```
import pandas as pd  #importing pandas which is powerful software library for data manipulati
```

```
df = pd.read_csv('/content/sample_data/housepricedata.csv')  #reading the csv file
```

```
df  #checking the contents of DataFrame df after reading the csv file.
```

	LotArea	OverallQual	OverallCond	TotalBsmtSF	FullBath	HalfBath	BedroomA
0	8450	7	5	856	2	1	
1	9600	6	8	1262	2	0	
2	11250	7	5	920	2	1	
3	9550	7	5	756	1	0	
4	14260	8	5	1145	2	1	
...	...	...	...	...	...	...	
1455	7917	6	5	953	2	1	
1456	13175	6	6	1542	2	0	
1457	9042	7	9	1152	2	0	
1458	9717	5	6	1078	1	0	
1459	9937	5	6	1256	1	1	

1460 rows × 11 columns

```
dataset=df.values #Pandas DataFrame df is a 2D size muatable tabular data structure with labe
                  #of the given DataFrame df
```

```
dataset   #to check the contents of the "dataset"
```

```
array([[ 8450,     7,     5, ...,     0,   548,     1],
       [ 9600,     6,     8, ...,     1,   460,     1],
       [11250,     7,     5, ...,     1,   608,     1],
       ...,
       [ 9042,     7,     9, ...,     2,   252,     1],
       [ 9717,     5,     6, ...,     0,   240,     0],
       [ 9937,     5,     6, ...,     0,   276,     0]])
```

```
x = dataset[:,0:10]
y = dataset[:,10]
```

```
from sklearn import preprocessing #provides several common utility functions and transformer
```

```
min_max_scaler = preprocessing.MinMaxScaler()      #Transform features by scaling each feaure
X_scale = min_max_scaler.fit_transform(x)          #Fit to data and then transform it x=input
```

```
X_scale    #load the transformed data
```

```
array([[0.0334198 , 0.66666667, 0.5       , ..., 0.5       , 0.        ,
        0.3864598 ],
       [0.03879502, 0.55555556, 0.875     , ..., 0.33333333, 0.33333333,
        0.32440056],
       [0.04650728, 0.66666667, 0.5       , ..., 0.33333333, 0.33333333,
        0.42877292],
       ...,
       [0.03618687, 0.66666667, 1.        , ..., 0.58333333, 0.66666667,
        0.17771509],
       [0.03934189, 0.44444444, 0.625     , ..., 0.25      , 0.        ,
        0.16925247],
       [0.04037019, 0.44444444, 0.625     , ..., 0.33333333, 0.        ,
        0.19464034]])
```

```
from sklearn.model_selection import train_test_split   #for splitting arrays or matrices into
```

```
X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, y, test_size=0.3
```

```
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.3
```

```
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)
```

```
(1022, 10) (306, 10) (132, 10) (1022,) (306,) (132,)
```

```
from keras.models import Sequential  #appropriate for plain stack of layers where ech layer h
 from keras.layers import Dense      #does operation on input to return the output :- output
```

```
#creating layers
model = Sequential([
Dense(32, activation='relu', input_shape=(10,)),    #relu -> applies rectified linear unit ac
Dense(32, activation='relu'),
Dense(1, activation='sigmoid'),       #sigmoid -> applies sigmoid activation function
])

model.compile(optimizer='sgd',    #String name of optimiser or optimiser instance SGD -> Grad

loss='binary_crossentropy',       #loss function. binary_crossentropy -> computes the cross e
metrics=['accuracy'])             #metrics accuracy -> calculates how often prediction equals

hist = model.fit(X_train, Y_train,    #display training data history and fit the model
batch_size=32, epochs=100,            #number of sapmples per gradient update. if unspecified
```

```
    validation_data=(X_val, Y_val))       #data on which to evalute a loss and any model metrics
```

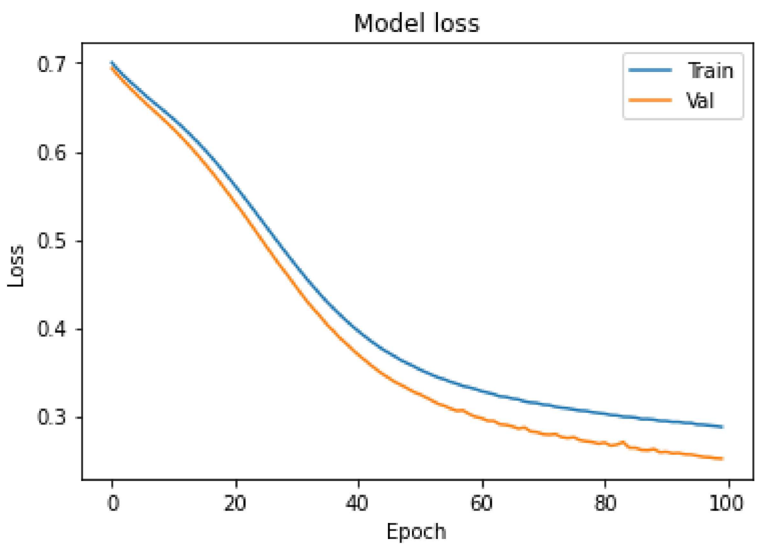
```
Epoch 36/100
32/32 [==============================] - 0s 2ms/step - loss: 0.4289 - accuracy: 0.861
Epoch 37/100
32/32 [==============================] - 0s 2ms/step - loss: 0.4219 - accuracy: 0.859
Epoch 38/100
32/32 [==============================] - 0s 3ms/step - loss: 0.4151 - accuracy: 0.864
Epoch 39/100
32/32 [==============================] - 0s 3ms/step - loss: 0.4085 - accuracy: 0.862
Epoch 40/100
32/32 [==============================] - 0s 2ms/step - loss: 0.4022 - accuracy: 0.865
Epoch 41/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3963 - accuracy: 0.865
Epoch 42/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3907 - accuracy: 0.867
Epoch 43/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3853 - accuracy: 0.865
Epoch 44/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3803 - accuracy: 0.866
Epoch 45/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3754 - accuracy: 0.867
Epoch 46/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3713 - accuracy: 0.867
Epoch 47/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3673 - accuracy: 0.867
Epoch 48/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3630 - accuracy: 0.866
Epoch 49/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3596 - accuracy: 0.867
Epoch 50/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3564 - accuracy: 0.868
Epoch 51/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3527 - accuracy: 0.870
Epoch 52/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3496 - accuracy: 0.869
Epoch 53/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3466 - accuracy: 0.873
Epoch 54/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3436 - accuracy: 0.868
Epoch 55/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3415 - accuracy: 0.871
Epoch 56/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3387 - accuracy: 0.874

Epoch 57/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3366 - accuracy: 0.875
Epoch 58/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3340 - accuracy: 0.873
Epoch 59/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3325 - accuracy: 0.870
Epoch 60/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3306 - accuracy: 0.870
Epoch 61/100
32/32 [==============================] - 0s 4ms/step - loss: 0.3282 - accuracy: 0.872
Epoch 62/100
32/32 [==============================] - 0s 3ms/step - loss: 0.3265 - accuracy: 0.872
Epoch 63/100
```

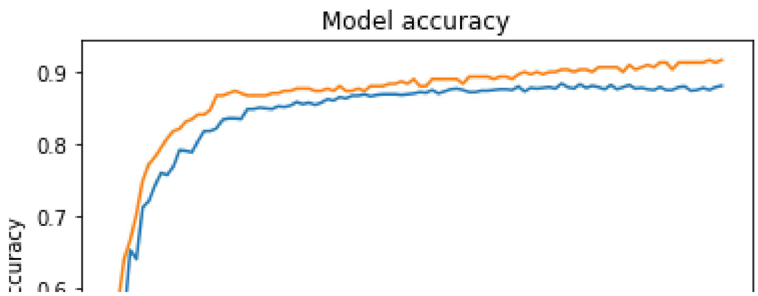
```
32/32 [==============================] - 0s 3ms/step - loss: 0.3248 - accuracy: 0.873
Epoch 64/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3223 - accuracy: 0.874
```

```
import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
#from the plot of model loss we can see that model has comaparable perfromance on both train
#if the parallel plots are depating continuosly then it might be the sign that to stop the tr
```




```
import matplotlib.pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
#from the plot of accuracy we can see that the model could probably be trained a little more
#it is like a method for measuring the classification model's performance based on training a
```




```
model_2 = Sequential([
Dense(1000, activation='relu', input_shape=(10,)),
Dense(1000, activation='relu'),
Dense(1000, activation='relu'),
Dense(1000, activation='relu'),
Dense(1, activation='sigmoid'),
])


model_2.compile(optimizer='adam',   #this time trying the adam optimiser , optimiser that is
loss='binary_crossentropy',
metrics=['accuracy'])


hist_2 = model_2.fit(X_train, Y_train,   #training the data as done previously, fitting it an
batch_size=32, epochs=100,
validation_data=(X_val, Y_val))
```

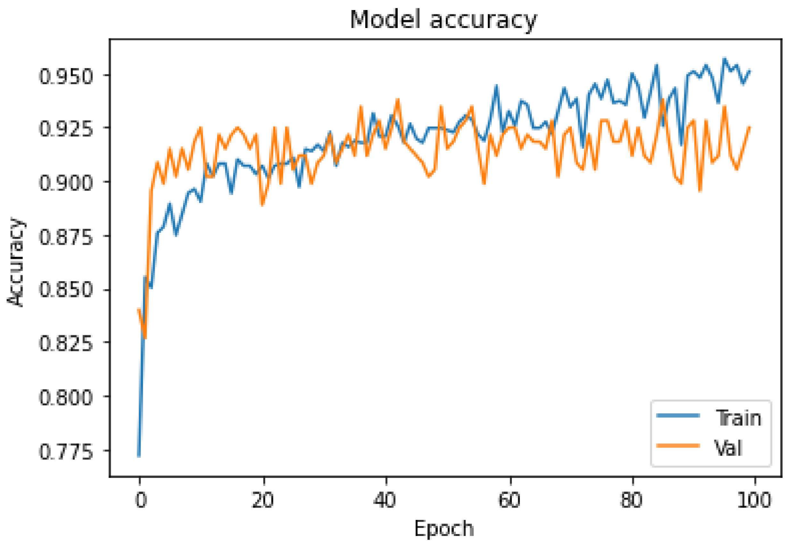
```
Epoch 71/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1500 - accuracy: 0.93

Epoch 72/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1440 - accuracy: 0.93
Epoch 73/100
32/32 [==============================] - 1s 41ms/step - loss: 0.1765 - accuracy: 0.91
Epoch 74/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1442 - accuracy: 0.94
Epoch 75/100
32/32 [==============================] - 1s 42ms/step - loss: 0.1399 - accuracy: 0.94
Epoch 76/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1426 - accuracy: 0.93
Epoch 77/100
32/32 [==============================] - 1s 38ms/step - loss: 0.1287 - accuracy: 0.94
Epoch 78/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1447 - accuracy: 0.93
Epoch 79/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1341 - accuracy: 0.93
Epoch 80/100
32/32 [==============================] - 1s 38ms/step - loss: 0.1594 - accuracy: 0.93
Epoch 81/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1268 - accuracy: 0.95
Epoch 82/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1519 - accuracy: 0.94
Epoch 83/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1477 - accuracy: 0.92
Epoch 84/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1448 - accuracy: 0.94
```

```
Epoch 85/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1192 - accuracy: 0.954
Epoch 86/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1717 - accuracy: 0.92
Epoch 87/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1357 - accuracy: 0.93
Epoch 88/100
32/32 [==============================] - 1s 42ms/step - loss: 0.1267 - accuracy: 0.94
Epoch 89/100
32/32 [==============================] - 1s 41ms/step - loss: 0.1822 - accuracy: 0.91
Epoch 90/100
32/32 [==============================] - 1s 41ms/step - loss: 0.1243 - accuracy: 0.949
Epoch 91/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1154 - accuracy: 0.95
Epoch 92/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1280 - accuracy: 0.94
Epoch 93/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1131 - accuracy: 0.954
Epoch 94/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1193 - accuracy: 0.94
Epoch 95/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1442 - accuracy: 0.93
Epoch 96/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1200 - accuracy: 0.95
Epoch 97/100
32/32 [==============================] - 1s 41ms/step - loss: 0.1067 - accuracy: 0.95
Epoch 98/100
32/32 [==============================] - 1s 40ms/step - loss: 0.1144 - accuracy: 0.954
Epoch 99/100
32/32 [==============================] - 1s 41ms/step - loss: 0.1294 - accuracy: 0.94
```

```
plt.plot(hist_2.history['loss'])
plt.plot(hist_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
#A model loss graph generated using adam optimiser
```


```
plt.plot(hist_2.history['accuracy'])
plt.plot(hist_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
#a model accuracy graph generated sing adam optimiser
```




```
from keras.layers import Dropout   #this layer randomly sets input units to 0 with a frequenc
from keras import regularizers     #alloes you to apply penalities on layer parameters or lay


model_3 = Sequential([
Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_shape=(10,)),
Dropout(0.3),
Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
Dropout(0.3),
Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
Dropout(0.3),
Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
Dropout(0.3),
Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])
model_3.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])


model_3.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy']),
```

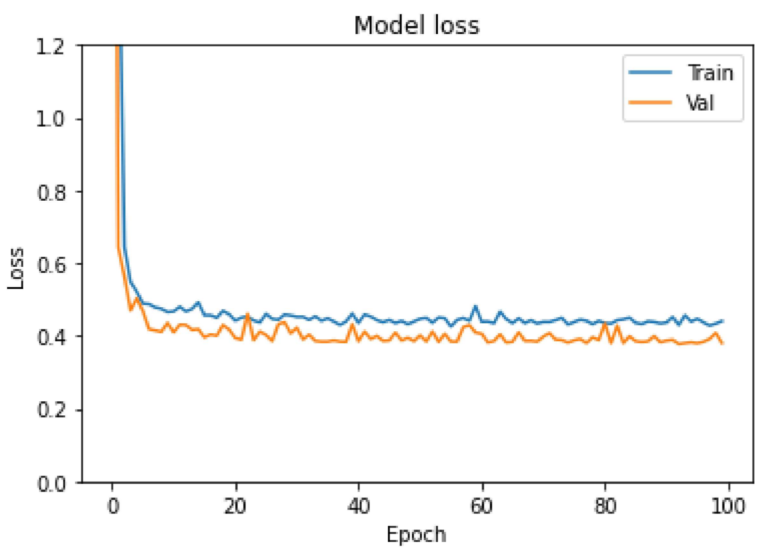
```
(None,)
```

```
hist_3 = model_3.fit(X_train, Y_train,
batch_size=32, epochs=100,
validation_data=(X_val, Y_val))
```

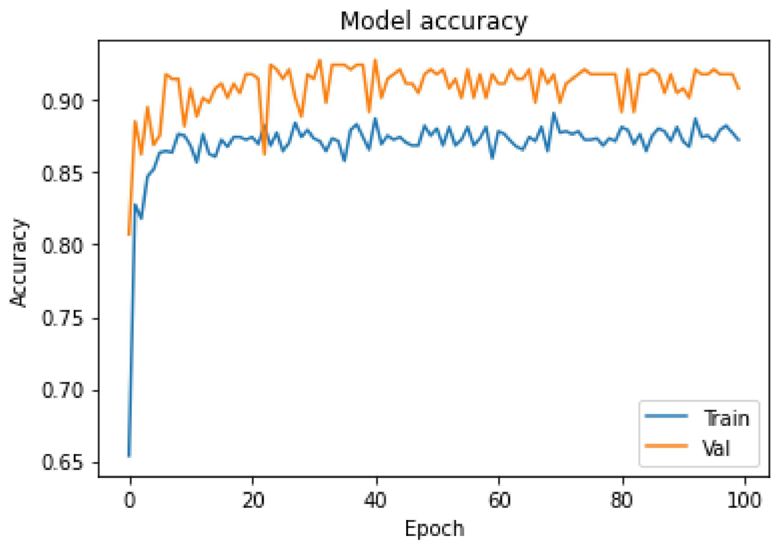
```
32/32 [==============================] - 2s 48ms/step - loss: 0.4496 - accuracy: 0.874
Epoch 19/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4689 - accuracy: 0.874
Epoch 20/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4593 - accuracy: 0.87
Epoch 21/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4419 - accuracy: 0.874
Epoch 22/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4502 - accuracy: 0.869
Epoch 23/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4532 - accuracy: 0.88
Epoch 24/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4421 - accuracy: 0.86
Epoch 25/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4372 - accuracy: 0.87
Epoch 26/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4605 - accuracy: 0.86
Epoch 27/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4466 - accuracy: 0.870
Epoch 28/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4445 - accuracy: 0.884
Epoch 29/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4585 - accuracy: 0.874
Epoch 30/100
32/32 [==============================] - 2s 48ms/step - loss: 0.4562 - accuracy: 0.879
Epoch 31/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4521 - accuracy: 0.87
Epoch 32/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4523 - accuracy: 0.87
Epoch 33/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4441 - accuracy: 0.86
Epoch 34/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4538 - accuracy: 0.87
Epoch 35/100
32/32 [==============================] - 2s 49ms/step - loss: 0.4411 - accuracy: 0.87
Epoch 36/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4486 - accuracy: 0.85
Epoch 37/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4395 - accuracy: 0.879
Epoch 38/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4302 - accuracy: 0.88
Epoch 39/100
32/32 [==============================] - 2s 52ms/step - loss: 0.4396 - accuracy: 0.874
Epoch 40/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4616 - accuracy: 0.86
Epoch 41/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4357 - accuracy: 0.88
Epoch 42/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4587 - accuracy: 0.869
Epoch 43/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4531 - accuracy: 0.87
```

```
Epoch 44/100
32/32 [==============================] - 2s 51ms/step - loss: 0.4432 - accuracy: 0.87
Epoch 45/100
32/32 [==============================] - 2s 52ms/step - loss: 0.4375 - accuracy: 0.87
Epoch 46/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4440 - accuracy: 0.87
Epoch 47/100
32/32 [==============================] - 2s 50ms/step - loss: 0.4353 - accuracy: 0.86
```

```
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.ylim(top=1.2, bottom=0)
plt.show()
#model loss graph generated after applying the kernel regularisers.
```




```
plt.plot(hist_3.history['accuracy'])
plt.plot(hist_3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
#model accuracy graph generated after applying the kernel regualisers
```


Model accuracy
0.90
0.85
0.80
0.75
0.70
0.65
Accuracy
0
20
40
60
80
100
Epoch
Train
Val