Forecast future sales with prophet library.

TNSDC IBM PROJECT

PHASE 2

TEAM MEMBERS:

A. THIRUKARTHIKEYAN[B.TECH IT]

V. PRAVEENKUMAR[B.TECH IT]

M .VIGNESH [B.TECH IT]

M .ABINESH [B.TECH IT]

A .VELUPRASAATH [B.TECH IT]

IMPLEMENTATION OF THE PROPHET IN FUTURE SALES(SALES FORECASTING)

- INSTALL PYTHON LIBRARIES (PANDAS, FBPROPHET).
- IMPORT THE LIBRARIES AND LOAD YOUR DATA SALES DATA FROM USING THE FBPROPHET DATAFRAME.
- CREATE AND FIT THE PROPHET MODEL.
- CREATE THE DATA FRAME FOR FUTURE SALES.
- GENERATE THE FORECASTS
- VISUALIZE THE RESULTS
- ACCESSS THE FORECAST VALUES

LEVEL OF PROPHET IN FUTURE SALES

- QUALITY OF DATA
- MODEL TUNING
- DATA CHARACTERSTICS
- DOMAIN KNOWLEDGE
- EVALUATION
- DATA AVAILABILITY
- EXTERNAL FACTORS
- AUTOMATION

What is Prophet Technique

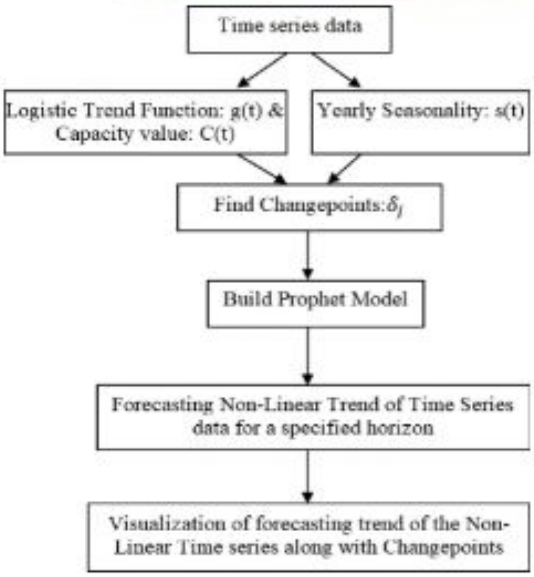
- Prophet is a time series forecasting technique developed by facebook's

Datascience team.

- It is designed to handle daily observations that display patterns on different time scales.
- **Overview of prophet techniques:**

1. Additive model
2. Trend component
3. Seasonality
4. Holiday effects
5. Automatic detection

Steps for Prophet technique



Scope of prophet techniques

- *Robust handling of seasonality:-*

Prophet is well suited for datasets with strong seasonal patterns, making it effective for industries where sales exhibit recurring trends.

- *Customization for holidays and special events:-*

the ability to include custom holidays and special events allows business to tailor the model to their specific industry, accounting for unique factors influencing sales

- *Uncertainty estimation:-*

The inclusion of uncertainty intervals in predictions provides a measure of the forecast's reliability .this is valuable for business that need to access the level of confidence in their sales predictions.

- *Demonstrated success stories:-*

numerous success stories across industries showcase the effectiveness of prophet in accurately predicting future sales ,in stilling confidence in its application for future forecasting endeavours.

Thank you

PROBLEM STATEMENT

Phase-2

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 is the facebooks' open source tool for making time series predictions.
- 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 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 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 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

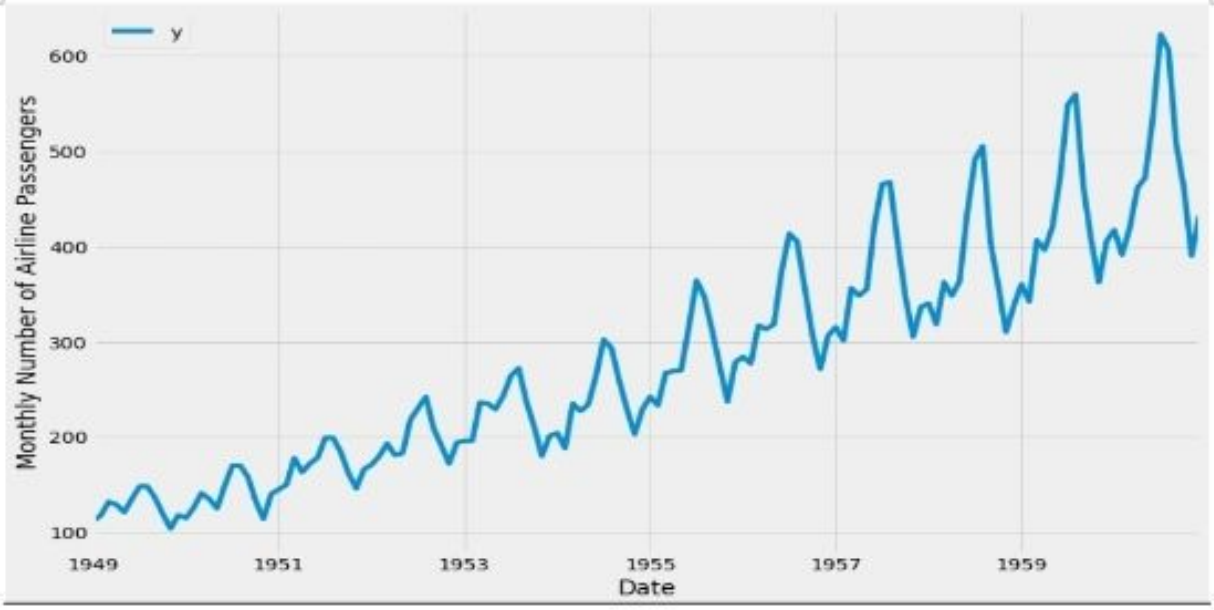
We can see that the column names are renamed accordingly.

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 library to predict future values of our time series data.
- The developers of 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:

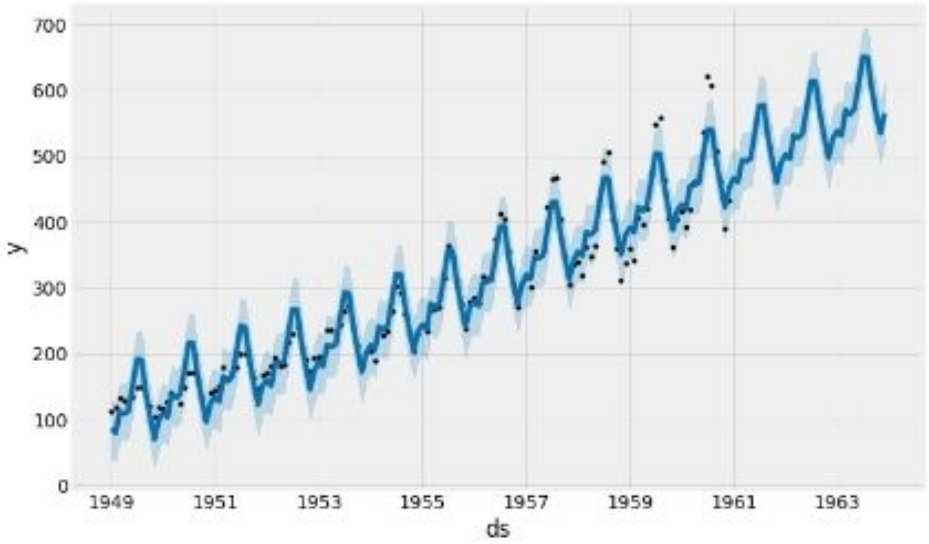
- **ds**: the datestamp of the forecasted value.
- **yhat**: the forecasted value of our metric (in Statistics, yhat is a notation traditionally used to represent the predicted values of a value y).
- **yhat_lower**: the lower bound of our forecasts.
- **yhat_upper**: the upper bound of our forecasts.
- 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).

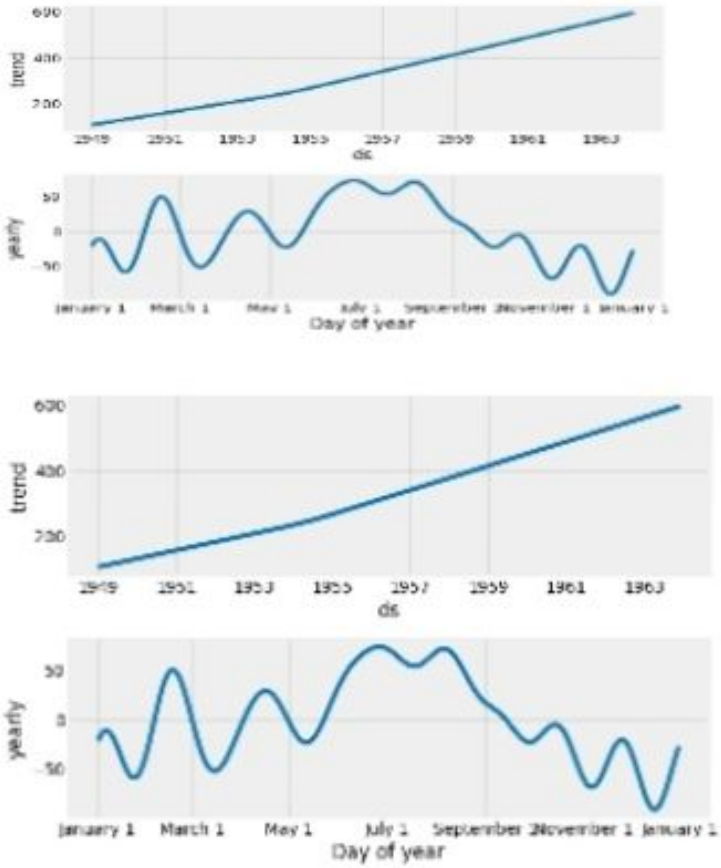
- One other particularly strong feature of Prophet is its ability to return the components of our forecasts.

- 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.

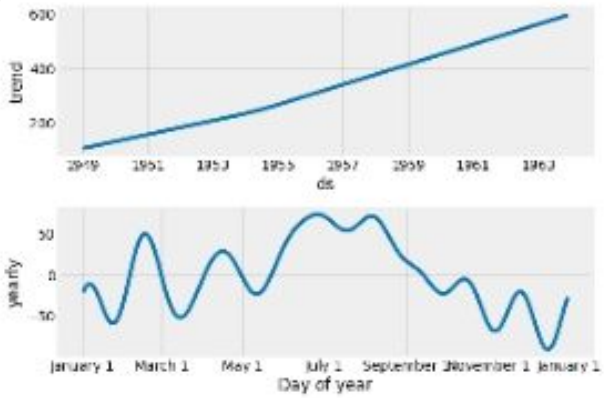
7. Plotting the forecasted components :

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

  Code:

```
fig1 = my_model.plot_components(forecast)
```

Output:



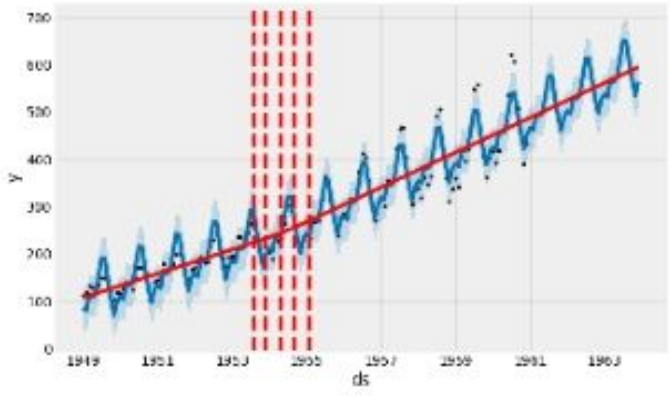
8. Adding ChangePoints to Prophet :

- 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:

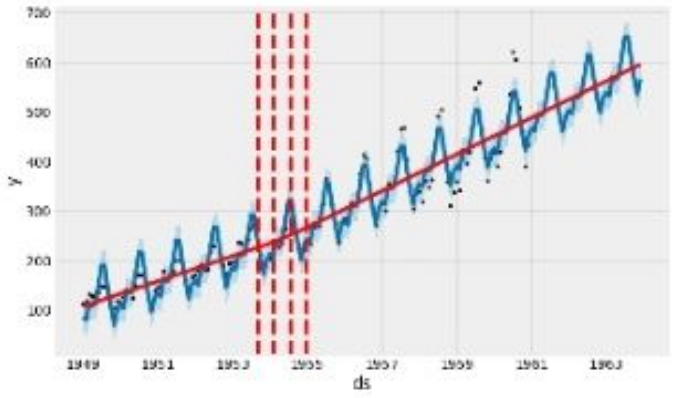
```
5    1949-06-01
9    1949-10-01
14   1950-03-01
18   1950-07-01
23   1950-12-01
27   1951-04-01
```

```
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)
```

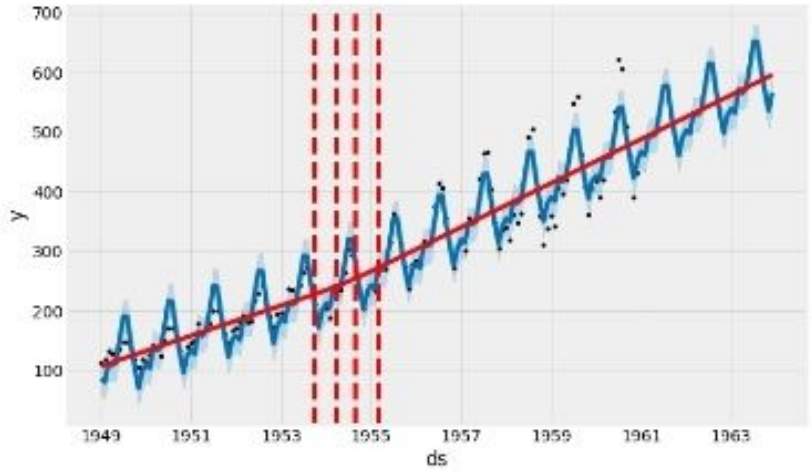
Output:



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:

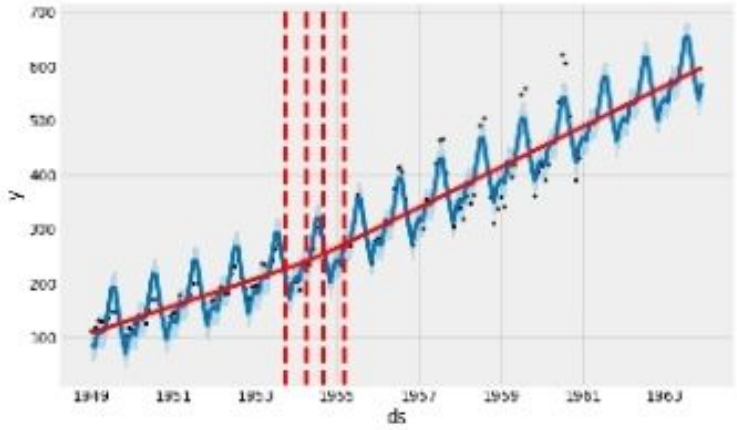


9. Adjusting Trend:

- 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_s
cale=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)
```

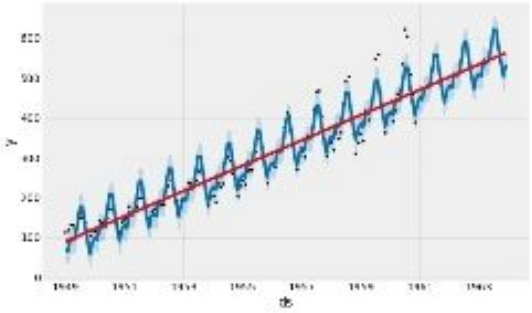
Output:



- 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_s
cale=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)

Phase 2.py

```
# check prophet version
import fbprophet
# print version number
print('Prophet %s' % fbprophet.__version__)
##
# load the car sales dataset
from pandas import read_csv
# load data
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# summarize shape
print(df.shape)
# show first few rows
print(df.head())
##
# load and plot the car sales dataset
from pandas import read_csv
from matplotlib import pyplot
# load data
path = 'https://raz.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# plot the time series
df.plot()
pyplot.show()
...
# prepare expected column names
df.columns = ['ds', 'y']
df['ds']= to_datetime(df['ds'])
# fit prophet model on the car sales dataset
from pandas import read_csv
from pandas import to_datetime
from fbprophet import Prophet
# load data
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# prepare expected column names
df.columns = ['ds', 'y']
df['ds']= to_datetime(df['ds'])
# define the model
model = Prophet()
# fit the model
model.fit(df)
...
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
	date = '1968-%02d' % i
	future.append([date])
```

```
future = DataFrame(future)
future.columns = ['ds']
future['ds']= to_datetime(future['ds'])
...
# summarize the forecast
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
...
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
# plot forecast
model.plot(forecast)
pyplot.show()

# make an in-sample forecast
from pandas import read_csv
from pandas import to_datetime
from pandas import DataFrame
from fbprophet import Prophet
from matplotlib import pyplot
# load data
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# prepare expected column names
df.columns = ['ds', 'y']
df['ds']= to_datetime(df['ds'])
# define the model
model = Prophet()
# fit the model
model.fit(df)
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
	date = '1968-%02d' % i
	future.append([date])
future = DataFrame(future)
future.columns = ['ds']
future['ds']= to_datetime(future['ds'])
# use the model to make a forecast
forecast = model.predict(future)
# summarize the forecast
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
# plot forecast
model.plot(forecast)
pyplot.show()

...
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
	date = '1969-%02d' % i
```

```
	future.append([date])
future = DataFrame(future)
future.columns = ['ds']
future['ds']= to_datetime(future['ds'])

# make an out-of-sample forecast
from pandas import read_csv
from pandas import to_datetime
from pandas import DataFrame
from fbprophet import Prophet
from matplotlib import pyplot
# load data
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# prepare expected column names
df.columns = ['ds', 'y']
df['ds']= to_datetime(df['ds'])
# define the model
model = Prophet()
# fit the model
model.fit(df)
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
	date = '1969-%02d' % i
	future.append([date])
future = DataFrame(future)
future.columns = ['ds']
future['ds']= to_datetime(future['ds'])
# use the model to make a forecast
forecast = model.predict(future)
# summarize the forecast
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
# plot forecast
model.plot(forecast)
pyplot.show()

...
# create test dataset, remove last 12 months
train = df.drop(df.index[-12:])
print(train.tail())

...
# calculate MAE between expected and predicted values for december
y_true = df['y'][-12:].values
y_pred = forecast['yhat'].values
mae = mean_absolute_error(y_true, y_pred)
print('MAE: %.3f' % mae)
```

```
...
# plot expected vs actual
pyplot.plot(y_true, label='Actual')
pyplot.plot(y_pred, label='Predicted')
pyplot.legend()
pyplot.show()

# evaluate prophet time series forecasting model on hold out dataset
from pandas import read_csv
from pandas import to_datetime
from pandas import DataFrame
from fbprophet import Prophet
from sklearn.metrics import mean_absolute_error
from matplotlib import pyplot
# load data
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'
df = read_csv(path, header=0)
# prepare expected column names
df.columns = ['ds', 'y']
df['ds']= to_datetime(df['ds'])
# create test dataset, remove last 12 months
train = df.drop(df.index[-12:])
print(train.tail())
# define the model
model = Prophet()
# fit the model
model.fit(train)
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
	date = '1968-%02d' % i
	future.append([date])
future = DataFrame(future)
future.columns = ['ds']
future['ds'] = to_datetime(future['ds'])
# use the model to make a forecast
forecast = model.predict(future)
# calculate MAE between expected and predicted values for december
y_true = df['y'][-12:].values
y_pred = forecast['yhat'].values
mae = mean_absolute_error(y_true, y_pred)
print('MAE: %.3f' % mae)
# plot expected vs actual
pyplot.plot(y_true, label='Actual')
pyplot.plot(y_pred, label='Predicted')
pyplot.legend()
pyplot.show()
```

Sales_data_csv.csv

sales_data.csv

sales_data.csv

	A	B	C	D	E
1	TV	Radio	Newspaper	Sales	
2	230.1	37.8	69.2	22.1	
3	44.5	39.3	45.1	10.4	
4	17.2	45.9	69.3	12	
5	151.5	41.3	58.5	16.5	
6	180.8	10.8	58.4	17.9	
7	8.7	48.9	75	7.2	
8	57.5	32.8	23.5	11.8	
9	120.2	19.6	11.6	13.2	
10	8.6	2.1	1	4.8	
11	199.8	2.6	21.2	15.6	
12	66.1	5.8	24.2	12.6	
13	214.7	24	4	17.4	
14	23.8	35.1	65.9	9.2	
15	97.5	7.6	7.2	13.7	
16	204.1	32.9	46	19	
17	195.4	47.7	52.9	22.4	
18	67.8	36.6	114	12.5	
19	281.4	39.6	55.8	24.4	
20	69.2	20.5	18.3	11.3	
21	147.3	23.9	19.1	14.6	
22	218.4	27.7	53.4	18	
23	237.4	5.1	23.5	17.5	
24	13.2	15.9	49.6	5.6	
25	228.3	16.9	26.2	20.5	
26	62.3	12.6	18.3	9.7	
27	262.9	3.5	19.5	17	
28	142.9	29.3	12.6	15	
29	240.1	16.7	22.9	20.9	
30	248.8	27.1	22.9	18.9	
31	70.6	16	40.8	10.5	
32	292.9	28.3	43.2	21.4	
33	112.9	17.4	38.6	11.9	
34	97.2	1.5	30	13.2	
35	265.6	20	0.3	17.4	
36	95.7	1.4	7.4	11.9	
37	290.7	4.1	8.5	17.8	
38	266.9	43.8	5	25.4	
39	74.7	49.4	45.7	14.7	
40	43.1	26.7	35.1	10.1	

sales_data.csv

sales_data.csv

	A	B	C	D	E
39	74.7	49.4	45.7	14.7	
40	43.1	26.7	35.1	10.1	
41	228	37.7	32	21.5	
42	202.5	22.3	31.6	16.6	
43	177	33.4	38.7	17.1	
44	293.6	27.7	1.8	20.7	
45	206.9	8.4	26.4	17.9	
46	25.1	25.7	43.3	8.5	
47	175.1	22.5	31.5	16.1	
48	89.7	9.9	35.7	10.6	
49	239.9	41.5	18.5	23.2	
50	227.2	15.8	49.9	19.8	
51	66.9	11.7	36.8	9.7	
52	199.8	3.1	34.6	16.4	
53	100.4	9.6	3.6	10.7	
54	216.4	41.7	39.6	22.6	
55	182.6	46.2	58.7	21.2	
56	262.7	28.8	15.9	20.2	
57	198.9	49.4	60	23.7	
58	7.3	28.1	41.4	5.5	
59	136.2	19.2	16.6	13.2	
60	210.8	49.6	37.7	23.8	
61	210.7	29.5	9.3	18.4	
62	53.5	2	21.4	8.1	
63	261.3	42.7	54.7	24.2	
64	239.3	15.5	27.3	20.7	
65	102.7	29.6	8.4	14	
66	131.1	42.8	28.9	16	
67	69	9.3	0.9	11.3	
68	31.5	24.6	2.2	11	
69	139.3	14.5	10.2	13.4	
70	237.4	27.5	11	18.9	
71	216.8	43.9	27.2	22.3	
72	199.1	30.6	38.7	18.3	
73	109.8	14.3	31.7	12.4	
74	26.8	33	19.3	8.8	
75	129.4	5.7	31.3	11	
76	213.4	24.6	13.1	17	
77	16.9	43.7	89.4	8.7	
78	27.5	1.6	20.7	6.9	
79	120.5	28.5	14.2	14.2	
80	5.4	29.9	9.4	5.3	

sales_data.csv

	A	B	C	D	E
80	5.4	29.9	9.4	5.3	
81	116	7.7	23.1	11	
82	76.4	26.7	22.3	11.8	
83	239.8	4.1	36.9	17.3	
84	75.3	20.3	32.5	11.3	
85	68.4	44.5	35.6	13.6	
86	213.5	43	33.8	21.7	
87	193.2	18.4	65.7	20.2	
88	76.3	27.5	16	12	
89	110.7	40.6	63.2	16	
90	88.3	25.5	73.4	12.9	
91	109.8	47.8	51.4	16.7	
92	134.3	4.9	9.3	14	
93	28.6	1.5	33	7.3	
94	217.7	33.5	59	19.4	
95	250.9	36.5	72.3	22.2	
96	107.4	14	10.9	11.5	
97	163.3	31.6	52.9	16.9	
98	197.6	3.5	5.9	16.7	
99	184.9	21	22	20.5	
100	289.7	42.3	51.2	25.4	
101	135.2	41.7	45.9	17.2	
102	222.4	4.3	49.8	16.7	
103	296.4	36.3	100.9	23.8	
104	280.2	10.1	21.4	19.8	
105	187.9	17.2	17.9	19.7	
106	238.2	34.3	5.3	20.7	
107	137.9	46.4	59	15	
108	25	11	29.7	7.2	
109	90.4	0.3	23.2	12	
110	13.1	0.4	25.6	5.3	
111	255.4	26.9	5.5	19.8	
112	225.8	8.2	56.5	18.4	
113	241.7	38	23.2	21.8	
114	175.7	15.4	2.4	17.1	
115	209.6	20.6	10.7	20.9	
116	78.2	46.8	34.5	14.6	
117	75.1	35	52.7	12.6	
118	139.2	14.3	25.6	12.2	
119	76.4	0.8	14.8	9.4	
120	125.7	36.9	79.2	15.9	

sales_data.csv

sales_data.csv

	A	B	C	D	E
120	125.7	36.9	79.2	15.9	
121	19.4	16	22.3	6.6	
122	141.3	26.8	46.2	15.5	
123	18.8	21.7	50.4	7	
124	224	2.4	15.6	16.6	
125	123.1	34.6	12.4	15.2	
126	229.5	32.3	74.2	19.7	
127	87.2	11.8	25.9	10.6	
128	7.8	38.9	50.6	6.6	
129	80.2	0	9.2	11.9	
130	220.3	49	3.2	24.7	
131	59.6	12	43.1	9.7	
132	0.7	39.6	8.7	1.6	
133	265.2	2.9	43	17.7	
134	8.4	27.2	2.1	5.7	
135	219.8	33.5	45.1	19.6	
136	36.9	38.6	65.6	10.8	
137	48.3	47	8.5	11.6	
138	25.6	39	9.3	9.5	
139	273.7	28.9	59.7	20.8	
140	43	25.9	20.5	9.6	
141	184.9	43.9	1.7	20.7	
142	73.4	17	12.9	10.9	
143	193.7	35.4	75.6	19.2	
144	220.5	33.2	37.9	20.1	
145	104.6	5.7	34.4	10.4	
146	96.2	14.8	38.9	12.3	
147	140.3	1.9	9	10.3	
148	240.1	7.3	8.7	18.2	
149	243.2	49	44.3	25.4	
150	38	40.3	11.9	10.9	
151	44.7	25.8	20.6	10.1	
152	280.7	13.9	37	16.1	
153	121	8.4	48.7	11.6	
154	197.6	23.3	14.2	16.6	
155	171.3	39.7	37.7	16	
156	187.8	21.1	9.5	20.6	
157	4.1	11.6	5.7	3.2	
158	93.9	43.5	50.5	15.3	
159	149.8	1.3	24.3	10.1	
160	11.7	36.9	45.2	7.3	

sales_data.csv

sales_data.csv

	A	B	C	D	E
159	149.8	1.3	24.3	10.1	
160	11.7	36.9	45.2	7.3	
161	131.7	18.4	34.6	12.9	
162	172.5	18.1	30.7	16.4	
163	85.7	35.8	49.3	13.3	
164	188.4	18.1	25.6	19.9	
165	163.5	36.8	7.4	18	
166	117.2	14.7	5.4	11.9	
167	234.5	3.4	84.8	16.9	
168	17.9	37.6	21.6	8	
169	206.8	5.2	19.4	17.2	
170	215.4	23.6	57.6	17.1	
171	284.3	10.6	6.4	20	
172	50	11.6	18.4	8.4	
173	164.5	20.9	47.4	17.5	
174	19.6	20.1	17	7.6	
175	168.4	7.1	12.8	16.7	
176	222.4	3.4	13.1	16.5	
177	276.9	48.9	41.8	27	
178	248.4	30.2	20.3	20.2	
179	170.2	7.8	35.2	16.7	
180	276.7	2.3	23.7	16.8	
181	165.6	10	17.6	17.6	
182	156.6	2.6	8.3	15.5	
183	218.5	5.4	27.4	17.2	
184	56.2	5.7	29.7	8.7	
185	287.6	43	71.8	26.2	
186	253.8	21.3	30	17.6	
187	205	45.1	19.6	22.6	
188	139.5	2.1	26.6	10.3	
189	191.1	28.7	18.2	17.3	
190	286	13.9	3.7	20.9	
191	18.7	12.1	23.4	6.7	
192	39.5	41.1	5.8	10.8	
193	75.5	10.8	6	11.9	
194	17.2	4.1	31.6	5.9	
195	166.8	42	3.6	19.6	
196	149.7	35.6	6	17.3	
197	38.2	3.7	13.8	7.6	
198	94.2	4.9	8.1	14	
199	177	9.3	6.4	14.8	
200	283.6	42	66.2	25.5	
201	232.1	8.6	8.7	18.4	