GENERATIVE ADVERSARIAL NETWORKS

SUBMITTED BY: PAOLO JOSHUA R. BILLONES
STUDENT NUMBER: 1911430
DATE: OCTOBER 22, 2022

SECTION: ECE41S11
INSTRUCTOR: ENGR. CHRISTIAN P. RIOFLORIDO

Before I Start with model construction, I installed y-data synthetic, which defines the TimeGAN to be used with ModelParameters being used to define the settings of the model. A complementary preprocessing function was also called.

```
!pip install ydata-synthetic
```

```
from os import path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from ydata_synthetic.synthesizers import ModelParameters
from ydata_synthetic.preprocessing.timeseries import processed_stock
from ydata_synthetic.synthesizers.timeseries import TimeGAN
```

```
seq_len=24
n_seq = 6
hidden_dim=24
gamma=1

noise_dim = 32
dim = 128
batch_size = 128

log_step = 100
learning_rate = 5e-4

gan_args = ModelParameters(batch_size=batch_size,
                           lr=learning_rate,
                           noise_dim=noise_dim,
                           layers_dim=dim)
```

The dataset for this application is the Robinson Retail Holdings Inc. This dataset contains a mix of float values and string, and the dates. Since preprocessing function requires a dataset that only contains pure numerical data, the RRHI dataset would have to be preprocessed first, by dropping the date column, converting the Vol. value of string to float, and removing the % in the Change % column:

```
stock = pd.read_csv("RRHI Historical Data Complete.csv")
stock["Change %"] = stock["Change %"].str.replace('%', '')
stock["Change %"] = pd.to_numeric(stock["Change %"])
for i in range(len(stock["Vol."])):
    if  "K" in stock.at[i,"Vol."]:
        stock.at[i,"Vol."] = stock.at[i,"Vol."].replace('K', '')
        stock.at[i,"Vol."] = float(stock.at[i,"Vol."])
    elif "M" in stock.at[i,"Vol."]:
        stock.at[i,"Vol."] = stock.at[i,"Vol."].replace('M', '')
        stock.at[i,"Vol."] = float(stock.at[i,"Vol."])*1000000/1000
stock = stock.drop(["Date"], axis=1)
stock.to_csv('Robinson Prep.csv', index=False)
```

```
stock_data = processed_stock("Robinson Prep.csv", seq_len=seq_len)
print(len(stock_data),stock_data[0].shape)
```

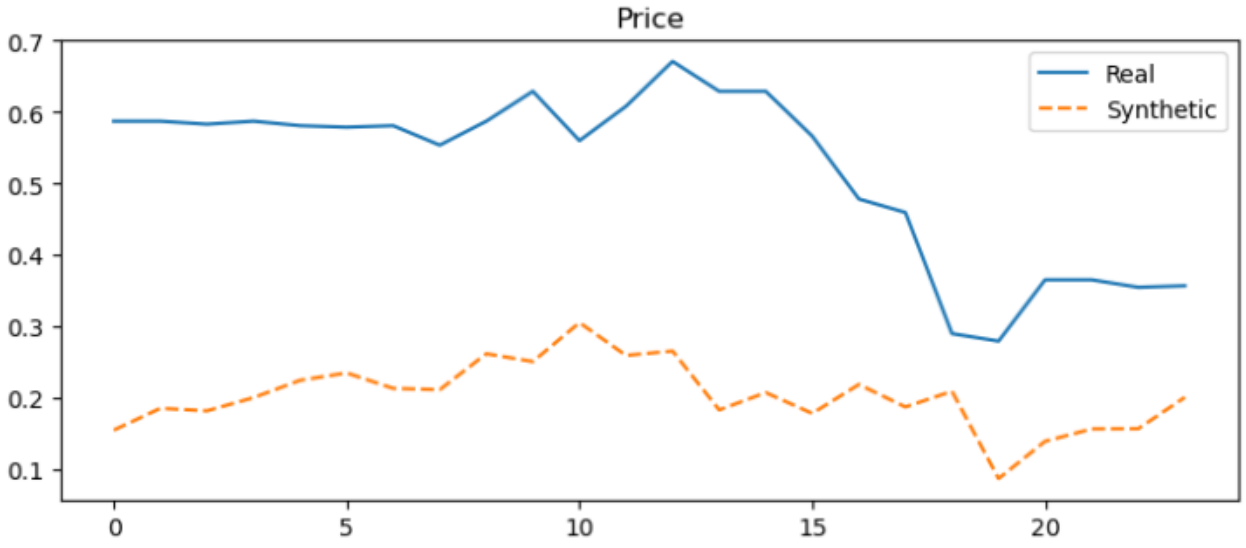
The preprocessed data can then be fit to the TimeGAN; with parameters defined from before; this process took over 30 hours to complete, so I attached the link to the pickle file in the notebook.

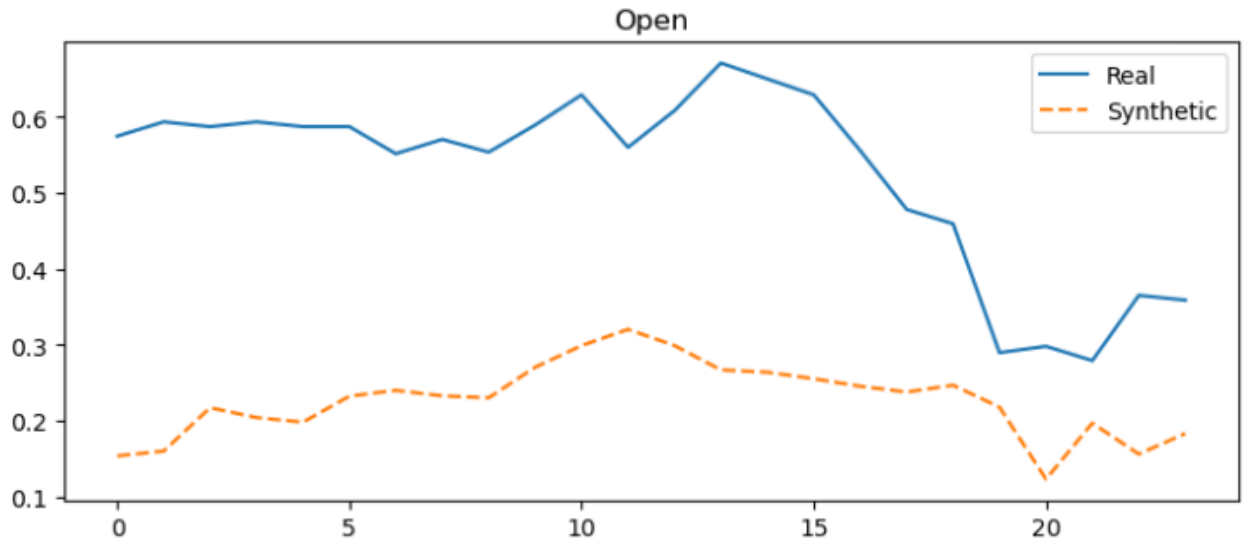
```
if path.exists('synthesizer_stock.pkl'):
    synth = TimeGAN.load('synthesizer_stock.pkl')
else:
    synth = TimeGAN(model_parameters=gan_args, hidden_dim=24, seq_len=seq_len, n_seq=n_seq, gamma=1)
    synth.train(stock_data, train_steps=50000)
    synth.save('synthesizer_stock.pkl')
```

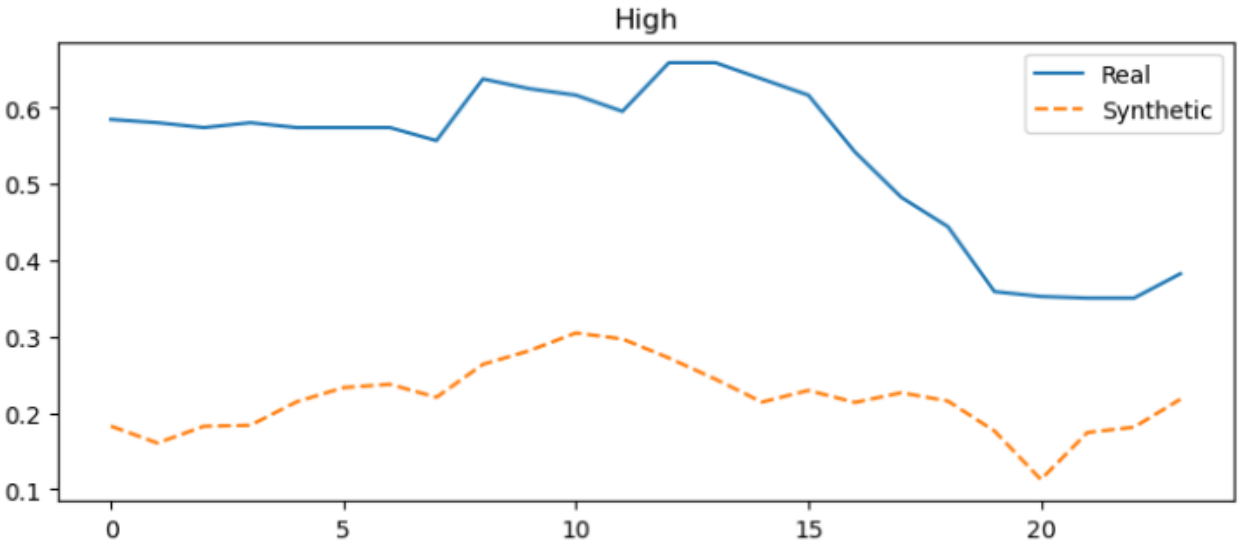
```
synth_data = synth.sample(len(stock_data))
synth_data.shape
```

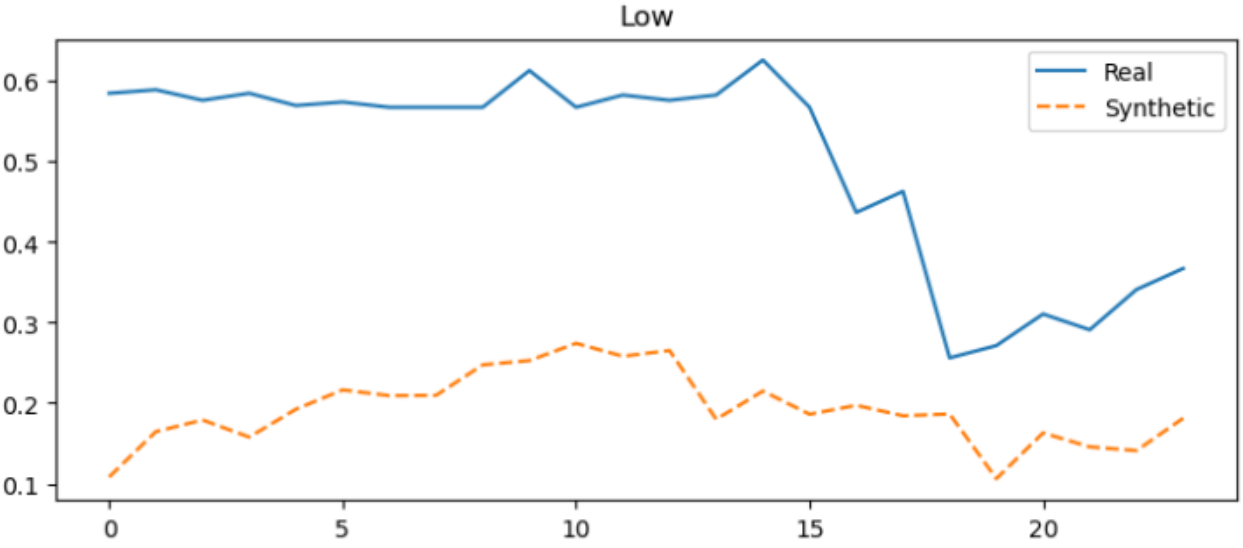
Once that is done, the difference between the real and synthetic data is mapped; the values range from 0 to 1 as it was processed beforehand through MinMaxScaler from the preprocessing function.

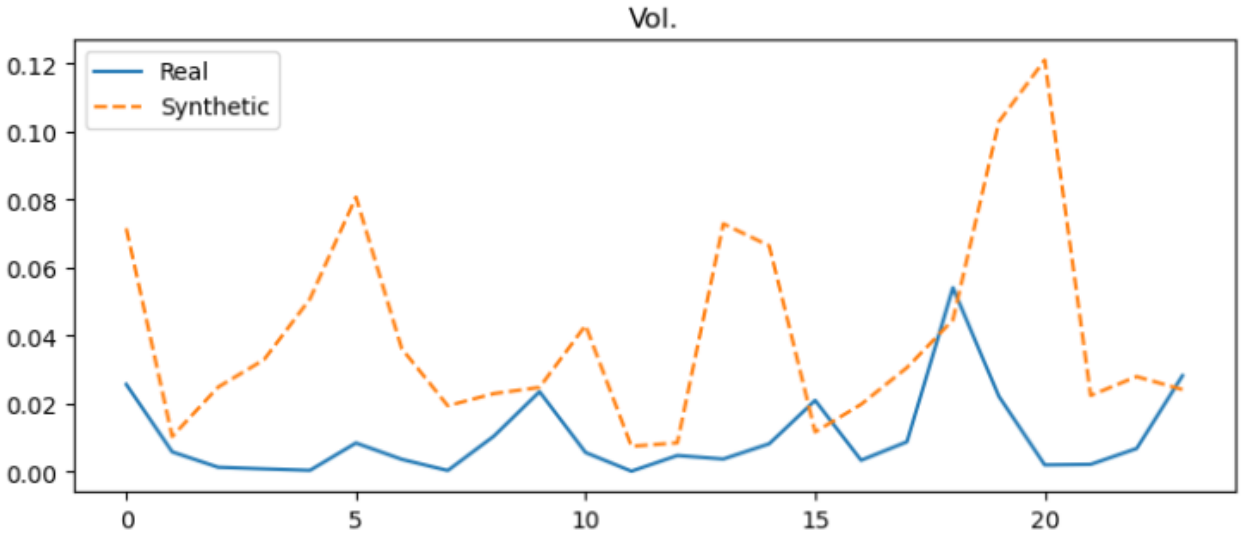
```
cols = ['Price','Open','High','Low','Vol.','Change%']

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 10))
axes=axes.flatten()

time = list(range(1,25))
obs = np.random.randint(len(stock_data))

for j, col in enumerate(cols):
    df = pd.DataFrame({'Real': stock_data[obs][:, j],
                       'Synthetic': synth_data[obs][:, j]})
    df.plot(ax=axes[j],
            title = col,
            secondary_y='Synthetic data', style=['-', '--'])
fig.tight_layout()
```

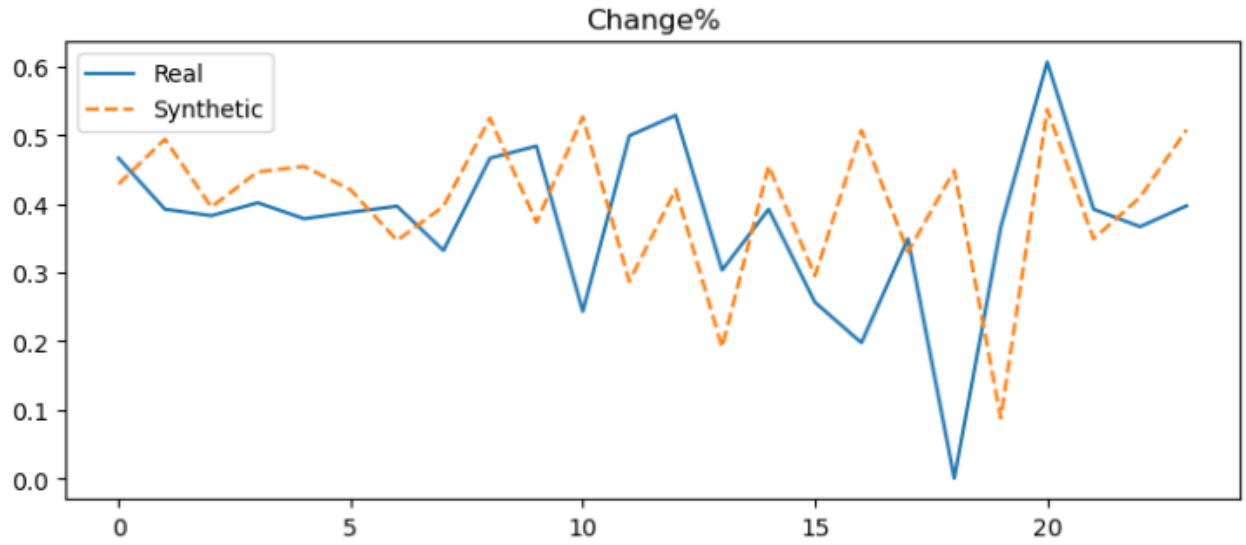



Open
Real
Synthetic
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20


High
Real
Synthetic
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20


Low
Real
Synthetic
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20





The dataset has a high dimensionality, so PCA and t-SNE were used to simplify the dataset generated:

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

sample_size = 250
idx = np.random.permutation(len(stock_data))[:sample_size]

real_sample = np.asarray(stock_data)[idx]
synthetic_sample = np.asarray(synth_data)[idx]

synth_data_reduced = real_sample.reshape(-1, seq_len)
stock_data_reduced = np.asarray(synthetic_sample).reshape(-1,seq_len)

n_components = 2
pca = PCA(n_components=n_components)
tsne = TSNE(n_components=n_components, n_iter=300)

pca.fit(stock_data_reduced)

pca_real = pd.DataFrame(pca.transform(stock_data_reduced))
pca_synth = pd.DataFrame(pca.transform(synth_data_reduced))

data_reduced = np.concatenate((stock_data_reduced, synth_data_reduced), axis=0)
tsne_results = pd.DataFrame(tsne.fit_transform(data_reduced))
```

Now that the dataset is simplified, we can start to preprocess it and feed it to the model. This section is for demonstration purposes only, so GRU was used for the models.

```
from tensorflow.keras import Input, Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import GRU, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanAbsoluteError

def RNN_regression(units):
    opt = Adam(name='AdamOpt')
    loss = MeanAbsoluteError(name='MAE')
    model = Sequential()
    model.add(GRU(units=units,
                  name=f'RNN_1'))
    model.add(Dense(units=6,
                    activation='sigmoid',
                    name='OUT'))
    model.compile(optimizer=opt, loss=loss)
    return model
```

```
stock_data=np.asarray(stock_data)
synth_data = synth_data[:len(stock_data)]
n_events = len(stock_data)

idx = np.arange(n_events)
n_train = int(.75*n_events)
train_idx = idx[:n_train]
test_idx = idx[n_train:]

X_stock_train = stock_data[train_idx, :seq_len-1, :]
X_synth_train = synth_data[train_idx, :seq_len-1, :]

X_stock_test = stock_data[test_idx, :seq_len-1, :]
y_stock_test = stock_data[test_idx, -1, :]

y_stock_train = stock_data[train_idx, -1, :]
y_synth_train = synth_data[train_idx, -1, :]

print('Synthetic X train: {}'.format(X_synth_train.shape))
print('Real X train: {}'.format(X_stock_train.shape))

print('Synthetic y train: {}'.format(y_synth_train.shape))
print('Real y train: {}'.format(y_stock_train.shape))

print('Real X test: {}'.format(X_stock_test.shape))
print('Real y test: {}'.format(y_stock_test.shape))
```

After preprocessing, one model was fitted with real data, and the other model was fitted with synthetic data:

```
ts_real = RNN_regression(12)
early_stopping = EarlyStopping(monitor='val_loss')
#training the model with real data
real_train = ts_real.fit(x=X_stock_train,
                         y=y_stock_train,
                         validation_data=(X_stock_test, y_stock_test),
                         epochs=200,
                         batch_size=32,
                         callbacks=[early_stopping])
```

```
Epoch 1/200
11/11 [==============================] - 2s 41ms/step - loss: 0.2282 - val_loss: 0.2312
Epoch 2/200
11/11 [==============================] - 0s 8ms/step - loss: 0.2198 - val_loss: 0.2229
```

```
#Training the model with the synthetic data
ts_synth = RNN_regression(12)
synth_train = ts_synth.fit(x=X_synth_train,
                         y=y_synth_train,
                         validation_data=(X_stock_test, y_stock_test),
                         epochs=200,
                         batch_size=128,
                         callbacks=[early_stopping])
```

```
Epoch 1/200
3/3 [==============================] - 2s 179ms/step - loss: 0.2557 - val_loss: 0.2502
Epoch 2/200
3/3 [==============================] - 0s 19ms/step - loss: 0.2514 - val_loss: 0.2467
```

The models are evaluated with using R2 score, mean absolute error, and mean squared log error.

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_log_error
real_predictions = ts_real.predict(X_stock_test)
synth_predictions = ts_synth.predict(X_stock_test)

metrics_dict = {'r2': [r2_score(y_stock_test, real_predictions),
                       r2_score(y_stock_test, synth_predictions)],
                'MAE': [mean_absolute_error(y_stock_test, real_predictions),
                        mean_absolute_error(y_stock_test, synth_predictions)],
                'MRLE': [mean_squared_log_error(y_stock_test, real_predictions),
                         mean_squared_log_error(y_stock_test, synth_predictions)]}

results = pd.DataFrame(metrics_dict, index=['Real', 'Synthetic'])

results
```

```
4/4 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 2ms/step
```

Which is shown here.

	r2	MAE	MRLE
Real	-3.734383	0.222878	0.040752
Synthetic	-4.080809	0.246716	0.044927

The GAN used for this application is TimeGAN; which was made suitable for synthesizing time-series forecasting. In this application, Robinson's Retail Holdings Inc stock prices were used as training data for the model. While TimeGAN proves useful for the upcoming Design Project, TimeGAN must be utilized as soon as possible, since the longest part of this activity was training the models, which reached up to 30+ hours in fitting the data alone.