Bearing Failure Anomaly Detection

In this workbook, we use an autoencoder neural network to identify vibrational anomalies from sensor readings in a set of bearings. The goal is to be able to predict future bearing failures before they happen. The vibrational sensor readings are from the NASA Acoustics and Vibration Database. Each data set consists of individual files that are 1-second vibration signal snapshots recorded at 10 minute intervals. Each file contains 20,480 sensor data points that were obtained by reading the bearing sensors at a sampling rate of 20 kHz.

This autoencoder neural network model is created using Long Short-Term Memory (LSTM) recurrent neural network (RNN) cells within the Keras / TensorFlow framework.

In [2]:
```
# import libraries
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.externals import joblib
import seaborn as sns
sns.set(color_codes=True)
import matplotlib.pyplot as plt
%matplotlib inline

from numpy.random import seed
from tensorflow import set_random_seed
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)


from keras.layers import Input, Dropout, Dense, LSTM, TimeDistributed, R
epeatVector
from keras.models import Model
from keras import regularizers
```

In [3]:
```
# set random seed
seed(10)
set_random_seed(10)
```

Data loading and pre-processing

An assumption is that mechanical degradation in the bearings occurs gradually over time; therefore, we use one datapoint every 10 minutes in the analysis. Each 10 minute datapoint is aggregated by using the mean absolute value of the vibration recordings over the 20,480 datapoints in each file. We then merge together everything in a single dataframe.

In [4]:
```
# load, average and merge sensor samples
data_dir = 'data/bearing_data'
merged_data = pd.DataFrame()

for filename in os.listdir(data_dir):
    dataset = pd.read_csv(os.path.join(data_dir, filename), sep='\t')
    dataset_mean_abs = np.array(dataset.abs().mean())
    dataset_mean_abs = pd.DataFrame(dataset_mean_abs.reshape(1,4))
    dataset_mean_abs.index = [filename]
    merged_data = merged_data.append(dataset_mean_abs)

merged_data.columns = ['Bearing 1', 'Bearing 2', 'Bearing 3', 'Bearing
 4']
```

In [5]:
```
# transform data file index to datetime and sort in chronological order
merged_data.index = pd.to_datetime(merged_data.index, format='%Y.%m.%d.%
H.%M.%S')
merged_data = merged_data.sort_index()
merged_data.to_csv('Averaged_BearingTest_Dataset.csv')
print("Dataset shape:", merged_data.shape)
merged_data.head()
```

```
Dataset shape: (982, 4)
```

Out[5]:

	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:52:39	0.060236	0.074227	0.083926	0.044443
2004-02-12 11:02:39	0.061455	0.073844	0.084457	0.045081
2004-02-12 11:12:39	0.061361	0.075609	0.082837	0.045118
2004-02-12 11:22:39	0.061665	0.073279	0.084879	0.044172
2004-02-12 11:32:39	0.061944	0.074593	0.082626	0.044659

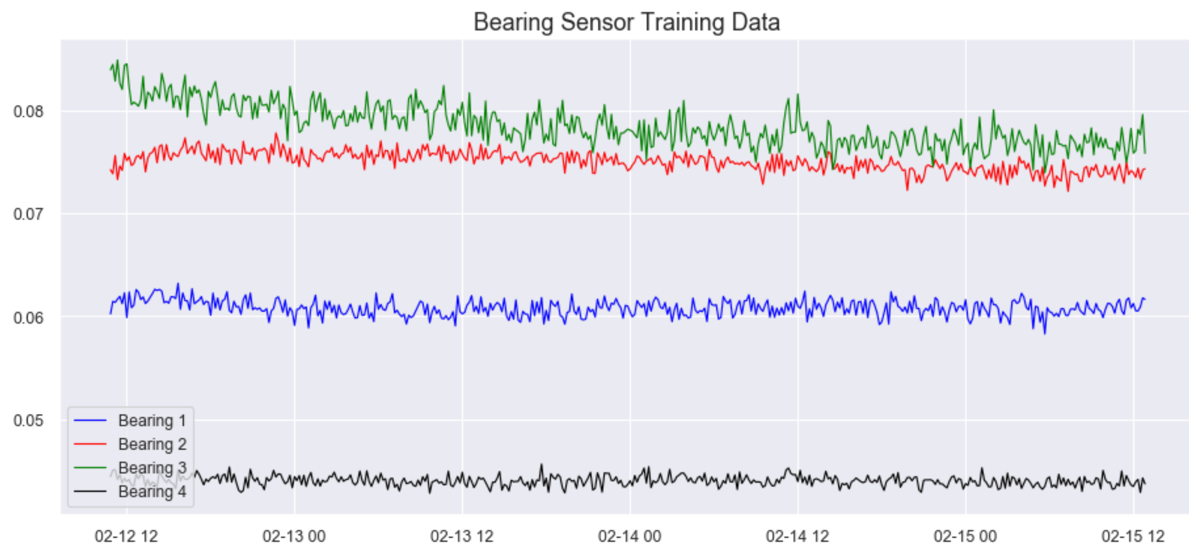
Define train/test data

Before setting up the models, we need to define train/test data. To do this, we perform a simple split where we train on the first part of the dataset (which should represent normal operating conditions) and test on the remaining parts of the dataset leading up to the bearing failure.

In [6]:
```
train = merged_data['2004-02-12 10:52:39': '2004-02-15 12:52:39']
test = merged_data['2004-02-15 12:52:39':]
print("Training dataset shape:", train.shape)
print("Test dataset shape:", test.shape)
```

```
Training dataset shape: (445, 4)
Test dataset shape: (538, 4)
```

In [7]:
```
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
ax.plot(train['Bearing 1'], label='Bearing 1', color='blue', animated = True, linewidth=1)
ax.plot(train['Bearing 2'], label='Bearing 2', color='red', animated = True, linewidth=1)
ax.plot(train['Bearing 3'], label='Bearing 3', color='green', animated = True, linewidth=1)
ax.plot(train['Bearing 4'], label='Bearing 4', color='black', animated = True, linewidth=1)
plt.legend(loc='lower left')
ax.set_title('Bearing Sensor Training Data', fontsize=16)
plt.show()
```

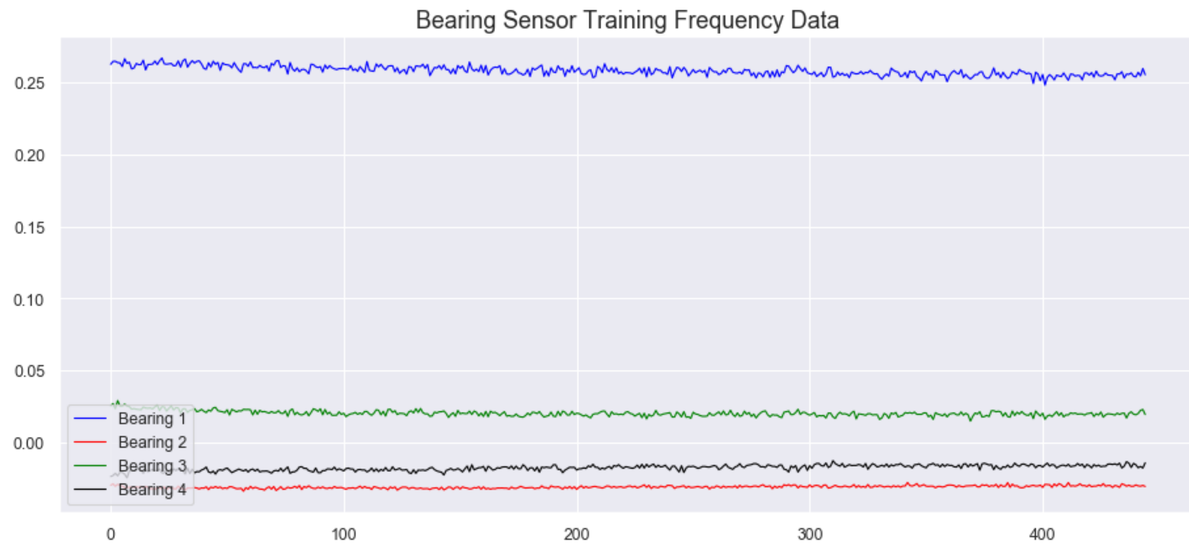


Let's get a different perspective of the data by transforming the signal from the time domain to the frequency domain using a discrete Fourier transform.

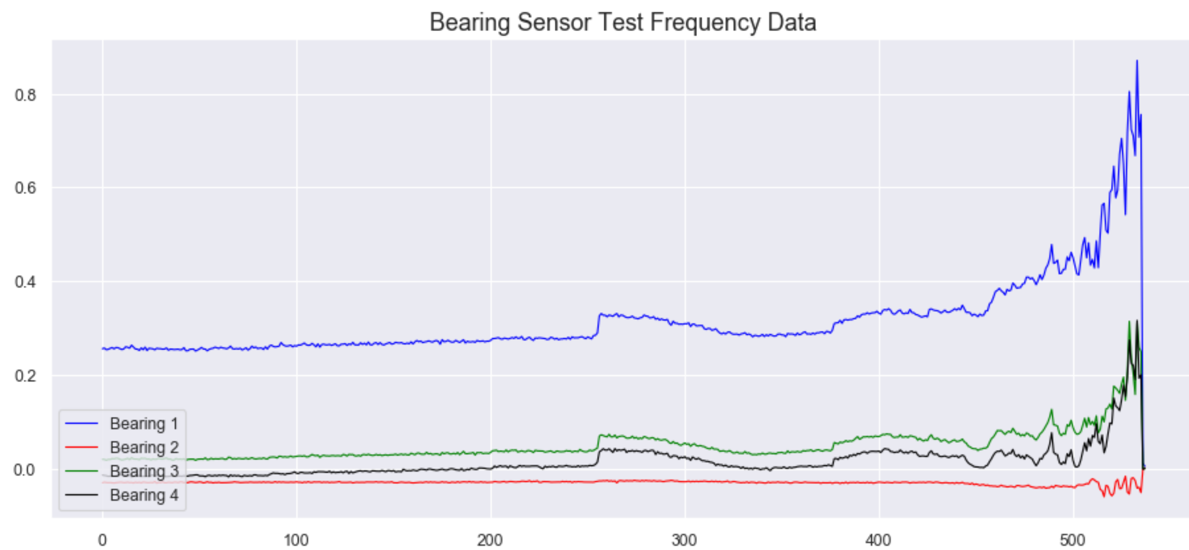
In [8]:
```
# transforming data from the time domain to the frequency domain using fast Fourier transform
train_fft = np.fft.fft(train)
test_fft = np.fft.fft(test)
```

In [9]:

```
# frequencies of the healthy sensor signal
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
ax.plot(train_fft[:,0].real, label='Bearing 1', color='blue', animated =
True, linewidth=1)
ax.plot(train_fft[:,1].imag, label='Bearing 2', color='red', animated =
True, linewidth=1)
ax.plot(train_fft[:,2].real, label='Bearing 3', color='green', animated
= True, linewidth=1)
ax.plot(train_fft[:,3].real, label='Bearing 4', color='black', animated
= True, linewidth=1)
plt.legend(loc='lower left')
ax.set_title('Bearing Sensor Training Frequency Data', fontsize=16)
plt.show()
```



In [10]:
```
# frequencies of the degrading sensor signal
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
ax.plot(test_fft[:,0].real, label='Bearing 1', color='blue', animated = True, linewidth=1)
ax.plot(test_fft[:,1].imag, label='Bearing 2', color='red', animated = True, linewidth=1)
ax.plot(test_fft[:,2].real, label='Bearing 3', color='green', animated = True, linewidth=1)
ax.plot(test_fft[:,3].real, label='Bearing 4', color='black', animated = True, linewidth=1)
plt.legend(loc='lower left')
ax.set_title('Bearing Sensor Test Frequency Data', fontsize=16)
plt.show()
```



In [11]:
```
# normalize the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(train)
X_test = scaler.transform(test)
scaler_filename = "scaler_data"
joblib.dump(scaler, scaler_filename)
```

Out[11]: ['scaler_data']

In [12]:
```
# reshape inputs for LSTM [samples, timesteps, features]
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
print("Training data shape:", X_train.shape)
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
print("Test data shape:", X_test.shape)
```

```
Training data shape: (445, 1, 4)
Test data shape: (538, 1, 4)
```

In [13]:
```python
# define the autoencoder network model
def autoencoder_model(X):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(16, activation='relu', return_sequences=True,
              kernel_regularizer=regularizers.l2(0.00))(inputs)
    L2 = LSTM(4, activation='relu', return_sequences=False)(L1)
    L3 = RepeatVector(X.shape[1])(L2)
    L4 = LSTM(4, activation='relu', return_sequences=True)(L3)
    L5 = LSTM(16, activation='relu', return_sequences=True)(L4)
    output = TimeDistributed(Dense(X.shape[2]))(L5)
    model = Model(inputs=inputs, outputs=output)
    return model
```

In [14]:
```python
# create the autoencoder model
model = autoencoder_model(X_train)
model.compile(optimizer='adam', loss='mae')
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 4)              0
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 16)             1344
_________________________________________________________________
lstm_2 (LSTM)                (None, 4)                 336
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 1, 4)              0
_________________________________________________________________
lstm_3 (LSTM)                (None, 1, 4)              144
_________________________________________________________________
lstm_4 (LSTM)                (None, 1, 16)             1344
_________________________________________________________________
time_distributed_1 (TimeDist (None, 1, 4)              68
=================================================================
Total params: 3,236
Trainable params: 3,236
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
# fit the model to the data
nb_epochs = 100
batch_size = 10
history = model.fit(X_train, X_train, epochs=nb_epochs, batch_size=batch
_size,
                    validation_split=0.05).history
```

```
Train on 422 samples, validate on 23 samples
Epoch 1/100
422/422 [==============================] - 3s 7ms/step - loss: 0.4473 -
val_loss: 0.3246
Epoch 2/100
422/422 [==============================] - 0s 769us/step - loss: 0.3903
- val_loss: 0.2589
Epoch 3/100
422/422 [==============================] - 0s 790us/step - loss: 0.3147
- val_loss: 0.1834
Epoch 4/100
422/422 [==============================] - 0s 746us/step - loss: 0.2171
- val_loss: 0.1514
Epoch 5/100
422/422 [==============================] - 0s 751us/step - loss: 0.1341
- val_loss: 0.1460
Epoch 6/100
422/422 [==============================] - 0s 756us/step - loss: 0.1102
- val_loss: 0.1240
Epoch 7/100
422/422 [==============================] - 0s 742us/step - loss: 0.1053
- val_loss: 0.1200
Epoch 8/100
422/422 [==============================] - 0s 803us/step - loss: 0.1044
- val_loss: 0.1178
Epoch 9/100
422/422 [==============================] - 0s 768us/step - loss: 0.1042
- val_loss: 0.1163
Epoch 10/100
422/422 [==============================] - 0s 766us/step - loss: 0.1034
- val_loss: 0.1175
Epoch 11/100
422/422 [==============================] - 0s 788us/step - loss: 0.1039
- val_loss: 0.1171
Epoch 12/100
422/422 [==============================] - 0s 869us/step - loss: 0.1032
- val_loss: 0.1151
Epoch 13/100
422/422 [==============================] - 0s 786us/step - loss: 0.1029
- val_loss: 0.1164
Epoch 14/100
422/422 [==============================] - 0s 820us/step - loss: 0.1024
- val_loss: 0.1130
Epoch 15/100
422/422 [==============================] - 0s 795us/step - loss: 0.1024
- val_loss: 0.1126
Epoch 16/100
422/422 [==============================] - 0s 842us/step - loss: 0.1021
- val_loss: 0.1129
Epoch 17/100
422/422 [==============================] - 0s 835us/step - loss: 0.1016
- val_loss: 0.1128
Epoch 18/100
422/422 [==============================] - 0s 829us/step - loss: 0.1014
- val_loss: 0.1117
Epoch 19/100
422/422 [==============================] - 0s 793us/step - loss: 0.1012
```

```
- val_loss: 0.1098
Epoch 20/100
422/422 [==============================] - 0s 790us/step - loss: 0.1017
- val_loss: 0.1123
Epoch 21/100
422/422 [==============================] - 0s 828us/step - loss: 0.1008
- val_loss: 0.1094
Epoch 22/100
422/422 [==============================] - 0s 803us/step - loss: 0.1005
- val_loss: 0.1113
Epoch 23/100
422/422 [==============================] - 0s 761us/step - loss: 0.0999
- val_loss: 0.1098
Epoch 24/100
422/422 [==============================] - 0s 845us/step - loss: 0.1000
- val_loss: 0.1090
Epoch 25/100
422/422 [==============================] - 0s 772us/step - loss: 0.0998
- val_loss: 0.1088
Epoch 26/100
422/422 [==============================] - 0s 849us/step - loss: 0.0990
- val_loss: 0.1086
Epoch 27/100
422/422 [==============================] - 0s 813us/step - loss: 0.0991
- val_loss: 0.1075
Epoch 28/100
422/422 [==============================] - 0s 779us/step - loss: 0.0986
- val_loss: 0.1056
Epoch 29/100
422/422 [==============================] - 0s 826us/step - loss: 0.0983
- val_loss: 0.1063
Epoch 30/100
422/422 [==============================] - 0s 886us/step - loss: 0.0982
- val_loss: 0.1067
Epoch 31/100
422/422 [==============================] - 0s 822us/step - loss: 0.0981
- val_loss: 0.1060
Epoch 32/100
422/422 [==============================] - 0s 861us/step - loss: 0.0978
- val_loss: 0.1047
Epoch 33/100
422/422 [==============================] - 0s 814us/step - loss: 0.0977
- val_loss: 0.1031
Epoch 34/100
422/422 [==============================] - 0s 854us/step - loss: 0.0975
- val_loss: 0.1029
Epoch 35/100
422/422 [==============================] - 0s 808us/step - loss: 0.0972
- val_loss: 0.1043
Epoch 36/100
422/422 [==============================] - 0s 879us/step - loss: 0.0970
- val_loss: 0.1035
Epoch 37/100
422/422 [==============================] - 0s 1ms/step - loss: 0.0970 -
val_loss: 0.1027
Epoch 38/100
422/422 [==============================] - 0s 893us/step - loss: 0.0968
```

```
- val_loss: 0.1043
Epoch 39/100
422/422 [==============================] - 0s 1ms/step - loss: 0.0969 -
val_loss: 0.1021
Epoch 40/100
422/422 [==============================] - 0s 1ms/step - loss: 0.0966 -
val_loss: 0.1009
Epoch 41/100
422/422 [==============================] - 0s 961us/step - loss: 0.0965
- val_loss: 0.1021
Epoch 42/100
422/422 [==============================] - 0s 774us/step - loss: 0.0967
- val_loss: 0.1022
Epoch 43/100
422/422 [==============================] - 0s 779us/step - loss: 0.0963
- val_loss: 0.1005
Epoch 44/100
422/422 [==============================] - 0s 739us/step - loss: 0.0962
- val_loss: 0.1018
Epoch 45/100
422/422 [==============================] - 0s 806us/step - loss: 0.0965
- val_loss: 0.1022
Epoch 46/100
422/422 [==============================] - 0s 841us/step - loss: 0.0962
- val_loss: 0.1003
Epoch 47/100
422/422 [==============================] - 0s 793us/step - loss: 0.0958
- val_loss: 0.0998
Epoch 48/100
422/422 [==============================] - 0s 811us/step - loss: 0.0958
- val_loss: 0.1002
Epoch 49/100
422/422 [==============================] - 0s 859us/step - loss: 0.0958
- val_loss: 0.0992
Epoch 50/100
422/422 [==============================] - 0s 748us/step - loss: 0.0959
- val_loss: 0.0997
Epoch 51/100
422/422 [==============================] - 0s 904us/step - loss: 0.0959
- val_loss: 0.0998
Epoch 52/100
422/422 [==============================] - 0s 797us/step - loss: 0.0959
- val_loss: 0.1002
Epoch 53/100
422/422 [==============================] - 0s 778us/step - loss: 0.0956
- val_loss: 0.0991
Epoch 54/100
422/422 [==============================] - 0s 786us/step - loss: 0.0954
- val_loss: 0.0994
Epoch 55/100
422/422 [==============================] - 0s 793us/step - loss: 0.0956
- val_loss: 0.0981
Epoch 56/100
422/422 [==============================] - 0s 754us/step - loss: 0.0952
- val_loss: 0.0988
Epoch 57/100
422/422 [==============================] - 0s 850us/step - loss: 0.0953
```

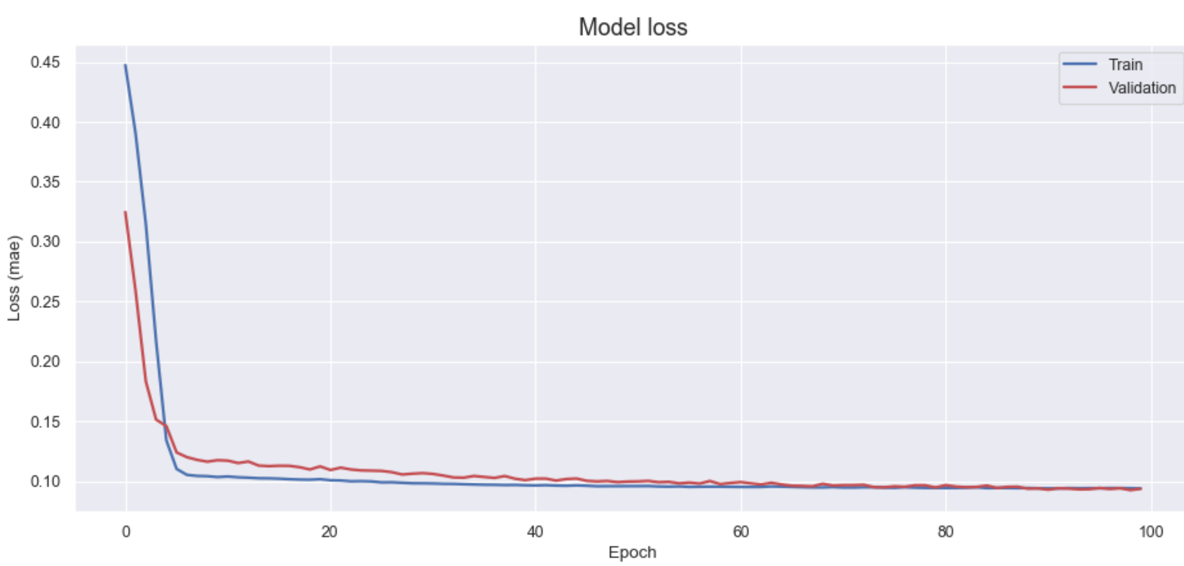
```
- val_loss: 0.0980
Epoch 58/100
422/422 [==============================] - 0s 724us/step - loss: 0.0954
- val_loss: 0.1001
Epoch 59/100
422/422 [==============================] - 0s 788us/step - loss: 0.0954
- val_loss: 0.0976
Epoch 60/100
422/422 [==============================] - 0s 814us/step - loss: 0.0953
- val_loss: 0.0985
Epoch 61/100
422/422 [==============================] - 0s 771us/step - loss: 0.0952
- val_loss: 0.0992
Epoch 62/100
422/422 [==============================] - 0s 754us/step - loss: 0.0952
- val_loss: 0.0981
Epoch 63/100
422/422 [==============================] - 0s 813us/step - loss: 0.0952
- val_loss: 0.0973
Epoch 64/100
422/422 [==============================] - 0s 758us/step - loss: 0.0956
- val_loss: 0.0986
Epoch 65/100
422/422 [==============================] - 0s 770us/step - loss: 0.0954
- val_loss: 0.0972
Epoch 66/100
422/422 [==============================] - 0s 830us/step - loss: 0.0952
- val_loss: 0.0962
Epoch 67/100
422/422 [==============================] - 0s 780us/step - loss: 0.0950
- val_loss: 0.0960
Epoch 68/100
422/422 [==============================] - 0s 780us/step - loss: 0.0948
- val_loss: 0.0957
Epoch 69/100
422/422 [==============================] - 0s 903us/step - loss: 0.0948
- val_loss: 0.0978
Epoch 70/100
422/422 [==============================] - 0s 888us/step - loss: 0.0951
- val_loss: 0.0964
Epoch 71/100
422/422 [==============================] - 0s 911us/step - loss: 0.0947
- val_loss: 0.0967
Epoch 72/100
422/422 [==============================] - 0s 797us/step - loss: 0.0947
- val_loss: 0.0967
Epoch 73/100
422/422 [==============================] - 0s 754us/step - loss: 0.0949
- val_loss: 0.0970
Epoch 74/100
422/422 [==============================] - 0s 782us/step - loss: 0.0949
- val_loss: 0.0950
Epoch 75/100
422/422 [==============================] - 0s 832us/step - loss: 0.0946
- val_loss: 0.0949
Epoch 76/100
422/422 [==============================] - 0s 758us/step - loss: 0.0944
```

```
- val_loss: 0.0957
Epoch 77/100
422/422 [==============================] - 0s 810us/step - loss: 0.0949
- val_loss: 0.0953
Epoch 78/100
422/422 [==============================] - 0s 817us/step - loss: 0.0946
- val_loss: 0.0966
Epoch 79/100
422/422 [==============================] - 0s 742us/step - loss: 0.0944
- val_loss: 0.0965
Epoch 80/100
422/422 [==============================] - 0s 754us/step - loss: 0.0944
- val_loss: 0.0947
Epoch 81/100
422/422 [==============================] - 0s 737us/step - loss: 0.0943
- val_loss: 0.0967
Epoch 82/100
422/422 [==============================] - 0s 757us/step - loss: 0.0943
- val_loss: 0.0955
Epoch 83/100
422/422 [==============================] - 0s 775us/step - loss: 0.0946
- val_loss: 0.0950
Epoch 84/100
422/422 [==============================] - 0s 753us/step - loss: 0.0949
- val_loss: 0.0952
Epoch 85/100
422/422 [==============================] - 0s 782us/step - loss: 0.0942
- val_loss: 0.0963
Epoch 86/100
422/422 [==============================] - 0s 714us/step - loss: 0.0943
- val_loss: 0.0943
Epoch 87/100
422/422 [==============================] - 0s 823us/step - loss: 0.0943
- val_loss: 0.0952
Epoch 88/100
422/422 [==============================] - 0s 733us/step - loss: 0.0942
- val_loss: 0.0954
Epoch 89/100
422/422 [==============================] - 0s 706us/step - loss: 0.0943
- val_loss: 0.0937
Epoch 90/100
422/422 [==============================] - 0s 749us/step - loss: 0.0941
- val_loss: 0.0940
Epoch 91/100
422/422 [==============================] - 0s 772us/step - loss: 0.0940
- val_loss: 0.0929
Epoch 92/100
422/422 [==============================] - 0s 749us/step - loss: 0.0940
- val_loss: 0.0941
Epoch 93/100
422/422 [==============================] - 0s 817us/step - loss: 0.0942
- val_loss: 0.0938
Epoch 94/100
422/422 [==============================] - 0s 784us/step - loss: 0.0941
- val_loss: 0.0932
Epoch 95/100
422/422 [==============================] - 0s 759us/step - loss: 0.0942
```

```
- val_loss: 0.0933
Epoch 96/100
422/422 [==============================] - 0s 767us/step - loss: 0.0941
- val_loss: 0.0943
Epoch 97/100
422/422 [==============================] - 0s 741us/step - loss: 0.0943
- val_loss: 0.0934
Epoch 98/100
422/422 [==============================] - 0s 743us/step - loss: 0.0942
- val_loss: 0.0941
Epoch 99/100
422/422 [==============================] - 1s 1ms/step - loss: 0.0941 -
val_loss: 0.0925
Epoch 100/100
422/422 [==============================] - 1s 1ms/step - loss: 0.0939 -
val_loss: 0.0936
```

In [16]:

```
# plot the training losses
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
ax.plot(history['loss'], 'b', label='Train', linewidth=2)
ax.plot(history['val_loss'], 'r', label='Validation', linewidth=2)
ax.set_title('Model loss', fontsize=16)
ax.set_ylabel('Loss (mae)')
ax.set_xlabel('Epoch')
ax.legend(loc='upper right')
plt.show()
```



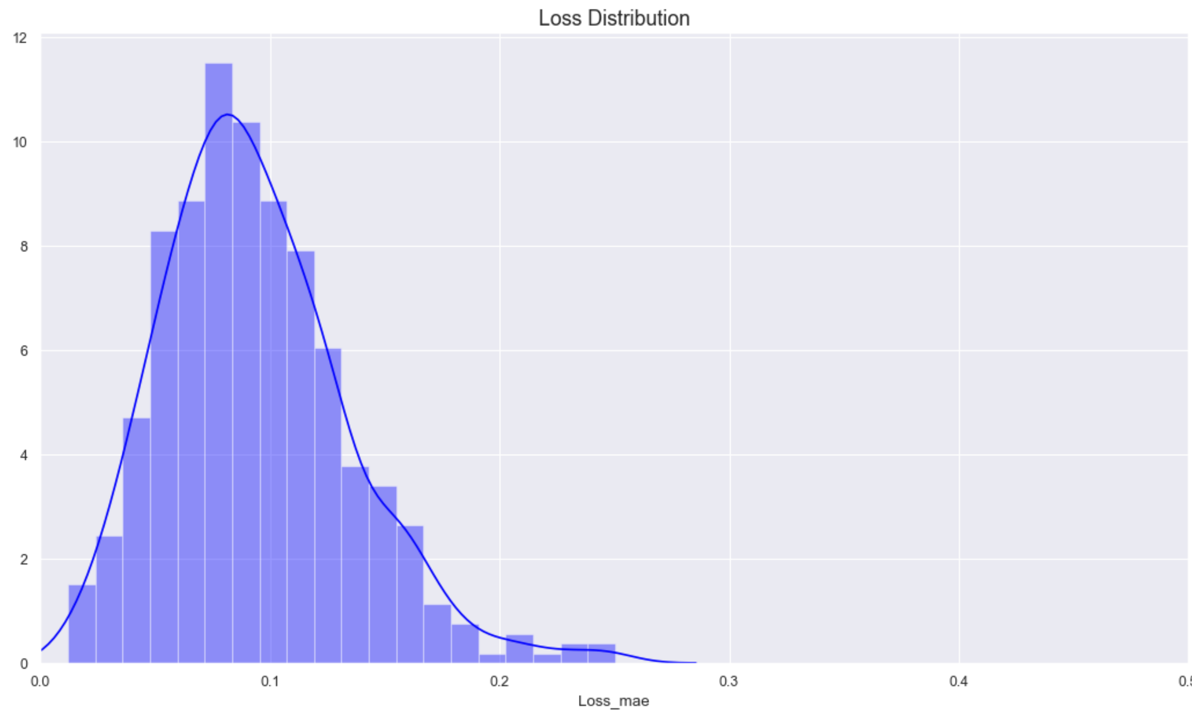
Distribution of Loss Function

By plotting the distribution of the calculated loss in the training set, one can use this to identify a suitable threshold value for identifying an anomaly. In doing this, one can make sure that this threshold is set above the “noise level” and that any flagged anomalies should be statistically significant above the background noise.

In [17]:
```
# plot the loss distribution of the training set
X_pred = model.predict(X_train)
X_pred = X_pred.reshape(X_pred.shape[0], X_pred.shape[2])
X_pred = pd.DataFrame(X_pred, columns=train.columns)
X_pred.index = train.index

scored = pd.DataFrame(index=train.index)
Xtrain = X_train.reshape(X_train.shape[0], X_train.shape[2])
scored['Loss_mae'] = np.mean(np.abs(X_pred-Xtrain), axis = 1)
plt.figure(figsize=(16,9), dpi=80)
plt.title('Loss Distribution', fontsize=16)
sns.distplot(scored['Loss_mae'], bins = 20, kde= True, color = 'blue');
plt.xlim([0.0,.5])
```

Out[17]:
```
(0.0, 0.5)
```



From the above loss distribution, let's try a threshold value of 0.275 for flagging an anomaly. We can then calculate the loss in the test set to check when the output crosses the anomaly threshold.

In [18]:

```
# calculate the loss on the test set
X_pred = model.predict(X_test)
X_pred = X_pred.reshape(X_pred.shape[0], X_pred.shape[2])
X_pred = pd.DataFrame(X_pred, columns=test.columns)
X_pred.index = test.index

scored = pd.DataFrame(index=test.index)
Xtest = X_test.reshape(X_test.shape[0], X_test.shape[2])
scored['Loss_mae'] = np.mean(np.abs(X_pred-Xtest), axis = 1)
scored['Threshold'] = 0.275
scored['Anomaly'] = scored['Loss_mae'] > scored['Threshold']
scored.head()
```

Out[18]:

	Loss_mae	Threshold	Anomaly
2004-02-15 12:52:39	0.096617	0.275	False
2004-02-15 13:02:39	0.178313	0.275	False
2004-02-15 13:12:39	0.065853	0.275	False
2004-02-15 13:22:39	0.049520	0.275	False
2004-02-15 13:32:39	0.043890	0.275	False

In [20]:

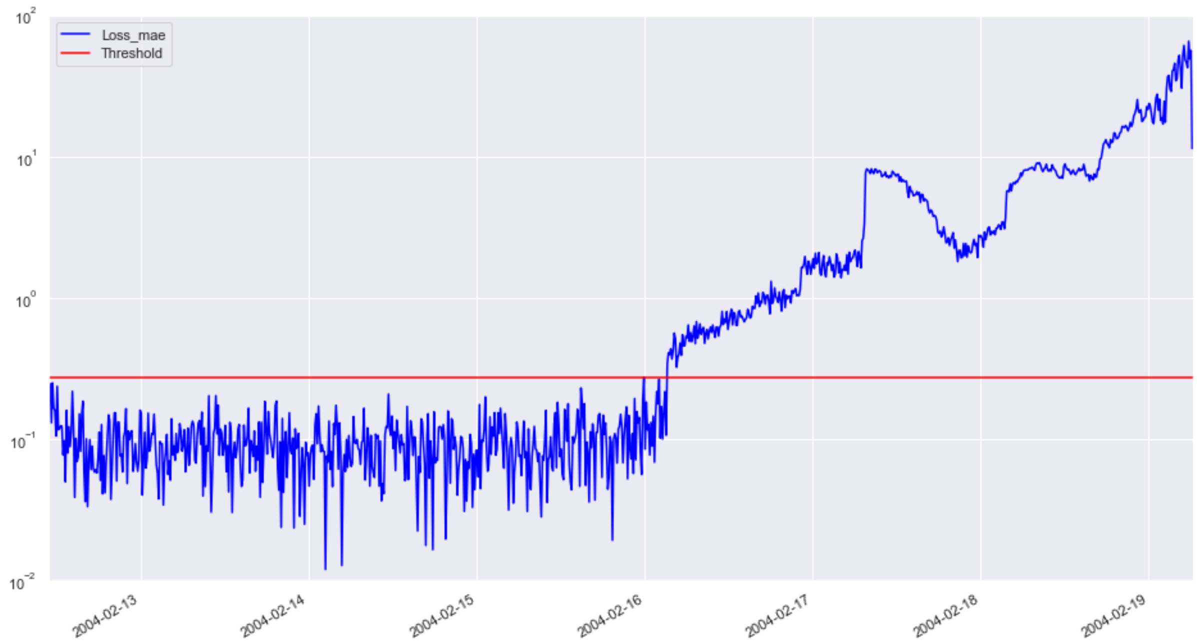
```
# calculate the same metrics for the training set
# and merge all data in a single dataframe for plotting
X_pred_train = model.predict(X_train)
X_pred_train = X_pred_train.reshape(X_pred_train.shape[0], X_pred_train.
shape[2])
X_pred_train = pd.DataFrame(X_pred_train, columns=train.columns)
X_pred_train.index = train.index

scored_train = pd.DataFrame(index=train.index)
scored_train['Loss_mae'] = np.mean(np.abs(X_pred_train-Xtrain), axis = 1
)
scored_train['Threshold'] = 0.275
scored_train['Anomaly'] = scored_train['Loss_mae'] > scored_train['Thres
hold']
scored = pd.concat([scored_train, scored])
```

Having calculated the loss distribution and the anomaly threshold, we can visualize the model output in the time leading up to the bearing failure.

In [21]:
```
# plot bearing failure time plot
scored.plot(logy=True,  figsize=(16,9), ylim=[1e-2,1e2], color=['blue',
'red'])
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2e130ef0>



This analysis approach is able to flag the upcoming bearing malfunction well in advance of the actual physical failure. It is important to define a suitable threshold value for flagging anomalies while avoiding too many false positives during normal operating conditions.

In [22]:
```
# save all model information, including weights, in h5 format
model.save("Cloud_model.h5")
print("Model saved")
```

Model saved

In [ ]:
```
```