Anomaly Detection in Time Series Data by Forecasting using Facebook Prophet

Mr. Atharva Khangar
AIDS
Vivekanand Education Society's
Institute of Technology (VESIT)
Mumbai, India
2020.atharva.khangar@ves.ac.in

Mrs. Sangeeta Oswal
AIDS
Vivekanand Education Society's
Institute of Technology (VESIT)
Mumbai, India
sangeeta.oswal@ves.ac.in

Mr. Shubham Hadawle
AIDS
Vivekanand Education Society's
Institute of Technology (VESIT)
Mumbai, India
2020.shubham.hadawle@ves.ac.in

***Abstract*—**

Anomaly detection is the process of identifying data points, observations or events which deviate from normal behaviour. Analysing Time-series data would help recognise the root cause of various trends or patterns that have surfaced over time. This is essential because the subsequent value in time-series data depends on the input from the previous value. Here we are detecting anomalies in time-series data using Facebook Prophet, an open-source library available for forecasting using R or Python.

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 along with holiday effects. It functions best with time-series data that have strong seasonal effects and several seasons of historical data. Prophet is stringent to missing data and shifts in the trend, and typically handles outliers well. In this paper we shall be applying the same on Real Ad Exchange dataset, which shows online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost-per thousand impressions (CPM).

I. Introduction

Identifying anomalous data can help indicate unusual incidents in a system, business or organisation. It can also provide us ways to use potential opportunities, which can greatly enhance the decision-making process. Time series data shows information of data over a set time period. In analysis and preprocessing of time series data, we can split and create samples according to randomness, unlike in normal data analysis. But as all the values are related to each other in a hierarchical order of time, the preprocessing should be done with care. Hidden patterns in data can be identified with different visualisation techniques like graphs, histograms, pie charts, etc., whereas to identify anomalies in a time series data, there are different models and algorithms.

II. Literature

"An anomaly is an observation or a sequence of observations which differ remarkably from the mainstream distribution of data."

The occurrence of anomalies in the data may be caused by numerous reasons such as Natural Variation, Data Measurement and Collection Errors, Data belonging to different Classes, etc. These anomalies are majorly categorised into three types; Point anomaly, Collective anomaly and Contextual Anomaly.

1. *Point Anomaly* -
   A single data point may deviate or differ significantly from the remnant of the data. Such an instance is referred to as Point Anomaly.
2. *Collective Anomaly* -
   At times, we come across cases where individual data points are not anomalous, rather we find a sequence of points that can be labelled as an Anomaly.
3. *Contextual Anomaly* -
   The values of some data points may seem suitable in some scenarios, while the same values may seem anomalous in other contexts.

The process of picking out or identifying such anomalies from the conventional data is called Anomaly Detection. There are three prime approaches for identifying anomalies: unsupervised, semi-supervised, supervised. Detecting Anomalies becomes substantive because anomalies help to either recognise a problem, a technical glitch, a potential opportunity or a new phenomenon that could influence the current business model. [3, 4]

➢ *Time Series Modelling*

Time Series is a collection of numeric values of the same entity, obtained by repeatedly measuring it at equal intervals of time. Time series data can be gathered yearly, quarterly, monthly, weekly, daily, hourly or even biennial or decennial. Example: For meteorological analysis, the rainfall of a particular geographical location needs to be noted at regular time periods.

Time Series Analysis attempts to acknowledge the underlying patterns in the time series data, to be able to make predictions for future events. Time Series Models are extensively used in statistics, economics, business, applied sciences to foretell the seasonal variability of a target entity, by using the previous values as input variables for the model. [5]

The Time Series Model has four key components:

1. **Trend** - gives the overall long-term direction of the series
2. **Seasonality** - occurs when there is repetition in the behaviour of the data, which occurs at regular intervals.

3. **Cycles** - occurs when the series follows some kind of an up and down pattern that isn't seasonal.
4. **Irregularity** - gives information about strange dips, jumps, fluctuations or non-seasonal patterns.

Time Series Data can be grouped into two classes:

1. *Univariate* -

This time series consists of sequential measurements of a single variable over time. Example: We track a person's body weight on a daily basis.

2. *Multivariate* -

Multivariate time series consists of regular measurements of multiple variables that are interrelated over time. Example: Say, along with the body weight, we measure the person's height, body mass index and calorie intake, which are all interconnected.

Anomaly Detection in Multivariate Time Series is more complex than in Univariate Time Series. This is because, as the number of random variables increases, the prospects of making prediction errors increases. [3]

Here, we would be working to detect anomalies in Multivariate Time-Series Data using the Prophet Model.

➢ *Prophet Model*

Facebook Prophet is an open source tool launched by Facebook (now Meta) in 2017, used for forecasting and goal setting, which can be implemented in Python and R. It is the predecessor to NeuralProphet. Prophet is an additive regression model. It identifies the seasonality and trend from the given data and then combines them to get the forecast. FB Prophet finds the best fitting curve by using a linear logistic curve component for external regressor. Prophet is a favourable toolkit as it has easily understandable parameters and does not require a lot of data to automatically identify seasonal trends.

The Prophet's Algorithm incorporates four vital ingredients:

1) A piecewise linear or logistic growth curve trend. (The model selects changepoints from the data and thus detects changes in the trends.)
2) A yearly seasonal component that has been modelled using Fourier Series.
3) A weekly seasonal component modelled using dummy variables.
4) A record of holidays that has been provided by the user.

Ergo, the algorithm can be mathematically represented as follows -

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \qquad \text{..... (1)}$$

where,

- g(t) is called the growth function that gives the trends in the data
- s(t) is the seasonality function
- h(t) gives the holiday effect
- ε_t is the error term that speaks for any feature of the data that wasn't fitted by the model [6]

❖ *Growth Function*

The growth function, also referred to as the trend factor, tries to acclimate the patterns or trends shown by the data. The Prophet Algorithm, by default, uses a 'Linear Model' to model its growth (signifies a constant rate in the growth). However, by using this function, we can apply either of two trend models; Piecewise Linear Model and Logistic Growth Model.

1) Piecewise Linear Model -

At times, only a single linear model isn't sufficient. This is because the time-series happens to have some abrupt changes in their trajectories. The algorithm identifies these changepoints where the slope changes, and uses different linear equations for different slopes, combining them to model the growth function. These changepoints can be specified by the user or else the Prophet recognises them automatically, if they haven't been specified explicitly. [6, 7] Mathematically, it can be formulated as:

$$g(t) = \beta_0 + \beta_1 x + \beta_2 (x - c)^+ + \varepsilon \qquad \text{.....(2)}$$

2) Logistic Growth Model -

This trend function is used to model non-linear growths with saturation limits. These saturations help to set an upper or lower bound (cap or floor) and limit the values of the trend. Thus, ensuring that the growth does not surpass a specific maximum or minimum value, we happen to formulate the trend into an S-Curve. Say 'L' gives the cap or floor, 'k' gives the growth rate, 't' time and 'm' the offset, then the mathematical formulation would be:

$$g(t) = L / 1 + e^{-k\,(t - m)} \qquad \text{.....(3)}$$

We can also choose to set the growth to be flat. In such a flat trend, there is no growth over time and the growth function is a constant value. [7, 9]

❖ *Seasonality Function*

For this component, a Fourier series is used to account for seasonal patterns. It was mainly designed to model yearly, weekly and daily seasonal effects. Although, apart from this, one can also use conditional seasonality.

The fourier series is formed by combining sine and cosine terms that have been multiplied by some coefficient, thus using it to analyse harmonic functions in our data. Associating sine and cosine waves of different frequencies, we can take after nearly any form of curve.

For the same, we make use of a partial Fourier sum (also referred to as the order), which is a parameter to discover how quickly the seasonality changes. Increasing the order allows the seasonality to fit faster-changing cycles. The function can be represented as:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right) \qquad \text{.....(4)}$$

Here, P gives the period of the particular seasonality. Like, for yearly seasonality, we have P = 354.25 and for weekly seasonality, P = 7. [1, 2, 6, 7, 9]

❖ ***Holiday Effect or Event Function***

The Holiday Effect component in the Prophet Algorithm, is a point intervention variable. That is, it acts as a binary indicator for a certain day, date or time, which can be used to highlight the respective data point. This allows the Prophet model to adjust the forecasting when a holiday or major occasion may tend to affect the forecast.

The function takes a matrix list of dates as input and adds or subtracts values from the growth, the seasonality terms and the forecast for those particular days, according to the historical data.

The major holidays for each country are provided by a package called ‘holidays’. One can use those or else create their own dataframe for the same. [2, 6, 7, 9]

Suppose Z(t) is a matrix of regressors with D as the set of holiday dates and $k \sim \text{Normal}(0, v^2)$ where v is the smoothing parameter, then:

$$Z(t) = [1(t \in D_1), \ldots, 1(t \in D_L)] \qquad \ldots..(5)$$

$$h(t) = Z(t)k \qquad \ldots..(6)$$

❖ ***Additive and Multiplicative Regressors***

Prophet allows us to add additive or multiplicative regressors to the usual linear regression model. The additive model in regression is given as the arithmetic summation of the individual effects of the predictor variables. The additive model can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \qquad \ldots..(7)$$

Likewise, the multiplicative model would be given by the arithmetic multiplication of the predictor variables’ individual effects.

$$y = \beta_0 * x_1^{\beta 1} * x_2^{\beta 2} * \ldots * x_n^{\beta n} * \varepsilon \qquad \ldots..(8)$$

These additive and multiplicative regressors make the linear model more flexible. [8]

Comprehensively, to summarise, Prophet uses a curve fitting approach to forecast. And these forecastings then just extend the curve into the future. The future predicted values can thus be used to detect anomalies by comparing them with the actual values in the given data.

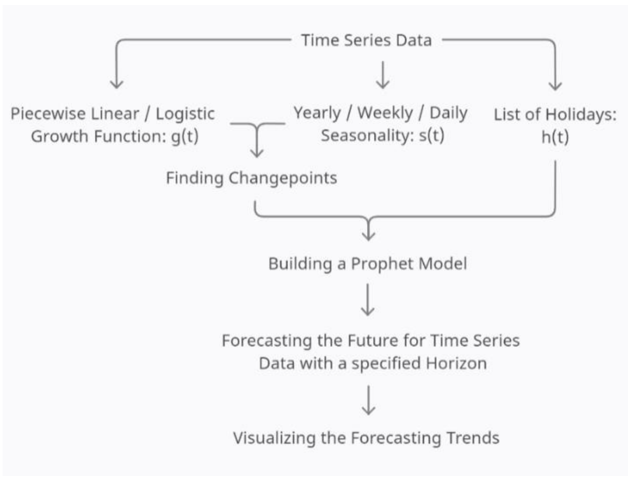

Time Series Data
Piecewise Linear / Logistic Growth Function: g(t)
Yearly / Weekly / Daily Seasonality: s(t)
List of Holidays: h(t)
Finding Changepoints
Building a Prophet Model
Forecasting the Future for Time Series Data with a specified Horizon
Visualizing the Forecasting Trends

Fig. 1: Work Flow Diagram of Prophet Model

➢ *Perquisites of Modelling with Prophet*

Not every type of forecasting can be favoured by Prophet. It excels in forecasting the data that characterises in the following traits:

1. trends with non-linear growth curves
2. history of various trend changes
3. weekly, daily or hourly observations with a good amount of history
4. holidays that although may be irregular, are known in advance

If one isn’t satisfied with the out-comes that are produced by the Prophet Algorithm automatically, the user or analyst can tune some parameters to tweak the forecast in the right direction. By affiliating the Model’s automatic forecasting with analyst-in-the-loop approach, we can handle a varied range of use-cases. []

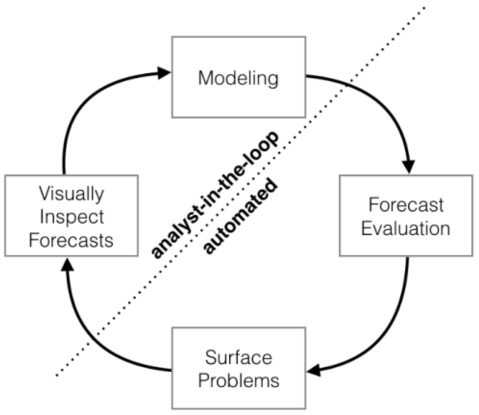


Fig. 2: Modelling with analyst-in-the-loop

Using Facebook Prophet is advantageous for the following reasons:

- Prophet makes fast and accurate predictions all by itself.
- It allows the user to adjust parameters and thus customise the seasonality or growth components in order to improve the forecast.
- Any person, without prior knowledge or experience in analytics or forecasting, can circumvent the modelling with Prophet.

Thus, Prophet helps both experts and non-experts to make high quality forecasts that keep up with demands as well.

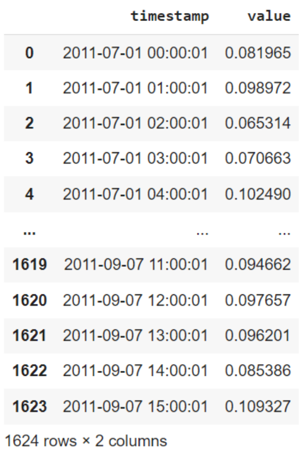
III. METHODOLOGY

1. *<u>Setting up the Environment</u>:*

We shall be using Facebook Prophet to detect the anomalies in time-series data. So, we first install the same. Then we import all the libraries required for the project. Also we have set the default parameters for the graphs throughout the project using matplotlib.rcParams. We are also including the Pandas library for working with dataframes and Plotly Express for data visualisation.

2. *Dataset:*

Anomaly Detection in datasets has broad applications in numerous domains such as medicine, cyber security, machine vision, statistics, neuroscience, law enforcement and financial fraud etc. Due to its wide range of applications, the problem of anomaly detection has been thoroughly researched by industry specialists and academia alike, and several algorithms have been introduced for determining anomalies in problem settings. The NAB (Numenta Anomaly Benchmark) is a standard, open source framework to evaluate and compare various anomaly detection algorithms. It contains a dataset with real-world and artificial, labelled time series data files across various platforms and a unique scoring mechanism that rewards early detection, penalises late or false results, and gives credit for on-line learning. Here, we are using the realAdExchange dataset, which shows online advertisement clicking rates, where the metrics are cost-per-click (CPC) and cost-per thousand impressions (CPM).



	timestamp	value
0	2011-07-01 00:00:01	0.081965
1	2011-07-01 01:00:01	0.098972
2	2011-07-01 02:00:01	0.065314
3	2011-07-01 03:00:01	0.070663
4	2011-07-01 04:00:01	0.102490
...	...	...
1619	2011-09-07 11:00:01	0.094662
1620	2011-09-07 12:00:01	0.097657
1621	2011-09-07 13:00:01	0.096201
1622	2011-09-07 14:00:01	0.085386
1623	2011-09-07 15:00:01	0.109327

1624 rows × 2 columns

Table 1: NAB Dataset

3. *Preprocessing:*

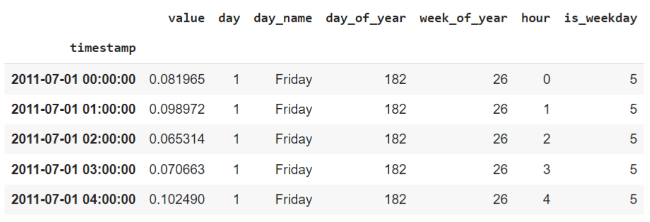
Data preprocessing is a technique that involves transforming raw data into an understandable conformation. Data in the real world is often incomplete, inconsistent, lacking certain behaviours or trends or may contain some errors. Preprocessing helps resolve such issues. It prepares the raw data for further processing. In Machine Learning processes, data preprocessing is critical to encode the dataset in a form that could be interpreted and parsed by the algorithm.

Here, we shall apply 'Feature Engineering' for the same, so as to convert the unrefined raw data into some meaningful features that the model can understand better and provide a better decision boundary. Using Feature Engineering, we shall generate more variables to be considered as factors affecting the model; since we wish to deal with 'Multivariate Time Series Data'.

The data can also be 'Resampled' to change the frequency at which the time series data is reported. This can be used to derive more reliable and consistent data. In this case, we have resampled the data to an 'hourly' basis for our convenience.

```
# Resampling the timeseries to hourly basis
df_hourly = df.resample('H').mean()

# Creating features from date
df_hourly['day'] = [i.day for i in df_hourly.index]
df_hourly['day_name'] = [i.day_name() for i in df_hourly.index]
df_hourly['day_of_year'] = [i.dayofyear for i in df_hourly.index]
df_hourly['week_of_year'] = [i.weekofyear for i in df_hourly.index]
df_hourly['hour'] = [i.hour for i in df_hourly.index]
df_hourly['is_weekday'] = [i.isoweekday() for i in df_hourly.index]
df_hourly.head()
```

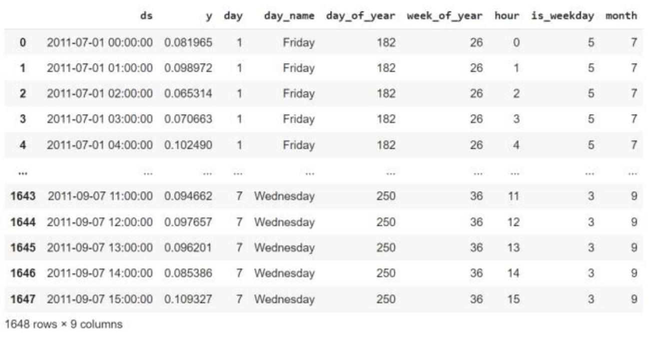


	value	day	day_name	day_of_year	week_of_year	hour	is_weekday
timestamp							
2011-07-01 00:00:00	0.081965	1	Friday	182	26	0	5
2011-07-01 01:00:00	0.098972	1	Friday	182	26	1	5
2011-07-01 02:00:00	0.065314	1	Friday	182	26	2	5
2011-07-01 03:00:00	0.070663	1	Friday	182	26	3	5
2011-07-01 04:00:00	0.102490	1	Friday	182	26	4	5

Table 2: Resampling & Feature Engineering

Any further variables required for 'Regression' can also be added. We shall add a 'month' column, that gives the number of the particular month and Reset the Indexing of the Dataframe.

```
# Adding a month column to the dataframe
df_final['month'] = df_final['ds'].dt.month
df_final
```



	ds	y	day	day_name	day_of_year	week_of_year	hour	is_weekday	month
0	2011-07-01 00:00:00	0.081965	1	Friday	182	26	0	5	7
1	2011-07-01 01:00:00	0.098972	1	Friday	182	26	1	5	7
2	2011-07-01 02:00:00	0.065314	1	Friday	182	26	2	5	7
3	2011-07-01 03:00:00	0.070663	1	Friday	182	26	3	5	7
4	2011-07-01 04:00:00	0.102490	1	Friday	182	26	4	5	7
...	...	...	...	...	...	...	...	...	...
1643	2011-09-07 11:00:00	0.094662	7	Wednesday	250	36	11	3	9
1644	2011-09-07 12:00:00	0.097657	7	Wednesday	250	36	12	3	9
1645	2011-09-07 13:00:00	0.096201	7	Wednesday	250	36	13	3	9
1646	2011-09-07 14:00:00	0.085386	7	Wednesday	250	36	14	3	9
1647	2011-09-07 15:00:00	0.109327	7	Wednesday	250	36	15	3	9

1648 rows × 9 columns

Table 3: Creating anymore required Variables

4. *Data Visualisation:*

For visualising or representing the data, we plot a line graph with: the 'timestamp' on the x-axis and the 'values' on the y-axis. We shall use the 'px.line' method, from Plotly Express, where each data point is represented as a vertex of a polyline mark in two dimensional space.

```
# Visualing the data using plotly
fig = px.line(df_final, x='ds', y='y', hover_data=['day', 'day_name', 'day_of_year', 'week_of_year',
                                                     'hour', 'is_weekday', 'month'], title='Real Ad Exchange')

fig.update_xaxes(
    rangeslider_visible=True,
    rangeselector = dict(
        buttons=list([
                     dict(count=1, label="1y", step="year", stepmode="backward"),
                     dict(count=2, label="3y", step="year", stepmode="backward"),
                     dict(count=3, label="5y", step="year", stepmode="backward"),
                     dict(step="all")
        ])
    )
)

fig.show()
```

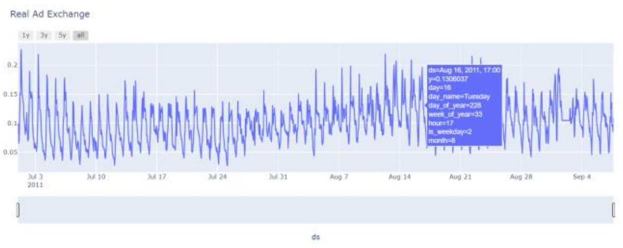


Fig. 3: Plotting 'Line Graph' to represent the Data

5. *Creating the Prophet Model:*

Now, we analyse the graph, so as to select various features as variables for our model.

For eg.: Let us consider how these values change for each hour of the day. Let's do so by grouping the value and the hour data by 'hour' and finding the maximum, minimum and average values.

```
# Checking the Maximum, Minimum, and Average value of 'y' for each hour of the day
df_final[['hour', 'y']].groupby('hour').agg({'y':{'max', 'min', 'mean'}})
```

	y		
	max	mean	min
hour			
0	0.126895	0.076766	0.040555
1	0.115210	0.070469	0.036329
2	0.104638	0.062447	0.029836
3	0.104638	0.055901	0.026843
4	0.136761	0.072798	0.039034
5	0.161497	0.091418	0.051258
6	0.183986	0.099622	0.049853
7	0.191602	0.121641	0.061923
8	0.218536	0.159612	0.104638
9	0.226598	0.141139	0.098608
10	0.181725	0.127746	0.093615
11	0.159902	0.122838	0.094026
12	0.176341	0.122931	0.089616
13	0.190700	0.124328	0.079773
14	0.207095	0.126643	0.074123
15	0.172836	0.123101	0.068836
16	0.166081	0.112077	0.060932
17	0.145224	0.101218	0.051182
18	0.149355	0.095511	0.058831
19	0.145936	0.092093	0.055302
20	0.165860	0.088118	0.050912
21	0.171937	0.086464	0.048883
22	0.183754	0.086494	0.044633
23	0.173424	0.083055	0.043706

Table 4: Analysing Data by Hours

Here, we notice that values normally range somewhere between 0.05 to 0.19. So we define a function to segregate the values that might lie outside this range, and add another column called 'hourly_analysis', which can be used as a regressor or variable for our model.

```
def abnormal_values_of_y_based_on_hourly_analysis(y):
  if y > 0.19 or y < 0.05:
    return 1
  else:
    return 0
```

```
# If value is outside the range 0.05-0.19, it will be regarded as 1
df_final['hourly_analysis'] = df_final['y'].apply(abnormal_values_of_y_based_on_hourly_analysis)
```

```
df_final
```

	ds	y	day	day_name	day_of_year	week_of_year	hour	is_weekday	month	hourly_analysis
0	2011-07-01 00:00:00	0.081965	1	Friday	182	26	0	5	7	0
1	2011-07-01 01:00:00	0.098972	1	Friday	182	26	1	5	7	0
2	2011-07-01 02:00:00	0.065314	1	Friday	182	26	2	5	7	0
3	2011-07-01 03:00:00	0.070663	1	Friday	182	26	3	5	7	0
4	2011-07-01 04:00:00	0.102490	1	Friday	182	26	4	5	7	0
...	...	...	...	...	...	...	...	...	...	...
1643	2011-09-07 11:00:00	0.094662	7	Wednesday	250	36	11	3	9	0
1644	2011-09-07 12:00:00	0.097657	7	Wednesday	250	36	12	3	9	0
1645	2011-09-07 13:00:00	0.096201	7	Wednesday	250	36	13	3	9	0
1646	2011-09-07 14:00:00	0.085386	7	Wednesday	250	36	14	3	9	0
1647	2011-09-07 15:00:00	0.109327	7	Wednesday	250	36	15	3	9	0

1648 rows × 10 columns

Table 5: Final Dataframe

Separating the data into 'train' and 'test'. We notice that the data ranges from 1st of July, 2011 to 7th of August, 2011 on hourly basis. So let's split the dataset from "01-07-2011 to 31-08-2011" into 'train' and the rest into 'test'.

```
train = df_final[(df_final['ds'] >= '2011-07-01') & (df_final['ds'] <= '2011-08-31')]
test = df_final[(df_final['ds'] > '2011-08-31')]
```

```
train.shape
```

```
(1465, 10)
```

```
train
```

	ds	y	day	day_name	day_of_year	week_of_year	hour	is_weekday	month	hourly_analysis
0	2011-07-01 00:00:00	0.081965	1	Friday	182	26	0	5	7	0
1	2011-07-01 01:00:00	0.098972	1	Friday	182	26	1	5	7	0
2	2011-07-01 02:00:00	0.065314	1	Friday	182	26	2	5	7	0
3	2011-07-01 03:00:00	0.070663	1	Friday	182	26	3	5	7	0
4	2011-07-01 04:00:00	0.102490	1	Friday	182	26	4	5	7	0
...	...	...	...	...	...	...	...	...	...	...
1460	2011-08-30 20:00:00	0.075349	30	Tuesday	242	35	20	2	8	0
1461	2011-08-30 21:00:00	0.065388	30	Tuesday	242	35	21	2	8	0
1462	2011-08-30 22:00:00	0.089247	30	Tuesday	242	35	22	2	8	0
1463	2011-08-30 23:00:00	0.081306	30	Tuesday	242	35	23	2	8	0
1464	2011-08-31 00:00:00	0.093861	31	Wednesday	243	35	0	3	8	0

1465 rows × 10 columns

```
test.shape
```

```
(183, 10)
```

```
test
```

	ds	y	day	day_name	day_of_year	week_of_year	hour	is_weekday	month	hourly_analysis
1465	2011-08-31 01:00:00	0.079748	31	Wednesday	243	35	1	3	8	0
1466	2011-08-31 02:00:00	0.066393	31	Wednesday	243	35	2	3	8	0
1467	2011-08-31 03:00:00	0.064618	31	Wednesday	243	35	3	3	8	0
1468	2011-08-31 04:00:00	0.087396	31	Wednesday	243	35	4	3	8	0
1469	2011-08-31 05:00:00	0.109590	31	Wednesday	243	35	5	3	8	0
...	...	...	...	...	...	...	...	...	...	...
1643	2011-09-07 11:00:00	0.094662	7	Wednesday	250	36	11	3	9	0
1644	2011-09-07 12:00:00	0.097657	7	Wednesday	250	36	12	3	9	0
1645	2011-09-07 13:00:00	0.096201	7	Wednesday	250	36	13	3	9	0
1646	2011-09-07 14:00:00	0.085386	7	Wednesday	250	36	14	3	9	0
1647	2011-09-07 15:00:00	0.109327	7	Wednesday	250	36	15	3	9	0

183 rows × 10 columns

Table 6: Splitting the Data into 'train' & 'test'

The Prophet model is an "additive regression model" (a type of regression analysis where the predictor does not take a predetermined form but is constructed according to information derived from the data).

If the time series is more than two cycles long, Prophet will fit the weekly and yearly seasonalities, by default. It also fits the daily seasonality for a sub-daily time-series. One can also add other seasonalities like hourly, monthly, quarterly using the 'add_seasonality' method.

Here, while initiating the model, we shall explicitly set the daily, weekly and yearly seasonalities to True.

The Prophet algorithm gives results according to the 'confidence interval', a range of values you expect your estimate to fall between. This would give us the predictions (yhat) and the upper & lower levels of uncertainty (yhat_upper & yhat_lower). The default confidence interval is set to 80%. However, here, we set it to 95%.

```
model = Prophet(interval_width=0.95, yearly_seasonality=True,
                weekly_seasonality=True, daily_seasonality=True)
```

One can add additional regressors to the model using the 'add_regressor' method. This helps to modulate as to how the forecast would be constructed and to make the prediction process more lucid. The column of the regressor value must be present in both the fitting and prediction dataframes. These extra regressors that are added must be known for both the history and future dates. Hence, it must be something either that has known future values, or that has separately been forecasted elsewhere.

Let's use the 'hourly_analysis' column, that we constructed earlier, and the 'day_of_year' column to be added as regressors in the additive fashion. Any regressor added is, by default, in the additive mode. That is the effect of these regressors would be added to the trend. We shall use the 'month' column in the multiplicative mode, which means that it would multiply the trend.

```
model.add_regressor('hourly_analysis', standardize=False)
model.add_regressor('day_of_year', standardize=False)
model.add_regressor('month', standardize=False, mode='multiplicative')
```

Some other time-series, that has been forecasted by a time-series model, can also be used as a regressor to the model.

We can now train the model using the 'fit' method by passing the Training DataFrame to it.

We shall now use this model to forecast for the future. We also have the 'test' dataset to validate our predictions. The 'test' dataset, as we saw earlier, includes 183 rows and 10 columns. Therefore, we shall forecast the next 183 periods using the 'make_future_dataframe' method.

We shall create a 'future' dataframe that only consists of dates of the 'test data' as timestamps. According to our 'test' data, our dataframe must include hourly frequency. By default, the method creates predictions on daily basis. Hence, we specify the frequency to predict for each hour.

```
future = model.make_future_dataframe(periods = 183, freq = 'H')
# Period is set to 183 as the 'test' data has 183 rows.
# Hence we tell the model to forecast for the next 183 predictions.
future.tail()
```

	ds
1643	2011-09-07 11:00:00
1644	2011-09-07 12:00:00
1645	2011-09-07 13:00:00
1646	2011-09-07 14:00:00
1647	2011-09-07 15:00:00

Table 7: Forecasting the Future Dataframe

In multivariate forecasting, when we are predicting our target data points, we must know the future values of the regressor that we have given. Hence we add the regressor columns containing their future values from the final dataframe to our future dataframe.

```
future['hourly_analysis'] = df_final['hourly_analysis']
future['day_of_year'] = df_final['day_of_year']
future['month'] = df_final['month']
```

Any missing values in this 'future' dataframe can be filled using 'Forward Fill' with the help of the 'fillna' method from pandas.

```
# Using the fillna() method from pandas, we will 'forward fill'
# i.e. ffill the Missing Value
future = future.fillna(method = 'ffill')
```

```
future
```

	ds	hourly_analysis	day_of_year	month
0	2011-07-01 00:00:00	0	182	7
1	2011-07-01 01:00:00	0	182	7
2	2011-07-01 02:00:00	0	182	7
3	2011-07-01 03:00:00	0	182	7
4	2011-07-01 04:00:00	0	182	7
...	...	...	...	...
1643	2011-09-07 11:00:00	0	250	9
1644	2011-09-07 12:00:00	0	250	9
1645	2011-09-07 13:00:00	0	250	9
1646	2011-09-07 14:00:00	0	250	9
1647	2011-09-07 15:00:00	0	250	9

1648 rows × 4 columns

Table 8: Added the Regressor Columns to the Future Dataframe

Now, we take the model and try to predict the target values for these dates from the 'future' dataframe to get the forecast. This forecast will then include:

1. *yhat* - Predicted value for the particular time
2. *yhat_lower* - Lower value of the confidence interval
3. *yhat_upper* - Upper value of the confidence interval

```
forecast = model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
1643	2011-09-07 11:00:00	0.194851	0.164599	0.224416
1644	2011-09-07 12:00:00	0.192848	0.162800	0.224523
1645	2011-09-07 13:00:00	0.196892	0.166847	0.225783
1646	2011-09-07 14:00:00	0.200424	0.169981	0.233106
1647	2011-09-07 15:00:00	0.197537	0.166180	0.229367

Table 9: Final Predictions

6. *Analysing Forecasted Data:*

Let us now plot the actual values and the predicted values together to check the difference between the two. We'll use the 'concat' function from the Pandas library to do so.

```
pd.concat([df_final.set_index('ds')['y'],
           forecast.set_index('ds')['yhat']], axis=1).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e447f1990>
```

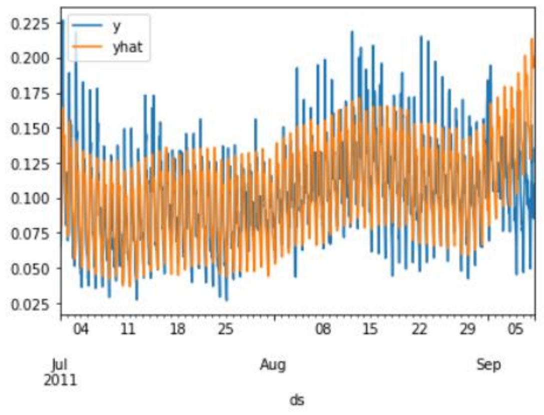


Fig. 4: Plotting the difference between Actual & Predicted values

Let us also visualise our 'forecasted' values to better understand these predictions.

```
fig1 = model.plot(forecast)
```

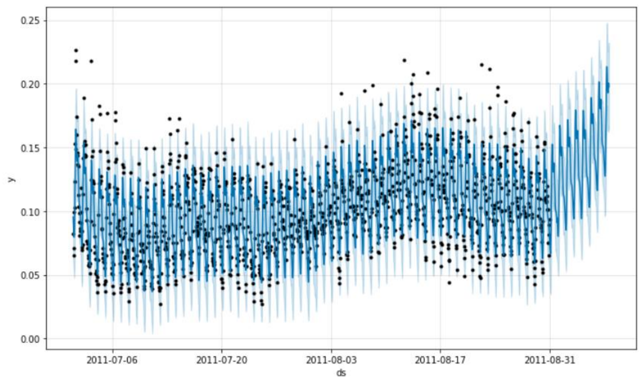


Fig. 5: Visualising the Forecasted Values

Here, the 'black points' represent the Actual values, whereas the 'dark blue' part of the graph shows the Predicted data points. The upper and lower shades of 'light blue' represent the Upper and Lower Confidence Intervals.

We can also check out the seasonality components of this forecasted time-series and visualise its trends.

```
fig2 = model.plot_components(forecast)
```

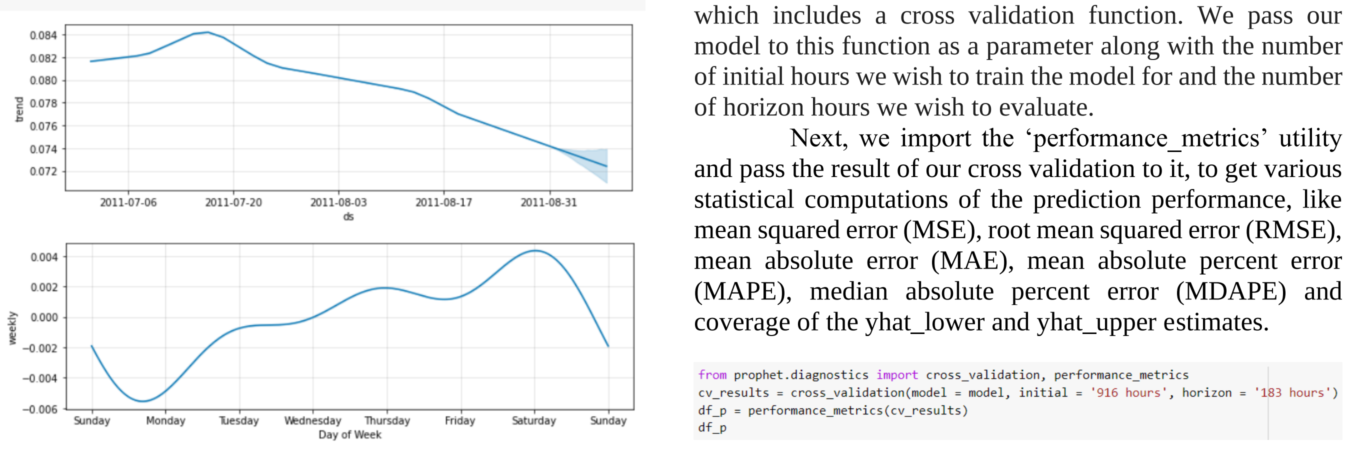


Fig. 6: Component Analysis of Forecasted Data

The first graph shows the trend in the predicted values. The data shows a rise for the first 15 days and then shows a gradual decline. The second, third and the fourth graphs represent the weekly, yearly and daily trends respectively. And finally, the last two graphs show the effects of the 'additive' and 'multiplicative' regressors.

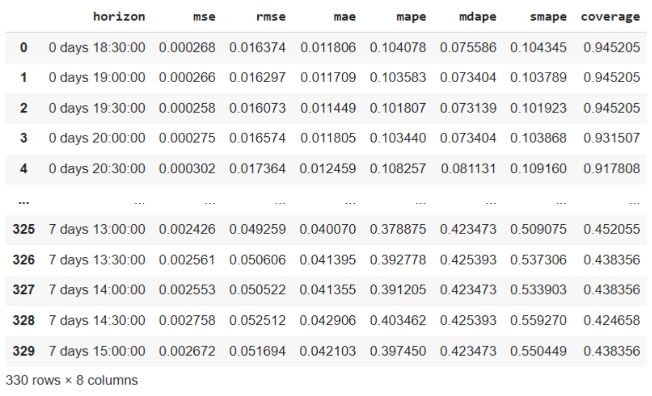
7. *Cross Validation*:

Now, we don't know how efficient our model is. We need some kind of a metric for measuring the same. This is where 'cross validation' comes into the picture. Cross Validation is a model validation technique where we divide the dataset into two segments; one to train the model and the other to assess our performance.

Facebook (Meta) provides a diagnostic package which includes a cross validation function. We pass our model to this function as a parameter along with the number of initial hours we wish to train the model for and the number of horizon hours we wish to evaluate.

Next, we import the 'performance_metrics' utility and pass the result of our cross validation to it, to get various statistical computations of the prediction performance, like mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), mean absolute percent error (MAPE), median absolute percent error (MDAPE) and coverage of the yhat_lower and yhat_upper estimates.

```
from prophet.diagnostics import cross_validation, performance_metrics
cv_results = cross_validation(model = model, initial = '916 hours', horizon = '183 hours')
df_p = performance_metrics(cv_results)
df_p
```



	horizon	mse	rmse	mae	mape	mdape	smape	coverage
0	0 days 18:30:00	0.000268	0.016374	0.011806	0.104078	0.075586	0.104345	0.945205
1	0 days 19:00:00	0.000266	0.016297	0.011709	0.103583	0.073404	0.103789	0.945205
2	0 days 19:30:00	0.000258	0.016073	0.011449	0.101807	0.073139	0.101923	0.945205
3	0 days 20:00:00	0.000275	0.016574	0.011805	0.103440	0.073404	0.103868	0.931507
4	0 days 20:30:00	0.000302	0.017364	0.012459	0.108257	0.081131	0.109160	0.917808
...	...	...	...	...	...	...	...	...
325	7 days 13:00:00	0.002426	0.049259	0.040070	0.378875	0.423473	0.509075	0.452055
326	7 days 13:30:00	0.002561	0.050606	0.041395	0.392778	0.425393	0.537306	0.438356
327	7 days 14:00:00	0.002553	0.050522	0.041355	0.391205	0.423473	0.533903	0.438356
328	7 days 14:30:00	0.002758	0.052512	0.042906	0.403462	0.425393	0.559270	0.424658
329	7 days 15:00:00	0.002672	0.051694	0.042103	0.397450	0.423473	0.550449	0.438356

330 rows × 8 columns

Table 10: Performance Metrics of our Model

The performance metrics of our cross validation can be visualised using 'plot_cross_validation_metric' function. Let's plot the Mean Absolute Percent Error (MAPE)

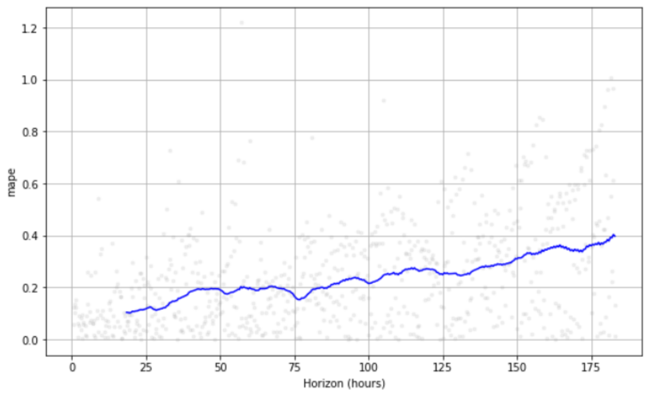


Fig. 7: Plotting MAPE of our Cross Validation

The dots represent the absolute percent error for each prediction made in our cross validation result, and the blue line shows the mean of all these absolute percent errors (MAPE).

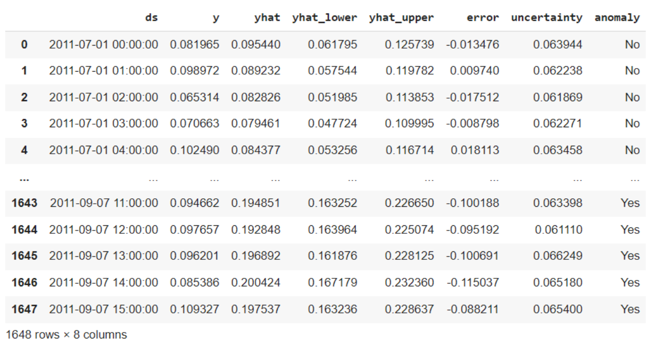
8. *Detecting Anomalies*:

Lastly, let's identify the anomalous data points in the dataset. The difference between the true values and the prophesied values would give us the errors. Similarly, the difference between the upper and lower confidence intervals would give us a value of the uncertainty range. We shall calculate the same for further processing.

```
result = pd.concat([df_final[['ds', 'y']],
                    forecast[['yhat', 'yhat_lower', 'yhat_upper']]], axis=1)
result['error'] = result['y'] - result['yhat']
result['uncertainty'] = result['yhat_upper'] - result['yhat_lower']
```

The error can be both positive and negative, hence we first take its absolute value and then compare it with the uncertainty. For an anomalous entity, the value of the absolute error would be greater than the uncertainty value. We can mark such a point as an Anomaly.

```
import numpy as np
# Here if the absolute of the error is found to be greater than the uncertainty then we consider it as an anomaly
result['anomaly'] = result.apply(lambda x: 'Yes' if(np.abs(x['error']) > x['uncertainty']) else 'No', axis = 1)
result
```



	ds	y	yhat	yhat_lower	yhat_upper	error	uncertainty	anomaly
0	2011-07-01 00:00:00	0.081965	0.095440	0.061795	0.125739	-0.013476	0.063944	No
1	2011-07-01 01:00:00	0.098972	0.089232	0.057544	0.119782	0.009740	0.062238	No
2	2011-07-01 02:00:00	0.065314	0.082826	0.051985	0.113853	-0.017512	0.061869	No
3	2011-07-01 03:00:00	0.070663	0.079461	0.047724	0.109995	-0.008798	0.062271	No
4	2011-07-01 04:00:00	0.102490	0.084377	0.053256	0.116714	0.018113	0.063458	No
...	...	...	...	...	...	...	...	...
1643	2011-09-07 11:00:00	0.094662	0.194851	0.163252	0.226650	-0.100188	0.063398	Yes
1644	2011-09-07 12:00:00	0.097657	0.192848	0.163964	0.225074	-0.095192	0.061110	Yes
1645	2011-09-07 13:00:00	0.096201	0.196892	0.161876	0.228125	-0.100691	0.066249	Yes
1646	2011-09-07 14:00:00	0.085386	0.200424	0.167179	0.232360	-0.115037	0.065180	Yes
1647	2011-09-07 15:00:00	0.109327	0.197537	0.163236	0.228637	-0.088211	0.065400	Yes

1648 rows × 8 columns

Table 11: Resultant Data Frame marked with the Anomalies

Ultimately, let's envision the Anomalies in our data by generating a scatter plot with the timestamp on the x-axis and the values of the data points on the y-axis. Here, we have colour coded the Anomalies with 'red', whereas the typical data values are marked with 'blue'.

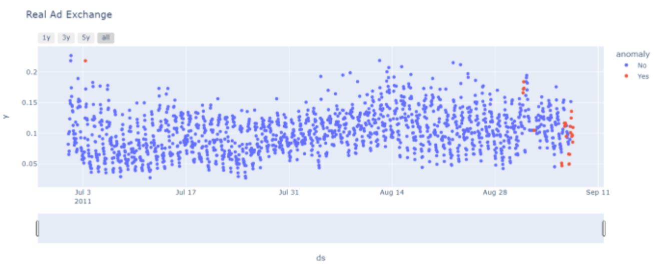


Fig. 8: Visualisation of Anomalies

IV. Conclusion And Further Work

Identifying anomalies depends on the baselining of our given data, numerous business scenarios, seasonalities and the user's requirements. These help to identify which and what type of data is not crucial for our needs. In a majority of these professions, these points are quite functional to drive some strategies or change the business model.

The fundamental purpose of this research was to use Facebook's (Meta) Prophet Algorithm to detect anomalies in a Multivariate Time-Series Dataset. Thus, we created a forecasting model using Prophet and cross-verified our forecasted values with the actual values to pick out the Anomalies. We managed to accomplish our goal with the help of the modules and features offered by the 'prophet' library. We also explored and analysed the NAB Real Ad Exchange datasets.

In our further studies, we would like to work with more datasets of similar type and also wish to compare Prophet Algorithm with other techniques like PyCaret and PyOD for detecting anomalies in time-series data.

V. References

[1] Time Series Facebook Prophet Model and Python for COVID-19 Outbreak Prediction. Khayyat, Mashael, Laabidi, Kaouther, Almalki, Nada, Al-Zahrani, Maysoon. Computers, Materials, & Continua ; 67(3):3781-3793, 2021. Artigo | ProQuest Central | ID: covidwho-1112965

[2] Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2; Facebook, Menlo Park, California, United States. doi:10.7287/peerj.preprints.3190v2

[3] Master thesis; December 2019. Author: Mohammad Braei. Technische Universität Darmstadt; Advisor: Prof. Max Mühlhäuser, Dr. Sebastian Wagner. doi:10.13140/RG.2.2.17687.80801

[4] Anomaly Detection Techniques Causes and Issues; G. Sandhya Madhuri, Dr. M. Usha Rani; International Journal of Engineering & Technology. doi:10.14419/ijet.v7i3.24.22791

[5] (2008) Time Series. In: The Concise Encyclopedia of Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1_401

[6] Almazrouee, A. I., Almeshal, A. M., Almutairi, A. S., Alenezi, M. R., & Alhajeri, S. N. (2020). Long-Term Forecasting of Electrical Loads in Kuwait Using Prophet and Holt–Winters Models. Applied Sciences, 10(16), 5627. doi:10.3390/app10165627

[7] Forecasting: Principles and Practice; Hyndman, R. & Athanasopoulos, G.; Econometrics & Business Statistics; 2021, 3rd ed. OTexts.

[8] Fahrmeir, Ludwig; Gieger, Christian; Klinger, Artur (1995): Additive, Dynamic and Multiplicative Regression. Collaborative Research Center 386, Discussion Paper 1. doi:10.5282/ubm/epub.1405

[9] Sujata Dash, Chinmay Chakraborty, Sourav K. Giri, Subhendu Kumar Pani; Intelligent computing on time-series data analysis and prediction of COVID-19 pandemics, Pattern Recognition Letters, Volume 151, 2021. https://doi.org/10.1016/j.patrec.2021.07.027.