In [1]:
```
from IPython.display import Image
url =( r"C:\Users\Sonu\OneDrive\Desktop\NIT\29th- REGRESSION PROJECT\29th- REGRESSI
Image(url,height=300,width=400)
```

Out[1]:

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#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\Sonu\OneDrive\Desktop\NIT\29th- REGRESSION PROJECT\29
# Check the data
data.info()
```

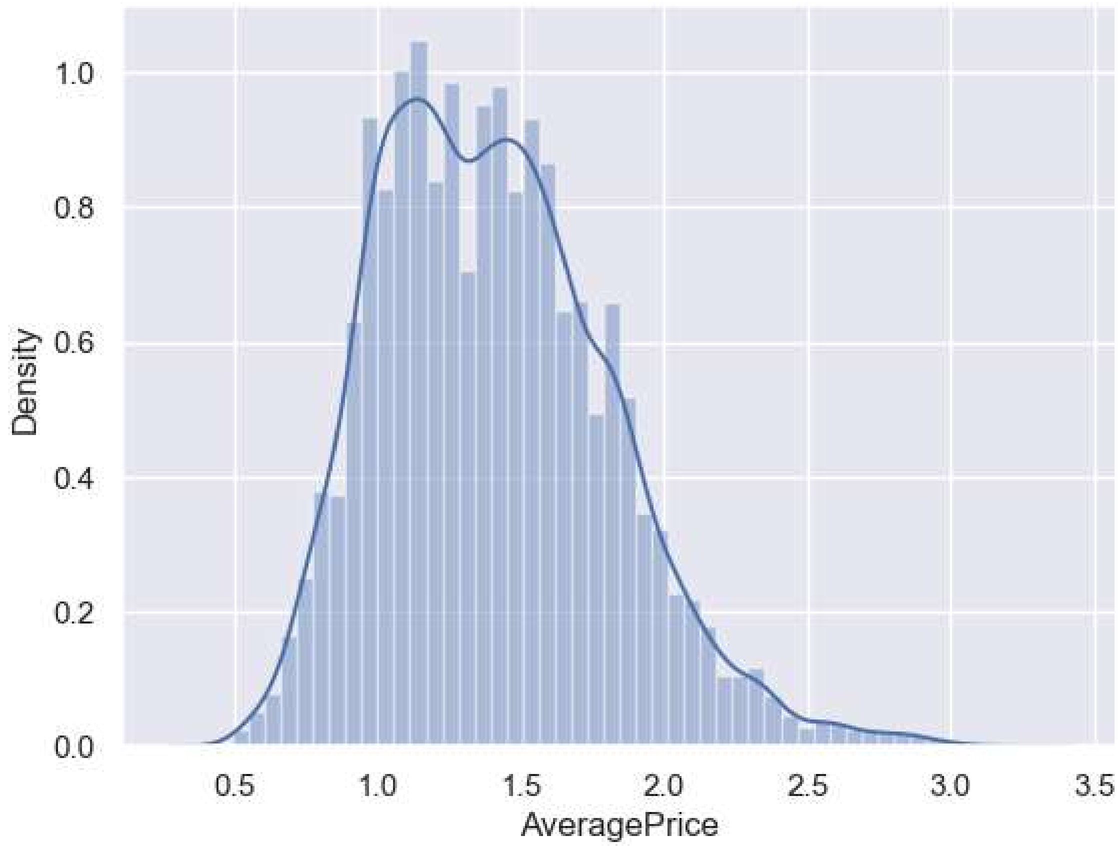
```
<class 'pandas.core.frame.DataFrame'>
Index: 18249 entries, 0 to 11
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Date          18249 non-null  object 
 1   AveragePrice  18249 non-null  float64
 2   Total Volume  18249 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 [4]:
```
data.head(3)
```

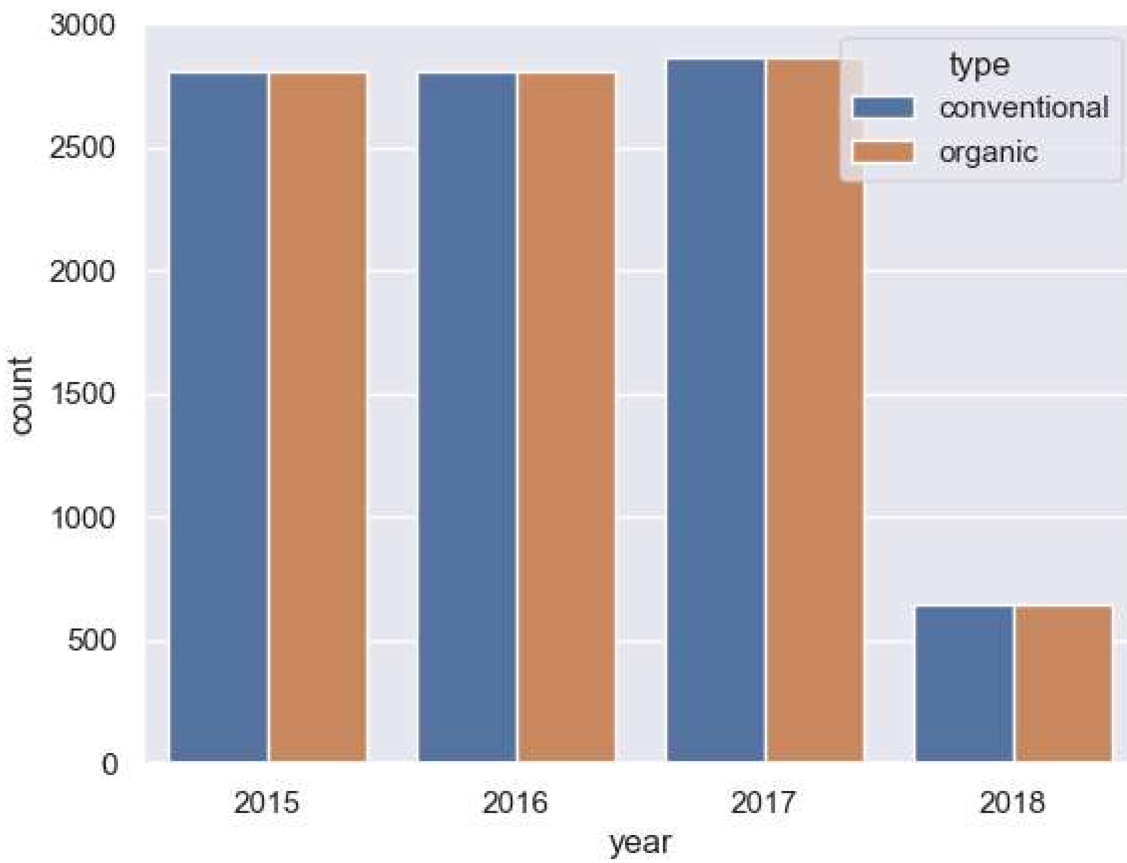
Out[4]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XL
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	

In [5]:
```
sns.distplot(data['AveragePrice']);
```



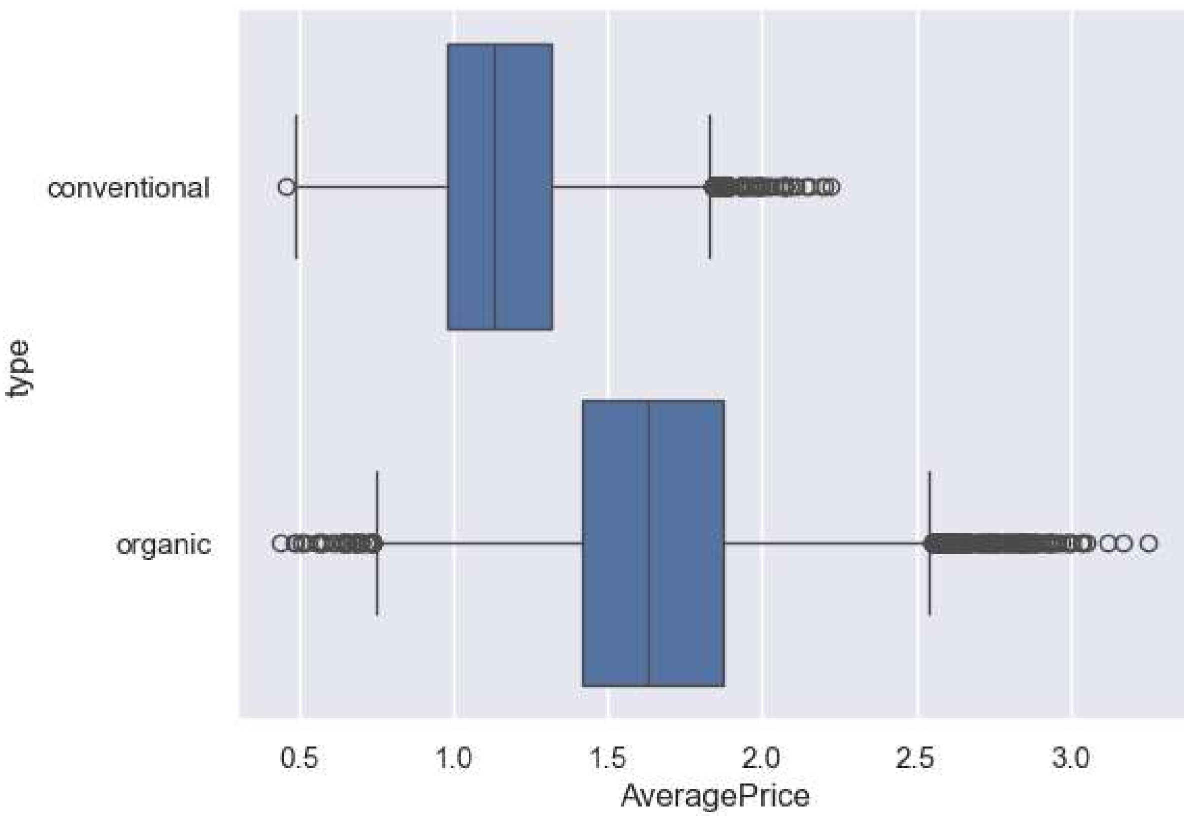
In [6]:
```
sns.countplot(x='year',data=data,hue='type');
```



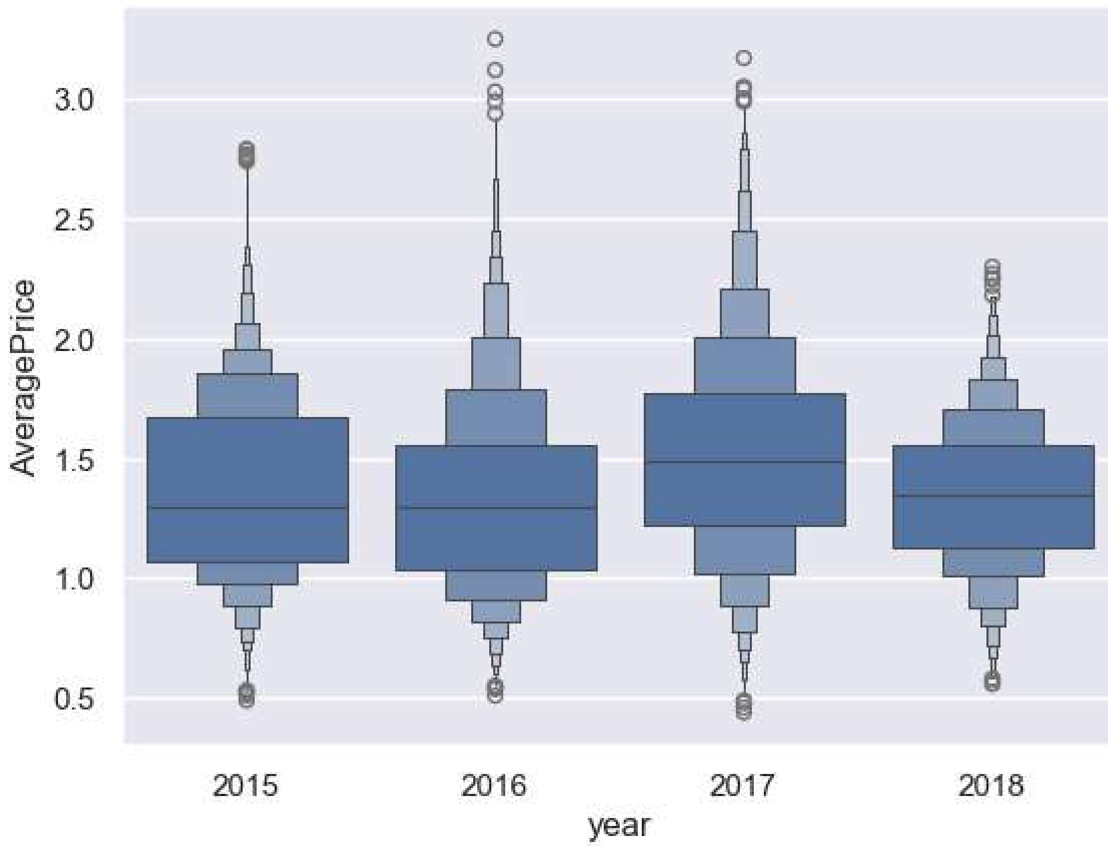
In [7]:
```
data.year.value_counts()
```

Out[7]:
```
year
2017    5722
2016    5616
2015    5615
2018    1296
Name: count, dtype: int64
```

In [8]:
```
sns.boxplot(y="type", x="AveragePrice", data=data);
```



In [9]:
```
data.year=data.year.apply(str)
sns.boxenplot(x="year", y="AveragePrice", data=data);
```

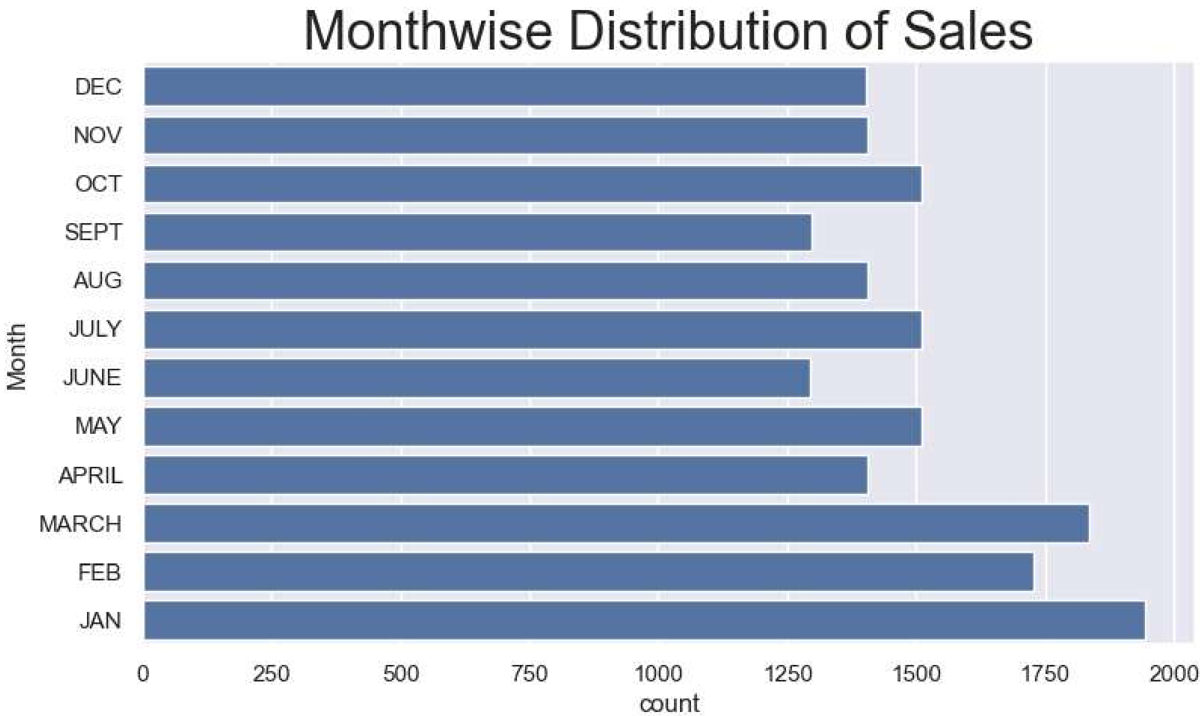


Dealing with categorical features

In [10]:
```
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 [11]:
```
plt.figure(figsize=(9,5))
sns.countplot(data['Month'])
plt.title('Monthwise Distribution of Sales',fontdict={'fontsize':25});
```



Preparing data for ML models

In [12]:
```
# 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 [13]:
```
#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 [14]:
```
#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)
                         ]
```

```
  File "C:\Users\Sonu\anaconda3\Lib\site-packages\joblib\externals\loky\backend\cont
ext.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^^
  File "C:\Users\Sonu\anaconda3\Lib\subprocess.py", line 548, in run
    with Popen(*popenargs, **kwargs) as process:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Sonu\anaconda3\Lib\subprocess.py", line 1026, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\Sonu\anaconda3\Lib\subprocess.py", line 1538, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Deep Neural Network

In [15]:

```
# 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 [17]:

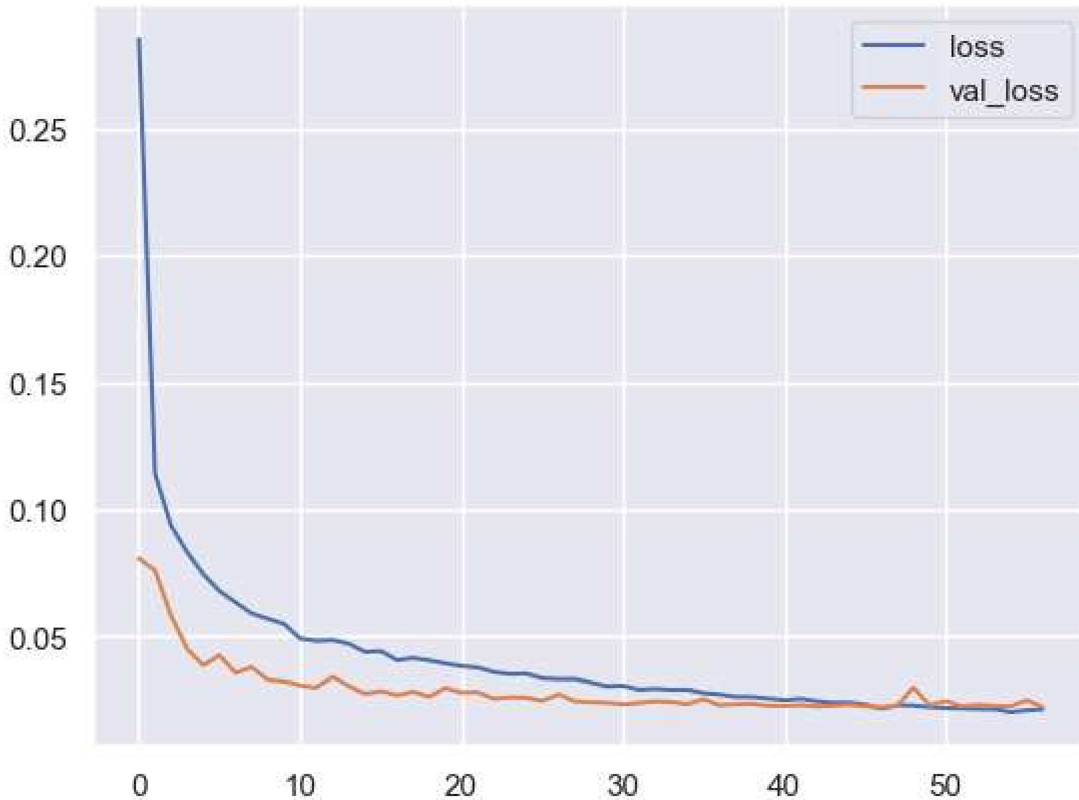
```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    batch_size=100,
    epochs=150,
    callbacks=[early_stop]
)
```

```
Epoch 1/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.2854 - val_loss: 0.0808
Epoch 2/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.1141 - val_loss: 0.0759
Epoch 3/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0935 - val_loss: 0.0580
Epoch 4/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0831 - val_loss: 0.0449
Epoch 5/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0744 - val_loss: 0.0386
Epoch 6/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0678 - val_loss: 0.0425
Epoch 7/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0634 - val_loss: 0.0356
Epoch 8/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0590 - val_loss: 0.0378
Epoch 9/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0569 - val_loss: 0.0328
Epoch 10/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0549 - val_loss: 0.0322
Epoch 11/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0491 - val_loss: 0.0307
Epoch 12/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0482 - val_loss: 0.0298
Epoch 13/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0485 - val_loss: 0.0341
Epoch 14/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0471 - val_loss: 0.0304
Epoch 15/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0438 - val_loss: 0.0273
Epoch 16/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0441 - val_loss: 0.0284
Epoch 17/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0406 - val_loss: 0.0268
Epoch 18/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0414 - val_loss: 0.0282
Epoch 19/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0405 - val_loss: 0.0262
Epoch 20/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0393 - val_loss: 0.0297
Epoch 21/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0383 - val_loss: 0.0278
Epoch 22/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0377 - val_loss: 0.0280
Epoch 23/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0359 - val_loss: 0.0254
Epoch 24/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0351 - val_loss: 0.0260
Epoch 25/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0353 - val_loss: 0.0259
Epoch 26/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0335 - val_loss: 0.0247
Epoch 27/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0331 - val_loss: 0.0272
Epoch 28/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0332 - val_loss: 0.0245
```

```
Epoch 29/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0318 - val_loss: 0.0240
Epoch 30/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0303 - val_loss: 0.0239
Epoch 31/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0305 - val_loss: 0.0233
Epoch 32/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0290 - val_loss: 0.0238
Epoch 33/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0293 - val_loss: 0.0245
Epoch 34/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0289 - val_loss: 0.0242
Epoch 35/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0290 - val_loss: 0.0234
Epoch 36/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0276 - val_loss: 0.0253
Epoch 37/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0271 - val_loss: 0.0229
Epoch 38/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0262 - val_loss: 0.0233
Epoch 39/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0262 - val_loss: 0.0235
Epoch 40/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0256 - val_loss: 0.0227
Epoch 41/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0249 - val_loss: 0.0227
Epoch 42/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0255 - val_loss: 0.0229
Epoch 43/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0245 - val_loss: 0.0225
Epoch 44/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0239 - val_loss: 0.0227
Epoch 45/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0239 - val_loss: 0.0231
Epoch 46/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0232 - val_loss: 0.0225
Epoch 47/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0216 - val_loss: 0.0221
Epoch 48/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0229 - val_loss: 0.0227
Epoch 49/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0228 - val_loss: 0.0298
Epoch 50/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0222 - val_loss: 0.0229
Epoch 51/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0218 - val_loss: 0.0245
Epoch 52/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0217 - val_loss: 0.0225
Epoch 53/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0214 - val_loss: 0.0231
Epoch 54/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0214 - val_loss: 0.0227
Epoch 55/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0202 - val_loss: 0.0225
Epoch 56/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0209 - val_loss: 0.0249
```

```
Epoch 57/150
103/103 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.0213 - val_loss: 0.0222
```

In [18]:
```
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot();
```



In [19]:
```
dnn_pred = model.predict(X_test)
```

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Results table

In [23]:
```
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[23]:

	MAE	MSE	R2-score
Linear Regression	0.182	0.058	0.638
Decision Tree	0.136	0.043	0.732
Random Forest	0.097	0.019	0.880
Support Vector Machines	0.118	0.028	0.824
K-nearest Neighbors	0.101	0.024	0.846
XGBoost	0.093	0.017	0.895
Deep Neural Network	0.111	0.024	0.847

In [24]:
```
f"10% of mean of target variable is {np.round(0.1 * data.AveragePrice.mean(),3)}"
```

Out[24]: '10% of mean of target variable is 0.141'

In [25]:
```
results.sort_values('R2-score',ascending=False).style.background_gradient(cmap='Gre
```

Out[25]:

	MAE	MSE	R2-score
XGBoost	0.093000	0.017000	0.895000
Random Forest	0.097000	0.019000	0.880000
Deep Neural Network	0.111000	0.024000	0.847000
K-nearest Neighbors	0.101000	0.024000	0.846000
Support Vector Machines	0.118000	0.028000	0.824000
Decision Tree	0.136000	0.043000	0.732000
Linear Regression	0.182000	0.058000	0.638000

In [ ]: