AVACADO PRICES PREDICTION

OBJECTIVE OF THE REPORT

Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation:

Accurate forecast of Avocado prices.

DESCRIPTION OF DATASET

Brief description of the data set you chose and a summary of its attributes

The table below represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass, club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

Some relevant columns in the dataset:

- Date - The date of the observation
- Average Price - the average price of a single avocado
- type - conventional or organic
- year - the year
- Region - the city or region of the observation
- Total Volume - Total number of avocados sold
- 4046 - Total number of avocados with PLU 4046 sold
- 4225 - Total number of avocados with PLU 4225 sold
- 4770 - Total number of avocados with PLU 4770 sold

PRE-DATA PROCESSING

Brief summary of data exploration and actions taken for data cleaning and feature engineering.

Columns_Name	No of msiing Values
Unnamed	0
Date	0
Average Price	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
year	0
region	0

- It was found that there no missing values.
- The whitespaces from the column names were removed.

Descriptive Statistics of the Variables

	count	mean	std	min	25%	50%	75%	max
AveragePrice	18249.000000	1.405978	0.402677	0.440000	1.100000	1.370000	1.660000	3.250000
Total Volume	18249.000000	850644.013009	3453545.355399	84.560000	10838.580000	107376.760000	432962.290000	62505646.520000
4046	18249.000000	293008.424531	1264989.081763	0.000000	854.070000	8645.300000	111020.200000	22743616.170000
4225	18249.000000	295154.568356	1204120.401135	0.000000	3008.780000	29061.020000	150206.860000	20470572.610000
4770	18249.000000	22839.735993	107464.068435	0.000000	0.000000	184.990000	6243.420000	2546439.110000
Total Bags	18249.000000	239639.202060	986242.399216	0.000000	5088.640000	39743.830000	110783.370000	19373134.370000
Small Bags	18249.000000	182194.686696	746178.514962	0.000000	2849.420000	26362.820000	83337.670000	13384586.800000
Large Bags	18249.000000	54338.088145	243965.964547	0.000000	127.470000	2647.710000	22029.250000	5719096.610000
XLarge Bags	18249.000000	3106.426507	17692.894652	0.000000	0.000000	0.000000	132.500000	551693.650000

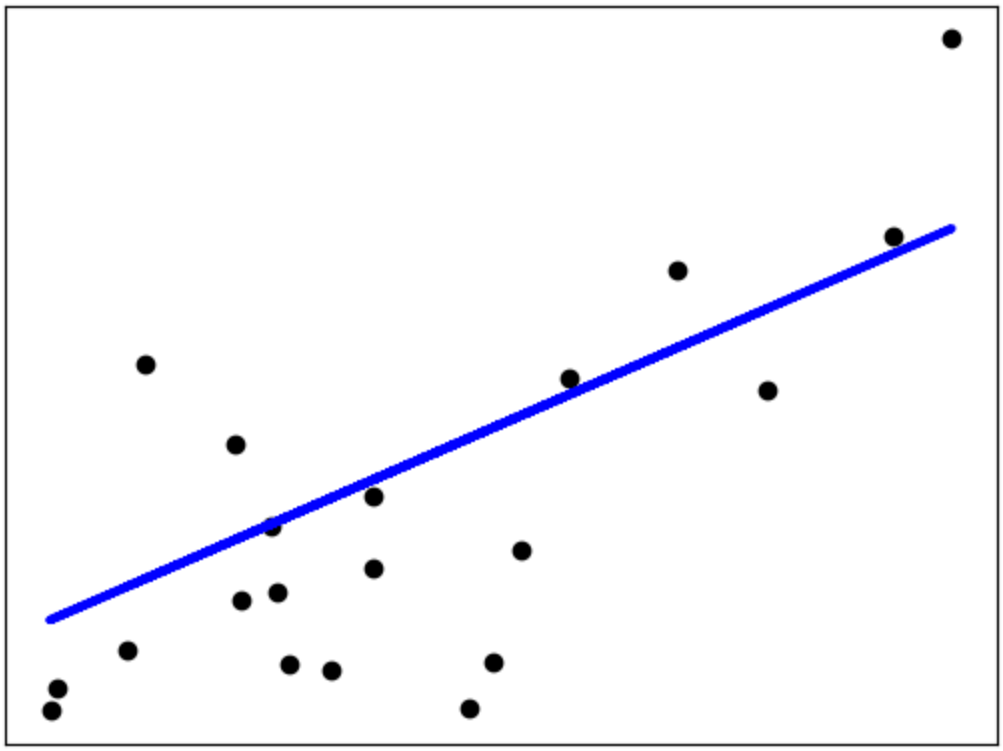
MODEL FITTING

Summary of training at least three linear regression models which should be variations that cover using a simple linear regression as a baseline, adding polynomial effects, and using a regularization regression. Preferably, all use the same training and test splits, or the same cross-validation method

Linear Regression-Ordinary Least Squares

LinearRegression fits a linear model with coefficients

to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:



Python code for Linear regression

```
# Create linear regression object
regr=linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train,y_train)

# Make predictions using the testing set
y_pred=regr.predict(x_test)

# The coefficients
print("Coefficients: \n",regr.coef_)

# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test,y_pred))
```

Polynomial Regression

Polynomial regression: extending linear models with basis functions

One common pattern within machine learning is to use linear models trained on nonlinear functions of the data. This approach maintains the generally fast performance of linear methods, while allowing them to fit a much wider range of data.

For example, a simple linear regression can be extended by constructing polynomial features from the coefficients.

```
# Polynomial regression model

features = PolynomialFeatures(degree=4)
x_train_transformed = features.fit_transform(x_train)
model = LinearRegression()
model.fit(x_train_transformed, y_train)


x_test_transformed = features.fit_transform(x_test)

train_pred = model.predict(x_train_transformed)
rmse_poly_4_train = mean_squared_error(y_train, train_pred, squared = False)
print("Train RMSE for Polynonial Regression of degree 4 is {}.".format(rmse_poly_4_tra

#test_pred = model.predict(x_test_transformed)
#rmse_poly_4 = mean_squared_error(y_test, test_pred, squared = False)
#print("Test RMSE for Polynonial Regression of degree 4 is {}.".format(rmse_poly_4))
```

Regularization techniques in the Linear regression

```
from sklearn import linear_model
reg = linear_model.Lasso(alpha=0.1)
reg.fit(x_train,y_train)
```

MODEL SELECTION

A paragraph explaining which of your regressions you recommend as a final model that best fits your needs in terms of accuracy and explainability.

The RMSE of linear regression is 15 %.

The RMSE of Polynomial features is 37%.

The linear regression model was final chosen to be the model to be deployed.