Aircraft Maintenance, Repair & Overhaul Solutions

Problem Statement

Determine if an engine will fail within a specific cycle based on its past cycles and sensory inputs.

Engine problems can affect aircraft extremely quickly. As a result, maintaining them in excellent shape is essential for the passengers' safety. The cost of maintaining an aircraft is quite high. However, we also don't want to take upkeep too seriously. If a problem is not found in a timely manner, maintaining and repairing the engines may become too costly, or they may need to be replaced.

Algorithm Used

Long Short-Term Memory (LSTM) and recurrent neural networks (RNN) for variables that change over time (time-dependent variables).

Import Libraries

```
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, recall_score, precision_score
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import Dense, Dropout, SimpleRNN, LSTM, GRU


np.random.seed(1234)
PYTHONHASHSEED = 0
```

Import Dataset

```
train_df = pd.read_csv('/content/PM_train.txt', sep=" ", header=None)
```

```
test_df = pd.read_csv('/content/PM_test.txt', sep=" ", header=None)
```

```
train_df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190	NaN	NaN
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236	NaN	NaN
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442	NaN	NaN
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739	NaN	NaN
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044	NaN	NaN

5 rows × 28 columns

```
train_df.dropna(axis=1, inplace=True)
test_df.dropna(axis=1, inplace=True)
```

```
print(len(train_df))
```

```
20631
```

```
print(len(test_df))
```

```
13096
```

```
cols_names = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                    's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                    's15', 's16', 's17', 's18', 's19', 's20', 's21']
train_df.columns = cols_names
test_df.columns = cols_names
```

```
train_df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	1(
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	1(
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	1(
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	1(
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	1(

5 rows × 26 columns

```
test_df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	
0	1	1	0.0023	0.0003	100.0	518.67	643.02	1585.29	1398.21	14.62	...	521.72	2388.03	8125.55	8.4052	0.03	392	2388	1(
1	1	2	-0.0027	-0.0003	100.0	518.67	641.71	1588.45	1395.42	14.62	...	522.16	2388.06	8139.62	8.3803	0.03	393	2388	1(
2	1	3	0.0003	0.0001	100.0	518.67	642.46	1586.94	1401.34	14.62	...	521.97	2388.03	8130.10	8.4441	0.03	393	2388	1(
3	1	4	0.0042	0.0000	100.0	518.67	642.44	1584.12	1406.42	14.62	...	521.38	2388.05	8132.90	8.3917	0.03	391	2388	1(
4	1	5	0.0014	0.0000	100.0	518.67	642.51	1587.19	1401.92	14.62	...	522.15	2388.03	8129.54	8.4031	0.03	390	2388	1(

5 rows × 26 columns

```
truth_df = pd.read_csv('/content/PM_truth.txt', sep=" ", header=None)
```

```
truth_df.head()
```

	0	1
0	112	NaN
1	98	NaN
2	69	NaN
3	82	NaN
4	91	NaN

```
truth_df.dropna(axis=1, inplace=True)
```

```
truth_df.head()
```

	0
0	112
1	98
2	69
3	82
4	91

Data Preprocessing

1. Generating Classification Target Variable

The cycles that are on the verge of breaking down will be categorised and assigned an id of 1 in the failure_within_w1 goal variable.

The window_1, which in this instance is 30, is denoted by w1.

IMP: In this instance, the aircraft was in poor condition before the most recent cycle, and repair was in fact necessary. Here, estimating the x number of cycles required for the aircraft to break down is crucial. We refer to the number of cycles prior to the breakdown as the window. The window in this instance may be 15, 30, 45, etc. It depends on the accuracy reliance and how early we want to notify the maintenance. Here, we're going to assume that the window is 30 in size. (Aim to find out if the maintenance requiered before 30 cycles)

id	cycle	remaining_useful_life = (max - cycle)	Other columns...
1	1	191	...
1	2	190	...
1	..	...	...
1	...	...	...
1	162	30	...
1	163	29	...
1	..	...	...
1	..	...	...
1	192	0	...
2	1	286	...
2	2	285	...
2	..	...	...
2	257	30	...
2	258	29	...
2	..	...	...
2	..	...	...
2	287	0	...

determine if it will fail in next w1 cycles here, w1 = 30

id	cycle	remaining_useful_life = (max - cycle)	failure_within_w1 (if RUL is greater than w1)	Other columns...
1	1	191	0	...
1	2	190	0	...
1	..	...	..	...
1	...	...	..	...
1	162	30	1	...
1	163	29	1	...
1	..	...	...	...
1	..	...	...	...
1	192	0	1	...
2	1	286	0	...
2	2	285	0	
2	..	...	...	...
2	257	30	1	...
2	258	29	1	...
2	..	...	...	...
2	..	...	...	...
2	287	0	1	...

To enable each engine's values to be sorted and saved in a single location, sort the data set according to cycles and id. Utilising sort_values().

```
train_df.sort_values(['id','cycle'], inplace=True)
test_df.sort_values(['id','cycle'], inplace=True)
```

Using groupby(), we first determine the maximum cycles seen for each engine. Then, we use merge() to combine these numbers for the corresponding engine data.

Next, we deduct the current cycle value from the maximum to get the remaining usable life (RUL). For instance, if a cycle has completed 50 of its 192-cycle total life, its residual useful life (RUL) is 192-50 = 142.

```
rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
```

```
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
```

```
train_df['RUL'] = train_df['max'] - train_df['cycle']
```

```
train_df[['id','cycle','max','RUL']].head()
```

	id	cycle	max	RUL
0	1	1	192	191
1	1	2	192	190
2	1	3	192	189
3	1	4	192	188
4	1	5	192	187

```
train_df.drop('max', axis=1, inplace=True)
```

Generate label column for training data

```
w1 = 30
train_df['failure_within_w1'] = np.where(train_df['RUL'] <= w1, 1, 0 )
```

2. Normalize Training Dataset

```
train_df['cycle_norm'] = train_df['cycle']

cols_normalize = train_df.columns.difference(['id','cycle','RUL','failure_within_w1'])

min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)

join_df = train_df[['id','cycle','RUL','failure_within_w1']].join(norm_train_df)
train_df = join_df.reindex(columns = train_df.columns)

train_df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s15	s16	s17	s18	s19	s20	s21
0	1	1	0.459770	0.166667	0.0	0.0	0.183735	0.406802	0.309757	0.0	...	0.363986	0.0	0.333333	0.0	0.0	0.713178	0.724662
1	1	2	0.609195	0.250000	0.0	0.0	0.283133	0.453019	0.352633	0.0	...	0.411312	0.0	0.333333	0.0	0.0	0.666667	0.731014
2	1	3	0.252874	0.750000	0.0	0.0	0.343373	0.369523	0.370527	0.0	...	0.357445	0.0	0.166667	0.0	0.0	0.627907	0.621375
3	1	4	0.540230	0.500000	0.0	0.0	0.343373	0.256159	0.331195	0.0	...	0.166603	0.0	0.333333	0.0	0.0	0.573643	0.662386
4	1	5	0.390805	0.333333	0.0	0.0	0.349398	0.257467	0.404625	0.0	...	0.402078	0.0	0.416667	0.0	0.0	0.589147	0.704502

5 rows × 29 columns

The test data should be normalised in the same way as the training data was.

```
test_df['cycle_norm'] = test_df['cycle']

norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)

test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns = test_df.columns)
test_df = test_df.reset_index(drop=True)
```

As with the training data, label the test set.

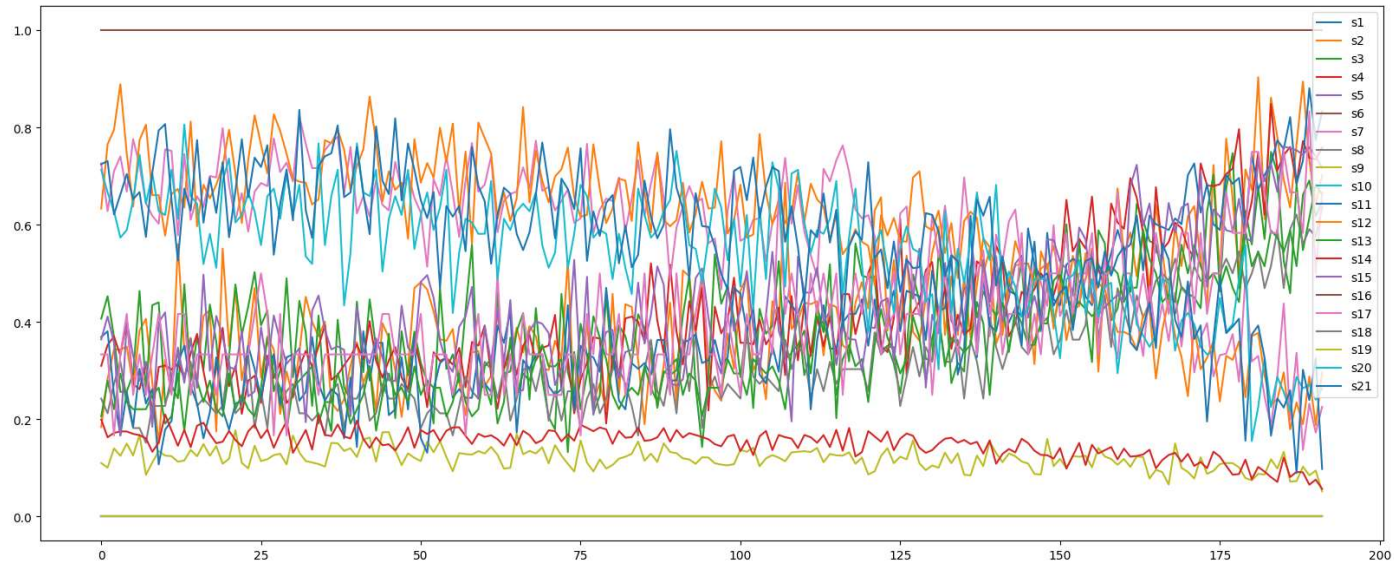
```
rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['additional_rul']

truth_df['id'] = truth_df.index + 1

truth_df['max'] = rul['max'] + truth_df['additional_rul']
truth_df.drop('additional_rul', axis=1, inplace=True)

test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)

test_df['failure_within_w1'] = np.where(test_df['RUL'] <= w1, 1, 0 )
test_df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s15	s16	s17	s18	s19	s20	s21
0	1	1	0.632184	0.750000	0.0	0.0	0.545181	0.310661	0.269413	0.0	...	0.308965	0.0	0.333333	0.0	0.0	0.558140	0.661834
1	1	2	0.344828	0.250000	0.0	0.0	0.150602	0.379551	0.222316	0.0	...	0.213159	0.0	0.416667	0.0	0.0	0.682171	0.686827

Exploratory Data Analysis - EDA

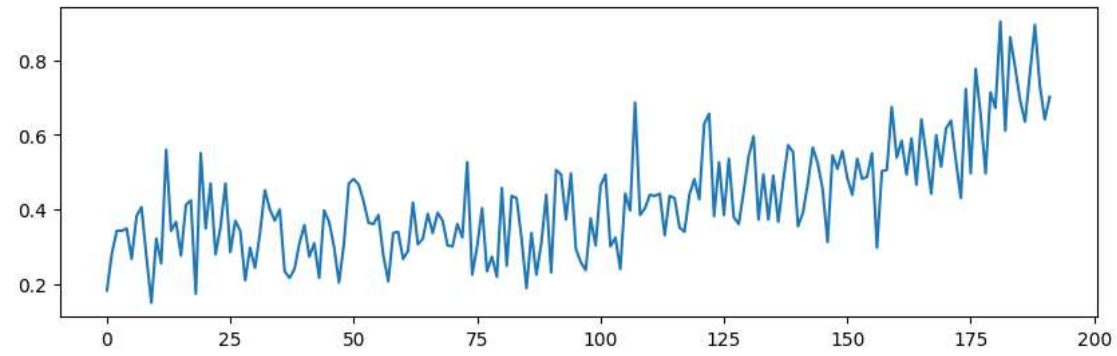
```
sensor_cols = cols_names[5:]
train_df[train_df.id==1][sensor_cols].plot(figsize=(20, 8))
```

```
<Axes: >
```




```
train_df[train_df.id==1][sensor_cols[1]].plot(figsize=(10, 3))
```

```
<Axes: >
```



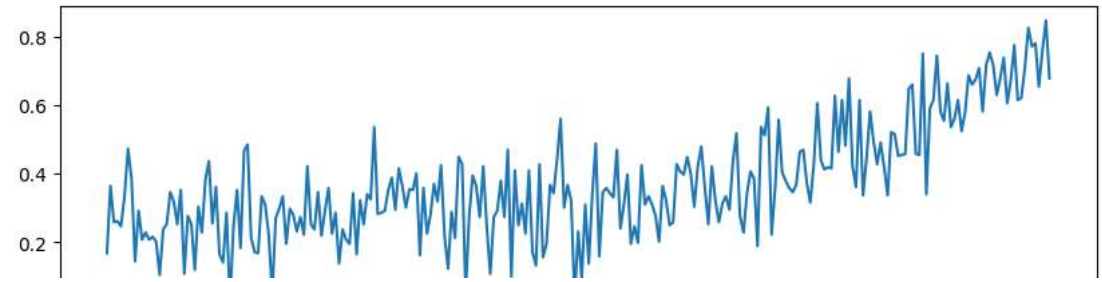
As the number cycle increases, Sensor 1 readings rise.

As the number cycle grows, Sensor 6 readings drop.

The majority of other sensors show a trend that is either rising or falling.

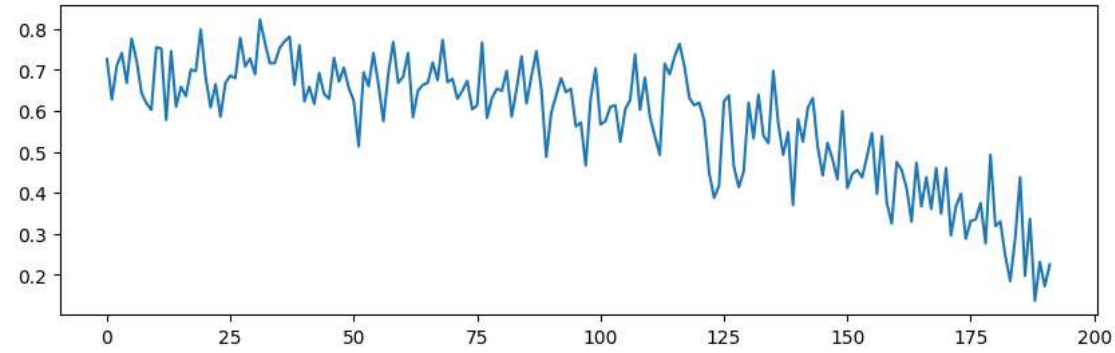
```
train_df[train_df.id==5][sensor_cols[1]].plot(figsize=(10, 3))
```

```
<Axes: >
```




```
train_df[train_df.id==1][sensor_cols[6]].plot(figsize=(10, 3))
```

```
<Axes: >
```

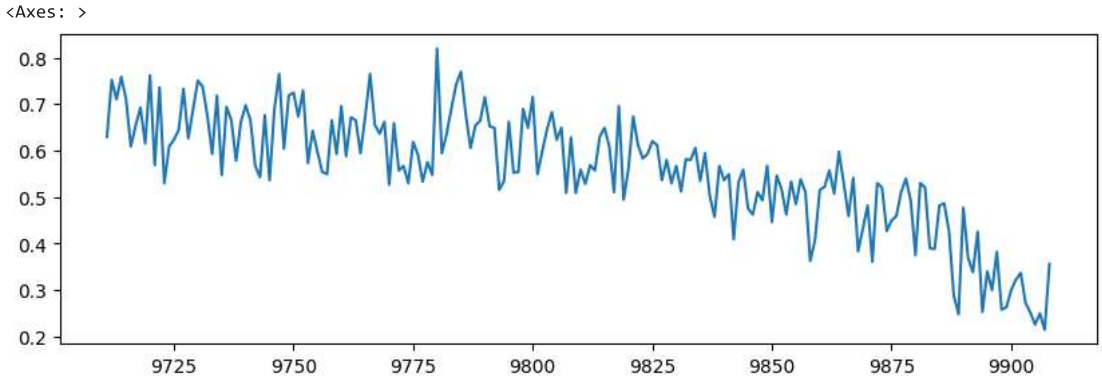


For IDs 1 and 5, we have displayed the sensor1 observations.

There is an increasing cyclic pattern in both observations.

```
train_df[train_df.id==50][sensor_cols[6]].plot(figsize=(10, 3))
```

```
<Axes: >
```



For ID 1 and ID 50, we have shown the sensor6 observations.

There is a declining tendency with more cycles in both observations.

We can infer that the aeroplane may soon stop functioning when the sensor data get close to a certain value.

Generating Input Sequence

```
sequence_length = 50
```

```
def sequence_generator(feature_df, seq_length, seq_cols):
    feature_array = feature_df[seq_cols].values
    num_elements = feature_array.shape[0]
    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield feature_array[start:stop, :]
```

```
seq_gen = (list(sequence_generator(train_df[train_df['id']==id], sequence_length, ["s2"]))
           for id in train_df['id'].unique())


seq_set = np.concatenate(list(seq_gen)).astype(np.float32)


seq_set.shape
```

```
(15631, 50, 1)
```

```
def label_generator(label_df, seq_length, label):
    label_array = label_df[label].values
    num_elements = label_array.shape[0]
    return label_array[seq_length:num_elements, :]


label_gen = [label_generator(train_df[train_df['id']==id], sequence_length, ['failure_within_w1'])
             for id in train_df['id'].unique()]


label_set = np.concatenate(label_gen).astype(np.float32)


label_set.shape
```

```
(15631, 1)
```

RNN Models

The following models will be created, trained, and assessed:

- Simple RNN [1 Feature]
- Simple RNN [25 Features]
- Bidirqactional RNN [25 Features]

1. Simple RNN [1 Feature]

```
out_dim = label_set.shape[1]
features_dim = seq_set.shape[2]


print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
```

```
Features dimension:  1
Output dimension:  1
```

```
RNN_fwd = Sequential()
RNN_fwd.add(SimpleRNN(
         input_shape=(sequence_length, features_dim),
         units=1,
         return_sequences=False))
RNN_fwd.add(Dropout(0.2))
RNN_fwd.add(Dense(units=out_dim, activation='sigmoid'))


RNN_fwd.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


print(RNN_fwd.summary())
RNN_fwd_path = '/content/RNN_fwd.h5'
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_3 (SimpleRNN)    (None, 1)                 3

 dropout_2 (Dropout)         (None, 1)                 0

 dense_2 (Dense)             (None, 1)                 2

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

```
Total params: 5 (20.00 Byte)
Trainable params: 5 (20.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

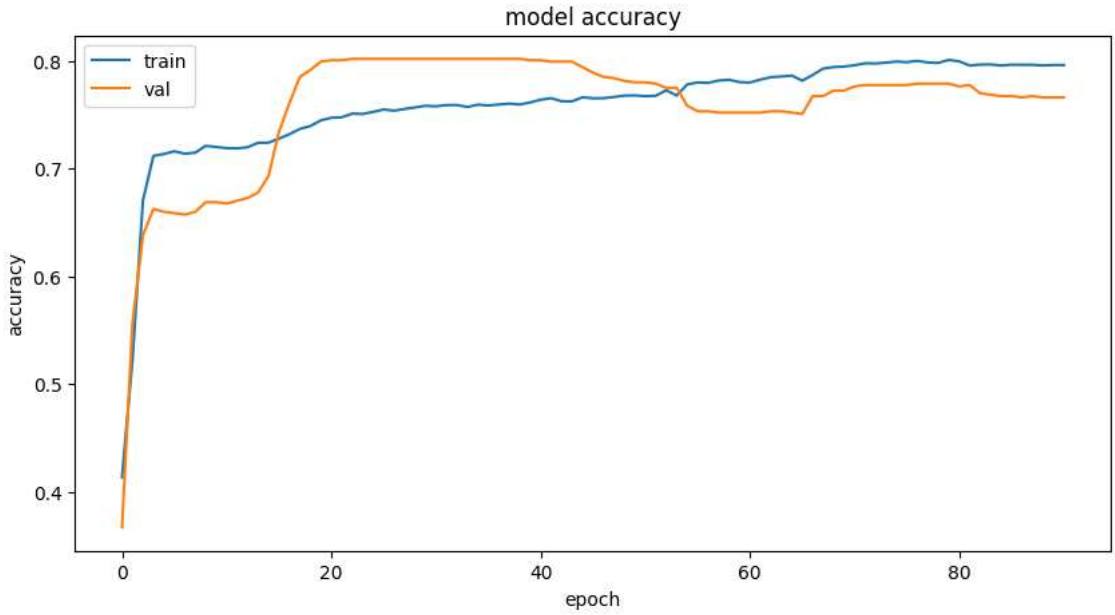
```
import time
epochs = 300
batch_size = 200
start = time.time()
RNN_fwd_history = RNN_fwd.fit(seq_set, label_set, epochs=epochs, batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(RNN_fwd_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )
end = time.time()
print("Total time taken for training: ", "{:.2f}".format((end-start)), " secs")
```

```
75/75 - 1s - loss: 0.4260 - accuracy: 0.7817 - val_loss: 0.4189 - val_accuracy: 0.7519 - 825ms/epoch - 11ms/step
Epoch 59/300
75/75 - 1s - loss: 0.4239 - accuracy: 0.7824 - val_loss: 0.4179 - val_accuracy: 0.7519 - 509ms/epoch - 7ms/step
Epoch 60/300
75/75 - 1s - loss: 0.4269 - accuracy: 0.7802 - val_loss: 0.4161 - val_accuracy: 0.7519 - 546ms/epoch - 7ms/step
Epoch 61/300
75/75 - 1s - loss: 0.4267 - accuracy: 0.7799 - val_loss: 0.4157 - val_accuracy: 0.7519 - 529ms/epoch - 7ms/step
Epoch 62/300
75/75 - 0s - loss: 0.4255 - accuracy: 0.7825 - val_loss: 0.4205 - val_accuracy: 0.7519 - 472ms/epoch - 6ms/step
Epoch 63/300
75/75 - 0s - loss: 0.4247 - accuracy: 0.7848 - val_loss: 0.4183 - val_accuracy: 0.7532 - 478ms/epoch - 6ms/step
Epoch 64/300
75/75 - 0s - loss: 0.4228 - accuracy: 0.7854 - val_loss: 0.4168 - val_accuracy: 0.7532 - 476ms/epoch - 6ms/step
Epoch 65/300
75/75 - 0s - loss: 0.4221 - accuracy: 0.7862 - val_loss: 0.4163 - val_accuracy: 0.7519 - 490ms/epoch - 7ms/step
Epoch 66/300
75/75 - 0s - loss: 0.4260 - accuracy: 0.7816 - val_loss: 0.4142 - val_accuracy: 0.7506 - 492ms/epoch - 7ms/step
Epoch 67/300
75/75 - 0s - loss: 0.4217 - accuracy: 0.7869 - val_loss: 0.4112 - val_accuracy: 0.7673 - 499ms/epoch - 7ms/step
Epoch 68/300
75/75 - 0s - loss: 0.4198 - accuracy: 0.7928 - val_loss: 0.4100 - val_accuracy: 0.7673 - 481ms/epoch - 6ms/step
Epoch 69/300
75/75 - 0s - loss: 0.4183 - accuracy: 0.7942 - val_loss: 0.4089 - val_accuracy: 0.7724 - 493ms/epoch - 7ms/step
Epoch 70/300
75/75 - 0s - loss: 0.4171 - accuracy: 0.7947 - val_loss: 0.4081 - val_accuracy: 0.7724 - 489ms/epoch - 7ms/step
Epoch 71/300
75/75 - 0s - loss: 0.4190 - accuracy: 0.7959 - val_loss: 0.4086 - val_accuracy: 0.7762 - 479ms/epoch - 6ms/step
Epoch 72/300
75/75 - 0s - loss: 0.4175 - accuracy: 0.7977 - val_loss: 0.4074 - val_accuracy: 0.7775 - 480ms/epoch - 6ms/step
Epoch 73/300
75/75 - 1s - loss: 0.4182 - accuracy: 0.7975 - val_loss: 0.4065 - val_accuracy: 0.7775 - 501ms/epoch - 7ms/step
Epoch 74/300
75/75 - 0s - loss: 0.4173 - accuracy: 0.7983 - val_loss: 0.4057 - val_accuracy: 0.7775 - 496ms/epoch - 7ms/step
Epoch 75/300
75/75 - 0s - loss: 0.4159 - accuracy: 0.7993 - val_loss: 0.4051 - val_accuracy: 0.7775 - 492ms/epoch - 7ms/step
Epoch 76/300
75/75 - 0s - loss: 0.4171 - accuracy: 0.7987 - val_loss: 0.4050 - val_accuracy: 0.7775 - 497ms/epoch - 7ms/step
Epoch 77/300
75/75 - 1s - loss: 0.4155 - accuracy: 0.7999 - val_loss: 0.4049 - val_accuracy: 0.7788 - 524ms/epoch - 7ms/step
Epoch 78/300
75/75 - 1s - loss: 0.4175 - accuracy: 0.7984 - val_loss: 0.4045 - val_accuracy: 0.7788 - 608ms/epoch - 8ms/step
Epoch 79/300
75/75 - 1s - loss: 0.4178 - accuracy: 0.7981 - val_loss: 0.4045 - val_accuracy: 0.7788 - 903ms/epoch - 12ms/step
Epoch 80/300
75/75 - 1s - loss: 0.4160 - accuracy: 0.8008 - val_loss: 0.4041 - val_accuracy: 0.7788 - 860ms/epoch - 11ms/step
Epoch 81/300
75/75 - 1s - loss: 0.4158 - accuracy: 0.7994 - val_loss: 0.4038 - val_accuracy: 0.7762 - 827ms/epoch - 11ms/step
Epoch 82/300
75/75 - 0s - loss: 0.4202 - accuracy: 0.7956 - val_loss: 0.4062 - val_accuracy: 0.7775 - 473ms/epoch - 6ms/step
Epoch 83/300
75/75 - 0s - loss: 0.4175 - accuracy: 0.7965 - val_loss: 0.4075 - val_accuracy: 0.7698 - 496ms/epoch - 7ms/step
Epoch 84/300
75/75 - 0s - loss: 0.4160 - accuracy: 0.7966 - val_loss: 0.4088 - val_accuracy: 0.7685 - 484ms/epoch - 6ms/step
Epoch 85/300
75/75 - 0s - loss: 0.4172 - accuracy: 0.7956 - val_loss: 0.4093 - val_accuracy: 0.7673 - 495ms/epoch - 7ms/step
Epoch 86/300
75/75 - 1s - loss: 0.4171 - accuracy: 0.7963 - val_loss: 0.4091 - val_accuracy: 0.7673 - 528ms/epoch - 7ms/step
```

Assessment of the Train and Validation Sets Models

```
def plot_model_accuracy(model_name_history, width = 10, height = 10):
  fig_acc = plt.figure(figsize=(width, height))
```

```
  plt.plot(model_name_history.history['accuracy'])
  plt.plot(model_name_history.history['val_accuracy'])
  plt.title('model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='upper left')
  plt.show()


plot_model_accuracy(RNN_fwd_history,10,5)
```



Training Curve

```
def plot_training_curve(model_name_history, width = 10, height = 10):
  fig_acc = plt.figure(figsize=(width, height))
  plt.plot(model_name_history.history['loss'])
  plt.plot(model_name_history.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='upper left')
  plt.show()


plot_training_curve(RNN_fwd_history,10,5)
```

Model Evaluation on the Train Set

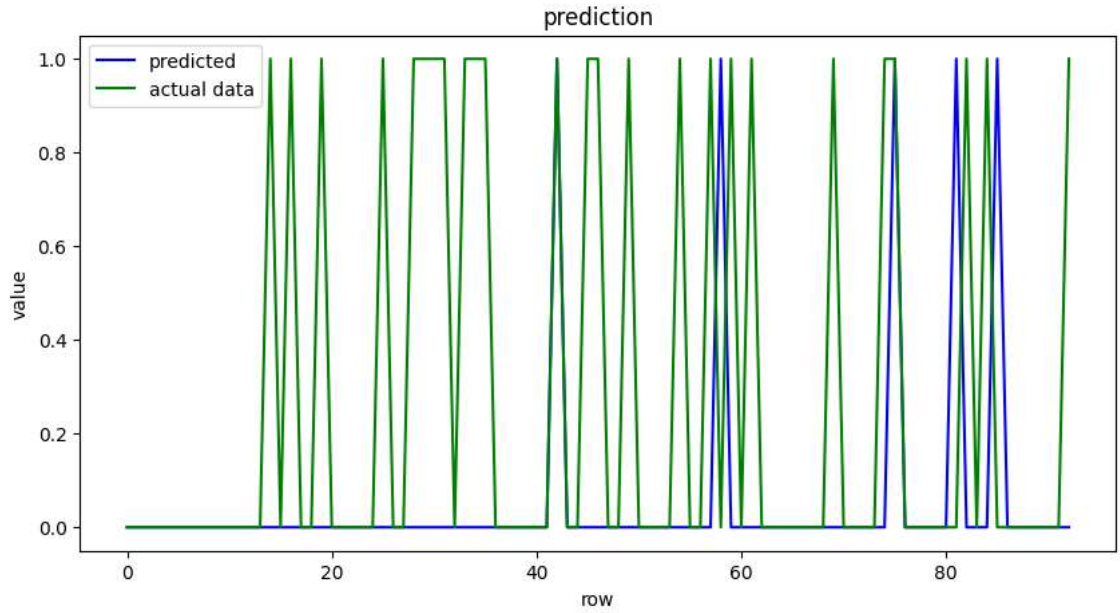
```
def analyze_model_on_train_set(input_sequence_set, model_name):
  model_history_scores = model_name.evaluate(input_sequence_set, label_set, verbose=1, batch_size=50)
  print('\nTrain Accurracy: {}'.format(model_history_scores[1]))
  print('\n')
  y_pred = (model_name.predict(input_sequence_set, verbose=1, batch_size=200) > 0.5).astype("int32")
  y_true = label_set
  test_set = pd.DataFrame(y_pred)
  test_set.to_csv('binary_submit_train.csv', index = None)
  print('\nConfusion matrix\n- x-axis is true labels.\n- y-axis is predicted labels')
  model_cm = confusion_matrix(y_true, y_pred)
  print(model_cm)
  model_precision = precision_score(y_true, y_pred)
  model_recall = recall_score(y_true, y_pred)
  print( '\nTrain Precision = ', model_precision, '\n', 'Train Recall = ', model_recall)

analyze_model_on_train_set(seq_set, RNN_fwd)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.4011 - accuracy: 0.7836

Train Accurracy: 0.7835711240768433


79/79 [==============================] - 0s 3ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[11630   901]
 [ 2482   618]]

Train Precision =  0.4068466096115866
 Train Recall =  0.1993548387096774
```

Model evaluation on test set

```
def analyze_model_on_test_set(input_sequence_columns, model_path, width= 10, height=5):
  last_test_seq = [test_df[test_df['id']==id][input_sequence_columns].values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df[test_df['id']==id]) >= sequence_length]
  last_test_seq = np.asarray(last_test_seq).astype(np.float32)
  y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
  last_test_label = test_df.groupby('id')['failure_within_w1'].nth(-1)[y_mask].values
  last_test_label = last_test_label.reshape(last_test_label.shape[0],1).astype(np.float32)
  if os.path.isfile(model_path):
    print("using " + model_path)
    model_estimator = load_model(model_path)
  start = time.time()
  scores_test = model_estimator.evaluate(last_test_seq, last_test_label, verbose=2)
  end = time.time()
  print("Total time taken for inferencing: ", "{:.2f}".format((end-start)), " secs")
  print('\nTest Accurracy: {}'.format(scores_test[1]))
  print('\n')
  y_model_estimator_pred_test = (model_estimator.predict(last_test_seq) >0.5).astype("int32")
  y_true_test = last_test_label
  test_set = pd.DataFrame(y_model_estimator_pred_test)
  test_set.to_csv('binary_submit_test.csv', index = None)
  print('\nConfusion matrix\n- x-axis is true labels.\n- y-axis is predicted labels')
  model_estimator_conf_m = confusion_matrix(y_true_test, y_model_estimator_pred_test)
  print(model_estimator_conf_m)
  model_estimator_precision_test = precision_score(y_true_test, y_model_estimator_pred_test)
  model_estimator_recall_test = recall_score(y_true_test, y_model_estimator_pred_test)
  f1_test = 2 * (model_estimator_precision_test * model_estimator_recall_test) / (model_estimator_precision_test + model_estimator_recall_tes
  print( '\nTest Precision: ', model_estimator_precision_test, '\n', 'Test Recall: ', model_estimator_recall_test,'\n', 'Test F1-score:', f1_
  fig_verify = plt.figure(figsize=(10, 5))
  plt.plot(y_model_estimator_pred_test, color="blue")
  plt.plot(y_true_test, color="green")
  plt.title('prediction')
  plt.ylabel('value')
  plt.xlabel('row')
  plt.legend(['predicted', 'actual data'], loc='upper left')
  plt.show()

analyze_model_on_test_set(["s2"], RNN_fwd_path, 10, 5)
```

```
using /content/RNN_fwd.h5
3/3 - 0s - loss: 0.4916 - accuracy: 0.7204 - 357ms/epoch - 119ms/step
Total time taken for inferencing:  0.45  secs

Test Accurracy: 0.7204301357269287


3/3 [==============================] - 0s 8ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[65  3]
 [23  2]]

Test Precision:  0.4
 Test Recall:  0.08
 Test F1-score: 0.13333333333333333
```



2. Simple RNN With 25 Features

```
sensor_cols = ['s' + str(i) for i in range(1,22)]
sequence_cols_25 = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols_25.extend(sensor_cols)


seq_gen = (list(sequence_generator(train_df[train_df['id']==id], sequence_length, sequence_cols_25))
           for id in train_df['id'].unique())


seq_set_f25 = np.concatenate(list(seq_gen)).astype(np.float32)


seq_set_f25.shape
```

```
(15631, 50, 25)
```

```
features_dim = seq_set_f25.shape[2]
out_dim = label_set.shape[1]


print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
```

```
Features dimension:  25
Output dimension:  1
```

```
RNN_fwd_2 = Sequential()
RNN_fwd_2.add(SimpleRNN(
         input_shape=(sequence_length, features_dim),
         units=5,
```

```
          return_sequences=True))
RNN_fwd_2.add(Dropout(0.2))


RNN_fwd_2.add(SimpleRNN(
          units=3,
          return_sequences=False))
RNN_fwd_2.add(Dropout(0.2))


RNN_fwd_2.add(Dense(units=out_dim, activation='sigmoid'))


RNN_fwd_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


print(RNN_fwd_2.summary())
```

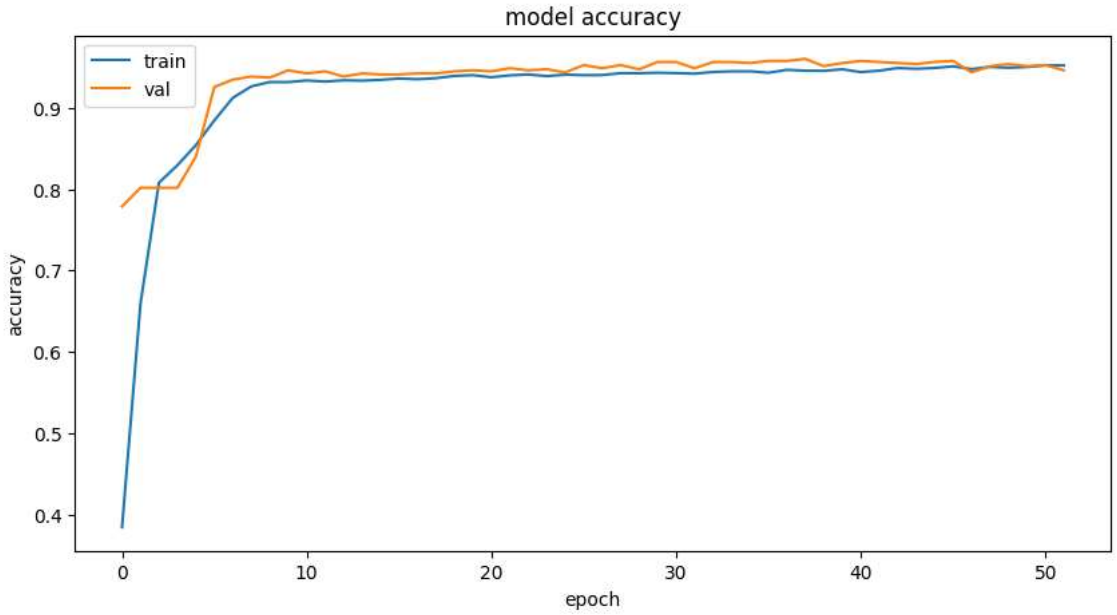
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_4 (SimpleRNN)    (None, 50, 5)             155

 dropout_3 (Dropout)         (None, 50, 5)             0

 simple_rnn_5 (SimpleRNN)    (None, 3)                 27

 dropout_4 (Dropout)         (None, 3)                 0

 dense_3 (Dense)             (None, 1)                 4

=================================================================
Total params: 186 (744.00 Byte)
Trainable params: 186 (744.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
RNN_fwd_2_path = '/content/RNN_fwd_2.h5'


import time
epochs = 200
batch_size = 200
start = time.time()
RNN_fwd_2_history = RNN_fwd_2.fit(seq_set_f25, label_set, epochs=epochs, batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(RNN_fwd_2_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )
end = time.time()
print("Total time taken for training: ", "{:.2f}".format((end-start)), " secs")
```

```
Epoch 40/200
75/75 - 2s - loss: 0.1470 - accuracy: 0.9475 - val_loss: 0.1180 - val_accuracy: 0.9552 - 2s/epoch - 26ms/step
Epoch 41/200
75/75 - 1s - loss: 0.1492 - accuracy: 0.9441 - val_loss: 0.1110 - val_accuracy: 0.9578 - 1s/epoch - 17ms/step
Epoch 42/200
75/75 - 1s - loss: 0.1464 - accuracy: 0.9459 - val_loss: 0.1069 - val_accuracy: 0.9565 - 1s/epoch - 18ms/step
Epoch 43/200
75/75 - 1s - loss: 0.1442 - accuracy: 0.9491 - val_loss: 0.1229 - val_accuracy: 0.9552 - 1s/epoch - 18ms/step
Epoch 44/200
75/75 - 1s - loss: 0.1452 - accuracy: 0.9481 - val_loss: 0.1142 - val_accuracy: 0.9540 - 1s/epoch - 17ms/step
Epoch 45/200
75/75 - 1s - loss: 0.1434 - accuracy: 0.9492 - val_loss: 0.1157 - val_accuracy: 0.9565 - 1s/epoch - 17ms/step
Epoch 46/200
75/75 - 1s - loss: 0.1384 - accuracy: 0.9512 - val_loss: 0.1152 - val_accuracy: 0.9578 - 1s/epoch - 17ms/step
Epoch 47/200
75/75 - 1s - loss: 0.1419 - accuracy: 0.9477 - val_loss: 0.1356 - val_accuracy: 0.9437 - 1s/epoch - 17ms/step
Epoch 48/200
75/75 - 2s - loss: 0.1386 - accuracy: 0.9505 - val_loss: 0.1314 - val_accuracy: 0.9514 - 2s/epoch - 29ms/step
Epoch 49/200
75/75 - 2s - loss: 0.1382 - accuracy: 0.9494 - val_loss: 0.1143 - val_accuracy: 0.9540 - 2s/epoch - 22ms/step
Epoch 50/200
75/75 - 1s - loss: 0.1364 - accuracy: 0.9504 - val_loss: 0.1236 - val_accuracy: 0.9514 - 1s/epoch - 17ms/step
Epoch 51/200
75/75 - 1s - loss: 0.1359 - accuracy: 0.9524 - val_loss: 0.1166 - val_accuracy: 0.9527 - 1s/epoch - 17ms/step
Epoch 52/200
75/75 - 1s - loss: 0.1354 - accuracy: 0.9523 - val_loss: 0.1436 - val_accuracy: 0.9463 - 1s/epoch - 17ms/step
Total time taken for training:  80.36  secs
```

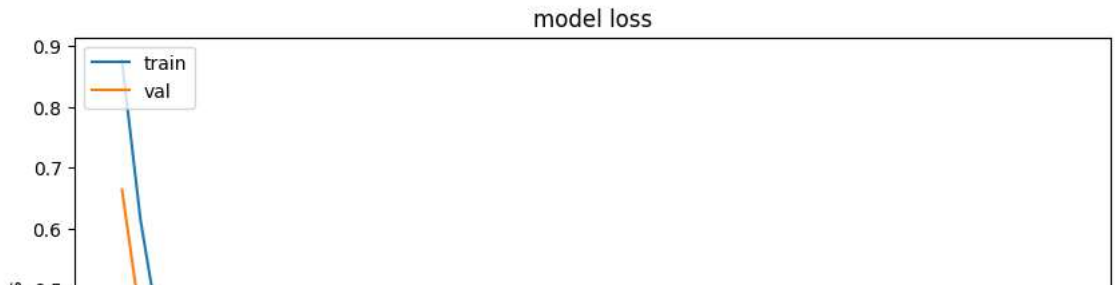
Plot Model Accuracy for the Train and Validation Sets

```
plot_model_accuracy(RNN_fwd_2_history, 10, 5)
```



Training Curve

```
plot_training_curve(RNN_fwd_2_history,10,5)
```



Model Evaluation on Train Set

```
analyze_model_on_train_set(seq_set_f25, RNN_fwd_2)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.1166 - accuracy: 0.9583

Train Accurracy: 0.9582880139350891


79/79 [==============================] - 1s 6ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12420   111]
 [  541  2559]]

Train Precision =  0.9584269662921349
 Train Recall =  0.8254838709677419
```

Model Evaluation on Test Set

```
analyze_model_on_test_set(sequence_cols_25, RNN_fwd_2_path, 10, 5)
```

```
using /content/RNN_fwd_2.h5
3/3 - 1s - loss: 0.2053 - accuracy: 0.9462 - 577ms/epoch - 192ms/step
```

3. Bidirectional RNN

```
from tensorflow.keras.layers import Bidirectional
```

```
features_dim = seq_set_f25.shape[2]
out_dim = label_set.shape[1]
```

```
print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
```

```
Features dimension:  25
Output dimension:  1
```

```
RNN_bi = Sequential()
```

```
RNN_bi.add(Bidirectional(
        SimpleRNN(
        input_shape=(sequence_length, features_dim),
        units=6,
        return_sequences=True)))
RNN_bi.add(Dropout(0.2))
```

```
RNN_bi.add(SimpleRNN(
         units=3,
         return_sequences=False))
RNN_bi.add(Dropout(0.2))
```

```
RNN_bi.add(Dense(units=out_dim, activation='sigmoid'))
```

```
RNN_bi.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
RNN_bi_path = '/content/RNN_bi.h5'
```

```
import time
epochs = 200
batch_size = 200
start = time.time()
RNN_bi_history = RNN_bi.fit(seq_set_f25, label_set, epochs=epochs, batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(RNN_bi_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )
end = time.time()
print("Total time taken for training: ", "{:.2f}".format((end-start)), " secs")
```

```
Epoch 7/200
75/75 - 3s - loss: 0.2193 - accuracy: 0.9362 - val_loss: 0.2013 - val_accuracy: 0.9399 - 3s/epoch - 36ms/step
Epoch 8/200
75/75 - 2s - loss: 0.2022 - accuracy: 0.9415 - val_loss: 0.1873 - val_accuracy: 0.9450 - 2s/epoch - 24ms/step
Epoch 9/200
75/75 - 2s - loss: 0.1915 - accuracy: 0.9419 - val_loss: 0.1805 - val_accuracy: 0.9476 - 2s/epoch - 24ms/step
Epoch 10/200
75/75 - 2s - loss: 0.1834 - accuracy: 0.9440 - val_loss: 0.1762 - val_accuracy: 0.9501 - 2s/epoch - 24ms/step
Epoch 11/200
75/75 - 2s - loss: 0.1780 - accuracy: 0.9455 - val_loss: 0.1590 - val_accuracy: 0.9527 - 2s/epoch - 24ms/step
Epoch 12/200
```

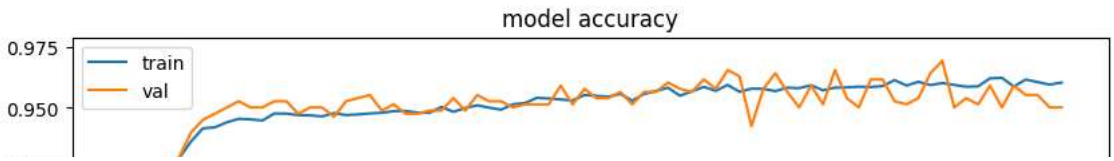
```
75/75 - 3s - loss: 0.1574 - accuracy: 0.9478 - val_loss: 0.1753 - val_accuracy: 0.9463 - 3s/epoch - 41ms/step
Epoch 20/200
75/75 - 2s - loss: 0.1574 - accuracy: 0.9470 - val_loss: 0.1447 - val_accuracy: 0.9527 - 2s/epoch - 24ms/step
Epoch 21/200
75/75 - 2s - loss: 0.1563 - accuracy: 0.9473 - val_loss: 0.1434 - val_accuracy: 0.9540 - 2s/epoch - 24ms/step
Epoch 22/200
75/75 - 2s - loss: 0.1555 - accuracy: 0.9477 - val_loss: 0.1342 - val_accuracy: 0.9552 - 2s/epoch - 25ms/step
Epoch 23/200
75/75 - 2s - loss: 0.1513 - accuracy: 0.9479 - val_loss: 0.1542 - val_accuracy: 0.9488 - 2s/epoch - 24ms/step
Epoch 24/200
75/75 - 2s - loss: 0.1548 - accuracy: 0.9486 - val_loss: 0.1482 - val_accuracy: 0.9514 - 2s/epoch - 24ms/step
Epoch 25/200
75/75 - 3s - loss: 0.1501 - accuracy: 0.9487 - val_loss: 0.1544 - val_accuracy: 0.9476 - 3s/epoch - 38ms/step
Epoch 26/200
75/75 - 2s - loss: 0.1539 - accuracy: 0.9480 - val_loss: 0.1557 - val_accuracy: 0.9476 - 2s/epoch - 27ms/step
Epoch 27/200
75/75 - 2s - loss: 0.1493 - accuracy: 0.9479 - val_loss: 0.1551 - val_accuracy: 0.9488 - 2s/epoch - 24ms/step
Epoch 28/200
75/75 - 2s - loss: 0.1437 - accuracy: 0.9504 - val_loss: 0.1411 - val_accuracy: 0.9488 - 2s/epoch - 23ms/step
Epoch 29/200
75/75 - 2s - loss: 0.1475 - accuracy: 0.9483 - val_loss: 0.1264 - val_accuracy: 0.9540 - 2s/epoch - 24ms/step
Epoch 30/200
75/75 - 2s - loss: 0.1437 - accuracy: 0.9499 - val_loss: 0.1371 - val_accuracy: 0.9488 - 2s/epoch - 25ms/step
Epoch 31/200
75/75 - 2s - loss: 0.1427 - accuracy: 0.9510 - val_loss: 0.1223 - val_accuracy: 0.9552 - 2s/epoch - 31ms/step
Epoch 32/200
75/75 - 3s - loss: 0.1461 - accuracy: 0.9500 - val_loss: 0.1179 - val_accuracy: 0.9527 - 3s/epoch - 35ms/step
Epoch 33/200
75/75 - 2s - loss: 0.1450 - accuracy: 0.9492 - val_loss: 0.1290 - val_accuracy: 0.9527 - 2s/epoch - 33ms/step
Epoch 34/200
75/75 - 2s - loss: 0.1412 - accuracy: 0.9514 - val_loss: 0.1309 - val_accuracy: 0.9501 - 2s/epoch - 24ms/step
Epoch 35/200
```

```
print(RNN_bi.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirection  (None, 50, 12)            384
 al)

 dropout_5 (Dropout)         (None, 50, 12)            0

 simple_rnn_7 (SimpleRNN)    (None, 3)                 48

 dropout_6 (Dropout)         (None, 3)                 0

 dense_4 (Dense)             (None, 1)                 4

=================================================================
Total params: 436 (1.70 KB)
Trainable params: 436 (1.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

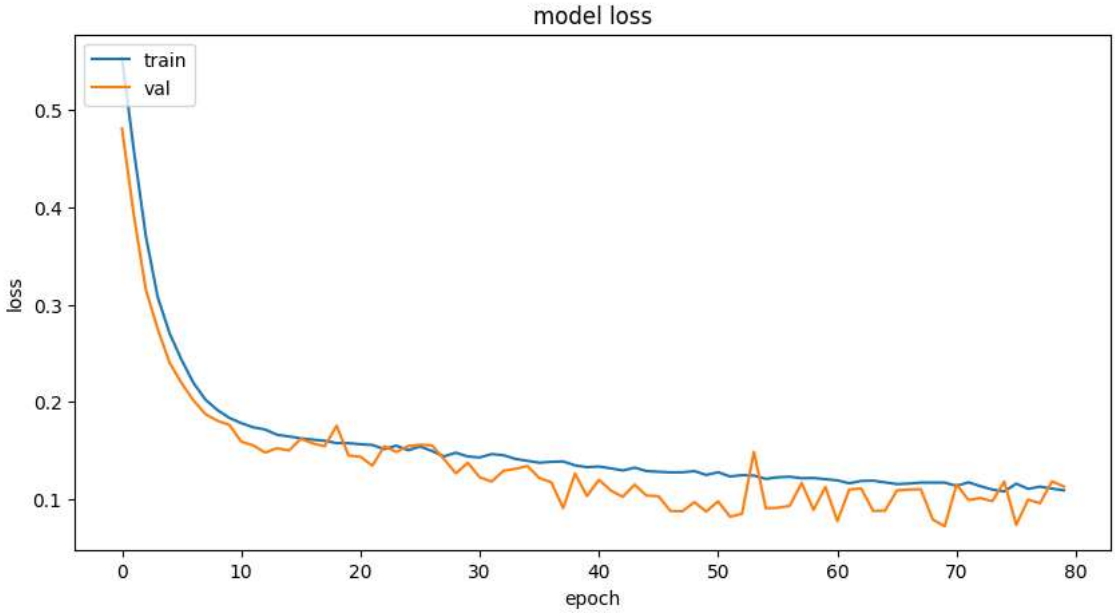
Plot model accuracy for train and validation set

```
plot_model_accuracy(RNN_bi_history,10,5)
```



Training curve

```
plot_training_curve(RNN_bi_history,10,5)
```



Model Evaluation on train set

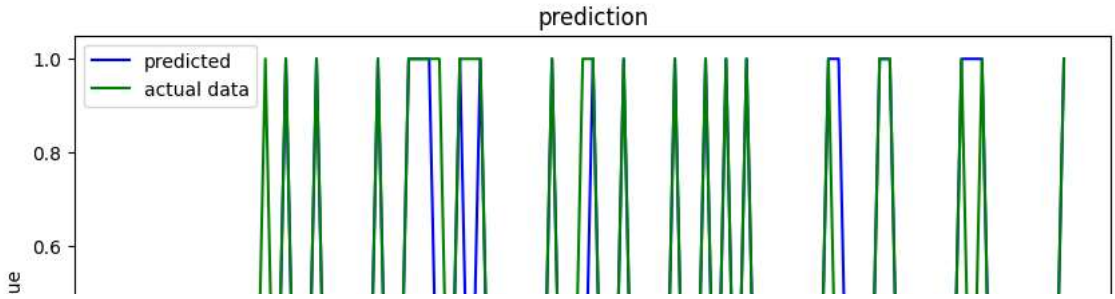
```
analyze_model_on_train_set(seq_set_f25, RNN_bi)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0928 - accuracy: 0.9666

Train Accurracy: 0.9666048288345337


79/79 [==============================] - 1s 9ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12409   122]
 [  400  2700]]

Train Precision =  0.9567682494684621
 Train Recall =  0.8709677419354839
```

Model Evaluation on test set

```
analyze_model_on_test_set(sequence_cols_25, RNN_bi_path,10,5)
```

```
using /content/RNN_bi.h5
3/3 - 1s - loss: 0.1362 - accuracy: 0.9355 - 681ms/epoch - 227ms/step
Total time taken for inferencing:  0.76  secs

Test Accurracy: 0.9354838728904724


3/3 [==============================] - 1s 7ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[66  2]
 [ 4 21]]

Test Precision:  0.9130434782608695
 Test Recall:  0.84
 Test F1-score: 0.8749999999999999
```



Overview of RNN

- An RNN with more complexity performs better on the test set than one with only one feature.
- More units in the RNN model with all 25 characteristics result in improved accuracy and other assessment measures.
- Return_sequences = True must be set in the preceding layer when using several layers.

LSTM Model

```
features_dim = seq_set_f25.shape[2]
out_dim = label_set.shape[1]


print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)
```

```
Features dimension:  25
Output dimension:  1
```

```
model = Sequential()


model.add(LSTM(
         input_shape=(sequence_length, features_dim),
         units=100,
         return_sequences=True))
model.add(Dropout(0.2))


model.add(LSTM(
          units=50,
          return_sequences=False))
model.add(Dropout(0.2))


model.add(Dense(units=out_dim, activation='sigmoid'))


model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


print(model.summary())
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 lstm (LSTM)                 (None, 50, 100)           50400

 dropout_7 (Dropout)         (None, 50, 100)           0

 lstm_1 (LSTM)               (None, 50)                30200

 dropout_8 (Dropout)         (None, 50)                0

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 80651 (315.04 KB)
Trainable params: 80651 (315.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

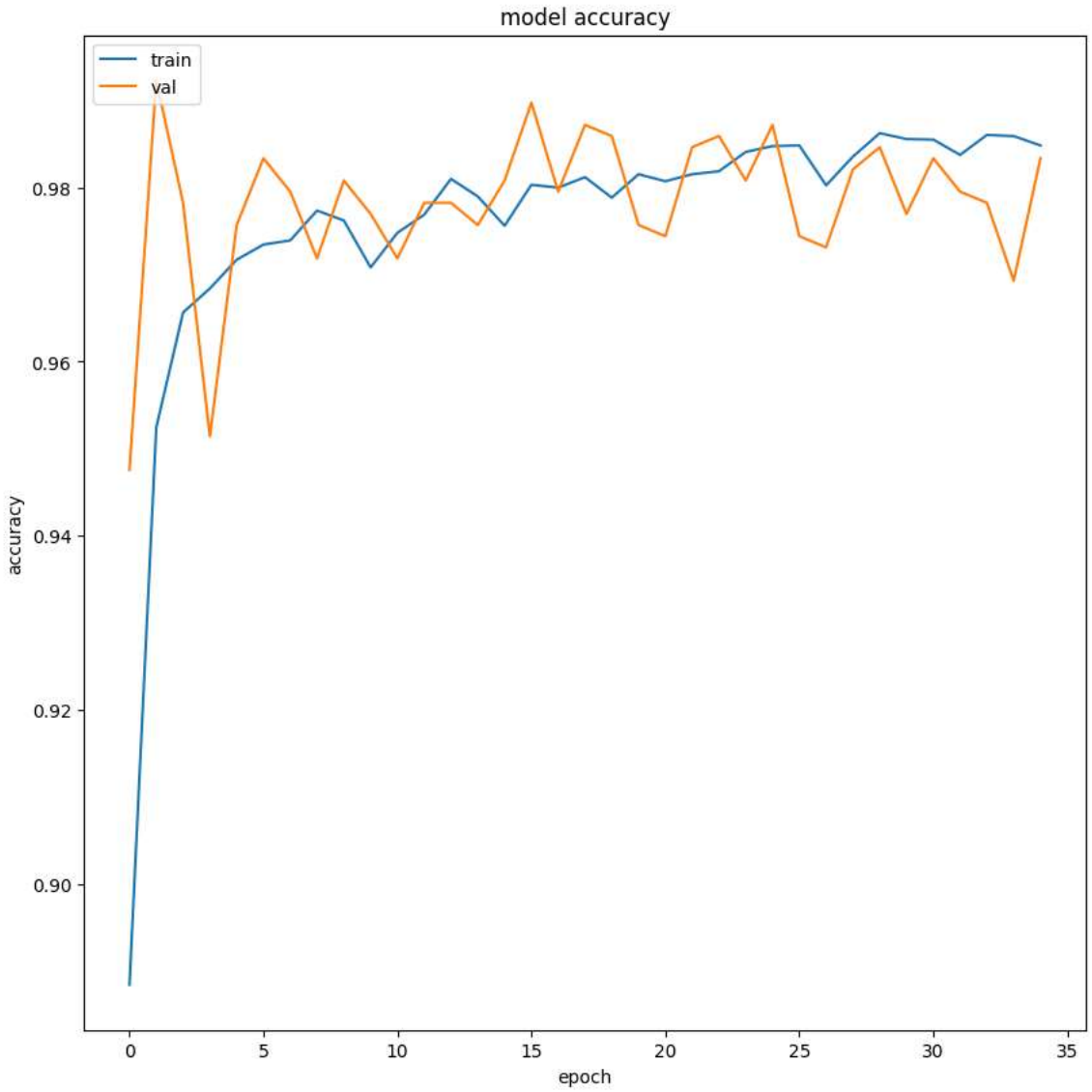
```
model_path = '/content/binary_model.h5'


import time
epochs = 200
batch_size = 200
start = time.time()
history = model.fit(seq_set_f25, label_set, epochs=epochs, batch_size=batch_size, validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
          )
end = time.time()
print("Total time taken for training: ", "{:.2f}".format((end-start)), " secs")
```

```
Epoch 8/200
75/75 - 16s - loss: 0.0555 - accuracy: 0.9774 - val_loss: 0.0606 - val_accuracy: 0.9719 - 16s/epoch - 217ms/step
Epoch 9/200
75/75 - 17s - loss: 0.0583 - accuracy: 0.9762 - val_loss: 0.0414 - val_accuracy: 0.9808 - 17s/epoch - 232ms/step
Epoch 10/200
75/75 - 16s - loss: 0.0666 - accuracy: 0.9708 - val_loss: 0.0541 - val_accuracy: 0.9770 - 16s/epoch - 218ms/step
Epoch 11/200
75/75 - 16s - loss: 0.0636 - accuracy: 0.9748 - val_loss: 0.0566 - val_accuracy: 0.9719 - 16s/epoch - 217ms/step
Epoch 12/200
75/75 - 18s - loss: 0.0536 - accuracy: 0.9769 - val_loss: 0.0480 - val_accuracy: 0.9783 - 18s/epoch - 233ms/step
Epoch 13/200
75/75 - 16s - loss: 0.0473 - accuracy: 0.9810 - val_loss: 0.0442 - val_accuracy: 0.9783 - 16s/epoch - 219ms/step
Epoch 14/200
75/75 - 16s - loss: 0.0519 - accuracy: 0.9790 - val_loss: 0.0463 - val_accuracy: 0.9757 - 16s/epoch - 216ms/step
Epoch 15/200
75/75 - 17s - loss: 0.0560 - accuracy: 0.9756 - val_loss: 0.0407 - val_accuracy: 0.9808 - 17s/epoch - 232ms/step
Epoch 16/200
75/75 - 16s - loss: 0.0441 - accuracy: 0.9803 - val_loss: 0.0278 - val_accuracy: 0.9898 - 16s/epoch - 216ms/step
Epoch 17/200
75/75 - 16s - loss: 0.0494 - accuracy: 0.9800 - val_loss: 0.0497 - val_accuracy: 0.9795 - 16s/epoch - 220ms/step
Epoch 18/200
75/75 - 18s - loss: 0.0455 - accuracy: 0.9812 - val_loss: 0.0295 - val_accuracy: 0.9872 - 18s/epoch - 241ms/step
Epoch 19/200
75/75 - 27s - loss: 0.0486 - accuracy: 0.9789 - val_loss: 0.0354 - val_accuracy: 0.9859 - 27s/epoch - 357ms/step
Epoch 20/200
75/75 - 29s - loss: 0.0455 - accuracy: 0.9815 - val_loss: 0.0480 - val_accuracy: 0.9757 - 29s/epoch - 391ms/step
Epoch 21/200
75/75 - 24s - loss: 0.0472 - accuracy: 0.9807 - val_loss: 0.0498 - val_accuracy: 0.9744 - 24s/epoch - 324ms/step
Epoch 22/200
75/75 - 26s - loss: 0.0426 - accuracy: 0.9815 - val_loss: 0.0342 - val_accuracy: 0.9847 - 26s/epoch - 342ms/step
Epoch 23/200
75/75 - 22s - loss: 0.0421 - accuracy: 0.9819 - val_loss: 0.0324 - val_accuracy: 0.9859 - 22s/epoch - 295ms/step
Epoch 24/200
75/75 - 22s - loss: 0.0379 - accuracy: 0.9841 - val_loss: 0.0368 - val_accuracy: 0.9808 - 22s/epoch - 291ms/step
Epoch 25/200
```

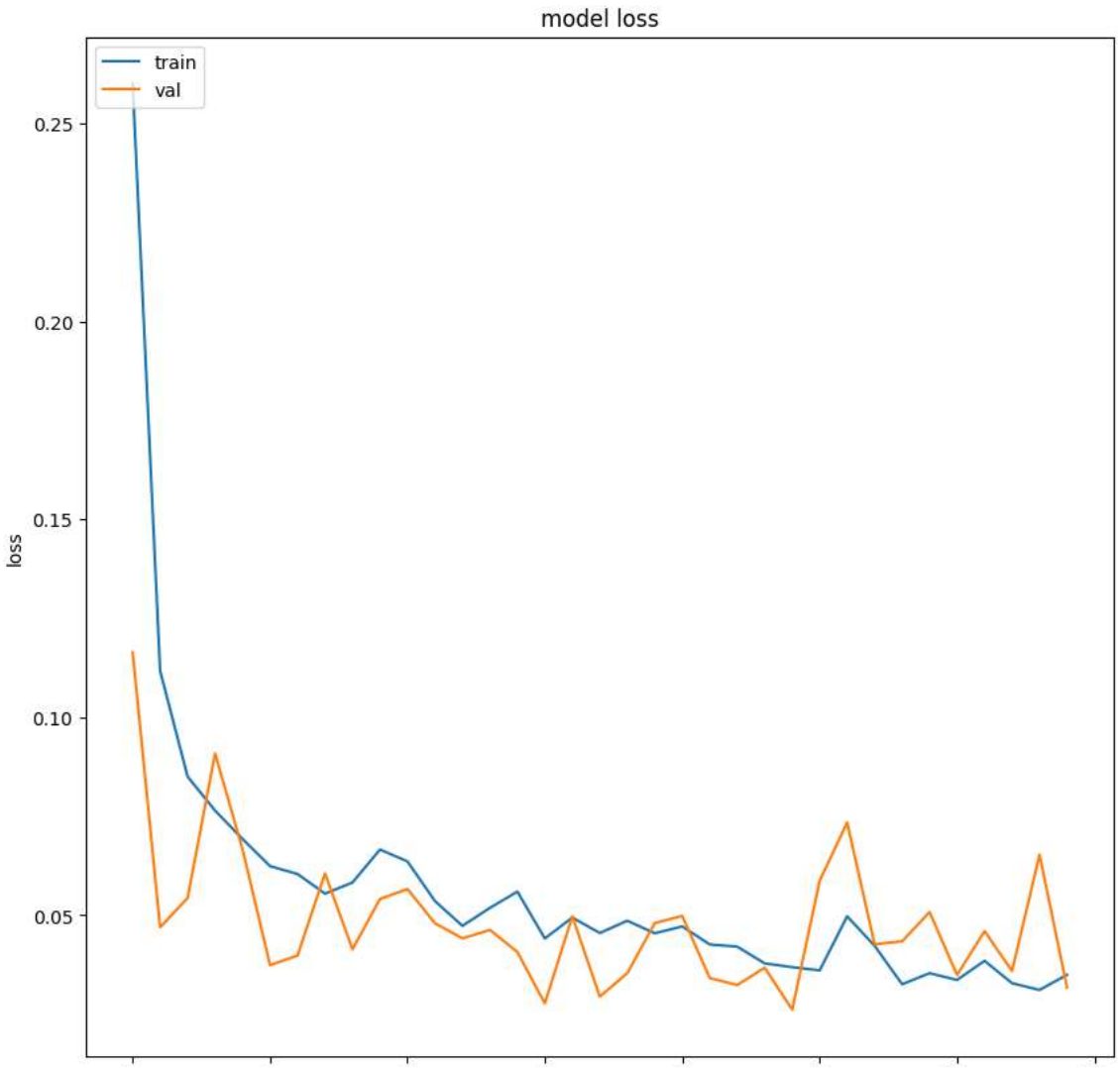
```
75/75 - 16s - loss: 0.0328 - accuracy: 0.9861 - val_loss: 0.0339 - val_accuracy: 0.9785 - 16s/epoch - 213ms/step
Epoch 34/200
75/75 - 16s - loss: 0.0311 - accuracy: 0.9859 - val_loss: 0.0653 - val_accuracy: 0.9693 - 16s/epoch - 213ms/step
Epoch 35/200
75/75 - 19s - loss: 0.0350 - accuracy: 0.9848 - val_loss: 0.0318 - val_accuracy: 0.9834 - 19s/epoch - 247ms/step
Total time taken for training:  653.40  secs
```

Model Evaluation on Validation set

```
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
scores = model.evaluate(seq_set_f25, label_set, verbose=1, batch_size=50)
print('Train Accurracy: {}'.format(scores[1]))
print('\n')
y_pred = (model.predict(seq_set_f25, verbose=1, batch_size=200) > 0.5).astype("int32")
y_true = label_set
test_set = pd.DataFrame(y_pred)
test_set.to_csv('binary_submit_train.csv', index = None)
print('\nConfusion matrix\n- x-axis is true labels.\n- y-axis is predicted labels')
cm = confusion_matrix(y_true, y_pred)
print(cm)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
print( '\nTrain Precision = ', precision, '\n', 'Train Recall = ', recall)
```

```
313/313 [==============================] - 10s 33ms/step - loss: 0.0292 - accuracy: 0.9876
Train Accurracy: 0.9875887632369995


79/79 [==============================] - 7s 81ms/step

Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[12476    55]
 [  139  2961]]

Train Precision =  0.9817639257294429
 Train Recall =  0.9551612903225807
```

Model Evaluation on Test set

```
last_test_seq = [test_df[test_df['id']==id][sequence_cols_25].values[-sequence_length:]
                 for id in test_df['id'].unique() if len(test_df[test_df['id']==id]) >= sequence_length]
```

```
last_test_seq = np.asarray(last_test_seq).astype(np.float32)


y_mask = [len(test_df[test_df['id']==id]) >= sequence_length for id in test_df['id'].unique()]
last_test_label = test_df.groupby('id')['failure_within_w1'].nth(-1)[y_mask].values
last_test_label = last_test_label.reshape(last_test_label.shape[0],1).astype(np.float32)


if os.path.isfile(model_path):
    estimator = load_model(model_path)


start = time.time()
scores_test = estimator.evaluate(last_test_seq, last_test_label, verbose=2)
end = time.time()
print("Total time taken for inferencing: ", "{:.2f}".format((end-start)), " secs")
```

```
3/3 - 1s - loss: 0.0422 - accuracy: 0.9785 - 868ms/epoch - 289ms/step
Total time taken for inferencing:  0.94  secs
```

```
print('Test Accurracy: {}'.format(scores_test[1]))
```

```
Test Accurracy: 0.9784946441650391
```

```
y_pred_test = (estimator.predict(last_test_seq) > 0.5).astype("int32")
y_true_test = last_test_label
```

```
3/3 [==============================] - 1s 23ms/step
```

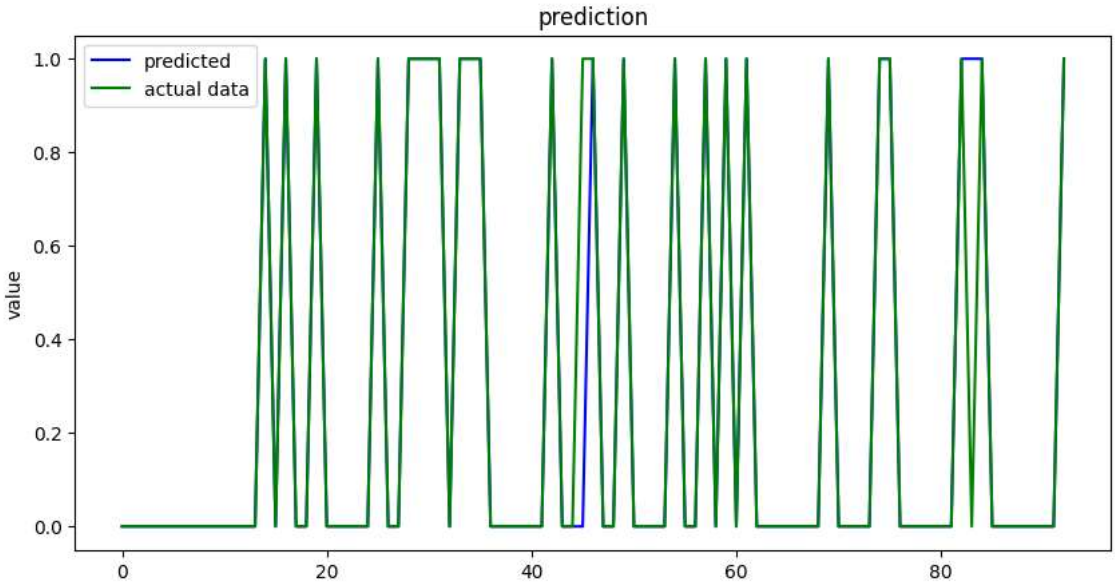
```
test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('binary_submit_test.csv', index = None)


print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted labels')
conf_m = confusion_matrix(y_true_test, y_pred_test)
print(conf_m)
```

```
Confusion matrix
- x-axis is true labels.
- y-axis is predicted labels
[[67  1]
 [ 1 24]]
```

```
precision_test = precision_score(y_true_test, y_pred_test)
recall_test = recall_score(y_true_test, y_pred_test)
f1_test = 2 * (precision_test * recall_test) / (precision_test + recall_test)
print( 'Test Precision: ', precision_test, '\n', 'Test Recall: ', recall_test,'\n', 'Test F1-score:', f1_test )
```

```
Test Precision:  0.96
 Test Recall:  0.96
 Test F1-score: 0.96
```

```
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
```


prediction
predicted
actual data
value
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80