1 Time Series Forecasting Using Facebook's Prophet

1.1 PROJECT OVERVIEW

This is a guided project to learn the basics of Facebook's Prophet Time Series Forecasting library. Prophet is an open source tool for Time Series Forecasting.

"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."

In this project, we will predict the future prices of Avocados using the Prophet library.

1.2 IMPORT LIBRARIES AND DATASET

Installing the necessary libraries and loading the dataset.

In [4]:
```
import pandas as pd # For data manipulation using dataframes
import numpy as np # For data statistical analysis
import matplotlib.pyplot as plt # For data visualisation
import random #To generate random numbers
import seaborn as sns #For better data visualization plots
from fbprophet import Prophet #For Time Series Forecasting

```
started 12:41:04 2021-12-27, finished in 16ms

In [3]:
```
# Load the dataset
df = pd.read_csv('avocado.csv')
```
started 12:41:00 2021-12-27, finished in 29ms

About the dataset

- Date: The date of the observation
- AveragePrice: the average price of a single avocado
- type: conventional or organic
- year: the year
- Region: the city or region of the observation
- Total Volume: Total number 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

1.3 DATA EXPLORATION

To get a better understanding of the dataset we're working with, let's do some basic data exploration.

In [5]:
```
# Let's view the head of the training dataset
df.head(5)
```

started 12:41:07 2021-12-27, finished in 29ms

Out[5]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26

In [6]:
```
# Let's view the last elements in the training dataset
df.tail(5)
```

Out[6]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.8
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.0
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.8
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.5
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.1

In [7]:
```
1 df.describe()
```

Out[7]:

	Unnamed: 0	AveragePrice	Total Volume	4046	4225	4770	Total
count	18249.000000	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824
mean	24.232232	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04	2.396
std	15.481045	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.862
min	0.000000	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000
25%	10.000000	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000e+00	5.088
50%	24.000000	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02	3.974
75%	38.000000	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03	1.107
max	52.000000	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06	1.937

In [8]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Unnamed: 0    18249 non-null  int64  
 1   Date          18249 non-null  object 
 2   AveragePrice  18249 non-null  float64
 3   Total Volume  18249 non-null  float64
 4   4046          18249 non-null  float64
 5   4225          18249 non-null  float64
 6   4770          18249 non-null  float64
 7   Total Bags    18249 non-null  float64
 8   Small Bags    18249 non-null  float64
 9   Large Bags    18249 non-null  float64
 10  XLarge Bags   18249 non-null  float64
 11  type          18249 non-null  object 
 12  year          18249 non-null  int64  
 13  region        18249 non-null  object 
dtypes: float64(9), int64(2), object(3)
memory usage: 1.9+ MB
```

In [10]:
```
1 df.isnull().sum()
```

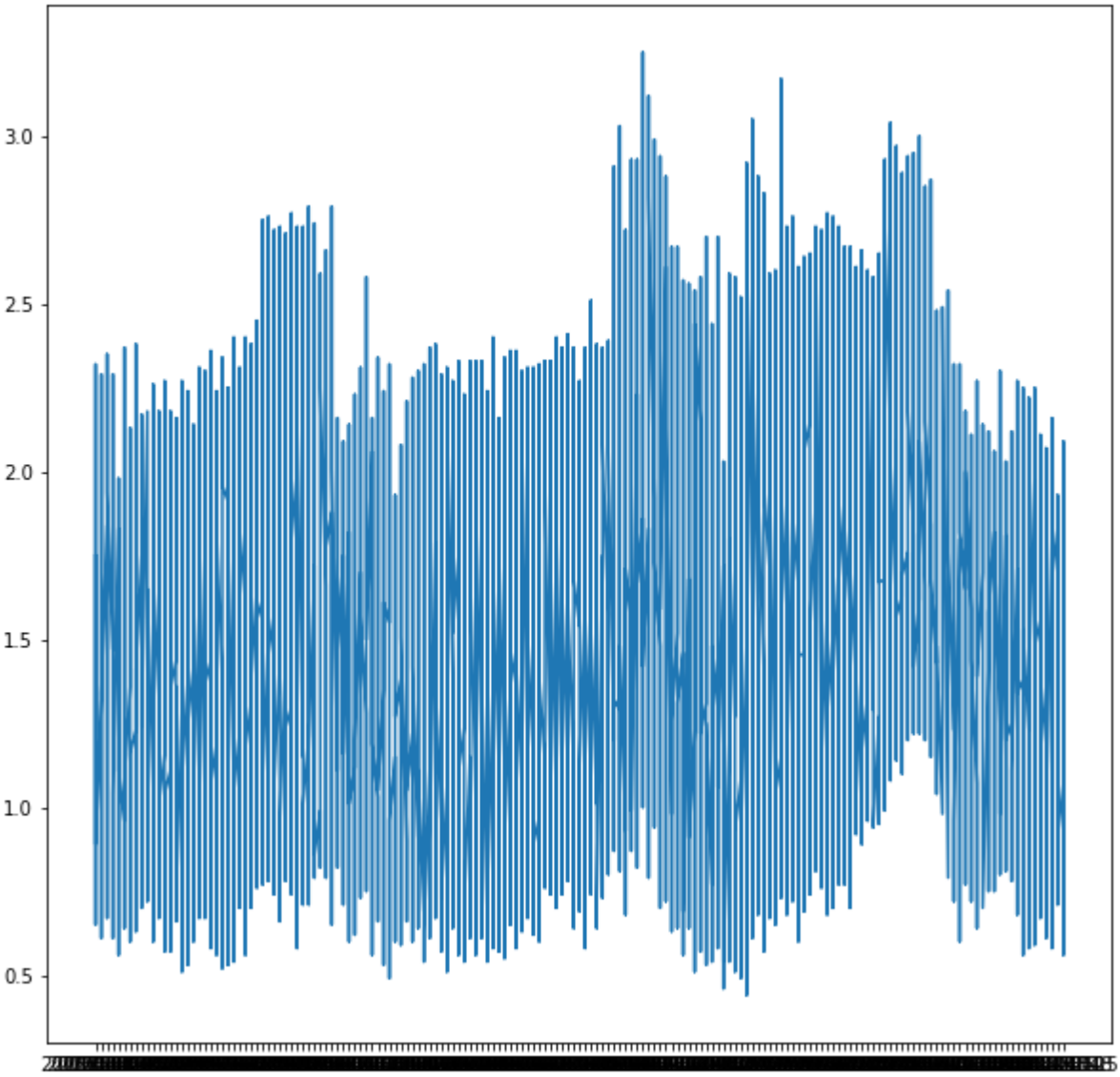
Out[10]:
```
Unnamed: 0      0
Date            0
AveragePrice    0
Total Volume    0
4046            0
4225            0
4770            0
Total Bags      0
Small Bags      0
Large Bags      0
XLarge Bags     0
type            0
year            0
region          0
dtype: int64
```

In [6]:
```
1 df1 = df.sort_values('Date')
```

started 12:41:14 2021-12-27, finished in 16ms

In [12]:
```
# Plot date and average price
plt.figure(figsize = (10,10))
plt.plot(df1['Date'],df1['AveragePrice'])
```

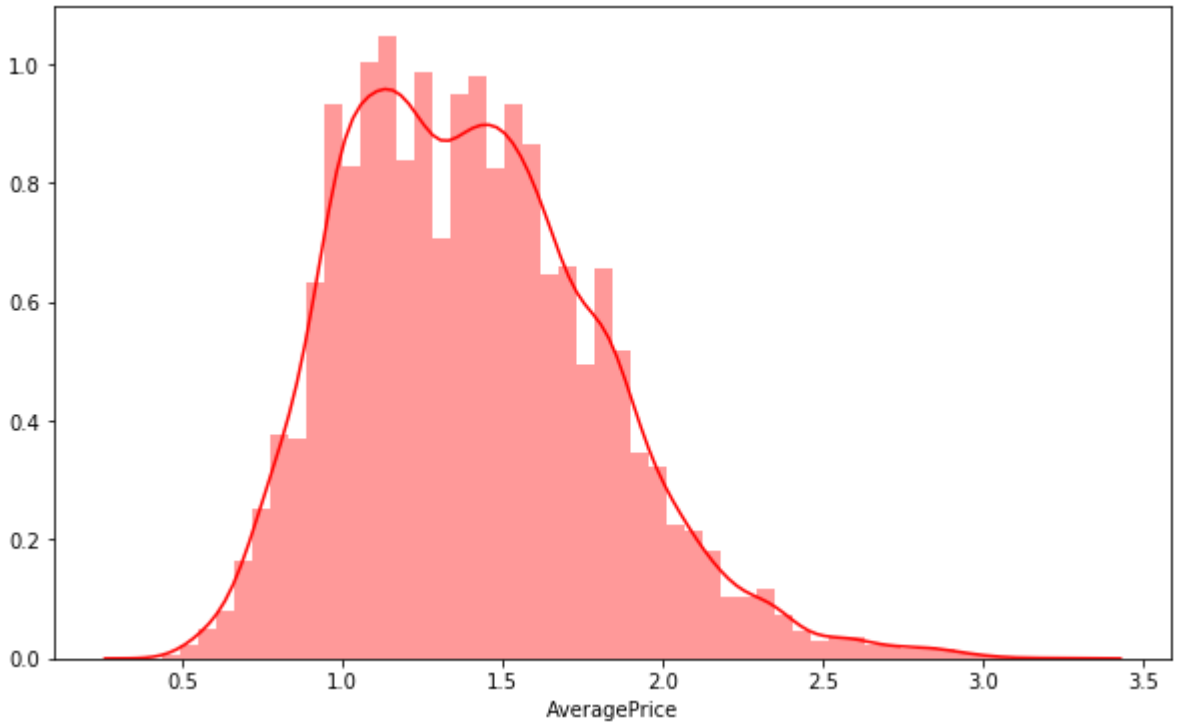
Out[12]: [<matplotlib.lines.Line2D at 0x24d3308b448>]



The above plot shows that the average price of Avocados vary between 0.5 and 3.5. Let's look at the price distribution closer.

In [13]:
```
# Plot distribution of the average price
plt.figure(figsize = (10,6))
sns.distplot(df1['AveragePrice'], color = 'r')
```

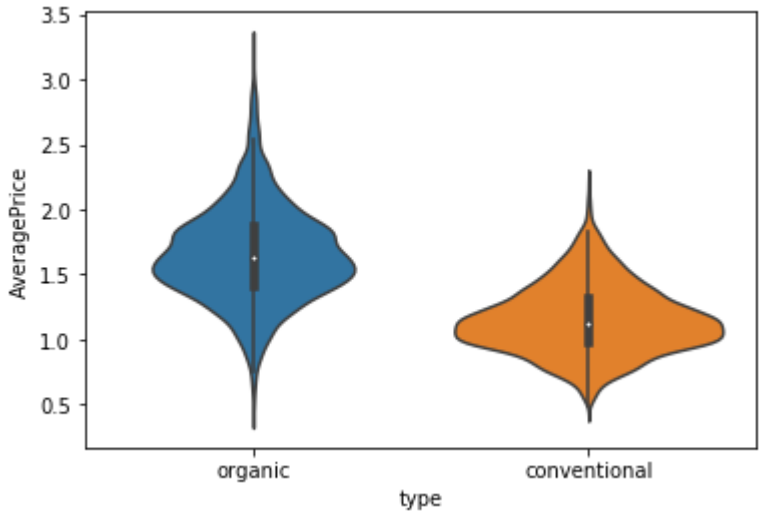
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24d33e8cd48>
```



The distribution plot shows that 0.5 and 3 are outliers and the average price of avocados lie between 1 and 1.5 dollars.

In [14]:
```
# Plot a violin plot of the average price vs. avocado type
sns.violinplot(y = 'AveragePrice', x = 'type', data = df1)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x24d33dcd548>



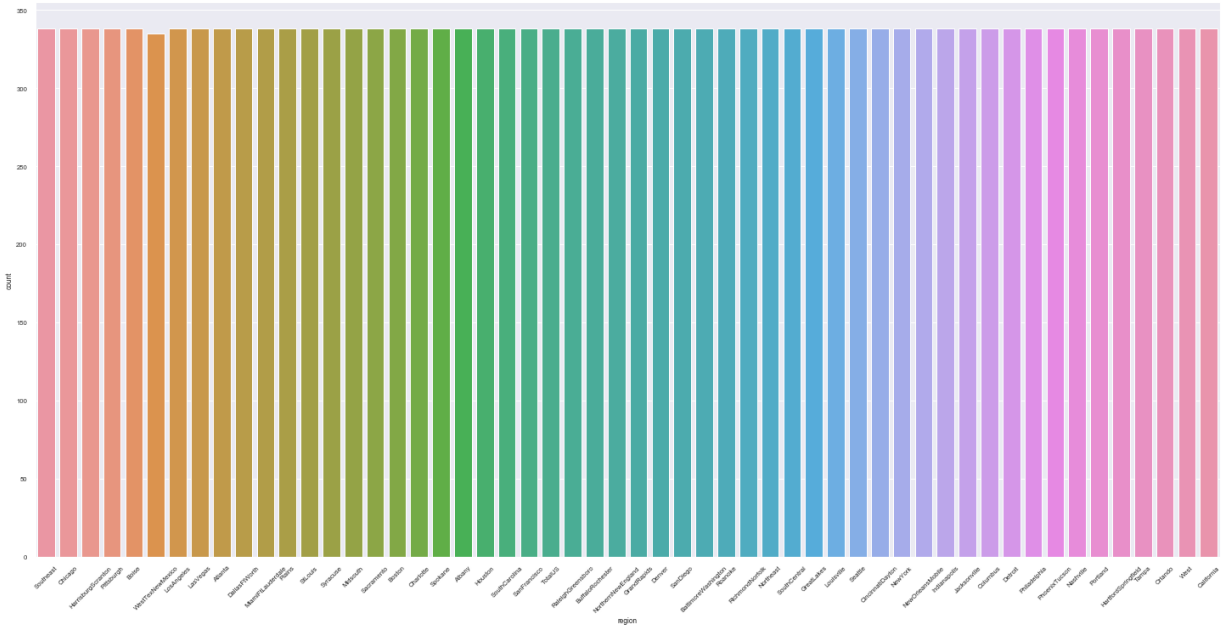
This violin plot shows that the average price of organic avocados is higher than conventional avocados.

In [16]:
```
# Bar Chart to indicate the number of regions

sns.set(font_scale=0.7)
plt.figure(figsize=[25,12])
sns.countplot(x = 'region', data = df1)
plt.xticks(rotation = 45)

```

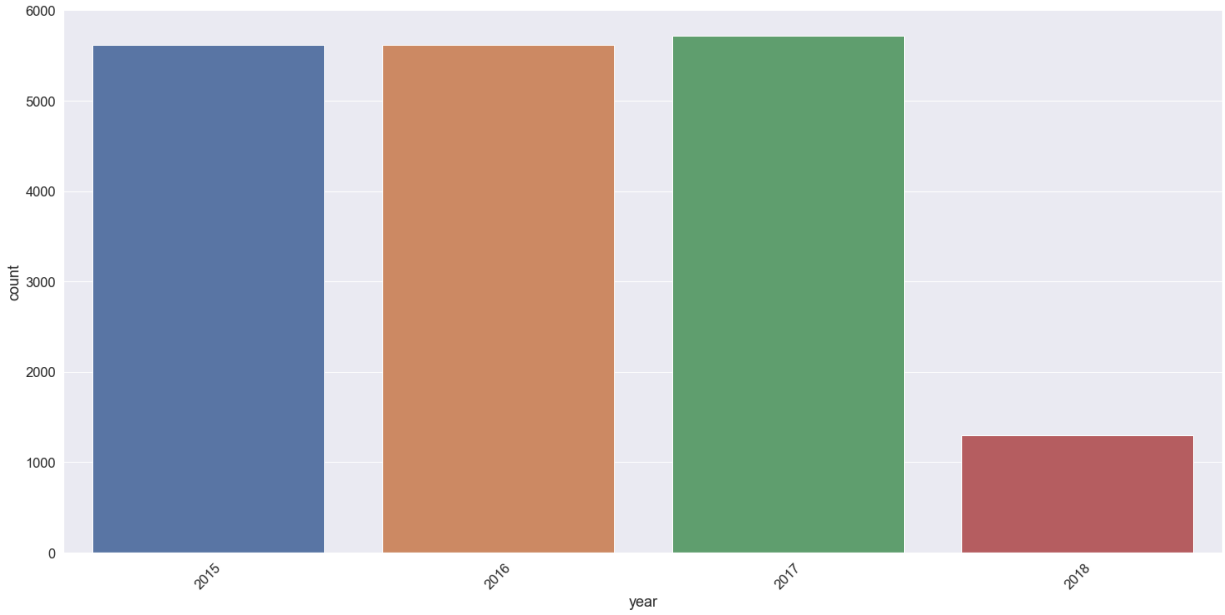
Out[16]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53]),
 <a list of 54 Text xticklabel objects>)
```



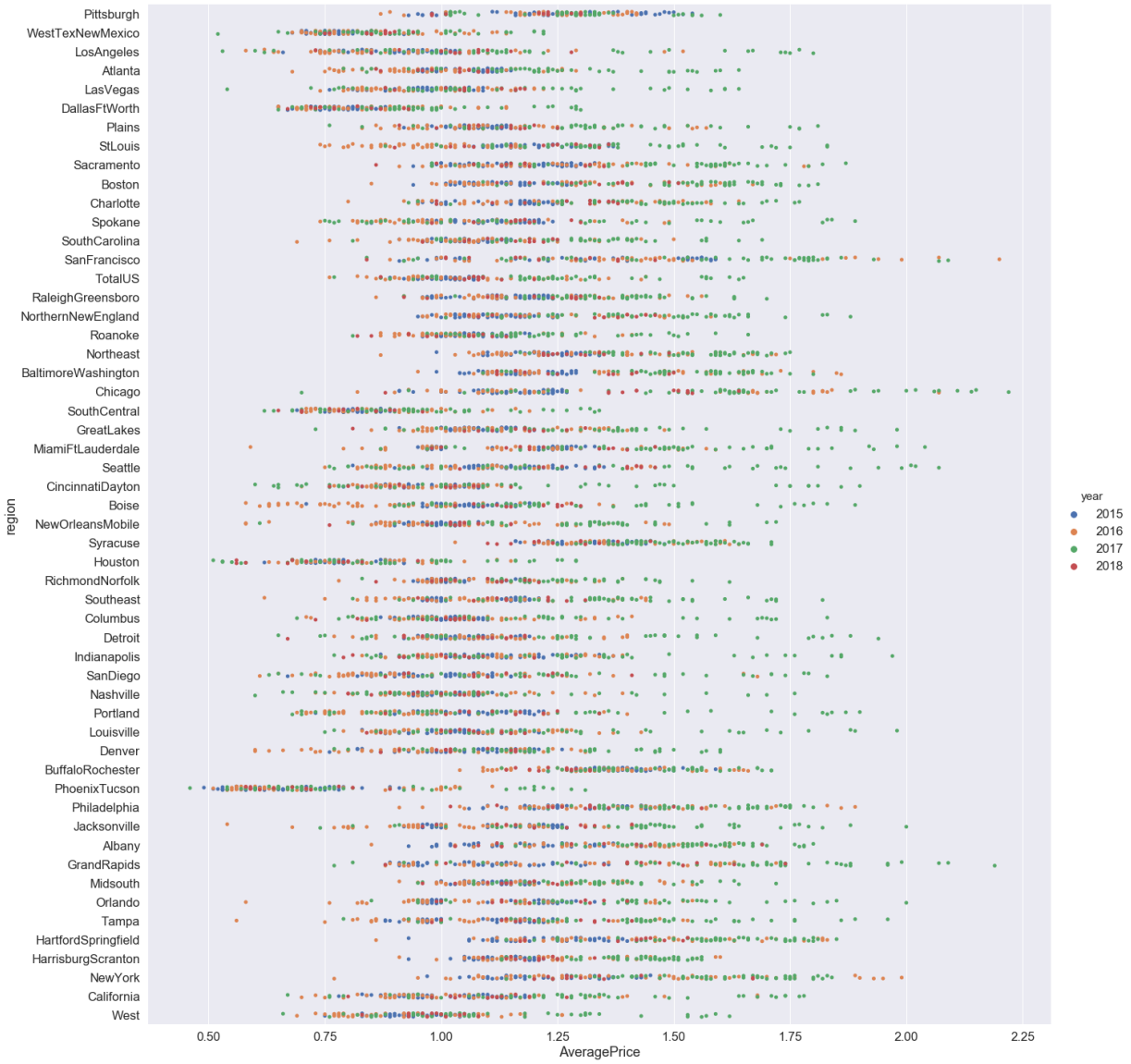
Loading [MathJax]/extensions/Safe.js

In [17]:
```
# Bar Chart to indicate the count in every year
sns.set(font_scale=1.5)
plt.figure(figsize=[25,12])
sns.countplot(x = 'year', data = df1)
plt.xticks(rotation = 45)
```

Out[17]: (array([0, 1, 2, 3]), <a list of 4 Text xticklabel objects>)



In [19]:
```
# plot the avocado prices vs. regions for conventional avocado
conventional = sns.catplot('AveragePrice', 'region', data = df1[df1['type']=
```



Loading [MathJax]/extensions/Safe.js

In [20]:
```
  # plot the avocado prices vs. regions for organic avocados
organic = sns.catplot('AveragePrice', 'region', data = df1[df1['type']=='org
```

1.4 DATA PREPARATION

In [7]:
```
1 df1.head()
```
started 12:41:20 2021-12-27, finished in 25ms

Out[7]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
11569	51	2015-01-04	1.75	27365.89	9307.34	3844.81	615.28	13598.46	13061
9593	51	2015-01-04	1.49	17723.17	1189.35	15628.27	0.00	905.55	905
10009	51	2015-01-04	1.68	2896.72	161.68	206.96	0.00	2528.08	2528
1819	51	2015-01-04	1.52	54956.80	3013.04	35456.88	1561.70	14925.18	11264
9333	51	2015-01-04	1.64	1505.12	1.27	1129.50	0.00	374.35	186

In [22]:
```
1 df1_prophet = df1[['Date', 'AveragePrice']]
```

In [24]:
```
1 df1_prophet
```

Out[24]:

	Date	AveragePrice
11569	2015-01-04	1.75
9593	2015-01-04	1.49
10009	2015-01-04	1.68
1819	2015-01-04	1.52
9333	2015-01-04	1.64
...	...	...
8574	2018-03-25	1.36
9018	2018-03-25	0.70
18141	2018-03-25	1.42
17673	2018-03-25	1.70
8814	2018-03-25	1.34

18249 rows × 2 columns

We only need the Date and AveragePrice columns for this time series forecasting. The columns

also need to be renamed to fit the Prophet algorithm.

In [25]:
```
df1_prophet = df1_prophet.rename(columns = {'Date': 'ds' , 'AveragePrice': '
```

In [26]:
```
df1_prophet
```

Out[26]:

	ds	y
11569	2015-01-04	1.75
9593	2015-01-04	1.49
10009	2015-01-04	1.68
1819	2015-01-04	1.52
9333	2015-01-04	1.64
...	...	...
8574	2018-03-25	1.36
9018	2018-03-25	0.70
18141	2018-03-25	1.42
17673	2018-03-25	1.70
8814	2018-03-25	1.34

18249 rows × 2 columns

1.5 MODEL IMPLEMENTATION AND FORECASTING

In [1]:
```
# Fit dataset to the model
model = Prophet()
model.fit(df1_prophet)
```

started 12:46:18 2021-12-27, finished in 17ms

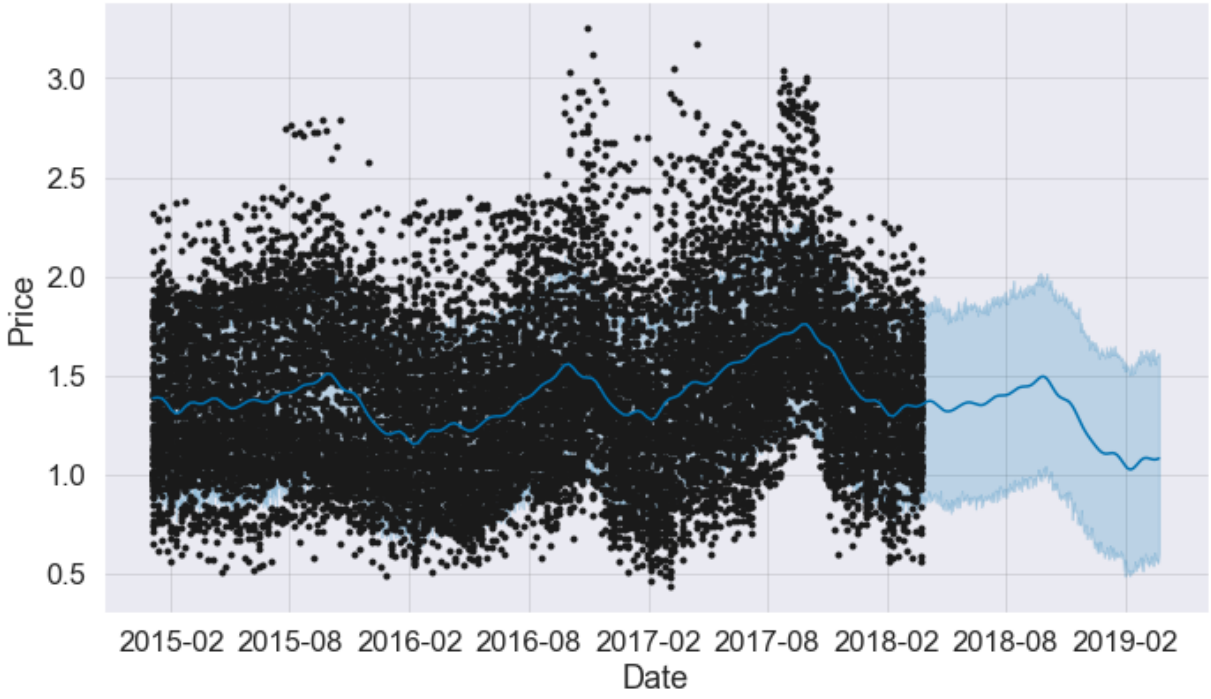
In [28]:
```
# Forcasting into the future for the next 365days
future = model.make_future_dataframe(periods = 365)
forecast = model.predict(future)
```

In [29]:
```
1 forecast
```

Out[29]:

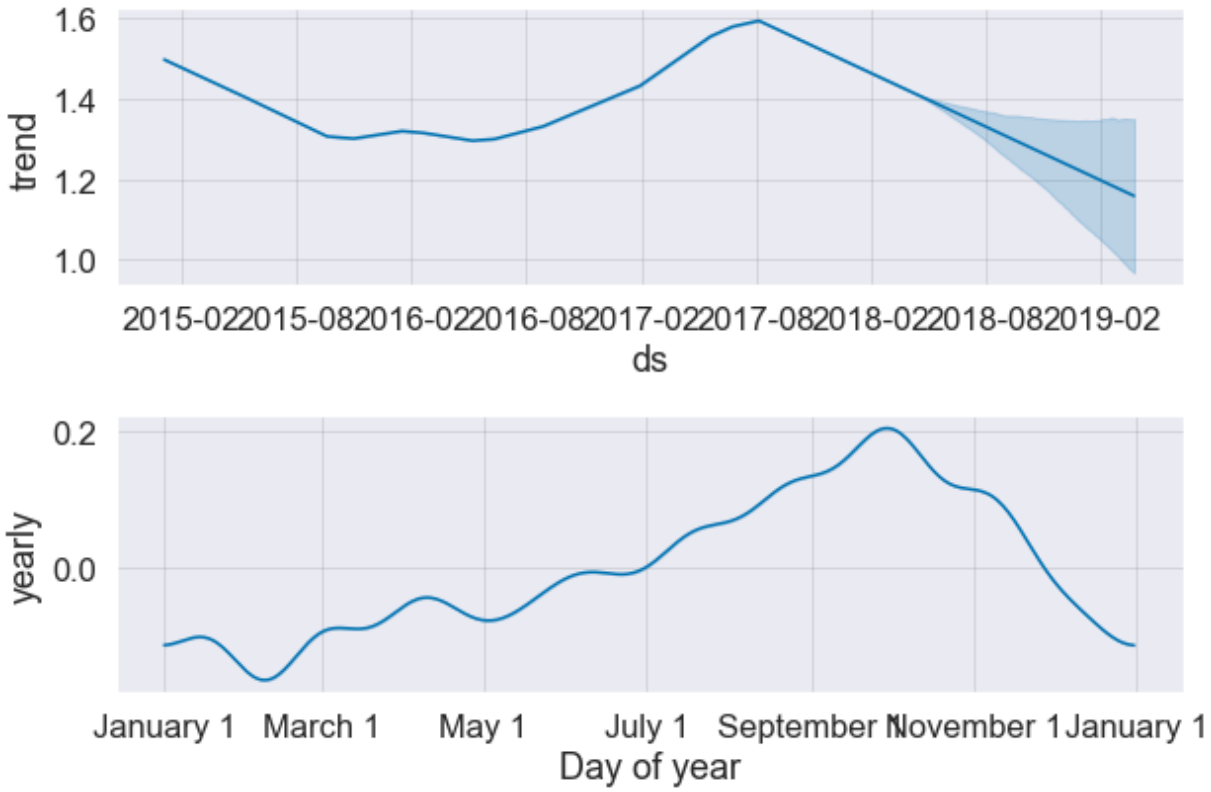
	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms
0	2015-01-04	1.497917	0.923582	1.882271	1.497917	1.497917	-0.113
1	2015-01-04	1.497917	0.893287	1.888370	1.497917	1.497917	-0.113
2	2015-01-04	1.497917	0.903401	1.892052	1.497917	1.497917	-0.113
3	2015-01-04	1.497917	0.901735	1.893753	1.497917	1.497917	-0.113
4	2015-01-04	1.497917	0.862839	1.851518	1.497917	1.497917	-0.113
...	...	...	...	...	...	...	
18609	2019-	1.161737	0.574797	1.597413	0.973498	1.349956	-0.086

In [30]:
```
1 figure = model.plot(forecast, xlabel = 'Date', ylabel = 'Price')
```



Loading [MathJax]/extensions/Safe.js

In [31]:
```
1 figure2 = model.plot_components(forecast)
```



The forecast and trend plot shows that the average price for avocado will follow the same seasonality as the past years in the next 365 days but the overall trend will continue to decrease.

1.6 REGION SPECIFIC FORECASTING

The model predicted the overall trend for avocado price to decrease in the next year. But is this true for all the regions? To check if there is a difference between the overall trend and individual regions, let us use the same model on one particular region.

1.6.1 West Region

In [ ]:
```
1 # Select specific region
```

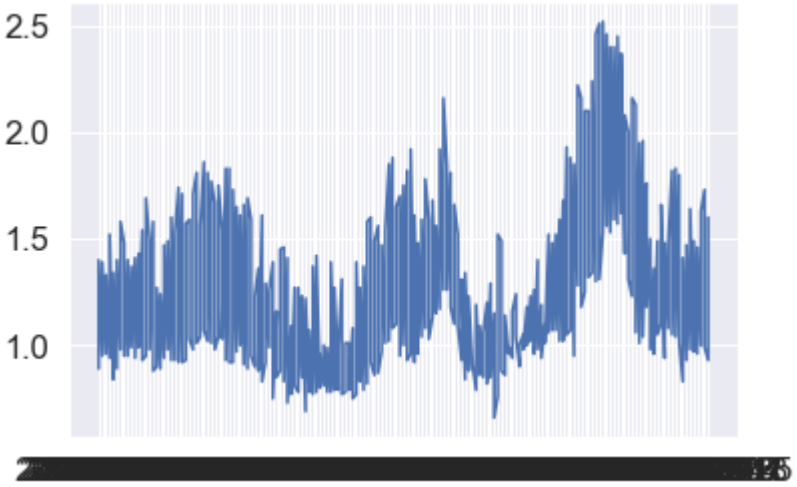
In [34]:
```
1 df_sample = df[df['region'] == 'West']
```

In [35]:
```
1 df_sample = df_sample.sort_values('Date')
```

In [36]:
```
1 plt.plot(df_sample['Date'], df_sample['AveragePrice'])
```

```
INFO:matplotlib.category:Using categorical units to plot a list of strings that
are all parsable as floats or dates. If these strings should be plotted as numb
ers, cast to the appropriate data type before plotting.
INFO:matplotlib.category:Using categorical units to plot a list of strings that
are all parsable as floats or dates. If these strings should be plotted as numb
ers, cast to the appropriate data type before plotting.
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x24d38142448>]
```

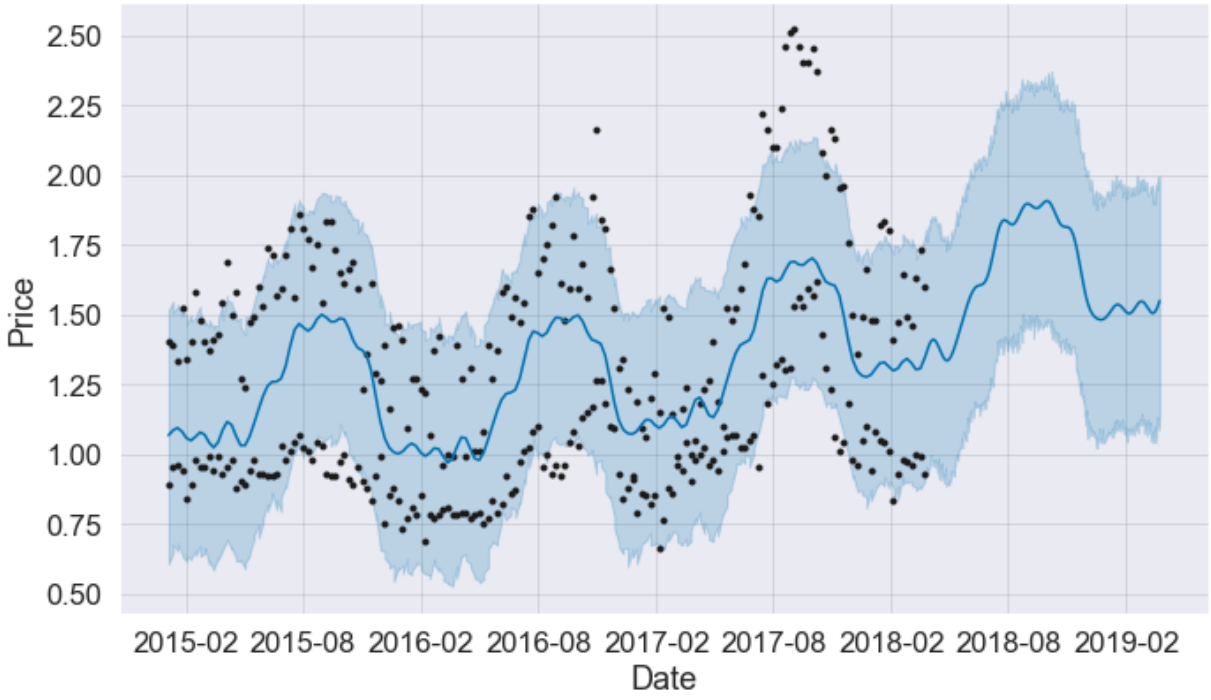


In [37]:
```
1 df_sample = df_sample.rename(columns = {'Date':'ds', 'AveragePrice':'y'})
```

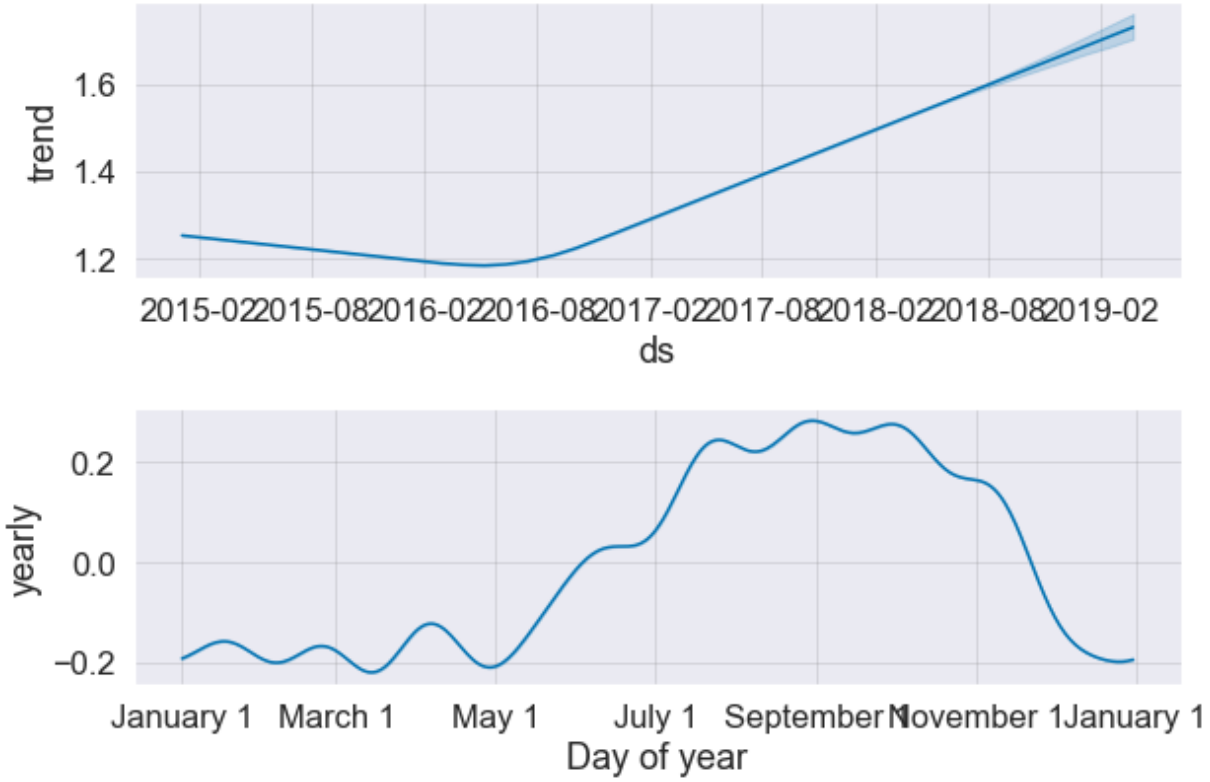
In [38]:
```
1 m = Prophet()
2 m.fit(df_sample)
3 # Forcasting into the future
4 future = m.make_future_dataframe(periods=365)
5 forecast = m.predict(future)
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonalit
y=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=
True to override this.
```

In [39]:
```
1 figure = m.plot(forecast, xlabel='Date', ylabel='Price')
```



In [40]:
```
1 figure3 = m.plot_components(forecast)
```



1.6.2 Chicago Region

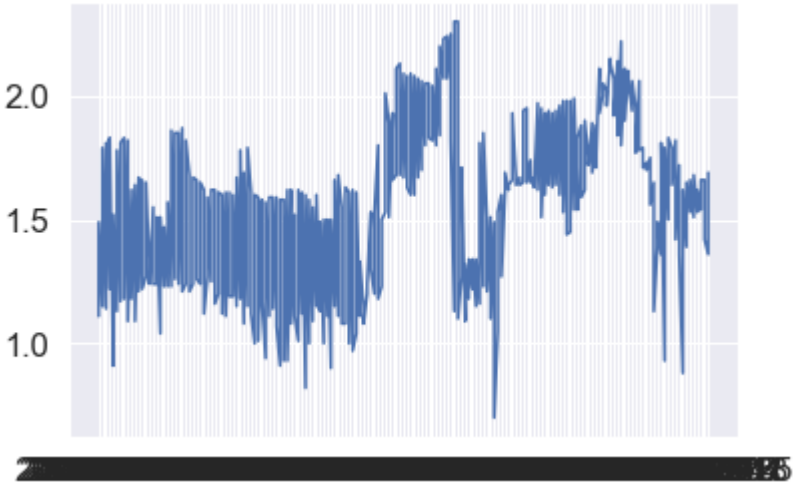
In [41]:
```
1 df_sample2 = df[df['region'] == 'Chicago']
```

In [42]:
```
df_sample2 = df_sample2.sort_values('Date')
```

In [43]:
```
plt.plot(df_sample2['Date'], df_sample2['AveragePrice'])
```

```
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x24d3b8d1688>]
```

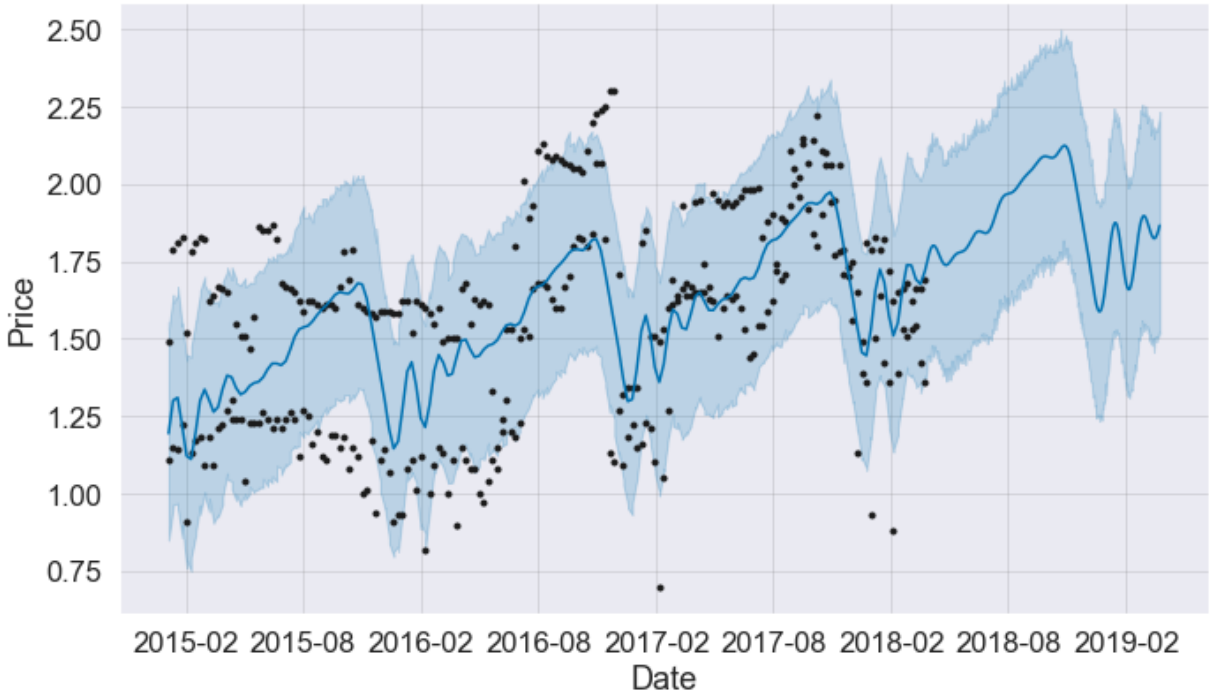


In [44]:
```
df_sample2 = df_sample2.rename(columns = {'Date':'ds', 'AveragePrice':'y'})
```

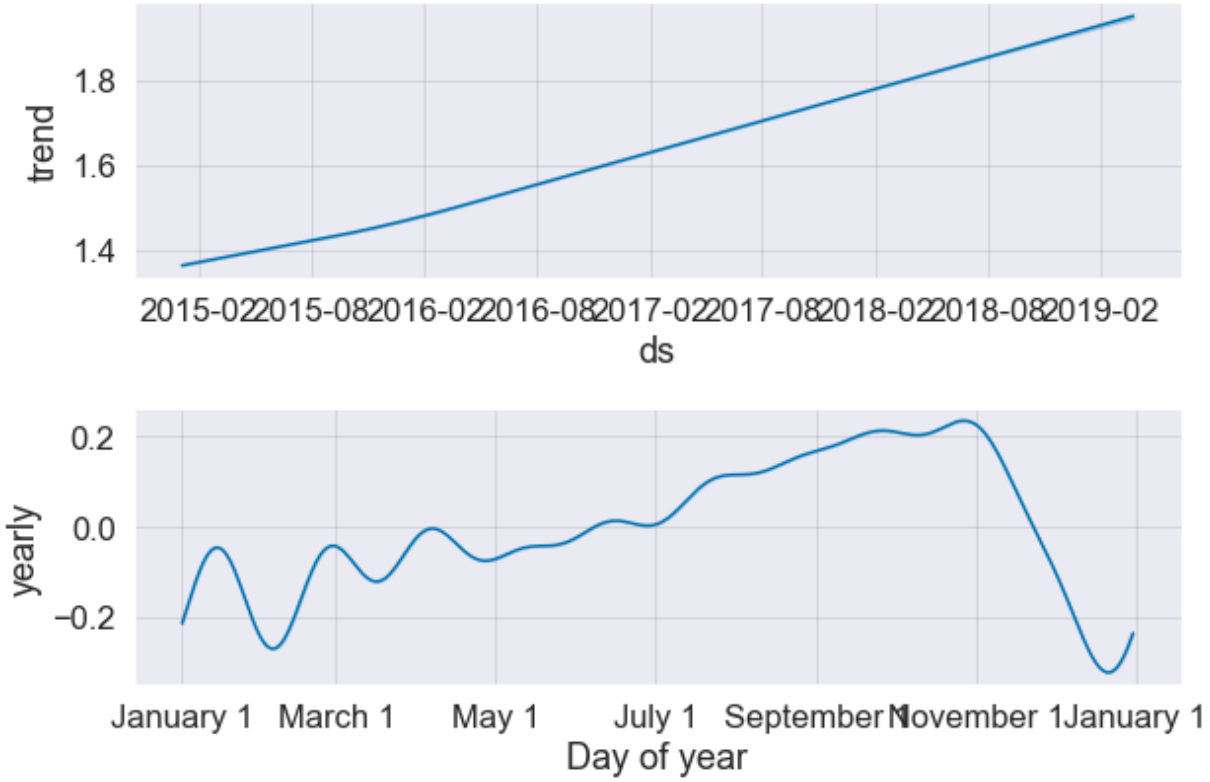
In [45]:
```
m = Prophet()
m.fit(df_sample2)
# Forcasting into the future
future = m.make_future_dataframe(periods=365)
forecast = m.predict(future)
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

In [46]:
```
1 figure = m.plot(forecast, xlabel='Date', ylabel='Price')
```



In [47]:
```
1 figure4 = m.plot_components(forecast)
```



The above two section results show us that even though the overall price for avocados was forecasted to go down, the individual regions West and Chicago have a steady upwards trend in the next 365 days. This shows that the data needs to be divided into smaller groups and the model needs to be implemented seperately on each group for more accurate forecasts.