In [1]:

```python
import pandas as pd
dataset = pd.read_csv('avocado.csv')
dataset.head()
```

Out[1]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
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	Albany
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	Albany
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	Albany
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

In [2]:

```python
dataset.tail()
```

Out[2]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	regio
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	organic	2018	WestTexNewMexic
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	organic	2018	WestTexNewMexic
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	organic	2018	WestTexNewMexic
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	organic	2018	WestTexNewMexic
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	organic	2018	WestTexNewMexic

In [3]:

```python
dataset.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 !

In [4]:

```python
dataset.head()
```

Out[4]:

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

In [5]:

```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          18249 non-null  object
 1   AveragePrice  18249 non-null  float64
 2   Total Volume  18249 non-null  float64
 3   4046          18249 non-null  float64
```

```
 4   4225          18249 non-null  float64
 5   4770          18249 non-null  float64
 6   Total Bags    18249 non-null  float64
 7   Small Bags    18249 non-null  float64
 8   Large Bags    18249 non-null  float64
 9   XLarge Bags   18249 non-null  float64
 10  type          18249 non-null  object
 11  year          18249 non-null  int64
 12  region        18249 non-null  object
dtypes: float64(9), int64(1), object(3)
memory usage: 1.8+ 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 so we can be able to use the day and the month columns in building our machine learning model later.

In [6]:

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

Out[6]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region	Month	Day
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany	12	27
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany	12	20
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany	12	13
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany	12	6
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany	11	29

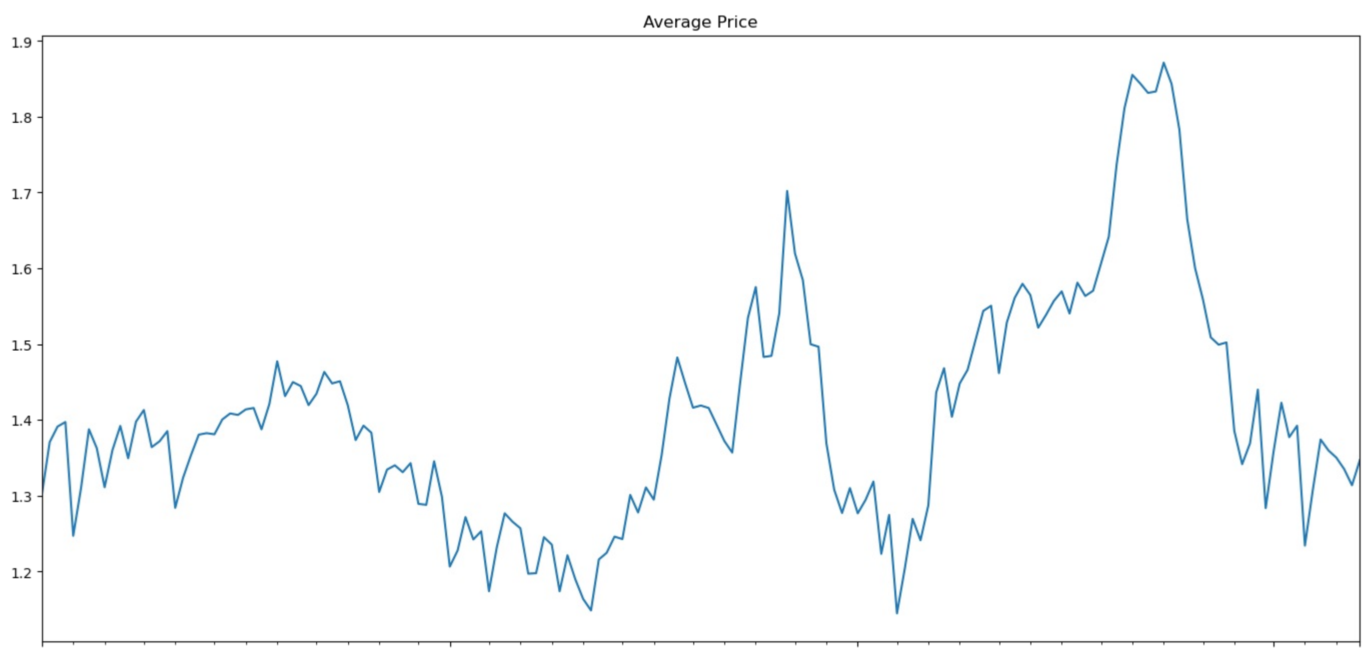
Here we add two more columns month and day where 6 in Month implies the month "June" & 15 in Day implies the 15th day of a month.

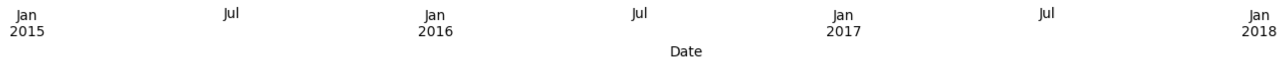
Step 2: Analysis of Average Prices

In [7]:

```
import matplotlib.pyplot as plt

byDate=dataset.groupby('Date').mean()
plt.figure(figsize=(17,8),dpi=100)
byDate['AveragePrice'].plot()
plt.title('Average Price')
```

Out[7]: `Text(0.5, 1.0, 'Average Price')`

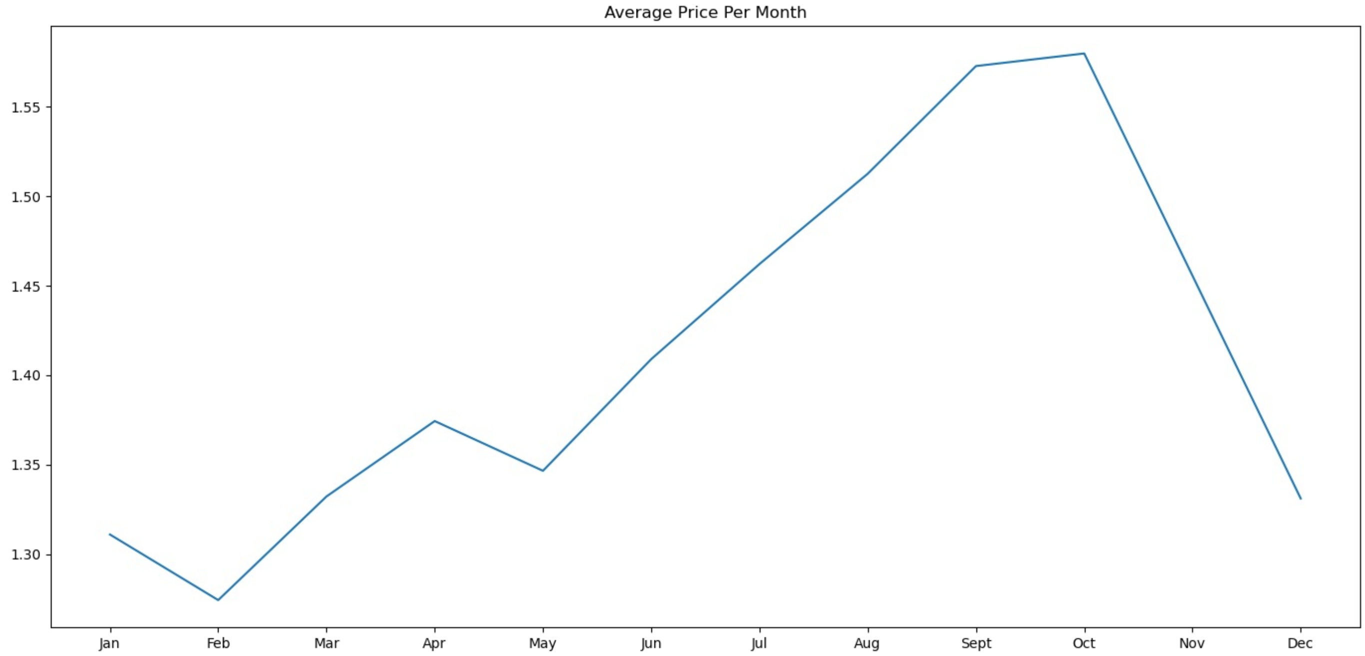




Hence the plot shows the average price of avocado at various points of time

In [8]:
```
byMonth = dataset.groupby("Month").mean()
plt.figure(figsize=(17,8),dpi=100)
plt.plot(["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sept","Oct","Nov","Dec"],byMonth['AveragePrice'])
plt.title('Average Price Per Month')
```

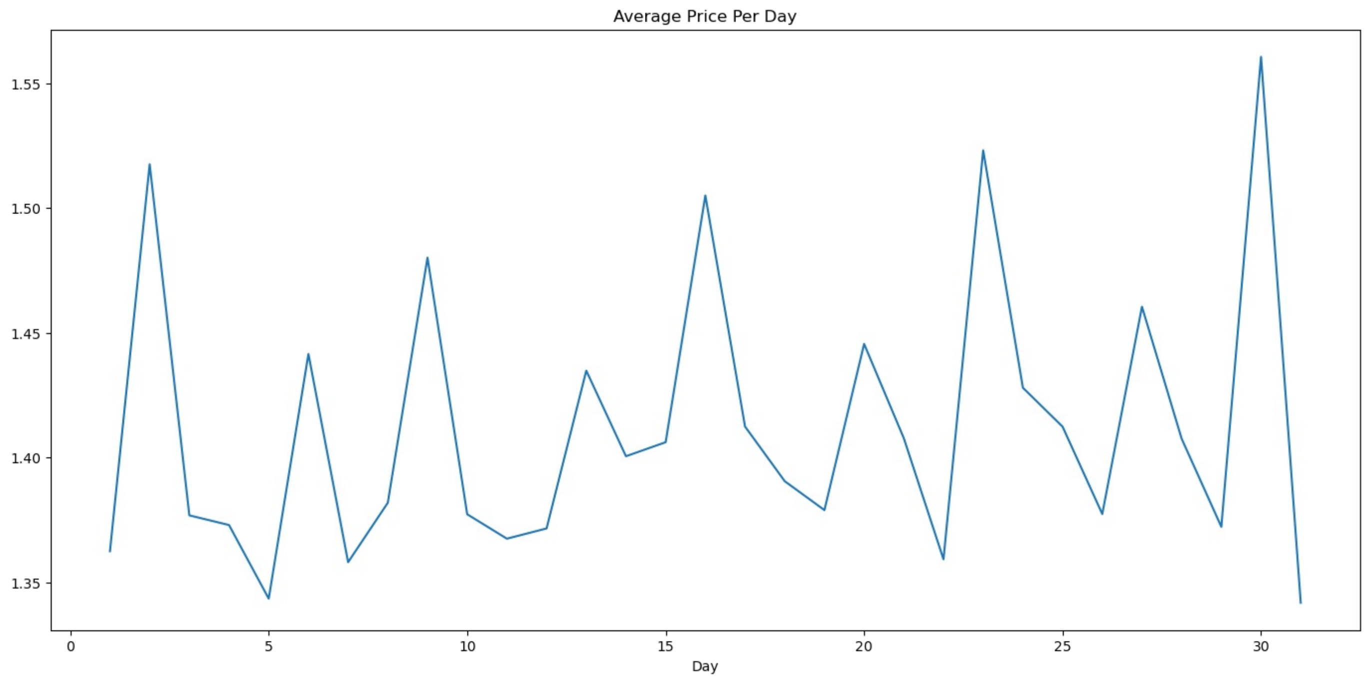
Out[8]: `Text(0.5, 1.0, 'Average Price Per Month')`



From the above graph plotted for average price of avocado per month we can observe that the price rises for a while in February to March then it falls in April and then the month of May witnesses a rise in the average price. This rise reaches its zenith in the month of October and henceforth it starts to fall.

In [9]:
```
byDay = dataset.groupby("Day").mean()
plt.figure(figsize=(17,8),dpi=100)
byDay['AveragePrice'].plot()
plt.title('Average Price Per Day')
```

Out[9]: `Text(0.5, 1.0, 'Average Price Per Day')`



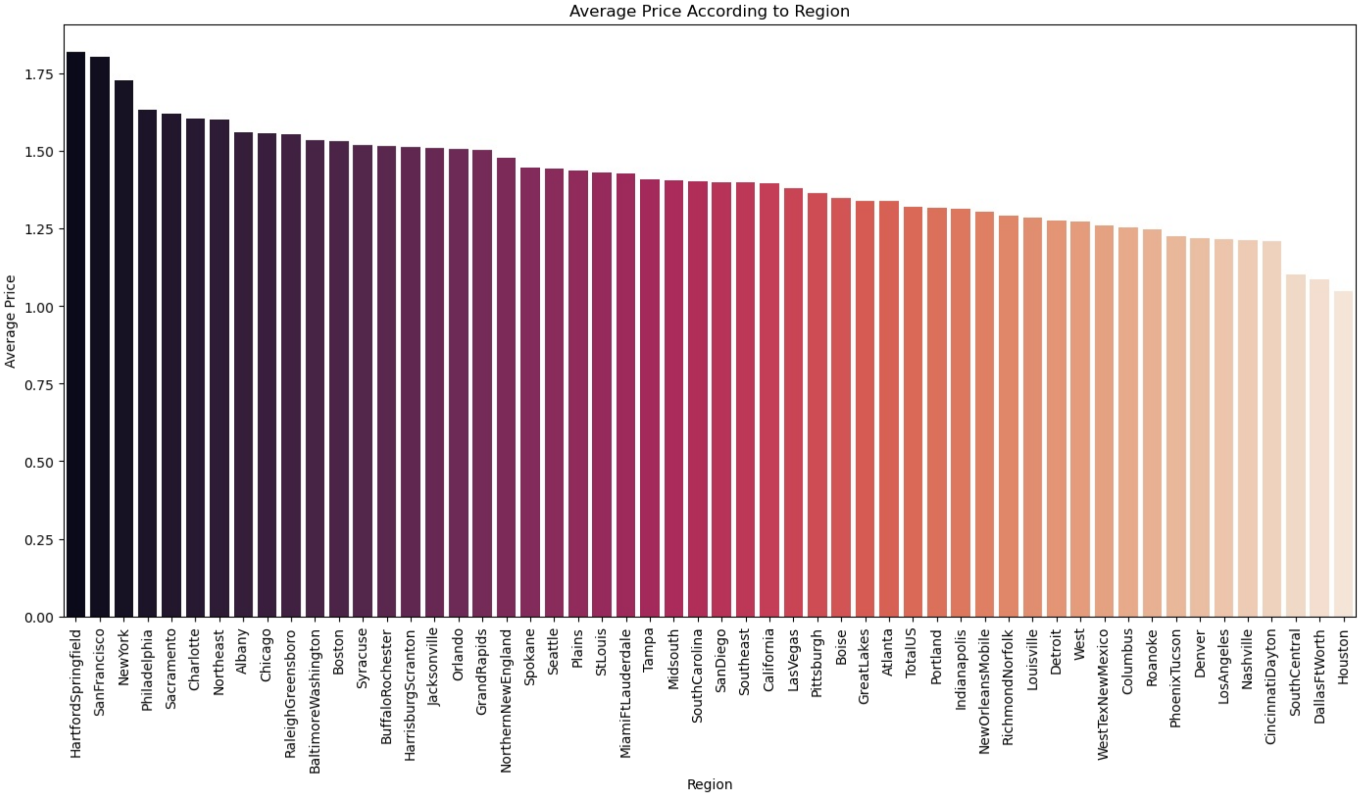
The above graph for average price per day implies that the price fluctuates in a similar manner at a regular interval.

In [10]:

```
import seaborn as sns

byRegion=dataset.groupby('region').mean()
byRegion.sort_values(by=['AveragePrice'], ascending=False, inplace=True)
plt.figure(figsize=(17,8),dpi=100)
sns.barplot(x = byRegion.index,y=byRegion["AveragePrice"],data = byRegion,palette='rocket')
plt.xticks(rotation=90)
plt.xlabel('Region')
plt.ylabel('Average Price')
plt.title('Average Price According to Region')
```

Out[10]:

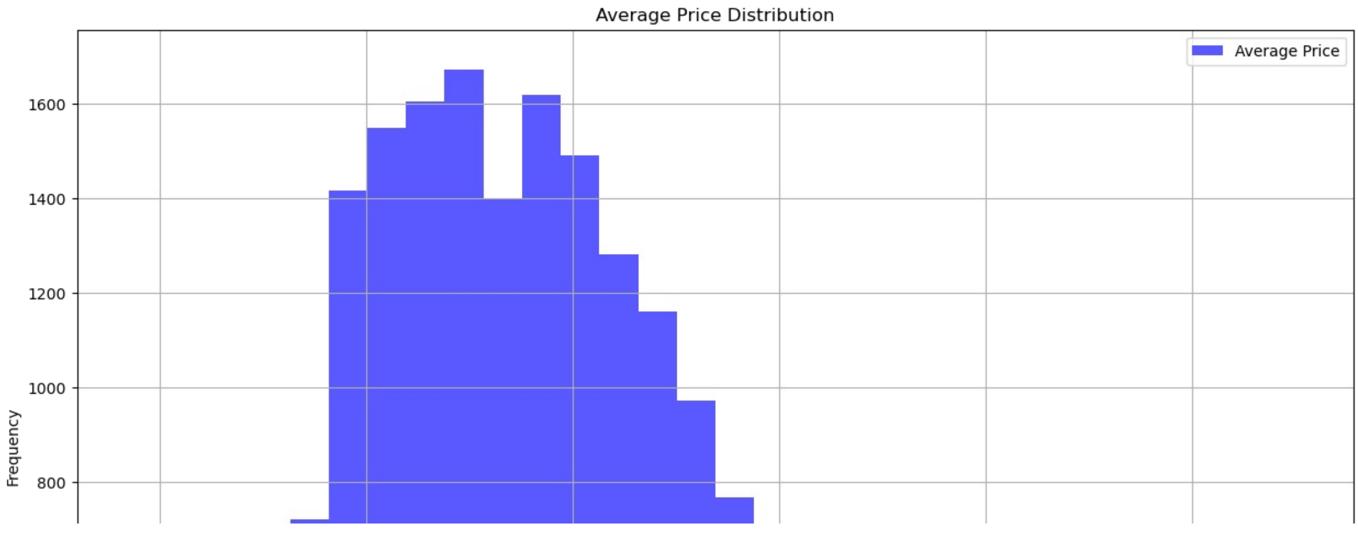
```
Text(0.5, 1.0, 'Average Price According to Region')
```

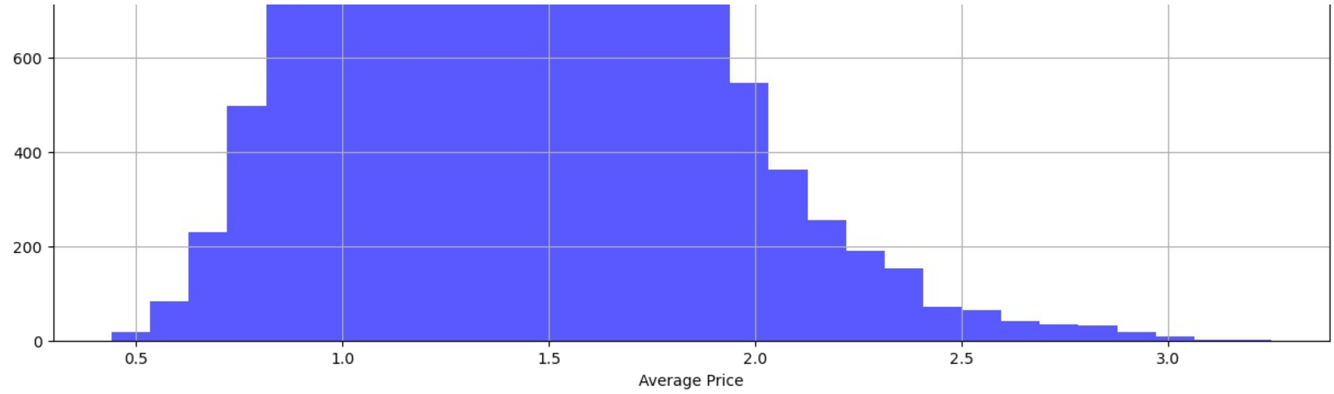


The barplot shows the average price of avocado at various regions in a ascending order. Clearly Hartford Springfield, SanFrancisco, NewYork are the regions with the highest avocado prices.

In [11]:

```
plt.figure(figsize=(15,10),dpi=100)
dataset["AveragePrice"].plot(kind="hist",color="blue",bins=30,grid=True,alpha=0.65,label="Average Price")
plt.legend()
plt.xlabel("Average Price")
plt.title("Average Price Distribution")
plt.show()
```





The above histogram for the average price of avocado suggests that its distribution is somewhat positively skewed.

In [12]:

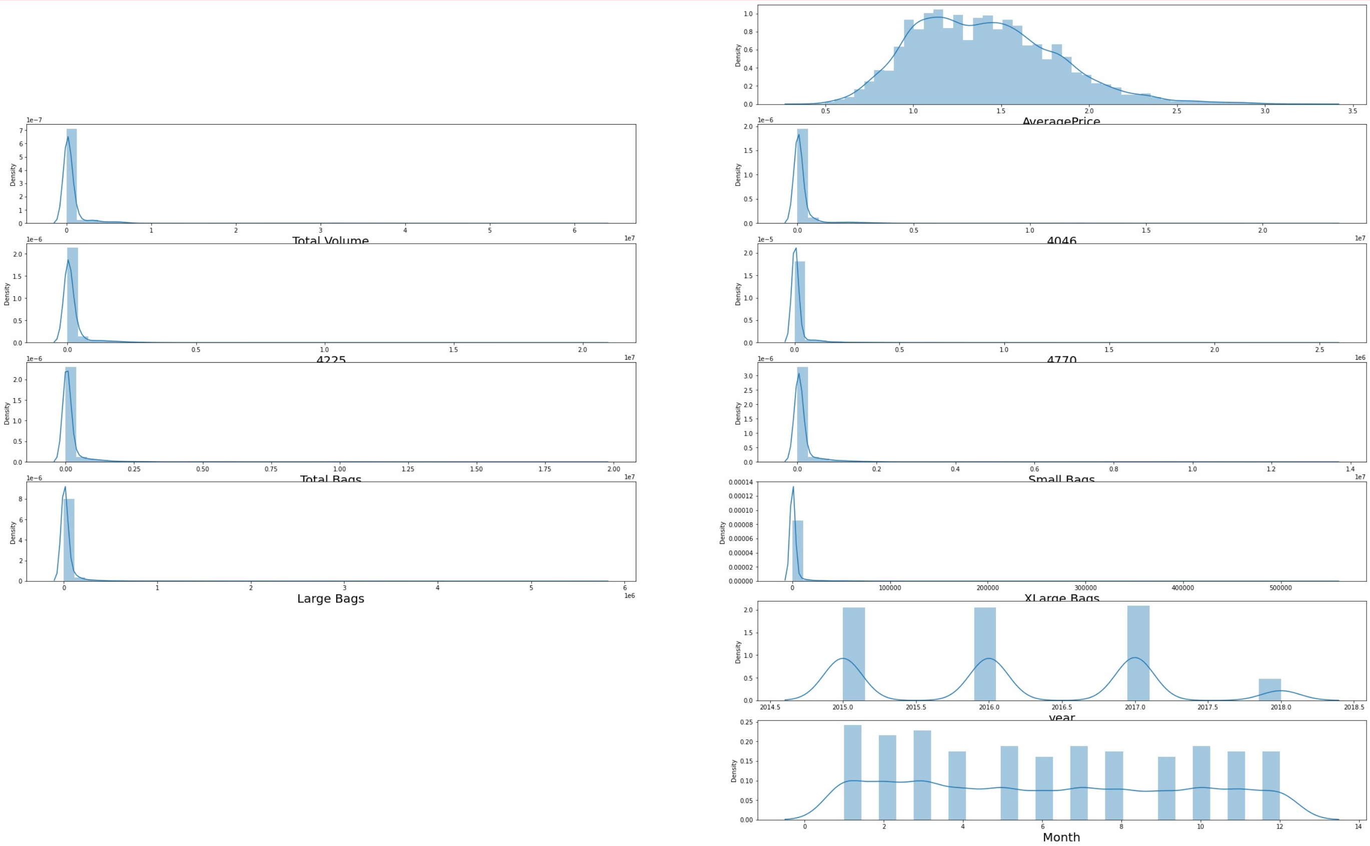
```
import pandas as pd
import numpy as np
```

In [13]:

```
#plotting distribution plot for my dataframe to check and remove outliers
plt.figure(figsize=(40,25),facecolor="white")
plotnumber = 1

for column in dataset:

    if(dataset[column].dtype == np.float64 or dataset[column].dtype == np.int64):
        if plotnumber<=14:
            ax = plt.subplot(7,2,plotnumber)
            sns.distplot(dataset[column])
            plt.xlabel(column,fontsize=20)

    plotnumber +=1
plt.show()
# importing module
import warnings

warnings.warn('Do not show this message')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecat
ed function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-lev
el function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

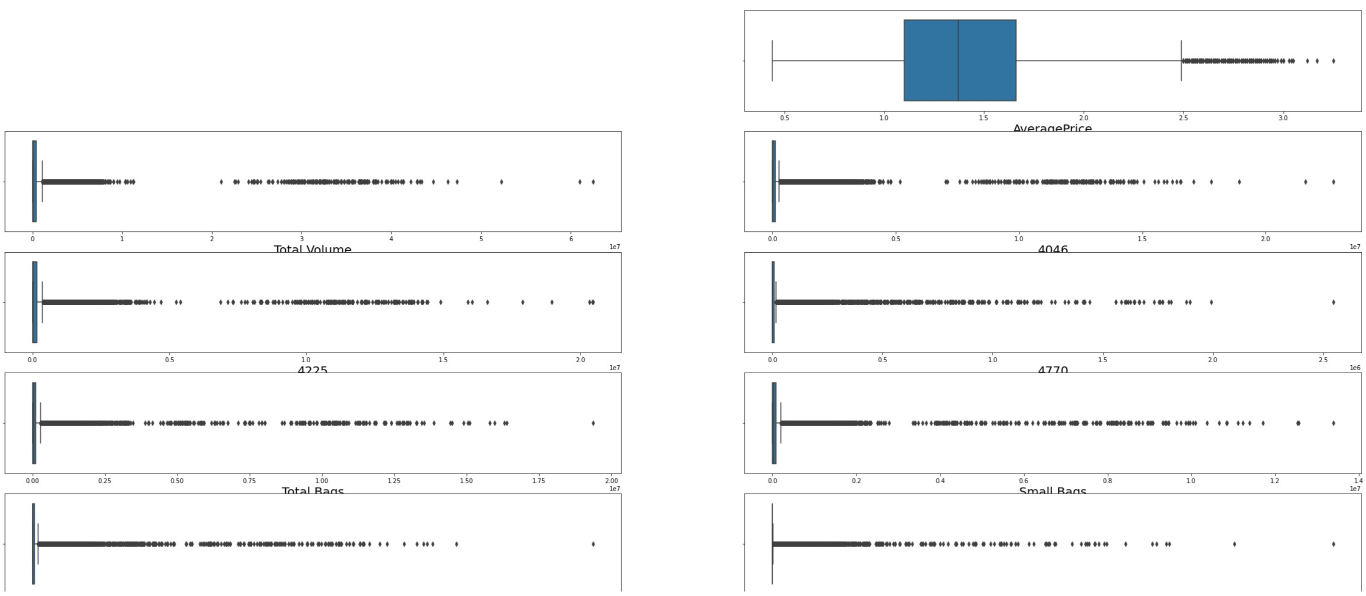
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

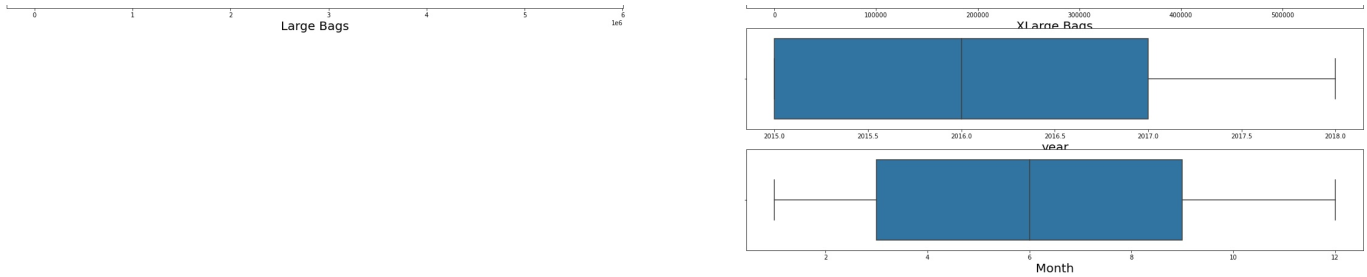



```
<ipython-input-13-31681bf7c4e7>:18: UserWarning: Do not show this message
  warnings.warn('Do not show this message')
```

In [14]:

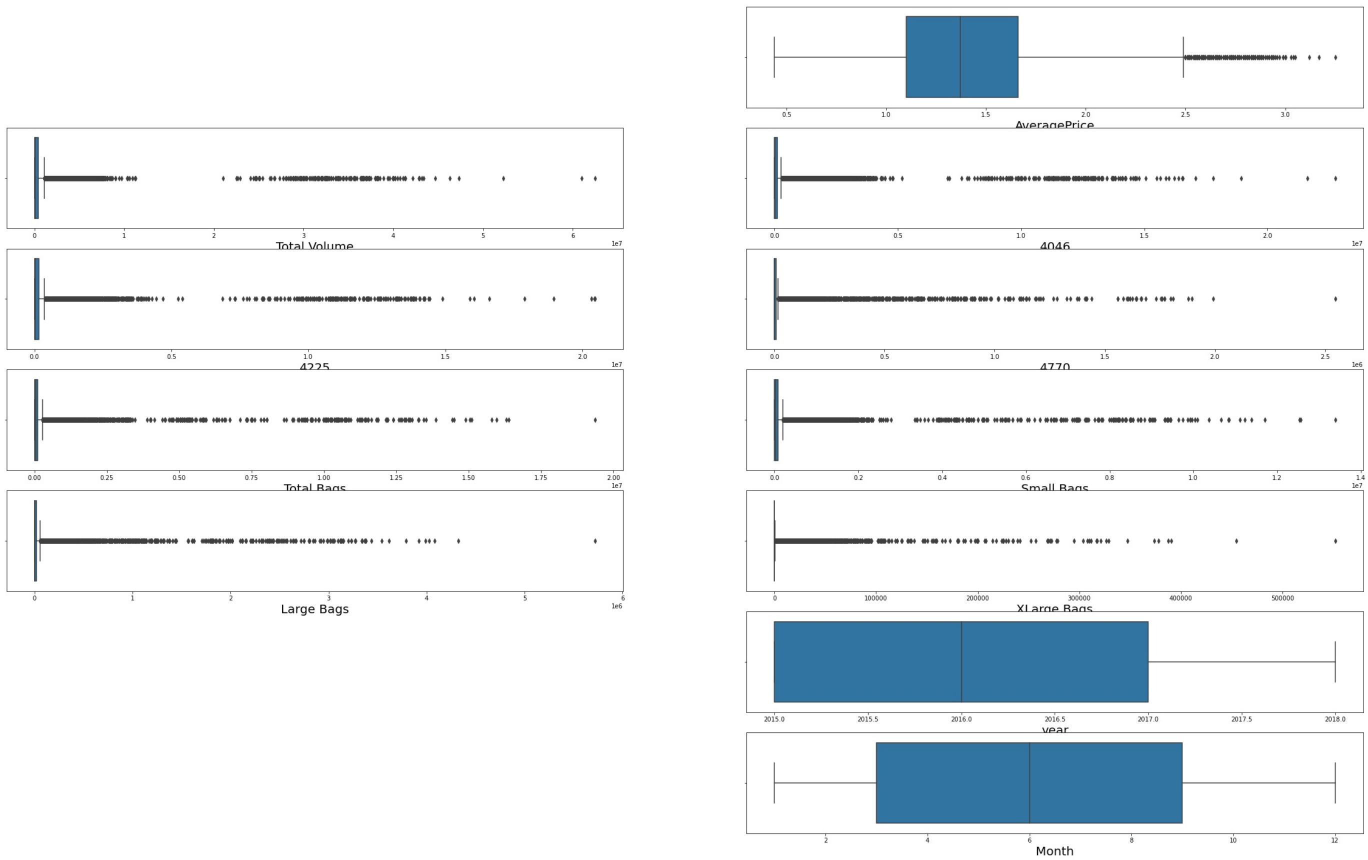
```python
import warnings
warnings.filterwarnings("ignore")
#plotting box plot for my dataframe to check and remove outliers
plt.figure(figsize=(40,25),facecolor="white")
plotnumber = 1

for column in dataset:

    if(dataset[column].dtype == np.float64 or dataset[column].dtype == np.int64):
        if plotnumber<=14:
            ax = plt.subplot(7,2,plotnumber)
            sns.boxplot(dataset[column])
            plt.xlabel(column,fontsize=20)

    plotnumber +=1
plt.show()
```





In [15]:

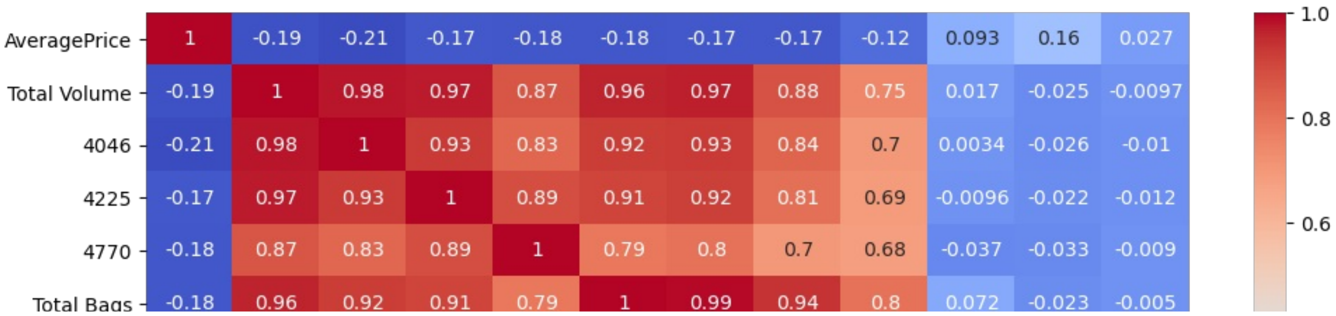
```
import warnings
warnings.filterwarnings("ignore")
#plotting box plot for my dataframe to check and remove outliers
plt.figure(figsize=(40,25),facecolor="white")
plotnumber = 1

for column in dataset:

    if(dataset[column].dtype == np.float64 or dataset[column].dtype == np.int64):
        if plotnumber<=14:
            ax = plt.subplot(7,2,plotnumber)
            sns.boxplot(dataset[column])
            plt.xlabel(column,fontsize=20)

    plotnumber +=1
plt.show()
```

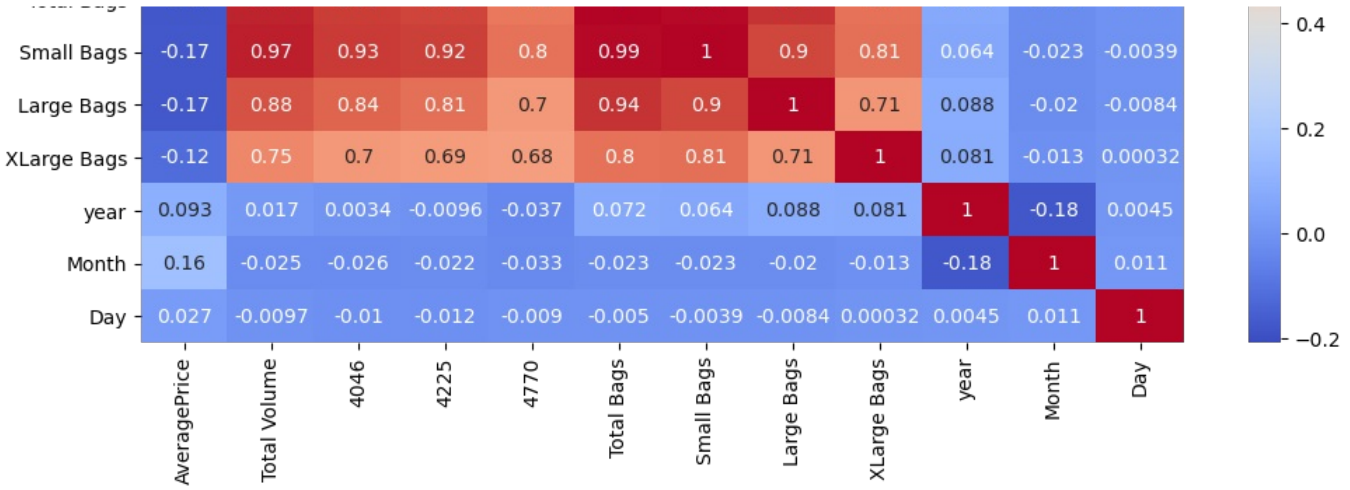


In [16]:

```
corr_df = dataset.corr(method='pearson')
plt.figure(figsize=(12,6),dpi=100)
sns.heatmap(corr_df,cmap='coolwarm',annot=True)
```

Out[16]: <AxesSubplot:>

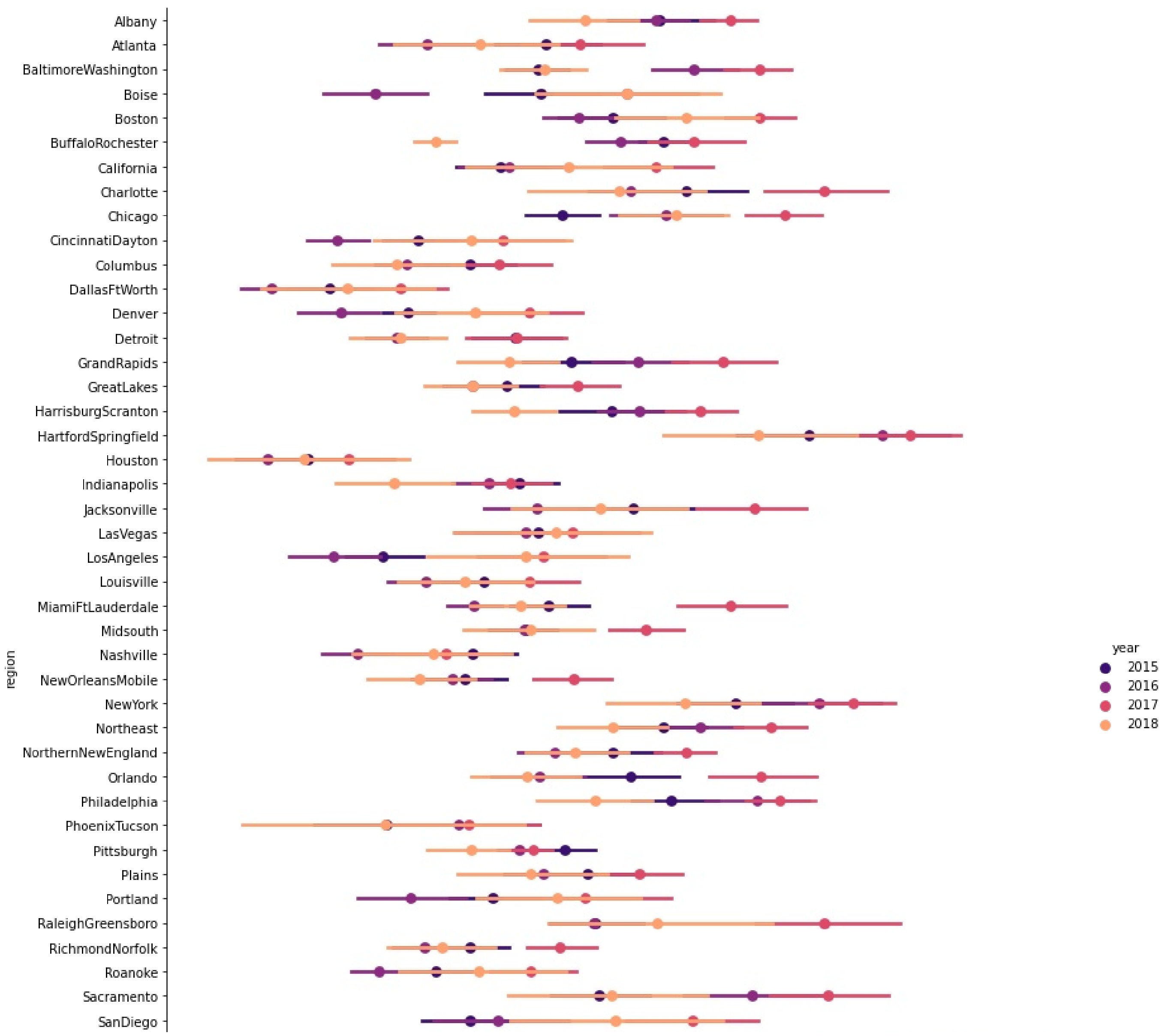


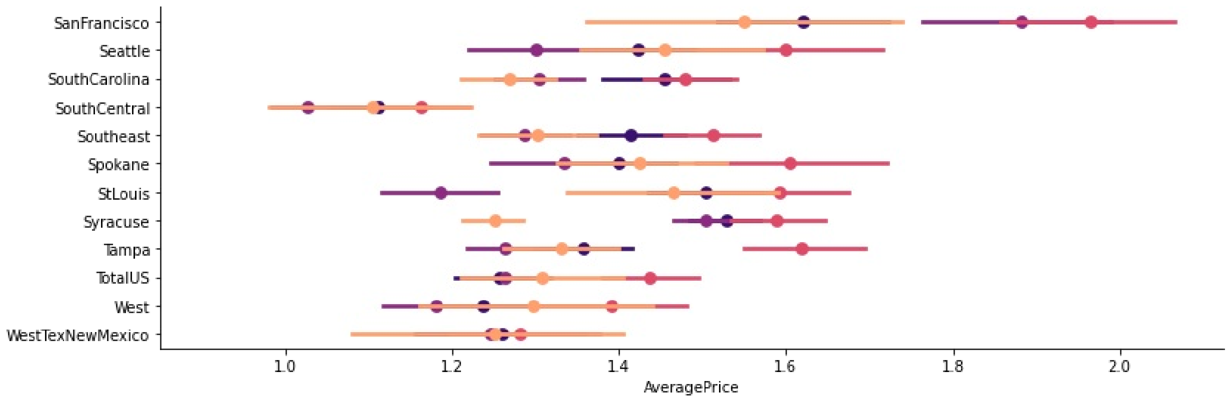


As we can from the heatmap above, all the Features are not correlated with the Average Price column, instead most of them are correlated with each other.

In [17]:
```
sns.factorplot('AveragePrice','region',data=dataset,
                   hue='year',
                   aspect=0.8,
                   height=15,
                   palette='magma',
                   join=False,
              )
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x245d8a33a90>





A factor plot is simply the same plot generated for different response and factor variables and arranged on a single page. The underlying plot generated can be any univariate or bivariate plot. The scatter plot is the most common application. The above plot is a factor plot of average avocado price for different regions classified by year.

In [18]:

```
dataset_vif = dataset.copy()
dataset_vif.drop(columns=['Date','type','region'],inplace = True)

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

Xf = add_constant(dataset_vif)
pd.Series([variance_inflation_factor(Xf.values, i)
               for i in range(Xf.shape[1])],
              index=Xf.columns)
```

Out[18]:

```
const           5.068485e+06
AveragePrice    1.099766e+00
Total Volume    4.918067e+09
4046            6.598339e+08
4225            5.978631e+08
4770            4.762133e+06
Total Bags      2.370316e+14
Small Bags      1.364727e+14
Large Bags      1.448103e+13
XLarge Bags     7.622174e+10
year            1.101665e+00
Month           1.071816e+00
Day             1.001467e+00
dtype: float64
```

The above code snippet calculates the variable inflation factor for the displayed variables.

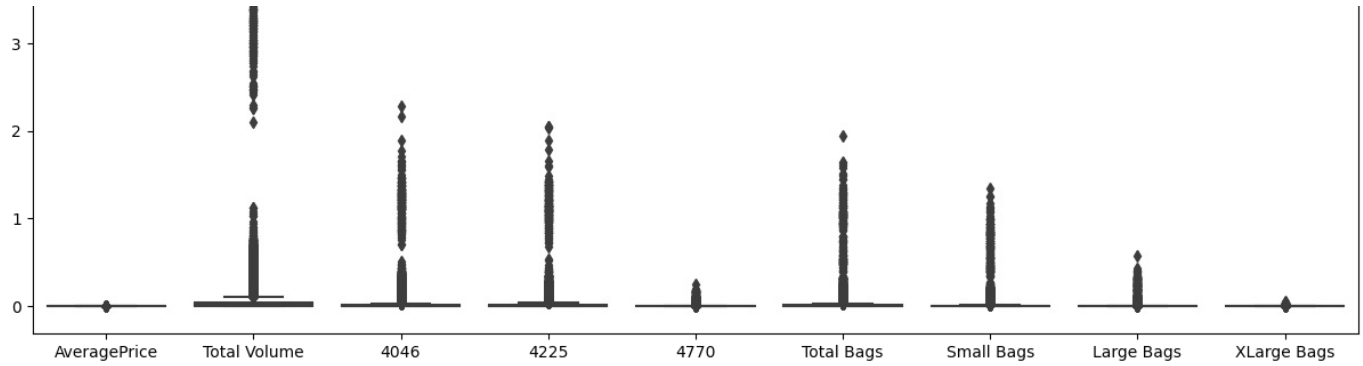
Step 3: Taking Care of the Outliers

In [19]:

```
plt.figure(figsize=(15,7),dpi=100)
sns.boxplot(data = dataset[[
 'AveragePrice',
 'Total Volume',
 '4046',
 '4225',
 '4770',
 'Total Bags',
 'Small Bags',
 'Large Bags',
 'XLarge Bags']])
```

Out[19]:

```
<AxesSubplot:>
```



Clearly the boxplot indicates that all the variables contains outliers. Now we need to take care of the outliers.

In [20]:

```
dataset.drop(columns=["Date"],inplace=True)
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   AveragePrice  18249 non-null  float64
 1   Total Volume  18249 non-null  float64
 2   4046          18249 non-null  float64
 3   4225          18249 non-null  float64
 4   4770          18249 non-null  float64
 5   Total Bags    18249 non-null  float64
 6   Small Bags    18249 non-null  float64
 7   Large Bags    18249 non-null  float64
 8   XLarge Bags   18249 non-null  float64
 9   type          18249 non-null  object 
 10  year          18249 non-null  int64  
 11  region        18249 non-null  object 
 12  Month         18249 non-null  int64  
 13  Day           18249 non-null  int64  
dtypes: float64(9), int64(3), object(2)
memory usage: 1.9+ MB
```

Before we go on to taking care of the outliers we removed the "Date" variable from our dataset as it is useless now.

In [21]:

```
import numpy as np
from numpy import percentile

columns = dataset.columns
for j in columns:
    if isinstance(dataset[j][0], str) :
        continue
    else:
        for i in range(len(dataset)):
            #defining quartiles
            quartiles = percentile(dataset[j], [25,75])
            # calculate min/max
            lower_fence = quartiles[0] - (1.5*(quartiles[1]-quartiles[0]))
            upper_fence = quartiles[1] + (1.5*(quartiles[1]-quartiles[0]))
            if dataset[j][i] > upper_fence:
                dataset[j][i] = upper_fence
            elif dataset[j][i] < lower_fence:
                dataset[j][i] = lower_fence
```

In the following code snippet we have we replaced the outliers higer than the upper whisker by the value of the upper whisker and the outliers lower than the lower whisker by the value of the lower whisker.

In [22]:

```
dataset.head()
```

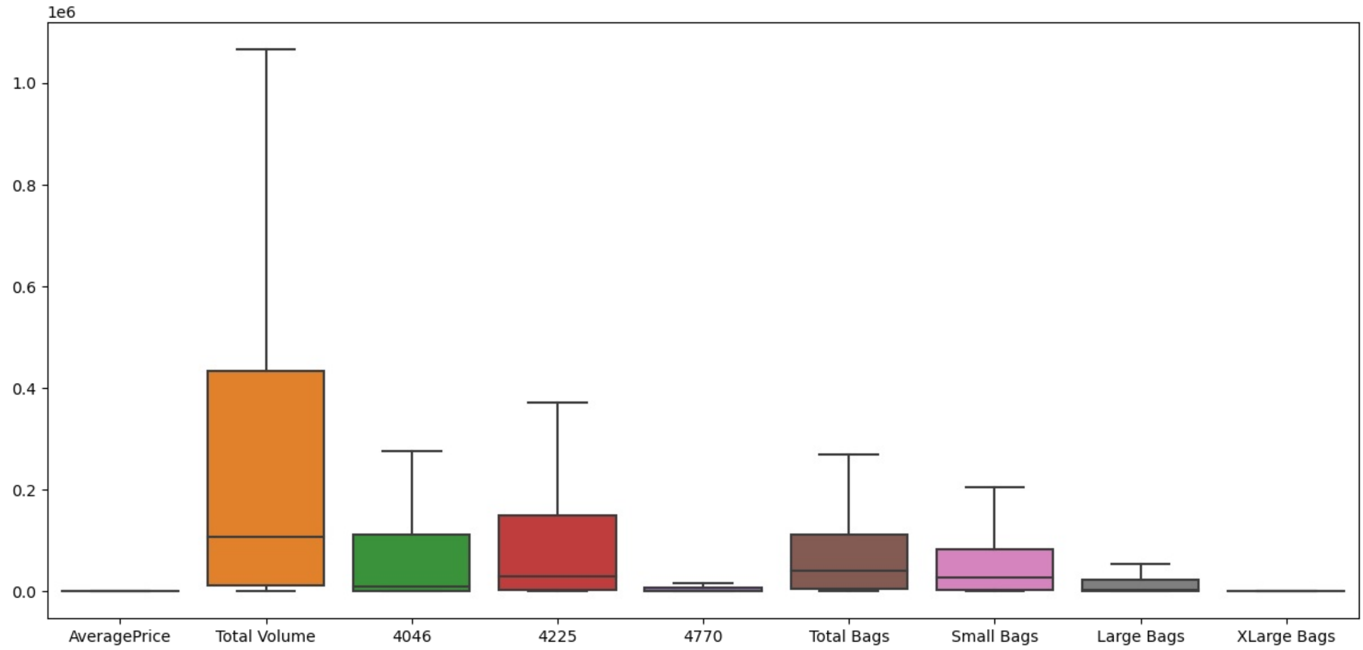
Out[22]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region	Month	Day
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany	12	27

1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany	12	20
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany	12	13
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany	12	6
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany	11	29

In [23]:
```
plt.figure(figsize=(15,7),dpi=100)
sns.boxplot(data = dataset[[
 'AveragePrice',
 'Total Volume',
 '4046',
 '4225',
 '4770',
 'Total Bags',
 'Small Bags',
 'Large Bags',
 'XLarge Bags']])
```

Out[23]: <AxesSubplot:>



Now clearly our data is free from outliers. Now we can fit our data to appropriate models.

Step 4: Taking Care of the Categorical Variables

Now since our data contains categorical variables like "type", "month" and "region" we apply one-hot encoding to our variables "region","month" and apply label encoding in variable "type".

One hot encoding creates equal number of columns, with 1's and 0's, as the number of categories in a categorical variable a column for a specific category contains 1's where the category is present and 0's elsewhere.

As for label encoding it asssigns numerical value to the categories of a categorical variable in their alphabetical order, the indexing starts with 0.

OneHotEncoder in Python can encode a specific number of categories since for the variable 'region' we have crossed that threshold we have used pandas.get_dummies instead. Had we use OneHotEncoder we would have eliminated one column to avoid dummy variable trap but here we have no use for that.

In [24]:
```
dataset['region'] = pd.Categorical(dataset['region'])
dfDummies_region = pd.get_dummies(dataset['region'], prefix = 'region')
dfDummies_region
```

Out[24]:

	region_Albany	region_Atlanta	region_BaltimoreWashington	region_Boise	region_Boston	region_BuffaloRochester	region_California	reg
0	1	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	

2	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...
18244	0	0	0	0	0	0	0
18245	0	0	0	0	0	0	0
18246	0	0	0	0	0	0	0
18247	0	0	0	0	0	0	0
18248	0	0	0	0	0	0	0

18249 rows × 54 columns

In [25]:

```
dataset = pd.concat([dataset, dfDummies_region], axis=1)
dataset.drop(columns="region",inplace=True)
dataset
```

Out[25]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	...	region_SouthCarolina	region_S
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	...	0	
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	...	0	
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	...	0	
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	...	0	
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
18244	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	organic	...	0	
18245	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	organic	...	0	
18246	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	organic	...	0	
18247	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	organic	...	0	
18248	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	organic	...	0	

18249 rows × 67 columns

Adding the one hot encoded columns for region into our data and dropping the region column from our dataset

In [26]:

```
dataset['Month'] = pd.Categorical(dataset['Month'])
dfDummies_month = pd.get_dummies(dataset['Month'], prefix = 'month')
dfDummies_month
```

Out[26]:

	month_1	month_2	month_3	month_4	month_5	month_6	month_7	month_8	month_9	month_10	month_11	month_12
0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
18244	0	1	0	0	0	0	0	0	0	0	0	0
18245	1	0	0	0	0	0	0	0	0	0	0	0
18246	1	0	0	0	0	0	0	0	0	0	0	0
18247	1	0	0	0	0	0	0	0	0	0	0	0
18248	1	0	0	0	0	0	0	0	0	0	0	0

18249 rows × 12 columns

Similarly applying one hot encoding on months.

In [27]:

```
dataset = pd.concat([dataset, dfDummies_month], axis=1)
dataset.drop(columns="Month",inplace=True)
```

```
dataset
```

Out[27]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	...	month_3	month_4	month_5
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	...	0	0	0
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	...	0	0	0
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	...	0	0	0
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	...	0	0	0
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18244	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	organic	...	0	0	0
18245	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	organic	...	0	0	0
18246	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	organic	...	0	0	0
18247	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	organic	...	0	0	0
18248	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	organic	...	0	0	0

18249 rows × 78 columns

Adding the one hot encoded columns for Month into our data and dropping the Month column from our dataset.

In [28]:

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
dataset['type']= label_encoder.fit_transform(dataset['type'])
dataset
```

Out[28]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	...	month_3	month_4	month_5	month_
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	0	...	0	0	0	
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	0	...	0	0	0	
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	0	...	0	0	0	
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	0	...	0	0	0	
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
18244	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	1	...	0	0	0	
18245	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	1	...	0	0	0	
18246	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	1	...	0	0	0	
18247	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	1	...	0	0	0	
18248	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	1	...	0	0	0	

18249 rows × 78 columns

Now label encoding on the variable "type"

Hence our preprocessing ends here!!!

Now its time that we fit multiple linear regression, decision tree regression and random forest regression onto our data.

Step 5: Model Fitting

In [29]:

```
dataset.head()
```

Out[29]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	...	month_3	month_4	month_5	month_6	mo
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	0	...	0	0	0	0	
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	0	...	0	0	0	0	
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	0	...	0	0	0	0	
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	0	...	0	0	0	0	
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	0	...	0	0	0	0	

5 rows × 78 columns

Having a look at our data after complete preprocessing.

Splitting our dataset to training and test numpy arrays with the names having their intended meaning. Where we are using 75% of our dataset for training and 25% of the data for testing

In [30]:
```
X=dataset.iloc[:,1:78]
y=dataset['AveragePrice']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=50)
y_test = np.array(y_test,dtype = float)
```

Normalizing our X_train and X_test using standard scaler.

In [31]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

The funtion regression_results defined below calculates and prints the following features of a model: explained_variance, r2, adjusted_r2, MAE, MSE, RMSE. It accepts the original and predicted values as its arguments.

In [32]:
```
import sklearn.metrics as metrics

def regression_results(y_true, y_pred):
    explained_variance=metrics.explained_variance_score(y_true, y_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_true, y_pred)
    mse=metrics.mean_squared_error(y_true, y_pred)
    r2=metrics.r2_score(y_true, y_pred)
    adjusted_r2 = 1 - (1-r2)*(len(y_true)-1)/(len(y_true)-X_test.shape[1]-1)

    print('Explained_variance: ', round(explained_variance,4))
    print('R2: ', round(r2,4))
    print('Adjusted_r2: ', round(adjusted_r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

Below is a function to find the accuracy of each model on the basis of K-fold cross validation.

In [33]:
```
from sklearn.model_selection import cross_val_score
def model_accuracy(model,X_train=X_train,y_train=y_train):
    accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
    print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
    print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Fitting Multiple Linear Regression Model

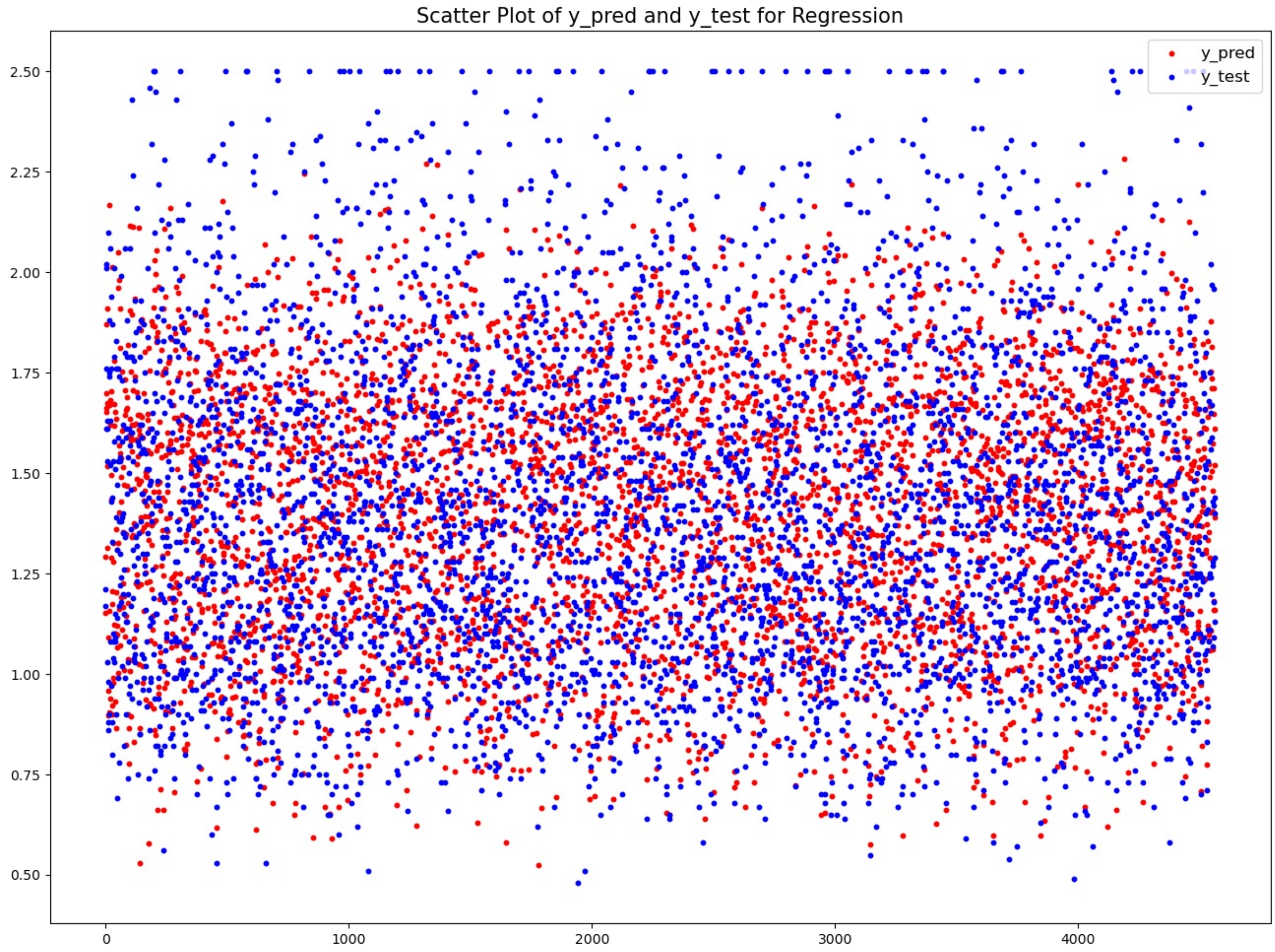
The following code snippet fits the multiple linear regression model on X_train and y_train and predicts the values for X_test and stores it in y_pred. It also prints the outputs of the functions defined above. Hence giving us a useful summary for the multiple linear regression model.

In [34]:
```
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm

regressor=LinearRegression()
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)
regression_results(y_test,y_pred)
model_accuracy(regressor)
```

```
Explained_variance:  0.6606
R2:  0.6606
Adjusted_r2:  0.6547
MAE:  0.1784
MSE:  0.054
RMSE:  0.2325
Accuracy: 64.10 %
Standard Deviation: 2.10 %
```

In [35]:
```
plt.figure(figsize=(16, 12),dpi=100)
red = plt.scatter(range(len(X_test)),y_pred,c='r',s = 10)
blue = plt.scatter(range(len(X_test)),y_test,c='b', s = 10)
plt.title("Scatter Plot of y_pred and y_test for Regression",fontsize=15)
plt.legend((red,blue),("y_pred","y_test"),scatterpoints=1, loc='upper right',fontsize=12)
```

Out[35]: <matplotlib.legend.Legend at 0x245dc921280>



The above scatterplot comprises of the original and predicted values of the multiple linear regression model.

Fitting Random Forest Regression Model.
The following code snippet fits the random forest regression model on X_train and y_train and predicts the values for X_test and stores it in y_pred_rf. It also prints the outputs of the functions defined above. Hence giving us a useful summary for the random forest regression model.

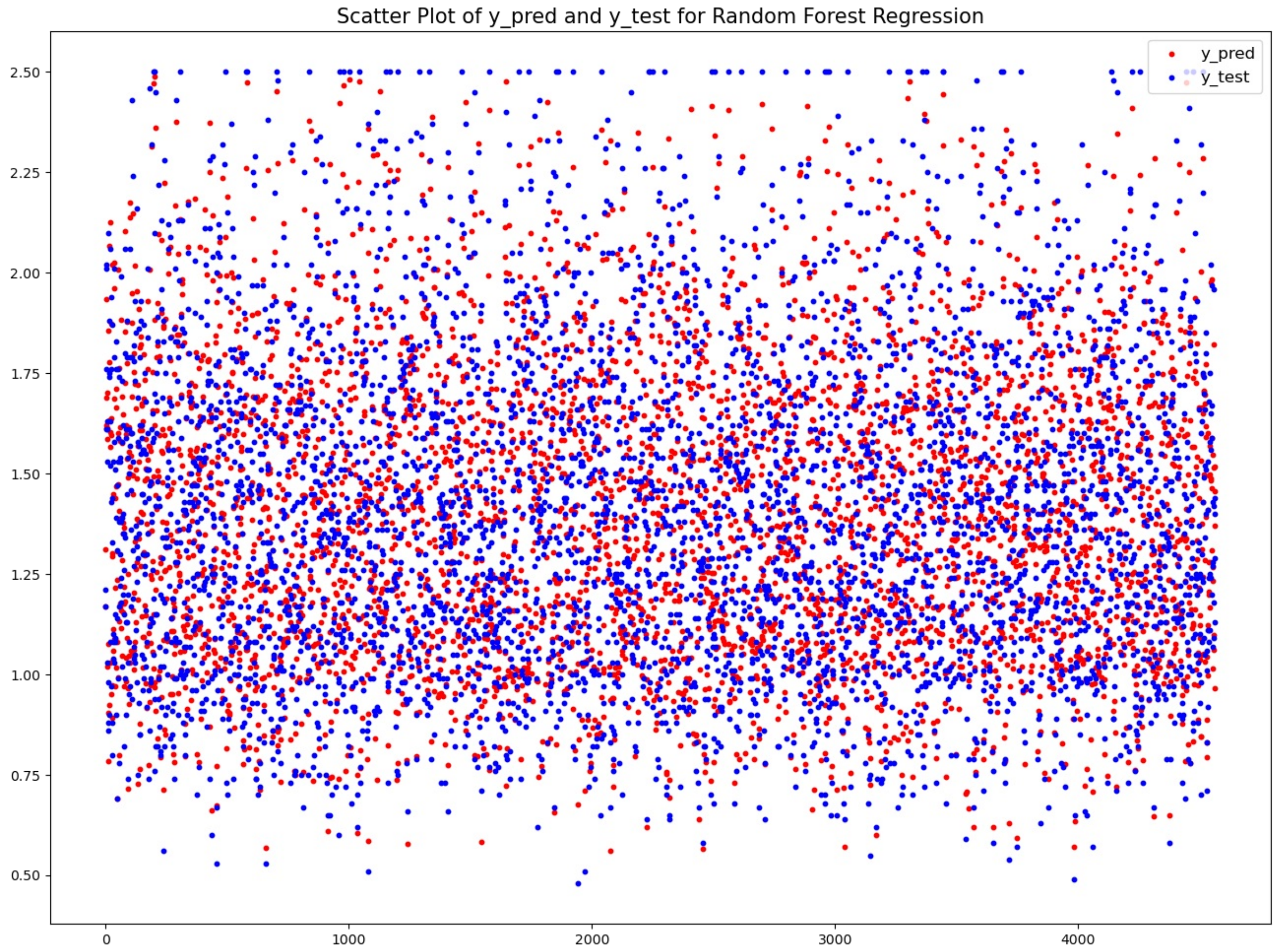
In [36]:
```
from sklearn.ensemble import RandomForestRegressor

classifier = RandomForestRegressor()
classifier.fit(X_train, y_train)
y_pred_rf = classifier.predict(X_test)
regression_results(y_test,y_pred_rf)
model_accuracy(classifier)
```

```
Explained_variance:  0.9003
R2:  0.9003
Adjusted_r2:  0.8985
MAE:  0.0908
MSE:  0.0159
RMSE:  0.126
Accuracy: 88.16 %
Standard Deviation: 1.00 %
```

In [37]:
```
plt.figure(figsize=(16, 12),dpi=100)
red=plt.scatter(range(len(X_test)),y_pred_rf,c='r',s = 10)
blue=plt.scatter(range(len(X_test)),y_test,c='b', s = 10)
```

```
plt.title("Scatter Plot of y_pred and y_test for Random Forest Regression",fontsize=15)
plt.legend((red,blue),("y_pred","y_test"),scatterpoints=1, loc='upper right',fontsize=12)
```

Out[37]: <matplotlib.legend.Legend at 0x245dcf42f70>



The above scatterplot comprises of the original and predicted values of the random forest regression model.

Fitting Decision Tree Regression Model

The following code snippet fits the decision tree regression model on X_train and y_train and predicts the values for X_test and stores it in y_pred_dt. It also prints the outputs of the functions defined above. Hence giving us a useful summary for the decision tree regression model.

In [38]:

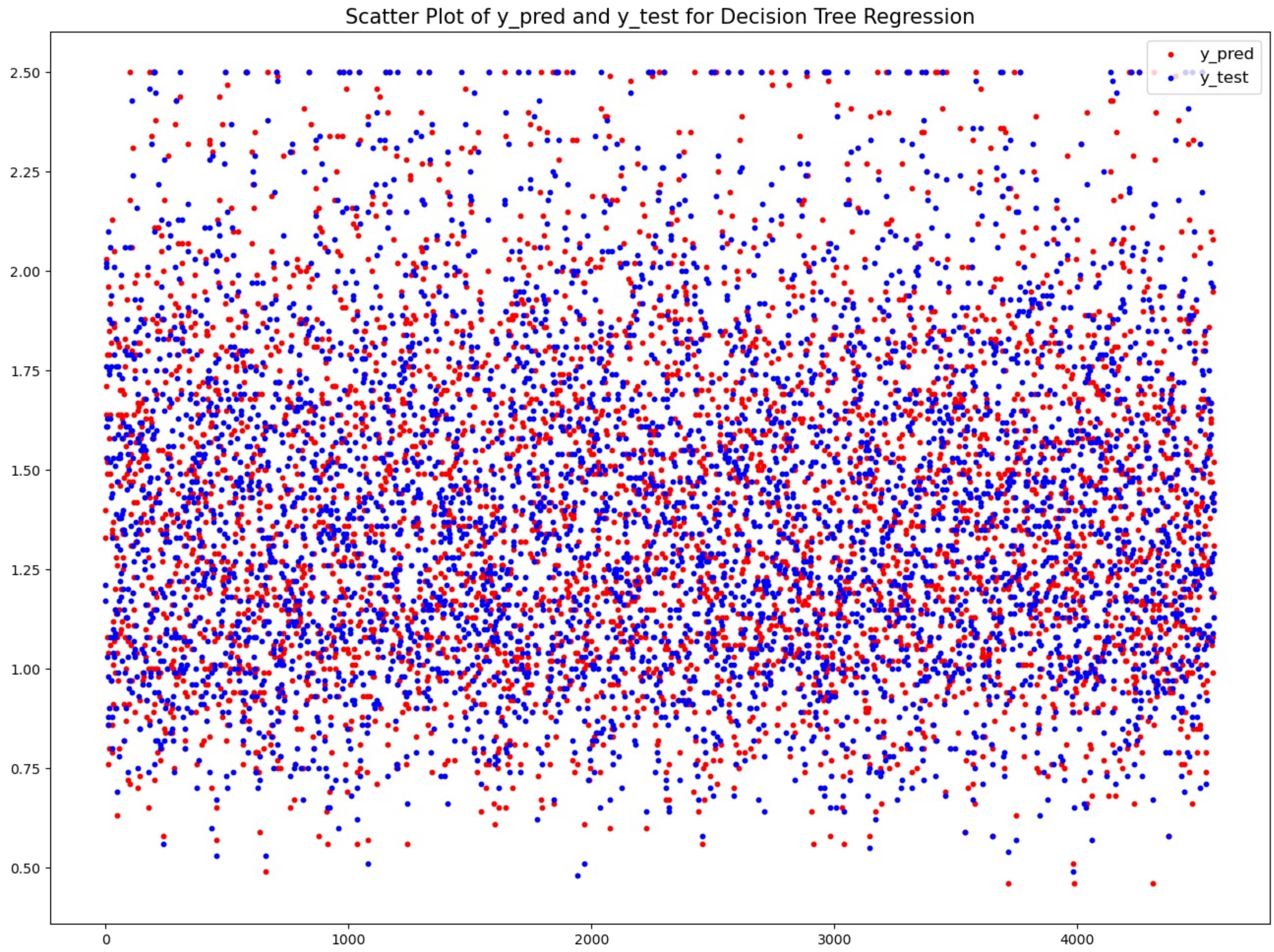
```
from sklearn.tree import DecisionTreeRegressor

decision_tree=DecisionTreeRegressor(criterion='mse',splitter='random',random_state=10)
decision_tree.fit(X_train, y_train)
y_pred_dt = decision_tree.predict(X_test)
regression_results(y_test,y_pred_dt)
model_accuracy(decision_tree)
```

```
Explained_variance:  0.8299
R2:  0.8298
Adjusted_r2:  0.8269
MAE:  0.109
MSE:  0.0271
RMSE:  0.1646
Accuracy: 80.41 %
Standard Deviation: 2.38 %
```

In [39]:

```
plt.figure(figsize=(16, 12),dpi=100)
red=plt.scatter(range(len(X_test)),y_pred_dt,c='r',s = 10)
blue=plt.scatter(range(len(X_test)),y_test,c='b', s = 10)
plt.title("Scatter Plot of y_pred and y_test for Decision Tree Regression",fontsize=15)
plt.legend((red,blue),("y_pred","y_test"),scatterpoints=1, loc='upper right',fontsize=12)
```

Out[39]: <matplotlib.legend.Legend at 0x245dcd4b7f0>



The above scatterplot comprises of the original and predicted values of the random forest regression model.

As our conclusion we proclaim that, using k-fold cross validation as the basis for model selection we declare random forest model as the best suited model for our purpose of predicting average avocado prices.

In [40]:

```
# Fit the model on training set
import pickle
model = RandomForestRegressor()
model.fit(X_train, y_train)
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

0.898496516757842

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js