Efficient Model Comparison for Time Series Analysis

YOUNGHOE KIM

younghoe.kim@unibo.it

20 December 2023

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Objectives

Forecasting

Forecasting vs Prediction

What is a time-series data?

What is the difference between a time series and a normal series?

Models for a time-series analysis

Performance Metrics for Time series data

Data

Exploratory Data Analysis

Arima vs Auto Arima vs Sarima

Prophet vs Neural Prophet

MLP vs CNN vs LSTIM vs CNN+LSTM

Conclusion

Advantages & Disadvantage of the Models

References

Objectives

Efficient model comparison for time series analysis is a crucial aspect of building accurate and effective predictive models. Time series data involves observations collected over time, and analyzing such data requires specialized techniques that account for temporal dependencies and trends.

By comparing a range of models, the project aims to discern which model is most effective for a given time series dataset and to identify the model that enhances predictive accuracy. The project seeks to analyze the strengths and limitations of each model, enabling an understanding of which model is more effective in specific scenarios.

An in-depth understanding of the strengths and limitations of each model will guide the selection of the most suitable model based on specific scenarios and data types.

This project consists of comprehensive Exploratory Data Analysis with different models for Store sales forecasting Challenge using 3 year history.

Forecasting

Many people wrongly assume that forecasts are not possible in a changing environment.
Every environment is changing, and a good forecasting model captures the way in which things are changing. Forecasts rarely assume that the environment is unchanging.

What is normally assumed is that the way in which the environment is changing will continue into the future. That is, a highly volatile environment will continue to be highly volatile; a business with fluctuating sales will continue to have fluctuating sales; and an economy that has gone through booms and busts will continue to go through booms and busts.

A forecasting model is intended to capture the way things move, not just where things are.

As Abraham Lincoln said, "If we could first know where we are and whither we are tending, we could better judge what to do and how to do it".

Source : Forecasting: Principles and Practice Rob J Hyndman and George Athanasopoulos

Forecasting vs Prediction

Forecasting :

Definition: Forecasting involves making predictions about future events or trends based on historical data and analysis.
It is commonly associated with time series data and aims to estimate the future values of a variable by identifying patterns, trends, and seasonality in the historical data.

Usage: stock prices, sales figures, weather conditions. It implies a systematic and methodical approach to predicting future outcomes.

Prediction:

Definition: Prediction, in a general sense, refers to the act of making statements about what will happen in the future. It is a broader term that can encompass various methods and approaches, including forecasting. Predictions can be based on data analysis, statistical models, machine learning algorithms, or expert judgment.

Usage: Prediction can be applied to a wide range of scenarios, not necessarily tied to time series data. For example, predicting whether a customer will churn, the outcome of a sports match, or the likelihood of success for a project are all instances where the term "prediction" is commonly used.

What is a time-series data?

Time-series data is a type of data where the values are observed, recorded, or measured over time at regular intervals. It is a sequence of data points collected or recorded in a sequential order, typically at equally spaced time intervals. Each data point in a time series is associated with a specific timestamp.

Common examples of time-series data include stock prices, temperature measurements, economic indicators, sensor data, and daily sales figures.

Key characteristics of time series data

Temporal Order: The data points in a time series are ordered chronologically, and the order of observations is significant.

Equally Spaced Intervals: Time-series data is often collected at regular time intervals, although irregular intervals are also possible.

Trends and Patterns: Time-series data may exhibit trends, patterns, or seasonality, allowing analysts to identify and understand underlying structures in the data.

Dependency on Time: The value of a data point in a time series is dependent on the time at which it was recorded. Past values can influence future values.

Key Concept in Time-series data

Trend: A trend represents the long-term movement or direction in a time series, indicating the underlying growth or decline in the data over an extended period.

Example: In retail sales, a positive trend might signify a consistent increase in sales over several months or years, while a negative trend could indicate a prolonged decrease.

Cycle: The cycle refers to a recurring pattern or wave-like movement observed over a more extended time frame than a typical seasonal pattern. It is not strictly regular and can span multiple years.

Example: Economic cycles, such as periods of economic expansion and contraction, exemplify cyclic patterns in time series data.

Seasonal Variations: Seasonal variations are regular, repeating patterns in a time series that occur at fixed intervals, often related to calendar events or specific seasons of the year.

Example: Retail sales of winter clothing might exhibit a seasonal peak in the colder months, reflecting consumer demand during the winter season.

Random Fluctuation: Random fluctuation, also known as irregular variation or noise, represents the unpredictable and erratic components in a time series that do not follow a discernible pattern.

Example: Sudden, unanticipated changes in stock prices driven by unforeseen events, such as unexpected news or market reactions, illustrate random fluctuations in financial time series data.

What is the difference between a time series and a normal series?

The main difference between a time series and a normal series lies in the incorporation of time as a key factor.

Time Series:
Definition: A time series is a sequence of data points collected or recorded over a time interval. Each data point corresponds to a specific time, and the order of the data points is important.

Example: Daily temperature readings over a year, monthly sales figures, or hourly stock prices are examples of time series data. In these cases, the time variable is explicit, and the data points are ordered chronologically.

Normal Series (Non-Time Series):
Definition: The term "normal series" is not commonly used in the context of data analysis. Instead, the opposite of a time series is often referred to as cross-sectional or cross-sectional data.

Example: In cross-sectional data, observations are taken at a single point in time, and there is no inherent time ordering. For instance, a survey conducted at a specific moment to collect information about individuals' income, age, and education level represents cross-sectional data.

Models for a time-series analysis

ARIMA (AutoRegressive Integrated Moving Average):
Type: Statistical model.
Description: ARIMA is a popular model for time series forecasting. It combines autoregressive (AR) and moving average (MA) components, and the integrated (I) part is used to make the time series stationary.

SARIMA (Seasonal AutoRegressive Integrated Moving Average):
Type: Statistical model.
Description: An extension of ARIMA that includes seasonal components, SARIMA is suitable for time series data with a clear seasonal pattern.

XGBoost and LightGBM:
Type: Machine learning model (ensemble method).
Description: These are gradient boosting algorithms that can be used for time series forecasting. They can handle complex relationships and capture non-linear patterns.

Prophet:
Type: Additive model.
Description: Developed by Facebook, Prophet is designed for forecasting with daily observations that display patterns on different time scales. It handles missing data and outliers well, making it suitable for real-world datasets.

LSTM (Long Short-Term Memory) Networks:
Type: Deep learning model.
Description: LSTM is a type of recurrent neural network (RNN) that can capture long-term dependencies in time series data. It is effective for sequential data and can learn complex patterns.

Performance Metric for Time series data

The choice of metric depends on the specific characteristics of a time series data and the goals of the analysis. Among them, I used the metrics as below.

Root Mean Squared Error (RMSE):

Formula: $RMSE = \sqrt{MSE}$

Mean Squared Error (MSE):

Formula: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Description: Measures the average squared difference between actual and predicted values.
It penalizes larger errors more than MAE.

Mean Absolute Error (MAE):

Formula: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

Description: Measures the average absolute difference between actual and predicted values.
It is less sensitive to outliers compared to MSE.

Description: It provides an interpretable scale as it is in the same units as the target variable.

R-squared (R^2):

Formula: $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$

Description: Measures the proportion of the variance in the dependent variable explained by the model.

Data Description

Data

	PRODUCT_CATEGORY	DATE	QTY	TRANSACTIONS	STORE_CODE
0	TOPS	18/05/16	5	5	85
1	TOPS	20/05/16	4	4	85
2	TOPS	21/05/16	13	13	85
3	TOPS	24/05/16	3	3	85
4	TOPS	25/05/16	1	1	85
...	...	...	...	...	...
28146	SWEATERS	13/06/19	3	3	801
28147	SWEATERS	14/06/19	5	5	801
28148	SWEATERS	15/06/19	2	2	801
28149	SWEATERS	16/06/19	1	1	801
28150	SWEATERS	17/06/19	1	1	801

28151 rows × 5 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28151 entries, 0 to 28150
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   PRODUCT_CATEGORY  28151 non-null  object
 1   DATE              28151 non-null  object
 2   QTY               28151 non-null  int64 
 3   TRANSACTIONS      28151 non-null  int64 
 4   STORE_CODE        28151 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 1.1+ MB
```

Description

PRODUCT_CATEGORY : dataset contains data about 5 distinct product categories

* Tops
* Jackets
* Shoes
* Accessories
* Sweaters

DATE : 3 year of records (5/2016 - 6/2019)

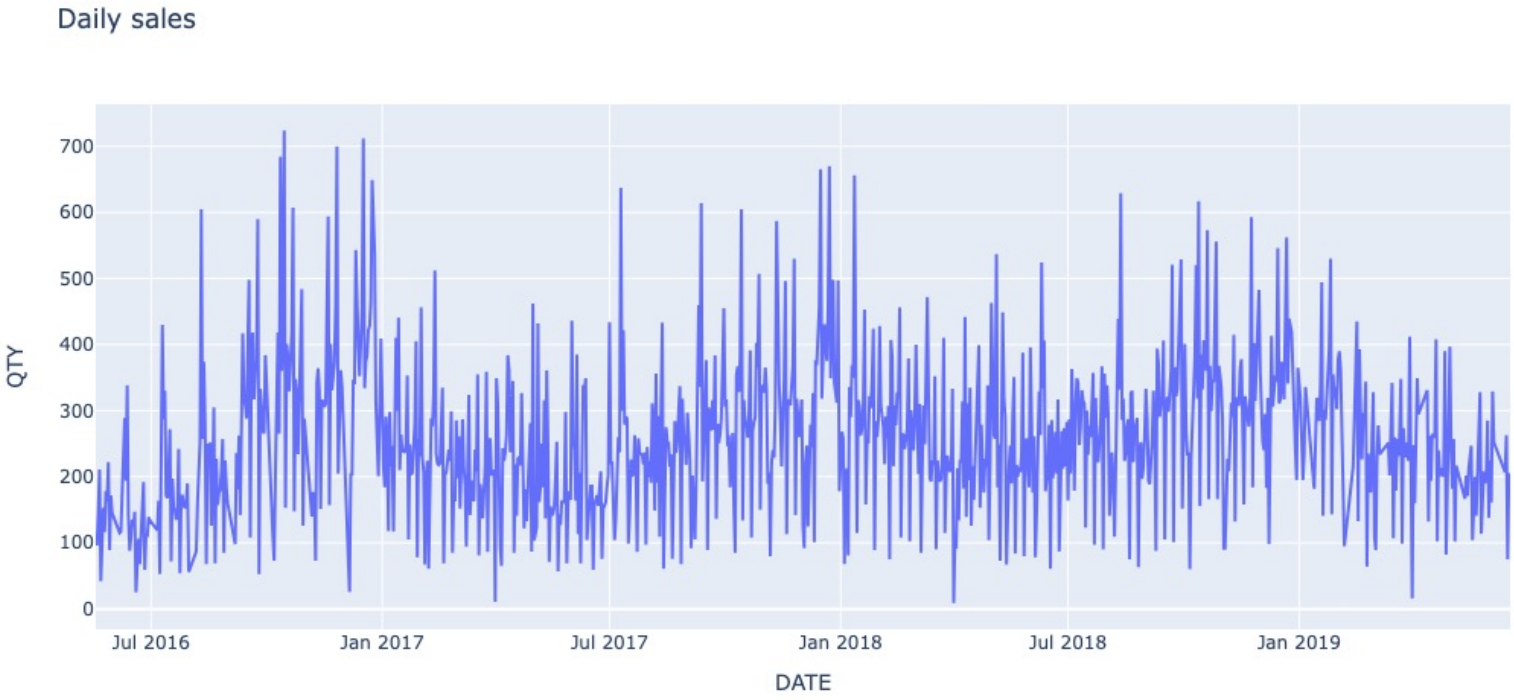
QTY = SALES : Quantity sold for each combination of store, category and date Ranges from -3 to 209 can be negative when returns are higher than products sold

TRANSACTIONS : Daily transactions for each combination of category and store Ranges from 1 to 227 generally higher than QTY because of potential multiple transactions for a single product and returns

STORE_CODE : 8 different stores are identified by a numeric code

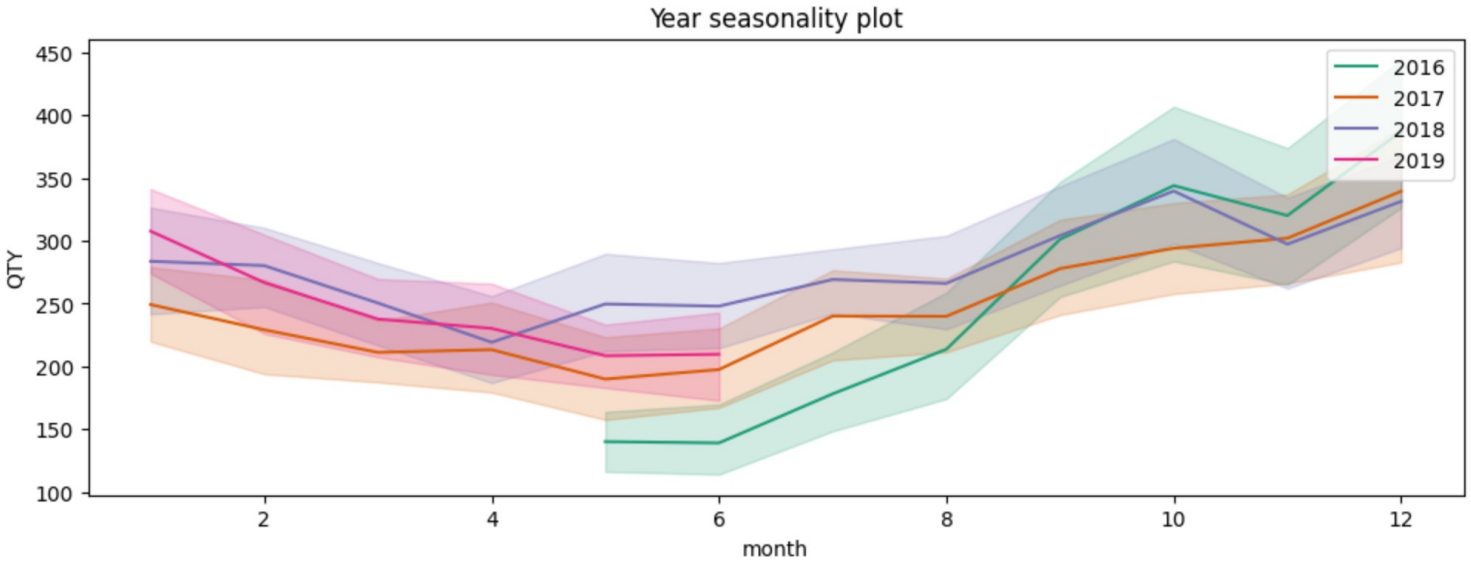
Exploratory Data Analysis

Exploratory Data Analysis



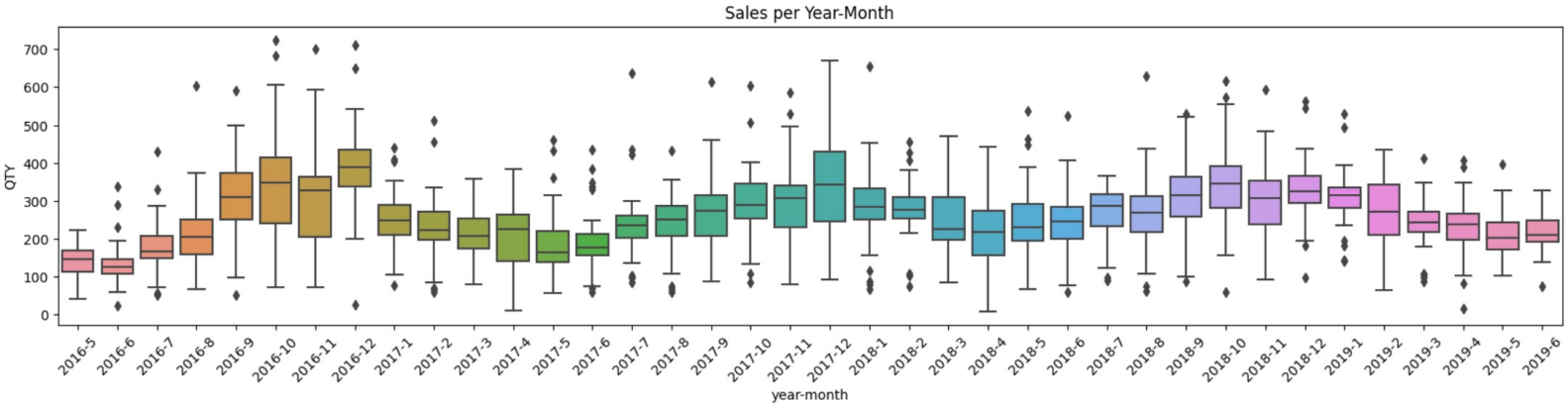
With the above graph, The pattern of sales per month seems to be pretty the same every year. We can also identify some seasonality over the year.

Exploratory Data Analysis

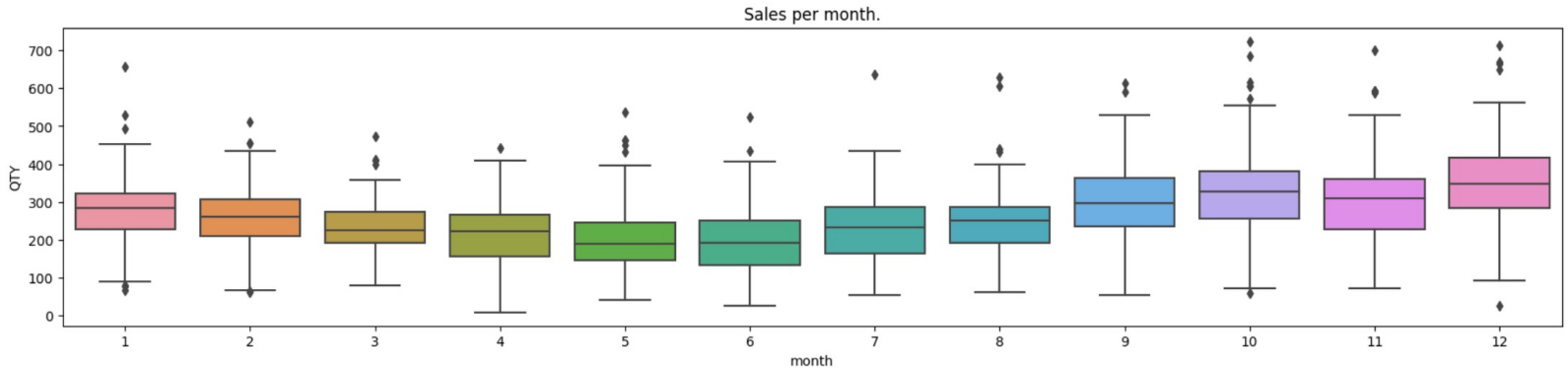


The pattern of sales is pretty much the same every year with higher number of sales in December.
In addition, it show us that the volume of sales is slightly increasing every year

Exploratory Data Analysis



We can identify again that the pattern of sales is pretty much the same every year with higher number of sales in December.



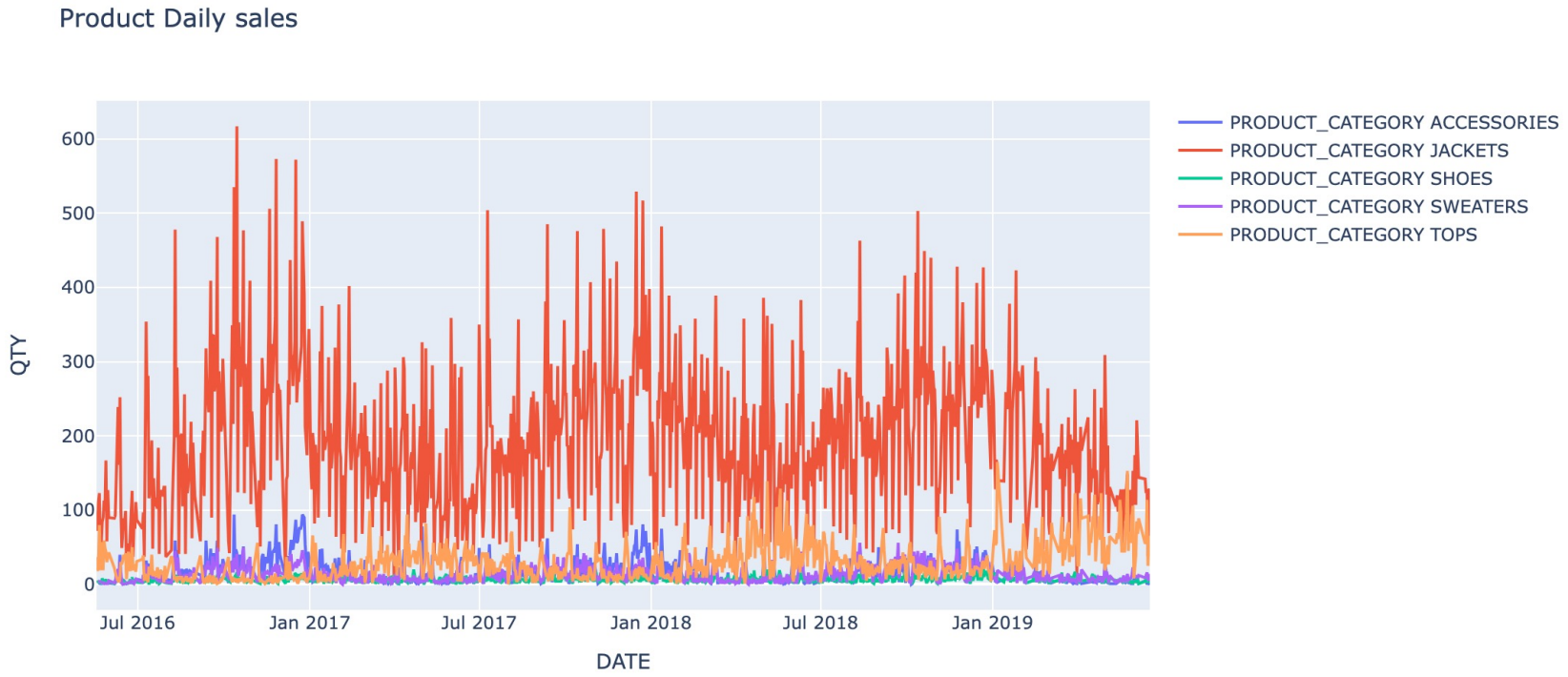
In Monthly level, Higher volume in sales is achieved between October and December and lowest between April and June. It's actually a normal pattern in Fashion industry.

Exploratory Data Analysis


```
The month with the highest sales: 12
```

More specifically, In the monthly graph, we can see that sales gradually decrease starting in February and then rise again starting in August, repeating the cycle. And we were able to confirm that December was the month with the most sales which tells us that sales tend to peak at the end of the year.

Exploratory Data Analysis

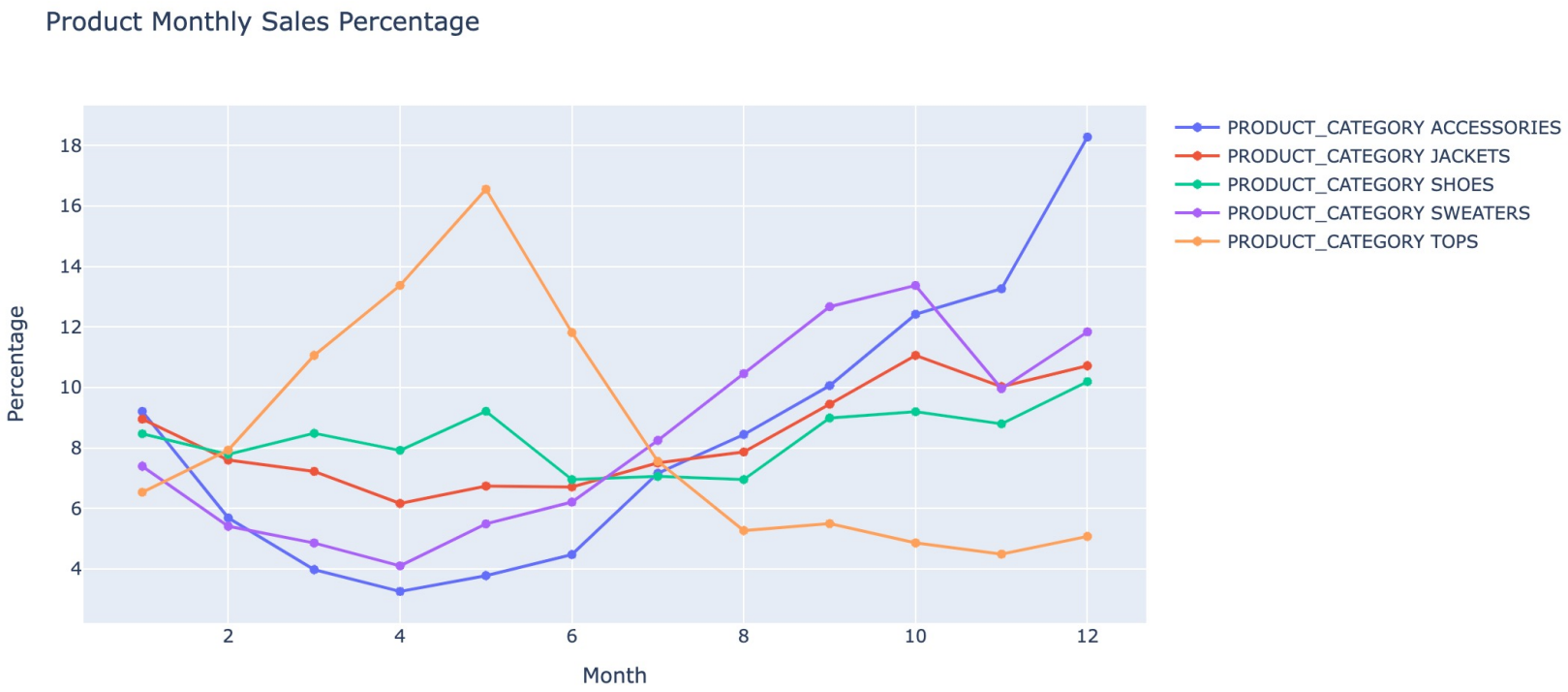


The graph above shows the sales volume for each product categories.
we can see that jacket sales are overwhelmingly higher than others.

Exploratory Data Analysis

More specifically, Looking at the graph above, we can see that jackets account for 75% of total sales, and the product with the lowest percentage is shoes.

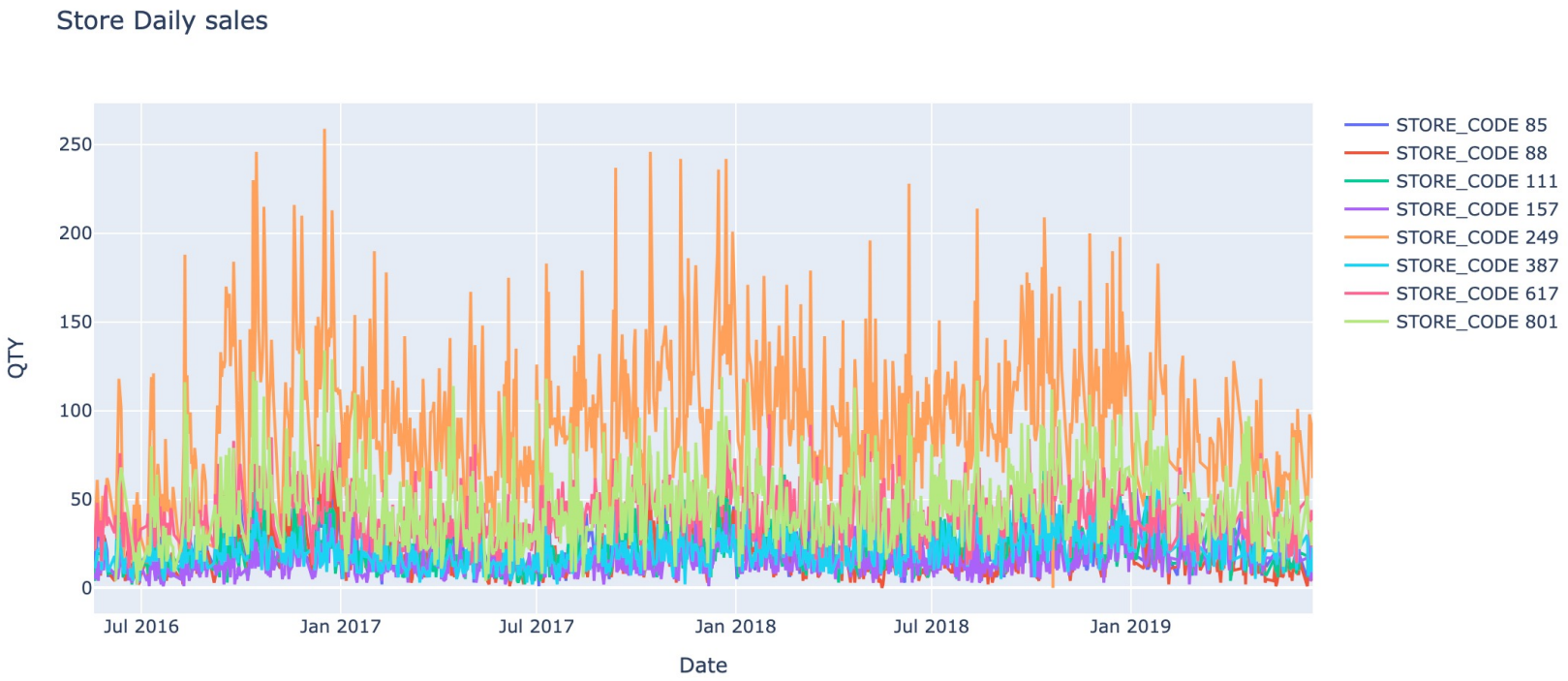
Exploratory Data Analysis



What's interesting about the monthly combined sales volume is that overall sales are mostly made in the fall and winter seasons but we can see that a lot of top sales are made in the spring and summer seasons.

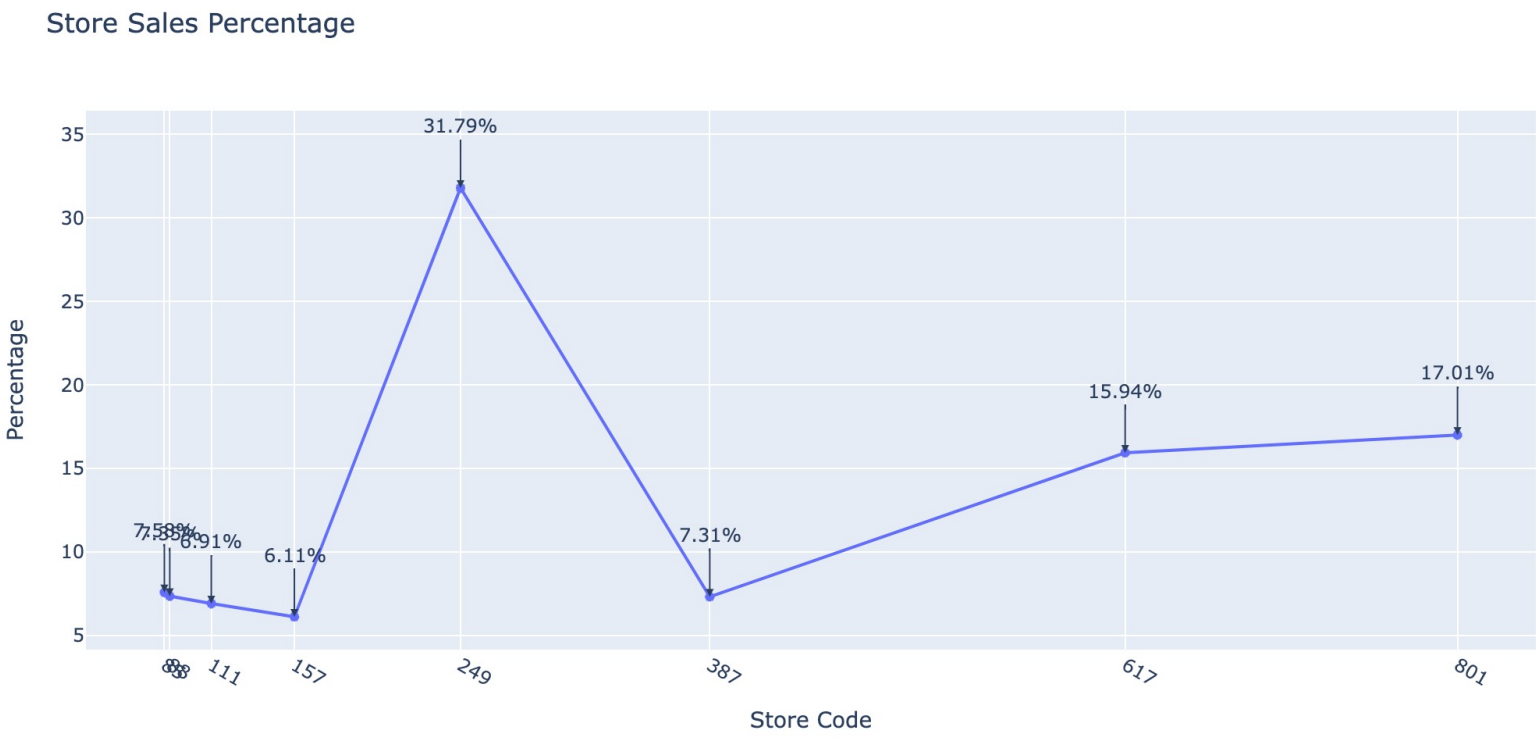
It is believed that this interpretation can be used to strategically produce and sell products that customers want. Additionally, accessories include many products that can be used in the winter, such as scarves and gloves, so if sales appear to be concentrated in the fall and winter seasons, we may want to consider developing spring and summer products as well.

Exploratory Data Analysis



The graph above shows sales volume for each store code.
It is confirmed that the store selling the most is number 249.

Exploratory Data Analysis



More specifically, we can identify that 249, 801, and 617 has highest sales in that order.

Data Preprocessing

Data Preprocessing

```
df.drop(['PRODUCT_CATEGORY','TRANSACTIONS','STORE_CODE'], axis=1, inplace=True)
df['DATE'] =  pd.to_datetime(df['DATE'])
start_date = '2016-05-18'
end_date = '2019-06-18'
df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]
df['QTY'] = abs(df['QTY'])
df = df.set_index('DATE')
df = df.sort_values(by='DATE')
df = df.groupby(by = ['DATE']).sum()
```

Converting the 'DATE' column to datetime format. This ensures that the 'DATE' column is treated as a datetime object.

Filtering the DataFrame to include only rows where the 'DATE' falls between '2016-05-18' and '2019-06-18'

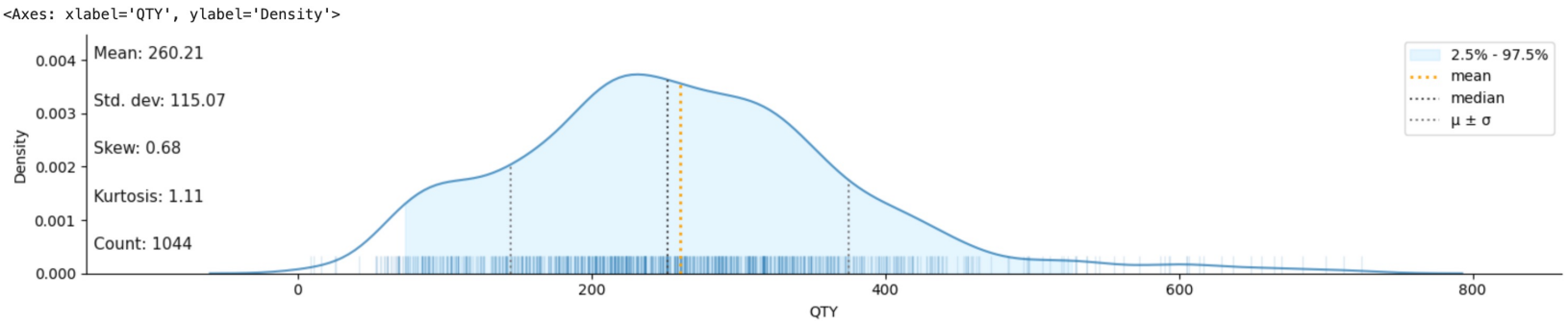
Taking absolute values of the 'QTY' column in order to avoid negative value error.

Setting the 'DATE' column as the index of the DataFrame
This makes it easier to work with time series data.

Sorting the DataFrame by the 'DATE' index in ascending order

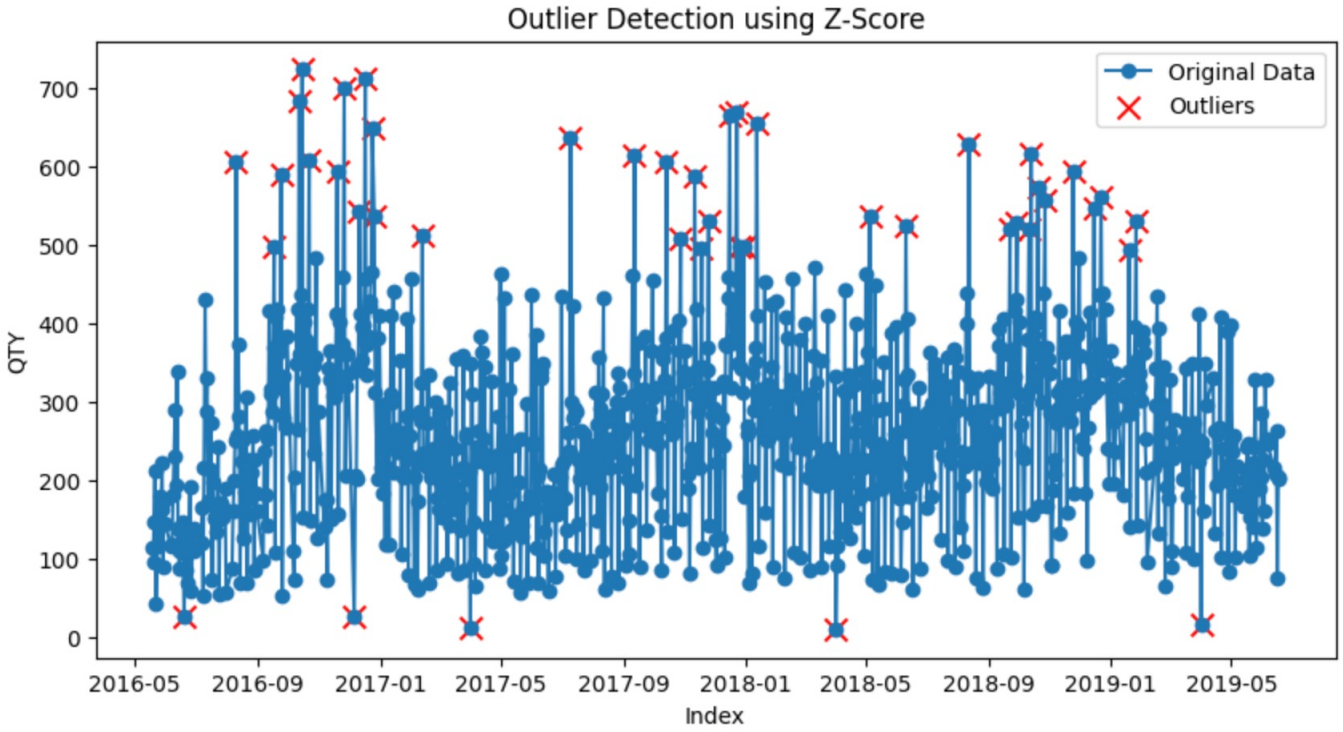
Grouping the DataFrame by the 'DATE' index and summing the values for each date.
This step is aggregating the data, likely combining rows with the same date and summing their corresponding values.

Exploratory Data Analysis



Through the plot above, we can check the distribution density and see that the mean is 260 and the standard deviation is 115 of Sales.

Detecting Outlier



The Z-score threshold of 2 is a common choice for detecting outlier.
In the given dataset, points with Z-scores greater than 2 or less than -2 are considered outliers.

Detected outliers are those 'QTY' values that deviate significantly from the mean, indicating potential unusual or extreme behavior in the quantity of interest.

However, since this data is actual data and the number is not large, it was used as it is rather than excluded or replaced.

Arima vs Auto Arima vs Sarima

What is Arima Model?

AR Model (AutoRegressive Model)

Autocorrelation is constructed as a time series model, and is a model that predicts the future value of a variable through a linear combination of past observed values of the variable.
It is based on the idea that previous observations influence subsequent observations.
p is used as the parameter value and can be expressed in the form of AR(p).

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t$$

- y_t : The value of the time series at time t .
- c : Constant term.
- $\phi_1, \phi_2, \ldots, \phi_p$: Autoregressive coefficients.
- $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$: Lagged values of the time series.
- ϵ_t : Error term at time t .

The error term is white noise that follows a standard normal distribution with mean 0 and variance 1.

MA Model (Moving Average Model)

This is a model that predicts future values using prediction error.
q is used as the parameter value and can be expressed in the form of MA(q).

$$y_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t$$

What is Arima Model?

ARMA model (steady time series model)

It means a linear combination of p own past values and q past white noise, and is a model that combines the AR(p) model and the MA(q) model.
The stationarity conditions of the AR(p) model and the reversibility conditions of the MA(q) model must be satisfied.
Parameter values are p,q, and can be expressed in the form of ARMA(p,q).

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$$

ARMA becomes identical to the AR and MA models when the value is given as 0.
ARMA(1, 0) = AR(1)
ARMA(0, 1) = MA(1)
ARMA(1,1) is the most common type of stationary time series model.

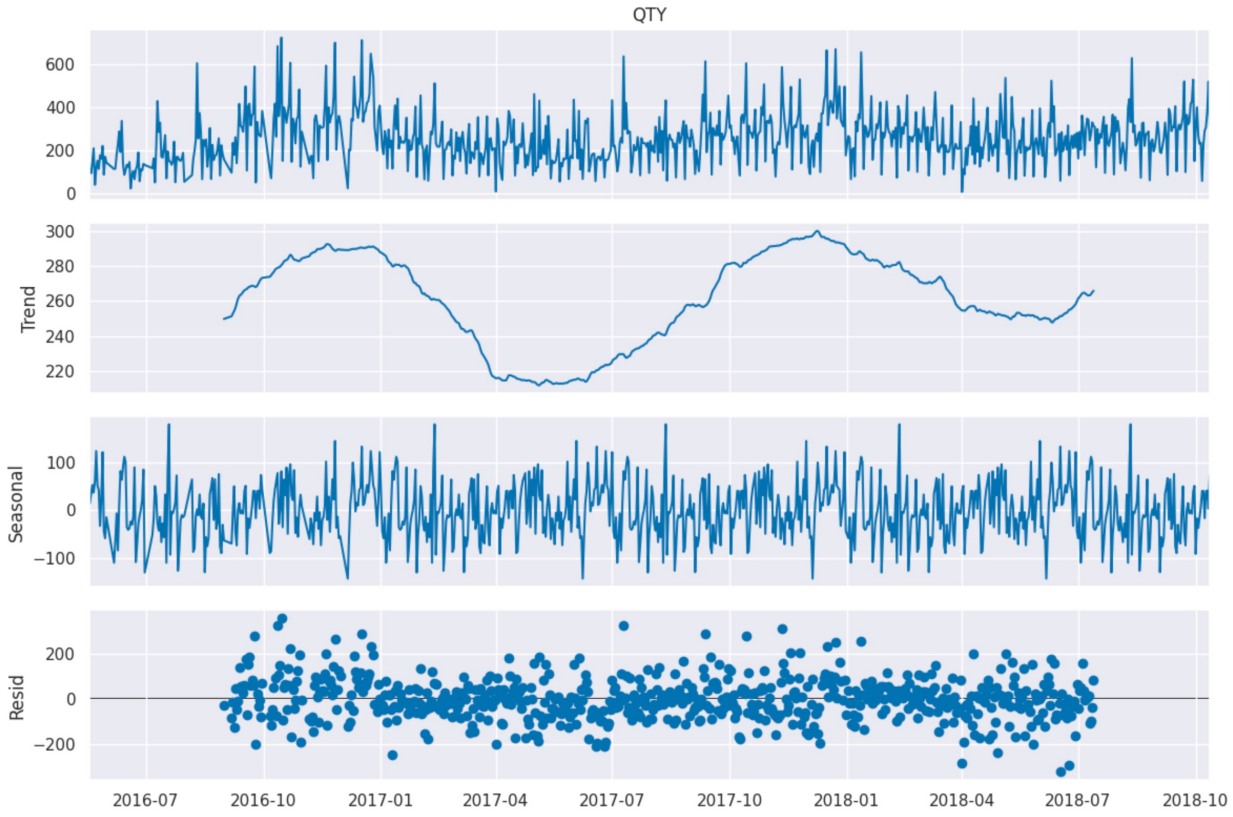
ARIMA model (unstable time series model)

This is a model that additionally applies the d-order difference to the ARMA model, and has the advantage of being directly applicable to non-stationary time series.
ARIMA(p,d,q): A model that combines the AR(p) model and the MA(q) model with d-order differential data. I stands for Integrated, which means 'cumulative'.

ARIMA model -> AR, MA
ARIMA(p,0,0) = AR(p)
ARIMA(0,0,q) = MA(q)

p,d,q values can be selected through various methods such as AIC, BIC, ACF, and PACF.
Autocorrelation (p, q) can be checked by drawing ACF and PACF graphs.

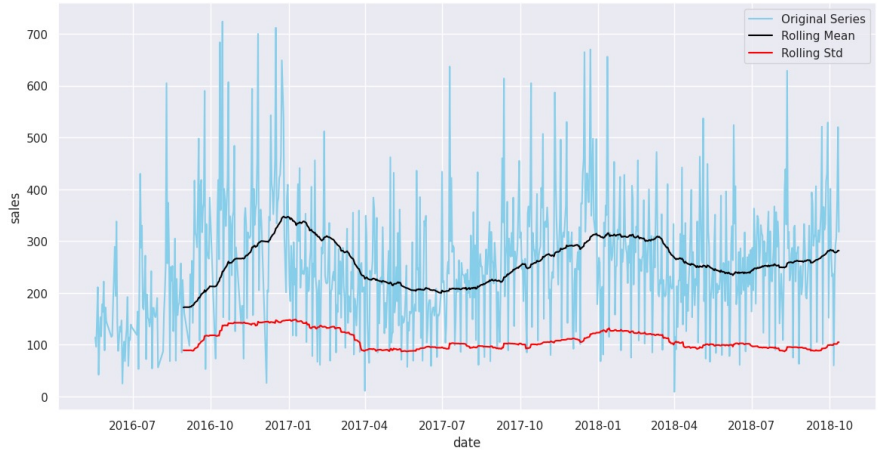
Decomposition Plot



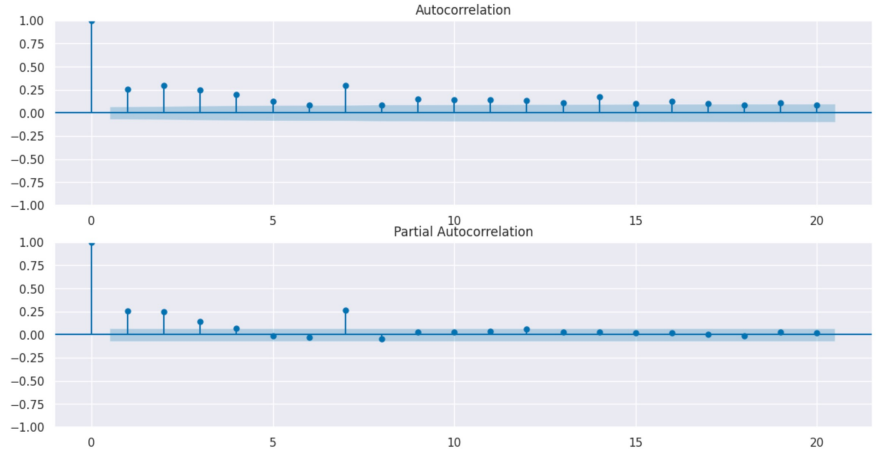
Trend Component: There is a cycle on sales. we can't conclude the data is stationary with this.

Seasonal Component: Since the seasonal component is similar (not getting multiplied) over the period of time, we can consider that the model can be additive.

Stationarity




```
Test statistic = -6.540
P-value = 0.000
Critical values :
        1%: -3.438272329479064 - The data is  stationary with 99% confidence
        5%: -2.8650369467745573 - The data is  stationary with 95% confidence
        10%: -2.568632068251301 - The data is  stationary with 90% confidence
```



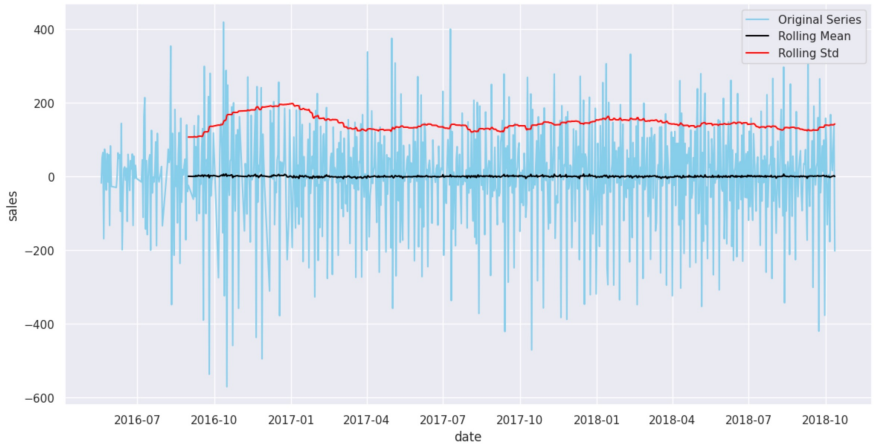
3 ways to test the stationarity of time series

Plotting rolling statistics: Since the rolling statistics exhibit a clear trend (upwards or downwards) and show varying variance (increasing or decreasing amplitude), we can assume that our data is likely not to be stationary.

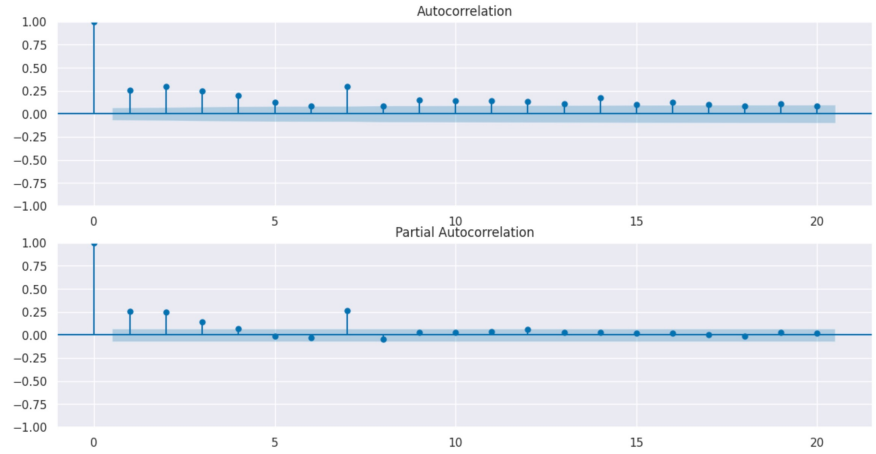
Augmented Dickey-Fuller Test: However, from this ADF test, It confirmed that our data is stationary.

ACF and PACF plots: Since all the spikes to fall in the blue region means that our data is stationary, we can assume that our data is likely not to be stationary.

Differencing




```
Test statistic = -10.568
P-value = 0.000
Critical values :
        1%: -3.4384289321263126 - The data is  stationary with 99% confidence
        5%: -2.865105979079729 - The data is  stationary with 95% confidence
        10%: -2.568668841575869 - The data is  stationary with 90% confidence
```



Differencing: Seasonal or cyclical patterns can be removed by substracting periodical values. If the data is 12-month seasonal, substracting the series with a 12-lag difference series will give a “flatter” series. Since we have aggregated the data to each day-level, we shift by 1.

After applying Differencing to our data, we can see from the results that the series now seem to be stationary because mean and variance are more constant over time.

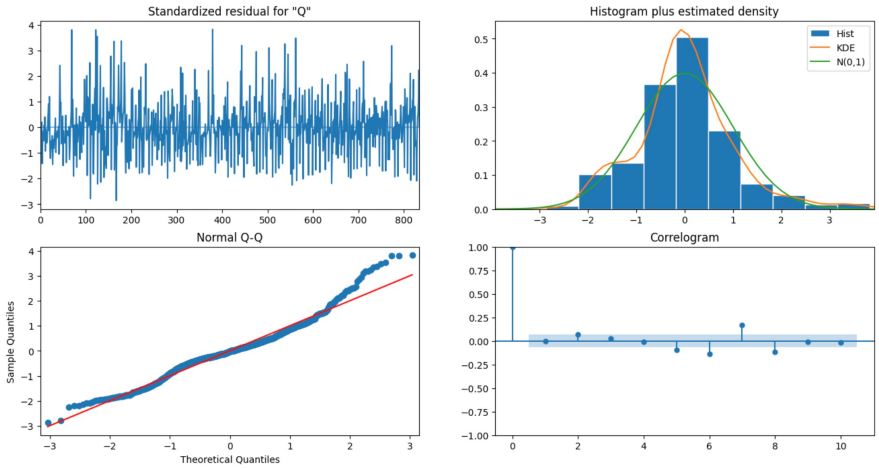
Arima Result

SARIMAX Results

Dep. Variable:	QTY	No. Observations:	835
Model:	ARIMA(1, 0, 1)	Log Likelihood	-5093.711
Date:	Sun, 19 Nov 2023	AIC	10195.422
Time:	17:44:32	BIC	10214.332
Sample:	0	HQIC	10202.672
	- 835		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	256.1712	14.217	18.019	0.000	228.306	284.036
ar.L1	0.9423	0.020	47.021	0.000	0.903	0.982
ma.L1	-0.7997	0.035	-22.599	0.000	-0.869	-0.730
sigma2	1.164e+04	459.121	25.350	0.000	1.07e+04	1.25e+04

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	102.63
Prob(Q):	0.91	Prob(JB):	0.00
Heteroskedasticity (H):	0.82	Skew:	0.50
Prob(H) (two-sided):	0.10	Kurtosis:	4.40



Although I could assumed that there is some seasonality but I first tried ARIMA model with pdq that I decided arbitrarily. The reason I decided this is because after the stationary test, our data was shown to be stationary even before differencing, and in the ACF and PACF tests, we confirmed that there is a sharp cutoff, dropping rapidly after 1.

Prob(Q) = 0.91 > 0.05. We shouldn't reject the null hypothesis that the residuals are uncorrelated so the residuals are not correlated.
Prob(JB) = 0.02 < 0.05. We reject the null hypothesis that the residuals are normally distributed. Therefore, the residuals are not normally distributed.

Standardized residual: There are no obvious patterns in the residuals. This points out to a good model

Histogram plus kde estimate: The green line shows a normal distribution. For a good model the orange line should be similar to the green line. The orange curve is a bit different to the green one.

Correlogram or ACF plot: 95% of correlations for lag greater than one should not be significant (inside the blue area). There are also some differnece.

Normal Q-Q: Most of the data points should lie on the straight line, indicating a normal distribution of the residuals. There are also some difference.

Therefore, the model doesn't seems to be a good model.

Auto Arima Result

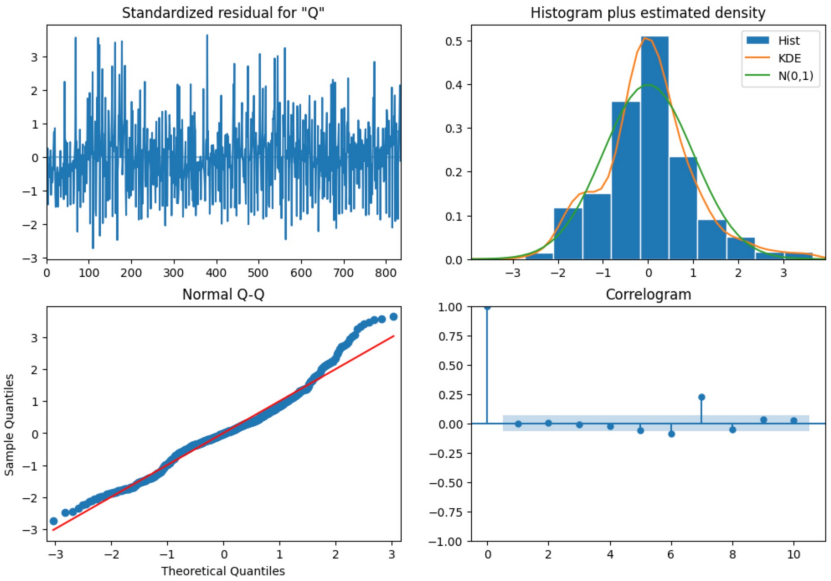
SARIMAX Results

Dep. Variable:	y	No. Observations:	835
Model:	SARIMAX(5, 1, 5)	Log Likelihood	-5051.856
Date:	Mon, 27 Nov 2023	AIC	10125.711
Time:	08:48:57	BIC	10177.700
Sample:	0	HQIC	10145.644
	- 835		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.4531	0.086	-16.911	0.000	-1.622	-1.285
ar.L2	-1.0972	0.164	-6.693	0.000	-1.418	-0.776
ar.L3	-0.6312	0.172	-3.675	0.000	-0.968	-0.295
ar.L4	-0.1498	0.114	-1.310	0.190	-0.374	0.074
ar.L5	0.2732	0.038	7.196	0.000	0.199	0.348
ma.L1	0.6060	0.088	6.919	0.000	0.434	0.778
ma.L2	-0.0816	0.102	-0.803	0.422	-0.281	0.117
ma.L3	-0.2303	0.096	-2.402	0.016	-0.418	-0.042
ma.L4	-0.4247	0.086	-4.929	0.000	-0.593	-0.256
ma.L5	-0.6065	0.076	-7.941	0.000	-0.756	-0.457
sigma2	1.072e+04	437.258	24.512	0.000	9861.171	1.16e+04

Ljung-Box (L1) (Q):	0.06	Jarque-Bera (JB):	64.10
Prob(Q):	0.80	Prob(JB):	0.00
Heteroskedasticity (H):	0.83	Skew:	0.32
Prob(H) (two-sided):	0.12	Kurtosis:	4.20

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



Auto Arima has a function that automatically finds the best pdq.

The results through auto arima this time, we can see that Prob(Q), AIC, and BIC have all improved slightly.

Looking at the correlogram, we can see that more spikes have entered the blue area.

We can conclude that we found better model by using Auto Arima.

SARIMA Result

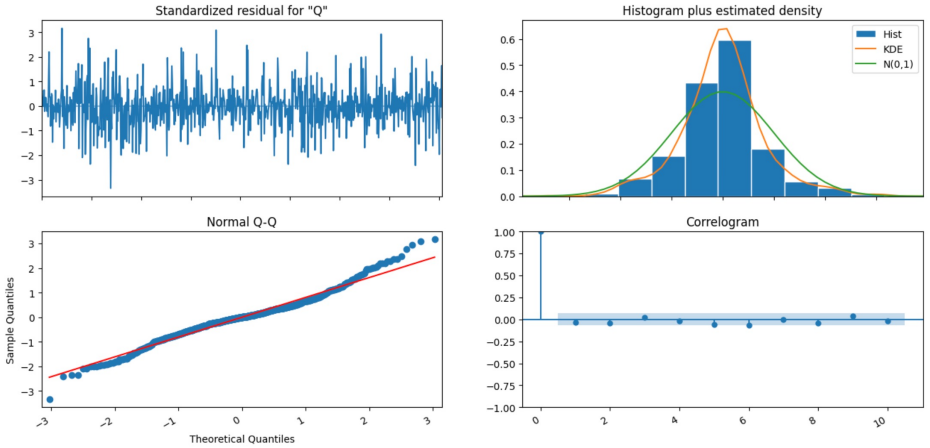
SARIMAX Results

Dep. Variable:	QTY	No. Observations:	835
Model:	SARIMAX(5, 1, 5)x(2, 1, [1, 2], 7)	Log Likelihood	-4941.877
Date:	Mon, 27 Nov 2023	AIC	9913.754
Time:	12:10:08	BIC	9984.154
Sample:	0 - 835	HQIC	9940.788
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4833	3.393	-0.142	0.887	-7.133	6.166
ar.L2	-0.2588	2.460	-0.105	0.916	-5.081	4.563
ar.L3	0.6163	1.332	0.463	0.643	-1.993	3.226
ar.L4	0.0293	1.907	0.015	0.988	-3.708	3.766
ar.L5	-0.0312	0.658	-0.047	0.962	-1.320	1.258
ma.L1	-0.3065	3.399	-0.090	0.928	-6.968	6.355
ma.L2	-0.0521	0.324	-0.161	0.872	-0.687	0.583
ma.L3	-0.9037	0.384	-2.355	0.019	-1.656	-0.152
ma.L4	0.3931	3.190	0.123	0.902	-5.859	6.645
ma.L5	-0.0843	0.621	-0.136	0.892	-1.302	1.133
ar.S.L7	0.3835	0.311	1.232	0.218	-0.226	0.993
ar.S.L14	-0.1725	0.070	-2.474	0.013	-0.309	-0.036
ma.S.L7	-1.1660	0.321	-3.636	0.000	-1.794	-0.537
ma.S.L14	0.2918	0.281	1.037	0.300	-0.260	0.844
sigma2	1.704e+04	1032.036	16.510	0.000	1.5e+04	1.91e+04

Ljung-Box (L1) (Q):	0.88	Jarque-Bera (JB):	94.77
Prob(Q):	0.35	Prob(JB):	0.00
Heteroskedasticity (H):	0.71	Skew:	0.16
Prob(H) (two-sided):	0.01	Kurtosis:	4.65

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



Sarima is an extension of the ARIMA (AutoRegressive Integrated Moving Average) model that incorporates seasonality into the time series forecasting process has a function that automatically finds the best pdq.

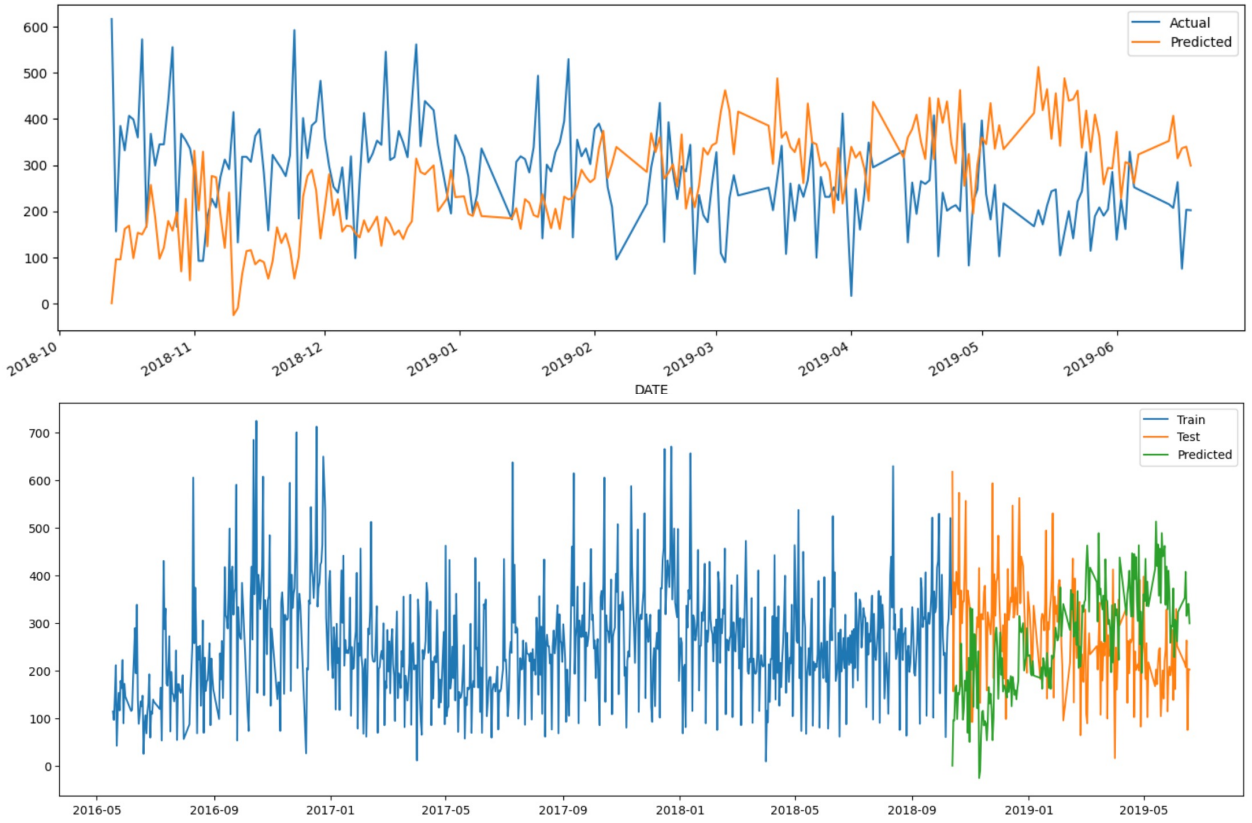
As we have already seen in the last ACF and PACF, there was a significant change in spike number 7. With this, I set the time step (m) to 7. Since I found that the best non-seasonal pdq (5, 1, 5) using auto arima, I searched for seasonal PDQ this time for Sarima Model.

The result through Sarima this time, we can see that Prob(Q), AIC, and BIC are all improved more.

Looking at the correlogram, we can see that all spikes have entered the blue area.

We can conclude that we found better Sarima model.

Sarima Prediction



The graph above shows our prediction results along with the original data.

Conclusion

I first checked stationarity and seasonality in order to apply the ARIMA model, a statistical model, to the time series data.

For this purpose, I used graphical and statistical methods to find the best fit model as follows

- Plotting rolling statistics plot analysis
- Augmented Dickey-Fuller Test
- ACF and PACF plots analysis
- Checking model summary
- Diagnostics plot analysis

ARIMA	AUTO ARIMA	SARIMA
AIC: 10195.422	AIC: 10125,711	AIC: 9913,754
RMSE: 12.0779	RMSE: 10.3959	RMSE: 13.4326
MSE: 145.8759	MSE: 108.0744	MSE: 180.4360
MAE: 10.9687	MAE: 9.2446	MAE: 12.1562
R2 Score: -0.9135	R2 Score: -0.0503	R2 Score: -1.9276

Although there is a SARIMA model with the lowest AIC value, Auto Arima is considered the best model based on R2 score.
Since each performance metric has meaning, interpreting the results and deciding on a good model requires domain knowledge and understanding.

Prophet vs Neural Prophet

What is Prophet?

Prophet

is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_i$$

The main components of the Prophet model are Trend, Seasonality, and Holidays. When combined, these three elements can be expressed in the following formula:

- $y(t)$: The dependent variable representing the predicted value at time
- $g(t)$: Represents the trend function.
- $s(t)$: Represents the seasonality function. This function captures periodic patterns that occur regularly, such as weekly or yearly variations.
- $h(t)$: Represents the holiday function. This function captures irregular events, such as holidays, that can cause abnormal increases or decreases in values during specific periods.
- ε_i : Represents the error term assumed to follow a normal distribution. This term accounts for the deviation of the model from the actual data.

By combining these functions, one can model time series data, considering non-periodic trends, periodic seasonal patterns, irregular holiday events, and the normal distribution error term.

Prophet Training

```
df = df.rename(columns={'DATE': 'ds', 'QTY': 'y'})
df = df.groupby('ds', as_index=False).sum()
df
```

Prophet requires as input a dataframe with two columns:
ds: datetime column.
y: numeric column which represents the measurement we wish to forecast.
I renamed columns date and qty, respectively, as ds and y.

```
p = Prophet()
p.fit(df_train)
future_prophet = p.make_future_dataframe(periods=365)
forecast_prophet = p.predict(future_prophet)
```

```
forecast_prophet
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	a
0	2016-05-18	217.354212	12.807560	255.887546	217.354212	217.354212	-79.051917	
1	2016-05-19	217.519386	24.133150	262.362951	217.519386	217.519386	-71.552160	
2	2016-05-20	217.684561	60.382972	287.480479	217.684561	217.684561	-40.296722	
3	2016-05-21	217.849735	124.292768	356.054830	217.849735	217.849735	24.053788	
4	2016-05-22	218.014909	-47.036969	205.335755	218.014909	218.014909	-142.162634	
...	...	...	...	...	...	...	...	
1195	2019-10-08	314.262473	228.167653	474.622212	309.410922	319.106095	31.413153	
1196	2019-10-09	314.329698	227.846854	465.875955	309.458676	319.203084	35.983923	

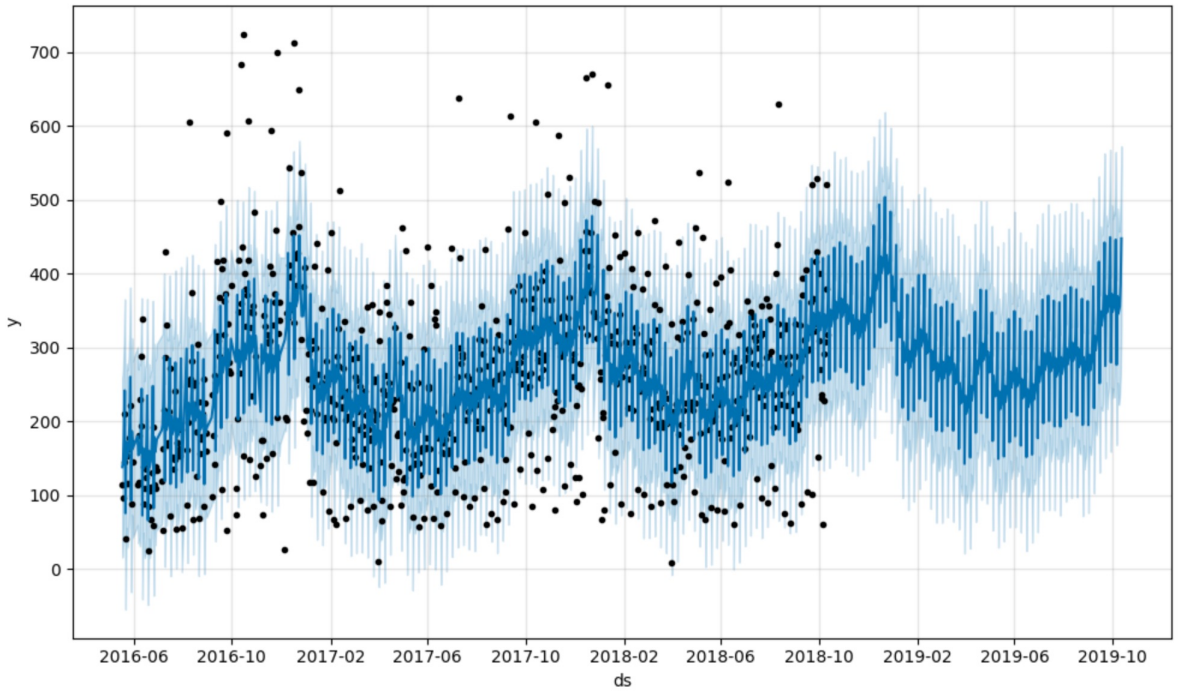
The initial model was executed with the default parameters since prophet is based on an additive model (our dataset is additive).

Make_future_dataframe builds a dataframe that extends into the future a specified number of days. I set 365 days in the future.

The forecast dataframe contains Prophet's prediction for sales qty.

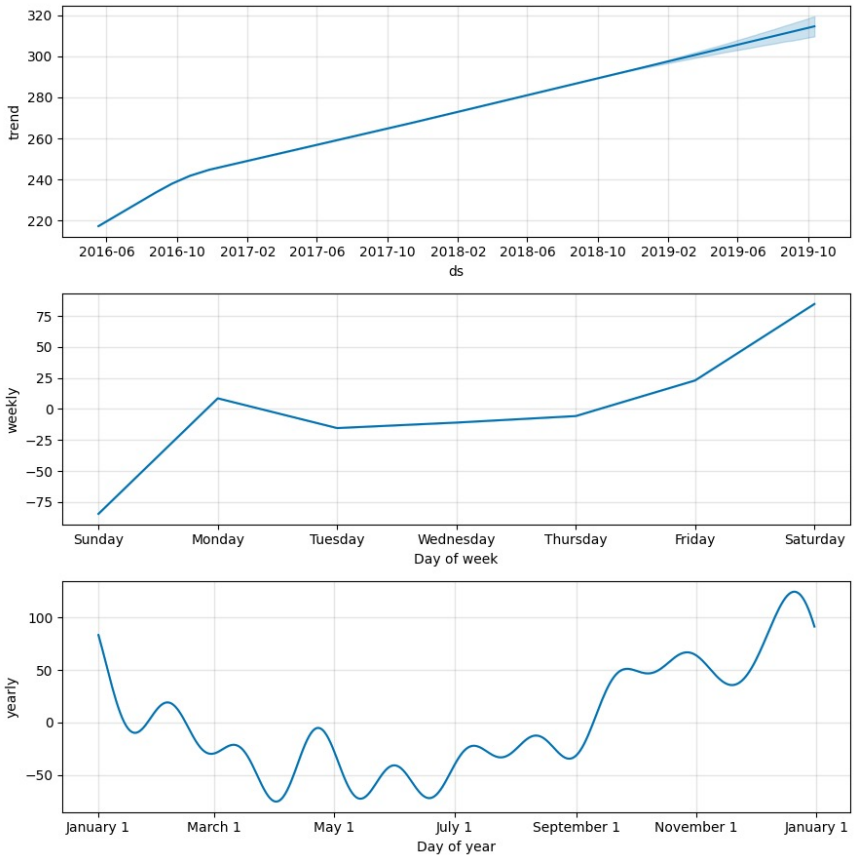
Forecast includes a column yhat with the forecast, as well as columns for components and uncertainty intervals as well.

Prophet Plot



In the plot above, deep blue line is forecast sales forecast['y_hat'], black dots are actual sales qty forecast['y']. The light blue shade is 95% confidence interval around the forecast. The uncertainty interval in this region is bounded by forecast['yhat_lower'] and forecast['yhat_upper'] values.

Prophet Components Plot

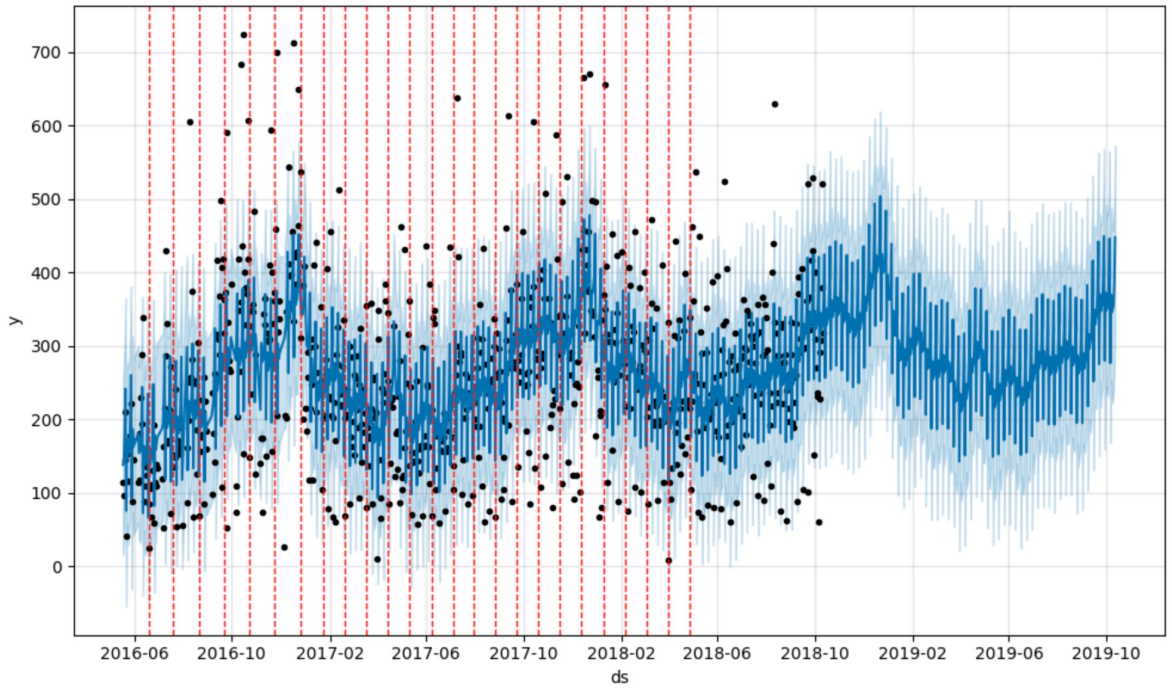


The first Trend plot shows that the total sales on a daily basis are increasing. Trend upward.

The second Weekly plot shows the sales are increasing from Thursday to Saturday and decreasing on Sunday.

The third Yearly plot shows that as observed previously sales is higher from September to January and lower from March to July.

Trend Change Points

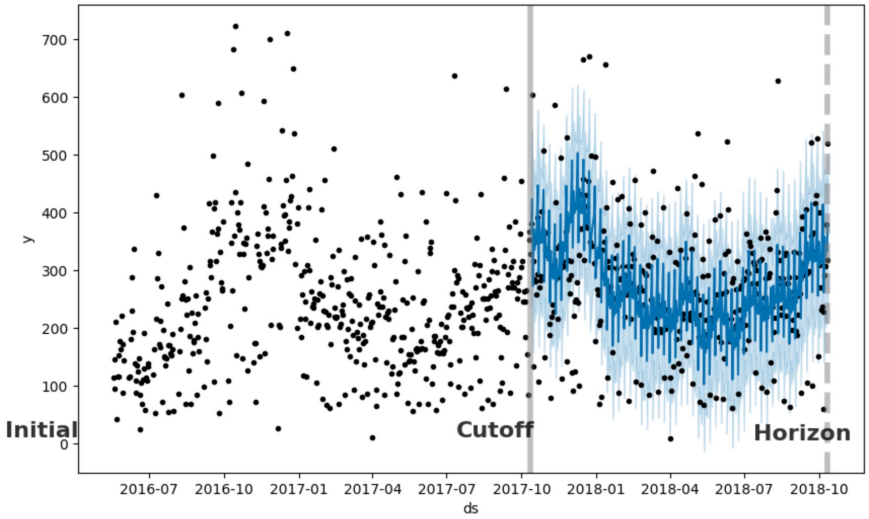


Prophet automatically detects the change points where the trend changes and predicts the trend.
By default, Prophet potentially assigns the change points at 80% of the size of the time series data.
If overfitting is severe, the graph will be too flexible and close to all values. If underfitting, flexibility is lacking. Default value is 0.05 Increasing this value makes the graph more flexible (=resolving underfitting), and decreasing this value decreases flexibility (=resolving overfitting).

Cross Validation

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2017-10-13	363.810445	241.675354	489.632680	381	2017-10-12
1	2017-10-14	423.467688	301.392884	542.362433	605	2017-10-12
2	2017-10-15	273.102344	156.915521	385.439386	134	2017-10-12
3	2017-10-16	371.909616	252.582389	491.916428	316	2017-10-12
4	2017-10-17	335.651874	224.683077	452.885622	284	2017-10-12
...	...	...	...	...	...	...
356	2018-10-08	359.154154	251.661065	475.038996	292	2017-10-12
357	2018-10-09	322.128713	198.361318	438.090843	308	2017-10-12
358	2018-10-10	341.977394	226.014246	462.372529	380	2017-10-12
359	2018-10-11	332.867963	211.268112	456.160382	520	2017-10-12
360	2018-10-12	380.100152	271.892574	504.229508	318	2017-10-12

361 rows × 6 columns



Prophet includes functionality for time series cross validation to measure forecast error by comparing the predicted values with the actual values.

The output of cross_validation is a data frame with the actual value (=y) and the out-of-sample predicted value (=yhat) for each simulated forecast date and each cutoff date. In particular, predictions are made for all observed points between the cutoff and the cutoff + horizon. This data frame can be used to measure the error between yhat and y values.

The cross-validation was performed to evaluate prediction performance starting with 500 days of training data at the first cutoff and making predictions 365 days.

What is Neural Prophet?

Neural Prophet

is a PyTorch implementation of a user-friendly time series forecasting tool for practitioners based on Neural Networks. This is heavily influenced by Prophet, a popular Facebook developed forecasting tool.

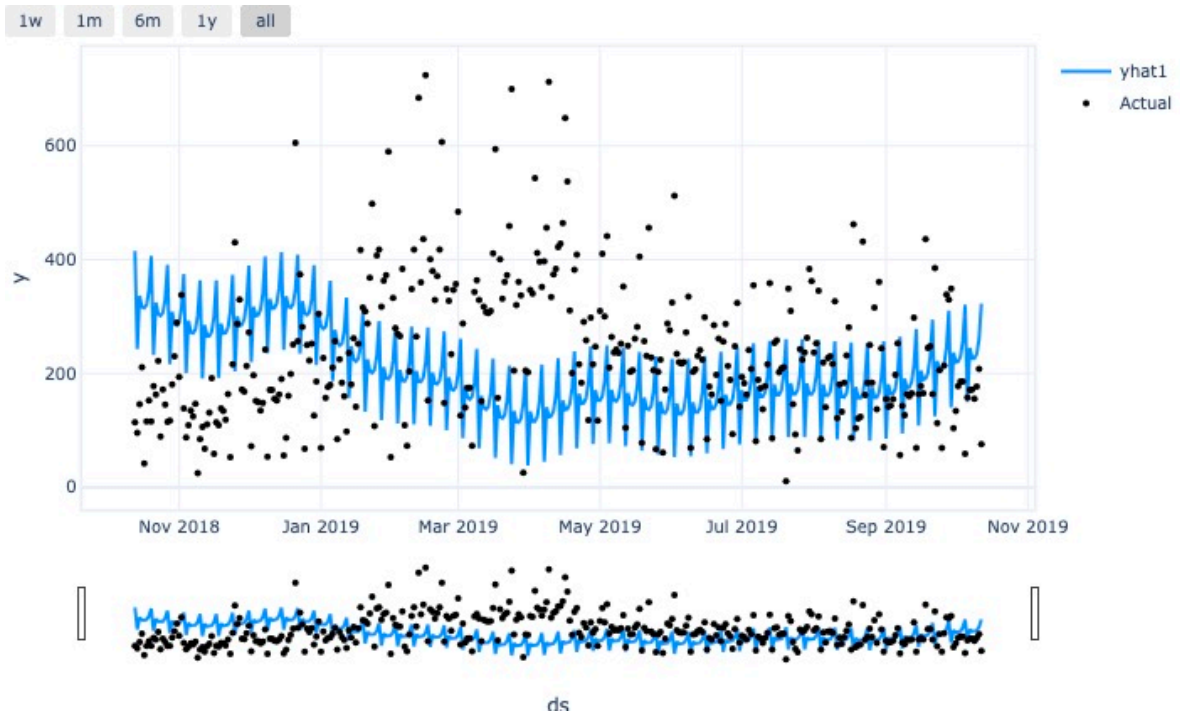
- NeuralProphet is highly scalable, easy to use, and extensible, as it is built entirely in PyTorch and trained with standard deep learning methods.
- NeuralProphet introduces local context with support for auto-regression and lagged covariates.
- NeuralProphet improves forecast accuracy with a hybrid model, where some model components can be configured as neural networks.

Advantages of NeuralProphet

- Auto-regression and covariates are supported.
- Hyperparameters related to training are automatically selected.
- Fourier term seasonality at various time scales, including yearly, daily, weekly, and hourly.
- Piecewise linear trend with automatic changepoint detection as an option.
- Forecast component plotting, model coefficient plotting, and final prediction plotting.
- Global modeling assistance.
- Future and lag regressors.
- Regularization reduces coefficient sparsity.

Neural Prophet

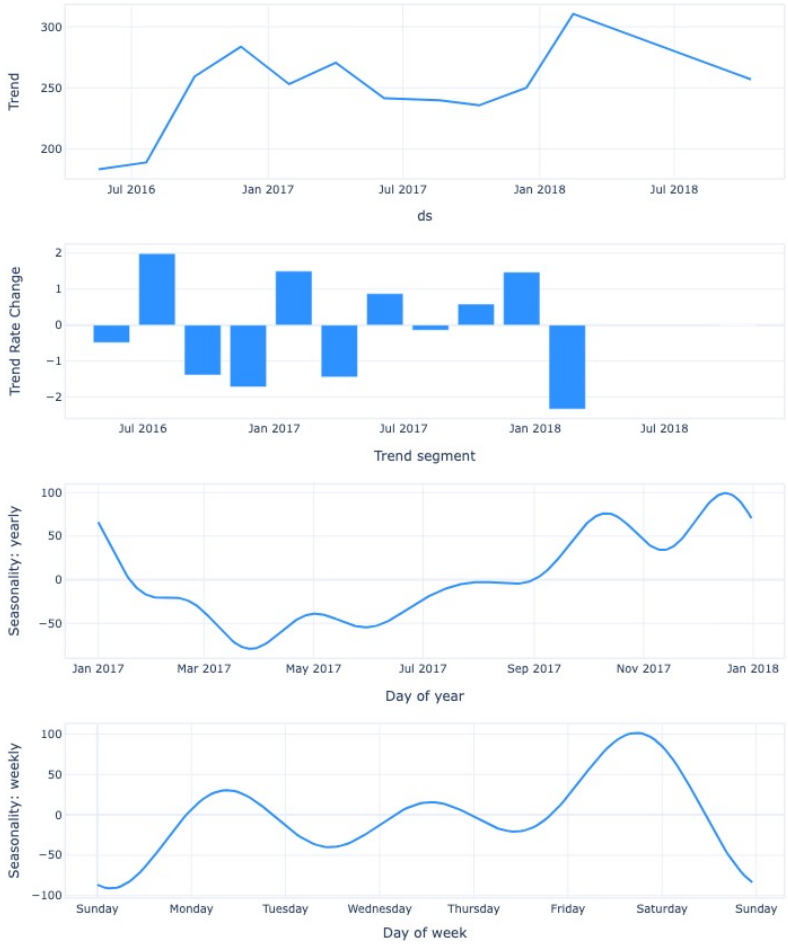
```
1 n = NeuralProphet()
2 n.fit(df_train)
3 future_neural = n.make_future_dataframe(df_train, periods=365)
4 forecast_neural = n.predict(future_neural)
```



The basic neural prophet was executed creating a data frame that is 365 periods into the future.

In the plot above, blue line is forecast sales qty forecast['yhat1'], black dots are actual sales qty forecast['y'].

Neural Prophet



The plot from Neural Prophet are almost same as the plot from Prophet

The first Trend plot shows that the total sales on a daily basis are increasing. Trend upward.

The second Trend rate change plot shows that There are some changes specific period. They could be seasonality of We can assume there is external events.

The third Yearly plot shows that as observed previously sales is higher from September to January and lower from March to July.

The fourth Weekly plot shows the sales are increasing from Thursday to Saturday and decreasing on Sunday.

Conclusion

I first tried basic prophet and turned hyper parameters but the performance hasn't been improved. The next model was NeuralProphet. Although I let Neural Prophet choose automatically the hyperparameters. I got better performance than SARIMA models. The possible explanation is that with enough training data, Prophet is more effective on very cyclical data.

ARIMA

AIC: 10195.422
RMSE: 12.0779
MSE: 145.8759
MAE: 10.9687
R2 Score: -0.9135

AUTO ARIMA

AIC: 10125,711
RMSE: 10.3959
MSE: 108.0744
MAE: 9.2446
R2 Score: -0.0503

SARIMA

AIC: 9913,754
RMSE: 13.4326
MSE: 180.4360
MAE: 12.1562
R2 Score: -1.9276

Prophet

RMSE: 12.0000
MSE: 151.0000
MAE: 11.0000
R2 Score: -1.0370

Neural Prophet

RMSE: 10.0000
MSE: 103.0000
MAE: 9.0000
R2 Score: 0.0492

In order to improve the performance of prophet model, we can modify the trend, seasonality, etc. when something seems to be different compared to the baseline model or if the prediction rate falls on a specific date, we can check and remove outliers. if the prediction rate falls at a specific cutoff (end of the year, etc.) we can create a better model by adding changepoints. Like this, the advantage of prophet is that we can use the prediction model without any expert skills. Moreover, Applying domain knowledge, we can make better model although prophet perform well the power of handling many things by itself.

MLP vs CNN vs LSTM vs CNN+LSTM

Transform the data into a time series problem

Splitting the DataFrame

```
[ ]  1 split_index = int(len(df) * 0.8)
     2 salesbyday_train = df.iloc[:split_index]
     3 salesbyday_test = df.iloc[split_index:]
     4 print('Training set shape:', salesbyday_train.shape)
     5 print('Testing set shape:', salesbyday_test.shape)
     6 print('Training set start date:', salesbyday_train['DATE'].min())
     7 print('Training set end date:', salesbyday_train['DATE'].max())
     8 print('Testing set start date:', salesbyday_test['DATE'].min())
     9 print('Testing set end date:', salesbyday_test['DATE'].max())
```

```
Training set shape: (21144, 5)
Testing set shape: (5287, 5)
Training set start date: 2016-05-18 00:00:00
Training set end date: 2018-10-20 00:00:00
Testing set start date: 2018-10-20 00:00:00
Testing set end date: 2019-06-18 00:00:00
```

Data Preprocessing

```
1 lag_size = (salesbyday_test['DATE'].max().date() - salesbyday_train['DATE'].max().date()).days
2 print('Max date from train set: %s' % salesbyday_train['DATE'].max().date())
3 print('Max date from test set: %s' % salesbyday_test['DATE'].max().date())
4 print('Forecast lag size', lag_size)
```

```
Max date from train set: 2018-10-20
Max date from test set: 2019-06-18
Forecast lag size 241
```

There is the time gap between the last day from training set from the last day of the test set, this will be lag size (the amount of day that need to be forecast).

Transform the data into a time series problem

```python
def series_to_supervised(data, window=1, lag=1, dropnan=True):
    cols, names = [], []
    for i in range(window, 0, -1):
        cols += [data.shift(i)]
        names += [f'{col}(t-{i})' for col in data.columns]

    cols += [data, data.shift(-lag)]
    names += [f'{col}(t)' for col in data.columns] + [f'{col}(t+{lag})' for col in data.columns]

    agg = pd.concat(cols, axis=1)
    agg.columns = names

    if dropnan:
        agg.dropna(inplace=True)
    return agg

window = 29
lag = lag_size = (salesbyday_test['DATE'].max().date() - salesbyday_train['DATE'].max().date()).days
series = series_to_supervised(train_gp.drop('DATE', axis=1), window=window, lag=lag)

last_item = 'PRODUCT_CATEGORY(t-%d)' % window
last_store = 'STORE_CODE(t-%d)' % window
series = series[(series['STORE_CODE(t)'] == series[last_store])]
series = series[(series['PRODUCT_CATEGORY(t)'] == series[last_item])]

columns_to_drop = [('%s(t+%d)' % (col, lag)) for col in ['STORE_CODE', 'PRODUCT_CATEGORY']]
for i in range(window, 0, -1):
    columns_to_drop += [('%s(t-%d)' % (col, i)) for col in ['STORE_CODE', 'PRODUCT_CATEGORY']]
series.drop(columns_to_drop, axis=1, inplace=True)
series.drop(['STORE_CODE(t)', 'PRODUCT_CATEGORY(t)'], axis=1, inplace=True)
```

It is designed to forecast 241 days ahead with last 30 days of data. The reason there is a difference of 241 days between the X value and the Y value is because the Test Set has the data for 241 days.
Since the data from October 20, 2018 to June 18, 2018 is the test set, there is no Y value, so the value on June 18, 2019 must be predicted using the data from October 21, 2018, the most recent data among the data we have. The difference between the two is 241 days.

Transform the data into a time series problem

										X	Y
	QTY(t-29)	QTY(t-28)	QTY(t-27)	QTY(t-26)	QTY(t-25)	...	QTY(t-3)	QTY(t-2)	QTY(t-1)	QTY(t)	QTY(t+241)
29	1.0	2.0	2.0	6.0	1.0	...	4.0	1.0	1.0	11	1.0
30	2.0	2.0	6.0	1.0	2.0	...	1.0	1.0	11.0	17	3.0
31	2.0	6.0	1.0	2.0	1.0	...	1.0	11.0	17.0	2	10.0
32	6.0	1.0	2.0	1.0	1.0	...	11.0	17.0	2.0	10	4.0
33	1.0	2.0	1.0	1.0	1.0	...	17.0	2.0	10.0	2	10.0
...	...	...	...	...	...	...	...	...	...	...	...
20898	1.0	13.0	2.0	7.0	3.0	...	1.0	2.0	1.0	1	4.0
20899	13.0	2.0	7.0	3.0	1.0	...	2.0	1.0	1.0	3	2.0
20900	2.0	7.0	3.0	1.0	2.0	...	1.0	1.0	3.0	3	10.0
20901	7.0	3.0	1.0	2.0	2.0	...	1.0	3.0	3.0	1	12.0
20902	3.0	1.0	2.0	2.0	3.0	...	3.0	3.0	1.0	1	9.0

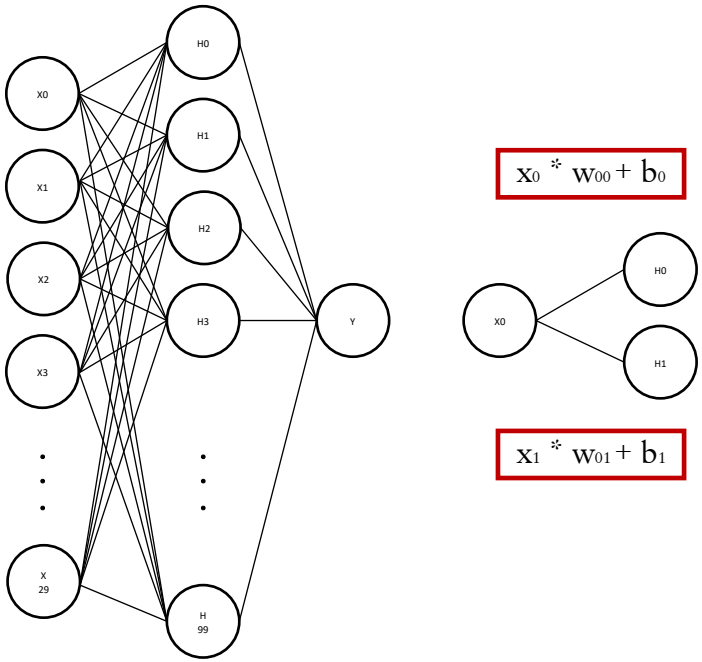
19743 rows × 31 columns

The function shifts a total of 29 days by one day and adds the value after 241 days.
In other words, based on time t, the value from t29 to t is the X value and the time t+241 is the Y value.

MLP Architecture

```
model_mlp = Sequential()
model_mlp.add(Dense(100, activation='relu', input_dim=X_train.shape[1]))
model_mlp.add(Dense(1))
model_mlp.compile(loss='mse', optimizer=adam)
model_mlp.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               3100

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 3201 (12.50 KB)
Trainable params: 3201 (12.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



$x_0 * w_{00} + b_0$

$x_1 * w_{01} + b_1$

MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer consists of nodes (or neurons), which are fully connected to nodes in the next layer. The core concept of MLP is “depth” or “layers,” which increases the number of nodes and complexity of combinations required to learn complex patterns.

The model has input features equal to the window size. Input shape [samples, timesteps]

This architecture has 30 matching data x_0 ~ x_{29} as input, one hidden layer with 100 neurons and one output.

To calculate x_0 as h_0, one weight is multiplied and one bias is added.

The required number of parameters requires a weight of (30X100) and a bias of (1X100) to be calculated from the input layer of (1X30) to the hidden layer of (1X100). In other words, 3100 parameters are needed up to Dense(100).

CNN Architecture

```
model_cnn = Sequential()
model_cnn.add(Conv1D(filters=64, kernel_size=2, activation='relu',
                     input_shape=(X_train_series.shape[1], X_train_series.shape[2])))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(50, activation='relu'))
model_cnn.add(Dense(1))
model_cnn.compile(loss='mse', optimizer=legacy.Adadelta(lr=0.01))
model_cnn.summary()
```

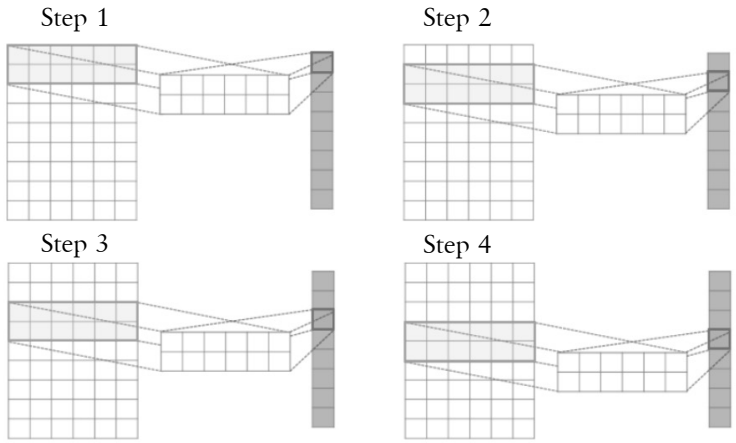
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_1 (Conv1D)           (None, 29, 64)            192

 max_pooling1d_1 (MaxPoolin  (None, 14, 64)            0
 g1D)

 flatten_1 (Flatten)         (None, 896)               0

 dense_4 (Dense)             (None, 50)                44850

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 45093 (176.14 KB)
Trainable params: 45093 (176.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Data preprocessing
Reshape from [samples, timesteps] into [samples, timesteps, features]. This same reshaped data will be used on both CNN and LSTM model.

One convolutional hidden layer followed by a max pooling layer and The filter maps are then flattened before being interpreted by a Dense layer and outputting a prediction.

The convolutional layer should be able to identify patterns between the timesteps.

1D Convolution

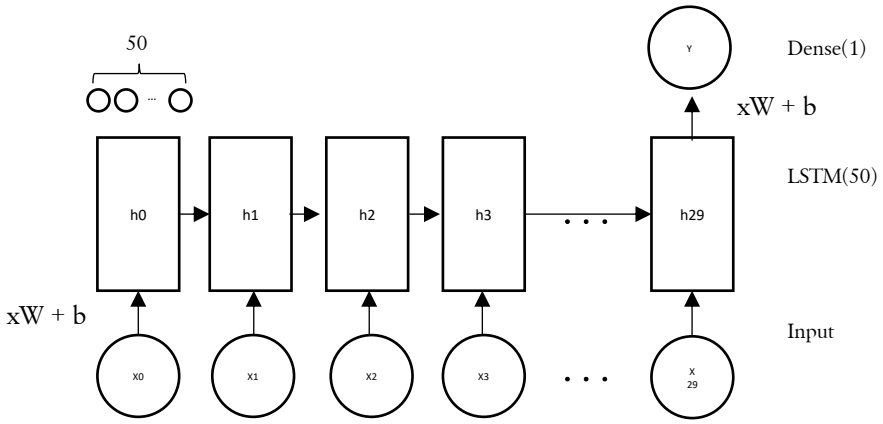


CNN 2 Dimension output shape consists of a total of 4 dimensions (Batch_size, width, height, depth). Since our Sales data is one-dimensional, the input shape consists of 30 heights and 1 depth without width.

LSTM Architecture

```
model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(X_train_series.shape[1],
                                                        X_train_series.shape[2])))
model_lstm.add(Dense(1))
model_lstm.compile(loss='mse', optimizer=legacy.Adam(lr=0.01))
model_lstm.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense_6 (Dense)             (None, 1)                 51

=================================================================
Total params: 10451 (40.82 KB)
Trainable params: 10451 (40.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



Recurrent neural networks have 'short term memory' in that they use persistent previous information to be used in the current neural network. Essentially, the previous information is used in the present task. LSTM introduces long-term memory into recurrent neural networks. It mitigates the vanishing gradient problem, which is where the neural network stops learning because the updates to the various weights within a given neural network become smaller and smaller.

LSTM model sees the input data as a sequence, so it's able to learn patterns from sequenced data better than the other ones, especially patterns from long sequences.

(1X1) Dimension x_0 is multiplied by (1X50) Weight and Bias is added to become (1X50) h_0.

Multiply x_1 by Weight, add Bias, and add h_0 to become h1.

Multiply h_{29} (1X50) by Weight (50X1) and add Bias (1X1) to get the final output of (1X1).

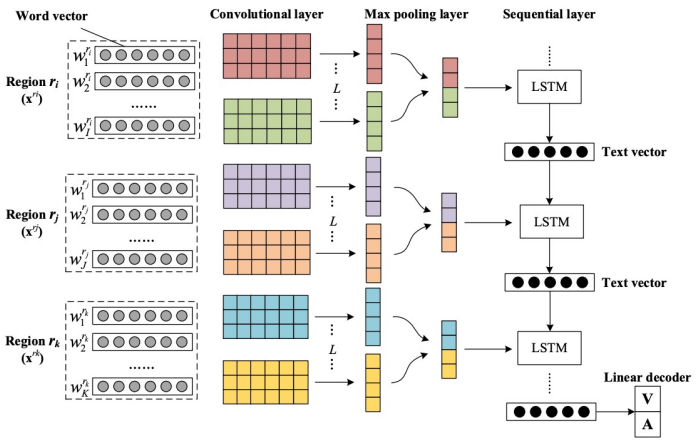
CNN+LSTM

Data preprocessing
Reshape from [samples, timesteps, features] into [samples, subsequences, timesteps, features].

```
subsequences = 2
timesteps = X_train_series.shape[1]//subsequences
X_train_series_sub = X_train_series.reshape((X_train_series.shape[0], subsequences, timesteps, 1))
X_valid_series_sub = X_valid_series.reshape((X_valid_series.shape[0], subsequences, timesteps, 1))
print('Train set shape', X_train_series_sub.shape)
print('Validation set shape', X_valid_series_sub.shape)
```

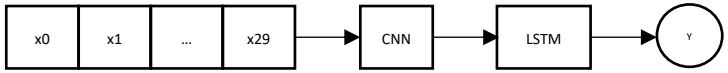
```
Train set shape (11845, 2, 15, 1)
Validation set shape (7898, 2, 15, 1)
```

The framework of the CNN-LSTM model

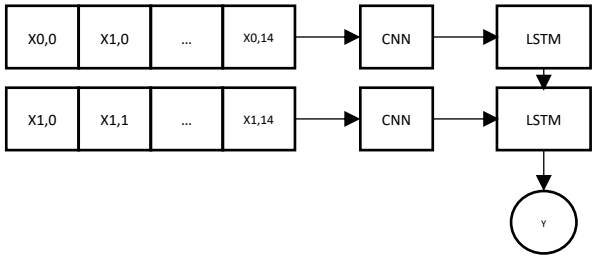


Sources: https://aclanthology.org/P16-2037/

LSTM is divided into two because each input must come in according to the time stamp.
If it had been entered into the input in 1X30 format without dividing, the architecture would have been as follows.



Since it has been changed to 2X15 form, the architecture has the following form.



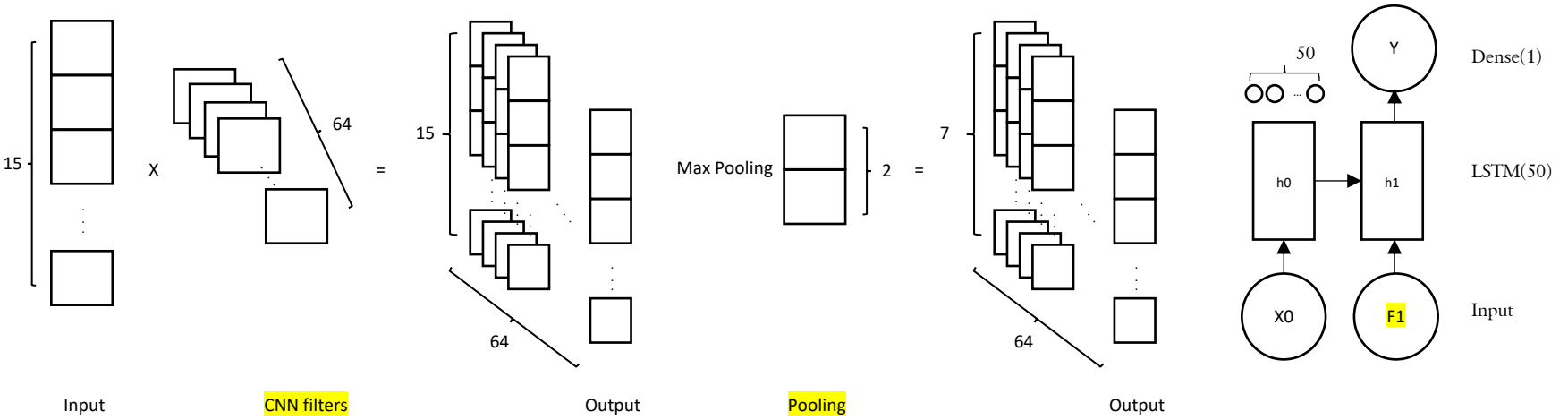
CNN+LSTM Architecture

```
model_cnn_lstm = Sequential()
model_cnn_lstm.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'),
                                   input_shape=(None, X_train_series_sub.shape[2],
                                                X_train_series_sub.shape[3])))
model_cnn_lstm.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model_cnn_lstm.add(TimeDistributed(Flatten()))
model_cnn_lstm.add(LSTM(50, activation='relu'))
model_cnn_lstm.add(Dense(1))
model_cnn_lstm.compile(loss='mse', optimizer=legacy.Adam(lr=0.01))
model_cnn_lstm.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDist  (None, None, 15, 64)      128
 ributed)

 time_distributed_1 (TimeDi  (None, None, 7, 64)       0
 stributed)

 time_distributed_2 (TimeDi  (None, None, 448)         0
 stributed)

 lstm_2 (LSTM)               (None, 50)                99800

 dense_10 (Dense)            (None, 1)                 51

=================================================================
Total params: 99979 (390.54 KB)
Trainable params: 99979 (390.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

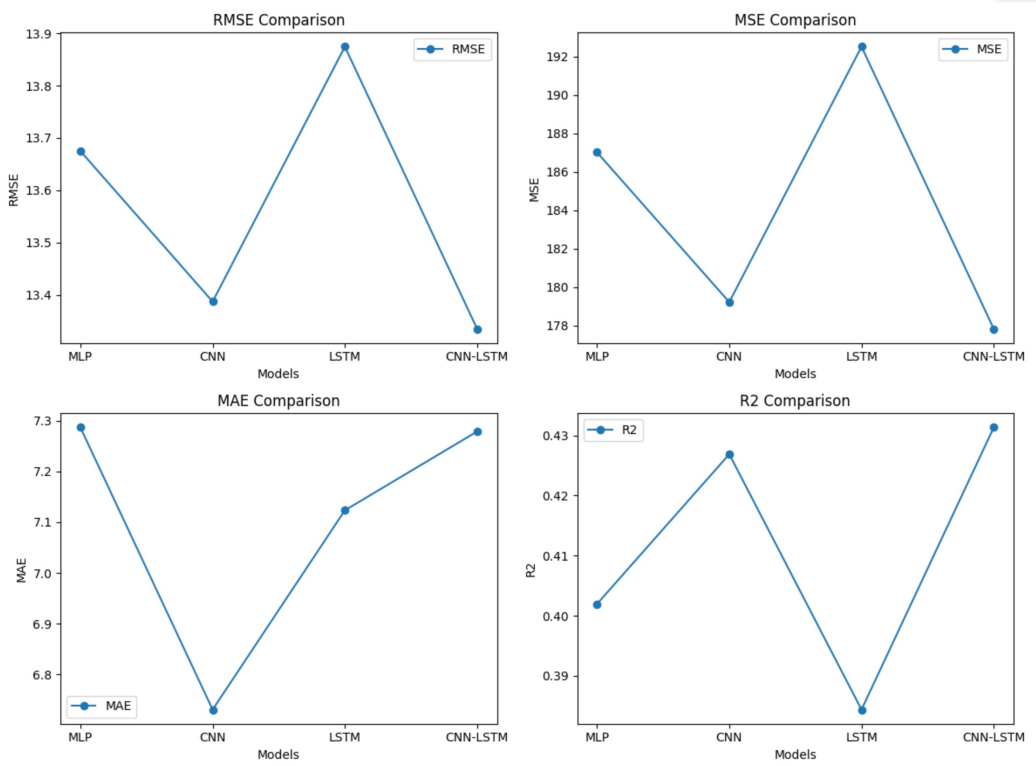
The benefit of this model is that the model can support very long input sequences that can be read as blocks or subsequences by the CNN model, then pieced together by the LSTM model. The CNN model interprets each sub-sequence and the LSTM pieces together the interpretations from the subsequences.

The CNN is defined to expect 2 time steps per subsequence with one feature. The entire CNN model is then wrapped in TimeDistributed wrapper layers so that it can be applied to each subsequence in the sample. Otherwise, the time steps are not divided and are learned with a single convolution filter. The results are then interpreted by the LSTM layer before the model outputs a prediction.



The first convolution layer applies 64 convolution filters of size 1 to an input of height 15 and depth 1, resulting in an output of 15X64. The size of the second MaxPooling Layer is 2, so the height, which was 15 becomes 7. When flattened into one line, 7*64=448 is obtained. If the result of this layer is expressed as F, LSTM(50) and Dense(1) are as follows. Considering that the subsequence of the Train Set is 2, it was diagrammed with 2 inputs.

Comparison



MLP :
RMSE: 13.6749
MSE: 187.0021
MAE: 7.2863
R2: 0.4020

CNN :
RMSE: 13.3870
MSE: 179.2110
MAE: 6.7308
R2: 0.4269

LSTM :
RMSE: 13.8745
MSE: 192.5004
MAE: 7.1232
R2: 0.3844

CNN-LSTM :
RMSE: 13.3349
MSE: 177.8190
MAE: 7.2783
R2: 0.4313

Conclusion

ARIMA	AUTO ARIMA	SARIMA	Prophet	Neural Prophet
AIC: 10195.422	AIC: 10125,711	AIC: 9913,754	RMSE: 12.0000	RMSE: 10.0000
RMSE: 12.0779	RMSE: 10.3959	RMSE: 13.4326	MSE: 151.0000	MSE: 103.0000
MSE: 145.8759	MSE: 108.0744	MSE: 180.4360	MAE: 11.0000	MAE: 9.0000
MAE: 10.9687	MAE: 9.2446	MAE: 12.1562	R2 Score: -1.0370	R2 Score: 0.0492
R2 Score: -0.9135	R2 Score: -0.0503	R2 Score: -1.9276		

MLP	CNN	LSTM	CNN-LSTM
RMSE: 13.6749	RMSE: 13.3870	RMSE: 13.8745	RMSE: 13.3349
MSE: 187.0021	MSE: 179.2110	MSE: 192.5004	MSE: 177.8190
MAE: 7.2863	MAE: 6.7308	MAE: 7.1232	MAE: 7.2783
R2: 0.4020	R2: 0.4269	R2: 0.3844	R2: 0.4313

Observations and Considerations:

AIC: SARIMA has the lowest AIC.
RMSE, MSE and MAE: Neural Prophet has the lowest RMSE, MSE and CNN has the lowest MAE.
R2 Score: CNN-LSTM have higher R2 Scores.

Considering these factors, the choice of the "best" model depends on specific goals and requirements. If simplicity and interpretability are important, SARIMA might be favored due to its lower AIC. If predictive accuracy is the primary concern, CNN-LSTM may be preferred based on their lower R2 score. Ultimately, it's often beneficial to choose a model that strikes a balance between interpretability and predictive performance, depending on the context of application.

Advantages & Disadvantage of the Models

ARIMA (Autoregressive Integrated Moving Average):
A : Captures trends and seasonality well.
Easy to understand and interpret.
D : Limited capability to handle nonlinear patterns.
May require significant preprocessing.

AUTO ARIMA:
A : Reduces the burden on the user for ARIMA model selection.
D : Does not guarantee the selection of the optimal model.
Can be computationally expensive.

SARIMA (Seasonal ARIMA):
A : Improves forecasting by considering seasonality.
D : Identifying and configuring seasonality can be challenging.

Prophet:
A : User-friendly and handles holidays and events well.
D : Performance may be limited for complex nonlinear patterns.

Neural Prophet:
A : Combines the simplicity of Prophet with the flexibility of neural networks. Can learn nonlinear patterns.
D : Prone to overfitting in some datasets.

MLP (Multi-Layer Perceptron):
A : Can learn complex nonlinear patterns.
Handles a variety of features well.
D : Requires careful parameter tuning, and can be prone to overfitting.

CNN (Convolutional Neural Network):
A : Excels at learning spatial patterns, suitable for image data.
Recognizes local patterns effectively.
D : Traditionally not well-suited for typical time series data.

LSTM (Long Short-Term Memory):
A : Captures long-term dependencies.
Strong for sequential data.
D : Requires extensive parameter tuning and can be computationally intensive.

CNN-LSTM:
A : Effectively processes time series data by learning spatial and temporal features together.
Captures complex patterns.
D : Requires significant parameter tuning and training time. May overfit on some datasets.

References

https://facebook.github.io/prophet/

https://arxiv.org/pdf/1911.12436.pdf

https://ai.meta.com/blog/neuralprophet-the-neural-evolution-of-facebooks-prophet/

https://paperswithcode.com/paper/neuralprophet-explainable-forecasting-at

https://machinelearningmastery.com/how-to-get-started-with-deep-learning-for-time-series-forecasting-7-day-mini-course/

https://aclanthology.org/P16-2037.pdf