Objective : Convert Celesius to Farhenheit data given in csv file by using Perceptron Algorithm of Neural Network. Use Tensorflow, Keras libaray to build the Neural Network.

```
import pandas as pd
import tensorflow as tf
```

```
from google.colab import files
files_upload = files.upload()
```

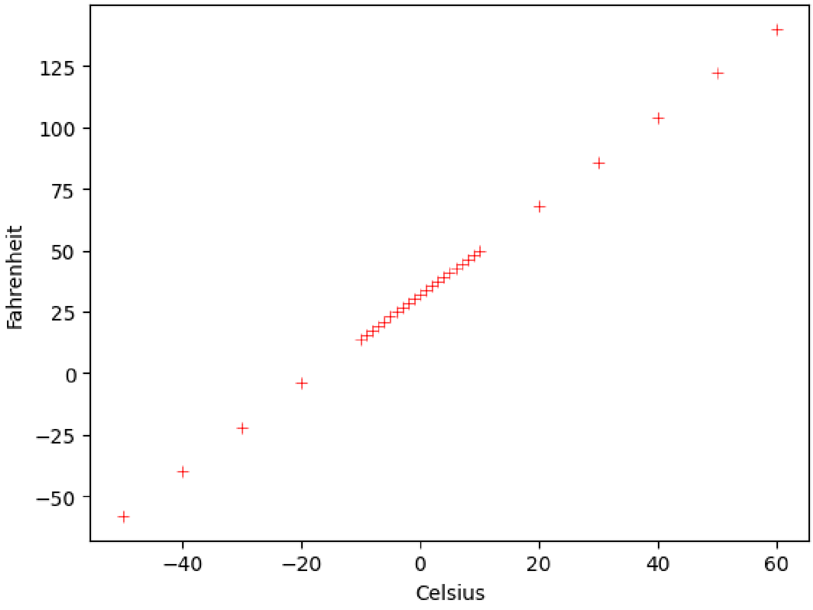
Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser

```
df = pd.read_csv('Celsius+to+Fahrenheit.csv')
df.head()
```

	Celsius	Fahrenheit
0	-50	-58.0
1	-40	-40.0
2	-30	-22.0
3	-20	-4.0
4	-10	14.0

```
import seaborn as sns
sns.scatterplot(x=df['Celsius'],y=df['Fahrenheit'],color='red',marker='+')
```

```
<Axes: xlabel='Celsius', ylabel='Fahrenheit'>
```



```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(df['Celsius'],df['Fahrenheit'],test_size=0.2,rand
```

```
X_train.shape,y_train.shape
```

```
((24,), (24,))
```

```
X_test.shape,y_test.shape
```

```
((6,), (6,))
```

```
#Build the model-Perceptron
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(units=1,input_shape=[1]))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.5),loss='mean_squared_error')
```

```
epochs_hist=model.fit(X_train,y_train,epochs=500)
```

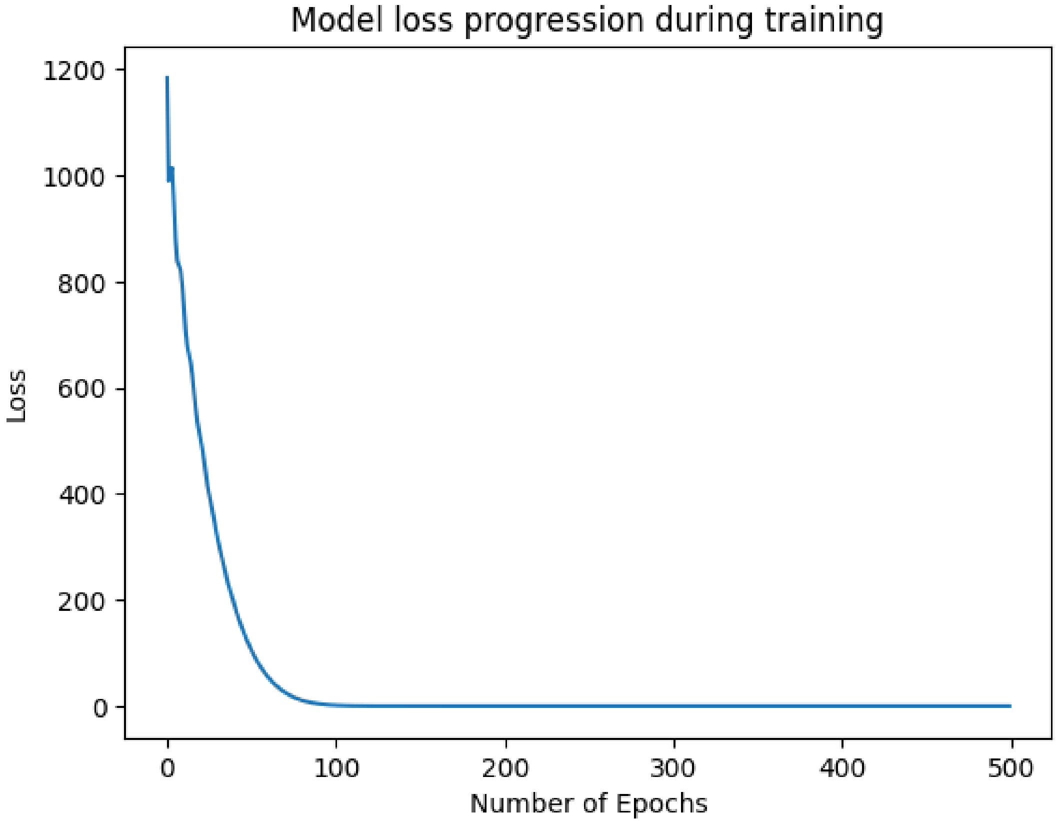
```
Epoch 147/500
1/1 [==============================] - 0s 6ms/step - loss: 9.3692e-04
Epoch 148/500
1/1 [==============================] - 0s 8ms/step - loss: 0.0011
Epoch 149/500
1/1 [==============================] - 0s 8ms/step - loss: 0.0012
Epoch 150/500
1/1 [==============================] - 0s 12ms/step - loss: 0.0013
Epoch 151/500
1/1 [==============================] - 0s 10ms/step - loss: 0.0014
Epoch 152/500
1/1 [==============================] - 0s 11ms/step - loss: 0.0014
Epoch 153/500
1/1 [==============================] - 0s 10ms/step - loss: 0.0015
Epoch 154/500
1/1 [==============================] - 0s 9ms/step - loss: 0.0015
Epoch 155/500
1/1 [==============================] - 0s 8ms/step - loss: 0.0015
Epoch 156/500
1/1 [==============================] - 0s 8ms/step - loss: 0.0015
Epoch 157/500
1/1 [==============================] - 0s 7ms/step - loss: 0.0015
Epoch 158/500
1/1 [==============================] - 0s 7ms/step - loss: 0.0015
Epoch 159/500
1/1 [==============================] - 0s 7ms/step - loss: 0.0015
Epoch 160/500
1/1 [==============================] - 0s 8ms/step - loss: 0.0014
Epoch 161/500
1/1 [==============================] - 0s 9ms/step - loss: 0.0014
Epoch 162/500
1/1 [==============================] - 0s 10ms/step - loss: 0.0013
Epoch 163/500
1/1 [==============================] - 0s 11ms/step - loss: 0.0013
Epoch 164/500
```

```
epochs_hist.history['loss']
```

```
[1183.69140625,
 989.644287109375,
 1007.7645874023438,
 1014.4931030273438,
 950.9345703125,
 878.4695434570312,
 838.819580078125,
 830.17822265625,
 821.28271484375,
 791.2744750976562,
 746.3317260742188,
 704.2516479492188,
 676.7549438476562,
 661.457763671875,
 646.3070678710938,
 622.2114868164062,
 590.4083862304688,
 558.9146118164062,
 534.5076293945312,
 517.351318359375,
 501.8316345214844,
 482.420654296875,
 458.5843811035156,
 434.3080139160156,
 413.81787109375,
 397.8353576660156,
 383.4305725097656,
 367.2420349121094,
 348.654052734375,
 329.901123046875,
 313.5269775390625,
 299.95233154296875,
 287.3757019042969,
 273.8359680175781,
 259.1446838378906,
 244.77978515625,
 232.15625,
 221.21372985839844,
 210.6864776611328,
 199.5709228515625,
 188.1104278564453,
 177.3350830078125,
 167.8485565185547,
 159.2420196533203,
 150.67481994628906,
 141.83917236328125,
 133.19989013671875,
 125.34940338134766,
 118.3181381225586,
 111.6152114868164,
 104.85223388671875,
 98.15711212158203,
 91.93209075927734,
 86.3317642211914,
 81.1090087890625,
 75.95719146728516,
 70.87117767333984,
 66.0922622680664
```

```
import matplotlib.pyplot as plt
plt.plot(epochs_hist.history['loss'])
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.title('Model loss progression during training')
```

```
Text(0.5, 1.0, 'Model loss progression during training')
```



```
model.get_weights()
```

```
[array([[1.8]], dtype=float32), array([32.000015], dtype=float32)]
```

```
C_F = 100
print(f'Our Perceptron model prediction is:{model.predict([C_F])}')
```

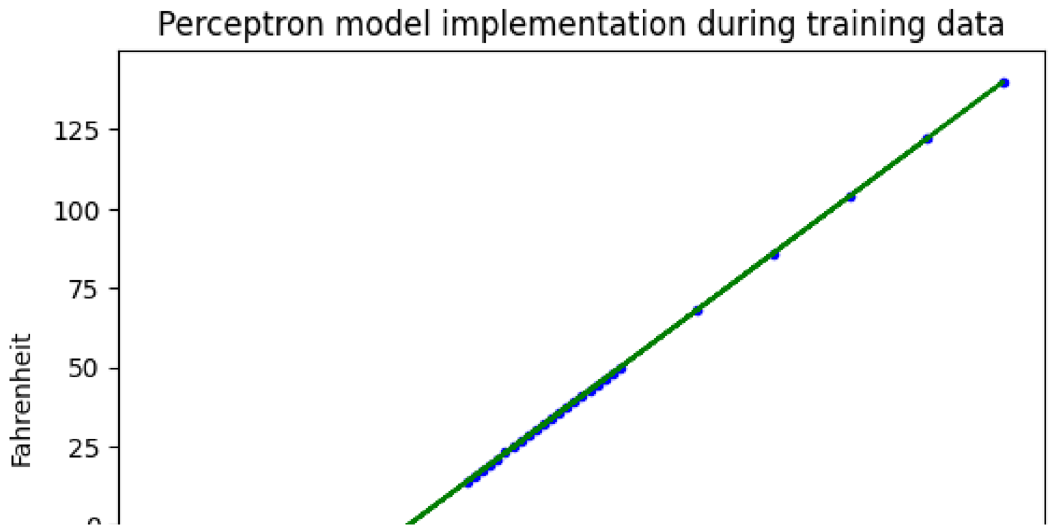
```
1/1 [==============================] - 0s 36ms/step
Our Perceptron model prediction is:[[212.00002]]
```

```
actual_true_prediction = 9/5 * C_F + 32
print(f'Our actual true value is:{actual_true_prediction}')
```

```
Our actual true value is:212.0
```

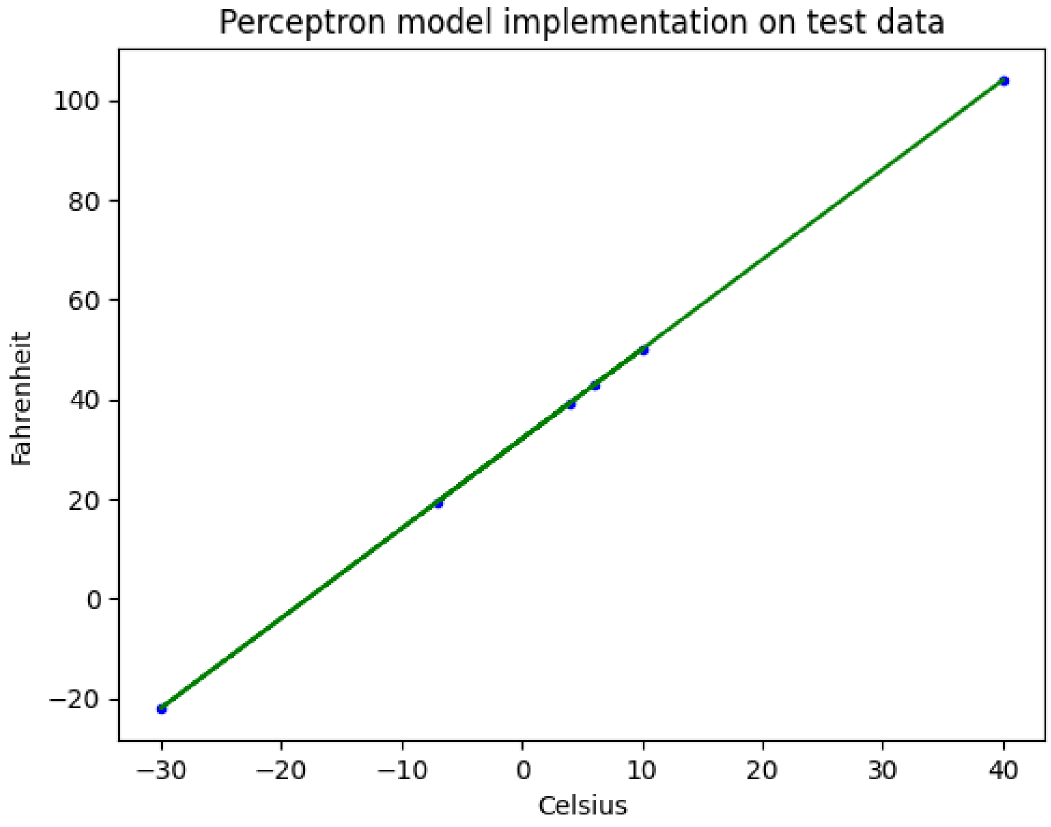
```
plt.scatter(df['Celsius'],df['Fahrenheit'],marker='.',color='b')
plt.plot(X_train,model.predict(X_train),color='green')
plt.xlabel('Celsius')
plt.ylabel('Fahrenheit')
plt.title('Perceptron model implementation during training data')
```

```
1/1 [==============================] - 0s 20ms/step
Text(0.5, 1.0, 'Perceptron model implementation during training data')
```



```
plt.scatter(X_test,y_test,marker='.',color='b')
plt.plot(X_test,model.predict(X_test),color='green')
plt.xlabel('Celsius')
plt.ylabel('Fahrenheit')
plt.title('Perceptron model implementation on test data')
```

```
1/1 [==============================] - 0s 24ms/step
Text(0.5, 1.0, 'Perceptron model implementation on test data')
```


Perceptron model implementation on test data
Fahrenheit
Celsius