In [5]:
```
import numpy as np

from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
```

In [6]:
```
from sklearn.datasets import fetch_california_housing
```

In [6]:

```
ca_house_db=fetch_california_housing()
print (ca_house_db.data.shape)
print (ca_house_db.target.shape)
print (ca_house_db.feature_names)
print (ca_house_db.DESCR)
print (ca_house_db)
```

```
(20640, 8)
(20640,)
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
'Latitude', 'Longitude']
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the targe
t

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html (https://ww
w.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html)

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per cen
sus
block group. A block group is the smallest geographical unit for which th
e U.S.
Census Bureau publishes sample data (a block group typically has a popula
tion
of 600 to 3,000 people).

An household is a group of people residing within a home. Since the avera
ge
number of rooms and bedrooms in this dataset are provided per household,
these
columns may take surpinsingly large values for block groups with few hous
eholds
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
```

```
Statistics and Probability Letters, 33 (1997) 291-297
```


```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,    2.5
5555556,
          37.88      , -122.23      ],
       [   8.3014    ,   21.        ,    6.23813708, ...,    2.10984183,
          37.86      , -122.22      ],
       [   7.2574    ,   52.        ,    8.28813559, ...,    2.80225989,
          37.85      , -122.24      ],
       ...,
       [   1.7       ,   17.        ,    5.20554273, ...,    2.3256351 ,
          39.43      , -121.22      ],
       [   1.8672    ,   18.        ,    5.32951289, ...,    2.12320917,
          39.43      , -121.32      ],
       [   2.3886    ,   16.        ,    5.25471698, ...,    2.61698113,
          39.37      , -121.24      ]]), 'target': array([4.526, 3.585,
3.521, ..., 0.923, 0.847, 0.894]), 'frame': None, 'target_names': ['MedHo
useVal'], 'feature_names': ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrm
s', 'Population', 'AveOccup', 'Latitude', 'Longitude'], 'DESCR': '.. _cal
ifornia_housing_dataset:\n\nCalifornia Housing dataset\n-----------------
---------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 2
0640\n\n    :Number of Attributes: 8 numeric, predictive attributes and t
he target\n\n    :Attribute Information:\n        - MedInc        median
income in block group\n        - HouseAge      median house age in block
group\n        - AveRooms      average number of rooms per household\n
- AveBedrms     average number of bedrooms per household\n        - Popul
ation    block group population\n        - AveOccup      average number o
f household members\n        - Latitude      block group latitude\n
- Longitude     block group longitude\n\n    :Missing Attribute Values: N
one\n\nThis dataset was obtained from the StatLib repository.\nhttps://ww
w.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe target variable
is the median house value for California districts,\nexpressed in hundred
s of thousands of dollars ($100,000).\n\nThis dataset was derived from th
e 1990 U.S. census, using one row per census\nblock group. A block group
is the smallest geographical unit for which the U.S.\nCensus Bureau publi
shes sample data (a block group typically has a population\nof 600 to 3,0
00 people).\n\nAn household is a group of people residing within a home.
Since the average\nnumber of rooms and bedrooms in this dataset are provi
ded per household, these\ncolumns may take surpisingly large values for
block groups with few households\nand many empty houses, such as vacation
resorts.\n\nIt can be downloaded/loaded using the\n:func:`sklearn.dataset
s.fetch_california_housing` function.\n\n.. topic:: References\n\n    - P
ace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,\n      S
tatistics and Probability Letters, 33 (1997) 291-297\n'}
```


In [10]:
```
housing = fetch_california_housing()
X = housing.data
y = housing.target
```

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:

```python
# Split the data into training (50%) and validation (50%) sets
X_train_val, X_test, y_train_val, y_test = train_test_split(housing.da

# Split the training set into training (80%) and validation (20%) subs
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_trair

# Print the shapes of the resulting subsets
print("Training set: X_train = {}, y_train = {}".format(X_train.shape,
print("Validation set: X_val = {}, y_val = {}".format(X_val.shape, y_v
print("Testing set: X_test = {}, y_test = {}".format(X_test.shape, y_t
```

```
Training set: X_train = (8256, 8), y_train = (8256,)
Validation set: X_val = (2064, 8), y_val = (2064,)
Testing set: X_test = (10320, 8), y_test = (10320,)
```

In [9]:

```python
import tensorflow as tf
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Load data
housing = fetch_california_housing()
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_

# Define model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation="relu", input_shape=X_train.s
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(1, activation="linear")
])

# Compile model
model.compile(loss="mse", optimizer="adam")

# Train model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_v

# Predict on validation set
y_valid_pred = model.predict(X_valid)

# Calculate R2 score on validation set
r2 = r2_score(y_valid, y_valid_pred)
print(f"Set 1 R2 score: {r2}")

```

```
Epoch 1/100
363/363 [==============================] - 2s 3ms/step - loss: 46.6198
- val_loss: 5.5429
Epoch 2/100
363/363 [==============================] - 1s 2ms/step - loss: 1.3149
- val_loss: 4.4744
Epoch 3/100
363/363 [==============================] - 1s 2ms/step - loss: 4.5052
- val_loss: 5.6807
Epoch 4/100
363/363 [==============================] - 1s 3ms/step - loss: 73.1594
- val_loss: 2.7875
Epoch 5/100
363/363 [==============================] - 1s 2ms/step - loss: 1.0399
- val_loss: 3.0086
Epoch 6/100
363/363 [==============================] - 1s 3ms/step - loss: 1.1106
- val_loss: 3.3342
Epoch 7/100
```

In [1]:

```python
import tensorflow as tf
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Load data
housing = fetch_california_housing()
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_

# Define model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation="sigmoid", input_shape=X_tra
    tf.keras.layers.Dense(128, activation="sigmoid"),
    tf.keras.layers.Dense(1, activation="linear")
])

# Compile model
model.compile(loss="mse", optimizer="sgd")

# Train model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_v

# Predict on validation set
y_valid_pred = model.predict(X_valid)

# Calculate R2 score on validation set
r2 = r2_score(y_valid, y_valid_pred)
print(f"Set 2 R2 score: {r2}")

```

```
Epoch 62/100
363/363 [==============================] - 1s 3ms/step - loss: 1.3447
- val_loss: 1.3416
Epoch 63/100
363/363 [==============================] - 1s 3ms/step - loss: 1.3427
- val_loss: 1.3253
Epoch 64/100
363/363 [==============================] - 1s 3ms/step - loss: 1.3435
- val_loss: 1.3147
Epoch 65/100
363/363 [==============================] - 1s 2ms/step - loss: 1.3438
- val_loss: 1.3204
Epoch 66/100
363/363 [==============================] - 1s 2ms/step - loss: 1.3434
- val_loss: 1.3290
Epoch 67/100
363/363 [==============================] - 1s 2ms/step - loss: 1.3430
- val_loss: 1.3145
Epoch 68/100
363/363 [==============================] - 1s 3ms/step - loss: 1.3434
```

In [10]:

```
from sklearn.metrics import r2_score

# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(8,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

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

# Train the model on the entire training set
model.fit(X_train, y_train, epochs=50, batch_size=128, verbose=0)

# Evaluate the model on the testing set
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R2 score on testing set:", r2)

```

```
162/162 [==============================] - 0s 1ms/step
R2 score on testing set: 0.4053987785465406
```

In [11]:

```
from sklearn.metrics import r2_score

# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(8,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

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

# Train the model on the entire training set
model.fit(X_train, y_train, epochs=50, batch_size=128, verbose=0)

# Evaluate the model on the testing set
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R2 score on testing set:", r2)

```

```
162/162 [==============================] - 0s 1ms/step
R2 score on testing set: 0.5257819877370892
```

In [12]:

```
# Apply top-ranked model over testing samples
y_pred_test = model.predict(X_test)

# Calculate absolute errors
abs_errors = np.abs(y_test - y_pred_test).flatten()

# Sort errors in descending order and select top 10
largest_errors_idx = np.argsort(abs_errors)[::-1][:10]
largest_errors = abs_errors[largest_errors_idx]
```

```
162/162 [==============================] - 0s 1ms/step
```

The R2 score on the testing set for the selected model is 0.525, which indicates a good fit of the model on the testing data.