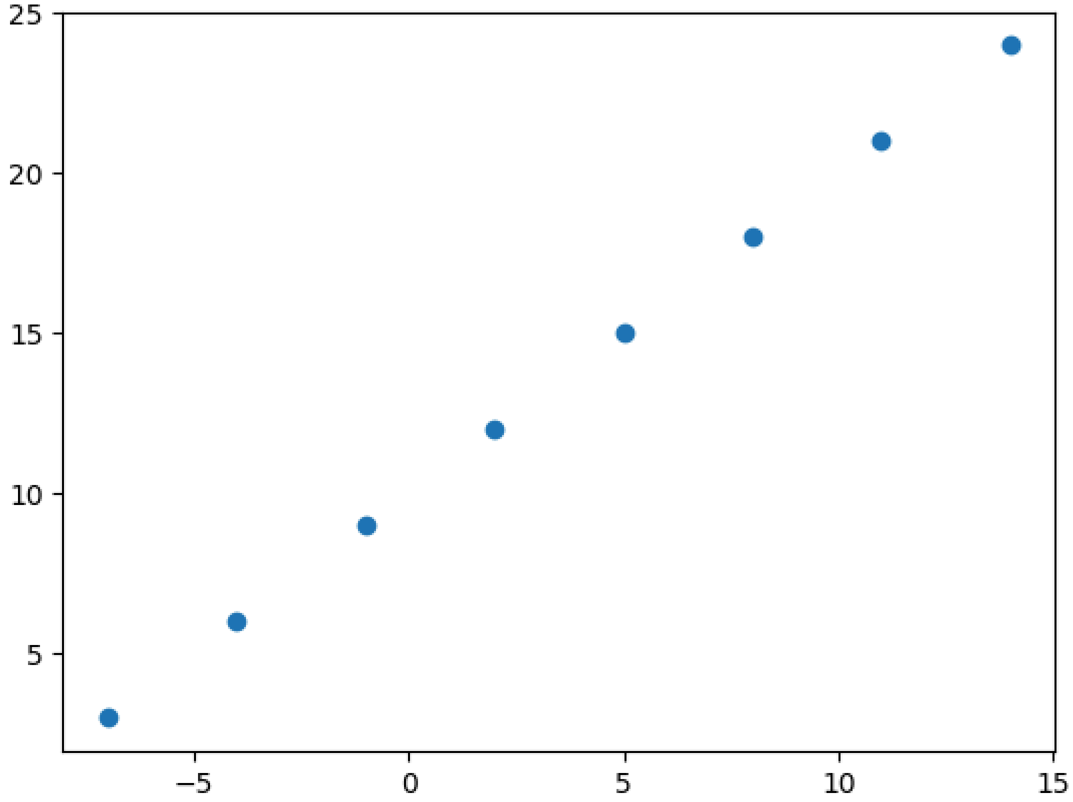
INTRO TO REGRESSION USING NEURAL NETWORKS

```
import tensorflow as tf
tf.__version__
```

```
'2.17.1'
```

```
import numpy as np
import matplotlib.pyplot as plt
X=np.array([-7.0,-4.0,-1.0,2.0,5.0,8.0,11.0,14.0])
Y= np.array([3.0,6.0,9.0,12.0,15.0,18.0,21.0,24.0])
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x7aab719ed720>
```




```
Y==X+10
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

I/p AND O/p SHAPES

```
#create a demo tensor
house_info = tf.constant(['bedroom','bathroom','garage'])
house_price = tf.constant([1000])
house_info,house_price
```

```
(<tf.Tensor: shape=(3,), dtype=string, numpy=array([b'bedroom', b'bathroom',
b'garage'], dtype=object)>,
```

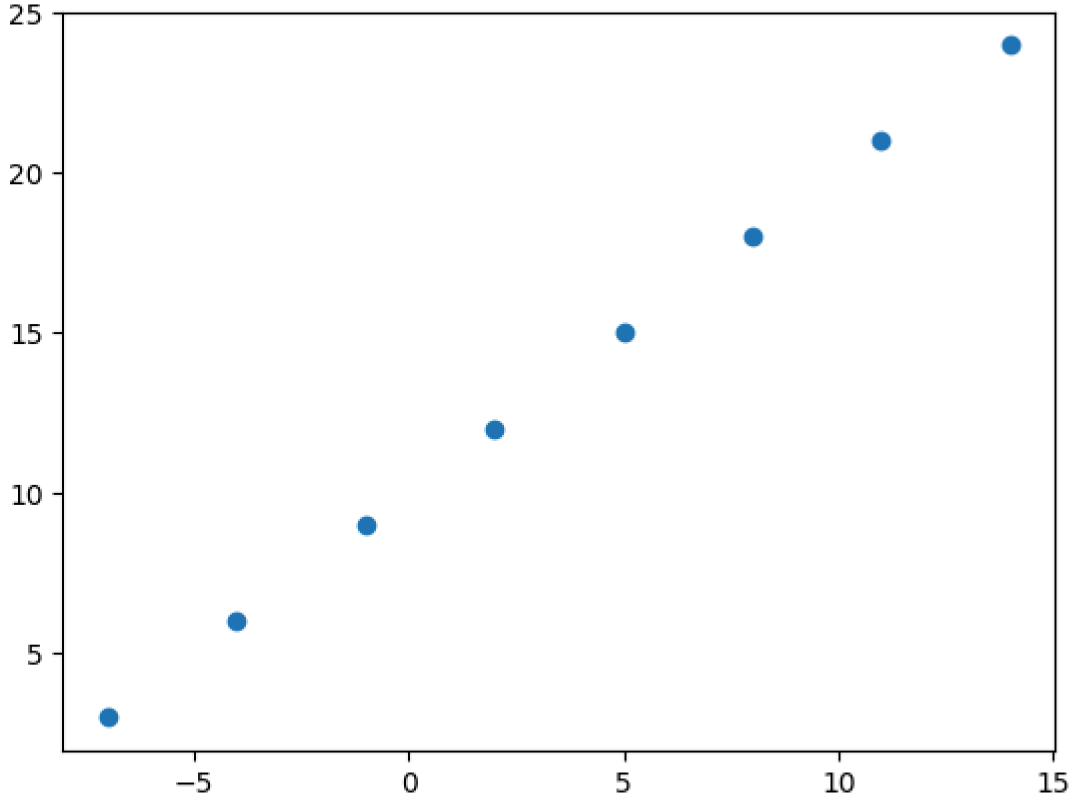
```
<tf.Tensor: shape=(1,), dtype=int32, numpy=array([1000], dtype=int32)>)
```

```
X= tf.constant(X)
Y= tf.constant(Y)
X,Y
```

```
(<tf.Tensor: shape=(8,), dtype=float64, numpy=array([-7., -4., -1.,  2.,  5.,  8.,
11., 14.])>,
 <tf.Tensor: shape=(8,), dtype=float64, numpy=array([ 3.,  6.,  9., 12., 15., 18.,
21., 24.])>)
```

```
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x7aab6f8ed3f0>
```



STEPS TO MODELLING IN TF

> 1. ***CREATE A MODEL*** (DEF I.P O/P LAYER,HIDDEN LAYER),
> 2. **COMPILING A MODEL** (DEFINE LOSS FUNCTION AND OPTIMIZER AND EVALUATION METRICS),
> 3. **FITTING A MODEL**

```
tf.random.set_seed(42)
#1.create a model using sequencial API
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])
#2.compile the model
```

```
model.compile(loss=tf.keras.losses.mae,#mean absolute error
              optimizer=tf.keras.optimizers.SGD(),#stocastic gradient descent
              metrics=['mae'])
#3.fit the model
model.fit(tf.expand_dims(X,axis=-1),Y,epochs=12)
```

```
Epoch 1/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - loss: 20.5018 - mae: 20.5018
Epoch 2/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 20.2205 - mae: 20.2205
Epoch 3/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 19.9393 - mae: 19.9393
Epoch 4/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 19.6580 - mae: 19.6580
Epoch 5/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 19.3768 - mae: 19.3768
Epoch 6/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 19.0955 - mae: 19.0955
Epoch 7/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 18.8143 - mae: 18.8143
Epoch 8/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 18.5330 - mae: 18.5330
Epoch 9/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 18.2518 - mae: 18.2518
Epoch 10/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 17.9705 - mae: 17.9705
Epoch 11/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 17.6893 - mae: 17.6893
Epoch 12/12
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - loss: 17.4080 - mae: 17.4080
<keras.src.callbacks.history.History at 0x7aab71aaa680>
```

try prediction

```
input_value = np.array([[17.0]])  # Add an extra dimension
predicted_value = model.predict(input_value)

print("Predicted value for X=17.0:", predicted_value[0][0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
Predicted value for X=17.0: -14.301005
```

IMPROVE OUR MODEL

1. CREATE A MODEL(ADD MORE LAYERS , INC HIDEN NEURONS, CHANGE ACTIVATION FUNCTION)
2. CHANGE OPTIMIZATION FUNCTION OR LEARNING RATE
3. MORE EPOCS OR MORE DATA

```
#1.create a model using sequencial API
model = tf.keras.Sequential([
```

```
    tf.keras.layers.Dense(50,activation=None),
    # tf.keras.layers.Dense(100,activation='relu'),
    # tf.keras.layers.Dense(100,activation='relu'),
    tf.keras.layers.Dense(1)
])
#2.compile the model
model.compile(loss=tf.keras.losses.mae,#mean absolute error
              #optimizer=tf.keras.optimizers.SGD(),#stocastic gradient descent(SGD)
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              metrics=['mae'])
#3.fit the model
model.fit(tf.expand_dims(X,axis=-1),Y,epochs=100)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.4162 - mae: 0.4162
Epoch 96/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.4815 - mae: 0.4815
Epoch 97/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.2519 - mae: 0.2519
Epoch 98/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2452 - mae: 0.2452
Epoch 99/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.3961 - mae: 0.3961
Epoch 100/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.2612 - mae: 0.2612
<keras.src.callbacks.history.History at 0x7aab71a91b10>
```

```
input_value = np.array([[17.0]])  # Add an extra dimension
predicted_value = model.predict(input_value)

print("Predicted value for X=17.0:", predicted_value[0][0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
Predicted value for X=17.0: 27.368765
```

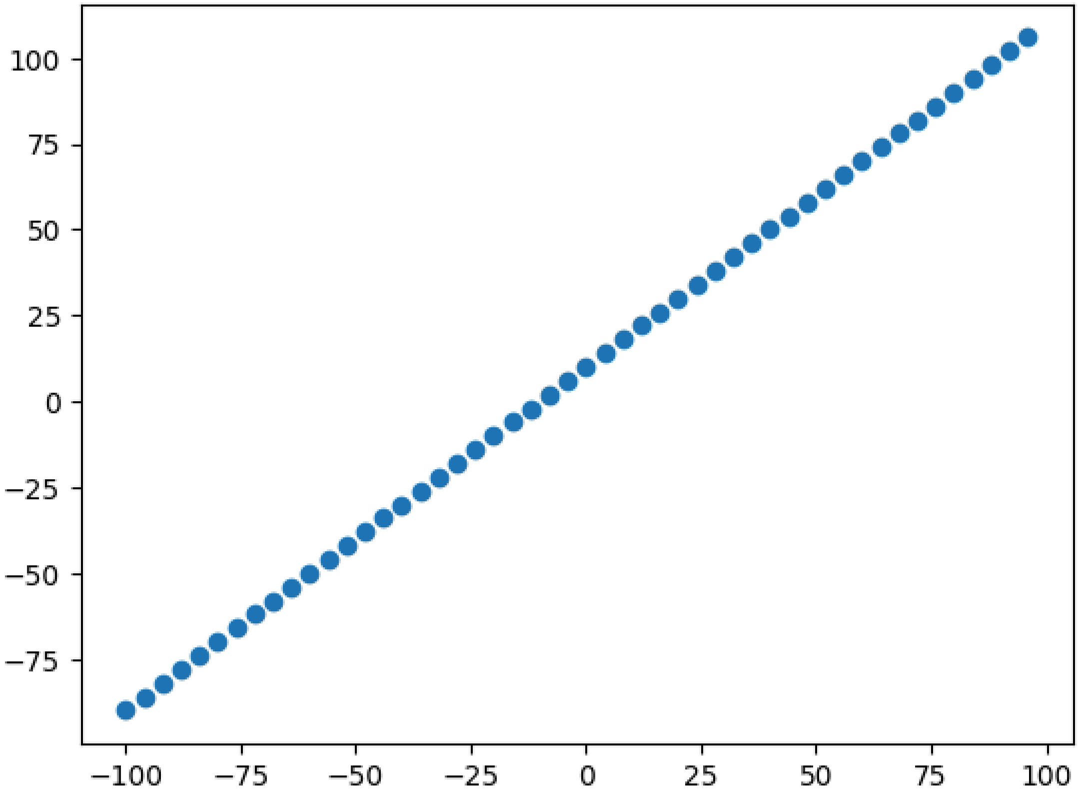
Evaluating our model

three sets...

1. training set 80%
2. validation set 10%(model is tuned on this data)
3. test set 10%

```
X= tf.range(-100,100,4)
Y= X+10
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x7aab6f8f51e0>
```



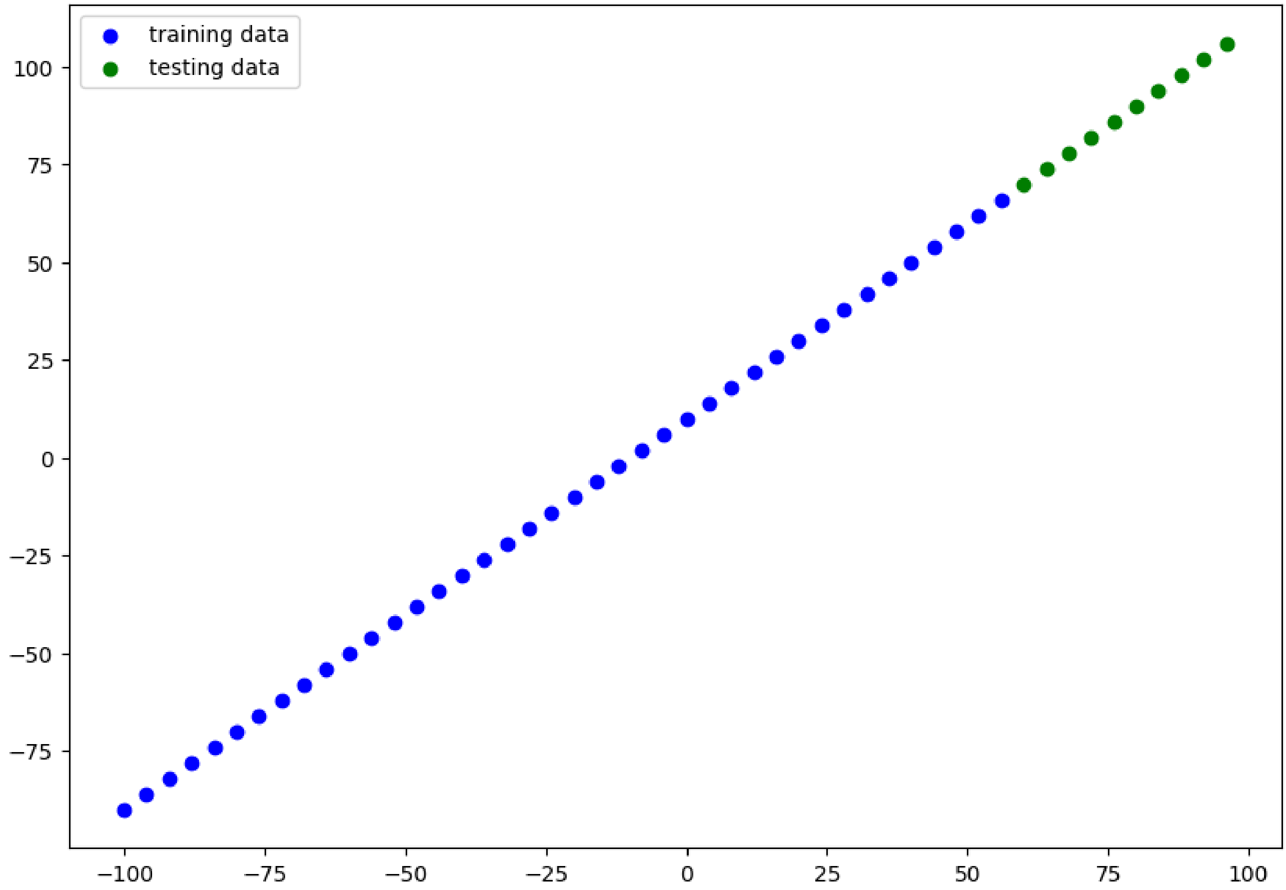

```
len(X)
```

```
50
```

```
X_train =X[:40]
Y_train = Y[:40]
X_test=X[40:]
Y_test=Y[40:]
```

```
plt.figure(figsize=(10,7))
plt.scatter(X_train,Y_train,c='b',label='training data')
plt.scatter(X_test,Y_test,c='g',label='testing data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7aab6c2a86a0>
```




```
#model creation
model = tf.keras.Sequential([

    tf.keras.layers.Dense(1)



])
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae']
              )
```

Visualizing our model

```
model.build()
```

create a model that builds automatically by defining the input shape argument in the first layer

```
tf.random.set_seed(42)
model = tf.keras.Sequential([

    tf.keras.layers.Dense(10,input_shape=[1],name="input_layer"),
    tf.keras.layers.Dense(1,name="output_layer")



],name="model_1")
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae']
              )
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

Model: "model_1"

Layer (type)	Output Shape	Param
input_layer (Dense)	(None, 10)	2
output_layer (Dense)	(None, 1)	1

Total params: 31 (124.00 B)
Trainable params: 31 (124.00 B)
Non-trainable params: 0 (0.00 B)

1. total params = total no of parameters in the model
2. trainable parameters = these parameters the model can update as it trains
3. non-trainable parameters = these parameters are not updated during training(params from other model which are already learned during transfer learning)

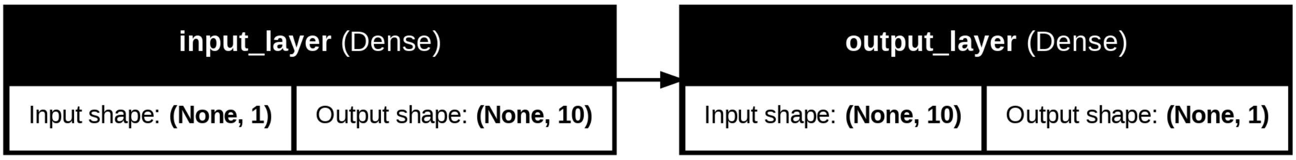
output shape only depends on the last layer of neurons is id = (batch_size,n) n is no of neurons, batch _size is no of samples used in one forward pass and backward pass

```
model.fit(tf.expand_dims(X_train,axis=-1),Y_train,epochs=100,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7aab6c3278b0>
```

```
from tensorflow.keras.utils import plot_model
plot_model(
    model=model,
    show_shapes=True,          # Show the shape of each layer
    show_layer_names=True,     # Display layer names
```

```
    rankdir="LR",             # Horizontal layout (use "TB" for vertical)
    to_file="model.png"       # Save the plot as a PNG file
)
```



Visualizing model's predictions

```
Y_pred= model.predict(tf.expand_dims(X_test,axis=-1))
Y_pred
```

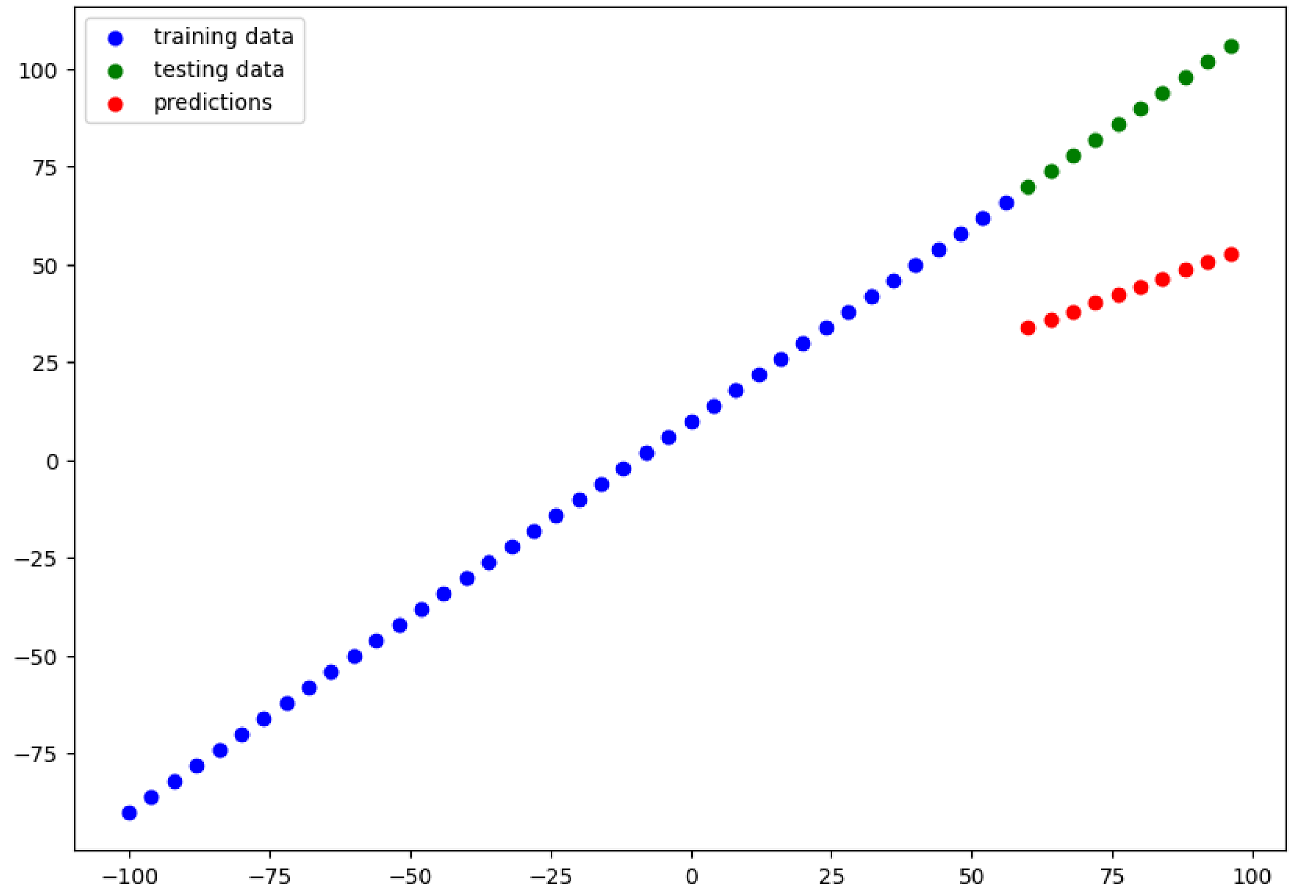
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
array([[33.757755],
       [35.882954],
       [38.00815 ],
       [40.133347],
       [42.258545],
       [44.383743],
       [46.50894 ],
       [48.63414 ],
       [50.759335],
       [52.884533]], dtype=float32)
```

```
Y_test
```

```
<tf.Tensor: shape=(10,), dtype=int32, numpy=array([ 70,  74,  78,  82,  86,  90,
 94,  98, 102, 106], dtype=int32)>
```

```
#plotting function
def plot_predictions(train_data=X_train,train_labels=Y_train,test_data=X_test,test_labels
  """
  Plots training data, test data and compares predictions.
  """
  plt.figure(figsize=(10,7))
  plt.scatter(train_data,train_labels,c='b',label='training data')
  plt.scatter(test_data,test_labels,c='g',label='testing data')
  plt.scatter(test_data,predictions,c='r',label='predictions')
  plt.legend()
```

```
plot_predictions()
```



Evaluate model with metrics

1. MAE- mean avg error (sigma |yi-xi|/n)
2. MSE - mean squared error 1/n(sigma(Yi=Ypredi)^2) , use when larger errors are more significant than smaller errors
3. Huber- combination of MSE AND MAE , less sensitive to outliers

```
model.evaluate (X_test,Y_test)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 241ms/step - loss: 44.6789 - mae: 44.6789
[44.678855895996094, 44.678855895996094]
```

```
tf.constant(Y_pred)
```

```
<tf.Tensor: shape=(10, 1), dtype=float32, numpy=
array([[33.757755],
       [35.882954],
       [38.00815 ],
       [40.133347],
       [42.258545],
       [44.383743],
       [46.50894 ],
       [48.63414 ],
       [50.759335],
       [52.884533]], dtype=float32)>
```

```
Y_test
```

```
<tf.Tensor: shape=(10,), dtype=int32, numpy=array([ 70,  74,  78,  82,  86,  90,
 94,  98, 102, 106], dtype=int32)>
```

```
tf.keras.losses.MAE(Y_test,Y_pred)
```

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([36.242245, 38.117046, 39.99185 , 41.866653, 43.741455, 45.616257,
       47.49106 , 49.36586 , 51.240665, 53.115467], dtype=float32)>
```

```
tf.squeeze(Y_pred)
```

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([33.757755, 35.882954, 38.00815 , 40.133347, 42.258545, 44.383743,
       46.50894 , 48.63414 , 50.759335, 52.884533], dtype=float32)>
```

WHILE COMPARING TENSORS ENSURE THEY ARE IN SAME SHAPE

```
tf.keras.losses.MAE(Y_test,tf.squeeze(Y_pred))
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=44.678856>
```

```
tf.keras.losses.MSE(Y_test,tf.squeeze(Y_pred))
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=2025.198>
```

```
def mae(y_true,y_pred):
  return tf.keras.losses.MAE(y_true=y_true,y_pred=y_pred)
def mse(y_true,y_pred):
  return tf.keras.losses.MSE(y_true=y_true,y_pred=y_pred)
```

3 models will be created

1. model -1 (100 epochs)
2. model -2 2 layers,100 epochs
3. model -3 2 layers, 500 epochs

MODEL1

```
tf.random.set_seed(42)
model_1 = tf.keras.Sequential([
    tf.keras.layers.Dense(1,input_shape=[1])

])
# model.compile(loss=tf.keras.losses.mae,
#               optimizer=tf.keras.optimizers.SGD(),
#               metrics['mae'])
```

```
model_1.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae']
              )
```

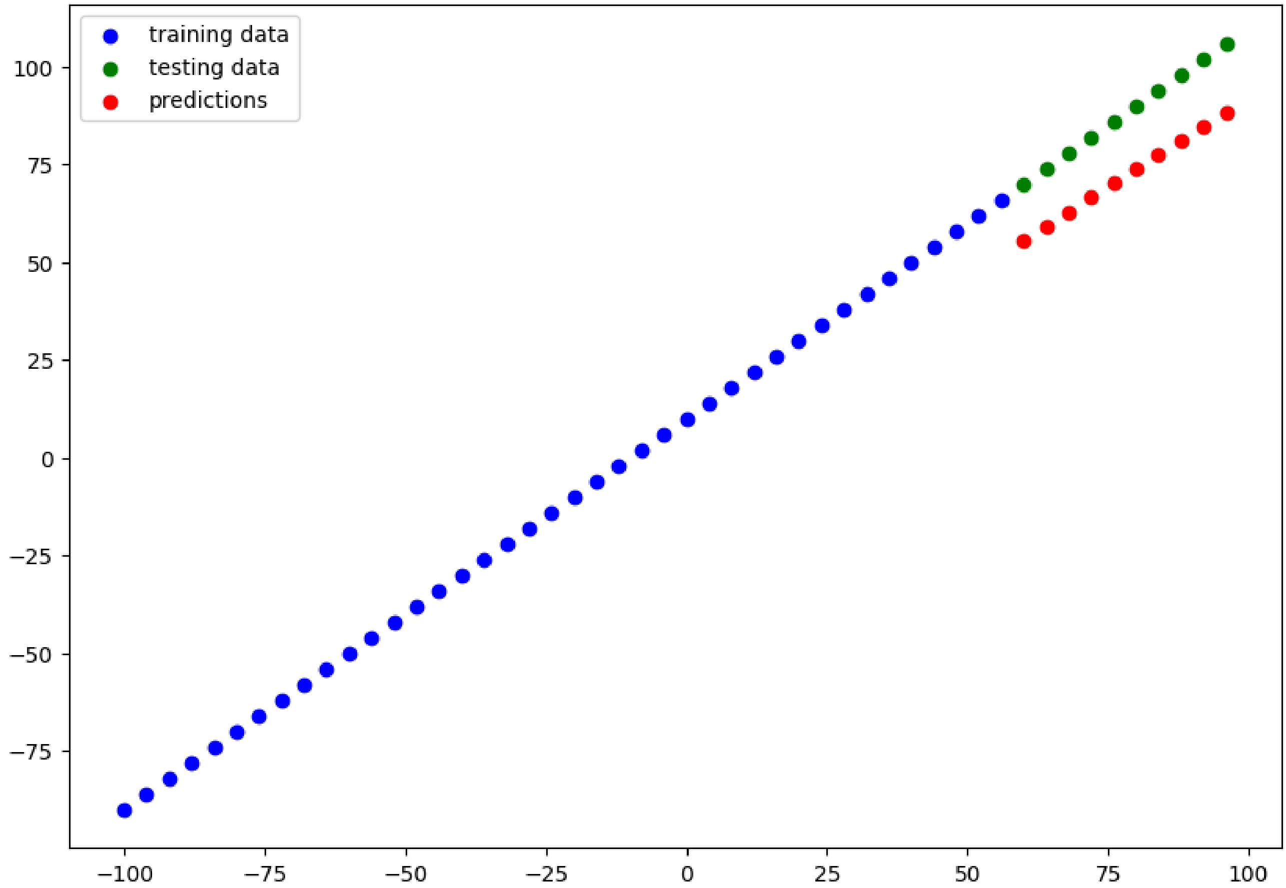
```
model_1.fit(tf.expand_dims(X_train,axis=-1),Y_train,epochs=100,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7aab6ce4d360>
```

```
Y_pred_1=model_1.predict(tf.expand_dims(X_test,axis=-1))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

```
plot_predictions(predictions=tf.squeeze(Y_pred_1))
```




```
mae_1=mae(Y_test,tf.squeeze(Y_pred)).numpy()
mse_1=mse(Y_test,tf.squeeze(Y_pred)).numpy()
mae_1,mse_1
```

```
(44.678856, 2025.198)
```

MODEL-2

```
tf.random.set_seed(42)
model_2 = tf.keras.Sequential([
```

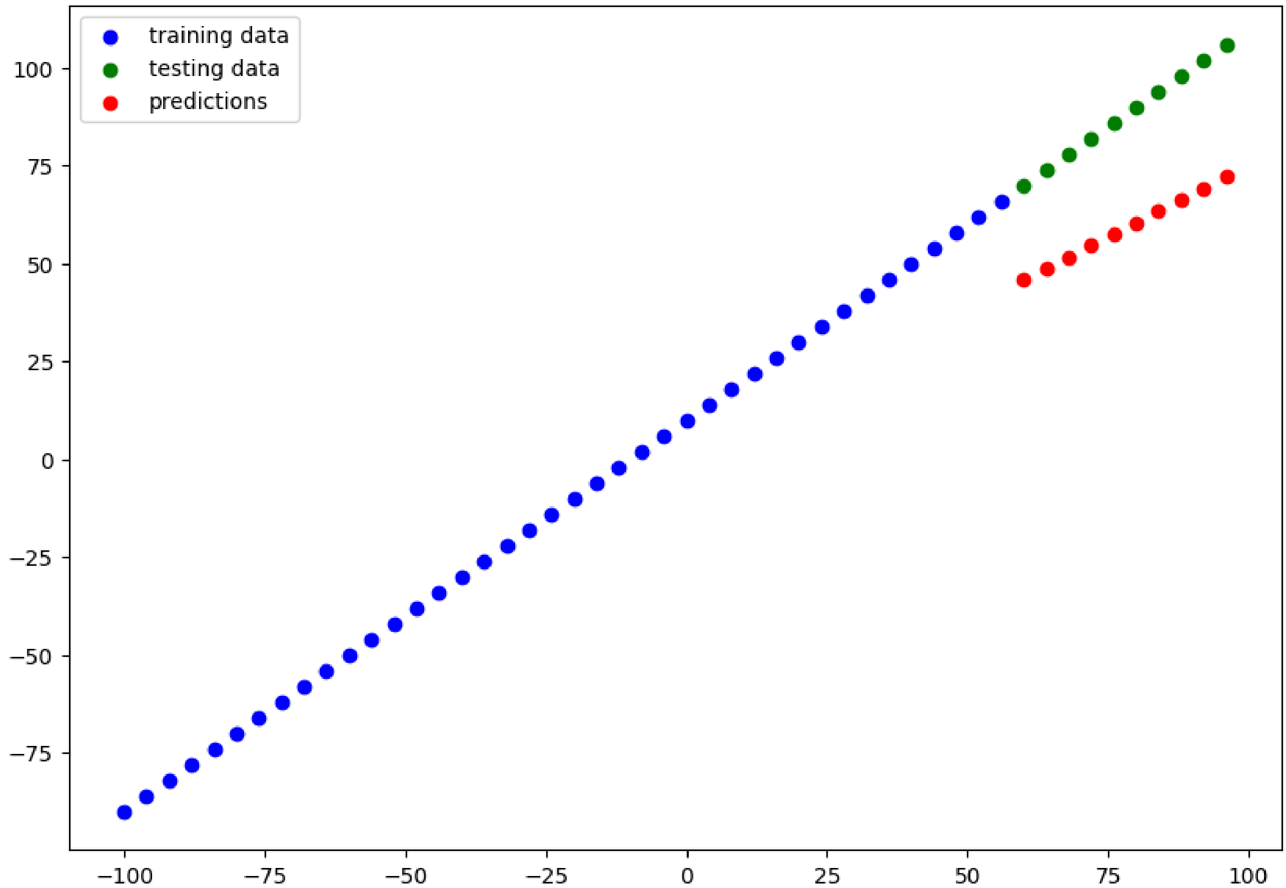
```
    tf.keras.layers.Dense(10,input_shape=[1]),
    tf.keras.layers.Dense(1)


])
model_2.compile(loss=tf.keras.losses.mae,
              optimizer= tf.keras.optimizers.SGD(),
              metrics=['mae'])


model_2.fit(tf.expand_dims(X_train,axis=-1),Y_train,epochs=100,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7aab6cbc67a0>
```

```
Y_pred_2=model_2.predict(tf.expand_dims(X_test,axis=-1))
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_prec
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step
```

```
plot_predictions(predictions=tf.squeeze(Y_pred_2))
```




```
mae_2=mae(Y_test,tf.squeeze(Y_pred_2)).numpy()
mse_2=mse(Y_test,tf.squeeze(Y_pred_2)).numpy()
mae_2,mse_2
```

```
(29.039108, 852.5791)
```

MODEL-3

```
tf.random.set_seed(42)
model_3=tf.keras.Sequential([
    tf.keras.layers.Dense(10,input_shape=[1]),
    tf.keras.layers.Dense(1)
])
model_3.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae'])
```

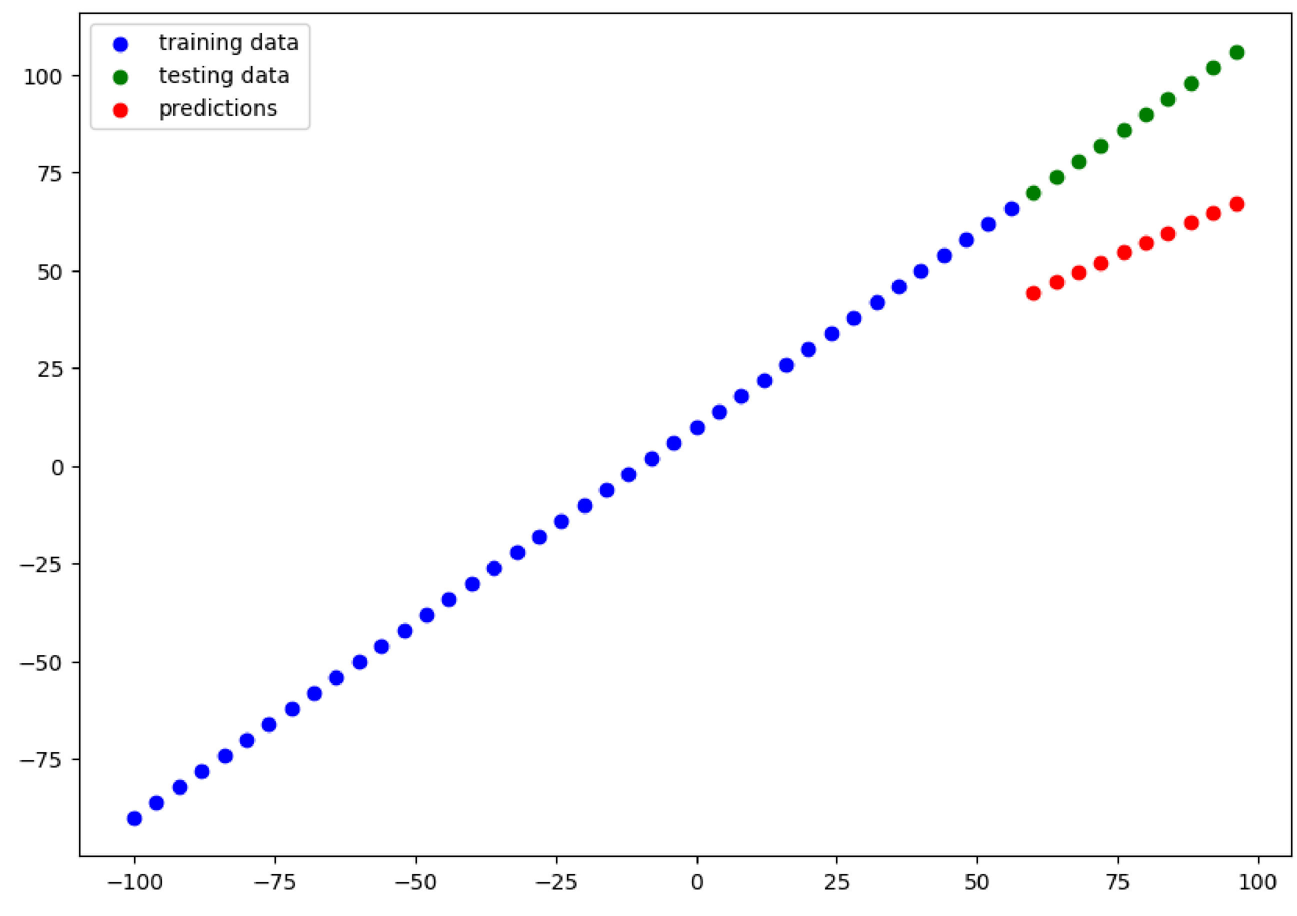
```
model_3.fit(tf.expand_dims(X_train,axis=-1),Y_train,epochs=500,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7aab6cb6a020>
```

```
Y_pred_3=model_3.predict(tf.expand_dims(X_test,axis=-1))
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_prec
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
```

```
plot_predictions(predictions=tf.squeeze(Y_pred_3))
```




```
mae_3=mae(Y_test,tf.squeeze(Y_pred_3)).numpy()
mse_3=mse(Y_test,tf.squeeze(Y_pred_3)).numpy()
```

```
mae_3,mse_3
```

```
(32.19206, 1054.2557)
```

```
import pandas as pd
tuples=[['mae',mae_1,mae_2,mae_3],['mse',mse_1,mse_2,mse_3]]
eval = pd.DataFrame(tuples,columns=['metric','model1','model2','model3'])
eval
```

	metric	model1	model2	model3
0	mae	44.678856	29.039108	32.192059
1	mse	2025.197998	852.579102	1054.255737

Next steps: Generate code with eval | View recommended plots | New interactive sheet

```
model_3.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param
dense_7 (Dense)	(None, 10)	2
dense_8 (Dense)	(None, 1)	1

```
Total params: 33 (136.00 B)
Trainable params: 31 (124.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
```

tracking experiments

1. TensorBoard- component of the tensorflow library
2. Weights& Biases-to track expriment

Saving our model allows us to use them outside google colab 2 main formats

1. keras FORMAT
2. THE HDF5 FORMAT

```
model_2.save("best_model.keras")
```

```
load_keras=tf.keras.models.load_model("best_model.keras")#load mdoel
```

```
load_keras.summary()
```

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param
dense_5 (Dense)	(None, 10)	2
dense_6 (Dense)	(None, 1)	1

```
Total params: 33 (136.00 B)
Trainable params: 31 (124.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
```

```
model_2.summary()
```

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param
dense_5 (Dense)	(None, 10)	2
dense_6 (Dense)	(None, 1)	1

```
Total params: 33 (136.00 B)
Trainable params: 31 (124.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)
```

```
expanded_X_test = tf.expand_dims(X_test, axis=-1)
model_2_preds = model_2.predict(expanded_X_test)
load_model_pred=load_keras.predict(expanded_X_test)
model_2_preds==load_model_pred
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
array([[ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True]])
```

download a model from google colab

1. files->file->download
2. from google.colab import files files.download("\filepath")

A LARGER DATA SET

```
#IMPORT LIBRARIES
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
insurance=pd.read_csv("https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-da
```

```
insurance
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Next steps: Generate code with `insurance` | View recommended plots | New interactive sheet

one hot encoding

```
insurance_onehot=pd.get_dummies(insurance)
insurance_onehot.head()
```

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	re
0	19	27.900	0	16884.92400	True	False	False	True	
1	18	33.770	1	1725.55230	False	True	True	False	
2	28	33.000	3	4449.46200	False	True	True	False	
3	33	22.705	0	21984.47061	False	True	True	False	
4	32	28.880	0	3866.85520	False	True	True	False	

Next steps: Generate code with insurance_onehot View recommended plots New interactive sheet

```
X=insurance_onehot.drop("charges",axis=1)
Y=insurance_onehot["charges"]
```

Generate randomly select 5 items from a list Close

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size=0.2,random_state=42)
```

```
tf.random.set_seed(42)
insurance_model = tf.keras.Sequential([
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])
insurance_model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=['mae']
              )
```

```
insurance_model.fit(X_train,Y_train,epochs=100,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7aab6c0ff580>
```

```
Y_pred=insurance_model.predict(X_test)
```

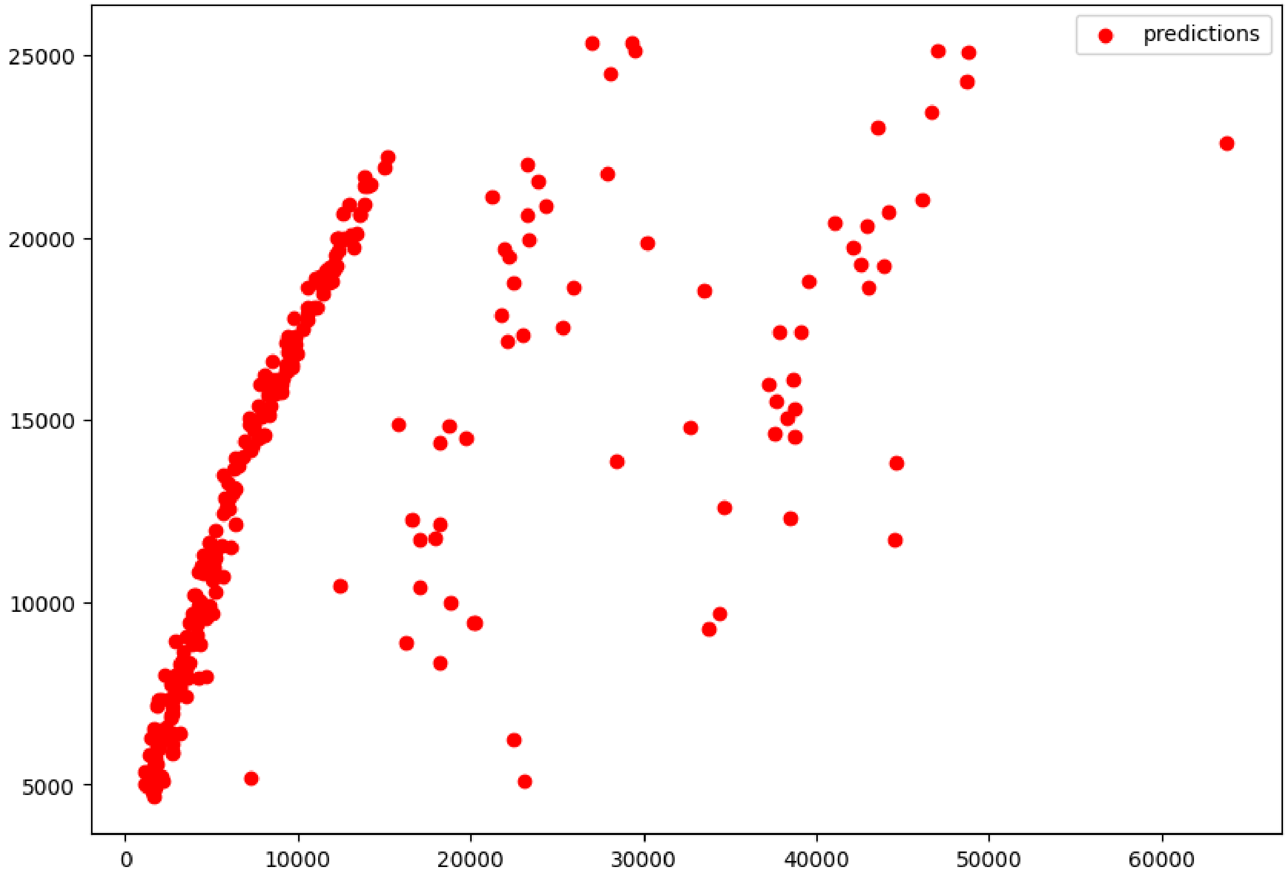
```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

```
print("Shape of X_test:", X_test.shape)
print("Shape of Y_test:", Y_test.shape)
print("Shape of Y_pred:", Y_pred.shape)
```

```
Shape of X_test: (268, 11)
Shape of Y_test: (268,)
Shape of Y_pred: (268, 1)
```

```
plt.figure(figsize=(10,7))


plt.scatter(Y_test,tf.squeeze(Y_pred),c='r',label='predictions')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7aab59f86fb0>
```



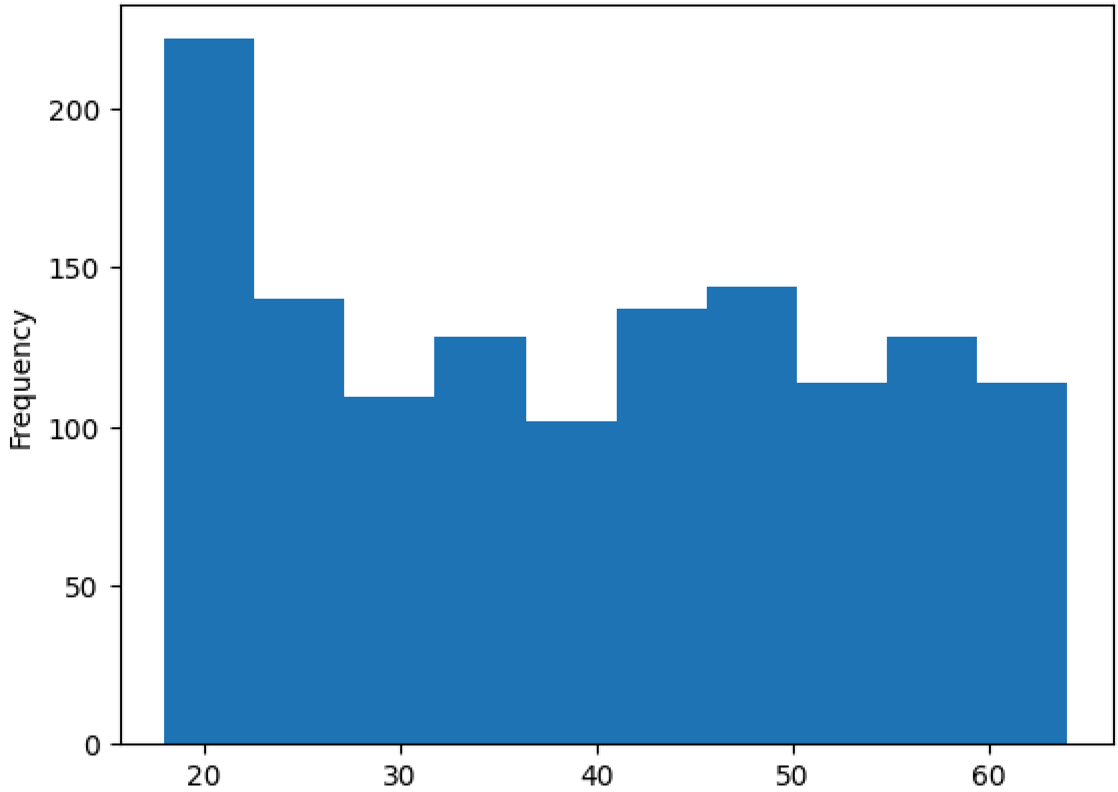
PREPROCESSING DATA(NORMALIZATION AND STANDARDIZATION)

from os import remove

1. MinMaxScaler - converts all values b/w 0 annd 1 while preserving original distribution
2. StandardSclaer - removes the mean and divides each value by the std dev

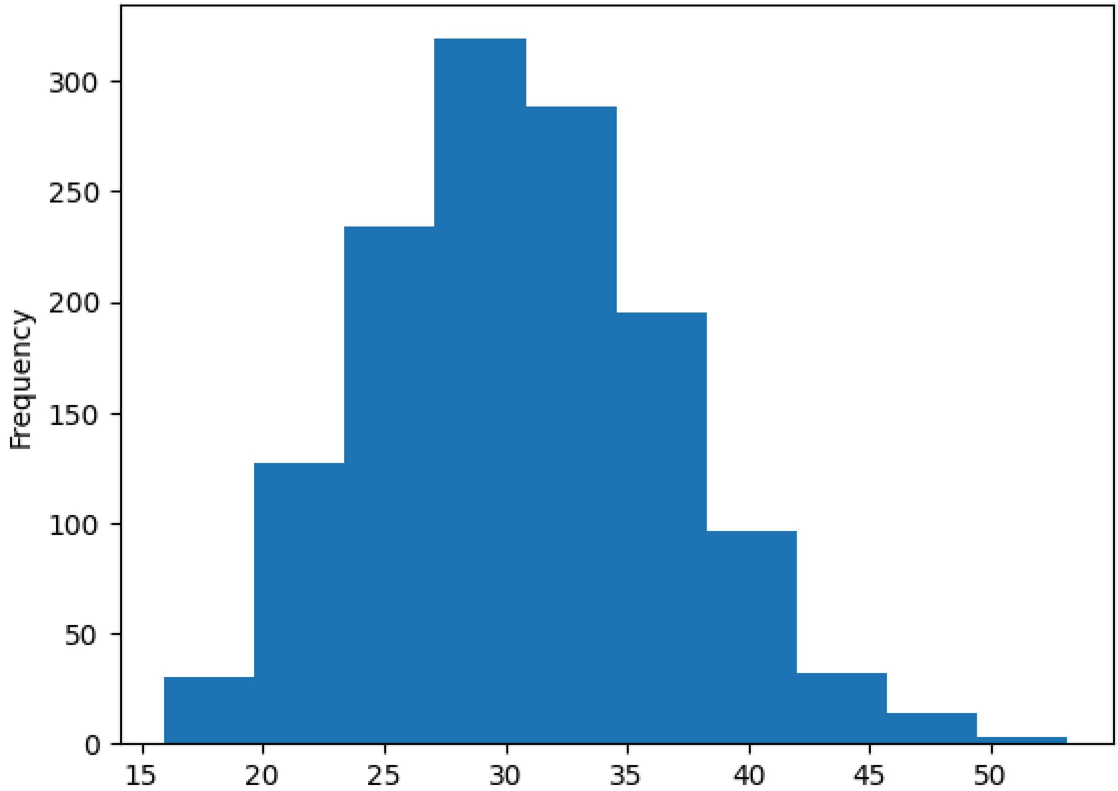
```
insurance['age'].plot(kind='hist')
```

```
<Axes: ylabel='Frequency'>
```




```
insurance['bmi'].plot(kind='hist')
```

```
<Axes: ylabel='Frequency'>
```




```
from sklearn.preprocessing import MinMaxScaler,StandardScaler,OneHotEncoder
from sklearn.compose import make_column_transformer
insurance.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Next steps: Generate code with insurance | View recommended plots | New interactive sheet

```
ct = make_column_transformer(
    (MinMaxScaler(),['age','bmi','children']),
    (OneHotEncoder(handle_unknown='ignore'),['sex','smoker','region'])
)
X=insurance.drop("charges",axis=1)
Y=insurance["charges"]
X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size=0.2,random_state=42)


# sclaer=MinMaxScaler()
# X_train_scaled=sclaer.fit_transform(X_train)
# X_test_scaled=sclaer.fit_transform(X_test)


# X_train_scaled,X_train
#fit he column transformer to our training data
ct.fit(X_train)
```

```
ColumnTransformer
minmaxscaler        onehotencoder
MinMaxScaler        OneHotEncoder
```

```
X_train_normal = ct.transform(X_train)
X_test_normal = ct.transform(X_test)
```

```
X_train_normal[0]
```

```
array([0.60869565, 0.10734463, 0.4       , 1.        , 0.        ,
       1.        , 0.        , 0.        , 1.        , 0.        ,
       0.        ])
```

```
X_train.shape,X_train_normal.shape#extra cols bec of one hot encoding
```

```
((1070, 6), (1070, 11))
```

```
tf.random.set_seed(42)
insurance_model_2 = tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)
])
insurance_model_2.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['mae']
              )
```

```
insurance_model_2.fit(X_train_normal,Y_train,epochs=100)
```

```
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3566.1824 - mae: 3566.1824
Epoch 73/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3565.5720 - mae: 3565.5720
Epoch 74/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 3564.9211 - mae: 3564.9211
Epoch 75/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 3565.2742 - mae: 3565.2742
Epoch 76/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3564.9783 - mae: 3564.9783
Epoch 77/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3565.1074 - mae: 3565.1074
Epoch 78/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3565.4009 - mae: 3565.4009
Epoch 79/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3565.8633 - mae: 3565.8633
Epoch 80/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 3565.9495 - mae: 3565.9495
Epoch 81/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3566.2512 - mae: 3566.2512
Epoch 82/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3566.4624 - mae: 3566.4624
Epoch 83/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3566.9065 - mae: 3566.9065
Epoch 84/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 3567.0105 - mae: 3567.0105
Epoch 85/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3567.3911 - mae: 3567.3911
Epoch 86/100
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3567.3857 - mae: 3567.3857
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.