Predicting the price of Avacados

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

Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings("ignore")
```

Import Dataset

In [2]:
```
data = pd.read_csv(r'E:\NIT(Data Science)by (prakash senapati sir (kodi))\Data science all class\August all
```

check the data

In [3]:
```
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
```

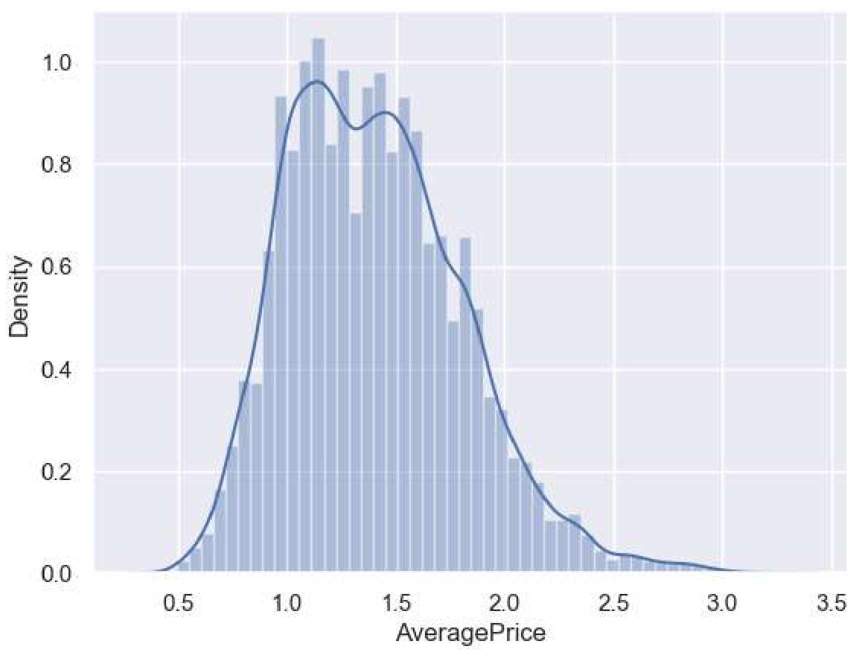
There are 3 categorical features and luckily no missing value. Let's explore the data further.

In [4]:
```
data.head(3)
```

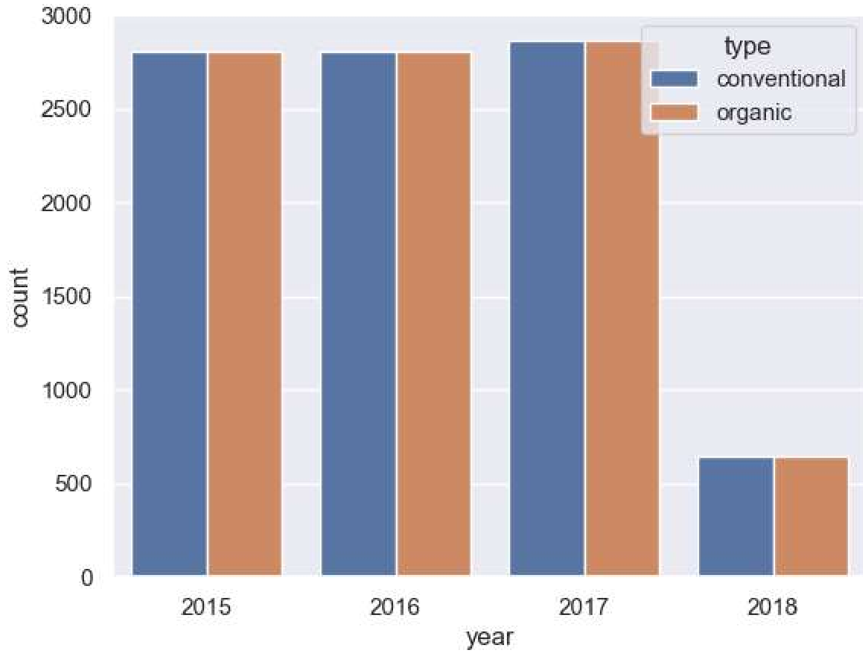
Out[4]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	regio
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albar
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albar
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albar

In [5]:
```
sns.distplot(data["AveragePrice"]);
```



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

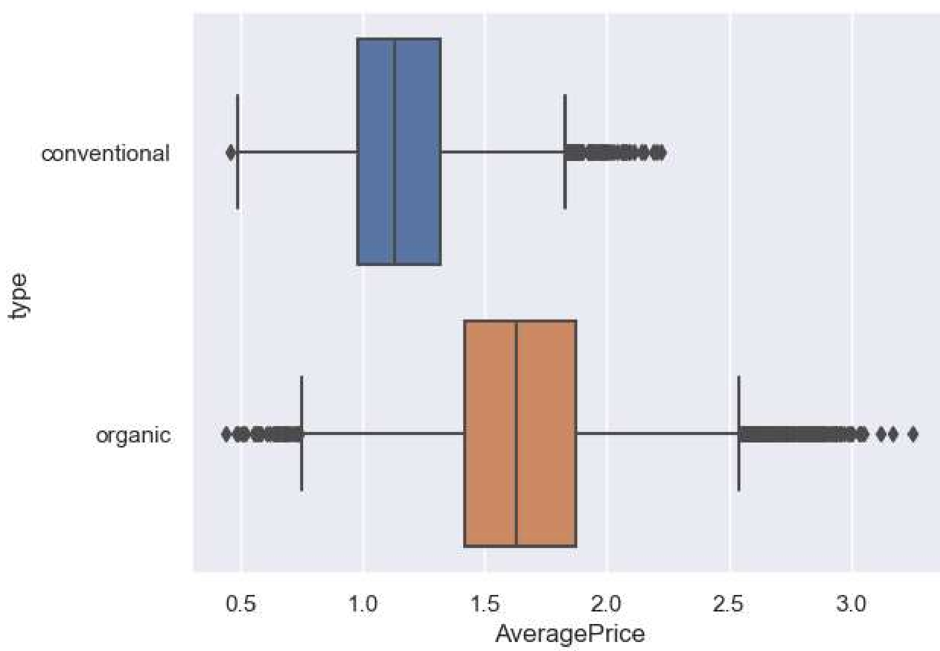


There are almost equal numbers of conventional and organic avacados. Though, there is very less observations in the year 2018.

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

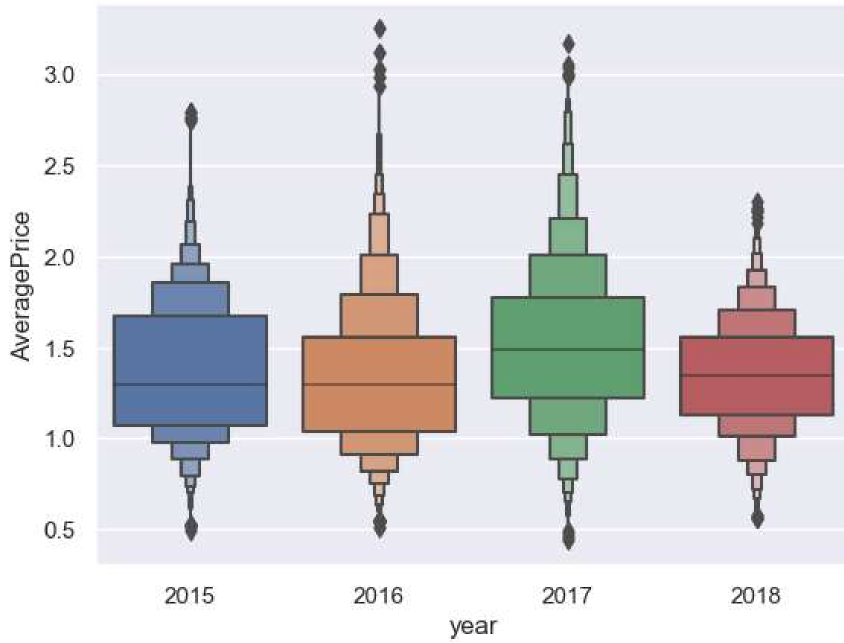
Out[7]:
```
2017    5722
2016    5616
2015    5615
2018    1296
Name: year, dtype: int64
```

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



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);
```



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 data column

In [11]:
```
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",7:"JULY",8:"AUG",9:"SEPT"
```

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(9, 5))
sns.barplot(x=data["Month"].value_counts().index, y=data["Month"].value_counts().values)
plt.title("Monthwise Distribution of Sales", fontdict={"fontsize": 25})
plt.xlabel("Month")
plt.ylabel("Number of Sales")
plt.xticks(rotation=45)
plt.show()
```

Preparing data for Ml MODEL

creating dummy variable

In [13]:
```
dummies = pd.get_dummies(data[["year","region","Month"]],drop_first=True)
df_dummies = pd.concat([data[["Total Volume", "4046", "4225", "4770", "Total Bags",
                              "Small Bags", "Large Bags", "XLarge Bags", "type"]],dummies],axis=1)
target = data["AveragePrice"]
```

Splitting data into training and test set

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_dummies,target,test_size=0.30)
```

Standardizing the data

In [15]:
```
cols_to_std = ["Total Volume", "4046", "4225", "4770", "Total Bags","Large Bags", "XLarge 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
```

In [17]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [18]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [19]:
```
from sklearn.svm import SVR
```

In [20]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [21]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in e:\anaconda3\lib\site-packages (1.7.6)
Requirement already satisfied: numpy in e:\anaconda3\lib\site-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in e:\anaconda3\lib\site-packages (from xgboost) (1.10.0)
Note: you may need to restart the kernel to use updated packages.
```

In [22]:
```
from xgboost import XGBRegressor
```

In [23]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

to save time all models can be applied once using for loop

In [24]:
```
regressors = {
    "Linear Regression" : LinearRegression(),
    "Decision Tree" : DecisionTreeRegressor(),
    "Random Forest" : RandomForestRegressor(),
    "Support Vactor Machines" : SVR(gamma=1),
    "K-nearest Nighbors" : 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)
                         ]
```

Deep Neural Network

Splitting train set into train and validation sets.

In [25]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.20)
```

In [29]:

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in e:\anaconda3\lib\site-packages (2.13.0)
Requirement already satisfied: tensorflow-intel==2.13.0 in e:\anaconda3\lib\site-packages (from tensorf
low) (2.13.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in e:\anaconda3\lib\site-packages (from tensorflow-i
ntel==2.13.0->tensorflow) (1.57.0)
Requirement already satisfied: tensorboard<2.14,>=2.13 in e:\anaconda3\lib\site-packages (from tensorfl
ow-intel==2.13.0->tensorflow) (2.13.0)
Requirement already satisfied: keras<2.14,>=2.13.1 in e:\anaconda3\lib\site-packages (from tensorflow-i
ntel==2.13.0->tensorflow) (2.13.1)
Requirement already satisfied: wrapt>=1.11.0 in e:\anaconda3\lib\site-packages (from tensorflow-intel==
2.13.0->tensorflow) (1.14.1)
Requirement already satisfied: opt-einsum>=2.3.2 in e:\anaconda3\lib\site-packages (from tensorflow-int
el==2.13.0->tensorflow) (3.3.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,
>=3.20.3 in e:\anaconda3\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow) (4.24.2)
Requirement already satisfied: absl-py>=1.0.0 in e:\anaconda3\lib\site-packages (from tensorflow-intel=
=2.13.0->tensorflow) (1.4.0)
Requirement already satisfied: h5py>=2.9.0 in e:\anaconda3\lib\site-packages (from tensorflow-intel==2.
13.0->tensorflow) (3.7.0)
```

Important Tensorflow libraries

In [27]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Creating Model

In [30]:

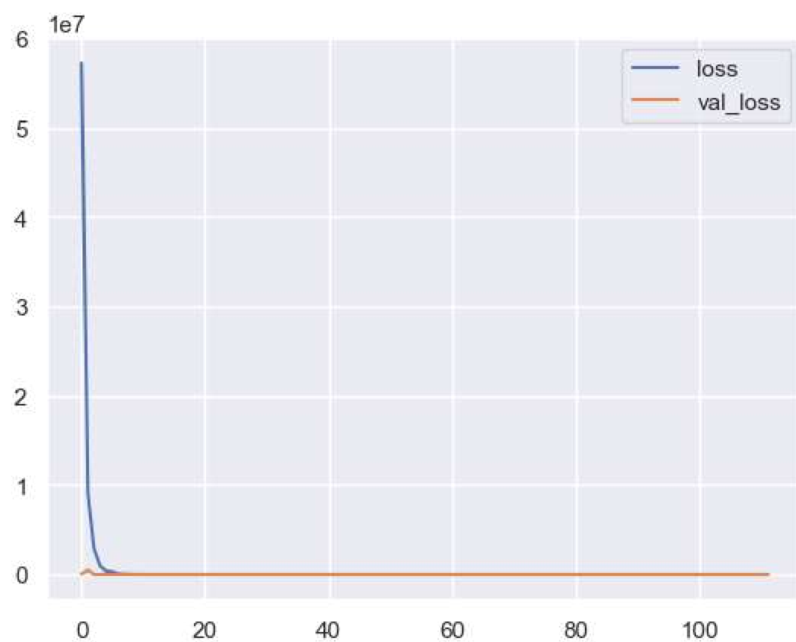
```
model = Sequential()
model.add(Dense(76,activation='relu',kernel_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(1))

model.compile(optimizer='Adam', loss='mean_squared_error')
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=0, patience=10)
```

In [31]:
```
model.fit(x=X_train.values,y=y_train.values,
          validation_data=(X_val.values,y_val.values),
          batch_size=100,epochs=150,callbacks=[early_stop])
```

```
Epoch 1/150
103/103 [==============================] - 5s 13ms/step - loss: 57242268.0000 - val_loss: 52982.3008
Epoch 2/150
103/103 [==============================] - 1s 9ms/step - loss: 9031492.0000 - val_loss: 471898.7812
Epoch 3/150
103/103 [==============================] - 1s 10ms/step - loss: 2872031.0000 - val_loss: 818.7524
Epoch 4/150
103/103 [==============================] - 1s 11ms/step - loss: 911147.7500 - val_loss: 3975.1584
Epoch 5/150
103/103 [==============================] - 1s 12ms/step - loss: 357832.1875 - val_loss: 7247.6147
Epoch 6/150
103/103 [==============================] - 1s 11ms/step - loss: 286699.2500 - val_loss: 3792.0518
Epoch 7/150
103/103 [==============================] - 1s 12ms/step - loss: 76566.8828 - val_loss: 183.0328
Epoch 8/150
103/103 [==============================] - 1s 12ms/step - loss: 40262.3828 - val_loss: 37.2701
Epoch 9/150
103/103 [==============================] - 1s 11ms/step - loss: 20512.2773 - val_loss: 74.5229
Epoch 10/150
```

In [32]:
```
losses = pd.DataFrame(model.history.history)
losses[["loss","val_loss"]].plot();
```



In [33]:
```
dnn_pred = model.predict(X_test)
```

```
172/172 [==============================] - 1s 3ms/step
```

Results Table

In [35]:
```
results.loc['Deep Neural Network']=[mean_absolute_error(y_test,dnn_pred).round(3),mean_squared_error(y_test
                                    r2_score(y_test,dnn_pred).round(3)]
results
```

Out[35]:

	MAE	MSE	R2-score
Linear Regression	0.187	0.061	0.628
Decision Tree	0.128	0.037	0.773
Random Forest	0.097	0.019	0.884
Support Vactor Machines	0.322	0.161	0.022
K-nearest Nighbors	0.358	0.213	-0.292
XGBoost	0.096	0.017	0.894
Deep Neural Network	0.283	0.131	0.205

In [36]:
```
f"10% of mean of target variable is {np.round(0.1 * data.AveragePrice.mean(),3)}"
```

Out[36]: '10% of mean of target variable is 0.141'

In [37]:
```
results.sort_values('R2-score',ascending=False).style.background_gradient(cmap='Greens',subset=['R2-score']
```

Out[37]:

	MAE	MSE	R2-score
XGBoost	0.096000	0.017000	0.894000
Random Forest	0.097000	0.019000	0.884000
Decision Tree	0.128000	0.037000	0.773000
Linear Regression	0.187000	0.061000	0.628000
Deep Neural Network	0.283000	0.131000	0.205000
Support Vactor Machines	0.322000	0.161000	0.022000
K-nearest Nighbors	0.358000	0.213000	-0.292000

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 [ ]: