**Deep Learning**
**Dr. DavoodAbadi**
**Fall 2022**
**Hoorieh Sabzevari - 98412004**
**HW5**

۱. ابتدا فایل را دانلود و unzip کرده و با کد زیر یک نمونه را نمایش می‌دهیم:

```
all_files = sorted(glob.glob('*.csv'))
data0 = pd.read_csv(all_files[0])

#view the first dataset
display(data0.head())
```

| | timestamp | value | label |
|---|---|---|---|
| 0 | 1493568000 | 1.901639 | 0 |
| 1 | 1493568060 | 1.786885 | 0 |
| 2 | 1493568120 | 2.000000 | 0 |
| 3 | 1493568180 | 1.885246 | 0 |
| 4 | 1493568240 | 1.819672 | 0 |

همانطور که می‌بینیم، هر دیتاست شامل سه ستون timestamp، value و label است.

با کد زیر نمودار ۵ نمونه از دیتاست‌ها را به صورت رندوم رسم می‌کنیم.

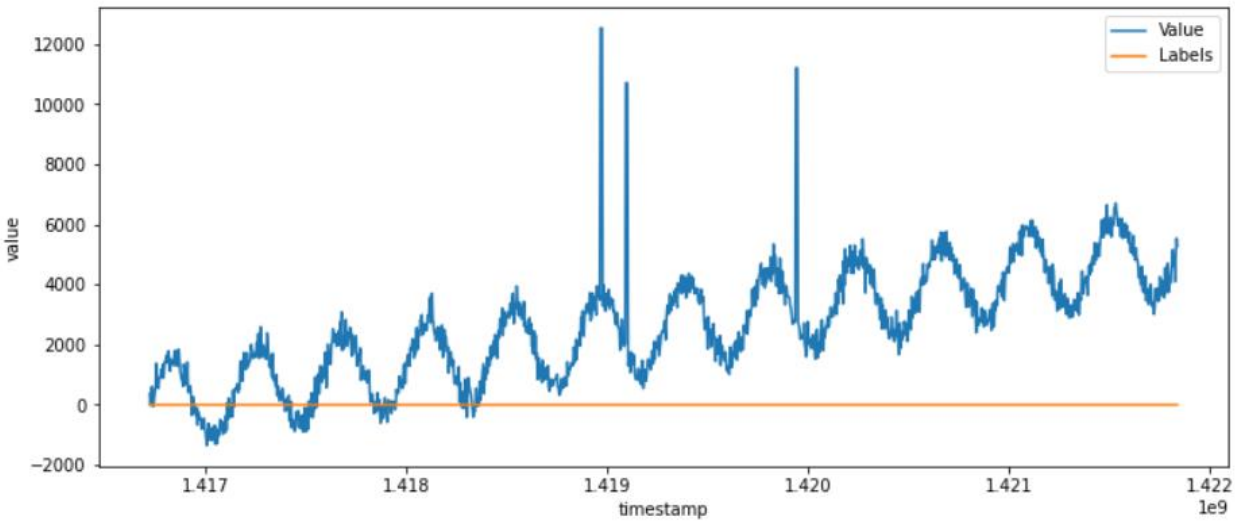
```
# show chart of 5 dataset(csv file) randomly
# https://www.geeksforgeeks.org/how-to-make-a-time-series-plot-with-rolling-average-in-python/

datasets = []
for f in all_files:
  datasets.append(pd.read_csv(f))

rnds = random.sample(range(0, len(datasets)), 5)

for i in rnds:
  plt.figure(figsize = ( 12, 5))
  sns.lineplot( x = 'timestamp',
                y = 'value',
                data = datasets[i],
                label = 'Value')
  sns.lineplot( x = 'timestamp',
                y = 'label',
                data = datasets[i],
                label = 'Labels')
```

خروجی یک نمونه:




۲. نمودارها به صورت دوره‌ای تکرار می‌شوند اما شبیه هم نیستند و نسبت متفاوتی دارند. علیرغم تعداد بالای دیتا، داده‌ها نیز بالانس نیستند. یعنی تعداد لیبل صفر بسیار بیشتر از ۱ است و این مورد ما را ملزم به استفاده از معیار f1_score می‌کند.

۳.

مدل simple RNN :

```
# Simple
model = Sequential()
model.add(SimpleRNN(128, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0834 - f1_score_m: 0.0115 - val_loss: 0.0779 - val_f1_score_m: 0.1177
Epoch 2/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0772 - f1_score_m: 0.1128 - val_loss: 0.0770 - val_f1_score_m: 0.1134
Epoch 3/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0765 - f1_score_m: 0.1307 - val_loss: 0.0766 - val_f1_score_m: 0.1589
Epoch 4/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0761 - f1_score_m: 0.1323 - val_loss: 0.0761 - val_f1_score_m: 0.1456
Epoch 5/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0757 - f1_score_m: 0.1336 - val_loss: 0.0758 - val_f1_score_m: 0.1586
Epoch 6/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0754 - f1_score_m: 0.1318 - val_loss: 0.0760 - val_f1_score_m: 0.0209
Epoch 7/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0752 - f1_score_m: 0.1339 - val_loss: 0.0759 - val_f1_score_m: 0.1578
Epoch 8/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0751 - f1_score_m: 0.1342 - val_loss: 0.0751 - val_f1_score_m: 0.1453
Epoch 9/10
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0749 - f1_score_m: 0.1374 - val_loss: 0.0750 - val_f1_score_m: 0.1453
Epoch 10/10
8869/8869 [==============================] - 42s 5ms/step - loss: 0.0749 - f1_score_m: 0.1341 - val_loss: 0.0749 - val_f1_score_m: 0.1319
```

مدل LSTM :

```
# LSTM
model = Sequential()
model.add(LSTM(128, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 54s 6ms/step - loss: 0.0775 - f1_score_m: 4.0745e-06 - val_loss: 0.0748 - val_f1_score_m: 0.00
Epoch 2/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0701 - f1_score_m: 0.0098 - val_loss: 0.0699 - val_f1_score_m: 0.0613
Epoch 3/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0689 - f1_score_m: 0.0591 - val_loss: 0.0680 - val_f1_score_m: 0.1003
Epoch 4/10
8869/8869 [==============================] - 50s 6ms/step - loss: 0.0680 - f1_score_m: 0.0890 - val_loss: 0.0673 - val_f1_score_m: 0.1178
Epoch 5/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0675 - f1_score_m: 0.1057 - val_loss: 0.0671 - val_f1_score_m: 0.1179
Epoch 6/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0672 - f1_score_m: 0.1154 - val_loss: 0.0671 - val_f1_score_m: 0.1182
Epoch 7/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0671 - f1_score_m: 0.1221 - val_loss: 0.0669 - val_f1_score_m: 0.1008
Epoch 8/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0669 - f1_score_m: 0.1182 - val_loss: 0.0683 - val_f1_score_m: 0.1320
Epoch 9/10
8869/8869 [==============================] - 46s 5ms/step - loss: 0.0668 - f1_score_m: 0.1202 - val_loss: 0.0668 - val_f1_score_m: 0.1465
Epoch 10/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0667 - f1_score_m: 0.1236 - val_loss: 0.0674 - val_f1_score_m: 0.1315
```

مدل GRU :

```
# GRU
model = Sequential()
model.add(GRU(128, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0746 - f1_score_m: 0.0150 - val_loss: 0.0705 - val_f1_score_m: 0.0410
Epoch 2/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0697 - f1_score_m: 0.0701 - val_loss: 0.0696 - val_f1_score_m: 0.0809
Epoch 3/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0686 - f1_score_m: 0.0910 - val_loss: 0.0683 - val_f1_score_m: 0.1178
Epoch 4/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0677 - f1_score_m: 0.1105 - val_loss: 0.0675 - val_f1_score_m: 0.1178
Epoch 5/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0673 - f1_score_m: 0.1221 - val_loss: 0.0673 - val_f1_score_m: 0.1313
Epoch 6/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0669 - f1_score_m: 0.1286 - val_loss: 0.0667 - val_f1_score_m: 0.1506
Epoch 7/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0668 - f1_score_m: 0.1319 - val_loss: 0.0674 - val_f1_score_m: 0.1152
Epoch 8/10
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0666 - f1_score_m: 0.1321 - val_loss: 0.0668 - val_f1_score_m: 0.1652
Epoch 9/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0665 - f1_score_m: 0.1358 - val_loss: 0.0668 - val_f1_score_m: 0.1318
Epoch 10/10
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0664 - f1_score_m: 0.1359 - val_loss: 0.0669 - val_f1_score_m: 0.1320
```

همانطور که واضح است نتایج حاصله نشان می‌دهد که ابتدا مدل سوم، سپس مدل اول و در انتها مدل دوم بهتر عمل کرده است.

۴. اولین روش پیش پردازش، روش MinMaxScaler است که داده‌ها را به محدوده‌ی خاصی می‌برد. مثلا صفر تا یک.

روش دوم نیز StandardScaler است که داده‌ها را به نحوی نرمالایز می‌کند که میانگین صفر و واریانس یک شود.

```
# Preprocces method 1
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train.reshape(-1,1)).reshape(-1,1,1)
x_test = scaler.transform(x_test.reshape(-1,1)).reshape(-1,1,1)
```

```
# Preprocces method 2
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train.reshape(-1,1))
x_test = scaler.transform(x_test.reshape(-1,1))
```

```
# Train 3 models again
# Simple
model = Sequential()
model.add(SimpleRNN(16, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 40s 4ms/step - loss: 0.0822 - f1_score_m: 0.0075 - val_loss: 0.0784 - val_f1_score_m: 0.0594
Epoch 2/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0778 - f1_score_m: 0.0974 - val_loss: 0.0780 - val_f1_score_m: 0.1182
Epoch 3/10
8869/8869 [==============================] - 38s 4ms/step - loss: 0.0776 - f1_score_m: 0.1185 - val_loss: 0.0780 - val_f1_score_m: 0.1310
Epoch 4/10
8869/8869 [==============================] - 39s 4ms/step - loss: 0.0775 - f1_score_m: 0.1204 - val_loss: 0.0779 - val_f1_score_m: 0.1314
Epoch 5/10
8869/8869 [==============================] - 38s 4ms/step - loss: 0.0774 - f1_score_m: 0.1251 - val_loss: 0.0778 - val_f1_score_m: 0.1319
Epoch 6/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0774 - f1_score_m: 0.1278 - val_loss: 0.0777 - val_f1_score_m: 0.1148
Epoch 7/10
8869/8869 [==============================] - 38s 4ms/step - loss: 0.0773 - f1_score_m: 0.1270 - val_loss: 0.0777 - val_f1_score_m: 0.1456
Epoch 8/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0772 - f1_score_m: 0.1333 - val_loss: 0.0775 - val_f1_score_m: 0.1456
Epoch 9/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0771 - f1_score_m: 0.1371 - val_loss: 0.0776 - val_f1_score_m: 0.1590
Epoch 10/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0771 - f1_score_m: 0.1367 - val_loss: 0.0775 - val_f1_score_m: 0.1456
```

```
# LSTM
model = Sequential()
model.add(LSTM(128, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 50s 5ms/step - loss: 0.0782 - f1_score_m: 4.6021e-06 - val_loss: 0.0712 - val_f1_score_m: 0.0
Epoch 2/10
8869/8869 [==============================] - 50s 6ms/step - loss: 0.0703 - f1_score_m: 0.0000e+00 - val_loss: 0.0699 - val_f1_score_m: 0.0
Epoch 3/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0693 - f1_score_m: 0.0049 - val_loss: 0.0688 - val_f1_score_m: 0.0408
Epoch 4/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0684 - f1_score_m: 0.0609 - val_loss: 0.0682 - val_f1_score_m: 0.0802
Epoch 5/10
8869/8869 [==============================] - 46s 5ms/step - loss: 0.0679 - f1_score_m: 0.0960 - val_loss: 0.0686 - val_f1_score_m: 0.0806
Epoch 6/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0675 - f1_score_m: 0.1141 - val_loss: 0.0705 - val_f1_score_m: 0.1004
Epoch 7/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0674 - f1_score_m: 0.1232 - val_loss: 0.0690 - val_f1_score_m: 0.1213
Epoch 8/10
8869/8869 [==============================] - 45s 5ms/step - loss: 0.0672 - f1_score_m: 0.1289 - val_loss: 0.0675 - val_f1_score_m: 0.1321
Epoch 9/10
8869/8869 [==============================] - 48s 5ms/step - loss: 0.0670 - f1_score_m: 0.1352 - val_loss: 0.0668 - val_f1_score_m: 0.1640
Epoch 10/10
8869/8869 [==============================] - 51s 6ms/step - loss: 0.0669 - f1_score_m: 0.1332 - val_loss: 0.0668 - val_f1_score_m: 0.1179
```

```
# GRU
model = Sequential()
model.add(GRU(128, input_shape=(1, 1)))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

history = model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/10
8869/8869 [==============================] - 49s 5ms/step - loss: 0.0743 - f1_score_m: 0.0185 - val_loss: 0.0706 - val_f1_score_m: 0.0599
Epoch 2/10
8869/8869 [==============================] - 42s 5ms/step - loss: 0.0686 - f1_score_m: 0.0865 - val_loss: 0.0677 - val_f1_score_m: 0.1005
Epoch 3/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0677 - f1_score_m: 0.1100 - val_loss: 0.0674 - val_f1_score_m: 0.1202
Epoch 4/10
8869/8869 [==============================] - 47s 5ms/step - loss: 0.0669 - f1_score_m: 0.1353 - val_loss: 0.0671 - val_f1_score_m: 0.1286
Epoch 5/10
8869/8869 [==============================] - 49s 6ms/step - loss: 0.0667 - f1_score_m: 0.1434 - val_loss: 0.0671 - val_f1_score_m: 0.1503
Epoch 6/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0666 - f1_score_m: 0.1489 - val_loss: 0.0668 - val_f1_score_m: 0.1704
Epoch 7/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0665 - f1_score_m: 0.1472 - val_loss: 0.0674 - val_f1_score_m: 0.1547
Epoch 8/10
8869/8869 [==============================] - 43s 5ms/step - loss: 0.0664 - f1_score_m: 0.1492 - val_loss: 0.0671 - val_f1_score_m: 0.1197
Epoch 9/10
8869/8869 [==============================] - 44s 5ms/step - loss: 0.0663 - f1_score_m: 0.1505 - val_loss: 0.0672 - val_f1_score_m: 0.1431
Epoch 10/10
8869/8869 [==============================] - 50s 6ms/step - loss: 0.0662 - f1_score_m: 0.1487 - val_loss: 0.0665 - val_f1_score_m: 0.1315
```

۵. زیرا داده‌ها بالانس نیستند و تعداد اعضای هر کلاس برابر نیست و در این شرایط accuracy و loss معیارهای مناسبی برای تعیین عملکرد مدل نیستند. مثلا اگر مدل ما تمام نمونه‌ها را کلاس صفر پیش‌بینی کند، به دقت بالایی می‌رسیم اما در بعضی مسائل اشتباهات هزینه‌های جبران‌ناپذیری دارند. مانند تشخیص یک بیماری نادر. در این گونه مسائل از معیار f1_score استفاده می‌کنیم که هم precision و هم recall را در نظر دارد.

۶. ابتدا یک تسک پیش‌بینی سری زمانی در استپ بعدی را تعریف می‌کنیم.

```
def pred(data):
    x = []
    y = []
    for i in range(0, len(data) - 1):
        x.append(data[i])
        y.append(data[i + 1])
    return np.array(x).reshape((-1, 1)), np.array(y)
```

```
x_train_ssl, y_train_ssl = pred(x_train.flatten())
x_test_ssl, y_test_ssl = pred(x_test.flatten())
```

مدل GRU خود را تعریف کرده و compile می‌کنیم. سپس روی داده‌هایی که تولید کرده‌ایم آموزش می‌دهیم. از ۶۴ نورون در لایه‌ی اول و ۳۲ نورون در لایه‌ی دوم استفاده می‌کنیم. همچنین چون تقریبا یک مسئله‌ی رگرسیون است از تابع ضرر mse استفاده می‌کنیم.

```
# compile and train the model
model = Sequential()
model.add(LSTM(64, input_shape=(1, 1), return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse', metrics=(f1_score_m,))
```

```
history = model.fit(x_train_ssl, y_train_ssl, epochs=5, batch_size=128, validation_data=(x_test_ssl, y_test_ssl))
```

```
Epoch 1/5
8869/8869 [==============================] - 53s 6ms/step - loss: 32793296896.0000 - f1_score_m: 0.9445 - val_loss: 33567098880.0000 - val_f1_score_m: 0.9442
Epoch 2/5
8869/8869 [==============================] - 51s 6ms/step - loss: 32792354816.0000 - f1_score_m: 0.9445 - val_loss: 33566119936.0000 - val_f1_score_m: 0.9442
Epoch 3/5
8869/8869 [==============================] - 51s 6ms/step - loss: 32791431168.0000 - f1_score_m: 0.9445 - val_loss: 33565171712.0000 - val_f1_score_m: 0.9442
Epoch 4/5
8869/8869 [==============================] - 51s 6ms/step - loss: 32790351872.0000 - f1_score_m: 0.9445 - val_loss: 33564289024.0000 - val_f1_score_m: 0.9442
Epoch 5/5
8869/8869 [==============================] - 51s 6ms/step - loss: 32789501952.0000 - f1_score_m: 0.9445 - val_loss: 33563400192.0000 - val_f1_score_m: 0.9442
```

همانطور که می‌بینیم به امتیاز بالای ۰.۹ رسیدیم.

حال با وزن‌های این مدل، مدل جدید خود را می‌سازیم.

```
new_model = Sequential()
```

```
# delete last layer of model
for layer in model.layers[:-1]:
  new_model.add(layer)
```

```
# freeze all remaining layers except the last one
for layer in new_model.layers[:-1]:
    layer.trainable = False
```

```
# add 2 dence layer to the model
new_model.add(Dense(16, activation='relu'))

new_model.add(Dense(1, activation='sigmoid'))
```

مدل جدید را آموزش می‌دهیم.

```
# train the main task(anomaly detection)
new_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=(f1_score_m,))

new_model.fit(x_train_ssl, y_train_ssl, epochs=5, batch_size=128, validation_data=(x_test_ssl, y_test_ssl))
```

```
Epoch 1/5
8869/8869 [==============================] - 50s 5ms/step - loss: -9524.0615 - f1_score_m: 0.9757 - val_loss: 3.1885 - val_f1_score_m: 0.0372
Epoch 2/5
8869/8869 [==============================] - 48s 5ms/step - loss: -31625.8027 - f1_score_m: 0.9363 - val_loss: 6.4256 - val_f1_score_m: 0.0372
Epoch 3/5
8869/8869 [==============================] - 48s 5ms/step - loss: -53998.7031 - f1_score_m: 0.9363 - val_loss: 9.7552 - val_f1_score_m: 0.0372
Epoch 4/5
8869/8869 [==============================] - 47s 5ms/step - loss: -75716.8828 - f1_score_m: 0.9363 - val_loss: 13.0244 - val_f1_score_m: 0.0372
Epoch 5/5
8869/8869 [==============================] - 47s 5ms/step - loss: -97655.0156 - f1_score_m: 0.9363 - val_loss: 16.3447 - val_f1_score_m: 0.0372
```

در داده‌های آموزشی به امتیاز خوبی رسیده اما مدل overfit شده است.

۷.down sampling تکنیکی است که تعداد نمونه‌های آموزشی را که در کلاس اکثریت قرار می‌گیرند کاهش می‌دهد تا تعداد متناسب شوند. همچنین با حذف داده‌های جمع‌آوری‌شده، اطلاعات ارزشمند زیادی را از دست می‌دهیم.

upsampling تکنیکی است که در آن نقاط داده تولید شده مصنوعی (مرتبط با کلاس اقلیت) به مجموعه‌ی داده اضافه می‌شود. پس از این فرآیند، تعداد هر دو کلاس تقریباً یکسان است. upsampling به دلیل اطلاعات اضافی، بایاس را به سیستم وارد می‌کند.

```
majority = df[df["label"] == 0]
minority = df[df["label"] == 1]

downsampled = resample(majority, replace=True, random_state=42, n_samples=len(minority))

upsampled = pd.concat([downsampled, minority])
x = upsampled["value"]
x = x.values.reshape(-1, 1, 1)
y = upsampled.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
```

```
338/338 [==============================] - 2s 5ms/step - loss: 0.4513 - f1_score_m: 0.8199 - val_loss: 0.4507 - val_f1_score_m: 0.8225
Epoch 9/10
338/338 [==============================] - 2s 5ms/step - loss: 0.4484 - f1_score_m: 0.8199 - val_loss: 0.4536 - val_f1_score_m: 0.8214
Epoch 10/10
338/338 [==============================] - 2s 5ms/step - loss: 0.4470 - f1_score_m: 0.8203 - val_loss: 0.4454 - val_f1_score_m: 0.8221
```

همانطور که مشاهده می‌کنیم، مدل به امتیاز بالای ۰.۸ رسیده است.

۸. Z-Score یک مقدار آماری است که به ما اعلام می‌کند یک مقدار خاص چند انحراف معیار از میانگین کل مجموعه داده فاصله دارد.

```
df = datasets[0]
df['z-score'] = stats.zscore(df['value'])
display(df)
```

| | timestamp | value | label | z-score |
|---|---|---|---|---|
| **0** | 1493568000 | 1.901639 | 0 | -0.074118 |
| **1** | 1493568060 | 1.786885 | 0 | -0.249654 |
| **2** | 1493568120 | 2.000000 | 0 | 0.076341 |
| **3** | 1493568180 | 1.885246 | 0 | -0.099195 |
| **4** | 1493568240 | 1.819672 | 0 | -0.199501 |
| **...** | ... | ... | ... | ... |
| **128557** | 1501475400 | 2.684211 | 0 | 1.122956 |
| **128558** | 1501475460 | 2.526316 | 0 | 0.881429 |
| **128559** | 1501475520 | 2.614035 | 0 | 1.015611 |
| **128560** | 1501475580 | 2.736842 | 0 | 1.203465 |
| **128561** | 1501475640 | 2.491228 | 0 | 0.827757 |