MACHINE LEARNING PROJECT

1. **What is Data Science?**

    a. The Data Science Lifecycle
    b. Prerequisites for Data Science
    c. What Does a Data Scientist Do?
    d. Why Become a Data Scientist?

2. **Introduction to Machine Learning**

    a. Types of machine learning
    b. Problems in Supervised Machine Learning

3. **Learning Dataset**

4. **What is Time Series Analysis?**

    a. Components of time series
    b. Why organizations use time series data analysis
    c. Examples of time series datasets

5. **Working with Time Series Models**

    a. ARIMA
    b. Facebook Prophet
    c. PyCaret (Library)

6. **Conclusion**

What is Data Science?

Data science is the domain of study that deals with vast volumes of data using modern tools and techniques to find unseen patterns, derive meaningful information, and make business decisions. Data science uses complex machine learning algorithms to build predictive models.

The data used for analysis can come from many different sources and presented in various formats.

Now that you know what data science is, let's see why data science is essential to today's IT landscape.

The Data Science Lifecycle:

Now that we know what is data science, next up let us focus on the data science lifecycle. Data science's lifecycle consists of five distinct stages, each with its own tasks:

1. Capture: Data Acquisition, Data Entry, Signal Reception, Data Extraction. This stage involves gathering raw structured and unstructured data.
2. Maintain: Data Warehousing, Data Cleansing, Data Staging, Data Processing, Data Architecture. This stage covers taking the raw data and putting it in a form that can be used.
3. Process: Data Mining, Clustering/Classification, Data Modeling, Data Summarization. Data scientists take the prepared data and examine its patterns, ranges, and biases to determine how useful it will be in predictive analysis.
4. Analyze: Exploratory/Confirmatory, Predictive Analysis, Regression, Text Mining, Qualitative Analysis. Here is the real meat of the lifecycle. This stage involves performing the various analyses on the data.
5. Communicate: Data Reporting, Data Visualization, Business Intelligence, Decision Making. In this final step, analysts prepare the analyses in easily readable forms such as charts, graphs, and reports.

Prerequisites for Data Science:

1. **Machine Learning** -

Machine learning is the backbone of data science. Data Scientists need to have a solid grasp of ML in addition to basic knowledge of statistics.

2. **Modeling** -

Mathematical models enable you to make quick calculations and predictions based on what you already know about the data. Modeling is also a part of Machine Learning and involves identifying which algorithm is the most suitable to solve a given problem and how to train these models.

3. **Statistics** -

Statistics are at the core of data science. A sturdy handle on statistics can help you extract more intelligence and obtain more meaningful results.

4. **Programming** -

Some level of programming is required to execute a successful data science project. The most common programming languages are Python, and R. Python is especially popular because it's easy to learn, and it supports multiple libraries for data science and ML.

5. **Databases** -

A capable data scientist needs to understand how databases work, how to manage them, and how to extract data from them.

What Does a Data Scientist Do?

You know what is data science, and you must be wondering what exactly is this job role like - here's the answer. A data scientist analyzes business data to extract meaningful insights. In other words, a data scientist solves business problems through a series of steps, including:

- Before tackling the data collection and analysis, the data scientist determines the problem by asking the right questions and gaining understanding.
- The data scientist then determines the correct set of variables and data sets.
- The data scientist gathers structured and unstructured data from many disparate sources—enterprise data, public data, etc.
- Once the data is collected, the data scientist processes the raw data and converts it into a format suitable for analysis. This involves cleaning and validating the data to guarantee uniformity, completeness, and accuracy.

- After the data has been rendered into a usable form, it's fed into the analytic system—ML algorithm or a statistical model. This is where the data scientists analyze and identify patterns and trends.
- When the data has been completely rendered, the data scientist interprets the data to find opportunities and solutions.
- The data scientists finish the task by preparing the results and insights to share with the appropriate stakeholders and communicating the results.
- Now we should be aware of some machine learning algorithms which are beneficial in understanding data science clearly.

Why Become a Data Scientist?

You learnt what is data science. Here's another solid reason why you should pursue data science as your work-field. According to Glassdoor and Forbes, demand for data scientists will increase by 28 percent by 2026, which speaks of the profession's durability and longevity, so if you want a secure career, data science offers you that chance.

Furthermore, the profession of data scientist came in second place in the Best Jobs in America for 2021 survey, with an average base salary of USD 127,500.

Introduction to Machine learning

Machine learning is a sub-domain of computer science which evolved from the study of pattern recognition in data, and also from the computational learning theory in artificial intelligence. It is the first-class ticket to most interesting careers in data analytics today. As data sources proliferate along with the computing power to process them, going straight to the data is one of the most straightforward ways to quickly gain insights and make predictions.

Types of Machine Learning:

There are two types of machine learning algorithms: Supervised and Unsupervised:

1. Supervised: Supervised learning algorithms are used when the data is labeled. There are two types:
    - Regression: When you need to predict continuous values and variables are linearly dependent, algorithms used are linear and multiple regression, decision trees and random forest

- Classification: When you need to predict categorical values, some of the classification algorithms used are KNN, logistic regression, SVM and Naïve-Bayes

2. Unsupervised: Unsupervised learning algorithms are used when the data is unlabeled, there is no labeled data to learn from. There are two types:
   - Clustering: This is the method of dividing the objects which are similar between them and dissimilar to others. K-Means and PCA clustering algorithms are commonly used.
   - Association-rule analysis: This is used to discover interesting relations between variables, Apriori and Hidden Markov Model algorithm can be used
   - Model Maintenance: After gathering data and performing the mining and model building, data scientists must maintain the model accuracy. Thus, they take the following steps:
   - Assess: Running a sample through the data occasionally to make sure it remains accurate

Retrain: When the results of the reassessment aren't right, the data scientist must retrain the algorithm to provide the correct results again

Rebuild: If retraining fails, rebuilding must occur.

Problems in Supervised Machine Learning:

Before we get started, we must know about how to pick a good machine learning algorithm for the given dataset. To intelligently pick an algorithm to use for a supervised learning task, we must consider the following factors:

1. **Heterogeneity of Data**: Many algorithms like neural networks and support vector machines like their feature vectors to be homogeneous numeric and normalized. The algorithms that employ distance metrics are very sensitive to this, and hence if the data is heterogeneous, these methods should be the afterthought. Decision Trees can handle heterogeneous data very easily.

2. **Redundancy of Data**: If the data contains redundant information, i.e. contain highly correlated values, then it's useless to use distance based methods because of numerical instability. In this case, some sort of Regularization can be employed to the data to prevent this situation.

3. **Dependent Features:** If there is some dependence between the feature vectors, then algorithms that monitor complex interactions like Neural Networks and Decision Trees fare better than other algorithms.

4. **Bias-Variance Tradeoff**: A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x, whereas a learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets. The prediction error of a learned classifier can be related to the sum of bias and variance of the learning algorithm, and neither can be high as they will make the prediction error to be high.

5. **Curse of Dimensionality**: If the problem has an input space that has a large number of dimensions, and the problem only depends on a subspace of the input space with small dimensions, the machine learning algorithm can be confused by the huge number of dimensions and hence the variance of the algorithm can be high. In practice, if the data scientist can manually remove irrelevant features from the input data, this is likely to improve the accuracy of the learned function.

6. **Overfitting**: The programmer should know that there is a possibility that the output values may constitute of an inherent noise which is the result of human or sensor errors. In this case, the algorithm must not attempt to infer the function that exactly matches all the data. Being too careful in fitting the data can cause overfitting, after which the model will answer perfectly for all training examples but will have a very high error for unseen samples. A practical way of preventing this is stopping the learning process prematurely, as well as applying filters to the data in the pre-learning phase to remove noises.

Learning Dataset

The dataset used contains generation of ticket for a particular query with a unique transaction id. The dataset is containing 2 years of data to work on. The main objective of this project is to predict the number of tickets generated in the future so that the company can take respective measures to solve the issues regarding the overworking of the employees. This dataset contains 17 columns namely:

1. Zone
2. Dealer Code
3. Dealer Name
4. Transaction No.
5. Description
6. Status
7. Posting Date
8. IRT

9. CATEGORY
10. IRTSTATUS
11. MPT
12. MPTSTATUS
13. Changed On
14. TRANSACTIONTYPE
15. Created By
16. Reported By
17. Object GUID

There are total of 97126 rows indicating ticket generation data for each. On preprocessing the data we get 793 rows in total to get number of tickets generated on each day.

What is Time Series Analysis?

A time series is a data set that looks at a certain metric over a period of time. An easy example of this would be the weather. If we want to predict what the temperature will be, we will have to analyze historical weather data over a period of time to learn the patterns in order to estimate what will happen. Certain variables that could influence temperature are the time of year, humidity levels, physical location in the world, altitude, and distance from a large body of water, to name a few. A time series analysis looks at the relationship between the dependent variable (in this case, temperature) in comparison to the independent variables (ex. time of year, latitude/longitude, humidity) to determine the impact of each. A time series analysis can also look at how timing impacts the dependent variable in the form of seasonality or overall upward or downward trends.

Seasonality are trends within the data that occur at specific times. In the northern hemisphere, it is expected for temperatures to drop from November through March. A time series analysis should be able to find that trend and incorporate it when forecasting temperature.

A time series should also be able to consider macro trends. In the weather example, a macro trend would be if a given area has been seeing an increase or decrease in temperature over time as a result of climate change.

Components of time series:

The various reasons or the forces which affect the values of an observation in a time series are the components of a time series. The four categories of the components of time series are

- Trend
- Seasonal Variations
- Cyclic Variations
- Random or Irregular movements

Why organizations use time series data analysis?

Time series analysis helps organizations understand the underlying causes of trends or systemic patterns over time. Using data visualizations, business users can see seasonal trends and dig deeper into why these trends occur. With modern analytics platforms, these visualizations can go far beyond line graphs.

When organizations analyze data over consistent intervals, they can also use time series forecasting to predict the likelihood of future events. Time series forecasting is part of predictive analytics. It can show likely changes in the data, like seasonality or cyclic behavior, which provides a better understanding of data variables and helps forecast better.

Time series analysis examples

Time series analysis is used for non-stationary data—things that are constantly fluctuating over time or are affected by time. Industries like finance, retail, and economics frequently use time series analysis because currency and sales are always changing. Stock market analysis is an excellent example of time series analysis in action, especially with automated trading algorithms. Likewise, time series analysis is ideal for forecasting weather changes, helping meteorologists predict everything from tomorrow's weather report to future years of climate change. Examples of time series analysis in action include:

- Weather data
- Rainfall measurements
- Temperature readings
- Heart rate monitoring (EKG)
- Brain monitoring (EEG)

- Quarterly sales
- Stock prices
- Automated stock trading
- Industry forecasts
- Interest rates

Working with Time series models

1. ARIMA :

An autoregressive integrated moving average – ARIMA model is a generalization of a simple autoregressive moving average – ARMA model. Both of these models are used to forecast or predict future points in the time-series data. ARIMA is a form of regression analysis that indicates the strength of a dependent variable relative to other changing variables.

The final objective of the model is to predict future time series movement by examining the differences between values in the series instead of through actual values. ARIMA models are applied in the cases where the data shows evidence of non-stationarity. In time series analysis, non-stationary data are always transformed into stationary data.

The common causes of non-stationary in time series data are the trend and the seasonal components. The way to transformed non-stationary data to stationary is to apply the differencing step. It is possible to apply one or more times of differencing steps to eliminate the trend component in the data. Similarly, to remove the seasonal components in data, seasonal differencing could be applied.

According to the name, we can split the model into smaller components as follow:

1. AR: an AutoregRegressive model which represents a type of random process. The output of the model is linearly dependent on its own previous value i.e. some number of lagged data points or the number of past observations.
2. MA: a Moving Average model which output is dependent linearly on the current and various past observations of a stochastic term.

3. I: integrated here means the differencing step to generate stationary time series data, i.e. removing the seasonal and trend components.

   ARIMA model is generally denoted as ARIMA (p, d, q) and parameter p, d, q are defined as follow:

   - p: the lag order or the number of time lag of autoregressive model AR(p)
   - d: degree of differencing or the number of times the data have had subtracted with past value
   - q: the order of moving average model MA(q)

Performing Time Series Analysis using ARIMA Model in Python

1. Import Necessary Libraries

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

2. Reading Dataset

```
[ ]  df_c=pd.read_csv("/content/drive/MyDrive/cleanedData.csv")
```

```
[ ]  df.head()
```

	Zone	Dealer Code	Dealer Name	Transaction No.	Description	Status	Posting Date	IRT	CATEGORY	IRTSTATUS	MPT	MPTSTATUS	Changed On	TRANSACTIONTYPE	Created By	R
0	NaN	NaN	NaN	2000095635	Medium	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	ANANDSHARMA	
1	NaN	NaN	NaN	1000003085	Very High	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	S309PRM0001	
2	NaN	NaN	NaN	1000003086	Medium	Open	2022-05-06	0	NaN	NaN	0	NaN	2022-05-06	MG Parts Support	WW14PRM0001	
3	NaN	NaN	NaN	2000095644	Very High	Open	2022-05-06	0	After Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DW02HSR0001	
4	NaN	NaN	NaN	2000095646	High	Open	2022-05-06	0	Sales	NaN	0	NaN	2022-05-06	MG_MOTORS_Incident	DE04CRE0001	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97125 entries, 0 to 97124
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Zone             95080 non-null  object
 1   Dealer Code      95080 non-null  object
 2   Dealer Name      95056 non-null  object
 3   Transaction No.  97125 non-null  int64
 4   Description      97118 non-null  object
 5   Status           97125 non-null  object
 6   Posting Date     97125 non-null  datetime64[ns]
 7   IRT              97125 non-null  int64
 8   CATEGORY         92095 non-null  object
 9   IRTSTATUS        3790 non-null   object
 10  MPT              97125 non-null  int64
 11  MPTSTATUS        10610 non-null  object
 12  Changed On       97125 non-null  datetime64[ns]
 13  TRANSACTIONTYPE  97124 non-null  object
 14  Created By       97125 non-null  object
 15  Reported By      90933 non-null  object
 16  Object GUID      97125 non-null  int64
dtypes: datetime64[ns](2), int64(4), object(11)
memory usage: 12.6+ MB
```

3. Data Preprocessing

```
a=np.arange(0,97125)
a.reshape(1,97125)
b=np.ones_like(a)
b
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
df["resolved"]=b
```

- A NumPy array is added to dataset to count number of tickets. Then using groupby and reset index functions we get the final result as follows-

```
[ ] dfs=df[["Posting Date","resolved"]].copy()
```

```
[ ] dfs.head()
```

	Posting Date	resolved
0	2022-05-06	1
1	2022-05-06	1
2	2022-05-06	1
3	2022-05-06	1
4	2022-05-06	1

```
[ ] dfs.columns=["Date","tickets"]
```

```
[ ] dfs["Date"]=pd.to_datetime(dfs["Date"])
```

```
[ ] dfs=dfs.groupby([dfs["Date"].dt.year,dfs["Date"].dt.month,dfs["Date"].dt.day]).agg({"tickets":sum})
```

```
dfs.head
```

```
<bound method NDFrame.head of                tickets
Date Date Date
2020 2    27          1
          28         14
          29          4
     3    2           1
          3           2
...                  ...
2022 5    2          83
          3          74
          4         116
          5         125
          6          95

[794 rows x 1 columns]>
```

```
dfs.reset_index(level=2,inplace=True)
dfs.columns=["day","sales"]
```

```
dfs.reset_index(level=1,inplace=True)
```

```
dfs.columns=["month","day","T"]
```

```
dfs.reset_index(inplace=True)
```

```
dfs.columns=["year","month","day","T"]
```

```
def ret_time(x):
  return pd.to_datetime(str(x["year"])+"-"+str(x["month"])+"-"+str(x["day"]))
```

```
dfs.drop(columns=["year","month","day"],inplace=True)
dfs=dfs[["date","T"]]
dfs.head()
```

	date	T
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

- So now the dataset is ready for using in arima model after saving it as .csv file.
- As per data it is observed that Sundays is having less number of tickets generated which is interfering with the consistency of data, so Sundays are removed using following code:

```
[ ]  df_c['D']=df_c['date'].apply(lambda time: time.dayofweek)
```

```
[ ]  df=df_c[(df_c['D']<6)]
```

```
[ ]  df
```

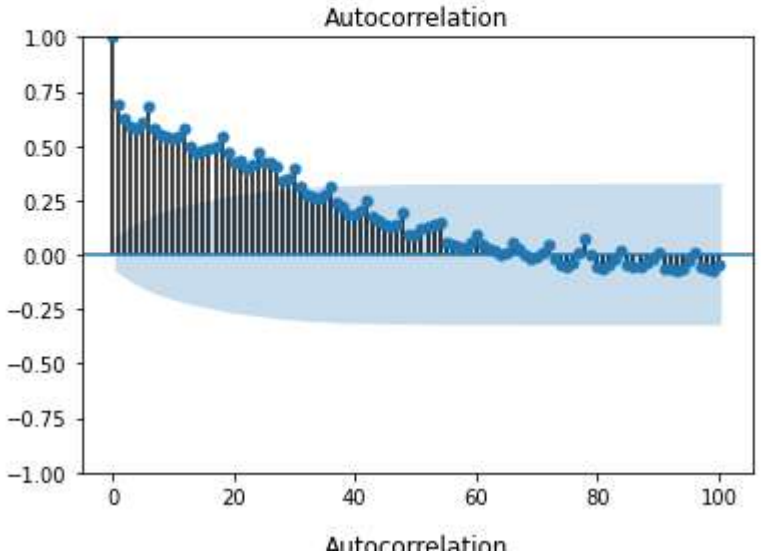
	date	T	D
0	2020-02-27	1	3
1	2020-02-28	14	4
2	2020-02-29	4	5
3	2020-03-02	1	0
4	2020-03-03	2	1
...	...	...	...
789	2022-05-02	83	0
790	2022-05-03	74	1
791	2022-05-04	116	2

- Next we will look at the Null Hypothesis test using ndiffs package.

```
from statsmodels.tsa.stattools import adfuller
result=adfuller(df["T"])
print("ADF-",result[0])
print("P-",result[1])
```

```
ADF- -2.479276409641882
P- 0.12059836111971262
```

```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(df["T"],lags=100)
```



- Installing the Library and fitting the dataset

```
!pip install pmdarima
```

```
[ ] from pmdarima import auto_arima
```

```
[ ] from inspect import trace
    model=auto_arima(df["T"],start_p=1,start_q=1,test="kpss",
                     max_p=10,max_q=10,
                     m=1,
                     d=None,
                     seasonal=True,
                     start_P=0,
                     D=0,
                     trace=True,
                     error_action='ignore',
                     suppress_warnings=True,
                     stepwise=True
                    )
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=6908.452, Time=1.29 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=7172.839, Time=0.09 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=7064.865, Time=0.37 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=6920.099, Time=0.79 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=7170.845, Time=0.08 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=6909.254, Time=2.10 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=6909.833, Time=2.84 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=6907.839, Time=1.00 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=6909.828, Time=1.53 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=5.06 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=6906.211, Time=0.33 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=6918.396, Time=0.28 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=6908.206, Time=0.57 sec
 ARIMA(0,1,3)(0,0,0)[0]             : AIC=6908.202, Time=0.46 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=6906.832, Time=0.52 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=inf, Time=2.64 sec

Best model:  ARIMA(0,1,2)(0,0,0)[0]
Total fit time: 20.140 seconds
```

- Dividing the dataset into training and test data. Then forecasting on trained data.

```
[ ] trained=df.iloc[85:600]
    testes=df.iloc[600:]
    trained.head()
```

	date	T	D
97	2020-06-09	148	1
98	2020-06-10	135	2
99	2020-06-11	136	3
100	2020-06-12	137	4
101	2020-06-13	102	5

```
[ ]  model=auto_arima(trained["T"],start_p=1,start_q=1,test="kpss",
                      max_p=10,max_q=5,
                      m=7,
                      d=None,
                      seasonal=False,
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True
                     )
```

```
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_validation.py:62: UserWarning: m (7) set for non-seasonal fit. Setting to 0
  warnings.warn("m (%i) set for non-seasonal fit. Setting to 0" % m)
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=5271.880, Time=0.38 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=6698.727, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=5453.674, Time=0.07 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=6242.833, Time=0.23 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=5262.764, Time=0.71 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=5384.035, Time=0.23 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=5262.826, Time=0.89 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=5263.777, Time=0.88 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=5261.799, Time=0.29 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=5983.760, Time=0.16 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=5263.753, Time=0.40 sec
 ARIMA(0,0,3)(0,0,0)[0]             : AIC=5824.100, Time=0.35 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=inf, Time=1.09 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=5252.499, Time=1.01 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=5318.172, Time=0.23 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=5260.964, Time=0.74 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=1.66 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=5254.338, Time=2.22 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=5338.161, Time=0.51 sec
 ARIMA(0,0,3)(0,0,0)[0] intercept   : AIC=5306.234, Time=0.67 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=5254.991, Time=1.31 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=inf, Time=1.23 sec

Best model:  ARIMA(1,0,2)(0,0,0)[0] intercept
Total fit time: 15.356 seconds
```

```
[ ]  model.fit(trained["T"])
```

```
ARIMA(order=(1, 0, 2), scoring_args={}, suppress_warnings=True)
```

```
[ ]  len(testes)
```

```
83
```

```
[ ]  forecast,confidence_interval=model.predict(n_periods=83,return_conf_int=True,alpha=0.05)
```

- Finally getting the forecasted data and comparing with test data.

```
testes["forecast"]=forecast
testes["lower"]=confidence_interval[:,0]
testes["higher"]=confidence_interval[:,1]
testes
```

	date	T	D	forecast	lower	higher
698	2022-01-31	243	0	156.915240	79.966474	233.864007
699	2022-02-01	137	1	153.852317	73.510950	234.193684
700	2022-02-02	134	2	153.843210	72.698010	234.988410
701	2022-02-03	172	3	153.834429	71.949132	235.719727
702	2022-02-04	190	4	153.825964	71.258685	236.393243
...	...	...	...	...	...	...
789	2022-05-02	83	0	153.613754	62.565843	244.661665
790	2022-05-03	74	1	153.613208	62.562735	244.663681
791	2022-05-04	116	2	153.612682	62.559826	244.665537
792	2022-05-05	125	3	153.612174	62.557105	244.667243
793	2022-05-06	95	4	153.611685	62.554557	244.668812

83 rows × 6 columns

```
from pmdarima.metrics import smape
```

```
mpe=smape(testes["T"],testes["forecast"])
print(mpe)
```

```
22.4979130944301
```

Finally by using smape function we get the accuracy as approx. 78%.

2. Prophet Model

FB Prophet is a forecasting package in both R and Python that was developed by Facebook's data science research team. The goal of the package is to give business users a powerful and easy-to-use tool to help forecast business results without needing to be an expert in time series analysis. There is thorough documentation of the package, how to use it, and examples on the Prophet website as well as other, third-party sites, such as GitHub and Kaggle. Once the data is cleaned and set up in the proper schema, the actual package is extremely easy to use and can run in 4 lines of code.

The underlying algorithm is a generalized additive model that is decomposable into three main components: trend, seasonality, and holidays. As I mentioned above, seasonality and trend are two important, but difficult to quantify, components of a time series analysis and FB Prophet does a great job capturing both.

Because it is a decomposable model, it is relatively easy to extract the coefficients of the model to understand the impact of seasonality, trend, holidays, and other regressor variables. For example, if a business team is trying to forecast sales, they can extract the price coefficient to see how impactful price is to forecast sales. Decomposition helps the teams understand the drivers of the business. It also helps to identify reasons why a forecast is off. For example, if there is a last-minute price increase that was not accounted for during forecasting, it can be identified relatively quickly when evaluating the model vs. actual performance.

One caveat about FB Prophet is that it is great for stationary data. Stationary data is time series data that follow similar behavior and have the same statistical properties throughout time.

- First we will import required libraries

```
import pandas as pd
import numpy as np
from fbprophet import Prophet
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
plt.rcParams['figure.figsize']=(20,10)
plt.style.use('ggplot')
pd.plotting.register_matplotlib_converters()
```

- Lets load the Dataset too and rename the column names to ds for datetime and y for frequency so that prophet can identify well.

```
[ ] df=pd.read_csv("/content/drive/MyDrive/cleanedData.csv")
```

```
[ ] df.head()
```

	date	T
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

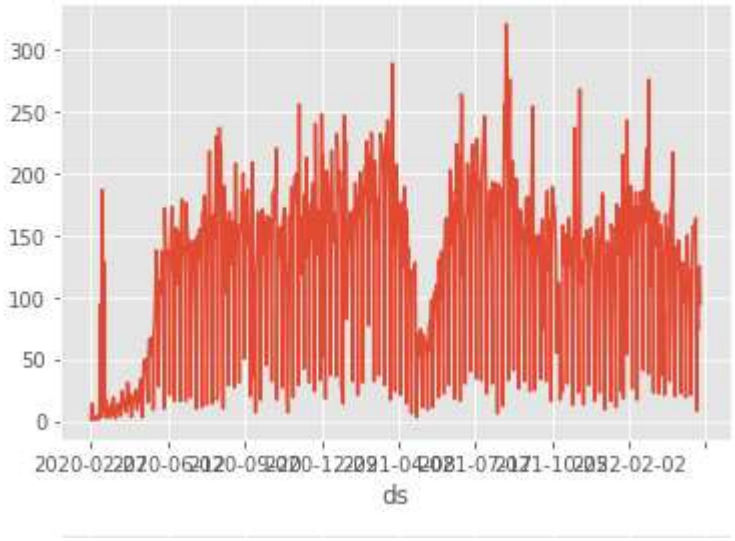
```
[ ]  df=df.rename(columns={'date':'ds', 'T':'y'})
```

```
df.head()
```

	ds	y
0	2020-02-27	1
1	2020-02-28	14
2	2020-02-29	4
3	2020-03-02	1
4	2020-03-03	2

- Also lets take a look at the dataset plotting

```
[ ]  plt.figure()
     df.set_index('ds').y.plot().get_figure()
```



- Now conversion of the dataset into trained and test data

```
[ ]  train_df=df[df["ds"]<'2021-12-15']
     test_df=df[df['ds']>='2021-12-15']
     test_df.shape
```

```
(143, 2)
```

The forecasting output looks something like the image below. It contains information for:

- The forecasted value (yhat)
- Range for the forecasted values (yhat_lower and yhat_upper)
- The overall trend for a given date (also incorporates seasonality)
- Additive terms to adjust the trend to get the forecasted value

```
[ ]  model = Prophet(daily_seasonality=True)
     model.fit(df);
     model.daily_seasonality
```

```
True
```

```
future = model.make_future_dataframe(periods=143, freq = 'D')
future.tail()
```

	ds
932	2022-09-22
933	2022-09-23
934	2022-09-24
935	2022-09-25
936	2022-09-26

```
[ ]  forecast=model.predict(future)
```

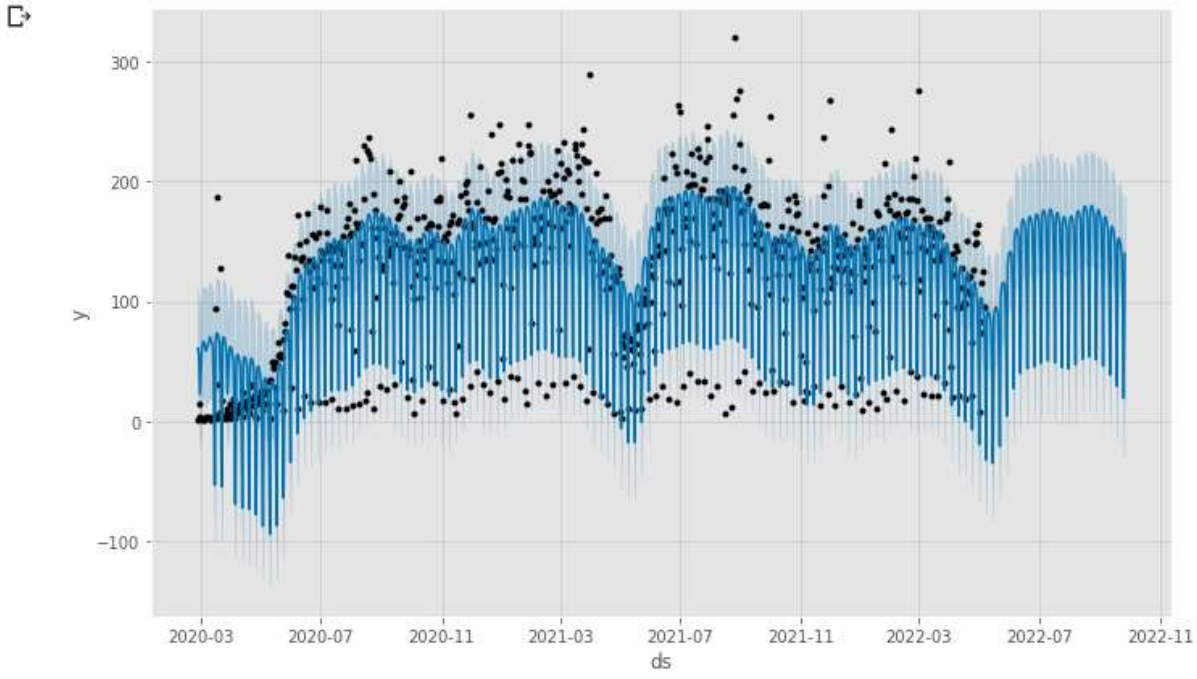
```
[ ]  forecast.head()
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower	additive_terms_upper	daily	...	weekly
0	2020-02-27	6.475570	16.039494	110.038616	6.475570	6.475570	54.341786	54.341786	54.341786	21.98348	...	23.019176
1	2020-02-28	6.994907	10.686622	102.100678	6.994907	6.994907	48.136317	48.136317	48.136317	21.98348	...	17.138321
2	2020-02-29	7.514244	-23.084572	67.067997	7.514244	7.514244	15.171661	15.171661	15.171661	21.98348	...	-15.568403
3	2020-03-02	8.552918	18.152706	111.129213	8.552918	8.552918	52.558237	52.558237	52.558237	21.98348	...	22.121135
4	2020-03-03	9.072255	21.016931	112.232033	9.072255	9.072255	56.980075	56.980075	56.980075	21.98348	...	26.588817

5 rows × 22 columns

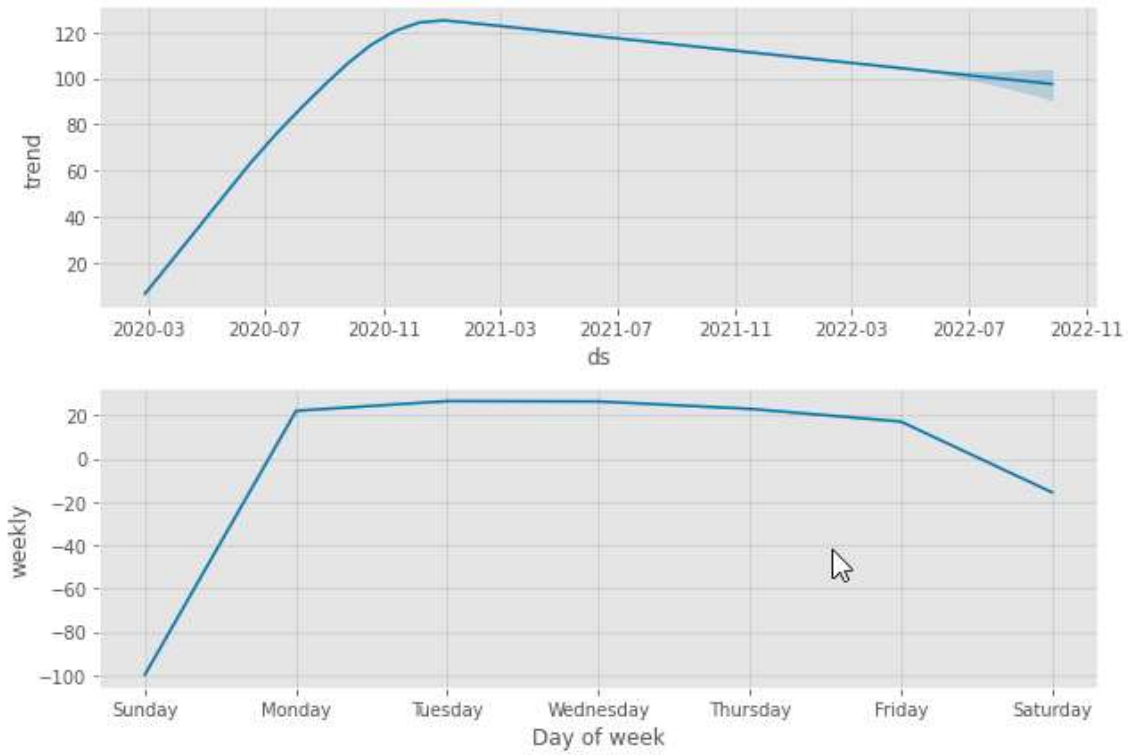
- The next step would be to visualize the training set to see the fit of the model. The blue line is the forecasted values while the black dots are the actual values. We can see that the blue line fits the overall trend of the data with some outliers still remaining.

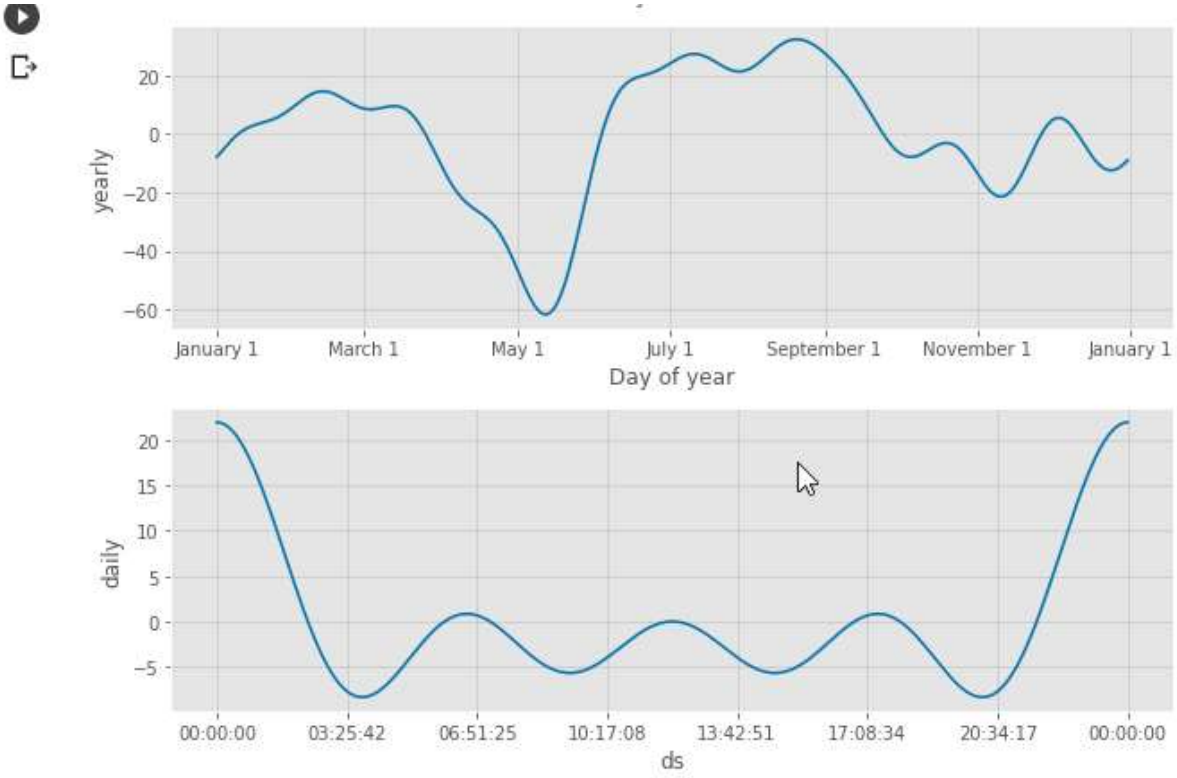
```
plot1=model.plot(forecast)
```



- Now lets see the result of the components

```
plot2=model.plot_components(forecast)
```





- When forecasting, we need to use 1 – weighted MAPE as an error metric to determine the fit of the model. I take the absolute error of the actual – predicted values at a daily level the aggregate all the errors up and divide by the total sales.

```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head()
```

	ds	yhat	yhat_lower	yhat_upper
0	2020-02-27	60.817356	16.039494	110.038616
1	2020-02-28	55.131224	10.686622	102.100678
2	2020-02-29	22.685905	-23.084572	67.067997
3	2020-03-02	61.111155	18.152706	111.129213
4	2020-03-03	66.052331	21.016931	112.232033

```
forecast_sub=forecast[["ds","yhat"]]
forecast_sub["ds"]=forecast_sub["ds"].astype(str)
test_df=test_df[['ds','y']]
metrics_df=test_df.merge(forecast_sub,on=['ds'],how='left')
metrics_df['abserror']=abs(metrics_df['y']-metrics_df['yhat'])
metrics_df['daily_df']=1-(metrics_df['abserror']/metrics_df['y'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
metrics_df.head()
```

	ds	y	yhat	abserror	daily_df
0	2021-12-15	156	153.048498	2.951502	0.981080
1	2021-12-16	144	148.297129	4.297129	0.970159
2	2021-12-17	128	141.172106	13.172106	0.897093
3	2021-12-18	114	107.327771	6.672229	0.941472
4	2021-12-19	16	22.143026	6.143026	0.616061

```
[ ] total_y=sum(metrics_df['y'])
    total_error=sum(metrics_df['abserror'])
    forecast_acc=1-(total_error/total_y)
    print(forecast_acc)
```

```
0.8148767666278307
```

With this we can conclude that the accuracy is nearly 81% for prophet model which is 3% more than ARIMA.

3. PyCaret Library

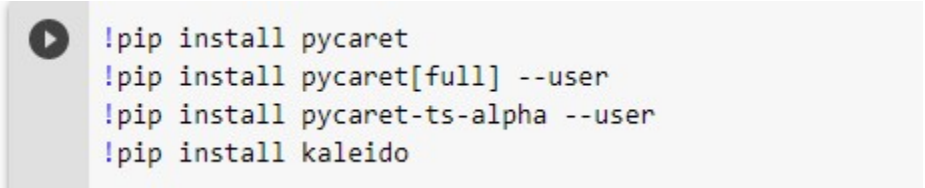
PyCaret is an open-source machine learning package written in low-code that enables Data Scientists to automate their machine learning processes. It reduces the model experimentation process, allowing for the achievement of specific outcomes with less code.

As more businesses shifted their focus to Machine Learning to address challenging issues, data scientists were expected to give results faster. This has increased the demand for automating important phases in data science projects so that data scientists may focus on the real problem at hand rather than writing hundreds of lines of code to identify the optimal model.

The time series forecasting module in PyCaret (pycaret.time_series.setup) is a machine learning module that is used to handle time series analysis problems. With PyCaret, a data scientist/user can do forecasting with several models, namely:

- Seasonal Naive Forecaster,
- Exponential Smoothing,
- ARIMA,
- Polynomial Trend Forecaster,
- K Neighbors w/ Cond. Deseasonalize & Detrending,
- Linear w/ Cond. Deseasonalize & Detrending,
- Elastic Net w/ Cond. Deseasonalize & Detrending,
- Ride w/ Cond. Deseasonalize & Detrending,
- Lasso Net w/ Cond. Deseasonalize & Detrending,
- Extreme Gradient Boosting w/ Cond. Deseasonalize & Detrending, and more.

- Installing PyCaret and importing libraries that will be used in this notebook.



```
!pip install pycaret
!pip install pycaret[full] --user
!pip install pycaret-ts-alpha --user
!pip install kaleido
```

```
[ ]  pip install markupsafe==2.0.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: markupsafe==2.0.1 in /usr/local/lib/python3.7/dist-packages (2.0.1)
```

```
[ ]
import jinja2
```

```
import datetime
import numpy as np
import pandas as pd
import warnings
import pycaret
import kaleido
import plotly.express as px

from pycaret.time_series import *
from pycaret.utils import enable_colab

# --- Libraries Settings ---
warnings.filterwarnings('ignore')
enable_colab()
```

```
/usr/local/lib/python3.7/dist-packages/distributed/config.py:20: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader is
  defaults = yaml.load(f)
Colab mode enabled.
```

- After importing libraries, the dataset that will be used will be imported.

```
df=pd.read_csv("/content/drive/MyDrive/cleanedData.csv")
df.reset_index()
df.head()
```

	Unnamed: 0	date	T
0	0	2020-02-27	1.0
1	1	2020-02-28	14.0
2	2	2020-02-29	4.0
3	3	2020-03-01	0.0
4	4	2020-03-02	1.0

- It can be seen that the dataset is successfully imported.
- Also, there is no null values in the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  800 non-null    int64
 1   date        800 non-null    datetime64[ns]
 2   T           800 non-null    float64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 18.9 KB
```

- A simple data pre-processing will be performed to prepare the dataset before implementing PyCaret time series module.
- Since the date column is detected as object, this section will change it into date time and set it as index.

```
[ ]  df.set_index('date', inplace=True, drop=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 800 entries, 2020-02-27 to 2022-05-06
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  800 non-null    int64
 1   T           800 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 18.8 KB
```

```
[ ]  df.drop(columns="Unnamed: 0",inplace=True)
```

- This section will implement PyCaret by calling TimeSeriesExperiment() function.
- For experiment purposes, the number of folds that used in cross validation will be set to 3 and the forecast horizon used will be set to 80 (last 80 points in the dataset will be set as test).

```
[ ]
    df=df.asfreq('D')
```

```
[ ] from pycaret.time_series import *
```

```
s = setup(df,fh = 80, fold = 3, session_id = 123)
```

	Description	Value
0	session_id	123
1	Target	T
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Data shape	(800, 1)
5	Train data shape	(720, 1)
6	Test data shape	(80, 1)
7	Fold Generator	ExpandingWindowSplitter
8	Fold Number	3
9	Enforce Prediction Interval	False
10	Seasonal Period(s) Tested	7

11	Seasonality Present	True
12	Seasonalities Detected	[7]
13	Primary Seasonality	7
14	Target Strictly Positive	False
15	Target White Noise	No
16	Recommended d	1
17	Recommended Seasonal D	0
18	Missing Values	0
19	Preprocess	True
20	CPU Jobs	-1
21	Use GPU	False
22	Log Experiment	False
23	Experiment Name	ts-default-name
24	USI	727b

- With check_stats(), statistical tests and results will be shown below.

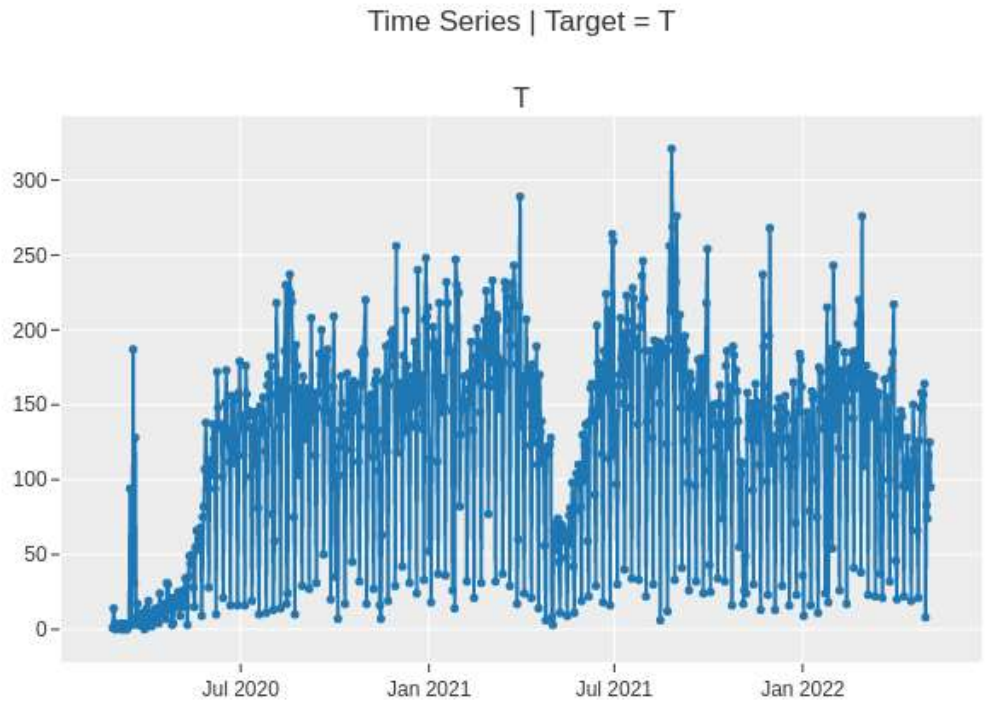
```
[ ]  check_stats()
```

	Test	Test Name	Data	Property	Setting	Value
0	Summary	Statistics	Actual	Length		800.0
1	Summary	Statistics	Actual	Mean		121.40625
2	Summary	Statistics	Actual	Median		138.0
3	Summary	Statistics	Actual	Standard Deviation		68.955219
4	Summary	Statistics	Actual	Variance		4754.822239
5	Summary	Statistics	Actual	Kurtosis		-0.870998
6	Summary	Statistics	Actual	Skewness		-0.292342
7	Summary	Statistics	Actual	# Distinct Values		223.0
8	White Noise	Ljung-Box	Actual	Test Statictic	{'alpha': 0.05, 'K': 24}	2602.040627
9	White Noise	Ljung-Box	Actual	Test Statictic	{'alpha': 0.05, 'K': 48}	3975.766819
10	White Noise	Ljung-Box	Actual	p-value	{'alpha': 0.05, 'K': 24}	0.0
11	White Noise	Ljung-Box	Actual	p-value	{'alpha': 0.05, 'K': 48}	0.0
12	White Noise	Ljung-Box	Actual	White Noise	{'alpha': 0.05, 'K': 24}	False

	13	White Noise	Ljung-Box	Actual	White Noise	{'alpha': 0.05, 'K': 48}	False
	14	Stationarity	ADF	Actual	Stationarity	{'alpha': 0.05}	False
	15	Stationarity	ADF	Actual	p-value	{'alpha': 0.05}	0.118263
	16	Stationarity	ADF	Actual	Test Statistic	{'alpha': 0.05}	-2.488639
	17	Stationarity	ADF	Actual	Critical Value 1%	{'alpha': 0.05}	-3.438783
	18	Stationarity	ADF	Actual	Critical Value 5%	{'alpha': 0.05}	-2.865262
	19	Stationarity	ADF	Actual	Critical Value 10%	{'alpha': 0.05}	-2.568752
	20	Stationarity	KPSS	Actual	Trend Stationarity	{'alpha': 0.05}	False
	21	Stationarity	KPSS	Actual	p-value	{'alpha': 0.05}	0.01
	22	Stationarity	KPSS	Actual	Test Statistic	{'alpha': 0.05}	0.651805
	23	Stationarity	KPSS	Actual	Critical Value 10%	{'alpha': 0.05}	0.119
	24	Stationarity	KPSS	Actual	Critical Value 5%	{'alpha': 0.05}	0.146
	25	Stationarity	KPSS	Actual	Critical Value 2.5%	{'alpha': 0.05}	0.176
	26	Stationarity	KPSS	Actual	Critical Value 1%	{'alpha': 0.05}	0.216
	27	Normality	Shapiro	Actual	Normality	{'alpha': 0.05}	False
	28	Normality	Shapiro	Actual	p-value	{'alpha': 0.05}	0.0

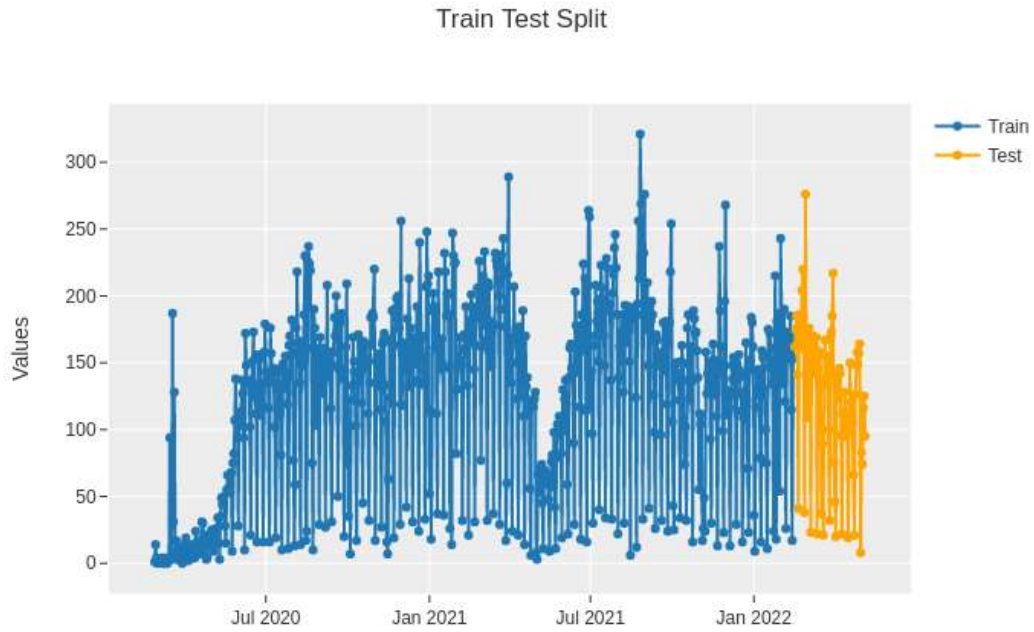
- From the statistical test results above, it can be concluded that:
  - The series is not stationary (ADF p-value more than 0.05)
  - The series is not adequate (Ljung-Box p-value less than 0.05)

```
plot_model(plot = 'ts', fig_kwargs = {'hoverinfo': 'none', 'big_data_threshold': 15})
```



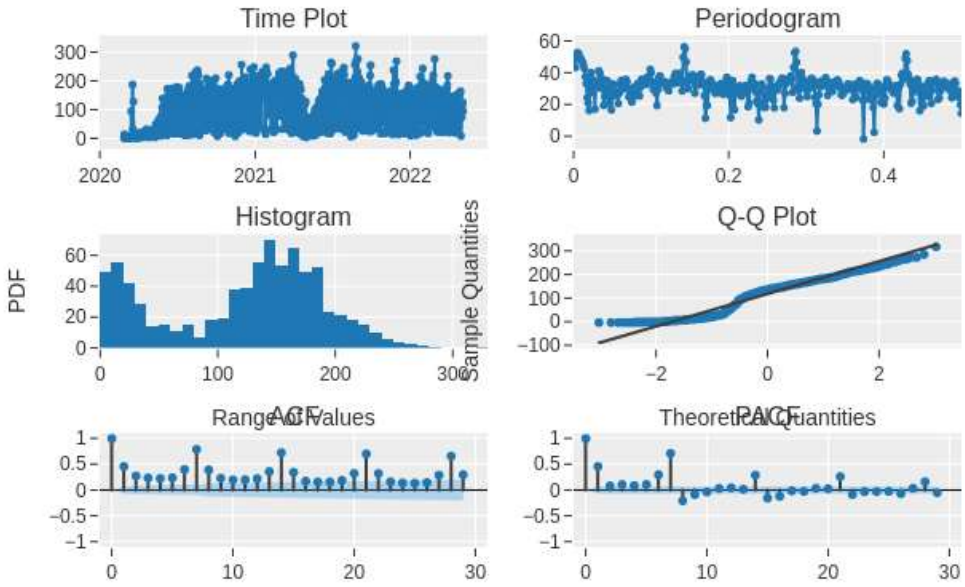
- It can be concluded that there are additive seasonal variability with additive trend.
- The next graph will show the train and test plot.

```
plot_model(plot = 'train_test_split', fig_kwargs = {'hoverinfo': 'none', 'big_data_threshold': 15})
```



- This section will show the diagnostics plot. In this plot, the time series plot, periodogram, ACF and PACF plot can be seen in one picture.

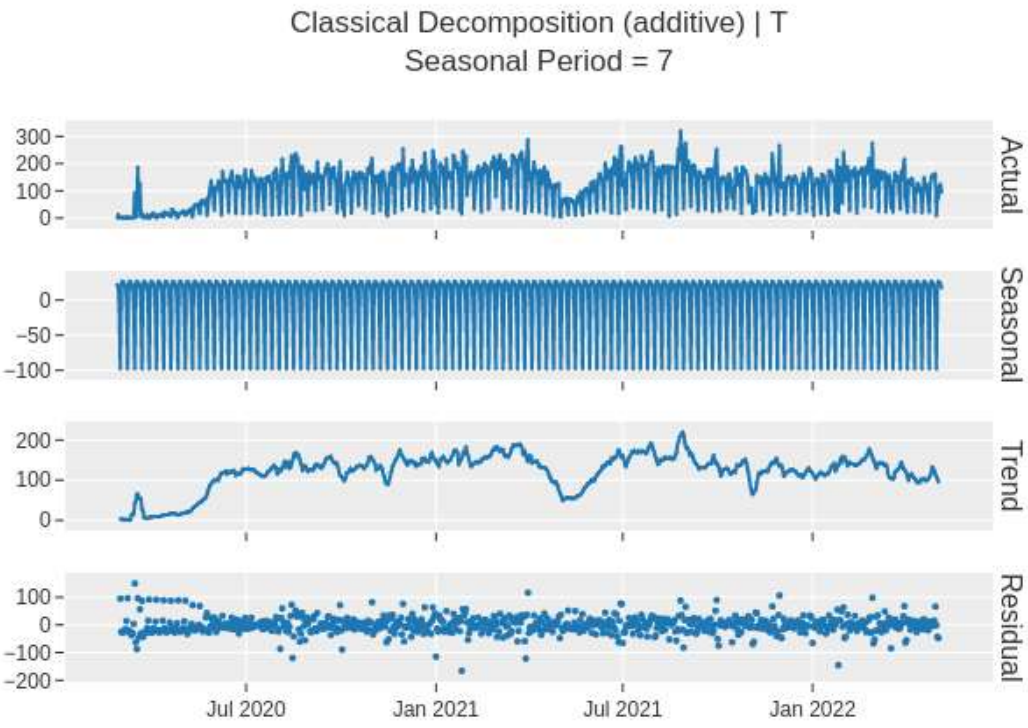
```
plot_model(plot = 'diagnostics', fig_kwargs = {'hoverinfo': 'none', 'big_data_threshold': 15})
```



- This section will show the decomposition in additive, decomposition in multiplicative, and STL (seasonal trend decomposition using LOESS) decomposition plot.

```
plot_model(plot = 'decomp', fig_kwargs = {'hoverinfo': 'none','big_data_threshold': 15})
plot_model(plot = 'decomp', data_kwargs={'type': 'multiplicative'},
           fig_kwargs = {'hoverinfo': 'none','big_data_threshold': 15})
plot_model(plot = 'decomp_stl', fig_kwargs = {'hoverinfo': 'none','big_data_threshold': 15})
```

[ ]



- With models() function, all time series models that available in PyCaret will be listed below.

```
models()
```

ID	Name	Reference	Turbo
naive	Naive Forecaster	sktime.forecasting.naive.NaiveForecaster	True
grand_means	Grand Means Forecaster	sktime.forecasting.naive.NaiveForecaster	True
snaive	Seasonal Naive Forecaster	sktime.forecasting.naive.NaiveForecaster	True
polytrend	Polynomial Trend Forecaster	sktime.forecasting.trend.PolynomialTrendForeca...	True
arima	ARIMA	sktime.forecasting.arima.ARIMA	True
auto_arima	Auto ARIMA	sktime.forecasting.arima.AutoARIMA	True
exp_smooth	Exponential Smoothing	sktime.forecasting.exp_smoothing.ExponentialSm...	True
croston	Croston	sktime.forecasting.croston.Croston	True
ets	ETS	sktime.forecasting.ets.AutoETS	True
theta	Theta Forecaster	sktime.forecasting.theta.ThetaForecaster	True
tbats	TBATS	sktime.forecasting.tbats.TBATS	False
bats	BATS	sktime.forecasting.bats.BATS	False
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
lar_cds_dt	Least Angular Regressor w/ Cond. Deseasonalize...	pycaret.containers.models.time_series.BaseCdsD...	True
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseaso...	pycaret.containers.models.time_series.BaseCdsD...	True
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detren...	pycaret.containers.models.time_series.BaseCdsD...	True
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
par_cds_dt	Passive Aggressive w/ Cond. Deseasonalize & De...	pycaret.containers.models.time_series.BaseCdsD...	True
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasona...	pycaret.containers.models.time_series.BaseCdsD...	True
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD	True
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Det...	pycaret.containers.models.time_series.BaseCdsD...	True
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	pycaret.containers.models.time_series.BaseCdsD...	True
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonali...	pycaret.containers.models.time_series.BaseCdsD...	True
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize...	pycaret.containers.models.time_series.BaseCdsD...	True
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & De...	pycaret.containers.models.time_series.BaseCdsD...	True

- The next section will compare which models has better results in forecasting

```
best = compare_models()
```

	Model	MAE	RMSE	MAPE	SMAPE	MASE	RMSSE	R2	TT (Sec)
snaive	Seasonal Naive Forecaster	36.2583	48.5518	0.4167	0.2904	1.2305	1.136	0.3085	0.0333
auto_arima	Auto ARIMA	33.0143	42.8777	0.5678	0.2908	1.1228	1.0047	0.4582	122.8400
arima	ARIMA	39.2938	51.4556	0.4904	0.2968	1.3315	1.203	0.1655	0.4267
ets	ETS	40.4523	50.4837	0.5817	0.3249	1.374	1.1827	0.1879	0.2900
exp_smooth	Exponential Smoothing	40.8564	50.8343	0.6006	0.3301	1.388	1.1909	0.1823	0.9300
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	39.4686	48.6492	0.6662	0.3428	1.3388	1.1386	0.2775	0.2767
lar_cds_dt	Least Angular Regressor w/ Cond. Deseasonalize...	43.1056	51.7342	0.7516	0.3669	1.4603	1.2099	0.1537	0.2233
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	43.1056	51.7342	0.7516	0.3669	1.4603	1.2099	0.1537	0.2500
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	43.1056	51.7342	0.7516	0.3669	1.4603	1.2099	0.1537	0.2233
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	43.1161	51.7424	0.7518	0.3669	1.4606	1.2101	0.1534	0.2500
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	43.1243	51.7491	0.752	0.367	1.4609	1.2103	0.1532	0.2300
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detren...	43.2894	51.872	0.756	0.3683	1.4664	1.2131	0.1488	0.2400
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasona...	45.1079	53.434	0.7837	0.3776	1.5264	1.2489	0.0868	0.1300
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	50.3004	59.8414	0.8422	0.3861	1.7126	1.4039	-0.2517	5.5667
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	49.5976	57.9444	0.8487	0.388	1.6887	1.3592	-0.1927	5.9333
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	49.314	58.0552	0.8454	0.3884	1.6791	1.3615	-0.1744	5.7633
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseaso...	46.6934	54.4451	0.8317	0.3935	1.5796	1.2723	0.0546	0.2300
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	50.8093	58.8558	0.8694	0.3969	1.7295	1.3805	-0.2403	0.4600
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & De...	50.9261	59.1961	0.8877	0.3992	1.7335	1.3883	-0.2266	3.3400
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	56.3806	69.4594	0.818	0.4122	1.9119	1.6247	-0.4854	0.1400
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize...	53.0029	61.8022	0.9211	0.4182	1.7969	1.4454	-0.2008	0.5067
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Det...	54.0836	61.849	0.9418	0.4228	1.8371	1.449	-0.3185	0.2600
polytrend	Polynomial Trend Forecaster	54.9612	73.7396	1.4477	0.4311	1.8655	1.7254	-0.5839	0.0500
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonali...	60.6143	70.3257	1.0526	0.4545	2.054	1.6446	-0.6026	0.9867
theta	Theta Forecaster	57.6233	72.1112	1.2237	0.4688	1.967	1.6913	-0.4725	0.1100
croston	Croston	57.7569	71.8298	1.1889	0.4751	1.9725	1.685	-0.4466	0.0333
grand_means	Grand Means Forecaster	57.2027	67.1544	1.0047	0.4933	1.9543	1.5757	-0.2438	0.0267
naive	Naive Forecaster	92.8833	105.4952	1.1549	0.8567	3.185	2.4826	-2.2407	1.8833
par_cds_dt	Passive Aggressive w/ Cond. Deseasonalize & De...	497.0458	827.5211	5.9658	1.0731	17.1821	19.5863	-264.755	0.2267

```
exps_tune = tune_model(best)
print(best)
print(exps_tune)
```

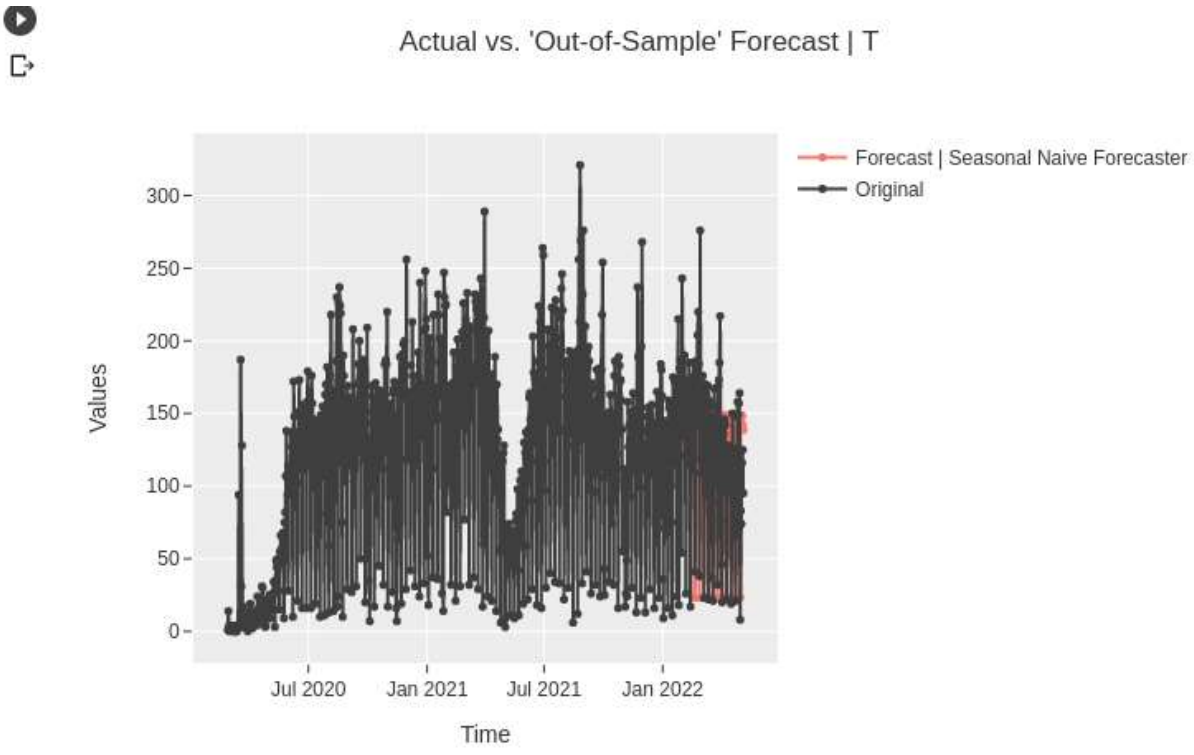
	cutoff	MAE	RMSE	MAPE	SMAPE	MASE	RMSSE	R2
0	2021-06-20	56.8163	67.5633	0.3526	0.3967	1.9841	1.6070	-0.0022
1	2021-09-08	24.3095	35.4433	0.3389	0.2249	0.8246	0.8319	0.5928
2	2021-11-27	23.1306	34.0914	0.3134	0.2197	0.7649	0.7829	0.6429
Mean	NaT	34.7521	45.6994	0.3349	0.2804	1.1912	1.0739	0.4112
SD	NaT	15.6092	15.4700	0.0162	0.0822	0.5612	0.3775	0.2930

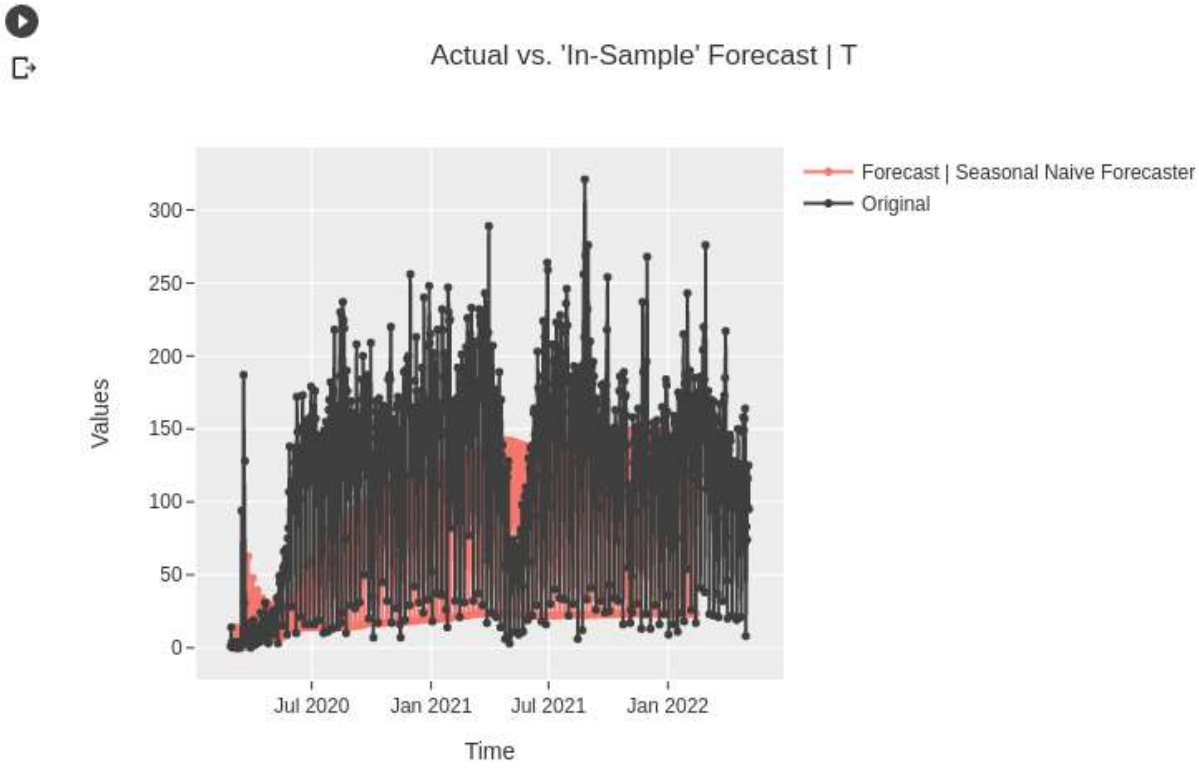
```
NaiveForecaster(sp=7)
NaiveForecaster(sp=7, strategy='mean')
```

```
predict_es = predict_model(exps_tune)
plot_model(estimator = exps_tune, fig_kwargs = {'hoverinfo': 'none','big_data_threshold': 15})
plot_model(exps_tune, plot = 'insample', fig_kwargs = {'hoverinfo': 'none','big_data_threshold': 15})
```

	Model	MAE	RMSE	MAPE	SMAPE	MASE	RMSSE	R2
0	Seasonal Naive Forecaster	27.1533	37.3963	0.2779	0.2358	0.8961	0.8635	0.5367

- Based on the all the models in the library the Seasonal naïve forecaster has good R2 score compared to others. Well that doesn't mean it has good accuracy because for good accuracy the r2 score should be in range of 0.9-1.


Actual vs. 'Out-of-Sample' Forecast | T
Forecast | Seasonal Naive Forecaster
Original
Values
300
250
200
150
100
50
0
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Time


Actual vs. 'In-Sample' Forecast | T
Forecast | Seasonal Naive Forecaster
Original
Values
300
250
200
150
100
50
0
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Time

6. Conclusion

Out of all the models that I have done prophet model has got the highest accuracy of 81% compared to ARIMA and pycaret library models. I have also done TBATS, Darts and auto time series libraries on this dataset but the data was not able to fit in TBATS. In the auto_time series library the package was not able get loader() installed in it. And finally in the darts library the prophet in it was not able to converge data in it so it started doing a re-run in an infinite loop. In the end I was able to successfully work on Facebook prophet model, ARIMA and PyCaret on the given 2 years dataset with accuracies 81%, 78%, 65% respectively.