Regression on Boston Housing price dataset

> > In a learnable approach brought you by Jayati Vijaywargiya

Import library

In [6]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
```

Import dataset

Boston housing dataset

- It contains information of various houses
- It has 506 data rows and 14 features
- It has features like, crime rate in town, proportion of residential land, average number of rooms per dwelling, etc

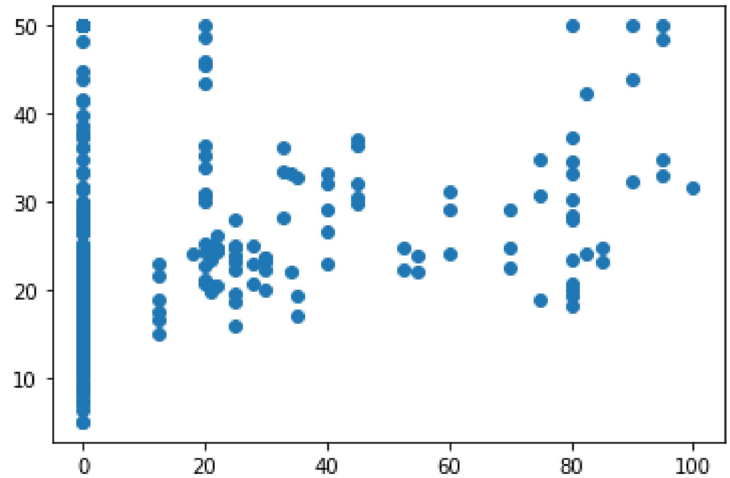
In [2]:
```
(train_x, train_y), (test_x, test_y) = datasets.boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
sets/boston_housing.npz
57344/57026 [==============================] - 0s 0us/step
```

Just to have vizualization plotting the y or the price in terms of few variables

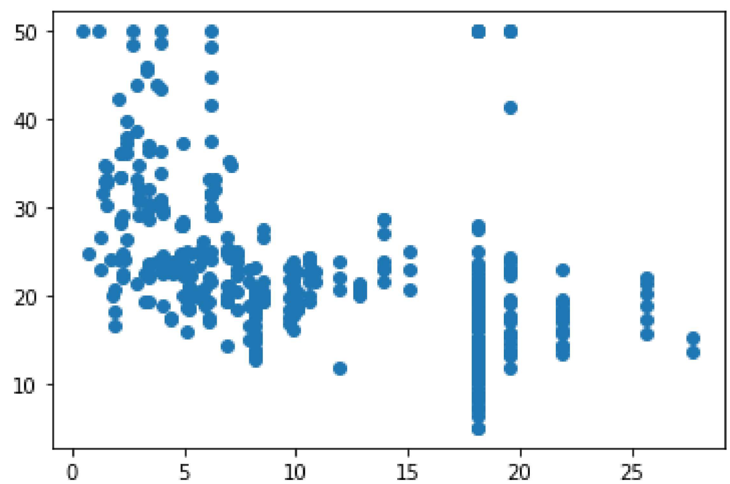
In [9]:
```
plt.scatter(train_x[:,1], train_y)
```

Out[9]: <matplotlib.collections.PathCollection at 0x7f03585845c0>



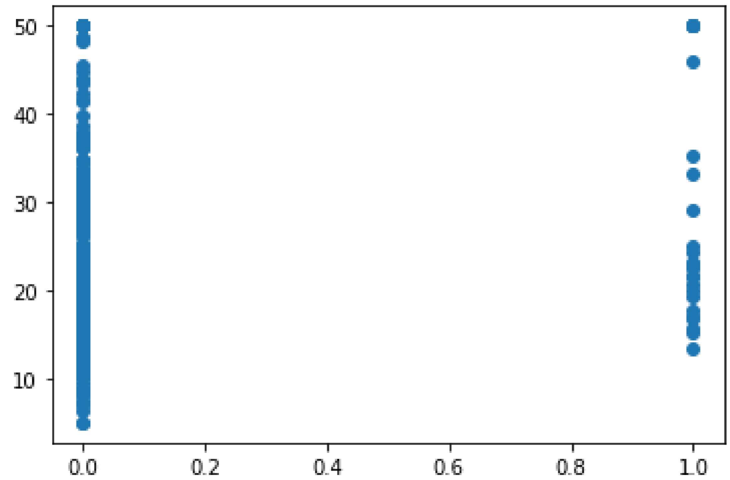
In [10]:
```
plt.scatter(train_x[:,2], train_y)
```

Out[10]: <matplotlib.collections.PathCollection at 0x7f03584ea0f0>



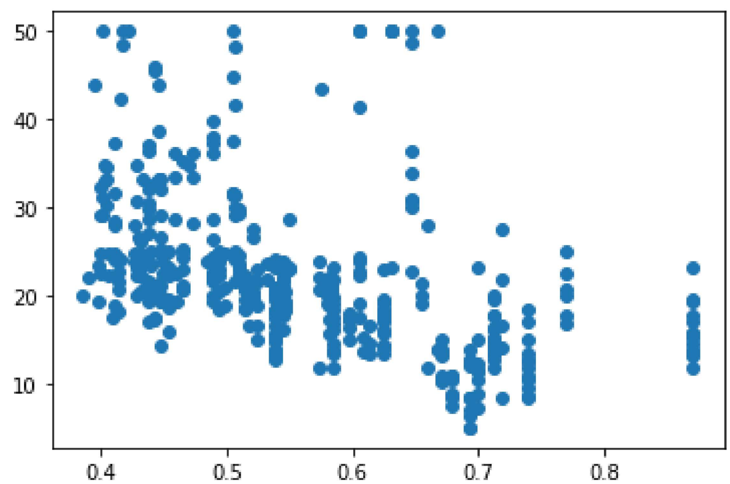
In [11]:
```
plt.scatter(train_x[:,3], train_y)
```

Out[11]: <matplotlib.collections.PathCollection at 0x7f03584d6438>



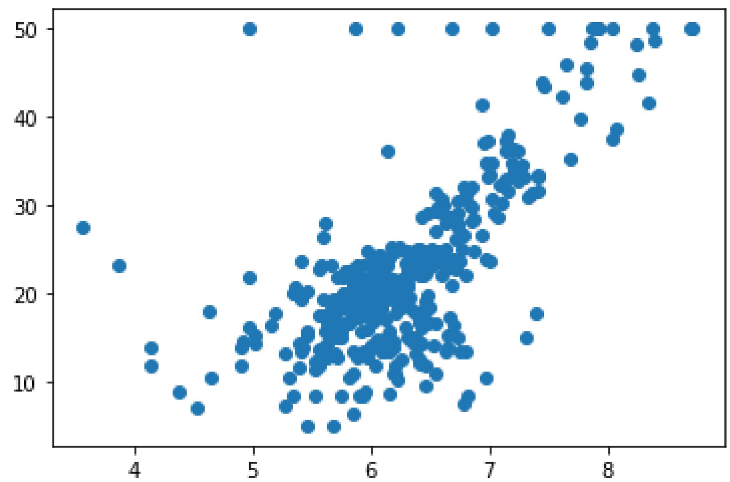
In [12]:
```
plt.scatter(train_x[:,4], train_y)
```

Out[12]: <matplotlib.collections.PathCollection at 0x7f0358428dd8>



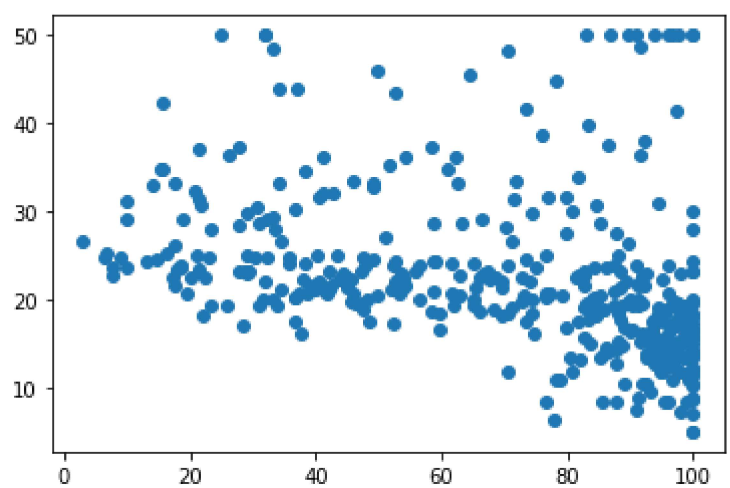
In [13]:
```
plt.scatter(train_x[:,5], train_y)
```

Out[13]: <matplotlib.collections.PathCollection at 0x7f03584091d0>



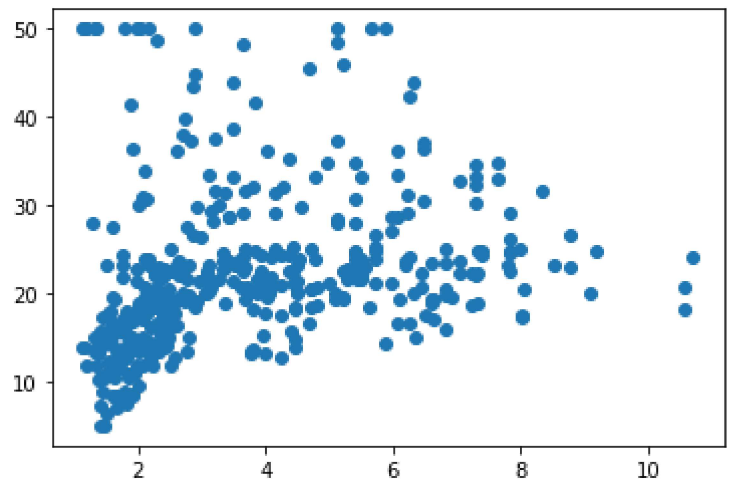
In [14]:
```
plt.scatter(train_x[:,6], train_y)
```

Out[14]: <matplotlib.collections.PathCollection at 0x7f03583dc470>



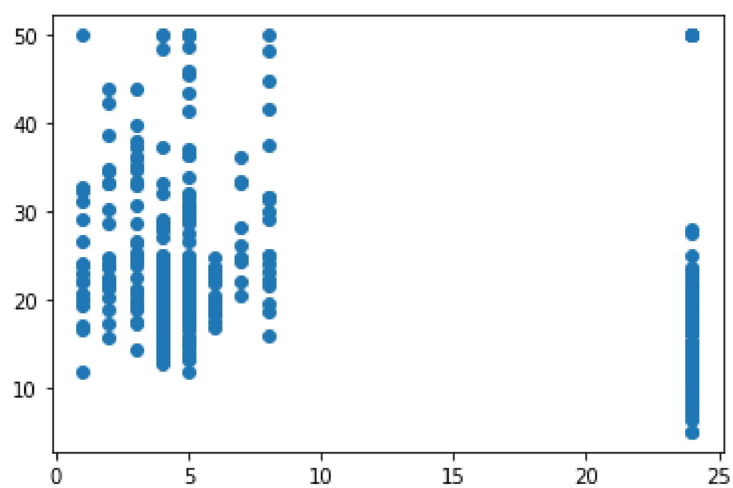
In [15]:
```
plt.scatter(train_x[:,7], train_y)
```

Out[15]: <matplotlib.collections.PathCollection at 0x7f0358330eb8>



In [16]:
```
plt.scatter(train_x[:,8], train_y)
```

Out[16]: <matplotlib.collections.PathCollection at 0x7f0358311240>



Creating a simple Neural Network

Here, the network is made from fully connected layer or Dense layer. It uses softmax activation function. It is like a regular densely-connected NN layer, In the abobe model the output of this layer will have 10 unique outputs or 10 classes

More detail about inbuilt function used

> > tf.keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)
> >
> > 1. units: Positive integer, dimensionality of the output space.
> > 2. activation: Activation function to use. If you don't specify then activation is applied (ie. "linear" activation: a(x) = x)
> > 3. use_bias: Boolean, whether the layer uses a bias vector.
> > 4. kernel_initializer: Initializer for the kernel weights matrix.
> > 5. bias_initializer: Initializer for the bias vector.

In [17]:
```
model = models.Sequential()
model.add(layers.Dense(51, activation='relu', input_shape=(13,)))
model.add(layers.Dense(37, activation='relu'))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(1))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 51)                714       
_________________________________________________________________
dense_1 (Dense)              (None, 37)                1924      
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 304       
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9         
=================================================================
Total params: 2,951
Trainable params: 2,951
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
model.compile(optimizer='Adam',loss='mse',metrics='mse')
history=model.fit(train_x, train_y,epochs=30,validation_data=(test_x, test_y))
```

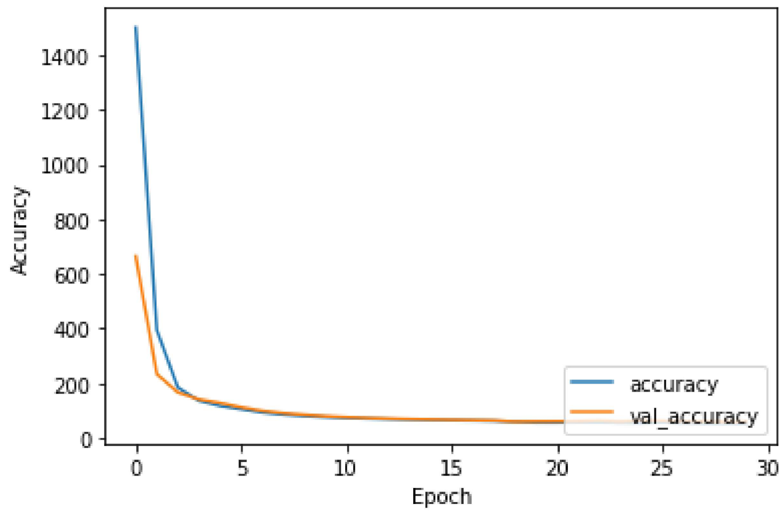
```
Epoch 1/30
13/13 [==============================] - 0s 9ms/step - loss: 1501.3920 - mse: 1501.3920 - val_loss: 664.4969 - val_mse: 664.4969
Epoch 2/30
13/13 [==============================] - 0s 2ms/step - loss: 394.7716 - mse: 394.7716 - val_loss: 234.0633 - val_mse: 234.0633
Epoch 3/30
13/13 [==============================] - 0s 2ms/step - loss: 184.4897 - mse: 184.4897 - val_loss: 166.6199 - val_mse: 166.6199
Epoch 4/30
13/13 [==============================] - 0s 2ms/step - loss: 136.2608 - mse: 136.2608 - val_loss: 141.1035 - val_mse: 141.1035
Epoch 5/30
13/13 [==============================] - 0s 2ms/step - loss: 117.5735 - mse: 117.5735 - val_loss: 128.2210 - val_mse: 128.2210
Epoch 6/30
13/13 [==============================] - 0s 2ms/step - loss: 105.3084 - mse: 105.3084 - val_loss: 112.2710 - val_mse: 112.2710
Epoch 7/30
13/13 [==============================] - 0s 2ms/step - loss: 93.2060 - mse: 93.2060 - val_loss: 99.2482 - val_mse: 99.2482
Epoch 8/30
13/13 [==============================] - 0s 2ms/step - loss: 85.7123 - mse: 85.7123 - val_loss: 90.6974 - val_mse: 90.6974
Epoch 9/30
13/13 [==============================] - 0s 2ms/step - loss: 80.2598 - mse: 80.2598 - val_loss: 84.7545 - val_mse: 84.7545
Epoch 10/30
13/13 [==============================] - 0s 2ms/step - loss: 76.3272 - mse: 76.3272 - val_loss: 79.6409 - val_mse: 79.6409
Epoch 11/30
13/13 [==============================] - 0s 2ms/step - loss: 73.0498 - mse: 73.0498 - val_loss: 76.0093 - val_mse: 76.0093
Epoch 12/30
13/13 [==============================] - 0s 2ms/step - loss: 71.0403 - mse: 71.0403 - val_loss: 72.9755 - val_mse: 72.9755
Epoch 13/30
13/13 [==============================] - 0s 2ms/step - loss: 69.0560 - mse: 69.0560 - val_loss: 70.8099 - val_mse: 70.8099
Epoch 14/30
13/13 [==============================] - 0s 2ms/step - loss: 67.3428 - mse: 67.3428 - val_loss: 69.0240 - val_mse: 69.0240
Epoch 15/30
13/13 [==============================] - 0s 2ms/step - loss: 66.0269 - mse: 66.0269 - val_loss: 67.5310 - val_mse: 67.5310
Epoch 16/30
13/13 [==============================] - 0s 3ms/step - loss: 64.7559 - mse: 64.7559 - val_loss: 66.1267 - val_mse: 66.1267
Epoch 17/30
13/13 [==============================] - 0s 2ms/step - loss: 63.9691 - mse: 63.9691 - val_loss: 64.9962 - val_mse: 64.9962
Epoch 18/30
13/13 [==============================] - 0s 2ms/step - loss: 62.7252 - mse: 62.7252 - val_loss: 63.9431 - val_mse: 63.9431
Epoch 19/30
13/13 [==============================] - 0s 2ms/step - loss: 59.2838 - mse: 59.2838 - val_loss: 59.4739 - val_mse: 59.4739
```

```
Epoch 20/30
13/13 [==============================] - 0s 2ms/step - loss: 56.8251 - mse: 5
6.8251 - val_loss: 60.2756 - val_mse: 60.2756
Epoch 21/30
13/13 [==============================] - 0s 2ms/step - loss: 56.7185 - mse: 5
6.7185 - val_loss: 60.7975 - val_mse: 60.7975
Epoch 22/30
13/13 [==============================] - 0s 2ms/step - loss: 57.2177 - mse: 5
7.2177 - val_loss: 60.9065 - val_mse: 60.9065
Epoch 23/30
13/13 [==============================] - 0s 2ms/step - loss: 57.8332 - mse: 5
7.8332 - val_loss: 63.0433 - val_mse: 63.0433
Epoch 24/30
13/13 [==============================] - 0s 2ms/step - loss: 55.4722 - mse: 5
5.4722 - val_loss: 59.7934 - val_mse: 59.7934
Epoch 25/30
13/13 [==============================] - 0s 2ms/step - loss: 54.8991 - mse: 5
4.8991 - val_loss: 63.4175 - val_mse: 63.4175
Epoch 26/30
13/13 [==============================] - 0s 2ms/step - loss: 57.6218 - mse: 5
7.6218 - val_loss: 62.2396 - val_mse: 62.2396
Epoch 27/30
13/13 [==============================] - 0s 2ms/step - loss: 54.8145 - mse: 5
4.8145 - val_loss: 60.8175 - val_mse: 60.8175
Epoch 28/30
13/13 [==============================] - 0s 2ms/step - loss: 52.9539 - mse: 5
2.9539 - val_loss: 59.7499 - val_mse: 59.7499
Epoch 29/30
13/13 [==============================] - 0s 2ms/step - loss: 53.0761 - mse: 5
3.0761 - val_loss: 58.9264 - val_mse: 58.9264
Epoch 30/30
13/13 [==============================] - 0s 2ms/step - loss: 52.3415 - mse: 5
2.3415 - val_loss: 58.4715 - val_mse: 58.4715
```

In [23]:
```
plt.plot(history.history['mse'], label='accuracy')
plt.plot(history.history['val_mse'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_x, test_y, verbose=2)
print(test_acc)
```

```
4/4 - 0s - loss: 58.4715 - mse: 58.4715
58.4715461730957
```



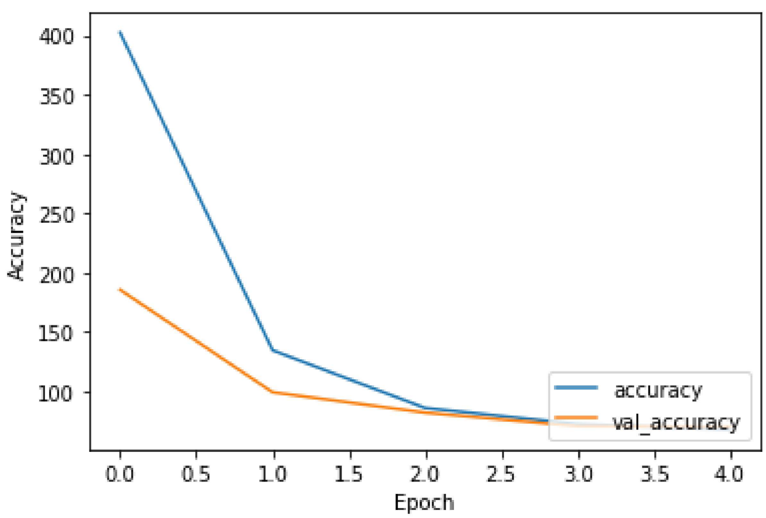
Here, taking more epochs doesnt seem to be a good idea,

lets model again

In [24]:
```
model2 = models.Sequential()
model2.add(layers.Dense(51, activation='relu', input_shape=(13,)))
model2.add(layers.Dense(37, activation='relu'))
model2.add(layers.Dense(8, activation='relu'))
model2.add(layers.Dense(1))
model2.summary()
model2.compile(optimizer='Adam',loss='mse',metrics='mse')
history2=model2.fit(train_x, train_y,epochs=5,validation_data=(test_x, test_y
))
plt.plot(history2.history['mse'], label='accuracy')
plt.plot(history2.history['val_mse'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

test_loss, test_acc = model2.evaluate(test_x, test_y, verbose=2)
print(test_acc)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 51)                714       
_________________________________________________________________
dense_5 (Dense)              (None, 37)                1924      
_________________________________________________________________
dense_6 (Dense)              (None, 8)                 304       
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 9         
=================================================================
Total params: 2,951
Trainable params: 2,951
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
13/13 [==============================] - 0s 7ms/step - loss: 402.2692 - mse: 
402.2692 - val_loss: 185.7450 - val_mse: 185.7450
Epoch 2/5
13/13 [==============================] - 0s 2ms/step - loss: 134.7648 - mse: 
134.7648 - val_loss: 99.4005 - val_mse: 99.4005
Epoch 3/5
13/13 [==============================] - 0s 2ms/step - loss: 86.1584 - mse: 8
6.1584 - val_loss: 82.2426 - val_mse: 82.2426
Epoch 4/5
13/13 [==============================] - 0s 2ms/step - loss: 72.6120 - mse: 7
2.6120 - val_loss: 70.8248 - val_mse: 70.8248
Epoch 5/5
13/13 [==============================] - 0s 2ms/step - loss: 68.0571 - mse: 6
8.0571 - val_loss: 70.2682 - val_mse: 70.2682
4/4 - 0s - loss: 70.2682 - mse: 70.2682
70.26815032958984
```



This was all about Regression using Neural Network on Boston Housing Dataset

Hope you like it ....To know more subscribe to our channel https://www.youtube.com/channel/UCfcRI0zl9RMOR11fHt (https://www.youtube.com/channel/UCfcRI0zl9RMOR11fH

In [ ]: