**PROBLEM STATEMENT**

**Topic: Forecast Future Sales With Prophet**

**Synopsis:**

* **AIM**
* **INTRODUCTION**
* **INSTALLATION OF PROPHET**
* **PYTHON API**
* **BASIC SETUP**
* **TIME SERIES FORECASTING WITH PROPHET**
* **PLOTTED THE FORECASTED COMPONENTS**
* **ADDING CHANGE POINTS TO PROPHET**
* **ADJUSTED TREND**
* **CONCLUSION**
* **REFRENCES**

**Aim:**

* In this section, we will explore using the Prophet to forecast the car sales dataset.
* Let’s start by fitting a model on the dataset.

# **1.Introduction to 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.

Prophet is open source software released by Facebook’s Core Data Science team. It is available for download on CRAN and PyPI.

* So, [Prophet](https://facebook.github.io/prophet/) is the facebooks’ open source tool for making time series predictions.
* [Prophet](https://facebook.github.io/prophet/) decomposes time series data into trend, seasonality and holiday effect.
* **Trend** models non periodic changes in the time series data.
* **Seasonality** is caused due to the periodic changes like daily, weekly, or yearly seasonality.
* **Holiday effect** which occur on irregular schedules over a day or a period of days.

# **2.Installation of Prophet** :

We can install Prophet using either command prompt or Anaconda prompt using pip as follows:

pip install fbprophet

Note: you may need to restart the kernel to use updated packages.

# **3. Python API** :

* [Prophet](https://facebook.github.io/prophet/docs/quick_start.html#python-api) follows the sklearn model API.
* First up, we create an instance of the Prophet class and then call its fit and predict methods.
* **The input to Prophet is always a dataframe with two columns** - **ds** and **y**.
* The **ds (datestamp)** column should be of a format expected by Pandas, ideally YYYY-MM-DD for a date or YYYY-MM-DD HH:MM:SS for a timestamp.
* The **y** column must be numeric, and represents the measurement we wish to forecast.

# **4. Basic Setup** :

* Now wel will dive right in and see how to make time series predictions using Prophet.
* We will explore the change points, how to include holidays and then add multiple regressors.
* First up, we will import the required libraries and the data.

### **Import libraries**

from fbprophet import Prophet

from fbprophet.plot import plot\_plotly

import plotly.offline as py

py.init\_notebook\_mode()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

plt.style.use('fivethirtyeight')

### **Import data**

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

file = '/kaggle/input/air-passengers/AirPassengers.csv'

df = pd.read\_csv(file)

### **Preview dataset:**

df.head()

**OUTPUT:**

| Month | #Passengers |
| --- | --- |
| 0 | 1949-01 | 112 |
| 1 | 1949-02 | 118 |
| 2 | 1949-03 | 132 |
| 3 | 1949-04 | 129 |
| 4 | 1949-05 | 121 |

We should rename the column name #Passenegrs as AirPassengers

df.rename(columns = {'#Passengers':'AirPassengers'}, inplace = True)

### **Summary of dataset:**

Now, we will print the information about the dataset that will tell us about the columns, data type of the columns and whether the column is null or not null.

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 144 entries, 0 to 143

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Month 144 non-null object

1 AirPassengers 144 non-null int64

dtypes: int64(1), object(1)

memory usage: 2.4+ KB

**EXPLAINATION:**

* We can see that the dataset contains a Month and AirPassengers column.
* Their data types are object and int64 respectively.
* The [Prophet](https://facebook.github.io/prophet/) library expects as input a dataframe with one column containing the time information, and another column containing the metric that we wish to forecast.
* The important thing to note is that, the Month column must be of the datetime type. But, we can see that it is of object data type. Now, because the Month column is not of the datetime type. So, we’ll need to convert it into datetime type.

df['Month'] = pd.DatetimeIndex(df['Month'])

df.dtypes

**OUTPUT:**

Month datetime64[ns]

AirPassengers int64

dtype: object

We can now see that our Month column is of the correct datetime type.

* [Prophet](https://facebook.github.io/prophet/) also imposes the strict condition that the input columns must be named as **ds (the time column)** and **y (the metric column)**.
* So, we must rename the columns in our dataframe.

df = df.rename(columns={'Month': 'ds','AirPassengers': 'y'})

df.head()

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
|  | ds | y |
| 0 | 1949-01-01 | 112 |
| 1 | 1949-02-01 | 118 |
| 2 | 1949-03-01 | 132 |
| 3 | 1949-04-01 | 129 |

We can see that the column names are renamed accordingly.

### **Visualize the data:**

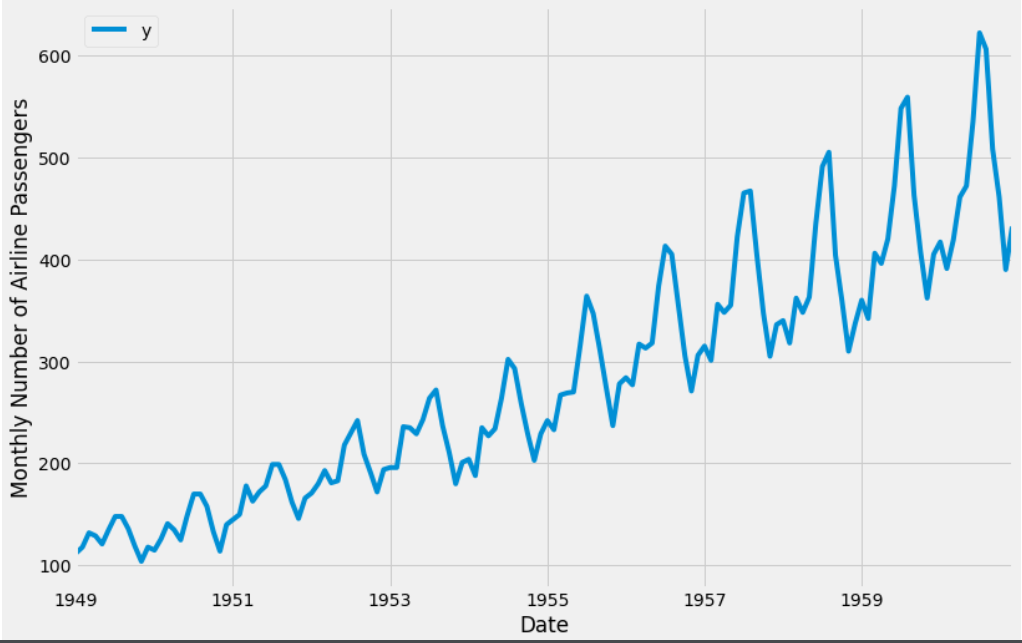
Now, it is considered a good practice to visualize the data at hand. So let’s plot our time series data:

ax = df.set\_index('ds').plot(figsize=(12, 8))

ax.set\_ylabel('Monthly Number of Airline Passengers')

ax.set\_xlabel('Date')

plt.show()



Now, our dataset is prepared and we are ready to use the Prophet library to produce forecasts of our time series.

# **6. Time Series Forecasting with Prophet:**

* Now, we will describe how to use the [Prophet](https://facebook.github.io/prophet/) library to predict future values of our time series data.
* The developers of [Prophet](https://facebook.github.io/prophet/) have made it more intuitive for analysts and developers alike to work with time series data.
* To begin, we must instantiate a new Prophet object. Prophet enables us to specify a number of arguments. For example, we can specify the desired range of our uncertainty interval by setting the interval\_width parameter.

*# set the uncertainty interval to 95% (the Prophet default is 80%)*

my\_model = Prophet(interval\_width=0.95)

* Now that our Prophet model has been initialized, we can call its fit method with our DataFrame as input.

my\_model.fit(df)

**Output:**

<fbprophet.forecaster.Prophet at 0x7f9255c62c90>

* In order to obtain forecasts of our time series, we must provide Prophet with a new DataFrame containing a ds column that holds the dates for which we want predictions.
* Conveniently, we do not have to concern ourselves with manually creating this DataFrame, as Prophet provides the make\_future\_dataframe helper function.

future\_dates = my\_model.make\_future\_dataframe(periods=36, freq='MS')

future\_dates.head()

**Output:**

|  |  |
| --- | --- |
|  | ds |
| **0** | 1949-01-01 |
| **1** | 1949-02-01 |
| **2** | 1949-03-01 |
| **3** | 1949-04-01 |
| **4** | 1949-05-01 |

* In the code snippet above, we instructed Prophet to generate 36 datestamps in the future.
* When working with Prophet, it is important to consider the frequency of our time series.
* Because we are working with monthly data, we clearly specified the desired frequency of the timestamps (in this case, MS is the start of the month).
* Therefore, the make\_future\_dataframe generated 36 monthly timestamps for us.
* In other words, we are looking to predict future values of our time series 3 years into the future.
* The DataFrame of future dates is then used as input to the predict method of our fitted model.

**Code:**

forecast = my\_model.predict(future\_dates)

forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].head()

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ds** | yhat | yhat\_lower | yhat\_upper |
| **0** | 1949-01-01 | 85.667868 | 40.711662 | 130.242936 |
| **1** | 1949-02-01 | 79.176553 | 35.772575 | 124.991217 |
| **2** | 1949-03-01 | 110.839332 | 69.379890 | 155.182425 |
| **3** | 1949-04-01 | 108.472210 | 67.188442 | 153.991365 |
| **4** | 1949-05-01 | 111.854130 | 69.954869 | 157.146512 |

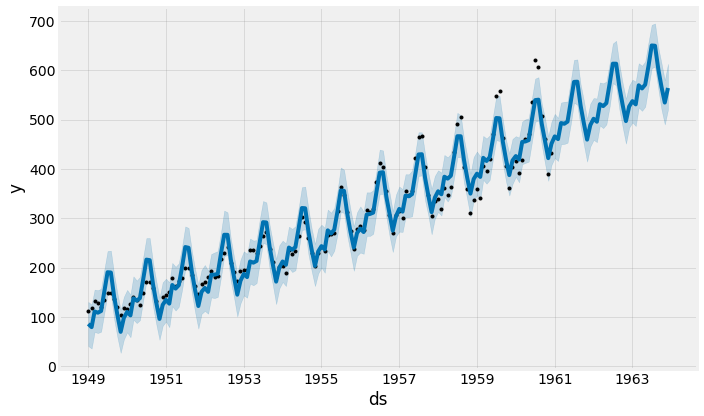
Prophet returns a large DataFrame with many interesting columns, but we subset our output to the columns most relevant to forecasting.

These are:

* **ds**: the datestamp of the forecasted value.
* **yhat**: the forecasted value of our metric (in Statistics, yhat is a notation traditionally used to represent the predicted values of a value y).
* **yhat\_lower**: the lower bound of our forecasts.
* **yhat\_upper**: the upper bound of our forecasts.
* A variation in values from the output presented is to be expected as Prophet relies on **Markov chain Monte Carlo (MCMC)** methods to generate its forecasts.
* MCMC is a stochastic process, so values will be slightly different each time
* Prophet also provides a convenient function to quickly plot the results of our forecasts as follows:

my\_model.plot(forecast, uncertainty=True)

**Output:**



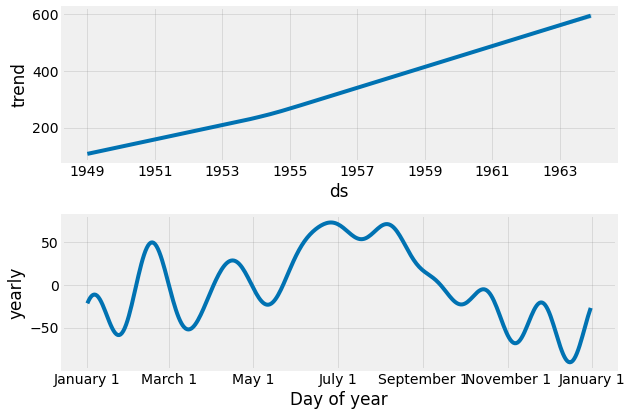
**Explaination:**

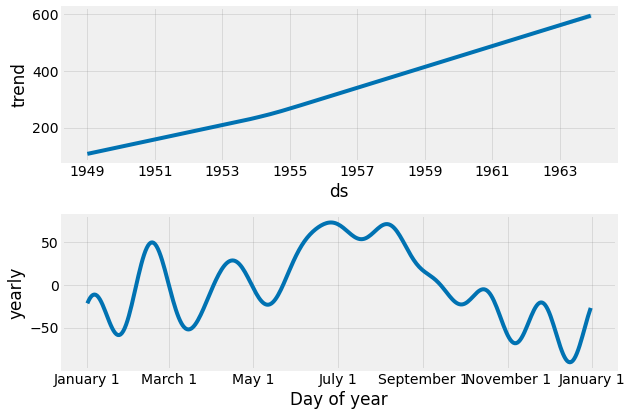
* Prophet plots the observed values of our time series (the black dots), the forecasted values (blue line) and the uncertainty intervals of our forecasts (the blue shaded regions).
* One other particularly strong feature of Prophet is its ability to return the components of our forecasts.
* This can help reveal how daily, weekly and yearly patterns of the time series contribute to the overall forecasted values

**Code:**

my\_model.plot\_components(forecast

**Output:**





**Explaination:**

* The above plot provides interesting insights.
* The first plot shows that the monthly volume of airline passengers has been linearly increasing over time.
* The second plot highlights the fact that the weekly count of passengers peaks towards the end of the week and on Saturday.
* The third plot shows that the most traffic occurs during the holiday months of July and August.

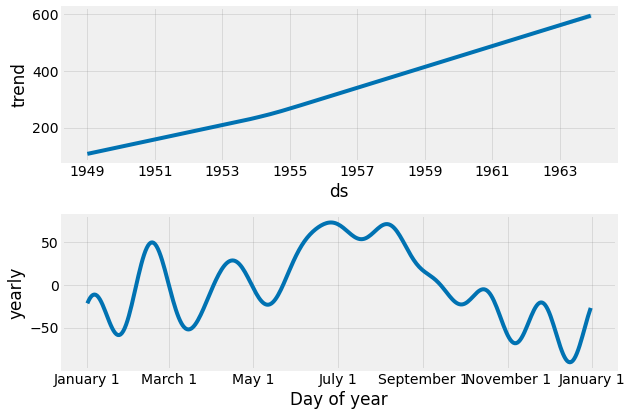
# **7. Plotting the forecasted components :**

* We can plot the trend and seasonality, components of the forecast as follows:

**Code:**

fig1 = my\_model.plot\_components(forecast)

**Output:**



# **8. Adding ChangePoints to Prophet** **:**

* Changepoints are the datetime points where the time series have abrupt changes in the trajectory.
* By default, Prophet adds 25 changepoints to the initial 80% of the data-set.
* Let’s plot the vertical lines where the potential changepoints occurred.

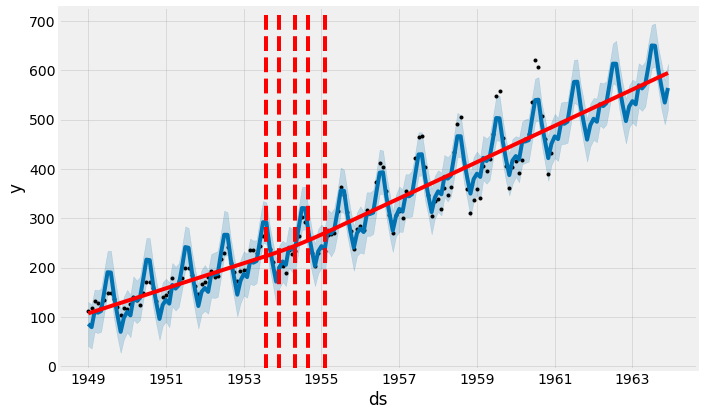
**Code:**

from fbprophet.plot import add\_changepoints\_to\_plot

fig = my\_model.plot(forecast)

a = add\_changepoints\_to\_plot(fig.gca(), my\_model, forecast)

**Output:**



We can view the dates where the chagepoints occurred:

**Code:**

my\_model.changepoints

**Output:**

5 1949-06-01

9 1949-10-01

14 1950-03-01

18 1950-07-01

23 1950-12-01

27 1951-04-01

32 1951-09-01

36 1952-01-01

41 1952-06-01

46 1952-11-01

50 1953-03-01

55 1953-08-01

59 1953-12-01

64 1954-05-01

68 1954-09-01

73 1955-02-01

78 1955-07-01

82 1955-11-01

87 1956-04-01

91 1956-08-01

96 1957-01-01

100 1957-05-01

105 1957-10-01

109 1958-02-01

114 1958-07-01

Name: ds, dtype: datetime64[ns]

* We can change the inferred changepoint range by setting the *changepoint\_range*

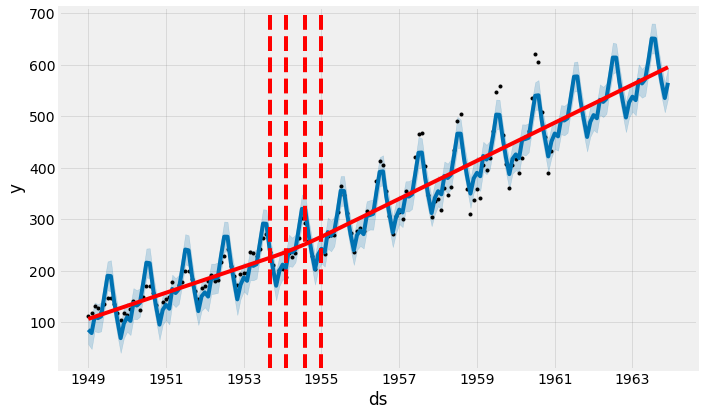
pro\_change= Prophet(changepoint\_range=0.9)

forecast = pro\_change.fit(df).predict(future\_dates)

fig= pro\_change.plot(forecast);

a = add\_changepoints\_to\_plot(fig.gca(), pro\_change, forecast)

**Output:**



The number of changepoints can be set by using the n\_changepoints parameter when initializing prophet.

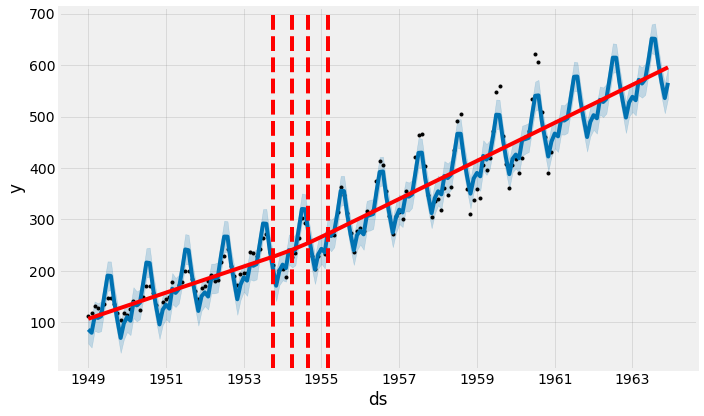
pro\_change= Prophet(n\_changepoints=20, yearly\_seasonality=True)

forecast = pro\_change.fit(df).predict(future\_dates)

fig= pro\_change.plot(forecast);

a = add\_changepoints\_to\_plot(fig.gca(), pro\_change, forecast)

**Output:**



# **9. Adjusting Trend:**

* Prophet allows us to adjust the trend in case there is an overfit or underfit.
* changepoint\_prior\_scale helps adjust the strength of the trend.
* Default value for changepoint\_prior\_scale is 0.05.
* Decrease the value to make the trend less flexible.
* Increase the value of changepoint\_prior\_scale to make the trend more flexible.
* Increasing the changepoint\_prior\_scale to 0.08 to make the trend flexible.

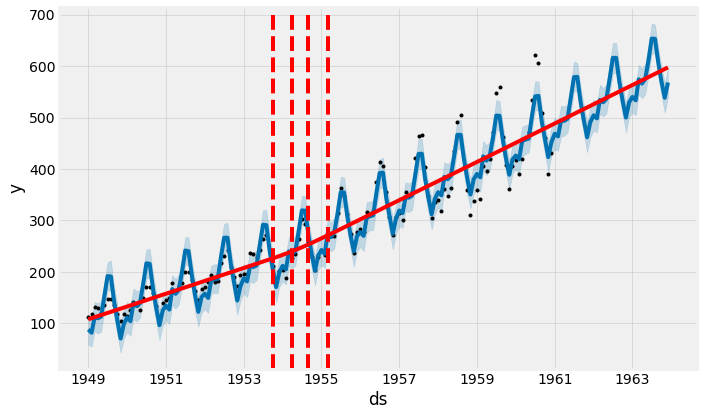
pro\_change= Prophet(n\_changepoints=20, yearly\_seasonality=True, changepoint\_prior\_scale=0.08)

forecast = pro\_change.fit(df).predict(future\_dates)

fig= pro\_change.plot(forecast);

a = add\_changepoints\_to\_plot(fig.gca(), pro\_change, forecast)

**Output:**



* Decreasing the *changepoint\_prior\_scale* to 0.001 to make the trend less flexible.

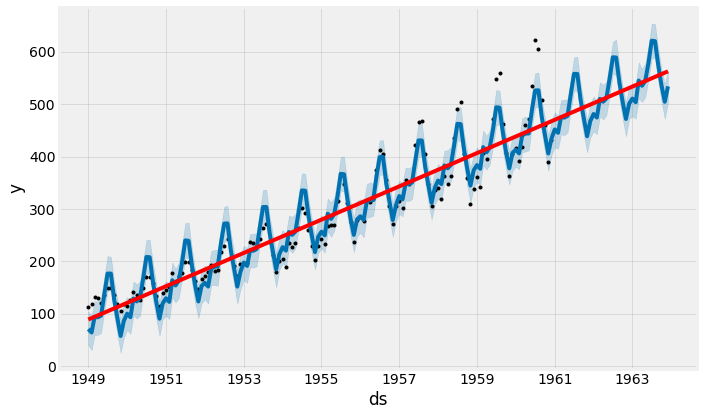
pro\_change= Prophet(n\_changepoints=20, yearly\_seasonality=True, changepoint\_prior\_scale=0.001)

forecast = pro\_change.fit(df).predict(future\_dates)

fig= pro\_change.plot(forecast);

a = add\_changepoints\_to\_plot(fig.gca(), pro\_change, forecast)

**Output:**



# **10. Conclusion**:

* In this tutorial, we described how to use the Prophet library to perform time series forecasting in Python.
* We have been using out-of-the box parameters, but Prophet enables us to specify many more arguments.
* In particular, Prophet provides the functionality to bring your own knowledge about time series to the table.

# **11. References:**

The concepts and ideas in this notebook are tgaken from the following websites-

1.<https://facebook.github.io/prophet/>

2.<https://facebook.github.io/prophet/docs/quick_start.html>

3.<https://peerj.com/preprints/3190.pdf>

4.<https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-prophet-in-python-3>

**By,**

**Team Members,**

**A Thirukarthikeyan B. Tech(Information Technology)**

**Veluprasath B. Tech(Information Technology)**

**Praveen kumar B. Tech(Information Technology)**

**Vignesh B.Tech(Information Technology)**

**Abinesh B.Tech(Information Technology)**