Avocado Dataset Analysis and ML Prediction<center>

Table of Contents

1. Problem Statement

2. Data Loading and Description

3. Data Cleaning

- Let's analyze and clean the data.

3.1 Understanding the Dataset

In [7]:
```
df.shape
```
Out[7]:
```
(18249, 14)
```

In [8]:
```
df.columns  # This will print the names of all columns.
```
Out[8]:
```
Index(['Unnamed: 0', 'Date', 'AveragePrice', 'Total Volume', '4046', '4225',
       '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type',
       'year', 'region'],
      dtype='object')
```

In [9]:
```
df.head()  # Will give you first 5 records
```
Out[9]:

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional

- The data set contians **18249** rows and **14** columns.

In [11]:
```
df.info() # This will give Index, Datatype and Memory information
```
```
<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
```

In [12]:
```
df.describe()
```
Out[12]:

	Unnamed: 0	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags
count	18249.000000	18249.000000	1.824900e+04	1.824900e+04	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	2.396392e+05	1.821947e+05
std	15.481045	0.402677	3.453545e+06	1.264989e+06	1.204120e+06	1.074641e+05	9.862424e+05	7.461785e+05
min	0.000000	0.440000	8.456000e+01	0.000000e+00	0.000000e+00	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	5.088640e+03	2.849420e+03
50%	24.000000	1.370000	1.073768e+05	8.645300e+03	2.906102e+04	1.849900e+02	3.974383e+04	2.636282e+04
75%	38.000000	1.660000	4.329623e+05	1.110202e+05	1.502069e+05	6.243420e+03	1.107834e+05	8.333767e+04
max	52.000000	3.250000	6.250565e+07	2.274362e+07	2.047057e+07	2.546439e+06	1.937313e+07	1.338459e+07

- We can see all columns having count **18249**. Looks like it doesn't contain missing values.

In [14]:
```
df.isnull().sum()  # Will show you null count for each column, but will not count Zeros(0) as null
```
Out[14]:
```
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
```

- We can obsrve that, there are **no missing values** exist in dataset, that's great!

3.2 Profiling

In [25]:
```
profile = pandas_profiling.ProfileReport(df)
profile.to_file(outputfile="avocado_before_preprocessing.html")
```

3.3 Preprocessing

- The Feature **"Unnamed:0"** is represented as a index, so let's remove it from the dataset!

In [26]:
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [27]:
```
df.head()
```
Out[27]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

In [28]:
```
#
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 [29]:
```
df.head()
```
Out[29]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

4. Data Visualisation

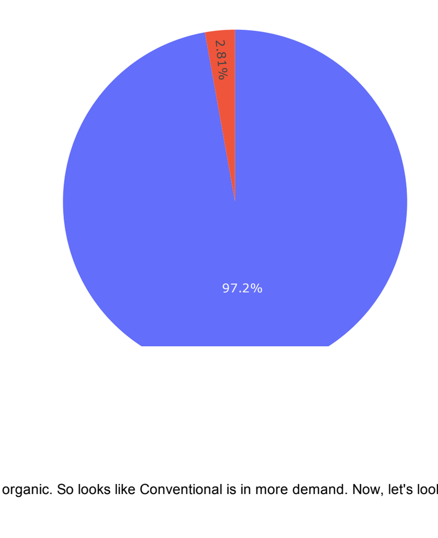
Organic vs Conventional : The main difference between organic and conventional food products are the chemicals involved during production and processing. The interest in organic food products has been rising steadily over the recent years with new health super fruits emerging. Let's see if this is also the case with our dataset

4.1 Which type of Avocados are more in demand (Conventional or Organic)?

In [38]:
```
type=df.groupby('type')['Total Volume'].agg('sum')

values=[type['conventional'],type['organic']]
labels=['conventional','organic']

trace=go.Pie(labels=labels,values=values)
py.iplot([trace])
```



- Just over **2%** of our dataset is organic. So looks like Conventional is in more demand. Now, let's look at the average price distribution

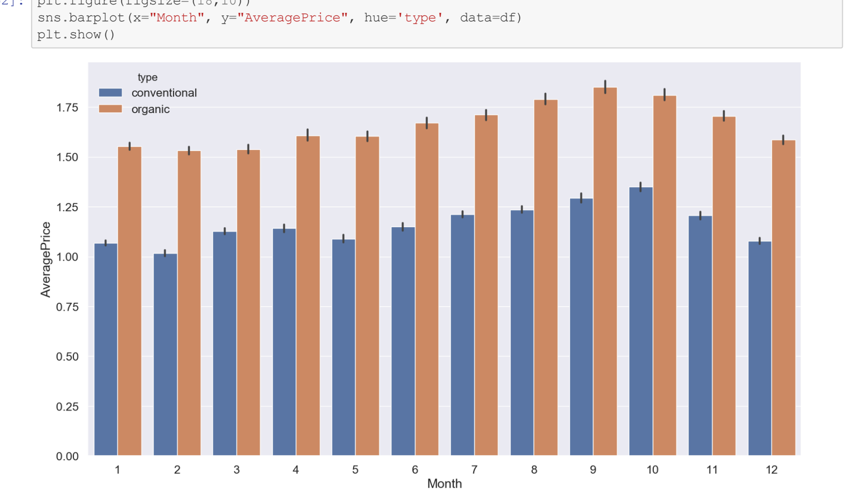
4.2 In which range Average price lies, what is distribution look like?

In [55]:
```
sns.set(font_scale=1.5)
from scipy.stats import norm
fig, ax = plt.subplots(figsize=(10, 5))
sns.distplot(a=df.AveragePrice, kde=False, fit=norm)
```
Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xce72cb3828>
```

- Average Price distribution shows that for most cases price of avocado is between 1.1, 1.4.

4.3 How Average price is distributed over the months for Conventional and Organic Types?

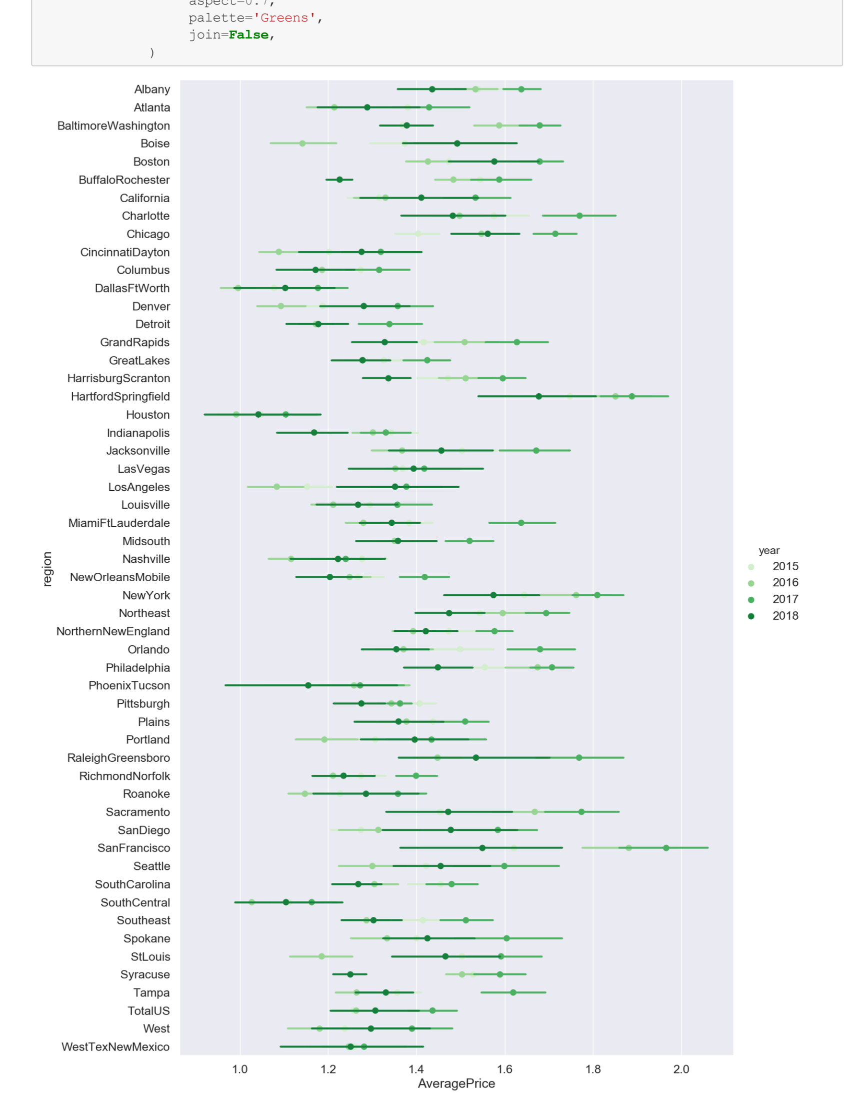
In [52]:
```
plt.figure(figsize=(18,10))
sns.barplot(x="Month", y="AveragePrice", hue='type', data=df)
plt.show()
```



- Looks like there was a hike between **months 8 – 10** for both Conventional and Organic type of Avocados prices

4.4 In which year and for which region was the Average price the highest?

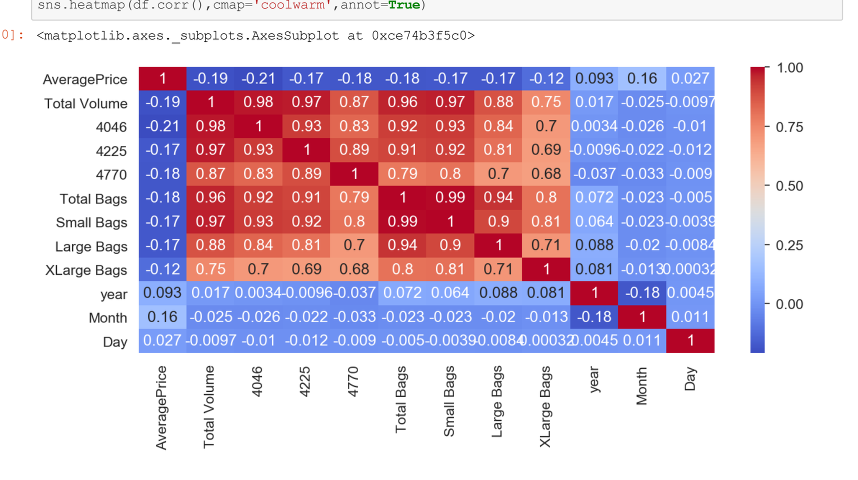
In [64]:
```
g = sns.factorplot('AveragePrice','region',data=df,
                   hue='year',
                   size=20,
                   aspect=0.7,
                   palette='Greens',
                   join=False,
              )
```



- Looks like there was a **huge increase in Avocado prices as the demand was little high in Year 2017 in SanFranciso region**.

4.5 How dataset features are correlated with each other?

In [70]:
```
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True)
```
Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xce74b3f5c0>
```



- As we can see from the heatmap above, all the features are not correleted with the **Average Price column**, instead most of them are correlated with each other, so we can say that it is not a good model.

5. Feature Engineering for Model building

In [71]:
```
df['region'].nunique()
```
Out[71]:
```
54
```

In [72]:
```
df['type'].nunique()
```
Out[72]:
```
2
```

- Type has 2 unique values, so it's going to be easy to to transform the type feature into dummies
- Region has 54 unique values, so I decided to drop the entire column.
- I will drop the Date Feature as well because I already have 3 other columns for the Year, Month and Day.

In [73]:
```
df_final=pd.get_dummies(df.drop(['region','Date'],axis=1),drop_first=True)
```

In [74]:
```
df_final.head()
```
Out[74]:

	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	year	Month	Day	type_organic
0	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	2015	12	27	0
1	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	2015	12	20	0
2	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	2015	12	13	0
3	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	2015	12	6	0
4	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	2015	11	29	0

6. Model selection/predictions

- Now our data is ready. Let's apply Linear Regression Algorithm because our Target variable 'AveragePrice' is continuous.
- Let's now begin training the regression model! We will need to first split the into an X array that contains the features to train on, and a y array with the target variable.

6.1 Average Price prediction using Linear Regression Regressor

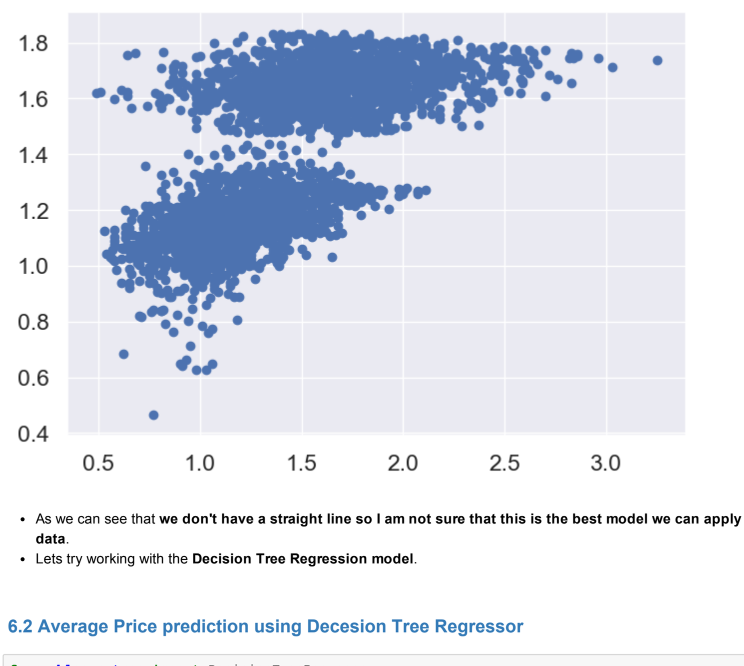
In [91]:
```
X=df_final.iloc[:,1:14]
y=df_final['AveragePrice']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [92]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
pred=lr.predict(X_test)
```

In [93]:
```
from sklearn import metrics
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: 0.23297133291665678
MSE: 0.09108802805350158
RMSE: 0.3018079323899582
```

- The **RMSE is low so we can say that we do have a good model, but lets check to be more sure**.
- Lets plot the **y_test vs the predictions**.

In [82]:
```
plt.scatter(x=y_test,y=pred)
```
Out[82]:
```
<matplotlib.collections.PathCollection at 0xce76b4a2e8>
```

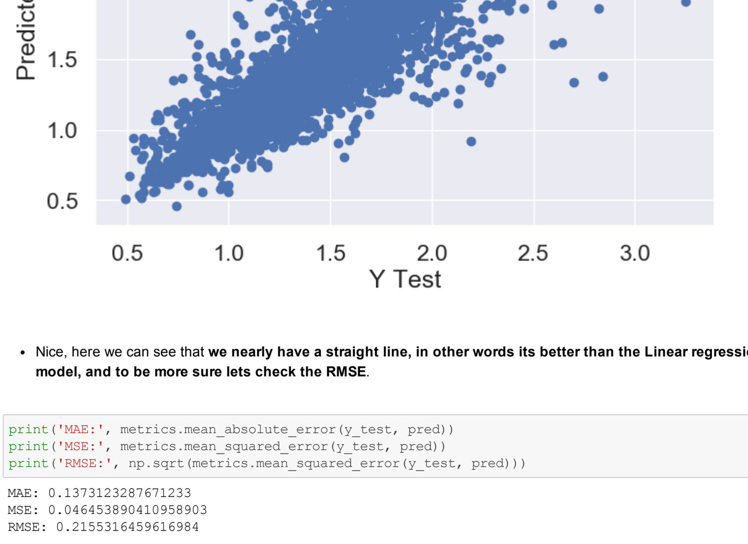


- As we can see that **we don't have a straight line so I am not sure that this is the best model we can apply on our data**.
- Lets try working with the **Decision Tree Regression model**.

6.2 Average Price prediction using Decesion Tree Regressor

In [95]:
```
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
dtr.fit(X_train,y_train)
pred=dtr.predict(X_test)
```

In [96]:
```
plt.scatter(x=y_test,y=pred)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```
Out[96]:
```
Text(0, 0.5, 'Predicted Y')
```



- Nice, here we can see that **we nearly have a straight line, in other words its better than the Linear regression model, and to be more sure lets check the RMSE**.

In [87]:
```
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: 0.1373123287671233
MSE: 0.046453890410958903
RMSE: 0.2155316459616984
```

- Very Nice, **RMSE is lower than as comapred to Linear Regression**.
- Now let's try one last model to see if the predictions are improved. For this, we will use **Random Forest Regressor. **

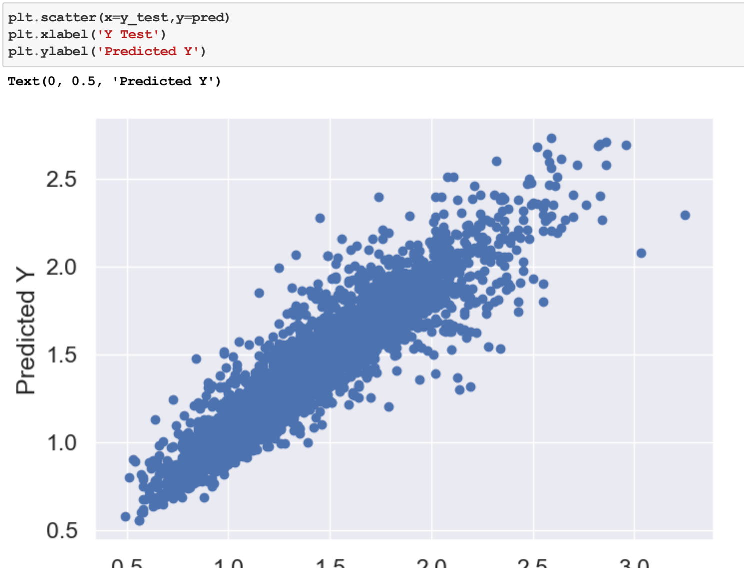
6.3 Average Price prediction using Random Forest Regressor

In [99]:
```
from sklearn.ensemble import RandomForestRegressor
rdr = RandomForestRegressor()
rdr.fit(X_train,y_train)
pred=rdr.predict(X_test)
```

In [89]:
```
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: 0.10660109589041096
MSE: 0.024027293698630138
RMSE: 0.15500739885124884
```

- **Well as we can see the RMSE is lower than the two previous models, so the Random Forest Regressor is the best model in this case.**

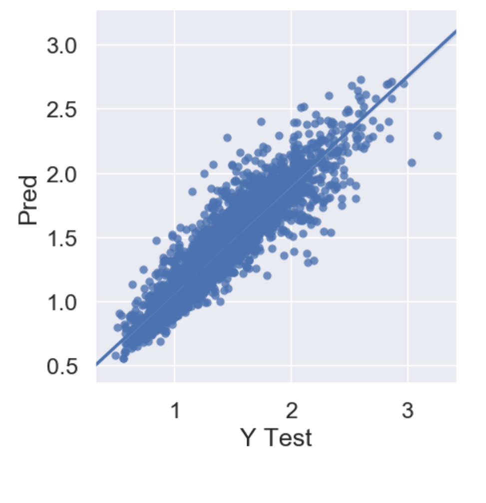
In [100]:
```
plt.scatter(x=y_test,y=pred)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```
Out[100]:
```
Text(0, 0.5, 'Predicted Y')
```



6.4 Lets see final Actual Vs Predicted sample.

In [122]:
```
data = pd.DataFrame({'Y Test':y_test , 'Pred':pred},columns=['Y Test','Pred'])
sns.lmplot(x='Y Test',y='Pred',data=data,palette='rainbow')
data.head(5)
```
Out[122]:

	Y Test	Pred
8604	0.82	0.918
2608	0.97	1.026
14581	1.44	1.416
4254	0.97	0.905
16588	1.45	1.444



7. Conclusions

- **Used Pandas and other plotting libraries like matplotlib, plotly and seaborn for EDA.**
- **Used profiler for data cleaning and identifying missing and incorrect values present in the data.**
- **Used dummification for categorical values like type.**
- **The most important inference drawn from all this analysis is, I get to know what are the features on which price is highly correlated with.**
- **Used various algorithms to know which model will work better with the help of low RMSE scores.**
- **This project helped me to gain lots of insights on how to choose the best model for predcition.**
- **Also get to know how to use Linear Regression, Decision Tree and Random Forest models to fine tune the predictions.**