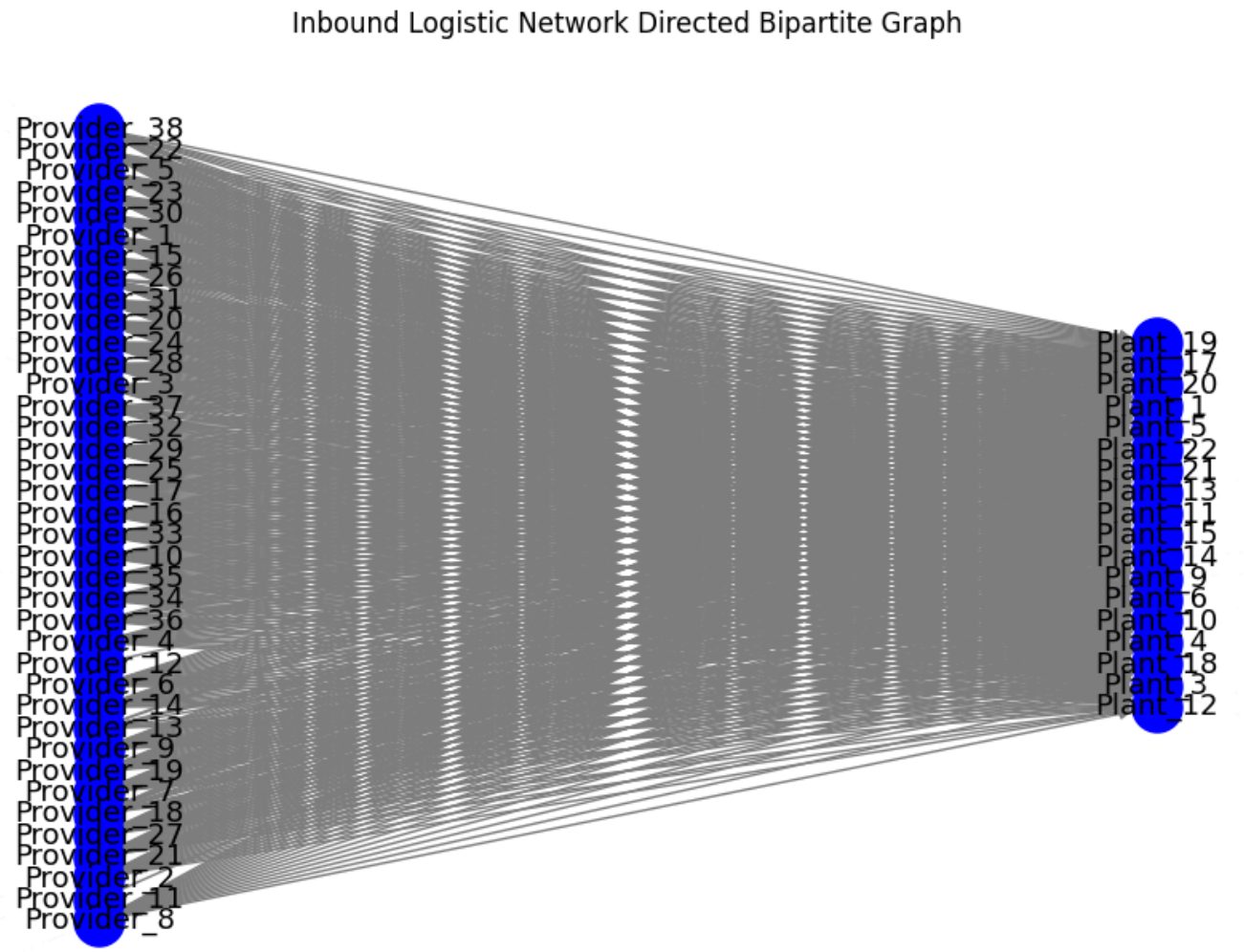
In [8]:
```
# Define nodes and edges
provider_nodes = df_vol['Provider'].to_list()
plant_nodes = df_vol['Plant'].to_list()
network_tuples_edges = [(x, y) for x, y in df_vol[['Provider', 'Plant']].drop_duplica
```

```
# Create Bipartite directed Graph
inbound_log_graph = nx.DiGraph()
inbound_log_graph.add_nodes_from(provider_nodes, bipartite=0)
inbound_log_graph.add_nodes_from(plant_nodes, bipartite=1)
inbound_log_graph.add_edges_from(network_tuples_edges)

# Graph Metrics
num_edges = inbound_log_graph.number_of_edges()
num_nodes = inbound_log_graph.number_of_nodes()
print(f"Number of main leg connection (edges) in the network: {num_edges}")
print(f"Number of nodes in the network: {num_nodes}")
```

```
Number of main leg connection (edges) in the network: 624
Number of nodes in the network: 56
```

The following image provide a good representation of the bipartite behaviour of the network. Since we only focus on the main leg, the network complexity is simplified. So that we can later either forecast the individual edges or also take into account cross-correlations among different Providers, when they deliver material flow to the same Plant.

In [10]:
```
file_name =  f"Inbound_Logistics_Network_Bipartite_Graph.png"
output_path = os.path.join("../img", file_name)

plt.figure(figsize=(8, 6))
pos = nx.multipartite_layout(inbound_log_graph, subset_key="bipartite")

# Draw the subgraph
nx.draw(
    inbound_log_graph,
    pos,
    with_labels=True,
    node_color="blue",
    edge_color="gray",
    node_size=500,
    font_size=13,
)

plt.title(f"Inbound Logistic Network Directed Bipartite Graph")
plt.savefig(output_path)
```



Data Preparation

Preparation Historical Volume Data

In [11]:
```
df_vol_bronze = df_vol.copy()
df_vol = preprocessing_volume_data(df_vol=df_vol_bronze)
```

In [12]:
```
print("The historical transport volume data contains data since", df_vol['Timestamp']
print("in Total it contains data for", df_vol['ts_key'].nunique(), " inbound logistic
print("in Total it contains data for", df_vol['Plant'].nunique(), " plants")
print("in Total it contains data for", df_vol['Provider'].nunique(), " Providers")
print("in Total it contains ", df_vol.shape[0], " rows.")
print("in Total it contains ", df_vol.shape[1], " columns.")
```

```
The historical transport volume data contains data since 2014-01-01 00:00:00  until  2
022-10-01 00:00:00
in Total it contains data for 624  inbound logistics Provider-Plant connections
in Total it contains data for 18  plants
in Total it contains data for 38  Providers
in Total it contains  47058  rows.
in Total it contains  10  columns.
```

In [13]:
```
# Store to silver
df_vol.to_parquet(config['preprocessing']['vol_silver_path'])
```

Data Quality and Data Completeness

One of the most important aspects in timeseries forecasting is to verify whether the timeseries have the same length.

In [14]:
```
df_vol_summary = data_quality_vol_analysis(df_vol=df_vol)

df_vol_gold = apply_data_quality_timeseries(df_vol=df_vol,
                                df_vol_summary=df_vol_summary,
                                ts_len_threshold=config['data_quality']['ts_len_thres
```

```
The min date available among all timeseries is:  2014-01-01 00:00:00
The max date available among all timeseries is:  2022-10-01 00:00:00
The min ts length is:  1
The max ts length is:  113
Number of time series with data until October 2022:  306
Number of Total Time Series Available:  624
Number of Total Time Series Available for Prediction:  49.0  %
Number of available timeseries after first filtering: 306
The min ts length is  1
The max ts length is  113
The mean ts length is  97.5268455014001
TS to forecast with Models 266
TS to forecast with Models 87.0  %
```

The previous analysis shows us how important it is to verify which time series actually meet the criteria and have the desired data quality for forecasting. In our case, we found out that only 266 out of 624, that is 42% of all time series are available at the last max date and meet the business criteria for forecasting. That is to say, most data was either outdated (material connections not existing anymore) or are not relevant for the business.

The 266 are the timeseries that will be relevant for forecasting. From now on, we will focus on these time series to analyze their patterns and create the forecast.

In [15]:
```
# Store clean volume data in gold layer
df_vol_gold.to_parquet(config['preprocessing']['vol_gold_path'])
```

Preparation Production Data

In [16]:
```
df_prod_bronze = df_prod.copy()
df_prod = preprocessing_production(df_prod=df_prod_bronze)
```

In [17]:
```
print("The historical production data contains data since", df_prod['Timestamp'].min(
print("in Total it contains ", df_prod.shape[0], " rows.")
print("in Total it contains ", df_prod.shape[1], " columns.")
print("Total available Plants are: ", df_prod['Plant'].nunique())
print("Max Production Volume was: ", df_prod['Production'].max(), " units. In",
    df_prod[df_prod['Production'] == df_prod['Production'].max()]['Timestamp'].values
print("Min Production Volume was: ", df_prod['Production'].min(), " units. In",
      df_prod[df_prod['Production'] == df_prod['Production'].min()]['Timestamp'].valu
```

```
The historical production data contains data since 2014-01-01 00:00:00  until  2023-12
-01 00:00:00
in Total it contains  2160  rows.
in Total it contains  3  columns.
Total available Plants are:  18
Max Production Volume was:  409207  units. In 2015-07-01T00:00:00.000000000
Min Production Volume was:  0  units. In 2020-04-01T00:00:00.000000000
```

In [18]:
```
# Store to silver
df_prod.to_parquet(config['preprocessing']['prod_silver_path'])
```

Preparation Covid Data

In [19]:
```
df_covid_bronze = df_covid.copy()
df_covid = preprocesing_covid(df_covid=df_covid_bronze)
```

```
The Covid data ranges from 2020-01-06 00:00:00  until  2023-11-20 00:00:00
The file contains data for  29  countries.
The file contains data for  6  age groups  ['<15yr' '15-24yr' '25-49yr' '50-64yr' '65-
79yr' '80+yr']
in Total it contains  35496  rows.
in Total it contains  9  columns.
```

In [20]:
```
print("Available countries ", df_covid.columns[1:])
```

```
Available countries  Index(['Belgium', 'Croatia', 'Cyprus', 'Czechia', 'Denmark', 'Est
onia',
       'Finland', 'France', 'Germany', 'Greece', 'Hungary', 'Iceland',
       'Ireland', 'Italy', 'Latvia', 'Liechtenstein', 'Lithuania',
       'Luxembourg', 'Malta', 'Netherlands', 'Norway', 'Poland', 'Portugal',
       'Romania', 'Slovakia', 'Slovenia', 'Spain', 'Sweden'],
      dtype='object', name='country')
```

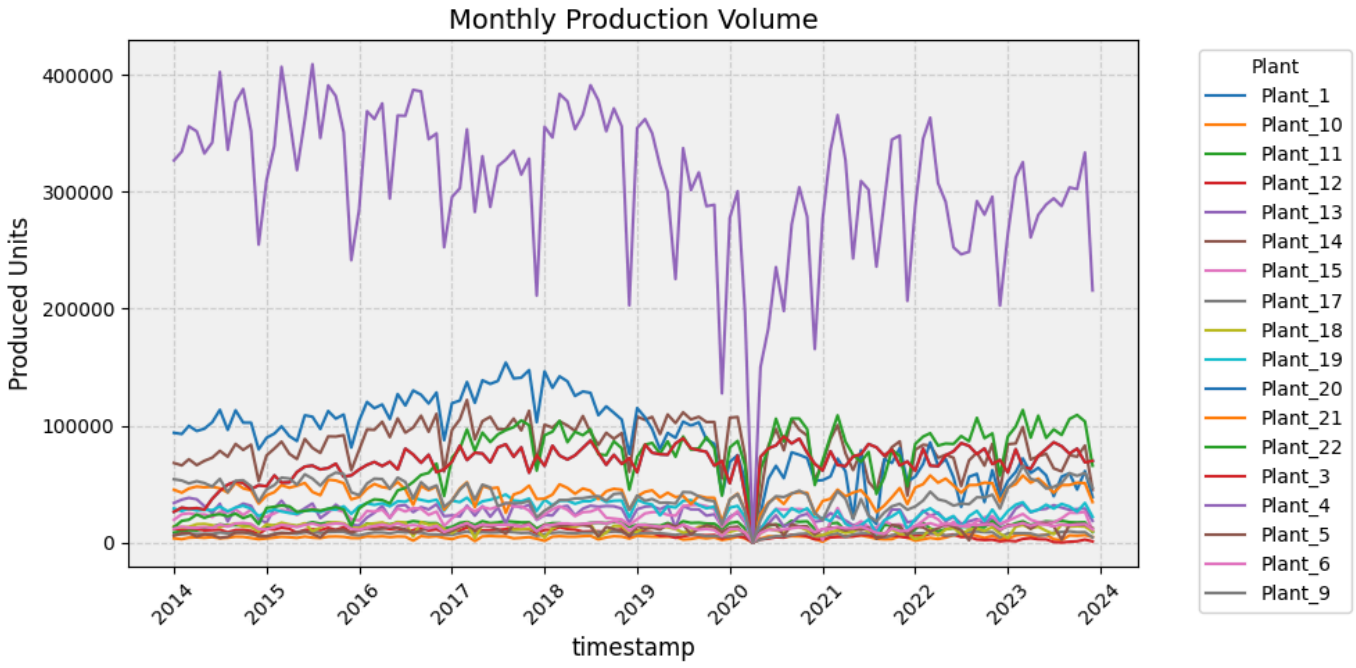
In [21]:
```
# Store to silver
df_covid.to_parquet(config['preprocessing']['covid_silver_path'])
```

Exploratory Data Analysis

Production Data

We can visualize all the productions volumes of all plants together.

In [23]:
```
# Free up memory and close all figures
plt.close('all')

plt.figure(figsize=(10, 5))

df_prod_pivot = df_prod.pivot(index='Timestamp', columns='Plant', values='Production'

# Plot each country's line
for plant in df_prod_pivot.columns:
    plt.plot(df_prod_pivot.index, df_prod_pivot[plant], label=plant)

plt.title('Monthly Production Volume', fontsize=14)
plt.xlabel('timestamp', fontsize=12)
plt.ylabel('Produced Units', fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.5)
plt.gca().set_facecolor('#f0f0f0')
plt.legend(title='Plant', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```

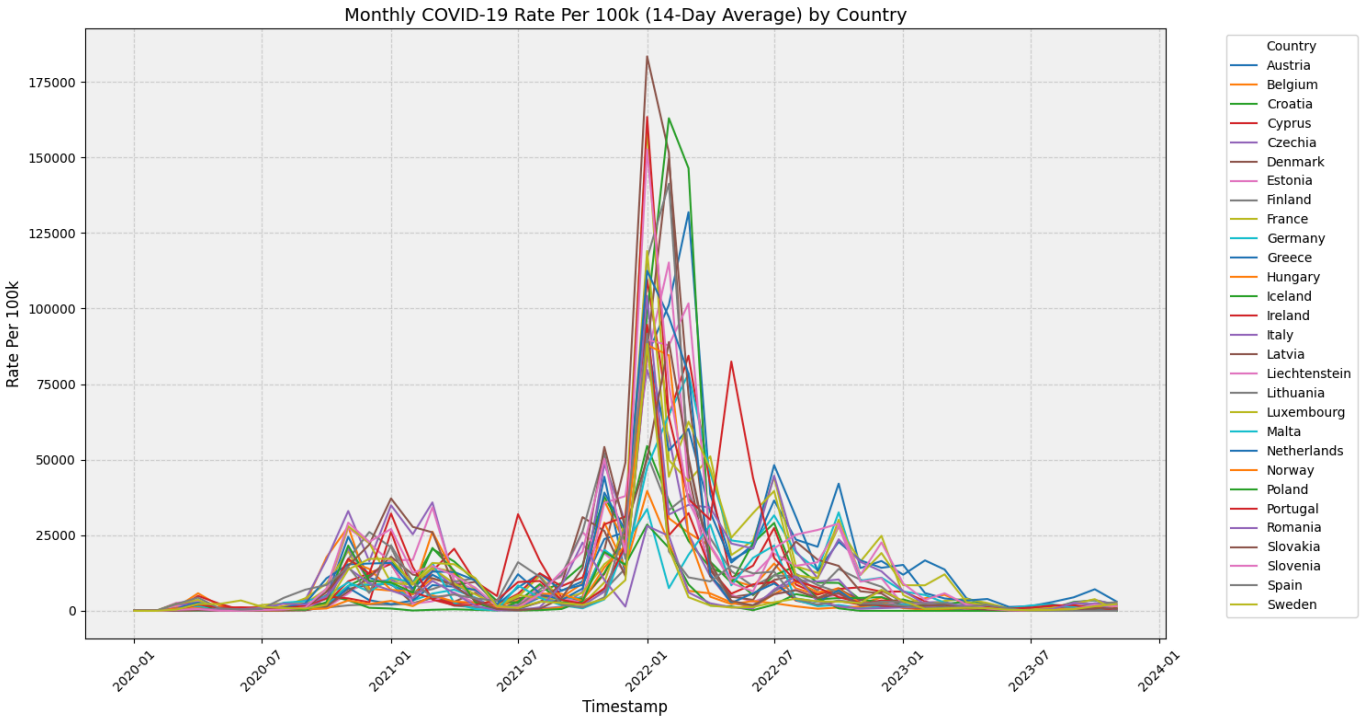


We can observe that data reveals a **strong 12-month seasonal pattern** in production volumes across the plants. Specifically, production levels tend to dip during the summer and winter months, indicating potential seasonality in demand, operational constraints, or planned maintenance periods during these times.

A significant anomaly is observed in April 2020, coinciding with the onset of the **COVID-19 outbreak and subsequent global lockdowns**. During this period, production dropped to nearly zero across all plants, suggesting a widespread halt in operations. This disruption is a clear outlier in the time series, with a sharp recovery observed in the following months as operations resumed.

Covid Data

In [25]:
```
# Free up memory and close all figures
plt.close('all')

plt.figure(figsize=(15, 8))

# Plot each country's line
for country in df_covid.columns:
    plt.plot(df_covid.index, df_covid[country], label=country)

plt.title('Monthly COVID-19 Rate Per 100k (14-Day Average) by Country', fontsize=14)
plt.xlabel('Timestamp', fontsize=12)
plt.ylabel('Rate Per 100k', fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.5)
plt.gca().set_facecolor('#f0f0f0')
plt.legend(title='Country', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```



This plot above illustrates the monthly COVID-19 rate per 100,000 inhabitants across various countries. The first notable peak occurs during the initial lockdowns in mid-2020 and early 2021, coinciding with the widespread implementation of strict public health measures. These peaks likely reflect the limited immunity in populations prior to the widespread availability of vaccines. World Data

Following the relaxation of restrictions, the graph shows a subsequent surge in case rates, particularly in late 2021 and early 2022. This trend aligns with the emergence of more transmissible variants, such as Delta and Omicron, and the lifting of social distancing measures. However, the overall rates stabilize over time, likely due to increased vaccine coverage and natural immunity from prior infections. The reduction in case rates in late 2022 and 2023 also suggests improved public health management, including booster campaigns and effective treatments. (Tartof SY et al., 2024) Estimated Effectiveness of the BNT162b2 XBB Vaccine Against COVID-19. link

This plot provides a clear temporal view of how the pandemic evolved, highlighting the impact of interventions like lockdowns and vaccinations on transmission rates.

We are interested in evaluating how much these numbers would influence the forecasting accuracy. As multiple disruptions occurred in the supply chain due to the covid lockdowns and restrictions. For example an increased volatility as researched by Capgemini.

Predictor Variable

In Timeseries forecasting it is common to carry out feature engineering using the response variable. For time dependent variables the common approach is to calculate lagged values as well as rolling statistics, e.g., rolling mean and rolling standard deviation. As for example explained by Manu Joseph in the chapter Feature Engineering for Timeseries in the book Modern Timeseries Forecasting with Python.

For the **historical transport volume** data the predictor variable would be:

- Provider
- Plant

- Expected Vol (Lagged Values, Rolling Mean, Rolling Std)
- Acutual Vol (Lagged Values, Rolling Mean, Rolling Std)
- Year
- Month

For the **Covid Data Set** after pivoting and transforming the data in order to match the timestamps values, I would use the following columns as predictors:

- monthly_rate_14_day_per_100k_per_country

Regarding the **Production Planning Data** it would be used as a smoothing factor to transform the response variable into the so called `Vol/Prod Ratio`. Because the production data and the inbound volume have a natural correlation. We can use this relationship to create a new target variable that will compensate for variations in the inbound volume. This new variable will be called `Vol/Prod Ratio`.

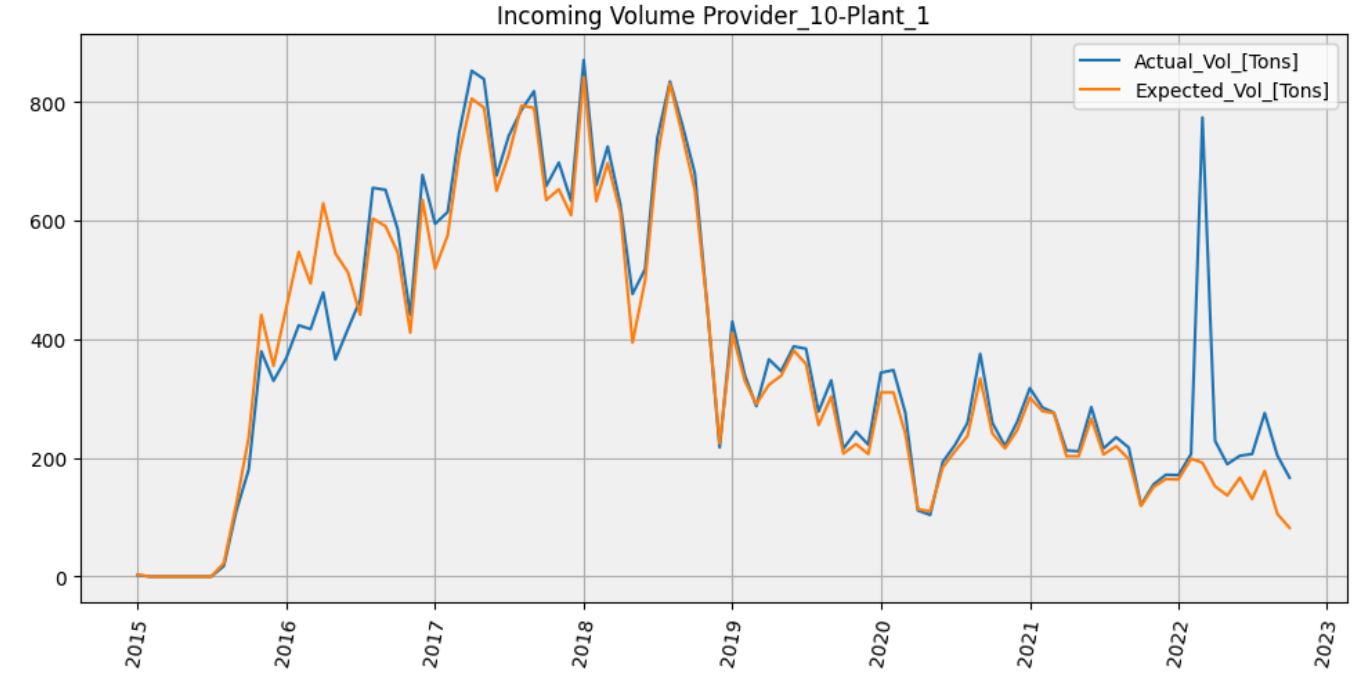
This approach is possible since the production planning data is always available on monthly basis for the next 12 months in the future. So if we use `Vol/Prod Ratio` instead of `Actual Vol [Kg]` to train the model, we can then easly multiply the forecast values of `Vol/Prod Ratio` with the Production Planning values to get the forecast of `Actual Vol [Kg]`.

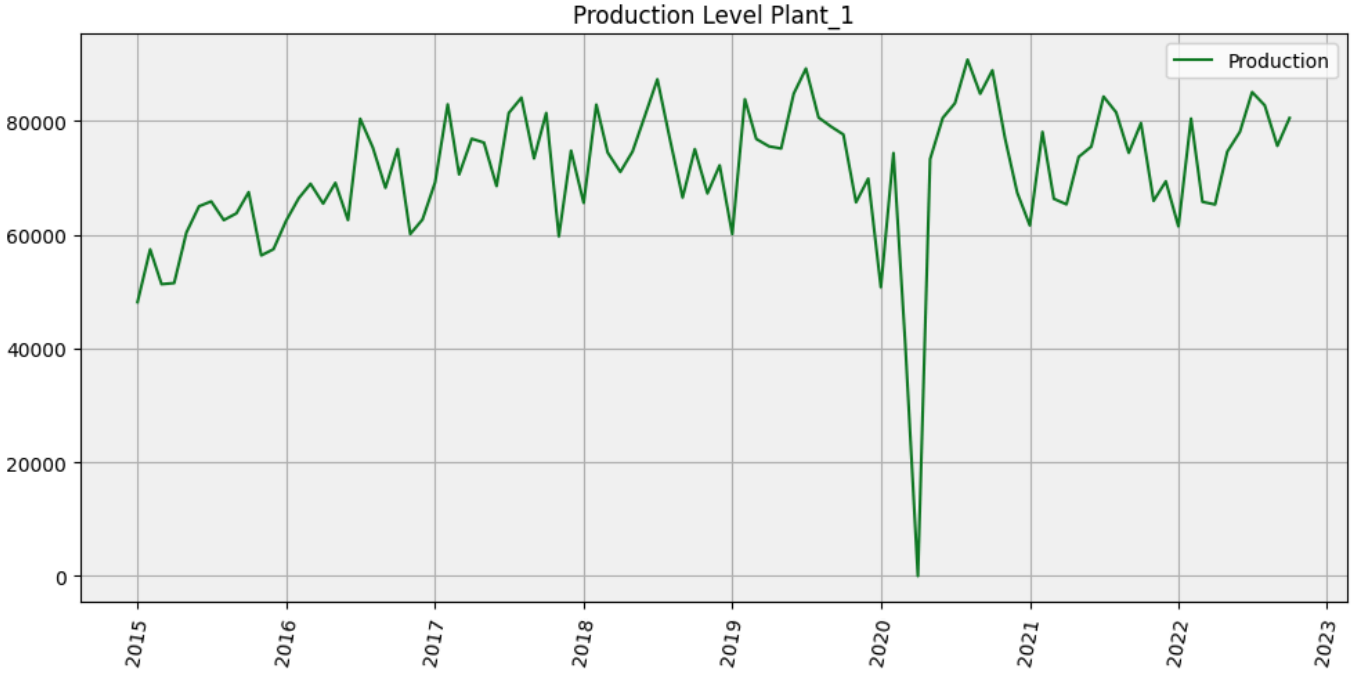
Merge Production Information to Historical Inbound Volume Data

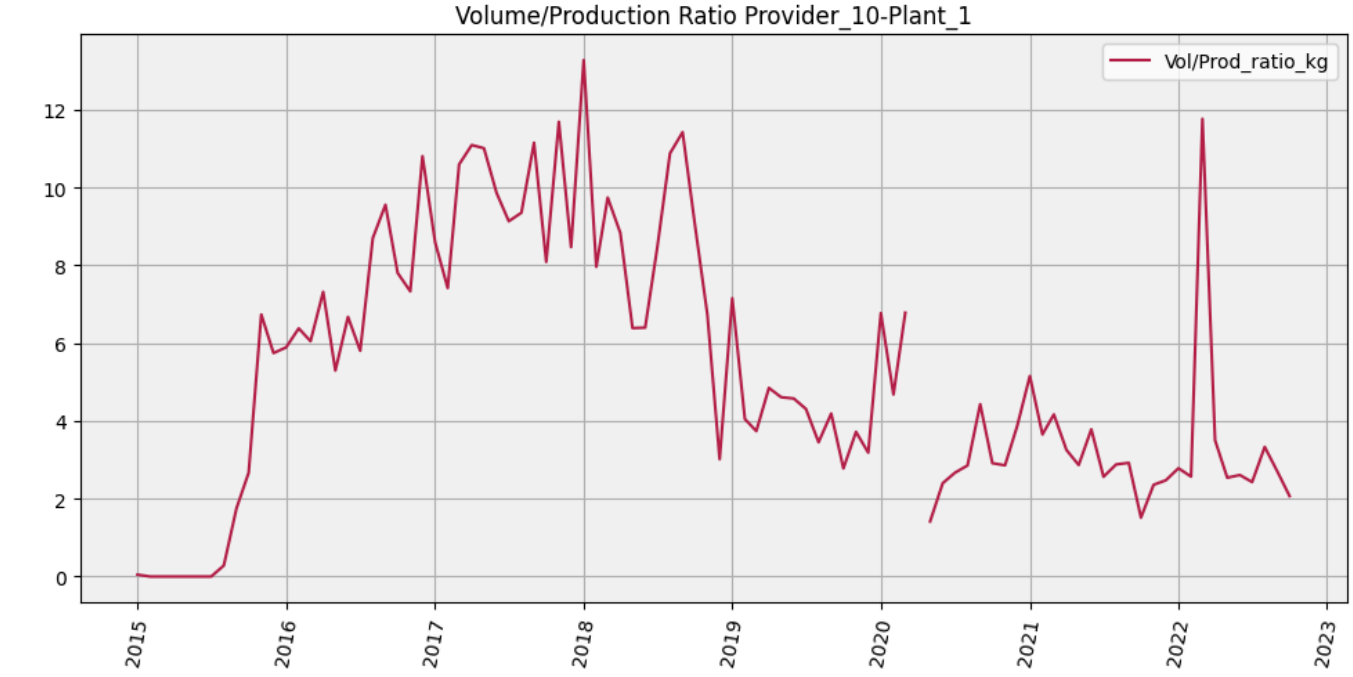
In [26]:
```
# Add production information to the timeseries
df_ratio_gold = pd.merge(df_vol_gold, df_prod, on=["Timestamp", "Plant"], how="left")

df_ratio_gold["Actual_Vol_[Kg]"] = df_ratio_gold["Actual_Vol_[Tons]"] * 1000
df_ratio_gold["Expected_Vol_[Kg]"] = df_ratio_gold["Expected_Vol_[Tons]"] * 1000

# Create New Feature:
# Volume / production Ratio
df_ratio_gold["Actual_Vol_[Kg]"] = df_ratio_gold["Actual_Vol_[Tons]"] * 1000
df_ratio_gold["Expected_Vol_[Kg]"] = df_ratio_gold["Expected_Vol_[Tons]"] * 1000
df_ratio_gold["Vol/Prod_ratio_ton"] = np.round(df_ratio_gold["Actual_Vol_[Tons]"] / d
df_ratio_gold["Vol/Prod_ratio_kg"] = np.round(df_ratio_gold["Actual_Vol_[Kg]"] / df_r
df_ratio_gold.replace([np.inf, -np.inf], np.nan, inplace=True)

# Add ts len
df_ratio_gold["ts_len"] = df_ratio_gold.groupby("ts_key")["Timestamp"].transform(lamb
df_ratio_gold.sort_values(by=["ts_key", "ts_len", "Timestamp"], inplace=True)
```

Volume/Production Ratio Analysis

In [27]:
```
ts_key = 'Provider_10-Plant_1'
plot_ratio_vol_prod(ts_key=ts_key, df_ratio=df_ratio_gold)
```


Incoming Volume Provider_10-Plant_1
Actual_Vol_[Tons]
Expected_Vol_[Tons]
800
600
400
200
0
2015
2016
2017
2018
2019
2020
2021
2022
2023


Production Level Plant_1
Production
80000
60000
40000
20000
0
2015
2016
2017
2018
2019
2020
2021
2022
2023


Volume/Production Ratio Provider_10-Plant_1
Vol/Prod_ratio_kg
12
10
8
6
4
2
0
2015
2016
2017
2018
2019
2020
2021
2022
2023

This visualization is very powerful. We can see how the Volume/Production ratio is closely related to the patterns from the production values as well as the historical inbound volume.

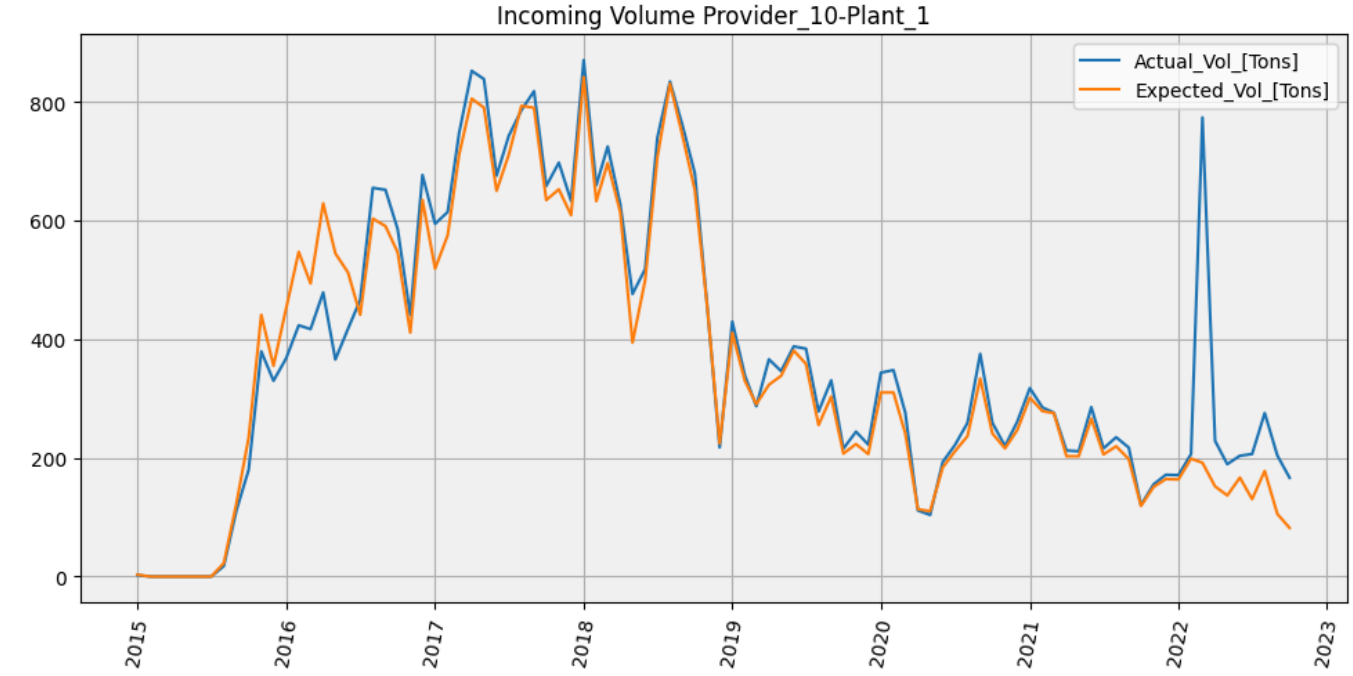
Linear Interpolation

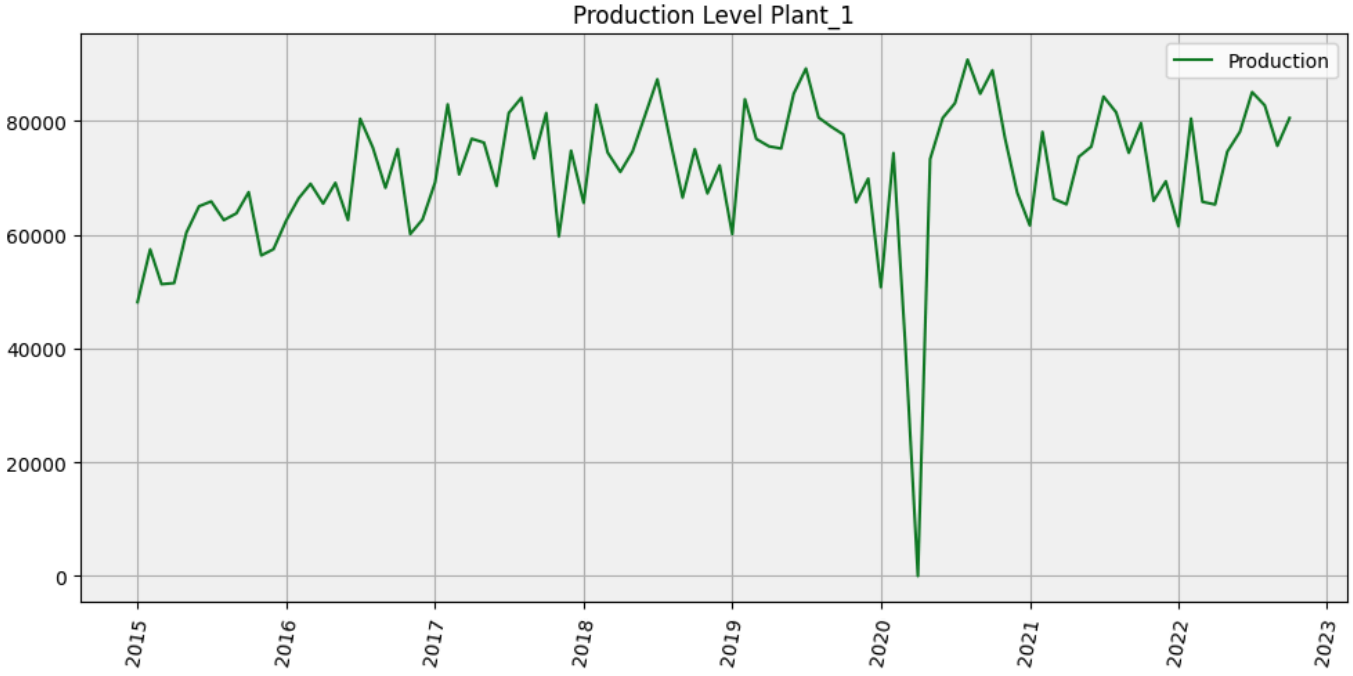
Another important factor when working with time series is to make sure we have data for all of our timestamps. Due to the Covid outbrake, April 2020 was a month in which no production took place. However, some inbound volume did flow into the plants. Therefore, when we calculate the volume/production ratio, we will not get any data because we would be calculating a division over 0.

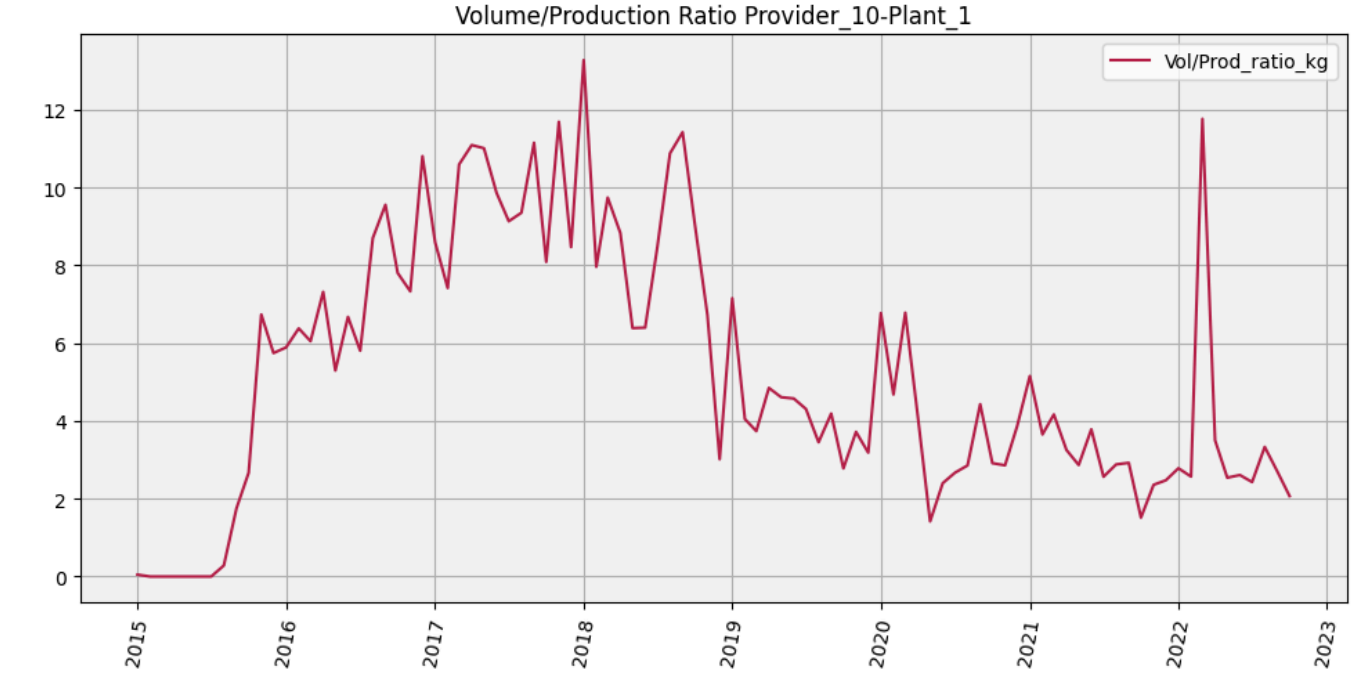
In [28]:
```
# Apply 1st order polynomial interpolation
df_ratio_gold['Vol/Prod_ratio_kg'] = df_ratio_gold['Vol/Prod_ratio_kg'].interpolate(m
```

In the following visualization we can observe how the linear interpolation created a smooth ratio for April 2020, even though the production value was 0.

In [29]:
```
ts_key = 'Provider_10-Plant_1'
plot_ratio_vol_prod(ts_key=ts_key, df_ratio=df_ratio_gold)
```







In [30]:
```
# Store ratio data to Gold layer
```

```
df_ratio_gold.to_parquet(config['preprocessing']['ratio_gold_path'])
```

In [31]:
```
# Create PDF Reaport of all ratios
plot_ratio_all_ts(df_ratio=df_ratio_gold, path=config['preprocessing']['pdf_report_ra
```

Timeseries Analysis

Based on `Konrad Banachewicz` full time series course on Kaggle. We can analize different factors in our timeseries. One powerful tool, is the seasonal decomposition; in particular the additve model.

The additive model for seasonal decomposition is used to break down a time series into three components:

- **T[t]**: The trend component, which captures the long-term movement or direction of the data over time.
- **S[t]**: The seasonal component, representing repeating patterns or cycles that occur at regular intervals (e.g., monthly, yearly).
- **e[t]**: The residual or error component, which accounts for random fluctuations or noise that cannot be explained by the trend or seasonality.

The model assumes that the time series $Y[t]$ is the sum of these components at any time t:

$$Y[t] = T[t] + S[t] + e[t]$$

Since we have a strong seasonal component in the data, i.e. incoming volume is lower in the winter and summer peaks due to vacations, and higher in other months. We can use this method to generate features that can then be used by machine learning models. When it comes to univariate statistical models, they should be able to pick out these trends by themselves. A good example of these models is the ARIMA model.

- Ref: https://www.kaggle.com/code/konradb/ts-0-the-basics
- Ref: https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html

We can use this data and back fill and forward fill the missing values (missing values due to the 12-month seasonal parameter) as new featuers and add them to new table `timeseries_features_gold`.

In [32]:
```
df_ts_decomposition = features_seasonal_decomposition(df_ratio_gold=df_ratio_gold, ta
```

In [33]:
```
df_ts_decomposition.head()
```

Out[33]:

	trend	sesonality	residuals	ts_key	Timestamp
20004	1.678919	1.267479	-0.87061	Provider_10-Plant_1	2015-01-01
20005	1.678919	-0.609561	-0.87061	Provider_10-Plant_1	2015-02-01
20006	1.678919	1.670471	-0.87061	Provider_10-Plant_1	2015-03-01
20007	1.678919	0.255907	-0.87061	Provider_10-Plant_1	2015-04-01
20008	1.678919	-0.979453	-0.87061	Provider_10-Plant_1	2015-05-01

In [34]:
```
# Store data to gold
df_ts_decomposition.to_parquet(config['preprocessing']['seasonal_feat_gold_path'])
```

Feature Engineering

Feature engineering in machine learning is the process of selecting, transforming, and creating new input variables (features) from raw data to improve the performance of models. It involves domain knowledge and various techniques like scaling, encoding, or generating new features, helping algorithms better capture underlying patterns in the data. The goal is to enhance model accuracy and predictive power by providing more meaningful data inputs.

In timeseries forecasting tasks common features engineering techniques are:

- Lag Features
- Rolling Features
- Statistical Analysis Features like mean, stadard deviation.

References can be found here Feature Engineering for Timeseries from the book Modern Timeseries Forecasting with Python

In [35]:
```
df_ts_decomposition = pd.read_parquet(config['preprocessing']['seasonal_feat_gold_pat
df_ratio_gold = pd.read_parquet(config['preprocessing']['ratio_gold_path'])
df_covid = pd.read_parquet(config['preprocessing']['covid_silver_path'])
```

In [36]:
```
df_ratio_gold.head()
```

Out[36]:

	Timestamp	ts_key	Actual_Vol_[Tons]	Expected_Vol_[Tons]	Plant	Production	Ac
20004	2015-01-01	Provider_10-Plant_1	2.344	3.109	Plant_1	48144	
20005	2015-02-01	Provider_10-Plant_1	0.000	0.000	Plant_1	57400	
20006	2015-03-01	Provider_10-Plant_1	0.000	0.000	Plant_1	51290	
20007	2015-04-01	Provider_10-Plant_1	0.000	0.000	Plant_1	51489	
20008	2015-05-01	Provider_10-Plant_1	0.000	0.000	Plant_1	60348	

In [37]:
```
df_timeseries_gold = apply_feature_eng(df_ratio_gold=df_ratio_gold,
                     df_ts_decomposition=df_ts_decomposition,
                     df_covid=df_covid,
                     config=config['feature_eng'],
                     verbosity=0)
```

In [38]:
```
df_timeseries_gold.to_parquet(config['preprocessing']['timeseries_gold_path'])
```

Data splitting

In time series analysis, data splitting differs from traditional regression and classification problems because we must respect the temporal dependencies in the data. Unlike random splits, the validation and test datasets must always occur after the training dataset to ensure that future data is not used to predict past events, which would violate the time series structure.

To evaluate the models effectively, I will define three separate datasets: training, validation, and testing. The test datasets will correspond to the following timeframes:

- January 2022 – April 2022
- May 2022 – August 2022
- July 2022 – October 2022

In [40]:
```
# Create the Timestamps splits for Train, Validation and Test
# Train = [: shard[0]]
# Validation = [shard[1] : shard[2]]
# Test = [shard[3] : shard[4]]
shards = [
    [datetime(2021,8,1), datetime(2021,9,1), datetime(2021,12,1), datetime(2022,1,1),
    [datetime(2021,12,1), datetime(2022,1,1), datetime(2022,4,1), datetime(2022,5,1),
    [datetime(2022,2,1), datetime(2022,3,1), datetime(2022,6,1), datetime(2022,7,1),
]
```

Model

Statistical Models

In this section I will explore the following statistical Models:

- **ARIMA**:(AutoRegressive Integrated Moving Average) is a forecasting method that combines autoregression, differencing, and moving averages to model and predict time series data based on its own past values and errors.
- **Exponential Smoothing Models**: The exponential smoothing forecasting model is a method that predicts future data points by weighting past observations, giving more importance to recent data while gradually reducing the influence of older data
- **WindowAverage**: The WindowAverage model forecasts future values by calculating the average of the most recent observations within a defined window of time, smoothing out short-term fluctuations.

The python package ecosystem `nixtla` contains multiple package for different timeseries tasks. The package statsforecast, is optimized for applying automated versions of these models.

In [41]:
```
# this make it so that the outputs of the predict methods have the id as a column
# instead of as the index
os.environ['NIXTLA_ID_AS_COL'] = '1'

sf, df_stats_forecast = train_test_stats_models(ts=df_timeseries_gold, shards=shards)
```

```
Forecasting for test frame 2022-01-01 - 2022-04-01
Forecasting for test frame 2022-05-01 - 2022-08-01
Forecasting for test frame 2022-07-01 - 2022-10-01
```

In [42]:
```
df_stats_forecast_melted = pd.melt(
        df_stats_forecast,
        id_vars=['ts_key', 'Timestamp', 'test_frame'],
        value_vars=['AutoARIMA', 'AutoETS', 'CES', 'SeasonalNaive', 'WindowAverage'],
        var_name='forecasting_model',
        value_name='y_hat'
    )
```

In [62]:
```
# Store Model
path="../models/stats_forecast.pkl"
```

```
store_pickle(obj=sf, path=path)

# Store Forecasts
df_stats_forecast.to_parquet("../data/forecasts/stats_forecast.parquet")
```

```
Object has been stored at ../models/stats_forecast.pkl.
```

In [63]:
```
df_stats_forecast = pd.read_parquet("../data/forecasts/stats_forecast.parquet")
```

Machine Learning

Now I will try out the Machine Learning Model known as boosted trees, in particular the algorithms LightGBM:

- LightGBM is a gradient boosting framework that uses tree-based learning algorithms to produce fast, efficient, and accurate predictions, particularly suited for large datasets and high-dimensional data

In [43]:
```
import optuna
optuna.logging.set_verbosity(optuna.logging.WARNING)
lgb_model, df_result_lgbm = train_test_lightgbm(ts=df_timeseries_gold, shards=shards)
```

```
Train-Testing for 2022-01-01 2022-04-01
Best hyperparameters: {'learning_rate': 0.07865596002539792, 'num_leaves': 416, 'subsa
mple': 0.23276419771978507, 'colsample_bytree': 0.22074001731969456, 'min_data_in_lea
f': 3}
Best MAE: 3.0415328870026515
Train-Testing for 2022-05-01 2022-08-01
Best hyperparameters: {'learning_rate': 0.06296003569452024, 'num_leaves': 475, 'subsa
mple': 0.8613250367904881, 'colsample_bytree': 0.5563141501728062, 'min_data_in_leaf':
54}
Best MAE: 4.18205484739683
Train-Testing for 2022-07-01 2022-10-01
Best hyperparameters: {'learning_rate': 0.0717134861701496, 'num_leaves': 670, 'subsam
ple': 0.41606647699215815, 'colsample_bytree': 0.8343966209261452, 'min_data_in_leaf':
78}
Best MAE: 3.993212552454154
```

In [44]:
```
# Store Model
path="../models/lightgbm.pkl"
store_pickle(obj=lgb_model, path=path)

# Store Forecasts
df_result_lgbm.to_parquet("../data/forecasts/lightgmb_forecast.parquet")
#df_result_lgbm = pd.read_parquet("../data/forecasts/lightgmb_forecast.parquet")
```

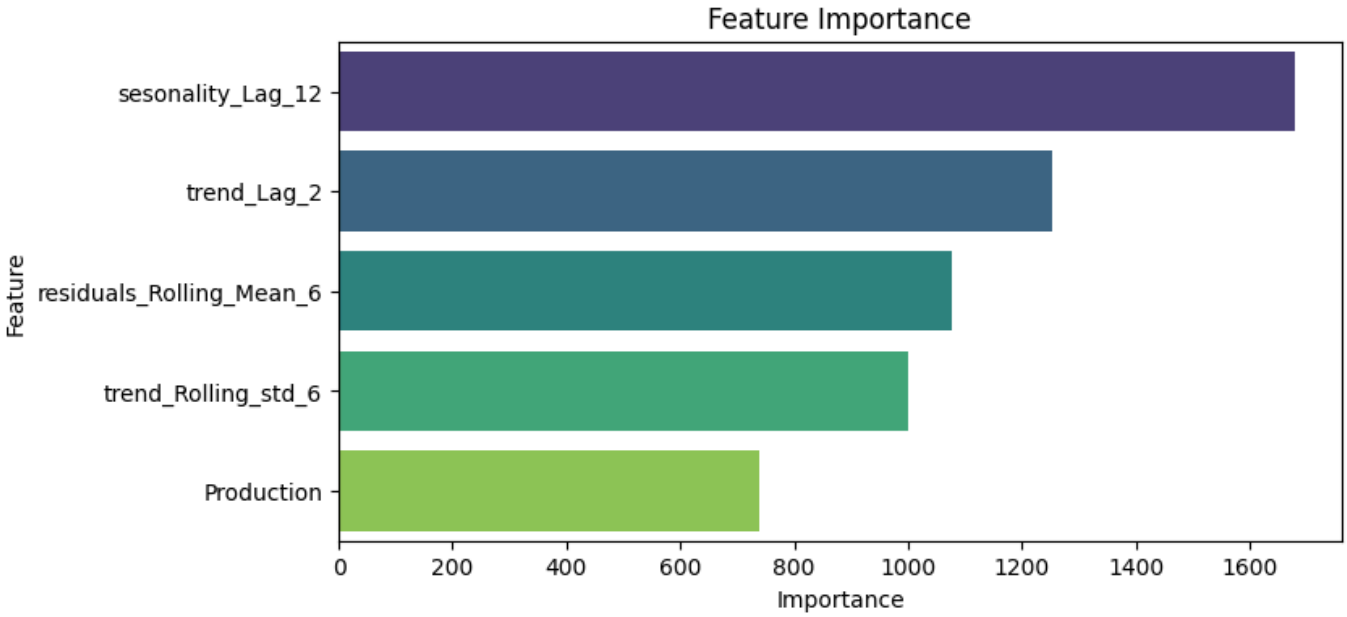
```
Object has been stored at ../models/lightgbm.pkl.
```

Feature Importance Analysis

In [45]:
```
import seaborn as sns

lgb_model = read_pickle(path="../models/lightgbm.pkl")
# Get feature importances
importance = lgb_model.feature_importance()
feature_names = lgb_model.feature_name()
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importa

feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=

# Plot feature importance with a different color palette
plt.figure(figsize=(8, 4))
```

```
sns.barplot(x='Importance', y='Feature', data=feature_importance_df.head(5), palette=
plt.title('Feature Importance')
plt.show()
```

```
Loaded object from ../models/lightgbm.pkl.
```



Based on the feature importance plot we can see that the most important features are the seasonal features. This is expected as the data is seasonal and the model is able to capture the seasonality of the data. The following features are the lag based features as well as the historical production levels. Interestingly, COVID-related features had only little impact, with the first such feature **Poland_Rolling_std_4** ranking 54th out of the 351 available features.

In [46]:
```
feature_importance_df.shape[0]
feature_importance_df.reset_index(drop=True, inplace=True)
```

In [50]:
```
# TODO
# add feature analysis of covid
feature_importance_df.head(5)
```

Out[50]:

	Feature	Importance
0	sesonality_Lag_12	1678
1	trend_Lag_2	1254
2	residuals_Rolling_Mean_6	1076
3	trend_Rolling_std_6	1000
4	Production	738

Ensemble Model

Finally, I can create one last model, which will be an average combination of all models.

This is a powerful technique proved by Claeskens *et al*, 2016 in their paper The forecast combination puzzle: A simple theoretical explanation

In [51]:
```
# Join all models in one single dataframe
model_names = ['LIGHTGBM'] + list(df_stats_forecast_melted['forecasting_model'].uniqu
df_forecats = (pd.merge(df_result_lgbm.reset_index(),
                        df_stats_forecast,
```

```
                        on=['ts_key','Timestamp', 'test_frame'],
                        how='inner' ))
```

In [52]:
```
df_ensemble_forecast = pd.melt(
    df_forecats,
    id_vars=['ts_key', 'Timestamp', 'y_true', 'test_frame'],  # Columns to keep
    value_vars=['LIGHTGBM', 'AutoARIMA', 'AutoETS', 'CES', 'SeasonalNaive', 'WindowAv
    var_name='Model',  # Name of the new column for model names
    value_name='y_pred'  # Name of the new column for predicted values
)

df_ensemble_forecast = (df_ensemble_forecast
                        .filter(['ts_key', 'Timestamp','test_frame','Model','y_pred']
                        .groupby(['ts_key', 'Timestamp', 'test_frame'],group_keys=Fal
                            ['y_pred'].mean().to_frame().reset_index())

model_names.append('Ensemble')
df_ensemble_forecast['Model'] = 'Ensemble'

df_ensemble_forecast = df_ensemble_forecast.drop(columns=['Model']).rename(columns={'
```

In [53]:
```
# Verify we joint data correctly, and there are not duplicated forecasts
assert df_forecats.shape[1] != df_ensemble_forecast.shape[1], "There are duplicated f
```

In [28]:
```
df_ensemble_forecast.head()
```

Out[28]:

	ts_key	Timestamp	test_frame	Ensemble
0	Provider_10-Plant_1	2022-01-01	2022-01-01 - 2022-04-01	3.142037
1	Provider_10-Plant_1	2022-02-01	2022-01-01 - 2022-04-01	2.823292
2	Provider_10-Plant_1	2022-03-01	2022-01-01 - 2022-04-01	3.461297
3	Provider_10-Plant_1	2022-04-01	2022-01-01 - 2022-04-01	2.877659
4	Provider_10-Plant_1	2022-05-01	2022-05-01 - 2022-08-01	4.664929

In [55]:
```
df_forecats = (pd.merge(df_forecats,
                        df_ensemble_forecast,
                        on=['ts_key','Timestamp', 'test_frame'],
                        how='inner' ))
```

In [56]:
```
df_ratio_gold = pd.read_parquet(config['preprocessing']['ratio_gold_path'])
df_true = (df_ratio_gold[['ts_key','Timestamp', 'Actual_Vol_[Kg]','Production']]
            .rename(columns={'Actual_Vol_[Kg]':'y_target'
                             }).copy())

evaluation_df = pd.merge(df_forecats.reset_index(),
                         df_true, on=['ts_key','Timestamp'],
                         how='left')

for model_name in model_names:
    evaluation_df[f'{model_name}_target'] = ((evaluation_df[model_name] * evaluation_
```

Results

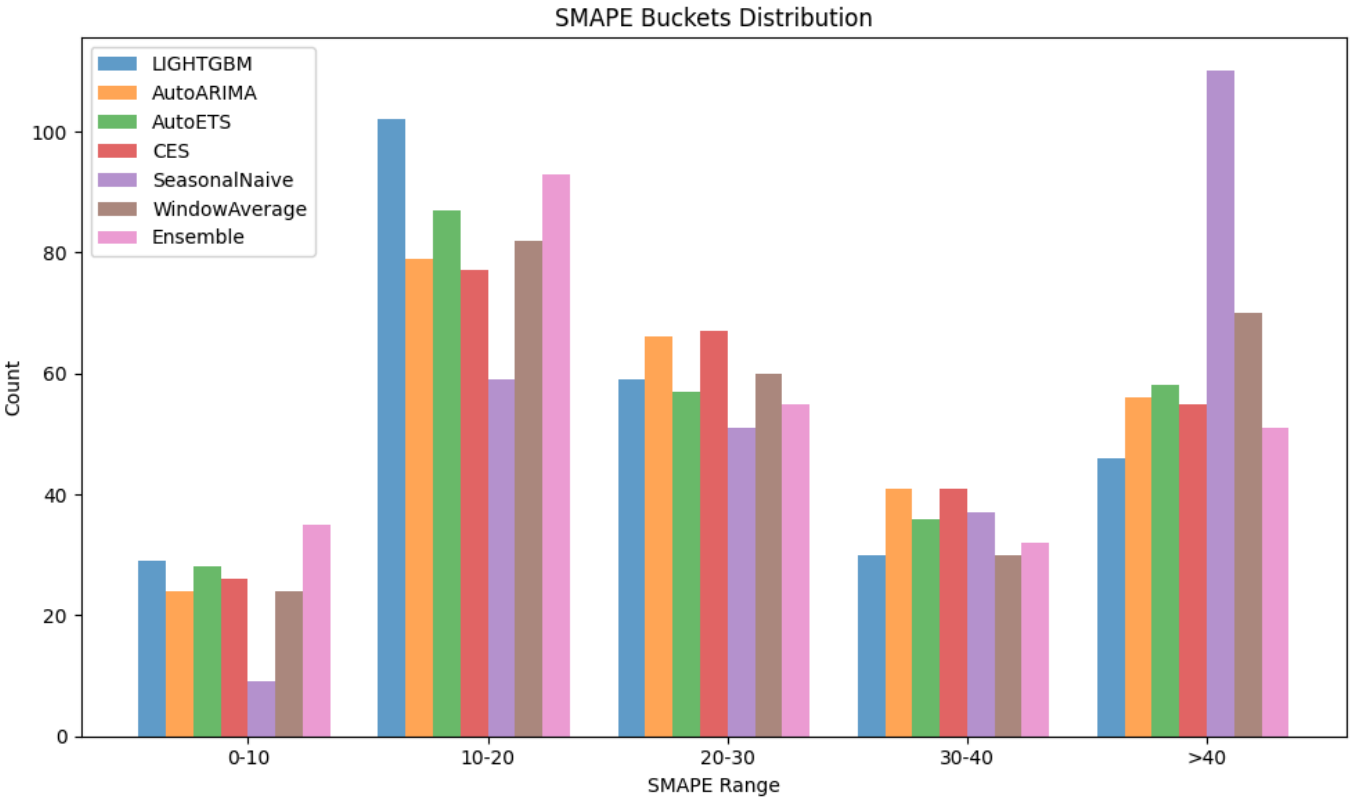
Evaluate Model Accuracy

Now we define the true values to evaluate the accuracy. Since we are using the `Vol/Prod_ratio_kg`. We have to multiply the forecast values with the Production values to get the forecast of `Actual Vol [Kg]`. These values will be called `y_target`. On the other hand, the `y_true` are the true `Vol/Prod_ratio_kg` values.

When using the models to forecast the real business data, this `Production` values are the `Production Planning` provided once a month for the following next 12 months.

Plot SMAPE Intervals

In [68]:
```
df_accuracy_smape, df_accuracy_mae = calculate_accuracy_metrics(evaluation_df, model_
buckets_data = plot_smape_buckets(df_accuracy_smape, model_names)
```



This plot shows us how the different models perform in different SMAPE intervals. We can highlight that lightGBM and Ensemble Model are in the lead, having more forecasts in the lower interval ranges than the other models. We can also spot that the Sesonal Naive model is the worst model, delivering more than 100 timeseries with an SMAPE greater than 40.

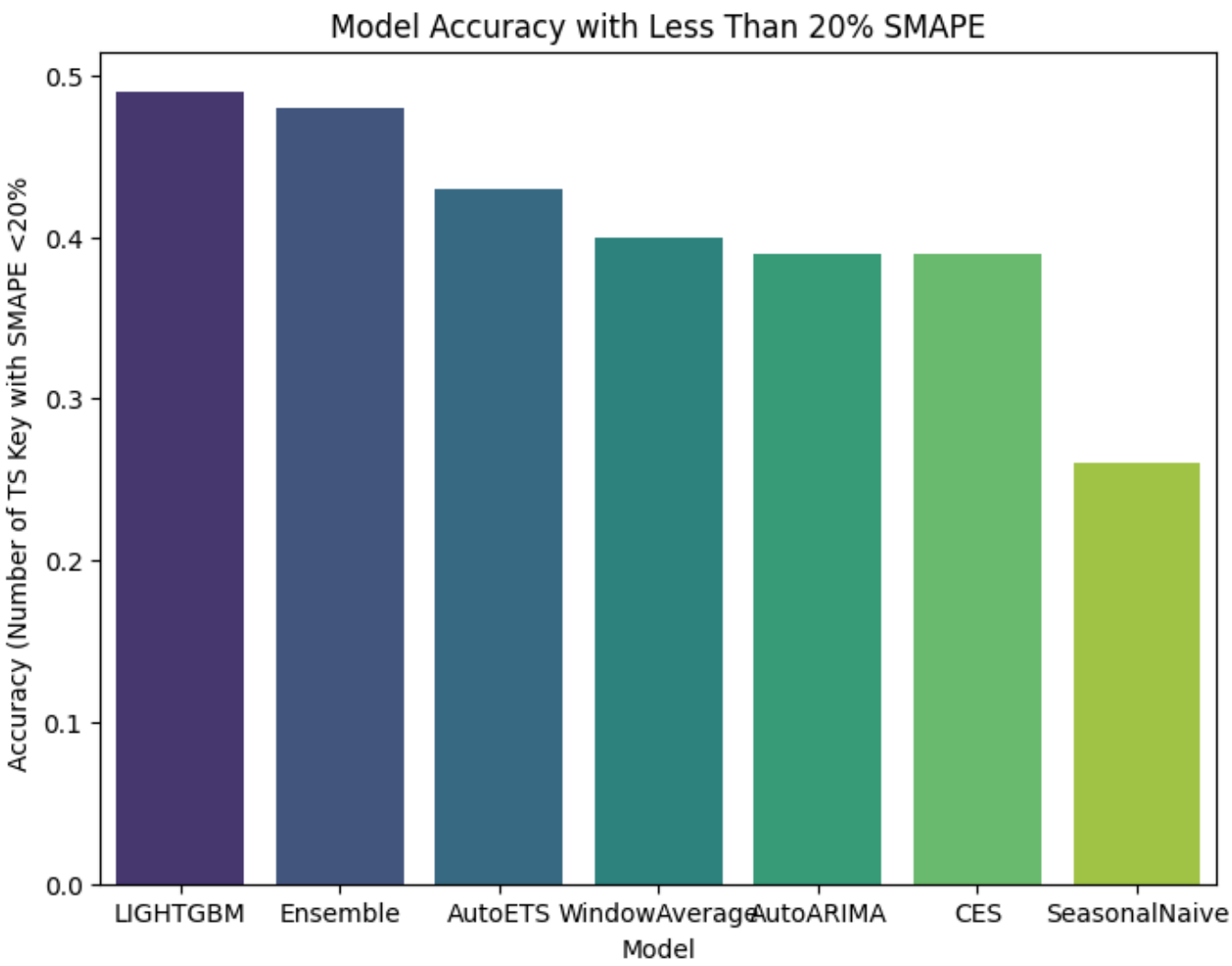
In [66]:
```
df_acc_stats_models = pd.merge(df_accuracy_smape, df_accuracy_mae, on=['test_frame','
df_acc_stats_models = df_acc_stats_models[['test_frame','ts_key','model_name','smape'
df_acc_stats_models[['model_name','test_frame','smape','mae']].groupby(['model_name',
```

Out[66]:

		smape	mae
model_name	**test_frame**		
AutoARIMA	**2022-01-01 - 2022-04-01**	26.032704	45484.217185
	2022-05-01 - 2022-08-01	22.323980	38017.296086
	2022-07-01 - 2022-10-01	25.068538	37749.917859
AutoETS	**2022-01-01 - 2022-04-01**	25.091556	40503.778049
	2022-05-01 - 2022-08-01	21.760051	32077.181910
	2022-07-01 - 2022-10-01	21.386382	34590.415032
CES	**2022-01-01 - 2022-04-01**	25.853458	41729.995234
	2022-05-01 - 2022-08-01	22.966287	32848.836165
	2022-07-01 - 2022-10-01	22.051627	34405.520286
Ensemble	**2022-01-01 - 2022-04-01**	20.266035	34124.811984
	2022-05-01 - 2022-08-01	19.720373	36102.451570
	2022-07-01 - 2022-10-01	23.058853	37710.196662
LIGHTGBM	**2022-01-01 - 2022-04-01**	24.529770	38562.352778
	2022-05-01 - 2022-08-01	18.113385	29095.776417
	2022-07-01 - 2022-10-01	21.365293	34126.068424
SeasonalNaive	**2022-01-01 - 2022-04-01**	30.885929	58590.279216
	2022-05-01 - 2022-08-01	31.831401	59052.629890
	2022-07-01 - 2022-10-01	37.614878	64212.855177
WindowAverage	**2022-01-01 - 2022-04-01**	25.120590	40488.944512
	2022-05-01 - 2022-08-01	21.964915	35242.703704
	2022-07-01 - 2022-10-01	24.492937	36821.276669

The previous dataframe shows us how the different models peform in the three test frames. We can spot how the LightGBM Model and the Ensemble model are consistently delivering values with low average SMAPE error.

In [71]:
```
plot_err_less_20_SMAPE(buckets_data=buckets_data)
```



Out[71]:

	model	err_less_20_perc_ts_key
0	LIGHTGBM	0.49
6	Ensemble	0.48
2	AutoETS	0.43
5	WindowAverage	0.40
1	AutoARIMA	0.39
3	CES	0.39
4	SeasonalNaive	0.26

Regarding the Business Accuracy target. We can see that the highest Number of timeseries with an SMAPE error of less than 20% is achieved by the LightGBM Model, with more than 50% of the forecast values, followed by the Ensemble Model. The worst performing model is the Seasonal Naive.

Forecating System - Analyze Best Forecasting Model per Timeseries

When it comes to forecasting systems. We can make the models compete to each other. For each timeseries we select the model with the lowest mean `SMAPE` across all test frames. With this we can increase the overall accuracy of all the predictions together, since some models would perform better than others in some timeseries. This is shown in my paper, however there I used a more compelx metric (EWMA SMAPE - Exponentially Weighted Moving Average SMAPE) to pick the best performing model, since I had access to much more data.

In [72]:
```
df_accuracy_smape.head()
```

Out[72]:

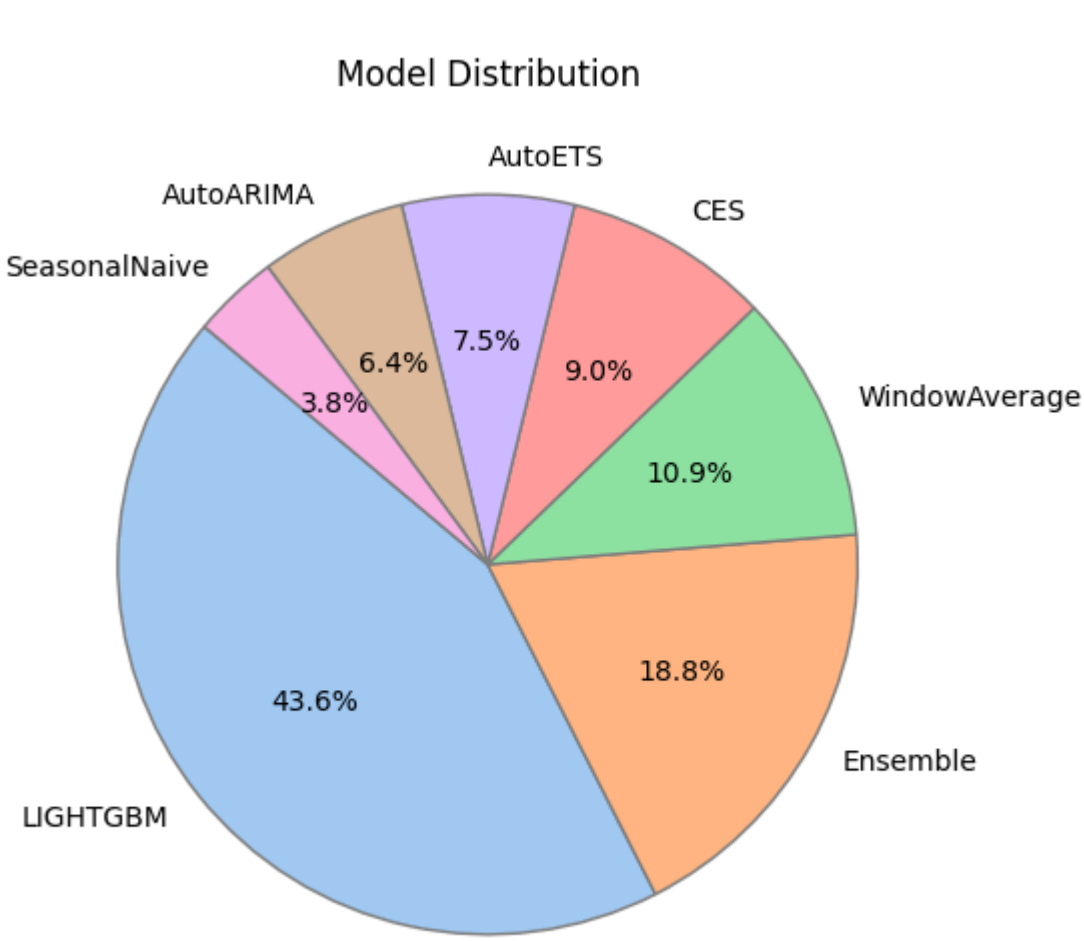
	test_frame	ts_key	smape	model_name
0	2022-01-01 - 2022-04-01	Provider_10-Plant_1	32.058585	LIGHTGBM
1	2022-01-01 - 2022-04-01	Provider_10-Plant_10	15.874883	LIGHTGBM
2	2022-01-01 - 2022-04-01	Provider_10-Plant_11	29.938502	LIGHTGBM
3	2022-01-01 - 2022-04-01	Provider_10-Plant_12	28.696125	LIGHTGBM
4	2022-01-01 - 2022-04-01	Provider_10-Plant_13	40.988182	LIGHTGBM

In [73]:
```
df_accuracy_smape['model_name'].unique()
```

Out[73]:
```
array(['LIGHTGBM', 'AutoARIMA', 'AutoETS', 'CES', 'SeasonalNaive',
       'WindowAverage', 'Ensemble'], dtype=object)
```

In [74]:
```
# mean of SMAPE over all test frames
df_best_models = df_accuracy_smape.groupby(['ts_key','model_name'])['smape'].mean().t
df_best_models['best_smape'] = df_best_models.groupby(['ts_key'])['smape'].transform(
df_best_models = df_best_models[df_best_models['smape'] == df_best_models['best_smape
df_model_per_ts  = df_best_models['model_name'].value_counts().to_frame().reset_index
```

In [77]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot
plt.figure(figsize=(6, 6))
colors = sns.color_palette("pastel", len(df_model_per_ts))

plt.pie(df_model_per_ts['count'], labels=df_model_per_ts['model_name'], autopct='%1.1

plt.title('Model Distribution')
plt.show()
```



The previous plot shows us a clear picture on which model is picked for which proportion of timeseries. 44.7% of timeseries are showing the lowest SMAPE when using the LightGBM Model, meanwhile 15.4% of timeseries perform the best when using the Ensemble model. Even though the seasonal Naive model is the worst model. This analysis shows us that 2.6% (7 timeseries) in total show better results when using this model instead of any other.

Average Accuracy of Best Forecasting Models - Forecasting System

In [78]:
```
df_all_forecast = pd.melt(
    df_forecats,
    id_vars=['ts_key', 'Timestamp', 'y_true', 'test_frame'],
    value_vars=['LIGHTGBM', 'AutoARIMA', 'AutoETS', 'CES', 'SeasonalNaive', 'WindowAv
    var_name='model_name',
    value_name='y_pred'
)

df_best_forecast = pd.merge(df_all_forecast,
                        df_best_models[['ts_key','model_name']]
                        .rename(columns={'model_name':'best_model_name'}),
                        on='ts_key',
                        how='left')

df_best_forecast = df_best_forecast[df_best_forecast['model_name'] == df_best_forecas
```

In [79]:
```
evaluation_df = pd.merge(df_best_forecast,
                        df_true, on=['ts_key','Timestamp'],
                        how='left')

evaluation_df['best_y_hat'] = evaluation_df['y_pred'] * evaluation_df['Production']
```

Calculate MAE and SMAPE of Forecasting System

In [80]:
```
df_accuracy_smape = (evaluation_df.groupby(['test_frame','ts_key'], group_keys=False)
            .apply(lambda x: smape(x['y_target'], x['best_y_hat']))
            .to_frame()
            .rename(columns={0:'smape'})
            .reset_index()
            )
df_accuracy_smape['model_name'] = 'best_model'

df_accuracy_mae = (evaluation_df.groupby(['test_frame','ts_key'], group_keys=False)
        .apply(lambda x: mean_absolute_error(x['y_target'], x['best_y_hat']))
        .to_frame()
        .rename(columns={0:'mae'})
        .reset_index()
        )
df_accuracy_mae['model_name'] = 'best_model'
```

In [81]:
```
df_acc_stats_models = pd.merge(df_accuracy_smape, df_accuracy_mae, on=['test_frame','
df_acc_stats_models = df_acc_stats_models[['test_frame','ts_key','model_name','smape'
df_acc_stats_models[['model_name','test_frame','smape','mae']].groupby(['model_name',
```

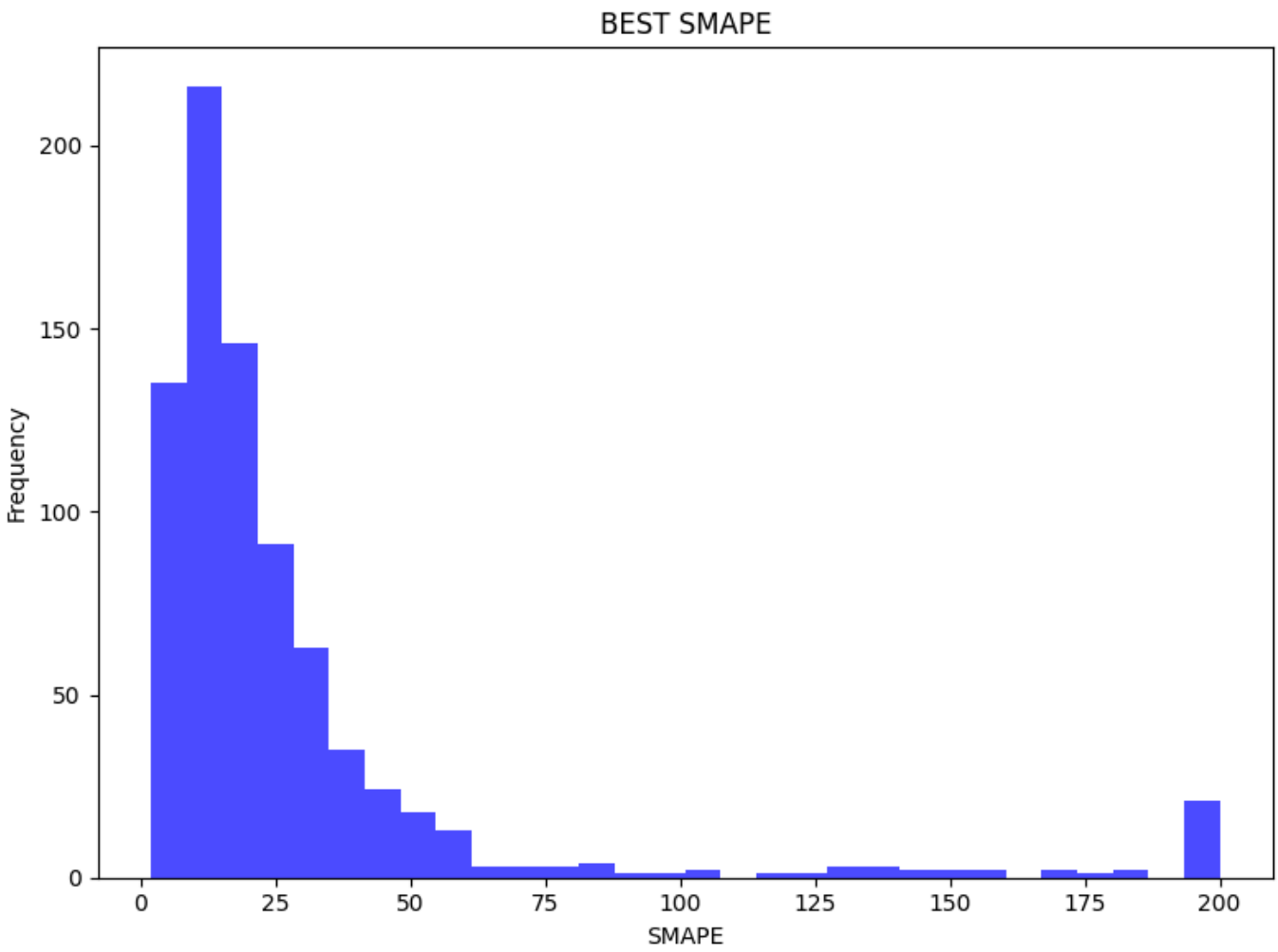
Out[81]:

		smape	mae
model_name	**test_frame**		
best_model	**2022-01-01 - 2022-04-01**	17.451533	29648.893874
	2022-05-01 - 2022-08-01	15.469940	26534.309421
	2022-07-01 - 2022-10-01	17.485574	31269.996501

This analysis proves our previous statement that combining the models into a Forecasting System helps improve the overall average SMAPE for all the timeseries. Which in our cases is the target of the business team.

Plot Accuracy Results - Forecasting System

In [82]:
```
# Prepare the data
data = df_accuracy_smape['smape']

# Create the figure for a single histogram
fig, ax = plt.subplots(figsize=(8, 6))

# Plot data
ax.hist(data, bins=30, color='blue', alpha=0.7)
ax.set_title('BEST SMAPE')
ax.set_xlabel('SMAPE')
ax.set_ylabel('Frequency')

# Display the plot
plt.tight_layout()
plt.show()
```

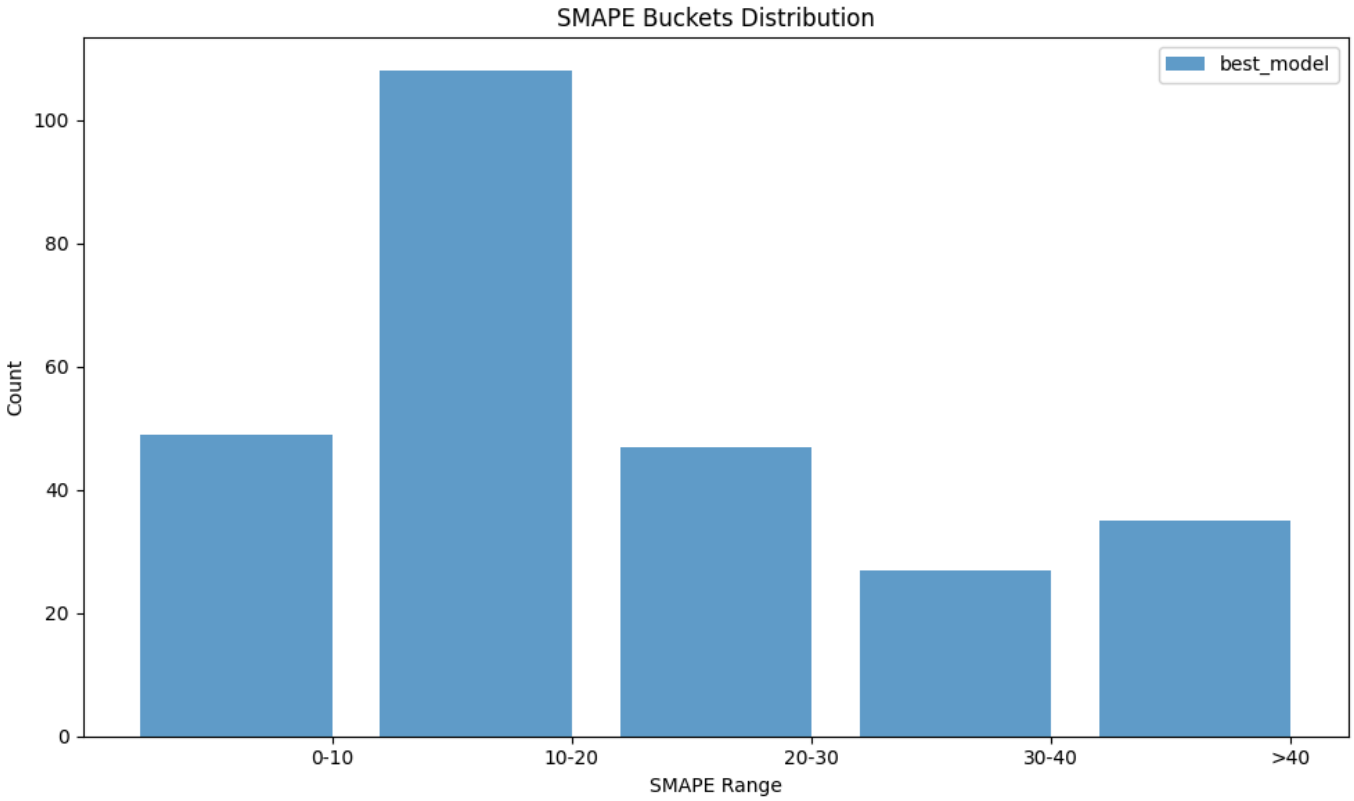


The histogram illustrates that the forecasting system achieves accurate predictions for most time series (low SMAPE), with some few outliers causing larger errors. The high frequency of low SMAPE values highlights the reliability of the selected models, while the outliers suggest areas where improvements may be needed for specific types of time series. This shows that our forecasting system is coming closer to the target, to get as much as possible error below the 20% SMAPE threshold.

Plot SMAPE Buckets Distributions

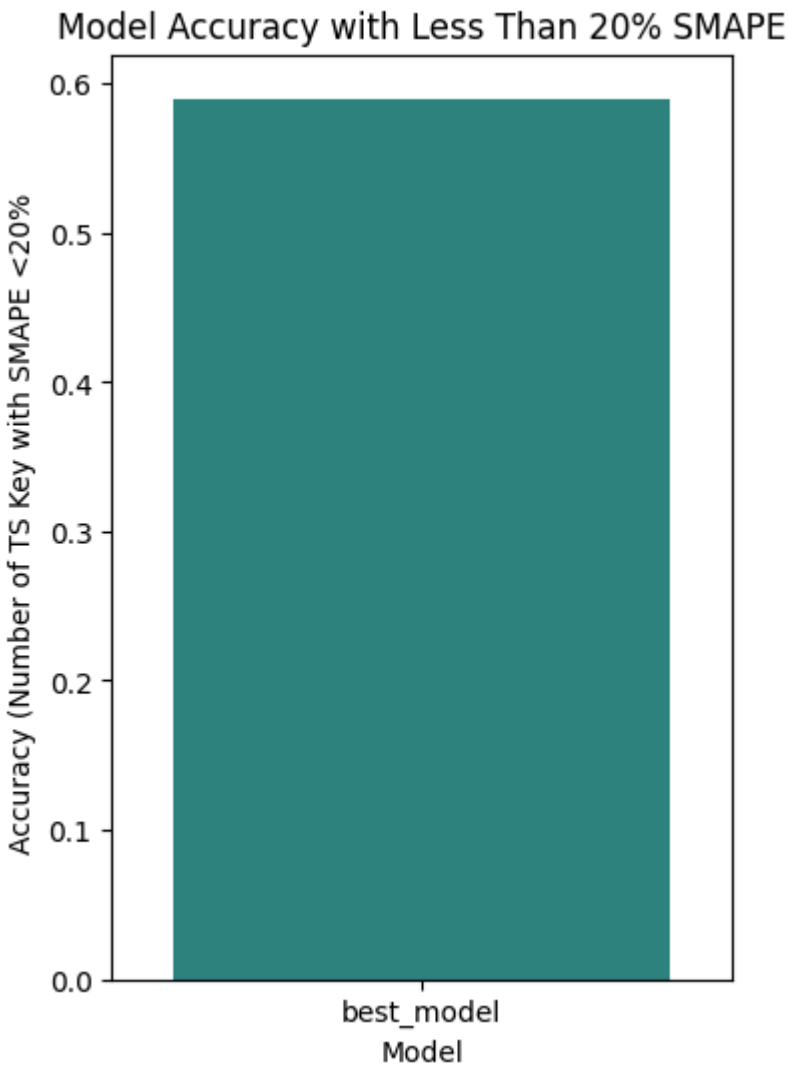
In [83]:
```
# Define SMAPE buckets
bins = [0, 10, 20, 30, 40, float('inf')]
labels = ['0-10', '10-20', '20-30', '30-40', '>40']
model_names = ['best_model']
buckets_data = generate_smape_err_buckets(df_accuracy=df_accuracy_smape,
                                          model_names=model_names,
                                          bins=bins,
                                          labels=labels)

colors = ['#7f7f7f', '#e377c2', '#8c564b', '#9467bd', '#d62728', '#2ca02c', '#ff7f0e'

x = np.arange(len(labels))  # x locations for the buckets
width = 0.8 / len(model_names)  # Adjust width based on number of models

fig, ax = plt.subplots(figsize=(10, 6))

for i, (model_name, data) in enumerate(buckets_data.items()):
    ax.bar(x + i * width - width * len(model_names) / 2, data.values, width, label=mo

# Add labels and title
ax.set_xlabel('SMAPE Range')
```

```
ax.set_ylabel('Count')
ax.set_title('SMAPE Buckets Distribution')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.tight_layout()
plt.show()
```



We can see that our system is performing quite well and most timeseries are falling below the 20% threshold. The next chart shows us this result better.

In [84]:
```
df_acc_less_20 = plot_err_less_20_SMAPE(buckets_data=buckets_data, figsize=(4,6))
```



In [85]:
```
df_acc_less_20
```

Out[85]:

	model	err_less_20_perc_ts_key
0	best_model	0.59

Finally, we can see that the forecasting system reaches a high accuracy level, with 61% of all timeseries with less than 20% of SMAPE.

Conclusions

Forecasting System Performance

With the selected best-performing models, we achieve a prediction error of less than 20% for 61% of all timeseries. That means 162 out of 266 time series. This is a good starting point for this simple forecasting system.

We observed the significant advantage of selecting the best-performing model across different test frames. This approach builds a robust system capable of handling outliers effectively while maintaining high accuracy. By leveraging the strengths of each model, we achieve a blended system that maximizes overall performance.

The results demonstrate that the LightGBM model is the dominant performer, delivering the lowest SMAPE values for 44.7% of the timeseries. The second-best model was the Ensemble, which balances multiple methods to perform well across various scenarios. Surprisingly, among the

statistical models, the simple WindowAverage outperformed more complex models like ARIMA and ETS in terms of the number of time series for which it delivered the best performance.

This finding underscores that in some cases, simpler models like WindowAverage can outperform more complex methods when data variability causes advanced models to overfit. This reinforces the importance of model selection and adaptability when dealing with diverse timeseries data.

Based on the paper Forecasting System for Inbound Logistics Material Flows at an International Automotive Company, the performance of the forecasting system version 3.0 was 97% for all time series that had a prediction error of less than 20%. This is due to several factors that were not included in this analysis for simplicity:

- **Selection of Best Performing Model**: I used the SMAPE to measure model accuracy, but the paper uses an Exponentially Weighted Moving Average SMAPE that takes into account the "age" of the error. The older the error, the less important it is in determining the best performing model. This exponential decay is based on a decay factor selected with the business experts.
- **Test Time Frames**: For simplicity, we analyzed only three test frames, namely: Jan 2022 - Apr 2022, May 2022 - Aug 2022, Jul 2022 - Oct 2022. However, the forecasting system in the paper had access to much more test data, as the system has been active since 2018.
- **Hyperparameter Tuning**: Hyperparameter tuning is a powerful tool for selecting the best parameters for each model, but it is also time-consuming. For simplicity, I let most models choose the best parameters using the automated routines provided by their libraries. Except for the tree-based models, where I used the package optuna. This is one additional reason why LightGBM were the best performing algorithms.
- **Analysis of Historical Production Planning**: For simplicity, we have only included a single production planning record. However, this file is available every month. As explained in the paper, if one analyzes the historical errors of the company's production planning and adds them as features to all models, the accuracy will improve significantly.
- **Model Pool**: So far I have only explored *statistical models* and *machine learning models*, however we still can explore *deep learning models* and *LLM foundation models* like Chronos, which might also deliver top performance. I will explore those in the final report.

These are definitely considerations that can be added to improve the overall performance of this forecasting system with Python.

Analysis Feature Importance

The feature importance analysis revealed that seasonal features are the most significant, as expected given the seasonality of the data and the model's ability to capture it. Lag-based features and historical production levels follow in importance. Interestingly, COVID-related features had only little impact, with the first such feature **Poland_Rolling_std_4** ranking 54th out of the 351 available features.

Data Quality and Data Preparation

Regarding **data quality and data preparation**, it is important to emphasize that this is a very important step before any model can be trained. My comments are:

- Involve the business experts to validate any inconsistencies in the data, such as outliers or inaccurate values.
- Agree with business experts on thresholds that may affect the output, e.g., time series length.
- Educate business experts about the limitations of predictive algorithms and create contingency plans for outlier processes.

In this project I could experience what Forbes stated in its article `Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task` Ref that in machine learning (ML) projects, a significant portion of the effort - often as much as 60% - is devoted to data engineering tasks such as data collection, cleaning, and preparation. This is because these phases ensure that the data is both accurate and usable, which is critical to creating meaningful ML models. Surveys from sources such as Anaconda and Kaggle have consistently shown that data scientists spend a large portion of their time on these tasks. Specifically, Anaconda found that data cleaning can take more than 45% of the time on many projects, and according to O'Reilly, handling upstream data unification and cleaning often requires complex ML pipelines, especially in large-scale data environments. It is important not to underestimate this critical step and plan accordingly.

Hypothesis Analysis

At the begining, we stated the hypothesis: "There are new forecasting methods which can deliver better accuracy than traditional statistical methods".

The results thus far have demonstrated that the LightGBM model consistently outperforms traditional statistical models such as ARIMA, ETS, CTS, and Window Average. This highlights the exceptional capability of tree-based models in the context of forecasting. This finding aligns with one of the key insights from the M5 Forecasting Competion , which also emphasized the effectiveness of machine learning approaches over conventional statistical methods.