Avacado Project

Problem Statement:

This data was downloaded from the Hass Avocado Board website in May of 2018 & compiled into a single CSV.

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.

Some relevant columns in 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

Inspiration /Label

The dataset can be seen in two angles to find the region and find the average price .

Task: One of Classification and other of Regression

Do both tasks in the same .ipynb file and submit at single file.

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'

Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import plotly.offline as py
import plotly.graph_objs as go
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Importing Dataset

In [2]:
```
df=pd.read_csv('avocado.csv')
df.head()
```

Out[2]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69

- 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

In [3]:
```
df.tail()
```

Out[3]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bag
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.8
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.8
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.3
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.0
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.0

In [4]:
```
df.shape
```

Out[4]:
```
(18249, 14)
```

- rows= 18149
- columns=14

In [5]:
```
df.columns  # This will print the names of all columns.
```

Out[5]:
```
Index(['Unnamed: 0', 'Date', 'AveragePrice', 'Total Volume', '4046', '4225',
       '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type',
       'year', 'region'],
      dtype='object')
```

In [6]:
```
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)

In [7]:
```
# 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()
```

Out[7]:

	Unnamed: 0	AveragePrice	Total Volume	4046	4225	4770	Tota
count	18249.000000	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.82490
mean	24.232232	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04	2.39639
std	15.481045	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.86242
min	0.000000	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
25%	10.000000	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000e+00	5.08864
50%	24.000000	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02	3.97438
75%	38.000000	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03	1.10783
max	52.000000	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06	1.93731

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

In [8]:
```
df.isnull().sum()  # Will show you null count for each column, but will not count Zero
```

Out[8]:
```
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!

Preprocessing

In [9]:
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

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

In [10]:
```
df.head() #check our data head again to make sure that the Feature Unnamed:0 is remove
```

Out[10]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conv
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conv
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conv
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conv
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conv

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

In [11]:
```
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)
```

In [12]:
```
df.head() #check the head to see what we have done
```

Out[12]:

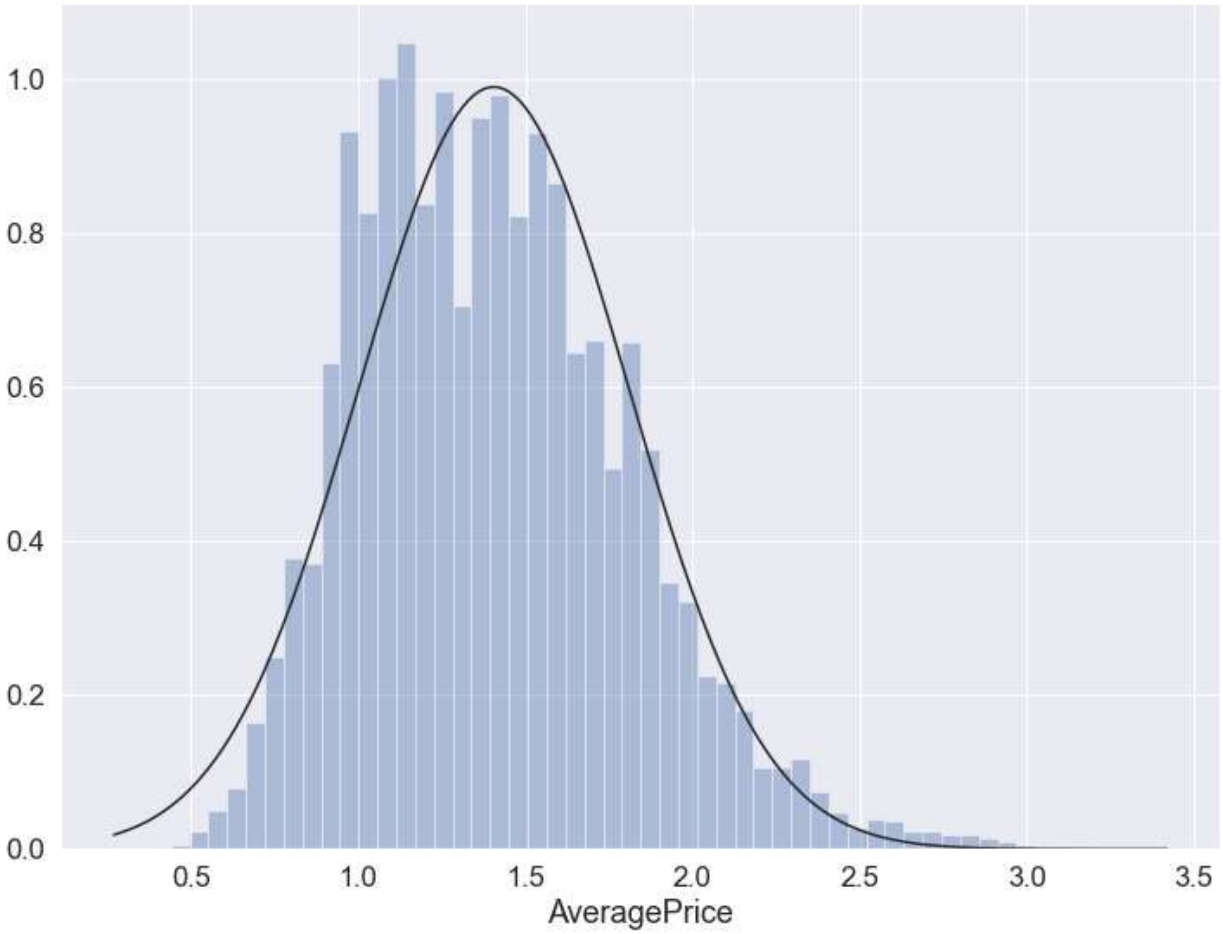
	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conv
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conv
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conv
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conv
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conv

Data Visualisation

In [14]:
```
sns.set(font_scale=1.5)
from scipy.stats import norm
```

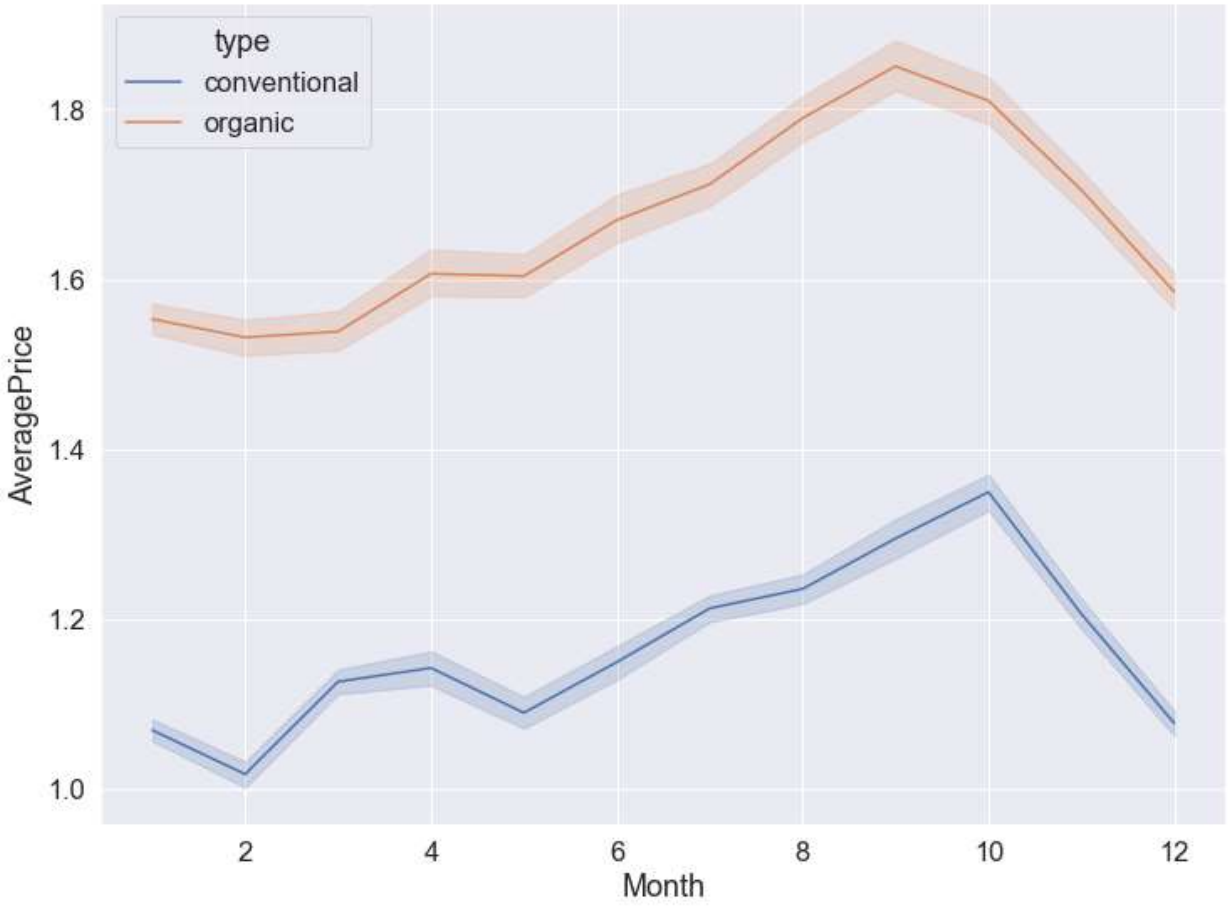
```
fig, ax = plt.subplots(figsize=(12, 9))
sns.distplot(a=df.AveragePrice, kde=False, fit=norm)
```

Out[14]: <AxesSubplot:xlabel='AveragePrice'>



- Average Price distribution shows that for most cases price of avocado is between 1.1, 1.4.
- Let's look at average price of conventional vs. organic.

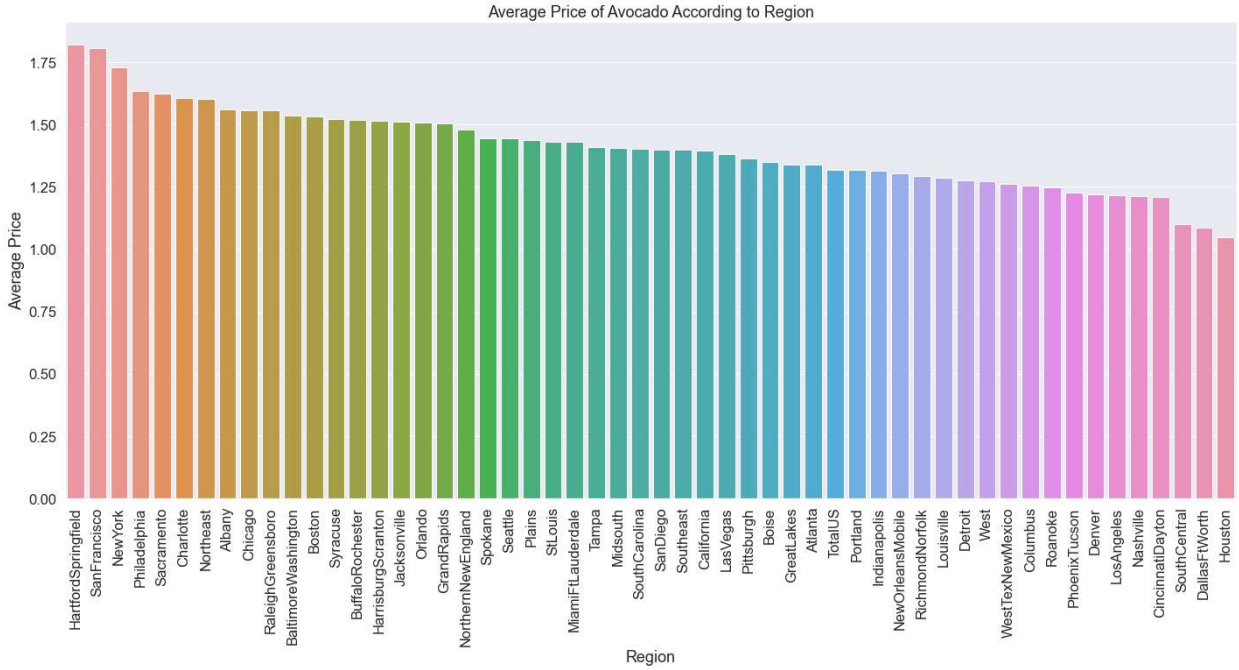
In [15]:
```
plt.figure(figsize=(12,9))
sns.lineplot(x="Month", y="AveragePrice", hue='type', data=df)
plt.show()
```



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

In [16]:
```
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')
```

Out[16]:
```
Text(0.5, 1.0, 'Average Price of Avocado According to Region')
```



- Looks like these region are where price is very high

  1.HartfordSpringfield

  2.SanFrancisco

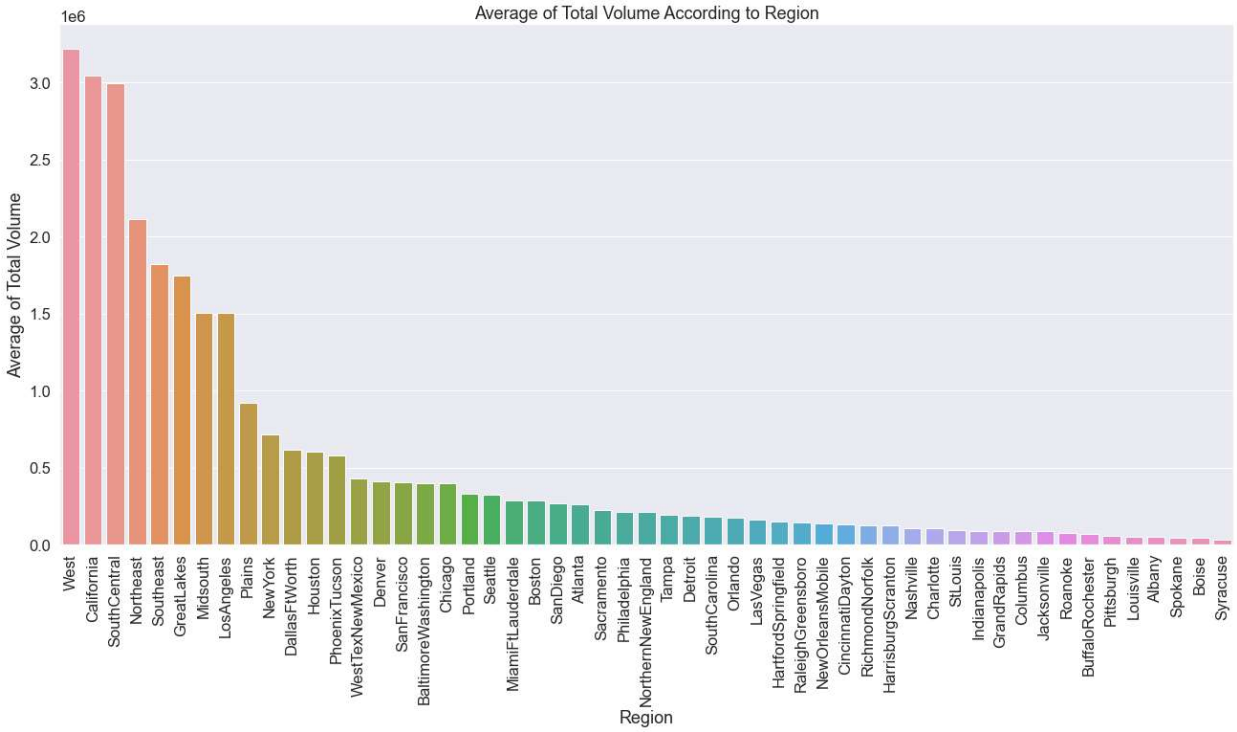
  3.NewYork

  4.Philadelphia

  5.Sacramento

In [17]:
```
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_volum

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')
```

Out[17]:
```
Text(0.5, 1.0, 'Average of Total Volume According to Region')
```



- Looks like these region are where Consumption is very high

  West

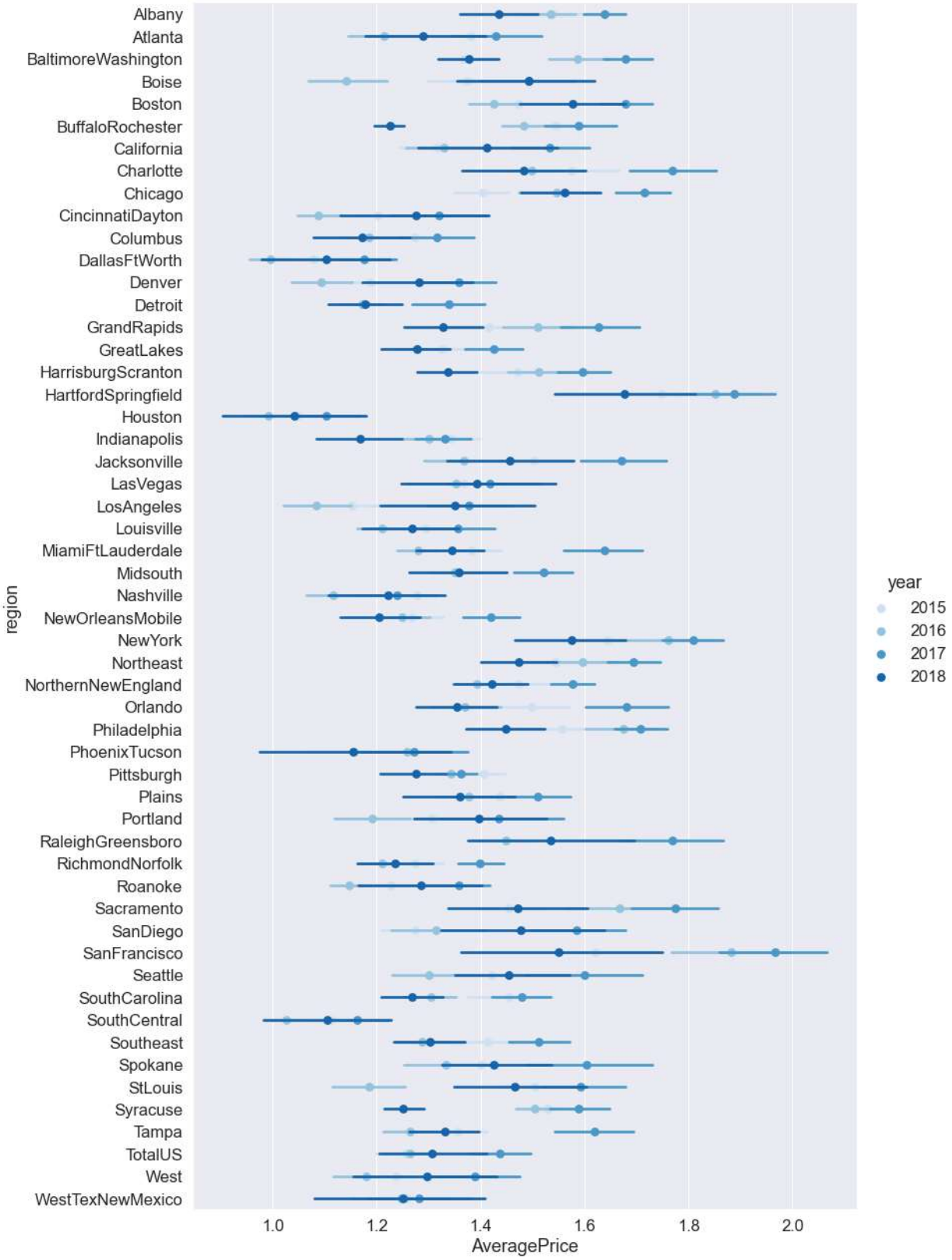
  California

  SouthCentral

  Northeast

  Southeast

In [18]:
```
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.

In [19]:
```
#Now lets do some plots!! I'll start by plotting the Avocado's Average Price through t
byDate=df.groupby('Date').mean()
plt.figure(figsize=(12,8))
```

```
byDate['AveragePrice'].plot()
plt.title('Average Price')
```

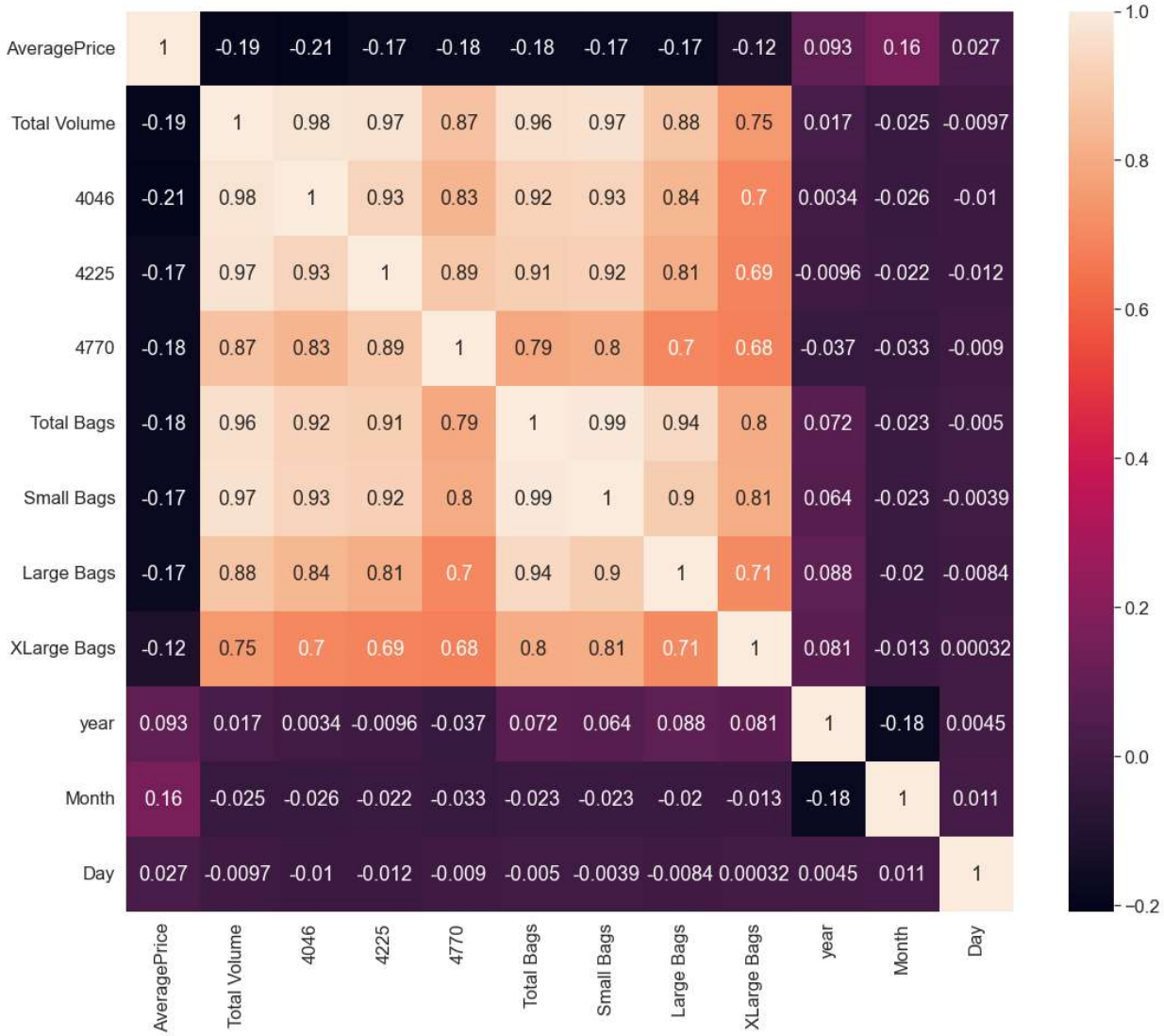
Out[19]:
```
Text(0.5, 1.0, 'Average Price')
```

- 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)

Correlation matrix

In [20]:
```
plt.figure(figsize=(18,15))
sns.heatmap(df.corr(),annot=True)
```

Out[20]:
```
<AxesSubplot:>
```



- 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. So now I am bit worried because that will not help us get a good model. Lets try and see.
- First we have to do some Feature Engineering on the categorical Features : region and type

Feature Engineering for Model building

In [21]:
```
df['region'].nunique()
```

Out[21]:
```
54
```

In [22]:
```
df['type'].nunique()
```

Out[22]:
```
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.

In [23]:
```
df_final=pd.get_dummies(df.drop(['region','Date'],axis=1),drop_first=True)
```

In [24]:
```
df_final.head()
```

Out[24]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	year	Mon
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	2015	
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	2015	
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	2015	
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	2015	
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	2015	

Model selection and 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.

In [25]:
```
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

Linear Regression

In [26]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
pred=lr.predict(X_test)
```

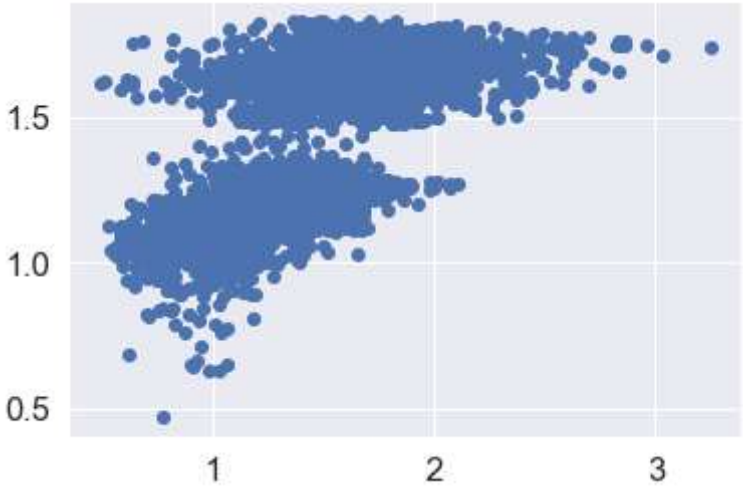
In [27]:
```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.23297133291682995
MSE: 0.09108802805350073
RMSE: 0.3018079323899568
```

- 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

In [28]:
```
plt.scatter(x=y_test,y=pred)
```

Out[28]:
```
<matplotlib.collections.PathCollection at 0x1a3ee844370>
```



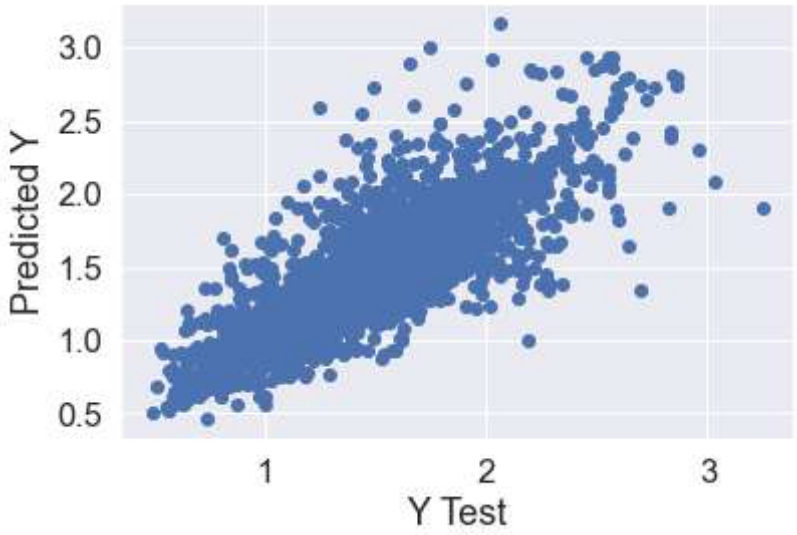
- As we can see that we don't have a straight line so I am not sure that this is the best model we can apply on our data
- Lets try working with the Decision Tree Regression model

Decision Tree Regressor

In [29]:
```
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
dtr.fit(X_train,y_train)
pred=dtr.predict(X_test)
```

In [30]:
```
plt.scatter(x=y_test,y=pred)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[30]:
```
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

In [31]:
```
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: 0.1352931506849315
MSE: 0.04398482191780822
RMSE: 0.20972558717955284
```

- 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

Random Forest Regressor

In [32]:
```
from sklearn.ensemble import RandomForestRegressor
rdr = RandomForestRegressor()
rdr.fit(X_train,y_train)
pred=rdr.predict(X_test)
```

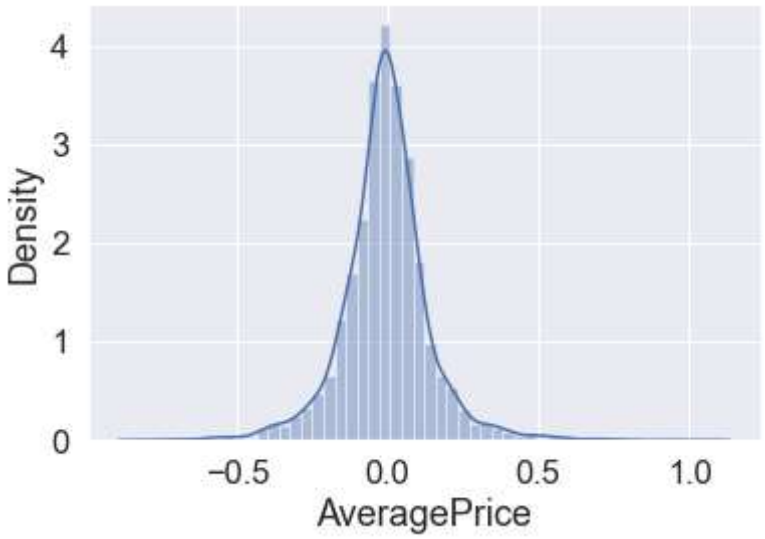
In [33]:
```
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.10070301369863009
MSE: 0.02133017963835615
RMSE: 0.14604855233228486
```

- 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.

In [34]:
```
sns.distplot((y_test-pred),bins=50)
```

Out[34]: <AxesSubplot:xlabel='AveragePrice', ylabel='Density'>



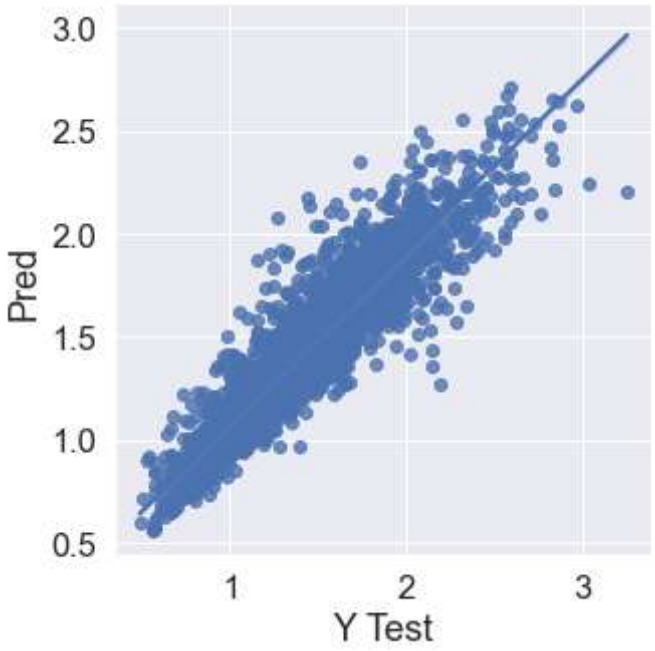
- Notice here that our residuals looked to be normally distributed and that's really a good sign which means that our model was a correct choice for the data.

Lets see final Actual Vs Predicted sample.

In [35]:
```
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()
```

Out[35]:

	Y Test	Pred
8604	0.82	0.9590
2608	0.97	0.9978
14581	1.44	1.3985
4254	0.97	0.9042
16588	1.45	1.4548



Conclusions

- With the help of notebook I learnt how EDA can be carried out using Pandas and other plotting libraries.
- Also I have seen making use of packages like matplotlib, plotly and seaborn to develop better insights about the data.
- I have also seen how preproceesing helps in dealing with missing values and irregualities present in the data. I also learnt how to create new features which will in turn help us to better predict the survival.
- I also make use of pandas profiling feature to generate an html report containing all the information of the various features present in the dataset.
- 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.

In [ ]: