# Facebook Prophet

A

Seminar

submitted

in partial fulfillment

for the award of the Degree of

**Bachelor of Technology**

**in Department of Computer Science And Engineering**


**Submitted To:**

Santosh Gupta

Assistant Professor

**Submitted By:**

Vaibhav Saran

18EJICS169

**Department of Computer Science And Engineering**

**Jodhpur Institute Of Engineering And Technology**

**Bikaner Technical University**

**10 2021**

# Candidates Declaration

I hereby declare that the work, which is being presented in the Seminar, entitled **"Facebook Prophet"** in partial fulfillment for the award of Degree of "Bachelor of Technology" in Computer science and Engineering **and submitted to the Department of Computer Science and Engineering,** Jodhpur Institute Of Engineering And Technology, BikanerTechnical University is a record of my own WORK carried under the Guidance of Mr. Santosh Gupta, Department of Computer Science and Engineering.

Vaibhav Saran
Computer Science and Engineering
Enrolment No.: 18EJICS169

**Counter Signed by**

Ms. Arshi Riyaz
Assistant Professor(Sr. Scale)

# ACKNOWLEDGEMENT

I am grateful to **Ms. Arshi Riyaz** AssociateProfessor (JIET) for giving me this opportunity to work under the guidance of renowned people in the field of Time Series Forecasting and FB Prophet and providing the necessary guidance and perspective to research in the field of forecasting.

**Vaibhav Saran**

**18EJICS169**

**Computer Science And Engineering**

# ABSTRACT

Facebook Prophet focuses on taking a time series data which comprises dates and some value associated with them, it can be sales, price of a product, stock values, etc., and then generate future predictions based on the previous data to tell where the value can go in the future. Time series forecasting can be performed using machine learning, deep learning, using custom ARIMA(AutoRegressive Integrated Moving Average) or Facebook prophet, an open-source package developed by Facebook.

**Chapter 1: Introduction to Facebook Prophet**

**Chapter 2: How Prophet works**

**Chapter 3:Understanding STAN**

**Chapter 4:Example of Forecasting with Facebook Prophet**

# Table Of Content

# Introduction

The purpose of the Seminar is to investigate the features and the working of Facebook Prophet. Time series forecasting requires a lot of mathematical computations to be performed as well as traditional LSTM and RNN consume a lot of computing resources, also are really difficult to design. Facebook launched prophet in February 2017 to make this process a lot easier as well as accommodate features like seasonality, holiday effects, etc. which makes the forecasting process a lot more accurate. It also provides options to plot the forecast which helps in better understanding of the forecast with probabilistic regions that indicate the minima and maxima of a forecast.

# Literature Survey

The term forecasting is often mistaken as an accurate prediction of future events but in reality that is not the case. While forecasting what we are doing is performing some mathematical calculations and predicting the possibility of occurrence of an event within a certain margin of change. Now while performing these calculations events like holidays or seasons tend to influence the output and sometimes these influences can drastically affect the output.

The automatic ARIMA(AutoRegressive Integrated Moving Average) forecasts are prone to large trend errors when there is a change in trend near the cutoff period and they fail to capture any seasonality

When a forecast is poor, we wish to be able to tune the parameters of the method to the problem at hand. Tuning these methods requires a thorough understanding of how the underlying time series models work. The first input parameters to automated ARIMA, for instance, are the maximum orders of the differencing, the auto-regressive components, and the moving average components. A typical analyst will not know how to adjust these orders to avoid the behavior, this is the type of expertise that is hard to scale. ARIMA models are capable of including seasonal covariates, but adding these covariates leads to extremely long fitting times and requires modeling expertise that many forecasting novices would not have

At this point, Facebook Prophet proves to be of great use as it incorporates these changes of seasons and holidays and comes very close to the actual possibility of a future event happening.

**Goal:** The Prophet library is an open-source library designed for making forecasts for univariate time-series datasets. It is easy to use and designed to automatically find a good set of hyperparameters for the model in an effort to make skillful forecasts for data with trends and seasonal structure by default.

The prophet aims to:

- ★ Make a forecast for sales/stocks or any other time-series data
- ★ Minimal user input while making predictions
- ★ Incorporate external factors like seasonality and holidays
- ★ Enable automation for forecasting
- ★ Allow tuning the forecasts as needed

# 1. Facebook Prophet

FB 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. It is open-source software released by Facebook's Core Data Science team. It is used in many applications across Facebook for producing reliable forecasts for planning and goal setting. It is found that prophet performs better than any other approach in the majority of cases. The models are fitted in Stan so that forecasts can be made in just a few seconds. It provides a reasonable forecast on messy data with no manual effort. The prophet is robust to outliers, missing data, and dramatic changes in time series. The Prophet procedure includes many possibilities for users to tweak and adjust forecasts. We can use human-interpretable parameters to improve forecasts by adding our domain knowledge. According to Sean Taylor, a research scientist at Facebook and Stan user told that Prophet uses Stan for optimization (and sampling if the user desires) in order to fit a non-linear additive model and generate uncertainty intervals. The advantages they got by using STAN was:

1) STAN does a great job of letting a user separate optimization from the model code.
2) It could share the same core implementation in both python and R.
3) FB Prophet is automatically detecting changepoints in the time series by specifying a sequence of potential parameter changes and shrinking the shifts using a Laplace prior. The user is allowed to adjust the flexibility of the model by tuning the precision of priors, which is intuitive for most users. At its core, the Prophet procedure is an additive regression model with four main components:
    a) A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
    b) A yearly seasonal component modeled using Fourier series.
    c) A weekly seasonal component using dummy variables.
    d) A user-provided list of important holidays. In layman terms, we can understand it as follows: FBProphet uses a decomposable time series model with 3 main components: seasonal, trends, holidays or events effect and error which are

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

    combined into this equation:

# 2. How Prophet Works

## 2.1 Understanding Maths Of FB Prophet

At its core is the sum of three functions of time plus an error term: growth g(t), seasonality s(t), holidays h(t) , and error e_t :

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

### 2.1.1 The Growth Function (and change points):

The growth function models the overall trend of the data. The old ideas should be familiar to anyone with a basic knowledge of linear and logistic functions. The new idea incorporated into Facebook prophet is that the growth trend can be present at all points in the data or can be altered at what Prophet calls “changepoints”.

Changepoints are moments in the data where the data shifts direction. Using new COVID-19 cases as an example, it could be due to new cases beginning to fall after hitting a peak once a vaccine is introduced. Or it could be a sudden pick up of cases when a new strain is introduced into the population and so on. Prophet can automatically detect change points or you can set them yourself. You can also adjust the power the change points have in altering the growth function and the amount of data taken into account in automatic changepoint detection.

The growth function has are three main options:

- Linear Growth: This is the default setting for Prophet. It uses a set of piecewise linear equations with differing slopes between change points. When linear growth is used, the growth term will look similar to the classic y = mx + b from middle school, except the slope(m) and offset(b) are variable and will change the value at each changepoint.

- Logistic Growth: This setting is useful when your time series has a gap or a floor in which the values you are modeling become saturated and can’t surpass a maximum or minimum value. When logistic growth is used, the growth term will look similar to a typical equation for a logistic curve, except it, the carrying capacity (C) will vary as a function of time and the growth rate (k) and the offset(m) are variable and will change the value at each change point.

$$g(t) = \frac{C(t)}{1 + x^{-k(t-m)}}$$

- ➢ Flat: Lastly, you can choose a flat trend when there is no growth over time (but there still may be seasonality). If set to flat the growth function will be a constant value.

### 2.1.2 The Seasonality Function:

The seasonality function is simply a Fourier Series as a function of time. An easy way to think about it is the sum of many successive sines and cosines. Each sine and cosine term is multiplied by some coefficient. This sum can approximate nearly any curve or in the case of Facebook Prophet, the seasonality (cyclical pattern) in our data. All together it looks like this:

$$s(t) = \sum_{n=1}^{N} (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P}))$$

Prophet will automatically detect an optimal number of terms in the series, also known as the Fourier order. You can also choose the Fourier order based on the needs of your particular data set. The higher the order the more terms in the series. You can also choose between additive and multiplicative seasonality.

### 2.1.3 The Holiday/Event Function:

The holiday function allows Facebook Prophet to adjust forecasting when a holiday or major event may change the forecast. It takes a list of dates (there are built-in dates of US holidays or you can define your own dates) and when each date is present in the forecast adds or subtracts value from the forecast from the growth and seasonality terms based on historical data on the identified holiday dates. You can also identify a range of days around dates (think the time between Christmas/New Year, holiday weekends, thanksgiving's association with Black Friday/Cyber Monday, etc).

## 2.2 Performance Metrics

Two performance metrics are used for measuring forecasting accuracy, calculating the expected level of forecasting accuracy and classifying product portfolio accordingly:

- ➢ the relative or percentage error (PE) for individual monthly/quarterly forecasts, and
- ➢ the mean absolute percentage error (MAPE) for quantifying the overall accuracy.

The percentage error (PE), which can be calculated as:

$$PE = \frac{y_{forecast} - y_{true}}{y_{true}} \cdot 100\%$$

is mainly used to measure the accuracy of individual monthly/quarterly forecasting outputs generated by the model.

While the mean absolute percentage error (MAPE), calculated as:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y^{i}_{forecast} - y^{i}_{true}}{y^{i}_{true}}\right| \cdot 100\%$$

is used to quantify the overall accuracy of the forecasting framework and calculate the expected level of reliability useful for classifying the product portfolio.

These metrics are selected for use because of their simplicity, very intuitive interpretation, as well as the fact they work well if there are no extremes in the data. It is concluded that a clear and intuitive interpretation of forecasting accuracy metrics in terms of relative error, plays a crucial role in the acceptance of the sales forecasting framework as a decision-making tool.

# 3. Understanding STAN

Stan, named after Stanislaw Ulam, a mathematician who was one of the developers of the Monte Carlo method in the 1940s (Metropolis & Ulam, 1949), is a C++ program to perform Bayesian inference. The code is open-source and is available at http://mc-stan.org/ along with instructions and a 500-page user manual. Stan 1.0 was released in 2012 and, as of this writing, is currently in version 2.6. To use Stan, a user writes a Stan program that directly computes the log-posterior density. This code is then compiled and run along with data. The result is a set of posterior simulations of the parameters in the model (or a point estimate, if Stan is set to optimize). Stan can be run from the command line, R, Python, Matlab, or Julia, and its output can be saved, printed, or graphed in R using shinytan. Stan can be thought of as similar to BUGS (Lunn, Thomas, Best, & Spiegelhalter, 2000) and Jags (Plummer, 2003) in that it allows a user to write a Bayesian model in a convenient language whose code looks like statistics notation. Or Stan can be thought of as an alternative to programming a sampler or optimizer oneself. Stan uses the no-U-turn sampler (Hoffman & Gelman, 2014), an adaptive variant of Hamiltonian Monte Carlo (Neal, 2011), which itself is a generalization of the familiar Metropolis algorithm, performing multiple steps per iteration to move more efficiently through the posterior distribution.

Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Thousands of users rely on Stan for statistical modeling, data analysis, and prediction in the social, biological, physical sciences, engineering, and businesses.

Users specify log density functions in Stan's probabilistic programming language and get:

- full Bayesian statistical inference with MCMC sampling (NUTS, HMC)
- approximate Bayesian inference with variational inference (ADVI)
- penalized maximum likelihood estimation with optimization (L-BFGS)

Stan's math library provides differentiable probability functions & linear algebra(C++ auto diff). Additional R packages provide expression-based linear modeling, poster visualization, and leave-one-out cross-validation.

Stan has several components:

• A flexible modeling language where the role of a Stan program is to compute a log posterior density. Stan programs are translated to templated C++ for efficient computation.

• An inference engine that performs Hamiltonian Monte Carlo using the no-U-turn sampler (Hoffman & Gelman, 2014) to get approximate simulations from the posterior distribution which is defined by the Stan program and by the data supplied to it from the calling program.

• In addition, an L-BFGS optimizer (Byrd, Lu, Nocedal, & Zhu, 1994) iterates to find a (local) maximum of the objective function (in Bayesian terms, a posterior mode) and can also be used as a stand-alone optimizer.

• The sampler and the optimizer both require gradients. Stan computes gradients using reverse-mode automatic differentiation (Griewank & Walther, 2008), a procedure in which Stan's 5 compiler takes the function defined by the called Stan program and analytically computes its derivative using an efficient procedure which can be much faster than numerical differentiation, especially when the number of parameters is large.

• Routines to monitor the convergence of parallel chains and compute inferences and effective sample sizes (Gelman et al., 2013).

• Wrappers to call Stan from R, Python, Stata, Matlab, Julia, and the command line, and to graph the results. Where possible these calling functions automatically run Stan in parallel; for example, on a laptop with four processors, Stan can be set up to run one chain on each.

Stan is open source, lives on GitHub, and is being developed by several groups of people. At the center is the core team, which currently numbers 17 people. The core team, or subsets of it, meets weekly via videoconference, directs Stan's research and development, and coordinates Stan's projects. The developer's group currently includes 46 people who exchange ideas on a listserv that is open to the public. Various features of Stan including a model for correlation matrices (Lewandowski, Kurowicka, & Joe, 2009), the higher default acceptance rate for adaptation, and vectorization of all the univariate distributions came from the developer's list. Stan also has a users group where people can post and answer questions. There are currently 1100 people on this list and it averages 15 messages per day.

## 4. Example Of Forecast With Facebook Prophet

For this Tesla Toy Data set will be used as a reference which is considered to be a beginner's dataset for introducing novice users to Facebook Prophet.

| Date | Store/Product | Value |
|---|---|---|
| 20180101 | LOS_ANGELES-TESLA_MODEL_X | 2926 |
| 20180102 | LOS_ANGELES-TESLA_MODEL_X | 2687.531 |
| 20180103 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180104 | LOS_ANGELES-TESLA_MODEL_X | 2394 |
| 20180105 | LOS_ANGELES-TESLA_MODEL_X | 2660 |
| 20180106 | LOS_ANGELES-TESLA_MODEL_X | 2527 |
| 20180107 | LOS_ANGELES-TESLA_MODEL_X | 2527 |
| 20180108 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180109 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180110 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180111 | LOS_ANGELES-TESLA_MODEL_X | 2926 |
| 20180112 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180113 | LOS_ANGELES-TESLA_MODEL_X | 2869.8208 |
| 20180114 | LOS_ANGELES-TESLA_MODEL_X | 3325 |
| 20180115 | LOS_ANGELES-TESLA_MODEL_X | 2261 |
| 20180116 | LOS_ANGELES-TESLA_MODEL_X | 2261 |
| 20180117 | LOS_ANGELES-TESLA_MODEL_X | 2253.685 |
| 20180118 | LOS_ANGELES-TESLA_MODEL_X | 2394 |
| 20180119 | LOS_ANGELES-TESLA_MODEL_X | 2394 |
| 20180120 | LOS_ANGELES-TESLA_MODEL_X | 2394 |
| 20180121 | LOS_ANGELES-TESLA_MODEL_X | 2527 |
| 20180122 | LOS_ANGELES-TESLA_MODEL_X | 2793 |
| 20180123 | LOS_ANGELES-TESLA_MODEL_X | 2660 |
| 20180124 | LOS_ANGELES-TESLA_MODEL_X | 2547.1362 |
| 20180125 | LOS_ANGELES-TESLA_MODEL_X | 2926 |
| 20180126 | LOS_ANGELES-TESLA_MODEL_X | 2926 |
| 20180127 | LOS_ANGELES-TESLA_MODEL_X | 3059 |

**Raw Dataset**

The dataset is loaded into a data frame as shown below:

```
tesla_df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/FACEBOOK PROPHET/Data/dataset.csv")
tesla_df.head()
```

| | Date | Store/Product | Value |
|---|---|---|---|
| 0 | 20180101 | LOS_ANGELES-TESLA_MODEL_X | 2926.000 |
| 1 | 20180102 | LOS_ANGELES-TESLA_MODEL_X | 2687.531 |
| 2 | 20180103 | LOS_ANGELES-TESLA_MODEL_X | 2793.000 |
| 3 | 20180104 | LOS_ANGELES-TESLA_MODEL_X | 2394.000 |
| 4 | 20180105 | LOS_ANGELES-TESLA_MODEL_X | 2660.000 |

Now the data is analyzed for any null values or missing values, etc. After the analysis we find that:

1) There are 3 features in the given data namely "Date", "Store/Product" and "Value".

2) There are 3240 data points in the data set.

3) The date has been provided in the form of integers but it needs to be changed into DateTime format as required by the prophet.

4) The Store/Product is a categorical feature that has categories namely:

SAN_FRANCISCO-TESLA_MODEL_S

LOS_ANGELES-TESLA_MODEL_X

LOS_ANGELES-TESLA_MODEL_S.

5) There are no duplicate and null values in the dataset which means it is a curated toy dataset.

After analyzing the dataset we now process the data as per the need of Facebook Prophet as shown :

```
tesla_df['Date'] = pd.to_datetime(tesla_df['Date'],format="%Y%m%d")
tesla_df.head()
```

| | Date | Store/Product | Value |
|---|---|---|---|
| 0 | 2018-01-01 | LOS_ANGELES-TESLA_MODEL_X | 2926.000 |
| 1 | 2018-01-02 | LOS_ANGELES-TESLA_MODEL_X | 2687.531 |
| 2 | 2018-01-03 | LOS_ANGELES-TESLA_MODEL_X | 2793.000 |
| 3 | 2018-01-04 | LOS_ANGELES-TESLA_MODEL_X | 2394.000 |
| 4 | 2018-01-05 | LOS_ANGELES-TESLA_MODEL_X | 2660.000 |

Once the date has been set as needed by the prophet we now proceed to create separate data frames for each model and store.

```
LA_TMX = tesla_df[tesla_df['Store/Product'] == 'LOS_ANGELES-TESLA_MODEL_X'].copy()
LA_TMS = tesla_df[tesla_df['Store/Product'] == 'LOS_ANGELES-TESLA_MODEL_S'].copy()
SF_TMS = tesla_df[tesla_df['Store/Product'] == 'SAN_FRANCISCO-TESLA_MODEL_S'].copy()
```

Now for each of these data frames, we drop the 'Store/Product' column and rename the 'Date' and 'Value' columns as 'ds' and 'y' respectively.

```
LA_TMX.drop('Store/Product', axis=1, inplace=True)
LA_TMS.drop('Store/Product', axis=1, inplace=True)
SF_TMS.drop('Store/Product', axis=1, inplace=True)
```

```
LA_TMX.columns = ['ds', 'y']
LA_TMS.columns = ['ds', 'y']
SF_TMS.columns = ['ds', 'y']
```

Now we create an instance of prophet class for each model and store and then fit them for training:

```
# Creating instance of prophet class for each model and store
LA_TMX_model = Prophet(interval_width=0.95)
LA_TMS_model = Prophet(interval_width=0.95)
SF_TMS_model = Prophet(interval_width=0.95)
```

Fitting the model for Each Model and store

```
training_LA_TMX = LA_TMX_model.fit(LA_TMX)
training_LA_TMS = LA_TMS_model.fit(LA_TMS)
training_SF_TMS = SF_TMS_model.fit(SF_TMS)
```

```
INFO:numexpr.utils:NumExpr defaulting to 2 threads.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

An important point to note here is that daily seasonality has been disabled because we are not taking into account any seasonal effects on a daily or any other basis.

Now let's take a look at the trained model for one of the products (for the rest it will be the same code-wise)

```
future_LA_TMX = LA_TMX_model.make_future_dataframe(periods=400, freq='D')
# periods tell how much far in future you want to predict and frequency tells how oftem do you want to predict, "D" means daily.
```

```
forecast = LA_TMX_model.predict(future_LA_TMX)
```

Here we are making future predictions for 400 days for the given product.

```
forecast.head()
```

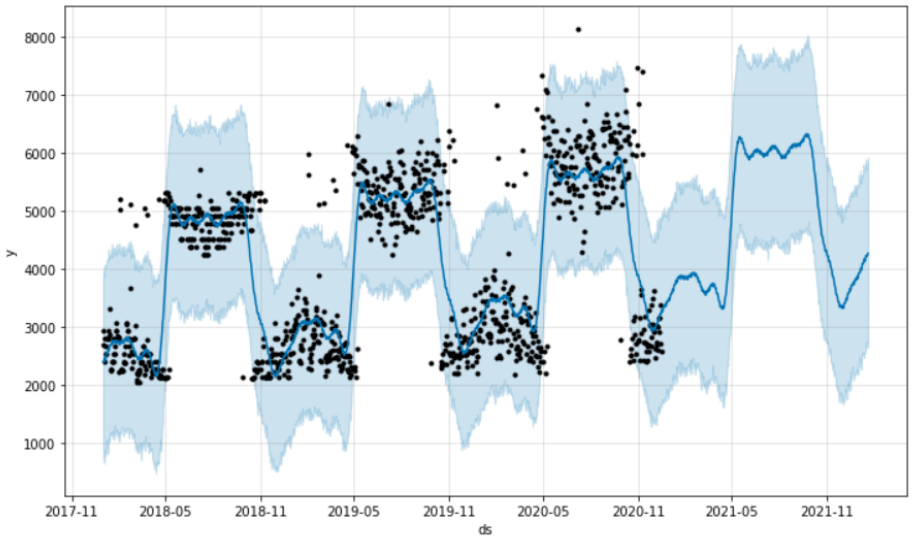
| | ds | trend | yhat_lower | yhat_upper | trend_lower | trend_upper | additive_terms | additive_terms_lower | additive_terms_upper | weekly | weekly_lower | weekly_upper | yearly | yearly_lower | yearly_upper |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2018-01-01 | 3510.912234 | 871.555187 | 3977.316382 | 3510.912234 | 3510.912234 | -1110.246686 | -1110.246686 | -1110.246686 | 16.628584 | 16.628584 | 16.628584 | -1126.875270 | -1126.875270 | -1126.875270 |
| 1 | 2018-01-02 | 3511.660929 | 710.023657 | 3868.435316 | 3511.660929 | 3511.660929 | -1120.378413 | -1120.378413 | -1120.378413 | -11.274964 | -11.274964 | -11.274964 | -1109.103449 | -1109.103449 | -1109.103449 |
| 2 | 2018-01-03 | 3512.409624 | 652.493080 | 4069.926535 | 3512.409624 | 3512.409624 | -1094.548052 | -1094.548052 | -1094.548052 | -4.403695 | -4.403695 | -4.403695 | -1090.144357 | -1090.144357 | -1090.144357 |
| 3 | 2018-01-04 | 3513.158319 | 876.447695 | 4036.116088 | 3513.158319 | 3513.158319 | -1047.593632 | -1047.593632 | -1047.593632 | 22.480903 | 22.480903 | 22.480903 | -1070.074535 | -1070.074535 | -1070.074535 |
| 4 | 2018-01-05 | 3513.907015 | 853.545053 | 4037.614734 | 3513.907015 | 3513.907015 | -1083.603163 | -1083.603163 | -1083.603163 | -34.579879 | -34.579879 | -34.579879 | -1049.023283 | -1049.023283 | -1049.023283 |

| multiplicative_terms | multiplicative_terms_lower | multiplicative_terms_upper | yhat |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 2400.665548 |
| 0.0 | 0.0 | 0.0 | 2391.282516 |
| 0.0 | 0.0 | 0.0 | 2417.861572 |
| 0.0 | 0.0 | 0.0 | 2465.564688 |
| 0.0 | 0.0 | 0.0 | 2430.303852 |

This is what a typical forecast data looks like, it consists of ‘ds’ column which consists of the date and ‘yhat’ is the forecast by the prophet. It also contains other data which shows the trend value, weekly values, etc. But here our main focus is on two columns namely ‘ds’ and ‘yhat’.

Plotting the forecast we get:

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

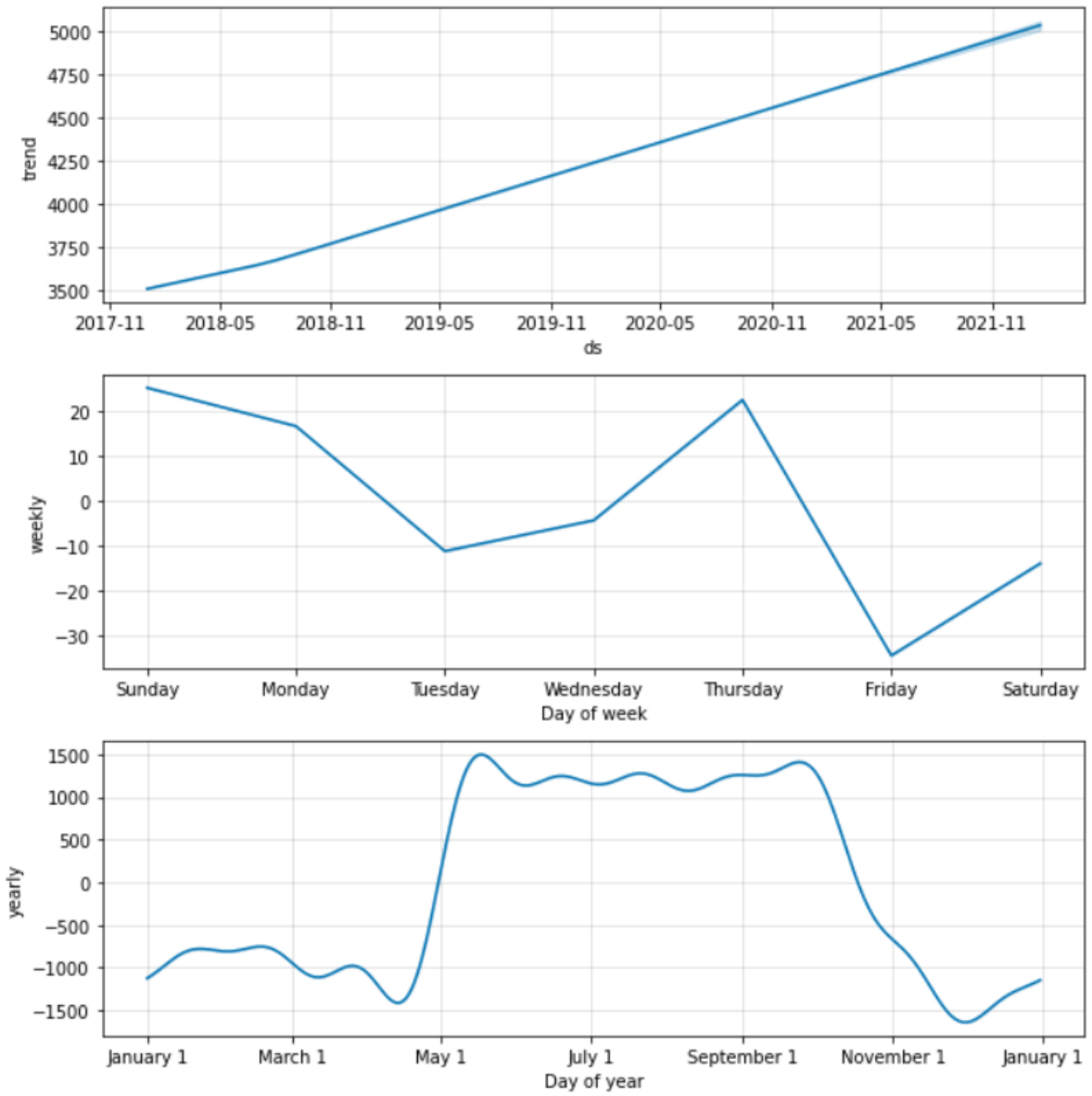



From the above graph, we can see that the black dots represent our training data and the blue line represents the forecast done by the model and it shows a pretty accurate forecast (with one or two outliers present in training data which can be ignored).

After training, it has also been forecasted for the next 400 days and the blue shaded region shows the region where the values can vary i.e. maxima and minima of the forecast. These maxima and minima indicate that if in the future some event happens on a miniature scale that may affect the product sales directly or indirectly this is the range in which the value will oscillate. However, it is to be noted that any sudden change or any big event that affects the sales of the product occurs then this forecast becomes irrelevant as the values will change drastically hence that event will have to be incorporated into the next forecast and then again we will have to make this plot.

Plotting the various forecast components we get:

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




Here we can see how the trend for a product has been over the year and the weekly as well as the yearly sales of the product.

From the graph, it can be seen that trend for Tesla cars has been increasing for the past couple of years and steady growth can be expected. The weekly and yearly sales graphs tell about when it is most likely that a Tesla car will be sold.

# References


1. Time Series Analysis with Facebook Prophet: How it works and How to use it
2. Prophet | Forecasting at scale.
3. Interfaces
4. 1.1 Linear regression
5. https://arxiv.org/ftp/arxiv/papers/2005/2005.07575.pdf
6. Stan: A probabilistic programming language for Bayesian inference and optimization∗ 1. What is Stan? Users' perspective