ne-age-prediction-sequential-model

February 18, 2024

1 Abalone Age Prediction Sequential Model TensorFlow

Generally, the age of an Abalone is determined by the physical examination of the abalone but this is a tedious task which is why we will try to build a regressor that can predict the age of abalone using some features which are easy to determine. ## Importing Libraries

[64]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split


import tensorflow as tf
from tensorflow import keras
from keras import layers

import warnings
warnings.filterwarnings('ignore')
```

1.0.1 Loading Dataset

[65]:
```
data = pd.read_csv("drive/MyDrive/abalone.csv")
data.head()
```

[65]:
```
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
```

```
3          0.155     10
4          0.055      7
```

[66]:
```python
# Shape
data.shape
```

[66]:
```
(4177, 9)
```

[67]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

[68]:
```python
data.describe().T
```

[68]:
```
                 count      mean       std     min     25%     50%     75%  \
Length          4177.0  0.523992  0.120093  0.0750  0.4500  0.5450   0.615
Diameter        4177.0  0.407881  0.099240  0.0550  0.3500  0.4250   0.480
Height          4177.0  0.139516  0.041827  0.0000  0.1150  0.1400   0.165
Whole weight    4177.0  0.828742  0.490389  0.0020  0.4415  0.7995   1.153
Shucked weight  4177.0  0.359367  0.221963  0.0010  0.1860  0.3360   0.502
Viscera weight  4177.0  0.180594  0.109614  0.0005  0.0935  0.1710   0.253
Shell weight    4177.0  0.238831  0.139203  0.0015  0.1300  0.2340   0.329
Rings           4177.0  9.933684  3.224169  1.0000  8.0000  9.0000  11.000

                   max
Length          0.8150
Diameter        0.6500
Height          1.1300
Whole weight    2.8255
Shucked weight  1.4880
Viscera weight  0.7600
Shell weight    1.0050
```

```
Rings           29.0000
```

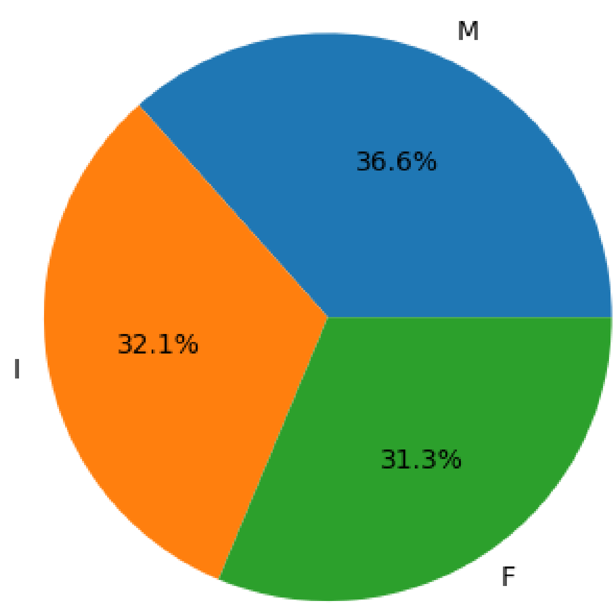
1.1 Exploratory Data Analysis

[69]:
```
data.isnull().sum()
```

[69]:
```
Sex                 0
Length              0
Diameter            0
Height              0
Whole weight        0
Shucked weight      0
Viscera weight      0
Shell weight        0
Rings               0
dtype: int64
```

Distribution of the data in male, female and infant

[70]:
```
x = data['Sex'].value_counts()
labels = x.index
values = x.values
plt.pie(values, labels = labels, autopct = '%1.1f%%')
plt.show()
```



by the look of the above the pie chat shows that we have equal amount of data for male, female, and infant abalone

[71]:
```
data.groupby('Sex').mean()
```

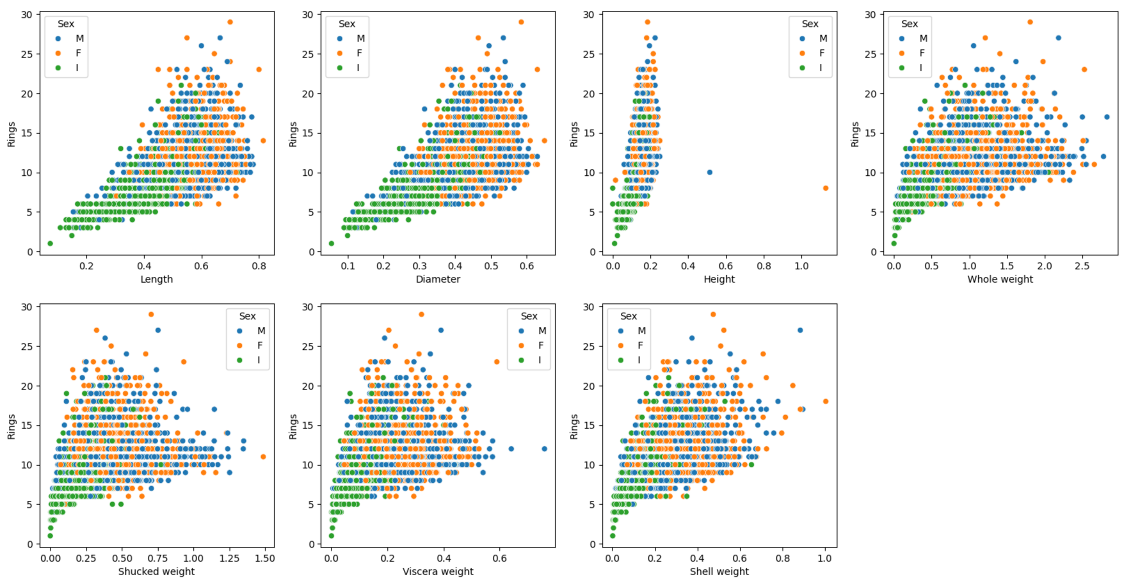
[71]:
```
       Length  Diameter    Height  Whole weight  Shucked weight  \
Sex
F    0.579093  0.454732  0.158011      1.046532        0.446188
I    0.427746  0.326494  0.107996      0.431363        0.191035
M    0.561391  0.439287  0.151381      0.991459        0.432946

     Viscera weight  Shell weight      Rings
Sex
F          0.230689      0.302010  11.129304
I          0.092010      0.128182   7.890462
M          0.215545      0.281969  10.705497
```

Here is an interesting observation that the life expectancy of the female abalone is higher than that of the male abalone. In the other features as well we can see that the height weight, as well as length in all the attributes of the numbers for female abalones, is on the higher sides.

[72]:
```
features = data.loc[:, 'Length':'Shell weight'].columns
plt.subplots(figsize=(20,10))
for i, feat in enumerate(features):
  plt.subplot(2, 4, i+1)
  sns.scatterplot(data = data, x = feat, y='Rings', hue = 'Sex')

plt.show()
```



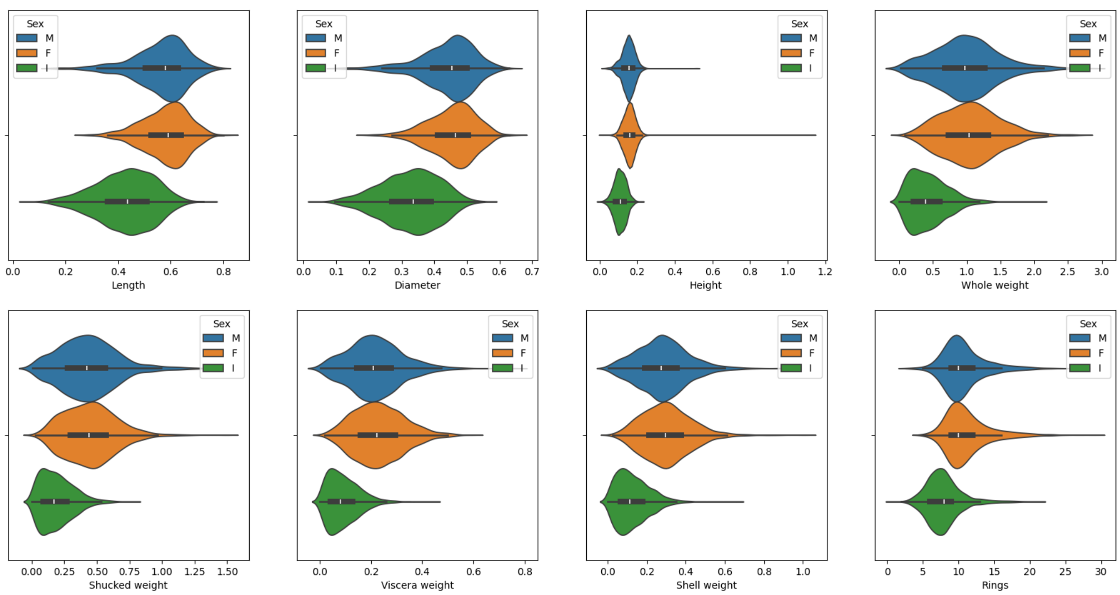
Observations from the above graph are as follows:

- A strong linear correlation between the age of the abalone and its height can be observed from the above graphs.
- Length and Diameter have the same kind of relation with age that is up to a certain age length increases and after that it became constant.

A similar kind of relationship is present between the weight and the age feature.

[73]:
```
plt.subplots(figsize=(20, 10))
for i, feat in enumerate(features):
  plt.subplot(2, 4, i+1)
  sns.violinplot(data = data, x = feat , hue ='Sex')

plt.subplot(2, 4, 8)
sns.violinplot(data = data, x = 'Rings', hue = 'Sex')
plt.show()
```



[74]:
```
data.head()
```

[74]:
```
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395
```

```
   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7
```

Perform a One Hot Encoding on the Sex column.

[75]:
```
data = pd.get_dummies(data, columns=['Sex'])
data.head()
```

[75]:
```
   Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
0   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell weight  Rings  Sex_F  Sex_I  Sex_M
0         0.150     15      0      0      1
1         0.070      7      0      0      1
2         0.210      9      1      0      0
3         0.155     10      0      0      1
4         0.055      7      0      1      0
```

Now I will separate the Features and target variables and split them into training and validation data.

[76]:
```
features = data.drop('Rings', axis = 1)
target = data['Rings']
t='BY JosephWathome'
X_train,X_val, Y_train, Y_val = train_test_split(features, target, test_size =␣
 ↪0.2,
                                                  random_state = 22)

X_train.shape, X_val.shape
```

[76]:
```
((3341, 10), (836, 10))
```

[77]:
```
X_train.head()
```

[77]:
```
      Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  \
3733   0.605     0.455   0.160        1.1215          0.5330          0.2730
3505   0.625     0.495   0.180        1.0815          0.4715          0.2540
3314   0.450     0.355   0.115        0.4385          0.1840          0.1080
1888   0.565     0.445   0.125        0.8305          0.3135          0.1785
3484   0.475     0.420   0.160        0.7095          0.3500          0.1505
```

```
      Shell weight  Sex_F  Sex_I  Sex_M
3733        0.2710      0      0      1
3505        0.3135      0      0      1
3314        0.1125      0      1      0
1888        0.2300      1      0      0
3484        0.1845      0      1      0
```

1.2 Model Architecture

I will impliment the **Sequential Model** that will contain the following parts: - I will have Tow Connected Layers - I have included some **BatchNormalization** layers to enable stable and fast training and a **Dropout** layer before the final layer to avoid any possibility of overfitting.

[78]:
```
model = keras.Sequential([layers.Dense(256, activation = 'relu', input_shape =␣
 ↪[10]),
                          layers.BatchNormalization(),
                          layers.Dense(256, activation='relu'),
                          layers.Dropout(0.3),
                          layers.BatchNormalization(),
                          layers.Dense(1, activation='relu')])

model.compile(loss = 'mae',
              optimizer = 'adam',
              metrics = ['mape'])
```

[79]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 256)               2816

 batch_normalization_8 (Bat  (None, 256)               1024
 chNormalization)

 dense_13 (Dense)            (None, 256)               65792

 dropout_4 (Dropout)         (None, 256)               0

 batch_normalization_9 (Bat  (None, 256)               1024
 chNormalization)

 dense_14 (Dense)            (None, 1)                 257

=================================================================
```

```
Total params: 70913 (277.00 KB)
Trainable params: 69889 (273.00 KB)
Non-trainable params: 1024 (4.00 KB)
_________________________________________________________________
```

converting the data to foat32 for the model

[80]:
```
X_train = X_train.astype('float32')
Y_train = Y_train.astype('float32')
X_val = X_val.astype('float32')
Y_val = Y_val.astype('float32')
```

1.3 Model Training

[81]:
```
history = model.fit(X_train, Y_train,
                    epochs=50,
                    verbose=1,
                    batch_size=64,
                    validation_data=(X_val, Y_val))
```

```
Epoch 1/50
53/53 [==============================] - 2s 17ms/step - loss: 5.2891 - mape:
57.2654 - val_loss: 7.8946 - val_mape: 79.3217
Epoch 2/50
53/53 [==============================] - 0s 7ms/step - loss: 4.1686 - mape:
47.0963 - val_loss: 7.1025 - val_mape: 71.5404
Epoch 3/50
53/53 [==============================] - 0s 6ms/step - loss: 4.0295 - mape:
45.9125 - val_loss: 6.1870 - val_mape: 62.5931
Epoch 4/50
53/53 [==============================] - 0s 7ms/step - loss: 3.9076 - mape:
44.6869 - val_loss: 5.1261 - val_mape: 51.8713
Epoch 5/50
53/53 [==============================] - 0s 6ms/step - loss: 3.3706 - mape:
38.6901 - val_loss: 1.8799 - val_mape: 16.3903
Epoch 6/50
53/53 [==============================] - 0s 7ms/step - loss: 2.7900 - mape:
31.8521 - val_loss: 1.8729 - val_mape: 19.3113
Epoch 7/50
53/53 [==============================] - 0s 6ms/step - loss: 2.4032 - mape:
26.8200 - val_loss: 2.1171 - val_mape: 23.3601
Epoch 8/50
53/53 [==============================] - 0s 7ms/step - loss: 2.2544 - mape:
25.0822 - val_loss: 2.0218 - val_mape: 22.1881
Epoch 9/50
53/53 [==============================] - 0s 6ms/step - loss: 2.2369 - mape:
24.8090 - val_loss: 1.8889 - val_mape: 20.3759
Epoch 10/50
```

```
53/53 [==============================] - 0s 6ms/step - loss: 2.1362 - mape:
23.4074 - val_loss: 1.8999 - val_mape: 20.6195
Epoch 11/50
53/53 [==============================] - 0s 7ms/step - loss: 2.0768 - mape:
22.4461 - val_loss: 1.9617 - val_mape: 21.9786
Epoch 12/50
53/53 [==============================] - 1s 10ms/step - loss: 1.9735 - mape:
21.5916 - val_loss: 1.6727 - val_mape: 17.3723
Epoch 13/50
53/53 [==============================] - 1s 10ms/step - loss: 1.9031 - mape:
20.6304 - val_loss: 1.8582 - val_mape: 20.4082
Epoch 14/50
53/53 [==============================] - 1s 10ms/step - loss: 1.8717 - mape:
20.0603 - val_loss: 1.6555 - val_mape: 17.2705
Epoch 15/50
53/53 [==============================] - 1s 10ms/step - loss: 1.7624 - mape:
18.3546 - val_loss: 1.5901 - val_mape: 15.9443
Epoch 16/50
53/53 [==============================] - 0s 9ms/step - loss: 1.7712 - mape:
18.3677 - val_loss: 1.6946 - val_mape: 17.7099
Epoch 17/50
53/53 [==============================] - 1s 10ms/step - loss: 1.6906 - mape:
16.9950 - val_loss: 1.5600 - val_mape: 14.7417
Epoch 18/50
53/53 [==============================] - 1s 10ms/step - loss: 1.6550 - mape:
16.5222 - val_loss: 1.6298 - val_mape: 15.6944
Epoch 19/50
53/53 [==============================] - 1s 17ms/step - loss: 1.6752 - mape:
16.9274 - val_loss: 1.5083 - val_mape: 14.3708
Epoch 20/50
53/53 [==============================] - 1s 16ms/step - loss: 1.6268 - mape:
16.3398 - val_loss: 1.7552 - val_mape: 16.7409
Epoch 21/50
53/53 [==============================] - 1s 10ms/step - loss: 1.6223 - mape:
16.1914 - val_loss: 1.5222 - val_mape: 14.8033
Epoch 22/50
53/53 [==============================] - 0s 8ms/step - loss: 1.5838 - mape:
15.9068 - val_loss: 1.5468 - val_mape: 15.1606
Epoch 23/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5931 - mape:
15.7406 - val_loss: 1.6474 - val_mape: 16.2875
Epoch 24/50
53/53 [==============================] - 0s 7ms/step - loss: 1.6036 - mape:
15.8766 - val_loss: 1.6406 - val_mape: 16.1085
Epoch 25/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5829 - mape:
15.4957 - val_loss: 1.4986 - val_mape: 14.3542
Epoch 26/50
```

```
53/53 [==============================] - 0s 7ms/step - loss: 1.5777 - mape:
15.5899 - val_loss: 1.5088 - val_mape: 14.2424
Epoch 27/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5696 - mape:
15.3485 - val_loss: 1.4703 - val_mape: 13.9026
Epoch 28/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5581 - mape:
15.3101 - val_loss: 1.4816 - val_mape: 13.9312
Epoch 29/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5493 - mape:
15.1414 - val_loss: 1.4758 - val_mape: 13.9861
Epoch 30/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5502 - mape:
15.0839 - val_loss: 1.5056 - val_mape: 14.3565
Epoch 31/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5452 - mape:
15.1466 - val_loss: 1.4956 - val_mape: 14.4843
Epoch 32/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5194 - mape:
14.9069 - val_loss: 1.5074 - val_mape: 13.8365
Epoch 33/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5433 - mape:
14.9819 - val_loss: 1.4817 - val_mape: 14.1558
Epoch 34/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5148 - mape:
14.9192 - val_loss: 1.4864 - val_mape: 14.3607
Epoch 35/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5150 - mape:
14.7361 - val_loss: 1.4988 - val_mape: 14.0730
Epoch 36/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5045 - mape:
14.6200 - val_loss: 1.5537 - val_mape: 15.3094
Epoch 37/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5367 - mape:
15.1302 - val_loss: 1.5128 - val_mape: 14.3585
Epoch 38/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5210 - mape:
14.8798 - val_loss: 1.4943 - val_mape: 14.1917
Epoch 39/50
53/53 [==============================] - 0s 6ms/step - loss: 1.5033 - mape:
14.7799 - val_loss: 1.5056 - val_mape: 13.8984
Epoch 40/50
53/53 [==============================] - 0s 7ms/step - loss: 1.5133 - mape:
14.8882 - val_loss: 1.4816 - val_mape: 13.8616
Epoch 41/50
53/53 [==============================] - 0s 8ms/step - loss: 1.5069 - mape:
14.7457 - val_loss: 1.5079 - val_mape: 14.5612
Epoch 42/50
```

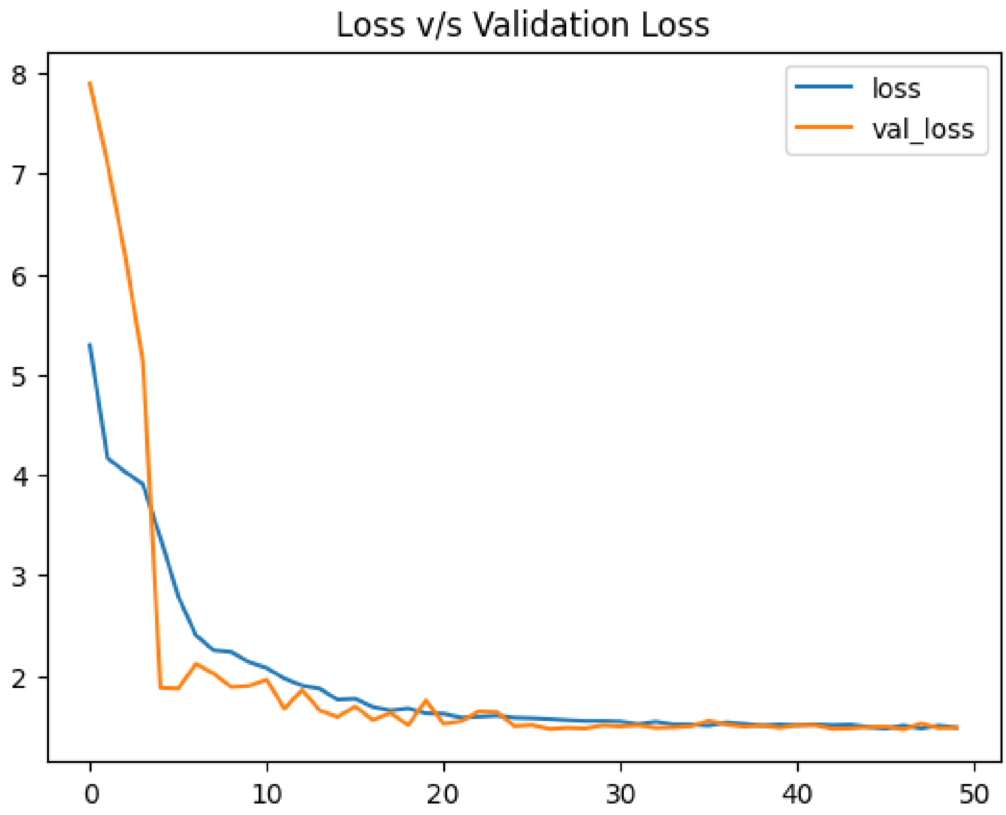
```
53/53 [==============================] - 0s 8ms/step - loss: 1.5146 - mape:
14.8047 - val_loss: 1.5062 - val_mape: 14.5307
Epoch 43/50
53/53 [==============================] - 1s 10ms/step - loss: 1.5073 - mape:
14.8130 - val_loss: 1.4719 - val_mape: 14.1408
Epoch 44/50
53/53 [==============================] - 1s 10ms/step - loss: 1.5124 - mape:
14.8445 - val_loss: 1.4771 - val_mape: 13.9903
Epoch 45/50
53/53 [==============================] - 1s 11ms/step - loss: 1.4892 - mape:
14.4589 - val_loss: 1.4877 - val_mape: 14.1112
Epoch 46/50
53/53 [==============================] - 1s 10ms/step - loss: 1.4748 - mape:
14.3367 - val_loss: 1.4938 - val_mape: 14.4995
Epoch 47/50
53/53 [==============================] - 1s 11ms/step - loss: 1.5032 - mape:
14.7512 - val_loss: 1.4663 - val_mape: 13.9024
Epoch 48/50
53/53 [==============================] - 1s 10ms/step - loss: 1.4766 - mape:
14.3811 - val_loss: 1.5241 - val_mape: 13.8843
Epoch 49/50
53/53 [==============================] - 0s 9ms/step - loss: 1.5035 - mape:
14.7135 - val_loss: 1.4785 - val_mape: 14.2238
Epoch 50/50
53/53 [==============================] - 0s 6ms/step - loss: 1.4829 - mape:
14.3986 - val_loss: 1.4838 - val_mape: 14.3104
```

[82]:
```python
hist_df = pd.DataFrame(history.history)
hist_df.head()
```

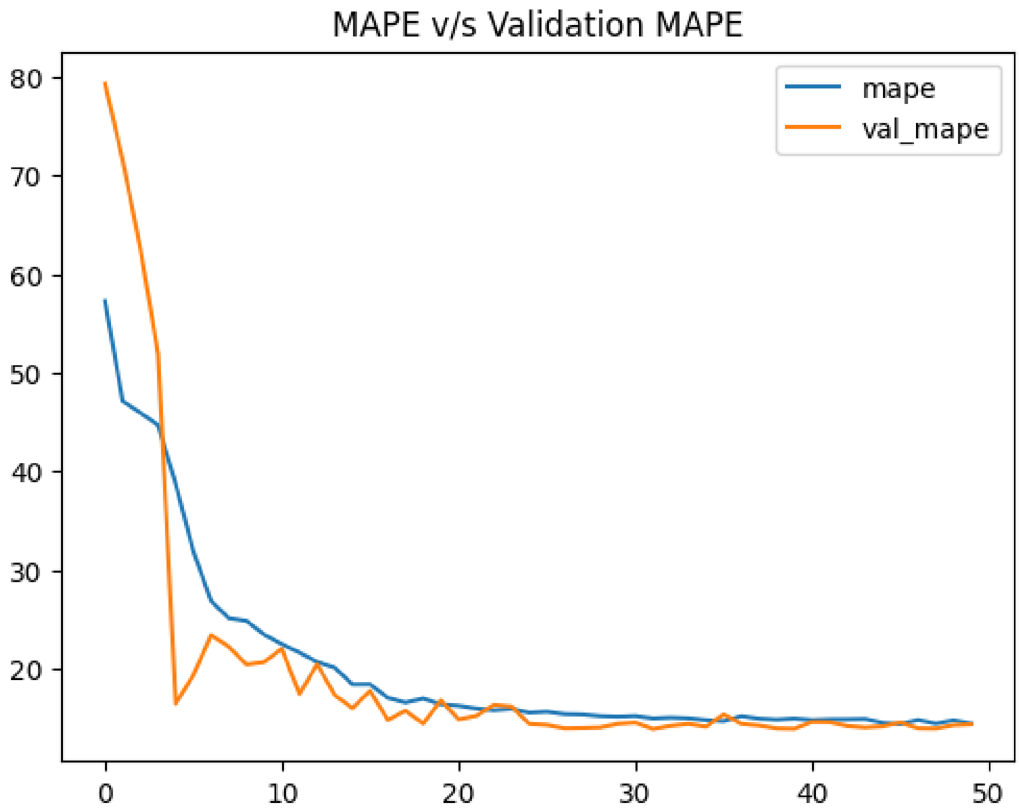
[82]:

	loss	mape	val_loss	val_mape
0	5.289099	57.265411	7.894604	79.321724
1	4.168559	47.096336	7.102536	71.540443
2	4.029542	45.912514	6.187005	62.593052
3	3.907557	44.686943	5.126121	51.871315
4	3.370569	38.690086	1.879891	16.390261

[83]:
```python
hist_df['loss'].plot()
hist_df['val_loss'].plot()
plt.title('Loss v/s Validation Loss')
plt.legend()
plt.show()
```



[84]:
```
hist_df['mape'].plot()
hist_df['val_mape'].plot()
plt.title('MAPE v/s Validation MAPE')
plt.legend()
plt.show()
```



From the above two graphs, we can certainly say that the two(mae and mape) error values have decreased simultaneously and continuously. Also, the saturation has been achieved after 15 epochs only.

By Joseph Wathome

[85]:
```
print(t)
```

```
BY JosephWathome
```