Prediction of Avocado Prices using Machine Learning

By Qiangjing Liang, Elouise Matthews, Luis Sejas, Bingqi Tang

21/12/2020

Table of Contents

Abstract

The selling price of avocado is potentially affected by various factors, such as the number of bags sold for different bag sizes, size of avocados, date, and region. To build a model for the prediction of the selling price, different machine learning *methods coded in Python were applied to a dataset regarding avocado prices in the U.S. from 2015 to 2018. To prepare the data for these machine learning methods, common approaches such as data cleaning and one-hot encoding were used to drop highly correlated features and convert data types.*

Before training machine learning models, a performance metric was selected for this project. After considering the variance-bias tradeoff, the Coefficient of Determination (R^2) was chosen.

After applying Python's various modelling packages to the data, the results returned showed that the methods of stacking and bagging had the highest R^2 for the test set—0.94 and 0.943 respectively. Both methods were ensemble methods, and both were a combination of Gradient Boosting and Random Forest models and Support Vector Machine.

While these results were already sufficient to call the project a success, further investigations were made regarding alternative ways of utilizing the data. More modelling was performed to analyze the effect of dropping correlated features and deleting outliers on the tree models used, and to explore other potential methods of handling the variable of region better, such as using alternative classifications for the regions and using proxy variables.

I. Introduction

This report is a detailed explanation of the project Prediction of Avocado Prices, in which the task was to build a machine learning model for a dataset containing the selling price of avocados at specified locations on specified days in the years 2015 to 2018. Additional information was provided regarding bag types and avocado types. The model is to treat all columns beside the price as "features" and the price column as the "target variable".

The following sections will discuss the initial preparation of data such as cleaning and modifying the data, and exploratory data analysis applied to investigate further handling of the data. Furthermore, the benefits of different performance metrics, namely the Mean Square Error, Mean Absolute Error and Coefficient of Determination, will be analyzed and compare to find out which is best suited to the problem at hand.

After the preparatory processes, machine learning models will be applied. The data is first split into training and test data. The training data is then fed to the algorithm to build a model, which is later applied to the test set. Using the evaluation metric of choice, which is R^2, the final results of different model types will be compared and discussed to find out which model is the most applicable. The discussion of result will be followed by a section of further investigation, which will evaluate some potential improvements.

II. Import necessary libraries and import data

The following python packages were first imported, and the data was read from a given csv file:

```
>> import os
>> import pandas as pd
>> import numpy as np
>> from sklearn.preprocessing import LabelBinarizer
>> import seaborn as sns
>> import matplotlib.pyplot as plt
>> from scipy.stats import anderson
>> from sklearn.model_selection import train_test_split

>> data_ori = pd.read_csv('AvocadoPrices.csv')
```

III. Data Preparation

i. Overview of Data

In this section, the data is inspected regarding its data types etc.:

```
>> print(data_ori.head(5))
```

```
  Unnamed: 0   Date     AveragePrice  ...      type          year    region
0   0       12/27/2015     1.33       ...     conventional   2015    Albany
1   1       12/20/2015     1.35       ...     conventional   2015    Albany
2   2       12/13/2015     0.93       ...     conventional   2015    Albany
3   3       12/6/2015      1.08       ...     conventional   2015    Albany
4   4       11/29/2015     1.28       ...     conventional   2015    Albany
```

From the first look, the column "Unnamed: 0" is only an index column, which apparently isn't a contributing factor to the avocado prices and thus should be removed later.

Meanwhile, it is clear that the target variable is stored in the column "AveragePrice".

Another observation regarding the "Date" and "year" variables can also be made: the year information stored in the "Date" variable seems to be the same as the "year" variable by inspecting the first 5 rows.

```
>> print(data_ori.describe())
```

```
      Unnamed: 0      AveragePrice ...   XLarge Bags     year
count 18249.000000    18249.000000 ...   18249.000000    18249.000000
mean    24.232232     1.405978     ...   3106.429119     2016.147899
std     15.481045     0.402677     ...   17692.899701    0.939938
min     0.000000      0.440000     ...   0.000000        2015.000000
25%     10.000000     1.100000     ...   0.000000        2015.000000
50%     24.000000     1.370000     ...   0.000000        2016.000000
75%     38.000000     1.660000     ...   133.000000      2017.000000
max     52.000000     3.250000     ...   551694.000000   2018.000000
```

The numerical analysis of the target variable reveals that its distribution is not symmetrical and is skewed to the right: compare to the differences between 1st and the 2nd quantiles (1.37-1.1=0.27), as well as between the 2nd and the 3rd quantiles (1.66-1.37=0.29) of "AveragePrice", both of which are less than 0.3 and relatively small, the difference between the 3rd and the 4th quantiles (3.25-1.66=1.59) is much higher.

Meanwhile, a similar observation can be made regarding the column "XLarge Bags", which shows a much more extreme skewness.

```
>> print(data_ori.dtypes)
```

```
Unnamed: 0      int64
Date            object
AveragePrice    float64
Total Volume    float64
4046            float64
4225            float64
4770            float64
Total Bags      int64
Small Bags      int64
Large Bags      int64
XLarge Bags     int64
type            object
year            int64
region          object
```

From the summary regarding the datatypes of different columns, it's worth noticing that only 3 of the variables are of the "object" type, namely the "Date", the "type", and the "region" columns, which indicate the need of further preprocessing for type conversion.

ii. Scan for Missing values

A quick scan was done to check for any NaN in the dataset:

```
>> print(data_ori.isnull().sum())

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
```

There are no NaN values in the dataset.

iii. Extract the month data from the "Date" variable

First, it was examined that whether the year information stored in the "Date" variable is the same as the value of the "year" variable:

```
>> extract_year = pd.DataFrame()
>> extract_year['Year'] = pd.DatetimeIndex(data_ori['Date']).year
>> extract_year['checkYear'] = np.where(extract_year['Year'] == data_ori['year'],
1, 0)
>> print("The number of non-matching (year information) records in the two
variables is", sum(extract_year['checkYear'] == 0))

The number of non-matching (year information) records in the two variables is 0
```

As they are consistent, the year information stored in the "Date" variable is redundant and should be removed. Meanwhile, since the exact date information is unlikely to contribute to the prediction of avocado prices, only the month information within the "Date" variable should be extracted:

```
>> data_rpdate = data_ori
>> data_rpdate['month'] = pd.DatetimeIndex(data_ori['Date']).month
>> data_rpdate = data_rpdate.drop('Date', axis = 1)
```

iv. Convert categorical values

a. Convert the "type" variable

First, the number of categories within the "type" variable was inspected:

```
>> print(data_rpdate['type'].value_counts())

conventional 9126
organic 9123
```

Since the "type" variable only has two categories, it can be converted using binary encoding:

```
>> data_rpdate['type'] = np.where(data_rpdate['type']=='conventional',1,0)
```

b. Convert the "region" variable

First, the number of categories within the "region" variable was inspected:

```
>> print(data_rpdate['region'].value_counts())
Syracuse               338
Nashville              338
Indianapolis           338
GreatLakes             338
Jacksonville           338
…
California             338
HarrisburgScranton     338
NewOrleansMobile       338
LasVegas               338
SanFrancisco           338
WestTexNewMexico       335
```

As there are 54 categories in the variable, only the first and the last few lines of the output are displayed.

Since there are more than 2 categories in this column (thus binary encoding can't be used), and that the variable is not continuous (thus label encoding shouldn't be used), the "region" variable was converted using one hot encoding:

```
>> region_names = pd.DataFrame(region_cat.index.tolist(), columns=['Region'])
>> region_names = region_names.sort_values('Region')
>> bi_Region = LabelBinarizer()
>> bi_dummys = bi_Region.fit_transform(data_rpdate['region'])
>> bi_dummys = pd.DataFrame(bi_dummys,
columns=region_names['Region'].tolist())
```

To avoid the dummy variable trap, the first column was dropped, and the rest were then concatenated with the other variables of the dataset:

```
>> bi_dummys = bi_dummys.drop('Albany', axis = 1)
>> data_reg = data_rpdate.drop('region', axis = 1)
>> data_reg = pd.concat([data_reg, bi_dummys], axis = 1)
```

v. Drop Irrelevant Variables

As shown in Section III(i), the first column "Unnamed: 0" is not a relevant variable and thus should be dropped:

```
>> data_reg = data_reg.drop('Unnamed: 0', axis = 1)
```

IV. Exploratory Data Analysis (EDA)

i. Non-graphical EDA

To analyze the data numerically without graphs, the same process shown in Section III(i) can be performed, i.e.:

```
>> print(data_ori.describe())
```

```
       Unnamed: 0      AveragePrice ...   XLarge Bags      year
count   18249.000000   18249.000000 ...   18249.000000   18249.000000
mean    24.232232      1.405978     ...   3106.429119    2016.147899
std     15.481045      0.402677     ...   17692.899701   0.939938
min     0.000000       0.440000     ...   0.000000       2015.000000
25%     10.000000      1.100000     ...   0.000000       2015.000000
50%     24.000000      1.370000     ...   0.000000       2016.000000
75%     38.000000      1.660000     ...   133.000000     2017.000000
max     52.000000      3.250000     ...   551694.000000  2018.000000
```

ii. Graphical EDA

a. Numerical values: pairplot, boxplot, and correlation matrix

To help create different graphs, all the numerical values of the dataset were first extracted:

```
>> numericals = pd.concat([data_reg.iloc[:, 0:9], data_reg.iloc[:, 10:12]], axis=1)
```

To gain an overview of these numerical variables, pairplots can be created:

```
>> sns.set(font_scale = 1.7)
>> sns.set_style("white")
>> sns.pairplot(numericals)
>> plt.show()
```

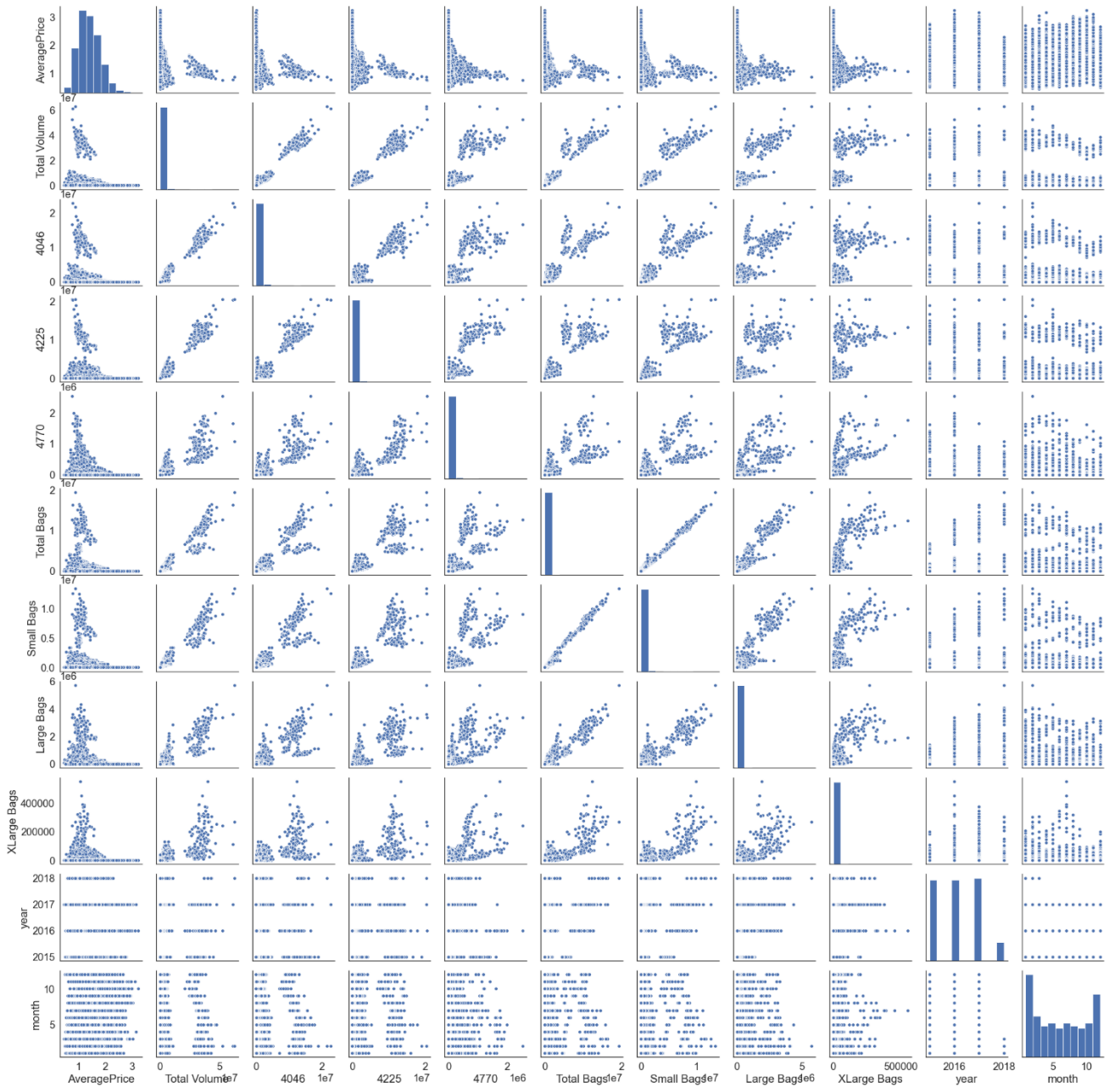

Figure 1: Pairplots of all numerical variables of the dataset

Distribution graphs, which are on the diagonal of the pairplots in Figure 1, are first inspected. As expected, the distribution of the target variable "AveragePrice", which is shown in the first diagonal graph, is skewed

to the right. Meanwhile, apart from the numerical variables “year” and “month”, more extreme skewness is observed in the other 8 numerical variables (e.g. “4046”, “Small Bags”, etc.)

From the other part of Figure 1, which are scatterplots of different pairs of the numerical values, it can be observed that a few of them have indicated high correlations between certain pairs, such as between the variables “4046” and “Total Volume”.

To further explore the distributions of the numerical values, a boxplot was created using the normalized data of the numerical values:

```
>> numericals_nm = (numericals-numericals.min())/(numericals.max()-
numericals.min())
>> sns.set_style("white")
>> plt.xticks(rotation = 90)
>> sns.boxplot(data=numericals_nm, orient='v', palette='Set2')
>> plt.show()
```

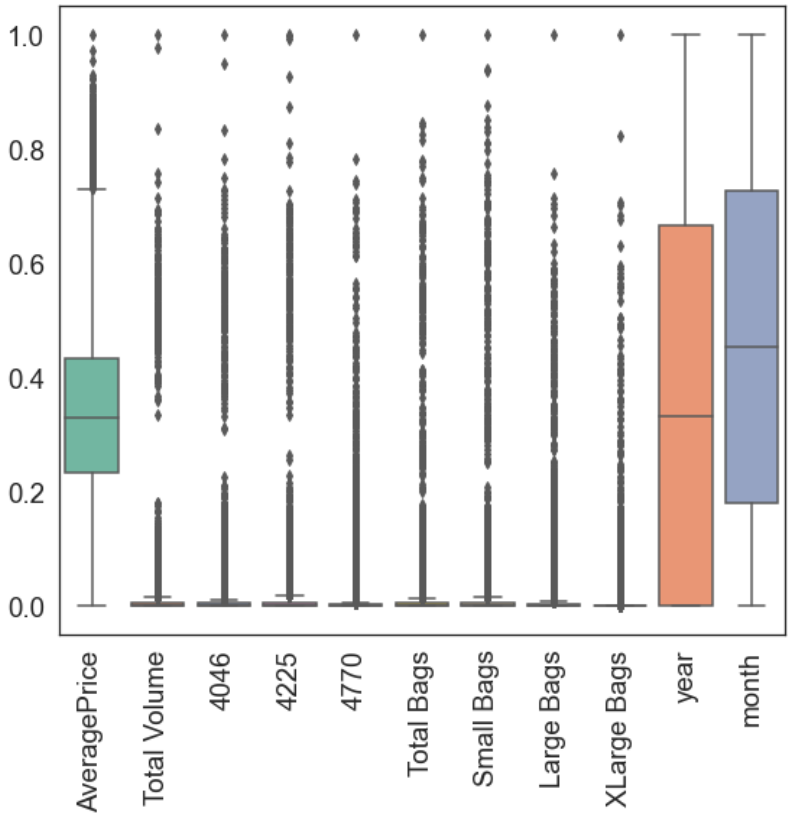

Figure 2: Boxplot of all numerical variables of the dataset after being normalized

As shown in Figure 2, all outliers of the “AveragePrice” variable are in the upper range, which corresponds with the above conclusion that its distribution is skewed to the right. Meanwhile, the variables that showed more extreme skewness in the pairplot are also displaying much more outliers in the boxplot, making their “boxes” almost invisible.

To further explore the correlations between different numerical variable pairs, a heatmap of their correlation matrix was made:

```
>> sns.set(rc={'figure.figsize':(9,8)})
>> corr_matrix = numericals.corr()
>> sns.heatmap(corr_matrix, cmap="mako", annot=True)
>> plt.show()
```

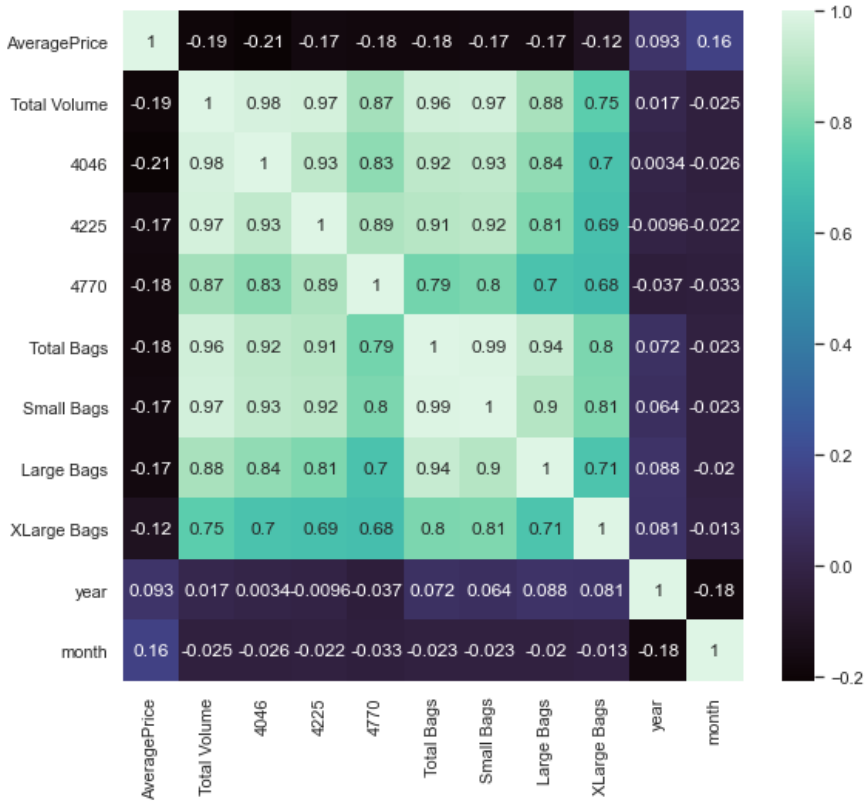

Figure 3: Heatmap of all numerical variables of the dataset

As shown in Figure 3, the variables "Total Volume" and "Total Bags" have much higher correlations with the others. This is reasonable, considering that these two variables are the aggregated amount of the others (e.g., "Total Volume" is the sum of "4046", "4225", and "4770").

b. Categorical values: bar plot

For the two columns with categorical values, bar graphs were used for visualization:

```
>> sns.set(font_scale = 1.5)
>> sns.set_style("white")
>> sns.countplot(x ="type", data=data_ori)
>> plt.show()
```

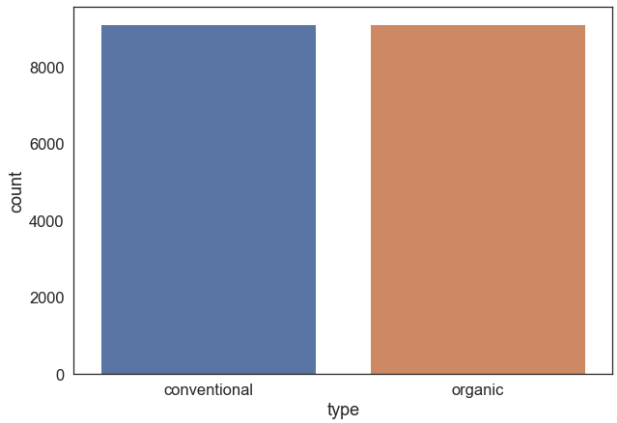

Figure 4: Bar plot of the variable "type"

```
>> plt.xticks(rotation=90)
>> sns.set(rc={'figure.figsize':(15,7)})
>> sns.countplot(x="region", data=data_ori)
>> plt.show()
```

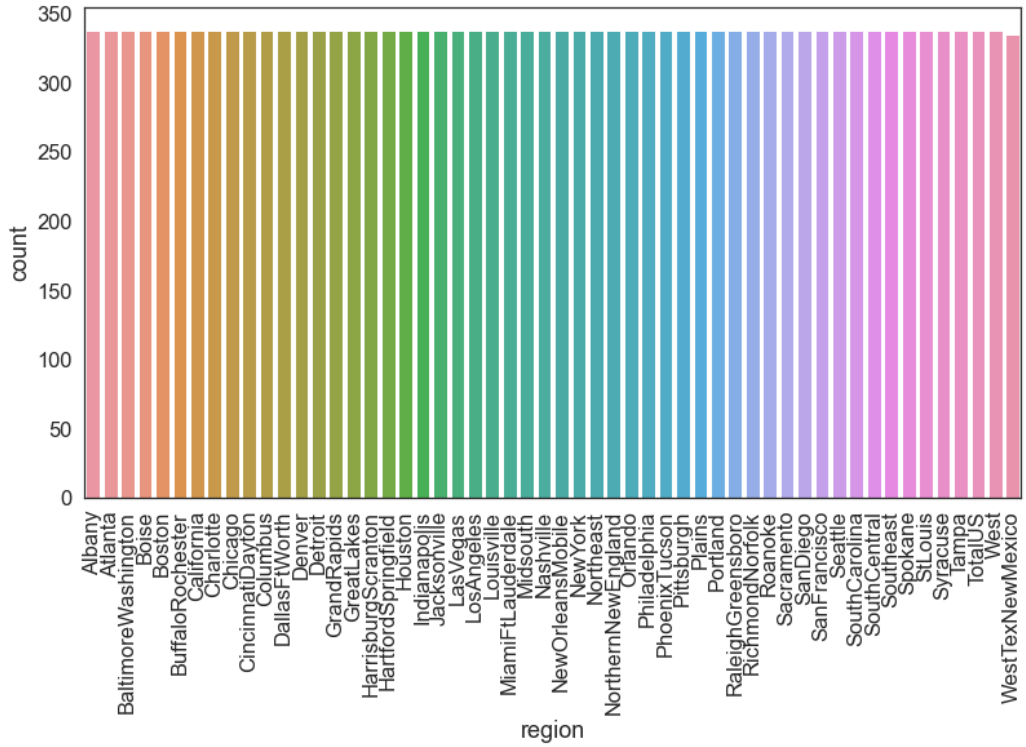

Figure 5: Bar plot of the variable "region"

From Figure 4 and Figure 5, it can be observed that both these variables have a balanced distribution among their individual categories.

iii. Handling Outliers

a. Delete outliers of target variable "AveragePrice"

As shown in the previous section, the variable "AveragePrice" is skewed to the right. A histogram was made to investigate the variable in more detail as shown in Figure 6:

```
>> plt.rcParams["figure.figsize"] = (8,5)
>> plt.hist(data=data_ori, x='AveragePrice', bins=30)
>> plt.xlabel("AveragePrice")
>> plt.ylabel("Count")
>> plt.show()
```

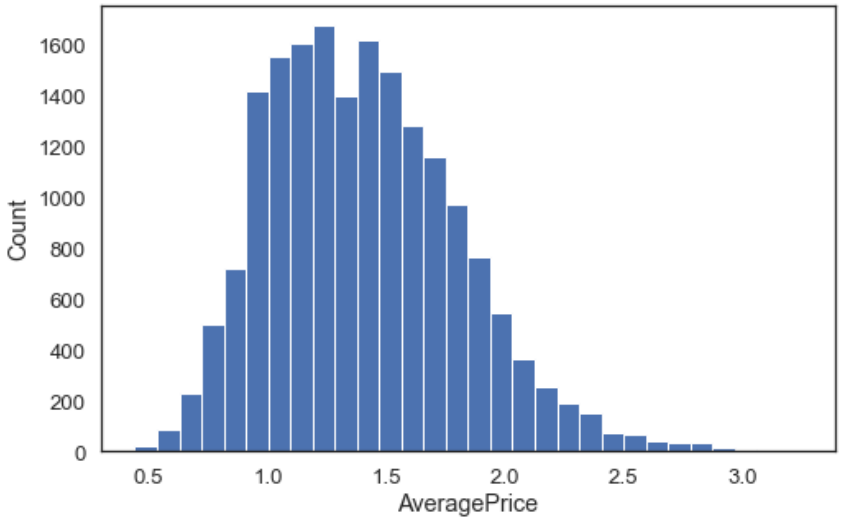

Figure 6: Histogram of the variable "AveragePrice" using its original value

In order to limit the effect of outliers on the model outputs and thus make the predictions more generalized, the top 5% of the target variable was dropped:

```
>> Price_95 = np.quantile(data_reg['AveragePrice'],0.95)
>> outlier_index = data_reg[data_reg['AveragePrice'] > Price_95].index
>> data_reg.drop(outlier_index, inplace=True)
```

After dropping the outliers, an Anderson-Darling test, which is used to test the degree of being normally distributed, was carried out:

```
>> print(anderson(data_reg['AveragePrice']))

AndersonResult(statistic=61.86218998759432, critical_values=array([0.576,
0.656, 0.787, 0.918, 1.092]), significance_level=array([15. , 10. ,  5. ,
2.5,  1. ]))
```

With a p-value over 61, it can be concluded that the resulting target variable (after removing the top 5%) can be considered normally distributed. This can also be observed in the histogram of the variable "AveragePrice" after the outlier removal as shown in Figure 7:

```
>> plt.rcParams["figure.figsize"] = (8,5)
>> plt.hist(data=data_reg, x='AveragePrice', bins=30)
>> plt.xlabel("AveragePrice")
>> plt.ylabel("Count")
>> plt.show()
```

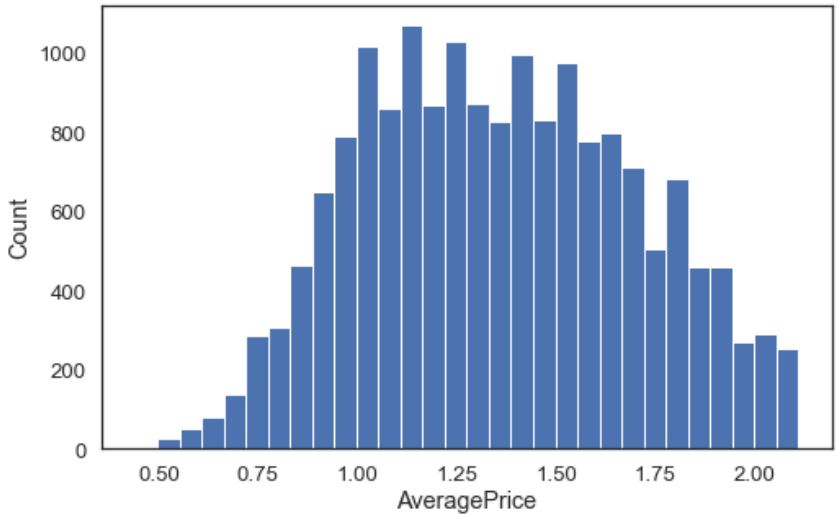

Figure 7: Histogram of the variable "AveragePrice" after removing the top 5%

b. Log transform the highly skewed variables

As shown in the previous sections, some numerical variables, such as "4046", "Total Bags", and "Small Bags", have an extremely high skewness. Upon further inspection of the data, their abnormal distributions are caused by the differences among the categories in the "region" variable. To further investigate, a scatterplot of the "4046" and "region" variables was made:

```
>> sns.set(rc={'figure.figsize':(10,12)})
>> sns.set(font_scale = 1.2)
>> sns.set_style("white")
>> sns.scatterplot(data=data_ori, x='4046', y='region')
```

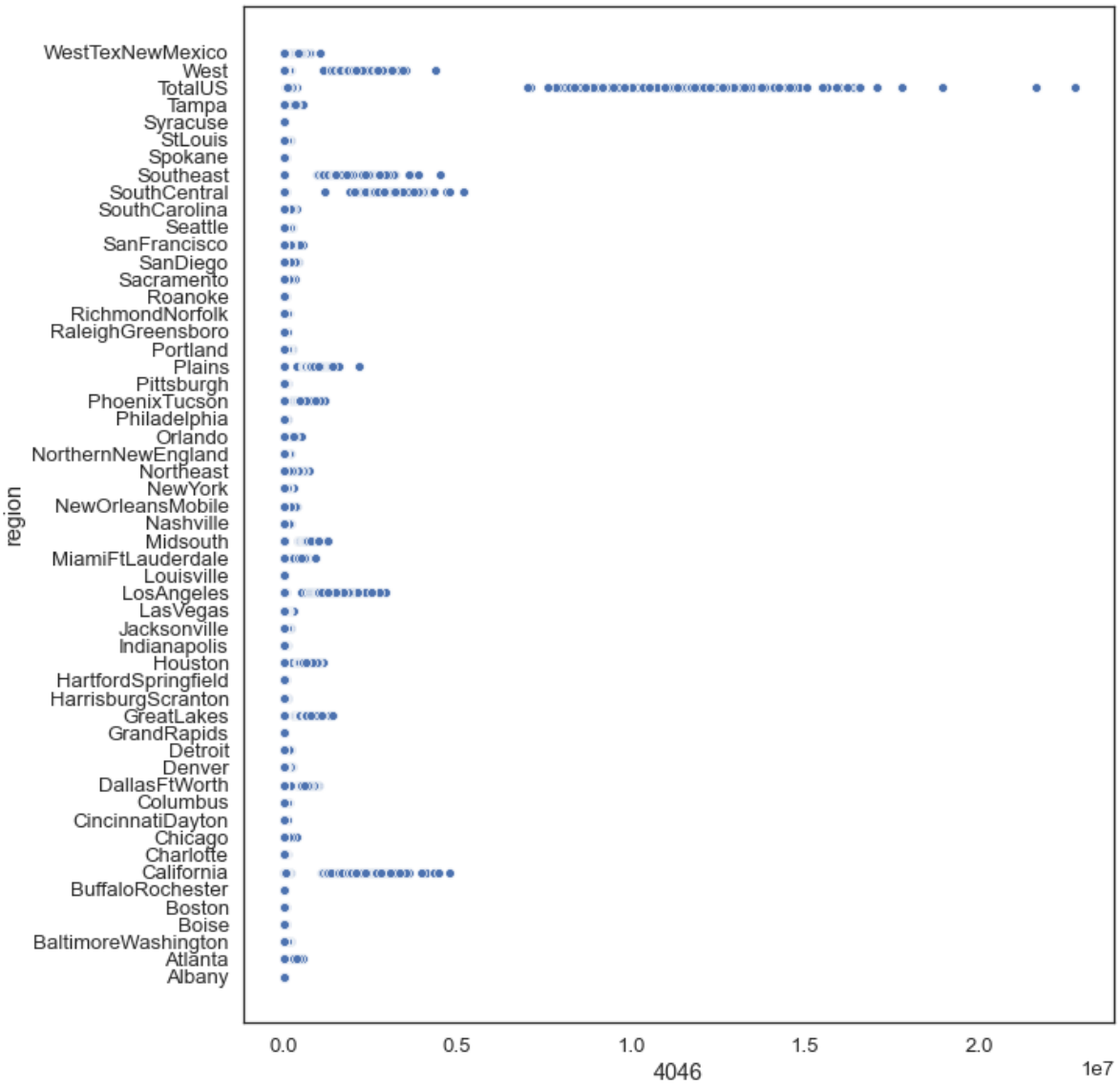

Figure 8: Scatterplot of the variables "4046" and "region"

As shown in Figure 8, most of the regions have an amount of "4046" below 5 million, which is much lower than the amount of "4046" from the region "TotalUS". Similar observations can be made when the amounts of "4046" are compared between a region at the state level (e.g., "California") and a region at the city level (e.g., "Albany") or even among regions at the same level (e.g., "Chicago" and "LosAngeles").

After inspecting the other numerical variables with abnormal distributions using scatterplots similar to the one above, it was concluded that such inconsistency regarding the region levels was the reason behind the high skewness. Therefore, it was apparent that the outliers of these variables shown in the boxplot in Figure 2 shouldn't be deleted. Moreover, these variables need to be transformed before being used to train the models, otherwise the great differences among them are likely to have an adverse effect on the performance of the models.

For this purpose, a log transformation was performed on these highly skewed variables.[1] The main reason is that the transformation would reflect the change in price relative to the independent variables while minimizing the region problem. To further explain, an example of a simple linear regression model is used here. Supposing the avocado price (Y) is predicted linearly by only one variable (C_1), we would have the following relationship:

$$Y_{AvocadoPrice} = C_0 + B_1\, ln(C_1) + E \qquad \textit{Equation 1}$$

Applying the first derivative of Y in respect to C_1 in Equation 1, we would have:

$$\frac{\partial Y}{\partial C_1} = \frac{B_1}{C_1} \qquad \textit{Equation 2}$$

Therefore, as C_1 gets higher, the rate of change in Y will be lower. On the other hand, a lower value of C_1 would lead to a higher rate of change in Y. Such relationships between the predictor (C_1) and the target (Y) variables, as shown in Equations 1 and 2, appear to better reflect the situation of the ones in our data. Compared to using the original predictors that are highly skewed, applying log transformation can moderate the range of the value of the predictor variables. In this way, the potential negative impact of the extreme values caused by the inconsistency in region levels can be reduced.

To avoid errors during the transformation due to zero values (negative values can also lead to errors, however, all values of the variables to be transformed are non-negative), every data point of these variables are increased by 1:

```
>> handle_zeros = data_reg.iloc[:, 1:9]+1
>> log_transf = pd.DataFrame(handle_zeros.apply(np.log))
```

Pairplots can be made to visualize the effect of the log transformation:

```
>> sns.set(font_scale = 1.7)
>> sns.set_style("white")
>> sns.pairplot(log_transf)
```

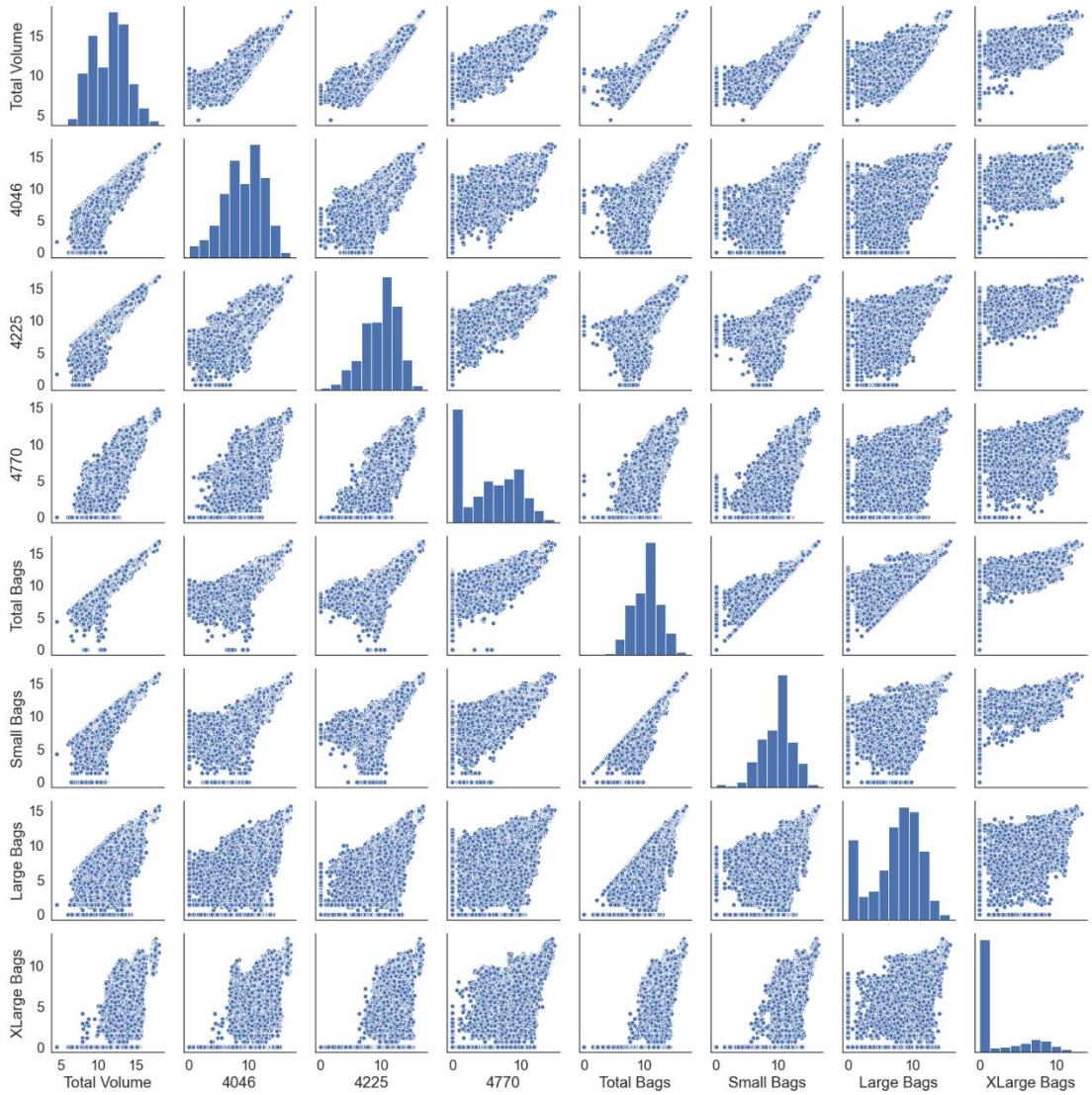

Figure 9: Pairplots of the previously highly skewed variables after being log transformed

Compared to Figure 1, the variables displayed in Figure 9 have shown much less skewness.

The original data is replaced by the log-transformed data:

```
>> data_log = data_reg.drop(data_reg.iloc[:, 1:9], axis=1)
>> data_log = pd.concat([data_log, log_transf], axis=1)
```

iv. Drop Highly Correlated Values

As shown in the pairplot and the heatmap of correlation matrix in section IV (ii), the variables "Total bags" and "Total volume" have high correlations with other variables. Therefore, to avoid multicollinearity, they should be dropped:

```
>> data_ = data_log
>> data_ = data_.drop('Total Bags', axis=1)
>> data_ = data_.drop('Total Volume', axis=1)
```

V. Pre-modelling processing and preparation

i. Selection of evaluation metrics

Now that the impact of the outliers is reduced, we can now start to choose the evaluation metric for our models. Three popular evaluation metrics for regression models include:

- Mean Square Error
- Mean Absolute Error
- Coefficient of Determination (R^2)

Compared to Mean Absolute Error, using Mean Square Error puts more penalty on predicted values that are far away from the true values than the ones that are closer, which aligns with the objective of the project. Therefore, the metric of Mean Absolute Error is not considered for this project, and thus the choice lies between Mean Square Error and Coefficient of Determination.

It should be taken into consideration that there is a relationship between the Mean Square Error and R^2, being that R^2 is the normalized version of the Mean Square Error. However, this doesn't mean that the two metrics are the same. The following section are intended for further explore the metrics.

a. Mean Square Error

Ideally, a prediction model should generate the result (denoted as $\hat{y}$) same as the true value (denoted as y):

$$y - \hat{y} = 0 \qquad \textit{Equation 3}$$

However, due to the existence of prediction errors, different models will perform differently, and one of the potential evaluations can be simply adding up the residuals of all predictions:

$$\sum_{i=1}^{n}(y - \hat{y}) \qquad \textit{Equation 4}$$

The problem with the metric shown in Equation 4 is that negative differences would offset the positive differences, thus reducing the actual value of the error. To avoid this, the difference can be squared and averaged, resulting in the Mean Square Error:

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y - \hat{y})^2 \qquad \textit{Equation 5}$$

Using Mean Square Error can not only better reflect the actual errors, but it can also account for the variance and bias of the predictions as shown in Equation 6:

$$(1)\quad E[y - \hat{y}]^2 = E[\hat{y}^2] + E[y^2] - 2yE[\hat{y}]$$
$$= Var(\hat{y}) + (E[\hat{y}])^2 + y^2 - 2yE[\hat{y}]$$

$$= Var(\hat{y}) + (E[\hat{y}] - y)^2$$
$$= Var(\hat{y}) + Bias^2(\hat{y}) \qquad \textit{Equation 6}$$

b. Coefficient of Determination (R^2)

The equation for calculating the Coefficient of Determination is given as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$
$$= 1 - \frac{SSR}{TSS} \qquad \textit{Equation 7}$$

In Equation 7, $\sum_{i=1}^{n}(y_i - \bar{y})^2$ is denoted as TSS (Total Sum of Squares), and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is denoted as SSR (Sum of Squares of Residuals). It should be noted that TSS is equal to n times the Variance of y. Moreover, SSR is closely related to MSE as defined in Equation 5, as SSR is equal to n times MSE. As the value of R^2 ranges between 0 and 1 and that it is related to MSE, it can therefore be regarded as the normalized value of MSE.

Meanwhile, R^2 can also be viewed in the following manner:

$$R^2 = 1 - FVU\ (Fraction\ of\ Variance\ Unexplained) \qquad \textit{Equation 8}$$

Compared to only using MSE, R^2 incorporates both MSE, which enables us to evaluate the variance-bias tradeoff, and the variance. Moreover, by normalizing, R^2 provides a more intuitive interpretation of the model outcomes. Therefore, the Coefficient of Determination (R^2) was deemed the most appropriate evaluation metric for this project.

ii. Scale data using normalization

Before partitioning the data, all the predictor variables are normalized:

```
>> data = data_
>> data.iloc[:, 2:4] = (data.iloc[:, 2:4]-data.iloc[:, 2:4].min()) /
(data.iloc[:, 2:4].max()-data.iloc[:, 2:4].min())
>> data.iloc[:, 57:63] = (data.iloc[:, 57:63]-data.iloc[:, 57:63].min()) /
(data.iloc[:, 57:63].max()-data.iloc[:, 57:63].min())
```

iii. Partition data

The data is first separated into predictor variable and target variable:

```
>> X = data.drop('AveragePrice', axis = 1)
>> Y = data['AveragePrice']
```

Then the data is further separated into training and testing data with a ratio of 75% training data vs. 25% testing data:

```
>> X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
random_state=0)
>> print(Y_train.shape)
>> print(Y_test.shape)

(13013,)
(4338,)
```

iv. Create a report for storing results

An empty report is created to store training and testing results of different models for final comparisons:

```
>> report = pd.DataFrame(columns=['Model','R2.Train','R2.Test'])
```

v. Calculate components of R^2

Some of the fixed components of the training and testing R^2 equations are also computed:

```
>> Y_train_mean = Y_train.mean()
>> print("Y_train_mean =", Y_train_mean)
>> Y_train_meandev = sum((Y_train-Y_train_mean) ** 2)
>> print("Y_train_meandev =", Y_train_meandev)
>> Y_test_meandev = sum((Y_test-Y_train_mean) ** 2)
>> print("Y_test_meandev =", Y_test_meandev)

Y_train_mean = 1.3556873895335433
Y_train_meandev = 1540.5051763005993
Y_test_meandev = 510.01502698900987
```

VI. Modeling

i. Selection of models

Considering that the target variable "AveragePrice" is continuous rather than discrete, several regression models will be used in this section, including individual and ensemble methods. The models selected are: Ordinary Least Square (or "OLS") Regression, Ridge Regression, Support Vector Regression, Neural Networks, Random Forests, Gradient Boosting, and two additional ensemble methods (bagging and stacking) that use the three best models from the ones previously listed.

ii. Training principles

For each of the models in Section VI (i) except for the two additional ensembles, the same training procedure was performed in order to find the best combination of their parameters (here the training of the SVR model will be used as an example):

a. Import the corresponding modelling package.

```
>> from sklearn.svm import SVR
>> RbfSVRregCV = SVR()
```

b. Create a parameter grid that has a relatively wide range and train the model using cross validation.

```
>> from sklearn.model_selection import GridSearchCV
>> param_grid = {
                'kernel': ["linear", "rbf"],
                'C': [1, 3, 5, 8, 10],
                'epsilon': [0.0, 0.025, 0.05, 0.075, 0.1],
                'gamma': [0., 1., 2., 3., 4.]
                 }
>> CV_svrmodel = GridSearchCV(estimator=RbfSVRregCV, param_grid=param_grid,
cv=10)
>> CV_svrmodel.fit(X_train, Y_train)
>> RbfSVRregCV = RbfSVRregCV.set_params(**CV_svrmodel.best_params_)
>> RbfSVRregCV.fit(X_train, Y_train)
>> Y_train_pred = RbfSVRregCV.predict(X_train)
>> Y_train_dev = sum((Y_train-Y_train_pred)**2)
>> r2 = 1 - Y_train_dev/Y_train_meandev
>> Y_test_pred = RbfSVRregCV.predict(X_test)
>> Y_test_dev = sum((Y_test-Y_test_pred)**2)
>> pseudor2 = 1 - Y_test_dev/Y_test_meandev
```

c. Store the result of every trial. Based on the output of GridSearchCV, which gives out the best parameter combination of the given grid, further adjust the parameter grid to make its range narrower. Then train the model again using the adjusted parameter grid.

```
>> print(CV_svrmodel.best_params_)
>> print("R2 =", r2)
>> print("Pseudo-R2 =", pseudor2)

{'C': 10, 'epsilon': 0.025, 'gamma': 2.0, 'kernel': 'rbf'}
R2 = 0.9617019655192849
Pseudo-R2 = 0.9194554029921328
```

In this case, since the best value of C lies at the end of the given range, the grid of the next trial will have a range of higher C's. Meanwhile, the best values of epsilon and gamma lie within the given range, therefore, the next trial will have narrower ranges of both of them. The parameter grid of the second trial for SVM therefore became:

```
>> param_grid = {
                'kernel': ["linear", "rbf"],
                'C': [9, 10, 11, 12],
                'epsilon': [0.015, 0.02, 0.025, 0.03, 0.035],
                'gamma': [1.5, 2., 2.5]
                 }
```

d. Repeat steps (b)-(c) to tune the parameters until:

- the best parameter values returned by the code have stabilized, or
- until the improvement from the last version (i.e., difference between the test R^2 of the current trial and the last one) has become very minor.

After using the adjusted parameter grid shown above and retraining, the following result was returned for the second trial:

```
{'C': 12, 'epsilon': 0.03, 'gamma': 1.5, 'kernel': 'rbf'}
R2 = 0.9574918198023498
Pseudo-R2 = 0.9196477843211218
```

Using the same principle, a third trial was carried out using the following grid:

```
>> param_grid = {
                'kernel': ["linear", "rbf"],
                'C': [12, 13, 14],
                'epsilon': [0.027, 0.03, 0.033],
                'gamma': [1.25, 1.5, 1.75]
                 }
```

The output from the third trial is:

```
{'C': 14, 'epsilon': 0.033, 'gamma': 1.5, 'kernel': 'rbf'}
R2 = 0.9589979889661289
Pseudo-R2 = 0.9198738025284898
```

Since the value of gamma has stabilized, it's not necessary to further adjust it for another trial (situation (1)). Meanwhile, the values of C and epsilon are still within a relatively wide range. However, considering that the improvement of the test R^2 (i.e., Pseudo R^2) is only around 0.0226% in the third trial compared to the second, no further tuning was deemed necessary (situation (2)).

iii. Training of the additional ensembles

As mentioned in Section VI (i), two additional ensemble methods (bagging and stacking) were trained using three best models from the others. In this case, the models Neural Network, Support Vector Regression, and Gradient Boosting generated the best results (the detailed results of each model will be shown in Section VII).

VII. Result

i. Comparison of different models

The results of all the trained models are as shown in Table 1, which are visualized in Figure 1. The R^2 value refer to the Coefficient of Determination (R^2) as determined earlier in the report.

Model	Training R^2	Testing R^2
OLS Regression	0.626	0.607
Ridge Regression	0.626	0.607
Support Vector Regression	0.959	0.920
Random Forest	0.984	0.891
Neural Network	0.959	0.938
Gradient Boosting	0.993	0.909
Ensemble-Bagging	0.979	**0.940**
Ensemble-Stacking	0.974	**0.943**

Table 1: Training and testing R^2 values of different models

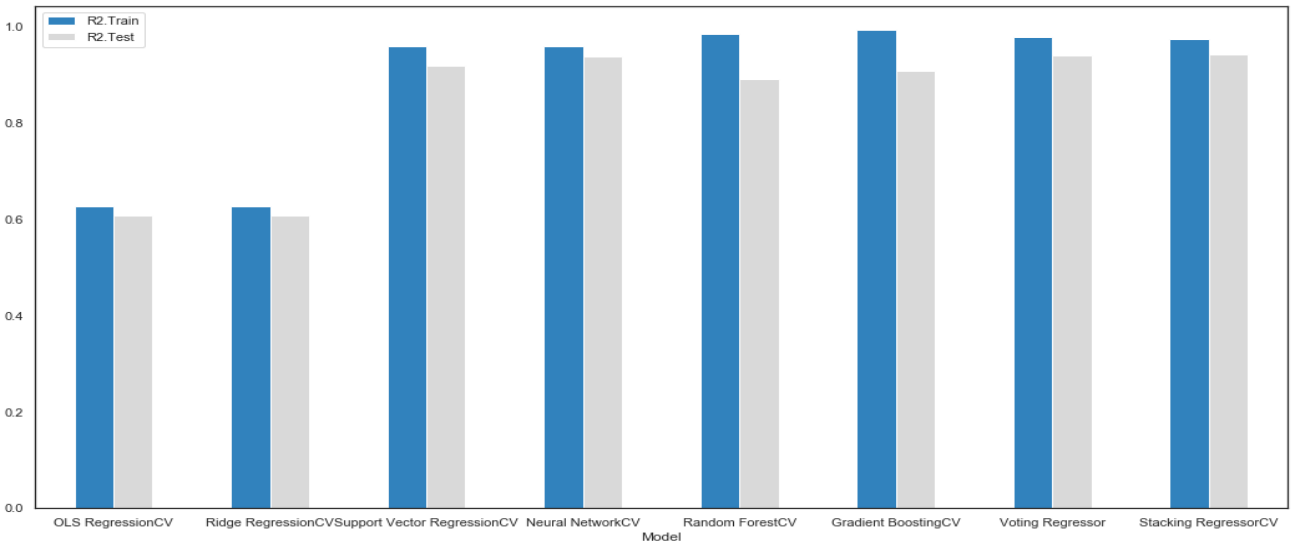

Figure 10: Training and testing R^2 values of different models

From Table 1 and Figure 10, it is apparent that the models Ensemble-Stacking and Ensemble-Bagging generate the highest test R^2 values.

ii. Analysis of individual models

a. As shown in Section VII (i), OLS and Ridge Regression have a much worse performance than the others, which is not surprising since they don't have many free parameters that can be tuned further. The low R^2 results from these methods also indicated that other models are more suitable for capturing the complexity of the data.

b. To explore the non-linear models further, the differences between their training and testing R^2 values are calculated.

Model	Training R^2	Testing R^2	Difference
Support Vector Regression	0.959	0.920	0.039
Random Forest	0.984	0.891	**0.093**
Neural Network	0.959	0.938	0.021
Gradient Boosting	0.993	0.909	**0.084**
Ensemble-Bagging	0.979	0.940	0.039
Ensemble-Stacking	0.974	0.943	0.031

Table 2: Training and testing R^2 values and their differences of different models

c. From Table 2, it can be observed that the values of "Difference" of Random Forest and Gradient Boosting are the highest. This indicates their tendency to overfit. Meanwhile, another disadvantage of these tree-models is that they might perform even worse in real-life application than what their testing R^2 suggest here. This is because tree-models can only predict by using the existing data and therefore are not able to extrapolate. This has 2 consequences:

- Firstly, the dataset of the project only contains data that is from 2015-2018. Since the variable "year" is very likely to be one of the splitting points of the tree models, if these models are used to predict current prices and encounter a new year value such as 2020, their performance is likely to be much worse as they do not know how to classify this value.
- Secondly, the dataset cannot output results outside the range of results of the training set. For example, if the input data contains prices between 1 Dollar and 5 Dollars, even if there are significant changes in the input features, and effects such as inflation are to be expected, it will never predict a result beyond the maximum of 5 Dollars.

d. It can also be observed from Table 2 that Neural Network has the smallest difference between its training and testing R^2, which indicates that the model performs the best among the others in terms of capturing the complexity of the data without overfitting.

e. As for Support Vector Regression, the results in Table 2 show that its performance is between that of the tree models and that of the Neural Network model.

f. Meanwhile, the results of the additional ensembles shown in Table 2 indicate the moderating effect of bagging and stacking. As mentioned before, the last two ensembles combined the models of Neural Network, Support Vector Regression, and Gradient Boosting. Compared to these individual models, both ensembles have shown improvements in their accuracy—while the highest testing R^2 of their components is 0.938, which is from the Neural Network model, the ensembles have achieved higher R^2 that exceed 0.94. In addition, the ensembles have moderated overfitting—after combining Gradient Boosting, which has a high tendency to overfit, with other models that have lower tendencies, the gap between training and testing R^2 has been reduced.

VIII. Alternatives

In this section, several alternatives are tried and/or discussed to further explore the other possibilities of training a prediction model for this project. In the following comparisons, the method discussed in the previous sections will be referred to as the "basic method".

i. Observing the effect of correlated variables and outliers

In the basic method, the highly correlated variables "Total Bags" and "Total Volume" were dropped from the features, as well as removing the top 5% of the target variable which were deemed to be outliers. As was mentioned in the previous section, robust tree models could possibly benefit from a larger range of target and feature variables.

Hence, to confirm these decisions, the models were trained on alternative datasets omitting each of these steps in the pre-processing. Tables 3a and 3b respectively show the training R^2 of each of these alternatives. Apart from the difference of including these steps or not, all the other data processing steps remained the same (e.g., log transformation). The training procedures are also the same as described in Section VII (ii).

Model	Basic method	Retrained with additional features	Difference
Random Forests	0.891	0.891	-
Gradient Boosting	0.909	0.915	0.006

Table 3a: Comparison of testing R^2 including "Total Bags" and "Total Volume"

Model	Basic method	Retrained while keeping outliers	Difference
Random Forests	0.891	0.893	0.002
Gradient Boosting	0.909	0.914	0.005

Table 3b: Comparison of testing R^2 including outliers

As can be observed in Table 3a, compared to the basic method, the retrained tree-models using the two additional variables have shown none or little improvement—they are insensitive to these additional columns. Similarly, they are also not heavily affected by the outliers, although the Gradient Boosting model did show a minor improvement.

ii. Handling the effect of regions

The most significant challenge of this dataset lay in the region's feature, as has been mentioned briefly in prior sections of the report. The basic method took the information of the region's name at face value and simply used one hot encoding to separate the information into a boolean class for each region. However, it was worth noting that the regions were not classified in a systematically logical way.

There were different tiers of information – i.e., some of the datapoints were assigned a broad geographical region (e.g., "GreatLakes"), others a county (e.g., "Detroit"), others a town (e.g., "Atlanta") and many no information at all (i.e., "TotalUS"). Even worse, some inputs were a combination of two towns (e.g., "BuffaloRochester") where one can only assume that this refers to the rough surrounding area centered on these towns, or what could be 2 towns or alternatively defined as a "metropolitan area" or airport (e.g., "BaltimoreWashington").

a. Reclassification of regions

The first approach to this was to reclassify the regions into broader regions, to “flatten the playing field”. All regions, except for “TotalUS”, were assigned to one of the 5 broader regions: Northeast, Southeast, West, Midwest, and Southwest.

After converting the “type” variable in III (iv) (a), the following reclassification was performed:

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

Southeast    5070
Northeast    4056
West         4056
Midwest      3042
Southwest    1687
TotalUS       338
```

The distribution of the reclassified regions is as shown in Figure 11. After the reclassification, the data is then used to retrain the models using the same approach as described in Section VI. The performance and time complexity of the retrained models were compared to those of the basic method, as shown in Figure 12a and Figure 12b. It can be concluded that while this approach showed great improvement in running time, it compromised the accuracy.

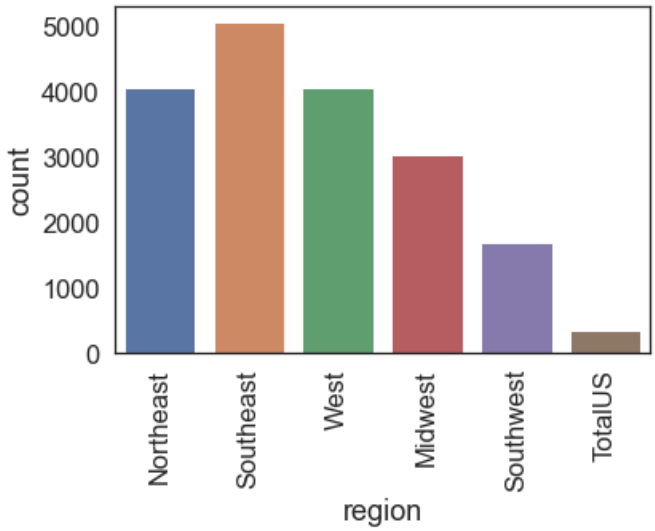

Figure 11: Distribution of regions after reclassification

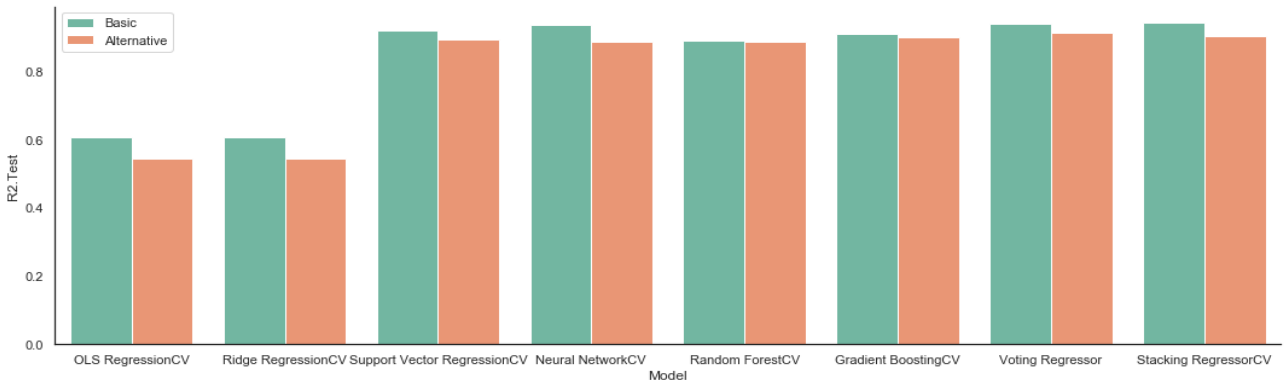

Figure 12a. R^2 before and after reclassification of regions

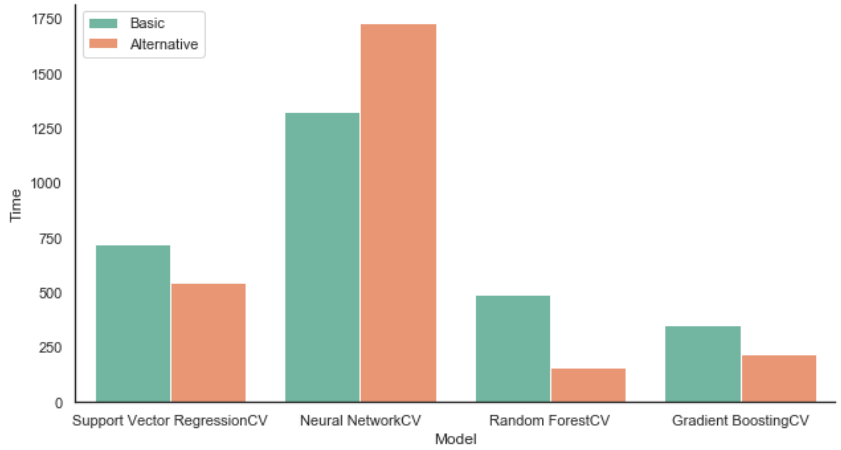

Figure 12b. Running time before and after reclassification of regions (in seconds)

b. Using a proxy for the variable “region”

Potential proxies for “region” include factors such as income and production of avocado in the respective region. Moreover, since the dataset contains information regarding the US, the distribution of ethnicities can also be considered. Given that the definition of regions was so unclear, the income per capita was chosen as it was possible to calculate estimates in cases of unclarity.

- For combinations of 2 towns, the weighted average of the average income for each town was calculated.
- For broad geographical regions, the average income per capita was calculated from the population, average number of people per household and average household income illustrated on the demographics information by Point2Homes[2], as the classification of regions was similar to that of the study's dataset.

After converting the "type" variable in III (iv) (a), the column "region" is replaced with the numerical variable for income per capita:

```
>> i = [28, 37.3, 60, 33, 79, 21, 66.6, 36.4, 35, 29, 46, 56, 44.8, 54.2,
31.2, 23.3, 20.5, 33, 35.2, 31.2, 34.4, 32, 28, 50.8, 36.4, 28, 37.6, 26.2,
72, 38.5, 38.5, 31.2, 63.5, 30.1, 38.4, 36, 41, 41, 26, 26, 32.6, 39, 50, 50,
45, 37, 28, 49, 37.3, 21, 33, 31, 31, 43]
>> r = data_rpdate.region.unique().tolist()
>> incomes = {}
>> for ind in range(len(i)):
        incomes[r[ind]] = i[ind]
>> def assignincome(row, incomes):
        return incomes[row['region']]
>> data_rpdate['income'] = data_rpdate.apply (lambda row: assignincome(row,
incomes) , axis=1)
>> data_rpdate = data_rpdate.drop(['region'], axis=1)
>> data_reg = data_rpdate
```

After replacing the "region" column, the data is then used to retrain the models using the same approach as described in Section VI. Figure 13 compares the testing R^2 for the basic method and the new model with proxy.

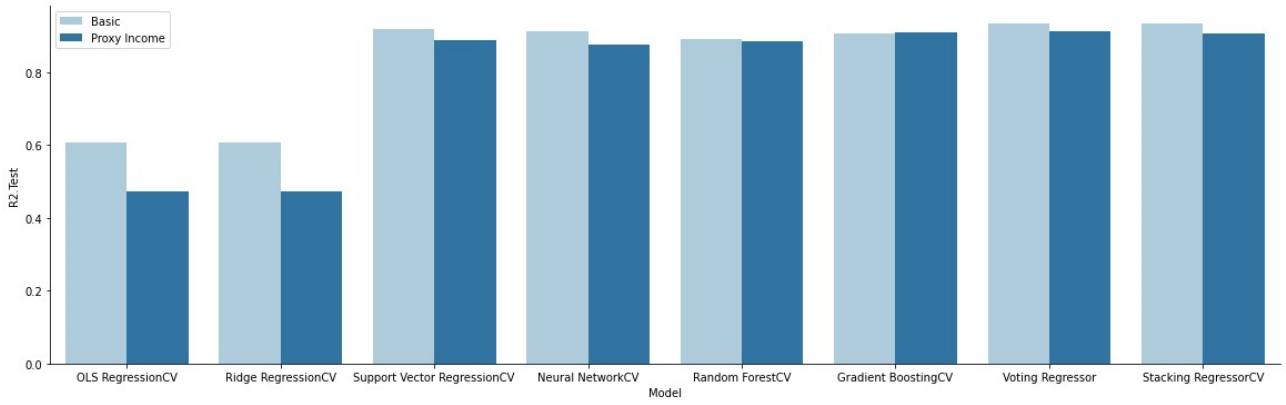

Figure 13. Testing R^2 of basic model vs. using income as proxy for region

As shown in Figure 13, the results were significantly worse for linear regression models, but barely differing for all others. While this can confirm that the research conducted to find the income per capita was on point, despite the large unclarity, it did not improve the results.

It is worth noticing that one model did marginally better with a proxy—the Gradient Boosted model, which suggests that further work in this area may produce viable results. It is also much better time-complexity-wise, as the many columns of regions are replaced by a simple numerical column.

iii. Future Work

Besides the methods applied in the study, there are many more alternatives that can be investigated and could potentially improve the model. One of these approaches would be Entity Embedding, which is a common method applied to categorical variables and has a tendency to reduce overfitting. This could be particularly useful for this case study, given that the major challenge of the project was capturing the effect that regions had

on the avocado price. It would also be possible to scale up the project and approach the problem using time-series analysis. This incorporates the element of time for predictions to become more sensitive over time.

IX. Conclusion

To build a model for predicting avocado prices, the study compared different machine learning methods using the performance metric of Coefficient of Determination (R^2). It was observed that the ensemble method of stacking, which incorporates the results of other individual models, had the overall best performance with a R^2 of 0.943.

To handle some of the problematic parts of the data and investigate the effect of some parts of the data preparation, further investigations and alternative approaches were carried out. However, the previous R^2 of 0.943 was not improved significantly. In most cases, there was a trade-off between time complexity and accuracy of the model, and for the purpose of this project, accuracy was prioritized.

While some concern remains regarding the applicability of the model in reality, as the ensemble method was built on several robust models, the project can conclude that machine learning can be applied to this dataset successfully, and it is indeed possible to predict the selling price of avocados given the features of this dataset. For future applications, the model can be retrained regularly using updated datasets so that it can perform better real-time predictions. Although the project has explored multiple possibilities of improving the model results, it should also be noted that there are still many other alternatives.

X. Reference

[1] Benoit, K. (2011, March 17). Linear Regression Models with Logarithmic Transformations. Retrieved from https://kenbenoit.net/assets/courses/ME104/logmodels2.pdf

[2] Retrieved from https://www.point2homes.com/US/Neighborhood/IL/Great-Lakes-Demographics.html. Accessed 2020, December 21.