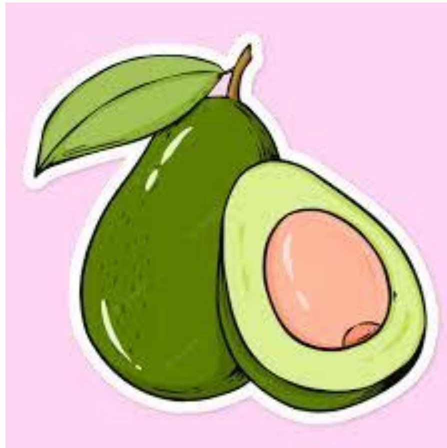
# A STEP WISE APPROACH TO CREATE A MACHINE LEARNING MODEL FOR AVOCADO DATASET

A Classification & Regression model building approach

# Table of Contents

## Introduction & Background



*It is a well-known fact that Millennials LOVE Avocado Toast. It's also a well-known fact that all Millennials live in their parents' basements.*

*Clearly, they aren't buying home because they are buying too much Avocado Toast!*

*But maybe there's hope… if a Millennial could find a city with cheap avocados, they could live out the Millennial American Dream.*

The aim of this blog is to help you understand how to create a Machine learning model & what are the necessary steps to predict the regions based on the dataset available at this link.

*(This blog is written with the assumption that the reader has preliminary understanding of machine learning terminologies & basic libraries machine learning libraries & intermediate level of understanding of python programming language.)*

We will understand all the necessary steps needed to clean & manipulate data & build ML model. So, lets go & dive into it…

# Problem Definition.

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 dataset 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., green skins) are not included in this dataset.

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

We will create one Classification model for predicting the region where avocados are sold & one regression model for predicting the average price for each avocado, type, year, volume, bag size and other features in the dataset.

We will import following machine learning libraries necessary for the model creation & validation;

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid')

import statsmodels.api as sm
from collections import Counter

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridSearchCV

#Logistic Regression
from sklearn.linear_model import LogisticRegression

# Random Forest Classifier & Gradient Boosting Classifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

# K Neighbors Classifier
from sklearn.neighbors import KNeighborsClassifier

#Classification model Evaluation metrics
from sklearn.metrics import confusion_matrix, classification_report, f1_score
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve, auc

import pickle

import warnings
warnings.filterwarnings('ignore')
```

```
#regression models
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor

#Regression evaluation metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

# Exploratory Data Analysis.

## Initial Data Analysis

- We will load the data using pandas ‘**pd.read_csv()**’ method.

```
data_url = "avocado.csv"
data0 = pd.read_csv(data_url)
```

- Reading the first & last 5 rows of data

```
data0.head()
```

| | 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 |

```
data0.tail()
```

| | Unnamed: 0 | Date | AveragePrice | Total Volume | 4046 | 4225 | 4770 | Total Bags | Small Bags | Large Bags | XLarge Bags | type | year | region |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | WestTexNewMexico |
| **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 | WestTexNewMexico |
| **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 | WestTexNewMexico |
| **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 | WestTexNewMexico |
| **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 | WestTexNewMexico |

**Observation:**

Here we see there are total 14 columns, out of which ‘region’ is our target/label column or dependent variable. We also see that there is a column named ‘Unnamed:’, it is an observation number from a specific region (you can check this using data0.head(55) & data0.sample(15)). Observations are taken on every 7th day from the last observation. We will remove this column as it brings no meaning our model.

Let’s understand our label column,

```
data0.region.nunique()
```

54

There are total 54 unique entries in target column, following are the values;

```
data0.region.unique()
```

```
array(['Albany', 'Atlanta', 'BaltimoreWashington', 'Boise', 'Boston',
       'BuffaloRochester', 'California', 'Charlotte', 'Chicago',
       'CincinnatiDayton', 'Columbus', 'DallasFtWorth', 'Denver',
       'Detroit', 'GrandRapids', 'GreatLakes', 'HarrisburgScranton',
       'HartfordSpringfield', 'Houston', 'Indianapolis', 'Jacksonville',
       'LasVegas', 'LosAngeles', 'Louisville', 'MiamiFtLauderdale',
       'Midsouth', 'Nashville', 'NewOrleansMobile', 'NewYork',
       'Northeast', 'NorthernNewEngland', 'Orlando', 'Philadelphia',
       'PhoenixTucson', 'Pittsburgh', 'Plains', 'Portland',
       'RaleighGreensboro', 'RichmondNorfolk', 'Roanoke', 'Sacramento',
       'SanDiego', 'SanFrancisco', 'Seattle', 'SouthCarolina',
       'SouthCentral', 'Southeast', 'Spokane', 'StLouis', 'Syracuse',
       'Tampa', 'TotalUS', 'West', 'WestTexNewMexico'], dtype=object)
```

**Observation:**

From above we see that, amongst the city name we also see there is an entry as 'TotalUS'. this means some observations were entered using total of all the regions in US. These entries should be dropped as we are predicting individual regions.

Going ahead with the next step we will check missing entries, data types, total number of unique entries in all features, column names & data size;

```
data0.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
```

```
data0.nunique()
```

```
Unnamed: 0         53
Date              169
AveragePrice      259
Total Volume    18237
4046            17702
4225            18103
4770            12071
Total Bags      18097
Small Bags      17321
Large Bags      15082
XLarge Bags      5588
type                2
year                4
region             54
dtype: int64
```

```
data0.columns
```

```
Index(['Unnamed: 0', 'Date', 'AveragePrice', 'Total Volume', '4046', '4225',
       '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type',
       'year', 'region'],
      dtype='object')
```

```
data0.shape
```

```
(18249, 14)
```

**Observation:**

- There are total **18249 observations** in dataset & **14 columns**, & at initial level there are no missing entries.
- **Date** is formatted as object datatype, we will convert it into week number & month number, which will be helpful for visualization. The data was gathered during 169 unique days spanned over **4 year**s.
- **Average price** is continuous type of numerical data.
- **type** is an object data type & it has 2 unique entries.
- type, year & region are the 3 categorical features from the dataset. from this we will remove year column as date is also there in the dataset.
- except Unnamed: 0, Date, type year & region all other features are float type continuous data.

## Descriptive Statistics of numerical data

```
#Descriptive statistics
data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **AveragePrice** | 18249.0 | 1.405978 | 4.026766e-01 | 0.44 | 1.10 | 1.37 | 1.66 | 3.25 |
| **Total Volume** | 18249.0 | 850644.013009 | 3.453545e+06 | 84.56 | 10838.58 | 107376.76 | 432962.29 | 62505646.52 |
| **4046** | 18249.0 | 293008.424531 | 1.264989e+06 | 0.00 | 854.07 | 8645.30 | 111020.20 | 22743616.17 |
| **4225** | 18249.0 | 295154.568356 | 1.204120e+06 | 0.00 | 3008.78 | 29061.02 | 150206.86 | 20470572.61 |
| **4770** | 18249.0 | 22839.735993 | 1.074641e+05 | 0.00 | 0.00 | 184.99 | 6243.42 | 2546439.11 |
| **Total Bags** | 18249.0 | 239639.202060 | 9.862424e+05 | 0.00 | 5088.64 | 39743.83 | 110783.37 | 19373134.37 |
| **Small Bags** | 18249.0 | 182194.686696 | 7.461785e+05 | 0.00 | 2849.42 | 26362.82 | 83337.67 | 13384586.80 |
| **Large Bags** | 18249.0 | 54338.088145 | 2.439660e+05 | 0.00 | 127.47 | 2647.71 | 22029.25 | 5719096.61 |
| **XLarge Bags** | 18249.0 | 3106.426507 | 1.769289e+04 | 0.00 | 0.00 | 0.00 | 132.50 | 551693.65 |

**Observation:**

- We see no missing observation from count. every feature has 18249 number of entries.
- There are some entries/observations where the minimum value of features was 0.0.

```
#percentage difference between mean & median
(abs((data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['mean'] -
      data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['50%'])*
     100/data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['mean'])).sort_values()
```

```
AveragePrice      2.558959
Total Bags       83.415138
Small Bags       85.530412
Total Volume     87.377004
4225             90.153966
Large Bags       95.127341
4046             97.049471
4770             99.190052
XLarge Bags     100.000000
dtype: float64
```

```
#percentage difference between 75% quantile & max
((data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['max'] -
  data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['75%'])*
 100/data0.drop(columns = ['Unnamed: 0', 'year', 'Date', 'type', 'region']).describe().T['max']).sort_values()
```

```
AveragePrice    48.923077
4225            99.266230
Total Volume    99.307323
Small Bags      99.377361
Total Bags      99.428160
4046            99.511862
Large Bags      99.614812
4770            99.754818
XLarge Bags     99.975983
dtype: float64
```

**Observation:**

- The mean & median % difference for all columns is more than 50% implying skewness in the dataset, except AveragePrice.
- features with % difference between 75% quantile & maximum value, more than 99% indicate possible outliers.

Dropping the 'Unnamed: 0' column & checking for duplicate & checking zero entry count.

## Checking for duplicate entries...

```
#dropping the unnamed & year column from the dataset.
data0.drop('Unnamed: 0', axis = 1, inplace =True)
```

```
data0.duplicated().sum()
```

```
0
```

### Observations:

1. No Duplicate entries.

## Checking entries with ZERO Value...

```
data0[data0 == 0.0].count()#*100/data0.shape[0]
```

```
Date              0
AveragePrice      0
Total Volume      0
4046            242
4225             61
4770           5497
Total Bags       15
Small Bags      159
Large Bags     2370
XLarge Bags   12048
type              0
year              0
region            0
dtype: int64
```

### Observation:

1. PLU 4046, 4225, & 4770 has 242, 61 & 5497 entries as zero respectively.
2. XLarge bags has 12048 entries with zero value, highest of the all features. Small, large & total bags also have zero entries.

We will rename all the columns to be able to easily understand & remove whitespaces. (This step is not necessary but I like to do it, whenever I see whitespace in column names)

## renaming column names to remove whitespaces...

```
columns = data0.columns.tolist()
columns_renamed = [column.strip().replace(" ", "_").lower() for column in columns]
data0.columns = columns_renamed
data0.rename(columns = {'4046':'plu_4046', '4225':'plu_4225', '4770':'plu_4770'}, inplace = True)
data0.columns
```

```
Index(['date', 'averageprice', 'total_volume', 'plu_4046', 'plu_4225',
       'plu_4770', 'total_bags', 'small_bags', 'large_bags', 'xlarge_bags',
       'type', 'year', 'region'],
      dtype='object')
```

Now we will copy the input data frame & do feature engineering for Date column on the copied dataframe as follows:

```python
#Copying dataframe
data1 = data0.copy()

data1['date'] = pd.to_datetime(data1['date'], errors='raise')
data1['week_number'] = data1['date'].dt.week
data1['month_number'] = data1['date'].dt.month
data1['year'] = data1['date'].dt.year

data1.drop('date', axis=1, inplace=True)
```

The new dataframe looks like as follows:

```python
data1
```

| | averageprice | total_volume | plu_4046 | plu_4225 | plu_4770 | total_bags | small_bags | large_bags | xlarge_bags | type | year | region | week_number | month_number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.33 | 64236.62 | 1036.74 | 54454.85 | 48.16 | 8696.87 | 8603.62 | 93.25 | 0.0 | conventional | 2015 | Albany | 52 | 12 |
| **1** | 1.35 | 54876.98 | 674.28 | 44638.81 | 58.33 | 9505.56 | 9408.07 | 97.49 | 0.0 | conventional | 2015 | Albany | 51 | 12 |
| **2** | 0.93 | 118220.22 | 794.70 | 109149.67 | 130.50 | 8145.35 | 8042.21 | 103.14 | 0.0 | conventional | 2015 | Albany | 50 | 12 |
| **3** | 1.08 | 78992.15 | 1132.00 | 71976.41 | 72.58 | 5811.16 | 5677.40 | 133.76 | 0.0 | conventional | 2015 | Albany | 49 | 12 |
| **4** | 1.28 | 51039.60 | 941.48 | 43838.39 | 75.78 | 6183.95 | 5986.26 | 197.69 | 0.0 | conventional | 2015 | Albany | 48 | 11 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18244** | 1.63 | 17074.83 | 2046.96 | 1529.20 | 0.00 | 13498.67 | 13066.82 | 431.85 | 0.0 | organic | 2018 | WestTexNewMexico | 5 | 2 |
| **18245** | 1.71 | 13888.04 | 1191.70 | 3431.50 | 0.00 | 9264.84 | 8940.04 | 324.80 | 0.0 | organic | 2018 | WestTexNewMexico | 4 | 1 |
| **18246** | 1.87 | 13766.76 | 1191.92 | 2452.79 | 727.94 | 9394.11 | 9351.80 | 42.31 | 0.0 | organic | 2018 | WestTexNewMexico | 3 | 1 |
| **18247** | 1.93 | 16205.22 | 1527.63 | 2981.04 | 727.01 | 10969.54 | 10919.54 | 50.00 | 0.0 | organic | 2018 | WestTexNewMexico | 2 | 1 |
| **18248** | 1.62 | 17489.58 | 2894.77 | 2356.13 | 224.53 | 12014.15 | 11988.14 | 26.01 | 0.0 | organic | 2018 | WestTexNewMexico | 1 | 1 |

18249 rows × 14 columns

## Data Visualization

We will make two group for data visualization purpose;

### Numerical features :

1. averageprice
2. total_volume
3. plu_4046
4. plu_4225
5. plu_4770
6. total_bags
7. small_bags
8. large_bags
9. xlarge_bags

### Categorical features :

1. type
2. year
3. region
4. week_number
5. month_number

Converting Some of the features in to proper data type for visualization purpose. & Creating two dataframes for numerical features & categorical features;

```python
data1['year'] = data1['year'].astype('object')
data1['week_number'] = data1['week_number'].astype('object')
data1['month_number'] = data1['month_number'].astype('object')
```

```python
month_dict = {1:'January',2:'February',3:'March',4:'April',5:'May',6:'June',
             7:'July',8:'August',9:'September',10:'October',11:'November',12:'December'}
```

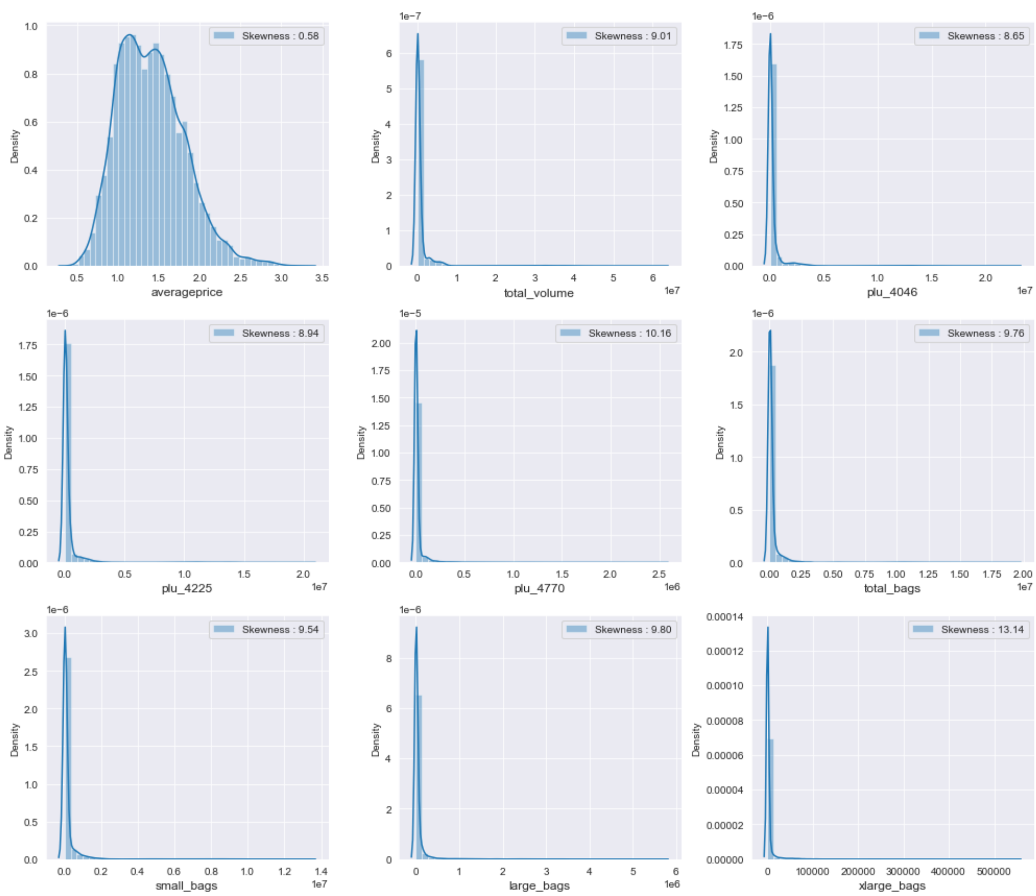
```python
data1['month_number'] = data1['month_number'].map(month_dict)
```

```python
data1_num = data1[num_cols]
data1_cat = data1[cat_cols]
```

## Univariate Analysis

### For Numerical features:

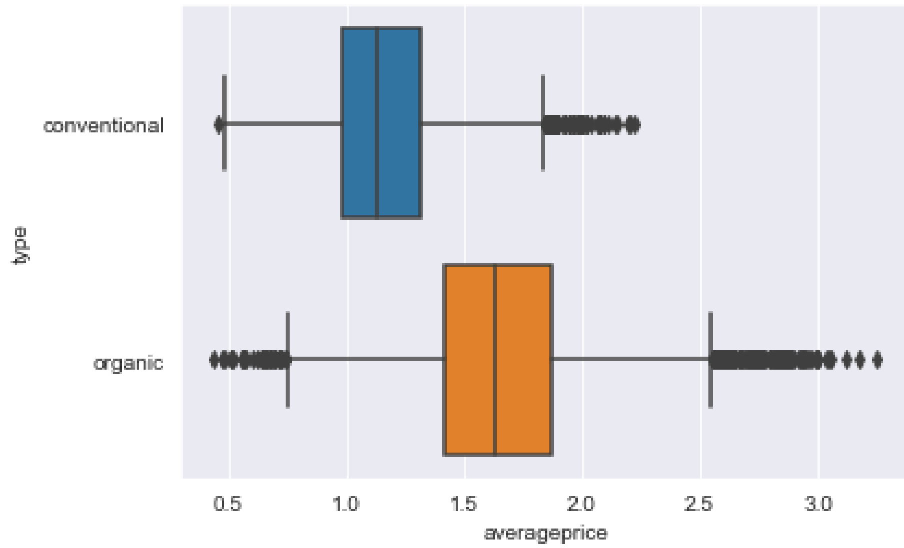
```python
#distribution of numerical features..
i=0
plt.figure(figsize=(14,12))
for column in data1_num.columns:
    plt.subplot(3,3,i+1)
    sns.distplot(data1_num[column], bins=40, label="Skewness : %.2f"%(data1_num[column].skew())).legend(loc="best")
    plt.xlabel(column,fontsize=12)
    i+=1

plt.tight_layout()
```




**Observation:**

- All the numerical features except average price are skewed & it is evident from the above distribution plots.

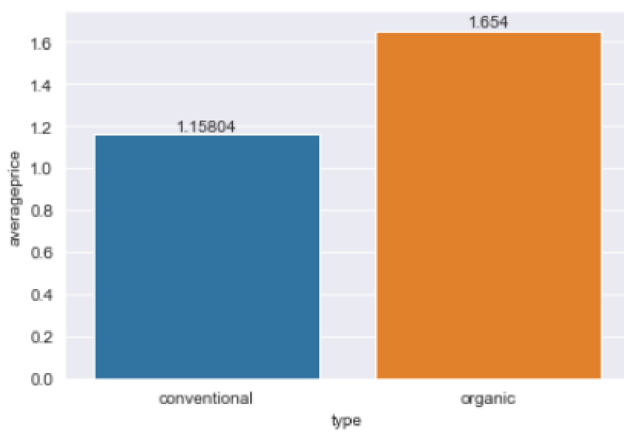
## For categorical features:




**Observation:**

- We can say that Conventional type Avocados are cheaper than the Organic type of Avocado.
- Organic type of avocado has average price ranging from 0.4 to 3.3, with mean value close to 1.65.
- Conventional type of avocado has avg price within 0.4 to 2.4, & mean being closer to 1.15.

```
Average price of avocado sold of each type

type
conventional    1.158040
organic         1.653999
Name: averageprice, dtype: float64
```

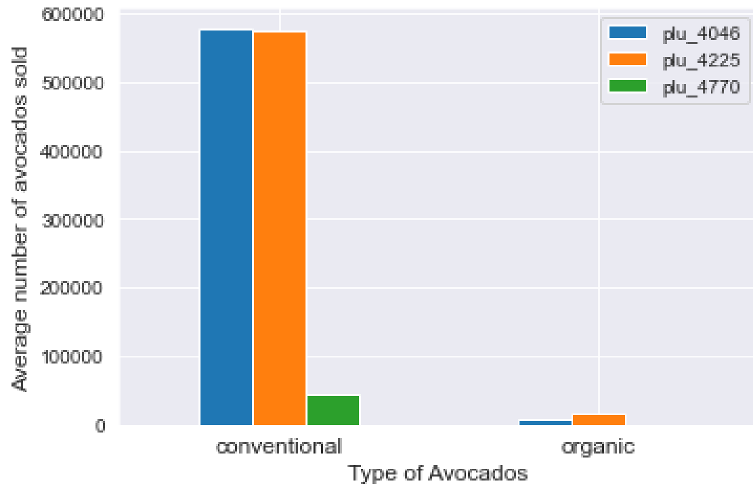



**Observation:**

- An organic type of Avocado is about 50 cents costlier than conventional type.

## Bivariate Analysis

1. Average number of avocados sold based on product lookup codes of each type;

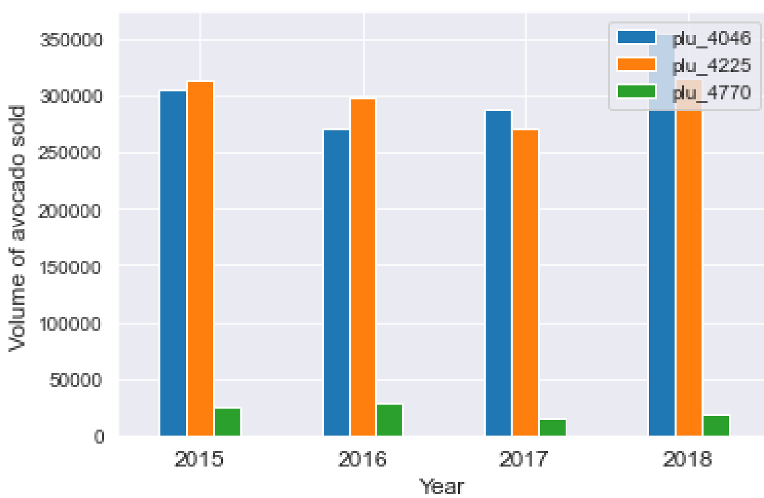



| type | plu_4046 | plu_4225 | plu_4770 |
|---|---|---|---|
| conventional | 578611.649925 | 574805.318859 | 45405.796798 |
| organic | 7311.281600 | 15411.857724 | 266.254582 |

**Observation:**

- Mostly Conventional type of avocados with package label 4046 & 4225 are sold.

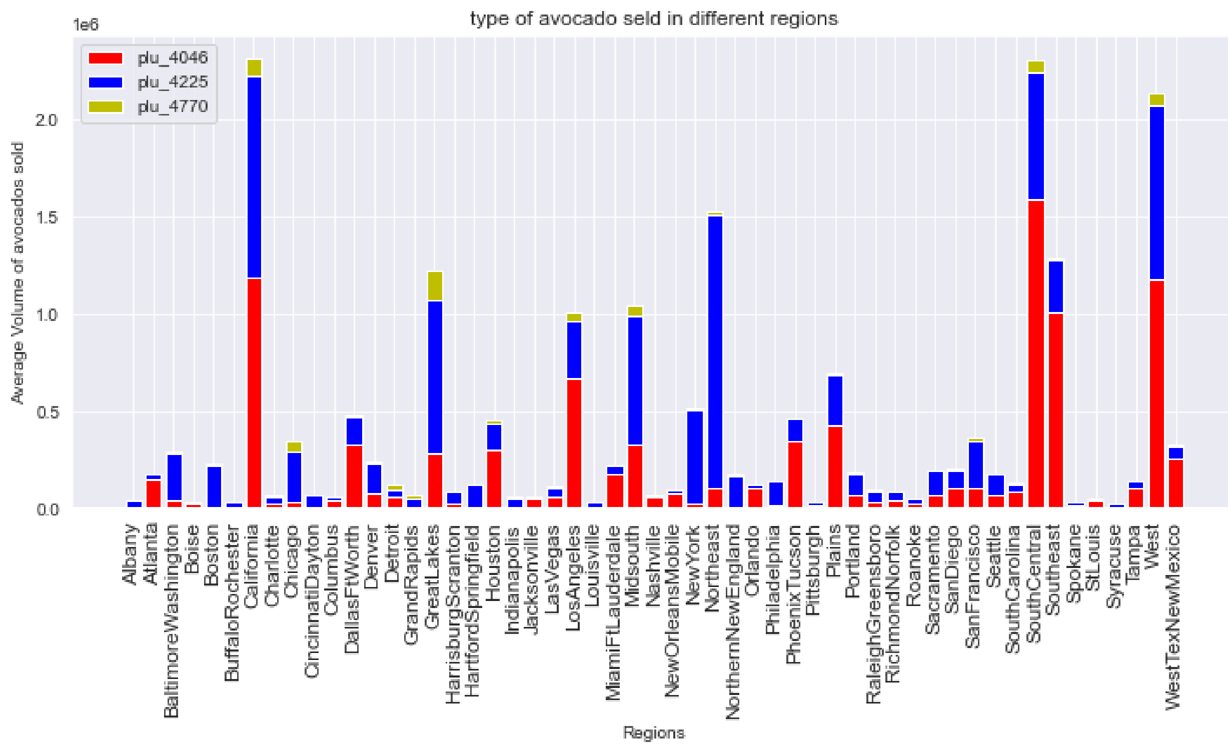
2. Average number of avocados sold based on bags size of each year




**Observation:**

- In all year of the data available, we see Conventional type of avocados with package label 4046 & 4225 are sold most in terms of volumes.

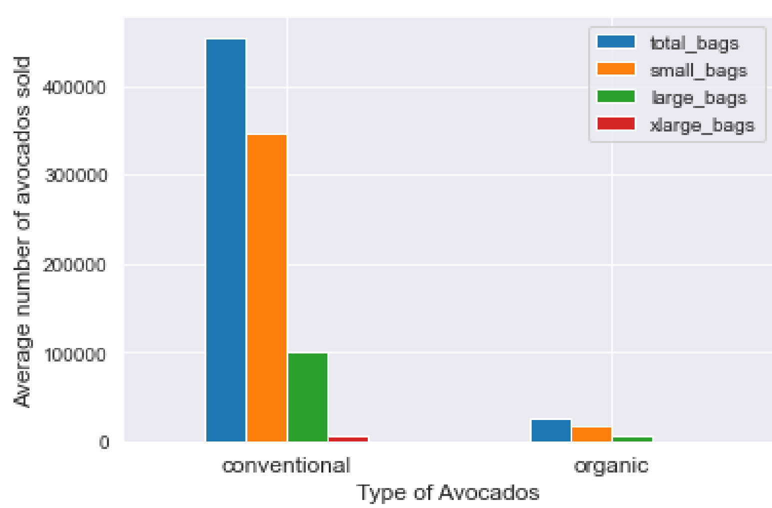
3. Average volume of avocado sold in different regions based on package label type;




**observation:**

- California, South central, South east & west regions are the regions with most consumption of avocados. i.e., higher demand from these 4 regions.

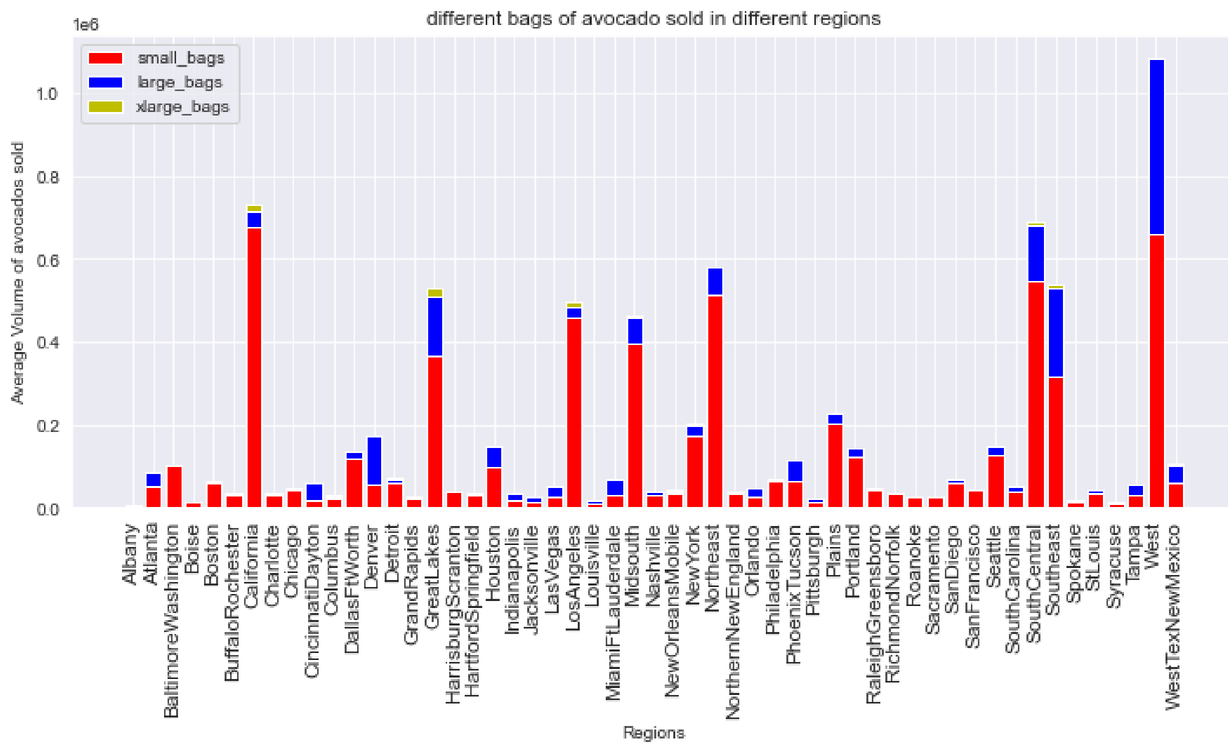
4. Average number of avocados sold based on bags size of each type;




**observation:**

- Organic type of avocado is very less consumptive but higher price.
- Conventional type of avocado is sold heavily with higher volume & lower price.
- Smaller bags of conventional avocado is sold in much bigger volume than other bag sizes.

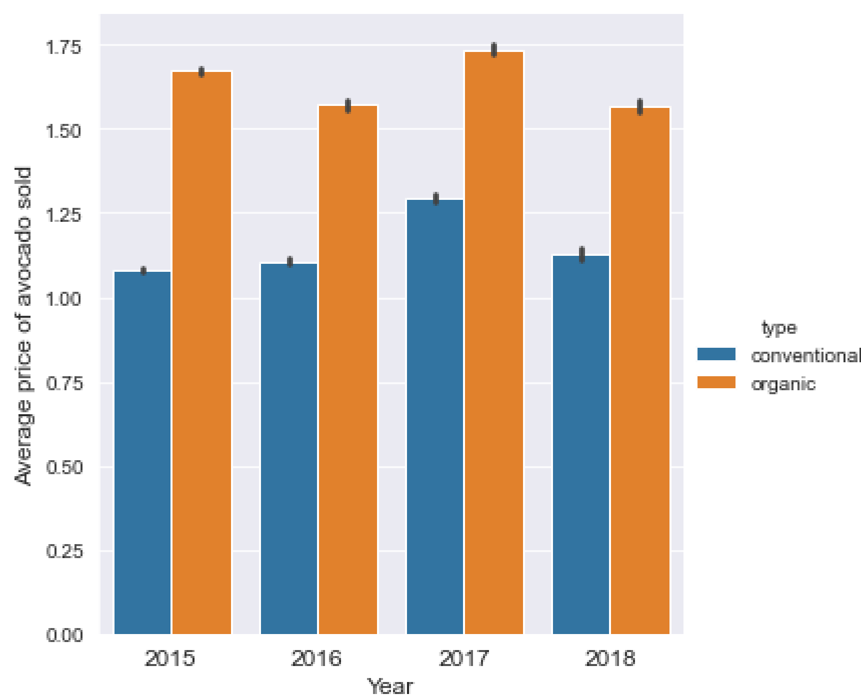
5. Different bags of avocado sold in different regions based on bag type;



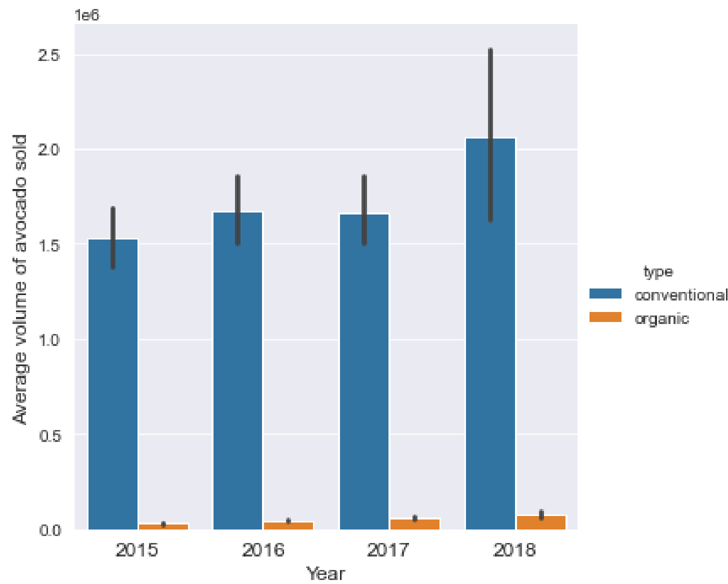

**Observation:**

- Smaller bags of avocados are sold heavily in all the regions.

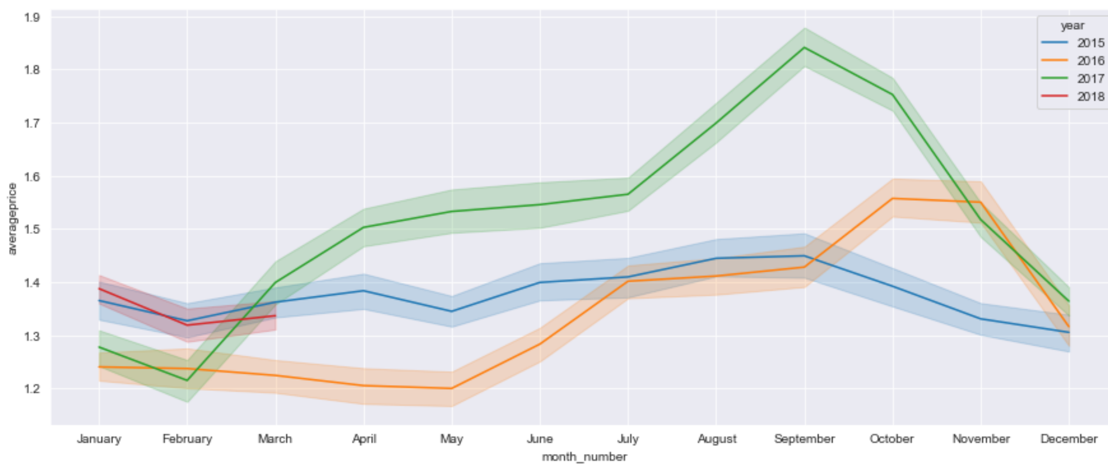
6. Average price of each type of avocado sold each year;



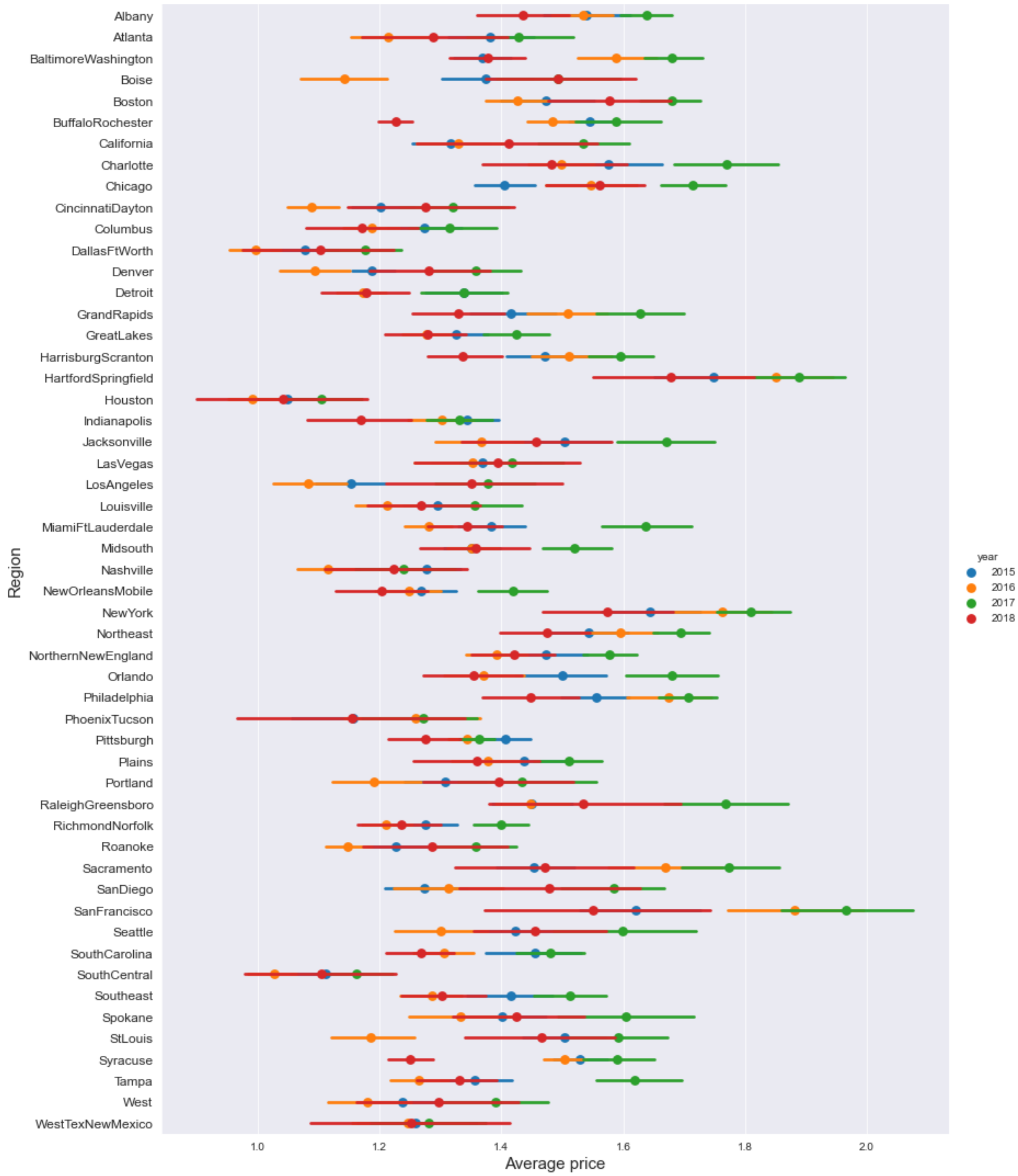
7. Average number of each type of avocado sold each year;




8. Average price variation during each year



9. Average price variation in all regions during each year




In 2015:

- average price of avocado was higher in Hart fort Springfield, New York, San Francisco, charlotte, philly. price was above 1.5 per unit.
- Houston, Dallas. South Central & LA had lower price per unit.
- Volume consumption was higher in California, West & South-central regions

In 2016:

- average price of avocado was higher in Hart fort Springfield, New York, San Francisco, philly, Sacramento & Northeast regions. price was above 1.6 per unit.
- Houston, Dallas. South Central & LA had lower price per unit.
- Volume consumption was higher in California, West & South-central regions

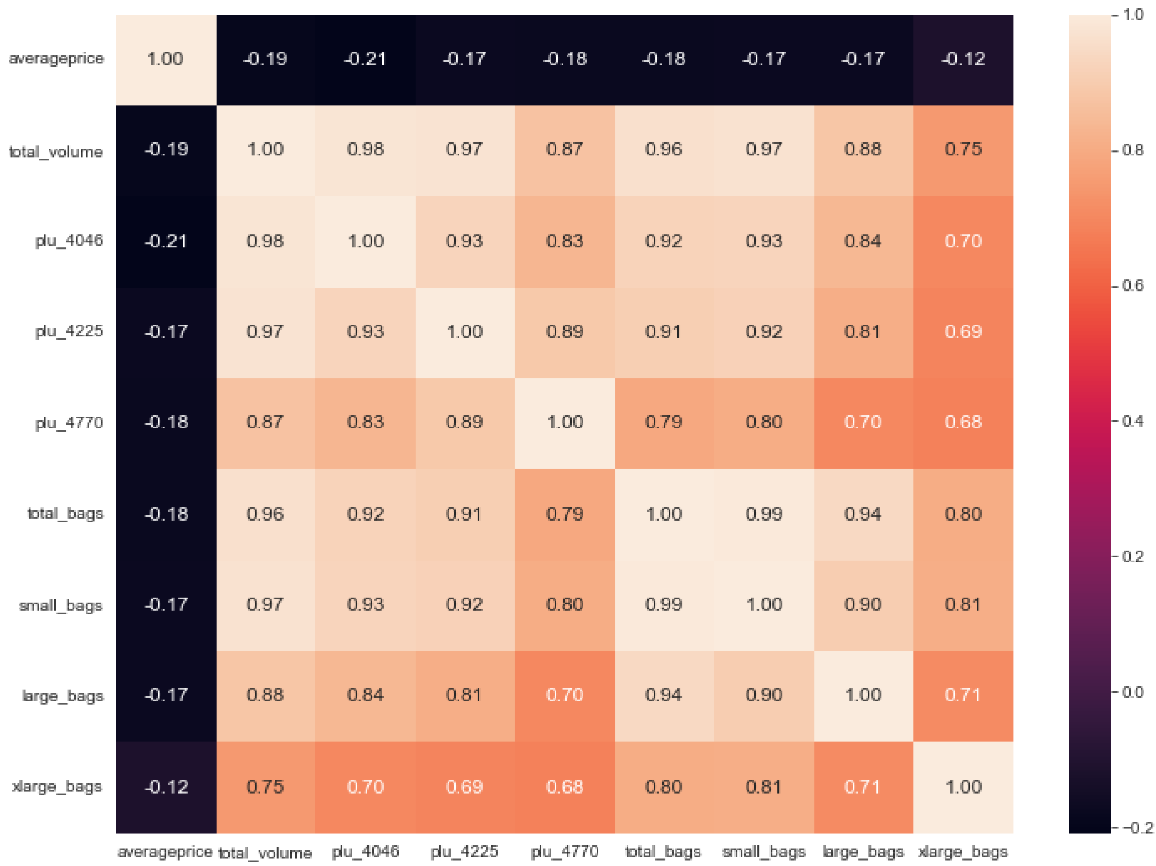
In 2017:

- average price of avocado was higher in hart fort Springfield, New York, San Francisco, charlotte, Sacramento regions. price was above 1.75 per unit.
- Houston, Dallas. South Central & Nashville had lower price per unit.
- Volume consumption was higher in California, West & South-central regions

In 2018:

- Data was only of 3 months, so it won't make sense to compare with other years data.

10. Correlation matrix of numerical features;




**Observation:**

- All Numerical features seem to have very low correlation with target variable 'Average Price' & high correlation with each other, hinting about multicollinearity.

# Pre-Processing Data.

1. In this step we will make week number, month number & year as categorical datatypes.

   Copying the dataframe & converting all the necessary features in to proper datatypes;

```
#Copying dataframe
data2 = data0.copy()

data2['date'] = pd.to_datetime(data2['date'], errors='raise')
data2['week_number'] = data2['date'].dt.week
data2['month_number'] = data2['date'].dt.month
data2['year'] = data2['date'].dt.year

data2.drop('date', axis=1, inplace=True)

data2['year'] = data2['year'].astype('str')
data2['week_number'] = data2['week_number'].astype('str')
data2['month_number'] = data2['month_number'].astype('str')
```

2. Checking Skewness & Removing using Power transformation

```
data2[['total_volume', 'plu_4046', 'plu_4225', 'plu_4770',
       'total_bags', 'small_bags', 'large_bags']].skew()
```

```
total_volume     9.007687
plu_4046         8.648220
plu_4225         8.942466
plu_4770        10.159396
total_bags       9.756072
small_bags       9.540660
large_bags       9.796455
dtype: float64
```

### Observation

1. All the numerical features have skewness. we will remove it using yeo-johnson transformation

```
x = data2[['total_volume', 'plu_4046', 'plu_4225', 'plu_4770',
       'total_bags', 'small_bags', 'large_bags']]
```

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='yeo-johnson', standardize=True)

x_pt = pt.fit_transform(x)
x_pt = pd.DataFrame(x_pt, columns = x.columns)
```

```
num_feat = ['total_volume', 'plu_4046', 'plu_4225', 'plu_4770', 'total_bags', 'small_bags', 'large_bags']
for col in num_feat:
    data2[col]=x_pt[col]
```

```
data2[['total_volume', 'plu_4046', 'plu_4225', 'plu_4770',
       'total_bags', 'small_bags', 'large_bags']].skew()
```

```
total_volume    0.011171
plu_4046       -0.022679
plu_4225       -0.017740
plu_4770        0.032184
total_bags     -0.001626
small_bags      0.007058
large_bags     -0.073809
dtype: float64
```

### Observation

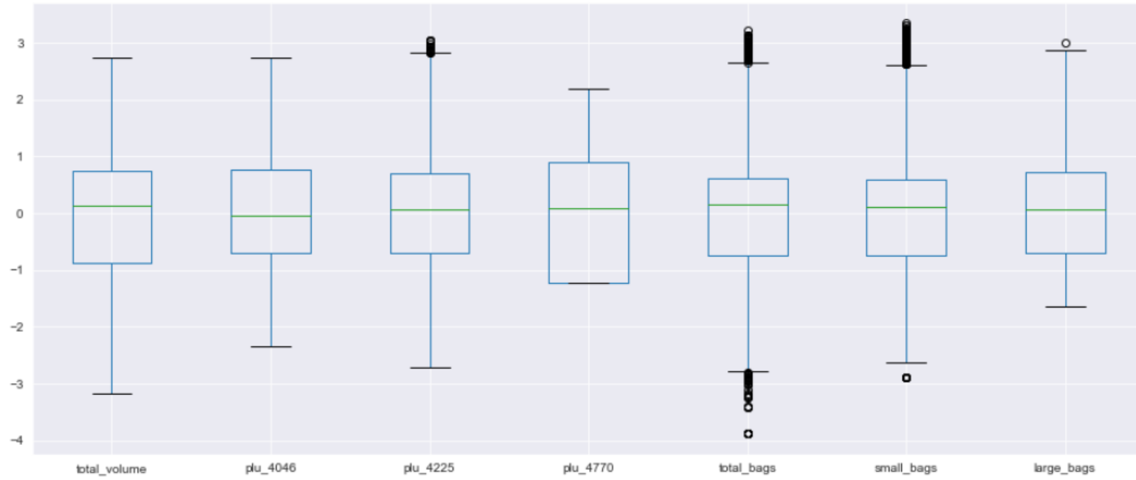
1. skewness in all numerical features is removed & are well within skewness of +/-0.5.

3. Checking outliers & removing if any using IQR Method

```
data3 = data2.copy()
```

```
data_out = data2[['total_volume', 'plu_4046', 'plu_4225', 'plu_4770',
       'total_bags', 'small_bags', 'large_bags']]
```

```
data_out.iloc[:,:].boxplot(figsize = (16,8))
plt.subplots_adjust(bottom=0.25)
plt.show()
```




```
#  creating a function to remove outliers using inter quantile range...
def get_outliers_iqr(data):

    q1 = data.quantile(0.25)
    q3 = data.quantile(0.75)
    iqr = q3-q1

    #empty list to store index values
    all_indices = []

    for column in data.columns:
        lower, upper = (q1[column] - (1.5*iqr[column])) , (q3[column] + (1.5*iqr[column]))

        index = np.where((data[column] < lower) | (data [column] > upper))

        all_indices.extend(index[0])

    set_res = set(all_indices)
    indices_to_remove = np.array(list(set_res))
    return indices_to_remove
```

```
data_out = data2[['total_volume', 'plu_4046', 'plu_4225', 'plu_4770',
       'total_bags', 'small_bags', 'large_bags']]
```

```
to_remove_iqr = get_outliers_iqr(data_out)
data_out = data_out.drop(data_out.index[to_remove_iqr])
data3_iqr = data3.drop(data3.index[to_remove_iqr])
```

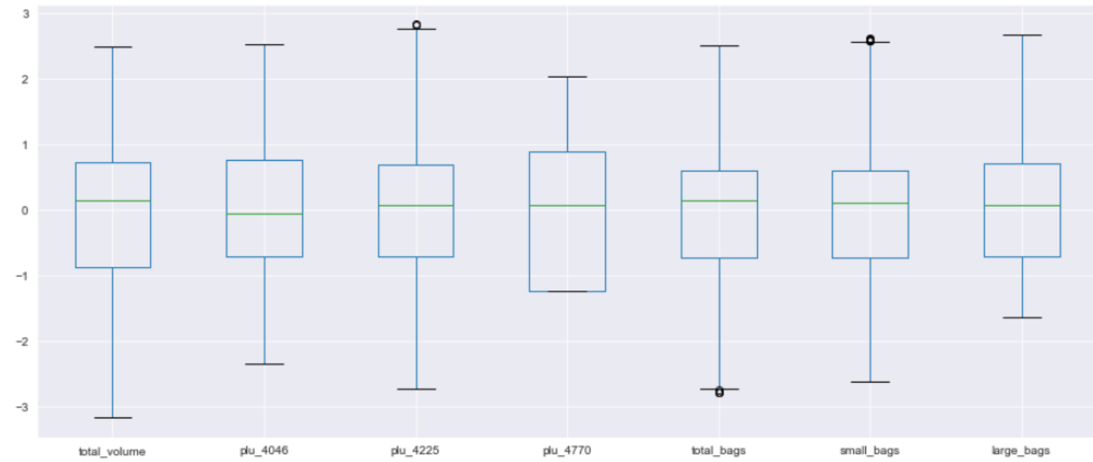
```
total_data_loss = (data3.shape[0] - data3_iqr.shape[0])*100/data3.shape[0]
print('Total Data loss after removing Outliers : ', round(total_data_loss,2))
```

```
Total Data loss after removing Outliers :  1.9
```

**Observation:**

- We see that total data loss after removing outliers was 1.9%.

```python
data_out.iloc[:,:].boxplot(figsize = (16,8))
plt.subplots_adjust(bottom=0.25)
plt.show()
```




### Observation

1. using IQR Method, outliers removed from all the columns
2. Data loss after removing outliers using IQR is : 1.9%

4. We will Remove all the observations where region column has ‘TotalUS’ entry;

```python
data5 = data3_iqr.copy()
data6 = data5.drop(columns = ['total_bags'])
data6 = data6.loc[data6['region'] != 'TotalUS']
data6
```

| | averageprice | total_volume | plu_4046 | plu_4225 | plu_4770 | small_bags | large_bags | xlarge_bags | type | year | region | week_number |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.33 | -0.086605 | -0.658619 | 0.307313 | -0.247250 | -0.340627 | -0.773351 | 0.0 | conventional | 2015 | Albany | 52 |
| **1** | 1.35 | -0.155746 | -0.776870 | 0.231647 | -0.201041 | -0.306425 | -0.763435 | 0.0 | conventional | 2015 | Albany | 51 |
| **2** | 0.93 | 0.179327 | -0.731947 | 0.579503 | -0.007072 | -0.366303 | -0.750820 | 0.0 | conventional | 2015 | Albany | 50 |
| **3** | 1.08 | 0.003876 | -0.634200 | 0.415100 | -0.148322 | -0.496853 | -0.691865 | 0.0 | conventional | 2015 | Albany | 49 |
| **4** | 1.28 | -0.187640 | -0.685293 | 0.224806 | -0.137918 | -0.477201 | -0.600975 | 0.0 | conventional | 2015 | Albany | 48 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18244** | 1.63 | -0.674587 | -0.467366 | -0.919470 | -1.238310 | -0.178839 | -0.410859 | 0.0 | organic | 2018 | WestTexNewMexico | 5 |
| **18245** | 1.71 | -0.767550 | -0.619883 | -0.665372 | -1.238310 | -0.325977 | -0.481465 | 0.0 | organic | 2018 | WestTexNewMexico | 4 |
| **18246** | 1.87 | -0.771505 | -0.619831 | -0.772496 | 0.401546 | -0.308728 | -0.943486 | 0.0 | organic | 2018 | WestTexNewMexico | 3 |
| **18247** | 1.93 | -0.698077 | -0.550282 | -0.710543 | 0.401246 | -0.248932 | -0.908510 | 0.0 | organic | 2018 | WestTexNewMexico | 2 |
| **18248** | 1.62 | -0.663809 | -0.367908 | -0.785173 | 0.122919 | -0.212590 | -1.042250 | 0.0 | organic | 2018 | WestTexNewMexico | 1 |

17725 rows × 13 columns

5. Using Label encoder for encoding region column.
   Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. Label Encoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels.

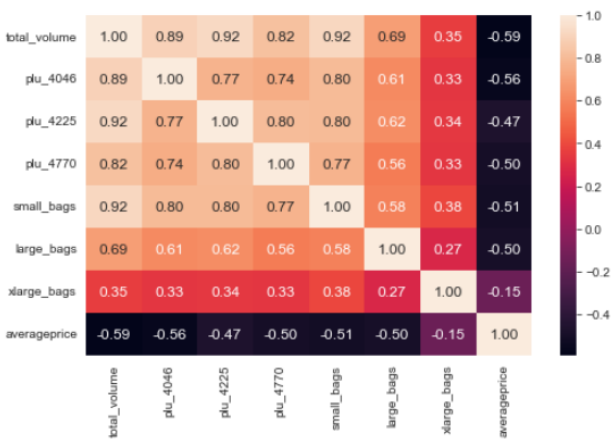
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data6['region']=le.fit_transform(data6['region'])
data7 = pd.get_dummies(data6, drop_first = True)
data7
```

| | averageprice | total_volume | plu_4046 | plu_4225 | plu_4770 | small_bags | large_bags | xlarge_bags | region | type_organic | year_2016 | year_2017 | year_2018 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.33 | -0.086605 | -0.658619 | 0.307313 | -0.247250 | -0.340627 | -0.773351 | 0.0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 1.35 | -0.155746 | -0.776870 | 0.231647 | -0.201041 | -0.306425 | -0.763435 | 0.0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0.93 | 0.179327 | -0.731947 | 0.579503 | -0.007072 | -0.366303 | -0.750820 | 0.0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 1.08 | 0.003876 | -0.634200 | 0.415100 | -0.148322 | -0.496853 | -0.691865 | 0.0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 1.28 | -0.187640 | -0.685293 | 0.224806 | -0.137918 | -0.477201 | -0.600975 | 0.0 | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18244** | 1.63 | -0.674587 | -0.467366 | -0.919470 | -1.238310 | -0.178839 | -0.410859 | 0.0 | 52 | 1 | 0 | 0 | 1 |
| **18245** | 1.71 | -0.767550 | -0.619883 | -0.665372 | -1.238310 | -0.325977 | -0.481465 | 0.0 | 52 | 1 | 0 | 0 | 1 |
| **18246** | 1.87 | -0.771505 | -0.619831 | -0.772496 | 0.401546 | -0.308728 | -0.943486 | 0.0 | 52 | 1 | 0 | 0 | 1 |
| **18247** | 1.93 | -0.698077 | -0.550282 | -0.710543 | 0.401246 | -0.248932 | -0.908510 | 0.0 | 52 | 1 | 0 | 0 | 1 |
| **18248** | 1.62 | -0.663809 | -0.367908 | -0.785173 | 0.122919 | -0.212590 | -1.042250 | 0.0 | 52 | 1 | 0 | 0 | 1 |

17725 rows × 76 columns

6. We will check for correlation & Multicollinearity using .corr() method & VIF resply.

```
x_check = data7[['total_volume','plu_4046','plu_4225','plu_4770','small_bags','large_bags','xlarge_bags','averageprice']]
x_check.corr()
plt.figure(figsize=(8,5))
sns.heatmap(x_check.corr(),annot=True, fmt = ".2f", annot_kws={'size':12});
plt.show()
```




**<u>Observation:</u>**

- We see there are some features with highly correlated with each other, We will confirm the same using Statsmodels' OLS result as follows.

```
model = sm.OLS(x_check['averageprice'], sm.add_constant(x_check.drop('averageprice', axis=1))).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           averageprice   R-squared:                       0.438
Model:                            OLS   Adj. R-squared:                  0.438
Method:                 Least Squares   F-statistic:                     1973.
Date:                Wed, 05 Oct 2022   Prob (F-statistic):               0.00
Time:                        23:29:06   Log-Likelihood:                -3935.4
No. Observations:               17725   AIC:                             7887.
Df Residuals:                   17717   BIC:                             7949.
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const            1.3947      0.002    592.150      0.000       1.390       1.399
total_volume    -0.6076      0.014    -44.473      0.000      -0.634      -0.581
plu_4046         0.0260      0.006      4.656      0.000       0.015       0.037
plu_4225         0.3082      0.007     42.910      0.000       0.294       0.322
plu_4770        -0.0721      0.004    -17.143      0.000      -0.080      -0.064
small_bags       0.1650      0.007     23.515      0.000       0.151       0.179
large_bags      -0.0506      0.003    -15.377      0.000      -0.057      -0.044
xlarge_bags   1.907e-06   3.03e-07      6.303      0.000    1.31e-06      2.5e-06
==============================================================================
Omnibus:                     1712.926   Durbin-Watson:                   0.253
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2638.681
Skew:                           0.728   Prob(JB):                         0.00
Kurtosis:                       4.206   Cond. No.                     6.00e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large,  6e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

**<u>Observation:</u>**

- Condition number is more than 10, this indicate there is multicollinearity issue.

7. We will remove multicollinearity using Variance inflation factor

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
df_feat = x_check.drop('averageprice', axis=1)
df_tgt = x_check['averageprice']

vif = pd.DataFrame()
vif["variables"] = df_feat.columns
vif["VIF"] = [variance_inflation_factor(df_feat.values, i) for i in range(df_feat.shape[1])]
vif.sort_values(by = 'VIF', ascending = False)
```

| | variables | VIF |
|---|---|---|
| 0 | total_volume | 34.198010 |
| 2 | plu_4225 | 9.428277 |
| 4 | small_bags | 8.120107 |
| 1 | plu_4046 | 5.774840 |
| 3 | plu_4770 | 3.349894 |
| 5 | large_bags | 1.986215 |
| 6 | xlarge_bags | 1.184613 |

**<u>Observation:</u>**

- We will remove total volume as it shows VIF more than 10.

```python
df_feat = x_check.drop(['averageprice', 'total_volume'], axis=1)
df_tgt = x_check['averageprice']

vif = pd.DataFrame()
vif["variables"] = df_feat.columns
vif["VIF"] = [variance_inflation_factor(df_feat.values, i) for i in range(df_feat.shape[1])]
vif.sort_values(by = 'VIF', ascending = False)
```

| | variables | VIF |
|---|---|---|
| **1** | plu_4225 | 4.020856 |
| **3** | small_bags | 3.862361 |
| **0** | plu_4046 | 3.432350 |
| **2** | plu_4770 | 3.349892 |
| **4** | large_bags | 1.763164 |
| **5** | xlarge_bags | 1.164698 |

### Observation

1. Multicollinearity is removed.. VIF for all the features is within 10.

8. Final data preparation for model training.
   We will separate dependant & independent features.

## Preparing data for ML model

```python
x = data7.drop(columns = ['region', 'averageprice', 'total_volume'])
y_class = data7['region']
y_reg = data7['averageprice']
```

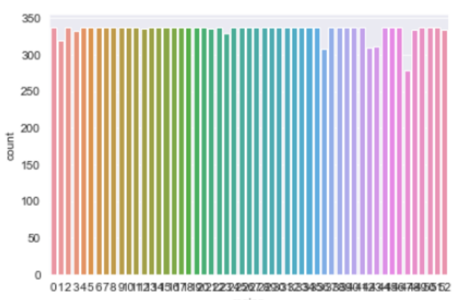
```python
x
```

| | plu_4046 | plu_4225 | plu_4770 | small_bags | large_bags | xlarge_bags | type_organic | year_2016 | year_2017 | year_2018 | week_number_10 | week_number_1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.658619 | 0.307313 | -0.247250 | -0.340627 | -0.773351 | 0.0 | 0 | 0 | 0 | 0 | 0 | ( |
| **1** | -0.776870 | 0.231647 | -0.201041 | -0.306425 | -0.763435 | 0.0 | 0 | 0 | 0 | 0 | 0 | ( |
| **2** | -0.731947 | 0.579503 | -0.007072 | -0.366303 | -0.750820 | 0.0 | 0 | 0 | 0 | 0 | 0 | ( |
| **3** | -0.634200 | 0.415100 | -0.148322 | -0.496853 | -0.691865 | 0.0 | 0 | 0 | 0 | 0 | 0 | ( |
| **4** | -0.685293 | 0.224806 | -0.137918 | -0.477201 | -0.600975 | 0.0 | 0 | 0 | 0 | 0 | 0 | ( |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **18244** | -0.467366 | -0.919470 | -1.238310 | -0.178839 | -0.410859 | 0.0 | 1 | 0 | 0 | 1 | 0 | ( |
| **18245** | -0.619883 | -0.665372 | -1.238310 | -0.325977 | -0.481465 | 0.0 | 1 | 0 | 0 | 1 | 0 | ( |
| **18246** | -0.619831 | -0.772496 | 0.401546 | -0.308728 | -0.943486 | 0.0 | 1 | 0 | 0 | 1 | 0 | ( |
| **18247** | -0.550282 | -0.710543 | 0.401246 | -0.248932 | -0.908510 | 0.0 | 1 | 0 | 0 | 1 | 0 | ( |
| **18248** | -0.367908 | -0.785173 | 0.122919 | -0.212590 | -1.042250 | 0.0 | 1 | 0 | 0 | 1 | 0 | ( |

17725 rows × 73 columns

9. Checking data imbalence for classification problem

```python
sns.countplot(x=y_class)
plt.show()
```




No need to do use SMOTE method as the data is perfectly balenced.

## 10. Applying Standard Scaling to Independent features

### using Standard Scaler to standardize the data

```
#Transforming data into Standard Norma Distribution
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X = pd.DataFrame(sc.fit_transform(x), columns=x.columns)
X
```

| | plu_4046 | plu_4225 | plu_4770 | small_bags | large_bags | xlarge_bags | type_organic | year_2016 | year_2017 | year_2018 | week_number_10 | week_number_1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.647482 | 0.347429 | -0.234376 | -0.358974 | -0.764661 | -0.241634 | -0.989393 | -0.668535 | -0.680842 | -0.278049 | -0.1556 | -0.15540 |
| **1** | -0.768411 | 0.269506 | -0.187556 | -0.321898 | -0.754458 | -0.241634 | -0.989393 | -0.668535 | -0.680842 | -0.278049 | -0.1556 | -0.15540 |
| **2** | -0.722471 | 0.627737 | 0.008975 | -0.386808 | -0.741477 | -0.241634 | -0.989393 | -0.668535 | -0.680842 | -0.278049 | -0.1556 | -0.15540 |
| **3** | -0.622510 | 0.458430 | -0.134141 | -0.528330 | -0.680813 | -0.241634 | -0.989393 | -0.668535 | -0.680842 | -0.278049 | -0.1556 | -0.15540 |
| **4** | -0.674761 | 0.262461 | -0.123600 | -0.507026 | -0.587286 | -0.241634 | -0.989393 | -0.668535 | -0.680842 | -0.278049 | -0.1556 | -0.15540 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **17720** | -0.451898 | -0.915943 | -1.238529 | -0.183588 | -0.391655 | -0.241634 | 1.010720 | -0.668535 | -0.680842 | 3.596491 | -0.1556 | -0.15540 |
| **17721** | -0.607869 | -0.654267 | -1.238529 | -0.343093 | -0.464309 | -0.241634 | 1.010720 | -0.668535 | -0.680842 | 3.596491 | -0.1556 | -0.15540 |
| **17722** | -0.607816 | -0.764586 | 0.422992 | -0.324393 | -0.939732 | -0.241634 | 1.010720 | -0.668535 | -0.680842 | 3.596491 | -0.1556 | -0.15540 |
| **17723** | -0.536692 | -0.700785 | 0.422687 | -0.259572 | -0.903741 | -0.241634 | 1.010720 | -0.668535 | -0.680842 | 3.596491 | -0.1556 | -0.15540 |
| **17724** | -0.350187 | -0.777641 | 0.140683 | -0.220177 | -1.041360 | -0.241634 | 1.010720 | -0.668535 | -0.680842 | 3.596491 | -0.1556 | -0.15540 |

17725 rows × 73 columns

# Building Machine Learning Models.

Please refer the notebook in the github link for detailed codes:

For Classification model;

We will test the following models (the detailed code has all algorithm run in a loop to get best random state & best cross validation fold). The code will generate a table consisting best random state & Cross validation fold & all related evaluation metrics of the respective model.

```
dtc = DecisionTreeClassifier()
etc = ExtraTreesClassifier()
knc = KNeighborsClassifier()
lgr = LogisticRegression(multi_class='ovr')
rfc = RandomForestClassifier()
```

```
# TO print classification report, confusion matrx, roc-auc curve
def print_score(clf,x_train, x_test, y_train, y_test, train=True):
    if train:
        y_pred = clf.predict(x_train)
        print(f"Accuracy score for Train : {accuracy_score(y_train,y_pred) * 100:.2f}%")
        return round(accuracy_score(y_train,y_pred)*100,2)
#         print('\n \n Train Classification report \n', classification_report(y_train,y_pred, digits=2))

    elif train==False:
        y_pred = clf.predict(x_test)
        acc = round(accuracy_score(y_test,y_pred)*100,2)
        print(f"Accuracy score for Test : {accuracy_score(y_test,y_pred) * 100:.2f}%")
        confusion_matrix_c(y_test, y_pred)
        print('\n \n Test Classification report \n', classification_report(y_test, y_pred, digits=2))

        diff = []
        best_cv = []
        for j in range(3, 12):
            cv_score = round(cross_val_score(clf, X, y_class, cv=j, scoring="accuracy").mean()*100,2)
            diff.append(abs(cv_score - acc))
            best_cv.append(j)
        k_f = best_cv[diff.index(min(diff))]
        cv_score = cross_val_score(clf,X, y_class, cv=k_f, scoring="accuracy").mean()
        print(f"Cross Validation score at best cv={k_f} is : {cv_score*100:.2f}%")
        return acc,k_f, min(diff),cv_score*100
```

```
models = [dtc,etc,knc,lgr,rfc]
models_name = ['DecisionTreeClassifier','ExtraTreesClassifier','KNeighborsClassifier',
               'LogisticRegression','RandomForestClassifier']

dummy_count = 0 #dummy variable for count purpose
for model in models:
    ### splitting with best random state
    x_train, x_test, y_train, y_test = train_test_split(X, y_class, random_state=best_i, test_size=.2, stratify=y_class)
    ### training the model
    model.fit(x_train, y_train)
    algo.append(models_name[dummy_count])
    tr_ac = print_score(model, x_train, x_test, y_train, y_test, train=True)
```

| | algo | best random state | train_accuracy | test_accuracy | Score_diff | best cv fold | cross_val_score |
|---|---|---|---|---|---|---|---|
| **3** | LogisticRegression | 78 | 40.97 | 30.69 | 0.20 | 9 | 30.894866 |
| **2** | KNeighborsClassifier | 60 | 22.03 | 0.20 | 9.54 | 3 | 9.737851 |
| **0** | DecisionTreeClassifier | 49 | 100.00 | 79.10 | 15.10 | 11 | 63.773091 |
| **4** | RandomForestClassifier | 82 | 100.00 | 85.42 | 15.72 | 11 | 69.453270 |
| **1** | ExtraTreesClassifier | 123 | 100.00 | 77.04 | 15.91 | 11 | 60.838086 |

**Observation:**

- we will select Random Forest classifier as it has higher test accuracy & lower difference between test accuracy & Cross value score amongst all models.
- It also has higher test accuracy of all the models tested.

Hyper parameter tuning for classification model;

```
param_grid_rfr = {'n_estimators':[200,300,350,450],
                  'max_depth':[50, 65, 75, 80],
                  'min_samples_split':[2, 3, 5],
                  'criterion':['gini', 'entropy']
                 }
```

```
rfc_tune = RandomForestClassifier()
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y_class, test_size = 0.2, random_state = 82, stratify=y_class)
```

```
rf_grid = GridSearchCV(estimator = rfc_tune,
                       param_grid = param_grid_rfr,
                       verbose = 2,
                       scoring = 'accuracy')
```

```
rf_grid.fit(x_train, y_train)
```

...

```
rf_grid.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 80,
 'min_samples_split': 3,
 'n_estimators': 450}
```

```
rf_grid.best_score_
```

```
0.8555007052186177
```

### Using hyper parameters obtained from gridsearch for classificaion model...

```
x_train, x_test_class, y_train, y_test_class = train_test_split(X, y_class,test_size = 0.2,
                                                                random_state = 82, stratify=y_class)
```

```
rfc_tune_final = RandomForestClassifier(criterion='entropy',max_depth= 80,min_samples_split= 3,n_estimators= 450)
```

```
rfc_tune_final.fit(x_train,y_train)
y_pred=rfc_tune_final.predict(x_test_class)
print('Accuracy Score: ', accuracy_score(y_test_class, y_pred))
```

```
Accuracy Score:  0.8750352609308886
```

```
confusion_matrix_c(y_test_class, y_pred)
print('\n \n Test Classification report \n', classification_report(y_test_class, y_pred, digits=2))

cv_score = cross_val_score(rfc_tune_final,X, y_class, cv=11, scoring="accuracy").mean()
print(f"Cross Validation score at best cv = 11 is : {cv_score*100:.2f}%")
```

```
====================
Confusion Matrix :

[[63  0  0 ...  0  0  0]
 [ 0 50  0 ...  0  0  0]
 [ 0  0 65 ...  0  0  0]
 ...
 [ 0  0  0 ... 23  0  0]
 [ 0  0  0 ...  0 68  0]
 [ 0  0  0 ...  0  0 52]]
====================

    accuracy                           0.88      3545
   macro avg       0.88      0.87      0.87      3545
weighted avg       0.88      0.88      0.87      3545

Cross Validation score at best cv = 11 is : 73.17%
```

### observation:

After tuning we see increase in test accuracy & cross value scores. Hence our model is good.

## Saving & predicting classification model

```
filename='avocado_class.pkl'
pickle.dump(rfc_tune_final,open(filename,'wb'))

filename='avocado_reg.pkl'
pickle.dump(xgb_tune_final,open(filename,'wb'))
```

```
model =pickle.load(open('avocado_class.pkl','rb'))
pred =model.predict(x_test_class)
result = pd.DataFrame(list(zip(y_test_class, pred)), columns = ['Actual', 'Predicted'])
result
```

| | Actual | Predicted |
|---|---|---|
| **0** | 35 | 35 |
| **1** | 12 | 12 |
| **2** | 2 | 2 |
| **3** | 8 | 8 |
| **4** | 27 | 27 |
| **...** | ... | ... |
| **3540** | 27 | 27 |
| **3541** | 3 | 3 |
| **3542** | 33 | 33 |
| **3543** | 23 | 3 |
| **3544** | 47 | 47 |

3545 rows × 2 columns

For Regression model;

We will test the following models (the detailed code has all algorithm run in a loop to get best random state & best cross validation fold). The code will generate a table consisting best random state & Cross validation fold & all related evaluation metrics of the respective model.

```
lr = LinearRegression()
ls = Lasso()
rd = Ridge()
rfr = RandomForestRegressor()
abr = AdaBoostRegressor()
gbr = GradientBoostingRegressor()
dtr = DecisionTreeRegressor()
knr = KNeighborsRegressor()
xgb = XGBRegressor()
```

```
models = [lr, ls, rd, abr, gbr, dtr, knr, xgb, rfr]
models_name = ['Linear Regression', 'Lasso', 'Ridge',
               'Ada-Boost Regressor', 'Gradient Boosting Regressor',
               'Decision Tree Regressor', 'KNeighbors Regressor',
               'XGB Regressor', 'Random Forest Regressor']
dummy_count = 0 #dummy variable for count purpose
for model in models:
```

```
    ### splitting the train7 set with best random state
    x_train, x_test, y_train, y_test = train_test_split(X, y_reg, random_state=best_i, test_size=.25)
```

```
    ### training the model
    model.fit(x_train, y_train)
    algo.append(models_name[dummy_count])
```

```
    ##showing the results in output
    print('::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::')
    print(' ')
    print(' ')
    print(models_name[dummy_count] + ' Model')
    print(' ')
    print('for '+ models_name[dummy_count] + ' model, Best Random_state number for splitting the data is: ', best_i)
    print(' ')
    print('===scores for training set===')
    print('r2 score for training set', r2_score(y_train, pred_train))
    print('MAE for training set: ', mean_absolute_error(y_train, pred_train))
    print('MSE for training set: ', mean_squared_error(y_train, pred_train))
    print('SMSE for training set: ', np.sqrt(mean_squared_error(y_train, pred_train)))
    print(' ')
    print('===scores for testing set===')
    print('r2 score for testing set : ', r2_score(y_test, pred_test))
    print('MAE for testing set: ', mean_absolute_error(y_test, pred_test))
    print('MSE for testing set: ', mean_squared_error(y_test, pred_test))
    print('SMSE for testing set: ', np.sqrt(mean_squared_error(y_test, pred_test)))
    print(' ')
    print(' ')
```

```python
    ### printing CV Score based on best CV fold value
    k_f = best_cv[cv_diff.index(min(cv_diff))]
    cv_fold.append(k_f)
    cv_score = cross_val_score(model, X, y_reg, cv=k_f).mean()
    crossvalscore.append(round(cv_score,2))
    score_diff.append(abs(round(cv_score - test_accuracy,2)))
    print(f"Cross Validation score at best cv={k_f} is : {cv_score*100:.2f}%")
    dummy_count+=1
    print(' ')
    print(' ')
```

| Sr. No. | algo | best random state | train_r2score | test_r2score | train_mae | train_mse | train_srmse | test_mae | test_mse | test_srmse | best cv fold | cross_val_score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | Random Forest Regressor | 92 | 0.98 | 0.86 | 0.04 | 0.00 | 0.06 | 0.11 | 0.02 | 0.15 | 11 | 0.39 |
| 8 | XGB Regressor | 63 | 0.93 | 0.85 | 0.08 | 0.01 | 0.10 | 0.12 | 0.03 | 0.16 | 12 | 0.51 |
| 6 | Decision Tree Regressor | 18 | 1.00 | 0.71 | 0.00 | 0.00 | 0.00 | 0.14 | 0.05 | 0.22 | 11 | -0.17 |
| 5 | Gradient Boosting Regressor | 76 | 0.71 | 0.71 | 0.16 | 0.05 | 0.22 | 0.16 | 0.05 | 0.21 | 9 | 0.38 |
| 7 | KNeighbors Regressor | 1 | 0.76 | 0.65 | 0.15 | 0.04 | 0.20 | 0.18 | 0.06 | 0.25 | 12 | 0.33 |
| 1 | Linear Regression | 20 | 0.57 | 0.57 | 0.21 | 0.07 | 0.27 | 0.20 | 0.07 | 0.26 | 9 | 0.15 |
| 3 | Ridge | 20 | 0.57 | 0.57 | 0.21 | 0.07 | 0.27 | 0.20 | 0.07 | 0.26 | 9 | 0.15 |
| 4 | Ada-Boost Regressor | 9 | 0.52 | 0.51 | 0.22 | 0.08 | 0.28 | 0.23 | 0.08 | 0.28 | 10 | -0.16 |
| 2 | Lasso | 23 | 0.00 | -0.00 | 0.32 | 0.16 | 0.40 | 0.33 | 0.16 | 0.40 | 9 | -1.18 |

**Observation:**

- we will select XGB Regressor as it has higher test & Cross validation scores, also it has lower MAE value.
- Random forest has higher test score but cross value score is very less. Hence, we will use XGB Regressor.

Hyper parameter tuning for regression model;

## Using hyper parameters obtained from gridsearch for regression model...

```python
x_train, x_test_reg, y_train, y_test_reg = train_test_split(X, y_reg, random_state=63, test_size=.25)
```

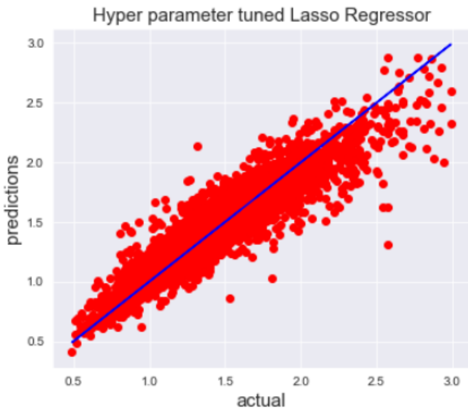
```python
xgb_tune_final = XGBRegressor(learning_rate= 0.1, max_depth=8, n_estimators=800)
```

```python
xgb_tune_final.fit(x_train,y_train)
y_pred=xgb_tune_final.predict(x_test_reg)
```

```python
print('r2 score for testing set : ', r2_score(y_test_reg, y_pred))
print('MAE for testing set: ', mean_absolute_error(y_test_reg, y_pred))
print('MSE for testing set: ', mean_squared_error(y_test_reg, y_pred))
print('SMSE for testing set: ', np.sqrt(mean_squared_error(y_test_reg, y_pred)))
```

```
r2 score for testing set :  0.874351595012883
MAE for testing set:  0.10166876581386539
MSE for testing set:  0.02122898504075049
SMSE for testing set:  0.14570169882589046
```

```python
##plotting the graph with bestfit line, actual & predicted values
plt.figure(figsize = (6,5))
plt.scatter(x =y_test_reg, y=y_pred, color = 'r')
plt.plot(y_test_reg, y_test_reg, color = 'b')
plt.xlabel('actual', fontsize = 15)
plt.ylabel('predictions', fontsize = 15)
plt.title('Hyper parameter tuned Lasso Regressor', fontsize = 15)
plt.show()
```




```
cross_val_score(xgb_tune_final, X, y_reg, cv = 12, scoring ='r2').mean()
```

```
0.5139095135730846
```

### observation:

After tuning we see increase in test score & cross value scores. Hence our model is good.

## Saving & predicting regression model

```
model =pickle.load(open('avocado_reg.pkl','rb'))
pred =model.predict(x_test_reg)
result = pd.DataFrame(list(zip(y_test_reg, pred)), columns = ['Actual', 'Predicted'])
result
```

| | Actual | Predicted |
|---|---|---|
| **0** | 1.60 | 1.655651 |
| **1** | 1.05 | 1.003266 |
| **2** | 1.78 | 1.507349 |
| **3** | 1.61 | 1.596217 |
| **4** | 1.60 | 1.697207 |
| **...** | ... | ... |
| **4427** | 1.26 | 1.178945 |
| **4428** | 1.26 | 1.179238 |
| **4429** | 1.55 | 1.754191 |
| **4430** | 1.17 | 1.011211 |
| **4431** | 1.70 | 1.568804 |

4432 rows × 2 columns