DL Pac 1

In [1]:

```
import numpy as np
import pandas as pd
```

In [13]:

```
from sklearn.datasets import fetch_openml

# Load Boston housing dataset
boston  = fetch_openml(name="boston")

# Converting the data into pandas DataFrame
data=  pd.DataFrame(boston.data, columns=boston.feature_names)

# Adding the target variable to the dataset
data['PRICE'] = boston.target

# First Look at the data
print(data.head())

# Shape of the data
print(data.shape)
```

```
      CRIM    ZN  INDUS CHAS    NOX     RM   AGE     DIS RAD    TAX  PTRATIO  \
0  0.00632  18.0   2.31    0  0.538  6.575  65.2  4.0900   1  296.0     15.3
1  0.02731   0.0   7.07    0  0.469  6.421  78.9  4.9671   2  242.0     17.8
2  0.02729   0.0   7.07    0  0.469  7.185  61.1  4.9671   2  242.0     17.8
3  0.03237   0.0   2.18    0  0.458  6.998  45.8  6.0622   3  222.0     18.7
4  0.06905   0.0   2.18    0  0.458  7.147  54.2  6.0622   3  222.0     18.7

        B  LSTAT  PRICE
0  396.90   4.98   24.0
1  396.90   9.14   21.6
2  392.83   4.03   34.7
3  394.63   2.94   33.4
4  396.90   5.33   36.2
(506, 14)
```

```
C:\Users\SOFT LAB11\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:301: Use
rWarning: Multiple active versions of the dataset matching the name boston exist. Ve
rsions may be fundamentally different, returning version 1.
  warn(
C:\Users\SOFT LAB11\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:968: Fut
ureWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'`
in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportEr
ror` will be raised from 1.4 if the dataset is dense and pandas is not installed. No
te that the pandas parser may return different data types. See the Notes Section in
fetch_openml's API doc for details.
  warn(
```

In [14]:

```
#Converting the data into pandas dataframe
data= pd.DataFrame(boston.data)
#First Look at the data
data.head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900		296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90

In [15]:

```
#Adding the feature names to the dataframe
data.columns= boston.feature_names
#Adding the target variable to the dataset
data['PRICE'] = boston.target
#Looking at the data with names and target variable
data.head(n=10)
```

Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900		296.0	15.3	396.90
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0	15.2	395.60
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0	15.2	386.63
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0	15.2	386.71

In [16]:

```
#Shape of the data
print(data.shape)
#Checking the null values in the dataset
data.isnull().sum()
```

```
(506, 14)
```

```
Out[16]: 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
```

```
In [17]: #Checking the statistics of the data
         data.describe()
```

Out[17]:

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.554695	6.284634	68.574901	3.795043
std	8.601545	23.322453	6.860353	0.115878	0.702617	28.148861	2.105710
min	0.006320	0.000000	0.460000	0.385000	3.561000	2.900000	1.129600
25%	0.082045	0.000000	5.190000	0.449000	5.885500	45.025000	2.100175
50%	0.256510	0.000000	9.690000	0.538000	6.208500	77.500000	3.207450
75%	3.677083	12.500000	18.100000	0.624000	6.623500	94.075000	5.188425
max	88.976200	100.000000	27.740000	0.871000	8.780000	100.000000	12.126500

```
In [18]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Rangeindex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
              --------------
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    category
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    category
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: category(2), float64(12)
memory usage: 49.0 KB
```

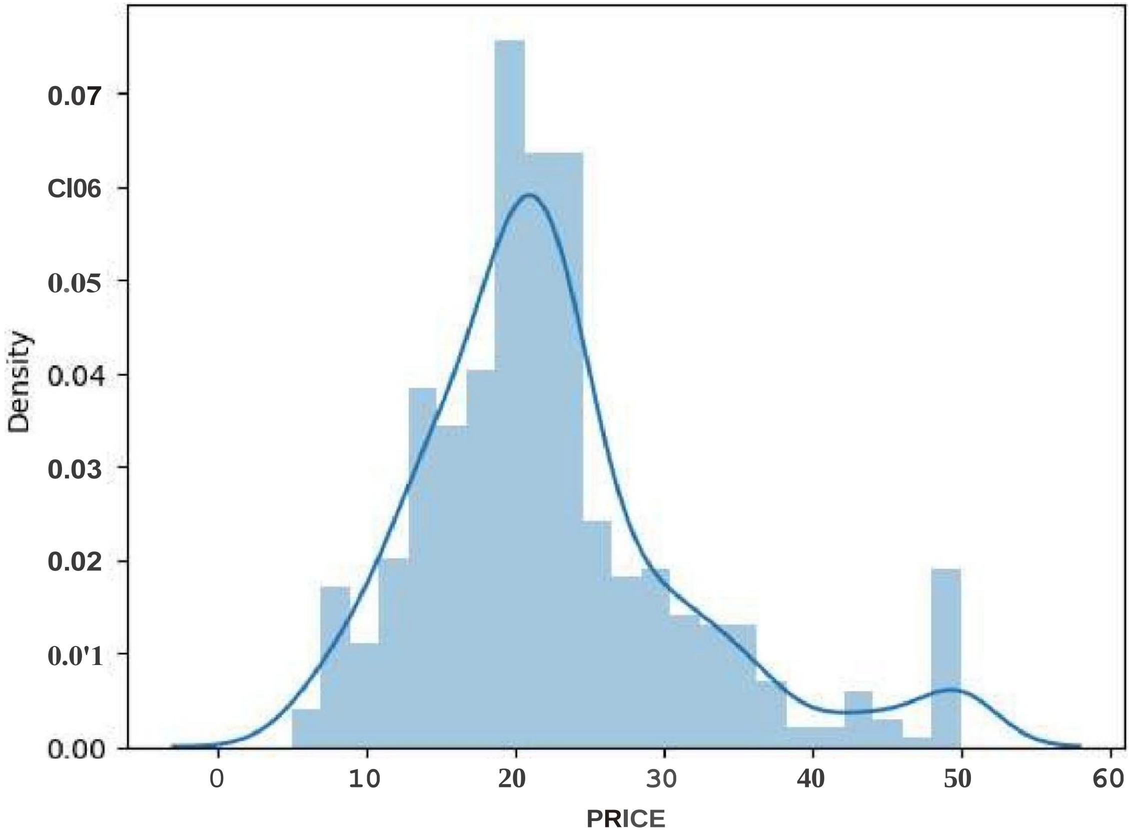
```
In [19]:  pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\soft labll\anaconda3\lib\site-pac
kages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in c:\users\soft labll\anaconda3
\lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=0.25 in c:\users\soft labll\anaconda3\lib\sit
e-packages (from seaborn) (2.1.4)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in c:\users\soft labll\anacon
da3\lib\site-packages (from seaborn) (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\soft labll\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\soft labll\anaconda3\lib\sit
e-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\soft labll\anaconda3\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\soft labll\anaconda3\li
b\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\soft labll\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\soft labll\anaconda3\lib\si
te-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\soft labll\anaconda3\lib
\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\soft labll\anaconda3
\lib\site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\soft labll\anaconda3\lib\sit
e-packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\soft labll\anaconda3\lib\s
ite-packages (from pandas>=0.25->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\soft labll\anaconda3\lib\site-pa
ckages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

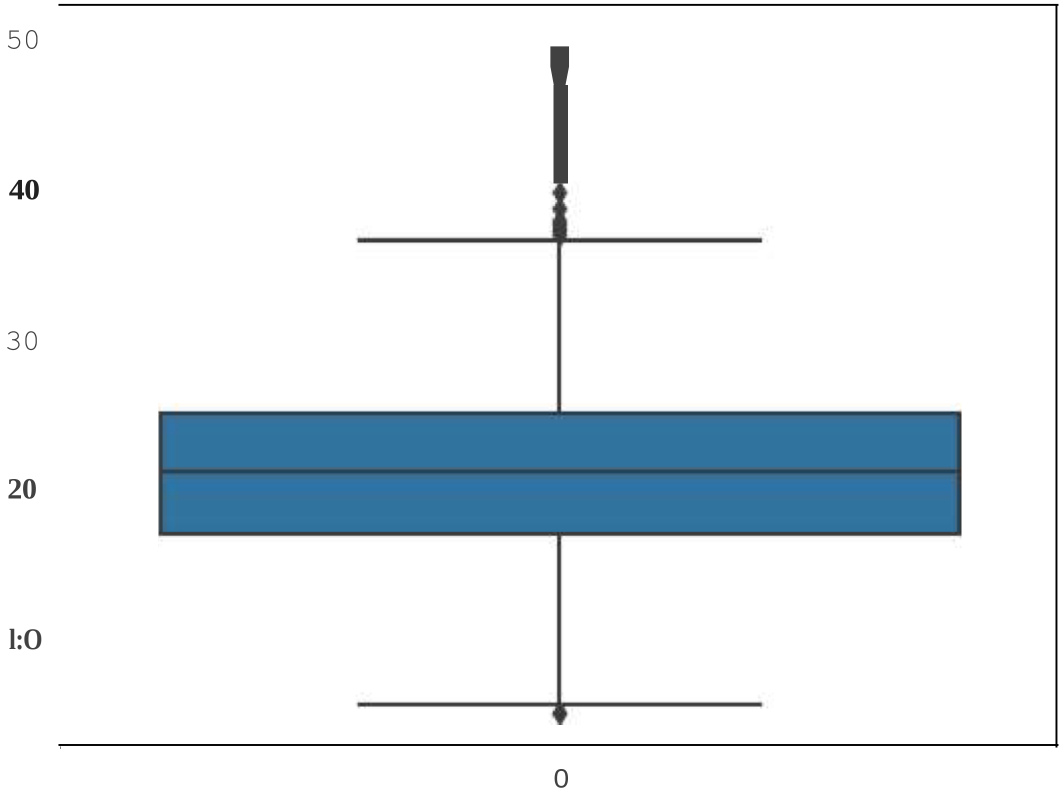
```
In [20]: #checking the distribution of the target variable
         import seaborn as sns
         sns.distplot(data.PRICE)
```

```
C:\Users\SOFT LAB11\AppData\Local\Temp\ipykernel_8924\4212025153.py:3: Userwarning:

'distplot' is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either 'displot' (a figure-level function with
similar flexibility) or 'histplot' (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.PRICE)
C:\Users\SOFT LAB11\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: Futurewarn
ing: use_inf_as_na option is deprecated and will be removed in a future version. Con
vert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

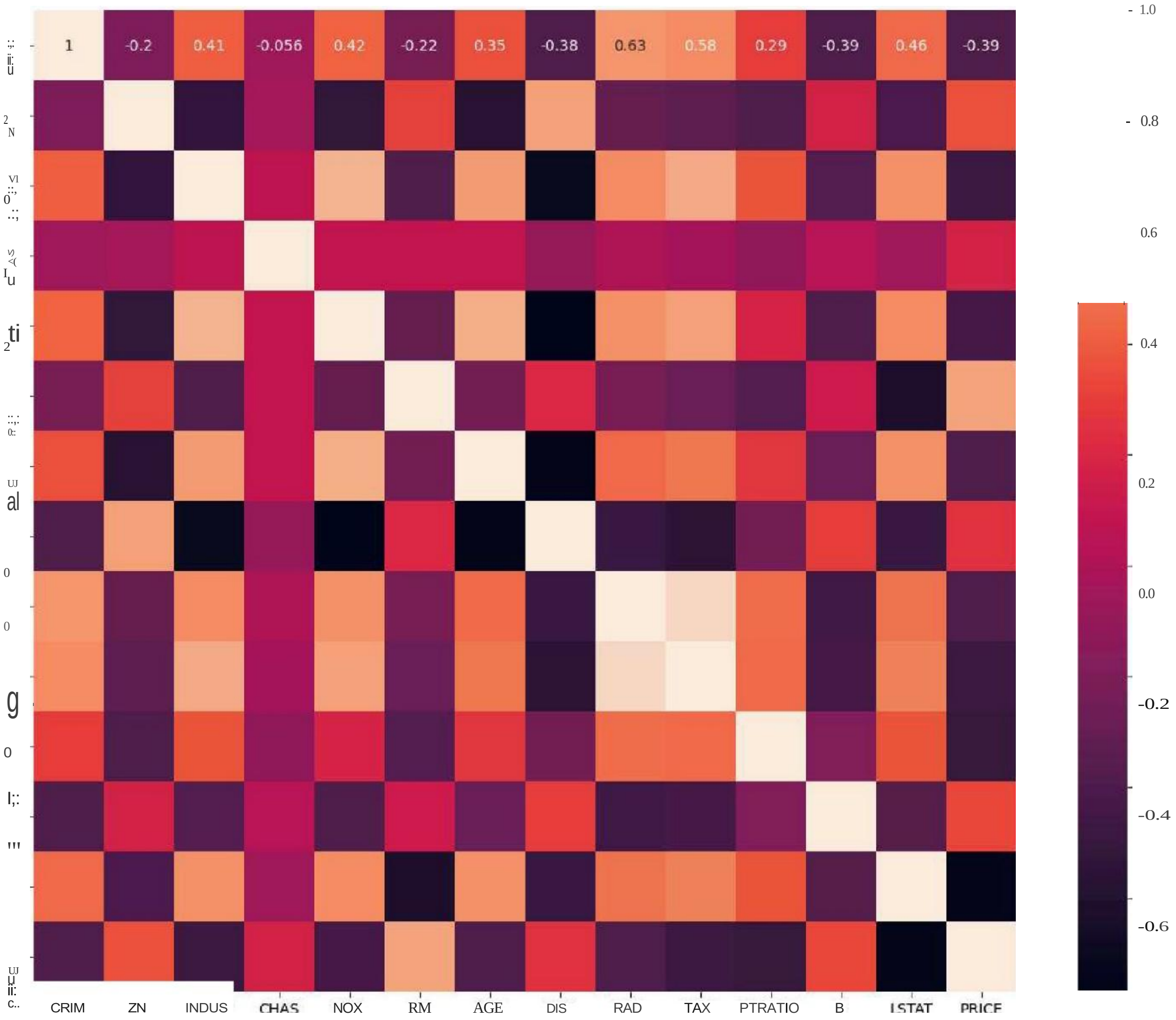
```
Out[20]: <Axes: xlabel='PRICE', ylabel='Density'>
```



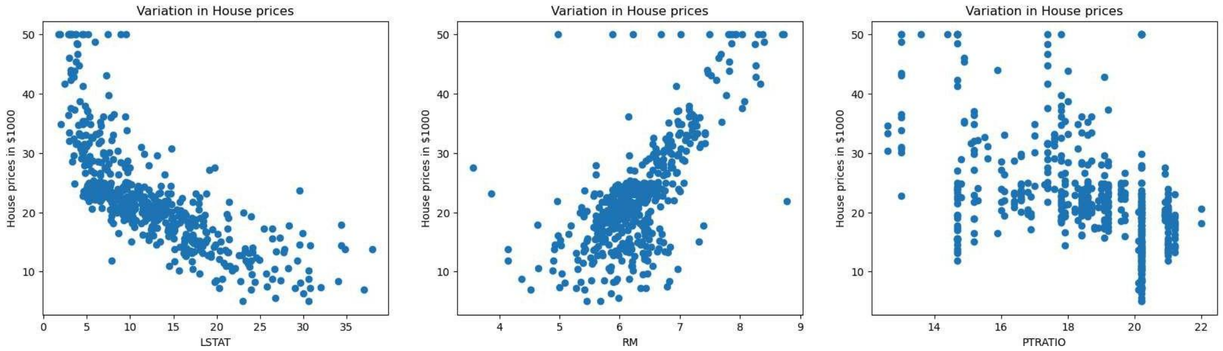
```
In [21]: sns.boxplot(data.PRICE)

Out[21]: <Axes: >
```



```
In [22]: correlation = data.corr()
         correlation.loc['PRICE']

Out[22]: CRIM      -0.388305
         ZN         0.360445
         INDUS     -0.483725
         CHAS       0.175260
         NOX       -0.427321
         RM         0.695360
         AGE       -0.376955
         DIS        0.249929
         RAD       -0.381626
         TAX       -0.468536
         PTRATIO   -0.507787
         B          0.333461
         LSTAT     -0.737663
         PRICE      1.000000

         Name: PRICE, dtype: float64

In [23]: # plotting the heatmap
         import matplotlib.pyplot as plt

In [24]: fig,axes = plt.subplots(figsize=(15,12))
         sns.heatmap(correlation,square = True,annot = True)

Out[24]: <Axes: >
```



```
In [25]: import matplotlib.pyplot as plt

         # Checking the scatter plot with the most correlated features
         plt.figure(figsize=(20, 5))
         features  = ['LSTAT', 'RM', 'PTRATIO']

         for i, col in enumerate(features):
             plt.subplot(l, len(features), i + 1)
             x = data[col]
             y = data.PRICE
             plt.scatter(x, y, marker='o')
             plt.title("Variation in House prices")
             plt.xlabel(col)
             plt.ylabel("House prices in $1000")

         pH.show()
```



In [26]:
```
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO'J]

from sklearn.model selection import train_test_split

X = data.iloc[:,:-1]
y= data.PRICE
```

In [27]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test  =  train_test_split(X, y, test_size=0.2, random_sta
```

In [28]:
```
# Assuming X_train and X_test are DataFrames

# Exclude non-numeric columns from standardization
numeric_columns = X_train.select_dtypes(include=[np.number]).columns

# Calculate mean and standard deviation only for numeric columns
mean= X_train[numeric_columns].mean(axis=0)
std= X_train[numeric_columns].std(axis=0)

# Standardize the training data for numeric columns
X_train_standardized  = (X_train[numeric_columns] -  mean)/ std

# Standardize the testing data using the mean and std from the training data for nu
X_test_standardized  = (X_test[numeric_columns] -  mean)/ std
```

In [46]:
```
# Assuming X and y are already defined and Loaded

# Exclude non-numeric columns from standardization
numeric_columns = X.select_dtypes(include=[np.number]).columns

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test  = train_test_split(X[numeric_columns], y, test siz

# Standardize the features
mean= X_train.mean(axis=0)
std= X_train.std(axis=0)
X_train = (X_train - mean)/ std
X_test = (X_test - mean)/ std

# Linear Regression
regressor = LinearRegression()

# Fitting the model
```

```
regressor.fit(X_train, y_train)

# Model Evaluation
# Prediction on the test dataset
y_pred = regressor.predict(X_test)

# Calculate RMSE (Root Mean Squared Error)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print( "RMSE:", rmse)

# Calculate R-squared (R2) score
r2 = r2_score(y_test, y_pred)
print("R-squared (R2) Score:", r2)
```

```
RMSE: 5.203503176683114
R-squared (R2) Score: 0.6307780105854284
```

```
In [29]: # Neural Networks
         #Scaling the dataset
         from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

```
In [30]: !pip install graphviz
         !pip install  ann_visualizer
```

```
Collecting graphviz
  Downloading graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
   ---------------------------------------- 0.0/47.1 kB? eta-:--:---
   ---------------- ----------------------- 20.5/47.1 kB 682.7 kB/s eta 0:00:01
   ---------------------------------------- 47.1/47.1 kB 594.9 kB/s eta 0:00:00
Installing collected packages: graphviz
Successfully installed graphviz-0.20.3
Collecting ann_visualizer

  Downloading ann_visualizer-2.5.tar.gz (4.7 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: ann_visualizer
  Building wheel for ann_visualizer (setup.py): started
  Building wheel for ann_visualizer (setup.py): finished with status 'done'
  Created wheel for ann_visualizer: filename=ann_visualizer-2.5-py3-none-any.whl siz
e=4181 sha256=150bbe984af6cfbf7807a5f643f6232f511556d1fcb33a3f41afd8c46d79ef32
  Stored in directory: c:\users\soft lab11\appdata\local\pip\cache\wheels\28\4a\ad\e
82da1aad2994e42bf0f4b1d403fdd8a64dfc38ae2c8a5daa4
Successfully built ann_visualizer
Installing collected packages: ann_visualizer
Successfully installed ann visualizer-2.5
```

```
In [32]:  pip install keras
```

```
In [34]:  pip install tensorflow
```

In [35]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [36]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [37]:
```
# Importing necessary libraries
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from ann_visualizer.visualize import ann_viz
import plotly.subplots as sp
import plotly.graph_objects as go

# Creating the neural network model
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))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

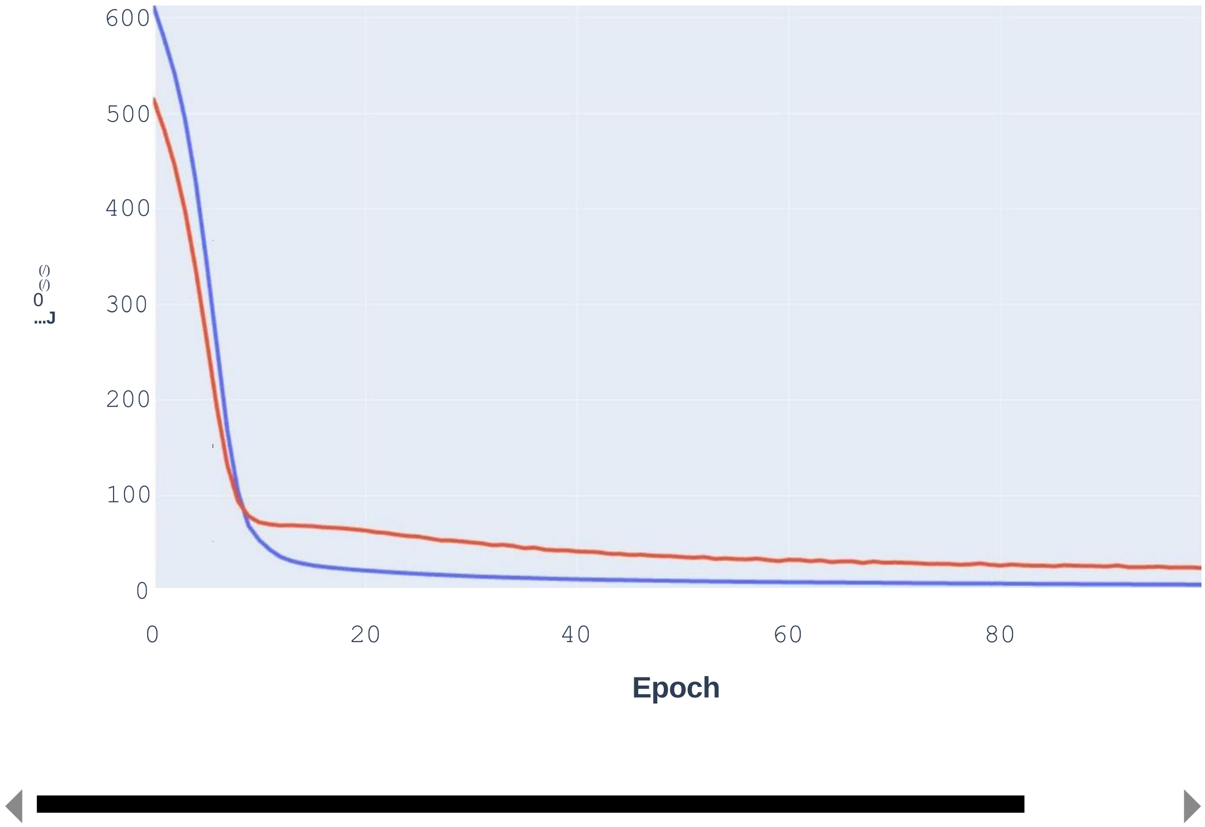
```
C:\Users\SOFT LAB11\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: U
serWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer. When using
Sequential models, prefer using an 'Input(shape)' object as the first layer in them
odel instead.
  super()._init_(activity_regularizer=activity_regularizer,   **kwargs)
```

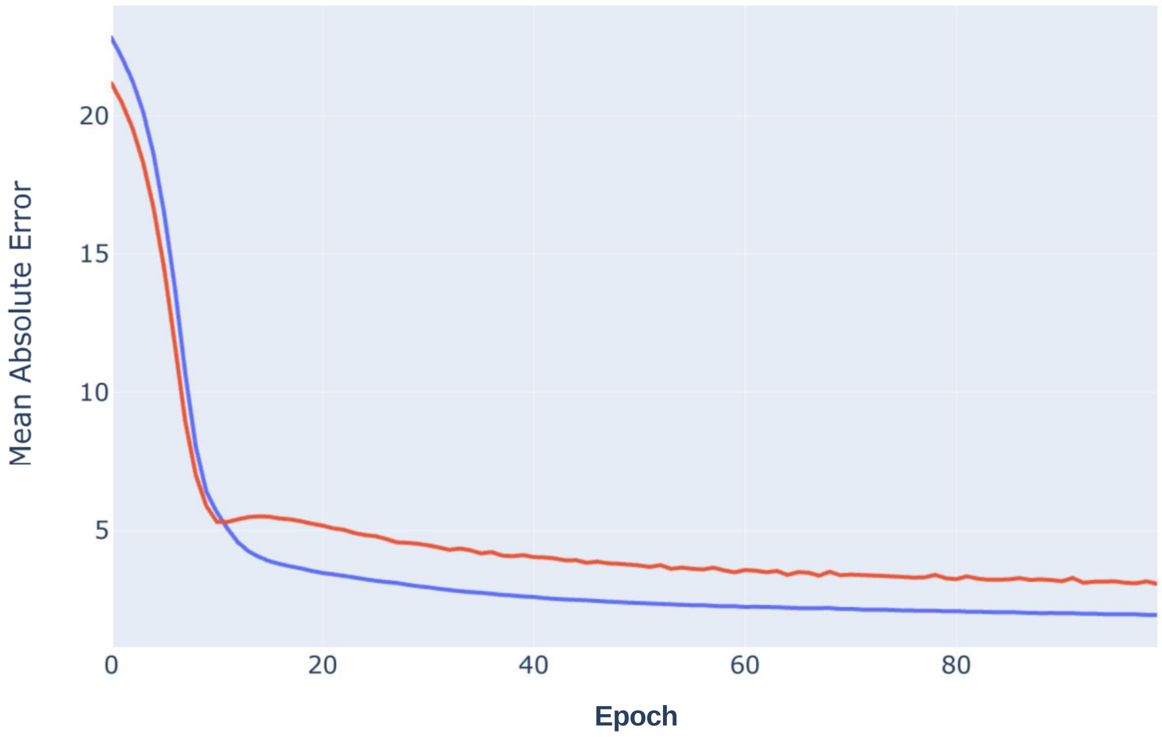
In [38]:
```
from ann_visualizer.visualize import ann viz
```

In [39]:
```
pip install keras matplotlib plotly
```

In [40]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
import plotly.graph_objects as go

# Load Boston housing dataset
boston  = fetch_openml(name="boston")

# Converting the data into pandas DataFrame
data=  pd.DataFrame(boston.data, columns=boston.feature_names)

# Adding the target variable to the dataset
data['PRICE'] = boston.target

#  Extract features (X) and target variable (y)
X = data.drop(columns=['PRICE'J).values
y = data['PRICE'J.values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler= StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the neural network model
model= Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[l],)),
    Dense(32, activation='relu'),
    Dense(l) # Output Layer with single neuron for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Training the model
history= model.fit(X_train_scaled, y_train, epochs=100, validation_split=0.05, cal

# Plotting the training and validation Loss
fig_loss = go.Figure()
fig_loss.add_trace(go.Scattergl(y=history.history['loss'J, name='Train'))
fig_loss.add_trace(go.Scattergl(y=history.history['val_loss'], name='Valid'))
fig_loss.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Los
fig_loss.show()

# Plotting the training and validation mean absolute error
fig_mae = go.Figure()
fig_mae.add_trace(go.Scattergl(y=history.history['mae'], name='Train'))
fig_mae.add_trace(go.Scattergl(y=history.history['val_mae'J, name='Valid'))
fig_mae.update_layout(height=500, width=700, xaxis_title='Epoch', yaxis_title='Mean
fig_mae.show()
```

```
Epoch 1/100
```


(/)
(/)
0
...J
600
500
400
300
200
100
0
0
20
40
60
80
Epoch



In [44]:
```
# Convert y_test to a numpy array if it's not already
y_test = np.array(y_test)

# Evaluate the model on the test data
mse_nn, mae nn = model.evaluate(X_test_scaled, y_test)

# Print the evaluation metrics
print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 9.2586 - mae: 2.2224
Mean squared error on test data:  12.195608139038086
Mean absolute error on test data:  2.3551769256591797
```

In [45]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
import plotly.graph_objects as go
```

```
# Load Boston housing dataset
boston = fetch_openml(name="boston")

# Converting the data into pandas DataFrame
data=  pd.DataFrame(boston.data,  columns=boston.feature_names)

# Adding the target variable to the dataset
data['PRICE'] =  boston.target

# Extract features (X) and target variable (y)
X = data.drop(columns=['PRICE'J).values
y = data['PRICE'J.values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test  =  train_test_split(X, y, test_size=0.2, random_sta

# Standardize the features
scaler= StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the neural network model
model= Sequential([
    Dense(64, activation='relu', input_shape=(X_train_scaled.shape[l],)),
    Dense(32, activation='relu'),
    Dense(l)  # Output Layer with single neuron for regression
])


# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'J)

# Training the model
history= model.fit(X_train_scaled, y_train, epochs=100, validation_split=0.05, cal

# Evaluation of the neural network model
y_pred_nn = model.predict(X_test_scaled)
mse_nn = mean_squared_error(y_test, y_pred_nn)
mae_nn = mean_absolute_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

print('Neural Network Model:')
print('Mean squared error on test data:', mse_nn)
print('Mean absolute error on test data:', mae_nn)
print('R-squared (R2) score:', r2_nn)

# Comparison with Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
y_pred_lr = lr_model.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print('\nlinear Regression Model:')
print('Mean squared error on test data:', mse_lr)
```

```
        print('Mean absolute error on test data:', mae_lr)
        print('R-squared (R2) score:', r2_lr)
```

```
In [46]: # Make predictions on new data
         import sklearn
         new_data = sklearn. preprocessing.StandardScaler(),fit_transform(([[0.1, 10.0,
         5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]))
         prediction= model.predict(new_data)
         print("Predicted house price:", prediction)
```

```
1/1                       0s 34ms/step
   -----------------------
Predicted house price: [[9.126371]]
```

```
In [ ]:
```