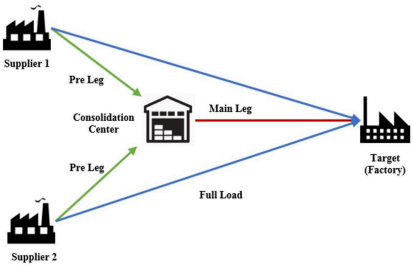
Report: Inbound Logistics Forecasting Benchmark

Introduction and data



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. For all main legs of the Area-Forwarding based Inbound Logistics Network, as shown in the image above. 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 warnings
warnings.filterwarnings('ignore')
import sys, os
sys.path.insert(0, os.path.abspath('..'))

# Import all the necessary libraries
from src.project_imports import *
```

Data Dictionary

Historical Volume Data

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

Name	Description	Role	Type	Format
Timestamp	Monthly date of the format YYYY-MM-DD	ID	ordinal	datetime
Provider	Logistics Provider ID	ID	nominal	category
Plant	Assembly Plant ID	ID	nominal	category
Actual_Vol_[Kg]	Actual transported volume from Provider to Plant in kg	response	numeric	float
Expected_Vol_[Kg]	Expected transported volume from Provider to Plant in kg	predictor	numeric	float
Year	Year in which transport took place	predictor	numeric	int
Month	Month in which transport took place	predictor	numeric	int
ts_key	Timeseries key	ID	numeric	category
Actual_Vol_[Tons]	Actual transported volume from Provider to Plant in tons	predictor	numeric	float
Expected_Vol_[Tons]	Expected transported volume from Provider to Plant in tons	predictor	numeric	float

Historical Production Planning

In [3]:
```
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]:
```
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

The modelling process is based on the approach used in my paper Forecasting System. That is, fitting several different models to the same training set and evaluating them on the same test set. Comparing their accuracy on the Symmetric Mean Absolute Error (SMAPE). Specifically, using the following intervals: 0% to 10%, 10 to 20%, 20 to 30%, 30 to 40%, greater than 40%. The economists are particularly interested in a forecasting system for which most of the time series have a SMAPE of less than or equal to 20%.

Finally, for each time series, the best performing model on the three test sets is selected. This creates what is known as a forecasting system in which each timeseries forecast is generated by the best performing model on the test sets.

We will evaluate the performance of 4 different types of models: Three based models, statistical models, deep learning models and LLM Foundational Models.

In order to control the flow of this report we will make use of a **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.

In [5]:
```
# Import Data
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 Preparation & Data Quality

In this step we adjust the data as well as apply multiple data quality checks, and data completeness to make sure that the timeseries contain data at all timestamps and that they all have the same length.

Preparation Historical Volume Data

In [6]:
```
df_vol_bronze = df_vol.copy()
df_vol_gold = data_preparation_and_data_quality(config=config, df_vol_bronze=df_vol_bronze)
```

```
The historical transport volume data contains data since 2014-01-01 00:00:00  until  2022-10-01 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.
 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.

Preparation Production Data

In [7]:
```
df_prod_bronze = df_prod.copy()
df_prod = preparation_production_data(config=config, df_prod_bronze=df_prod_bronze)
```

```
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
```

Preparation Covid Data

In [8]:
```
df_covid_bronze = df_covid.copy()
df_covid = preprocesing_covid(df_covid=df_covid_bronze)
df_covid.to_parquet(config['preprocessing']['covid_silver_path'])
```

```
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.
```

Exploratory Data Analysis

In [9]:
```
df_vol_gold.head(2)
```

Out[9]:

	Timestamp	ts_key	Actual_Vol_[Tons]	Expected_Vol_[Tons]	Plant
0	2014-01-01	Provider_10-Plant_10	476.199005	482.835999	Plant_10
1	2014-02-01	Provider_10-Plant_10	388.113007	388.118011	Plant_10

The Response variable is the **Actual_Vol_[Tons]**. However, the **Production Planning Data** 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]`.

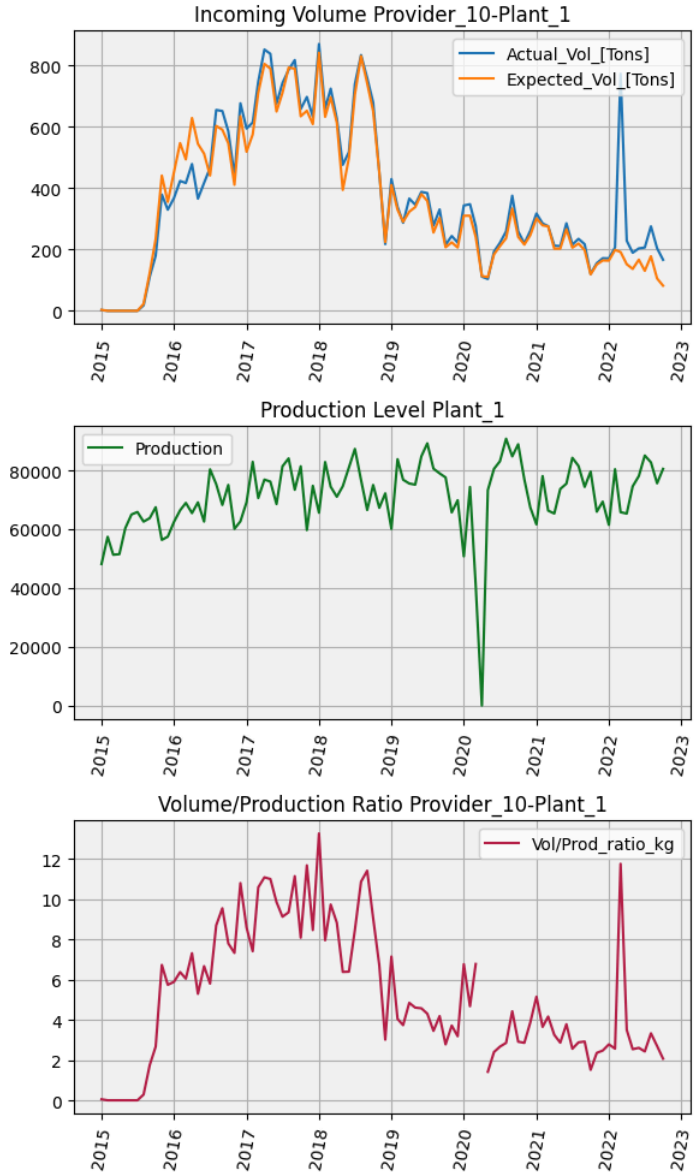
Response Variable

Merge Production Information to Historical Inbound Volume Data

In [10]:
```
df_ratio_gold = generate_vol_prod_ratio_gold(df_vol_gold=df_vol_gold, df_prod=df_prod)
```

Volume/Production Ratio Analysis

In [11]:
```
ts_key = 'Provider_10-Plant_1'
plot_ratio_vol_prod(ts_key=ts_key, df_ratio=df_ratio_gold)
```



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. One technique we can use to solve this problem is to apply a first-oder linear interpolation, this is demonstrated to be most effective that non-linear interpolation Ref.

In [12]:
```
# Apply 1st order polynomial interpolation
df_ratio_gold['Vol/Prod_ratio_kg'] = df_ratio_gold['Vol/Prod_ratio_kg'].interpolate(method='polynomial', order=1)
# Store ratio data to Gold layer
df_ratio_gold.to_parquet(config['preprocessing']['ratio_gold_path'])
# Create PDF Reaport of all ratios
#plot_ratio_all_ts(df_ratio=df_ratio_gold, path=config['preprocessing']['pdf_report_ratios_path'])
```

The PDF Report `Timeseries_Vol_Prod_Ratio.pdf` contains all the timeseries plots. This can be analyzed to understand a bit deeper particular patterns in the data.

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. 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 [13]:
```
df_ts_decomposition = features_seasonal_decomposition(df_ratio_gold=df_ratio_gold, target_col=config['feature_eng']['target_col'])
df_ts_decomposition.to_parquet(config['preprocessing']['seasonal_feat_gold_path'])
df_ts_decomposition.head(2)
```

Out[13]:

	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

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 [14]:
```
df_timeseries_gold = main_feature_engineering(config=config)
df_timeseries_gold.head(2)
```

Out[14]:

	Timestamp	ts_key	Plant	Production	Vol/Prod_ratio_kg	ts_len	Provider	Month	Year	Vol/Prod_ratio_kg_Lag_2	...	Spain_Rolling_std_1
0	2015-01-01	Provider_10-Plant_1	Plant_1	48144	0.04869	94	Provider_10	1	2015	0.04869	...	0.
1	2015-02-01	Provider_10-Plant_1	Plant_1	57400	0.00000	94	Provider_10	2	2015	0.04869	...	0.

2 rows × 320 columns

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 [15]:
```
# 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(2022,4,1)],
    [datetime(2021,12,1), datetime(2022,1,1), datetime(2022,4,1), datetime(2022,5,1), datetime(2022,8,1)],
    [datetime(2022,2,1), datetime(2022,3,1), datetime(2022,6,1), datetime(2022,7,1), datetime(2022,10,1) ],
]
```

Models

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.

Machine Learning Models

We will try out the Machine Learning Model known as boosted trees, in particular the algorithm 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

Deep Learning Models

NeuralForecast offers a large collection of neural forecasting models focused on their usability, and robustness. The models range from classic networks like MLP, RNNs to novel proven contributions like NBEATS, NHITS, TFT and other architectures. link

N-BEATS (Neural Basis Expansion Analysis Time Series) is a deep learning model specifically designed for time series forecasting, focusing on trend and seasonality extraction through basis expansion.

N-HiTS (Neural Hierarchical Interpolation for Time Series) is a neural model that improves hierarchical time series prediction accuracy by generating fine-grained interpolations.

TFT (Temporal Fusion Transformer) is a model designed for interpretable, multivariate time series forecasting that combines temporal and static variables, using attention mechanisms to adaptively focus on relevant time points. However, we won`t be able to implement this model on the Mac ARM achitecture since the MPS device is still not supported in certain operations mps_issue. Therefore we will use a Docker DevContainer.

LLM Models

AWS Model: Chronos Bolt: Chronos-Bolt is a family of pretrained time series forecasting models which can be used for zero-shot forecasting. It is based on the T5 encoder-decoder architecture and has been trained on nearly 100 billion time series observations. It chunks the historical time series context into patches of multiple observations, which are then input into the encoder. The decoder then uses these representations to directly generate quantile forecasts across multiple future steps—a method known as direct multi-step forecasting. Chronos-Bolt models are up to 250 times faster and 20 times more memory-efficient than the original Chronos models of the same size. Ref

Salesforece Model: Morai MOE: Moirai, the Masked Encoder-based Universal Time Series Forecasting Transformer is a Large Time Series Model pre-trained on LOTSA data. Moirai-Mo is the first mixture-of-experts time series foundation model, achieving token-level model specialization in a data-driven manner. Extensive experiments on 39 datasets reveal that Moirai-MoE delivers up to 17% performance improvements over Moirai at the same level of model size and outperforms other time series foundation models Ref

In [61]:
```
stats_models, df_stats_forecast = main_stats_models(df_timeseries_gold=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
Object has been stored at ../models/stats_forecast.pkl.
```

In [62]:
```
lgbm_model, df_result_lgbm = main_lightgbm(df_timeseries_gold=df_timeseries_gold, shards=shards)
```

```
Train-Testing for 2022-01-01 2022-04-01
Best hyperparameters: {'learning_rate': 0.09260476161219532, 'num_leaves': 572, 'subsample': 0.2720140768577476, 'colsample_bytree':
0.5367103814708704, 'min_data_in_leaf': 86}
Best MAE: 2.7300686223038437
Train-Testing for 2022-05-01 2022-08-01
Best hyperparameters: {'learning_rate': 0.035268402255054156, 'num_leaves': 602, 'subsample': 0.998653106682182, 'colsample_bytree':
0.491474567857771, 'min_data_in_leaf': 56}
Best MAE: 4.096085600436597
Train-Testing for 2022-07-01 2022-10-01
Best hyperparameters: {'learning_rate': 0.09661309686765497, 'num_leaves': 806, 'subsample': 0.0620562262722254, 'colsample_bytree':
0.6857407119477501, 'min_data_in_leaf': 97}
Best MAE: 4.196667573807564
Object has been stored at ../models/lgbm_forecast.pkl.
```

In [ ]:
```
nf_model, df_result_deepl = main_deepl_models(df_timeseries_gold=df_timeseries_gold, shards=shards)
```

In [16]:
```
df_result_chronos = main_chronos(df_timeseries_gold=df_timeseries_gold, shards=shards)
```

```
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers v4.48.0. You should pass an instance of `Encod
erDecoderCache` instead, e.g. `past_key_values=EncoderDecoderCache.from_legacy_cache(past_key_values)`.
Train-Testing for 2022-01-01 2022-04-01
Train-Testing for 2022-05-01 2022-08-01
Train-Testing for 2022-07-01 2022-10-01
```

Morai Model

To run Morai model, create a separate python env with the file **requirements/morai_requirements.txt** then run:

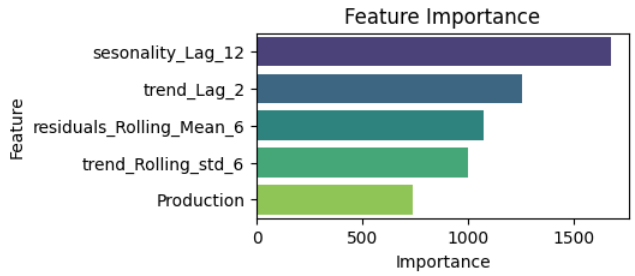
```
python forecast_morai.py
```

In [6]:
```
# Load all Results
df_stats_forecast = pd.read_parquet("../data/forecasts/stats_forecast.parquet")
df_result_lgbm = pd.read_parquet("../data/forecasts/lightgmb_forecast.parquet")
df_result_deepl = pd.read_parquet("../data/forecasts/deepl_forecast.parquet")
df_result_chronos = pd.read_parquet("../data/forecasts/chronos_bolt_forecast.parquet")
df_result_morai = pd.read_parquet("../data/forecasts/morai_forecast.parquet")
```

Feature Importance Analysis

In [27]:
```
# TODO: Analyze COVID Impact
model_path="../models/lightgbm.pkl"
feature_importance_df = feature_importance_analysis(model_path=model_path, top=5, figsize=(4,2))
```

```
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.

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 [7]:
```
df_true, df_forecats, evaluation_df, model_names = ensemble_model(config=config,
                                                  df_result_lgbm=df_result_lgbm,
                                                  df_stats_forecast=df_stats_forecast,
                                                  df_result_deepl=df_result_deepl,
                                                  df_result_chronos=df_result_chronos,
                                                  df_result_morai=df_result_morai)
```

In [8]:
```
evaluation_df.head(2)
```

Out[8]:

	index	ts_key	Timestamp	y_true	LIGHTGBM	test_frame	AutoARIMA	AutoETS	CES	SeasonalNaive	...	AutoETS_target	CES_targ
0	0	Provider_10-Plant_1	2022-01-01	2.78201	4.079643	2022-01-01 - 2022-04-01	2.423550	2.37877	2.378803	5.15235	...	146158.783727	146160.79363
1	1	Provider_10-Plant_1	2022-02-01	2.56811	3.616801	2022-01-01 - 2022-04-01	2.444547	2.37877	2.378812	3.65084	...	191215.072035	191218.4298

2 rows × 33 columns

Results

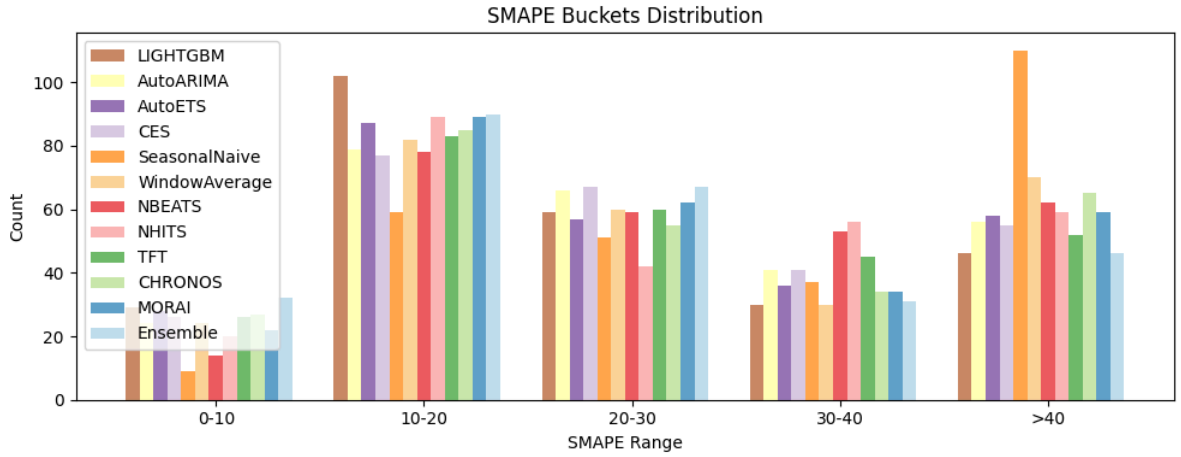
Evaluate Model Accuracy

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 [30]:
```
df_accuracy_smape, df_accuracy_mae = calculate_accuracy_metrics(evaluation_df, model_names)
buckets_data = plot_smape_buckets(df_accuracy_smape=df_accuracy_smape, model_names=model_names, figsize=(10,4))
```



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.

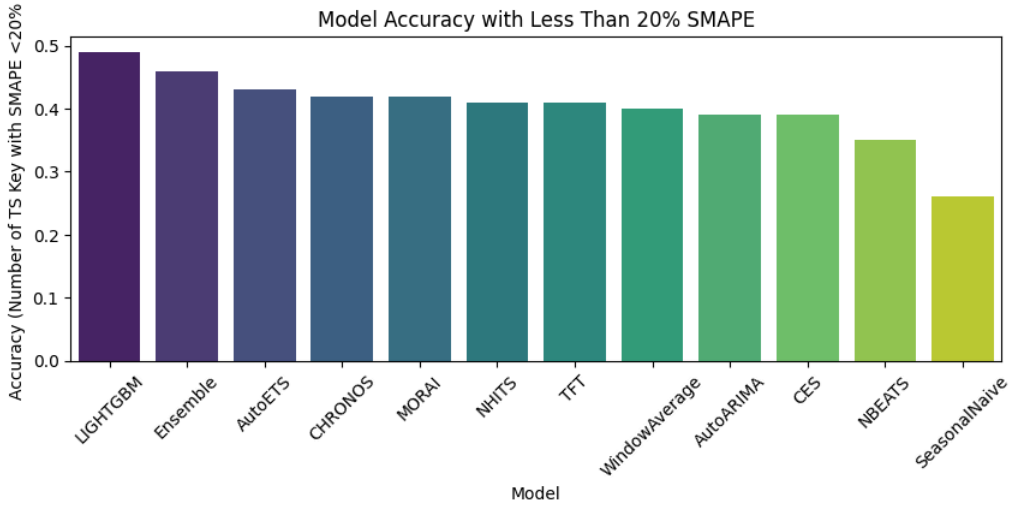
In [31]:
```
df_acc_stats_models = pd.merge(df_accuracy_smape, df_accuracy_mae, on=['test_frame','ts_key','model_name'], how='left')
df_acc_stats_models = df_acc_stats_models[['test_frame','ts_key','model_name','smape','mae']].copy()
df_acc_stats_models[['model_name','smape','mae']].groupby(['model_name']).agg('median').sort_values(by='smape', ascending=True)
```

Out[31]:

model_name	smape	mae
LIGHTGBM	20.859102	33736.530003
Ensemble	21.539527	36284.052896
AutoETS	22.290389	35840.304868
CES	23.224507	36491.425524
TFT	23.425203	36427.128403
WindowAverage	23.587861	36878.734352
MORAI	23.790733	36892.736424
AutoARIMA	24.229489	40510.010984
CHRONOS	24.840997	37413.759097
NHITS	26.817846	36743.909294
NBEATS	27.682582	39243.517394
SeasonalNaive	32.709670	60456.741264

The previous dataframe shows us the average (median) performance of the models 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 [34]:
```
df_acc_less_20 = plot_err_less_20_SMAPE(buckets_data=buckets_data, figsize=(10,3.5))
```

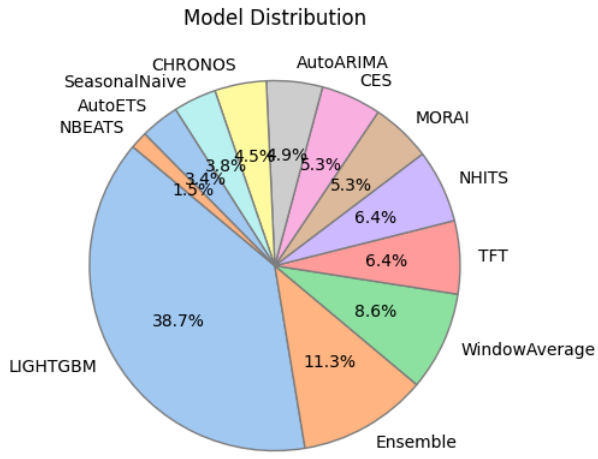


Regarding the Business Accuracy target. We can see that the highest Number of timeseries with an SMAPE error of less than 20% is achieved by the LightGBM Model, with more than 50% of the forecast values, followed by the Ensemble Model. The worst performing model is the Seasonal Naive.

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 complex metric (EWMA SMAPE - Exponentially Weighted Moving Average SMAPE) to pick the best performing mode.

In [36]:
```
df_best_models, df_model_per_ts = calculate_best_models(df_accuracy_smape)
plot_model_distribution(df_model_per_ts=df_model_per_ts,figsize=(5,5))
```



The previous plot shows us a clear picture on which model is picked for which proportion of timeseries. 38.7% of timeseries are showing the lowest SMAPE when using the LightGBM Model, meanwhile 11.3% 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 3.8% (10 timeseries) in total show better results when using this model instead of any other.

Average Accuracy of Best Forecasting Models - Forecasting System

In [17]:
```
f_system_evaluation_df = forecast_system_evaluation_df(df_forecats, df_best_models, df_true)
```

Calculate MAE and SMAPE of Forecasting System

In [18]:
```
df_accuracy_smape, df_accuracy_mae = forecast_system_accuracy_metrics(evaluation_df=f_system_evaluation_df)
df_acc_stats_models = pd.merge(df_accuracy_smape, df_accuracy_mae, on=['test_frame','ts_key','model_name'], how='left')
df_acc_stats_models = df_acc_stats_models[['test_frame','ts_key','model_name','smape','mae']].copy()
df_acc_stats_models[['model_name','test_frame','smape','mae']].groupby(['model_name','test_frame']).agg('median')
```

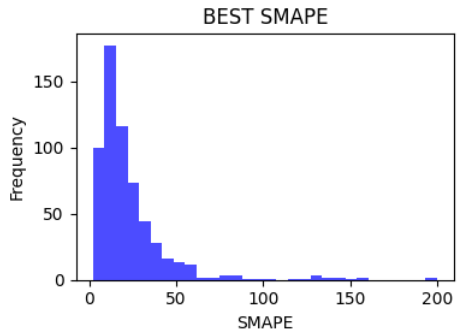
Out[18]:

		smape	mae
model_name	**test_frame**		
best_model	**2022-01-01 - 2022-04-01**	16.483380	28272.534277
	2022-05-01 - 2022-08-01	15.256857	25798.740495
	2022-07-01 - 2022-10-01	17.316490	29279.507427

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

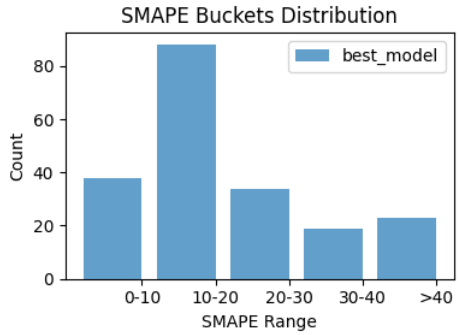
In [21]:
```
plot_smape_histogram(data=df_accuracy_smape['smape'], figsize=(4, 3))
```



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 [22]:
```
# Example usage
buckets_data = plot_f_system_smape_buckets_distribution(df_accuracy_smape=df_accuracy_smape, figsize=(4, 3))
```



We can see that our system is performing quite well and most timeseries are falling below the 20% threshold.

In [80]:
```
df_acc_less_20 = plot_err_less_20_SMAPE(buckets_data=buckets_data, figsize=(2,4), no_plot=True)
df_acc_less_20
```

Out[80]:

	model	err_less_20_perc_ts_key
0	best_model	0.62

Finally, we can see that the forecasting system reaches a high accuracy level, with 62% of all timeseries with less than 20% of SMAPE.

Discussion & Conclusions

Forecasting System Performance

With the selected best-performing models, we achieve a prediction error of less than 20% for 62% of all timeseries. That means 164 out of 266 time series. This is a good starting point for this 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 38.7% 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. In addition, we observed that the LLM models did not perform better than the WindowsAverage model, which is a much simpler model. This showed that LLM models still struggle with timeseries forecasting tasks and consume much more computational resources, as shown by Tan, M., Merrill *et al*, 2024 Ref.

Based on the paper Forecasting System..., 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.
- **Exogenous Variables**: Only LightGBM Model used exogenous data from the feature engineering process. Additional analysis with models like NBEATSx can be carried out to validate the inpact of these features in other models. Ref.
- **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.

Feature Importance Analysis on LightGBM

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 Analysis

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.

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 Competition, which also emphasized the effectiveness of machine learning approaches over conventional statistical methods.