Pandas 101: One-stop Shop for Data Science

This notebook can be treated as pandas cheatsheet or a beginner-friendly guide to learn from basics.

In [2]:
```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
```

Creating DataFrames

- From a list of dictionaries (constructed row by row)

In [4]:
```
list_of_dicts = [
     {"name": "Ginger", "breed": "Dachshund", "height_cm": 22,"weight_kg": 10, "
    {"name": "Scout", "breed": "Dalmatian", "height_cm": 59,"weight_kg": 25, "da
]
new_dogs = pd.DataFrame(list_of_dicts)
new_dogs
```

Out[4]:

	name	breed	height_cm	weight_kg	date_of_birth
0	Ginger	Dachshund	22	10	2019-03-14
1	Scout	Dalmatian	59	25	2019-05-09

In [5]:
```
dict_of_lists = {
     "name": ["Ginger", "Scout"],
     "breed": ["Dachshund", "Dalmatian"],
     "height_cm": [22, 59],
     "weight_kg": [10, 25],
     "date_of_birth": ["2019-03-14","2019-05-09"]  }
new_dogs = pd.DataFrame(dict_of_lists)
new_dogs
```

Out[5]:

	name	breed	height_cm	weight_kg	date_of_birth
0	Ginger	Dachshund	22	10	2019-03-14
1	Scout	Dalmatian	59	25	2019-05-09

Reading and writing CSVs

- CSV = comma-separated values
- Designed for DataFrame-like data
- Most database and spreadsheet programs can use them or create them

In [7]:
```
# read CSV from using pandas
avocado = pd.read_csv(r"D:\NIT Daily Task\Oct\4th- REGRESSION PROJECT\4th- REGRE
# print the first few rows of the dataframe
avocado.head()
```

Out[7]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

Read CSV and assign index

You can assign columns as index using "index_col" attribute.

Since I want to index Date there is another helpful function called "parse_date" which will parse the date in the rows such that we can perform more complex subsetting(eg monthly, weekly etc).

In [9]:
```
avocado.head()
```

Out[9]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

Remove index from dataframe .reset_index(drop)

To reset the index use this function

In [11]:
```
avocado = avocado.reset_index(drop=True)
avocado.head()
```

Out[11]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

In [12]:
```
avocado.to_csv("test_write.csv")
```

Some useful pandas function

- **.head()** or **.head(x)** is used to get the first x rows of the DataFrame (x = 5 by default)

In [14]:
```
avocado = pd.read_csv(r"D:\NIT Daily Task\Oct\4th- REGRESSION PROJECT\4th- REGRE
avocado.head()
```

Out[14]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986

```
In [15]: avocado.tail(10)
```

Out[15]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	
18239	2	2018-03-11	1.56	22128.42	2162.67	3194.25	8.93	16762.57	1
18240	3	2018-03-04	1.54	17393.30	1832.24	1905.57	0.00	13655.49	1
18241	4	2018-02-25	1.57	18421.24	1974.26	2482.65	0.00	13964.33	1
18242	5	2018-02-18	1.56	17597.12	1892.05	1928.36	0.00	13776.71	1
18243	6	2018-02-11	1.57	15986.17	1924.28	1368.32	0.00	12693.57	1
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67	1
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84	
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11	
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54	1
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15	1

```
In [16]: avocado.info()
```

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

In [17]:
```
print(avocado.shape)
```

```
(18249, 14)
```

In [18]:
```
avocado.describe()
```

Out[18]:

	Unnamed: 0	AveragePrice	Total Volume	4046	4225	
count	18249.000000	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900
mean	24.232232	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974
std	15.481045	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641
min	0.000000	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000
25%	10.000000	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000
50%	24.000000	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900
75%	38.000000	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420
max	52.000000	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439

In [19]:
```
avocado.values
```

Out[19]:
```
array([[0, '2015-12-27', 1.33, ..., 'conventional', 2015, 'Albany'],
       [1, '2015-12-20', 1.35, ..., 'conventional', 2015, 'Albany'],
       [2, '2015-12-13', 0.93, ..., 'conventional', 2015, 'Albany'],
       ...,
       [9, '2018-01-21', 1.87, ..., 'organic', 2018, 'WestTexNewMexico'],
       [10, '2018-01-14', 1.93, ..., 'organic', 2018, 'WestTexNewMexico'],
       [11, '2018-01-07', 1.62, ..., 'organic', 2018, 'WestTexNewMexico']],
      dtype=object)
```

In [20]:
```
print(avocado.columns)
```

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

Appending & Concatenating Series

append(): Series & DataFrame method

- Invocation:
- s1.append(s2)
- Stacks rows of s2 below s1

concat(): pandas module function

* Invocation: * pd.concat([s1, s2, s3]) * Can stack row-wise or column-wise

In [22]:
```
even = pd.Series([2, 4, 6, 8, 10])
odd = pd.Series([1, 3, 5, 7, 9])

# Use pd.concat instead of append
res = pd.concat([even, odd])
print(res)
```

```
0     2
1     4
2     6
3     8
4    10
0     1
1     3
2     5
3     7
4     9
dtype: int64
```

Sorting

syntax:

> DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

- by: Single/List of column names to sort Data Frame by.
- axis: 0 or 'index' for rows and 1 or 'columns' for Column.
- ascending: Boolean value which sorts Data frame in ascending order if True.
- inplace: Boolean value. Makes the changes in passed data frame itself if True.
- kind: String which can have three inputs('quicksort', 'mergesort' or 'heapsort') of algorithm used to sort data frame.

- na_position: Takes two string input 'last' or 'first' to set position of Null values. Default is 'last'.

In [24]:
```python
# sort values based on "AveragePrice" (ascending) and "year" (descending)
avocado.sort_values(["AveragePrice", "year"], ascending=[True, False])
```

Out[24]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770
15261	43	2017-03-05	0.44	64057.04	223.84	4748.88	0.00
7412	47	2017-02-05	0.46	2200550.27	1200632.86	531226.65	18324.93
15473	43	2017-03-05	0.48	50890.73	717.57	4138.84	0.00
15262	44	2017-02-26	0.49	44024.03	252.79	4472.68	0.00
1716	0	2015-12-27	0.49	1137707.43	738314.80	286858.37	11642.46
...	...	...	...	...	...	...	...
16720	18	2017-08-27	3.04	12656.32	419.06	4851.90	145.09
16055	42	2017-03-12	3.05	2068.26	1043.83	77.36	0.00
14124	7	2016-11-06	3.12	19043.80	5898.49	10039.34	0.00
17428	37	2017-04-16	3.17	3018.56	1255.55	82.31	0.00
14125	8	2016-10-30	3.25	16700.94	2325.93	11142.85	0.00

18249 rows × 14 columns

Subsetting

Subsetting is used to get a slice of the original dataframe

In [26]:
```python
avocado["AveragePrice"]
```

Out[26]:
```
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
Name: AveragePrice, Length: 18249, dtype: float64
```

In [27]:
```
# Subsetting multiple columns
avocado[["AveragePrice","Date"]]
```

Out[27]:

	AveragePrice	Date
0	1.33	2015-12-27
1	1.35	2015-12-20
2	0.93	2015-12-13
3	1.08	2015-12-06
4	1.28	2015-11-29
...	...	...
18244	1.63	2018-02-04
18245	1.71	2018-01-28
18246	1.87	2018-01-21
18247	1.93	2018-01-14
18248	1.62	2018-01-07

18249 rows × 2 columns

Subsetting rows

In [29]:
```
# Subsetting rows
avocado["AveragePrice"]<1
```

Out[29]:
```
0        False
1        False
2         True
3        False
4        False
         ...
18244    False
18245    False
18246    False
18247    False
18248    False
Name: AveragePrice, Length: 18249, dtype: bool
```

In [30]:
```
# This will print only the rows with price < 1
avocado[avocado["AveragePrice"]<1]
```

Out[30]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	To Ba
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.
6	6	2015-11-15	0.99	83453.76	1368.92	73672.72	93.26	8318.
7	7	2015-11-08	0.98	109428.33	703.75	101815.36	80.00	6829.
13	13	2015-09-27	0.99	106803.39	1204.88	99409.21	154.84	6034.
43	43	2015-03-01	0.99	55595.74	629.46	45633.34	181.49	9151.
...	...	...	...	...	...	...	...	
17169	43	2017-03-05	0.99	155011.12	35367.23	5175.81	5.91	114462.
17170	44	2017-02-26	0.99	171145.00	34520.03	6936.39	0.00	129688.
17536	39	2017-04-02	0.98	402676.23	34093.33	58330.53	207.85	310044.
17537	40	2017-03-26	0.90	456645.91	36169.35	51398.72	139.55	368938.
17540	43	2017-03-05	0.99	367519.17	61166.48	55123.99	126.80	251101.

2796 rows × 14 columns

Subsetting based on text data

In [32]:
```
# it will print all the rows with "type" = "organic"
avocado[avocado["type"]=="organic"]
```

Out[32]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags
9126	0	2015-12-27	1.83	989.55	8.16	88.59	0.00	892.80
9127	1	2015-12-20	1.89	1163.03	30.24	172.14	0.00	960.65
9128	2	2015-12-13	1.85	995.96	10.44	178.70	0.00	806.82
9129	3	2015-12-06	1.84	1158.42	90.29	104.18	0.00	963.95
9130	4	2015-11-29	1.94	831.69	0.00	94.73	0.00	736.96
...	...	...	...	...	...	...	...	...
18244	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0.00	13498.67
18245	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0.00	9264.84
18246	9	2018-01-21	1.87	13766.76	1191.92	2452.79	727.94	9394.11
18247	10	2018-01-14	1.93	16205.22	1527.63	2981.04	727.01	10969.54
18248	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224.53	12014.15

9123 rows × 14 columns

Subsetting based on dates

In [34]:

```
# it will print all the rows with "Date" <= 2015-02-04
avocado[avocado["Date"]<="2015-02-04"]
```

Out[34]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags
47	47	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18
48	48	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82
49	49	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05
50	50	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77
51	51	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46
...	...	...	...	...	...	...	...	...
11928	46	2015-02-01	1.77	7210.19	1634.42	3012.44	0.00	2563.33
11929	47	2015-01-25	1.63	7324.06	1934.46	3032.72	0.00	2356.88
11930	48	2015-01-18	1.71	5508.20	1793.64	2078.72	0.00	1635.84
11931	49	2015-01-11	1.69	6861.73	1822.28	2377.54	0.00	2661.91
11932	50	2015-01-04	1.64	6182.81	1561.30	2958.17	0.00	1663.34

540 rows × 14 columns

Subsetting based on multiple conditions

You can use the logical operators to define a complex condition

- "&" and
- "|" or
- "~" not

> ** SEPERATE EACH CONDITION WITH PARENTHESES TO AVOID ERRORS**

In [36]:

```
# it will print all the rows with "Date" before 2015-02-04 and "type" == "organi
avocado[(avocado["Date"]<"2015-02-04") & (avocado["type"]=="organic")]
```

Out[36]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sm Ba
9173	47	2015-02-01	1.83	1228.51	33.12	99.36	0.0	1096.03	1096
9174	48	2015-01-25	1.89	1115.89	14.87	148.72	0.0	952.30	952
9175	49	2015-01-18	1.93	1118.47	8.02	178.78	0.0	931.67	931
9176	50	2015-01-11	1.77	1182.56	39.00	305.12	0.0	838.44	838
9177	51	2015-01-04	1.79	1373.95	57.42	153.88	0.0	1162.65	1162
...	...	...	...	...	...	...	...	...	
11928	46	2015-02-01	1.77	7210.19	1634.42	3012.44	0.0	2563.33	2563
11929	47	2015-01-25	1.63	7324.06	1934.46	3032.72	0.0	2356.88	2320
11930	48	2015-01-18	1.71	5508.20	1793.64	2078.72	0.0	1635.84	1620
11931	49	2015-01-11	1.69	6861.73	1822.28	2377.54	0.0	2661.91	2656
11932	50	2015-01-04	1.64	6182.81	1561.30	2958.17	0.0	1663.34	1663

270 rows × 14 columns

Subsetting using .isin()

isin() method helps in selecting rows with having a particular(or Multiple) value in a particular column

> Syntax: DataFrame.isin(values)
>
> Parameters: values: iterable, Series, List, Tuple, DataFrame or dictionary to check in the caller Series/Data Frame.
>
> Return Type: DataFrame of Boolean of Dimension.

In [38]:
```
# subset the avocado in the region Boston or SanDiego
regionFilter = avocado["region"].isin(["Boston", "SanDiego"])
avocado[regionFilter]
```

Out[38]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	To Ba
208	0	2015-12-27	1.13	450816.39	3886.27	346964.70	13952.56	86012.
209	1	2015-12-20	1.07	489802.88	4912.37	390100.99	5887.72	88901.
210	2	2015-12-13	1.01	549945.76	4641.02	455362.38	219.40	89722.
211	3	2015-12-06	1.02	488679.31	5126.32	407520.22	142.99	75889.
212	4	2015-11-29	1.19	350559.81	3609.25	272719.08	105.86	74125.
...	...	...	...	...	...	...	...	
18100	7	2018-02-04	1.81	17454.74	1158.41	7388.27	0.00	8908.
18101	8	2018-01-28	1.91	17579.47	1145.64	8284.41	0.00	8149.
18102	9	2018-01-21	1.95	18676.37	1088.49	9282.37	0.00	8305.
18103	10	2018-01-14	1.81	21770.02	3285.98	14338.52	0.00	4145.
18104	11	2018-01-07	2.06	16746.82	5150.82	9366.31	0.00	2229.

676 rows × 14 columns

Multiple parameter Filtering

Use logical operators to combine different filters

In [40]:
```
# subset the avocado in the region Boston or SanDiego in the year 2016 or 2017
regionFilter = avocado["region"].isin(["Boston", "SanDiego"])
yearFilter = avocado["year"].isin(["2016", "2017"])
avocado[regionFilter & yearFilter]
```

Out[40]:

Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLa B

Detecting missing values .isna()

.isna() is a method used to find is there exist any NaN values in the DataFrame

It will give a True bool value if a cell has a NaN value

In [42]:
```
avocado.isna()
```

Out[42]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bag
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	..
18244	False	False	False	False	False	False	False	False	False	False
18245	False	False	False	False	False	False	False	False	False	False
18246	False	False	False	False	False	False	False	False	False	False
18247	False	False	False	False	False	False	False	False	False	False
18248	False	False	False	False	False	False	False	False	False	False

18249 rows × 14 columns

We can use .any() function to get a consise info

In [44]:
```
avocado.isna().any()
```

Out[44]:
```
Unnamed: 0      False
Date            False
AveragePrice    False
Total Volume    False
4046            False
4225            False
4770            False
Total Bags      False
Small Bags      False
Large Bags      False
XLarge Bags     False
type            False
year            False
region          False
dtype: bool
```

Counting missing values

In [46]:
```
avocado.isna().sum()
```

Out[46]:
```
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
```

Removing missing values

- Drop NaN ** .dropna() **
- Fill NaN with value x ** .fillna(x) **

In [48]:
```
# Luckily we don't have any NaN but if we have we can use any of the two methods

avocado.dropna()

# ****  OR  ****

meanVal = avocado["AveragePrice"].mean()
avocado.fillna(meanVal)
```

Out[48]:

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

18249 rows × 14 columns

Adding a new column

It can easily be done using the [ ] brackets

Lets add a new column to our dataframe called AveragePricePer100

In [50]:

```
avocado["AveragePricePer100"] = avocado["AveragePrice"] * 100
avocado
```

Out[50]:

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

18249 rows × 15 columns

Deleting columns in DataFrame .drop(lst,axis = 1)

> dataFrame.drop(['COLUMN_NAME'], axis = 1)

- the first parameter is a list of columns to be deleted
- axis = 1 means delete column
- axis = 0 means delete row

In [52]:
```
avocado.drop(["AveragePricePer100"],axis = 1)
```

Out[52]:

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

18249 rows × 14 columns

Summary statistics

Some of the functions availabe in pandas are:

> .median() .mode() .min() .max() .var() .std() .sum() .quantile()

In [54]:
```
# mean of the AveragePrice of avocado
avocado["AveragePrice"].mean()
```

Out[54]:
```
1.405978409775878
```

Summarizing dates

To find the min or max date in a dataframe

In [56]:
```
avocado["Date"].max()
```

Out[56]:
```
'2018-03-25'
```

.agg() method

Pandas Series.agg() is used to pass a function or list of function to be applied on a series or even each element of series separately.

> Syntax: Series.agg(func, axis=0)
>
> Parameters: func: Function, list of function or string of function name to be called on Series. axis:0 or 'index' for row wise operation and 1 or 'columns' for column wise operation.
>
> Return Type: The return type depends on return type of function passed as parameter.

In [58]:
```
def pct30(column):
    #return the 0.3 quartile
    return column.quantile(0.3)
def pct50(column):
    #return the 0.5 quartile
    return column.quantile(0.5)

avocado[["AveragePrice","Total Bags"]].agg([pct30,pct50])
```

Out[58]:

	AveragePrice	Total Bags
pct30	1.15	7316.634
pct50	1.37	39743.830

Dropping duplicate names .drop_duplicates(lst)

Delete all the duplicate names from the dataframe

In [60]:
```
temp = avocado.drop_duplicates(subset=["year"])
temp
```

Out[60]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87
2808	0	2016-12-25	1.52	73341.73	3202.39	58280.33	426.92	11432.09
5616	0	2017-12-31	1.47	113514.42	2622.70	101135.53	20.25	9735.94
8478	0	2018-03-25	1.57	149396.50	16361.69	109045.03	65.45	23924.33

Count categorical data .value_counts()

Pandas Series.value_counts() function return a Series containing counts of unique values.

> Syntax: Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)
>
> Parameter :

normalize : If True then the object returned will contain the relative frequencies of the unique values. sort : Sort by values. ascending : Sort in ascending order. bins : Rather than count values, group them into half-open bins, a convenience for pd.cut, only works with numeric data. dropna : Don’t include counts of NaN.

> Returns : counts : Series

In [62]:
```
# count number of avocado in each year in descending order
avocado["year"].value_counts(sort=True, ascending = False)
```

Out[62]:
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```

Grouped summaries .groupby(col)

This function will group similar categories into one and then we can perform some summary statistics

> Syntax: DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)

> Parameters : by : mapping, function, str, or iterable
>
> axis : int, default 0
>
> level : If the axis is a MultiIndex (hierarchical), group by a particular level or levels
>
> as_index : For aggregated output, return object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively “SQL-style” grouped output
>
> sort : Sort group keys. Get better performance by turning this off. Note this does not influence the order of observations within each group. groupby preserves the order of rows within each group.
>
> group_keys : When calling apply, add group keys to index to identify pieces
>
> squeeze : Reduce the dimensionality of the return type if possible, otherwise return a consistent type
>
> Returns : GroupBy object

In [64]:
```
# group by multiple columns and perform multiple summary statistic operations
avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.median]
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\3377443975.py:2: FutureWarning:
The provided callable <built-in function min> is currently using SeriesGroupBy.mi
n. In a future version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "min" instead.
  avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.media
n])
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\3377443975.py:2: FutureWarning:
The provided callable <built-in function max> is currently using SeriesGroupBy.ma
x. In a future version of pandas, the provided callable will be used directly. To
keep current behavior pass the string "max" instead.
  avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.media
n])
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\3377443975.py:2: FutureWarning:
The provided callable <function mean at 0x000001D5993E6DE0> is currently using Se
riesGroupBy.mean. In a future version of pandas, the provided callable will be us
ed directly. To keep current behavior pass the string "mean" instead.
  avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.media
n])
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\3377443975.py:2: FutureWarning:
The provided callable <function median at 0x000001D599565B20> is currently using
SeriesGroupBy.median. In a future version of pandas, the provided callable will b
e used directly. To keep current behavior pass the string "median" instead.
  avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.media
n])
```

Out[64]:

year	type	min	max	mean	median
2015	**conventional**	0.49	1.59	1.077963	1.08
	organic	0.81	2.79	1.673324	1.67
2016	**conventional**	0.51	2.20	1.105595	1.08
	organic	0.58	3.25	1.571684	1.53
2017	**conventional**	0.46	2.22	1.294888	1.30
	organic	0.44	3.17	1.735521	1.72
2018	**conventional**	0.56	1.74	1.127886	1.14
	organic	1.01	2.30	1.567176	1.55

group by multiple columns and perform multiple summary statistic operations

avocado.groupby(["year","type"])["AveragePrice"].agg([min,max,np.mean,np.median])

In [66]:
```
# this is the same table we build in the previous cell but using pivot table
avocado.pivot_table(index=["year","type"], aggfunc=[min,max,np.mean,np.median],
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\762502195.py:2: FutureWarning:
The provided callable <built-in function min> is currently using DataFrameGroupB
y.min. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "min" instead.
  avocado.pivot_table(index=["year","type"], aggfunc=[min,max,np.mean,np.median],
values="AveragePrice")
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\762502195.py:2: FutureWarning:
The provided callable <built-in function max> is currently using DataFrameGroupB
y.max. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "max" instead.
  avocado.pivot_table(index=["year","type"], aggfunc=[min,max,np.mean,np.median],
values="AveragePrice")
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\762502195.py:2: FutureWarning:
The provided callable <function mean at 0x000001D5993E6DE0> is currently using Da
taFrameGroupBy.mean. In a future version of pandas, the provided callable will be
used directly. To keep current behavior pass the string "mean" instead.
  avocado.pivot_table(index=["year","type"], aggfunc=[min,max,np.mean,np.median],
values="AveragePrice")
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\762502195.py:2: FutureWarning:
The provided callable <function median at 0x000001D599565B20> is currently using
DataFrameGroupBy.median. In a future version of pandas, the provided callable wil
l be used directly. To keep current behavior pass the string "median" instead.
  avocado.pivot_table(index=["year","type"], aggfunc=[min,max,np.mean,np.median],
values="AveragePrice")
```

Out[66]:

		min	max	mean	median
		AveragePrice	AveragePrice	AveragePrice	AveragePrice
year	type				
2015	conventional	0.49	1.59	1.077963	1.08
	organic	0.81	2.79	1.673324	1.67
2016	conventional	0.51	2.20	1.105595	1.08
	organic	0.58	3.25	1.571684	1.53
2017	conventional	0.46	2.22	1.294888	1.30
	organic	0.44	3.17	1.735521	1.72
2018	conventional	0.56	1.74	1.127886	1.14
	organic	1.01	2.30	1.567176	1.55

Explicit indexes

Indexes make subsetting simpler using .loc and .iloc

Setting column as the index

In [69]:
```
regionIndex = avocado.set_index(["region"])
regionIndex
```

Out[69]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4
region							
Albany	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48
Albany	1	2015-12-20	1.35	54876.98	674.28	44638.81	58
Albany	2	2015-12-13	0.93	118220.22	794.70	109149.67	130
Albany	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72
Albany	4	2015-11-29	1.28	51039.60	941.48	43838.39	75
...	...	...	...	...	...	...	
WestTexNewMexico	7	2018-02-04	1.63	17074.83	2046.96	1529.20	(
WestTexNewMexico	8	2018-01-28	1.71	13888.04	1191.70	3431.50	(
WestTexNewMexico	9	2018-01-21	1.87	13766.76	1191.92	2452.79	72
WestTexNewMexico	10	2018-01-14	1.93	16205.22	1527.63	2981.04	72
WestTexNewMexico	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224

18249 rows × 14 columns

In [70]:
```python
# Insted of doing this
avocado[avocado["region"].isin(["Albany", "WestTexNewMexico"])]
```

Out[70]:

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

673 rows × 15 columns

In [71]:
```
# we can simply do
regionIndex.loc[["Albany", "WestTexNewMexico"]]
```

Out[71]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4
region							
Albany	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48
Albany	1	2015-12-20	1.35	54876.98	674.28	44638.81	58
Albany	2	2015-12-13	0.93	118220.22	794.70	109149.67	13
Albany	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72
Albany	4	2015-11-29	1.28	51039.60	941.48	43838.39	75
...	...	...	...	...	...	...	
WestTexNewMexico	7	2018-02-04	1.63	17074.83	2046.96	1529.20	0
WestTexNewMexico	8	2018-01-28	1.71	13888.04	1191.70	3431.50	0
WestTexNewMexico	9	2018-01-21	1.87	13766.76	1191.92	2452.79	72
WestTexNewMexico	10	2018-01-14	1.93	16205.22	1527.63	2981.04	72
WestTexNewMexico	11	2018-01-07	1.62	17489.58	2894.77	2356.13	224

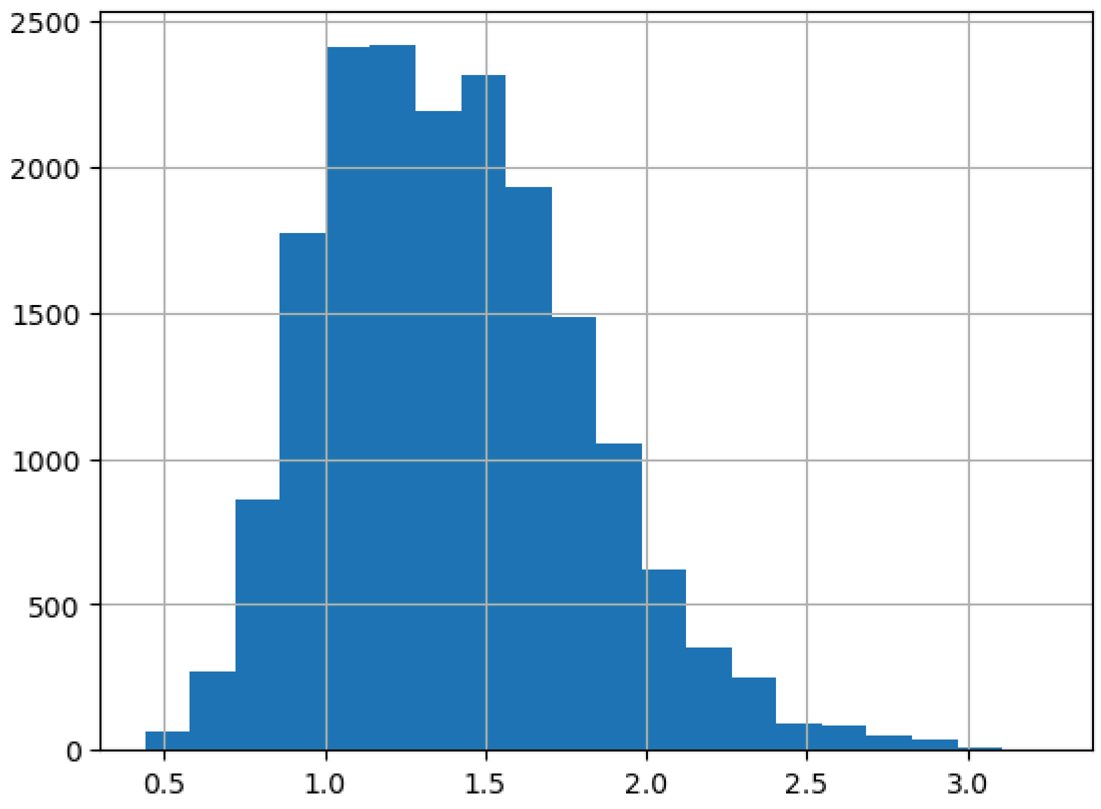
673 rows × 14 columns

Visualizing your data

Histograms

use the function .hist()

In [74]:
```
avocado["AveragePrice"].hist(bins=20)
plt.show()
```



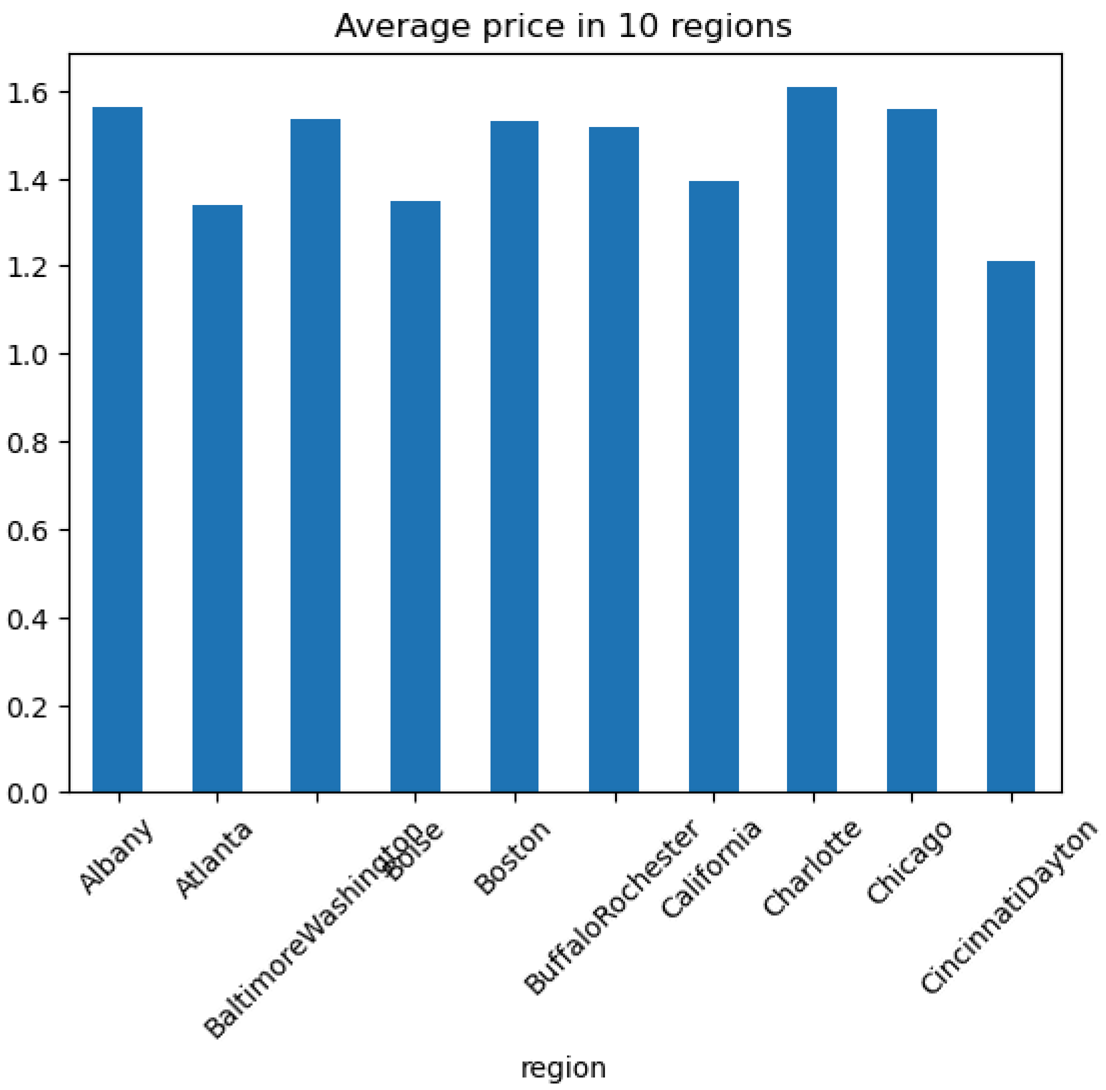
Bar plots

In [76]:
```
regionFilter = avocado.groupby("region")["AveragePrice"].mean().head(10)
regionFilter
```

Out[76]:
```
region
Albany                 1.561036
Atlanta                1.337959
BaltimoreWashington    1.534231
Boise                  1.348136
Boston                 1.530888
BuffaloRochester       1.516834
California             1.395325
Charlotte              1.606036
Chicago                1.556775
CincinnatiDayton       1.209201
Name: AveragePrice, dtype: float64
```

In [77]:
```
regionFilter.plot(kind = "bar",rot=45,title="Average price in 10 regions")
```

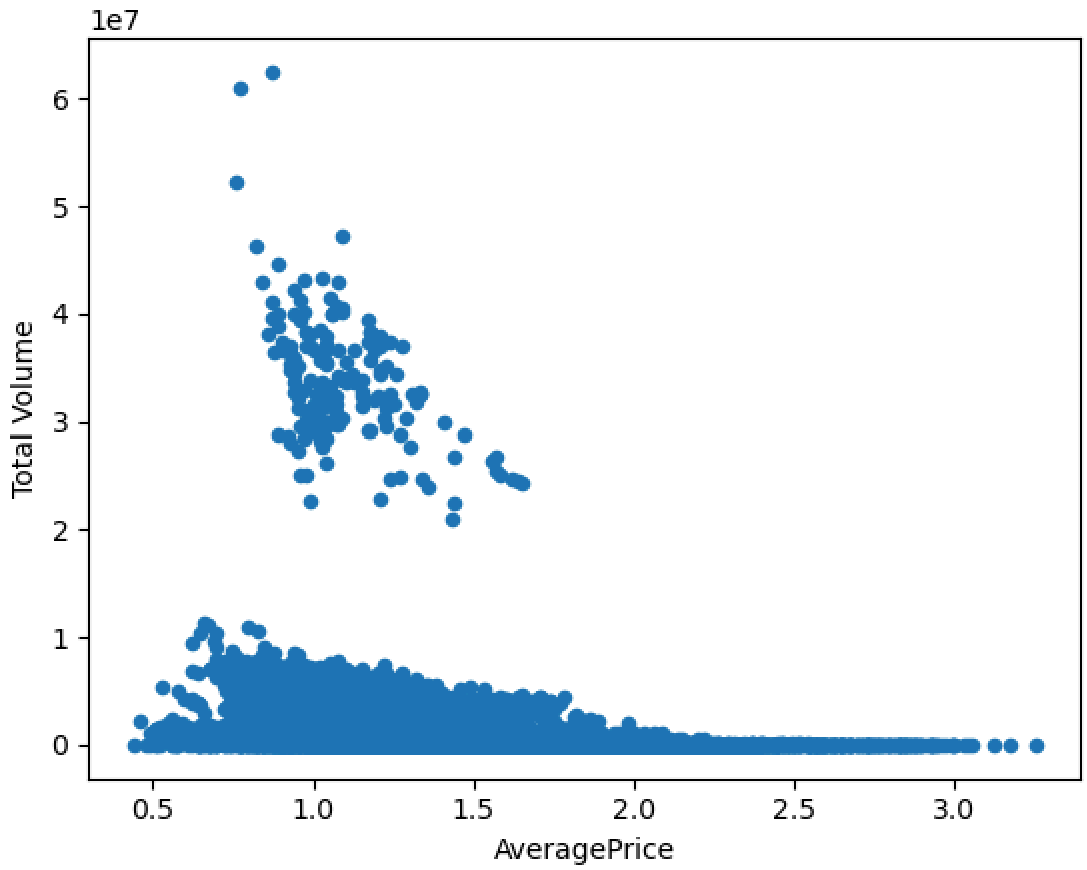
Out[77]:
```
<Axes: title={'center': 'Average price in 10 regions'}, xlabel='region'>
```



Scatter plot

In [79]:
```
avocado.plot(x="AveragePrice", y="Total Volume", kind="scatter")
```

Out[79]:
```
<Axes: xlabel='AveragePrice', ylabel='Total Volume'>
```



Arithmetic with Series & DataFrames

You can use arithmetic operators directly on series but sometimes you need more control while performing these operations, here is where these explicit arithmetic functions come into the picture

Add/Subtract function (just replece add with sub)

```
Syntax: Series.add(other, level=None, fill_value=None, axis=0)

Parameters:
other: other series or list type to be added into caller series
fill_value: Value to be replaced by NaN in series/list before
adding
level: integer value of level in case of multi index

Return type: Caller series with added values
```

Multiplication function

```
Syntax: Series.mul(other, level=None, fill_value=None, axis=0)

Parameters:
other: other series or list type to be added into caller series
fill_value: Value to be replaced by NaN in series/list before
adding
level: integer value of level in case of multi index
```

```
Return type: Caller series with added values
```

Division function

```
Syntax: Series.div(other, level=None, fill_value=None, axis=0)

Parameters:
other: other series or list type to be divided by the caller
series
fill_value: Value to be replaced by NaN in series/list before
division
level: integer value of level in case of multi index

Return type: Caller series with divided values
```

In [81]:
```
# subtract AveragePrice with AveragePrice :P
# Dah its 0
avocado["AveragePrice"].sub(avocado["AveragePrice"])
```

Out[81]:
```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
        ...
18244    0.0
18245    0.0
18246    0.0
18247    0.0
18248    0.0
Name: AveragePrice, Length: 18249, dtype: float64
```

Merge DataFrames

Syntax:

> DataFrame.merge(self, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None) → 'DataFrame'[source]¶

Merge DataFrame or named Series objects with a database-style join.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes will be ignored. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

Parameters right: DataFrame or named Series Object to merge with.

how{'left', 'right', 'outer', 'inner'}, default 'inner' on: label or list Column or index level names to join on. These must be found in both DataFrames. If on is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

left_on: label or list, or array-like Column or index level names to join on in the left DataFrame. Can also be an array or list of arrays of the length of the left DataFrame. These arrays are treated as if they are columns.

right_on: label or list, or array-like Column or index level names to join on in the right DataFrame. Can also be an array or list of arrays of the length of the right DataFrame. These arrays are treated as if they are columns.

left_index: bool, default False Use the index from the left DataFrame as the join key(s). If it is a MultiIndex, the number of keys in the other DataFrame (either the index or a number of columns) must match the number of levels.

right_index: bool, default False Use the index from the right DataFrame as the join key. Same caveats as left_index.

sort: bool, default False Sort the join keys lexicographically in the result DataFrame. If False, the order of the join keys depends on the join type (how keyword).

suffixes: tuple of (str, str), default (‘_x’, ‘_y’) Suffix to apply to overlapping column names in the left and right side, respectively. To raise an exception on overlapping columns use (False, False).

Avocado Data Analysis

Business Understanding

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

Data Understanding

The Avocado dataset was been used in this project.

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

In [85]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

In [86]:
```
df = pd.read_csv(r"D:\NIT Daily Task\Oct\4th- REGRESSION PROJECT\4th- REGRESSION
df
```

Out[86]:

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

18249 rows × 14 columns

Missing value checking

In [88]:
```
df.isnull().sum()
```

Out[88]:
```
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
```

Dropping Unnecessary columns

df = df.drop(['Unnamed: 0','4046','4225','4770','Date'],axis=1)

In [91]:
```
df.head()
```

Out[91]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

Answering questions

In [93]:
```
def get_avarage(df,column):
    """
    Description: This function to return the average value of the column

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

In [94]:
```
def get_avarge_between_two_columns(df,column1,column2):
    """
    Description: This function calculate the average between two columns in the

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

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

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

    df_column1_column2=pd.DataFrame({'column1':List,'column2':average})
    column1_column2_sorted_index=df_column1_column2.column2.sort_values(ascendin
    column1_column2_sorted_data=df_column1_column2.reindex(column1_column2_sorte

    return column1_column2_sorted_data
```

In [95]:
```
def plot(data,xlabel,ylabel):
    """
    Description: This function to draw a barplot

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

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

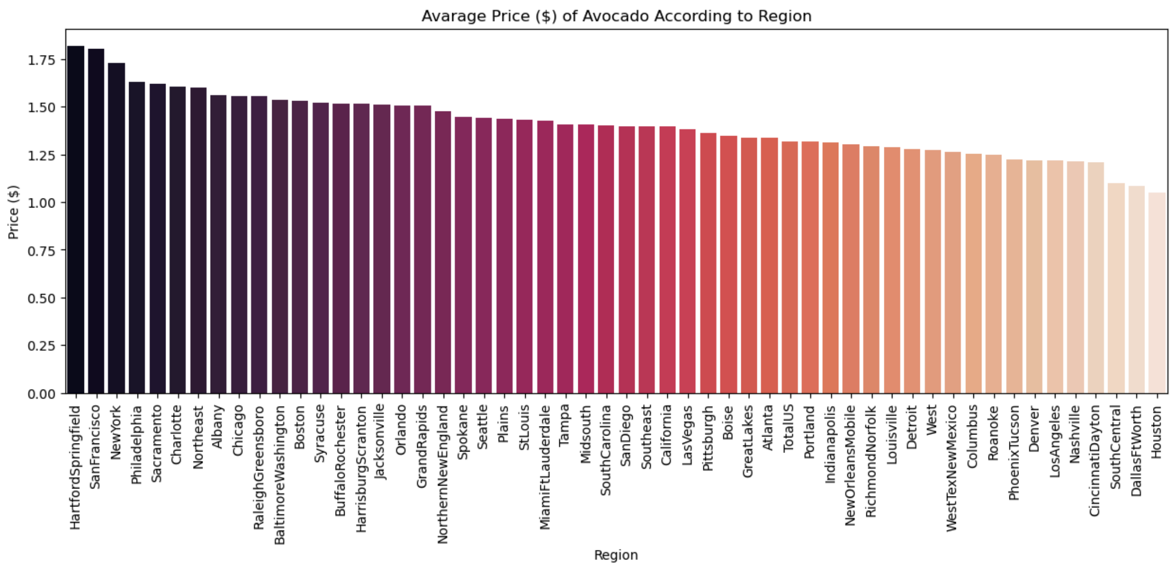
Which region are the lowest and highest prices of Avocado?

In [97]:
```
data1 = get_avarge_between_two_columns(df,'region','AveragePrice')
plot(data1,'Region','Price ($)')
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\640296719.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

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



In [98]:
```
print(data1['column1'].iloc[-1], " is the region producing avocado with the lowe
```

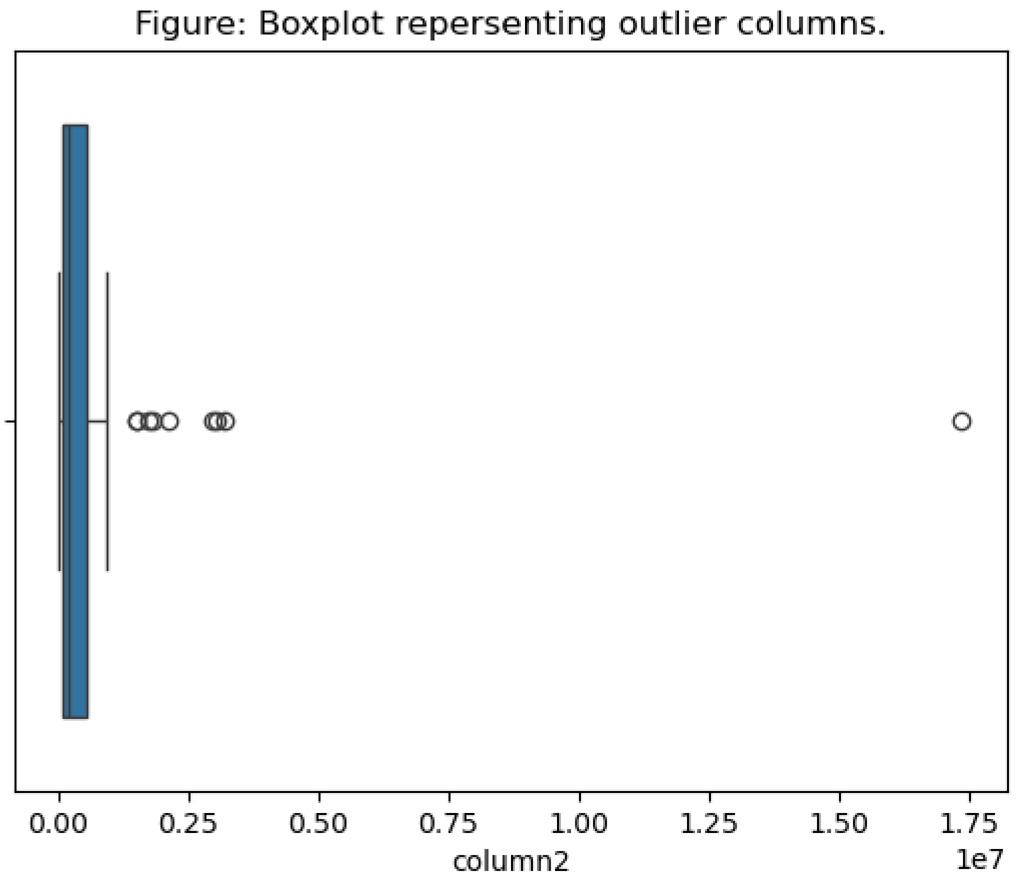
```
Houston  is the region producing avocado with the lowest price.
```

What is the highest region of avocado production?

Checking if there are outlier values or not.

In [100…
```
data2 = get_avarge_between_two_columns(df,'region','Total Volume')
sns.boxplot(x=data2.column2).set_title("Figure: Boxplot repersenting outlier col
```

Out[100…
```
Text(0.5, 1.0, 'Figure: Boxplot repersenting outlier columns.')
```



```
In [101…  outlier_region = data2[data2.column2>10000000]
          print(outlier_region['column1'].iloc[-1], "is outlier value")
```

```
TotalUS is outlier value
```

Remove the outlier Values

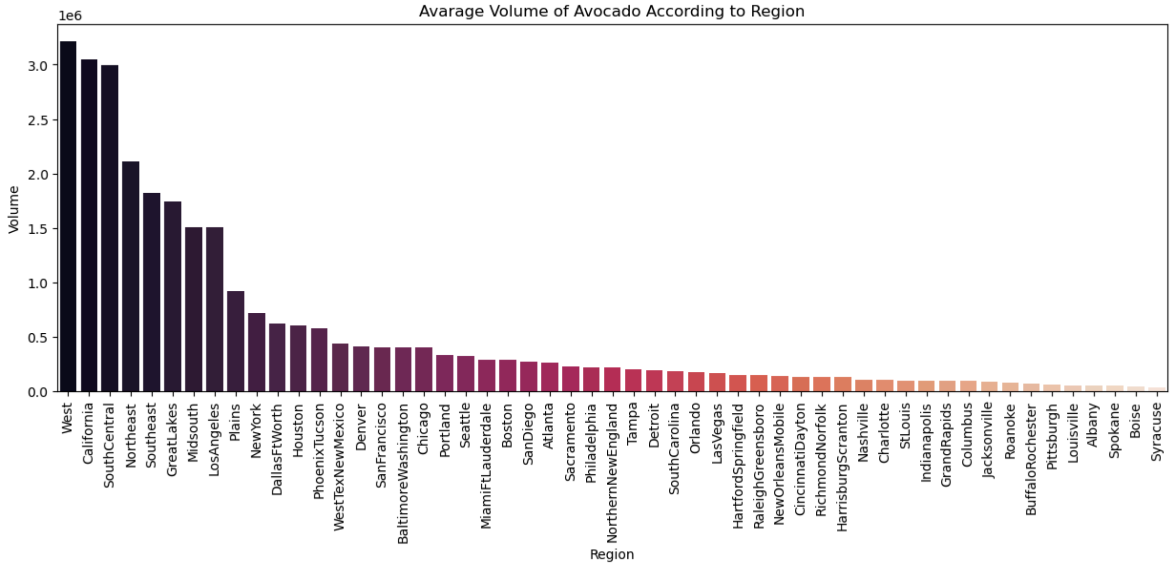
```
In [103…  outlier_region.index
          data2 = data2.drop(outlier_region.index,axis=0)
```

```
In [104…  plot(data2,'Region','Volume')
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\640296719.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

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



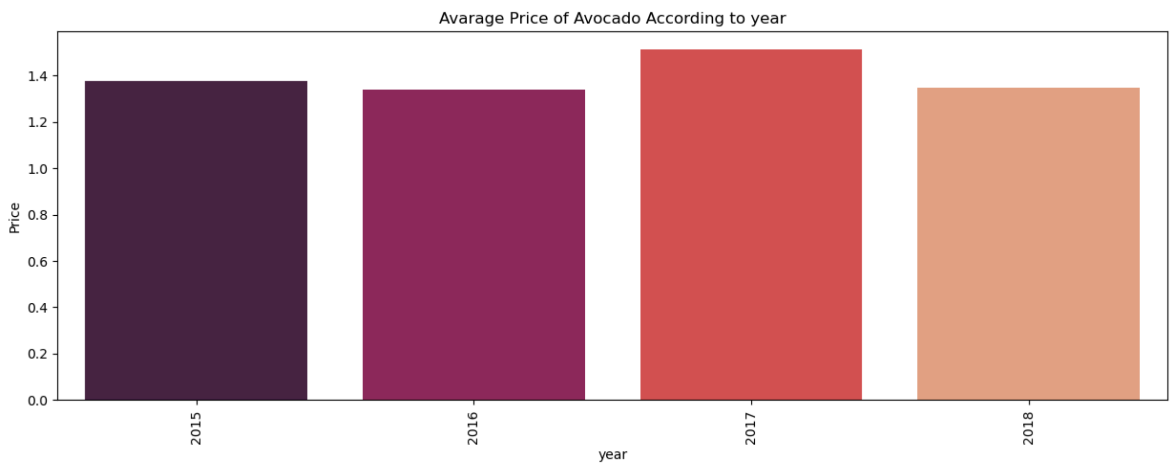
What is the average avocado prices in each year?

```
In [106…  data3 = get_avarge_between_two_columns(df,'year','AveragePrice')
          plot(data3,'year','Price')
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\640296719.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

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



What is the average avocado volume in each year?

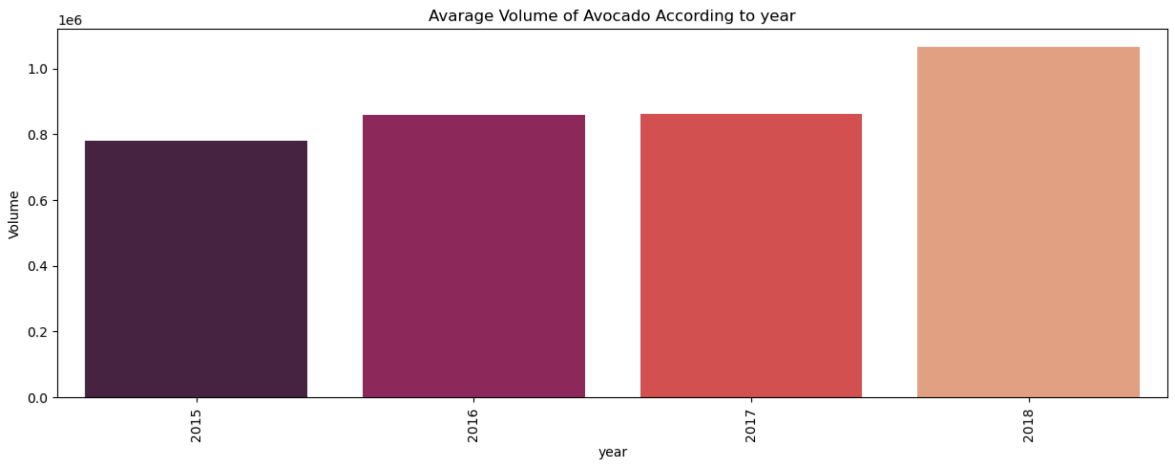
In [108…

```
data4 = get_avarge_between_two_columns(df,'year','Total Volume')
plot(data4,'year','Volume')
```

```
C:\Users\chitt\AppData\Local\Temp\ipykernel_19420\640296719.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

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



Data Modeling

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

Changing some column types to categories

In [112…

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

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

In [113…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Unnamed: 0    18249 non-null  int64  
 1   Date          18249 non-null  object 
 2   AveragePrice  18249 non-null  float64
 3   Total Volume  18249 non-null  float64
 4   4046          18249 non-null  float64
 5   4225          18249 non-null  float64
 6   4770          18249 non-null  float64
 7   Total Bags    18249 non-null  float64
 8   Small Bags    18249 non-null  float64
 9   Large Bags    18249 non-null  float64
 10  XLarge Bags   18249 non-null  float64
 11  type          18249 non-null  int8   
 12  year          18249 non-null  int64  
 13  region        18249 non-null  int8   
dtypes: float64(9), int64(2), int8(2), object(1)
memory usage: 1.7+ MB
```

In [114…

```
df.head()
```

Out[114…

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Sr B
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	598

In [115…

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

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

In [116…

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

```
training set: (12774, 13)  -  12774  samples
testing set: (5475, 13)  -  5475  samples
```

Evaluate the Results

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

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

Predicting the prices of Avacados

About the data-

> The dataset represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass, club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

Some relevant columns in the dataset:

- Date - The date of the observation
- AveragePrice - the average price of a single avocado
- type - conventional or organic
- year - the year
- Region - the city or region of the observation
- Total Volume - Total number of avocados sold
- 4046 - Total number of avocados with PLU 4046 sold
- 4225 - Total number of avocados with PLU 4225 sold
- 4770 - Total number of avocados with PLU 4770 sold

In [118…
```
from IPython.display import Image
url = 'https://img.etimg.com/thumb/msid-71806721,width-650,imgsize-807917,,resiz
Image(url,height=300,width=400)
```

Out[118…

In [120…

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
#importing the dataset
data = pd.read_csv(r"D:\NIT Daily Task\Oct\4th- REGRESSION PROJECT\4th- REGRESSI
# Check the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18249 entries, 0 to 11
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: 1.9+ MB
```

In [122…

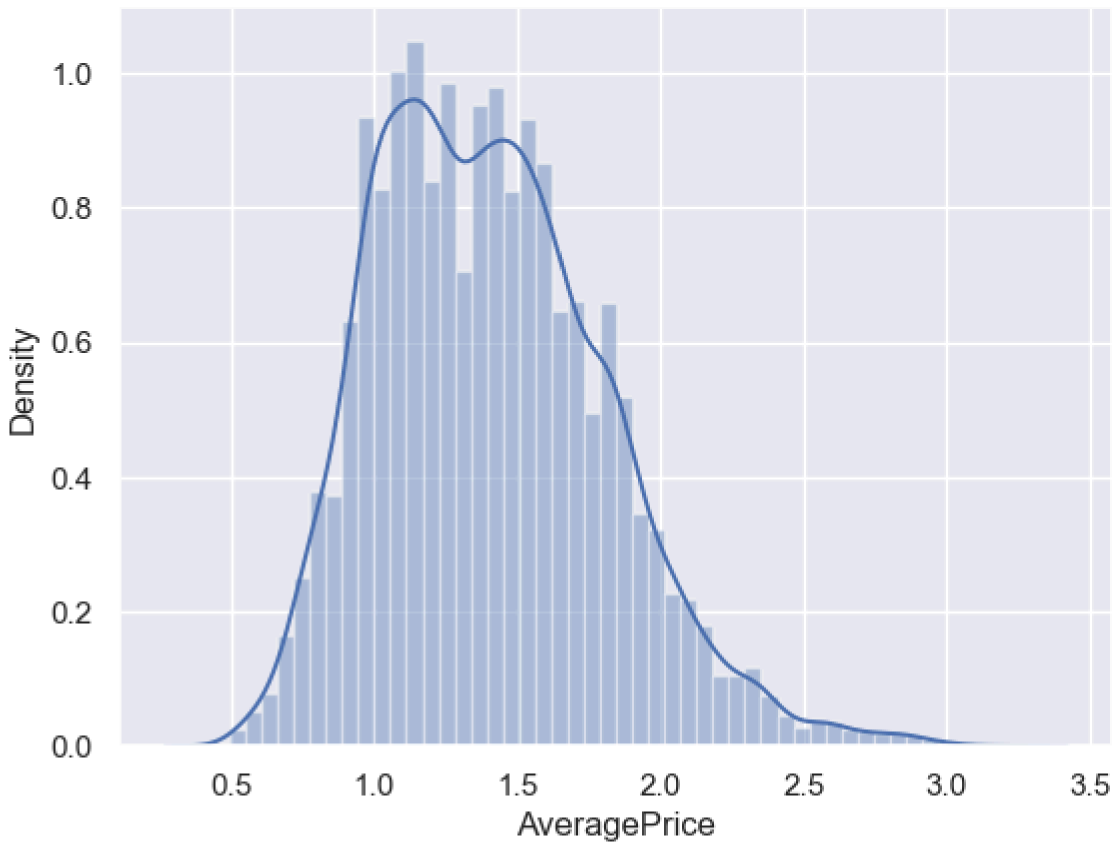
```
data.head(3)
```

Out[122…

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

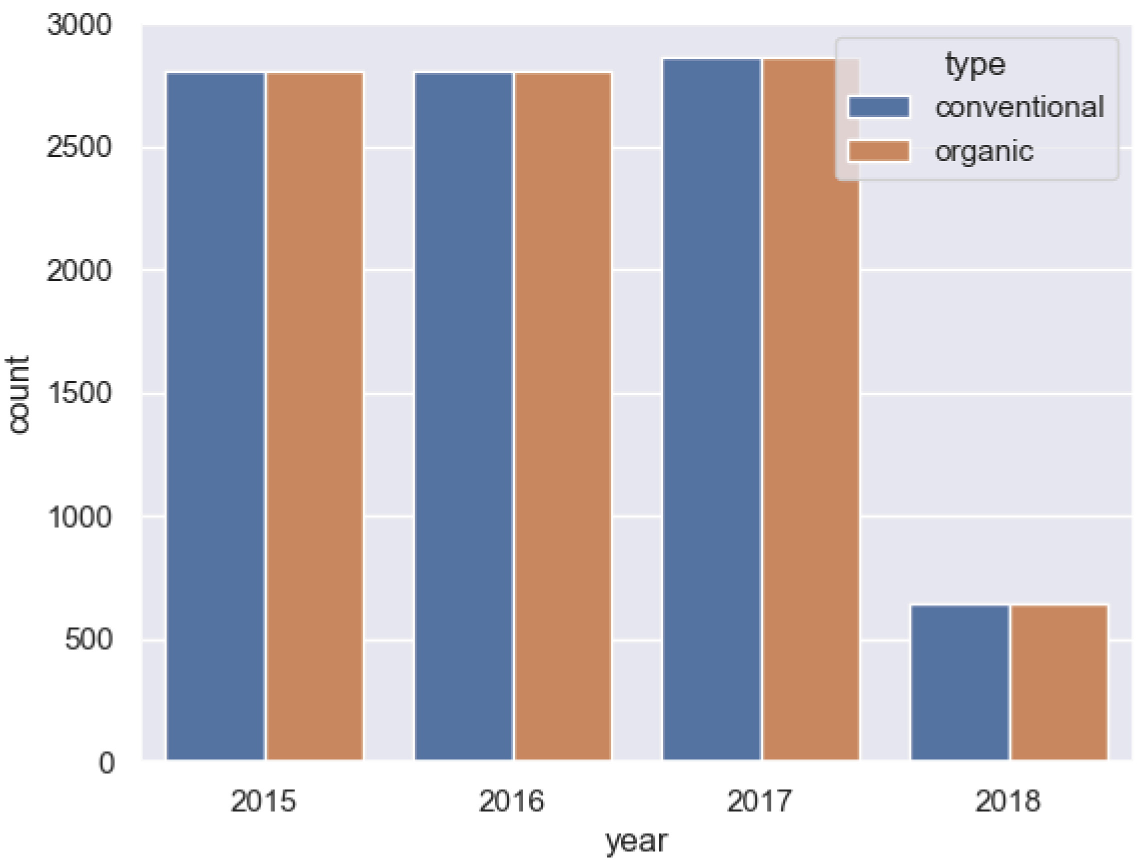
In [124…

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



In [126…

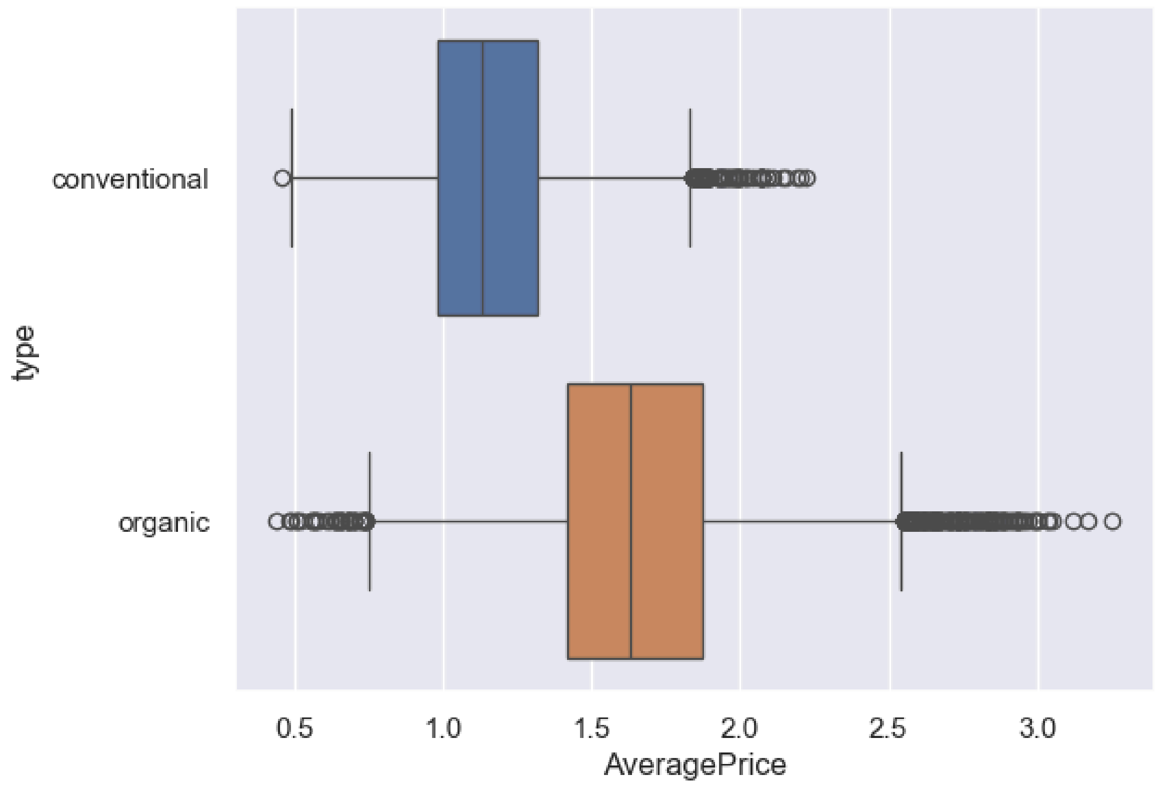
```
sns.countplot(x='year',data=data,hue='type');
```



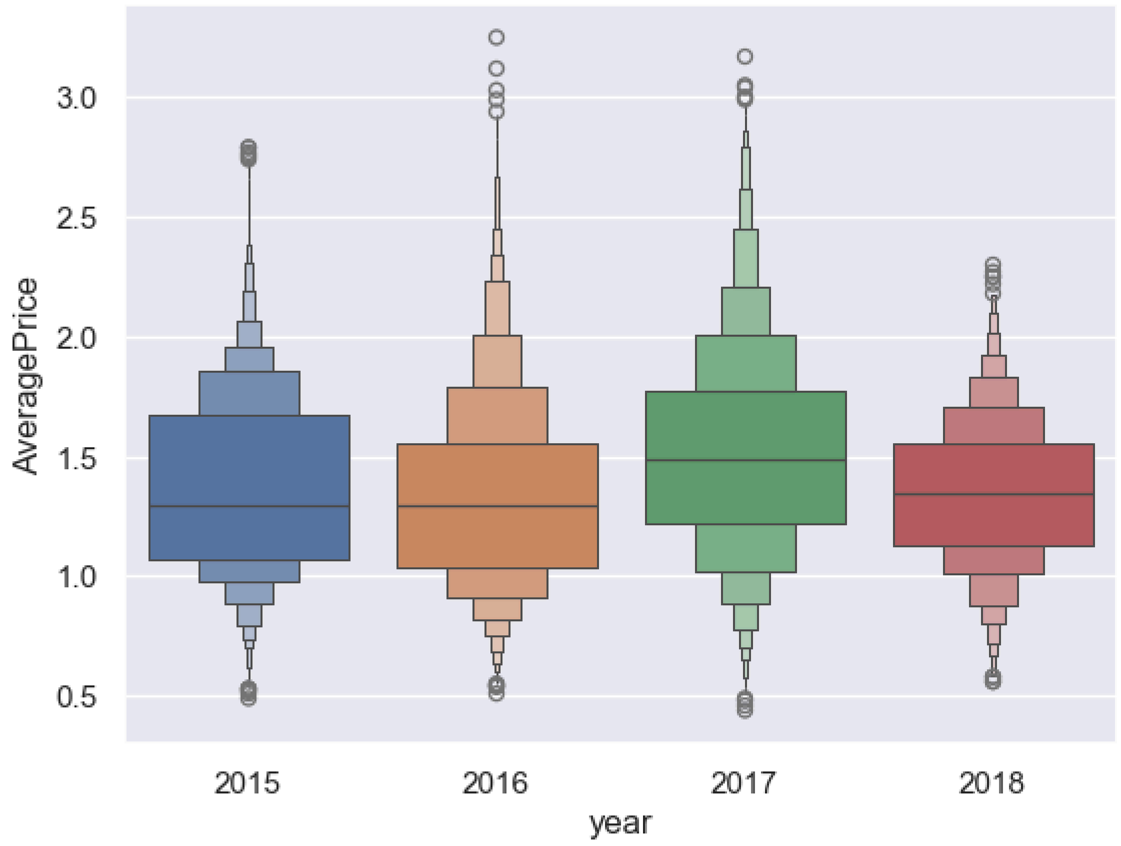
In [127…
```
data.year.value_counts()
```

Out[127…
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```

In [128…
```
sns.boxplot(y="type", x="AveragePrice",hue='type', data=data);
```



In [140…
```
data.year=data.year.apply(str)
sns.boxenplot(x="year", y="AveragePrice",hue='year', data=data);
```



Dealing with categorical features.

In [143…
```
data['type']= data['type'].map({'conventional':0,'organic':1})
```

```
# Extracting month from date column.
data.Date = data.Date.apply(pd.to_datetime)
data['Month']=data['Date'].apply(lambda x:x.month)
data.drop('Date',axis=1,inplace=True)
data.Month = data.Month.map({1:'JAN',2:'FEB',3:'MARCH',4:'APRIL',5:'MAY',6:'JUNE
```

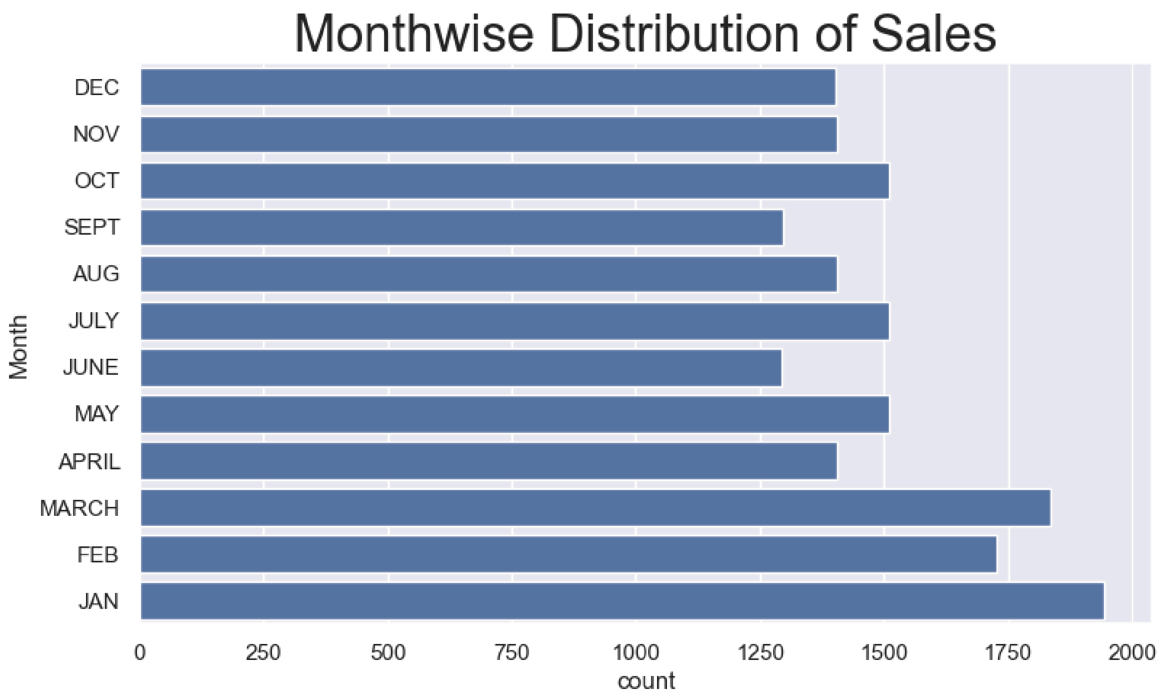
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19420\1249567319.py in ?()
      1 data['type']= data['type'].map({'conventional':0,'organic':1})
      2 
      3 # Extracting month from date column.
----> 4 data.Date = data.Date.apply(pd.to_datetime)
      5 data['Month']=data['Date'].apply(lambda x:x.month)
      6 data.drop('Date',axis=1,inplace=True)
      7 data.Month = data.Month.map({1:'JAN',2:'FEB',3:'MARCH',4:'APRIL',5:'MAY',
6:'JUNE',7:'JULY',8:'AUG',9:'SEPT',10:'OCT',11:'NOV',12:'DEC'})

~\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(nam
e)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'Date'
```

In [145…

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



Preparing data for ML models

In [148…

```
# Creating dummy variables
dummies = pd.get_dummies(data[['year','region','Month']],drop_first=True)
df_dummies = pd.concat([data[['Total Volume', '4046', '4225', '4770', 'Total Bag
       'Small Bags', 'Large Bags', 'XLarge Bags', 'type']],dummies],axis=1)
target = data['AveragePrice']

# Splitting data into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_dummies,target,test_size=

# Standardizing the data
cols_to_std = ['Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train[cols_to_std])
X_train[cols_to_std] = scaler.transform(X_train[cols_to_std])
X_test[cols_to_std] = scaler.transform(X_test[cols_to_std])
```

In [150…

```
#importing ML models from scikit-learn
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [186…

```
#to save time all models can be applied once using for loop
regressors = {
    'Linear Regression' : LinearRegression(),
    'Decision Tree' : DecisionTreeRegressor(),
    'Random Forest' : RandomForestRegressor(),
    'Support Vector Machines' : SVR(gamma=1),
    'K-nearest Neighbors' : KNeighborsRegressor(n_neighbors=1),
    'XGBoost' : XGBRegressor()
}
results=pd.DataFrame(columns=['MAE','MSE','R2-score'])
for method,func in regressors.items():
    model = func.fit(X_train,y_train)
    pred = model.predict(X_test)
    results.loc[method]= [np.round(mean_absolute_error(y_test,pred),3),
                          np.round(mean_squared_error(y_test,pred),3),
                          np.round(r2_score(y_test,pred),3)
                         ]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[186], line 12
     10 results=pd.DataFrame(columns=['MAE','MSE','R2-score'])
     11 for method,func in regressors.items():
---> 12     model = func.fit(X_train,y_train)
     13     pred = model.predict(X_test)
     14     results.loc[method]= [np.round(mean_absolute_error(y_test,pred),3),
     15                           np.round(mean_squared_error(y_test,pred),3),
     16                           np.round(r2_score(y_test,pred),3)
     17                          ]

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1474, in _fit_context.<locals
>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\Lib\site-packages\sklearn\linear_model\_base.py:578, in LinearRe
gression.fit(self, X, y, sample_weight)
    574 n_jobs_ = self.n_jobs
    576 accept_sparse = False if self.positive else ["csr", "csc", "coo"]
--> 578 X, y = self._validate_data(
    579     X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True
    580 )
    582 has_sw = sample_weight is not None
    583 if has_sw:

File ~\anaconda3\Lib\site-packages\sklearn\base.py:650, in BaseEstimator._validat
e_data(self, X, y, reset, validate_separately, cast_to_ndarray, **check_params)
    648         y = check_array(y, input_name="y", **check_y_params)
    649     else:
--> 650         X, y = check_X_y(X, y, **check_params)
    651     out = X, y
    653 if not no_val_X and check_params.get("ensure_2d", True):

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1263, in check_X_y
(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite,
ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_num
eric, estimator)
   1258         estimator_name = _check_estimator_name(estimator)
   1259     raise ValueError(
   1260         f"{estimator_name} requires y to be passed, but the target y is N
one"
   1261     )
-> 1263 X = check_array(
   1264     X,
   1265     accept_sparse=accept_sparse,
   1266     accept_large_sparse=accept_large_sparse,
   1267     dtype=dtype,
   1268     order=order,
   1269     copy=copy,
   1270     force_all_finite=force_all_finite,
   1271     ensure_2d=ensure_2d,
   1272     allow_nd=allow_nd,
   1273     ensure_min_samples=ensure_min_samples,
   1274     ensure_min_features=ensure_min_features,
   1275     estimator=estimator,
   1276     input_name="X",
   1277 )
   1279 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator
=estimator)
   1281 check_consistent_length(X, y)

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1049, in check_arr
ay(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finit
e, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input
_name)
   1043     raise ValueError(
   1044         "Found array with dim %d. %s expected <= 2."
   1045         % (array.ndim, estimator_name)
   1046     )
   1048 if force_all_finite:
-> 1049     _assert_all_finite(
   1050         array,
   1051         input_name=input_name,
   1052         estimator_name=estimator_name,
   1053         allow_nan=force_all_finite == "allow-nan",
   1054     )
   1056 if copy:
   1057     if _is_numpy_namespace(xp):
   1058         # only make a copy if `array` and `array_orig` may share memory`

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:126, in _assert_al
l_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    123 if first_pass_isfinite:
    124     return
--> 126 _assert_all_finite_element_wise(
    127     X,
    128     xp=xp,
    129     allow_nan=allow_nan,
    130     msg_dtype=msg_dtype,
    131     estimator_name=estimator_name,
    132     input_name=input_name,
    133 )

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:175, in _assert_al
l_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name, input_name)
    158 if estimator_name and input_name == "X" and has_nan_error:
    159     # Improve the error message on how to handle missing values in
    160     # scikit-learn.
    161     msg_err += (
    162         f"\n{estimator_name} does not accept missing values"
    163         " encoded as NaN natively. For supervised learning, you might wan
t"
   (...)
    173         "#estimators-that-handle-nan-values"
    174     )
--> 175 raise ValueError(msg_err)

ValueError: Input X contains NaN.
LinearRegression does not accept missing values encoded as NaN natively. For supe
rvised learning, you might want to consider sklearn.ensemble.HistGradientBoosting
Classifier and Regressor which accept missing values encoded as NaNs natively. Al
ternatively, it is possible to preprocess the data, for instance by using an impu
ter transformer in a pipeline or drop samples with missing values. See https://sc
```

```
ikit-learn.org/stable/modules/impute.html You can find a list of all estimators t
hat handle NaN values at the following page: https://scikit-learn.org/stable/modu
les/impute.html#estimators-that-handle-nan-values
```

Deep Natural Network

In [155…

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.20
```

In [157…

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

In [159…

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

model.compile(optimizer='Adam', loss='mean_squared_error')
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=0, patience=1
```

In [161…

```
print(X_train.isnull().sum())
print(y_train.isnull().sum())
print(X_val.isnull().sum())
print(y_val.isnull().sum())
```

```
Total Volume    0
4046            0
4225            0
4770            0
Total Bags      0
               ..
Month_MARCH     0
Month_MAY       0
Month_NOV       0
Month_OCT       0
Month_SEPT      0
Length: 76, dtype: int64
0
Total Volume    0
4046            0
4225            0
4770            0
Total Bags      0
               ..
Month_MARCH     0
Month_MAY       0
Month_NOV       0
Month_OCT       0
Month_SEPT      0
Length: 76, dtype: int64
0
```

In [163…

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

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

In [165…

```
X_train = X_train.astype(float)
y_train = y_train.astype(float)
X_val = X_val.astype(float)
y_val = y_val.astype(float)
```

In [167…

```
X_train_values = X_train.to_numpy(dtype='float32')
y_train_values = y_train.to_numpy(dtype='float32')
X_val_values = X_val.to_numpy(dtype='float32')
y_val_values = y_val.to_numpy(dtype='float32')
```

In [169…

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

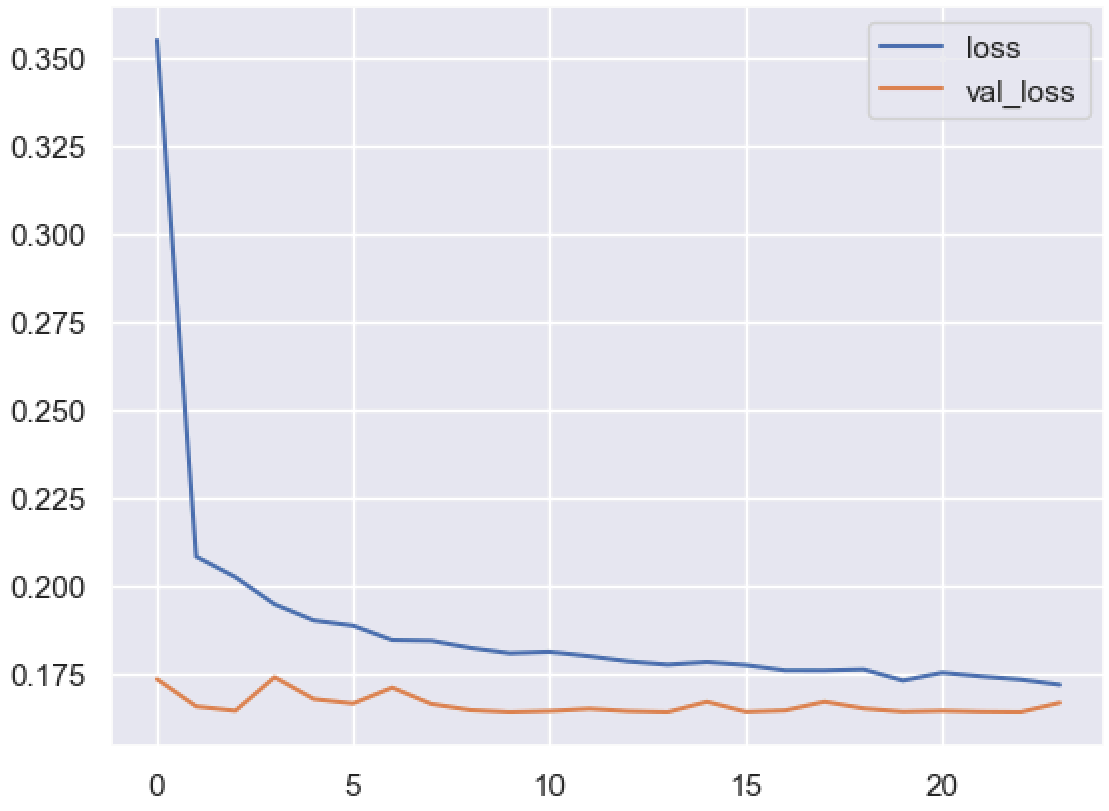
```
Epoch 1/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - loss: 0.6576 - val_loss: 0.1733
Epoch 2/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.2114 - val_loss: 0.1656
Epoch 3/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.2022 - val_loss: 0.1644
Epoch 4/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1968 - val_loss: 0.1739
Epoch 5/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.1896 - val_loss: 0.1677
Epoch 6/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1847 - val_loss: 0.1664
Epoch 7/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1857 - val_loss: 0.1709
Epoch 8/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1878 - val_loss: 0.1663
Epoch 9/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1829 - val_loss: 0.1645
Epoch 10/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.1785 - val_loss: 0.1640
Epoch 11/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1812 - val_loss: 0.1643
Epoch 12/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1815 - val_loss: 0.1649
Epoch 13/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1790 - val_loss: 0.1642
Epoch 14/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1753 - val_loss: 0.1640
Epoch 15/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.1758 - val_loss: 0.1669
Epoch 16/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 0.1734 - val_loss: 0.1640
Epoch 17/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1778 - val_loss: 0.1645
Epoch 18/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1780 - val_loss: 0.1669
Epoch 19/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1771 - val_loss: 0.1650
Epoch 20/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 0.1738 - val_loss: 0.1641
Epoch 21/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - loss: 0.1743 - val_loss: 0.1643
Epoch 22/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1730 - val_loss: 0.1641
Epoch 23/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1686 - val_loss: 0.1640
Epoch 24/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1707 - val_loss: 0.1666
```

Out[169…

```
<keras.src.callbacks.history.History at 0x1d5a222a2a0>
```

In [170…

```python
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot();
```



In [173…
```
dnn_pred = model.predict(X_test)
```

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

Results table

In [176…
```
results.loc['Deep Neural Network']=[mean_absolute_error(y_test,dnn_pred).round(3
                                    r2_score(y_test,dnn_pred).round(3)]
results
```

Out[176…

	MAE	MSE	R2-score
Deep Neural Network	0.324	0.167	-0.021

In [178…
```
f"10% of mean of target variable is {np.round(0.1 * data.AveragePrice.mean(),3)}
```

Out[178…
```
'10% of mean of target variable is 0.141'
```

In [184…
```
results.sort_values('R2-score',ascending=False).style.background_gradient(cmap='
```

Out[184…

	MAE	MSE	R2-score
Deep Neural Network	0.324000	0.167000	-0.021000

Conclusion:

- Except linear regression model, all other models have mean absolute error less than

10% of mean of target variabl.

- For this dataset, XGBoost and Random Forest algorithms have shown best results..

Completed

```
In [ ]:
```