Importing Libraries

```
import tensorflow as tf #models
import seaborn as sns #visuals
from tensorflow.keras.layers import Normalization, Dense, InputLayer
import pandas as pd
from tensorflow.keras.losses import MeanSquaredError, BinaryCrossentropy, Hu
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import RootMeanSquaredError
import matplotlib.pyplot as plt #visuals
import numpy as np
```

DATA PREPARATION

Importing Dataset

```
data  = pd.read_csv('train.csv')
data.head()
```

	v.id	on road old	on road now	years	km	rating	condition	economy	top speed	hp	torque
0	1	535651	798186	3	78945	1	2	14	177	73	123
1	2	591911	861056	6	117220	5	9	9	148	74	95
2	3	686990	770762	2	132538	2	8	15	181	53	97

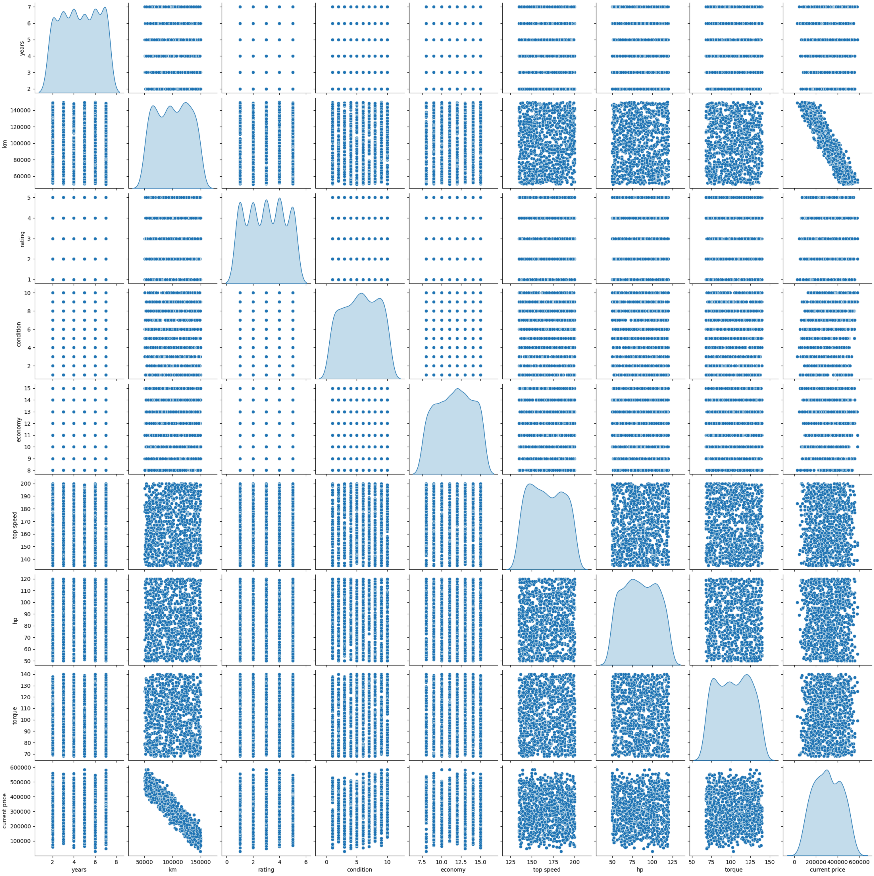
Next steps: Generate code with data | View recommended plots | New interactive sheet

from google.colab import sheets sheet = sheets.InteractiveSheet(df=data)MMMMMMMMMMMMMMMMMMMMMMMMMMMMMG

Visualising dataset

```
sns.pairplot(data[['years', 'km', 'rating', 'condition', 'economy', 'top spe
```

```
<seaborn.axisgrid.PairGrid at 0x783cd39d9b70>
```



Converting pd.DataFrame to Tensor

```
tensorData = tf.constant(data)
tensorData = tf.cast(tensorData, tf.float64)
print(tensorData[:5])
```

```
tf.Tensor(
[[1.000000e+00 5.356510e+05 7.981860e+05 3.000000e+00 7.894500e+04
  1.000000e+00 2.000000e+00 1.400000e+01 1.770000e+02 7.300000e+01
  1.230000e+02 3.513180e+05]
 [2.000000e+00 5.919110e+05 8.610560e+05 6.000000e+00 1.172200e+05
  5.000000e+00 9.000000e+00 9.000000e+00 1.480000e+02 7.400000e+01
  9.500000e+01 2.850015e+05]
 [3.000000e+00 6.869900e+05 7.707620e+05 2.000000e+00 1.325380e+05
  2.000000e+00 8.000000e+00 1.500000e+01 1.810000e+02 5.300000e+01
  9.700000e+01 2.153860e+05]
 [4.000000e+00 5.739990e+05 7.223810e+05 4.000000e+00 1.010650e+05
  4.000000e+00 3.000000e+00 1.100000e+01 1.970000e+02 5.400000e+01
  1.160000e+02 2.442955e+05]
 [5.000000e+00 6.913880e+05 8.113350e+05 6.000000e+00 6.155900e+04
  3.000000e+00 9.000000e+00 1.200000e+01 1.600000e+02 5.300000e+01
  1.050000e+02 5.311145e+05]], shape=(5, 12), dtype=float64)
```

Shuffling the order of dataset

```
tensorData = tf.random.shuffle(tensorData)
print(tensorData[:5])
```

```
tf.Tensor(
[[9.800000e+02 6.336660e+05 8.060520e+05 7.000000e+00 1.151760e+05
  5.000000e+00 2.000000e+00 1.100000e+01 1.520000e+02 7.100000e+01
  9.500000e+01 2.525495e+05]
 [6.000000e+00 6.500070e+05 8.448460e+05 6.000000e+00 1.488460e+05
  2.000000e+00 9.000000e+00 1.300000e+01 1.380000e+02 6.100000e+01
  1.090000e+02 1.779335e+05]
 [7.580000e+02 6.585050e+05 8.153720e+05 3.000000e+00 9.668300e+04
  3.000000e+00 9.000000e+00 1.300000e+01 1.510000e+02 9.900000e+01
  1.010000e+02 3.799910e+05]
 [5.570000e+02 5.534900e+05 8.944150e+05 3.000000e+00 5.633100e+04
  3.000000e+00 9.000000e+00 1.300000e+01 1.830000e+02 1.070000e+02
  7.200000e+01 5.284185e+05]
 [6.580000e+02 5.880350e+05 7.550160e+05 3.000000e+00 1.481670e+05
  1.000000e+00 6.000000e+00 1.000000e+01 1.980000e+02 7.400000e+01
  7.500000e+01 8.284800e+04]], shape=(5, 12), dtype=float64)
```

Splitting Dataset into Features and Labels

```
X = tensorData[:, 3:-1]
print(X[:5])
```

```
tf.Tensor(
[[7.00000e+00 1.15176e+05 5.00000e+00 2.00000e+00 1.10000e+01 1.52000e+02
  7.10000e+01 9.50000e+01]
 [6.00000e+00 1.48846e+05 2.00000e+00 9.00000e+00 1.30000e+01 1.38000e+02
  6.10000e+01 1.09000e+02]
 [3.00000e+00 9.66830e+04 3.00000e+00 9.00000e+00 1.30000e+01 1.51000e+02
  9.90000e+01 1.01000e+02]
 [3.00000e+00 5.63310e+04 3.00000e+00 9.00000e+00 1.30000e+01 1.83000e+02
  1.07000e+02 7.20000e+01]
 [3.00000e+00 1.48167e+05 1.00000e+00 6.00000e+00 1.00000e+01 1.98000e+02
  7.40000e+01 7.50000e+01]], shape=(5, 8), dtype=float64)
```

```
y = tensorData[:, -1]
print(y.shape)
y = tf.expand_dims(y, axis=-1)
print(y.shape)
```

```
(1000,)
(1000, 1)
```

Splitting Dataset into Train Dataset, Validation Dataset, Test Dataset

```
TRAIN_RATIO = 0.8
VAL_RATIO = 0.1
TEST_RATIO = 0.1
DATASET_SIZE = len(X)
```

```
X_train = X[:int(TRAIN_RATIO * DATASET_SIZE)]
y_train = y[:int(TRAIN_RATIO * DATASET_SIZE)]
print(X_train.shape, y_train.shape)
```

```
(800, 8) (800, 1)
```

```
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=len(X_train),reshuffle_eac
train_dataset.element_spec
```

```
(TensorSpec(shape=(None, 8), dtype=tf.float64, name=None),
 TensorSpec(shape=(None, 1), dtype=tf.float64, name=None))
```

```
X_val = X[int(DATASET_SIZE * TRAIN_RATIO):int(DATASET_SIZE * (TRAIN_RATIO +
y_val = y[int(DATASET_SIZE * TRAIN_RATIO):int(DATASET_SIZE * (TRAIN_RATIO +
print(X_val.shape, y_val.shape)
```

```
(100, 8) (100, 1)
```

```
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))
val_dataset = val_dataset.shuffle(buffer_size=8, reshuffle_each_iteration=Tr
val_dataset.element_spec
```

```
(TensorSpec(shape=(None, 8), dtype=tf.float64, name=None),
 TensorSpec(shape=(None, 1), dtype=tf.float64, name=None))
```

```
X_test = X[int(DATASET_SIZE * (TRAIN_RATIO + VAL_RATIO)):]
y_test = y[int(DATASET_SIZE * (TRAIN_RATIO + VAL_RATIO)):]
print(X_test.shape, y_test.shape)
```

```
(100, 8) (100, 1)
```

```
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
test_dataset = test_dataset.shuffle(buffer_size=8, reshuffle_each_iteration=
test_dataset.element_spec
```

```
(TensorSpec(shape=(None, 8), dtype=tf.float64, name=None),
 TensorSpec(shape=(None, 1), dtype=tf.float64, name=None))
```

Normalizing Data

```
normalizer = Normalization()
normalizer.adapt(X)
X_normalized = normalizer(X)
```

MODEL CREATION

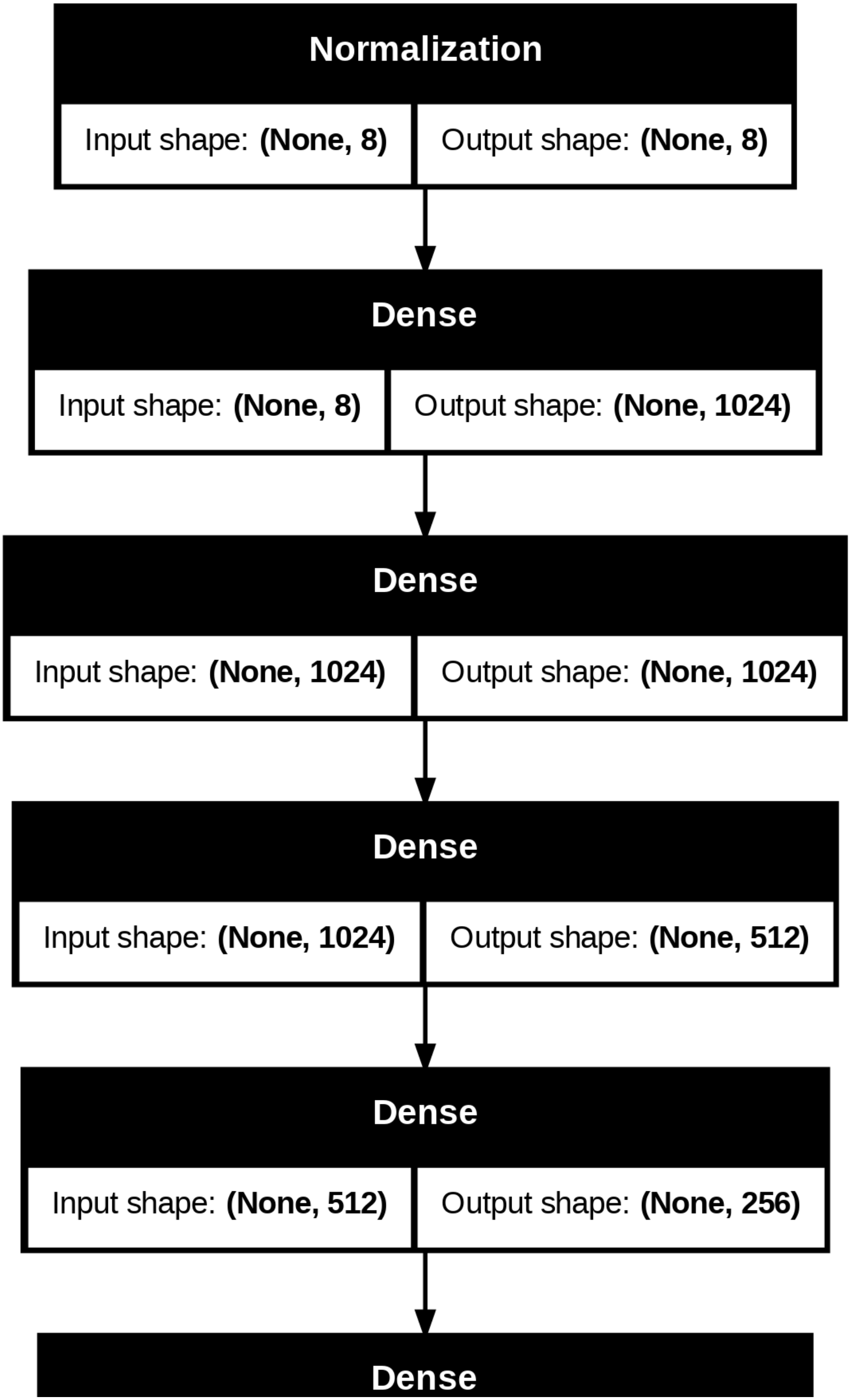
```
model = tf.keras.Sequential([
    InputLayer(input_shape =(8,)),
    normalizer,
    Dense(1024, activation='relu'),
    Dense(1024, activation='relu'),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(1, activation='linear')
])
model.summary()
```

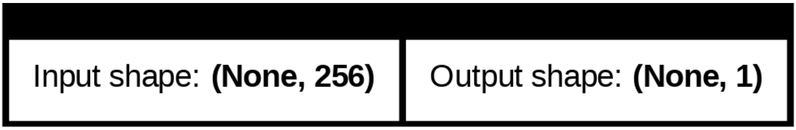
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/input_layer.py:26: UserW
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
normalization (Normalization)	(None, 8)	17
dense (Dense)	(None, 1024)	9,21
dense_1 (Dense)	(None, 1024)	1,049,60
dense_2 (Dense)	(None, 512)	524,80
dense_3 (Dense)	(None, 256)	131,32
dense_4 (Dense)	(None, 1)	25

```
tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=True)
```


Normalization
Input shape: (None, 8)
Output shape: (None, 8)
Dense
Input shape: (None, 8)
Output shape: (None, 1024)
Dense
Input shape: (None, 1024)
Output shape: (None, 1024)
Dense
Input shape: (None, 1024)
Output shape: (None, 512)
Dense
Input shape: (None, 512)
Output shape: (None, 256)
Dense



Compiling Model

```
model.compile(optimizer=Adam(),
              loss=MeanAbsoluteError(),
              metrics = [RootMeanSquaredError()])
```

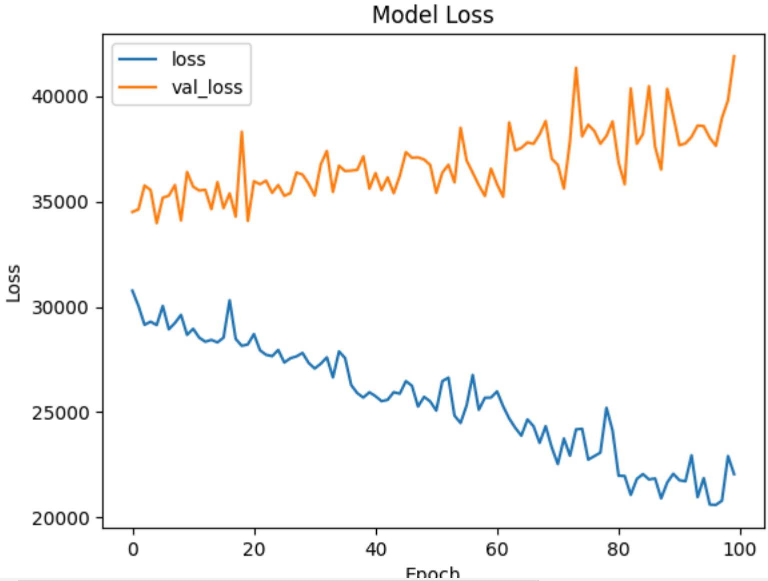
```
1 history = model.fit(train_dataset, validation_data=val_dataset, epochs=100,
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 27829.0527 - root_mean_squared_erro
Epoch 28/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 26643.8691 - root_mean_squared_erro
Epoch 29/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 28613.7578 - root_mean_squared_erro
Epoch 30/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 26513.7754 - root_mean_squared_erro
Epoch 31/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 26795.4102 - root_mean_squared_erro
Epoch 32/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 26993.3867 - root_mean_squared_erro
Epoch 33/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 28345.8477 - root_mean_squared_erro
Epoch 34/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 26034.2090 - root_mean_squared_erro
Epoch 35/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 27248.2930 - root_mean_squared_erro
Epoch 36/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 28574.3711 - root_mean_squared_erro
Epoch 37/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 26055.3047 - root_mean_squared_erro
Epoch 38/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 26702.4336 - root_mean_squared_erro
Epoch 39/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 24706.5703 - root_mean_squared_erro
Epoch 40/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 25718.0020 - root_mean_squared_erro
Epoch 41/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 25895.6250 - root_mean_squared_erro
Epoch 42/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - loss: 24564.2812 - root_mean_squared_erro
Epoch 43/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 24295.1602 - root_mean_squared_erro
Epoch 44/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - loss: 25121.5566 - root_mean_squared_erro
Epoch 45/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - loss: 24974.7188 - root_mean_squared_erro
Epoch 46/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 27825.6426 - root_mean_squared_erro
Epoch 47/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 26589.1094 - root_mean_squared_erro
Epoch 48/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 24495.5938 - root_mean_squared_erro
Epoch 49/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 26002.7344 - root_mean_squared_erro
Epoch 50/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 23549.6016 - root_mean_squared_erro
Epoch 51/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 24980.4980 - root_mean_squared_erro
Epoch 52/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 27307.6055 - root_mean_squared_erro
Epoch 53/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 26422.3125 - root_mean_squared_erro
Epoch 54/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - loss: 24914.9473 - root_mean_squared_erro
Epoch 55/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 23820.3906 - root_mean_squared_erro
```

```
#history.history
```

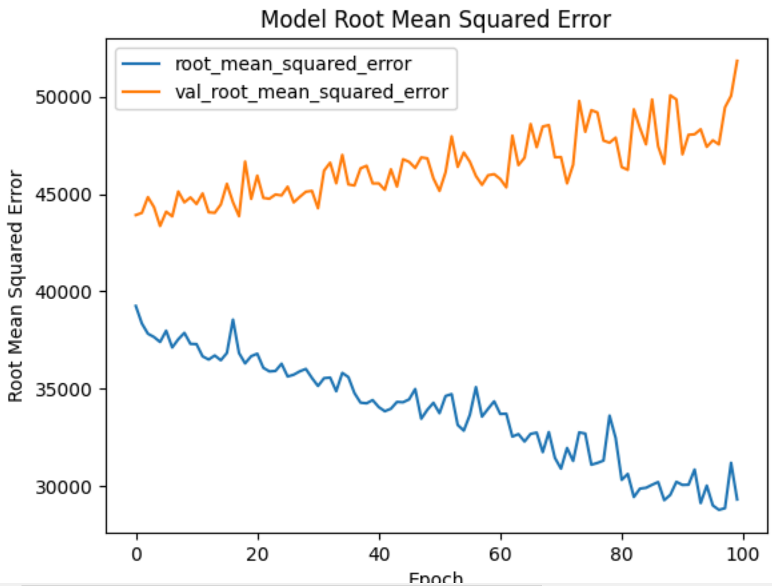
Plotting loss and Val_loss

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



Plotting Root Mean Squared Error and Val_Root_Mean_Squared_Error

```
plt.plot(history.history['root_mean_squared_error'])
plt.plot(history.history['val_root_mean_squared_error'])
plt.legend(['root_mean_squared_error', 'val_root_mean_squared_error'])
plt.title('Model Root Mean Squared Error')
plt.ylabel('Root Mean Squared Error')
plt.xlabel('Epoch')
plt.show()
```



Evaluating Model

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

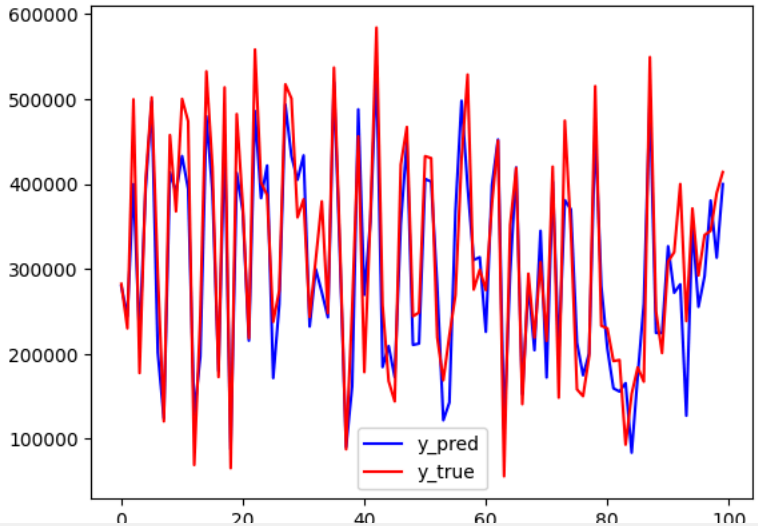
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 41266.8086 - root_mean_squared_error: 516
[41904.8359375, 51674.4609375]
```

PREDICTION

```
y_pred = list(model.predict(X_test)[:,0])
y_true = list(y_test[:,0].numpy())
print(y_pred, '\n', y_true)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
[279772.94, 244537.52, 399883.25, 217920.66, 393987.6, 497010.8, 201745.88, 124099.26,
 [282418.5, 230090.5, 499647.0, 177509.5, 403886.5, 501920.5, 295021.0, 120583.5, 4575
```

```
plt.plot(y_pred,color='b')
plt.plot(y_true,color='r')
plt.legend(['y_pred', 'y_true'])
plt.show()
```



Checking accuracy of model

```
y_true = np.array(y_true)
y_pred = np.array(y_pred)

mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
print(f'Mean Absolute Percentage Error (MAPE): {mape}%')
```

```
Mean Absolute Percentage Error (MAPE): 16.632037151871014%
```

```
model.save('CarPricePrediction.keras')
```

Main

```
def main():
    # Load the trained model
    with tf.keras.utils.CustomObjectScope({'MyCustomMetric': MyCustomMetric
        model = tf.keras.models.load_model('CarPricePrediction.keras')

    # Prompt user for input
    years = float(input("Enter years: "))
    km = float(input("Enter kilometers: "))
    rating = float(input("Enter rating: "))
    condition = float(input("Enter condition: "))
    economy = float(input("Enter economy: "))
```

```
    top_speed = float(input("Enter top speed: "))
    hp = float(input("Enter horsepower: "))
    torque = float(input("Enter torque: "))

    # Create a TensorFlow constant from user input
    test_input = tf.constant([[years, km, rating, condition, economy, top_speed, hp, torque]], dtype=tf.float64)

    # Predict the value
    prediction = model.predict(test_input)

    # Calculate the margin of error based on MAPE
    margin_of_error = prediction[0][0] * (17.56103471303923 / 100)

    # Print the prediction with the margin of error
    print("Predicted Value: {:.2f} ± {:.2f}".format(prediction[0][0], margin_of_error))
```