Predicting the prices of Avocados

About the data-

> The dataset 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

In [1]:

```
# display image using python
from PIL import Image
from IPython.display import display
img = Image.open(r"C:\Users\ankus\Desktop\NareshIT\2. Notes\11.Machine learning\
display(img)
```

Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# import dataset
data = pd.read_csv(r"C:\Users\ankus\Desktop\NareshIT\2. Notes\11.Machine learnin
data.head()
```

Out[3]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

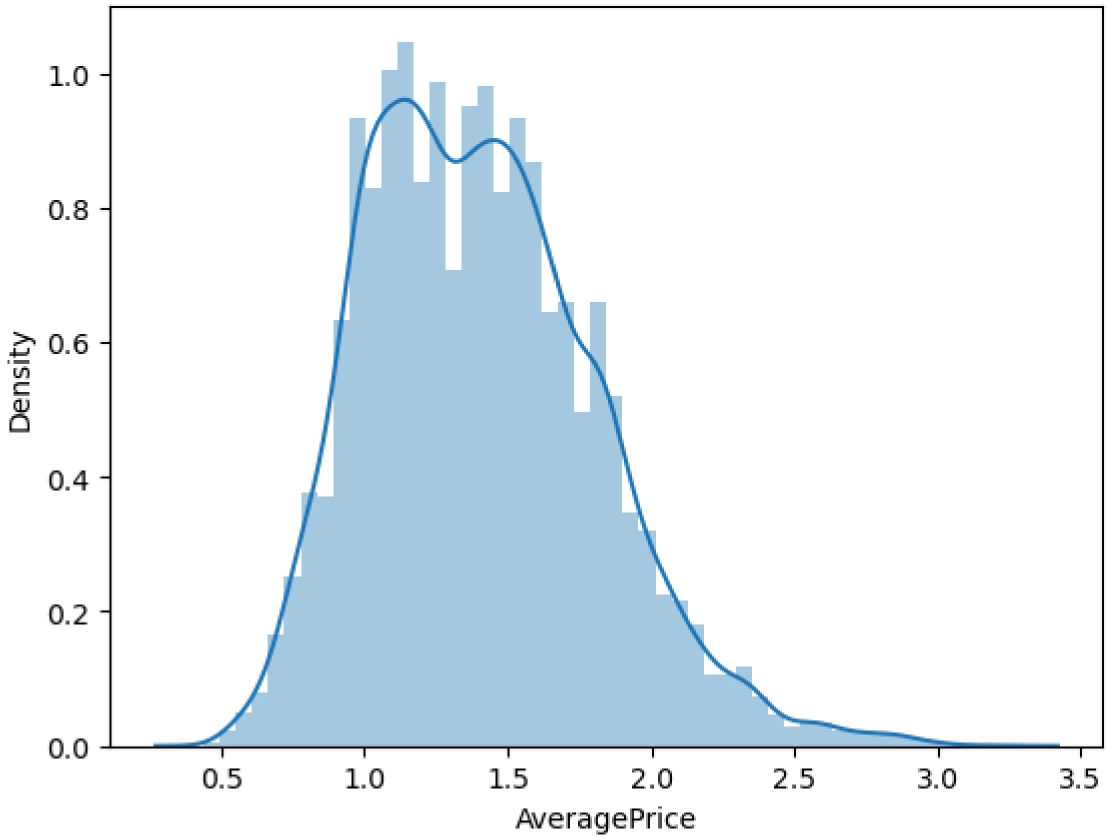
In [4]:
```python
data.info()
```

```
<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
```

There are three categorical features and luckily no missing value.Let's explore the data further.

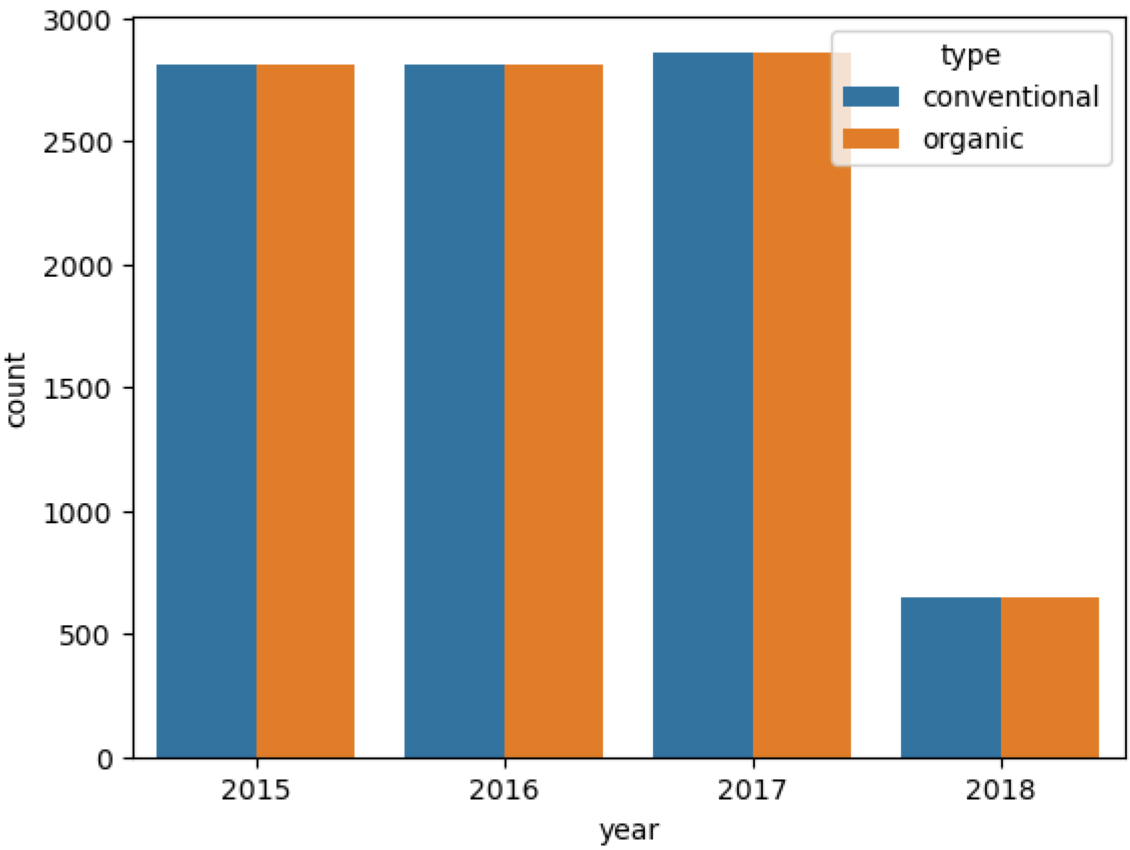
In [5]:
```python
sns.distplot(data['AveragePrice'])
```

Out[5]: <Axes: xlabel='AveragePrice', ylabel='Density'>



In [6]:
```python
sns.countplot(x='year',data=data,hue='type')
```

Out[6]: <Axes: xlabel='year', ylabel='count'>



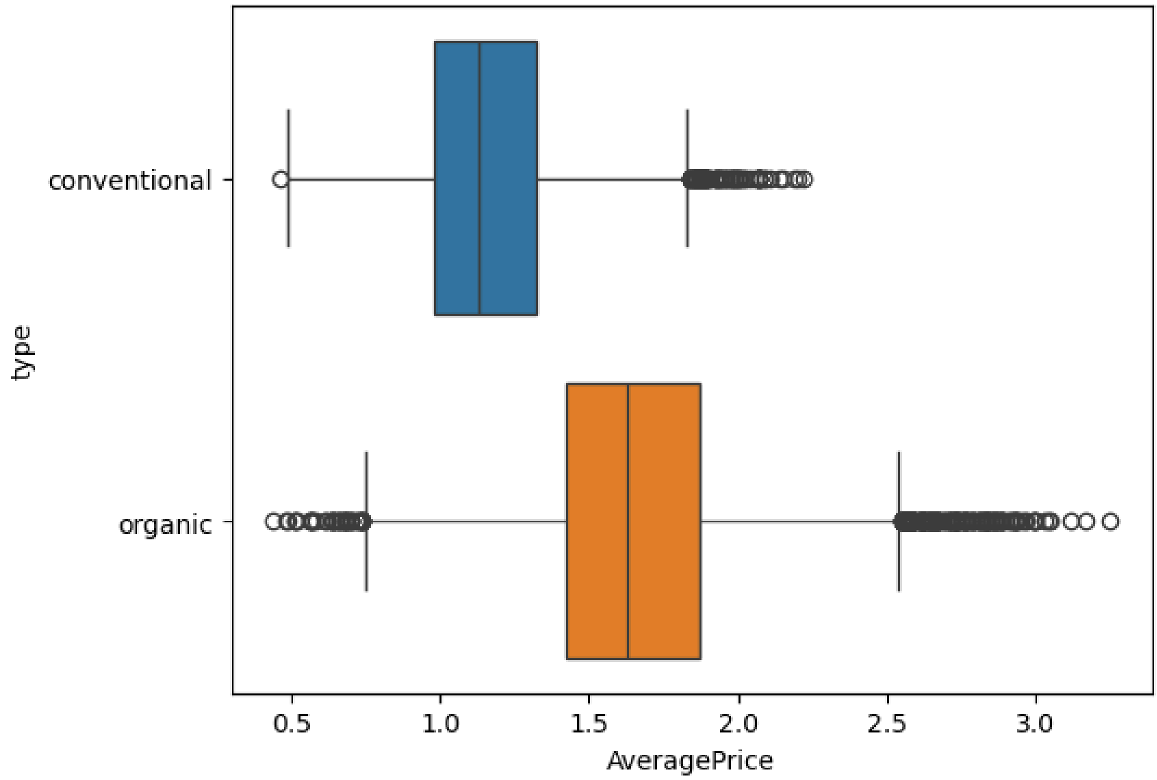
There are almost equal numbers of conventional and organic avacados.Though,there is very less observation in the year 2018

In [7]:
```
data.year.value_counts()
```

Out[7]:
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```

In [8]:
```
sns.boxplot(y='type',x='AveragePrice',data=data,hue='type')
```

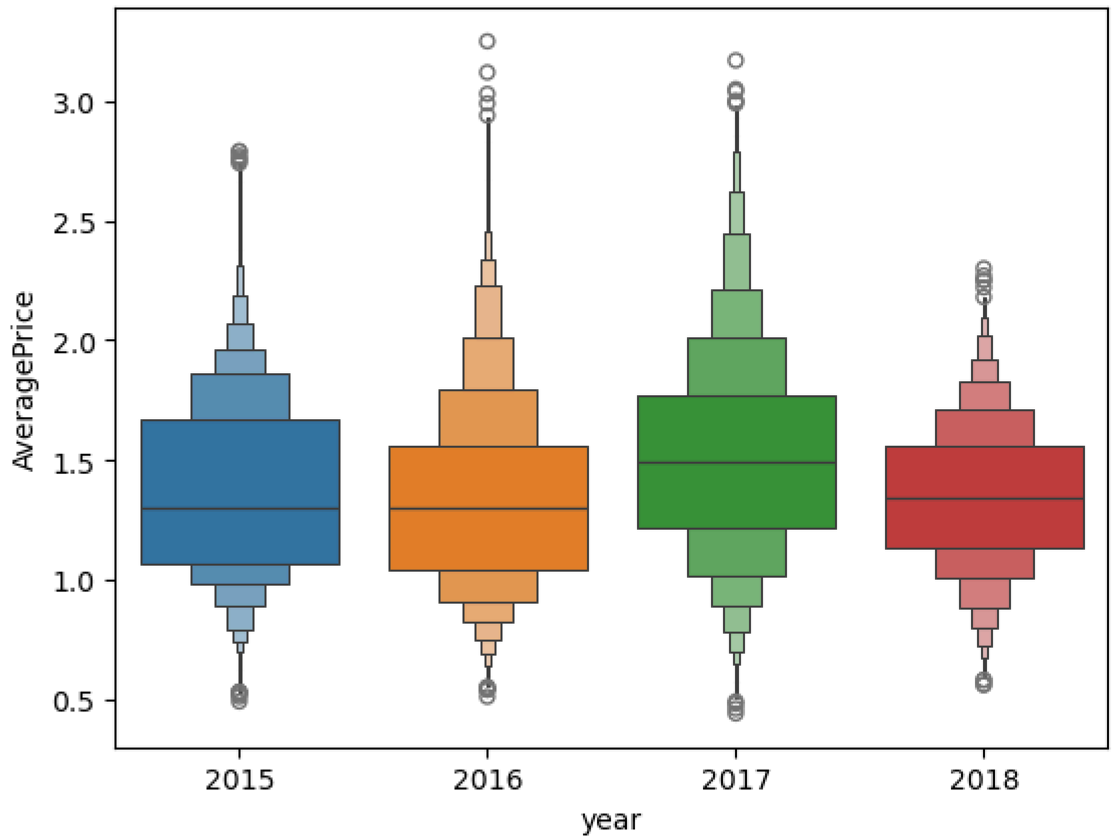
Out[8]: <Axes: xlabel='AveragePrice', ylabel='type'>



Organic avocados are more expensive. This is obvious, because their cultivation is more expensive and we all love natural products and are willing to pay a higher price for them.

In [9]:
```
data.year = data.year.apply(str)
sns.boxenplot(x='year',y='AveragePrice',data=data,hue='year')
```

Out[9]: `<Axes: xlabel='year', ylabel='AveragePrice'>`



Avacados were slightly more expensive in the year 2017.(as there was shortage due to some reasons)

Dealing with categorical features

In [10]:
```
data['type'] = data['type'].map({'conventional':0,'organic':1})

# Extracting month from date column
data.Date = data.Date.apply(pd.to_datetime)
data['Month'] = data['Date'].apply(lambda x:x.month)
data.drop('Date',axis=1,inplace=True)
data.Month = data.Month.map({1:'JAN',2:'FEB',3:'MARCH',4:'APRIL',5:'MAY',6:'JUNE
```

In [20]:
```
plt.figure(figsize=(9,5))
sns.countplot(x='count',y='Month',data=data)
plt.title('Monthwise Distribution of Sales',fontdict = {'fontsize':25})
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 2
      1 plt.figure(figsize=(9,5))
----> 2 sns.countplot(x='count',y='Month',data=data)
      3 plt.title('Monthwise Distribution of Sales',fontdict = {'fontsize':25})

File ~\anaconda3\Lib\site-packages\seaborn\categorical.py:2629, in countplot(dat
a, x, y, hue, order, hue_order, orient, color, palette, saturation, fill, hue_nor
m, stat, width, dodge, gap, log_scale, native_scale, formatter, legend, ax, **kwa
rgs)
   2627     y = 1 if list(x) else None
   2628 elif x is not None and y is not None:
-> 2629     raise TypeError("Cannot pass values for both `x` and `y`.")
   2631 p = _CategoricalAggPlotter(
   2632     data=data,
   2633     variables=dict(x=x, y=y, hue=hue),
   (...)
   2637     legend=legend,
   2638 )
   2640 if ax is None:

TypeError: Cannot pass values for both `x` and `y`.
```

```
<Figure size 900x500 with 0 Axes>
```

It implies that sales of avacado see a rise in January,February and March

Preparing data for Ml models

In [12]:
```
# creating dummy variables
dummies = pd.get_dummies(data[['year','region','Month']],drop_first=True)
data_dummies = pd.concat([data[['Total Volume','4046','4225','4770','Total Bags'
                                'Large Bags','XLarge Bags','type']],dummies],axi
target = data['AveragePrice']
```

In [13]:
```
# Splitting data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(data_dummies,target,test_size=0
```

```
#Standardizing the data
cols_to_std = ['Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train[cols_to_std])
X_train[cols_to_std] = scaler.transform(X_train[cols_to_std])
X_test[cols_to_std] = scaler.transform(X_test[cols_to_std])
```

importing ML models from scikit-learn

In [16]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [18]:
```
#to save time all models can be applied once using for loop
regressors = {
    'Linear Regression' : LinearRegression(),
    'Decision Tree' : DecisionTreeRegressor(),
    'Random Forest' : RandomForestRegressor(),
    'Support Vector Machines' : SVR(gamma=1),
    'K-nearest Neighbors' : KNeighborsRegressor(n_neighbors=1),
    'XGBoost' : XGBRegressor()
}

results = pd.DataFrame(columns=['MAE','MSE','R2-score'])
for method,func in regressors.items():
    model = func.fit(X_train,y_train)
    pred = model.predict(X_test)
    results.loc[method] = [np.round(mean_absolute_error(y_test,pred),3),
                           np.round(mean_squared_error(y_test,pred),3),
                           np.round(r2_score(y_test,pred),3)
                          ]
results
```

Out[18]:

	MAE	MSE	R2-score
Linear Regression	0.180	0.058	0.635
Decision Tree	0.133	0.043	0.732
Random Forest	0.096	0.019	0.878
Support Vector Machines	0.116	0.027	0.830
K-nearest Neighbors	0.100	0.023	0.854
XGBoost	0.094	0.017	0.895

Conclusion:

- Except linear regression model, all other models have mean absolute error less than 10% of mean of target variable.

- For this dataset, XGBoost and Random Forest algorithms have shown best results.

```
In [ ]:
```