Avocado Dataset Analysis and ML Predictions

November 21, 2020

1 Avocado Dataset Analysis and ML Prediction

1.1 Table of Contents

1.1.1 * Problem Statement

- In this study, we can predict the Avocado's Average Price based on different features. The features are different (Total Bags,Date,Type,Year,Region…).

The variables of the dataset are the following:

- Categorical: 'region','type'
- Date: 'Date'
- Numerical:'Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags','Large Bags','XLarge Bags','Year'
- Target:'AveragePrice'

1.1.2 * Data Loading and Description

- Database columns are explained in the next section:

Features

Description

'Unamed: 0'

Its just a useless index feature that will be removed later

'Total Volume'

Total sales volume of avocados

'4046'

Total sales volume of Small/Medium Hass Avocado

‘4225’

Total sales volume of Large Hass Avocado

‘4770’

Total sales volume of Extra Large Hass Avocado

‘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 XLarge Bags sold

1.1.3 * Importing packages

[2]:
```
import pandas as pd
import matplotlib
matplotlib.use("Agg", warn=False)
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas_profiling
%matplotlib inline

import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
from plotly import tools

import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

```
<ipython-input-2-f398dba02087>:3: MatplotlibDeprecationWarning: The 'warn'
parameter of use() is deprecated since Matplotlib 3.1 and will be removed in
3.3.  If any parameter follows 'warn', they should be pass as keyword, not
positionally.
  matplotlib.use("Agg", warn=False)
```

- Read in the Avocado Prices csv file as a DataFrame called df

[3]:
```
df= pd.read_csv("avocado.csv")
```

1.1.4 * Data Profiling

[3]:
```
df.shape
```

[3]:
```
(18249, 14)
```

[4]:
```
df.columns  # This will print the names of all columns.
```

[4]:
```
Index(['Unnamed: 0', 'Date', 'AveragePrice', 'Total Volume', '4046', '4225',
       '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type',
       'year', 'region'],
      dtype='object')
```

[5]:
```
df.head()  # Will give you first 5 records
```

[5]:
```
   Unnamed: 0        Date  AveragePrice  Total Volume     4046       4225  \
0           0  2015-12-27          1.33      64236.62  1036.74   54454.85
1           1  2015-12-20          1.35      54876.98   674.28   44638.81
2           2  2015-12-13          0.93     118220.22   794.70  109149.67
3           3  2015-12-06          1.08      78992.15  1132.00   71976.41
4           4  2015-11-29          1.28      51039.60   941.48   43838.39

     4770  Total Bags  Small Bags  Large Bags  XLarge Bags          type  \
0   48.16     8696.87     8603.62       93.25          0.0  conventional
1   58.33     9505.56     9408.07       97.49          0.0  conventional
2  130.50     8145.35     8042.21      103.14          0.0  conventional
3   72.58     5811.16     5677.40      133.76          0.0  conventional
4   75.78     6183.95     5986.26      197.69          0.0  conventional

   year  region
0  2015  Albany
1  2015  Albany
2  2015  Albany
3  2015  Albany
4  2015  Albany
```

- The Feature "Unnamed:0" is just a representation of the indexes, so it's useless to keep it, we'll remove it in pre-processing !

[6]:
```
df.tail()  # This will print the last n rows of the Data Frame
```

[6]:
```
       Unnamed: 0        Date  AveragePrice  Total Volume     4046     4225  \
18244           7  2018-02-04          1.63      17074.83  2046.96  1529.20
18245           8  2018-01-28          1.71      13888.04  1191.70  3431.50
18246           9  2018-01-21          1.87      13766.76  1191.92  2452.79
18247          10  2018-01-14          1.93      16205.22  1527.63  2981.04
18248          11  2018-01-07          1.62      17489.58  2894.77  2356.13
```

```
         4770  Total Bags  Small Bags  Large Bags  XLarge Bags     type  year  \
18244    0.00    13498.67    13066.82      431.85          0.0  organic  2018
18245    0.00     9264.84     8940.04      324.80          0.0  organic  2018
18246  727.94     9394.11     9351.80       42.31          0.0  organic  2018
18247  727.01    10969.54    10919.54       50.00          0.0  organic  2018
18248  224.53    12014.15    11988.14       26.01          0.0  organic  2018

                 region
18244  WestTexNewMexico
18245  WestTexNewMexico
18246  WestTexNewMexico
18247  WestTexNewMexico
18248  WestTexNewMexico
```

[7]:
```python
df.info() # This will give Index, Datatype and Memory information
```

```
<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
```

- Well as a first observation we can see that we are lucky, we dont have any missing values (**18249** complete data) and **13 columns**. Now let's do some Feature Engineering on the Date Feature in **pre-processing** later so we can be able to use the day and the month columns in building our machine learning model later. (I didn't mention the year because its already there in data frame)

[8]:
```python
# Use include='all' option to generate descriptive statistics for all columns
# You can get idea about which column has missing values using this
df.describe()
```

[8]:
```
        Unnamed: 0  AveragePrice  Total Volume          4046          4225  \
count  18249.000000  18249.000000  1.824900e+04  1.824900e+04  1.824900e+04
mean      24.232232      1.405978  8.506440e+05  2.930084e+05  2.951546e+05
std       15.481045      0.402677  3.453545e+06  1.264989e+06  1.204120e+06
min        0.000000      0.440000  8.456000e+01  0.000000e+00  0.000000e+00
25%       10.000000      1.100000  1.083858e+04  8.540700e+02  3.008780e+03
50%       24.000000      1.370000  1.073768e+05  8.645300e+03  2.906102e+04
75%       38.000000      1.660000  4.329623e+05  1.110202e+05  1.502069e+05
max       52.000000      3.250000  6.250565e+07  2.274362e+07  2.047057e+07

               4770    Total Bags    Small Bags    Large Bags    XLarge Bags  \
count  1.824900e+04  1.824900e+04  1.824900e+04  1.824900e+04   18249.000000
mean   2.283974e+04  2.396392e+05  1.821947e+05  5.433809e+04    3106.426507
std    1.074641e+05  9.862424e+05  7.461785e+05  2.439660e+05   17692.894652
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
25%    0.000000e+00  5.088640e+03  2.849420e+03  1.274700e+02       0.000000
50%    1.849900e+02  3.974383e+04  2.636282e+04  2.647710e+03       0.000000
75%    6.243420e+03  1.107834e+05  8.333767e+04  2.202925e+04     132.500000
max    2.546439e+06  1.937313e+07  1.338459e+07  5.719097e+06  551693.650000

               year
count  18249.000000
mean    2016.147899
std        0.939938
min     2015.000000
25%     2015.000000
50%     2016.000000
75%     2017.000000
max     2018.000000
```

- We can see all columns having count **18249**. Looks like it doesn't contain missing values

[9]:
```
df.isnull().sum()   # Will show you null count for each column, but will not␣
 ↪count Zeros(0) as null
```

[9]:
```
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
```

- We can see that **no missing values** exist in dataset, that's great!

1.1.5 * Preprocessing

- The Feature **"Unnamed:0"** is just a representation of the indexes, so it's useless to keep it, lets remove it now !

[11]:
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

- Lets check our data head again to make sure that the Feature **Unnamed:0 is removed**

[12]:
```
df.head()
```

[12]:
```
         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

   Total Bags  Small Bags  Large Bags  XLarge Bags          type  year  region
0     8696.87     8603.62       93.25          0.0  conventional  2015  Albany
1     9505.56     9408.07       97.49          0.0  conventional  2015  Albany
2     8145.35     8042.21      103.14          0.0  conventional  2015  Albany
3     5811.16     5677.40      133.76          0.0  conventional  2015  Albany
4     6183.95     5986.26      197.69          0.0  conventional  2015  Albany
```

- Earlier in **info** we have seen that **Date** is **Object** type not the date type. We have to change its type to date type.

[13]:
```
df['Date']=pd.to_datetime(df['Date'])
df['Month']=df['Date'].apply(lambda x:x.month)
df['Day']=df['Date'].apply(lambda x:x.day)
```

- Lets check the head to see what we have done:

[14]:
```
df.head()
```

[14]:
```
        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
```

```
   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

   region  Month  Day
0  Albany     12   27
1  Albany     12   20
2  Albany     12   13
3  Albany     12    6
4  Albany     11   29
```

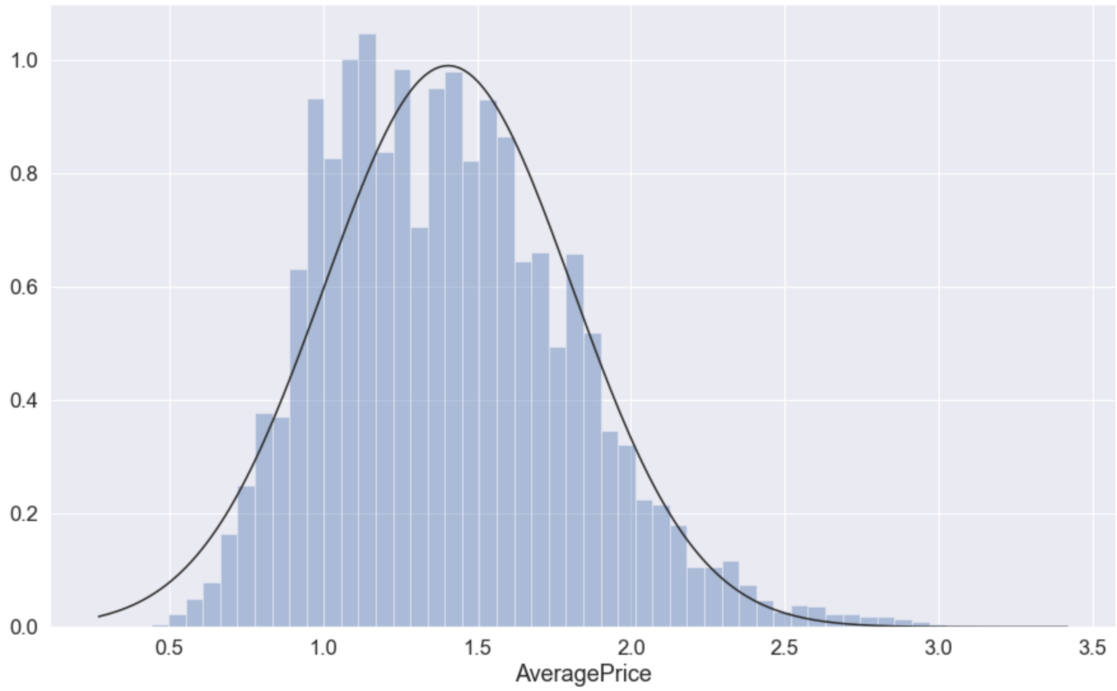
- **Q.1 Which type of Avocados are more in demand (Conventional or Organic)?**

[15]:
```
Type=df.groupby('type')['Total Volume'].agg('sum')

values=[Type['conventional'],Type['organic']]
labels=['conventional','organic']

trace=go.Pie(labels=labels,values=values)
py.iplot([trace])
```

- Just over **2% of our dataset is organic**. So looks like **Conventional is in more demand**. Now, let's look at the average price distribution
- **Q.2 In which range Average price lies, what is distribution look like?**

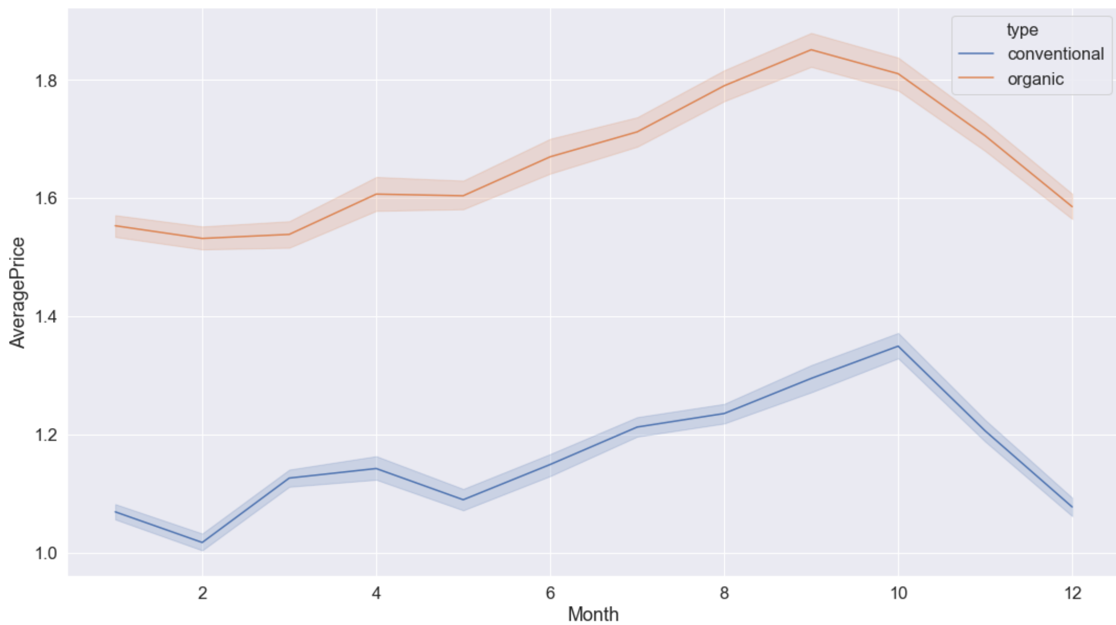
[16]:
```
sns.set(font_scale=1.5)
from scipy.stats import norm
fig, ax = plt.subplots(figsize=(15, 9))
sns.distplot(a=df.AveragePrice, kde=False, fit=norm)
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15d5b159940>
```



- Average Price distribution shows that for most cases **price of avocado is between 1.1, 1.4.**
- **Q.3 How Average price is distributed over the months for Conventional and Organic Types?**

[17]:
```
plt.figure(figsize=(18,10))
sns.lineplot(x="Month", y="AveragePrice", hue='type', data=df)
plt.show()
```



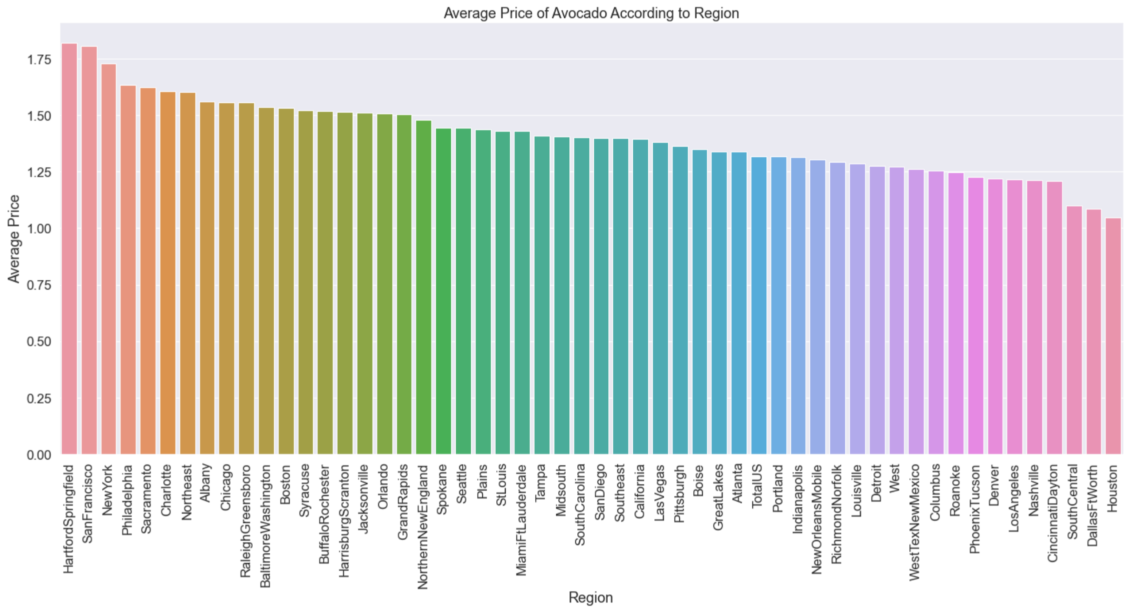
- Looks like there was a **hike between months 8 – 10 for both Conventional and Organic type** of Avocados prices

1.1.6 * Now lets plot Average price distribution based on region

- **Q.4 What are TOP 5 regions where Average price are very high?**

[18]:
```
region_list=list(df.region.unique())
average_price=[]

for i in region_list:
    x=df[df.region==i]
    region_average=sum(x.AveragePrice)/len(x)
    average_price.append(region_average)

df1=pd.DataFrame({'region_list':region_list,'average_price':average_price})
new_index=df1.average_price.sort_values(ascending=False).index.values
sorted_data=df1.reindex(new_index)

plt.figure(figsize=(24,10))
ax=sns.barplot(x=sorted_data.region_list,y=sorted_data.average_price)

plt.xticks(rotation=90)
plt.xlabel('Region')
plt.ylabel('Average Price')
plt.title('Average Price of Avocado According to Region')
```

[18]: Text(0.5, 1.0, 'Average Price of Avocado According to Region')



- Looks like these region are where price is very high

  HartfordSpringfield

  SanFrancisco

  NewYork

  Philadelphia

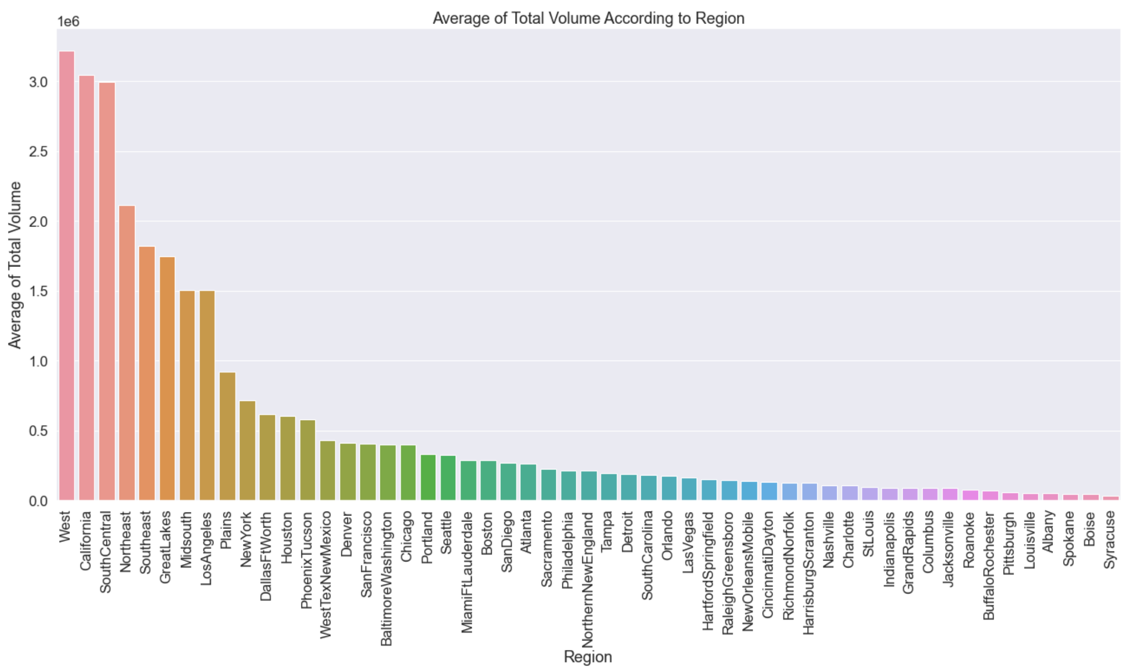
  Sacramento

- **Q.5 What are TOP 5 regions where Average consumption is very high?**

[19]:
```
filter1=df.region!='TotalUS'
df1=df[filter1]

region_list=list(df1.region.unique())
average_total_volume=[]

for i in region_list:
    x=df1[df1.region==i]
    average_total_volume.append(sum(x['Total Volume'])/len(x))
df3=pd.DataFrame({'region_list':region_list,'average_total_volume':
 ↪average_total_volume})

new_index=df3.average_total_volume.sort_values(ascending=False).index.values
sorted_data1=df3.reindex(new_index)
```

```
plt.figure(figsize=(22,10))
ax=sns.barplot(x=sorted_data1.region_list,y=sorted_data1.average_total_volume)

plt.xticks(rotation=90)
plt.xlabel('Region')
plt.ylabel('Average of Total Volume')
plt.title('Average of Total Volume According to Region')
```

[19]: Text(0.5, 1.0, 'Average of Total Volume According to Region')



- Looks like these region are where Consumption is very high

  West

  California

  SouthCentral

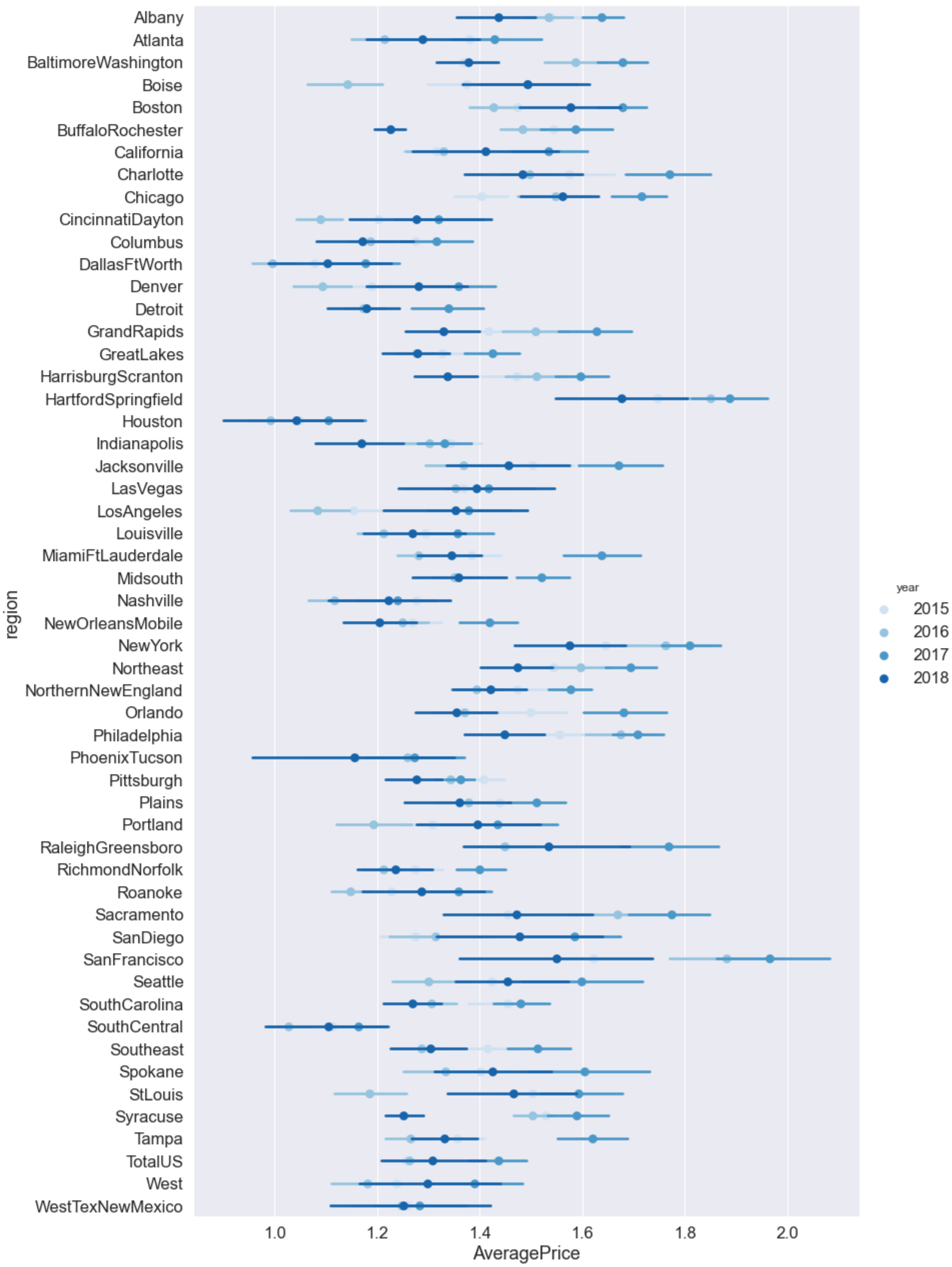
  Northeast

  Southeast

- **Q.6 In which year and for which region was the Average price the highest?**

[20]:
```
g = sns.factorplot('AveragePrice','region',data=df,
                   hue='year',
                   size=18,
                   aspect=0.7,
```

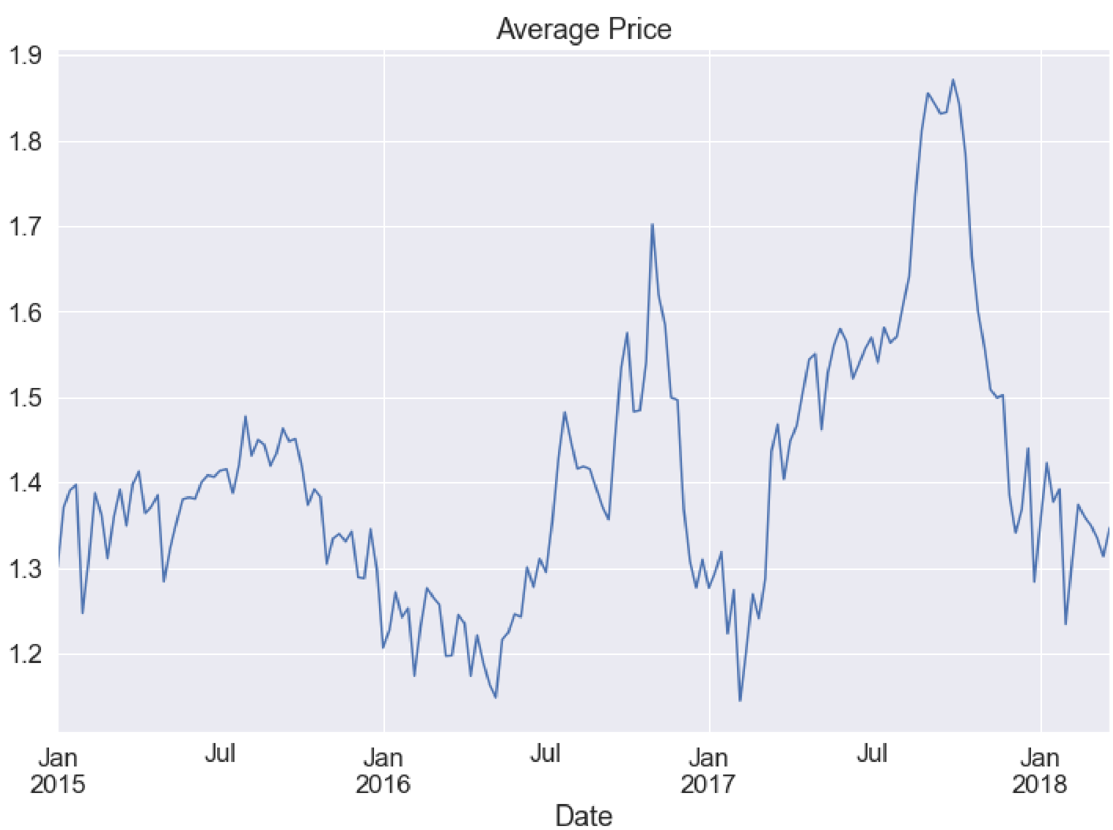
```
                palette='Blues',
                join=False,
            )
```



- Looks like there was a **huge increase in Avocado prices as the demand was little high in Year 2017 in SanFranciso region. If you'll search it on google, you'll find the same.**
- **Q.7 How price is distributed over the date column?**
- Now lets do some plots!! I'll start by plotting the Avocado's Average Price through the Date column

[21]:
```
byDate=df.groupby('Date').mean()
plt.figure(figsize=(12,8))
byDate['AveragePrice'].plot()
plt.title('Average Price')
```

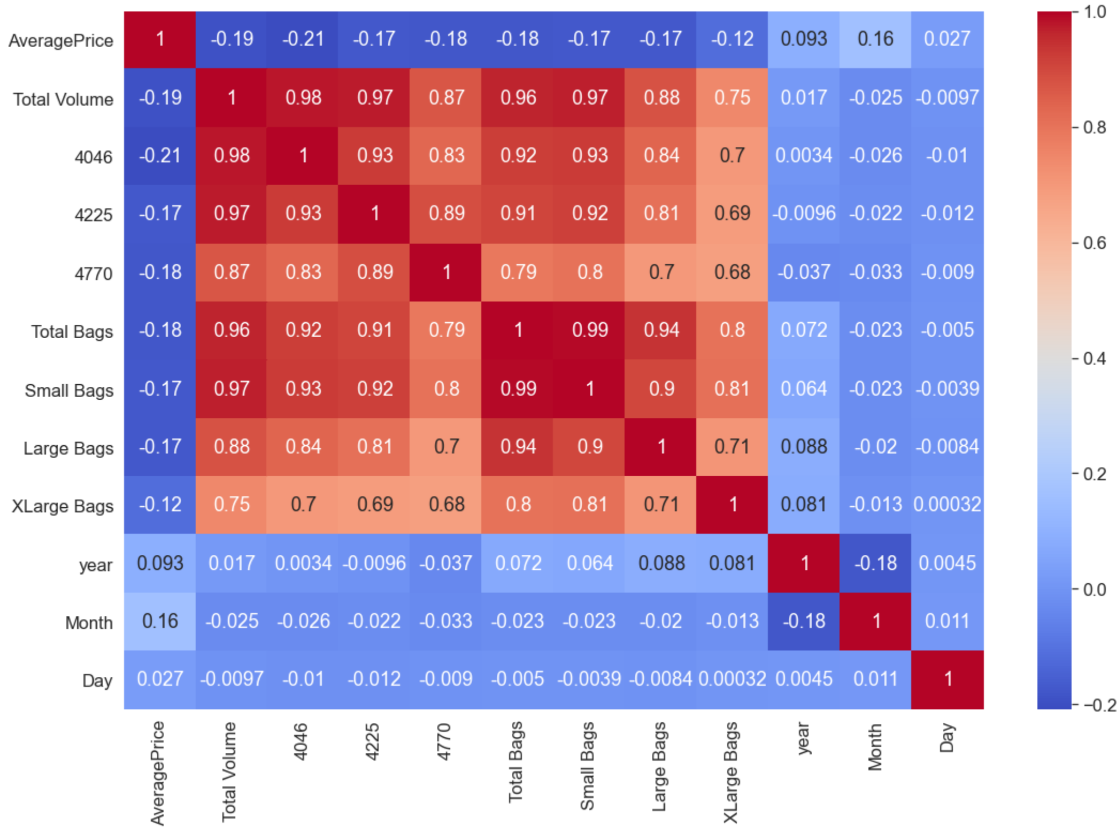
[21]: Text(0.5, 1.0, 'Average Price')



- This also shows there was a huge hike in prices after **July 2017 and before Jan 2018**. This was also confirmed in earlier graph too.
- Cool right? now lets have an idea about the relationship between our Features(Correlation)
- **Q.8 How dataset features are correlated with each other?**

[23]:
```
plt.figure(figsize=(18,12))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True)
```

[23]: <matplotlib.axes._subplots.AxesSubplot at 0x15d5c8fe400>



- As we can from the heatmap above, all the Features are not correleted with the **Average Price column**, instead most of them are correlated with each other.

1.2 * Feature Engineering for Model building

[24]:
```
df['region'].nunique()
```

[24]: 54

[25]:
```
df['type'].nunique()
```

[25]: 2

- As we can see we have **54 regions** and **2 unique types**, so it's going to be easy to to transform the **type feature** to dummies, but for the region its going to be a bit complex, so I decided to drop the entire column.

- I will drop the Date Feature as well because I already have **3 other columns for the Year, Month and Day**.

[26]:
```
df_final=pd.get_dummies(df.drop(['region','Date'],axis=1),drop_first=True)
```

[27]:
```
df_final.head()
```

[27]:
```
   AveragePrice  Total Volume     4046       4225    4770  Total Bags  \
0          1.33      64236.62  1036.74   54454.85   48.16     8696.87
1          1.35      54876.98   674.28   44638.81   58.33     9505.56
2          0.93     118220.22   794.70  109149.67  130.50     8145.35
3          1.08      78992.15  1132.00   71976.41   72.58     5811.16
4          1.28      51039.60   941.48   43838.39   75.78     6183.95

   Small Bags  Large Bags  XLarge Bags  year  Month  Day  type_organic
0     8603.62       93.25          0.0  2015     12   27             0
1     9408.07       97.49          0.0  2015     12   20             0
2     8042.21      103.14          0.0  2015     12   13             0
3     5677.40      133.76          0.0  2015     12    6             0
4     5986.26      197.69          0.0  2015     11   29             0
```

[28]:
```
df_final.tail()
```

[28]:
```
       AveragePrice  Total Volume     4046     4225    4770  Total Bags  \
18244          1.63      17074.83  2046.96  1529.20    0.00    13498.67
18245          1.71      13888.04  1191.70  3431.50    0.00     9264.84
18246          1.87      13766.76  1191.92  2452.79  727.94     9394.11
18247          1.93      16205.22  1527.63  2981.04  727.01    10969.54
18248          1.62      17489.58  2894.77  2356.13  224.53    12014.15

       Small Bags  Large Bags  XLarge Bags  year  Month  Day  type_organic
18244    13066.82      431.85          0.0  2018      2    4             1
18245     8940.04      324.80          0.0  2018      1   28             1
18246     9351.80       42.31          0.0  2018      1   21             1
18247    10919.54       50.00          0.0  2018      1   14             1
18248    11988.14       26.01          0.0  2018      1    7             1
```

1.3 * Model selection/predictions

- Now our data are ready! lets apply our model which is going to be the **Linear Regression because our Target variable 'AveragePrice' is continuous**.
- Let's now begin to train out regression model! We will need to first split up our data into an **X array that contains the features to train on**, and a **y array with the target variable**.

- **P.1 Are we good with Linear Regression? Lets find out.**

[29]:
```python
X=df_final.iloc[:,1:14]
y=df_final['AveragePrice']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪2,random_state=42)
```

- Creating and Training the Model

[30]:
```python
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
pred=lr.predict(X_test)
```

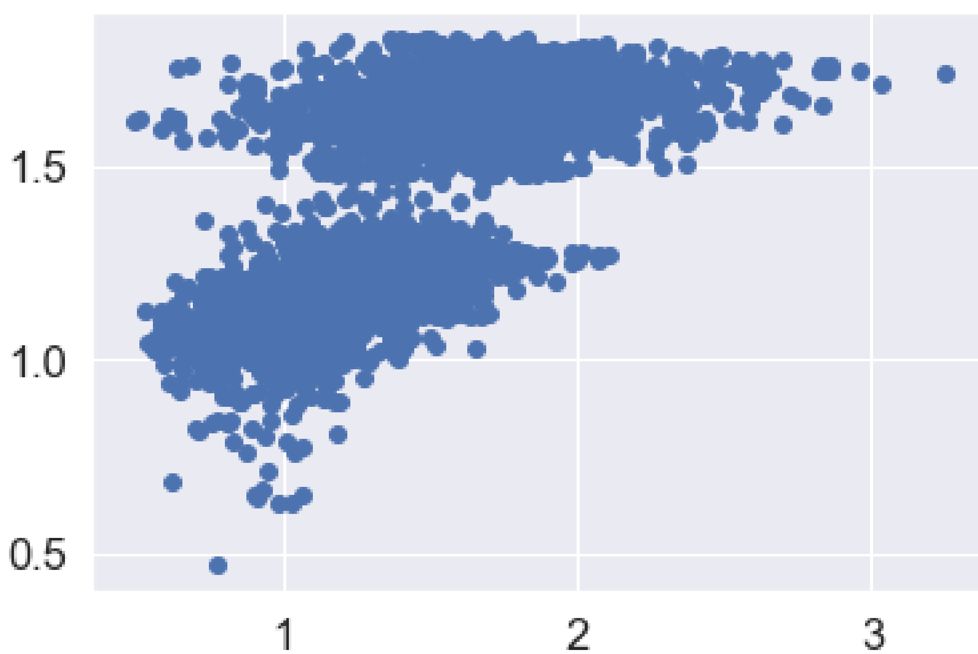
[31]:
```python
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.2329713329170077
MSE: 0.0910880280536491
RMSE: 0.3018079323902026
```

- The **RMSE is low so we can say that we do have a good model, but lets check to be more sure**.
- Lets plot the **y_test vs the predictions**

[32]:
```python
plt.scatter(x=y_test,y=pred)
```

[32]: <matplotlib.collections.PathCollection at 0x15d5e7cc9d0>



- As we can see that **we don't have a straight line so I am not sure that this is the best model we can apply on our data**
- **P.2 Are we good with Decision Tree Regression? Lets find out.**

[33]:
```python
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
dtr.fit(X_train,y_train)
pred=dtr.predict(X_test)
```

[34]:
```python
plt.scatter(x=y_test,y=pred)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

[34]: Text(0, 0.5, 'Predicted Y')

- Nice, here we can see that **we nearly have a straight line**, in other words **its better than the Linear regression model, and to be more sure lets check the RMSE**

[35]:
```python
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.13646301369863012
MSE: 0.04481915068493151
RMSE: 0.2117053392924503
```

- Very Nice, our **RMSE is lower than the previous one we got with Linear Regression**. Now I am going to try one last model to see if I can **improve my predictions for this data which is the RandomForestRegressor**
- **P.3 Are we good with Random Forest Regressor? Lets find out.**

[36]:
```
from sklearn.ensemble import RandomForestRegressor
rdr = RandomForestRegressor()
rdr.fit(X_train,y_train)
pred=rdr.predict(X_test)
```

[37]:
```
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.10090912328767121
MSE: 0.021606730369863002
RMSE: 0.14699227996688466
```

- Well as we can see the **RMSE is lower than the two previous models**, so the **RandomForest Regressor is the best model in this case.**

[38]:
```
sns.distplot((y_test-pred),bins=50)
```

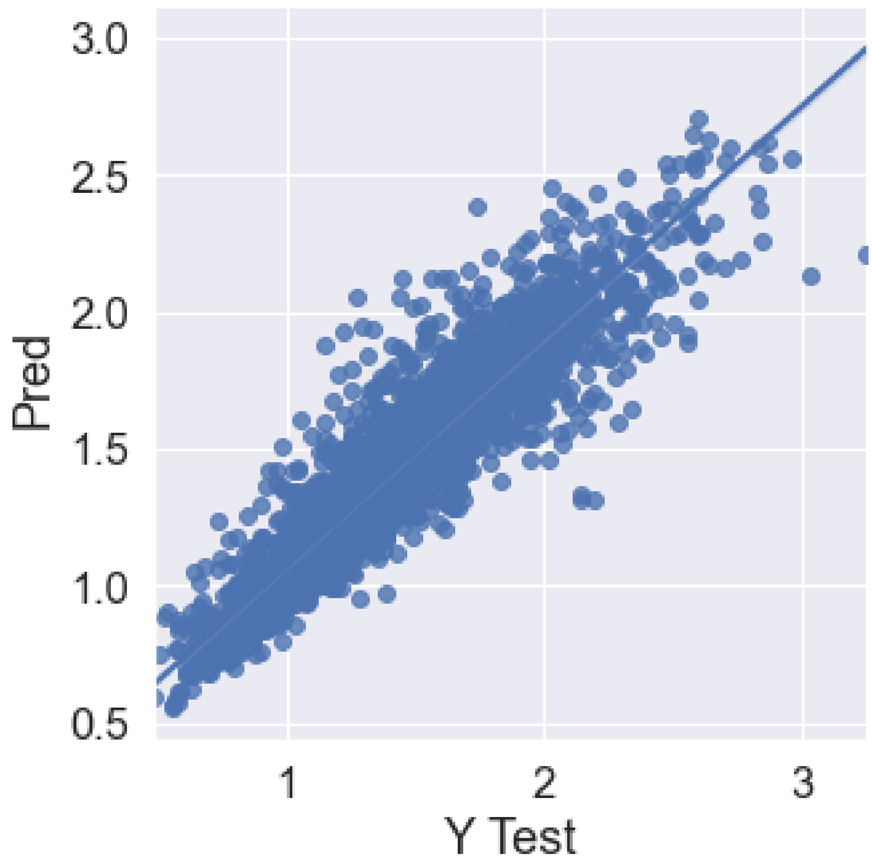
[38]: <matplotlib.axes._subplots.AxesSubplot at 0x15d5fd24c40>

- Notice here that our **residuals looked to be normally distributed** and that's really a **good sign which means that our model was a correct choice for the data.**
- **Lets see final Actual Vs Predicted sample.**

[39]:
```
data = pd.DataFrame({'Y Test':y_test , 'Pred':pred},columns=['Y Test','Pred'])
sns.lmplot(x='Y Test',y='Pred',data=data,palette='rainbow')
data.head()
```

[39]:

	Y Test	Pred
8604	0.82	0.9310
2608	0.97	1.0016
14581	1.44	1.3943
4254	0.97	0.9001
16588	1.45	1.4467



1.4 * Conclusions

- I have seen the impact of columns like **type, year/date** on the **Average price increase/decrease rate**.
- The most important inference drawn from all this analysis is, I get to know what are the **features on which price is highly positively and negatively coorelated with.**
- I came to know through analysis which model will be work with better accuracy with the help of **low residual and RMSE scores**.
- This project helped me to gain insights and how I should go with flow, which model to choose first and go step by step to attain results with good accuracy. Also get to know **where to use Linear, Decision Tree and other applicable and required models to fine tune the predictions**.

```
[ ]:
```