Avocado Data Analysis

Business Understanding

The aim of this project is to answer the following four questions: 1. Which region are the lowest and highest prices of Avocado? 2. What is the highest region of avocado production? 3. What is the average avocado prices in each year? 4. What is the average avocado volume in each year?

Data Understanding

The Avocado dataset was been used in this project.

This dataset contains 13 columns: 1. Date - The date of the observation 2. AveragePrice: the average price of a single avocado 3. Total Volume: Total number of avocados sold 4. Total Bags: Total number o bags 5. Small Bags: Total number of Small bags 6. Large Bags: Total number of Large bags 7. XLarge Bags: Total number of XLarge bags 8. type: conventional or organic 9. year: the year 10. region: the city or region of the observation 11. 4046: Total number of avocados with PLU 4046 sold 12. 4225: Total number of avocados with PLU 4225 sold 13. 4770: Total number of avocados with PLU 4770 sold

1.Import necessary libraries

import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns from sklearn.linear_model import LinearRegression from sklearn.model_selection import train_test_split from sklearn.metrics import r2_score

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

2.Data preparation

Load data

In [10]:
```
df = pd.read_csv(r"C:\Users\SHAIK BASHEER\Downloads\Avacodo.csv")
```
In [11]:
```
df
```

Out[11]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	
0	0	27-12-2015	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	co
1	1	20-12-2015	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	co
2	2	13-12-2015	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	co
3	3	06-12-2015	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	co
4	4	29-11-2015	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	co
...	...	...	...	...	...	...	...	...	...	...	...	
18244	7	04-02-2018	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	
18245	8	28-01-2018	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	
18246	9	21-01-2018	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	
18247	10	14-01-2018	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	
18248	11	07-01-2018	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	

18249 rows × 14 columns

Explore the data

In [12]:

```
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 [13]:

```
df.head()
```

Out[13]:

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

3.Missing value checking

In [14]:
```
df.isnull().sum()
```
Out[14]:
```
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
```

4.Dropping unnecessary columns

In [15]:
```
df = df.drop(['Unnamed: 0','4046','4225','4770','Date'],axis=1)
```
In [16]:
```
df.head()
```
Out[16]:

	AveragePrice	Total Volume	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	1.33	64236.62	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
1	1.35	54876.98	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
2	0.93	118220.22	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
3	1.08	78992.15	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	1.28	51039.60	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

Answering questions

In [17]:

```python
def get_avarage(df,column):
    """
    Description: This function to return the average value of the column

    Arguments:
        df: the DataFrame.
        column: the selected column.
    Returns:
        column's average
    """
    return sum(df[column])/len(df)
```

In [18]:

```python
def get_avarge_between_two_columns(df,column1,column2):
    """
    Description: This function calculate the average between two columns in the dataset

    Arguments:
        df: the DataFrame.
        column1:the first column.
        column2:the scond column.
    Returns:
        Sorted data for relation between column1 and column2
    """

    List=list(df[column1].unique())
    average=[]

    for i in List:
        x=df[df[column1]==i]
        column1_average= get_avarage(x,column2)
        average.append(column1_average)

    df_column1_column2=pd.DataFrame({'column1':List,'column2':average})
    column1_column2_sorted_index=df_column1_column2.column2.sort_values(ascending=False).index.values
    column1_column2_sorted_data=df_column1_column2.reindex(column1_column2_sorted_index)

    return column1_column2_sorted_data
```

In [19]:

```python
def plot(data,xlabel,ylabel):
    """
    Description: This function to draw a barplot

    Arguments:
        data: the DataFrame.
        xlabel: the label of the first column.
        ylabel: the label of the second column.
    Returns:
        None
    """

    plt.figure(figsize=(15,5))
    ax=sns.barplot(x=data.column1,y=data.column2,palette='rocket')
    plt.xticks(rotation=90)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(('Avarage '+ylabel+' of Avocado According to '+xlabel))
```

5.Which region are the lowest and highest prices of Avocado?

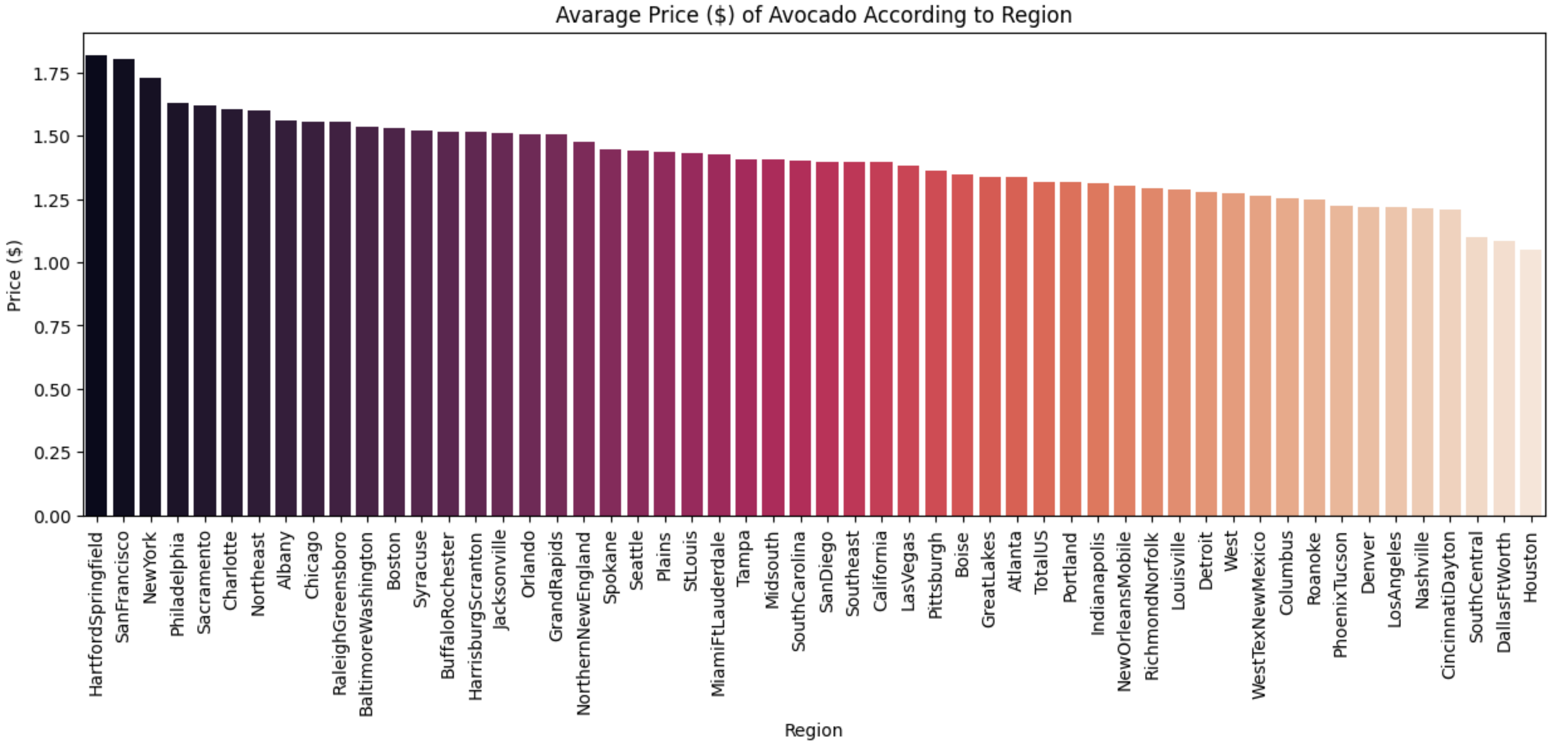
In [20]:

```python
data1 = get_avarge_between_two_columns(df,'region','AveragePrice')
plot(data1,'Region','Price ($)')
```

C:\Users\SHAIK BASHEER\AppData\Local\Temp\ipykernel_58184\430274990.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
  ax=sns.barplot(x=data.column1,y=data.column2,palette='rocket')
```



In [21]:

```
print(data1['column1'].iloc[-1], " is the region producing avocado with the lowest price.")
```

Houston is the region producing avocado with the lowest price.

6.What is the highest region of avocado production?

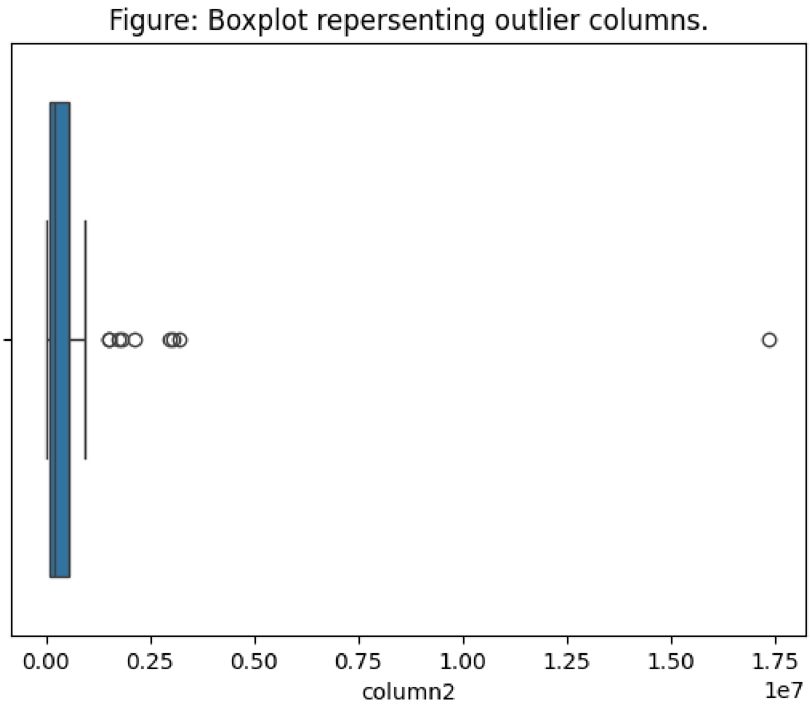
Checking if there are outlier values or not.

In [22]:

```
data2 = get_avarge_between_two_columns(df,'region','Total Volume')
sns.boxplot(x=data2.column2).set_title("Figure: Boxplot repersenting outlier columns.")
```

Out[22]:

Text(0.5, 1.0, 'Figure: Boxplot repersenting outlier columns.')



In [23]:

```
outlier_region = data2[data2.column2>10000000]
print(outlier_region['column1'].iloc[-1], "is outlier value")
```

TotalUS is outlier value

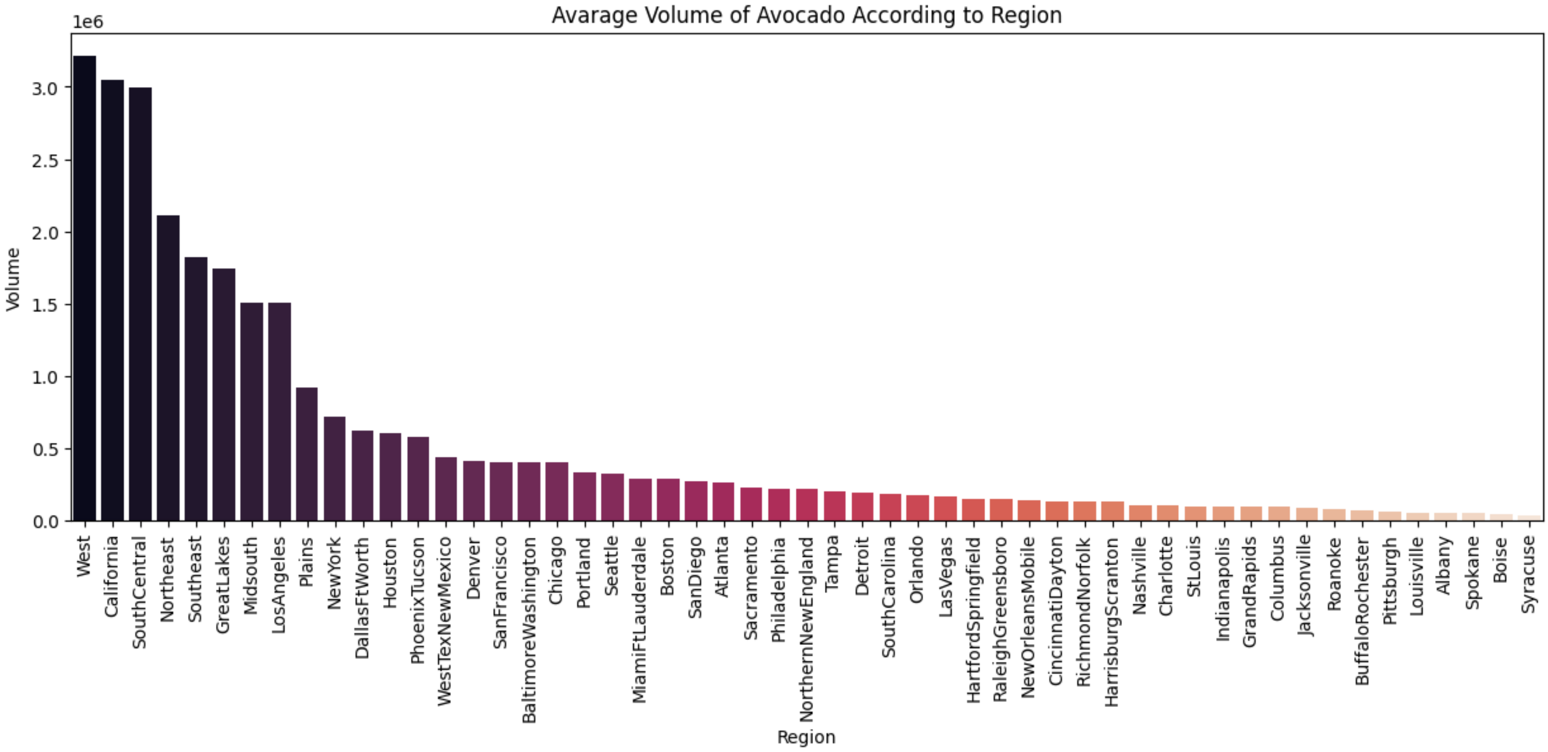
Remove the outlier values

In [24]:
outlier_region.index
data2 = data2.drop(outlier_region.index,axis=0)
In [25]:
plot(data2,'Region','Volume')

C:\Users\SHAIK BASHEER\AppData\Local\Temp\ipykernel_58184\430274990.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax=sns.barplot(x=data.column1,y=data.column2,palette='rocket')



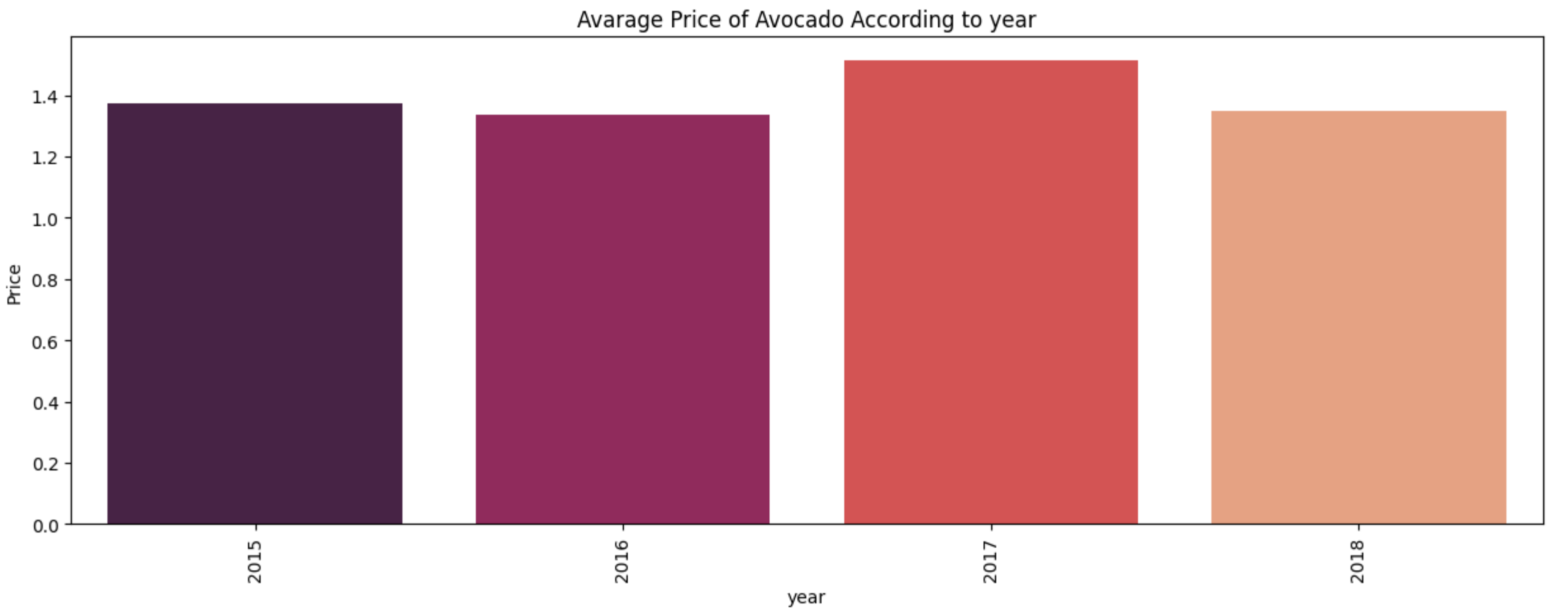
7.What is the average avocado prices in each year?

In [26]:
data3 = get_avarge_between_two_columns(df,'year','AveragePrice')
plot(data3,'year','Price')

C:\Users\SHAIK BASHEER\AppData\Local\Temp\ipykernel_58184\430274990.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax=sns.barplot(x=data.column1,y=data.column2,palette='rocket')



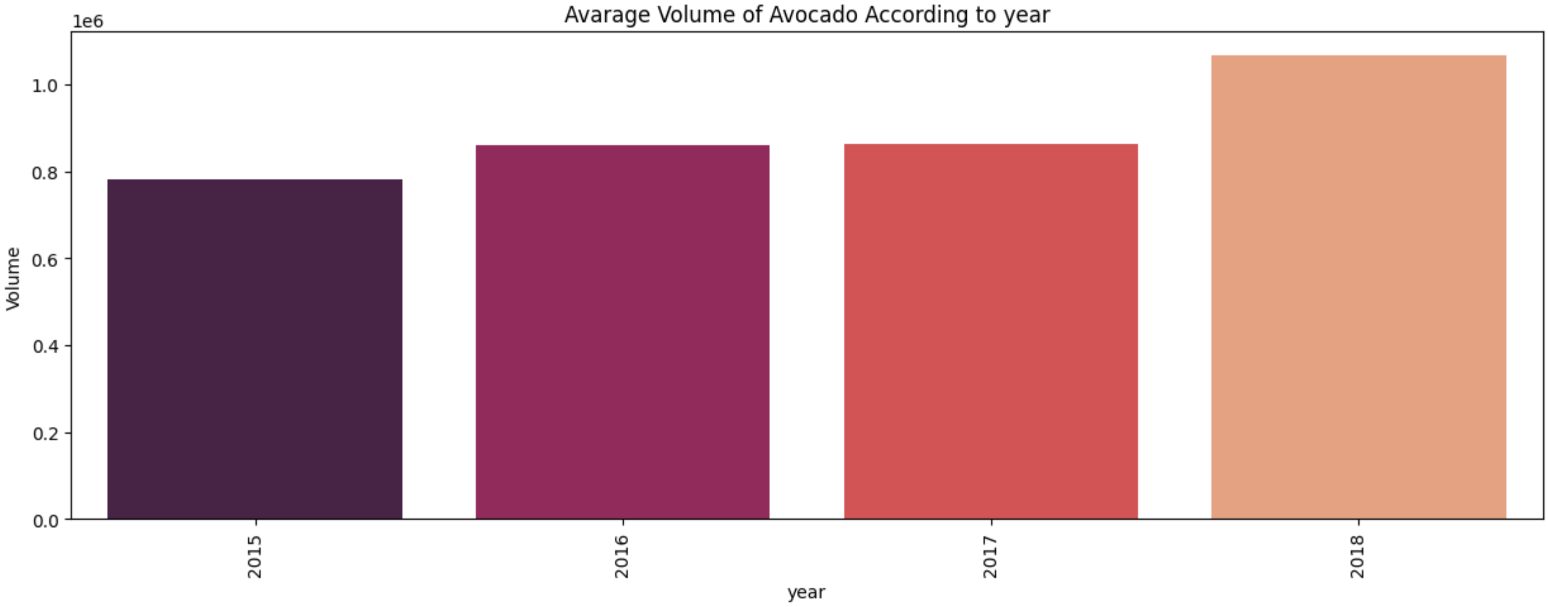
8.What is the average avocado volume in each year?

In [27]:
data4 = get_avarge_between_two_columns(df,'year','Total Volume')
plot(data4,'year','Volume')

C:\Users\SHAIK BASHEER\AppData\Local\Temp\ipykernel_58184\430274990.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

ax=sns.barplot(x=data.column1,y=data.column2,palette='rocket')



Data Modeling

We bulit the regrestion model by used Linear regresion from sklearn to predict the avocado price.

Changing some column types to categories

In [28]:

```
df['region'] = df['region'].astype('category')
df['region'] = df['region'].cat.codes

df['type'] = df['type'].astype('category')
df['type'] = df['type'].cat.codes
```

In [29]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   AveragePrice  18249 non-null  float64
 1   Total Volume  18249 non-null  float64
 2   Total Bags    18249 non-null  float64
 3   Small Bags    18249 non-null  float64
 4   Large Bags    18249 non-null  float64
 5   XLarge Bags   18249 non-null  float64
 6   type          18249 non-null  int8
 7   year          18249 non-null  int64
 8   region        18249 non-null  int8
dtypes: float64(6), int64(1), int8(2)
memory usage: 1.0 MB
```

In [30]:

```
df.head()
```

Out[30]:

	AveragePrice	Total Volume	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	1.33	64236.62	8696.87	8603.62	93.25	0.0	0	2015	0
1	1.35	54876.98	9505.56	9408.07	97.49	0.0	0	2015	0
2	0.93	118220.22	8145.35	8042.21	103.14	0.0	0	2015	0
3	1.08	78992.15	5811.16	5677.40	133.76	0.0	0	2015	0
4	1.28	51039.60	6183.95	5986.26	197.69	0.0	0	2015	0

In [31]:

```python
# split data into X and y
X = df.drop(['AveragePrice'],axis=1)
y = df['AveragePrice']

# split data into traing and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=15)
```

In [32]:

```python
print("training set:",X_train.shape,' - ',y_train.shape[0],' samples')
print("testing set:",X_test.shape,' - ',y_test.shape[0],' samples')
```

```
training set: (12774, 8)  -  12774  samples
testing set: (5475, 8)  -  5475  samples
```

In [33]:

```python
from sklearn.linear_model import LinearRegression

# Create and fit the model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[33]:

▾ LinearRegression
i ?

LinearRegression()

Evaluate the Results

In [34]:

```python
# prediction and calculate the accuracy for the testing dataset
test_pre = model.predict(X_test)
test_score = r2_score(y_test,test_pre)
print("The accuracy of testing dataset ",test_score*100)
```

```
The accuracy of testing dataset  38.58074176446672
```

In [35]:

```python
# prediction and calculate the accuracy for the testing dataset
train_pre = model.predict(X_train)
train_score = r2_score(y_train,train_pre)
print("The accuracy of training dataset ",train_score*100)
```

```
The accuracy of training dataset  39.70686042410747
```

Predicting the prices of Avacados

About the data-

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 [36]:

```python
from PIL import Image
```

In [37]:

```
#display image using python
from IPython.display import Image
url = 'https://img.etimg.com/thumb/msid-71806721,width-650,imgsize-807917,,resizemode-4,quality-100/avocados.jpg'
Image(url,height=300,width=400)
```

Out[37]:

No description has been provided for this image

In [38]:

```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
```

In [39]:

```
data=pd.read_csv(r"C:\Users\SHAIK BASHEER\Downloads\Avacodo.csv")
```

In [40]:

```
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 [41]:

```
data.head(3)
```

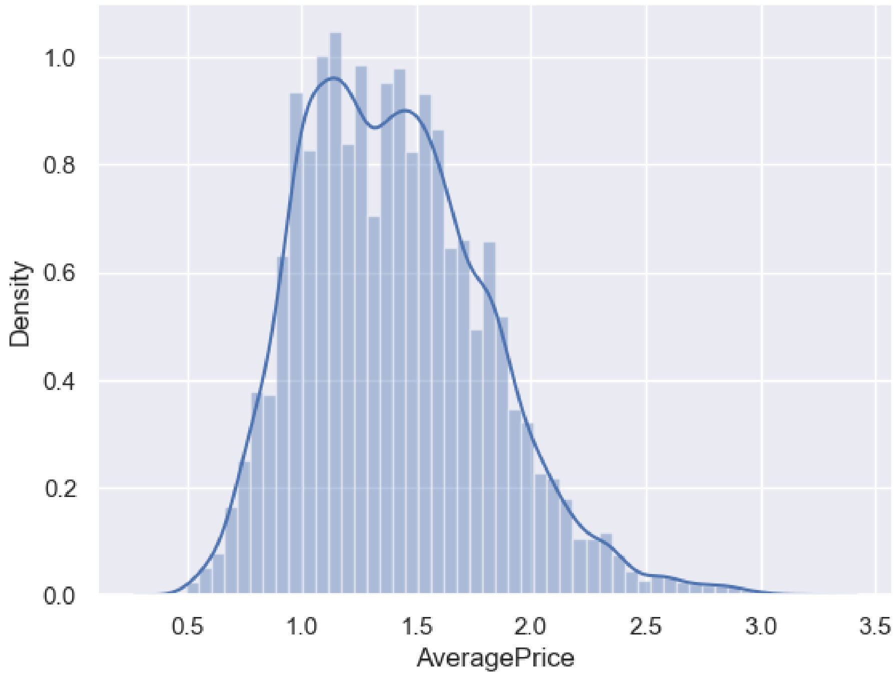
Out[41]:

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

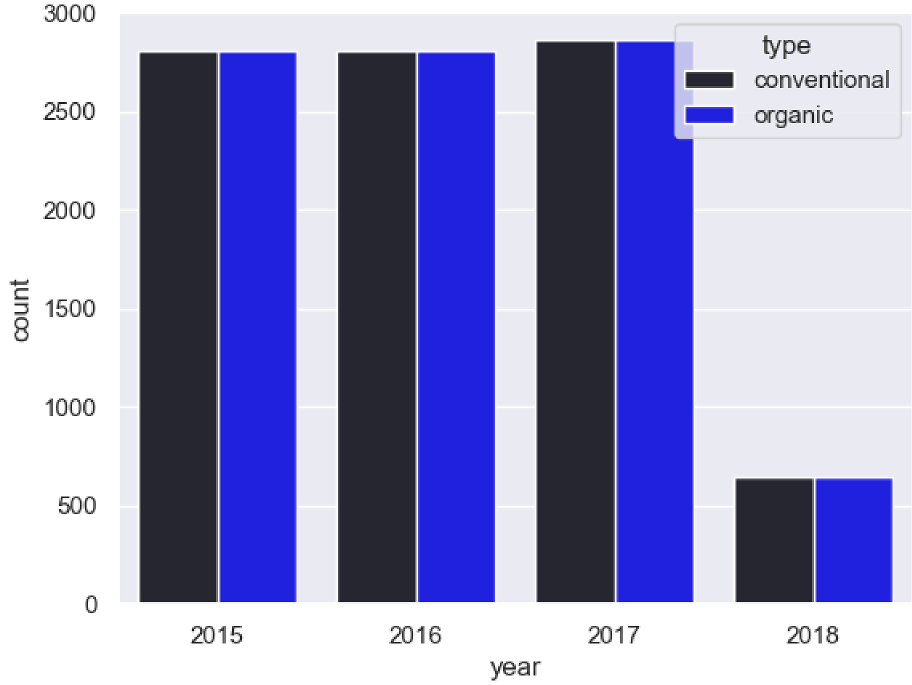
In [42]:

```
sns.distplot(data['AveragePrice'])
```

Out[42]:
<Axes: xlabel='AveragePrice', ylabel='Density'>



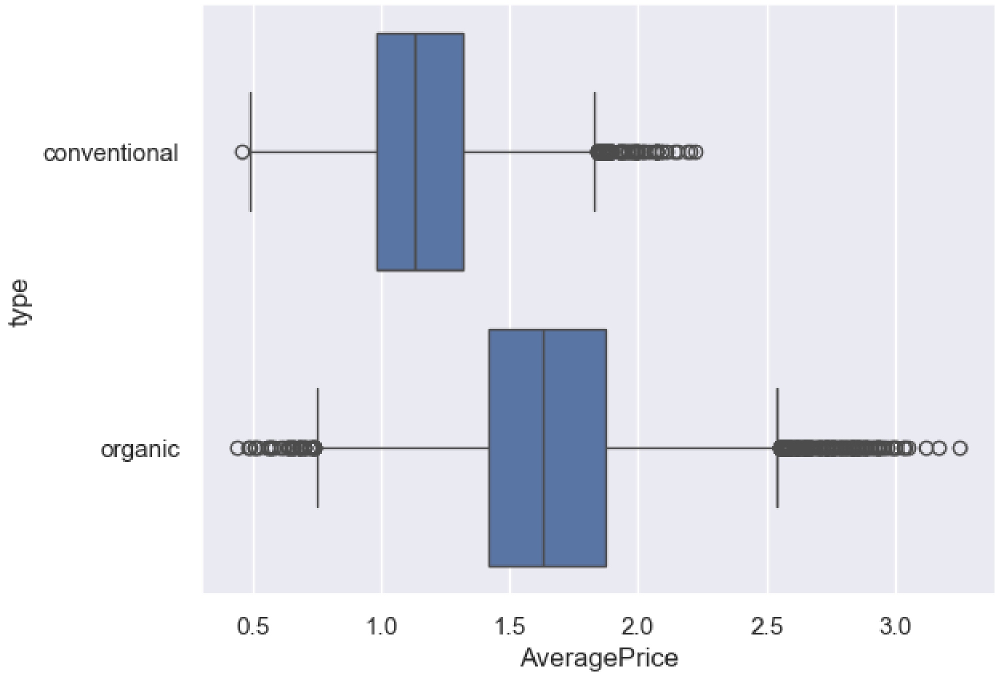
In [43]:
```
sns.countplot(x='year',data=data,hue='type',color='blue')
```
Out[43]:
<Axes: xlabel='year', ylabel='count'>



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

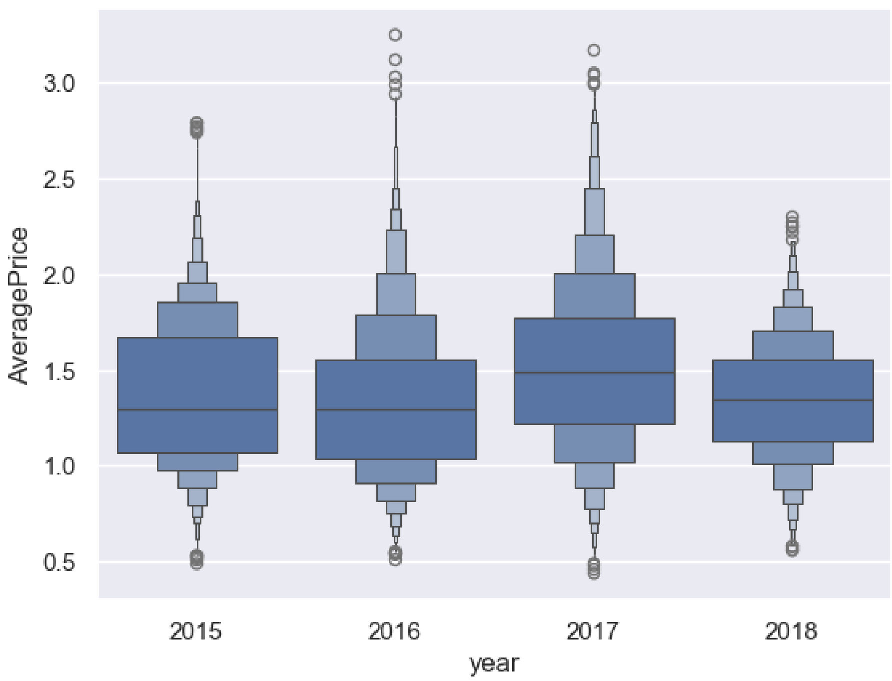
In [44]:
```
data.year.value_counts()
```
Out[44]:
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```
In [45]:
```
sns.boxplot(y='type',data=data,x='AveragePrice')
```

Out[45]:
<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 [46]:
```
data.year=data.year.apply(str)
sns.boxenplot(x="year", y="AveragePrice", data=data)
```
Out[46]:
<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 [47]:
```
data['type']=data['type'].map({'conventional':0,'organic':1})
```

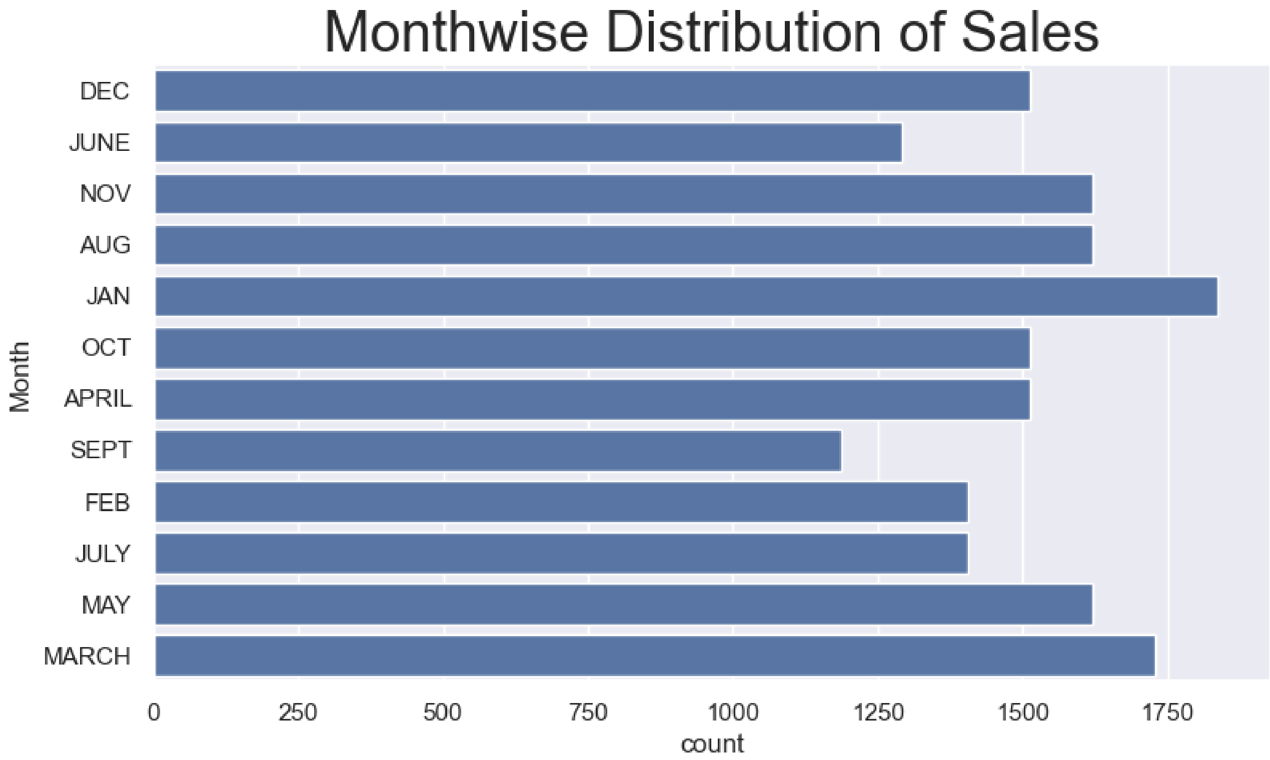
Month wise disribution

In [48]:
```
# 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',7:'JULY',8:'AUG',9:'SEPT',10:'OCT',11:'NOV',12:'DEC
```
In [49]:

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

Out[49]:

```
Text(0.5, 1.0, 'Monthwise Distribution of Sales')
```



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

Preparing data for ML models

In [50]:

```
# Creating dummy variables
```

In [51]:

```
# Creating dummy variables
dummies = pd.get_dummies(data[['year','region','Month']],drop_first=True)
```

In [52]:

```
df_dummies=pd.concat([data[['Total Volume','4046','4225','4770','Total Bags','Small Bags','Large Bags','XLarge Bags','type']],dummies],axi
target=data['AveragePrice']
```

In [53]:

```
# Splitting data into training and test set
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)
```

In [54]:

```
# Standardizing the data
cols_to_std = ['Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small 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])
```

In [55]:

```
#importing ML models from scikit-learn
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
```

In [56]:

```
from xgboost import XGBRegressor
```

In [57]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [58]:

```
#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)
                         ]
```

Deep Neural Network

In [59]:

```
# Splitting train set into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.20)
```

In [60]:

```
#importing tensorflow libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

In [61]:

```
# Splitting train set into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.20)

#importing tensorflow libraries
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
model = Sequential()
model.add(Dense(76,activation='relu',kernel_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1),
    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=0.1),
    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=0.1),
    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=0.1),
    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 [62]:

```
df=pd.read_csv(r"C:\Users\SHAIK BASHEER\Downloads\Avacodo.csv")
```

In [63]:

```
import os
print(os.getcwd())  # This will print the current working directory
```

```
c:\Users\SHAIK BASHEER\OneDrive\Desktop\projects
```

In [64]:

```
print(X_train.dtypes)
print(y_train.dtypes)
```

```
Total Volume    float64
4046            float64
4225            float64
4770            float64
Total Bags      float64
                 ...
Month_MARCH       bool
Month_MAY         bool
Month_NOV         bool
Month_OCT         bool
Month_SEPT        bool
Length: 76, dtype: object
float64
```

In [65]:

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

```
Epoch 1/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - loss: 0.6291 - val_loss: 0.1020
Epoch 2/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.1301 - val_loss: 0.1025
Epoch 3/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1122 - val_loss: 0.0710
Epoch 4/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0987 - val_loss: 0.0734
Epoch 5/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0894 - val_loss: 0.0744
Epoch 6/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0855 - val_loss: 0.0754
Epoch 7/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0802 - val_loss: 0.0586
Epoch 8/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0782 - val_loss: 0.0589
Epoch 9/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0746 - val_loss: 0.0558
Epoch 10/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0716 - val_loss: 0.0610
Epoch 11/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0710 - val_loss: 0.0563
Epoch 12/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0675 - val_loss: 0.0565
Epoch 13/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0660 - val_loss: 0.0541
Epoch 14/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0641 - val_loss: 0.0552
Epoch 15/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0612 - val_loss: 0.0567
Epoch 16/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0624 - val_loss: 0.0527
Epoch 17/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0562 - val_loss: 0.0546
Epoch 18/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0562 - val_loss: 0.0518
Epoch 19/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0577 - val_loss: 0.0528
Epoch 20/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0552 - val_loss: 0.0534
Epoch 21/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0581 - val_loss: 0.0543
Epoch 22/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0530 - val_loss: 0.0515
Epoch 23/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0529 - val_loss: 0.0545
Epoch 24/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0520 - val_loss: 0.0517
Epoch 25/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0495 - val_loss: 0.0532
Epoch 26/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0508 - val_loss: 0.0552
Epoch 27/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0510 - val_loss: 0.0528
Epoch 28/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0496 - val_loss: 0.0515
Epoch 29/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0465 - val_loss: 0.0511
Epoch 30/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0472 - val_loss: 0.0509
Epoch 31/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0453 - val_loss: 0.0513
Epoch 32/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0436 - val_loss: 0.0514
Epoch 33/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0473 - val_loss: 0.0533
Epoch 34/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0454 - val_loss: 0.0510
Epoch 35/150
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0431 - val_loss: 0.0521
```

Epoch 36/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - loss: 0.0433 - val_loss: 0.0516
Epoch 37/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0426 - val_loss: 0.0502
Epoch 38/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0429 - val_loss: 0.0532
Epoch 39/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 7ms/step - loss: 0.0413 - val_loss: 0.0522
Epoch 40/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 7ms/step - loss: 0.0396 - val_loss: 0.0509
Epoch 41/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 7ms/step - loss: 0.0426 - val_loss: 0.0515
Epoch 42/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 7ms/step - loss: 0.0389 - val_loss: 0.0510
Epoch 43/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0399 - val_loss: 0.0538
Epoch 44/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0395 - val_loss: 0.0524
Epoch 45/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - loss: 0.0373 - val_loss: 0.0513
Epoch 46/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **0s** 5ms/step - loss: 0.0359 - val_loss: 0.0522
Epoch 47/150
82/82 ━━━━━━━━━━━━━━━━━━━━ **0s** 6ms/step - loss: 0.0359 - val_loss: 0.0521

Out[65]:
<keras.src.callbacks.history.History at 0x1bab8d75eb0>

In [66]:
```
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot()
```
Out[66]:
<Axes: >

In [67]:
```
dnn_pred = model.predict(X_test)
```
172/172 ━━━━━━━━━━━━━━━━━━━━ **0s** 2ms/step

In [95]:
```
plt.figure(figsize=(12,20))
sns.set_style('whitegrid')
```
<Figure size 1200x2000 with 0 Axes>

Results Table

In [68]:
```
results.loc['Deep Neural Network']=[mean_absolute_error(y_test,dnn_pred),mean_squared_error(y_test,dnn_pred),
                           r2_score(y_test,dnn_pred)]
results
```

Out[68]:

	MAE	MSE	R2-score
Linear Regression	0.190000	0.064000	0.595000
Decision Tree	0.139000	0.044000	0.719000
Random Forest	0.107000	0.023000	0.851000
Support Vector Machines	0.159000	0.053000	0.663000
K-nearest Neighbors	0.154000	0.058000	0.628000
XGBoost	0.113000	0.024000	0.845000
Deep Neural Network	0.149162	0.046675	0.702841

In [69]:

```
import numpy as np
import pandas as pd

# Example: load your dataset (replace with your actual dataset)
data = pd.read_csv(r"C:\Users\SHAIK BASHEER\Downloads\Avacodo.csv")  # or any other data loading method

# Now, run the calculation
f"10% of mean of target variable is {np.round(0.1 * data.AveragePrice.mean(), 3)}"
```

Out[69]:

'10% of mean of target variable is 0.141'

Let's have a look at methods performing best as they have R2-score close to 1.

In [70]:

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

Out[70]:

	MAE	MSE	R2-score
Random Forest	0.107000	0.023000	0.851000
XGBoost	0.113000	0.024000	0.845000
Decision Tree	0.139000	0.044000	0.719000
Deep Neural Network	0.149162	0.046675	0.702841
Support Vector Machines	0.159000	0.053000	0.663000
K-nearest Neighbors	0.154000	0.058000	0.628000
Linear Regression	0.190000	0.064000	0.595000

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.
- Columns like Type of avocado, size and bags have impact on Average Price, lesser the MSE value accurate the model is, when we consider Small Hass in Small Bags.
- Random forest classifier model predicts the type of Avocado more accurately than Logistic regression model.
- Random Forest Regressor model predicts the average price more accurately than linear regression model.