Open in Colab

T81-558: Applications of Deep Neural Networks

Module 7: Generative Adversarial Networks

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 7 Material

- Part 7.1: Introduction to GANs for Image and Data Generation [Video] [Notebook]
- Part 7.2: Train StyleGAN3 with your Own Images [Video] [Notebook]
- Part 7.3: Exploring the StyleGAN Latent Vector [Video] [Notebook]
- Part 7.4: GANs to Enhance Old Photographs Deoldify [Video] [Notebook]
- **Part 7.5: GANs for Tabular Synthetic Data Generation** [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow. Running the following code will map your GDrive to `/content/drive`.

In [1]:
```
try:
    from google.colab import drive
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

Part 7.5: GANs for Tabular Synthetic Data Generation

Typically GANs are used to generate images. However, we can also generate tabular data from a GAN. In this part, we will use the Python tabgan utility to create fake data from tabular data. Specifically, we will use the Auto MPG dataset to train a GAN to generate fake cars. Cite:ashrapov2020tabular

Installing Tabgan

Pytorch is the foundation of the tabgan neural network utility. The following code installs the needed software to run tabgan in Google Colab.

In [2]:
```
# HIDE OUTPUT
CMD = "wget https://raw.githubusercontent.com/Diyago/"\
  "GAN-for-tabular-data/master/requirements.txt"

!{CMD}
!pip install -r requirements.txt
!pip install tabgan
```

```
--2022-04-03 18:53:04--  https://raw.githubusercontent.com/Diyago/GAN-for-
tabular-data/master/requirements.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 197 [text/plain]
Saving to: ‘requirements.txt.1’

requirements.txt.1  100%[===================>]     197  --.-KB/s    in 0s

2022-04-03 18:53:04 (8.18 MB/s) - ‘requirements.txt.1’ saved [197/197]

Requirement already satisfied: scipy==1.4.1 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 1)) (1.4.1)
Requirement already satisfied: category_encoders==2.1.0 in /usr/local/lib/
python3.7/dist-packages (from -r requirements.txt (line 2)) (2.1.0)
Requirement already satisfied: numpy==1.18.1 in /usr/local/lib/python3.7/d
ist-packages (from -r requirements.txt (line 3)) (1.18.1)
Requirement already satisfied: torch==1.6.0 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 4)) (1.6.0)
Requirement already satisfied: pandas==1.2.2 in /usr/local/lib/python3.7/d
ist-packages (from -r requirements.txt (line 5)) (1.2.2)
Requirement already satisfied: lightgbm==2.3.1 in /usr/local/lib/python
3.7/dist-packages (from -r requirements.txt (line 6)) (2.3.1)
Requirement already satisfied: scikit_learn==0.23.2 in /usr/local/lib/pyth
on3.7/dist-packages (from -r requirements.txt (line 7)) (0.23.2)
Requirement already satisfied: torchvision>=0.4.2 in /usr/local/lib/python
3.7/dist-packages (from -r requirements.txt (line 8)) (0.7.0)
Requirement already satisfied: python-dateutil==2.8.1 in /usr/local/lib/py
thon3.7/dist-packages (from -r requirements.txt (line 11)) (2.8.1)
Requirement already satisfied: tqdm==4.61.1 in /usr/local/lib/python3.7/di
st-packages (from -r requirements.txt (line 12)) (4.61.1)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.7/di
st-packages (from category_encoders==2.1.0->-r requirements.txt (line 2))
(0.5.2)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/lib/python
3.7/dist-packages (from category_encoders==2.1.0->-r requirements.txt (lin
e 2)) (0.10.2)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-pac
kages (from torch==1.6.0->-r requirements.txt (line 4)) (0.16.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/di
st-packages (from pandas==1.2.2->-r requirements.txt (line 5)) (2018.9)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.7/dist-packages (from scikit_learn==0.23.2->-r requirements.txt (line
7)) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/di
st-packages (from scikit_learn==0.23.2->-r requirements.txt (line 7))
(1.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-p
ackages (from python-dateutil==2.8.1->-r requirements.txt (line 11)) (1.1
5.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/d
ist-packages (from torchvision>=0.4.2->-r requirements.txt (line 8))
(7.1.2)
Requirement already satisfied: tabgan in /usr/local/lib/python3.7/dist-pac
kages (1.2.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-pack
ages (from tabgan) (1.18.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-pac
kages (from tabgan) (1.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packa
```

```
ges (from tabgan) (4.61.1)
Requirement already satisfied: scikit-learn==0.23.2 in /usr/local/lib/pyth
on3.7/dist-packages (from tabgan) (0.23.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python
3.7/dist-packages (from tabgan) (2.8.1)
Requirement already satisfied: category-encoders in /usr/local/lib/python
3.7/dist-packages (from tabgan) (2.1.0)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-p
ackages (from tabgan) (2.3.1)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dis
t-packages (from tabgan) (0.7.0)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-pack
ages (from tabgan) (1.6.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/di
st-packages (from scikit-learn==0.23.2->tabgan) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.7/dist-packages (from scikit-learn==0.23.2->tabgan) (3.1.0)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/d
ist-packages (from scikit-learn==0.23.2->tabgan) (1.4.1)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.7/di
st-packages (from category-encoders->tabgan) (0.5.2)
Requirement already satisfied: statsmodels>=0.6.1 in /usr/local/lib/python
3.7/dist-packages (from category-encoders->tabgan) (0.10.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/di
st-packages (from pandas->tabgan) (2018.9)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packag
es (from patsy>=0.4.1->category-encoders->tabgan) (1.15.0)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-pac
kages (from torch->tabgan) (0.16.0)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/d
ist-packages (from torchvision->tabgan) (7.1.2)
```

Note, after installing; you may see this message:

- You must restart the runtime in order to use newly installed versions.

If so, click the "restart runtime" button just under the message. Then rerun this notebook, and you should not receive further issues.

Loading the Auto MPG Data and Training a Neural Network

We will begin by generating fake data for the Auto MPG dataset we have previously seen. The tabgan library can generate categorical (textual) and continuous (numeric) data. However, it cannot generate unstructured data, such as the name of the automobile. Car names, such as "AMC Rebel SST" cannot be replicated by the GAN, because every row has a different car name; it is a textual but non-categorical value.

The following code is similar to what we have seen before. We load the AutoMPG dataset. The tabgan library requires Pandas dataframe to train. Because of this, we keep both the Pandas and Numpy values.

In [3]:

```python
# HIDE OUTPUT
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import pandas as pd
import io
import os
import requests
import numpy as np
from sklearn import metrics

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

COLS_USED = ['cylinders', 'displacement', 'horsepower', 'weight',
          'acceleration', 'year', 'origin','mpg']
COLS_TRAIN = ['cylinders', 'displacement', 'horsepower', 'weight',
          'acceleration', 'year', 'origin']

df = df[COLS_USED]

# Handle missing value
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())


# Split into training and test sets
df_x_train, df_x_test, df_y_train, df_y_test = train_test_split(
    df.drop("mpg", axis=1),
    df["mpg"],
    test_size=0.20,
    #shuffle=False,
    random_state=42,
)

# Create dataframe versions for tabular GAN
df_x_test, df_y_test = df_x_test.reset_index(drop=True), \
  df_y_test.reset_index(drop=True)
df_y_train = pd.DataFrame(df_y_train)
df_y_test = pd.DataFrame(df_y_test)

# Pandas to Numpy
x_train = df_x_train.values
x_test = df_x_test.values
y_train = df_y_train.values
y_test = df_y_test.values

# Build the neural network
model = Sequential()
# Hidden 1
model.add(Dense(50, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(25, activation='relu')) # Hidden 2
model.add(Dense(12, activation='relu')) # Hidden 2
model.add(Dense(1)) # Output
model.compile(loss='mean_squared_error', optimizer='adam')

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
        patience=5, verbose=1, mode='auto',
        restore_best_weights=True)
model.fit(x_train,y_train,validation_data=(x_test,y_test),
        callbacks=[monitor], verbose=2,epochs=1000)
```

```
Epoch 1/1000
10/10 - 1s - loss: 139176.5625 - val_loss: 40689.0703 - 1s/epoch - 148ms/s
tep
Epoch 2/1000
10/10 - 0s - loss: 19372.2285 - val_loss: 3346.7378 - 108ms/epoch - 11ms/s
tep
Epoch 3/1000
10/10 - 0s - loss: 873.7932 - val_loss: 769.1017 - 109ms/epoch - 11ms/step
Epoch 4/1000
10/10 - 0s - loss: 1485.8730 - val_loss: 1525.9556 - 136ms/epoch - 14ms/st
ep
Epoch 5/1000
10/10 - 0s - loss: 866.6918 - val_loss: 195.6039 - 155ms/epoch - 15ms/step
Epoch 6/1000
10/10 - 0s - loss: 142.9136 - val_loss: 177.2400 - 96ms/epoch - 10ms/step
Epoch 7/1000
10/10 - 0s - loss: 193.9373 - val_loss: 142.7312 - 113ms/epoch - 11ms/step
Epoch 8/1000
10/10 - 0s - loss: 116.1862 - val_loss: 89.0451 - 79ms/epoch - 8ms/step
Epoch 9/1000
10/10 - 0s - loss: 106.6868 - val_loss: 95.9191 - 174ms/epoch - 17ms/step
Epoch 10/1000
10/10 - 0s - loss: 104.5894 - val_loss: 87.7888 - 111ms/epoch - 11ms/step
Epoch 11/1000
10/10 - 0s - loss: 100.0589 - val_loss: 88.2749 - 96ms/epoch - 10ms/step
Epoch 12/1000
10/10 - 0s - loss: 99.6257 - val_loss: 87.3040 - 115ms/epoch - 11ms/step
Epoch 13/1000
10/10 - 0s - loss: 99.4177 - val_loss: 86.6027 - 103ms/epoch - 10ms/step
Epoch 14/1000
10/10 - 0s - loss: 98.5141 - val_loss: 86.1316 - 97ms/epoch - 10ms/step
Epoch 15/1000
10/10 - 0s - loss: 98.0732 - val_loss: 85.9742 - 108ms/epoch - 11ms/step
Epoch 16/1000
10/10 - 0s - loss: 97.4856 - val_loss: 85.1393 - 76ms/epoch - 8ms/step
Epoch 17/1000
10/10 - 0s - loss: 97.1630 - val_loss: 84.7279 - 85ms/epoch - 8ms/step
Epoch 18/1000
10/10 - 0s - loss: 96.7964 - val_loss: 84.2021 - 160ms/epoch - 16ms/step
Epoch 19/1000
10/10 - 0s - loss: 96.3048 - val_loss: 83.9871 - 92ms/epoch - 9ms/step
Epoch 20/1000
10/10 - 0s - loss: 95.4462 - val_loss: 83.0952 - 121ms/epoch - 12ms/step
Epoch 21/1000
10/10 - 0s - loss: 94.9444 - val_loss: 82.6417 - 91ms/epoch - 9ms/step
Epoch 22/1000
10/10 - 0s - loss: 94.3362 - val_loss: 82.0355 - 132ms/epoch - 13ms/step
Epoch 23/1000
10/10 - 0s - loss: 93.8082 - val_loss: 81.6199 - 89ms/epoch - 9ms/step
Epoch 24/1000
10/10 - 0s - loss: 93.2513 - val_loss: 81.1020 - 82ms/epoch - 8ms/step
Epoch 25/1000
10/10 - 0s - loss: 92.6264 - val_loss: 80.3359 - 121ms/epoch - 12ms/step
Epoch 26/1000
10/10 - 0s - loss: 92.2328 - val_loss: 79.8444 - 121ms/epoch - 12ms/step
Epoch 27/1000
10/10 - 0s - loss: 91.4926 - val_loss: 79.1404 - 109ms/epoch - 11ms/step
Epoch 28/1000
10/10 - 0s - loss: 90.7999 - val_loss: 78.6531 - 118ms/epoch - 12ms/step
Epoch 29/1000
10/10 - 0s - loss: 90.1882 - val_loss: 78.1106 - 112ms/epoch - 11ms/step
Epoch 30/1000
10/10 - 0s - loss: 89.8745 - val_loss: 77.8685 - 116ms/epoch - 12ms/step
```

```
Epoch 31/1000
10/10 - 0s - loss: 89.4765 - val_loss: 76.8118 - 94ms/epoch - 9ms/step
Epoch 32/1000
10/10 - 0s - loss: 88.4912 - val_loss: 76.5078 - 87ms/epoch - 9ms/step
Epoch 33/1000
10/10 - 0s - loss: 88.0864 - val_loss: 75.5026 - 102ms/epoch - 10ms/step
Epoch 34/1000
10/10 - 0s - loss: 86.9415 - val_loss: 75.0887 - 90ms/epoch - 9ms/step
Epoch 35/1000
10/10 - 0s - loss: 86.7026 - val_loss: 74.8265 - 129ms/epoch - 13ms/step
Epoch 36/1000
10/10 - 0s - loss: 86.5384 - val_loss: 73.6916 - 97ms/epoch - 10ms/step
Epoch 37/1000
10/10 - 0s - loss: 85.6226 - val_loss: 73.5788 - 105ms/epoch - 11ms/step
Epoch 38/1000
10/10 - 0s - loss: 84.6683 - val_loss: 72.4751 - 91ms/epoch - 9ms/step
Epoch 39/1000
10/10 - 0s - loss: 83.8491 - val_loss: 71.7716 - 90ms/epoch - 9ms/step
Epoch 40/1000
10/10 - 0s - loss: 83.1613 - val_loss: 71.0936 - 119ms/epoch - 12ms/step
Epoch 41/1000
10/10 - 0s - loss: 82.5631 - val_loss: 70.6658 - 89ms/epoch - 9ms/step
Epoch 42/1000
10/10 - 0s - loss: 81.8695 - val_loss: 69.8167 - 163ms/epoch - 16ms/step
Epoch 43/1000
10/10 - 0s - loss: 81.1869 - val_loss: 69.2964 - 91ms/epoch - 9ms/step
Epoch 44/1000
10/10 - 0s - loss: 80.8101 - val_loss: 68.9843 - 88ms/epoch - 9ms/step
Epoch 45/1000
10/10 - 0s - loss: 80.6469 - val_loss: 67.9292 - 143ms/epoch - 14ms/step
Epoch 46/1000
10/10 - 0s - loss: 79.4096 - val_loss: 67.6057 - 102ms/epoch - 10ms/step
Epoch 47/1000
10/10 - 0s - loss: 78.5745 - val_loss: 66.5229 - 69ms/epoch - 7ms/step
Epoch 48/1000
10/10 - 0s - loss: 78.8939 - val_loss: 66.6657 - 95ms/epoch - 10ms/step
Epoch 49/1000
10/10 - 0s - loss: 76.9754 - val_loss: 65.2553 - 106ms/epoch - 11ms/step
Epoch 50/1000
10/10 - 0s - loss: 76.6228 - val_loss: 64.6849 - 81ms/epoch - 8ms/step
Epoch 51/1000
10/10 - 0s - loss: 76.0204 - val_loss: 64.7692 - 120ms/epoch - 12ms/step
Epoch 52/1000
10/10 - 0s - loss: 74.8868 - val_loss: 63.3094 - 119ms/epoch - 12ms/step
Epoch 53/1000
10/10 - 0s - loss: 74.4092 - val_loss: 62.8904 - 136ms/epoch - 14ms/step
Epoch 54/1000
10/10 - 0s - loss: 73.6486 - val_loss: 62.5721 - 98ms/epoch - 10ms/step
Epoch 55/1000
10/10 - 0s - loss: 72.7242 - val_loss: 61.3689 - 131ms/epoch - 13ms/step
Epoch 56/1000
10/10 - 0s - loss: 72.2849 - val_loss: 61.0335 - 120ms/epoch - 12ms/step
Epoch 57/1000
10/10 - 0s - loss: 72.1777 - val_loss: 60.2657 - 163ms/epoch - 16ms/step
Epoch 58/1000
10/10 - 0s - loss: 71.7879 - val_loss: 59.4650 - 127ms/epoch - 13ms/step
Epoch 59/1000
10/10 - 0s - loss: 71.8203 - val_loss: 60.6488 - 83ms/epoch - 8ms/step
Epoch 60/1000
10/10 - 0s - loss: 69.9323 - val_loss: 58.5242 - 135ms/epoch - 13ms/step
Epoch 61/1000
10/10 - 0s - loss: 70.4658 - val_loss: 58.6250 - 153ms/epoch - 15ms/step
Epoch 62/1000
```

```
10/10 - 0s - loss: 68.6058 - val_loss: 57.0953 - 202ms/epoch - 20ms/step
Epoch 63/1000
10/10 - 0s - loss: 67.7657 - val_loss: 56.8579 - 110ms/epoch - 11ms/step
Epoch 64/1000
10/10 - 0s - loss: 67.2709 - val_loss: 56.0743 - 134ms/epoch - 13ms/step
Epoch 65/1000
10/10 - 0s - loss: 66.5735 - val_loss: 55.5872 - 115ms/epoch - 12ms/step
Epoch 66/1000
10/10 - 0s - loss: 66.1336 - val_loss: 54.8934 - 84ms/epoch - 8ms/step
Epoch 67/1000
10/10 - 0s - loss: 65.7582 - val_loss: 54.4984 - 142ms/epoch - 14ms/step
Epoch 68/1000
10/10 - 0s - loss: 65.1338 - val_loss: 53.6615 - 151ms/epoch - 15ms/step
Epoch 69/1000
10/10 - 0s - loss: 63.7764 - val_loss: 54.1908 - 107ms/epoch - 11ms/step
Epoch 70/1000
10/10 - 0s - loss: 63.6102 - val_loss: 52.6200 - 88ms/epoch - 9ms/step
Epoch 71/1000
10/10 - 0s - loss: 62.9163 - val_loss: 52.3956 - 92ms/epoch - 9ms/step
Epoch 72/1000
10/10 - 0s - loss: 62.3272 - val_loss: 51.6602 - 99ms/epoch - 10ms/step
Epoch 73/1000
10/10 - 0s - loss: 63.4992 - val_loss: 51.2628 - 161ms/epoch - 16ms/step
Epoch 74/1000
10/10 - 0s - loss: 62.8709 - val_loss: 50.4873 - 111ms/epoch - 11ms/step
Epoch 75/1000
10/10 - 0s - loss: 60.9686 - val_loss: 51.4177 - 82ms/epoch - 8ms/step
Epoch 76/1000
10/10 - 0s - loss: 59.7037 - val_loss: 49.4762 - 89ms/epoch - 9ms/step
Epoch 77/1000
10/10 - 0s - loss: 59.5827 - val_loss: 49.6083 - 121ms/epoch - 12ms/step
Epoch 78/1000
10/10 - 0s - loss: 59.7795 - val_loss: 48.5477 - 115ms/epoch - 11ms/step
Epoch 79/1000
10/10 - 0s - loss: 58.4787 - val_loss: 48.2142 - 113ms/epoch - 11ms/step
Epoch 80/1000
10/10 - 0s - loss: 58.1287 - val_loss: 48.3080 - 93ms/epoch - 9ms/step
Epoch 81/1000
10/10 - 0s - loss: 57.5325 - val_loss: 47.2556 - 77ms/epoch - 8ms/step
Epoch 82/1000
10/10 - 0s - loss: 56.7754 - val_loss: 47.1473 - 109ms/epoch - 11ms/step
Epoch 83/1000
10/10 - 0s - loss: 56.4003 - val_loss: 46.8065 - 101ms/epoch - 10ms/step
Epoch 84/1000
10/10 - 0s - loss: 56.7061 - val_loss: 45.9990 - 185ms/epoch - 18ms/step
Epoch 85/1000
10/10 - 0s - loss: 56.1603 - val_loss: 45.8791 - 197ms/epoch - 20ms/step
Epoch 86/1000
10/10 - 0s - loss: 55.1062 - val_loss: 45.0677 - 126ms/epoch - 13ms/step
Epoch 87/1000
10/10 - 0s - loss: 54.8889 - val_loss: 44.7583 - 120ms/epoch - 12ms/step
Epoch 88/1000
10/10 - 0s - loss: 54.1313 - val_loss: 44.8168 - 82ms/epoch - 8ms/step
Epoch 89/1000
10/10 - 0s - loss: 53.5392 - val_loss: 44.2517 - 76ms/epoch - 8ms/step
Epoch 90/1000
10/10 - 0s - loss: 53.6703 - val_loss: 44.3647 - 86ms/epoch - 9ms/step
Epoch 91/1000
10/10 - 0s - loss: 53.1194 - val_loss: 43.2339 - 164ms/epoch - 16ms/step
Epoch 92/1000
10/10 - 0s - loss: 52.4162 - val_loss: 43.0597 - 115ms/epoch - 12ms/step
Epoch 93/1000
10/10 - 0s - loss: 52.0441 - val_loss: 42.6480 - 83ms/epoch - 8ms/step
```

```
Epoch 94/1000
10/10 - 0s - loss: 51.5989 - val_loss: 42.4377 - 42ms/epoch - 4ms/step
Epoch 95/1000
10/10 - 0s - loss: 51.3697 - val_loss: 41.9103 - 48ms/epoch - 5ms/step
Epoch 96/1000
10/10 - 0s - loss: 51.3203 - val_loss: 41.6717 - 55ms/epoch - 6ms/step
Epoch 97/1000
10/10 - 0s - loss: 51.1632 - val_loss: 41.1382 - 65ms/epoch - 7ms/step
Epoch 98/1000
10/10 - 0s - loss: 50.2788 - val_loss: 40.7239 - 47ms/epoch - 5ms/step
Epoch 99/1000
10/10 - 0s - loss: 49.7021 - val_loss: 40.8461 - 50ms/epoch - 5ms/step
Epoch 100/1000
10/10 - 0s - loss: 50.6249 - val_loss: 40.8916 - 67ms/epoch - 7ms/step
Epoch 101/1000
10/10 - 0s - loss: 49.9470 - val_loss: 40.2612 - 47ms/epoch - 5ms/step
Epoch 102/1000
10/10 - 0s - loss: 49.3327 - val_loss: 39.3642 - 53ms/epoch - 5ms/step
Epoch 103/1000
10/10 - 0s - loss: 49.4299 - val_loss: 39.4563 - 67ms/epoch - 7ms/step
Epoch 104/1000
10/10 - 0s - loss: 48.4410 - val_loss: 39.5185 - 44ms/epoch - 4ms/step
Epoch 105/1000
10/10 - 0s - loss: 47.7434 - val_loss: 39.1029 - 49ms/epoch - 5ms/step
Epoch 106/1000
10/10 - 0s - loss: 47.3096 - val_loss: 38.3037 - 64ms/epoch - 6ms/step
Epoch 107/1000
10/10 - 0s - loss: 47.3403 - val_loss: 38.1661 - 50ms/epoch - 5ms/step
Epoch 108/1000
10/10 - 0s - loss: 47.3158 - val_loss: 39.3938 - 44ms/epoch - 4ms/step
Epoch 109/1000
10/10 - 0s - loss: 47.2465 - val_loss: 37.4724 - 63ms/epoch - 6ms/step
Epoch 110/1000
10/10 - 0s - loss: 46.1793 - val_loss: 38.2548 - 46ms/epoch - 5ms/step
Epoch 111/1000
10/10 - 0s - loss: 45.9742 - val_loss: 37.9052 - 48ms/epoch - 5ms/step
Epoch 112/1000
10/10 - 0s - loss: 46.8534 - val_loss: 36.6737 - 51ms/epoch - 5ms/step
Epoch 113/1000
10/10 - 0s - loss: 45.6568 - val_loss: 37.2436 - 43ms/epoch - 4ms/step
Epoch 114/1000
10/10 - 0s - loss: 46.1722 - val_loss: 37.9826 - 59ms/epoch - 6ms/step
Epoch 115/1000
10/10 - 0s - loss: 45.0864 - val_loss: 36.1506 - 45ms/epoch - 5ms/step
Epoch 116/1000
10/10 - 0s - loss: 44.5590 - val_loss: 36.2634 - 42ms/epoch - 4ms/step
Epoch 117/1000
10/10 - 0s - loss: 44.0101 - val_loss: 36.1932 - 50ms/epoch - 5ms/step
Epoch 118/1000
10/10 - 0s - loss: 44.5253 - val_loss: 36.1185 - 55ms/epoch - 6ms/step
Epoch 119/1000
10/10 - 0s - loss: 43.6802 - val_loss: 35.3576 - 49ms/epoch - 5ms/step
Epoch 120/1000
10/10 - 0s - loss: 43.8521 - val_loss: 35.2081 - 63ms/epoch - 6ms/step
Epoch 121/1000
10/10 - 0s - loss: 42.8944 - val_loss: 35.2362 - 63ms/epoch - 6ms/step
Epoch 122/1000
10/10 - 0s - loss: 43.0618 - val_loss: 34.6546 - 46ms/epoch - 5ms/step
Epoch 123/1000
10/10 - 0s - loss: 42.5577 - val_loss: 34.5727 - 46ms/epoch - 5ms/step
Epoch 124/1000
10/10 - 0s - loss: 42.0112 - val_loss: 35.2444 - 47ms/epoch - 5ms/step
Epoch 125/1000
```

```
10/10 - 0s - loss: 41.5351 - val_loss: 33.8780 - 50ms/epoch - 5ms/step
Epoch 126/1000
10/10 - 0s - loss: 43.1731 - val_loss: 36.7196 - 42ms/epoch - 4ms/step
Epoch 127/1000
10/10 - 0s - loss: 44.9588 - val_loss: 34.4649 - 67ms/epoch - 7ms/step
Epoch 128/1000
10/10 - 0s - loss: 41.6290 - val_loss: 35.5199 - 74ms/epoch - 7ms/step
Epoch 129/1000
10/10 - 0s - loss: 40.7516 - val_loss: 33.2187 - 43ms/epoch - 4ms/step
Epoch 130/1000
10/10 - 0s - loss: 42.1431 - val_loss: 35.9299 - 65ms/epoch - 6ms/step
Epoch 131/1000
10/10 - 0s - loss: 40.5715 - val_loss: 32.7406 - 45ms/epoch - 4ms/step
Epoch 132/1000
10/10 - 0s - loss: 40.3439 - val_loss: 32.6067 - 71ms/epoch - 7ms/step
Epoch 133/1000
10/10 - 0s - loss: 39.9940 - val_loss: 32.7292 - 47ms/epoch - 5ms/step
Epoch 134/1000
10/10 - 0s - loss: 40.0634 - val_loss: 32.1410 - 48ms/epoch - 5ms/step
Epoch 135/1000
10/10 - 0s - loss: 40.4516 - val_loss: 32.3407 - 69ms/epoch - 7ms/step
Epoch 136/1000
10/10 - 0s - loss: 39.1197 - val_loss: 32.0680 - 61ms/epoch - 6ms/step
Epoch 137/1000
10/10 - 0s - loss: 38.6692 - val_loss: 31.8778 - 51ms/epoch - 5ms/step
Epoch 138/1000
10/10 - 0s - loss: 38.7935 - val_loss: 32.9095 - 59ms/epoch - 6ms/step
Epoch 139/1000
10/10 - 0s - loss: 38.6567 - val_loss: 31.1871 - 54ms/epoch - 5ms/step
Epoch 140/1000
10/10 - 0s - loss: 38.1735 - val_loss: 31.1331 - 63ms/epoch - 6ms/step
Epoch 141/1000
10/10 - 0s - loss: 37.6601 - val_loss: 31.2389 - 44ms/epoch - 4ms/step
Epoch 142/1000
10/10 - 0s - loss: 37.5095 - val_loss: 31.1678 - 62ms/epoch - 6ms/step
Epoch 143/1000
10/10 - 0s - loss: 37.3672 - val_loss: 30.4313 - 61ms/epoch - 6ms/step
Epoch 144/1000
10/10 - 0s - loss: 37.9671 - val_loss: 30.2334 - 50ms/epoch - 5ms/step
Epoch 145/1000
10/10 - 0s - loss: 37.7869 - val_loss: 32.6676 - 61ms/epoch - 6ms/step
Epoch 146/1000
10/10 - 0s - loss: 37.3247 - val_loss: 30.0956 - 49ms/epoch - 5ms/step
Epoch 147/1000
10/10 - 0s - loss: 37.0411 - val_loss: 29.7544 - 63ms/epoch - 6ms/step
Epoch 148/1000
10/10 - 0s - loss: 37.8974 - val_loss: 34.1898 - 55ms/epoch - 6ms/step
Epoch 149/1000
10/10 - 0s - loss: 35.6341 - val_loss: 31.0651 - 50ms/epoch - 5ms/step
Epoch 150/1000
10/10 - 0s - loss: 38.9956 - val_loss: 32.1005 - 52ms/epoch - 5ms/step
Epoch 151/1000
10/10 - 0s - loss: 35.7875 - val_loss: 28.9734 - 65ms/epoch - 7ms/step
Epoch 152/1000
10/10 - 0s - loss: 35.7318 - val_loss: 29.1119 - 48ms/epoch - 5ms/step
Epoch 153/1000
10/10 - 0s - loss: 35.2600 - val_loss: 28.6848 - 65ms/epoch - 6ms/step
Epoch 154/1000
10/10 - 0s - loss: 35.9957 - val_loss: 29.1977 - 42ms/epoch - 4ms/step
Epoch 155/1000
10/10 - 0s - loss: 35.7540 - val_loss: 29.7204 - 72ms/epoch - 7ms/step
Epoch 156/1000
10/10 - 0s - loss: 34.8676 - val_loss: 28.1050 - 44ms/epoch - 4ms/step
```

```
Epoch 157/1000
10/10 - 0s - loss: 34.6044 - val_loss: 29.6049 - 49ms/epoch - 5ms/step
Epoch 158/1000
10/10 - 0s - loss: 34.8734 - val_loss: 27.8684 - 49ms/epoch - 5ms/step
Epoch 159/1000
10/10 - 0s - loss: 34.2168 - val_loss: 27.5564 - 61ms/epoch - 6ms/step
Epoch 160/1000
10/10 - 0s - loss: 34.3384 - val_loss: 27.3708 - 64ms/epoch - 6ms/step
Epoch 161/1000
10/10 - 0s - loss: 33.9496 - val_loss: 28.5652 - 47ms/epoch - 5ms/step
Epoch 162/1000
10/10 - 0s - loss: 33.4599 - val_loss: 27.3174 - 58ms/epoch - 6ms/step
Epoch 163/1000
10/10 - 0s - loss: 33.8438 - val_loss: 27.6048 - 45ms/epoch - 5ms/step
Epoch 164/1000
10/10 - 0s - loss: 33.1440 - val_loss: 26.6754 - 71ms/epoch - 7ms/step
Epoch 165/1000
10/10 - 0s - loss: 33.6024 - val_loss: 28.4600 - 45ms/epoch - 4ms/step
Epoch 166/1000
10/10 - 0s - loss: 33.0155 - val_loss: 26.3480 - 75ms/epoch - 8ms/step
Epoch 167/1000
10/10 - 0s - loss: 33.6239 - val_loss: 27.4919 - 43ms/epoch - 4ms/step
Epoch 168/1000
10/10 - 0s - loss: 33.7240 - val_loss: 28.3199 - 51ms/epoch - 5ms/step
Epoch 169/1000
10/10 - 0s - loss: 33.5670 - val_loss: 25.8991 - 62ms/epoch - 6ms/step
Epoch 170/1000
10/10 - 0s - loss: 31.7415 - val_loss: 26.0897 - 62ms/epoch - 6ms/step
Epoch 171/1000
10/10 - 0s - loss: 31.5303 - val_loss: 25.4196 - 49ms/epoch - 5ms/step
Epoch 172/1000
10/10 - 0s - loss: 31.8498 - val_loss: 26.9220 - 50ms/epoch - 5ms/step
Epoch 173/1000
10/10 - 0s - loss: 31.6775 - val_loss: 25.7945 - 62ms/epoch - 6ms/step
Epoch 174/1000
10/10 - 0s - loss: 31.3026 - val_loss: 25.3169 - 53ms/epoch - 5ms/step
Epoch 175/1000
10/10 - 0s - loss: 30.8672 - val_loss: 25.7407 - 69ms/epoch - 7ms/step
Epoch 176/1000
10/10 - 0s - loss: 31.5805 - val_loss: 24.5653 - 72ms/epoch - 7ms/step
Epoch 177/1000
10/10 - 0s - loss: 31.6575 - val_loss: 24.7597 - 53ms/epoch - 5ms/step
Epoch 178/1000
10/10 - 0s - loss: 31.0366 - val_loss: 26.0440 - 61ms/epoch - 6ms/step
Epoch 179/1000
10/10 - 0s - loss: 30.4223 - val_loss: 24.2567 - 65ms/epoch - 7ms/step
Epoch 180/1000
10/10 - 0s - loss: 29.6591 - val_loss: 24.0481 - 62ms/epoch - 6ms/step
Epoch 181/1000
10/10 - 0s - loss: 29.5555 - val_loss: 23.9970 - 65ms/epoch - 7ms/step
Epoch 182/1000
10/10 - 0s - loss: 29.4170 - val_loss: 23.5796 - 58ms/epoch - 6ms/step
Epoch 183/1000
10/10 - 0s - loss: 29.2324 - val_loss: 24.1796 - 52ms/epoch - 5ms/step
Epoch 184/1000
10/10 - 0s - loss: 28.8220 - val_loss: 23.2315 - 46ms/epoch - 5ms/step
Epoch 185/1000
10/10 - 0s - loss: 29.0582 - val_loss: 24.1037 - 61ms/epoch - 6ms/step
Epoch 186/1000
10/10 - 0s - loss: 28.7244 - val_loss: 22.9213 - 64ms/epoch - 6ms/step
Epoch 187/1000
10/10 - 0s - loss: 28.4226 - val_loss: 23.6453 - 52ms/epoch - 5ms/step
Epoch 188/1000
```

```
10/10 - 0s - loss: 30.4988 - val_loss: 22.3928 - 67ms/epoch - 7ms/step
Epoch 189/1000
10/10 - 0s - loss: 29.2073 - val_loss: 22.8565 - 46ms/epoch - 5ms/step
Epoch 190/1000
10/10 - 0s - loss: 27.8428 - val_loss: 24.4894 - 61ms/epoch - 6ms/step
Epoch 191/1000
10/10 - 0s - loss: 28.7934 - val_loss: 21.9677 - 46ms/epoch - 5ms/step
Epoch 192/1000
10/10 - 0s - loss: 29.0026 - val_loss: 23.3321 - 68ms/epoch - 7ms/step
Epoch 193/1000
10/10 - 0s - loss: 28.7747 - val_loss: 21.6816 - 60ms/epoch - 6ms/step
Epoch 194/1000
10/10 - 0s - loss: 27.4620 - val_loss: 21.3625 - 48ms/epoch - 5ms/step
Epoch 195/1000
10/10 - 0s - loss: 26.9269 - val_loss: 21.5949 - 46ms/epoch - 5ms/step
Epoch 196/1000
10/10 - 0s - loss: 27.0653 - val_loss: 21.2440 - 66ms/epoch - 7ms/step
Epoch 197/1000
10/10 - 0s - loss: 26.5345 - val_loss: 21.7376 - 60ms/epoch - 6ms/step
Epoch 198/1000
10/10 - 0s - loss: 26.6457 - val_loss: 20.7840 - 43ms/epoch - 4ms/step
Epoch 199/1000
10/10 - 0s - loss: 26.1900 - val_loss: 20.5610 - 63ms/epoch - 6ms/step
Epoch 200/1000
10/10 - 0s - loss: 26.4989 - val_loss: 21.2801 - 73ms/epoch - 7ms/step
Epoch 201/1000
10/10 - 0s - loss: 25.7045 - val_loss: 20.3552 - 51ms/epoch - 5ms/step
Epoch 202/1000
10/10 - 0s - loss: 26.2127 - val_loss: 20.3249 - 47ms/epoch - 5ms/step
Epoch 203/1000
10/10 - 0s - loss: 26.5411 - val_loss: 23.3969 - 43ms/epoch - 4ms/step
Epoch 204/1000
10/10 - 0s - loss: 26.0485 - val_loss: 19.6083 - 52ms/epoch - 5ms/step
Epoch 205/1000
10/10 - 0s - loss: 25.7512 - val_loss: 21.4974 - 61ms/epoch - 6ms/step
Epoch 206/1000
10/10 - 0s - loss: 27.9661 - val_loss: 24.6909 - 73ms/epoch - 7ms/step
Epoch 207/1000
10/10 - 0s - loss: 27.1431 - val_loss: 20.3076 - 41ms/epoch - 4ms/step
Epoch 208/1000
10/10 - 0s - loss: 25.5981 - val_loss: 20.7630 - 43ms/epoch - 4ms/step
Epoch 209/1000
10/10 - 0s - loss: 25.4804 - val_loss: 18.8038 - 65ms/epoch - 7ms/step
Epoch 210/1000
10/10 - 0s - loss: 26.6374 - val_loss: 26.9133 - 63ms/epoch - 6ms/step
Epoch 211/1000
10/10 - 0s - loss: 26.2150 - val_loss: 18.8805 - 48ms/epoch - 5ms/step
Epoch 212/1000
10/10 - 0s - loss: 25.7188 - val_loss: 20.6097 - 50ms/epoch - 5ms/step
Epoch 213/1000
10/10 - 0s - loss: 24.9249 - val_loss: 18.8219 - 50ms/epoch - 5ms/step
Epoch 214/1000
Restoring model weights from the end of the best epoch: 209.
10/10 - 0s - loss: 24.0144 - val_loss: 19.2638 - 50ms/epoch - 5ms/step
Epoch 214: early stopping
```

```
Out[3]: <keras.callbacks.History at 0x7f126e090b90>
```

We now evaluate the trained neural network to see the RMSE. We will use this trained neural network to compare the accuracy between the original data and the GAN-generated data. We will later see that you can use such comparisons for anomaly detection. We can use this technique can be used for security systems. If a neural network trained on original data does not perform well on new data, then the new data may be suspect or fake.

In [4]:
```
pred = model.predict(x_test)
score = np.sqrt(metrics.mean_squared_error(pred,y_test))
print("Final score (RMSE): {}".format(score))
```

```
Final score (RMSE): 4.33633936452545
```

Training a GAN for Auto MPG

Next, we will train the GAN to generate fake data from the original MPG data. There are quite a few options that you can fine-tune for the GAN. The example presented here uses most of the default values. These are the usual hyperparameters that must be tuned for any model and require some experimentation for optimal results. To learn more about tabgab refer to its paper or this Medium article, written by the creator of tabgan.

In [5]:
```
from tabgan.sampler import GANGenerator
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

gen_x, gen_y = GANGenerator(gen_x_times=1.1, cat_cols=None,
           bot_filter_quantile=0.001, top_filter_quantile=0.999, \
              is_post_process=True,
           adversarial_model_params={
               "metrics": "rmse", "max_depth": 2, "max_bin": 100,
               "learning_rate": 0.02, "random_state": \
                42, "n_estimators": 500,
           }, pregeneration_frac=2, only_generated_data=False,\
           gan_params = {"batch_size": 500, "patience": 25, \
          "epochs" : 500,}).generate_data_pipe(df_x_train, df_y_train,\
          df_x_test, deep_copy=True, only_adversarial=False, \
          use_adversarial=True)
```

```
Fitting CTGAN transformers for each column:   0%|          | 0/8 [00:00<?,
?it/s]
Training CTGAN, epochs::   0%|          | 0/500 [00:00<?, ?it/s]
```

Note: if you receive an error running the above code, you likely need to restart the runtime. You should have a "restart runtime" button in the output from the second cell. Once you restart the runtime, rerun all of the cells. This step is necessary as tabgan requires specific versions of some packages.

Evaluating the GAN Results

If we display the results, we can see that the GAN-generated data looks similar to the original. Some values, typically whole numbers in the original data, have fractional values in the synthetic data.

In [6]:

```
gen_x
```

Out[6]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	5	296.949632	106.872450	2133	18.323035	73	2
1	5	247.744505	97.532052	2233	19.490136	75	2
2	4	259.648421	108.111921	2424	19.898952	79	3
3	5	319.208637	93.764364	2054	19.420225	78	3
4	4	386.237667	129.837418	1951	20.989091	82	2
...	...	...	...	...	...	...	...
542	8	304.000000	150.000000	3672	11.500000	72	1
543	8	304.000000	150.000000	3433	12.000000	70	1
544	4	98.000000	80.000000	2164	15.000000	72	1
545	4	97.500000	80.000000	2126	17.000000	72	1
546	5	138.526374	68.958515	2497	13.495784	71	1

547 rows × 7 columns

Finally, we present the synthetic data to the previously trained neural network to see how accurately we can predict the synthetic targets. As we can see, you lose some RMSE accuracy by going to synthetic data.

In [7]:

```
# Predict
pred = model.predict(gen_x.values)
score = np.sqrt(metrics.mean_squared_error(pred,gen_y.values))
print("Final score (RMSE): {}".format(score))
```

```
Final score (RMSE): 9.083745225633098
```