Import Packages

```
# Standard Math and Data packages
import numpy as np
import pandas as pd

# Input data retrieval package
from google.colab import drive

# Plotting package
import matplotlib.pyplot as plt
# Scaling Package
from sklearn.preprocessing import MinMaxScaler

# Keras Network @ https://www.tensorflow.org/guide/keras/gru
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Set Random seed
import random
random.seed(42)
```

Retrieve the Data

```
drive.mount('/content/gdrive')
feeds = pd.read_csv('gdrive/My Drive/feeds.csv', index_col='created_at', parse_dates=['created_at'])
feeds.head()
```

```
Mounted at /content/gdrive
```

created_at	device_id	feed_entry_id	luminosity_value	vibration_value	alert
2024-11-26 20:51:07+00:00	1885576	499284	489	123	0
2024-11-26 20:51:27+00:00	1885576	499285	609	0	0
2024-11-26 20:51:39+00:00	1885576	499286	580	0	0
2024-11-26 20:51:46+00:00	1885576	499287	536	0	0
2024-11-26 20:52:03+00:00	1885576	499288	610	0	0

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

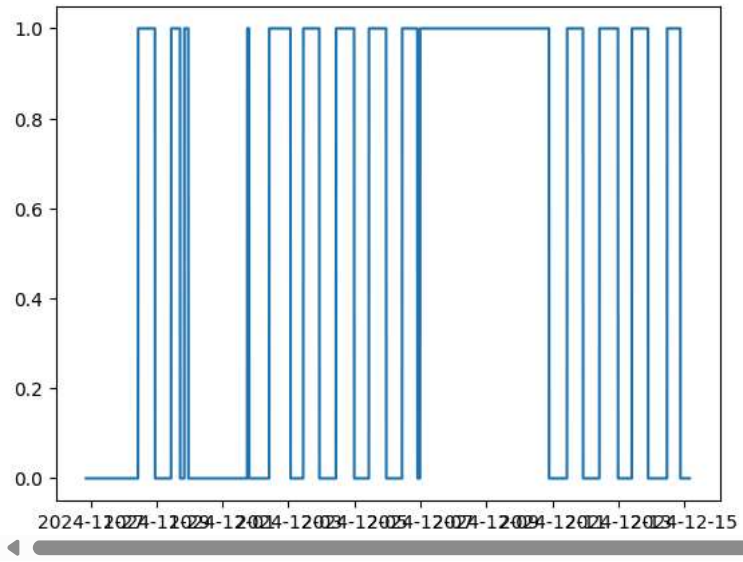
```
feeds = feeds.drop(columns=['device_id', 'feed_entry_id'])
feeds.head()
```

created_at	luminosity_value	vibration_value	alert
2024-11-26 20:51:07+00:00	489	123	0
2024-11-26 20:51:27+00:00	609	0	0
2024-11-26 20:51:39+00:00	580	0	0
2024-11-26 20:51:46+00:00	536	0	0
2024-11-26 20:52:03+00:00	610	0	0

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

```
plt.plot(feeds['alert'])
```

```
[<matplotlib.lines.Line2D at 0x7d8ad7a671c0>]
```



Remove Invalid Entries

```
feeds['luminosity_value'] = pd.to_numeric(feeds['luminosity_value'], errors='coerce')
feeds['vibration_value'] = pd.to_numeric(feeds['vibration_value'], errors='coerce')

feeds = feeds.dropna(subset=['luminosity_value'])
feeds.loc[:, 'vibration_value'] = feeds['vibration_value'].fillna(0)

feeds.head()
```

created_at	luminosity_value	vibration_value	alert
2024-11-26 20:51:07+00:00	489.0	123.0	0
2024-11-26 20:51:27+00:00	609.0	0.0	0
2024-11-26 20:51:39+00:00	580.0	0.0	0
2024-11-26 20:51:46+00:00	536.0	0.0	0
2024-11-26 20:52:03+00:00	610.0	0.0	0

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

Convert Numbers to Percentages

```
alerts = feeds['alert']

x = feeds.values
min_max_scaler = MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)

pct_change_feeds = pd.DataFrame(x_scaled)
pct_change_feeds.columns = feeds.columns

pct_change_feeds.head()
```

	luminosity_value	vibration_value	alert
0	0.788026	0.004176	0.0
1	0.982201	0.000000	0.0
2	0.935275	0.000000	0.0
3	0.864078	0.000000	0.0
4	0.983819	0.000000	0.0

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

Univariate Forecasting (GRU)

[ ] ↳ *7 cells hidden*

Multi-Step Forecast

[ ] ↳ *3 cells hidden*

Multi-Variate Forecasting

```
# Need the data to be in the form [sample, time steps, features (dimension of each element)].
samples = 20 # Number of samples (in past).
steps = 1 # Number of steps (in future).
X = []
Y = []
for i in range(pct_change_feeds.shape[0] - samples):
    X.append(pct_change_feeds.iloc[i:i+samples, 0:2].values) # Independent Samples.
    Y.append(pct_change_feeds.iloc[i+samples, 2:].values) # Dependent Samples.
print('Training data length is', len(X[0:1][0]), ':', X[0:1])
print('Testing data length is', len(Y[0:1]), ':', Y[0:1])
```

```
Training data length is 20 : [array([[0.78802589, 0.00417615],
            [0.98220065, 0.        ],
            [0.93527508, 0.        ],
            [0.86407767, 0.        ],
            [0.98381877, 0.        ],
            [0.9579288 , 0.00546634],
            [0.9789644 , 0.00179948],
            [0.98867314, 0.        ],
            [0.98867314, 0.        ],
            [0.99190939, 0.        ],
            [0.9789644 , 0.        ],
            [0.96278317, 0.        ],
            [1.        , 0.        ],
            [0.97087379, 0.        ],
            [0.97734628, 0.        ],
            [0.98867314, 0.        ],
            [0.97734628, 0.        ],
            [0.80097087, 0.        ],
            [0.76213592, 0.        ],
            [0.78964401, 0.        ]])]
     Testing data length is 1 : [array([0.])]
```

```
# Reshape the data so that the inputs will be acceptable to the model.
X = np.array(X)
Y = np.array(Y)
print('Dimensions of X', X.shape, 'Dimensions of Y', Y.shape)
```

```
Dimensions of X (48374, 20, 2) Dimensions of Y (48374, 1)
```

```
# Get the training and testing set.
threshold = round(0.9 * X.shape[0])
trainX, trainY = X[:threshold], Y[:threshold]
testX, testY =  X[threshold:], Y[threshold:]
print('Training length', trainX.shape, trainY.shape, 'Testing length:', testX.shape, testY.shape)
```

```
Training length (43537, 20, 2) (43537, 1) Testing length: (4837, 20, 2) (4837, 1)
```

```
# Build the GRU.
model = keras.Sequential()

# Add a GRU layer with 15 units.
model.add(layers.GRU(15,
                     activation = "tanh",
                     recurrent_activation = "sigmoid",
                     input_shape=(X.shape[1], X.shape[2])))
# Add a dropout layer (penalizing more complex models), prevents overfitting.
model.add(layers.Dropout(rate=0.2))


# Add a Dense layer with 1 units (because we are doing a regression task).
model.add(layers.Dense(1))

# Evaluating loss function of MSE using the adam optimizer.
model.compile(loss='binary_crossentropy', optimizer = 'adam')
```

```
# Print out architecture.
model.summary()
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
gru_11 (GRU)	(None, 15)	855
dropout_11 (Dropout)	(None, 15)	0
dense_11 (Dense)	(None, 1)	16

Total params: 871 (3.40 KB)
Trainable params: 871 (3.40 KB)

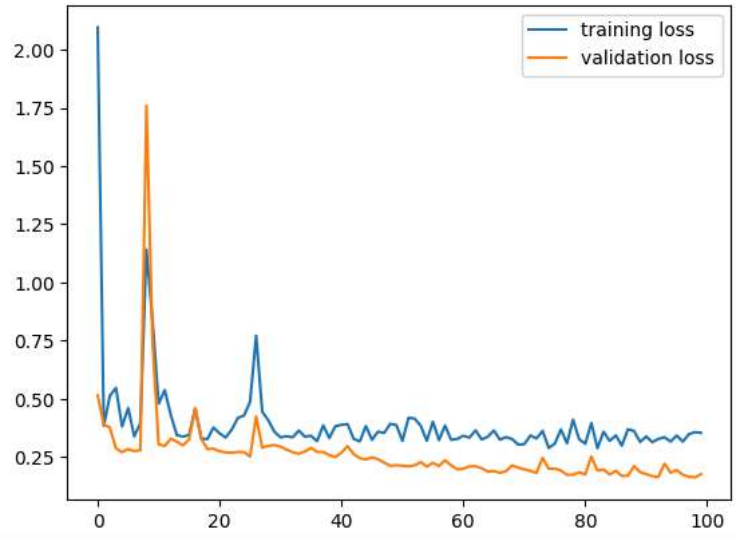
```
# Fitting the data.
history = model.fit(trainX,
                    trainY,
                    shuffle = False, # Since this is time series data.
                    epochs=100,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=1) # Verbose outputs data.
```

```
Epoch 72/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.6778 - val_loss: 0.1895
Epoch 73/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.6784 - val_loss: 0.1815
Epoch 74/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.7452 - val_loss: 0.2463
Epoch 75/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.4569 - val_loss: 0.1989
Epoch 76/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 0.6114 - val_loss: 0.2003
Epoch 77/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - loss: 0.7588 - val_loss: 0.1917
Epoch 78/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - loss: 0.6489 - val_loss: 0.1731
Epoch 79/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.9247 - val_loss: 0.1736
Epoch 80/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - loss: 0.6436 - val_loss: 0.1842
Epoch 81/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - loss: 0.5778 - val_loss: 0.1750
Epoch 82/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - loss: 0.7865 - val_loss: 0.2526
Epoch 83/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 0.4220 - val_loss: 0.1915
Epoch 84/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - loss: 0.6663 - val_loss: 0.1962
Epoch 85/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 11ms/step - loss: 0.5811 - val_loss: 0.1750
Epoch 86/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.6545 - val_loss: 0.1905
Epoch 87/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 0.5597 - val_loss: 0.1684
Epoch 88/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - loss: 0.7837 - val_loss: 0.1692
Epoch 89/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - loss: 0.7279 - val_loss: 0.2106
Epoch 90/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - loss: 0.5690 - val_loss: 0.1851
Epoch 91/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - loss: 0.6644 - val_loss: 0.1763
Epoch 92/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.6066 - val_loss: 0.1673
Epoch 93/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - loss: 0.6710 - val_loss: 0.1641
Epoch 94/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.6565 - val_loss: 0.2212
Epoch 95/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.5305 - val_loss: 0.1826
Epoch 96/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.6407 - val_loss: 0.1936
Epoch 97/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.5760 - val_loss: 0.1736
Epoch 98/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - loss: 0.7231 - val_loss: 0.1649
Epoch 99/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.7549 - val_loss: 0.1624
Epoch 100/100
1089/1089 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 0.7398 - val_loss: 0.1763
```

```
# Plotting the loss iteration.
plt.plot(history.history['loss'], label = 'training loss')
plt.plot(history.history['val_loss'], label ='validation loss')
plt.legend()
```

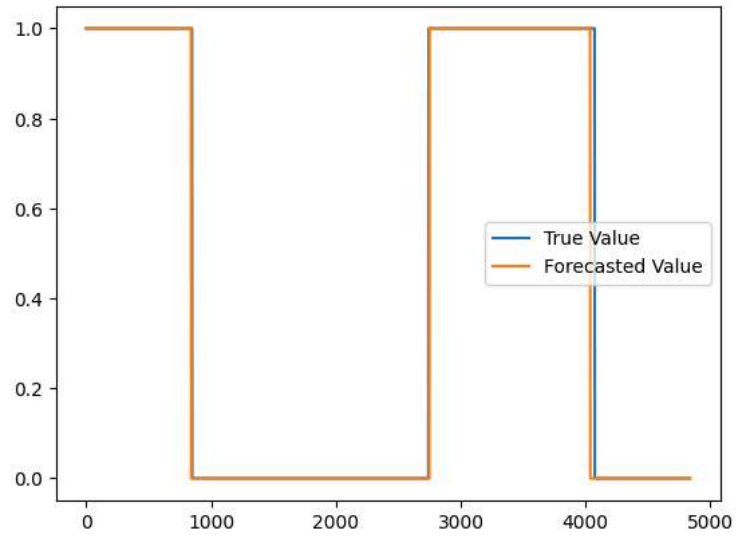
```
# Note:
# if training loss >> validation loss -> Underfitting.
# if training loss << validation loss -> Overfitting (i.e model is smart enough to have mapped the entire dataset...).
# Several ways to address overfitting:
# - Reduce complexity of model (hidden layers, neurons, parameters input etc).
# - Add dropout and tune rate.
# - More data.
```

```
<matplotlib.legend.Legend at 0x7d8ac4644e50>
```




```
# This is a one step forecast (based on how the model was constructed).
y_pred = model.predict(testX)
y_pred = (y_pred > 0.3).astype(int) # Classifier.
plt.plot(testY, label = 'True Value')
plt.plot(y_pred, label = 'Forecasted Value')
plt.legend()
```

```
152/152 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
<matplotlib.legend.Legend at 0x7d8ad4ab2e90>
```



Save the Model

```
model.save('/content/gdrive/MyDrive/tcc3_2.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

.

.

.