In [3]:

```python
# Vishakha Dhonde
# COBB055

# Importing Necesarry Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import keras
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential
import warnings
warnings.filterwarnings("ignore")
```

In [4]:

```python
# Loading Data
boston = load_boston()
data = pd.DataFrame(boston.data)
data.columns = boston.feature_names
data['PRICE'] = boston.target
data.head()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [5]:

```python
# Data Exploration
print(data.shape)
print(data.dtypes)
print(data.isnull().sum())
print(data.describe())
```

```
(506, 14)
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
```

```
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT       PRICE
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

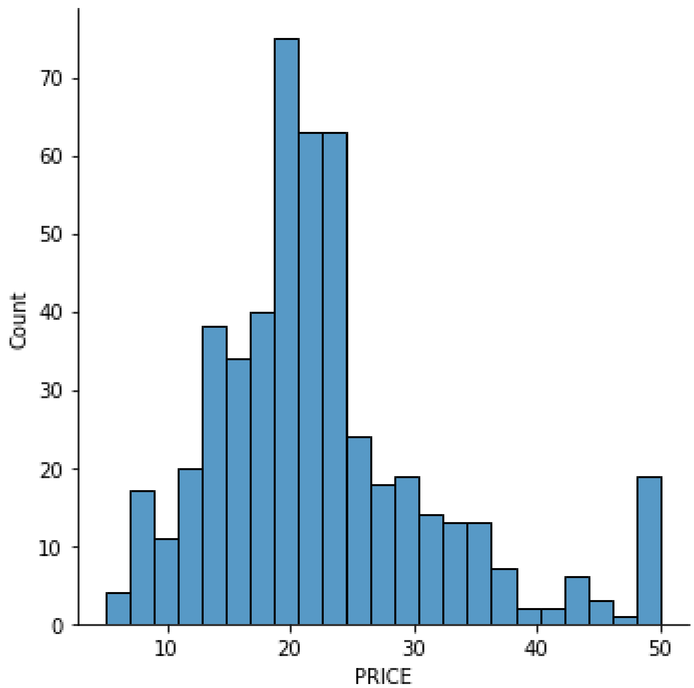
In [6]:

```
# Data Visualization
sns.displot(data.PRICE)

correlation = data.corr()
correlation.loc['PRICE']

fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

Out[6]: <AxesSubplot:>



In [7]:

```python
# Splitting Data into testing and training data
X = data.iloc[:,:-1]
y= data.PRICE
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state

# Normalizing the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [9]:

```python
# Model Building
model = Sequential()
model.add(Dense(128,activation  = 'relu',input_dim =13))
model.add(Dense(64,activation  = 'relu'))
model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation  = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'adam',loss = 'mean_squared_error')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1792
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 12,673
Trainable params: 12,673
Non-trainable params: 0
_________________________________________________________________
```

In [10]:

```python
# Fitting the data to the model
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
13/13 [==============================] - 0s 4ms/step - loss: 542.9215
Epoch 2/100
13/13 [==============================] - 0s 4ms/step - loss: 419.2375
Epoch 3/100
13/13 [==============================] - 0s 4ms/step - loss: 204.8196
Epoch 4/100
13/13 [==============================] - 0s 5ms/step - loss: 88.1948
Epoch 5/100
13/13 [==============================] - 0s 2ms/step - loss: 50.2675
Epoch 6/100
13/13 [==============================] - 0s 2ms/step - loss: 32.7460
Epoch 7/100
13/13 [==============================] - 0s 1ms/step - loss: 25.4227
```

```
Epoch 97/100
13/13 [==============================] - 0s 1ms/step - loss: 2.8114
Epoch 98/100
13/13 [==============================] - 0s 2ms/step - loss: 2.8003
Epoch 99/100
13/13 [==============================] - 0s 1ms/step - loss: 2.8641
Epoch 100/100
13/13 [==============================] - 0s 2ms/step - loss: 2.8603
```

Out[10]:

```
<tensorflow.python.keras.callbacks.History at 0x192c96d1640>
```

In [11]:

```
# Evaluating the model
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print("R2 Score = ", r2)
print("RMSE Score = ", rmse)
```

```
R2 Score =  0.9001743852285309
RMSE Score =  3.045114763857895
```

In [ ]: