Avacado Project taking the AveragePrice as Target column

In [2]:

```
#importing the necessary libraries for loading the data
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
# Creating the variable df and loading the dataset to the variable df
df=pd.read_excel('Avacoda.xlsx')
df
```

Out[3]:

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

18249 rows × 14 columns

In [4]:

```
#Checking the dataset for null values, if null value present we need to remove the null va
df.isnull().sum()
```

Out[4]:

```
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
```

From above we come to know that we donot have the null values in the data frame df

In [5]:

```
#cheking the datatype of df dataframe columns
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  datetime64[ns]
 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: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 1.9+ MB
```

In [6]:

```
# column unnamed does not have any impact on target column so we can drop that column
df.drop('Unnamed: 0',axis=1,inplace=True)
df.shape
```

Out[6]:

```
(18249, 13)
```

after deleting the unnamed column from df dataframe we have 18249rows and 13columns

In [8]:

```
# adding new columns month and date
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)
df
```

Out[8]:

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

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	ty
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventior
...	...	...	...	...	...	...	...	...	...	...	
18244	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	orgar
18245	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	orgar
18246	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	orgar
18247	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	orgar
18248	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	orgar

18249 rows × 15 columns

In [9]:

```
#checking the mean of price at each different years for organic type avacoda fruit
df.groupby('year')['AveragePrice'].mean()
```

Out[9]:

```
year
2015    1.375590
2016    1.338640
2017    1.515128
2018    1.347531
Name: AveragePrice, dtype: float64
```

average rate of avacoda in each year is shown in above table

In [10]:

```
# finding how much type of avacova is selling in each year
df.groupby('year')['type'].value_counts()
```

Out[10]:

```
year  type
2015  conventional    2808
      organic         2807
2016  conventional    2808
      organic         2808
2017  conventional    2862
      organic         2860
2018  conventional     648
      organic          648
Name: type, dtype: int64
```

selling rate for both type of avacova is almost same for every year (which we can see from the above table)

In [11]:

```
# to know the statistical data we use describe function
df.describe()
```

Out[11]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Lar
count	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04	1.824
mean	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04	2.396392e+05	1.821947e+05	5.433
std	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.862424e+05	7.461785e+05	2.439
min	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000

Loading [MathJax]/extensions/Safe.js

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Lar
25%	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000e+00	5.088640e+03	2.849420e+03	1.274
50%	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02	3.974383e+04	2.636282e+04	2.647
75%	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03	1.107834e+05	8.333767e+04	2.202
max	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06	1.937313e+07	1.338459e+07	5.719

1) Number of counts in each columns are same which means there is no null value in the data set

2) by seeing the 75% percentail value and max value we can say that outliers are present in the dataset which need to be removed

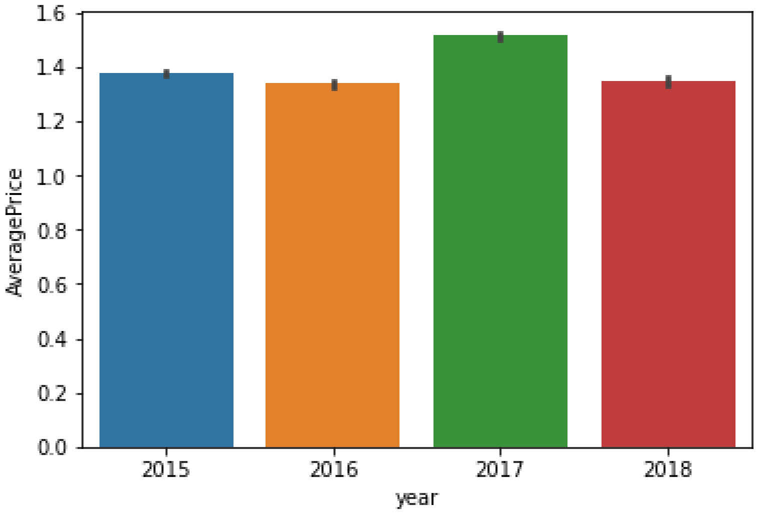
3) With the help of obove table we can know the statistical information of each columns in the data set

In [12]:

```
# checking for which year the max rate given for avacoda
# importing the matplotlip and seaborn library for visualization of data
import seaborn as sns
sns.barplot(x='year',y='AveragePrice',data=df)
```

Out[12]:

```
<AxesSubplot:xlabel='year', ylabel='AveragePrice'>
```



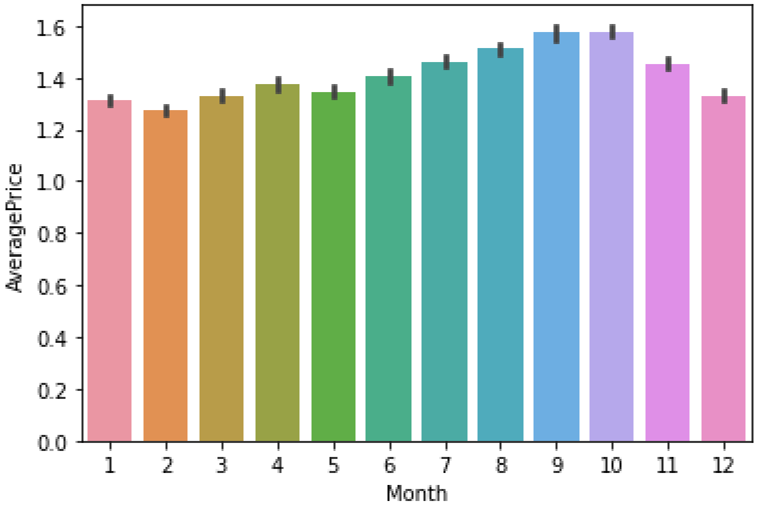
From the above graph we can say that 2017 is the year, avacoda got the maximun average price

In [13]:

```
sns.barplot(x='Month',y='AveragePrice',data=df)
```

Out[13]:

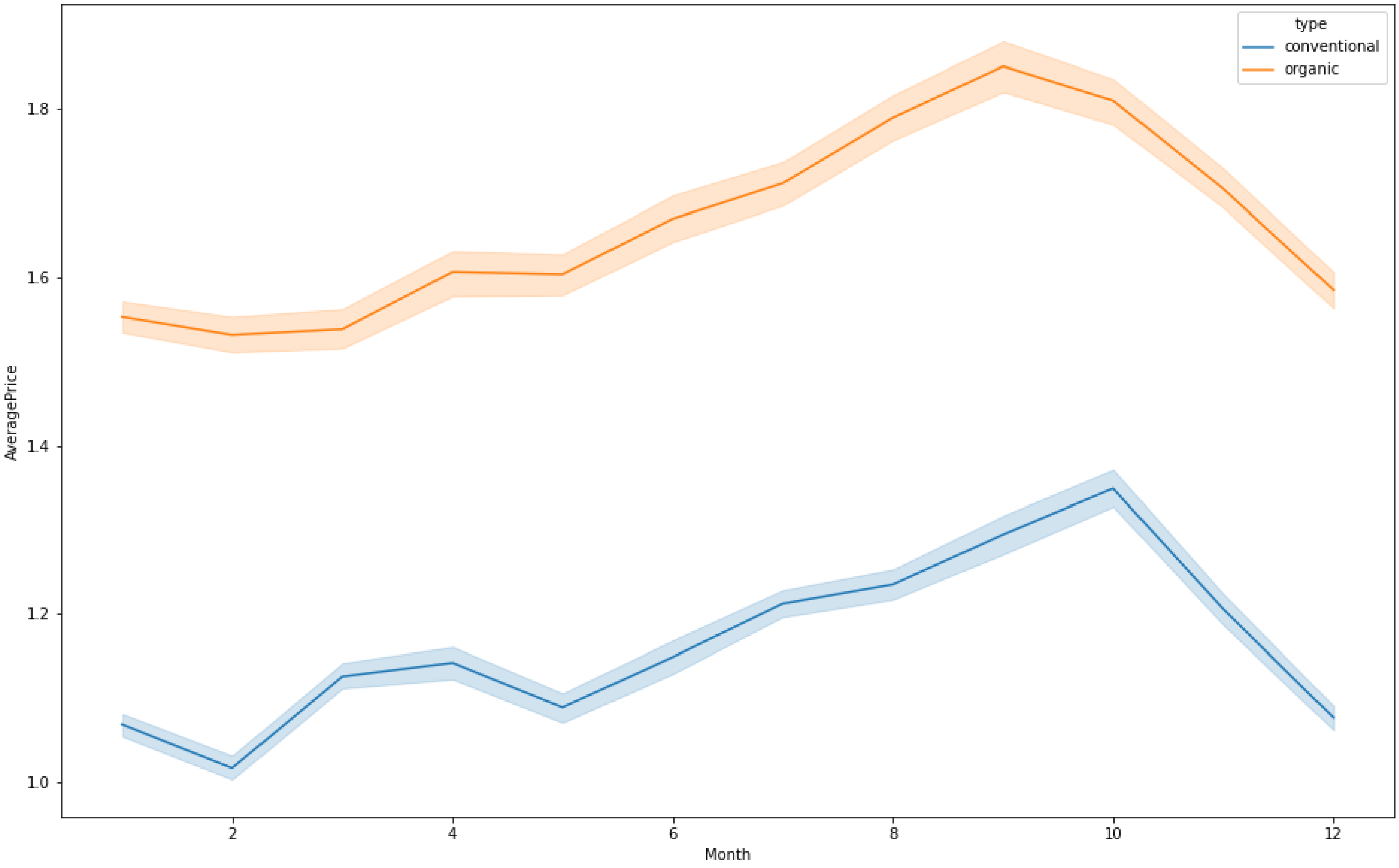
```
<AxesSubplot:xlabel='Month', ylabel='AveragePrice'>
```



From graph we can notice that the maximum average price in the 9th and 10th month and also there is no much difference in the average price if we consider all the months

In [14]:

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(16,10))
sns.lineplot(x='Month',y='AveragePrice',hue='type',data=df)
plt.show()
```



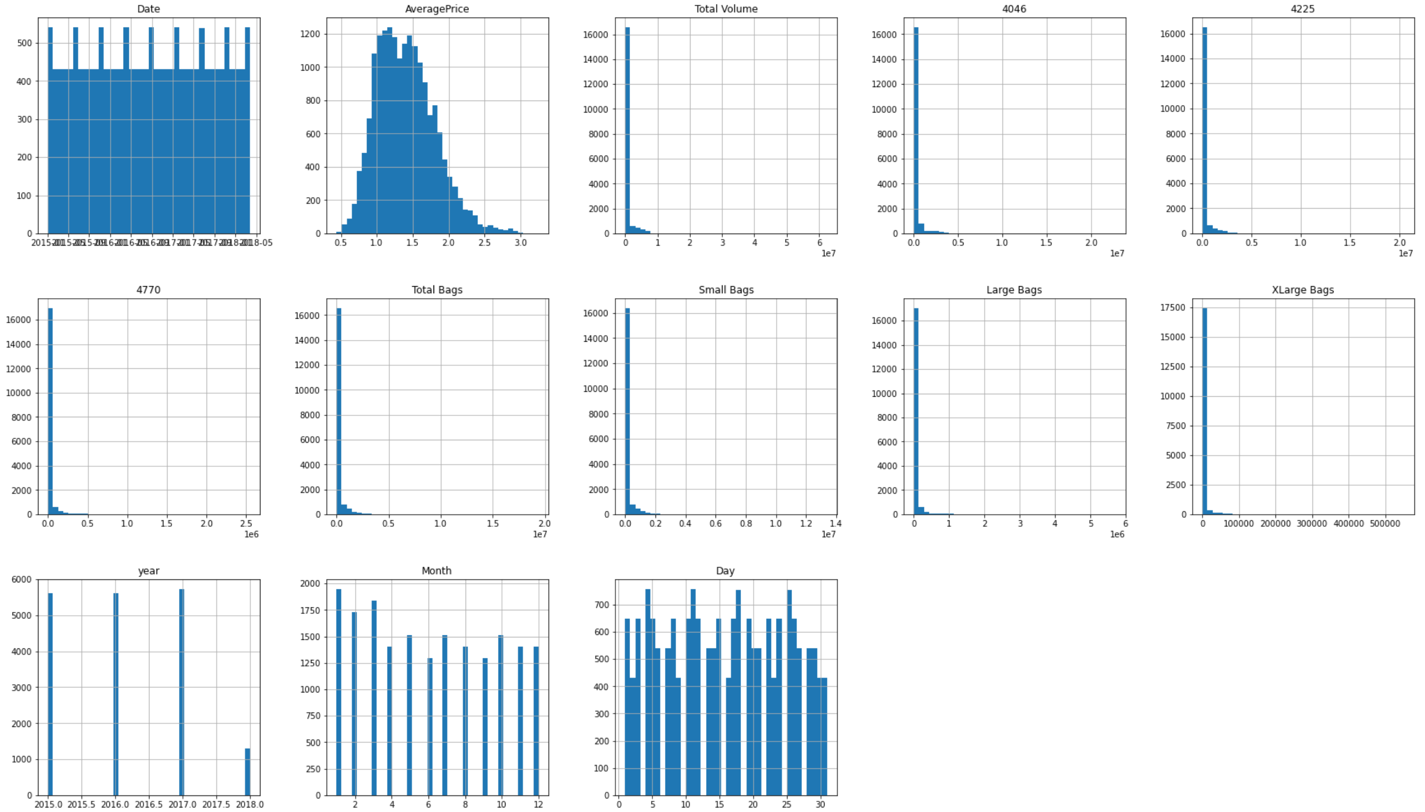
The averege price is more from 8th to 10th month which means that the sale is more in that months

In [15]:

```python
# plotting the histogram for univariant analysis and checking the normal distribution
df.hist(figsize=(30,30),grid=True,bins=40,layout=(5,5))
```

Out[15]:

```
array([[<AxesSubplot:title={'center':'Date'}>,
        <AxesSubplot:title={'center':'AveragePrice'}>,
        <AxesSubplot:title={'center':'Total Volume'}>,
        <AxesSubplot:title={'center':'4046'}>,
        <AxesSubplot:title={'center':'4225'}>],
       [<AxesSubplot:title={'center':'4770'}>,
        <AxesSubplot:title={'center':'Total Bags'}>,
        <AxesSubplot:title={'center':'Small Bags'}>,
        <AxesSubplot:title={'center':'Large Bags'}>,
        <AxesSubplot:title={'center':'XLarge Bags'}>],
       [<AxesSubplot:title={'center':'year'}>,
        <AxesSubplot:title={'center':'Month'}>,
        <AxesSubplot:title={'center':'Day'}>, <AxesSubplot:>,
        <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
        <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



From the graph we can say that the data is left skewed, skewness need to be removed

In [16]:

```
# changing the categorical value to numerical value by labelencoder tecnique
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
df['type']=le.fit_transform(df['type'])
df['region']=le.fit_transform(df['region'])
# we can drop the date column also beacuse it doesnot have much impact on target column
df.drop('Date',axis=1,inplace=True)
```

In [17]:

```
# all the categorical columns converted into numerical columns
df
```

Out[17]:

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

Loading [MathJax]/extensions/Safe.js

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	regi
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	0	2015	
...	...	...	...	...	...	...	...	...	...	...	...	
18244	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	1	2018	
18245	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	1	2018	
18246	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	1	2018	
18247	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	1	2018	
18248	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	1	2018	

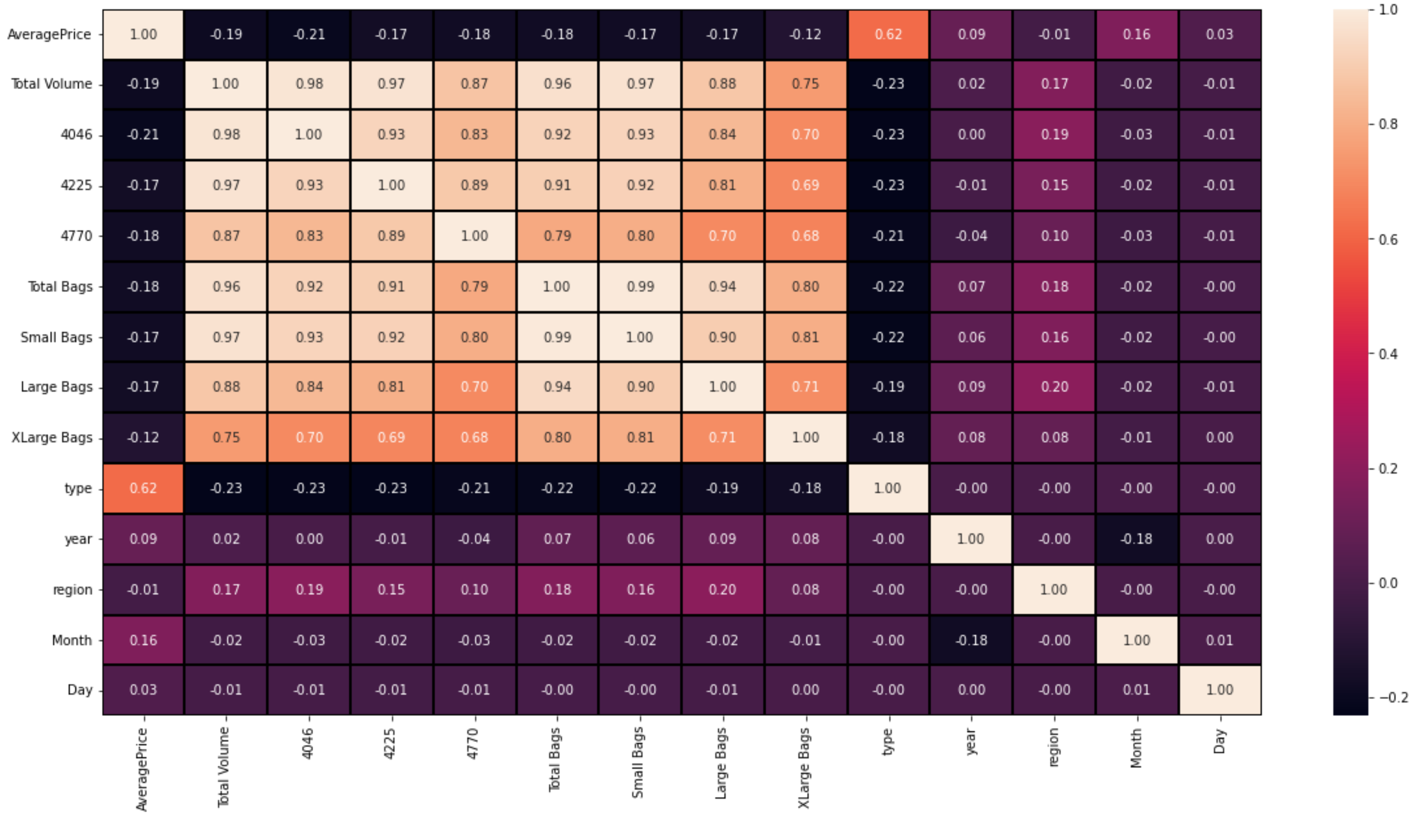
18249 rows × 14 columns

In [18]:

```
# checking for co-relation using heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True,linewidth=0.1,linecolor='black',fmt='0.2f')
```

Out[18]:

```
<AxesSubplot:>
```



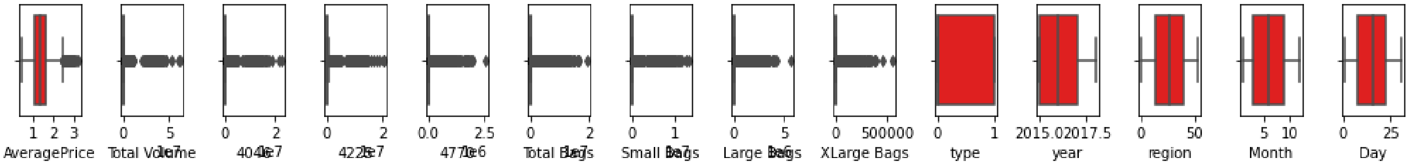
	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region	Month	Day
AveragePrice	1.00	-0.19	-0.21	-0.17	-0.18	-0.18	-0.17	-0.17	-0.12	0.62	0.09	-0.01	0.16	0.03
Total Volume	-0.19	1.00	0.98	0.97	0.87	0.96	0.97	0.88	0.75	-0.23	0.02	0.17	-0.02	-0.01
4046	-0.21	0.98	1.00	0.93	0.83	0.92	0.93	0.84	0.70	-0.23	0.00	0.19	-0.03	-0.01
4225	-0.17	0.97	0.93	1.00	0.89	0.91	0.92	0.81	0.69	-0.23	-0.01	0.15	-0.02	-0.01
4770	-0.18	0.87	0.83	0.89	1.00	0.79	0.80	0.70	0.68	-0.21	-0.04	0.10	-0.03	-0.01
Total Bags	-0.18	0.96	0.92	0.91	0.79	1.00	0.99	0.94	0.80	-0.22	0.07	0.18	-0.02	-0.00
Small Bags	-0.17	0.97	0.93	0.92	0.80	0.99	1.00	0.90	0.81	-0.22	0.06	0.16	-0.02	-0.00
Large Bags	-0.17	0.88	0.84	0.81	0.70	0.94	0.90	1.00	0.71	-0.19	0.09	0.20	-0.02	-0.01
XLarge Bags	-0.12	0.75	0.70	0.69	0.68	0.80	0.81	0.71	1.00	-0.18	0.08	0.08	-0.01	0.00
type	0.62	-0.23	-0.23	-0.23	-0.21	-0.22	-0.22	-0.19	-0.18	1.00	-0.00	-0.00	-0.00	-0.00
year	0.09	0.02	0.00	-0.01	-0.04	0.07	0.06	0.09	0.08	-0.00	1.00	-0.00	-0.18	0.00
region	-0.01	0.17	0.19	0.15	0.10	0.18	0.16	0.20	0.08	-0.00	-0.00	1.00	-0.00	-0.00
Month	0.16	-0.02	-0.03	-0.02	-0.03	-0.02	-0.02	-0.02	-0.01	-0.00	-0.18	-0.00	1.00	0.01
Day	0.03	-0.01	-0.01	-0.01	-0.01	-0.00	-0.00	-0.01	0.00	-0.00	0.00	-0.00	0.01	1.00

type column have high co relation with the target column(average price)

In [19]:

```
#check for outliers with the help of box plot
columns_list=df.columns.values
ncol=250
nrows=150
plt.figure(figsize=(ncol,ncol))
for i in range (0,len(columns_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df[columns_list[i]],color='red',orient='h')
    plt.tight_layout()
```

Loading [MathJax]/extensions/Safe.js



From the box plot we can conclude that outliers present in the dataset ,we need to remove the outliers

In [20]:

```
# checking how much number of outliers are present in the dataframe
from scipy.stats import zscore
z=np.abs(zscore(df))
print(df.shape)
print(z.shape)
threshold=3
print(np.where(z>3))
len(np.where(z>3)[0])
```

```
(18249, 14)
(18249, 14)
(array([  346,   359,   780, ..., 17304, 17402, 17428], dtype=int64), array([2, 2, 8, ...,
0, 0, 0], dtype=int64))
```

Out[20]:

```
1773
```

In [21]:

```
# 1773 outliers are present in the df dataset df
df_new=df[(z<3).all(axis=1)]
print(df_new.shape)
```

```
(17651, 14)
```

In [22]:

```
#df_new is the data frame after removing the outliers
df_new
```

Out[22]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	regi
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	0	2015	
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	0	2015	
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	0	2015	
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	0	2015	
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	0	2015	
...	...	...	...	...	...	...	...	...	...	...	...	
18244	1.63	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	1	2018	
18245	1.71	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	1	2018	
18246	1.87	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	1	2018	
18247	1.93	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	1	2018	
18248	1.62	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	1	2018	

17651 rows × 14 columns

In [23]:

```
# df_new1 is the dataframe after droping the Averageprice columns for assigning the x valu
df_new1=df_new.drop(columns=['AveragePrice'])
df_new1
```

Loading [MathJax]/extensions/Safe.js

Out[23]:

	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region	Month	Da
0	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	0	2015	0	12	2
1	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	0	2015	0	12	2
2	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	0	2015	0	12	1
3	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	0	2015	0	12	
4	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	0	2015	0	11	2
...	...	...	...	...	...	...	...	...	...	...	...	...	
18244	17074.83	2046.96	1529.20	0.00	13498.67	13066.82	431.85	0.0	1	2018	53	2	
18245	13888.04	1191.70	3431.50	0.00	9264.84	8940.04	324.80	0.0	1	2018	53	1	2
18246	13766.76	1191.92	2452.79	727.94	9394.11	9351.80	42.31	0.0	1	2018	53	1	2
18247	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	0.0	1	2018	53	1	1
18248	17489.58	2894.77	2356.13	224.53	12014.15	11988.14	26.01	0.0	1	2018	53	1	

17651 rows × 13 columns

df_new is the dataframe which is having the averageprice column df_new1 is the dataframe which is nothaving the averageprice column

In [24]:

```
# assigning the x and y values
x=df_new1.iloc[:,:]
y=df_new.iloc[:,0:1]
print(x)
print(y)
```

```
       Total Volume     4046       4225    4770  Total Bags  Small Bags  \
0          64236.62  1036.74   54454.85   48.16     8696.87     8603.62   
1          54876.98   674.28   44638.81   58.33     9505.56     9408.07   
2         118220.22   794.70  109149.67  130.50     8145.35     8042.21   
3          78992.15  1132.00   71976.41   72.58     5811.16     5677.40   
4          51039.60   941.48   43838.39   75.78     6183.95     5986.26   
...             ...      ...        ...     ...         ...         ...   
18244      17074.83  2046.96    1529.20    0.00    13498.67    13066.82   
18245      13888.04  1191.70    3431.50    0.00     9264.84     8940.04   
18246      13766.76  1191.92    2452.79  727.94     9394.11     9351.80   
18247      16205.22  1527.63    2981.04  727.01    10969.54    10919.54   
18248      17489.58  2894.77    2356.13  224.53    12014.15    11988.14   

       Large Bags  XLarge Bags  type  year  region  Month  Day  
0           93.25          0.0     0  2015       0     12   27  
1           97.49          0.0     0  2015       0     12   20  
2          103.14          0.0     0  2015       0     12   13  
3          133.76          0.0     0  2015       0     12    6  
4          197.69          0.0     0  2015       0     11   29  
...           ...          ...   ...   ...     ...    ...  ...  
18244      431.85          0.0     1  2018      53      2    4  
18245      324.80          0.0     1  2018      53      1   28  
18246       42.31          0.0     1  2018      53      1   21  
18247       50.00          0.0     1  2018      53      1   14  
18248       26.01          0.0     1  2018      53      1    7  

[17651 rows x 13 columns]
       AveragePrice
0              1.33
1              1.35
```

```
2              0.93
3              1.08
4              1.28
...             ...
18244          1.63
18245          1.71
18246          1.87
18247          1.93
18248          1.62

[17651 rows x 1 columns]
```

In [25]:

```
#removing the skewness by yeo johnson method
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
x
```

Out[25]:

```
array([[-0.07532391, -0.65742071,  0.35142978, ..., -2.01058988,
         1.49138301,  1.22756521],
       [-0.14749743, -0.77717852,  0.26951276, ..., -2.01058988,
         1.49138301,  0.53822879],
       [ 0.20555557, -0.7317763 ,  0.64933326, ..., -2.01058988,
         1.49138301, -0.21895878],
       ...,
       [-0.77527398, -0.61796649, -0.77889327, ...,  1.57828237,
        -1.64028838,  0.64019459],
       [-0.70179733, -0.54701087, -0.71637127, ...,  1.57828237,
        -1.64028838, -0.10529902],
       [-0.66738095, -0.35967077, -0.79165015, ...,  1.57828237,
        -1.64028838, -0.9581001 ]])
```

In [26]:

```
# checking the skweness after removing the skewness by yeo-johnson method
pd.DataFrame(x).skew()
```

Out[26]:

```
0    -0.008642
1    -0.039282
2    -0.044675
3     0.024168
4    -0.022204
5    -0.024405
6    -0.110913
7     0.853758
8    -0.037741
9     0.227731
10   -0.257799
11   -0.146554
12   -0.208926
dtype: float64
```

In [27]:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x1=mms.fit_transform(x)
```

In [28]:

```
# importing all the algorithems for checking the R2_score and model perforfance
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
```

Loading [MathJax]/extensions/Safe.js

In [29]:
```
model=[DecisionTreeRegressor(),KNeighborsRegressor(),AdaBoostRegressor(),LinearRegression(
max_r2_score=0
for i_state in range(10,20):
    x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=i_state,test_size=0.3
    for i in model:
        i.fit(x_train,y_train)
        pred=i.predict(x_test)
        r2_score1=r2_score(y_test,pred)
        #print('r2_score for random_state',i_state,'is',r2_score1)
        if r2_score1>max_r2_score:
            max_r2_score=r2_score1
            Final_state=i_state
            Final_model=i
print('R2_score is ',max_r2_score,'for random state ',Final_state, 'and model is ',Final_m
```

```
R2_score is  0.7764024736138327 for random state  18 and model is  KNeighborsRegressor()
```

In [30]:
```
# we are training the model with KNeighborsRegressor for randomstate 18 and checking the
kn=KNeighborsRegressor()
x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=18,test_size=0.33)
kn.fit(x_train,y_train)
kn.score(x_train,y_train)
pred_y=kn.predict(x_test)
kns=r2_score(y_test,pred_y)
print('r2_score =',kns*100)
```

```
r2_score = 77.64024736138327
```

In [31]:
```
# KNeighborsRegressor is giving the good R2_score so we are performing the hyperparameter
# for selecting the best parameters
knn_prams={'n_neighbors':range(1,30,2)}
knn=KNeighborsRegressor()
grid_search=GridSearchCV(knn,knn_prams)
grid_results=grid_search.fit(x_train,y_train)
grid_results.best_params_
```

Out[31]:
```
{'n_neighbors': 5}
```

In [32]:
```
knn1=KNeighborsRegressor(n_neighbors=5)
knn1.fit(x_train,y_train)
pred_y1=knn1.predict(x_test)
knn1s=r2_score(y_test,pred_y1)
print('R2_score ',knn1s*100)
```

```
R2_score  77.64024736138327
```

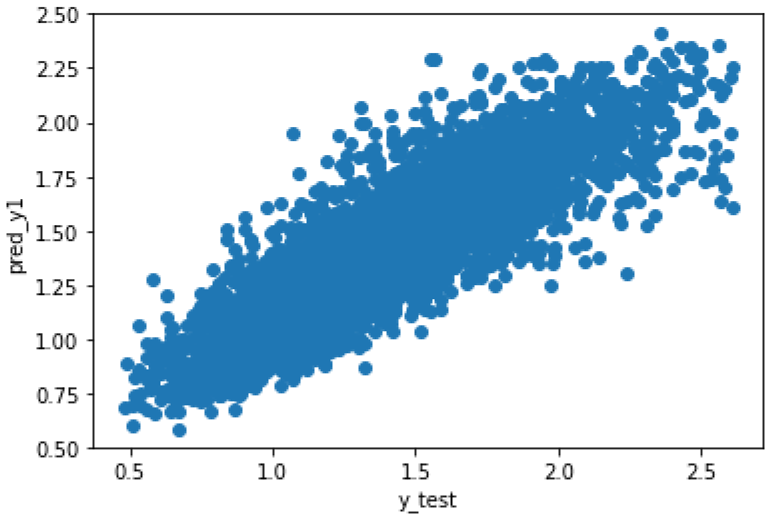
In [33]:
```
#checking for error values
print('mean_absolute_error is',mean_absolute_error(y_test,pred_y1))
print('mean_squared_error',mean_squared_error(y_test,pred_y1))
print('root mean absolute error',np.sqrt(mean_absolute_error(y_test,pred_y1)))
```

```
mean_absolute_error is 0.1345902145922747
mean_squared_error 0.033244514334763954
root mean absolute error 0.36686539028951026
```

In [34]:
```
#plotting the scatter plot
plt.scatter(x=y_test,y=pred_y1)
plt.xlabel('y_test')
plt.ylabel('pred_y1')
```

Loading [MathJax]/extensions/Safe.js

```
Out[34]: Text(0, 0.5, 'pred_y1')
```



In the problem the target column was continues so i have done the regression tecnique and for the KNeighborsRegressor i am getting the good R2 score (77.64) for the randomstate value 18 and hyperparameter tuning also done

From the above graph we can say that ,the actual values and predicted values are very close to each other so we can say that the line is best fit line

KNeighborsRegressor is giving the best R2_score, so i am saving the KNeighborsRegressor model