

Avocado Price **Regression**

1. | Introduction 



Dataset Problems

👉 It is required to **predict the price of avocado** from various regions, types, years, total volumes and many more. The price prediction will use regression technique **with auto ML library called PyCaret**. In addition, this notebook will also perform simple data pre-processing before calling PyCaret regression module.

Objectives of Notebook

👉 **This notebook aims to:**

- Perform **EDA** on avocado dataset,
- Perform **data pre-processing** before using PyCaret module, and
- **Implementing PyCaret regression module** to predict avocado price,

Machine Learning Modules

👉 The model used in this notebook:

1. **PyCaret Regression Module**

Dataset Description

👉 There are **14 variables** in this dataset:

- **3 categorical** variables,
- **9 continuous** variables,
- **1 variable contains date of observation**, and
- **1 variable as index** of dataset.

👉 The following is the **structure of the dataset**.

Variable Name	Description	Sample Data
...	Index	1; 2; ...
Date	Observation date (yyyy-mm-dd format)	27-12-2015; 20-12-2015; ...
AveragePrice	Average price of an avocado	1.33; 0.93; ...
Total Volume	Total number of avocados sold	64236.62; 118220.22; ...
4046	Total number of avocados with PLU 4046 sold	2695; 263807; ...
4225	Total number of avocados with PLU 4225 sold	80596; 32457; ...
4770	Total number of avocados with PLU 4770 sold	43; 1390; ...
Total Bags	Total of Small Bags, Large Bags, and XLarge Bags combined	8696.87; 9505.56; ...
Small Bags	Total of Small Bags	8603.62; 9408.07; ...

Large Bags	Total Large Bags	93.25; 103.14; ...
XLarge Bags	Total XLarge Bags	0; 33.33; ...
type	Conventional or organic avocado	conventional; organic
year	Year from date	2015; 2017; ...
region	The city or region of the observation	Albany; Boston; ...

👉 Like this notebook? You can support me by giving upvote 😊👍▲ :.

🔗 More about myself: linktr.ee/caesarmario_

1.1 | What is PyCaret ?

PyCaret is an **open-source machine learning package written in low-code**

that enables Data Scientists to **automate their machine learning processes**. It reduces the model experimentation process, allowing for the achievement of specific outcomes with less code.


1.2 | Why using PyCaret ?

As more businesses shifted their focus to Machine Learning to address challenging issues, data scientists were expected to give results faster. This has increased the demand for automating important phases in data science projects so that **data scientists may focus on the real problem at hand rather than writing hundreds of lines of code** to identify the optimal model.

1.3 | A Quick Overview of the PyCaret Regression Module

The regression module in PyCaret is **pycaret.regression**. It is a **supervised machine learning module** for predicting values or outcomes using a variety of methods and methodologies. It includes approximately **25 algorithms** and **ten graphs** for analyzing the models' performance. PyCaret is another source for all machine learning solutions, whether it's assembly, hyper-parameter tweaking, or advanced tuning such as stacking. With PyCaret, a data scientist/user can implement **various regression modules**, such as:

- Linear Regression,
- Lasso Regression,
- Ridge Regression,
- Elastic Net,
- Decision Tree Regressor,
- Support Vector Regressor,
- AdaBoost Regressor,
- Gradient Boosting Regressor,
- Decision Tree Regressor, and many more.

∴  Further information about PyCaret [here](#).

2. | Installing and Importing Libraries

📄 **Installing** PyCaret & other libraries and **importing them** to be used in this notebook.

In [2]:

```
# --- Installing Libraries ---
```

```
!pip install pycaret
```

```
!pip install markupsafe==2.0.1
```

```
!pip jinja2
```

unfold_more Show hidden output

In [3]:

```
# --- Importing Libraries ---
```

```
import datetime
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import scipy
```

```
import pycaret
```

```
import warnings
```

```
import jinja2
```

```
from sklearn.preprocessing import LabelEncoder
```

```
from scipy import stats
```

```
from scipy.stats import *  
  
from pycaret.regression import *  
  
# --- Libraries Settings ---  
  
warnings.filterwarnings('ignore')  
  
sns.set_style('whitegrid')  
  
plt.rcParams['figure.dpi'] = 100
```

3. | Color Palettes


 This section will create some **color palettes** that will be used in this notebook.

unfold_more [Show hidden code](#)





4. | Reading Dataset

 After importing libraries, the dataset that will be used will be imported.

unfold_more Show hidden code

Out[5]:

Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small B
0	2015-12-27	1.330000	64236.620000	1036.740000	54454.850000	48.160000	8696.870000	8603.620000
1	2015-12-20	1.350000	54876.980000	674.280000	44638.810000	58.330000	9505.560000	9408.070000
2	2015-12-13	0.930000	118220.220000	794.700000	109149.670000	130.500000	8145.350000	8042.210000
3	2015-12-06	1.080000	78992.150000	1132.000000	71976.410000	72.580000	5811.160000	5677.400000

4	2015-11-29	1.280000	51039.600000	941.480000	43838.390000	75.780000	6183.950000	5986.26
---	------------	----------	--------------	------------	--------------	-----------	-------------	---------

unfold_more [Show hidden code](#)

.: Imported Dataset Info .:

Total Rows: **18249**

Total Columns: **14**

.: Dataset Details .:

<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)
```

👉 It can be seen that dataset has successfully imported.

👉 In the dataset, there are **14 columns** with **18249 observations**.

👉 Also, there are **no null values** in this dataset.

👉 The **details of each variables** also can be seen above.

5. | Initial Data Exploration 🔍

👉 This section will focused on **initial data exploration** before implementing PyCaret regression module.

5.1 | Categorical Variable 📊

👉 The first type of variable that will be explored is **categorical variable**.

5.1.1 | Type

`unfold_more` show hidden code

∴ Total Avocado based on each Type ∴

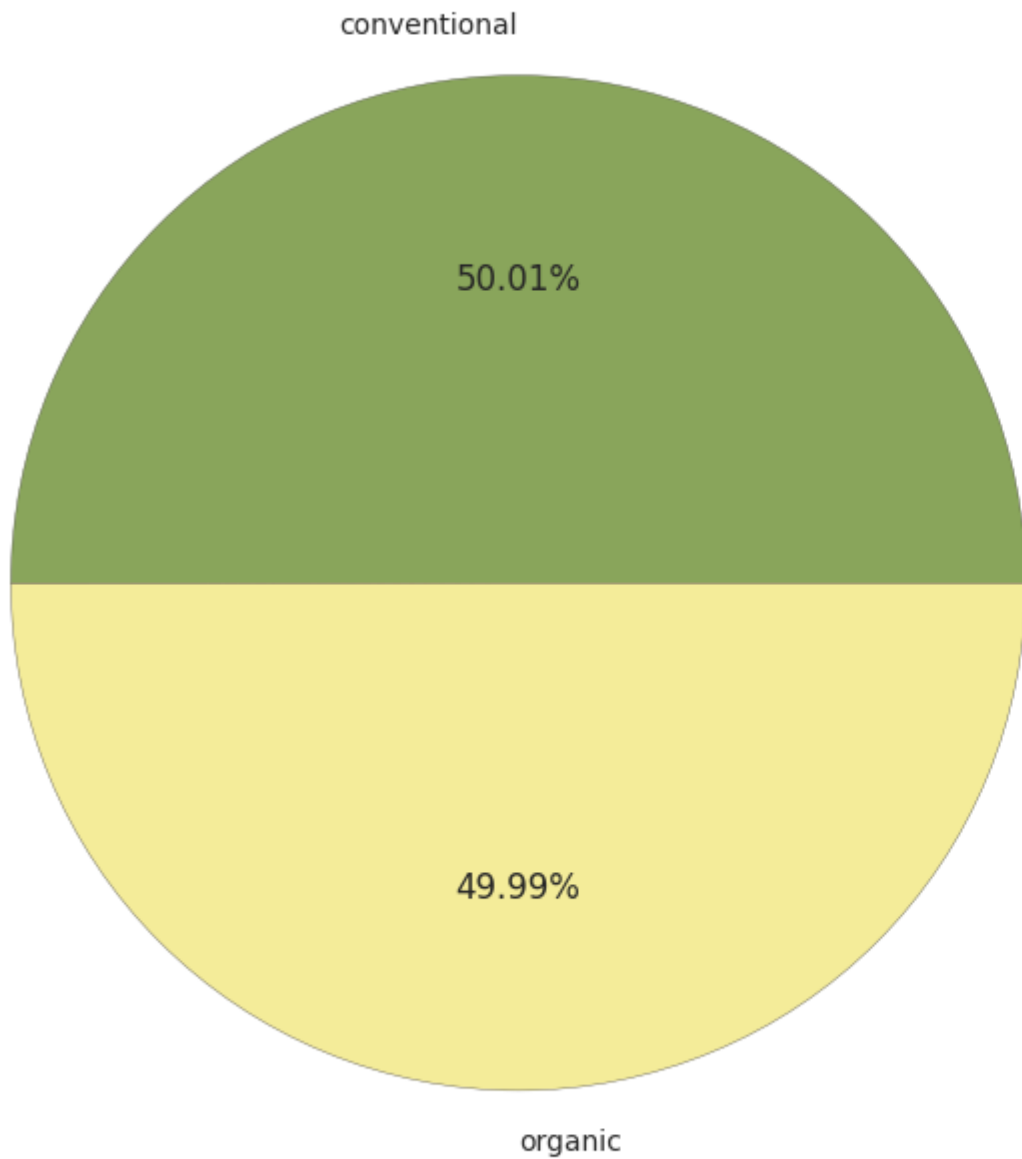
Out[7]:

conventional 9126

organic 9123

Name: type, dtype: int64

Avocado Type Percentage



📌 The distribution of conventional and organic avocados are **equally distributed**.

unfold_more Show hidden code

.: Total Avocado based on Year .:

Out[8]:

2017 5722

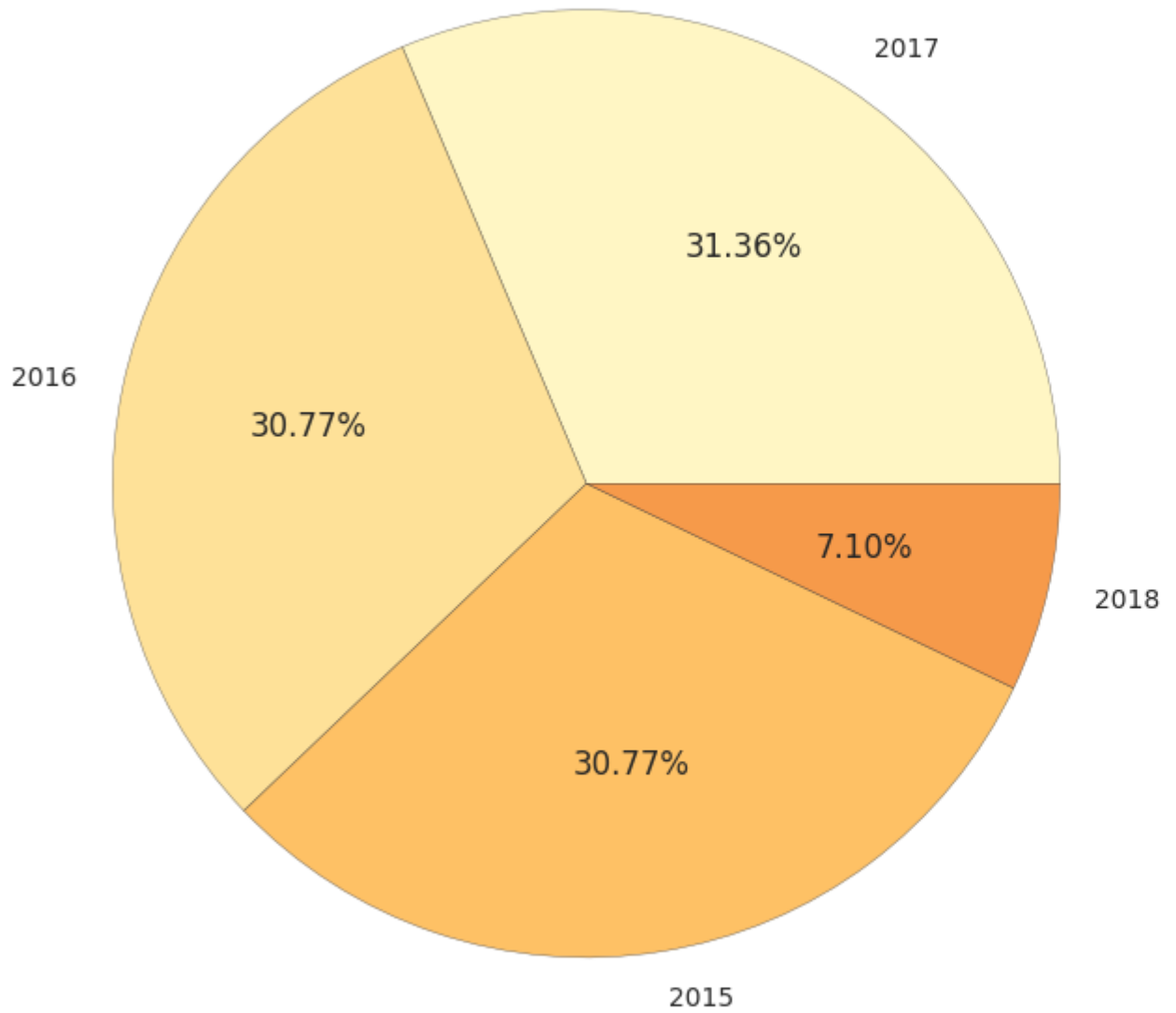
2016 5616

2015 5615

2018 1296

Name: year, dtype: int64

Avocado Years Percentage



🏠 The number of avocados in **2017 is the highest** compared to other years (with 31.36%).

🔗 However, the number of avocados in **2018 is the lowest**, only 7.10%.

5.1.3 | Region

unfold_more>Show hidden code

```
*****
```

```
.: Total Avocado based on Regions :.
```

```
*****
```

Out[9]:

```
Albany          338
Sacramento     338
Northeast      338
NorthernNewEngland 338
Orlando        338
Philadelphia    338
PhoenixTucson  338
Pittsburgh     338
Plains         338
Portland       338
RaleighGreensboro 338
RichmondNorfolk 338
Roanoke        338
SanDiego       338
```




Atlanta	338
SanFrancisco	338
Seattle	338
SouthCarolina	338
SouthCentral	338
Southeast	338
Spokane	338
StLouis	338
Syracuse	338
Tampa	338
TotalUS	338
West	338
NewYork	338
NewOrleansMobile	338
Nashville	338
Midsouth	338
BaltimoreWashington	338
Boise	338
Boston	338
BuffaloRochester	338
California	338
Charlotte	338
Chicago	338
CincinnatiDayton	338

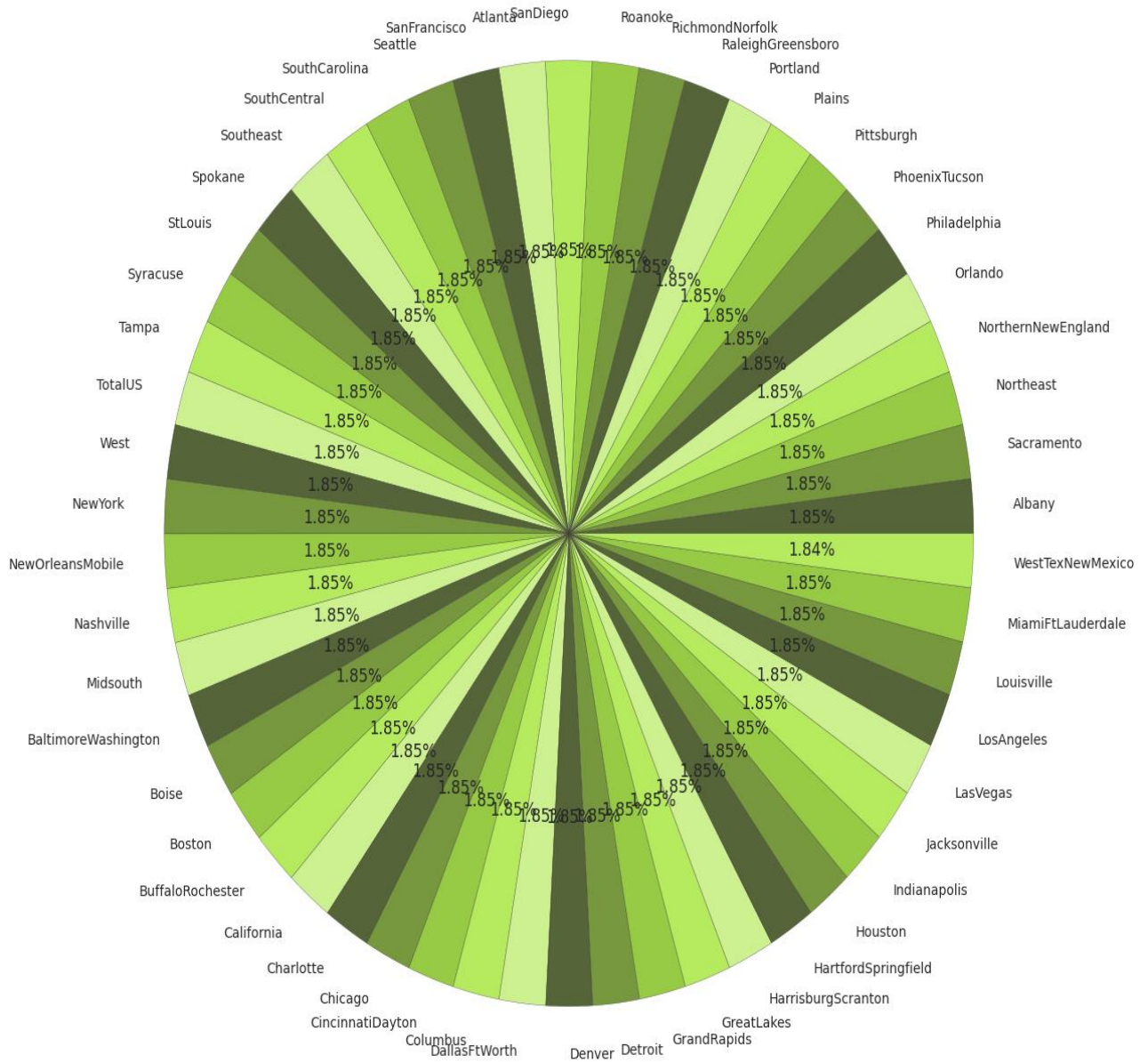


Columbus	338
DallasFtWorth	338
Denver	338
Detroit	338
GrandRapids	338
GreatLakes	338
HarrisburgScranton	338
HartfordSpringfield	338
Houston	338
Indianapolis	338
Jacksonville	338
LasVegas	338
LosAngeles	338
Louisville	338
MiamiFtLauderdale	338
WestTexNewMexico	335

Name: region, dtype: int64

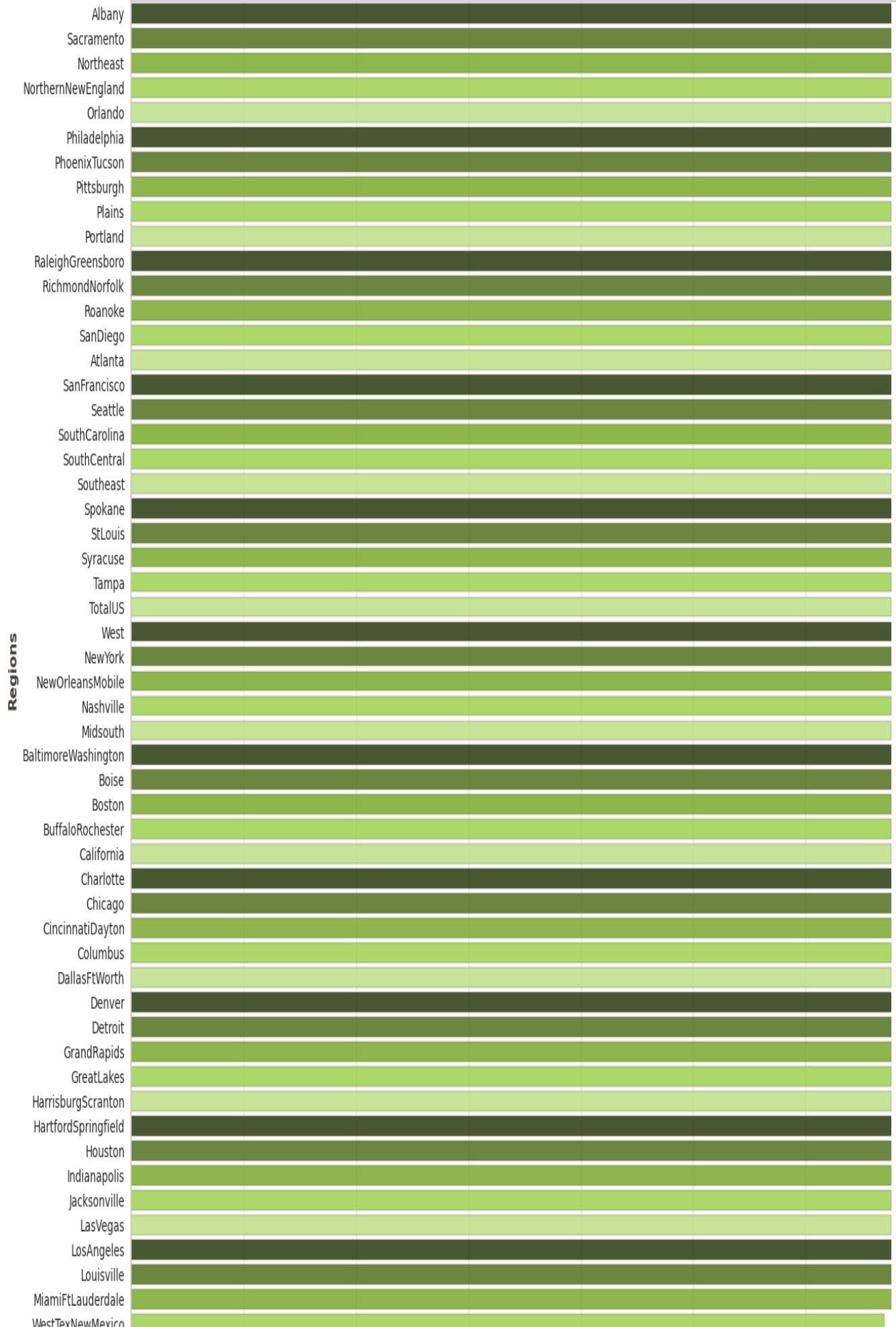
[unfold_more](#)Show hidden code

Regions Percentage



[unfold_more](#) [show hidden code](#)

Regions Bar Chart



☞ The number of avocados from various regions are **equally distributed** (with percentage of 1.85%).

☞ However, in **WestTexNewMexico**, the number of avocados slightly **lower** (335 avocados).

5.2 | Numerical Variable 1 2 3 4

☞ The second variable that will be explored is **numerical variable**.

5.2.1 | Descriptive Statistics

☞ This section will show **descriptive statistics** of numerical variables.

unfold_more Show hidden code

Out[12]:

	count	mean	std	min	25%	50%	75%	max
AveragePrice	18249.000000	1.405978	0.402677	0.440000	1.100000	1.370000	1.660000	3.250000
Total Volume	18249.000000	850644.013009	3453545.355399	84.560000	10838.580000	107376.760000	432962.290000	62505646.520000
4046	18249.000000	293008.424531	1264989.081763	0.000000	854.070000	8645.300000	111020.200000	22743616.170000
4225	18249.000000	295154.568356	1204120.401135	0.000000	3008.780000	29061.020000	150206.860000	20470572.610000

4770	18249.000 000	22839.7359 93	107464.0684 35	0.00000 0	0.000000	184.990000	6243.42000 0	2546439.1100 00
Total Bags	18249.000 000	239639.202 060	986242.3992 16	0.00000 0	5088.6400 00	39743.8300 00	110783.370 000	19373134.370 000
Small Bags	18249.000 000	182194.686 696	746178.5149 62	0.00000 0	2849.4200 00	26362.8200 00	83337.6700 00	13384586.800 000
Large Bags	18249.000 000	54338.0881 45	243965.9645 47	0.00000 0	127.47000 0	2647.71000 0	22029.2500 00	5719096.6100 00
XLarge Bags	18249.000 000	3106.42650 7	17692.89465 2	0.00000 0	0.000000	0.000000	132.500000	551693.65000 0

☞ From the descriptive statistics, it can be seen that **Average Price is lack of variation.**

☞ Furthermore, it can be seen that **the rest of the columns have more variation.**

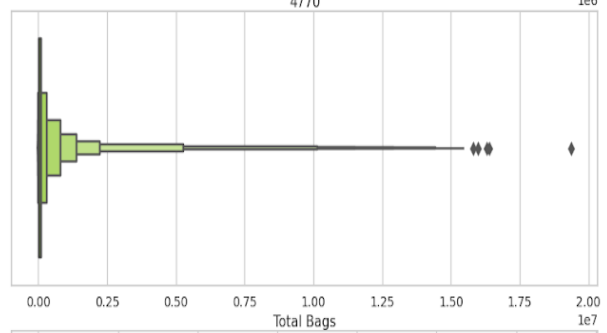
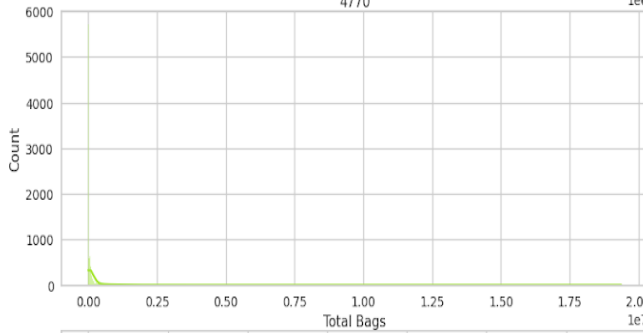
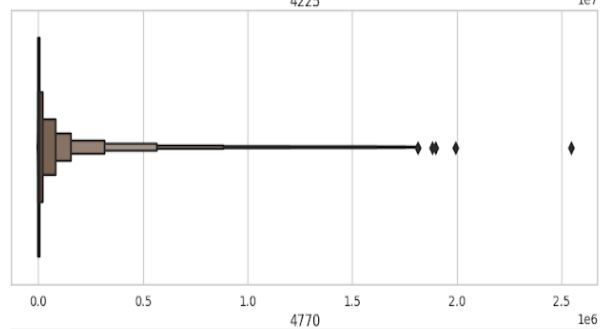
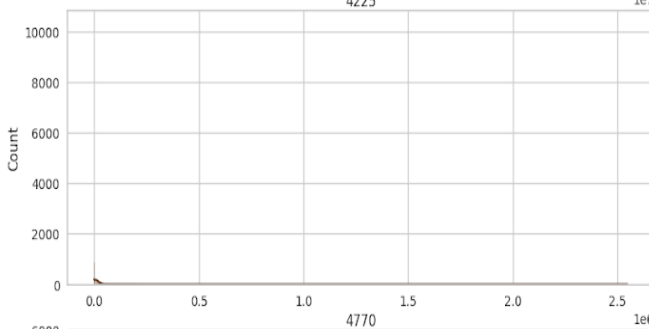
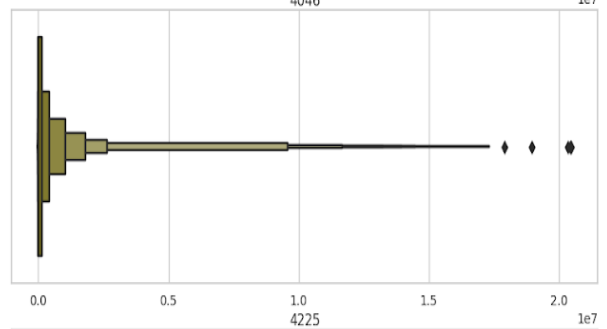
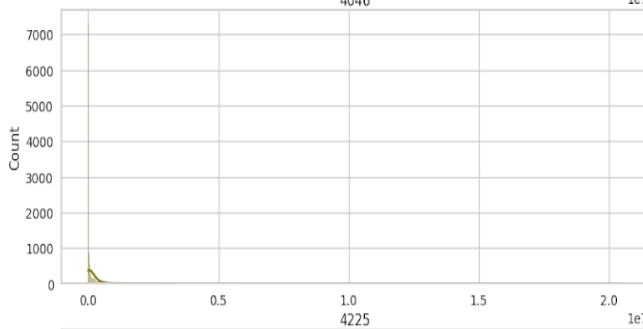
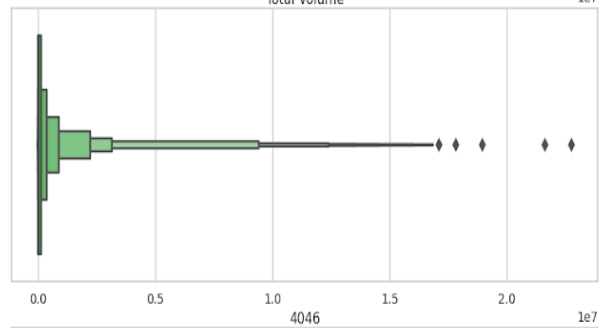
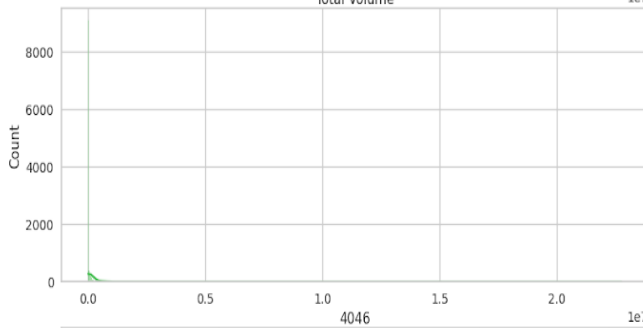
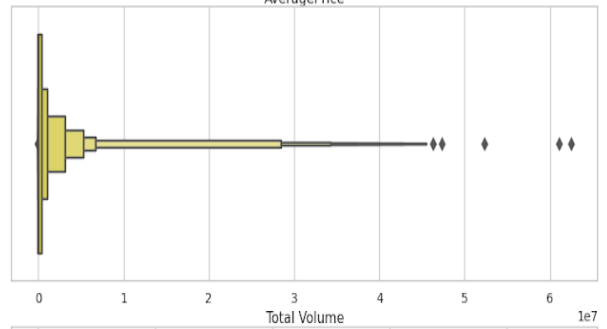
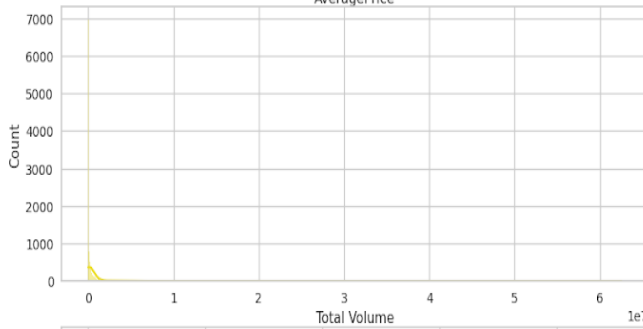
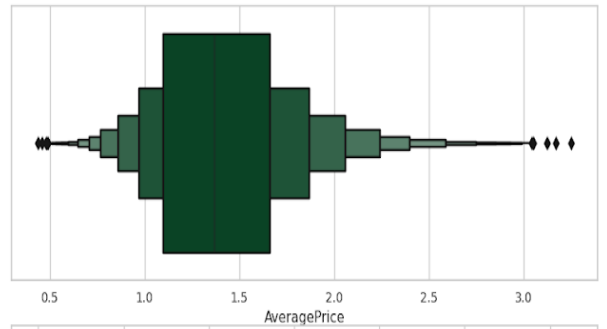
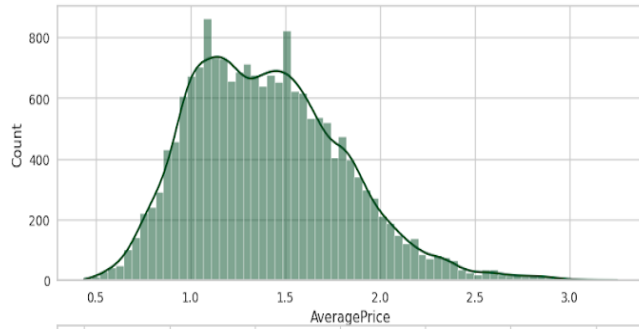
✦ **Low standard deviation** means data are **clustered around the mean** (lack of variation), and high standard deviation indicates data are more spread out (more variation).

5.2.2 | Continuous Column Distribution

☞ This section will show the **distribution of numerical variables** in histogram and boxenplot.

unfold_more Show hidden code

Histogram of Continuous Columns



☞ From the boxenplots and histogram, it can be seen that **most of the columns has extreme outliers and heavily right-skewed.**

☞ However, in Average Price column, the distribution is moderately skewed to the right and it has outliers.

5.2.3 | Skewness and Kurtosis

☞ This section will show the **numerical variables skewness and kurtosis value.**

`unfold_more` Show hidden code

∴ Continuous Columns Skewness ∴

Out[14]:

```
AveragePrice    0.580303
Total Volume    9.007687
4046             8.648220
4225             8.942466
4770             10.159396
Total Bags      9.756072
Small Bags      9.540660
Large Bags      9.796455
XLarge Bags    13.139751
dtype: float64
```

☞ As can be seen from skewness results, **all columns beside Average Price are highly right skewed** (the skewness value is > 1 , and the tail of distribution is on the right side of histogram).

unfold_more Show hidden code

∴ Continuous Columns Kurtosis ∴

Out[15]:

```
AveragePrice    0.325196
Total Volume    92.104458
4046            86.809113
4225            91.949022
4770            132.563441
Total Bags      112.272156
Small Bags     107.012885
Large Bags     117.999481
XLarge Bags    233.602612

dtype: float64
```

👉 From results above, it can be seen that **all columns beside Average Price is leptokurtic**. While **Average Price itself is platikurtic**.

👉 **Kurtosis** values used to show **tailedness of a column**. The value of normal distribution (mesokurtotic) should be equal to 3. If kurtosis value is more than 3, it is called leptokurtic. Meanwhile, if kurtosis value is less than 3, then it is called platikurtic.

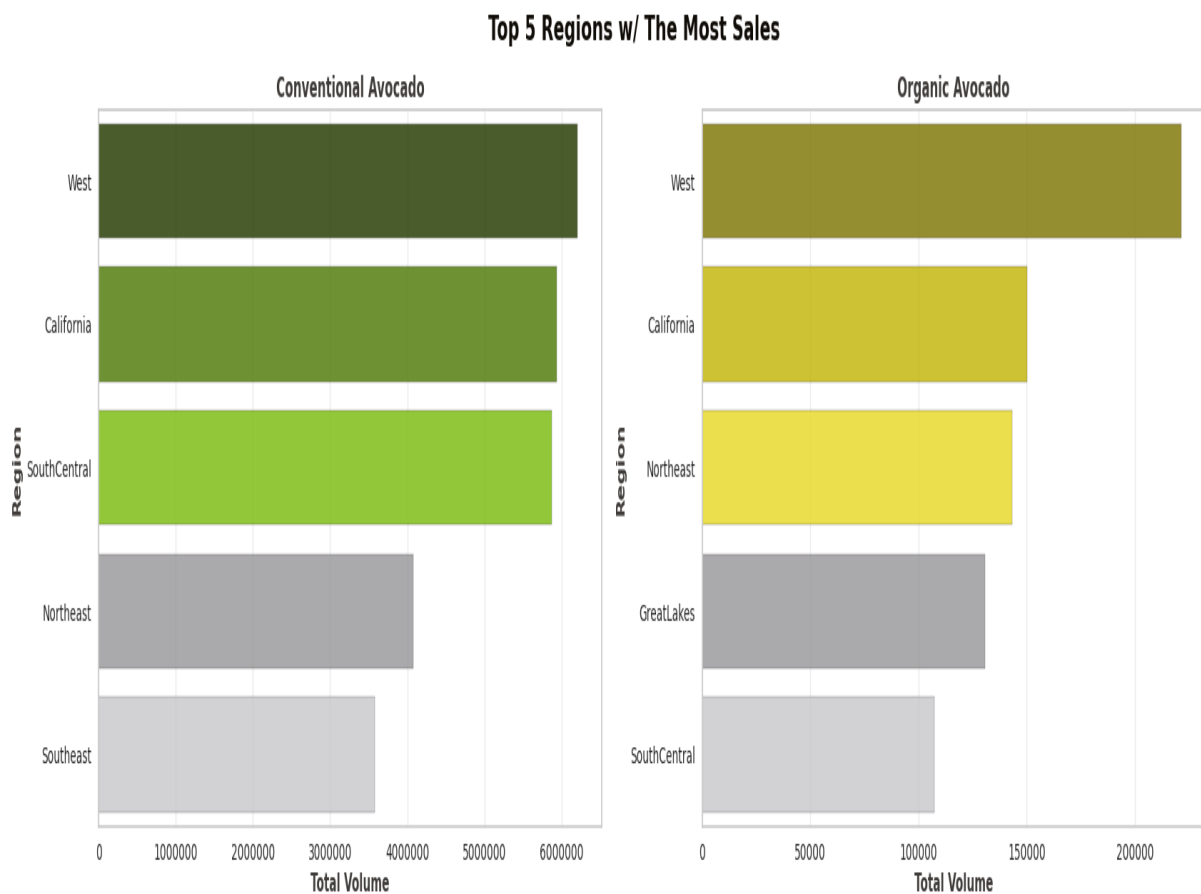
6. | EDA 

👉 This section will perform some **EDA** to get more insights about dataset.

6.1 | Top 5 Regions w/ the Most Sold

☞ Bar charts below shows **top 5 regions with most sales for each avocado types**. In this case, "TotalUS" isn't included since it is combination value from different regions in the dataset.

unfold_more [Show hidden code](#)



☞ From bar charts above, **conventional avocado has highest sales compared to organic avocados**. In conventional avocado, the highest sales is above 6.000.000, while the highest sales in organic avocado is only above 200.000 (the sales difference is about 5.800.000).

☞ Both **West and California have the highest number of avocado sold in both types**. With West in 1st position and California in 2nd position for both avocado types.

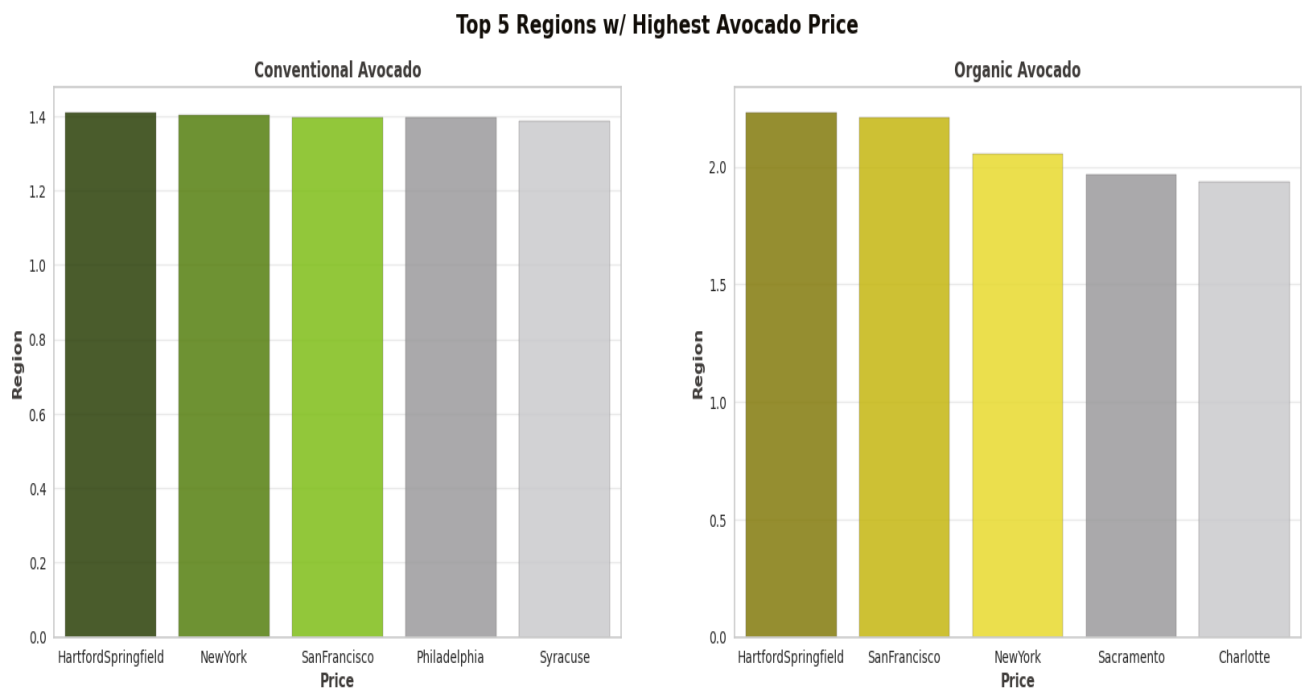
☞ It also can be seen that **conventional avocados in South Central are popular (in 3rd place)** compared to organic avocados (in 5th place).

☞ Otherwise, **in North East, organic avocados very popular (in 3rd place)** compared to conventional avocados (in 5th place).

6.2 | Top 5 Regions w/ Highest Avocado Price

☞ This section will show the **top 5 regions with highest avocado prices for each avocado types**.

unfold_more Show hidden code



☞ In general, it can be seen that **organic avocados are more expensive compared to conventional avocados**.

☞ The price differences in conventional avocados are very small compared to organic avocados, which indicates that **distribution of conventional avocados price is almost the same in every region**.

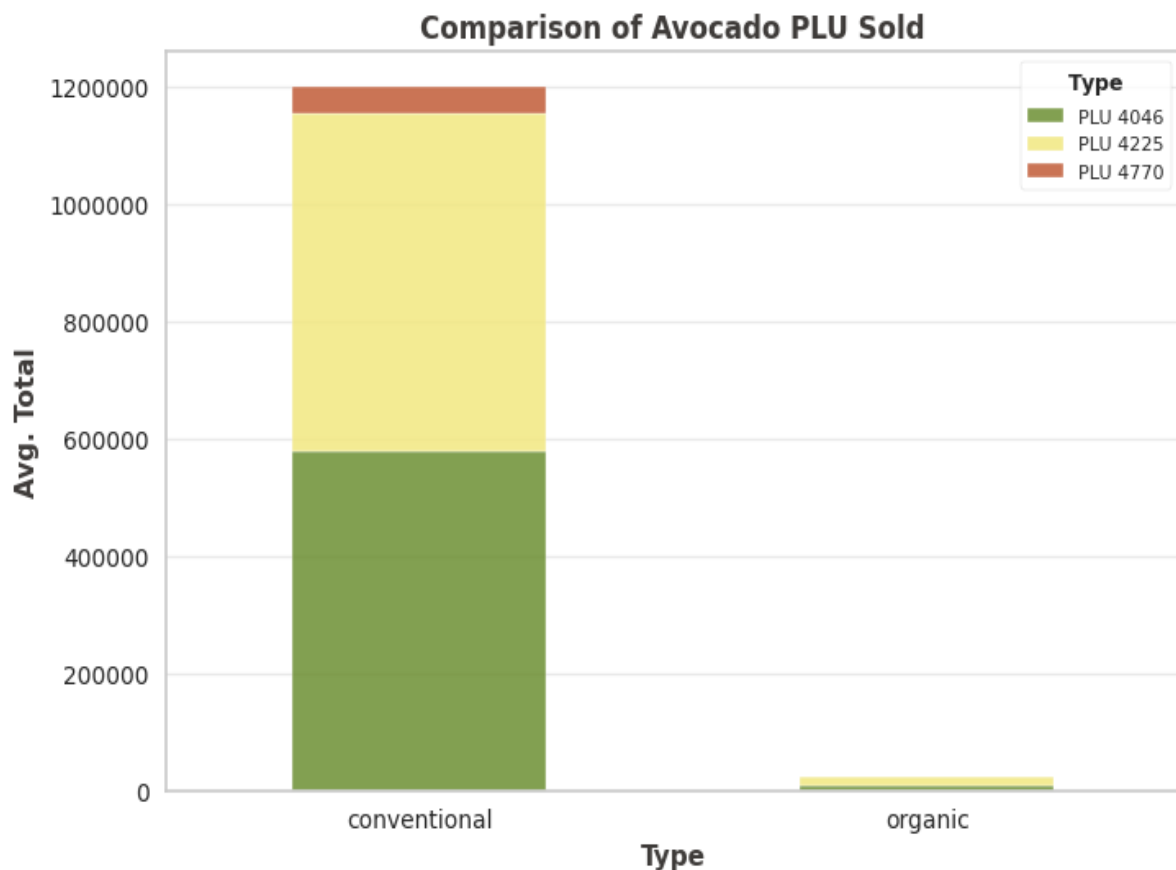
Both **Hartford Springfield, New York, and San Francisco** always become **regions with highest avocado prices**. In addition, Hartford Springfield is in the 1st position for both conventional and avocado types.

However, for **New York and San Francisco** positions are **different in conventional and organic avocado types**. In conventional avocados, New York placed 2nd, while in organic avocados, New York placed 3rd. Similarly in San Francisco, the positions of San Francisco in conventional avocados is in the 3rd place, while in organic avocados, San Francisco placed 2nd.

6.3 | Avocado PLU Sold Comparison between Avocado Types

This section will **compare total avocado sold based on PLU**. There are three PLUs, which are PLU 4046, PLU 4225, and PLU 4770.

[unfold_more](#) Show hidden code



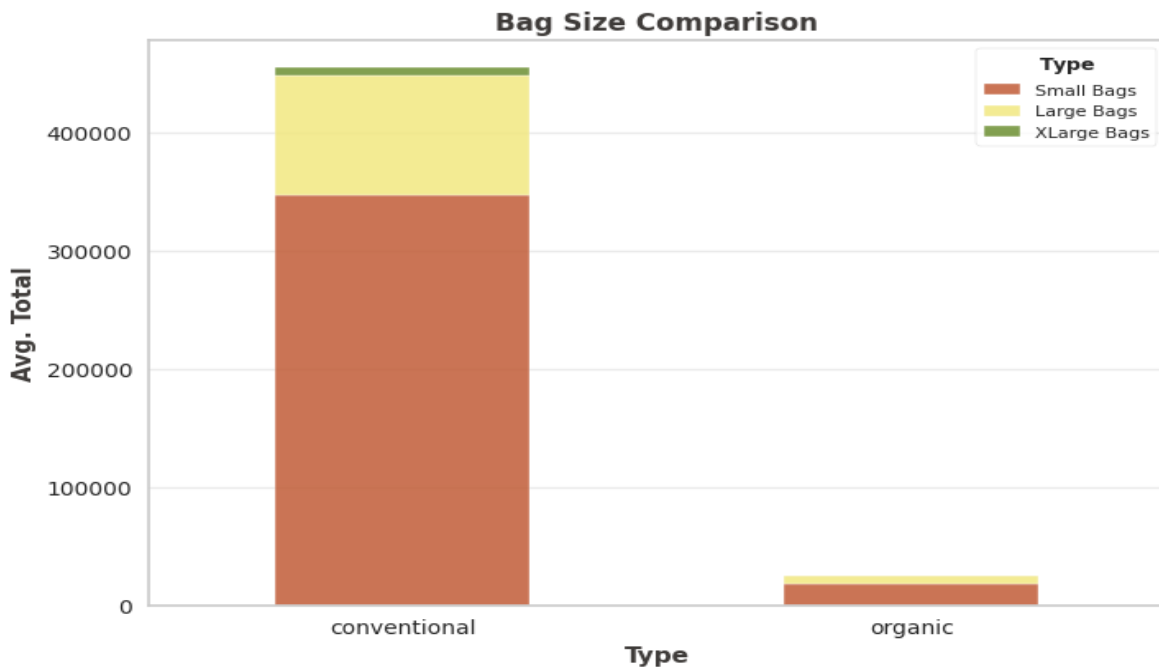
Based on stacked bar chart above, it can be seen that **conventional avocado with PLU 4046 & PLU 4225 have almost the same quantity of avocado sold**. For PLU 4770, the quantity of avocado sold is very small compared to other PLU.

In organic type, the avocado with **PLU 4225 is the highest**, followed by PLU 4046.

6.4 | Bag Size Comparison between Avocado Types

This section will **compare total avocado sold based on bags size**. There are three size of bags, small bags, large bags and extra large bags.

`unfold_more` [Show hidden code](#)

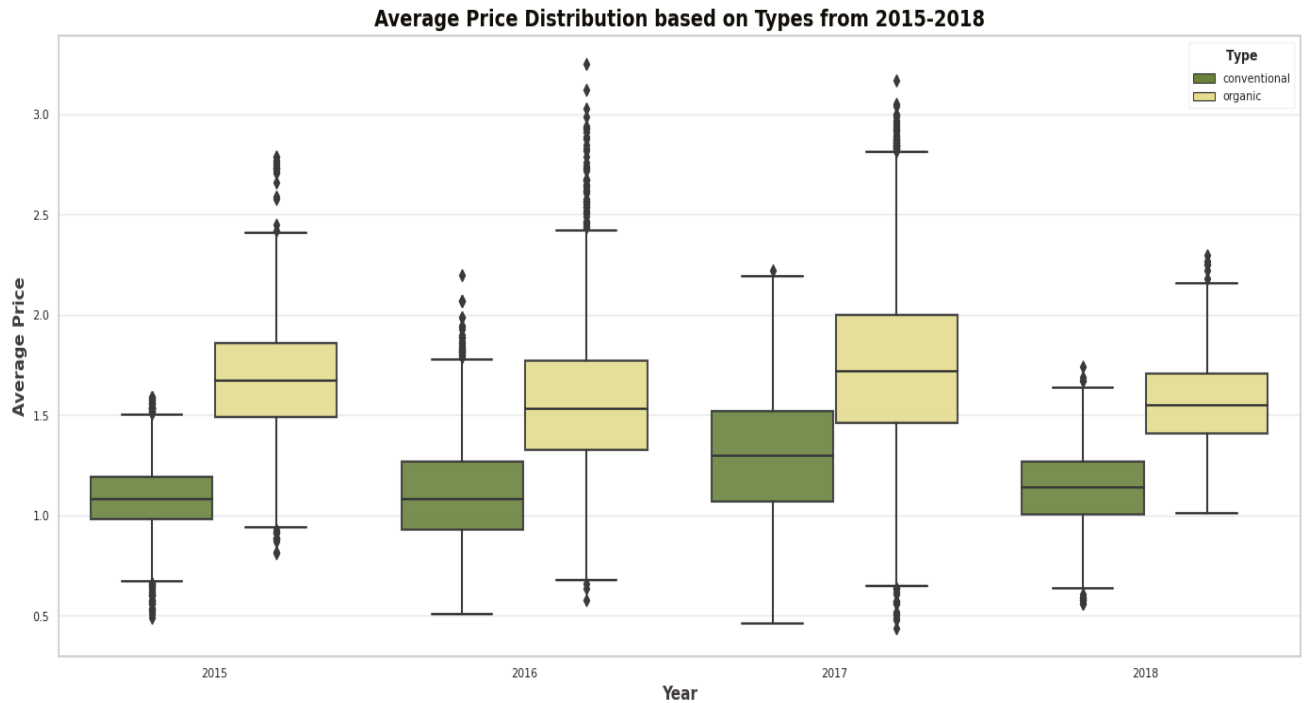


From bar above, it can be seen that the number of **small bags in both avocado types are the highest**, followed by large bags. The XLarge bags has the smallest number from all bags.

6.5 | Average Price Distribution based on Types from 2015-2018

👉 This section will show **box plots** to see the price distributions for each **avocado types** from 2015 until 2018.

unfold_more Show hidden code



👉 As mentioned in previous section, **the price of organic avocados are more expensive compared to conventional avocados.**

👉 In general, the price distribution for both avocado types decreased in 2016, then **reached its peak 2017** and finally **declined again in 2018.**

👉 The highest price of an **organic avocado** was in 2016, while the cheapest price was in 2017. Meanwhile for **conventional avocado**, the most expensive price were in 2016 and 2017, while the cheapest price was in 2017 too.



☞ Below will show the **scatter plot between total avocados sold and total avocado bags** in order to determine the heteroscedasticity from these two variables.

☞ This scatter plot will also be separated with based on avocado types with different colors.

☞ **Heteroscedasticity** is a concept that describes to **circumstances in which the residual variance is uneven throughout a spectrum of measured data**. A fan or cone form on a plot of the residuals suggests the existence of heteroscedasticity (as the variable values rise, the variance of the residuals increases proportionately). Heteroscedasticity is considered a concern in statistics because regressions using ordinary least squares (OLS) presume that the residuals are generated from a population with constant variance.

unfold_more Show hidden code



☞ In general, it can be seen that the scatter plot clearly **shows heteroscedasticity**, since the variable values increases, the distribution/residual variance also increases until form a cone shape.

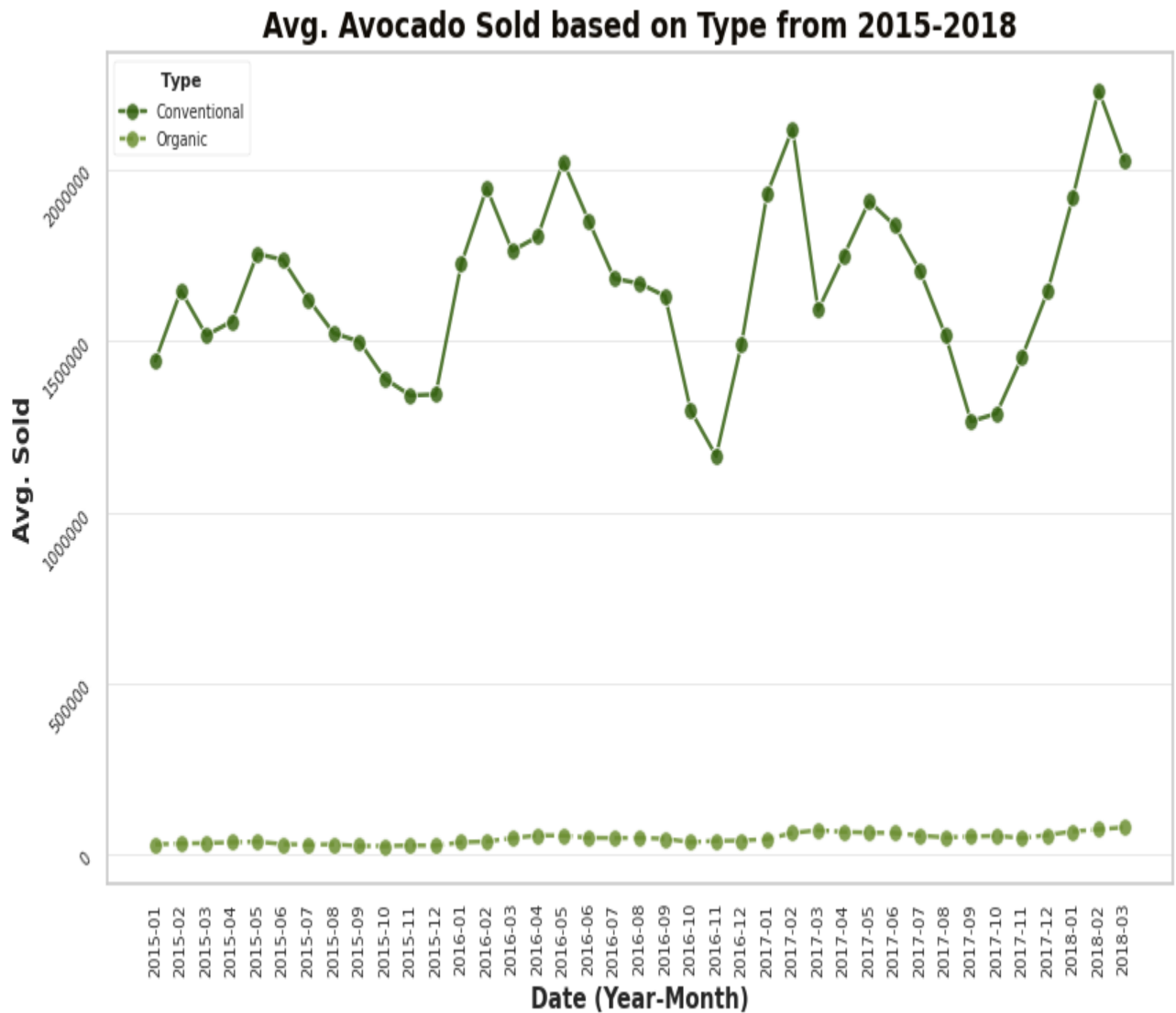
☞ From this plot, it can be concluded that **if the total avocado sold for both types increases, then the total bags also increases.**

☞ The distribution for **organic avocados relatively lower and congregate at the same spot** compared to **conventional avocados which more spread out.**

6.7 | Time Series Plot of Total Avocado Sold

Below is **times series plot about total avocado sold** from 2015-2018 based on avocado type.

unfold_more [Show hidden code](#)



As previously mentioned in previous EDA, conventional avocado has highest sales compared to organic avocados.

☞ In further analysis, it can be seen that **at the end of the year until February**

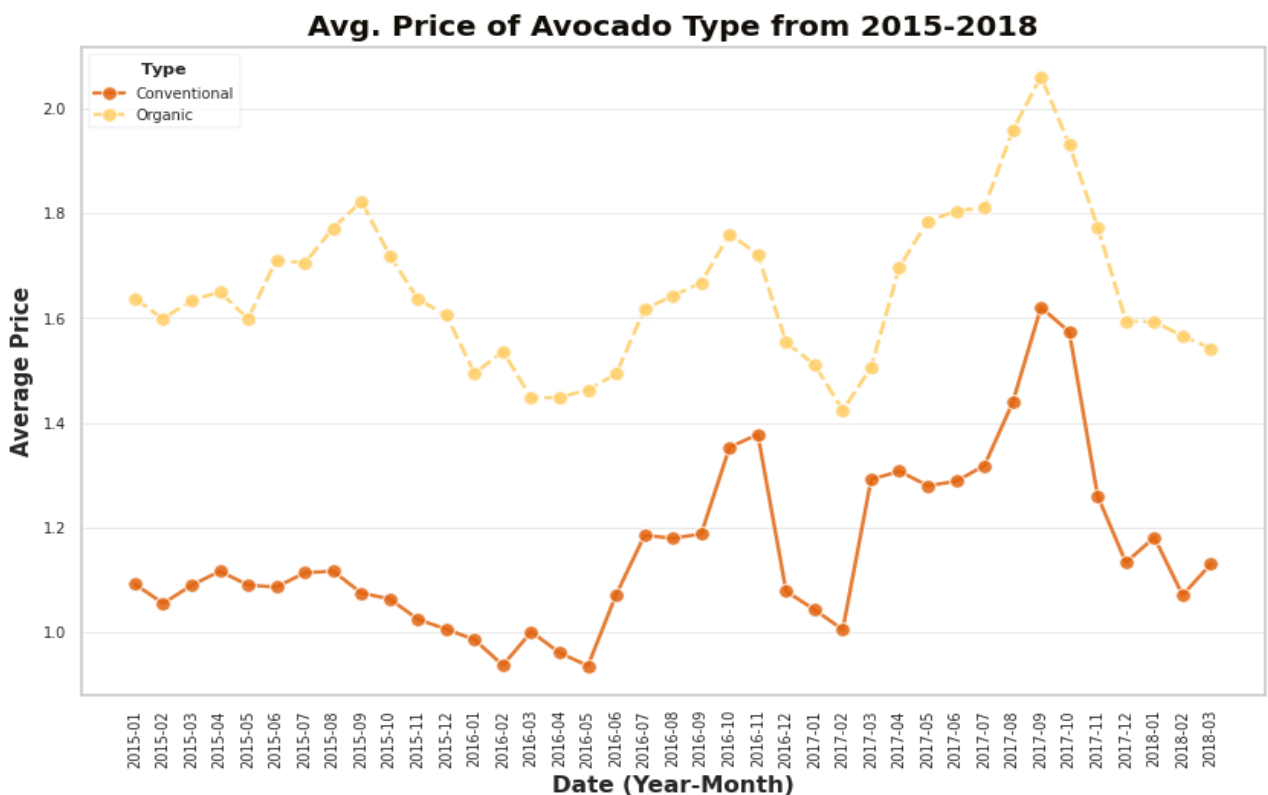
the beginning of the following year there is **an upward trend in conventional avocado sales**. The highest sales of conventional avocado was in February 2018 while the lowest sales was in November 2016.

☞ However, there is **no significant increase or decrease in organic avocado sales** from 2015 until beginning of 2018.

6.8 | Time Series Plot of Avg. Price

☞ Below is **average price time series plot of avocado** from 2015-2018 based on avocado type.

`unfold_more` [Show hidden code](#)



☞ As previously mentioned in previous section, organic avocados are more expensive compared to conventional avocados.

☞ It also can be seen that the **avocado price fluctuations for both types are similar**.

👉 In further analysis, the price of **organic avocados** always **reached its peak**

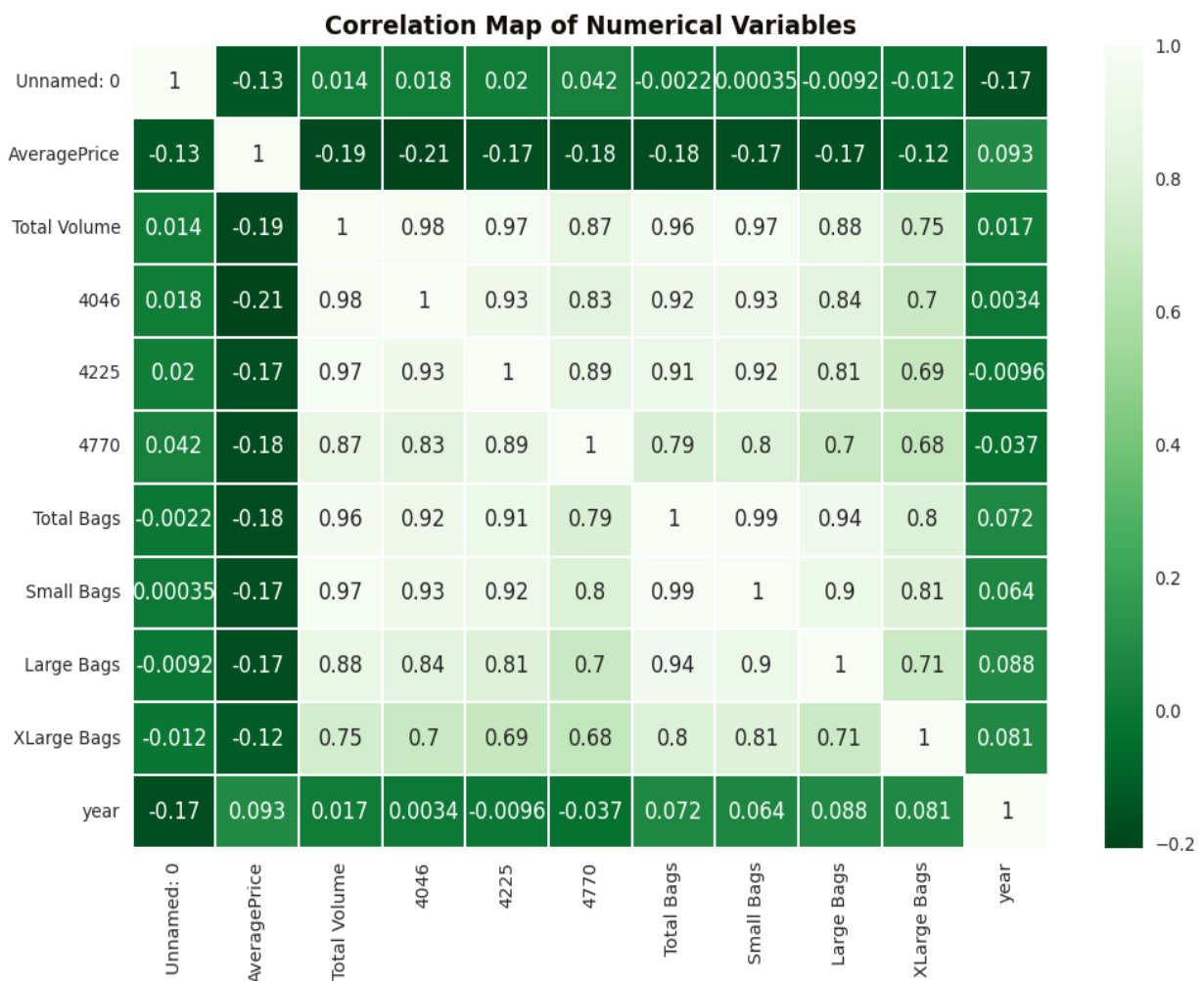
between August-September for past 3 years (2015-2017). Different case with **conventional avocados** which **doesn't have seasonality**.

👉 The highest price of organic and conventional avocados were on September 2017. However, in the next month the prices for both types declined.

6.9 | Heatmap 🔄

👉 Below is **correlation map/heatmap of numerical variables** to show correlation level/values for each variables with others.

`unfold_more`Show hidden code



☞ From plot above, **Total Volume, 4046, 4225, 4770, Total Bags, Small Bags,**

Large Bags, and XLarge Bags are highly correlated to each other since the correlation value is above 0.6 (brighter means have high correlation).

☞ Meanwhile, **Unnamed: 0, AveragePrice, and year are low correlated** since the correlation value is less than 0.1 (darker means have low correlation).

6.10 | EDA Conclusion

☞ Based on EDAs above, it can be concluded that:

- Conventional avocados have the lowest price and the most sales with little difference. Meanwhile, organic avocados have high prices and the least sales with a significant price difference. **People are more likely to choose conventional avocados because they are cheap.**
- From the time series plot of avocado prices, the lowest prices for conventional avocados fell in February and May 2016 and February 2017. Meanwhile, in the time series plots of avocado sales, sales of conventional & organic avocados increased dramatically in those months. It can be concluded that **the best time to sell conventional avocado is from the end of the year to the beginning (February).**
- The price of organic avocado follows the price of conventional avocado, but **it does not affect the total sales of avocado**, which is relatively the same every month.
- In **2017**, the average avocado price for both avocado types became the most expensive compared to the previous and following years.
- **West and California** had the highest avocado sales rates for both types, followed by South Central and North East.
- The highest prices for both avocado types are in the **Springfield, New York, and San Francisco** regions.
- Avocado with **PLU 4046 and 4225** had the highest average number of avocados compared to avocado with PLU 4770. In addition, **small avocado bags** had the highest average number compared to other avocado bags.
- **The higher the whole avocado sold, the more avocado bags available.** This is indicated by heteroscedasticity between the total volume and total bags in the scatter plot. In addition, the high correlation value in the correlation map also indicates a reasonably high correlation between these two variables.

7. | Dataset Pre-processing

👉 This section will **pre-process** the dataset before implemented into PyCaret module. A "**month**" **column will be added** into dataframe by extracting month number from "Date" column.

In [25]:

```
# --- Change `Date` Format to 'datetime' ---  
  
ds.Date = pd.to_datetime(ds.Date)  
  
# --- Extracting Month Number from `Date` ---  
  
ds['month'] = pd.DatetimeIndex(ds['Date']).month
```

8. | PyCaret Setup

👉 This section will **implement** the PyCaret regression module.

👉 In addition, this section will also **do some experiments**, including creating another models from the module and tuning it.

8.1 | Setup PyCaret Environment

👉 First, it is required to **setup** the module by defining the target, train size and etc. The **configuration** are as follows:

- **Target variable is average price,**
- **Train test size ratio is 80% train and 20% test,**
- **Defining the categorical variables** (type, year, region, and month),
- Since there are outliers from previous observation, the **normalization method will using robust technique,** and
- **Low variance features will be ignored.**

☞ To setup the PyCaret environment, `setup()` function will be used and the configuration will be added inside the bracket.

In [26]:

```
# --- Setup PyCaret Regression Module ---
```

```
avc = setup(data = ds, target = 'AveragePrice', train_size = 0.8,
```

```
          categorical_features = ['type', 'year', 'region', 'month'], normalize = True, normalize_method =  
'robust',
```

```
          silent = True, ignore_low_variance = True, session_id = 123)
```

	Description	Value
0	session_id	123
1	Target	AveragePrice
2	Original Data	(18249, 15)
3	Missing Values	False
4	Numeric Features	9

5	Categorical Features	4
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(14599, 83)
10	Transformed Test Set	(3650, 83)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10

15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	reg-default-name
19	USI	ccb8
20	Imputation Type	simple
21	Iterative Imputation Iteration	None
22	Numeric Imputer	mean
23	Iterative Imputation Numeric Model	None
24	Categorical Imputer	constant

25	Iterative Imputation Categorical Model	None
26	Unknown Categoricals Handling	least_frequent
27	Normalize	True
28	Normalize Method	robust
29	Transformation	False
30	Transformation Method	None
31	PCA	False
32	PCA Method	None
33	PCA Components	None
34	Ignore Low Variance	True

35	Combine Rare Levels	False
36	Rare Level Threshold	None
37	Numeric Binning	False
38	Remove Outliers	False
39	Outliers Threshold	None
40	Remove Multicollinearity	False
41	Multicollinearity Threshold	None
42	Remove Perfect Collinearity	True
43	Clustering	False
44	Clustering Iteration	None

45	Polynomial Features	False
46	Polynomial Degree	None
47	Trigonometry Features	False
48	Polynomial Threshold	None
49	Group Features	False
50	Feature Selection	False
51	Feature Selection Method	classic
52	Features Selection Threshold	None
53	Feature Interaction	False
54	Feature Ratio	False

55	Interaction Threshold	None
56	Transform Target	False
57	Transform Target Method	box-cox

8.2 | PyCaret Regression Models

🔗 Now, this section will show **list of models** that PyCaret regression have. `models()` will be used to list down all the models available.

In [27]:

```
# --- List PyCaret Regression Models ---
```

```
models()
```

Out[27]:

	Name	Reference	Turbo
ID			
lr	Linear Regression	sklearn.linear_model_base.LinearRegression	True

lasso	Lasso Regression	sklearn.linear_model._coordinate_descent.Lasso	True
ridge	Ridge Regression	sklearn.linear_model._ridge.Ridge	True
en	Elastic Net	sklearn.linear_model._coordinate_descent.Elast...	True
lar	Least Angle Regression	sklearn.linear_model._least_angle.Lars	True
llar	Lasso Least Angle Regression	sklearn.linear_model._least_angle.LassoLars	True
omp	Orthogonal Matching Pursuit	sklearn.linear_model._omp.OrthogonalMatchingPu...	True
br	Bayesian Ridge	sklearn.linear_model._bayes.BayesianRidge	True
ard	Automatic Relevance Determination	sklearn.linear_model._bayes.ARDRRegression	False
par	Passive Aggressive Regressor	sklearn.linear_model._passive_aggressive.Passi...	True
ransac	Random Sample Consensus	sklearn.linear_model._ransac.RANSACRegressor	False

tr	TheilSen Regressor	sklearn.linear_model._theil_sen.TheilSenRegressor	False
huber	Huber Regressor	sklearn.linear_model._huber.HuberRegressor	True
kr	Kernel Ridge	sklearn.kernel_ridge.KernelRidge	False
svm	Support Vector Regression	sklearn.svm._classes.SVR	False
knn	K Neighbors Regressor	sklearn.neighbors._regression.KNeighborsRegressor	True
dt	Decision Tree Regressor	sklearn.tree._classes.DecisionTreeRegressor	True
rf	Random Forest Regressor	sklearn.ensemble._forest.RandomForestRegressor	True
et	Extra Trees Regressor	sklearn.ensemble._forest.ExtraTreesRegressor	True
ada	AdaBoost Regressor	sklearn.ensemble._weight_boosting.AdaBoostRegr...	True
gbr	Gradient Boosting Regressor	sklearn.ensemble._gb.GradientBoostingRegressor	True

mlp	MLP Regressor	sklearn.neural_network._multilayer_perceptron....	False
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBRegressor	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMRegressor	True
catboost	CatBoost Regressor	catboost.core.CatBoostRegressor	True
dummy	Dummy Regressor	sklearn.dummy.DummyRegressor	True

8.3 | Comparing All Models

👉 `compare_models()` will be used to **evaluate all models performance** after all models successfully running. In the table will show the MAE, MSE, RMSE, R2, RMSLE, and MAPE score of each models. It also show total of time (in sec) needed to execute the models.

👉 For this experiment, **R2 score** will be used to evaluate the model.

In [28]:

```
# --- Comparing All Models ---
```

```
best_models = compare_models(sort='R2')
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	0.0729	0.0116	0.1076	0.9284	0.0432	0.0539	5.7420
catboost	CatBoost Regressor	0.0840	0.0132	0.1146	0.9187	0.0458	0.0615	4.3090
xgboost	Extreme Gradient Boosting	0.0877	0.0147	0.1213	0.9090	0.0484	0.0638	33.1610
rf	Random Forest Regressor	0.0851	0.0150	0.1223	0.9073	0.0492	0.0631	6.2110
lightgbm	Light Gradient Boosting Machine	0.0990	0.0178	0.1332	0.8904	0.0537	0.0732	0.2660
dt	Decision Tree Regressor	0.1169	0.0322	0.1794	0.8010	0.0721	0.0855	0.1230
knn	K Neighbors Regressor	0.1356	0.0353	0.1876	0.7825	0.0768	0.1021	0.6520
gbr	Gradient Boosting Regressor	0.1537	0.0411	0.2025	0.7465	0.0816	0.1155	2.1100
br	Bayesian Ridge	0.1819	0.0582	0.2411	0.6409	0.0974	0.1372	0.0790

ridge	Ridge Regression	0.1819	0.0582	0.2411	0.6409	0.0974	0.1372	0.0320
lr	Linear Regression	0.1820	0.0583	0.2413	0.6403	0.0975	0.1372	0.5380
ada	AdaBoost Regressor	0.2110	0.0677	0.2602	0.5811	0.1085	0.1683	1.1120
huber	Huber Regressor	0.2101	0.0773	0.2779	0.5227	0.1123	0.1590	0.6530
omp	Orthogonal Matching Pursuit	0.2159	0.0782	0.2796	0.5171	0.1145	0.1663	0.0290
en	Elastic Net	0.3194	0.1584	0.3980	0.0221	0.1636	0.2522	0.0290
lasso	Lasso Regression	0.3217	0.1602	0.4001	0.0114	0.1646	0.2542	0.0270
llar	Lasso Least Angle Regression	0.3243	0.1622	0.4027	-0.0013	0.1655	0.2561	0.4980
dummy	Dummy Regressor	0.3243	0.1622	0.4027	-0.0013	0.1655	0.2561	0.0270
par	Passive Aggressive Regressor	0.7842	18.2459	3.0802	-115.4978	0.3362	0.6811	0.0450

👉 From the models, it can be seen that the **top 3 models** based on PyCaret regression models are:

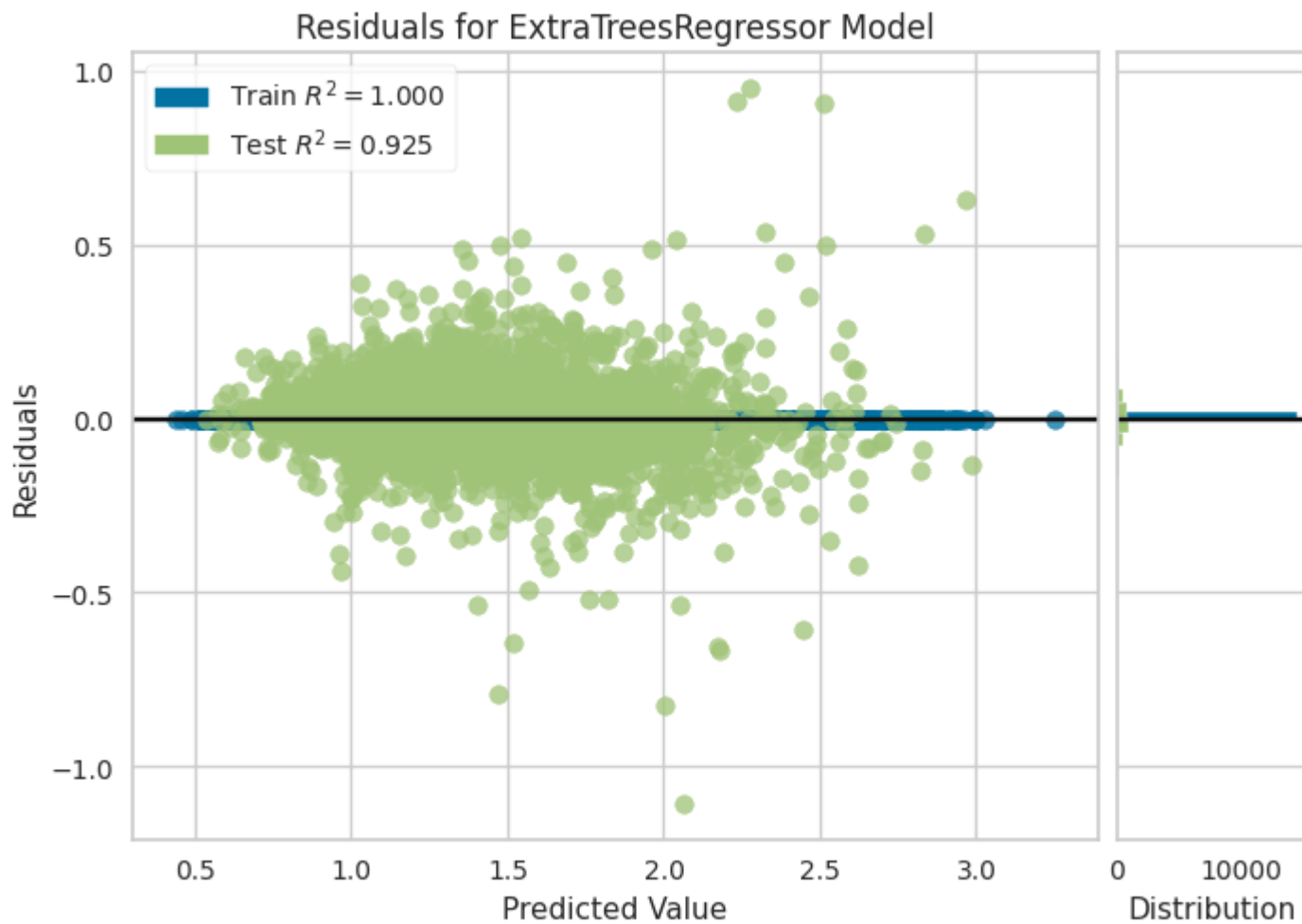
- **Extra Tree Regressor** (0.9284),
- **Random Forest Regressor** (0.9073), and
- **Light Gradient Boosting Machine** (0.8904).

☞ Now, let's **plot the best model** (extra tree regressor) to see the residuals and its performance in train and test set.

In [29]:

```
# --- Plot the Residual of Best Model (et) ---
```

```
plot_model(best_models)
```



☞ From the plot, it can be seen that the **test set is randomly dispersed around horizontal axis**, which indicates good fit of model.

It also can be seen that the best model can achieve **1.000** R2 score in train

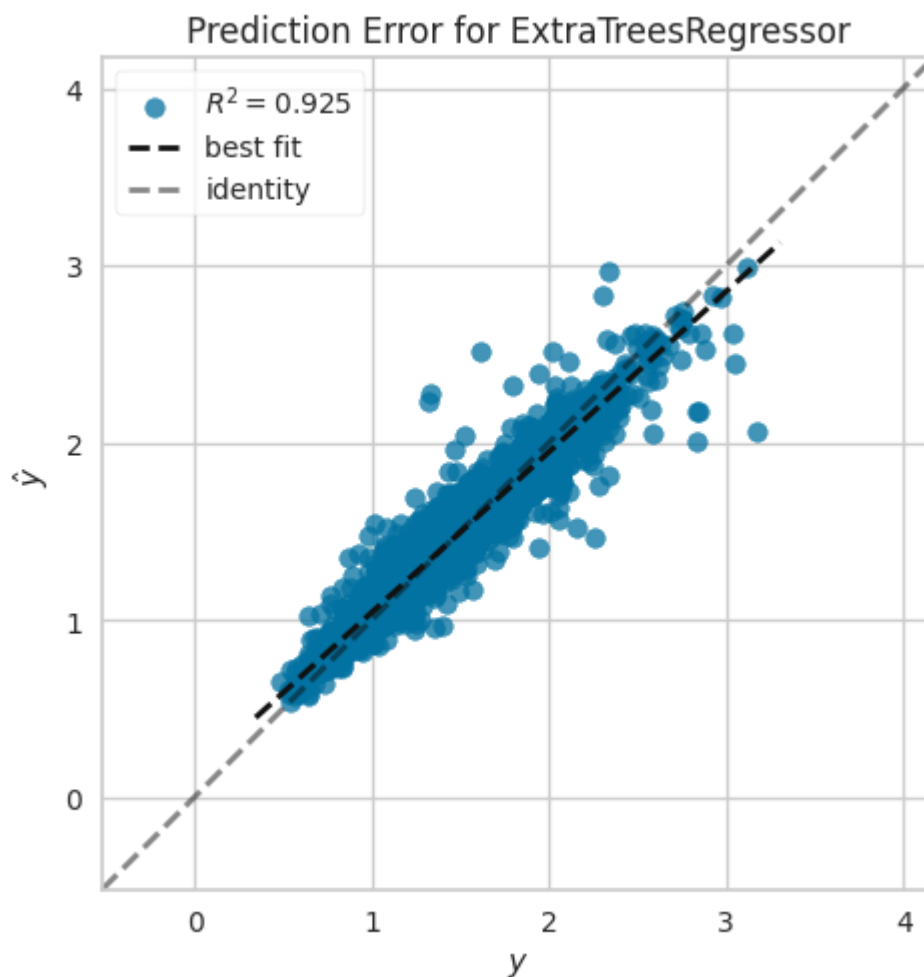
set, and **0.925** in test set. This means that there is **no overfitting/underfitting** happened which indicates very good performance.

A **residual plot** is a graph that **shows the residuals on the vertical axis and the independent variable on the horizontal axis**. If the points in a residual plot are randomly dispersed around the horizontal axis, a linear regression model is appropriate for the data; otherwise, a nonlinear model is more appropriate.

In [30]:

```
# --- Plot Error Prediction for Best Model ---
```

```
plot_model(best_models, plot = 'error')
```



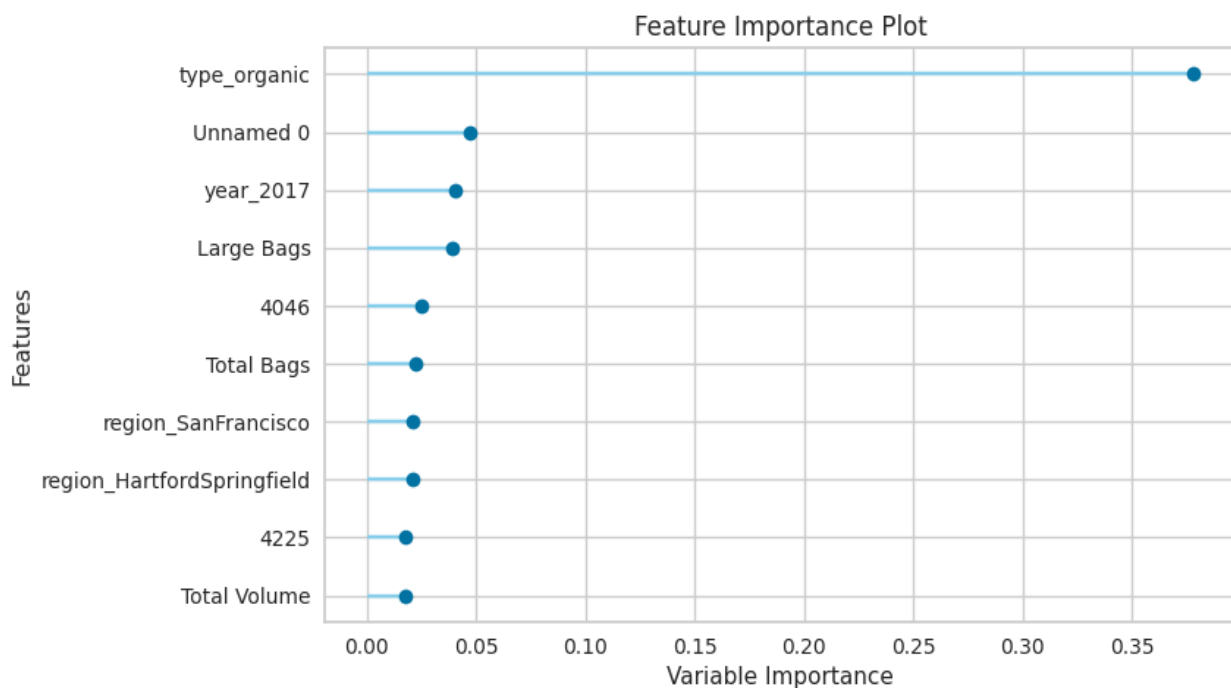
☞ From the error plot, it can be seen that there **a little gap** between the best fit line (predicted values) against the identity line (actual targets). This means that **the accuracy of extra tree regressor is very good for prediction.**

☞ A **prediction error plot** shows **the actual targets against the predicted values** generated by our model from the dataset. This allows to see **how much variance is in the model.**

In [31]:

--- Plot Feature Importance for Best Model ---

```
plot_model(best_models, plot = 'feature')
```



☞ In extra tree regressor, the **importance features for prediction** can be seen above.

☞ The most importance features from the best model is **organic type.**

☞ Below will **tuned this model** to get better performance.

--- Tuning Best Model ---

tuned_best = tune_model(best_models)

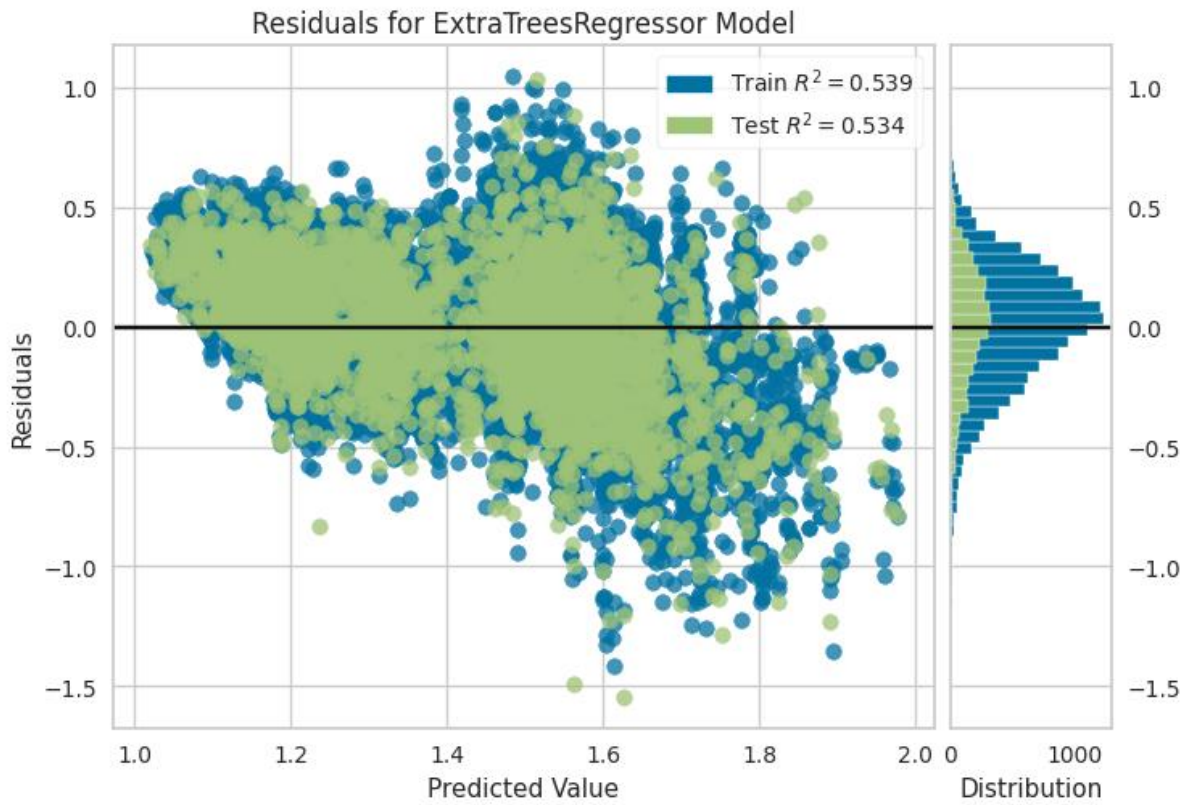
	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	0.2114	0.0771	0.2776	0.5397	0.1111	0.1613
1	0.2190	0.0795	0.2819	0.5396	0.1136	0.1669
2	0.2058	0.0687	0.2621	0.5445	0.1095	0.1645
3	0.2095	0.0756	0.2750	0.5403	0.1112	0.1618
4	0.2130	0.0738	0.2716	0.5492	0.1105	0.1645
5	0.2057	0.0702	0.2650	0.5364	0.1096	0.1634
6	0.2146	0.0762	0.2760	0.5246	0.1138	0.1699

7	0.2144	0.0826	0.2874	0.5130	0.1141	0.1612
8	0.2149	0.0776	0.2785	0.5082	0.1144	0.1688
9	0.2100	0.0746	0.2731	0.5395	0.1121	0.1664
Mean	0.2118	0.0756	0.2748	0.5335	0.1120	0.1649
Std	0.0040	0.0039	0.0071	0.0129	0.0018	0.0029

In [33]:

--- Plot the Residual of Tuned Best Model ---

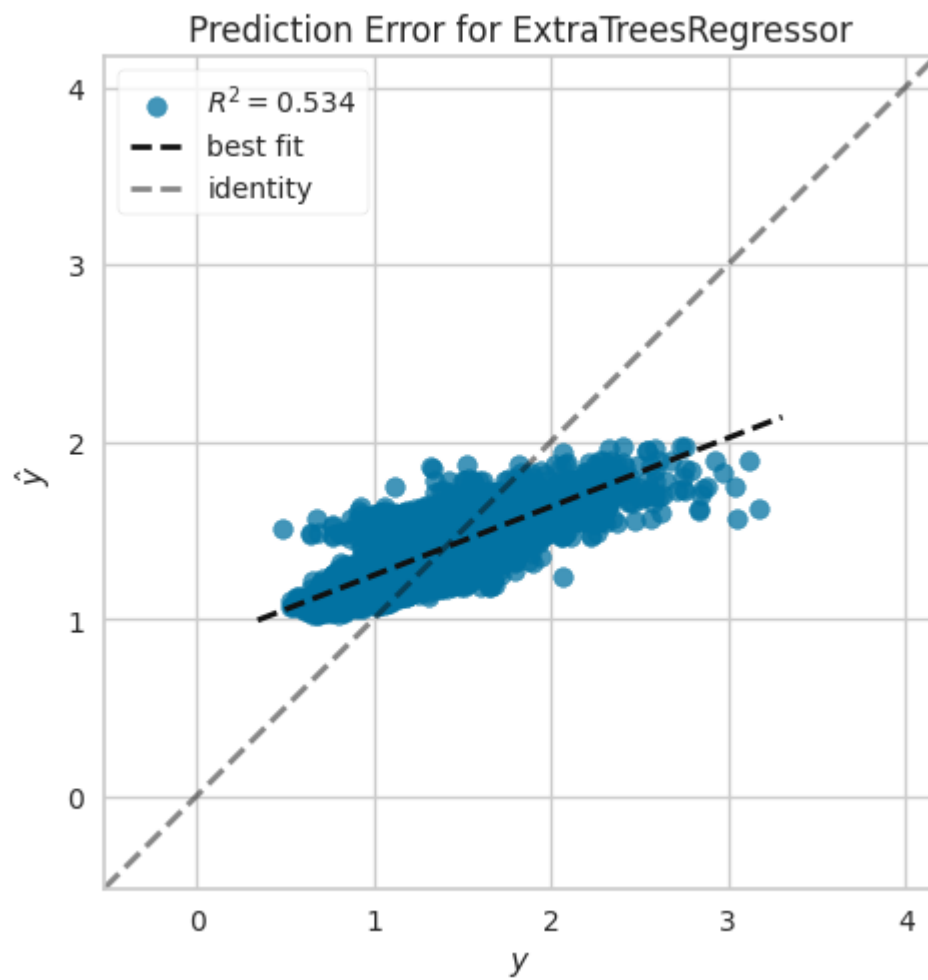
`plot_model(tuned_best)`



In [34]:

--- Plot Error Prediction for Tuned Best Model ---

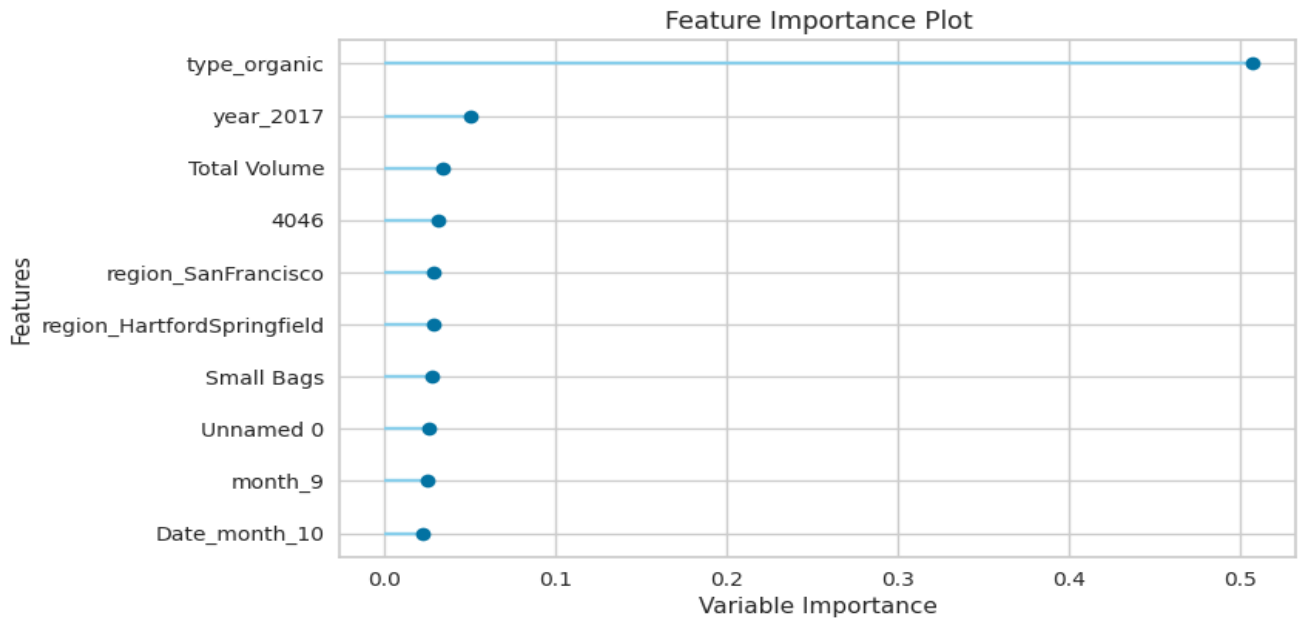
```
plot_model(tuned_best, plot = 'error')
```



In [35]:

--- Plot Feature Importance for Tuned Best Model ---

```
plot_model(tuned_best, plot = 'feature')
```

👉 After running the code to tuning the best model, the accuracy is **decreased until 0.5335** 🤔, which means PyCaret failed to optimize extra tree regressor.

👉 In the next section, will do experiments to create another models and then tuning it.

8.4 | Create Model 📄

👉 In this section, **two regression models** will be created, namely:

- Random Forest Regressor, and
- Light Gradient Boosting Machine

8.4.1 | Create Random Forest Regressor Model 📄 🌲

👉 This section will **create the Random Forest regressor model** first to see the early performance.

In [36]:

```
# --- Create RFR Model ---
```

```
rf = create_model('rf')
```

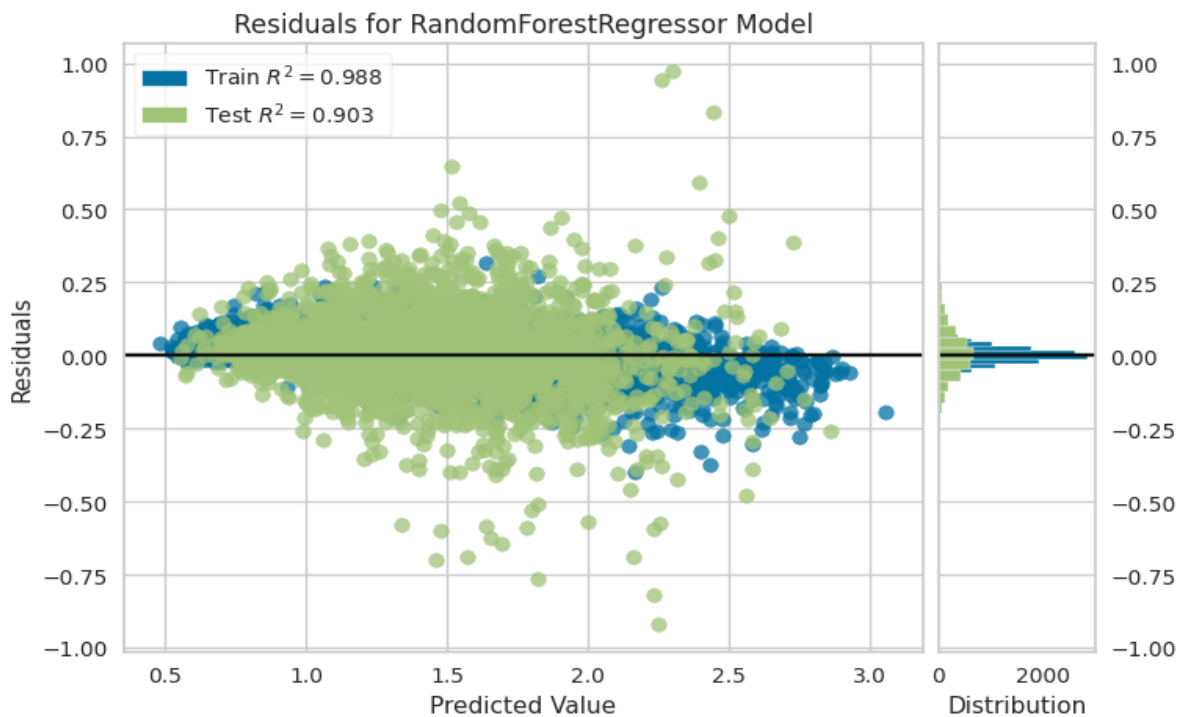
	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	0.0855	0.0153	0.1237	0.9086	0.0488	0.0625
1	0.0866	0.0144	0.1202	0.9164	0.0493	0.0645
2	0.0846	0.0145	0.1202	0.9041	0.0496	0.0638
3	0.0833	0.0145	0.1206	0.9116	0.0483	0.0617
4	0.0797	0.0126	0.1123	0.9230	0.0446	0.0584
5	0.0820	0.0140	0.1184	0.9075	0.0479	0.0611
6	0.0870	0.0164	0.1281	0.8976	0.0512	0.0645
7	0.0872	0.0160	0.1265	0.9057	0.0501	0.0635

8	0.0893	0.0164	0.1280	0.8961	0.0521	0.0669
9	0.0855	0.0157	0.1255	0.9028	0.0505	0.0643
Mean	0.0851	0.0150	0.1223	0.9073	0.0492	0.0631
Std	0.0027	0.0011	0.0047	0.0078	0.0020	0.0022

In [37]:

--- Plot the Residual of RFR Model ---

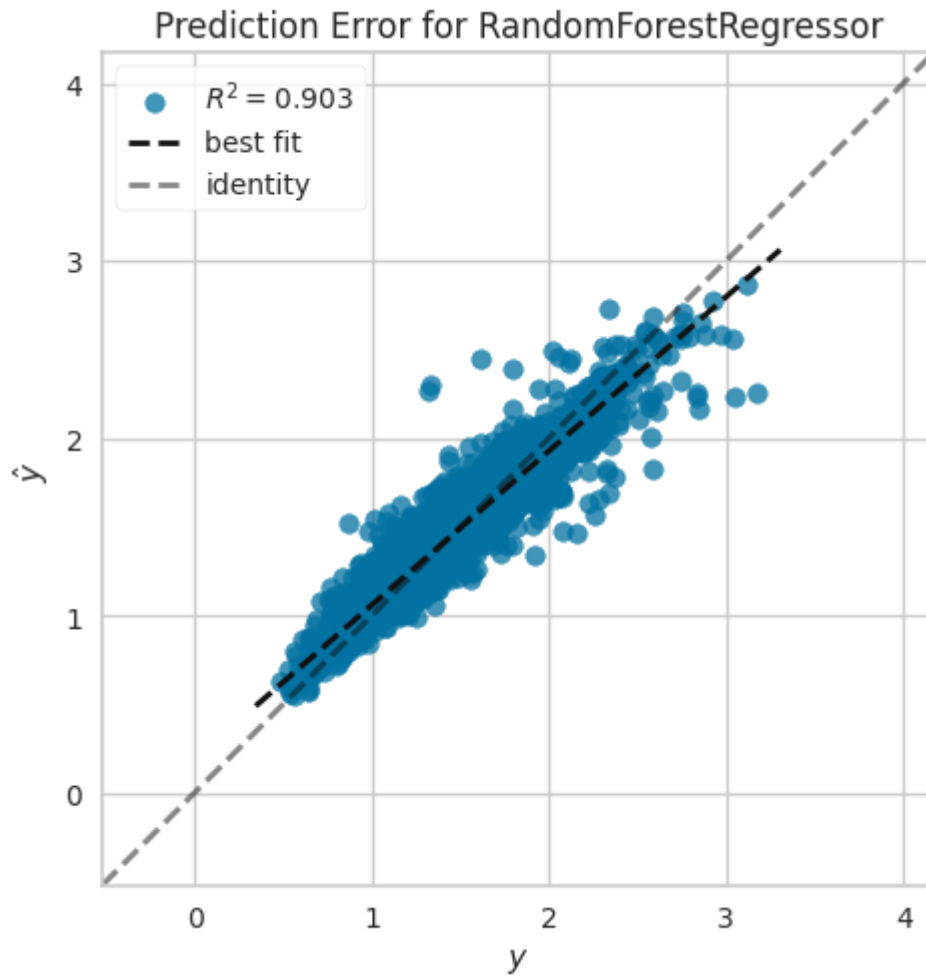
`plot_model(rf)`



In [38]:

--- Plot Error Prediction for RFR Model ---

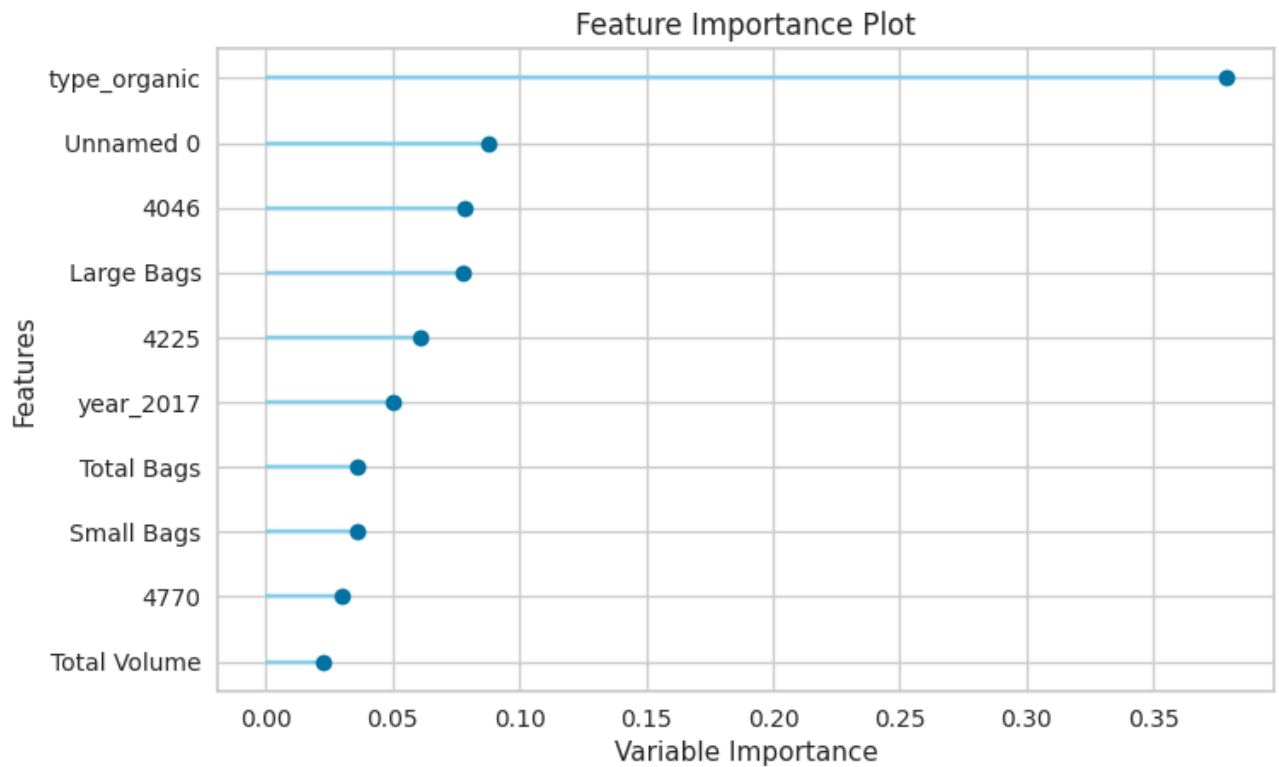
```
plot_model(rf, plot = 'error')
```



In [39]:

--- Plot Feature Importance for RFR Model --

```
plot_model(rf, plot = 'feature')
```



☞ From the plots, the RF model can achieve **0.988 R2 Score in train set** and **0.903 in test**, which indicates very good performance. However, **these numbers are still below the best model (extra tree regressor)**.

☞ There gap between the predicted values and the actual targets are **slightly bigger** compared to extra tree regressor.

☞ The importance features for RF regressor can be seen above. The **organic type of avocado still become the most importance features** in this model.

8.4.2 | Tuning Random Forest Regressor Model

☞ This section will do **tuning for Random Forest regressor** to achieve better results.

In [40]:

```
# --- Tuning RFR Model ---
```

```
tune_rf = tune_model(rf)
```

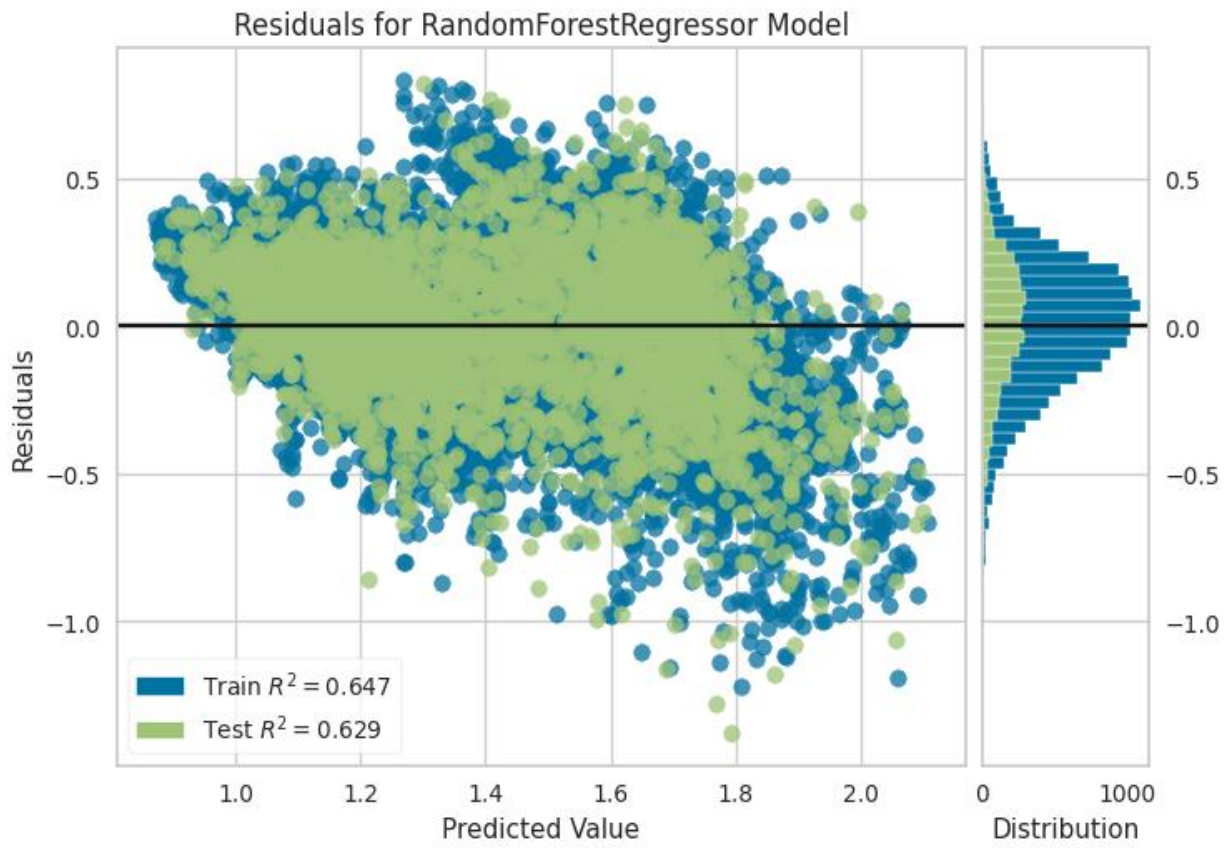
	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	0.1856	0.0597	0.2444	0.6432	0.0974	0.1393
1	0.1928	0.0613	0.2475	0.6451	0.0994	0.1449
2	0.1851	0.0547	0.2338	0.6375	0.0974	0.1450
3	0.1810	0.0557	0.2360	0.6615	0.0953	0.1378
4	0.1875	0.0576	0.2401	0.6480	0.0975	0.1429
5	0.1852	0.0564	0.2375	0.6277	0.0976	0.1435
6	0.1919	0.0613	0.2475	0.6177	0.1014	0.1490
7	0.1919	0.0650	0.2550	0.6167	0.1011	0.1423

8	0.1928	0.0617	0.2485	0.6087	0.1018	0.1490
9	0.1919	0.0614	0.2478	0.6211	0.1014	0.1498
Mean	0.1886	0.0595	0.2438	0.6327	0.0990	0.1443
Std	0.0040	0.0031	0.0064	0.0160	0.0022	0.0039

In [41]:

```
# --- Plot the Residual of RFR Model ---
```

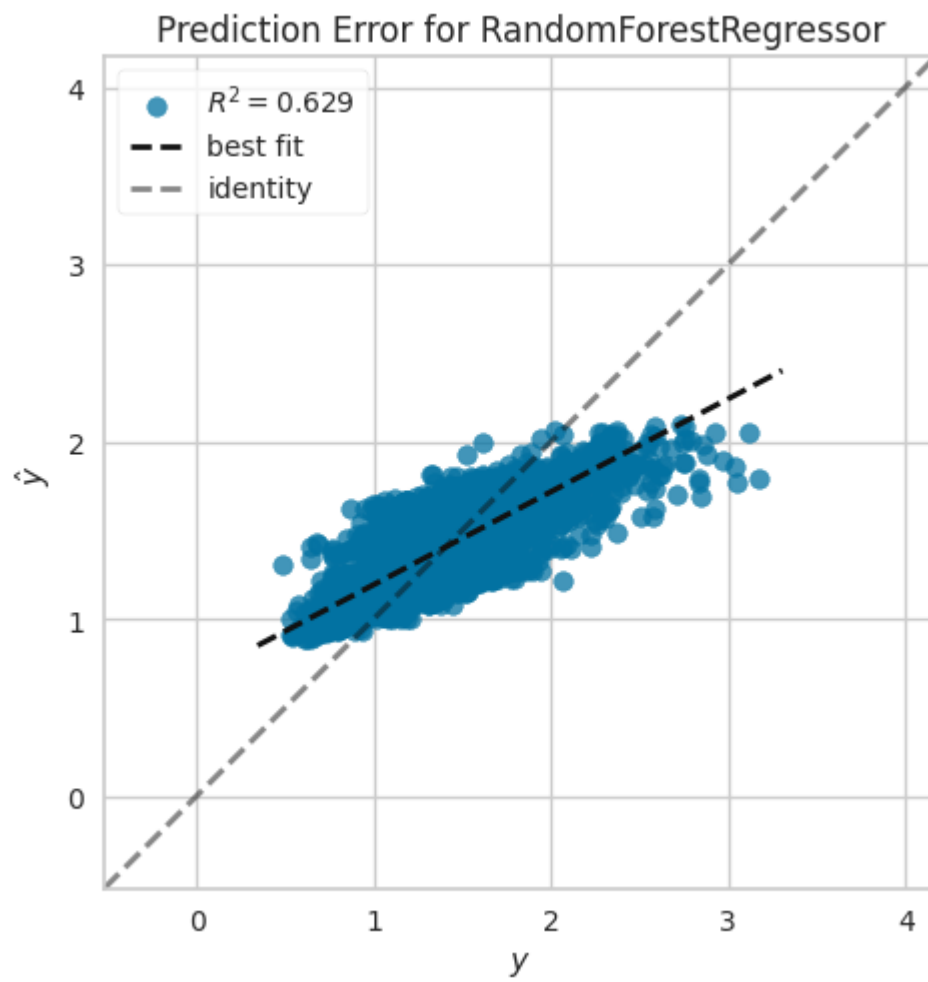
```
plot_model(tune_rf)
```



In [42]:

--- Plot Error Prediction for Tuned RFR Model ---

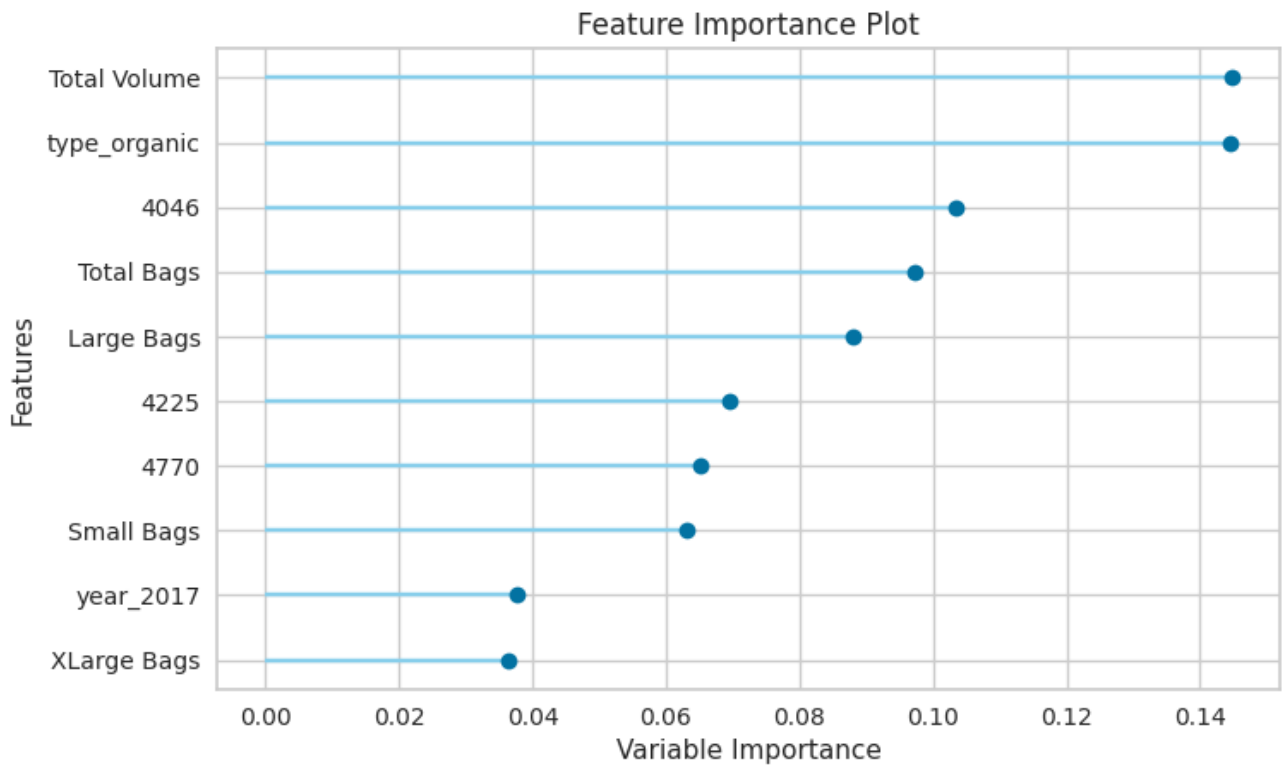
```
plot_model(tune_rf, plot = 'error')
```

In [43]:

--- Plot Feature Importance for Tuned RFR Model ---

```
plot_model(tune_rf, plot = 'feature')
```



➤ After running the code to tuning the RFR model, the accuracy is **decreased until 0.629** (PyCaret failed for the 2nd time to optimize RF regressor).

8.4.3 | Create Light Gradient Boosting Model

➤ The second model is **light gradient boosting**.

➤ This section will **create the light gradient boosting model** first to see the early performance.

In [44]:

```
# --- Create Light GBM ---
```

```
lgbm = create_model('lightgbm')
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
--	-----	-----	------	----	-------	------

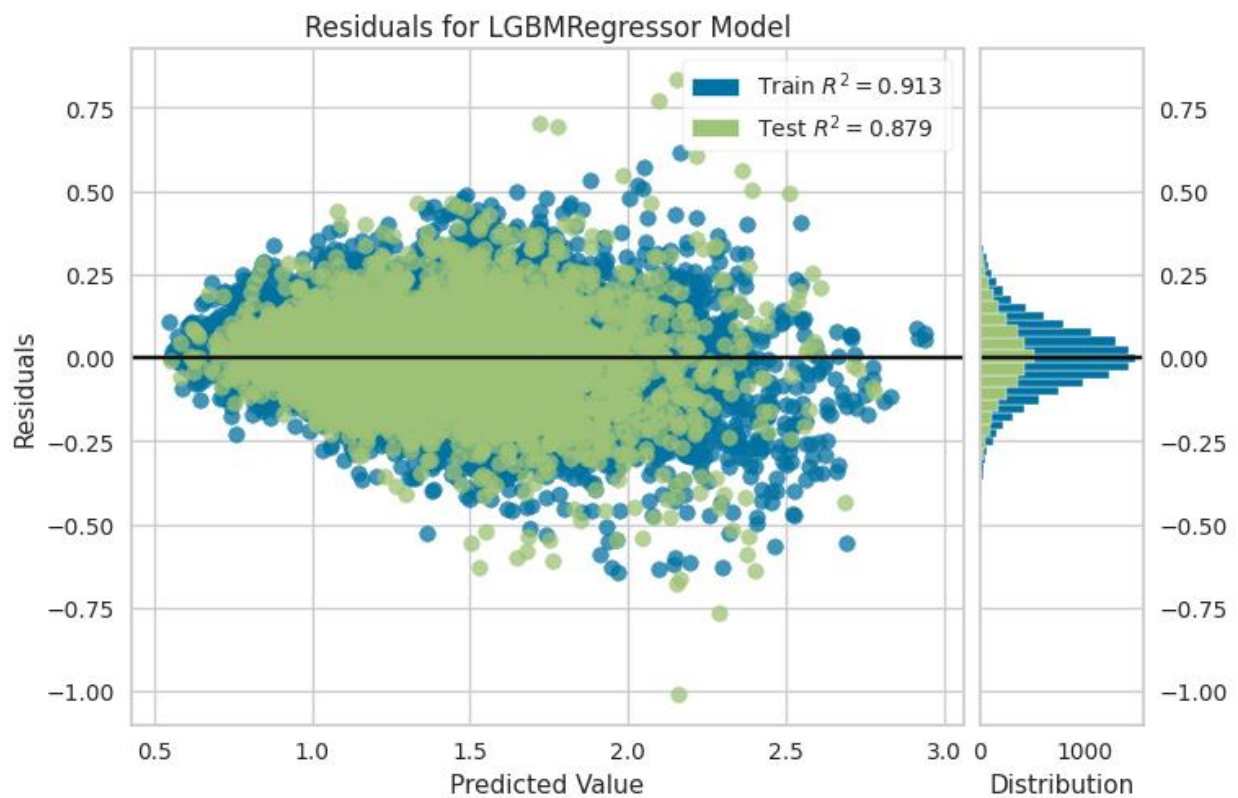
Fold						
0	0.0989	0.0178	0.1334	0.8936	0.0533	0.0725
1	0.1015	0.0177	0.1332	0.8972	0.0542	0.0750
2	0.0977	0.0164	0.1282	0.8911	0.0530	0.0740
3	0.0953	0.0171	0.1308	0.8961	0.0521	0.0695
4	0.0954	0.0161	0.1271	0.9014	0.0506	0.0698
5	0.0978	0.0173	0.1315	0.8859	0.0532	0.0721
6	0.1014	0.0183	0.1354	0.8856	0.0548	0.0752
7	0.1037	0.0205	0.1431	0.8794	0.0568	0.0753
8	0.0999	0.0181	0.1344	0.8855	0.0548	0.0747

9	0.0986	0.0182	0.1348	0.8879	0.0544	0.0739
Mean	0.0990	0.0178	0.1332	0.8904	0.0537	0.0732
Std	0.0025	0.0011	0.0042	0.0064	0.0016	0.0020

In [45]:

--- Plot the Residual of Light GBM ---

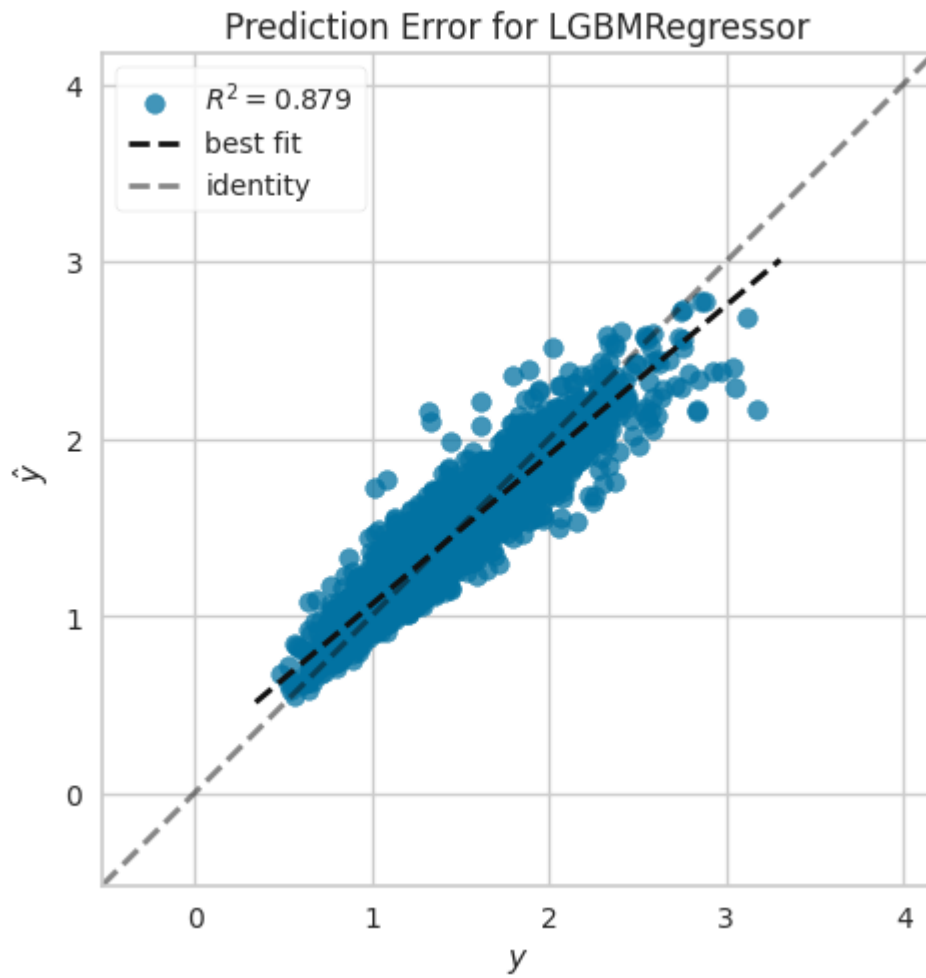
`plot_model(lgbm)`



In [46]:

--- Plot Error Prediction for Light GBM ---

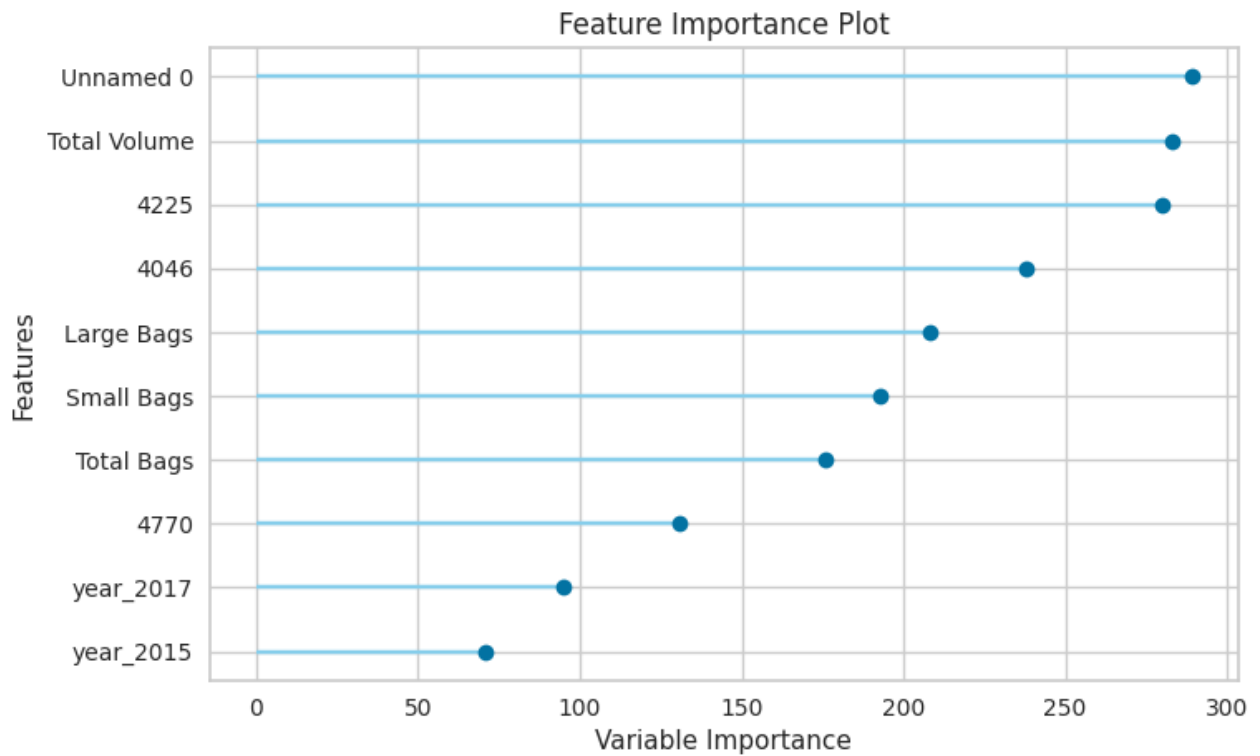
```
plot_model(lgbm, plot = 'error')
```



In [47]:

```
# --- Plot Feature Importance for Light GBM ---
```

```
plot_model(lgbm, plot = 'feature')
```



☞ From the plots, the light gradient boosting model can achieve **0.913 R2 score in train set** and **0.879 in test**. The numbers are **still not the best score compared to the best model**.

☞ The importance features for light gradient boosting can be seen above. The **total volume of avocado become the most importance features** for light gradient bosting model.

8.4.4 | Tuning Light Gradient Boosting Model

☞ This section will do **tuning for light gradient boosting** to achieve better results.

In [48]:

```
# --- Tuning Light Gradient Boost ---
```

```
tune_lgbm = tune_model(lgbm)
```

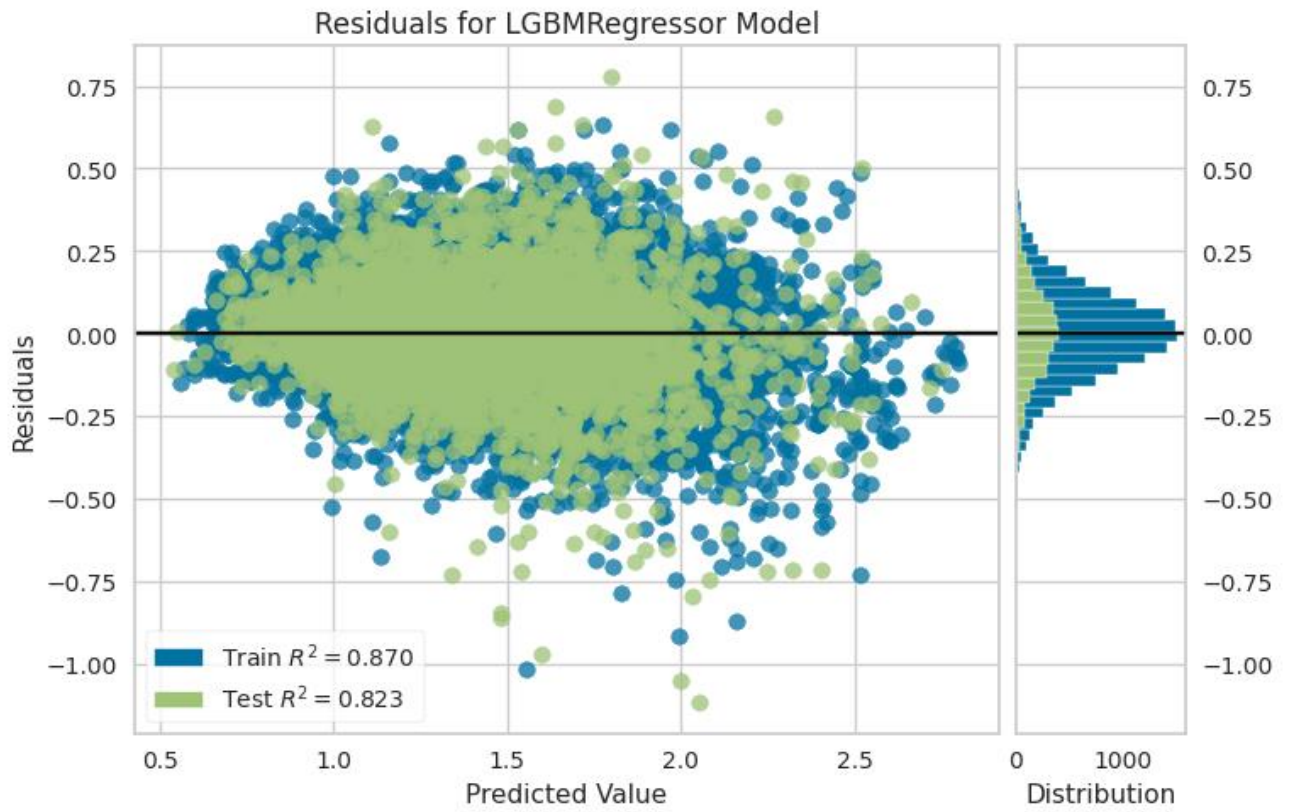
	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	0.1194	0.0264	0.1626	0.8421	0.0644	0.0872
1	0.1309	0.0291	0.1706	0.8315	0.0693	0.0973
2	0.1242	0.0270	0.1645	0.8207	0.0689	0.0962
3	0.1184	0.0265	0.1629	0.8386	0.0650	0.0874
4	0.1205	0.0256	0.1600	0.8435	0.0644	0.0892
5	0.1234	0.0267	0.1635	0.8235	0.0667	0.0924
6	0.1244	0.0266	0.1630	0.8343	0.0673	0.0950
7	0.1298	0.0310	0.1762	0.8171	0.0708	0.0961

8	0.1298	0.0295	0.1717	0.8132	0.0704	0.0981
9	0.1260	0.0280	0.1675	0.8269	0.0692	0.0967
Mean	0.1247	0.0277	0.1662	0.8291	0.0676	0.0936
Std	0.0042	0.0016	0.0048	0.0100	0.0023	0.0040

In [49]:

--- Plot the Residual of Tuned Light Gradient Boost ---

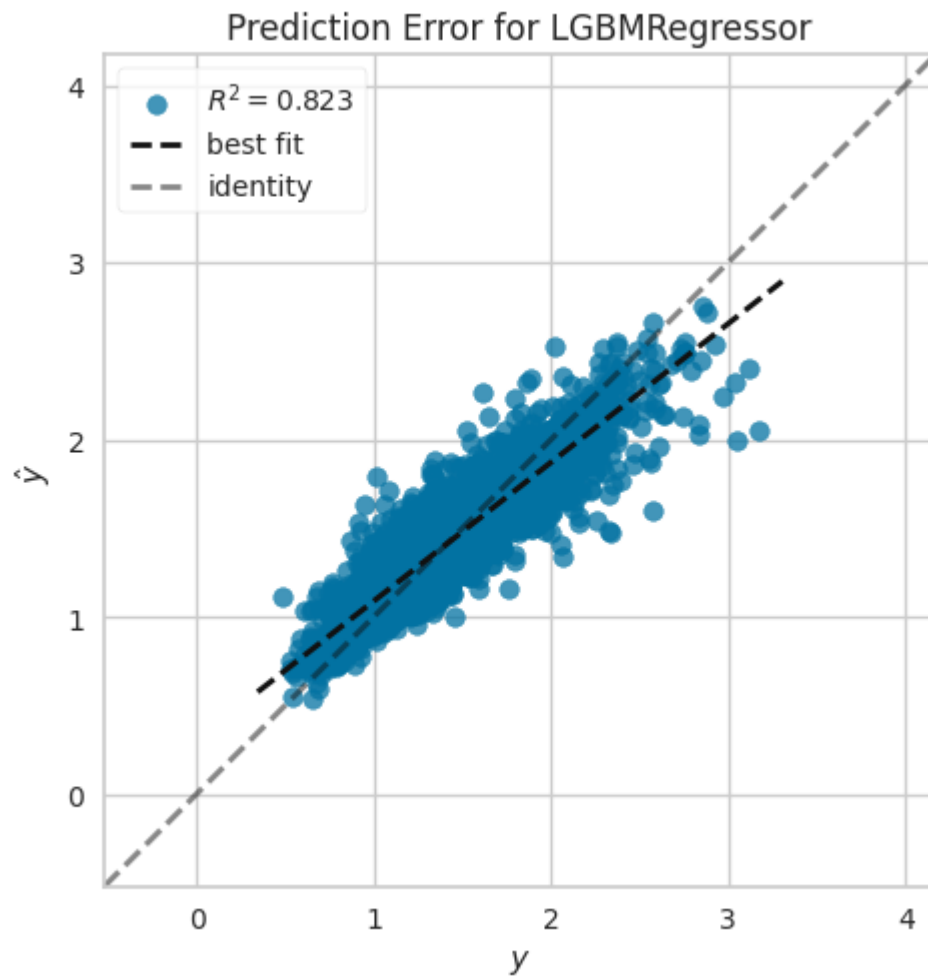
`plot_model(tune_lgbm)`



In [50]:

--- Plot Error Prediction for Tuned Light Gradient Boost ---

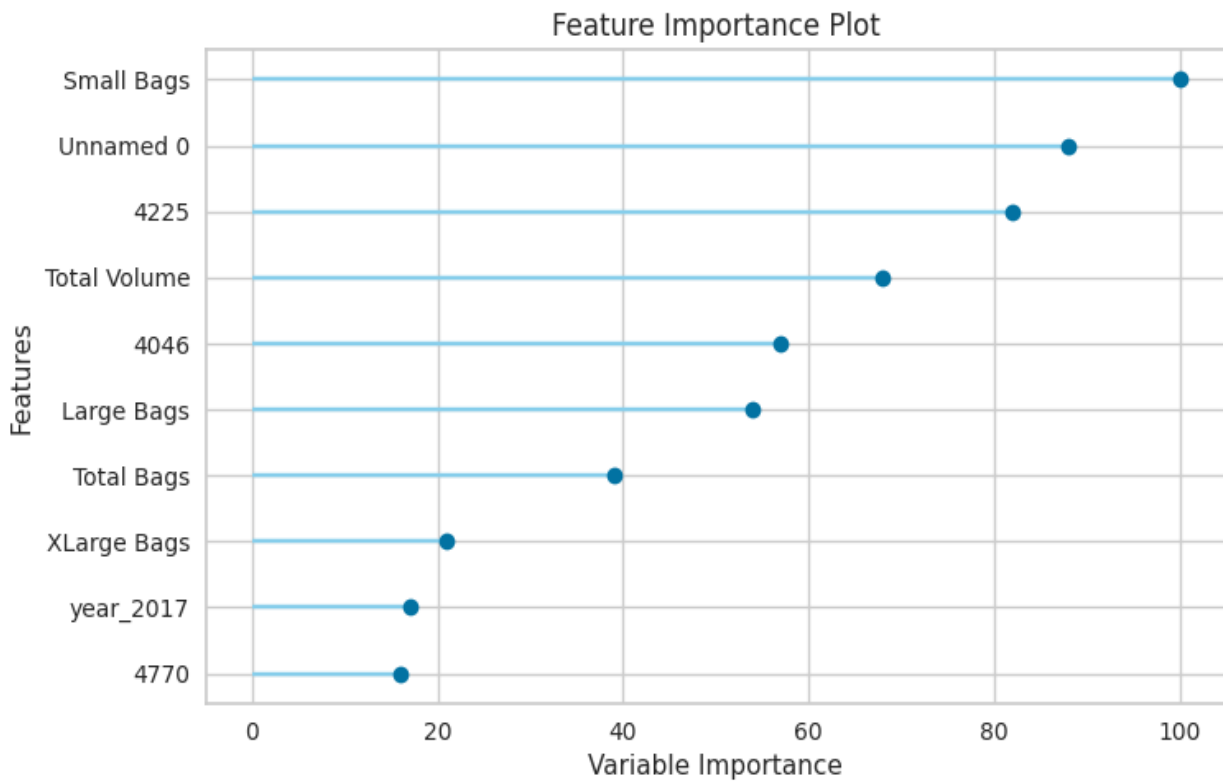
```
plot_model(tune_lgbm, plot = 'error')
```



In [51]:

--- Plot Feature Importance for Tuned Light Gradient Boost ---

`plot_model(tune_lgbm, plot = 'feature')`



☞ There is a **slight decrease for both train and test R2 score.**

☞ However, these numbers are still far from extra tree regressor.

8.4.5 | Comparison of Created Models

☞ Based on experiments from models creation above, below are the **summary of train and test R2 score**

in table form:

Model Name	Tuned/Not Tuned	R2 Score	
		Train	Test

Extra Tree Regressor	Not Tuned	1.000	0.925
	Tuned	0.539	0.534
Random Forest Regressor	Not Tuned	0.988	0.903
	Tuned	0.647	0.629
Light Gradient Boosting	Not Tuned	0.913	0.879
	Tuned	0.870	0.823

☞ From table above, it can be seen that **all tuned models accuracy (R2 score) are decreasing** 😞.

☞ **Extra tree regressor model still become the best model** since it has high accuracy in both train and test R2 score.

8.5 | Prediction on Test Sample 🧠

☞ This section will used the best experiments from three models to do a **prediction in test sample**. Those predictions are:

- **Normal** Extra tree regressor,
- **Normal** Random forest regressor, and
- **Normal** Light gradient boosting.

--- Prediction using Best Model ---

`predict_model(best_models)`

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extra Trees Regressor	0.0719	0.0122	0.1104	0.9247	0.0437	0.0529

Out[52]:

	Unnamed 0	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags
0	0.357143	1.770992	0.749676	1.618697	1.809093	4.169797	5.326726	1.068416	3.828886
1	-0.464286	-0.237191	-0.079796	-0.196736	-0.030364	-0.292676	-0.219610	-0.123732	0.000000
2	0.321429	0.082122	-0.049090	0.292166	-0.030364	0.257952	0.032864	1.636757	0.000000
3	0.035714	0.422099	0.783756	0.593626	2.410570	0.183641	0.400364	-0.098366	0.000000
4	0.285714	-0.237167	-0.079152	-0.188752	-0.030364	-0.304423	-0.301527	0.124106	0.000000

...
3645	0.964286	9.355392	5.424141	14.544580	87.213806	6.352094	5.082823	6.684702	1021.323364
3646	-0.357143	10.996881	15.420648	14.749743	43.922821	4.805529	6.212162	0.672725	40.000706
3647	0.964286	-0.010962	0.158737	-0.155775	-0.030364	0.301584	0.534609	-0.020226	0.000000
3648	-0.107143	0.014786	0.053533	-0.007713	-0.030107	0.304855	0.539354	-0.021873	0.000000
3649	0.071429	0.943723	1.711117	1.297896	3.435804	0.251382	0.494806	-0.119202	0.179307

3650 rows × 85 columns

In [53]:

--- Prediction using RFR Model ---

`predict_model(rf)`

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
--	-------	-----	-----	------	----	-------	------

0	Random Forest Regressor	0.0850	0.0157	0.1253	0.9031	0.0496	0.0624
---	-------------------------	--------	--------	--------	--------	--------	--------

Out[53]:

	Unnamed 0	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags
0	0.357143	1.770992	0.749676	1.618697	1.809093	4.169797	5.326726	1.068416	3.828886
1	-0.464286	-0.237191	-0.079796	-0.196736	-0.030364	-0.292676	-0.219610	-0.123732	0.000000
2	0.321429	0.082122	-0.049090	0.292166	-0.030364	0.257952	0.032864	1.636757	0.000000
3	0.035714	0.422099	0.783756	0.593626	2.410570	0.183641	0.400364	-0.098366	0.000000
4	0.285714	-0.237167	-0.079152	-0.188752	-0.030364	-0.304423	-0.301527	0.124106	0.000000
...
3645	0.964286	9.355392	5.424141	14.544580	87.213806	6.352094	5.082823	6.684702	1021.323364
3646	-0.357143	10.996881	15.420648	14.749743	43.922821	4.805529	6.212162	0.672725	40.000706

3647	0.964286	-0.010962	0.158737	-0.155775	-0.030364	0.301584	0.534609	-0.020226	0.000000
3648	-0.107143	0.014786	0.053533	-0.007713	-0.030107	0.304855	0.539354	-0.021873	0.000000
3649	0.071429	0.943723	1.711117	1.297896	3.435804	0.251382	0.494806	-0.119202	0.179307

3650 rows × 85 columns

In [54]:

--- Prediction using Light Gradient Boosting Model ---

`predict_model(lgbm)`

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Light Gradient Boosting Machine	0.1024	0.0197	0.1402	0.8786	0.0556	0.0747

Out[54]:

	Unnamed 0	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags
0	0.357143	1.770992	0.749676	1.618697	1.809093	4.169797	5.326726	1.068416	3.828886

1	-0.464286	-0.237191	-0.079796	-0.196736	-0.030364	-0.292676	-0.219610	-0.123732	0.000000
2	0.321429	0.082122	-0.049090	0.292166	-0.030364	0.257952	0.032864	1.636757	0.000000
3	0.035714	0.422099	0.783756	0.593626	2.410570	0.183641	0.400364	-0.098366	0.000000
4	0.285714	-0.237167	-0.079152	-0.188752	-0.030364	-0.304423	-0.301527	0.124106	0.000000
...
3645	0.964286	9.355392	5.424141	14.544580	87.213806	6.352094	5.082823	6.684702	1021.323364
3646	-0.357143	10.996881	15.420648	14.749743	43.922821	4.805529	6.212162	0.672725	40.000706
3647	0.964286	-0.010962	0.158737	-0.155775	-0.030364	0.301584	0.534609	-0.020226	0.000000
3648	-0.107143	0.014786	0.053533	-0.007713	-0.030107	0.304855	0.539354	-0.021873	0.000000
3649	0.071429	0.943723	1.711117	1.297896	3.435804	0.251382	0.494806	-0.119202	0.179307

3650 rows × 85 columns

8.6 | Finalize Model

☞ Since in the previous section mentioned that normal extra tree regressor has the best accuracy compared to other models and experiments, this section will **finalize extra tree regressor model, do prediction on the test sample, and save it to pickle file** (can be used for future production).

In [55]:

```
# --- Finalize Best Model ---
```

```
final_best = finalize_model(best_models)
```

```
# --- Final Best Model Parameters for Deployment ---
```

```
plot_model(best_models, plot='parameter')
```

	Parameters
bootstrap	False
ccp_alpha	0.0
criterion	mse
max_depth	None

max_features	auto
max_leaf_nodes	None
max_samples	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	-1

oob_score	False
random_state	123
verbose	0
warm_start	False

In [56]:

```
# --- Prediction using Final Model ---
```

```
predict_model(final_best)
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Extra Trees Regressor	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000

Out[56]:

	Unnamed 0	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags
0	0.357143	1.770992	0.749676	1.618697	1.809093	4.169797	5.326726	1.068416	3.828886

1	-0.464286	-0.237191	-0.079796	-0.196736	-0.030364	-0.292676	-0.219610	-0.123732	0.000000
2	0.321429	0.082122	-0.049090	0.292166	-0.030364	0.257952	0.032864	1.636757	0.000000
3	0.035714	0.422099	0.783756	0.593626	2.410570	0.183641	0.400364	-0.098366	0.000000
4	0.285714	-0.237167	-0.079152	-0.188752	-0.030364	-0.304423	-0.301527	0.124106	0.000000
...
3645	0.964286	9.355392	5.424141	14.544580	87.213806	6.352094	5.082823	6.684702	1021.323364
3646	-0.357143	10.996881	15.420648	14.749743	43.922821	4.805529	6.212162	0.672725	40.000706
3647	0.964286	-0.010962	0.158737	-0.155775	-0.030364	0.301584	0.534609	-0.020226	0.000000
3648	-0.107143	0.014786	0.053533	-0.007713	-0.030107	0.304855	0.539354	-0.021873	0.000000
3649	0.071429	0.943723	1.711117	1.297896	3.435804	0.251382	0.494806	-0.119202	0.179307

3650 rows × 85 columns

```
# --- Save Final Model into Pickle File ---
```

```
save_model(final_best, 'Final_Best_Model_caesarmario_06May2022')
```

Transformation Pipeline and Model Successfully Saved

Out[57]:




```
(Pipeline(memory=None,  
  steps=[('dtypes',  
    DataTypes_Auto_infer(categorical_features=['type', 'year',  
      'region', 'month'],  
        display_types=False, features_todrop=[],  
        id_columns=[], ml_usecase='regression',  
        numerical_features=[],  
        target='AveragePrice',  
        time_features=[])),  
  ('imputer',  
    Simple_Imputer(categorical_strategy='not_available',  
      fill_value_categorical=None,  
      fill_v...  
    ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,  
      criterion='mse', max_depth=None,  
      max_features='auto', max_leaf_nodes=None,  
      max_samples=None,
```

```
min_impurity_decrease=0.0,  
  
min_impurity_split=None,  
  
min_samples_leaf=1, min_samples_split=2,  
  
min_weight_fraction_leaf=0.0,  
  
n_estimators=100, n_jobs=-1,  
  
oob_score=False, random_state=123,  
  
verbose=0, warm_start=False)]],
```

```
verbose=False),
```

```
'Final_Best_Model_caesarmario_06May2022.pkl')
```

9. | References

- **Kaggle Notebook**  What Visualizations Should You Use? by Vivek Chowdhury
- How to Make Your Viz' Stand Out by Dimitri Irfan
- Statistical Avo: EDA, Analysis and ML by Jaime Becerra Guerrero
- Price of Avocados || Pattern Recognition Analysis by Janio Martinez Bachmann
- **GitHub Tutorial**  Demo 2 - Time Series.ipynb by PyCaret
- PyCaret Regression FIFA Market Value by Ojaas Hampiholi
- **Website**  PyCaret documentation - Regression
- Regression Tutorial (REG101) - Level Beginner

 Like this notebook? You can support me by giving **upvote**   

More about myself: linktr.ee/caesarmario_

