Predicting the Prices of Avacados

In [1]:
```
from IPython.display import Image
url = "C:/Users/JANHAVI/Desktop/FSDS/29th- REGRESSION PROJECT/RESUME PROJECT -- PRI
Image(url, height=300, width=400)
```

Out[1]:

In [2]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
#importing the dataset
data = pd.read_csv(r"C:\Users\JANHAVI\Desktop\avocado.csv",index_col=0)
# Check the data
data.info()
```

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

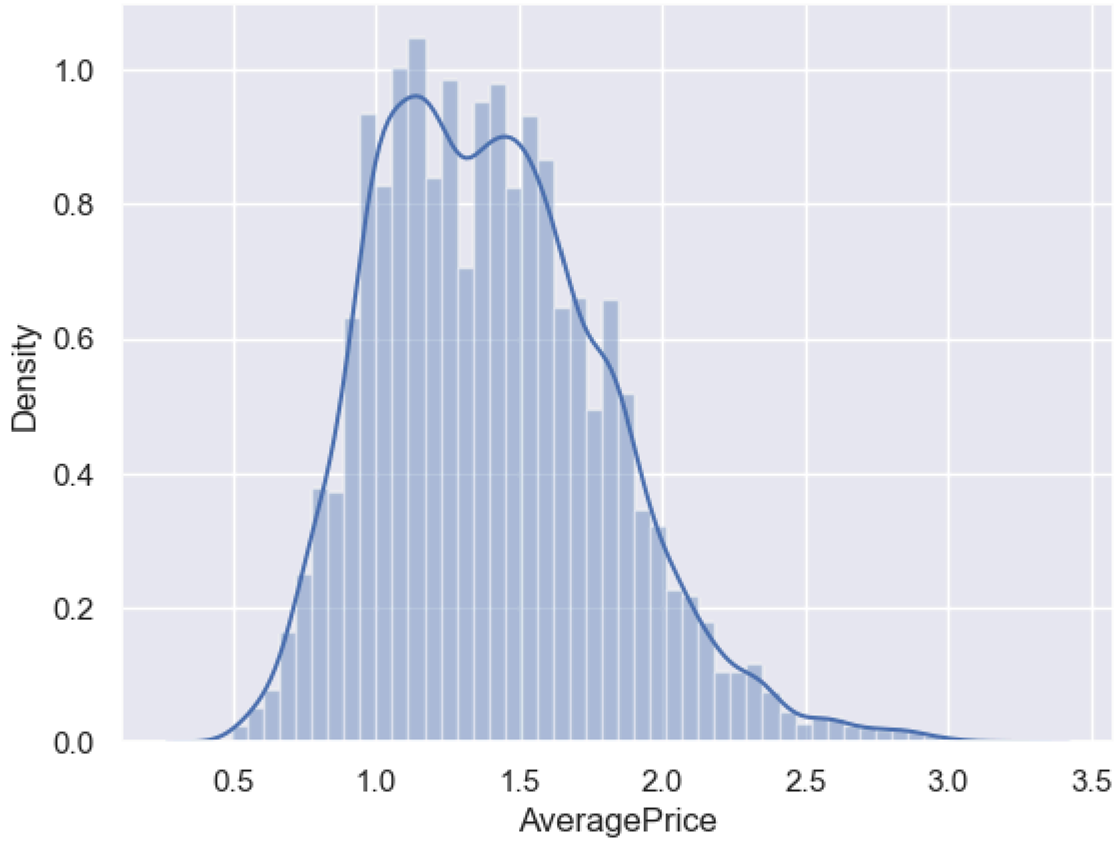
In [3]:
```
data.head(3)
```

Out[3]:

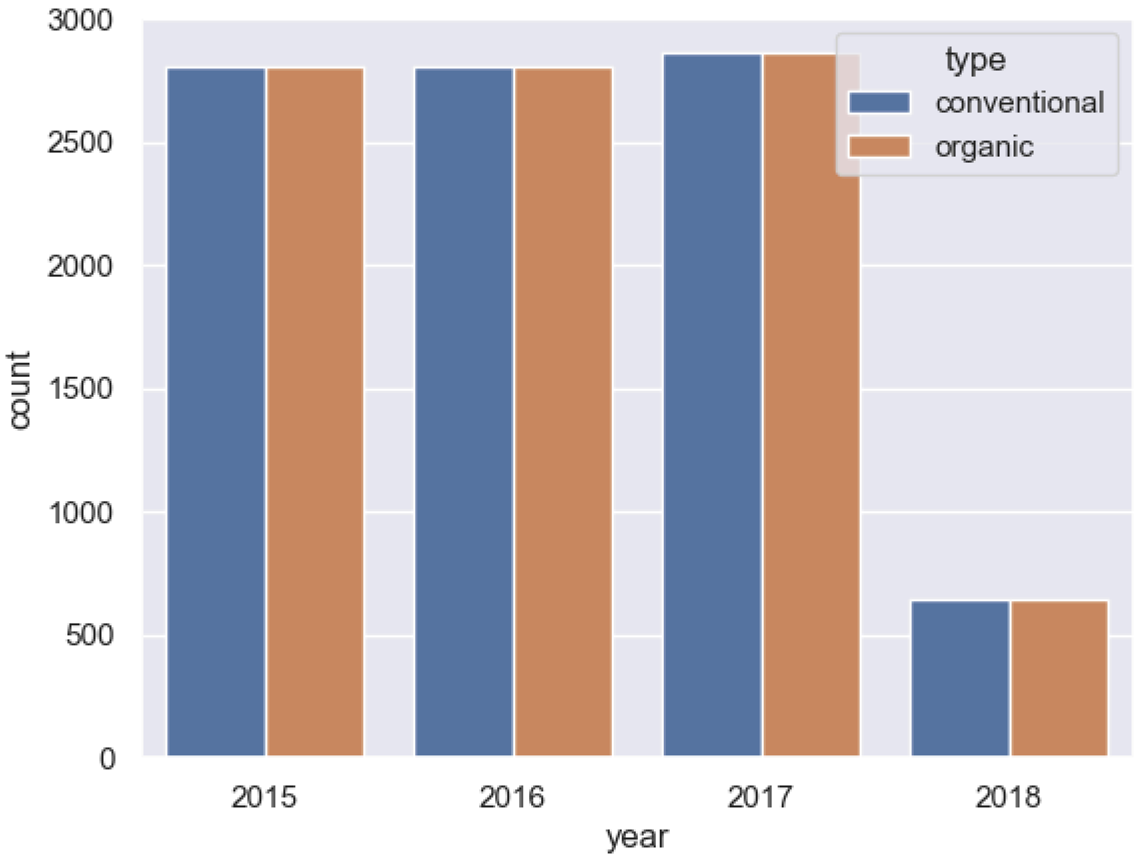
	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	
0	27-12-2015	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	cc
1	20-12-2015	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	cc
2	13-12-2015	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	cc

In [4]:
```
sns.distplot(data['AveragePrice'])
```

Out[4]:
```
<Axes: xlabel='AveragePrice', ylabel='Density'>
```



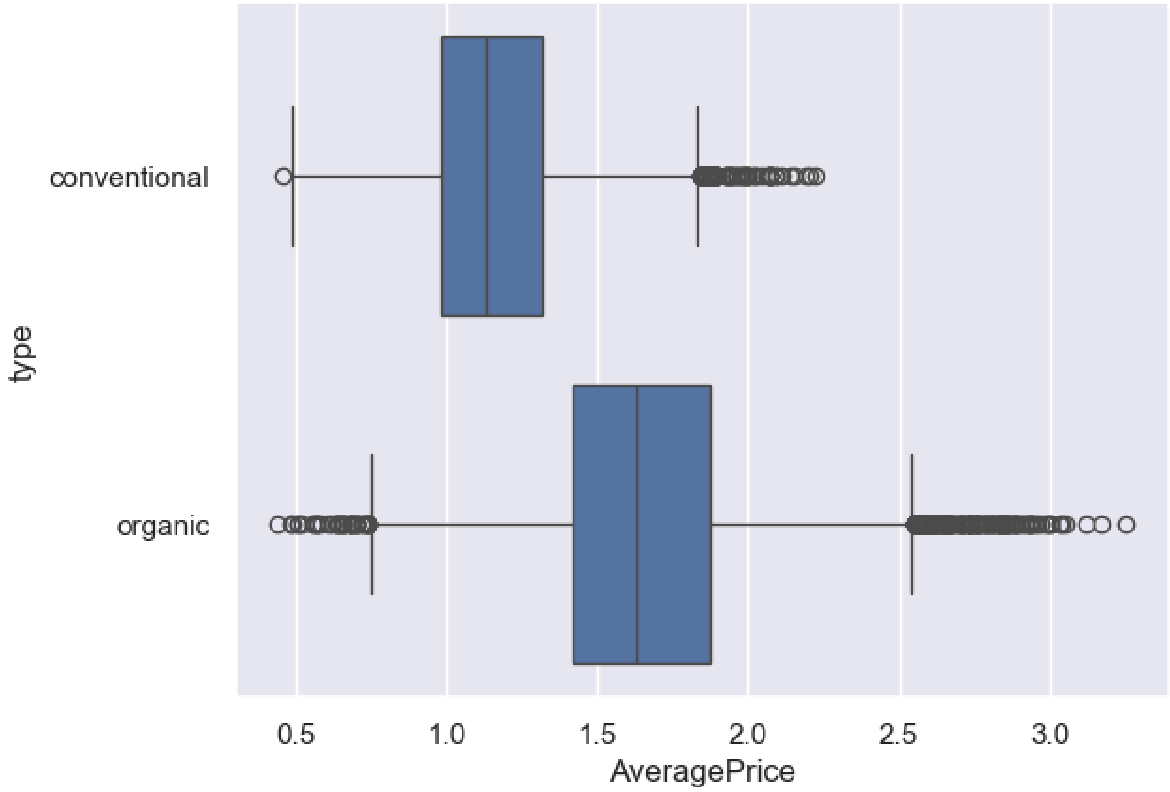
In [5]:
```
sns.countplot(x='year',data=data,hue='type');
```



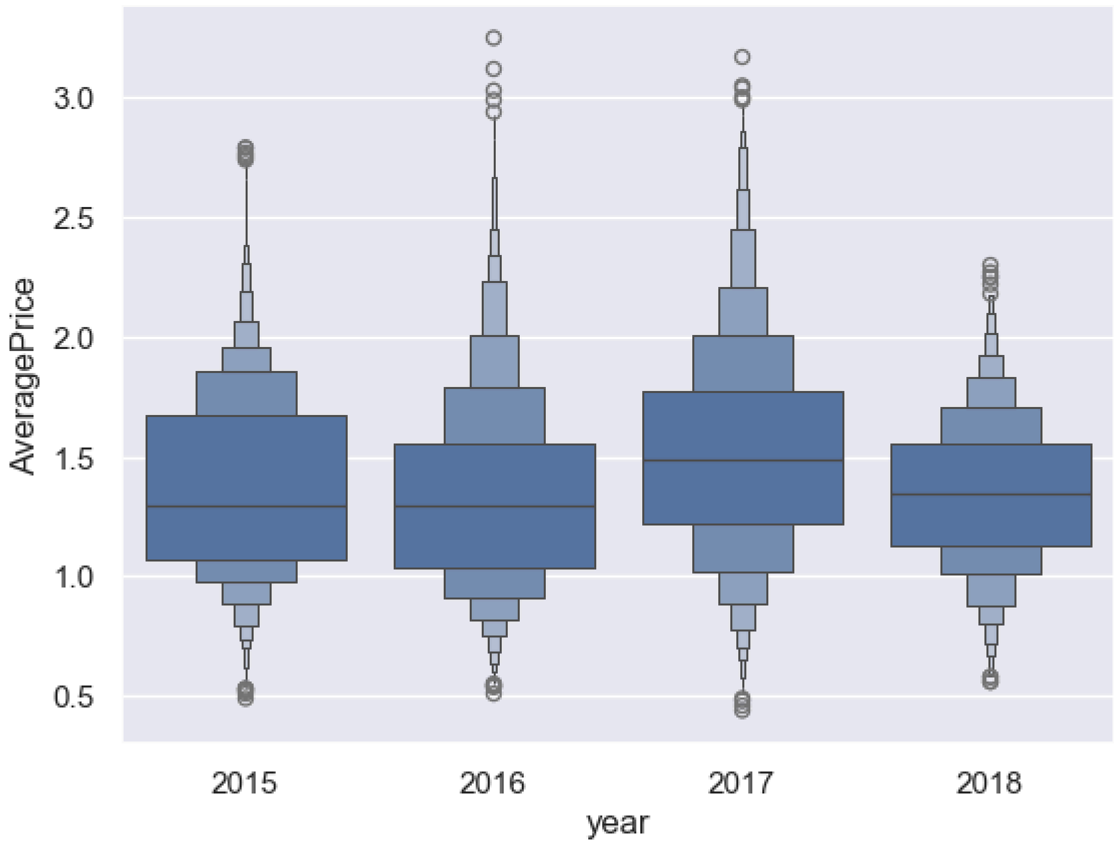
In [6]:
```
data.year.value_counts()
```

Out[6]:
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```

In [7]:
```
sns.boxplot(y="type", x="AveragePrice", data=data);
```



In [8]:
```
data.year=data.year.apply(str)
sns.boxenplot(x="year", y="AveragePrice", data=data);
```



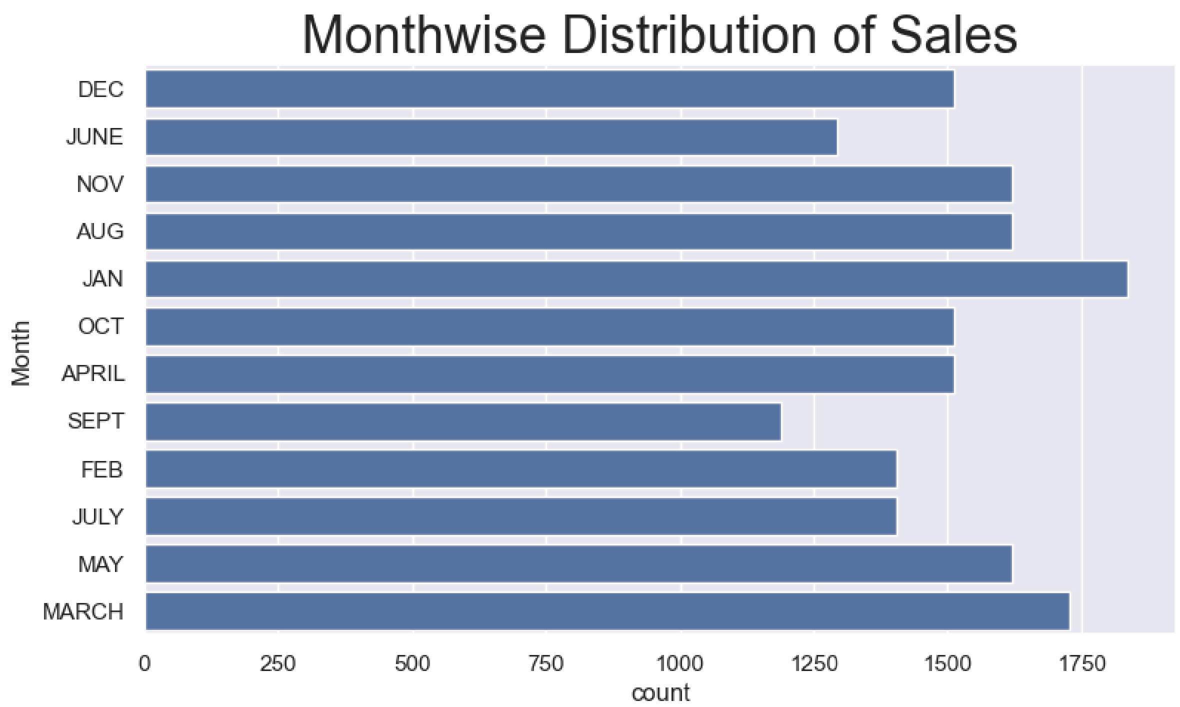
Categorical Features

In [9]:
```
data['type']= data['type'].map({'conventional':0,'organic':1})
```

```
# Extracting month from date column.
data.Date = data.Date.apply(pd.to_datetime)
data['Month']=data['Date'].apply(lambda x:x.month)
data.drop('Date',axis=1,inplace=True)
data.Month = data.Month.map({1:'JAN',2:'FEB',3:'MARCH',4:'APRIL',5:'MAY',6:'JUNE',7
```

In [10]:

```
plt.figure(figsize=(9,5))
sns.countplot(data['Month'])
plt.title('Monthwise Distribution of Sales',fontdict={'fontsize':25});
```



Preparing Data for ML Models

In [11]:

```
# Creating dummy variables
dummies = pd.get_dummies(data[['year','region','Month']],drop_first=True)
df_dummies = pd.concat([data[['Total Volume', '4046', '4225', '4770', 'Total Bags',
       'Small Bags', 'Large Bags', 'XLarge Bags', 'type']],dummies],axis=1)
target = data['AveragePrice']

# Splitting data into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_dummies,target,test_size=0.3

# Standardizing the data
cols_to_std = ['Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags','
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train[cols_to_std])
X_train[cols_to_std] = scaler.transform(X_train[cols_to_std])
X_test[cols_to_std] = scaler.transform(X_test[cols_to_std])
```

In [12]:
```
#importing ML models from scikit-learn
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [13]:
```
#to save time all models can be applied once using for loop
regressors = {
    'Linear Regression' : LinearRegression(),
    'Decision Tree' : DecisionTreeRegressor(),
    'Random Forest' : RandomForestRegressor(),
    'Support Vector Machines' : SVR(gamma=1),
    'K-nearest Neighbors' : KNeighborsRegressor(n_neighbors=1),
    'XGBoost' : XGBRegressor()
}
results=pd.DataFrame(columns=['MAE','MSE','R2-score'])
for method,func in regressors.items():
    model = func.fit(X_train,y_train)
    pred = model.predict(X_test)
    results.loc[method]= [np.round(mean_absolute_error(y_test,pred),3),
                          np.round(mean_squared_error(y_test,pred),3),
                          np.round(r2_score(y_test,pred),3)
                         ]
```

Deep Neural Network

In [40]:
```
# Splitting train set into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.20)

#importing tensorflow libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

#creating model
model = Sequential()
model.add(Dense(76,activation='relu',kernel_initializer=tf.random_uniform_initializ
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initiali
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initiali
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(200,activation='relu',kernel_initializer=tf.random_uniform_initiali
    bias_initializer=tf.random_uniform_initializer(minval=-0.1, maxval=0.1)))
model.add(Dropout(0.5))
model.add(Dense(1))

model.compile(optimizer='Adam', loss='mean_squared_error')
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=0, patience=10)
```

In [41]:
```
print(X_train.dtypes)
print(y_train.dtypes)
X_train = X_train.astype("float32")
```

```
X_val = X_val.astype("float32")
y_train = y_train.astype("float32")
y_val = y_val.astype("float32")
y_train = y_train.astype("float32")
y_val = y_val.astype("float32")
model.fit(
    x=X_train,
    y=y_train,
    validation_data=(X_val, y_val),
    batch_size=100,
    epochs=150,
    callbacks=[early_stop]
)
```

```
Total Volume    float32
4046            float32
4225            float32
4770            float32
Total Bags      float32
                 ...
Month_MARCH     float32
Month_MAY       float32
Month_NOV       float32
Month_OCT       float32
Month_SEPT      float32
Length: 76, dtype: object
float32
Epoch 1/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.4281 - val_loss: 0.1173
Epoch 2/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.1525 - val_loss: 0.1040
Epoch 3/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1288 - val_loss: 0.0671
Epoch 4/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1135 - val_loss: 0.0810
Epoch 5/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.1049 - val_loss: 0.0586
Epoch 6/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0998 - val_loss: 0.0552
Epoch 7/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0998 - val_loss: 0.0888
Epoch 8/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0885 - val_loss: 0.0673
Epoch 9/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0845 - val_loss: 0.0538
Epoch 10/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0845 - val_loss: 0.0565
Epoch 11/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0783 - val_loss: 0.0520
Epoch 12/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0769 - val_loss: 0.0543
Epoch 13/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0776 - val_loss: 0.0492
Epoch 14/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0759 - val_loss: 0.0502
Epoch 15/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0713 - val_loss: 0.0510
Epoch 16/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0684 - val_loss: 0.0521
Epoch 17/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0655 - val_loss: 0.0506
Epoch 18/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0677 - val_loss: 0.0501
Epoch 19/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0646 - val_loss: 0.0517
Epoch 20/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0656 - val_loss: 0.0497
Epoch 21/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0614 - val_loss: 0.0523
Epoch 22/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0626 - val_loss: 0.0551
Epoch 23/150
42/42 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0627 - val_loss: 0.0507
```

Out[41]:

```
<keras.src.callbacks.history.History at 0x1bc3e87e8d0>
```

In [42]:

```python
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot();
```

In [43]:
```
dnn_pred = model.predict(X_test)
```

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 891us/step
```

In [44]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

results.loc['Deep Neural Network'] = [
    round(mean_absolute_error(y_test, dnn_pred), 3),
    round(mean_squared_error(y_test, dnn_pred), 3),
    round(r2_score(y_test, dnn_pred), 3)
]

results
```

Out[44]:

	MAE	MSE	R2-score
Linear Regression	0.191	0.066	0.598
Decision Tree	0.143	0.048	0.706
Random Forest	0.108	0.024	0.854
Support Vector Machines	0.160	0.054	0.669
K-nearest Neighbors	0.156	0.061	0.629
XGBoost	0.114	0.025	0.849
Deep Neural Network	0.172	0.056	0.658

In [45]:
```
f"10% of mean of target variable is {np.round(0.1 * data.AveragePrice.mean(),3)}"
```

Out[45]:
```
'10% of mean of target variable is 0.141'
```

In [46]:
```
results.sort_values('R2-score',ascending=False).style.background_gradient(cmap='Gre
```

Out[46]:

	MAE	MSE	R2-score
Random Forest	0.108000	0.024000	0.854000
XGBoost	0.114000	0.025000	0.849000
Decision Tree	0.143000	0.048000	0.706000
Support Vector Machines	0.160000	0.054000	0.669000
Deep Neural Network	0.172000	0.056000	0.658000
K-nearest Neighbors	0.156000	0.061000	0.629000
Linear Regression	0.191000	0.066000	0.598000

In [ ]: