In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import metrics
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import mean_squared_error
import math
```

In [2]:
```
#Load the avacado dataset
data = pd.read_csv('avocado.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Lar Ba
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 14 columns):
Unnamed: 0      18249 non-null int64
Date            18249 non-null object
AveragePrice    18249 non-null float64
Total Volume    18249 non-null float64
4046            18249 non-null float64
4225            18249 non-null float64
4770            18249 non-null float64
Total Bags      18249 non-null float64
Small Bags      18249 non-null float64
Large Bags      18249 non-null float64
XLarge Bags     18249 non-null float64
type            18249 non-null object
year            18249 non-null int64
region          18249 non-null object
dtypes: float64(9), int64(2), object(3)
memory usage: 1.9+ MB
```

Data Preparation

The irrelavant variable in the avacado dataset is 'Unnamed: 0'

- "The 'Unnamed: 0' variable is dropped from the dataset"

In [5]:
```
data=data.drop(['Unnamed: 0'],axis = 1)
```

The columns '4046', '4225', '4770' are renamed as 'Small Hass', 'Large Hass' and 'Extra Large Hass'

In [6]:
```
data=data.rename(columns={"4046": "Small Hass","4225": "Large Hass","4770": "E
xtra Large Hass"})
```

The values of the variable 'type' is mapped as {'conventional':'1','organic':'0'}

In [7]:
```
change_values = {'conventional' : 1, 'organic' : 0}
data['type'] = data['type'].map(change_values)
```

The format of the observations in the Date variable (YYYY-MM-DD) is coverted as (YYYY-MM) for data analysis by month

In [8]:
```
data['Date']=data['Date'].str[0:-3]
```

The given dataset contains details about price and volume of avocado for all weeks in each month from Jan 2015 to March 2018

- The data per each month in every region for both conventional and organic is determined by calculating the mean of all the weeks in each month
- The data is grouped by the variables 'type', 'Date', 'region'

In [9]:
```
group_data=data.groupby(['type','Date','region']).mean()
group_data.to_csv('Avacado_grouped_data.csv')
```

In [10]:
```
groupdata = pd.read_csv('Avacado_grouped_data.csv')
groupdata.head()
```

Out[10]:

	type	Date	region	AveragePrice	Total Volume	Small Hass	Large Hass	Extra Large Hass	
0	0	2015-01	Albany	1.8450	1197.7175	29.8275	196.6250	0.00	97
1	0	2015-01	Atlanta	1.8325	3732.9750	1707.7350	967.6750	0.00	105
2	0	2015-01	BaltimoreWashington	1.3325	18647.8375	9136.2625	5850.2275	583.91	307
3	0	2015-01	Boise	1.5850	2008.0675	2.8400	1606.1650	0.00	39
4	0	2015-01	Boston	1.9450	2141.8925	7.2100	872.5575	0.00	126

In [11]:
```
data_organic = groupdata.groupby('type').get_group(0)
data_conventional = groupdata.groupby('type').get_group(1)
```

In [12]:
```
data_organic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2106 entries, 0 to 2105
Data columns (total 13 columns):
type                2106 non-null int64
Date                2106 non-null object
region              2106 non-null object
AveragePrice        2106 non-null float64
Total Volume        2106 non-null float64
Small Hass          2106 non-null float64
Large Hass          2106 non-null float64
Extra Large Hass    2106 non-null float64
Total Bags          2106 non-null float64
Small Bags          2106 non-null float64
Large Bags          2106 non-null float64
XLarge Bags         2106 non-null float64
year                2106 non-null int64
dtypes: float64(9), int64(2), object(2)
memory usage: 230.3+ KB
```

In [13]:
```
data_conventional.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2106 entries, 2106 to 4211
Data columns (total 13 columns):
type                2106 non-null int64
Date                2106 non-null object
region              2106 non-null object
AveragePrice        2106 non-null float64
Total Volume        2106 non-null float64
Small Hass          2106 non-null float64
Large Hass          2106 non-null float64
Extra Large Hass    2106 non-null float64
Total Bags          2106 non-null float64
Small Bags          2106 non-null float64
Large Bags          2106 non-null float64
XLarge Bags         2106 non-null float64
year                2106 non-null int64
dtypes: float64(9), int64(2), object(2)
memory usage: 230.3+ KB
```

In [14]:
```
np.count_nonzero(data_organic['XLarge Bags'])
```

Out[14]: 53

In [15]:
```
np.count_nonzero(data_conventional['XLarge Bags'])
```

Out[15]: 1686

Missing values

- The missing values in this dataset are assigned with '0'
- The variable 'XLarge Bags' has more than 50% missing values
- If we group the data separately by the variable 'type' and calculate the missing values in the column 'XLarge Bags' then there are 2053 missing values for organic and 420 missing values for conventional.
- We can drop the column 'XLarge Bags' for further analysis

In [16]:
```
# Dropping the variable 'XLarge Bags' from the dataset - groupdata
groupdata = groupdata.drop(['XLarge Bags'], axis = 1)
```

Data Exploration

Descriptive Statistics

In [17]:
```
data_organic[['Total Volume', 'Small Hass', 'Large Hass', 'Extra Large Hass']]
.describe()
```

Out[17]:

	Total Volume	Small Hass	Large Hass	Extra Large Hass
count	2.106000e+03	2106.000000	2106.000000	2106.000000
mean	4.793476e+04	7308.324154	15416.058715	261.064457
std	1.418299e+05	22911.891722	44861.141104	1012.231646
min	7.070975e+02	0.000000	0.000000	0.000000
25%	4.894299e+03	181.388000	735.240000	0.000000
50%	1.087757e+04	967.186250	3144.562750	0.000000
75%	3.078097e+04	4482.272250	9945.638250	37.766625
max	1.633609e+06	256211.472000	457878.716000	17333.770000

In [18]:
```
data_organic[['Total Bags', 'Small Bags', 'Large Bags']].describe()
```

Out[18]:

	Total Bags	Small Bags	Large Bags
count	2.106000e+03	2106.000000	2106.000000
mean	2.494523e+04	17695.938297	7248.062845
std	8.162639e+04	60672.354774	25740.824943
min	7.192500e+00	0.000000	0.000000
25%	1.886388e+03	880.594375	11.876000
50%	5.284485e+03	3060.487500	599.737500
75%	1.506474e+04	10005.244000	3202.279375
max	1.094251e+06	879146.160000	402012.160000

In [19]:
```
data_conventional[['Total Volume', 'Small Hass', 'Large Hass', 'Extra Large Ha
ss']].describe()
```

Out[19]:

	Total Volume	Small Hass	Large Hass	Extra Large Hass
count	2.106000e+03	2.106000e+03	2.106000e+03	2.106000e+03
mean	1.655341e+06	5.799370e+05	5.743132e+05	4.549175e+04
std	4.727242e+06	1.734367e+06	1.637842e+06	1.466775e+05
min	3.762428e+04	6.066240e+02	1.852114e+03	5.460000e+00
25%	2.015394e+05	3.310541e+04	5.213619e+04	6.334365e+02
50%	4.158724e+05	1.071119e+05	1.387030e+05	6.329908e+03
75%	1.029565e+06	3.626902e+05	4.219316e+05	2.126050e+04
max	4.560122e+07	1.614587e+07	1.500885e+07	1.823240e+06

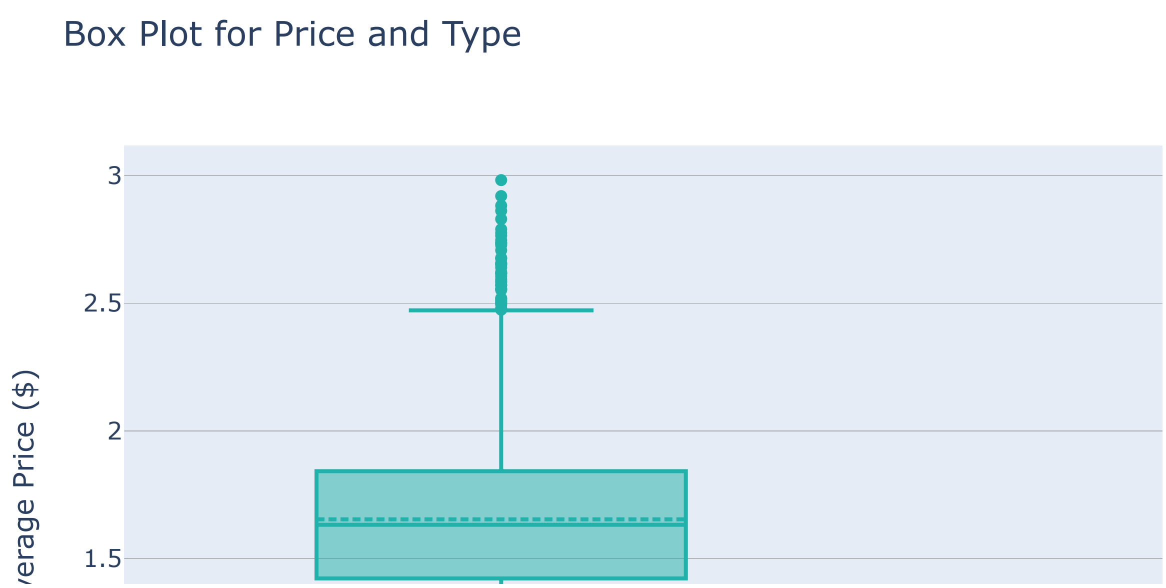
In [20]:
```
data_conventional[['Total Bags', 'Small Bags', 'Large Bags']].describe()
```

Out[20]:

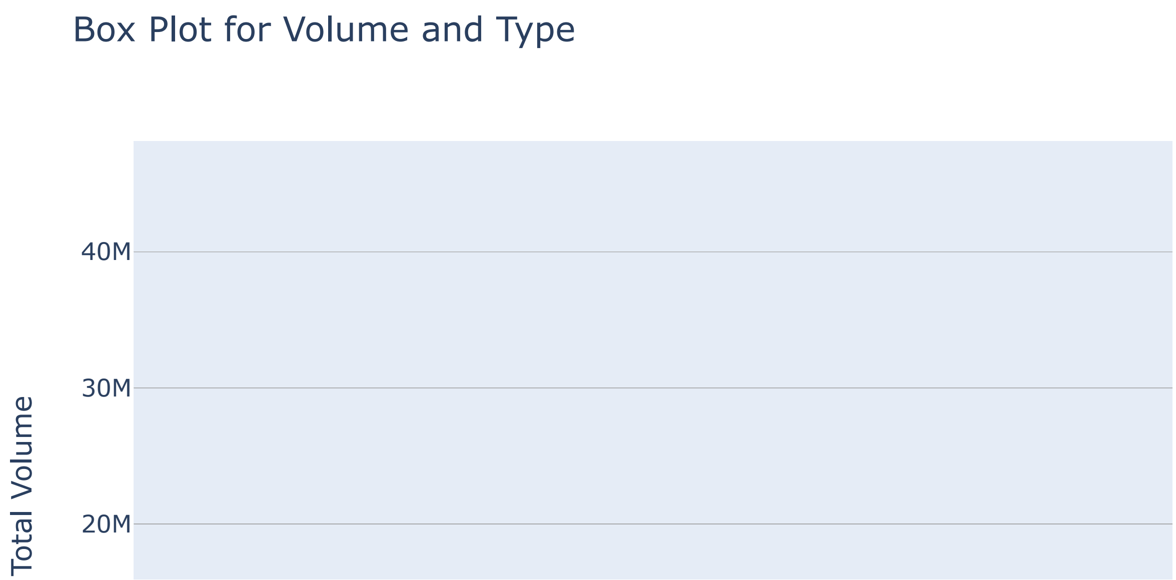
	Total Bags	Small Bags	Large Bags
count	2.106000e+03	2.106000e+03	2.106000e+03
mean	4.555986e+05	3.475746e+05	1.018100e+05
std	1.356075e+06	1.025119e+06	3.339136e+05
min	4.929124e+03	3.122030e+03	0.000000e+00
25%	5.796445e+04	4.410697e+04	2.970971e+03
50%	1.002049e+05	7.632378e+04	1.540124e+04
75%	2.954301e+05	2.043858e+05	6.049880e+04
max	1.548242e+07	1.161972e+07	4.243784e+06

In [21]:
```
dataOrg = groupdata.loc[groupdata['type'] == 0]
dataCon = groupdata.loc[groupdata['type'] == 1]
```

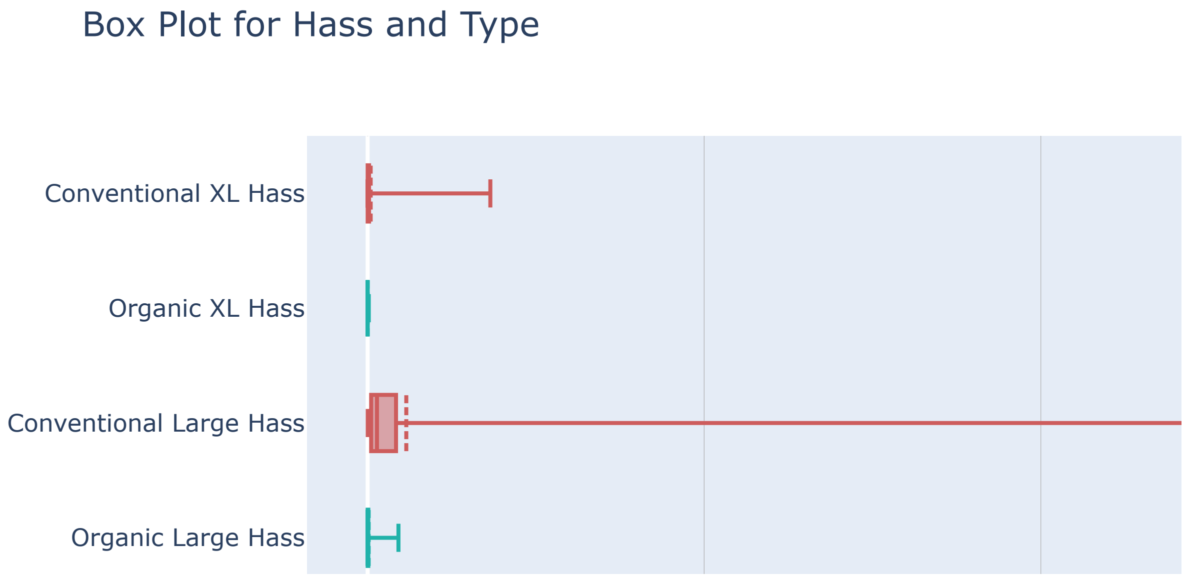
In [22]:
```
import plotly.graph_objects as go

x0 = dataOrg['type']
x1 = dataCon['type']
y0 = dataOrg['AveragePrice']
y1 = dataCon['AveragePrice']

fig = go.Figure()
fig.add_trace(go.Box(x=x0, y=y0, name = "Organic", boxmean=True,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box(x=x1, y=y1, name = "Conventional", boxmean=True,
                marker_color = 'indianred'))

fig.update_layout(yaxis_title='Average Price ($)', xaxis_title='Type of Avocad
o', title_text="Box Plot for Price and Type")
fig.show()
```



In [23]:
```
x0 = dataOrg['type']
x1 = dataCon['type']
y0 = dataOrg['Total Volume']
y1 = dataCon['Total Volume']

fig = go.Figure()
fig.add_trace(go.Box(x=x0, y=y0, name = "Organic", boxmean=True, boxpoints=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box(x=x1, y=y1, name = "Conventional", boxmean=True, boxpoints=False,
                marker_color = 'indianred'))

fig.update_layout(yaxis_title='Total Volume', xaxis_title='Type of Avocado', title_text="Box Plot for Volume and Type")
fig.show()
```



In [24]:
```
y00 = dataOrg['Small Hass']
y01 = dataCon['Small Hass']
y10 = dataOrg['Large Hass']
y11 = dataCon['Large Hass']
y20 = dataOrg['Extra Large Hass']
y21 = dataCon['Extra Large Hass']

fig = go.Figure()
fig.add_trace(go.Box( x=y00, name = "Organic Small Hass", boxmean=True, boxpoi
nts=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box( x=y01, name = "Conventional Small Hass", boxmean=True, b
oxpoints=False,
                marker_color = 'indianred'))
fig.add_trace(go.Box( x=y10, name = "Organic Large Hass", boxmean=True, boxpoi
nts=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box( x=y11, name = "Conventional Large Hass", boxmean=True, b
oxpoints=False,
                marker_color = 'indianred'))
fig.add_trace(go.Box( x=y20, name = "Organic XL Hass", boxmean=True, boxpoints
=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box( x=y21, name = "Conventional XL Hass", boxmean=True, boxp
oints=False,
                marker_color = 'indianred'))

fig.update_layout(xaxis_title='Amount of Hass', title_text="Box Plot for Hass
 and Type")
fig.show()
```


Box Plot for Hass and Type
Conventional XL Hass
Organic XL Hass
Conventional Large Hass
Organic Large Hass

In [25]:
```
y00 = dataOrg['Small Bags']
y01 = dataCon['Small Bags']
y10 = dataOrg['Large Bags']
y11 = dataCon['Large Bags']

fig = go.Figure()
fig.add_trace(go.Box( x=y00, name = "Organic Small Bags", boxmean=True, boxpoints=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box( x=y01, name = "Conventional Small Bags", boxmean=True, boxpoints=False,
                marker_color = 'indianred'))
fig.add_trace(go.Box( x=y10, name = "Organic Large Bags", boxmean=True, boxpoints=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box( x=y11, name = "Conventional Large Bags", boxmean=True, boxpoints=False,
                marker_color = 'indianred'))

fig.update_layout(xaxis_title='Amount of Bags', title_text="Box Plot for Bags and Type")
fig.show()
```

In [26]:
```
ryo = pd.pivot_table(dataOrg, index=['region', 'year'],
                     values=['AveragePrice'],
                     aggfunc = np.mean).reset_index()
ryo.head()
```

Out[26]:

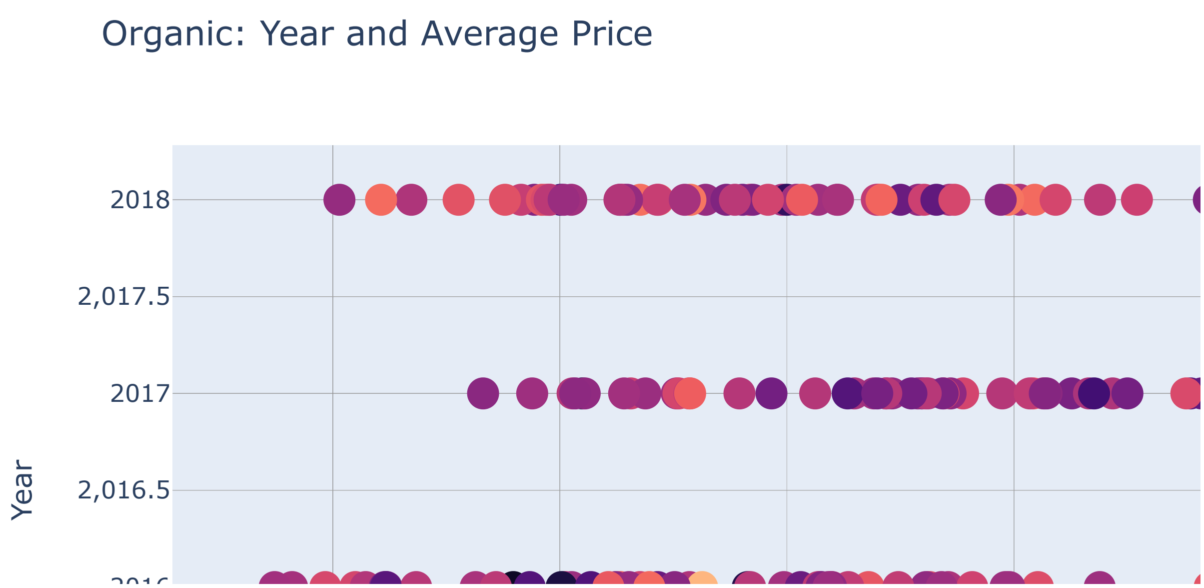
	region	year	AveragePrice
0	Albany	2015	1.906042
1	Albany	2016	1.725667
2	Albany	2017	1.755208
3	Albany	2018	1.528333
4	Atlanta	2015	1.711000

In [27]:
```
ryc = pd.pivot_table(dataCon, index=['region', 'year'],
                     values=['AveragePrice'],
                     aggfunc = np.mean).reset_index()
ryc.head()
```

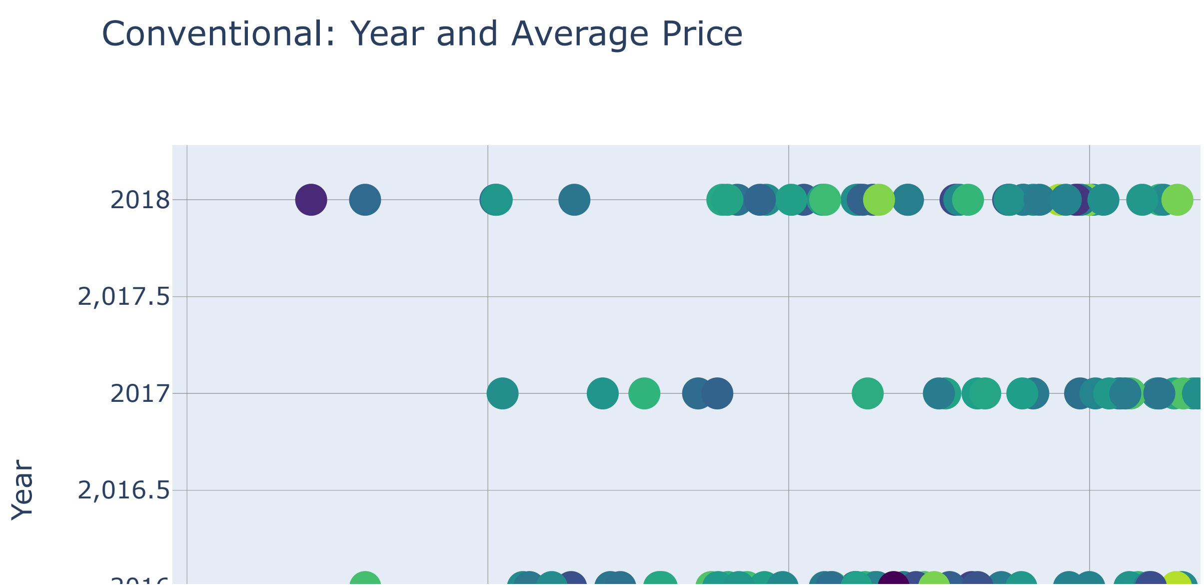
Out[27]:

	region	year	AveragePrice
0	Albany	2015	1.171833
1	Albany	2016	1.349417
2	Albany	2017	1.528458
3	Albany	2018	1.343333
4	Atlanta	2015	1.051083

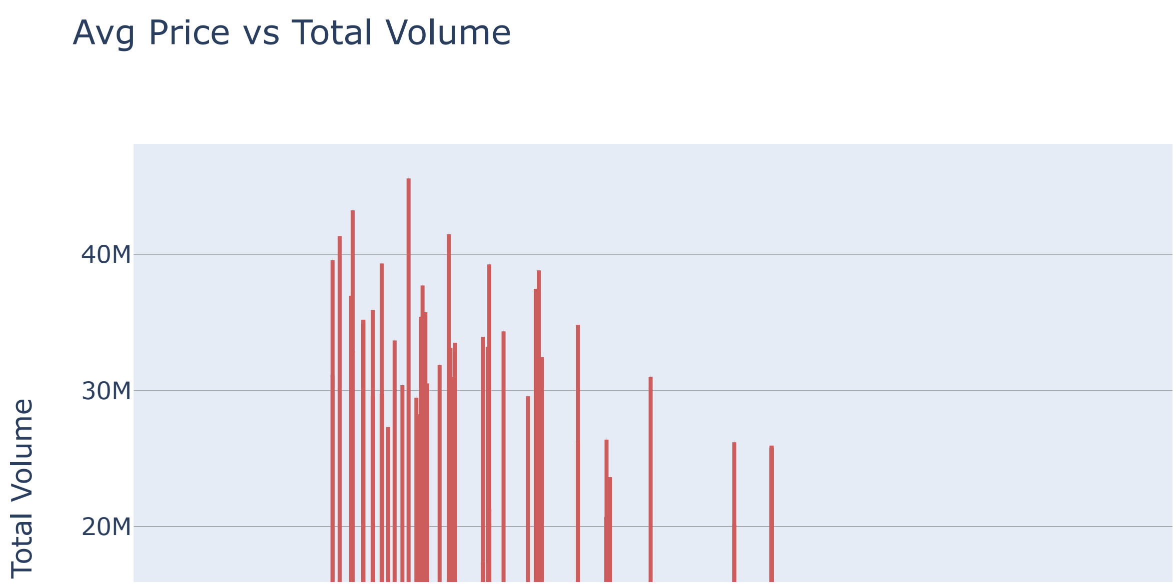
In [28]:
```
fig = go.Figure(data=go.Scatter(y=ryo['year'], x=ryo['AveragePrice'], mode='ma
rkers',marker=dict(
        size=16,
        color=np.random.randn(1000), #set color equal to a variable
        colorscale='Magma', # one of plotly colorscales
        showscale=False
    ),text=ryo['region']))

fig.update_layout(yaxis_title='Year', xaxis_title = 'Average Price', title_tex
t = 'Organic: Year and Average Price')
fig.show()
```



In [29]:
```
fig = go.Figure(data=go.Scatter(y=ryc['year'], x=ryc['AveragePrice'], mode='ma
rkers',marker=dict(
        size=16,
        color=np.random.randn(1000), #set color equal to a variable
        colorscale='Viridis', # one of plotly colorscales
        showscale=False
    ),text=ryc['region']))

fig.update_layout(yaxis_title='Year', xaxis_title = 'Average Price', title_tex
t = 'Conventional: Year and Average Price')
fig.show()
```



In [30]:
```
x0 = dataOrg['AveragePrice']
x1 = dataCon['AveragePrice']
y0 = dataOrg['Total Volume']
y1 = dataCon['Total Volume']

fig = go.Figure()
fig.add_trace(go.Box(x=x0, y=y0, name = "Organic", boxmean=True, boxpoints=False,
                marker_color = 'lightseagreen'))
fig.add_trace(go.Box(x=x1, y=y1, name = "Conventional", boxmean=True, boxpoints=False,
                marker_color = 'indianred'))

fig.update_layout(yaxis_title='Total Volume', xaxis_title='Average Price', title_text="Avg Price vs Total Volume")
fig.show()
```



The size of the hass avocado and the volume play an important role in determining the average price

Data Modeling

Partitioned the given dataset into train set and test set

- The dataset is partitioned using Holdout method with the ratio 80:20 (train - 80, test - 20) and random_state = 0

In [31]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 13 columns):
Date                18249 non-null object
AveragePrice        18249 non-null float64
Total Volume        18249 non-null float64
Small Hass          18249 non-null float64
Large Hass          18249 non-null float64
Extra Large Hass    18249 non-null float64
Total Bags          18249 non-null float64
Small Bags          18249 non-null float64
Large Bags          18249 non-null float64
XLarge Bags         18249 non-null float64
type                18249 non-null int64
year                18249 non-null int64
region              18249 non-null object
dtypes: float64(9), int64(2), object(2)
memory usage: 1.8+ MB
```

In [32]:
```
data.columns
```

Out[32]:
```
Index(['Date', 'AveragePrice', 'Total Volume', 'Small Hass', 'Large Hass',
       'Extra Large Hass', 'Total Bags', 'Small Bags', 'Large Bags',
       'XLarge Bags', 'type', 'year', 'region'],
      dtype='object')
```

In [33]:
```
data1 = data[['Date', 'AveragePrice', 'Total Volume', 'Small Hass', 'Large Has
s', 'Extra Large Hass', 'Total Bags', 'Small Bags', 'Large Bags', 'year', 'reg
ion']]
x_train, x_test, y_train, y_test = train_test_split(data1, data['type'], test_
size = 0.2, random_state = 0)
```

Partitioning the train dataset into training set and validation set

- The dataset is partitioned using Holdout method with the ratio 80:20 (training - 80, - 20) and random_state = 0

In [34]:
```
x_training, x_valid, y_training, y_valid = train_test_split(x_train, y_train,
test_size = 0.2, random_state = 0)
```

In [35]:
```
x_training.head()
```

Out[35]:

	Date	AveragePrice	Total Volume	Small Hass	Large Hass	Extra Large Hass	Total Bags	Small Bags	
15913	2017-11	1.99	61451.23	6477.51	19956.35	0.00	35017.37	35003.39	
14499	2016-08	1.53	2472.90	7.34	157.78	0.00	2307.78	2307.78	
288	2015-06	1.34	149376.79	1183.90	48286.51	4432.54	95473.84	95473.84	
426	2015-10	1.15	690302.58	25883.13	482992.35	118311.09	63116.01	62076.66	[illegible]
7013	2017-08	1.07	261757.88	114191.45	37848.66	52.60	109665.17	90141.14	17

In [36]:
```
x_training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11679 entries, 15913 to 3682
Data columns (total 11 columns):
Date                    11679 non-null object
AveragePrice            11679 non-null float64
Total Volume            11679 non-null float64
Small Hass              11679 non-null float64
Large Hass              11679 non-null float64
Extra Large Hass        11679 non-null float64
Total Bags              11679 non-null float64
Small Bags              11679 non-null float64
Large Bags              11679 non-null float64
year                    11679 non-null int64
region                  11679 non-null object
dtypes: float64(8), int64(1), object(2)
memory usage: 1.1+ MB
```

In [37]:
```
x_valid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2920 entries, 8068 to 1213
Data columns (total 11 columns):
Date                2920 non-null object
AveragePrice        2920 non-null float64
Total Volume        2920 non-null float64
Small Hass          2920 non-null float64
Large Hass          2920 non-null float64
Extra Large Hass    2920 non-null float64
Total Bags          2920 non-null float64
Small Bags          2920 non-null float64
Large Bags          2920 non-null float64
year                2920 non-null int64
region              2920 non-null object
dtypes: float64(8), int64(1), object(2)
memory usage: 273.8+ KB
```

In [38]:
```
std_train=x_training.iloc[:,2:-2]
std_valid=x_valid.iloc[:,2:-2]
scaler = StandardScaler()
scaler.fit(std_train)
x_training_scaled = scaler.transform(std_train)
x_validate_scaled = scaler.transform(std_valid)
x_training_scaled_df=pd.DataFrame(x_training_scaled,index=std_train.index,colu
mns=std_train.columns)
x_validate_scaled_df=pd.DataFrame(x_validate_scaled,index=std_valid.index,colu
mns=std_valid.columns)
```

In [39]:
```
x_training_scaled_df.head()
```

Out[39]:

	Total Volume	Small Hass	Large Hass	Extra Large Hass	Total Bags	Small Bags	Large Bags
15913	-0.228025	-0.224122	-0.226961	-0.214315	-0.209275	-0.198210	-0.226348
14499	-0.245127	-0.229200	-0.243183	-0.214315	-0.243333	-0.242874	-0.226408
288	-0.202530	-0.228277	-0.203748	-0.172756	-0.146326	-0.115605	-0.226408
426	-0.045679	-0.208890	0.152440	0.894975	-0.180018	-0.161227	-0.225185
7013	-0.169943	-0.139577	-0.212300	-0.213822	-0.131550	-0.122890	-0.150898

Regression Models

1) Simple linear regression model - predicting Average price

In [40]:
```
#We choose the variable 'Total Volume' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Total Volume'].values.resha
pe(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Total Volume'].values.r
eshape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.07725022]
R sqaure - 0.035827999998724749
Adjusted R square - 0.03439322022532376
RMSE - 0.40033086604582935
```

In [41]:
```
#We choose the variable 'Small Hass' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Small Hass'].values.reshape
(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Small Hass'].values.res
hape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.0831554]
R sqaure - 0.04432098644249519
Adjusted R square - 0.04289884505327257
RMSE - 0.3986462608246278
```

In [42]:
```
#We choose the variable 'Large Hass' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Large Hass'].values.reshape
(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Large Hass'].values.res
hape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.06881659]
R sqaure - 0.02809263133642437
Adjusted R square - 0.02664634060924642
RMSE - 0.4019661966144799
```

In [43]:
```
#We choose the variable 'Extra Large Hass' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Extra Large Hass'].values.r
eshape(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Extra Large Hass'].valu
es.reshape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.07211424]
R sqaure - 0.03328330014678438
Adjusted R square - 0.03184473362914564
RMSE - 0.4009502934583451
```

In [44]:
```
#We choose the variable 'Total Bags' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Total Bags'].values.reshape
(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Total Bags'].values.res
hape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.07162492]
R sqaure - 0.028344943615073722
Adjusted R square - 0.026899028352596166
RMSE - 0.4018522982186152
```

In [45]:
```
#We choose the variable 'Small Bags' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Small Bags'].values.reshape
(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Small Bags'].values.res
hape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.07015113]
R sqaure - 0.02892493689780641
Adjusted R square - 0.027479884720571057
RMSE - 0.40172502817846917
```

In [46]:
```
#We choose the variable 'Large Bags' as predictor
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df['Large Bags'].values.reshape
(-1,1), y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df['Large Bags'].values.res
hape(-1,1))
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/672)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.07118648]
R sqaure - 0.023874263188135653
Adjusted R square - 0.022421695127403662
RMSE - 0.40304754640535556
```

2) Multi-linear regression model - predicting Average price

In [47]:
```
#We choose all the variables as predictors
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df, y = x_training['AveragePric
e'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df)
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/666)
print("Adjusted R square -",adjusted_r)

rmse = np.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[ 5.24446868e+02 -1.93877725e+02 -1.85501638e+02 -1.62713195e+01
 -1.44815488e+02 -9.20150894e-01 -3.37229882e-01]
R sqaure - 0.05033258495171705
Adjusted R square - 0.040351095604362786
RMSE - 0.39727893353139965
```

In [48]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass', 'Small Bags'
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass', 'Small Bags']], y = x_training['AverageP
rice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass', 'Small Bags']])

corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/668)
print("Adjusted R square -",adjusted_r)

rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[-0.5335518   0.05808837  0.2669596  -0.0245042   0.1674232 ]
R sqaure - 0.04823183517434387
Adjusted R square - 0.041107821964570856
RMSE - 0.3977105867215314
```

In [49]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass', 'Large Bags'
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass', 'Large Bags']], y = x_training['AverageP
rice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass', 'Large Bags']])

corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/668)
print("Adjusted R square -",adjusted_r)

rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[ 0.2493176  -0.23017558 -0.00854469 -0.05043016 -0.05594876]
R sqaure - 0.04836773856548718
Adjusted R square - 0.041244742596665884
RMSE - 0.3976813083565826
```

In [50]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Extra Large Hass'
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Hass', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small Hass', 'Extra Large Hass']])

corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/670)
print("Adjusted R square -",adjusted_r)

rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))

print('RMSE -',rmse)
```

```
[ 0.13158878 -0.18447713 -0.03217562]
R sqaure - 0.05239386909627955
Adjusted R square - 0.04815085656984486
RMSE - 0.39717080143144706
```

3) LASSO regression model - predicting Average price

In [51]:
```
#we choose all the predictors

model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df, y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df)
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/666)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.010510510510510551
RMSE - 0.4076618306824906
```

In [52]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass', 'Small Bags'

model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass', 'Small Bags']], y = x_training['AverageP
rice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass', 'Small Bags']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/668)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.007485029940119681
RMSE - 0.4076618306824906
```

In [53]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass'

model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/669)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.005979073243647326
RMSE - 0.4076618306824906
```

In [54]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Extra Large Hass'

model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/670)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.004477611940298498
RMSE - 0.4076618306824906
```

4) Ridge regression model - predicting Average price

In [55]:
```
#with all the predictors

model = linear_model.Ridge(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df, y = x_training['AveragePric
e'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df)
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/666)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[ 0.08620264 -0.16711481  0.05423291 -0.04876033  0.46180985 -0.32885188
 -0.15608599]
R sqaure - 0.04919364212870972
Adjusted R square - 0.03920018191084318
RMSE - 0.3975127368889688
```

In [56]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass', 'Small Bags'

model = linear_model.Ridge(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass', 'Small Bags']], y = x_training['AverageP
rice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass', 'Small Bags']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/668)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0.47498552  0.03628241  0.24678023 -0.02629816  0.15084199]
R sqaure - 0.04871940790516923
Adjusted R square - 0.04159904419188465
RMSE - 0.3976146686822127
```

In [57]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass'

model = linear_model.Ridge(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/669)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[ 0.0114049  -0.13851118  0.08722055 -0.04313346]
R sqaure - 0.05086329237917996
Adjusted R square - 0.04518833448608084
RMSE - 0.3972568525245051
```

In [58]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Extra Large Hass'

model = linear_model.Ridge(alpha = 1)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/670)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[ 0.13084887 -0.18383312 -0.03206369]
R sqaure - 0.052379201038907315
Adjusted R square - 0.048136122834604
RMSE - 0.3971751025607456
```

5) Elastic Net regression model - predicting Average price

In [59]:
```
#with all the predictors
model = linear_model.ElasticNet(alpha = 1, l1_ratio = 0.5)
fitted_model = model.fit(X = x_training_scaled_df, y = x_training['AveragePric
e'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df)
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/666)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.010510510510510551
RMSE - 0.4076618306824906
```

In [60]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass', 'Small Bags'

model = linear_model.ElasticNet(alpha = 1, l1_ratio = 0.5)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass', 'Small Bags']], y = x_training['AverageP
rice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass', 'Small Bags']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/668)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.007485029940119681
RMSE - 0.4076618306824906
```

In [61]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Large Hass', 'Extra
 Large Hass'

model = linear_model.ElasticNet(alpha = 1, l1_ratio = 0.5)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Large Hass', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Large Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/669)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.005979073243647326
RMSE - 0.4076618306824906
```

In [62]:
```
#We choose the variables - 'Total Volume', 'Small Hass', 'Extra Large Hass'

model = linear_model.ElasticNet(alpha = 1, l1_ratio = 0.5)
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Extra Large Hass']], y = x_training['AveragePrice'])
print(fitted_model.coef_)
predicted = fitted_model.predict(x_validate_scaled_df[['Total Volume', 'Small
 Hass', 'Extra Large Hass']])
corr_coef = np.corrcoef(predicted,x_valid['AveragePrice'])[1, 0]
R_squared = corr_coef ** 2
print("R sqaure -",R_squared)
adjusted_r = 1 - (((1-R_squared)*(673))/670)
print("Adjusted R square -",adjusted_r)
rmse = math.sqrt(mean_squared_error(x_valid['AveragePrice'], predicted))
print('RMSE -',rmse)
```

```
[-0. -0. -0.]
R sqaure - 2.691468863004993e-32
Adjusted R square - -0.004477611940298498
RMSE - 0.4076618306824906
```

Result of prediction made by different regression models:

1. For evaluation of the performance of regression models, we consider the values of **Adjusted R-squared** and **RMSE**
2. The best regression model to predict the average price of an avacado is **Multi-Linear Regression model**
3. The best predictors are **'Total Volume', 'Small Hass', 'Extra Large Hass'**

Classification models

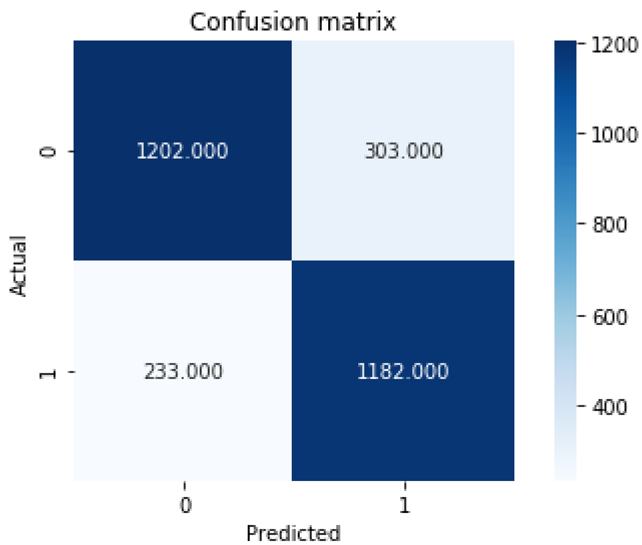
Decision Tree

1) Decesion Tree Classifire 1st Try.
Parameteres : Default
Variables : All variables

In [63]:
```
classifier = DecisionTreeClassifier(criterion="entropy",random_state = 0)
classifier.fit(x_training_scaled_df.loc[:,['Total Bags']],y_training)
```

Out[63]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [64]:
```
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Bags']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [65]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

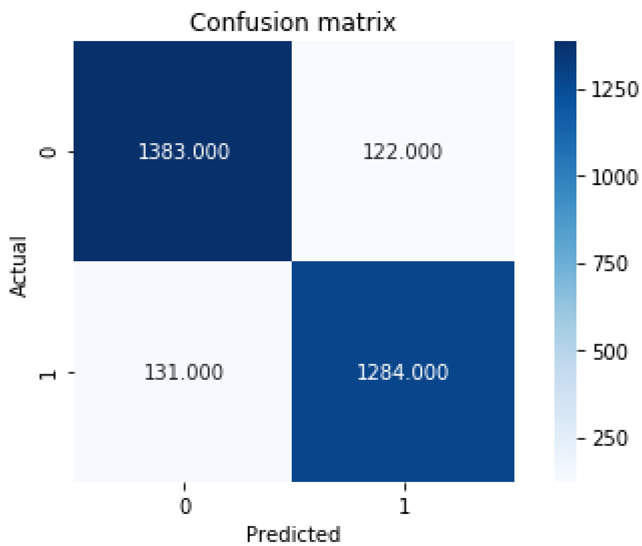
```
[0.8164383561643835, 0.18356164383561646, array([0.83763066, 0.7959596 ]), ar
ray([0.7986711 , 0.83533569]), array([0.81768707, 0.81517241])]
```

2) Descision Tree 2nd Try.
Parameters : criterion = gini, min_samples_leaf=2
Variables : Total Bags, Total Volume

In [66]:
```
classifier = DecisionTreeClassifier(criterion="gini",random_state = 0, min_sam
ples_leaf=2)
classifier.fit(x_training_scaled_df.loc[:,['Total Bags','Total Volume']],y_tra
ining)
```

Out[66]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [67]:
```
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Bags','Total Vo
lume']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [68]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

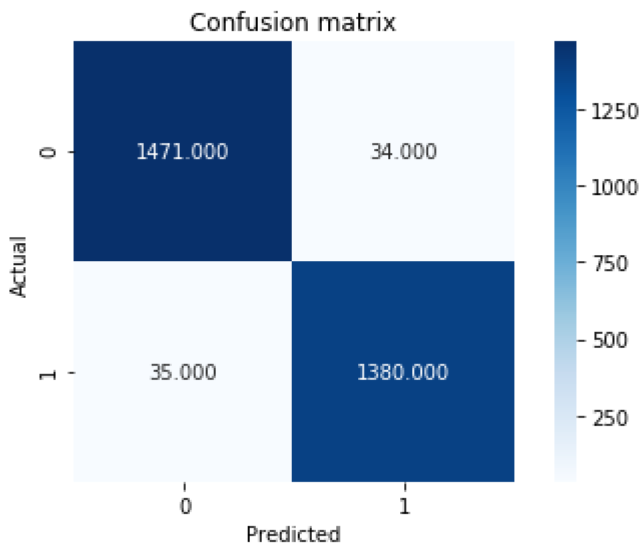
```
[0.9133561643835616, 0.0866438356164384, array([0.91347424, 0.91322902]), arr
ay([0.91893688, 0.90742049]), array([0.91619742, 0.91031549])]
```

3) Decesion Tree 3rd Try. (**Best Model**)
Parameters: criterion = entropy, min_sample_leaf = 2, splitter = best,
Variables : Large Bags, Total Bags, Total Volume, Small Hass, Large Hass, Extra Large Hass

In [69]:
```
classifier = DecisionTreeClassifier(criterion="entropy",random_state = 0, min_
samples_leaf=2,splitter = "best")
classifier.fit(x_training_scaled_df.loc[:,['Large Bags','Total Bags','Total Vo
lume','Small Hass','Large Hass','Extra Large Hass']],y_training)
```

Out[69]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [70]:
```
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Large Bags','Total Bags','Total Volume','Small Hass','Large Hass','Extra Large Hass']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



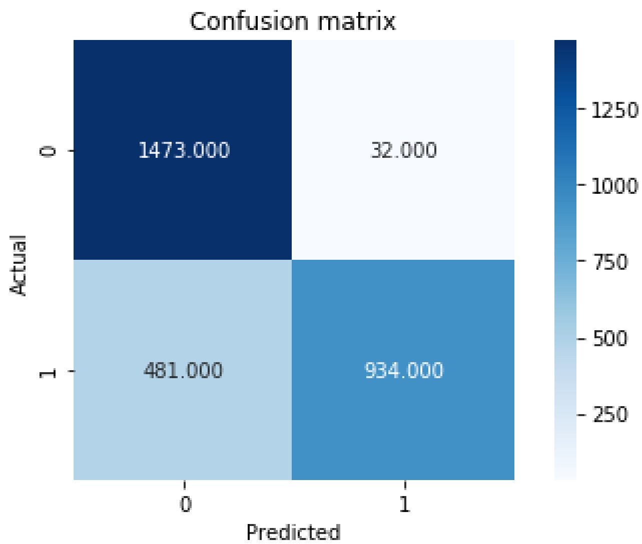
In [71]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9763698630136987, 0.023630136986301342, array([0.97675963, 0.97595474]), array([0.97740864, 0.97526502]), array([0.97708403, 0.97560976])]
```

Naive Bayes

1)Naive Bayes 1st Try.
Parameters : Default
Variables : All Variables

In [72]:
```
classifier = GaussianNB()
classifier.fit(x_training_scaled_df,y_training)
y_pred = classifier.predict(x_validate_scaled_df)
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [73]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

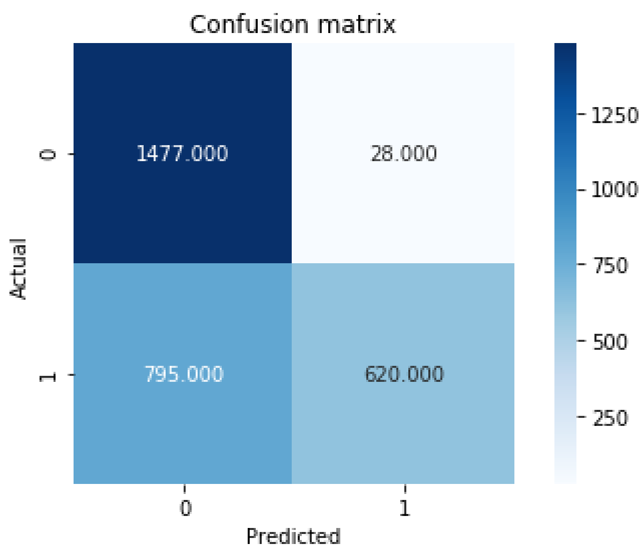
```
[0.8243150684931507, 0.17568493150684927, array([0.75383828, 0.96687371]), ar
ray([0.97873754, 0.66007067]), array([0.85169124, 0.78454431])]
```

2) Naive Bayes 2nd Try.

Parameters : Default

Variables : Total Bags, Total Volume

In [74]:
```
classifier = GaussianNB()
classifier.fit(x_training_scaled_df.loc[:,['Total Bags','Total Volume']],y_tra
ining)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Bags','Total Vo
lume']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

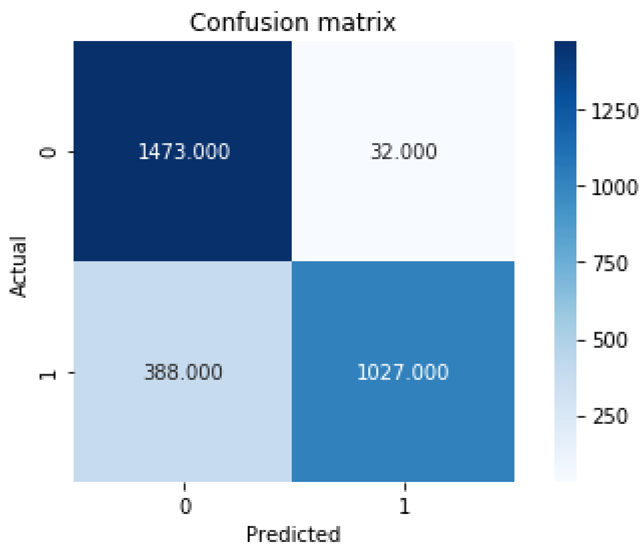


In [75]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7181506849315068, 0.2818493150684932, array([0.65008803, 0.95679012]), arr
ay([0.98139535, 0.43816254]), array([0.7821022 , 0.60106641])]
```

3) Naive Bayes 3rd Try.
Parameters : var_smoothing = 3e-09
Variables : Large Hass , Small Hass , Extra Large Hass

In [76]:
```
classifier = GaussianNB(var_smoothing=3e-09)
classifier.fit(x_training_scaled_df.loc[:,['Large Hass','Small Hass','Extra La
rge Hass']],y_training)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Large Hass','Small Ha
ss','Extra Large Hass']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



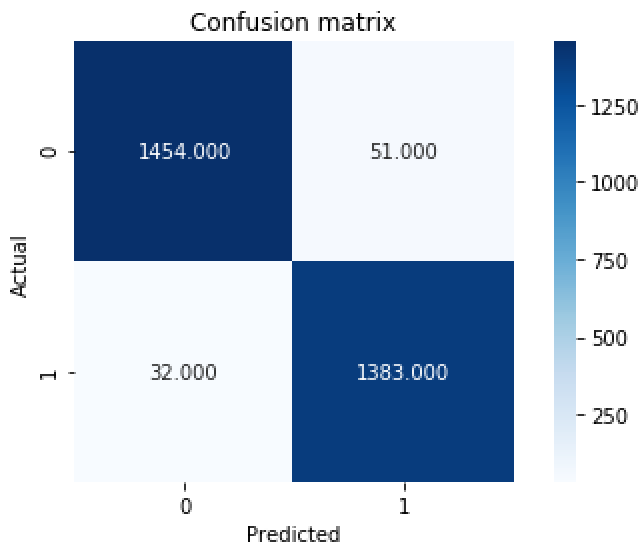
In [77]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8561643835616438, 0.14383561643835618, array([0.79150994, 0.96978281]), ar
ray([0.97873754, 0.72579505]), array([0.87522282, 0.83023444])]
```

K - Nearest Neighbours

1) K - Nearest Neighbors 1st try.
parameters : n_neighbors = 3
variables : All the variables

In [78]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_training_scaled_df,y_training)
y_pred = classifier.predict(x_validate_scaled_df)
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

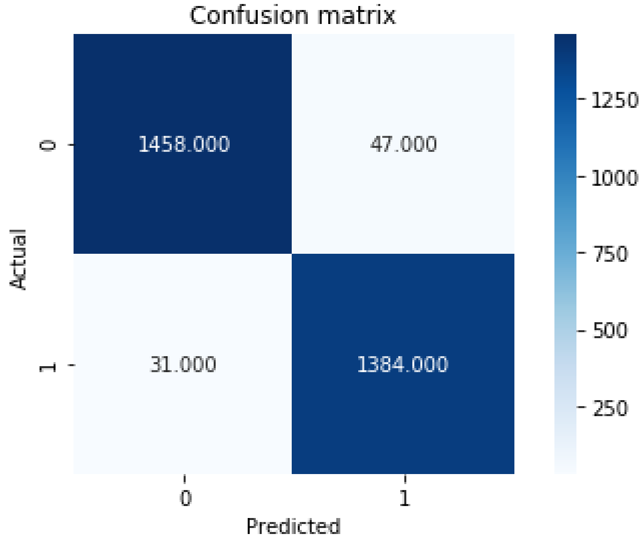


In [79]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9715753424657534, 0.02842465753424661, array([0.97846568, 0.96443515]), ar
ray([0.96611296, 0.97738516]), array([0.97225008, 0.97086697])]
```

2) K - Nearest Neighbors 2nd try. (Best Model)
parameters : n_neighbors = 3
variables : Large Bags,Total Bags,Total Volume,Large Hass,Small Hass,Extra Large Hass

In [80]:
```
classifier = KNeighborsClassifier(n_neighbors=3,algorithm='kd_tree')
classifier.fit(x_training_scaled_df.loc[:,['Large Bags','Total Bags','Total Vo
lume','Large Hass','Small Hass','Extra Large Hass']],y_training)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Large Bags','Total Ba
gs','Total Volume','Large Hass','Small Hass','Extra Large Hass']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

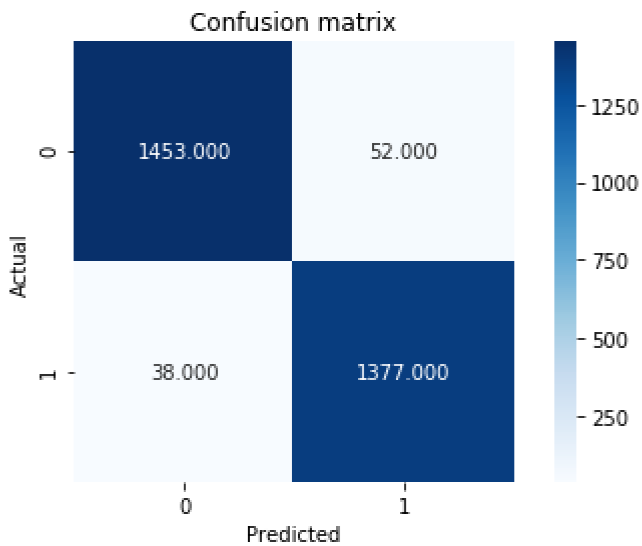


In [81]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9732876712328767, 0.02671232876712326, array([0.97918066, 0.96715584]), ar
ray([0.96877076, 0.97809187]), array([0.9739479 , 0.97259311])]
```

3) K - Nearest Neighbors 3rd try.
parameters : n_neighbors = 5,algorithm = kd_tree ,weights = distance
variables : Total Bags,Total Volume,Large Hass,Small Hass,Extra Large Hass

In [82]:
```
classifier = KNeighborsClassifier(n_neighbors=5,algorithm='kd_tree',weights='distance')
classifier.fit(x_training_scaled_df.loc[:,['Total Bags','Total Volume','Large Hass','Small Hass','Extra Large Hass']],y_training)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Bags','Total Volume','Large Hass','Small Hass','Extra Large Hass']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



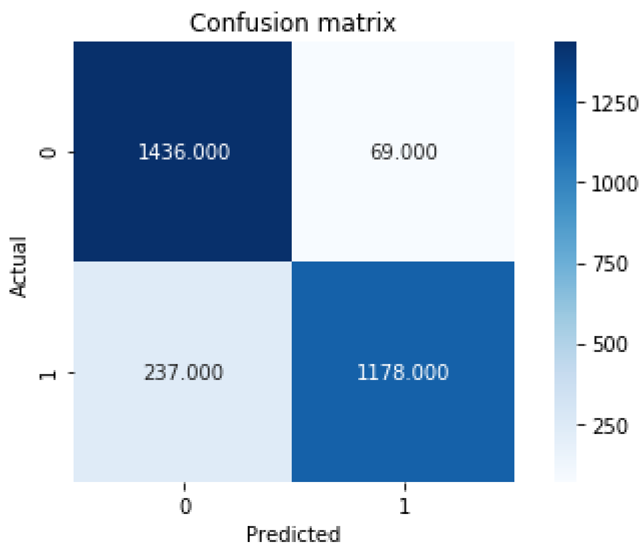
In [83]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.9691780821917808, 0.03082191780821919, array([0.97451375, 0.96361092]), array([0.9654485 , 0.97314488]), array([0.96995995, 0.96835443])]
```

Support Vector Machines

1) Support Vector Machine 1st try.
parameters : kernel = rbf
variables : All the variables

In [84]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_training_scaled_df,y_training)
y_pred = classifier.predict(x_validate_scaled_df)
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

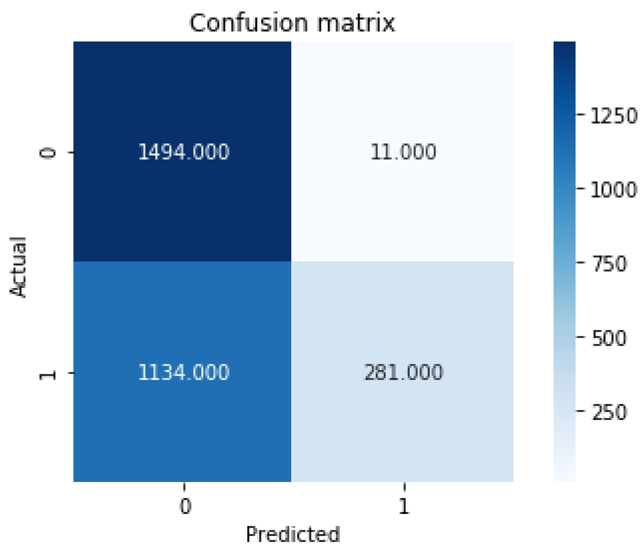


In [85]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8952054794520548, 0.10479452054794525, array([0.85833831, 0.9446672 ]), ar
ray([0.95415282, 0.83250883]), array([0.90371303, 0.88504884])]
```

2) Support Vector Machine 2nd try.
parameters : kernel = poly,probability=True,shrinking=True
variables :Total Bags, Large Bags, Small Bags, XLarge Bags

In [86]:
```
classifier = SVC(kernel='poly',probability=True,shrinking=True)
classifier.fit(x_training_scaled_df.loc[:,['Total Volume','Large Bags','Small
 Bags']],y_training)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Volume','Large
 Bags','Small Bags']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

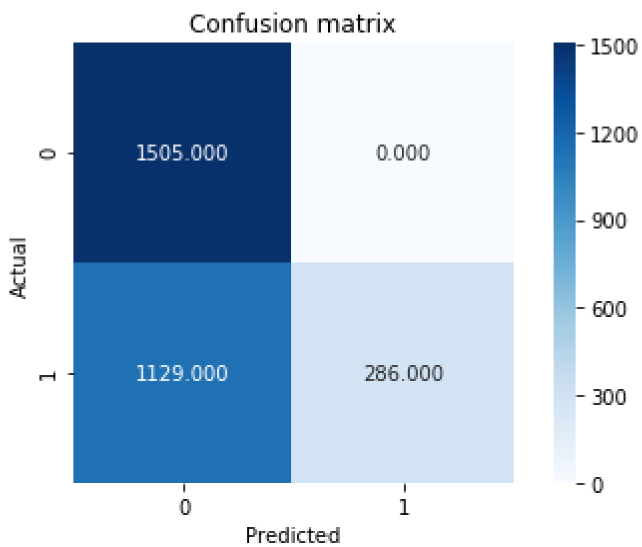


In [87]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.6078767123287672, 0.39212328767123283, array([0.56849315, 0.96232877]), ar
ray([0.99269103, 0.19858657]), array([0.72296153, 0.32923257])]
```

3) Support Vector Machine 3rd try.
parameters : kernel = rbf, decision_function_shape = ovo
variables : Total Bags,Large Hass,Small Hass,Extra Large Hass

In [88]:
```
classifier = SVC(kernel='poly',decision_function_shape='ovo')
classifier.fit(x_training_scaled_df.loc[:,['Total Bags','Large Hass','Small Ha
ss','Extra Large Hass']],y_training)
y_pred = classifier.predict(x_validate_scaled_df.loc[:,['Total Bags','Large Ha
ss','Small Hass','Extra Large Hass']])
conf_matrix = metrics.confusion_matrix(y_valid,y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [89]:
```
accuracy = metrics.accuracy_score(y_valid,y_pred)
error = 1 - metrics.accuracy_score(y_valid,y_pred)
precision = metrics.precision_score(y_valid,y_pred, average = None)
recall = metrics.recall_score(y_valid,y_pred, average = None)
F1_score = metrics.f1_score(y_valid,y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.6133561643835617, 0.38664383561643834, array([0.57137434, 1.        ]), ar
ray([1.        , 0.20212014]), array([0.7272288 , 0.33627278])]
```

Result of classification by different models:

1. For evaluation of the performance of classification models, we consider the values of **Accuracy**, **Error**, **Precision**, **Recall** and **F1-Score**
2. The best classification model to classify the avacado into conventional or oraganic is **Decision Tree classification model**
3. The parameters used are (criterion = entropy, min_sample_leaf = 2, splitter = best)
4. The variables used are - Large Bags, Total Bags, Total Volume, Small Hass, Large Hass, Extra Large Hass

Prediction and classification for Test data

The best regression model and classification model is used for avacado average price prediction and classification of type

In [90]:
```
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3650 entries, 9181 to 13532
Data columns (total 11 columns):
Date                3650 non-null object
AveragePrice        3650 non-null float64
Total Volume        3650 non-null float64
Small Hass          3650 non-null float64
Large Hass          3650 non-null float64
Extra Large Hass    3650 non-null float64
Total Bags          3650 non-null float64
Small Bags          3650 non-null float64
Large Bags          3650 non-null float64
year                3650 non-null int64
region              3650 non-null object
dtypes: float64(8), int64(1), object(2)
memory usage: 342.2+ KB
```

In [91]:
```
test = x_test[['Date', 'Total Volume', 'Small Hass', 'Large Hass', 'Extra Larg
e Hass', 'Total Bags', 'Small Bags', 'Large Bags', 'year', 'region']]
test.head()
```

Out[91]:

	Date	Total Volume	Small Hass	Large Hass	Extra Large Hass	Total Bags	Small Bags	Large Bags	year
9181	2015-12	4400.25	1358.53	1735.98	0.00	1305.74	130.00	1175.74	2015
1013	2015-07	190716.43	4890.33	119457.27	13495.86	52872.97	30631.37	21037.53	2015
14625	2016-03	1045450.41	105069.07	352698.21	9425.64	578257.49	252881.52	325375.97	2016
15234	2017-09	9883.59	313.75	4230.58	0.00	5339.26	2166.91	3172.35	2017
18247	2018-01	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	2018

In [92]:
```
std_test = test.iloc[:,1:-2]
scaler = StandardScaler()
scaler.fit(std_train)
x_test_scaled = scaler.transform(std_test)
x_test_scaled_df=pd.DataFrame(x_test_scaled,index=std_test.index,columns=std_t
est.columns)
x_test_scaled_df.head()
```

Out[92]:

	Total Volume	Small Hass	Large Hass	Extra Large Hass	Total Bags	Small Bags	Large Bags
9181	-0.244568	-0.228140	-0.241890	-0.214315	-0.244377	-0.245849	-0.221373
1013	-0.190542	-0.225367	-0.145432	-0.087778	-0.190684	-0.204183	-0.136317
14625	0.057302	-0.146737	0.045680	-0.125940	0.356360	0.099421	1.166974
15234	-0.242978	-0.228960	-0.239846	-0.214315	-0.240177	-0.243067	-0.212822
18247	-0.241145	-0.228007	-0.240870	-0.207499	-0.234315	-0.231110	-0.226194

In [93]:
```
#Using multi-linear regression model to predict Avacado average price
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_training_scaled_df[['Total Volume', 'Small Has
s', 'Extra Large Hass']], y = x_training['AveragePrice'])
predicted = fitted_model.predict(x_test_scaled_df[['Total Volume', 'Small Has
s', 'Extra Large Hass']])
```

In [94]:
```
#Using Decision tree classifier for avacado classification
classifier = DecisionTreeClassifier(criterion="entropy",random_state = 0, min_
samples_leaf=2,splitter = "best")
classifier.fit(x_training_scaled_df.loc[:,['Large Bags','Total Bags','Total Vo
lume','Small Hass','Large Hass','Extra Large Hass']],y_training)
pred_type = classifier.predict(x_test_scaled_df.loc[:,['Large Bags','Total Bag
s','Total Volume','Small Hass','Large Hass','Extra Large Hass']])
```

In [95]:
```
test['AveragePrice'] = predicted
test['type'] = pred_type
```

In [96]:
```
test.head()
```

Out[96]:

	Date	Total Volume	Small Hass	Large Hass	Extra Large Hass	Total Bags	Small Bags	Large Bags	year
9181	2015-12	4400.25	1358.53	1735.98	0.00	1305.74	130.00	1175.74	2015
1013	2015-07	190716.43	4890.33	119457.27	13495.86	52872.97	30631.37	21037.53	2015
14625	2016-03	1045450.41	105069.07	352698.21	9425.64	578257.49	252881.52	325375.97	2016
15234	2017-09	9883.59	313.75	4230.58	0.00	5339.26	2166.91	3172.35	2017
18247	2018-01	16205.22	1527.63	2981.04	727.01	10969.54	10919.54	50.00	2018

In [97]:
```
result = test[['Date', 'region', 'AveragePrice', 'type']]
```

In [98]:
```
result.to_csv("test_output.csv")
```

In [ ]: