Part 1 : Machine Learning

Data Preprocessing

In [55]:
```
import numpy as np
import pandas as pd
raw_train  = pd.read_csv('train_FD001.csv')
raw_train
```

Out[55]:

	engine	cycle	setting 1	setting 2	setting 3	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	...	sensor 13	sensor 14	sensor 15	sensor 16	sensor 17	sensor 18	sensor 19	sensor 20	sensor 21	RUL
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190	191.0
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236	190.0
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442	189.0
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739	188.0
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044	187.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735	4.0
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594	3.0
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333	2.0
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640	1.0
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522	0.0

20631 rows × 27 columns

In [56]:
```
raw_test  = pd.read_csv('test_FD001.csv')
raw_test
```

Out[56]:

	engine	cycle	setting 1	setting 2	setting 3	sensor 1	sensor 2	sensor 3	sensor 4	sensor 5	...	sensor 13	sensor 14	sensor 15	sensor 16	sensor 17	sensor 18	sensor 19	sensor 20	sensor 21	RUL
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	2388.03	8125.55	8.4052	0.03	392	2388	100.0	38.86	23.3735	142.0
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	2388.06	8139.62	8.3803	0.03	393	2388	100.0	39.02	23.3916	141.0
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	2388.03	8130.10	8.4441	0.03	393	2388	100.0	39.08	23.4166	140.0
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	2388.05	8132.90	8.3917	0.03	391	2388	100.0	39.00	23.3737	139.0
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	2388.03	8129.54	8.4031	0.03	390	2388	100.0	38.99	23.4130	138.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13091	100	194	0.0049	0.0000	100.0	518.67	643.24	1599.45	1415.79	14.62	...	2388.00	8213.28	8.4715	0.03	394	2388	100.0	38.65	23.1974	24.0
13092	100	195	-0.0011	-0.0001	100.0	518.67	643.22	1595.69	1422.05	14.62	...	2388.09	8210.85	8.4512	0.03	395	2388	100.0	38.57	23.2771	23.0
13093	100	196	-0.0006	-0.0003	100.0	518.67	643.44	1593.15	1406.82	14.62	...	2388.04	8217.24	8.4569	0.03	395	2388	100.0	38.62	23.2051	22.0
13094	100	197	-0.0038	0.0001	100.0	518.67	643.26	1594.99	1419.36	14.62	...	2388.08	8220.48	8.4711	0.03	395	2388	100.0	38.66	23.2699	21.0
13095	100	198	0.0013	0.0003	100.0	518.67	642.95	1601.62	1424.99	14.62	...	2388.05	8214.64	8.4903	0.03	396	2388	100.0	38.70	23.1855	20.0

13096 rows × 27 columns

In [57]:
```
X_train = (raw_train.drop(columns=['cycle', 'setting 3', 'sensor 1', 'sensor 5', 'sensor 6', 'sensor 10', 'sensor 14', 'sensor 16', 'sensor 18', 'sensor 19','engine','RUL'])).copy()
X_test = (raw_test.drop(columns=['cycle', 'setting 3', 'sensor 1', 'sensor 5', 'sensor 6', 'sensor 10', 'sensor 14' , 'sensor 16', 'sensor 18', 'sensor 19','engine','RUL'])).copy()
Y_train = raw_train[['RUL']]
Y_test = raw_test[['RUL']]
```

In [58]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
scaler = MinMaxScaler()
#scaler = StandardScaler()

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train_scaled),columns = X_train_scaled.columns)
X_test_scaled = pd.DataFrame(scaler.transform(X_test_scaled),columns = X_test_scaled.columns)

Y_train_scaled = Y_train.copy()
Y_test_scaled = Y_test.copy()
Y_train_scaled = pd.DataFrame(scaler.fit_transform(Y_train_scaled),columns = Y_train_scaled.columns)
Y_test_scaled = pd.DataFrame(scaler.fit_transform(Y_test_scaled),columns = Y_test_scaled.columns)

#X_train_scaled = scaler.fit_transform(X_train_scaled)   # fit_transform
#X_test_scaled = scaler.transform(X_test_scaled)
```

Multi-layer Perceptron

In [59]:

```
import tensorflow as tf
from tensorflow import keras

#input = keras.layers.Input(shape=X_train_scaled.shape[1:])
#hidden1 = keras.layers.Dense(30, activation="relu")(input)
#hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
#concat = keras.layers.Concatenate()([input, hidden2])
#output = keras.layers.Dense(1)(concat)
#model = keras.models.Model(inputs=[input], outputs=[output])
model = keras.models.Sequential([keras.layers.Dense(30, activation="relu", input_shape=X_train_scaled.shape[1:])
                                 ,keras.layers.Dense(1)])
model.compile(loss="mean_squared_error", optimizer="sgd", metrics=["accuracy"])
history = model.fit(X_train_scaled, Y_train_scaled, epochs=20)
```

```
Epoch 1/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0275 - accuracy: 0.0048
Epoch 2/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0167 - accuracy: 0.0048
Epoch 3/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0161 - accuracy: 0.0048
Epoch 4/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0157 - accuracy: 0.0048
Epoch 5/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0154 - accuracy: 0.0048
Epoch 6/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0152 - accuracy: 0.0048
Epoch 7/20
645/645 [==============================] - 1s 993us/step - loss: 0.0150 - accuracy: 0.0048
Epoch 8/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0148 - accuracy: 0.0048
Epoch 9/20
645/645 [==============================] - 1s 991us/step - loss: 0.0147 - accuracy: 0.0048
Epoch 10/20
645/645 [==============================] - 1s 984us/step - loss: 0.0146 - accuracy: 0.0048
Epoch 11/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0145 - accuracy: 0.0048
Epoch 12/20
645/645 [==============================] - 1s 991us/step - loss: 0.0145 - accuracy: 0.0048
Epoch 13/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0144 - accuracy: 0.0048
Epoch 14/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0144 - accuracy: 0.0048
Epoch 15/20
645/645 [==============================] - 1s 996us/step - loss: 0.0143 - accuracy: 0.0048
Epoch 16/20
645/645 [==============================] - 1s 988us/step - loss: 0.0143 - accuracy: 0.0048
Epoch 17/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0143 - accuracy: 0.0048
Epoch 18/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0142 - accuracy: 0.0048
Epoch 19/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0142 - accuracy: 0.0048
Epoch 20/20
645/645 [==============================] - 1s 1000us/step - loss: 0.0142 - accuracy: 0.0048
```

In [60]:

```
train_predict_MLP = model.predict(X_train_scaled)
test_predict_MLP = model.predict(X_test_scaled)

train_predict_MLP_unscaled = (train_predict_MLP*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
test_predict_MLP_unscaled = (test_predict_MLP*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
test_predict_MLP_unscaled
```

```
645/645 [==============================] - 1s 857us/step
410/410 [==============================] - 0s 833us/step
```

Out[60]:

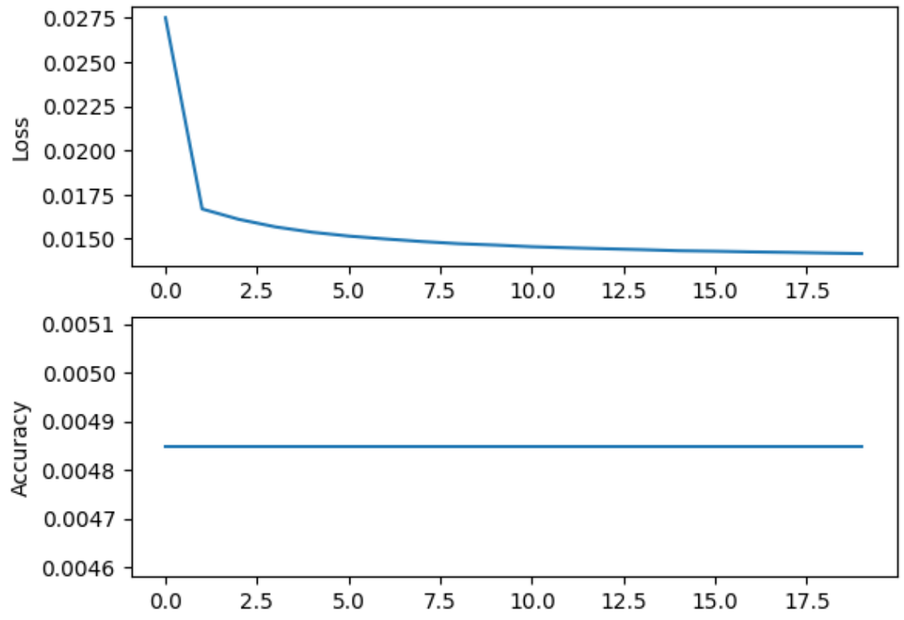
```
array([[162.56404 ],
       [172.17946 ],
       [151.38202 ],
       ...,
       [ 30.834398],
       [  2.680902],
       [ -3.967549]], dtype=float32)
```

In [61]:

```
import matplotlib.pyplot as plt
plt.subplot(2,1,1)
plt.plot(history.history['loss'])
plt.ylabel('Loss')

plt.subplot(2,1,2)
plt.plot(history.history['accuracy'])
plt.ylabel('Accuracy')
```

Out[61]:

```
Text(0, 0.5, 'Accuracy')
```



In [62]:
```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_21 (Dense)            (None, 30)                480

 dense_22 (Dense)            (None, 1)                 31

=================================================================
Total params: 511
Trainable params: 511
Non-trainable params: 0
_________________________________________________________________
```

Recurrent Neural Networks

In [63]:
```
X_train_RNN = np.zeros([len(X_train_scaled),1,len(X_train_scaled.columns)])
X_test_RNN = np.zeros([len(X_test_scaled),1,len(X_test_scaled.columns)])

for i in range(len(X_train_scaled)):
    X_train_RNN[i,0,:] = X_train_scaled.to_numpy()[i,:]

for i in range(len(X_test_scaled)):
    X_test_RNN[i,0,:] = X_test_scaled.to_numpy()[i,:]

X_train_RNN.shape
```

Out[63]:
```
(20631, 1, 15)
```

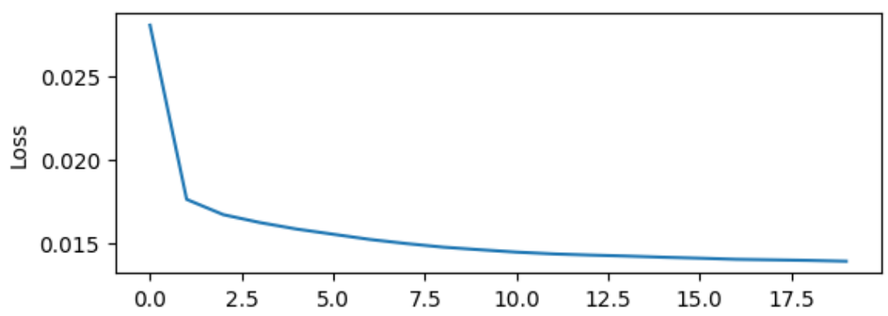
In [64]:
```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from sklearn.metrics import mean_squared_error
import math

Y_train_scaled = Y_train.copy()
Y_train_scaled = pd.DataFrame(scaler.fit_transform(Y_train_scaled),columns = Y_train_scaled.columns)

hidden_units=3
dense_units=1
input_shape= (1, 15)
activation=['tanh', 'tanh']
```

```
model_RNN = Sequential()
model_RNN.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
model_RNN.add(Dense(units=dense_units, activation=activation[1]))
model_RNN.compile(loss='mean_squared_error', optimizer='adam')
history_RNN = model_RNN.fit(X_train_RNN, Y_train_scaled, epochs=20)
```

```
Epoch 1/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0280
Epoch 2/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0176
Epoch 3/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0167
Epoch 4/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0162
Epoch 5/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0158
Epoch 6/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0155
Epoch 7/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0152
Epoch 8/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0150
Epoch 9/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0147
Epoch 10/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0146
Epoch 11/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0145
Epoch 12/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0143
Epoch 13/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0143
Epoch 14/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0142
Epoch 15/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0141
Epoch 16/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0141
Epoch 17/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0140
Epoch 18/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0140
Epoch 19/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0140
Epoch 20/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0139
```

In [65]:
```
plt.subplot(2,1,1)
plt.plot(history_RNN.history['loss'])
plt.ylabel('Loss')
```

Out[65]:
```
Text(0, 0.5, 'Loss')
```



In [66]:
```
train_predict_RNN = model_RNN.predict(X_train_RNN)
test_predict_RNN = model_RNN.predict(X_test_RNN)
```

```
645/645 [==============================] - 1s 894us/step
410/410 [==============================] - 0s 959us/step
```

In [67]:
```
train_predict_RNN_unscaled = (train_predict_RNN*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
test_predict_RNN_unscaled = (test_predict_RNN*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
test_predict_RNN_unscaled
```

Out[67]:
```
array([[173.2057  ],
       [188.10455 ],
       [162.69337 ],
       ...,
       [ 17.342054],
       [ 17.368032],
       [ 19.666815]], dtype=float32)
```

In [68]:
```
model_RNN.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_3 (SimpleRNN)    (None, 3)                 57

 dense_23 (Dense)            (None, 1)                 4

=================================================================
Total params: 61
Trainable params: 61
Non-trainable params: 0
_________________________________________________________________
```

Autoencoders

In [69]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import ReLU
from tensorflow.keras.models import Model
from tensorflow.keras.utils import plot_model
import h5py

n_inputs = len(X_train_scaled.columns)

# define encoder
visible = Input(shape=(n_inputs,))
e = Dense(n_inputs*2)(visible)
e = BatchNormalization()(e)
e = ReLU()(e)
# define bottleneck
n_bottleneck = n_inputs
bottleneck = Dense(n_bottleneck)(e)
# define decoder
d = Dense(n_inputs*2)(bottleneck)
d = BatchNormalization()(d)
d = ReLU()(d)
```

In [70]:
```
# output layer
output = Dense(n_inputs, activation='linear')(d)

# define autoencoder model
model_AutoEn = Model(inputs=visible, outputs=output)

# compile autoencoder model
model_AutoEn.compile(optimizer='adam', loss='mse', metrics=["accuracy"])

# plot the autoencoder
plot_model(model_AutoEn, 'autoencoder.png', show_shapes=True)

# fit the autoencoder model to reconstruct input
history_AutoEn = model_AutoEn.fit(X_train_scaled, Y_train_scaled, epochs=20)
```

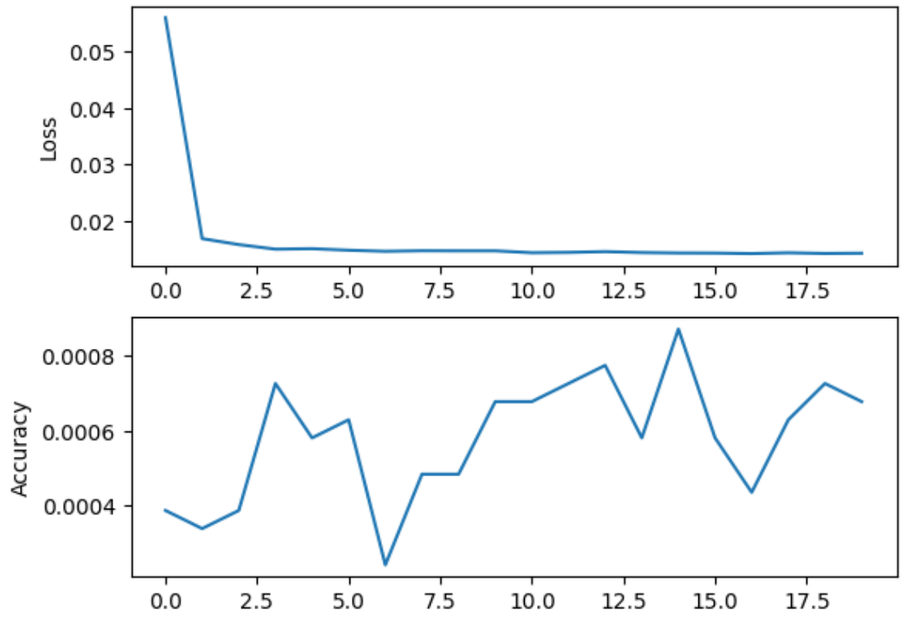
```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model to work.
Epoch 1/20
645/645 [==============================] - 2s 2ms/step - loss: 0.0560 - accuracy: 3.8777e-04
Epoch 2/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0169 - accuracy: 3.3930e-04
Epoch 3/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0159 - accuracy: 3.8777e-04
Epoch 4/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0151 - accuracy: 7.2706e-04
Epoch 5/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0152 - accuracy: 5.8165e-04
Epoch 6/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0149 - accuracy: 6.3012e-04
Epoch 7/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0147 - accuracy: 2.4235e-04
Epoch 8/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0148 - accuracy: 4.8471e-04
Epoch 9/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0148 - accuracy: 4.8471e-04
Epoch 10/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0148 - accuracy: 6.7859e-04
Epoch 11/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0145 - accuracy: 6.7859e-04
Epoch 12/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0145 - accuracy: 7.2706e-04
Epoch 13/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0146 - accuracy: 7.7553e-04
Epoch 14/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0145 - accuracy: 5.8165e-04
Epoch 15/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0144 - accuracy: 8.7247e-04
Epoch 16/20
645/645 [==============================] - 1s 2ms/step - loss: 0.0144 - accuracy: 5.8165e-04
Epoch 17/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0143 - accuracy: 4.3624e-04
Epoch 18/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0144 - accuracy: 6.3012e-04
Epoch 19/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0143 - accuracy: 7.2706e-04
Epoch 20/20
645/645 [==============================] - 1s 1ms/step - loss: 0.0144 - accuracy: 6.7859e-04
```

In [71]:

```
plt.subplot(2,1,1)
plt.plot(history_AutoEn.history['loss'])
plt.ylabel('Loss')

plt.subplot(2,1,2)
plt.plot(history_AutoEn.history['accuracy'])
plt.ylabel('Accuracy')
```

Out[71]:

```
Text(0, 0.5, 'Accuracy')
```



In [72]:
```
# define an encoder model (without the decoder)
encoder = Model(inputs=visible, outputs=bottleneck)
plot_model(encoder, 'encoder.png', show_shapes=True)
# save the encoder to file
encoder.save('encoder.h5')
```

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model to work.
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

In [73]:
```
from keras import models
from sklearn.svm import SVR

encoder = models.load_model('encoder.h5')

# encode the train data
X_train_encode = encoder.predict(X_train_scaled)
# encode the test data
X_test_encode = encoder.predict(X_test_scaled)

# define model
model = SVR()
# fit model on the training dataset
model.fit(X_train_encode, Y_train_scaled)
# make prediction on test set
test_predict_AutoEn = model.predict(X_test_encode)
train_predict_AutoEn = model.predict(X_train_encode)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
645/645 [==============================] - 1s 923us/step
410/410 [==============================] - 0s 924us/step
C:\Users\Niroshan\anaconda3\lib\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  y = column_or_1d(y, warn=True)
```

In [74]:
```
test_predict_AutoEn_unscaled = (test_predict_AutoEn*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
train_predict_AutoEn_unscaled = (train_predict_AutoEn*(Y_train.max()[0] - Y_train.min()[0])) + Y_train.min()[0]
train_predict_AutoEn_unscaled
```

Out[74]:
```
array([163.22781783, 161.75421858, 158.1683973 , ...,  33.83431462,
       32.77203608,  32.47559834])
```

In [92]:
```
data = {'Model':
            ['Multi Layer Percetron'
            , 'Recurrent Neural Networks'
            , 'Autoencoders with SVR  '
            ],
```

```
        'Correlation of real and prediced values (Train)':
            [np.corrcoef(train_predict_MLP_unscaled[:,0],Y_train.to_numpy()[:,0])[0,1]
            , np.corrcoef(train_predict_RNN_unscaled[:,0],Y_train.to_numpy()[:,0])[0,1]
            , np.corrcoef(train_predict_AutoEn_unscaled,Y_train.to_numpy()[:,0])[0,1]
            ],
        'Correlation of real and prediced values (Test)':
            [np.corrcoef(test_predict_MLP_unscaled[:,0],Y_test.to_numpy()[:,0])[0,1]
            , np.corrcoef(test_predict_RNN_unscaled[:,0],Y_test.to_numpy()[:,0])[0,1]
            , np.corrcoef(test_predict_AutoEn_unscaled,Y_test.to_numpy()[:,0])[0,1]
            ],
        'RMSE (Train)':
            [np.sqrt(mean_squared_error(train_predict_MLP_unscaled,Y_train))
            , np.sqrt(mean_squared_error(train_predict_RNN_unscaled,Y_train))
            , np.sqrt(mean_squared_error(train_predict_AutoEn_unscaled,Y_train))
            ],
        'RMSE (Test)': [np.sqrt(mean_squared_error(test_predict_MLP_unscaled,Y_test))
            , np.sqrt(mean_squared_error(test_predict_RNN_unscaled,Y_test))
            ,np.sqrt(mean_squared_error(test_predict_AutoEn_unscaled,Y_test))
            ]}

summary = pd.DataFrame(data)
summary
```

Out[92]:

	Model	Correlation of real and prediced values (Train)	Correlation of real and prediced values (Test)	RMSE (Train)	RMSE (Test)
0	Multi Layer Percetron	0.782452	0.601245	42.952092	47.186879
1	Recurrent Neural Networks	0.787412	0.609949	42.854230	46.864389
2	Autoencoders with SVR	0.798791	0.631809	41.934953	46.027255

In [ ]: