```
df=pd.read_csv('/content/drive/MyDrive/insurance.csv')
df.head()
```

	id	Date	number of bedrooms	number of bathrooms	living area	lot area	number of floors	waterfront present	number of views	co
0	6762810145	42491	5	2.50	3650	9050	2.0	0	4	
1	6762810635	42491	4	2.50	2920	4000	1.5	0	0	
2	6762810998	42491	5	2.75	2910	9480	1.5	0	0	
3	6762812605	42491	4	2.50	3310	42998	2.0	0	0	
4	6762812919	42491	3	2.00	2710	4500	1.5	0	0	

5 rows × 23 columns

```
#import all the libraries
import pandas as pd
import numpy as np
import tensorflow as tf
```

```
df.shape
```

```
(14620, 23)
```

```
#checking null values
df.isnull().sum()
```

```
id                                    0
Date                                  0
number of bedrooms                    0
number of bathrooms                   0
living area                           0
lot area                              0
number of floors                      0
waterfront present                    0
number of views                       0
condition of the house                0
grade of the house                    0
Area of the house(excluding basement)    0
Area of the basement                  0
Built Year                            0
Renovation Year                       0
Postal Code                           0
Lattitude                             0
Longitude                             0
living_area_renov                     0
lot_area_renov                        0
Number of schools nearby              0
Distance from the airport             0
Price                                 0
dtype: int64
```

```
#checking the datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14620 entries, 0 to 14619
Data columns (total 23 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   id                                    14620 non-null  int64  
 1   Date                                  14620 non-null  int64  
 2   number of bedrooms                    14620 non-null  int64  
 3   number of bathrooms                   14620 non-null  float64
 4   living area                           14620 non-null  int64  
 5   lot area                              14620 non-null  int64  
 6   number of floors                      14620 non-null  float64
 7   waterfront present                    14620 non-null  int64  
 8   number of views                       14620 non-null  int64  
 9   condition of the house                14620 non-null  int64  
 10  grade of the house                    14620 non-null  int64  
 11  Area of the house(excluding basement)  14620 non-null  int64
```

```
 12  Area of the basement                14620 non-null  int64
 13  Built Year                          14620 non-null  int64
 14  Renovation Year                     14620 non-null  int64
 15  Postal Code                         14620 non-null  int64
 16  Lattitude                           14620 non-null  float64
 17  Longitude                           14620 non-null  float64
 18  living_area_renov                   14620 non-null  int64
 19  lot_area_renov                      14620 non-null  int64
 20  Number of schools nearby            14620 non-null  int64
 21  Distance from the airport           14620 non-null  int64
 22  Price                               14620 non-null  int64
dtypes: float64(4), int64(19)
memory usage: 2.6 MB
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
#convert the string type to int type
df['Price']=le.fit_transform(df['Price'])
```

```
#split the data independent& dependent)
x=df.iloc[:,0:-1].values
x
```

```
array([[6.76281014e+09, 4.24910000e+04, 5.00000000e+00, ...,
        5.40000000e+03, 2.00000000e+00, 5.80000000e+01],
       [6.76281064e+09, 4.24910000e+04, 4.00000000e+00, ...,
        4.00000000e+03, 2.00000000e+00, 5.10000000e+01],
       [6.76281100e+09, 4.24910000e+04, 5.00000000e+00, ...,
        6.60000000e+03, 1.00000000e+00, 5.30000000e+01],
       ...,
       [6.76283062e+09, 4.27340000e+04, 2.00000000e+00, ...,
        6.12000000e+03, 2.00000000e+00, 6.40000000e+01],
       [6.76283071e+09, 4.27340000e+04, 4.00000000e+00, ...,
        6.63100000e+03, 3.00000000e+00, 5.40000000e+01],
       [6.76283146e+09, 4.27340000e+04, 3.00000000e+00, ...,
        3.48000000e+03, 2.00000000e+00, 5.50000000e+01]])
```

```
y=df.iloc[:,4:5].values
```

```
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=0)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print(x_train)
```

```
[[-1.46639097 -1.16155586  0.65582076 ... -0.17007044 -1.23664001
  -0.99104498]
 [-0.32364482 -1.08747669 -0.39868565 ... -0.25454935  1.2121438
  -0.99104498]
 [ 1.57399277  0.67560763 -0.39868565 ... -0.13110387  1.2121438
   0.91013359]
 ...
 [ 0.71357026  0.55708096 -0.39868565 ... -0.19790921  1.2121438
   0.12729536]
 [ 0.61620624  0.79413431  1.71032717 ... -0.18693542  1.2121438
  -0.43187481]
 [-0.46744892 -1.01339751  0.65582076 ... -0.27934626 -0.01224811
   0.46279746]]
```

```
# one-encode the geograpy column
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
x = np.array(ct.fit_transform(x))
print(x)
```

```
  (0, 0)        1.0
  (0, 241)      6762810145.0
  (0, 242)      5.0
  (0, 243)      2.5
```

```
(0, 244)      3650.0
(0, 245)      9050.0
(0, 246)      2.0
(0, 248)      4.0
(0, 249)      5.0
(0, 250)      10.0
(0, 251)      3370.0
(0, 252)      280.0
(0, 253)      1921.0
(0, 255)      122003.0
(0, 256)      52.8645
(0, 257)      -114.557
(0, 258)      2880.0
(0, 259)      5400.0
(0, 260)      2.0
(0, 261)      58.0
(1, 0)        1.0
(1, 241)      6762810635.0
(1, 242)      4.0
(1, 243)      2.5
(1, 244)      2920.0
:     :
(14618, 256)  52.7157
(14618, 257)  -114.411
(14618, 258)  1420.0
(14618, 259)  6631.0
(14618, 260)  3.0
(14618, 261)  54.0
(14619, 240)  1.0
(14619, 241)  6762831463.0
(14619, 242)  3.0
(14619, 243)  1.0
(14619, 244)  900.0
(14619, 245)  4770.0
(14619, 246)  1.0
(14619, 249)  3.0
(14619, 250)  6.0
(14619, 251)  900.0
(14619, 253)  1969.0
(14619, 254)  2009.0
(14619, 255)  122018.0
(14619, 256)  52.5338
(14619, 257)  -114.552
(14619, 258)  900.0
(14619, 259)  3480.0
(14619, 260)  2.0
(14619, 261)  55.0
```

Build the ANN Model

```
import keras
from keras.models import Sequential
from keras.layers import Dense


#initializing the Ann
ann=tf.keras.models.Sequential()
```

Input layer

```
# add input layer
ann.add(tf.keras.layers.Dense(units=12,activation='relu',input_shape=x_train[0].shape))
```

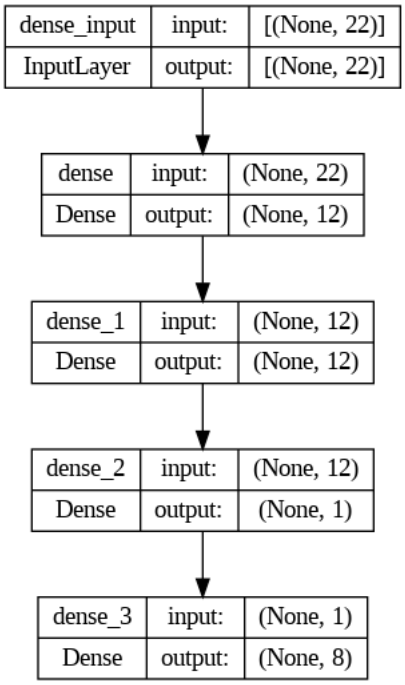
Hidden Layer

```
#hidding two layers
ann.add(tf.keras.layers.Dense(units=8, activation='relu'))
```

Output Layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
from tensorflow.keras.utils import plot_model
plot_model(ann,
           to_file="model.png",
           show_shapes=True,
           show_layer_names=True,
          )
```



Test Model

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
ann.fit(x_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
366/366 [==============================] - 3s 2ms/step - loss: 4548.2520 - accuracy: 0.0000e+00
Epoch 2/100
366/366 [==============================] - 1s 2ms/step - loss: -3840.6453 - accuracy: 0.0000e+00
Epoch 3/100
366/366 [==============================] - 1s 3ms/step - loss: -6026.0781 - accuracy: 0.0000e+00
Epoch 4/100
366/366 [==============================] - 1s 4ms/step - loss: -7861.8940 - accuracy: 0.0000e+00
Epoch 5/100
366/366 [==============================] - 1s 3ms/step - loss: -7861.9023 - accuracy: 0.0000e+00
Epoch 6/100
366/366 [==============================] - 1s 3ms/step - loss: -7861.9092 - accuracy: 0.0000e+00
Epoch 7/100
366/366 [==============================] - 1s 2ms/step - loss: -7861.9058 - accuracy: 0.0000e+00
Epoch 8/100
366/366 [==============================] - 1s 2ms/step - loss: -7861.9102 - accuracy: 0.0000e+00
Epoch 9/100
366/366 [==============================] - 1s 2ms/step - loss: -7861.9351 - accuracy: 0.0000e+00
Epoch 10/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.1113 - accuracy: 0.0000e+00
Epoch 11/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.1108 - accuracy: 0.0000e+00
Epoch 12/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.1216 - accuracy: 0.0000e+00
Epoch 13/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.1240 - accuracy: 0.0000e+00
Epoch 14/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3057 - accuracy: 0.0000e+00
Epoch 15/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3032 - accuracy: 0.0000e+00
Epoch 16/100
```

```
366/366 [==============================] - 1s 2ms/step - loss: -7862.3066 - accuracy: 0.0000e+00
Epoch 17/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3066 - accuracy: 0.0000e+00
Epoch 18/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3066 - accuracy: 0.0000e+00
Epoch 19/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3057 - accuracy: 0.0000e+00
Epoch 20/100
366/366 [==============================] - 1s 3ms/step - loss: -7862.3066 - accuracy: 0.0000e+00
Epoch 21/100
366/366 [==============================] - 1s 3ms/step - loss: -7862.3042 - accuracy: 0.0000e+00
Epoch 22/100
366/366 [==============================] - 1s 3ms/step - loss: -7862.3057 - accuracy: 0.0000e+00
Epoch 23/100
366/366 [==============================] - 1s 3ms/step - loss: -7862.3101 - accuracy: 0.0000e+00
Epoch 24/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3042 - accuracy: 0.0000e+00
Epoch 25/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3066 - accuracy: 0.0000e+00
Epoch 26/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3042 - accuracy: 0.0000e+00
Epoch 27/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3076 - accuracy: 0.0000e+00
Epoch 28/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3091 - accuracy: 0.0000e+00
Epoch 29/100
366/366 [==============================] - 1s 2ms/step - loss: -7862.3091 - accuracy: 0.0000e+00
```

```
y_pred = ann.predict(x_test)
y_pred = (y_pred > 0.5)
pd.DataFrame(list(zip(y_test, y_pred)), columns=['Actual', 'Predicted'])
```

```
92/92 [==============================] - 1s 5ms/step
```

	Actual	Predicted
0	[1440]	[False, False, True, True, True, True, True, F...
1	[4270]	[False, False, True, True, True, True, True, F...
2	[1010]	[False, False, True, True, True, True, True, F...
3	[1970]	[False, False, True, True, True, True, True, F...
4	[2320]	[False, False, True, True, True, True, True, F...
...	...	...
2919	[4980]	[False, False, True, True, True, True, True, F...
2920	[2360]	[False, False, True, True, True, True, True, F...
2921	[2230]	[False, False, True, True, True, True, True, F...
2922	[2710]	[False, False, True, True, True, True, True, F...
2923	[1240]	[False, False, True, True, True, True, True, F...

2924 rows × 2 columns