Prediction of price for second hand cars using `Regression`

Imports

In [1]:
```
import tensorflow as tf
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
from tensorflow.keras.layers import Normalization, Dense, InputLayer
from tensorflow.keras.losses import MeanAbsoluteError, MeanSquaredError, Huber
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import RootMeanSquaredError, Accuracy
import numpy as np
```

Read Data

In [2]:
```
data = pd.read_csv("SecondHandCarDataSet.csv")
data.head()
```

Out[2]:

	v.id	on road old	on road now	years	km	rating	condition	economy	top speed	hp	torque	current price
0	1	535651	798186	3	78945	1	2	14	177	73	123	351318.0
1	2	591911	861056	6	117220	5	9	9	148	74	95	285001.5
2	3	686990	770762	2	132538	2	8	15	181	53	97	215386.0
3	4	573999	722381	4	101065	4	3	11	197	54	116	244295.5
4	5	691388	811335	6	61559	3	9	12	160	53	105	531114.5

Data Processing

In [3]:
```
tensor_data = tf.constant(data)
tensor_data = tf.random.shuffle(tensor_data)
```

In [4]:
```
X = tensor_data[:, 3:-1]
Y = tensor_data[:,-1]
Y = tf.expand_dims(Y,axis=-1)
```

Splitting the data into training, testing and validation data

In [5]:
```
TRAIN_RATIO = 0.8
TEST_RATIO = 0.1
VALIDATION_RATIO = 0.1
DATASET_SIZE = len(data)
```

In [6]:
```
#Training Data
x_train=X[:int(DATASET_SIZE*TRAIN_RATIO)]
y_train=Y[:int(DATASET_SIZE*TRAIN_RATIO)]

#Testing Data
x_test = X[int(DATASET_SIZE*TRAIN_RATIO):int(DATASET_SIZE*(TRAIN_RATIO+TEST_RATIO))]
y_test = Y[int(DATASET_SIZE*TRAIN_RATIO):int(DATASET_SIZE*(TRAIN_RATIO+TEST_RATIO))]

#Validation Data
x_validate = X[int(DATASET_SIZE*(TRAIN_RATIO+TEST_RATIO)):]
y_validate = Y[int(DATASET_SIZE*(TRAIN_RATIO+TEST_RATIO)):]
```

Normalize the data

In [7]:
```
normalizer = Normalization()
normalizer.adapt(x_train)
```

Creating a model

In [8]:
```
model = tf.keras.Sequential(
    [
        InputLayer(shape=(8,)),
        normalizer,
        Dense (128, activation='relu') ,
        Dense (128, activation = 'relu') ,
        Dense (128, activation = 'relu') ,
        Dense (1) ,
    ]
)
model. summary ()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 8)	17
dense (Dense)	(None, 128)	1,152
dense_1 (Dense)	(None, 128)	16,512
dense_2 (Dense)	(None, 128)	16,512
dense_3 (Dense)	(None, 1)	129

Total params: 34,322 (134.07 KB)

Trainable params: 34,305 (134.00 KB)

Non-trainable params: 17 (72.00 B)

Compiling and fitting the model

In [9]:
```
model.compile(
    optimizer= Adam(learning_rate=0.1),
    loss = MeanAbsoluteError(),
    metrics= [RootMeanSquaredError()]
)
```

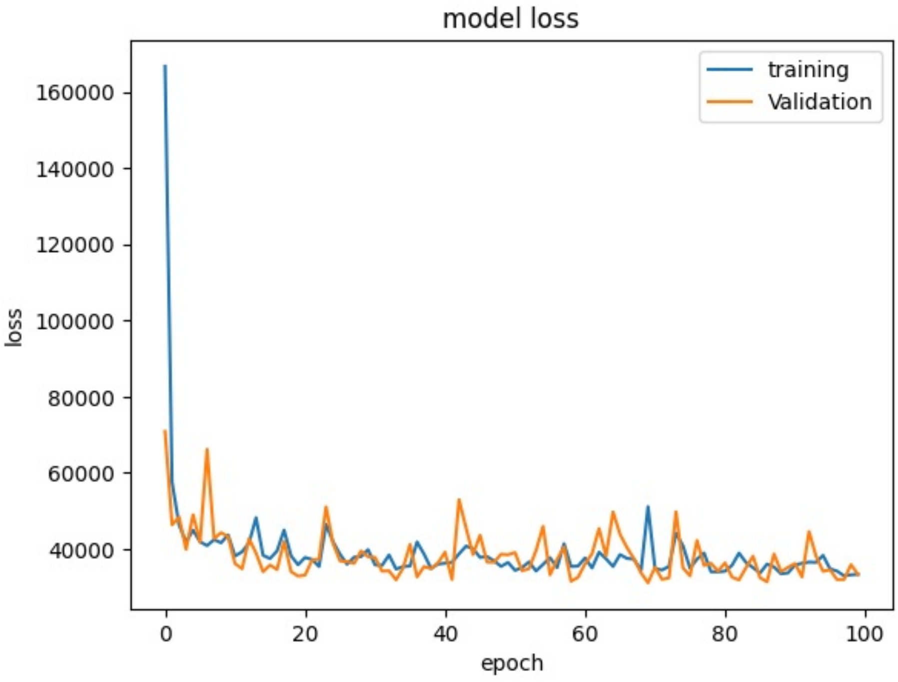
In [10]:
```
history = model.fit(x_train,y_train,validation_data=(x_validate,y_validate),epochs=100,verbose=1)
```

```
Epoch 1/100

 1/25 ━━━━━━━━━━━━━━━━━━━━ 13s 571ms/step - loss: 322586.2188 - root_mean_squared_error: 338913.2188

 5/25 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 332477.3125 - root_mean_squared_error: 351293.2188

10/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 311415.8438 - root_mean_squared_error: 333668.0000

15/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 285040.1562 - root_mean_squared_error: 312035.6875

21/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 258718.4375 - root_mean_squared_error: 289946.8125

25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 241827.3125 - root_mean_squared_error: 275261.2500 - val_loss: 7
0898.0781 - val_root_mean_squared_error: 95216.2422
Epoch 2/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 69136.1406 - root_mean_squared_error: 91435.4141

 6/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 65426.3516 - root_mean_squared_error: 81465.0156

12/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 64000.1562 - root_mean_squared_error: 79450.9531

18/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 62415.8789 - root_mean_squared_error: 77701.7031

24/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 61238.1055 - root_mean_squared_error: 76329.9922

25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 60952.6602 - root_mean_squared_error: 75977.7266 - val_loss: 463
36.7539 - val_root_mean_squared_error: 60648.1914
Epoch 3/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 42258.9141 - root_mean_squared_error: 57050.6680

 7/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 44746.5781 - root_mean_squared_error: 57257.1680

12/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 43810.4531 - root_mean_squared_error: 55564.1289

18/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 43718.4414 - root_mean_squared_error: 55362.1602

24/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 43931.5664 - root_mean_squared_error: 55491.7383

25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 44116.7031 - root_mean_squared_error: 55672.7656 - val_loss: 484
08.5547 - val_root_mean_squared_error: 57142.0938
```

```
Epoch 99/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 31547.9238 - root_mean_squared_error: 38936.7734
 7/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 33135.2227 - root_mean_squared_error: 41485.8242
13/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 33699.7148 - root_mean_squared_error: 41985.9961
19/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 33503.9453 - root_mean_squared_error: 41787.1055
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 33387.6680 - root_mean_squared_error: 41634.7070
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 33381.7930 - root_mean_squared_error: 41622.4492 - val_loss: 358
87.7031 - val_root_mean_squared_error: 47676.9141
Epoch 100/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 36376.9141 - root_mean_squared_error: 44244.2969
 7/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 35325.3398 - root_mean_squared_error: 42602.6133
13/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 34735.6406 - root_mean_squared_error: 42111.9258
19/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 34448.3906 - root_mean_squared_error: 41937.6445
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 34249.0117 - root_mean_squared_error: 41892.3320
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 34215.7188 - root_mean_squared_error: 41880.6367 - val_loss: 333
21.3203 - val_root_mean_squared_error: 45155.1914
```

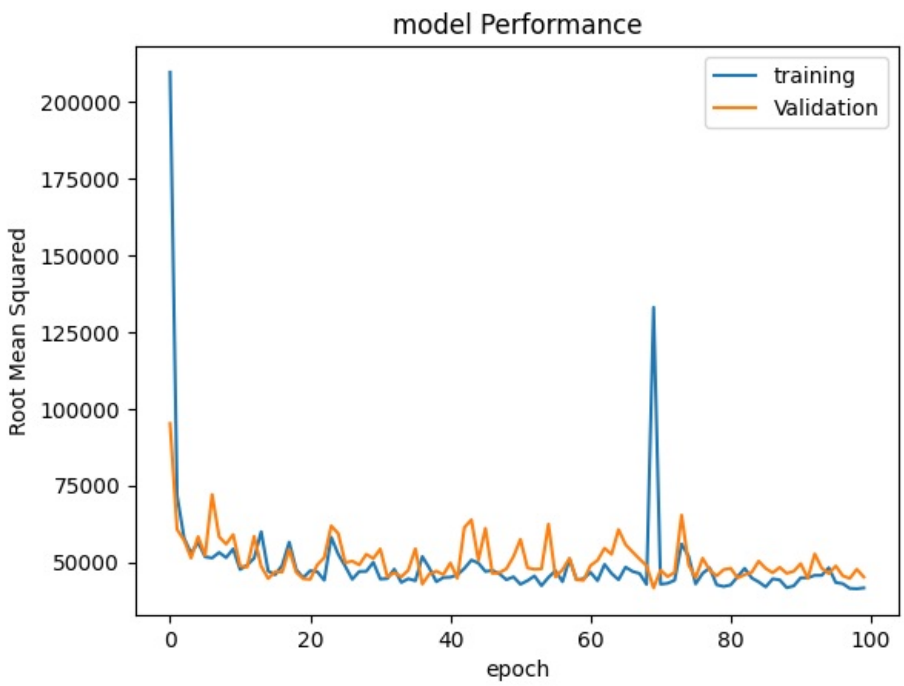
Visualiztion of `loss` values during `training` and `validation`

In [11]:
```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['training','Validation'])
plt.show()
```



Visualiztion of `RMS` during `training` and `validation`

In [12]:
```python
plt.plot(history.history['root_mean_squared_error'])
plt.plot(history.history['val_root_mean_squared_error'])
plt.title('model Performance')
plt.ylabel('Root Mean Squared')
plt.xlabel('epoch')
plt.legend(['training','Validation'])
plt.show()
```



Evaluation and prediction

In [13]:
```
model.evaluate(x_test,y_test)
```

```
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 29214.6953 - root_mean_squared_error: 37767.1289

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 34239.4102 - root_mean_squared_error: 43914.4258
```

Out[13]:
```
[34738.046875, 45548.31640625]
```

In [14]:
```
model.predict(tf.expand_dims(x_test[0], axis = 0 ))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```

Out[14]:
```
array([[223938.44]], dtype=float32)
```

In [15]:
```
y_test[0]
```

Out[15]:
```
<tf.Tensor: shape=(1,), dtype=float64, numpy=array([232192.])>
```

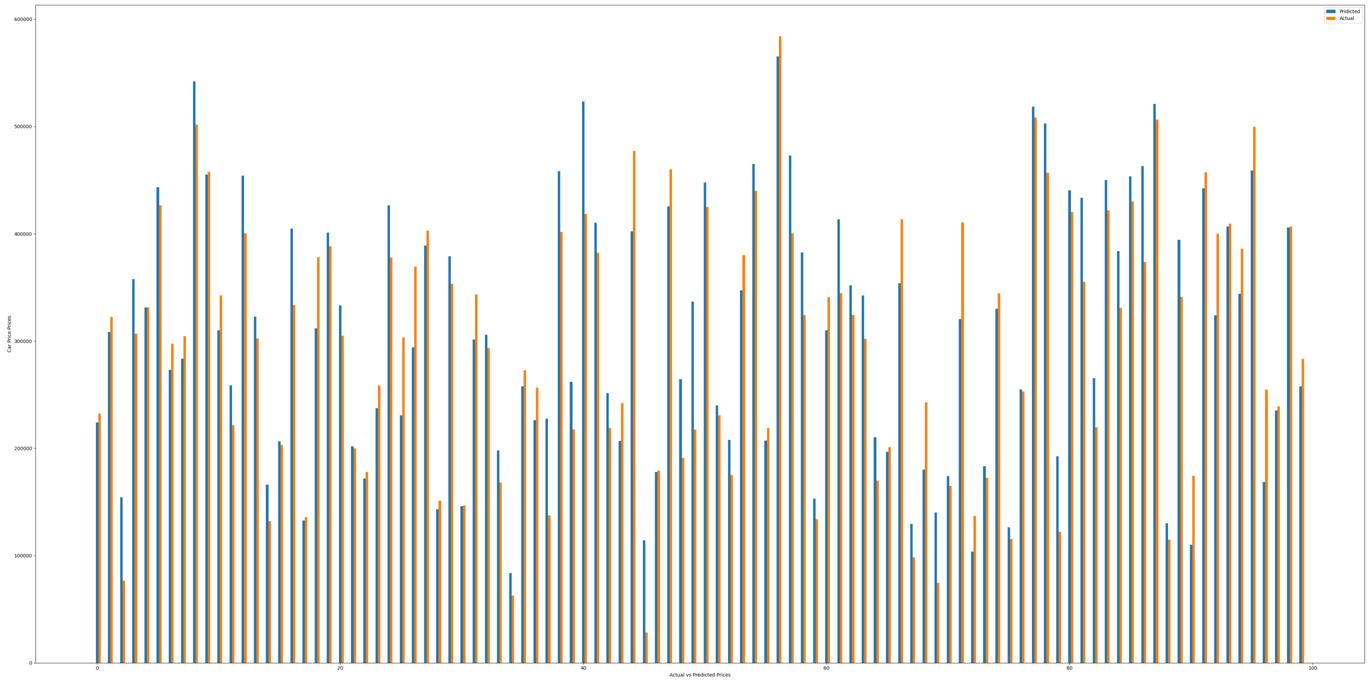
In [16]:
```
y_true = list(y_test[:,0] .numpy())
```

In [17]:
```
y_pred = list(model. predict(x_test) [:,0])
```

```
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

Visualization of actual values vs predicted values

In [18]:
```
ind = np.arange (100)
plt. figure(figsize=(40,20))
width = 0.2
plt.bar(ind, y_pred, width, label='Predicted Car Price')
plt.bar(ind + width, y_true, width, label='Actual Car Price')
plt.xlabel('Actual vs Predicted Prices')
plt.ylabel( 'Car Price Prices')
plt.legend(['Pridicted','Actual'])
plt.tight_layout()
plt.show()
```


Predicted
Actual
Car Price Prices
Actual vs Predicted Prices
0
100000
200000
300000
400000
500000
600000
0
20
40
60
80
100

In [ ]: