*Machine Learning Project*

# Time Series Forecasting

Justine Diliberto
Anna Nikiforovskaja
Cindy Pereira

May 2021

## Contents

# 1 Introduction

This report aims at understanding Times Series Forecasting through an implementation with the Python library Prophet created by Facebook.
This first part will be focusing on Time Series Forecasting. Then, the dataset used for the implementation will be presented. The algorithm of Prophet will be explained in the third part. Finally, the Python implementation will be demonstrated.

## 1.1 What is Time Series Forecasting (TSF)?

To begin with, examining the phrase TSF gives many hints on what this method is about. "Forecasting" means it is a prediction tool. "Time" suggests the prediction will be over time. "Series" is a synonym for sequences or sets, and is applied to "Time".

TSF consists in the prediction of future events, as explained in *Machine Learning for Time Series Data* [4]. This prediction is based on the analysis of data through time.
As said in *Machine Learning for Time Series Forecasting* [3], several factors are investigated in the data while implementing a TSF model, such as trends, seasonal patterns, cyclic patterns, or regularity.
TSF can be used in many domains including weather forecasting, geology, astronomy, finance, marketing, signal processing or statistics (see [5]).
The predictions can be computed with ML algorithms, like Artificial Neural Networks, Support Vector Machines, Fuzzy logic, Gaussian Processes and Hidden Markov Models.

## 1.2 Why is TSF interesting for Machine Learning?

It is interesting for students in Machine Learning to learn TSF, because predictions over time is a comprehensible concept, even for beginners. Moreover, it is a good way to be acquainted with some complex Machine Learning methods, as said in *Generate Quick and Accurate Time Series Forecasts using Facebook's Prophet* [2].
As said before, many domains can use TSF. Thus, it is a very handy technique, and considered a must-know by many data scientists.
In addition to this, datasets for TSF include a lot of instances, and handling Big Data can be a good challenge.

# 2 Dataset

## 2.1 Description of the dataset

We base our implementation on the Avocado Prices[1] dataset. Data have been collected in May of 2018 from the Hass Avocado Board[2] which supplies all kind of information about avocados and describes the data this way:

> "The table below represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass, club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table."

The dataset is composed of 13 columns and 18249 entries (see the two first entries in Figure 1). The columns are as follow:

[1] Available here: https://www.kaggle.com/neuromusic/avocado-prices#avocado.csv
[2] https://hassavocadoboard.com/

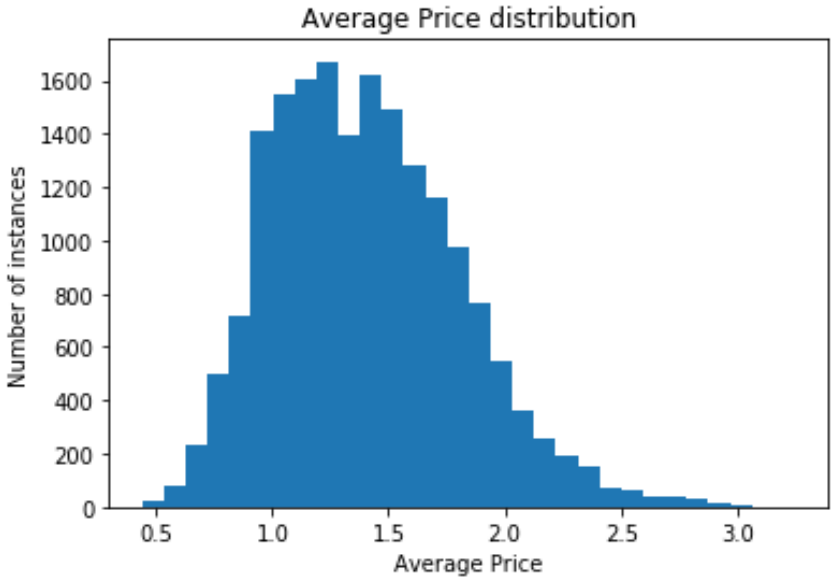
| | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags | XLarge Bags | type | year | region |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 12/27/2015 | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | 2015 | Albany |
| 1 | 12/20/2015 | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 | 0.0 | conventional | 2015 | Albany |

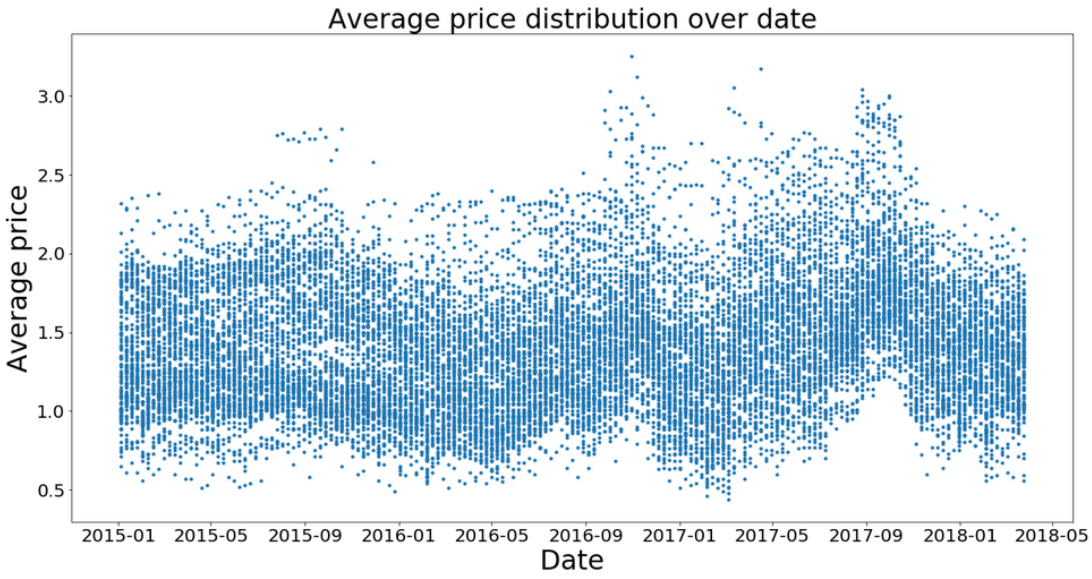
- **Date**: The date of the observation
- **AveragePrice**: The average price of a single avocado
- **Total Volume**: Total volume of avocados sold
- **4046**: Total number of avocados with PLU 4046 sold
- **4225**: Total number of avocados with PLU 4225 sold
- **4770**: Total number of avocados with PLU 4770 sold
- **Total Bags**: Total number of bags sold
- **Small Bags**: Total number of small bags sold
- **Large Bags**: Total number of large bags sold
- **XLarge Bags**: Total number of extra large bags sold
- **type**: Convention or Organic
- **year**: The year
- **region**: The city or region of the observation

In our implementation, we want to predict the average price of a single avocado based on all the information collected.

## 2.2 Basic statistical analysis

The average price distribution can be seen in Figure 2. The mean of the average price is 1.41. The median is 1.37 and the standard deviation is 0.40. The mode is 1.15 and there are 202 instances of an average price of 1.15.






The Figure 3 shows the distribution of the average price of a single avocado over the date from January of 2015 to May of 2018. We can observe that the average price follows a periodic scheme, corresponding to seasonality.

# 3 Algorithm: Prophet

As described in the Facebook paper [1], the Prophet forecasting model is a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

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

where

- $g(t)$ is a trend function, it is a piecewise linear function or a logistic growth curve for modelling non-periodic changes in time series. By default in Prophet there is a piecewise linear function, however there is a parameter that enables the usage of a logistic growth curve.
- $s(t)$ represents periodic changes (e.g. weekly/yearly seasonality).
- $h(t)$ represents effects of holidays (user provided) with irregular schedules.
- $\epsilon_t$ is an error term which accounts for any unusual changes not accommodated by the model. In the model they make an assumption that this error is distributed normally.

In the following sections we will explain what is actually inside each of the main components and how the model is trained.

## 3.1 Trend

In the Prophet library, they have implemented two different trend functions.

### 3.1.1 Logistic growth function

Non-linear growth in Prophet is represented as a following kind of generalized logistic growth curve.

$$g(t) = \frac{C(t)}{1 + \exp\left(-(k + a(t)^T\delta)(t - (m + a(t)^T\gamma))\right)}$$

It is basically a piecewise logistic growth function where

- $C(t)$ is a time-varying capacity. For example, a growth of a specific website users is limited with the number of people who currently have access to the Internet. However, the last number can also change with the time, so that is why we need a capacity, which is changing with time.
- $k$ is the growth rate.
- $m$ is an offset parameter.
- $a(t), \delta$ and $\gamma$ are here to represent the piecewise structure itself, and all result in vectors of length $S$, considering we have $S$ change points at times $s_i$. More specifically they are defined as the following:
  - $a_j(t) = \begin{cases} 1, \text{ if } t \geq s_j \\ 0, \text{ otherwise.} \end{cases}$
  - $\delta_j$ is the change in rate that occurs at time $s_j$.
  - $\gamma$ is the parameter to adjust $m$ so that the endpoints of segments were connected.

#### 3.1.2 Piecewise linear function

The other introduced trend model by Prophet assumes that forecasting problems do not exhibit saturating growth. And the trend function now looks as a piecewise linear function

$$g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma),$$

where, as before, $k$ is the growth rate, $\delta$ has the rate adjustments, $m$ is the offset parameter, and $\gamma_j$ is set to $-s_j\delta_j$ to make the function continuous.

### 3.2 Seasonality

Real world data can have multi-period seasonality, so they have introduced a seasonality term in Prophet. This seasonality term is modeled with Fourier series. Specifically, we have

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

where $P$ is the regular period we expect the time series to have. If we consider $a_n$ and $b_n$ as one vector $\beta$ and sine and cosine functions as one vector $X(t)$, the seasonal term can be rewritten as $s(t) = X(t)\beta$, where $X(t)$ can be calculated and $\beta$ is a learnable parameter. In the generative model $\beta$ is assumed to be distributed normally with $\mathcal{N}(0, \sigma^2)$

### 3.3 Holidays

A user can pass to the Prophet model holidays and events, as in real life these days can influence the data.

For each holiday $i$, let $D_i$ be the set of past and future dates for that holiday. They add an indicator function representing whether time $t$ is during holiday $i$, and assign each holiday a parameter $\kappa_i$ which is the corresponding change in the forecast. This is done in a similar way as seasonality by generating a matrix of regressors $Z(t) = [1(t \in D_1), ..., 1(t \in D_L)]$ and taking

$$h(t) = Z(t)\kappa.$$

As with seasonality, they use a prior $\kappa \sim \mathcal{N}(0, \nu^2)$.

### 3.4 Model fitting

The parameters of the different terms and an error term are fitted with Stan's L-BFGS. It is a Limited-memory BFGS (Broyden–Fletcher–Goldfarb–Shanno) which uses an estimate of the inverse Hessian matrix to steer its search through variable space.

As the model is divided onto different terms the scientist can look into what each term consist of after fitting and make appropriate changes to improve the fitting.

## 4 Python implementation of Time Series Forecast using Prophet

Because of low-memory, we only managed to calculate everything using one thread. These three exports allow us to restrict the algorithms of Numpy and other libraries to one thread.

[6]:
```
!export MKL_NUM_THREADS=1
!export NUMEXPR_NUM_THREADS=1
!export OMP_NUM_THREADS=1
```

Prophet is a Facebook library.

[8]:
```
from fbprophet import Prophet
from fbprophet.plot import plot_plotly
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.offline as py
py.init_notebook_mode()
%matplotlib inline
```

The dataset used is the Avocado dataset, which is composed of 18249 rows and 13 columns.

The aim of the implementation is to predict the price of avocados over time, based on several features.

[3]:
```
df = pd.read_csv('avocado.csv', index_col=0)
```

[46]:
```
df
```

[46]:

| | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | \ |
|---|---|---|---|---|---|---|---|
| 0 | 2015-12-27 | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | |
| 1 | 2015-12-20 | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | |
| 2 | 2015-12-13 | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | |
| 3 | 2015-12-06 | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | |
| 4 | 2015-11-29 | 1.28 | 51039.60 | 941.48 | 43838.39 | 75.78 | |
| .. | ... | ... | ... | ... | ... | ... | |
| 7 | 2018-02-04 | 1.63 | 17074.83 | 2046.96 | 1529.20 | 0.00 | |
| 8 | 2018-01-28 | 1.71 | 13888.04 | 1191.70 | 3431.50 | 0.00 | |
| 9 | 2018-01-21 | 1.87 | 13766.76 | 1191.92 | 2452.79 | 727.94 | |
| 10 | 2018-01-14 | 1.93 | 16205.22 | 1527.63 | 2981.04 | 727.01 | |
| 11 | 2018-01-07 | 1.62 | 17489.58 | 2894.77 | 2356.13 | 224.53 | |

| | Total Bags | Small Bags | Large Bags | XLarge Bags | type | year | \ |
|---|---|---|---|---|---|---|---|
| 0 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | 2015 | |

```
1      9505.56     9408.07       97.49          0.0  conventional  2015
2      8145.35     8042.21      103.14          0.0  conventional  2015
3      5811.16     5677.40      133.76          0.0  conventional  2015
4      6183.95     5986.26      197.69          0.0  conventional  2015
..         ...         ...         ...          ...           ...   ...
7     13498.67    13066.82      431.85          0.0       organic  2018
8      9264.84     8940.04      324.80          0.0       organic  2018
9      9394.11     9351.80       42.31          0.0       organic  2018
10    10969.54    10919.54       50.00          0.0       organic  2018
11    12014.15    11988.14       26.01          0.0       organic  2018

              region
0             Albany
1             Albany
2             Albany
3             Albany
4             Albany
..               ...
7   WestTexNewMexico
8   WestTexNewMexico
9   WestTexNewMexico
10  WestTexNewMexico
11  WestTexNewMexico

[18249 rows x 13 columns]
```

[5]:
```
df.describe()
```

[5]:
```
        AveragePrice  Total Volume          4046          4225          4770  \
count   18249.000000  1.824900e+04  1.824900e+04  1.824900e+04  1.824900e+04
mean        1.405978  8.506440e+05  2.930084e+05  2.951546e+05  2.283974e+04
std         0.402677  3.453545e+06  1.264989e+06  1.204120e+06  1.074641e+05
min         0.440000  8.456000e+01  0.000000e+00  0.000000e+00  0.000000e+00
25%         1.100000  1.083858e+04  8.540700e+02  3.008780e+03  0.000000e+00
50%         1.370000  1.073768e+05  8.645300e+03  2.906102e+04  1.849900e+02
75%         1.660000  4.329623e+05  1.110202e+05  1.502069e+05  6.243420e+03
max         3.250000  6.250565e+07  2.274362e+07  2.047057e+07  2.546439e+06

         Total Bags    Small Bags    Large Bags    XLarge Bags          year
count  1.824900e+04  1.824900e+04  1.824900e+04   18249.000000  18249.000000
mean   2.396392e+05  1.821947e+05  5.433809e+04    3106.426507   2016.147899
std    9.862424e+05  7.461785e+05  2.439660e+05   17692.894652      0.939938
min    0.000000e+00  0.000000e+00  0.000000e+00       0.000000   2015.000000
25%    5.088640e+03  2.849420e+03  1.274700e+02       0.000000   2015.000000
50%    3.974383e+04  2.636282e+04  2.647710e+03       0.000000   2016.000000
75%    1.107834e+05  8.333767e+04  2.202925e+04     132.500000   2017.000000
max    1.937313e+07  1.338459e+07  5.719097e+06  551693.650000   2018.000000
```

[9]:
```
from sklearn.preprocessing import LabelEncoder
```

We want to convert all values of features into numeric values.

We use Label Encoder to do that.

[11]:
```
le = LabelEncoder()
df.iloc[:,10] = le.fit_transform(df.iloc[:,10])
df.shape
```

[11]:
```
(18249, 13)
```

We split the price (Y) from the other features (X) so that we can train our model.

[12]:
```
X = df[['Date', 'Total Volume', '4046', '4225', '4770', 'Small Bags', 'Large Bags',␣
 ↪'XLarge Bags', 'type']]
y = df.iloc[:,1]
```

We convert the string-formatted dates into Numpy Date format to allow Prophet understanding the date.

[13]:
```
train = pd.DataFrame()
train['ds'] = pd.to_datetime(X['Date'])
train['y'] = y
train.shape
```

[13]:
```
(18249, 2)
```

Our train dataset is composed of average prices and corresponding dates.

[15]:
```
train
```

[15]:

| | ds | y |
|---|---|---|
| 0 | 2015-12-27 | 1.33 |
| 1 | 2015-12-20 | 1.35 |
| 2 | 2015-12-13 | 0.93 |
| 3 | 2015-12-06 | 1.08 |
| 4 | 2015-11-29 | 1.28 |
| .. | ... | ... |
| 7 | 2018-02-04 | 1.63 |
| 8 | 2018-01-28 | 1.71 |
| 9 | 2018-01-21 | 1.87 |
| 10 | 2018-01-14 | 1.93 |
| 11 | 2018-01-07 | 1.62 |

[18249 rows x 2 columns]

We create the Prophet model with the basic parameters and we fit it with the training data.

[72]:
```
prophet_basic = Prophet()
prophet_basic.fit(train)
```

[72]:
```
<fbprophet.forecaster.Prophet at 0x7f93e8f5a760>
```

We complete the dates column with the next 300 days.

The whole history is not displayed in future because make_future_dataframe method removes multiple occurrences of the same date.

[80]:
```
future = prophet_basic.make_future_dataframe(periods=300)
future
```

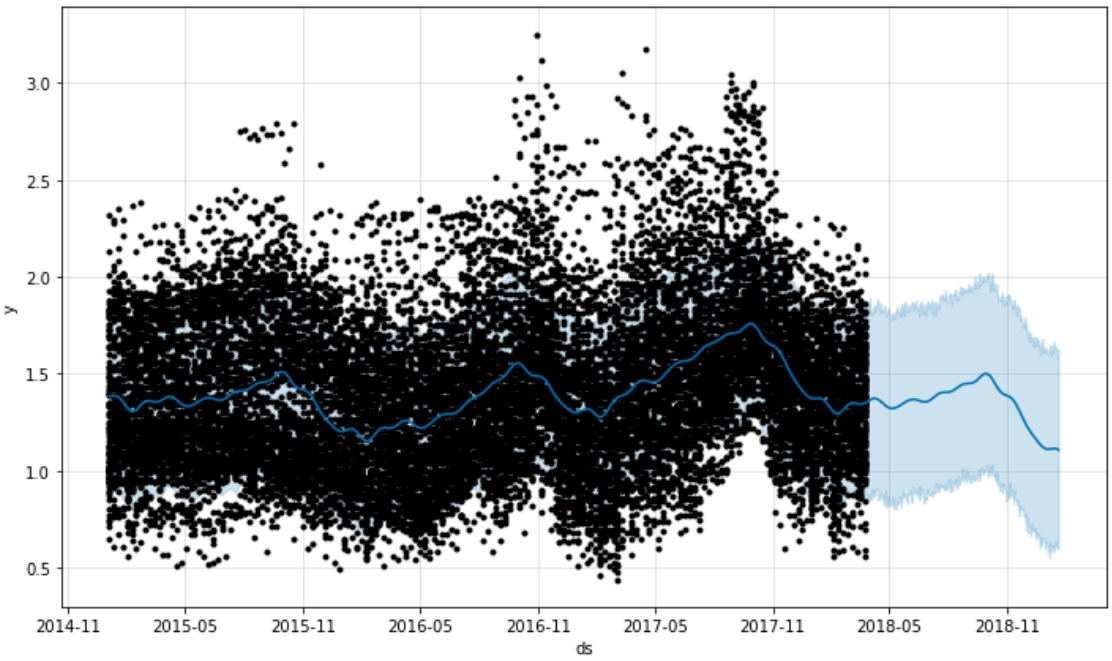
[80]:
```
            ds
0   2015-01-04
1   2015-01-11
2   2015-01-18
3   2015-01-25
4   2015-02-01
..         ...
464 2019-01-15
465 2019-01-16
466 2019-01-17
467 2019-01-18
468 2019-01-19

[469 rows x 1 columns]
```

We predict the future average prices over the dates created in the `future` dataframe using the basic Prophet model.

[81]:
```
forecast = prophet_basic.predict(future)
```

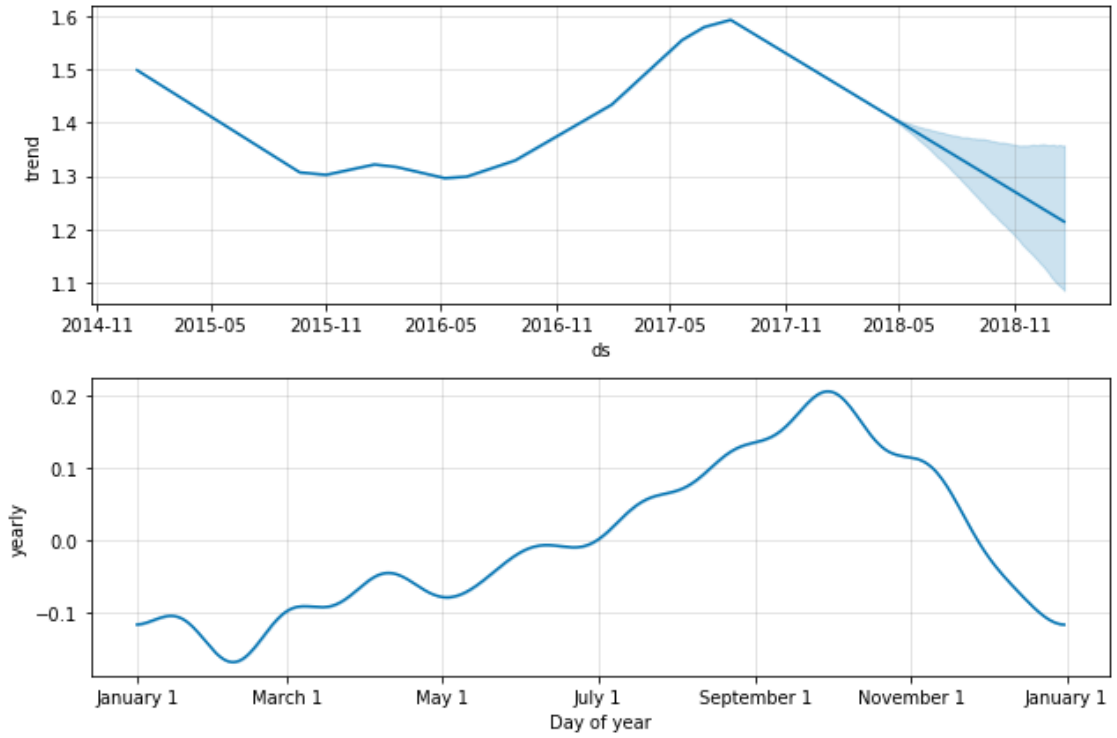
We plot the forecast.
The black dots correspond to the real prices for each past date.
The light blue area represents the prediction for the future prices.
A trend line is displayed in blue.

[82]:
```
fig1 = prophet_basic.plot(forecast)
```




We plot the components of the prediction which are the trend and seasonality.

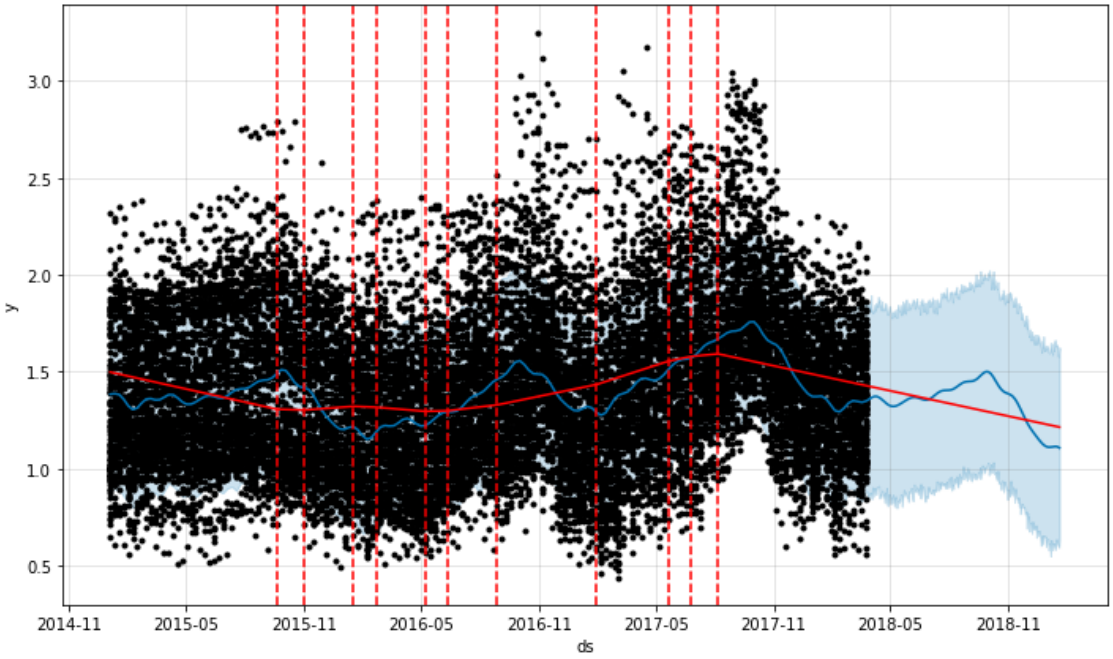
[83]:
```
fig1 = prophet_basic.plot_components(forecast)
```




[84]:
```
from fbprophet.plot import add_changepoints_to_plot
```

We display some changepoints on the results graph from the previous prediction. By default, there are 25 changepoints.

[85]:
```
fig = prophet_basic.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), prophet_basic, forecast)
```




We print the changepoints to check them.

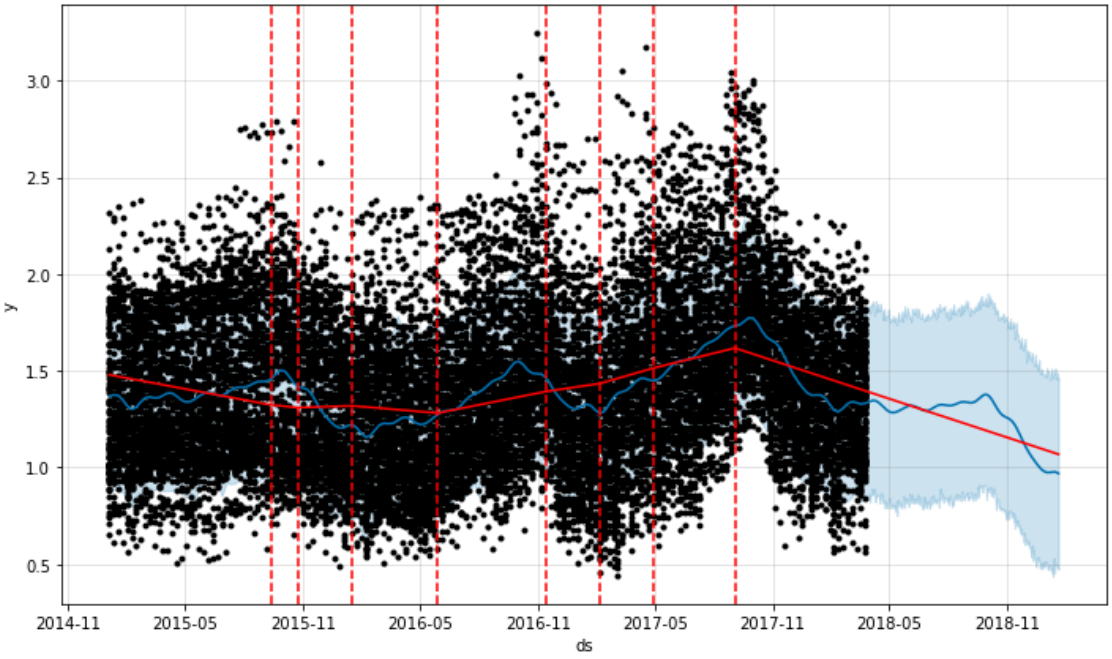
[86]:
```
prophet_basic.changepoints
```

[86]:
```
584      2015-02-08
1168     2015-03-15
1752     2015-04-26
2336     2015-05-31
2920     2015-07-12
3504     2015-08-16
4087     2015-09-20
4671     2015-11-01
5255     2015-12-06
5839     2016-01-17
6423     2016-02-21
7007     2016-03-27
7591     2016-05-08
8175     2016-06-12
8759     2016-07-24
9343     2016-08-28
9927     2016-10-02
10511    2016-11-13
11094    2016-12-18
11678    2017-01-29
12262    2017-03-05
12846    2017-04-09
13430    2017-05-21
14014    2017-06-25
14598    2017-08-06
```

```
Name: ds, dtype: datetime64[ns]
```

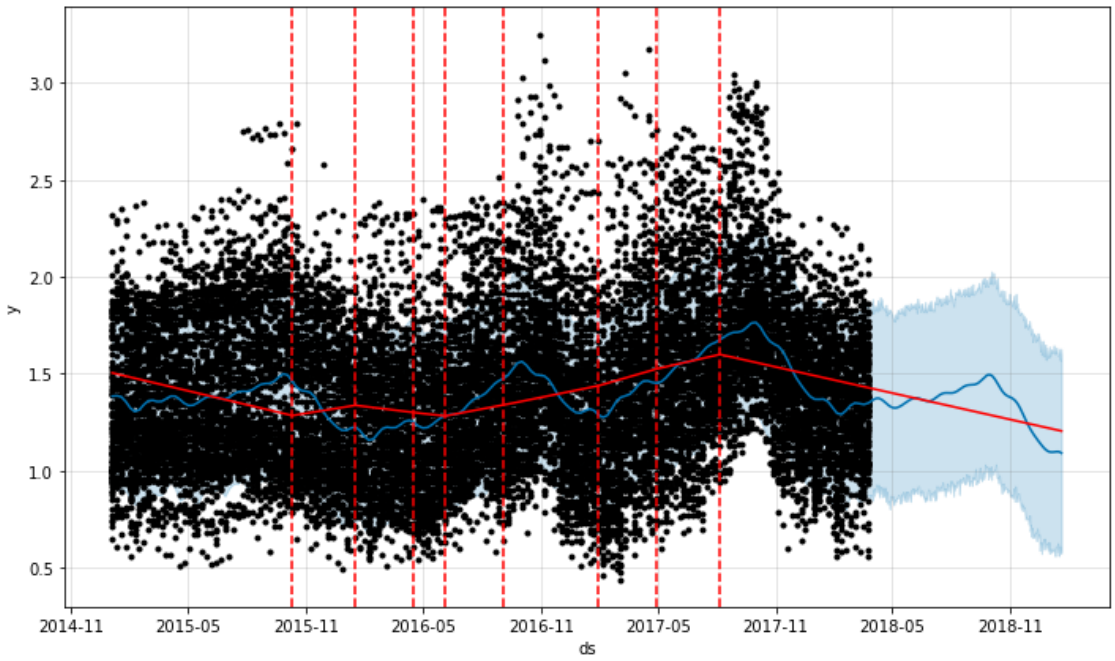
We create a Prophet model using changepoints at a range of 0.9 (= using 90% of the history).
We fit this new model on the training data, predict the future prices, and we plot the forecast.

[93]:
```
pro_change = Prophet(changepoint_range=0.9)
forecast = pro_change.fit(train).predict(future)
fig = pro_change.plot(forecast);
a = add_changepoints_to_plot(fig.gca(), pro_change, forecast)
```



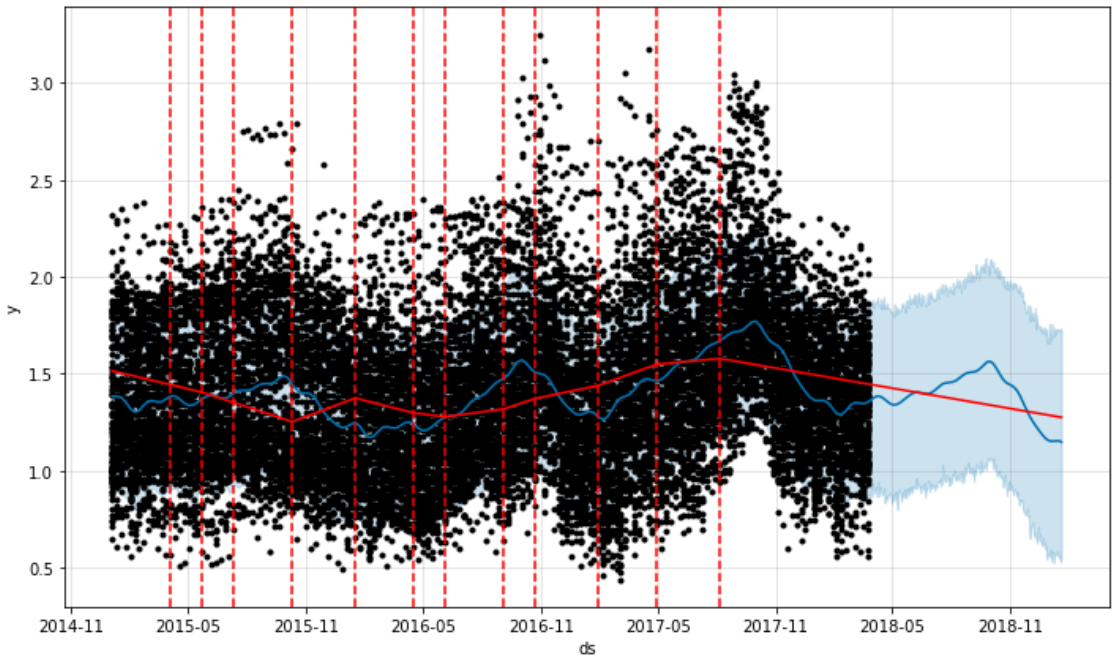

We create a Prophet model using 20 changepoints and a yearly seasonality (to take into account seasonal events such as holidays).
We fit this new model on the training data, predict the future prices, and we plot the forecast.

[94]:
```
pro_change = Prophet(n_changepoints=20, yearly_seasonality=True)
forecast = pro_change.fit(train).predict(future)
fig = pro_change.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), pro_change, forecast)
```



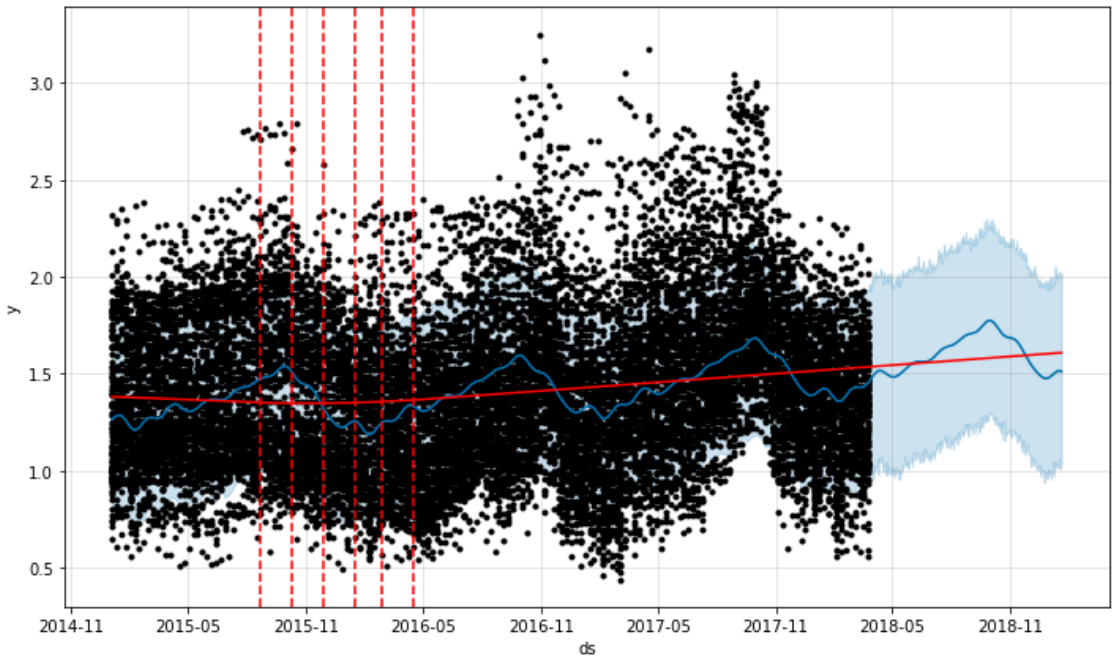

We create a Prophet model using 20 changepoints and a yearly seasonality (to take into account seasonal events such as holidays). We set the changepoints prior scale to 0.08 so that the trend will be more flexible (the default is 0.05).
We fit this new model on the training data, predict the future prices, and we plot the forecast.

[95]:
```
pro_change = Prophet(n_changepoints=20, yearly_seasonality=True,␣
 ↪changepoint_prior_scale=0.08)
forecast = pro_change.fit(train).predict(future)
fig = pro_change.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), pro_change, forecast)
```




We create a Prophet model using 20 changepoints and a yearly seasonality (to take into account seasonal events such as holidays). We set the changepoints prior scale to 0.001 so that the trend will be less flexible (the default is 0.05).
We fit this new model on the training data, predict the future prices, and we plot the forecast.

[97]:
```
pro_change = Prophet(n_changepoints=20, yearly_seasonality=True,␣
 ↪changepoint_prior_scale=0.001)
forecast = pro_change.fit(train).predict(future)
fig = pro_change.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), pro_change, forecast)
```

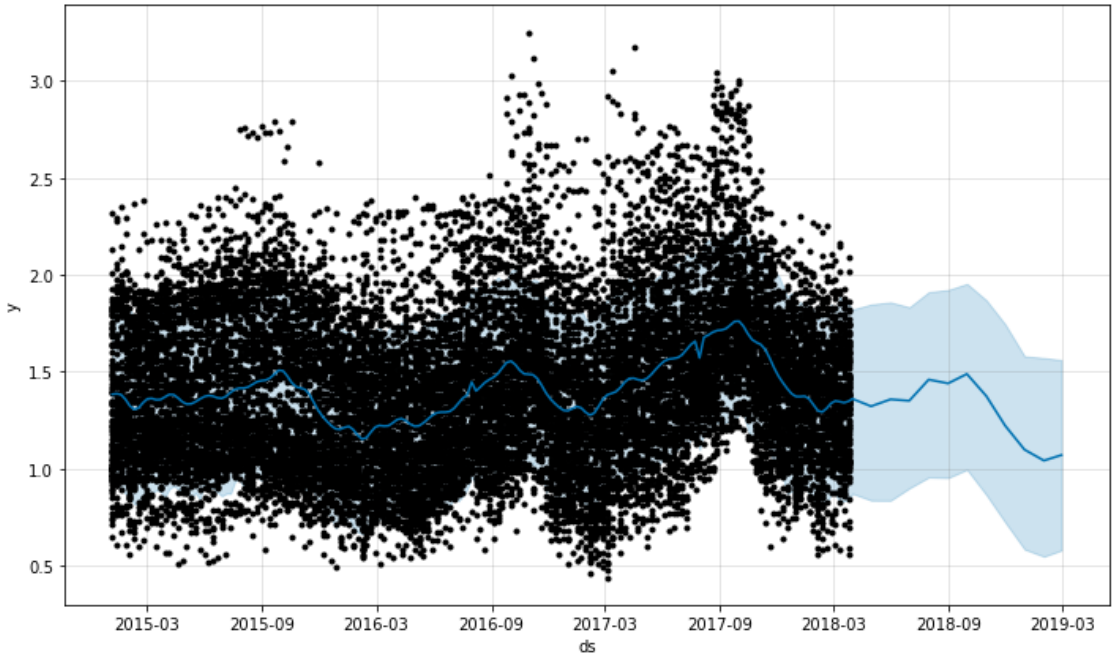



We create a dataframe containing two dates impacting the price of avocados, because these dates correspond to the National Avocado Day and the Guacamole Day (in the US). The prices on the day before will be also impacted by the holiday so we use the parameter lower window to consider this.

[98]:
```
avocado_season = pd.DataFrame({
  'holiday': 'avocado season',
  'ds': pd.to_datetime(['2014-07-31', '2014-09-16',
                        '2015-07-31', '2015-09-16',
                        '2016-07-31', '2016-09-16',
                        '2017-07-31', '2017-09-16',
                       '2018-07-31', '2018-09-16',
                        '2019-07-31', '2019-09-16']),
  'lower_window': -1,
  'upper_window': 0,
})
```

We create a Prophet model precising these impacting dates.
We fit this new model on the training data, predict the future monthly prices, and we plot the forecast.

[102]:
```
pro_holiday= Prophet(holidays=avocado_season)
pro_holiday.fit(train)
future_data = pro_holiday.make_future_dataframe(periods=12, freq = 'm')

#forecast the data for future data
forecast_data = pro_holiday.predict(future_data)
fig = pro_holiday.plot(forecast_data)
```



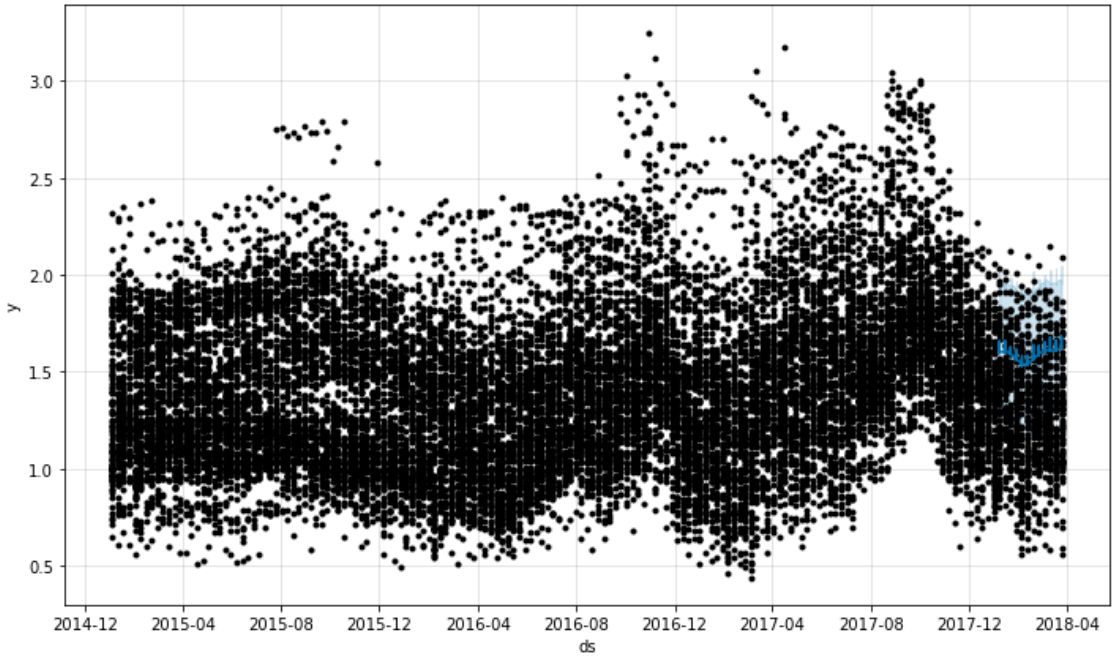

We split the data between train and test to fit a new model and evaluate it.

[103]:
```
train['type'] = X['type']
train['Total Volume'] = X['Total Volume']
train['4046'] = X['4046']
train['4225'] = X['4225']
train['4770'] = X['4770']
train['Small Bags'] = X['Small Bags']

train_X = train[:18000]
test_X = train[18000:]
```

We create a new Prophet model with additional regressors to include information about other feature such as the type, volume, PLU, and size of bags.
We fit this new model on the training part of the dataset and make predictions over 249 days using the test data.
We plot these predictions in blue.

[104]:
```
#Additional Regressor
pro_regressor= Prophet()
pro_regressor.add_regressor('type')
pro_regressor.add_regressor('Total Volume')
pro_regressor.add_regressor('4046')
pro_regressor.add_regressor('4225')
pro_regressor.add_regressor('4770')
pro_regressor.add_regressor('Small Bags')

#Fitting the data
pro_regressor.fit(train_X)
future_data = pro_regressor.make_future_dataframe(periods=249)
```

```
#forecast the data for Test  data
forecast_data = pro_regressor.predict(test_X)
pro_regressor.plot(forecast_data);
```




## References


[1] Benjamin Letham Sean J. Taylor. "Forecasting at Scale". In: *PeerJ Preprints* (2017). DOI: 10.7287/peerj.preprints.3190v2.

[2] Ankit Choudhary. *Generate Quick and Accurate Time Series Forecasts using Facebook's Prophet (with Python & R codes)*. URL: https://www.analyticsvidhya.com/blog/2018/05/generate-accurate-forecasts-facebook-prophet-python-r/.

[3] Oleksandr Gerasymov. *Machine Learning for Time Series Forecasting*. URL: https://codeit.us/blog/machine-learning-time-series-forecasting.

[4] *Machine Learning for Time Series Data*. URL: https://medium.com/@ODSC/machine-learning-for-time-series-data-e3971d38005b.

[5] *time series forecasting*. URL: https://whatis.techtarget.com/definition/time-series-forecasting.