In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

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

Loading the data file into Jupyter notebook

In [3]:

```
df=pd.read_csv('Avacado.csv')
```

In [4]:

```
df
```

Out[4]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	S B
0	0	27-12-2015	1.33	64236.62	1036.74	54454.85	48.16	8696.87	860
1	1	20-12-2015	1.35	54876.98	674.28	44638.81	58.33	9505.56	940
2	2	13-12-2015	0.93	118220.22	794.70	109149.67	130.50	8145.35	804
3	3	06-12-2015	1.08	78992.15	1132.00	71976.41	72.58	5811.16	567
4	4	29-11-2015	1.28	51039.60	941.48	43838.39	75.78	6183.95	598
...	...	...	...	...	...	...	...	...	
18244	7	04-02-2018	1.63	17074.83	2046.96	1529.20	0.00	13498.67	1306
18245	8	28-01-2018	1.71	13888.04	1191.70	3431.50	0.00	9264.84	894
18246	9	21-01-2018	1.87	13766.76	1191.92	2452.79	727.94	9394.11	935
18247	10	14-01-2018	1.93	16205.22	1527.63	2981.04	727.01	10969.54	1091
18248	11	07-01-2018	1.62	17489.58	2894.77	2356.13	224.53	12014.15	1198

18249 rows × 14 columns

Trying to understand the Avocado dataset

In [5]:

```
df.shape      #finding the data shape
```

Out[5]:

```
(18249, 14)
```

In [6]:

```
df.columns   # finding out the name of the columns
```

Out[6]:

```
Index(['Unnamed: 0', 'Date', 'AveragePrice', 'Total Volume', '4046', '422
5',
       '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 't
ype',
       'year', 'region'],
      dtype='object')
```

In [7]:

```
df.info()  #finding the index,data-type,& memory information
```

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

In [8]:

```
df.describe()
```

Out[8]:

	Unnamed: 0	AveragePrice	Total Volume	4046	4225	4770
count	18249.000000	18249.000000	1.824900e+04	1.824900e+04	1.824900e+04	1.824900e+04
mean	24.232232	1.405978	8.506440e+05	2.930084e+05	2.951546e+05	2.283974e+04
std	15.481045	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05
min	0.000000	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	0.000000e+00
25%	10.000000	1.100000	1.083858e+04	8.540700e+02	3.008780e+03	0.000000e+00
50%	24.000000	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02
75%	38.000000	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03
max	52.000000	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06

In [9]:

```
df.isnull().sum() # trying to find out if there is any null value in any columns
```

Out[9]:

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

In [10]:

```
df.drop('Unnamed: 0',axis=1,inplace=True) #dropping unnecessary data from the dataset.
```

In [11]:

```
df.head()
```

Out[11]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLar Ba
0	27-12-2015	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	(
1	20-12-2015	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	(
2	13-12-2015	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	(
3	06-12-2015	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	(
4	29-11-2015	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	(

In [12]:

```
df['Date']=pd.to_datetime(df['Date'])
df['Month']=df['Date'].apply(lambda x:x.month)
df['Day']=df['Date'].apply(lambda x:x.day)
```

In [13]:

```
df.head(10)
```

Out[13]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XI
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	
3	2015-06-12	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	
5	2015-11-22	1.26	55979.78	1184.27	48067.99	43.61	6683.91	6556.47	127.44	
6	2015-11-15	0.99	83453.76	1368.92	73672.72	93.26	8318.86	8196.81	122.05	
7	2015-08-11	0.98	109428.33	703.75	101815.36	80.00	6829.22	6266.85	562.37	
8	2015-01-11	1.02	99811.42	1022.15	87315.57	85.34	11388.36	11104.53	283.83	
9	2015-10-25	1.07	74338.76	842.40	64757.44	113.00	8625.92	8061.47	564.45	

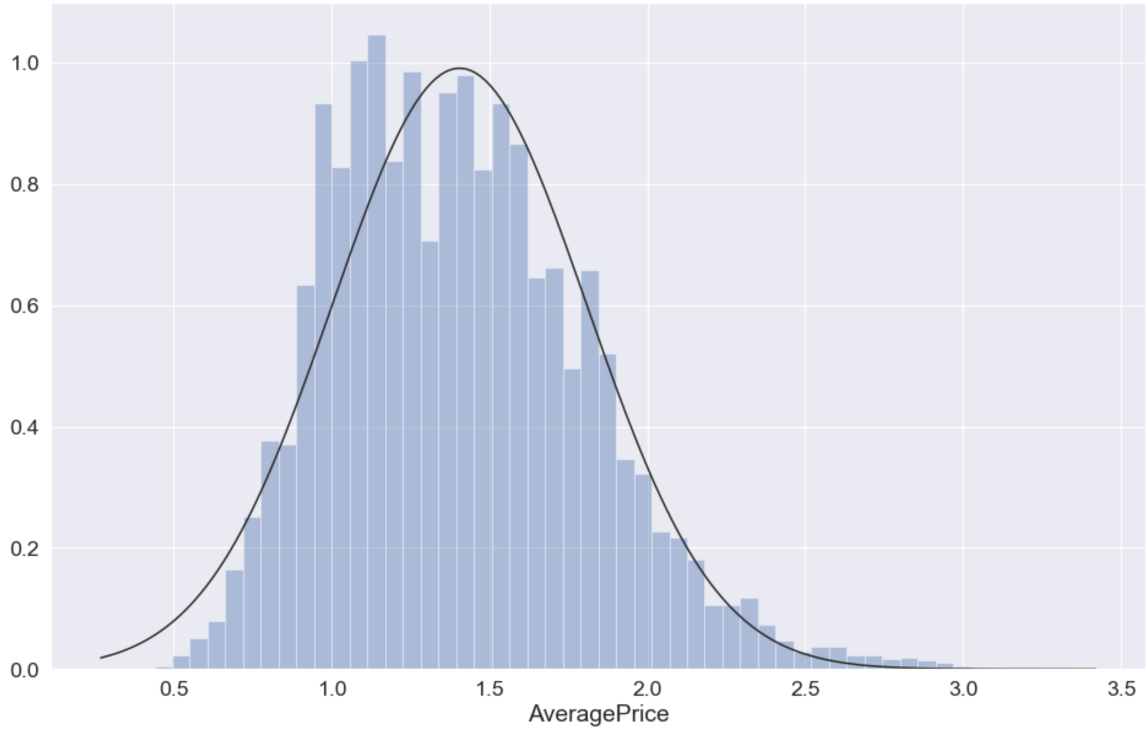
Data Visualization process

In [18]:

```
sns.set(font_scale=1.6)
from scipy.stats import norm
fig, ax = plt.subplots(figsize=(16, 10))
sns.distplot(a=df.AveragePrice, kde=False, fit=norm)
```

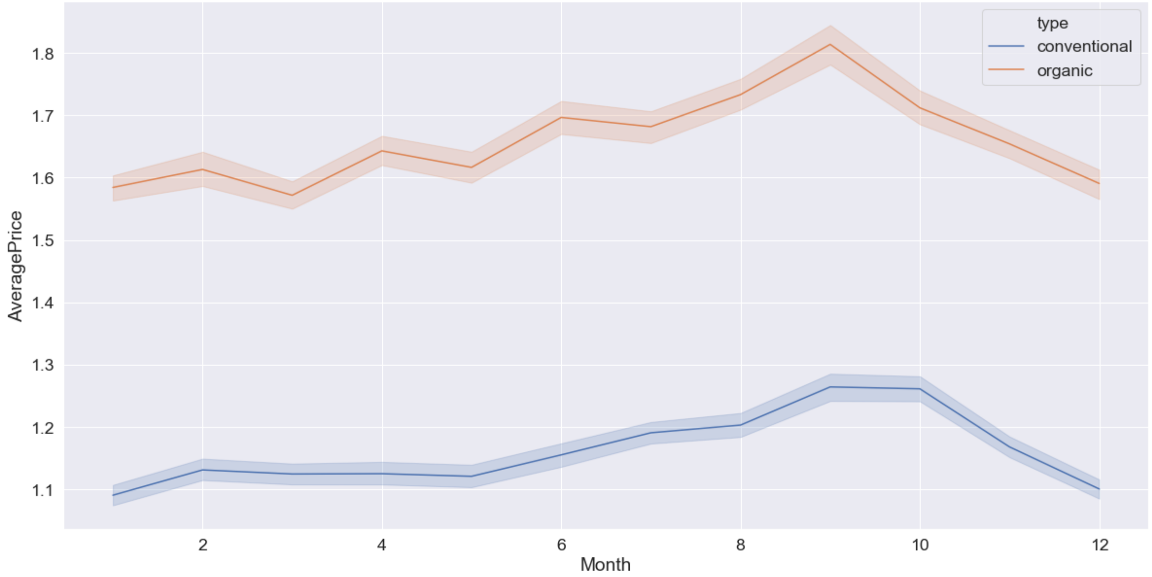
Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2786f4d0f88>
```



In [19]:

```
plt.figure(figsize=(20,10))
sns.lineplot(x="Month", y="AveragePrice", hue='type', data=df)
plt.show()
```

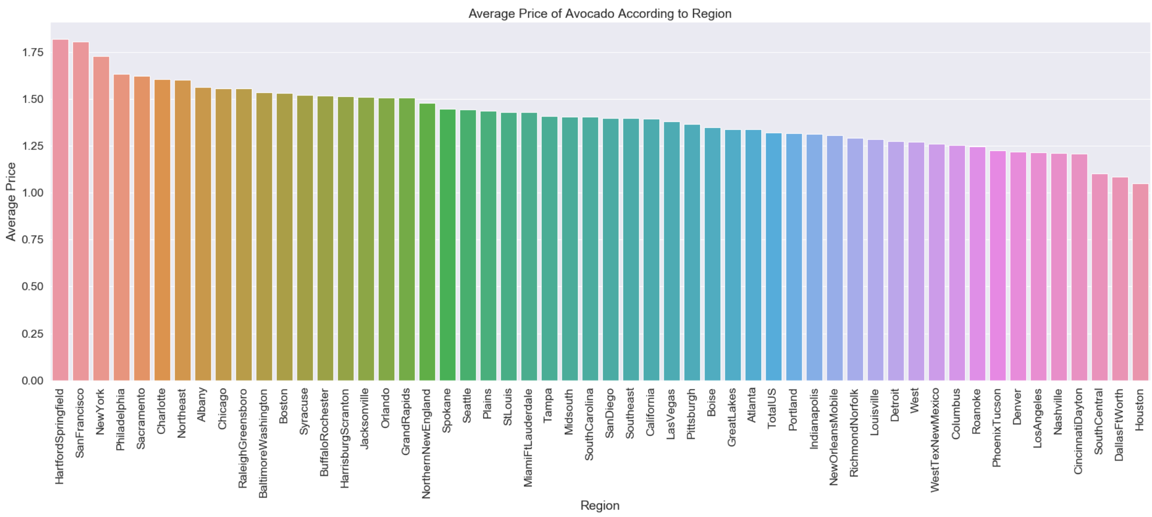


In [21]:

```
region_list=list(df.region.unique())
average_price=[]

for i in region_list:
    x=df[df.region==i]
    region_average=sum(x.AveragePrice)/len(x)
    average_price.append(region_average)

df1=pd.DataFrame({'region_list':region_list,'average_price':average_price})
new_index=df1.average_price.sort_values(ascending=False).index.values
sorted_data=df1.reindex(new_index)

plt.figure(figsize=(30,10))
ax=sns.barplot(x=sorted_data.region_list,y=sorted_data.average_price)

plt.xticks(rotation=90)
plt.xlabel('Region')
plt.ylabel('Average Price')
plt.title('Average Price of Avocado According to Region')
```

Out[21]:

```
Text(0.5, 1.0, 'Average Price of Avocado According to Region')
```

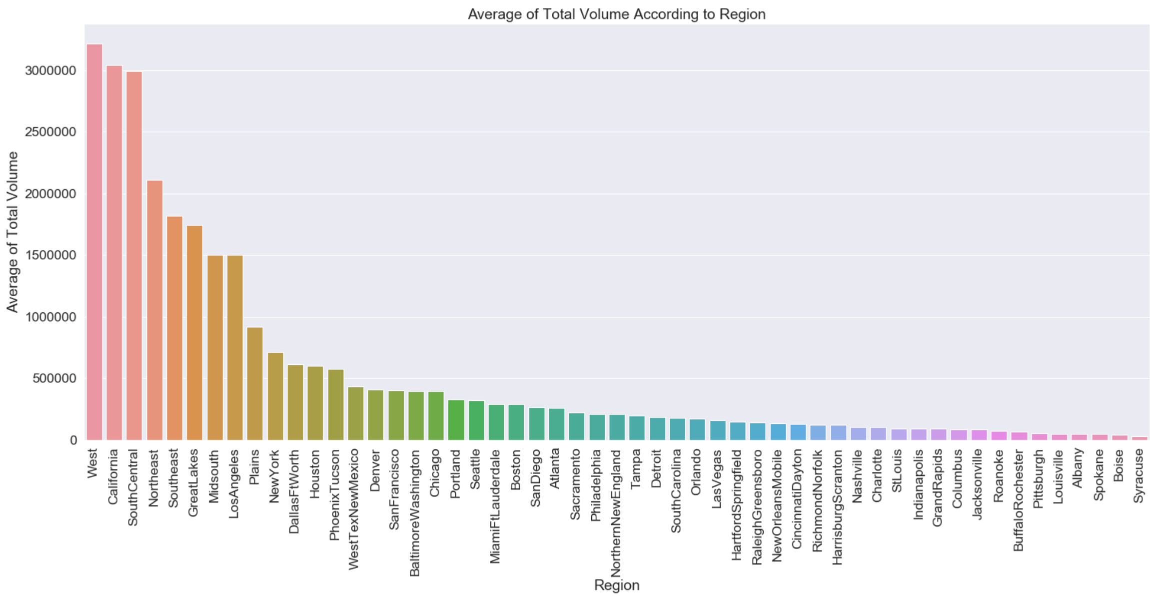


In [22]:

```
filter1=df.region!='TotalUS'
df1=df[filter1]

region_list=list(df1.region.unique())
average_total_volume=[]

for i in region_list:
    x=df1[df1.region==i]
    average_total_volume.append(sum(x['Total Volume'])/len(x))
df3=pd.DataFrame({'region_list':region_list,'average_total_volume':average_total_volume
})

new_index=df3.average_total_volume.sort_values(ascending=False).index.values
sorted_data1=df3.reindex(new_index)

plt.figure(figsize=(25,10))
ax=sns.barplot(x=sorted_data1.region_list,y=sorted_data1.average_total_volume)

plt.xticks(rotation=90)
plt.xlabel('Region')
plt.ylabel('Average of Total Volume')
plt.title('Average of Total Volume According to Region')
```

Out[22]:

```
Text(0.5, 1.0, 'Average of Total Volume According to Region')
```

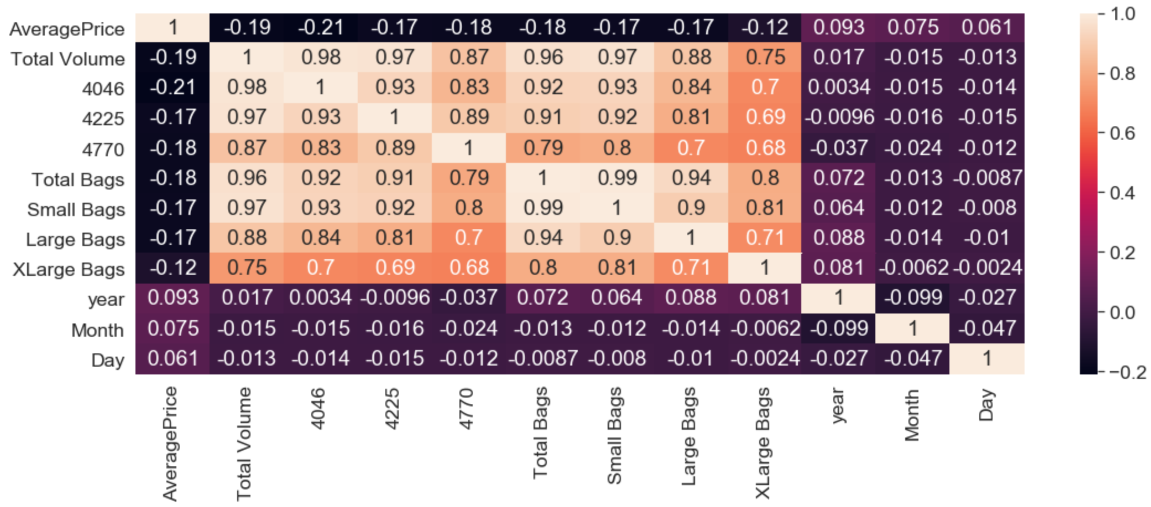


In [25]:

```
plt.figure(figsize=(18,6))
sns.heatmap(df.corr(),annot=True) # Finding how datasets are correlated wth each other
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x278705b9dc8>
```



In [26]:

```
df['region'].nunique()
```

Out[26]:

```
54
```

In [27]:

```
df['type'].nunique()
```

Out[27]:

```
2
```

In [28]:

```
df_data=pd.get_dummies(df.drop(['region','Date'],axis=1),drop_first=True)
```

In [29]:

```
df_data.head(10)
```

Out[29]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0
5	1.26	55979.78	1184.27	48067.99	43.61	6683.91	6556.47	127.44	0.0
6	0.99	83453.76	1368.92	73672.72	93.26	8318.86	8196.81	122.05	0.0
7	0.98	109428.33	703.75	101815.36	80.00	6829.22	6266.85	562.37	0.0
8	1.02	99811.42	1022.15	87315.57	85.34	11388.36	11104.53	283.83	0.0
9	1.07	74338.76	842.40	64757.44	113.00	8625.92	8061.47	564.45	0.0

Finding out the best suitable model for the datasets

In [31]:

```
X=df_data.iloc[:,1:14]
```

In [32]:

```
y=df_data['AveragePrice']
```

In [33]:

```
from sklearn.model_selection import train_test_split
```

In [34]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.12,random_state=42)
```

In [35]:

```
from sklearn.linear_model import LinearRegression
```

In [36]:

```
lr=LinearRegression()
```

In [37]:

```
lr.fit(X_train,y_train)
```

Out[37]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

In [38]:

```
pred=lr.predict(X_test)
```

In [56]:

```
pred
```

Out[56]:

```
array([0.8798, 1.0184, 1.435 , ..., 1.4975, 2.0281, 1.0068])
```

In [39]:

```
from sklearn import metrics
```

In [40]:

```
print('MAE:', metrics.mean_absolute_error(y_test, pred))
```

```
MAE: 0.23617077359997626
```

In [41]:

```
print('MSE:', metrics.mean_squared_error(y_test, pred))
```

```
MSE: 0.09431161995978325
```

In [42]:

```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
RMSE: 0.3071019699705348
```

In [43]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [44]:

```
dr=DecisionTreeRegressor()
```

In [45]:

```
dr.fit(X_train,y_train)
```

Out[45]:

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      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='deprecated',
                      random_state=None, splitter='best')
```

In [47]:

```
pred=dr.predict(X_test)
```

In [55]:

```
pred
```

Out[55]:

```
array([0.8798, 1.0184, 1.435 , ..., 1.4975, 2.0281, 1.0068])
```

In [48]:

```
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.1502511415525114
MSE: 0.05246351598173516
RMSE: 0.22904915625632669
```

In [49]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [50]:

```
rder = RandomForestRegressor()
```

In [51]:

```
rder.fit(X_train,y_train)
```

Out[51]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=
None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [53]:

```
pred=rder.predict(X_test)
```

In [54]:

```
pred
```

Out[54]:

```
array([0.8798, 1.0184, 1.435 , ..., 1.4975, 2.0281, 1.0068])
```

In [57]:

```
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 0.11386292237442923
MSE: 0.02746618539726027
RMSE: 0.16572925329361823
```

Conclusion: From the datasets it assist me to obtain the actionable insights about the data and also which model to choose in a datasets with a normal process flow. it also help me where to use LinearRegression, Decision Tree, and additional required models to find out the predictions of the datasets in the best possible way.