Configuration Manual

MSc Research Project
Master of Science in Data Analytics

Bingwei Wang
Student ID: X21111596

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Bingwei Wang
Student ID:	X21111596
Programme:	Master of Science in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Catherine Mulwa
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	8948
Page Count:	57

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Bingwei Wang
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☑
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☑
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Bingwei Wang
x21111596

1. Config Manual Introduction

This configuration manual provides information about the software and hardware tools used in this project implementation. This manual also includes step by step installation of required software and packages, loading the code into systems and execution of project to get the results.

2. Hardware details

Specification

Deep learning and machine learning need a highly configured computer very much. The following hardware configuration is used in my project, which is not the minimum required configuration, but can ensure the normal operation of the whole project.

Hardware

Figure 1: Hardware details

Processor	11th Gen Intel(R) Core (TM) i7-11370H @ 3.30GHz 3.00 GHz
Installed RAM	16.0GB
System type	64-bit operating system, x64-based processor
GPU	None/GTX1060

The operation system information is shown in Table 1, and the version is show in Figure 1.In this project, repeated training of neural networks, machine learning, data visualization analysis and so on are used, so the equipment has high requirements. If there is a good GPU, the speed of operation will be greatly accelerated.

Edition	Windows 11 Home
Version	21H2
Installed	2022/5/11
OS build	22000.795

Figure 2: windows software version

In theory, all the major operating systems which can install python will be able to be used in this project, because Python will be able to use virtual environments. However, if you want good GPU acceleration, such as NVIDIA CUDA, Tensor RT, etc., you will have very different configurations for different operating systems. If you don't have a separate graphics card, you can also use the CPU, but it will take a long time.

3. Required software and dataset

The main development software used in this project is Python3. With Anaconda Navigator, the operation environment can be set up very quickly and conveniently. At the same time, Excel and SPSS are used as auxiliary data analysis software. The following is the software installation for building the project environment.

3.1 Python

Open-source software and available for free download. We used Python 3 during development because it is new and support for newer software. I haven't tested my code with python 2.[1]

Figure 2: python download representational image[1]

As shown in Figure 2, you can download Python from the official website. Once you have downloaded the Python installation package, select the default configuration step by step and install it on your computer. This step is the same as any software installation. Because Python2 and Python3 have many incompatibilities, the current mainstream software development has abandoned Python2, TensorFlow and Keras and other software packages are not compatible with Python2, so in this project, we need to download and install Python3.0 and above versions. Some compilers do not need to download Python, Anoconda or Visual Studio integrate Python, or even choose a different version, so this step is not necessary and can be more personal.

With the rise of artificial intelligence in recent years, python has gained great popularity compared with programming languages such as C++ and Java. Its advantages in artificial intelligence are as follows:

1. simple grammar, less coding
2. Almost all AI project libraries are built in
3. Open source and can be used for extensive programming

[1]Python official download url: `https://www.python.org/downloads/`

3.2 Anaconda Navigator

As it is shown in Figure 3., you can download the package from Anaconda Navigator's website, depending on your operating system.[2]

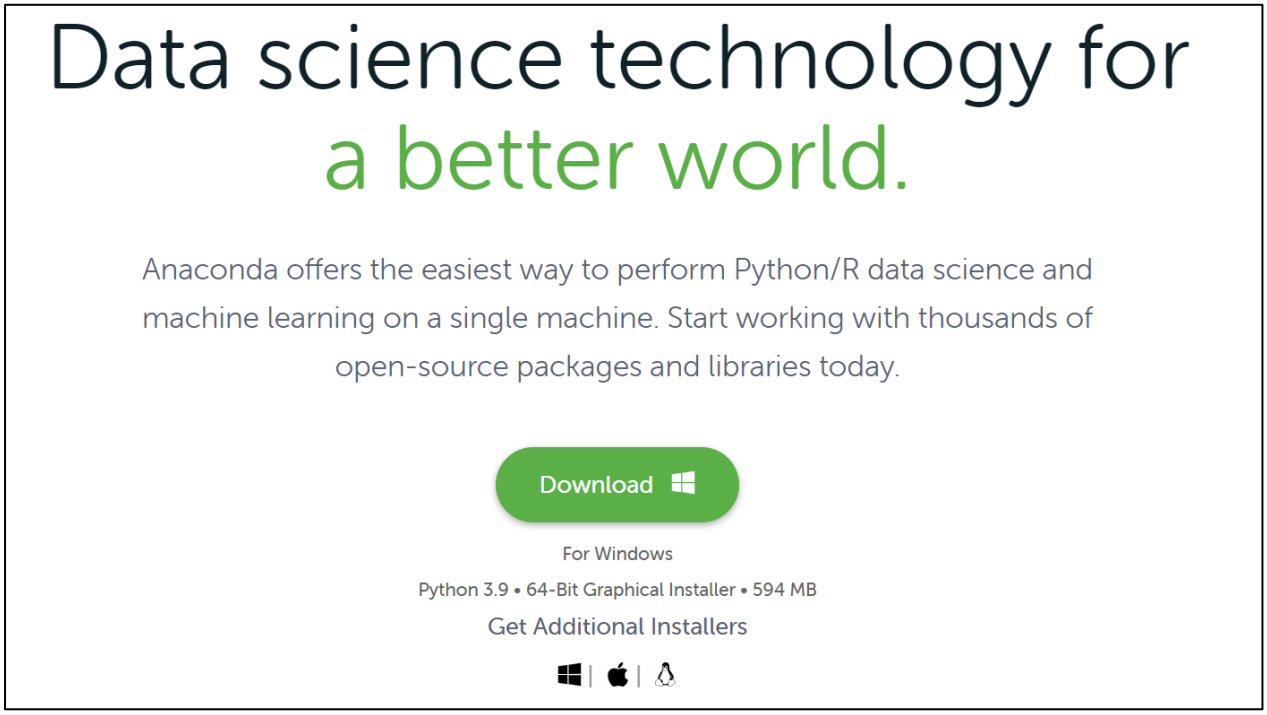

Figure 3: Anaconda download representational image[2]

Anaconda is a Python distribution for scientific computing for Linux, Mac, and Windows, and contains many popular Python packages for scientific computing and data analysis. Install Anaconda, and you don't need to install Python separately.

Anaconda's advantages:

1. Anaconda integrates a lot of third-party libraries about Python scientific computing, which is mainly easy to install. Python is a compiler.

2. Libraries for common scientific computing classes are included, making installation easier than a regular Python installation

3.3 Dataset

For this research, the credit card transactions fraud detection dataset that is readily available on Kaggle (Credit Card Fraud Analysis) is used, which is contained more than 550,000 records. The data contains both a training dataset and a testing dataset. The objective is to utilize the testing dataset to assess the model's performance after it has been trained on the training data set. The training dataset comprises 23 columns that include information about the credit card users' names, genders, localities, and birthdays as well as the merchant, spending category, transaction amount, and time of the credit card transaction. It has been downloaded in the ICT packages[3].

[2]Anaconda official url: https://www.anaconda.com

[3]Dataset details and download url: https://www.kaggle.com/code/nathanxiang/credit-card-fraud-analysis-and-modeling/data

3.4 Compiler

Compiler options include PyCharm, VScode, Visual Studio, IntelliJ IDEA, etc. These run the same as long as the virtual environment for the Anaconda configuration is selected. This article uses Visual Studio 2019 as an example to show the process of running and implementing the entire project.

3.5 Install Software dependency library in Anaconda

The Table below is all the packages in this project:

Table 2. Necessary Packages for python in the project

Num.	Packages and Version	
1.	absl-py	1.1.0
2.	astunparse	1.6.3
3.	cachetools	5.2.0
4.	certifi	2022.6.15
5.	charset-normalizer	2.1.0
6.	cycler	0.11.0
7.	flatbuffers	1.12
8.	fonttools	4.34.4
9.	gast	0.4.0
10.	google-auth	2.9.1
11.	google-auth-oauthlib	0.4.6
12.	google-pasta	0.2.0
13.	grpcio	1.47.0
14.	h5py	3.7.0
15.	idna	3.3
16.	imbalanced-learn	0.9.0
17.	importlib-metadata	4.12.0
18.	joblib	1.1.0
19.	keras	2.9.0
20.	Keras-Preprocessing	1.1.2
21.	kiwisolver	1.4.3
22.	libclang	14.0.1
23.	lightgbm	3.3.2
24.	Markdown	3.4.1
25.	matplotlib	3.5.2
26.	numpy	1.21.6
27.	oauthlib	3.2.0
28.	opt-einsum	3.3.0
29.	packaging	21.3
30.	pandas	1.3.5

31.	Pillow	9.2.0
32.	pip	22.1.2
33.	plotly	5.9.0
34.	protobuf	3.19.4
35.	pyasn1	0.4.8
36.	pyasn1-modules	0.2.8
37.	pyparsing	3.0.9
38.	python-dateutil	2.8.2
39.	pytz	2022.1
40.	requests	2.28.1
41.	requests-oauthlib	1.3.1
42.	rsa	4.8
43.	scikit-learn	1.0.2
44.	scipy	1.7.3
45.	seaborn	0.11.2
46.	setuptools	61.2.0
47.	six	1.16.0
48.	tenacity	8.0.1
49.	tensorboard	2.9.1
50.	tensorboard-data-server	0.6.1
51.	tensorboard-plugin-wit	1.8.1
52.	tensorflow	2.9.1
53.	tensorflow-estimator	2.9.0
54.	tensorflow-io-gcs-filesystem	0.26.0
55.	termcolor	1.1.0
56.	threadpoolctl	3.1.0
57.	typing_extensions	4.3.0
58.	urllib3	1.26.10
59.	Werkzeug	2.1.2
60.	wheel	0.37.1
61.	wincertstore	0.2
62.	wrapt	1.14.1
63.	xgboost	1.6.1
64.	zipp	3.8.1

All the packages are available in Anoconda, but sometimes there are incompatible versions.

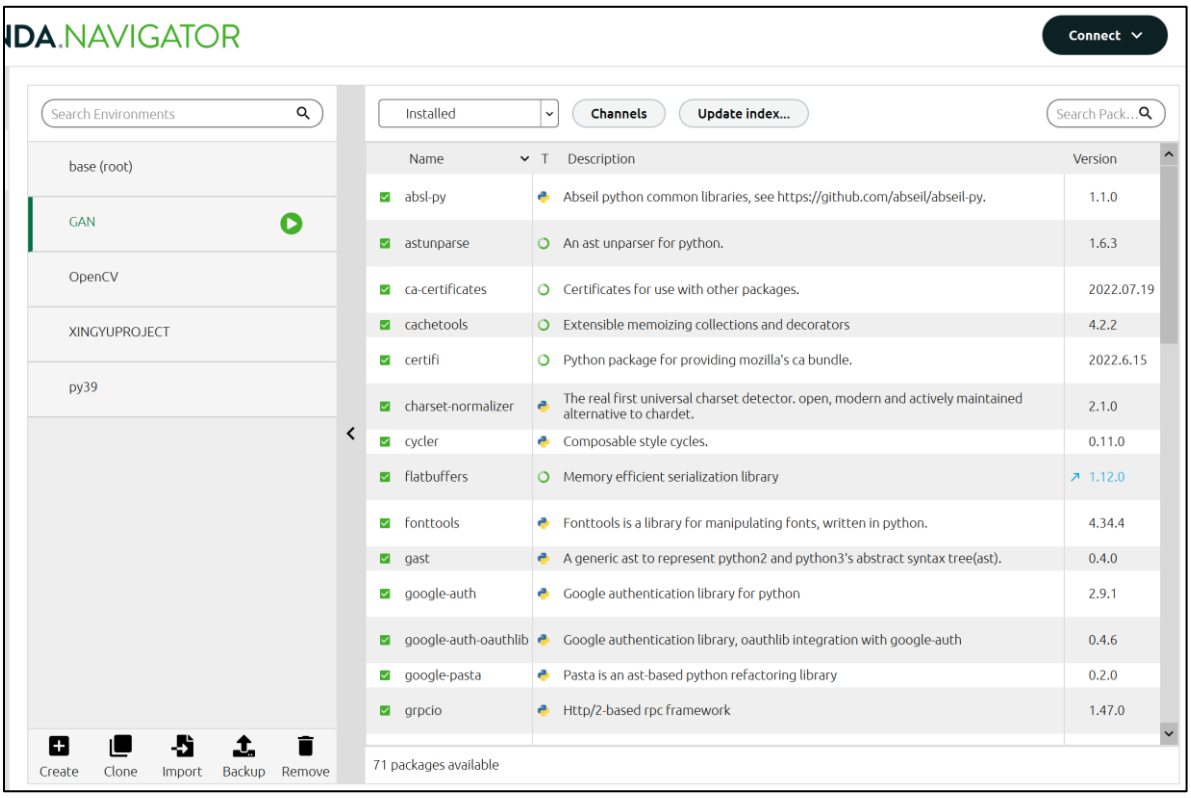

Figure 4. Anaconda Home Page

You can create a unique virtual environment by clicking Creat, and then you can find various software packages on the right side of the interface. However, some software packages have compatibility problems, and this method often fails to install, or the corresponding version cannot be found. So I recommend using Pip install, all the environment packages have been edited in the ICT Codes package, it is named as requirements.txt.

1.	Check Version of conda
	conda --version
2.	View existing virtual environments
	conda info --envs
3.	Update conda package
	conda update conda
4.	Install all packages for this project
	pip install -r requirements.txt

Notes: You should change your path, when you type pip install -r XXXX, XXXX is the full path in your own PC for requirements.txt.

4. Files of Smart Synthetic Data Project

Information of ICT Solution Artefact is shown In Figure 5., these files are included in all data source and some visual output.

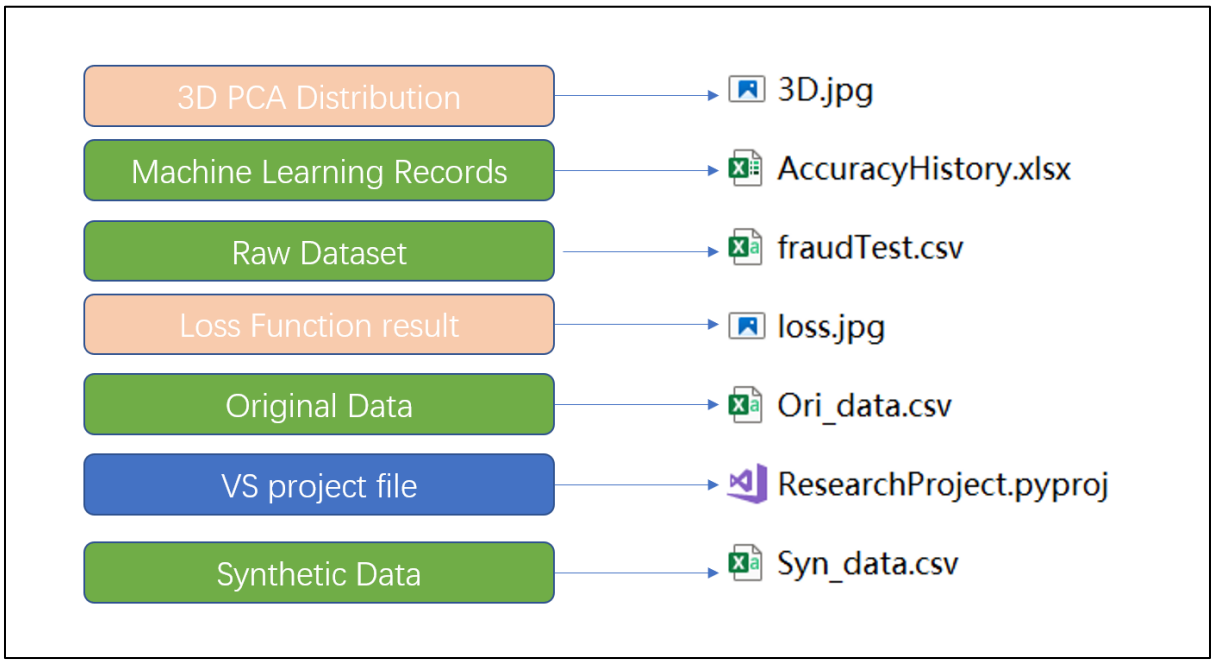

Figure 5. Data and results files in ICT Solution artefact

In Figure 6., these files are included in all code and setting files.

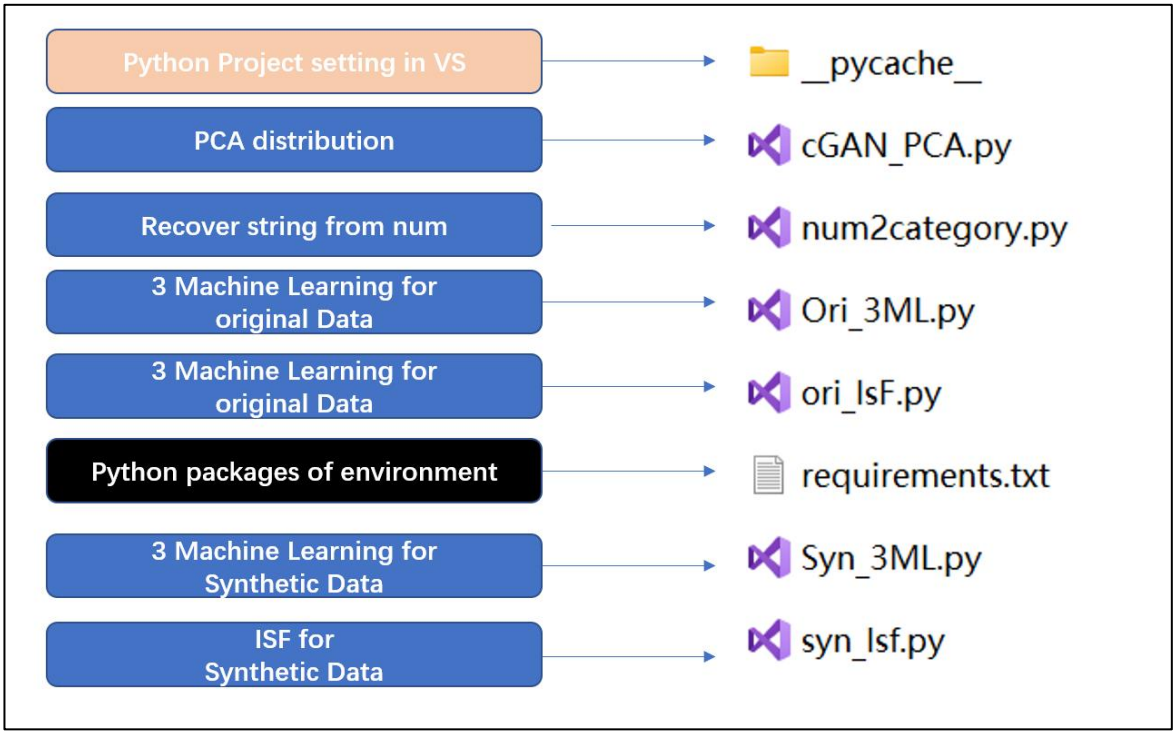

Figure 6. Project code file

Since there is a lot of code, here are a few core code explanations：

A. Data clean：

```
df.isna().sum()# check null data
df.drop_duplicates(inplace=True)# drop duplicate
```

Remove null and duplicate data， which I use panda function here.

B. Data format conversion：

```
df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
```

C. Feature extract:

```
#Feature extract
train=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','
```

D. Discriminator function

```
def dis():
    # define the feature input
    feature = keras.Input(shape=(24,))

    # define the labels  input
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([feature, labels])

    # add one hidden layer
    model = keras.layers.Dense(200)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)#LeakyReLU

    model = keras.layers.Dense(200)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # add the output layer
    model = keras.layers.Dense(1, activation='sigmoid')(model)

    # create a model from the pipeline
    d_model = keras.Model(inputs=[feature, labels], outputs=model, name='discriminator')

    # compile the model
    d_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002), loss='binary_crossentropy',
                    metrics=['accuracy'])

    # return the model
    return d_model
```

The 2 hidden layer is set as 200x200, 24 features is extracted here.

Sigmoid is used for the the activation function.

LeakyreLU is used for loss function. The reason is explained in the report.

E. Generator Function:

```
def gen():
    # define the latent space
    latent = keras.Input(shape=(99,))

    # define the label vector
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([latent, labels])

    # create 1 hidden layer   1-1000
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # model = keras.layers.Dense(128)(merge)
    # model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 2 hidden layer
    model = keras.layers.Dense(200)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 3 hidden layer
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create an ouptput layer
    model = keras.layers.Dense(24, activation='tanh')(model)

    # create a model from the pipeline
    g_model = keras.Model(inputs=[latent, labels], outputs=model, name='generator')

    # return the model
    return g_model
```

The 3 hidden layer is set as 150x200x150, 24 features is extracted here.
Tanh is used for the activation function.
LeakyreLU is used for loss function. The reason is explained in the report.

F. Integrate generators and discriminators

```
def gan(discriminator, generator):
    # Make discriminator as non-trainable
    discriminator.trainable = False

    # get inputs from the generator
    gen_latent, gen_labels = generator.input

    # get output from the generator
    gen_feature = generator.output

    # get discriminator predictions
    dis_predict = discriminator([gen_feature, gen_labels])

    # create a model from the pipeline
    cgan_model = keras.Model(inputs=[gen_latent, gen_labels], outputs=dis_predict, name='cGAN')

    # compile the model
    cgan_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002), loss='binary_crossentropy')

    # return the model
    return cgan_model
```

G. Training of generators and discriminators

```python
def train(discriminator, generator, cgan, batch_size=512, epochs=6):

    count = 0
    half_batch = int(batch_size / 2)
    batch_per_epoch = int(X.shape[0] / batch_size)

    d_real = []
    d_fake = []
    g = []
    for i in range(epochs):
        for j in range(batch_per_epoch):

            # real
            [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
            # loss on real
            d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
            # fake
            [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
                                                                    half_batch)
            # loss on fake
            d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)

            # Generate latent points
            [X_lat, y_lat] = generate_latent_point(batch_size)
            # Give them '1' labels
            y_labels_lat = np.ones((batch_size, 1))
            # calculate generator loss
            gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)

            # print the results
            count += 1
            if count % 1000 == 0:
                print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'.format(count, d1_loss, d2_loss, gan_loss))

            # store the losses
            d_real.append(d1_loss)
            d_fake.append(d2_loss)
            g.append(gan_loss)

    # plot a summary of loss
    plot_history(d_real, d_fake, g)
```

H. PCA 3D distribution

```python
# PCA
pca = PCA(n_components = 3)
X = pca.fit_transform(X)
X=X[0:100000]
# plot the 'real' fraud data
fig = plt.figure(figsize = (15,15))
axs = plt.axes(projection = '3d')
for i in range(X.shape[0]) :
    if y[i] == 1 :
        axs.scatter3D(X[i,0], X[i,1], X[i,2], color = 'orange')

# Synthetic data from stand-alone generator
[X_syn, y_syn] = generate_new_samples(500)

# PCA
pca = PCA(n_components = 3)
X_syn = pca.fit_transform(X_syn)

# plot
for i in range(X_syn.shape[0]) :
    if y_syn[i] == 1.0 :
        axs.scatter3D(X_syn[i,0], X_syn[i,1], X_syn[i,2], color = 'blue')
plt.title('Scatter plot of real and synthetic fraudulent samples')
plt.savefig('3D.jpg')
plt.show()
```

More detailed information can be found in the code, which is commented on almost every line.

5. Run the code in the project

6.1 Step 1. Import Project files

First you need to unzip the ICT Solution Artefact folder and then import the file into your compiler. As shown in Figure 7., you can create a Python Application.

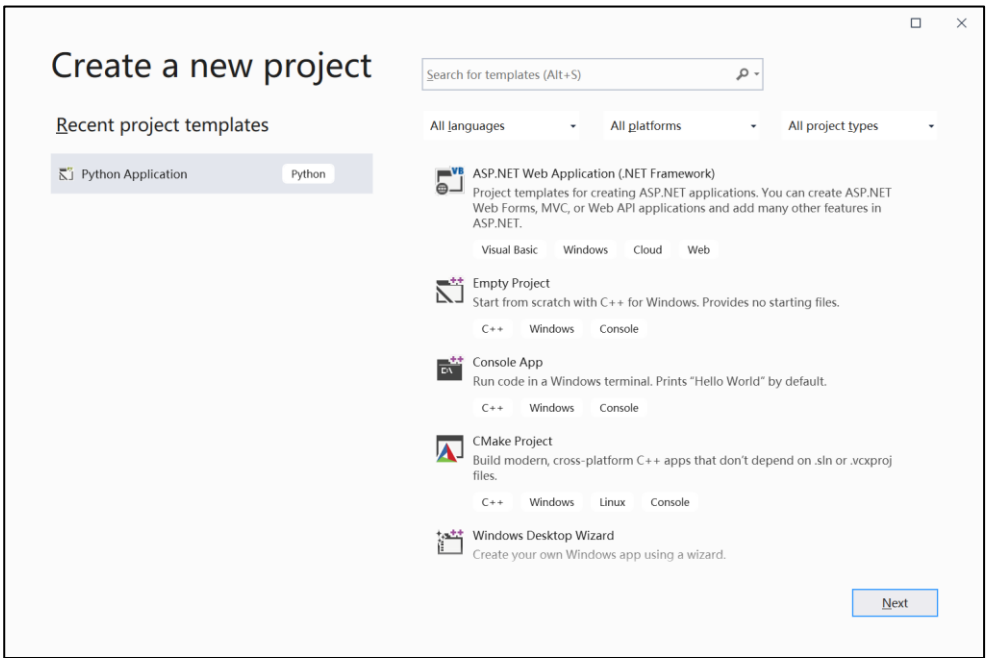

Figure 7. Home Page of Visual Studio

Click Next here:

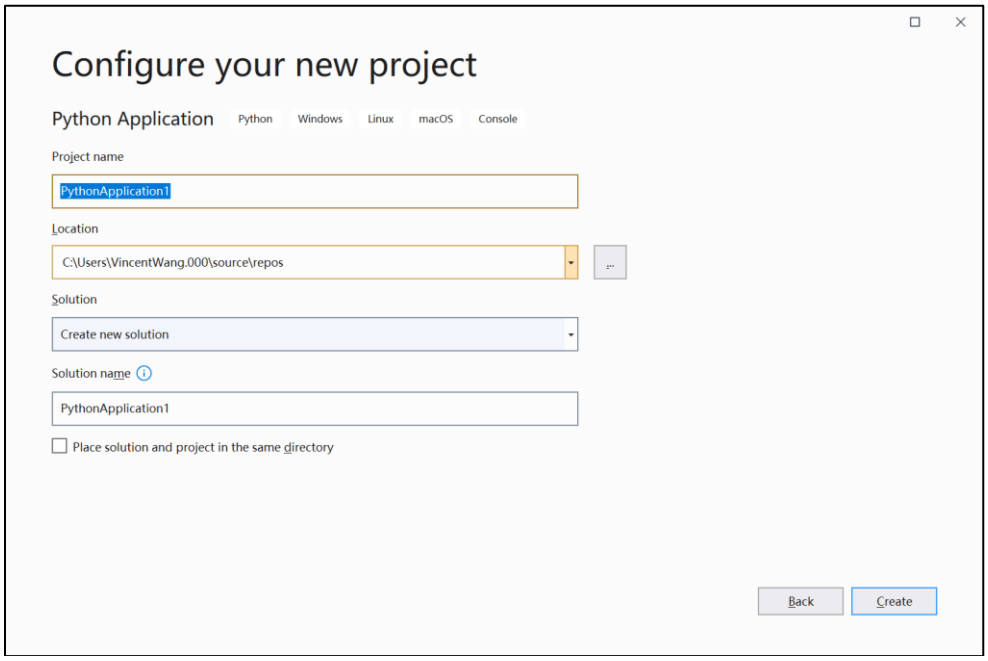

Figure 8. Create a Python Project

Click Next here, as shown in Figure 8.

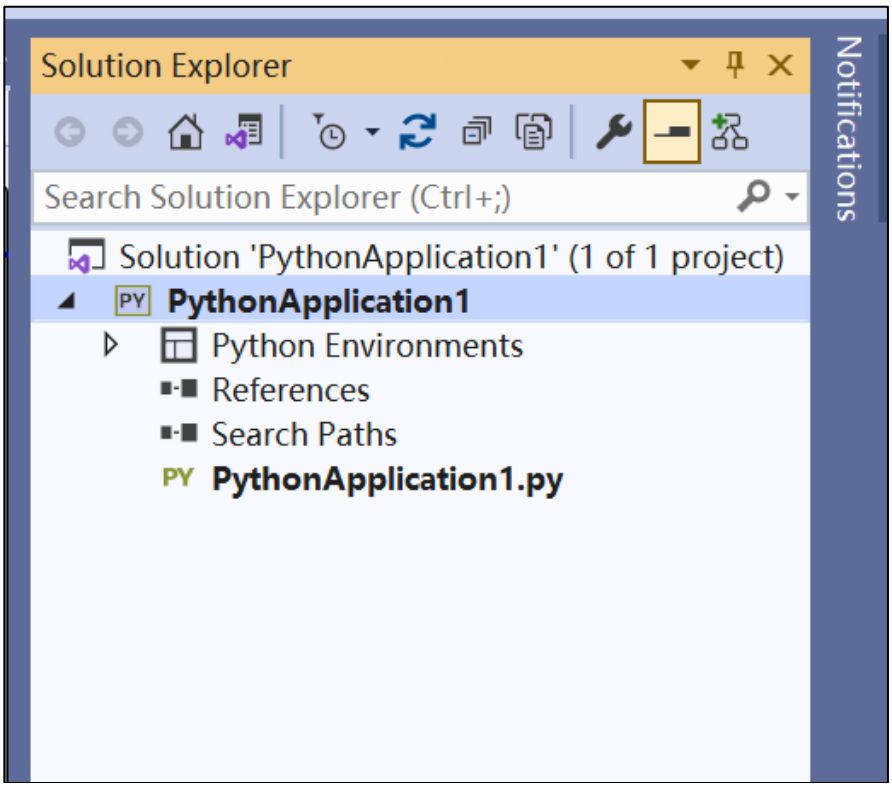

Figure 9. Project solution file list in Visual Studio

Than right click the solution file named PythonApplication(You can give the porject a new name), selected add->Existing Item.

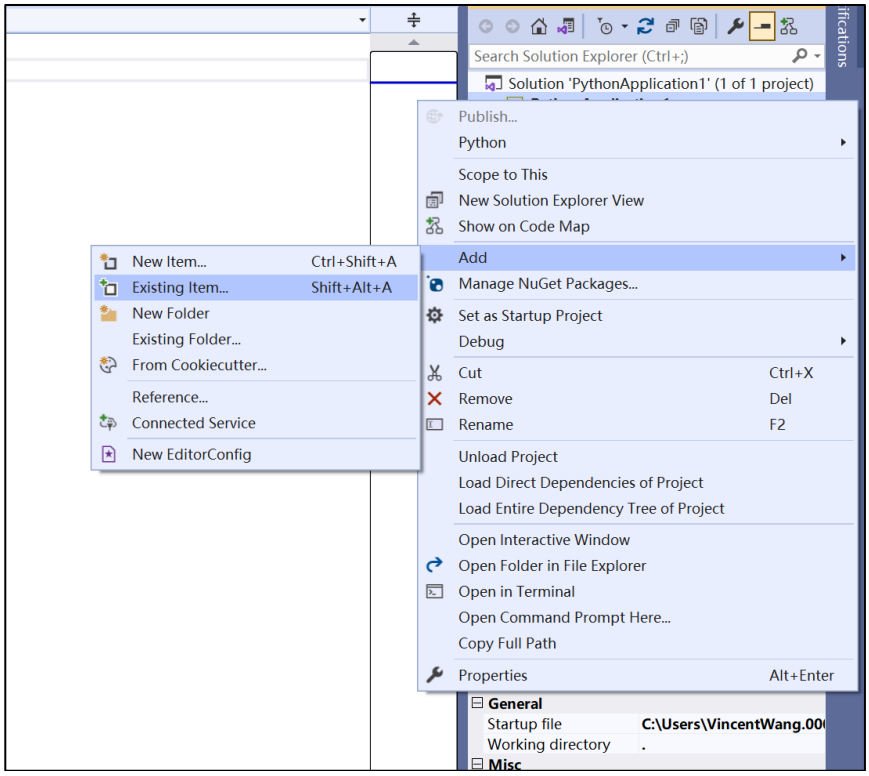

Figure 10. Import items to VS project.

After you finish the step in figure 10., right now you can import all files to our python solution files.

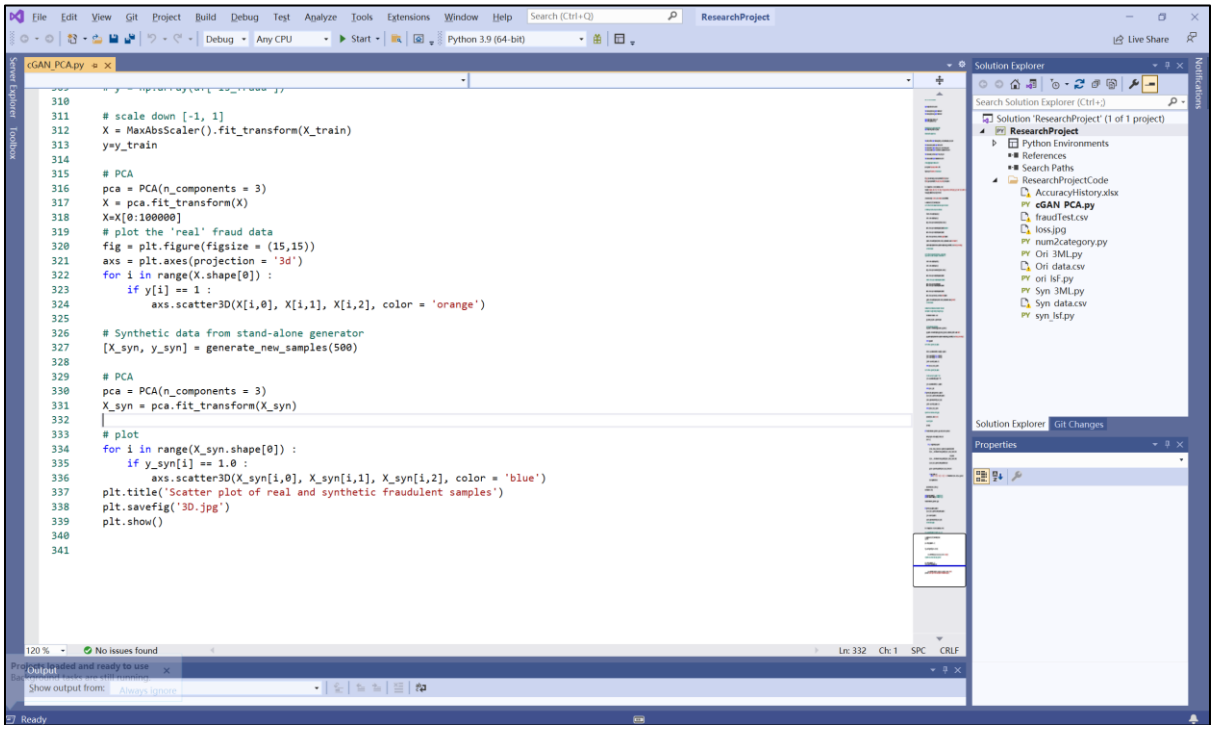

Figure 11. Files list system in VS.

You can see all files under your solution file, as shown in figure 11.

6.2 Step 2. Import the Anaconda environment to compiler

If you finish the process in section 3.5, you can select your own environment for this project.

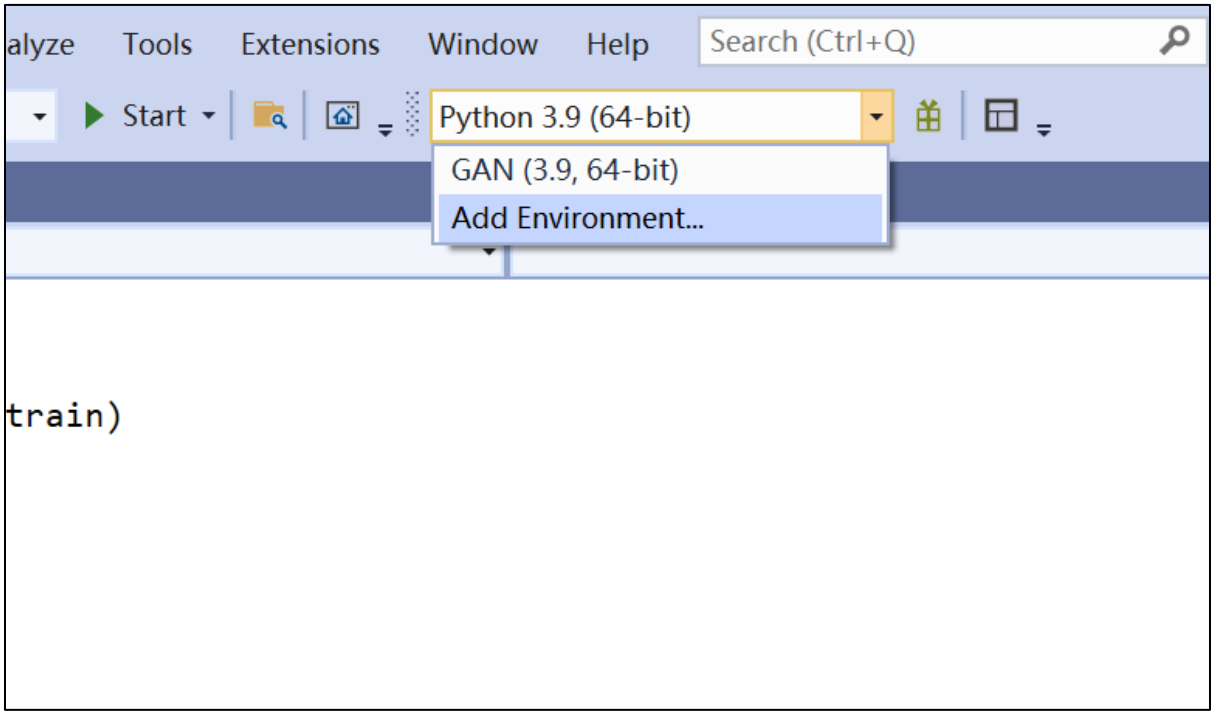

Figure 12. Set environment for python in VS.

As shown in Figure 12.and 13, you can add environmental here, and add the existing environment as you created in section 3.5.

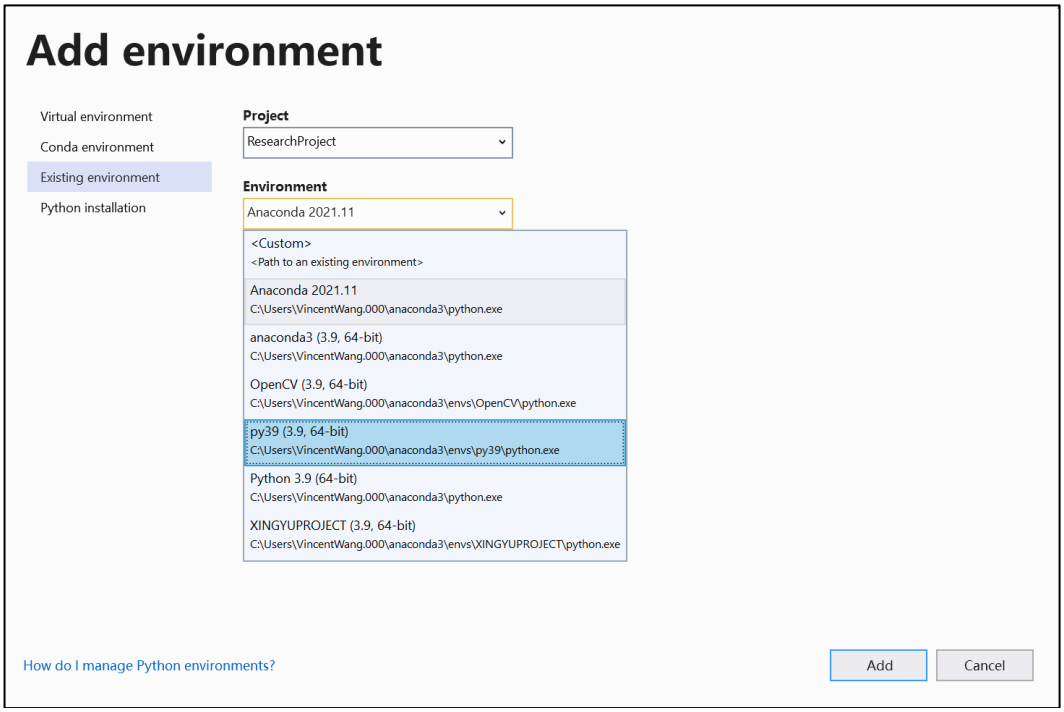

Figure 13. Select an environment created by Anaconda.

As long as you make sure all the packages are installed in 3.5, you can run all the code in this project. You can use the conda list command to view the installed software version.

6.3 Step 3. Run the code for each section

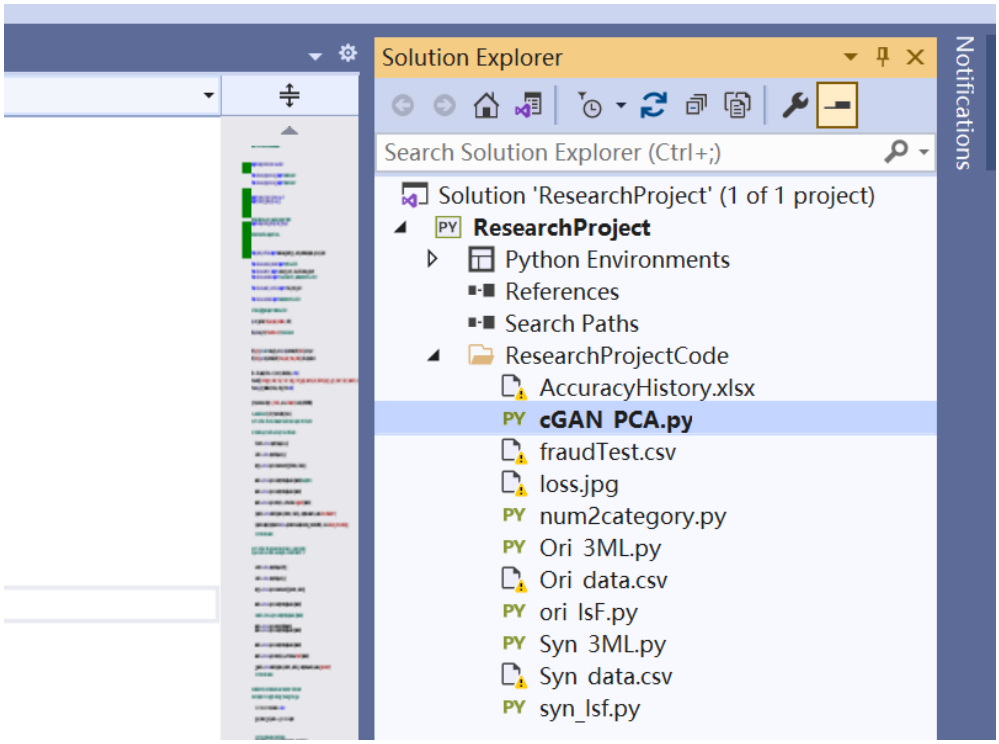

Figure 14. Set a startup file in VS

Python does not require precompilation, so it does not require main functions

As shown in Figure 14. and Figure 15., select a python file(xxx.py), right click it, Set as Startup File.

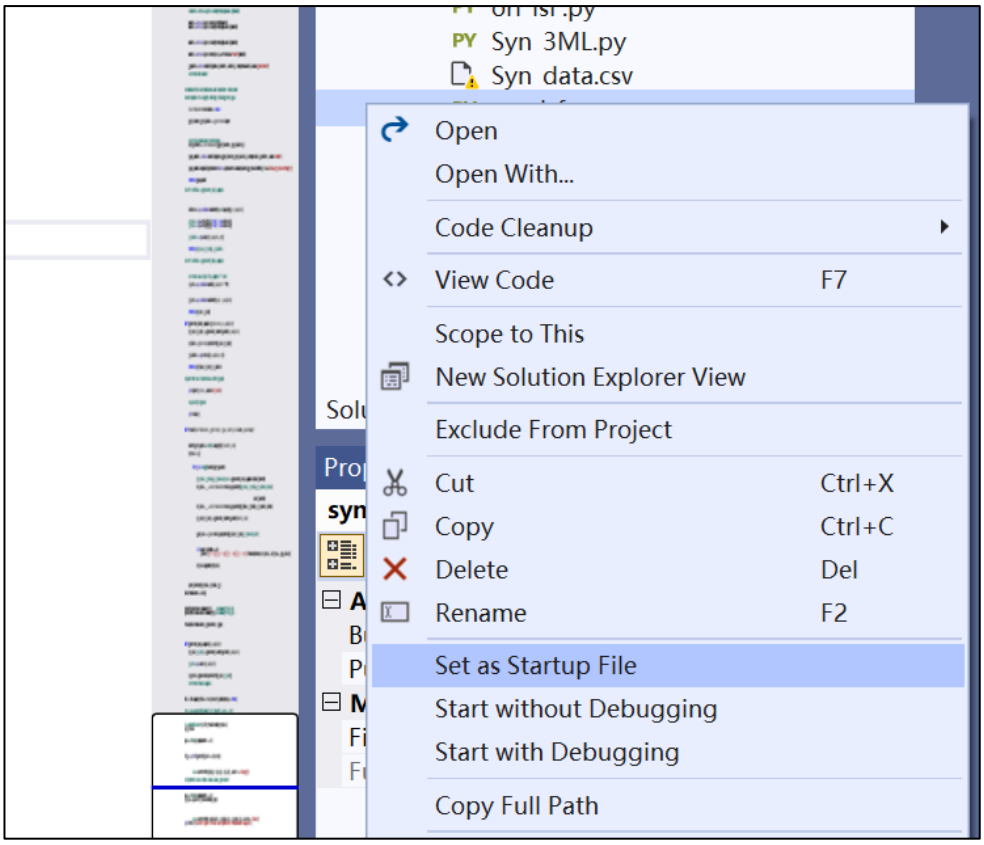

Figure 15. Set a startup file in VS

Finally, you can run all the code in this project. Click the start here, as shown in Figure 16..

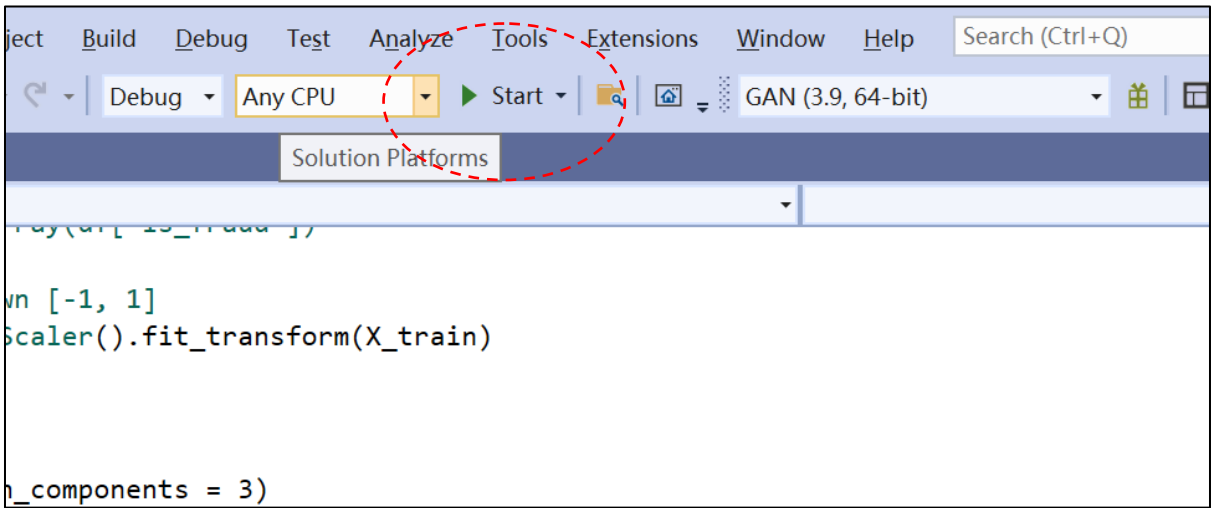

Figure 16. Run a python file in VS.

6. The corresponding result of Smart Synthetic Data project

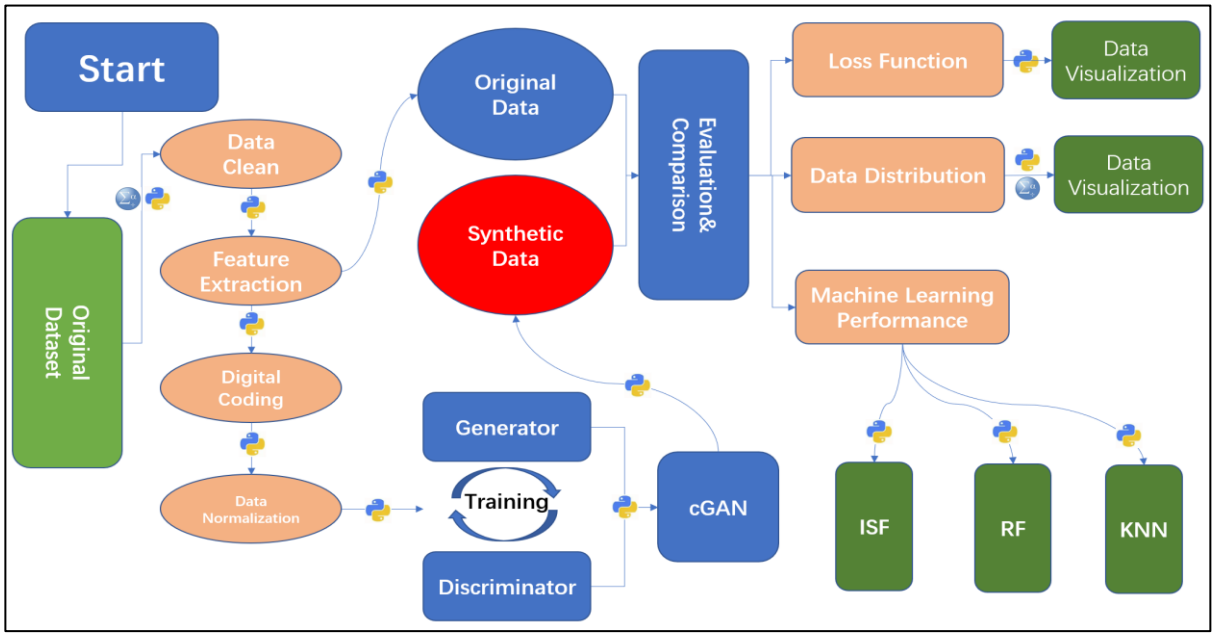

Figure 17. The implementation process flow chat of the whole project

In this study, an original data set containing user information was used and pushed into the generator and generator of cGAN after a series of data pre-processing. The cGAN model was trained and synthesized, and the data was generated and synthesized. Finally, the research topic is verified by comparing the model's loss function, data distribution, and machine learning performance. All process is shown in Figure 17.

The results of the software output can be obtained from:

1. Loss function of cGAN model, as shown in Figure 18:
   Run cGAN_PCA.py, Syn_IsF.py and Syn_3ML.py

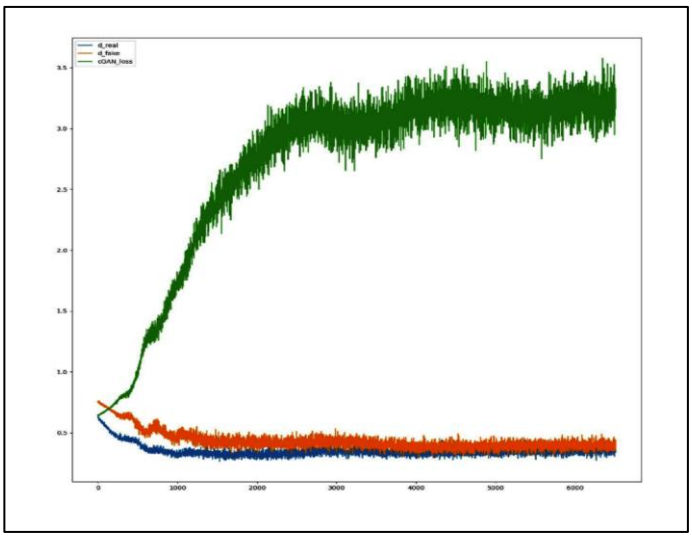

Figure 18. An example of Loss function

2. Analysis for data distribution, as shown in Figure 19.
   Run cGAN_PCA.py and analysis from SPSS

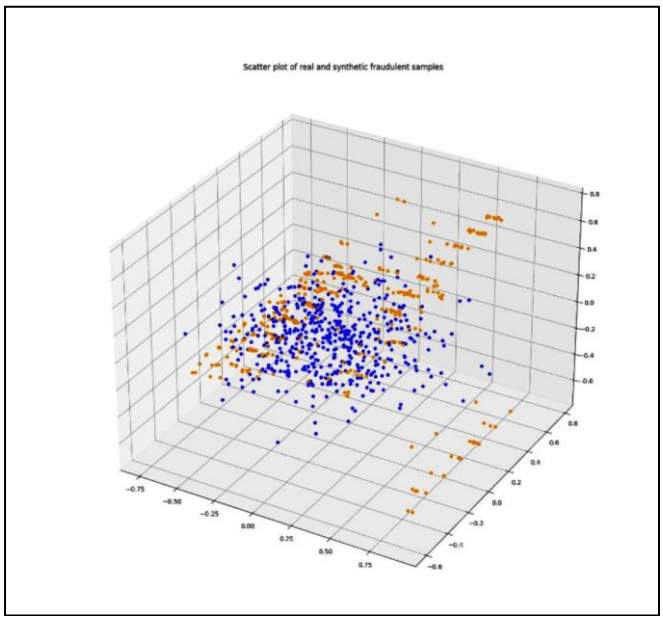

Figure 19. An example of PCA 3D distribution

3. Machine learning performance, As show in Figure 18.
   Run Syn_3ML.py, Ori_3ML.py, ori_Isf.py, Syn_IsF.py
   Output records can be obtained from the command, while the code is running.

```
IsF : 0.7714672075726843
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       999
           1       0.70      0.51      0.59       480

    accuracy                           0.77      1479
   macro avg       0.75      0.70      0.72      1479
weighted avg       0.76      0.77      0.76      1479
```

Figure 18. An example of accuracy of machine learning for synthetic data.

7. Appendix-code

7.1 cGAN_PCA.PY

This part mainly completes the generation of PCA 3D distribution map and the generation of loss function image. Due to the large amount of generated data, when more than 1000 groups of data are generated, it is difficult to display on the 3D image, so this part is separated separately.

```
#### PCA visualization#######
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import plotly.express as px
# for scaling
from sklearn.preprocessing import MaxAbsScaler
import datetime as dt
from sklearn.preprocessing import RobustScaler
import seaborn as sns
#%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.graph_objs as go
import plotly
import seaborn as sns
import plotly.express as px
from collections import Counter
#from imblearn.over_sampling import SMOTE
import matplotlib.gridspec as gridspec
import plotly.figure_factory as ff
# For GANs
#from tensorflow import keras
import keras
# for PCA
from sklearn.decomposition import PCA
from plotly import tools
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
#init_notebook_mode(connected=True)
# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

```
# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier
import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
# import lightgbm as lgb
# from lightgbm import LGBMClassifier
#import xgboost as xgb

pd.set_option('display.max_columns', 100)

#sns.set(rc = {'figure.figsize':(15,8)})
df=pd.read_csv('fraudTest.csv')#read dataset
#Data Clean
df.isna().sum()# check null data
df.drop_duplicates(inplace=True)# drop duplicate
df.info()
df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month

# Shuffle the dataframe df
df = df.sample(frac = 1).reset_index(drop = True)
#Feature extract
train=df[['category','amt','zip','lat','long','city_pop','merch_lat',
'merch_long','age','hour','day','month','is_fraud']]
#convert category to dummy variables
train=pd.get_dummies(train, drop_first=True)

y_train=train['is_fraud'].values[0:300000]
y=y_train
X_train=train.drop("is_fraud", axis='columns').values[0:300000]

# scale b/w [-1, 1]
X = MaxAbsScaler().fit_transform(X_train)

# Let's define the discriminator which takes inputs the feature
```

```
# matrix and class vector and predicts the probability of being
# fake or real (0 or 1)
# 2 Hidden Layer with each layer has 200 nodes
def dis():
    # define the feature input
    feature = keras.Input(shape=(24,))

    # define the labels  input
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([feature, labels])

    # add one hidden layer
    model = keras.layers.Dense(200)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)#LeakyReLU

    model = keras.layers.Dense(200)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # add the output layer
    model = keras.layers.Dense(1, activation='sigmoid')(model)

    # create a model from the pipeline
    d_model = keras.Model(inputs=[feature, labels], outputs=model,
     name='discriminator')

    # compile the model
    d_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002),
    loss='binary_crossentropy',metrics=['accuracy'])

    # return the model
    return d_model

# Let's define the generator which takes as input latent
# space and class labels and ouputs a feature matrix 'X'
# generator has 3 hidden layer with 150x200x150 nodes
def gen():
    # define the latent space
    latent = keras.Input(shape=(99,))

    # define the label vector
```

```
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([latent, labels])

    # create 1 hidden layer   1-1000
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # model = keras.layers.Dense(128)(merge)
    # model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 2 hidden layer
    model = keras.layers.Dense(200)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 3 hidden layer
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create an ouptput layer
    model = keras.layers.Dense(24, activation='tanh')(model)

    # create a model from the pipeline
    g_model = keras.Model(inputs=[latent, labels],
    outputs=model, name='generator')

    # return the model
    return g_model

# Combine the discriminator and Generator into cGAN
# Keep discriminator as non-trainable, so it does
# not update its weights during training the cgan.
def gan(discriminator, generator):
    # Make discriminator as non-trainable
    discriminator.trainable = False

    # get inputs from the generator
    gen_latent, gen_labels = generator.input

    # get output from the generator
    gen_feature = generator.output
```

```

    # get discriminator predictions
    dis_predict = discriminator([gen_feature, gen_labels])

    # create a model from the pipeline
    cgan_model = keras.Model(inputs=[gen_latent, gen_labels],
    outputs=dis_predict, name='cGAN')

    # compile the model
    cgan_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002),
    loss='binary_crossentropy')

    # return the model
    return cgan_model

# Let's define a generate_real_samples
# method to generate random samples as
# real data
def generate_real_samples(n_samples):
    # generate n_samples number of random indices
    indices = np.random.randint(0, X.shape[0], n_samples)

    # get the newly indexed matrix and vector
    X_real = np.array([X[j] for j in indices])
    y_real = np.array([y[j] for j in indices])

    # classify them as '1' (real)
    y_labels = np.ones((n_samples, 1))

    # return the computed arrays
    return [X_real, y_real], y_labels

# Let's define a generate_fake_samples
# method to generate random samples as
# fake data from latent sapce
def generate_latent_point(n_samples):
    # create an array of n_samples * late
    # -nt_space (= 99)
    Z_lat = np.random.randn(n_samples * 99)
    X_lat = Z_lat.reshape(n_samples, 99)

    # randomly generate the class labels 'y'
```

```
    y_lat = np.random.randint(0, 2, n_samples)

    # return the points
    return [X_lat, y_lat]

def generate_fake_samples(generator, n_samples):
    # generate n_samples points in latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # predict on them using the generator
    X_fake = generator.predict([X_lat, y_lat])

    # classify them as '0' (fake)
    y_labels = np.zeros((n_samples, 1))

    # return the computed arrays
    return [X_fake, y_lat], y_labels

# plot the loss functions on the graph
def plot_history(d_real, d_fake, g):
    fig = plt.figure(figsize=(15, 15))
    plt.plot(d_real, label='d_real')
    plt.plot(d_fake, label='d_fake')
    plt.plot(g, label='cGAN_loss')
    # specify legend
    plt.legend()
    # plot the graph
    plt.show()

# train the model
def train(discriminator, generator, cgan, batch_size=256, epochs=6):

    count = 0
    half_batch = int(batch_size / 2)
    batch_per_epoch = int(X.shape[0] / batch_size)

    d_real = []
    d_fake = []
    g = []
    for i in range(epochs):
        for j in range(batch_per_epoch):

```

```
        # real
        [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
        # loss on real
        d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
        # fake
        [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
                                                half_batch)
        # loss on fake
        d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)

        # Generate latent points
        [X_lat, y_lat] = generate_latent_point(batch_size)
        # Give them '1' labels
        y_labels_lat = np.ones((batch_size, 1))
        # calculate generator loss
        gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)

        # print the results
        count += 1
        if count % 1000 == 0:
            print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'
            .format(count, d1_loss, d2_loss, gan_loss))

        # store the losses
        d_real.append(d1_loss)
        d_fake.append(d2_loss)
        g.append(gan_loss)

    # plot a summary of loss
    plot_history(d_real, d_fake, g)

discriminator = dis()
generator = gen()
cgan = gan(discriminator, generator)

print(generator.summary())     # summary for dis
print(discriminator.summary()) # summary for gen
print(cgan.summary())          # summary for gan
# train the cGAN model
train(discriminator, generator, cgan)

# generate the synthetic fraudulent
```

```
# data
def generate_new_samples(n_samples):
    # get n_samples for latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # specify the labels
    y_lat = np.ones(n_samples)

    # predict on generator
    X_pred = generator.predict([X_lat, y_lat])

    # return fraud samples
    return [X_pred, y_lat]

# shuffle the dataframe df
df = df.sample(frac = 1.0).reset_index(drop = True)
# get 'X' and 'y'
# X = np.array(df.drop(['is_fraud'], axis = 1))
# y = np.array(df['is_fraud'])
# scale down [-1, 1]
X = MaxAbsScaler().fit_transform(X_train)
y=y_train
# PCA
pca = PCA(n_components = 3)
X = pca.fit_transform(X)
X=X[0:100000]
# plot the 'real' fraud data
fig = plt.figure(figsize = (15,15))
axs = plt.axes(projection = '3d')
for i in range(X.shape[0]) :
    if y[i] == 1 :
        axs.scatter3D(X[i,0], X[i,1], X[i,2], color = 'orange')

# Synthetic data from stand-alone generator
[X_syn, y_syn] = generate_new_samples(500)

# PCA
pca = PCA(n_components = 3)
X_syn = pca.fit_transform(X_syn)

# plot
for i in range(X_syn.shape[0]) :
```

```
    if y_syn[i] == 1.0 :
        axs.scatter3D(X_syn[i,0], X_syn[i,1], X_syn[i,2], color = 'blue')
plt.title('Scatter plot of real and synthetic fraudulent samples')
plt.savefig('3D.jpg')
plt.show()
```

7.2 num2category.py

Due to the standardization of data, all data are converted into data between [-1,1], so this part is used for data recovery and numbers are converted into strings.

```
###recover string from num #####
import numpy as np
import pandas as pd

def num2cate(m):
    num_list=[]
    List1 = ['personal_care', 'health_fitness', 'misc_pos', 'travel', 'kids_pets',
            'shopping_pos', 'food_dining', 'home', 'entertainment', 'shopping_net',
            'misc_net', 'grocery_pos', 'gas_transport', 'grocery_net']
    List2 = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
    category = dict(zip(List2, List1))
    for i in m:
        if i<0:
            i=0
        if i>12:
            i=12
        a=category[str(round(i))]
        num_list.append(a)
    return num_list
```

7.3 Ori_3ML.py

This part is the prediction result of the original data in machine learning in 3. They are ISF,RF and KNN.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import plotly.express as px
from sklearn.decomposition import PCA
# for scaling
```

```
from sklearn.preprocessing import MaxAbsScaler
import datetime as dt
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
import seaborn as sns
# For GANs
#from tensorflow import keras
import keras
from plotly import tools
import num2category as num2cate
# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
# for scaling
from sklearn.preprocessing import MaxAbsScaler,StandardScaler
import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
# import lightgbm as lgb
# from lightgbm import LGBMClassifier
# import xgboost as xgb

pd.set_option('display.max_columns', 100)

#sns.set(rc = {'figure.figsize':(15,8)})
def get_data():
    df=pd.read_csv('fraudTest.csv')
    df.isna().sum()
    df.drop_duplicates(inplace=True)

    df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
    df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
    df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
    df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
```

```
    df = df.sample(frac = 1).reset_index(drop = True)
    df=df[['category','amt','zip','lat','long','city_pop',
    'merch_lat','merch_long','age','hour','day','month','is_fraud']]
    #convert category to dummy variables，
    #df=pd.get_dummies(df, drop_first=True)#one-hot
    num, index = pd.factorize(df['category'])
    df['category'] = num
    # Extract from the dataframe, class 1s and 0s
    df_1 = df[df['is_fraud'] == 1].sample(frac = 1.0).reset_index(drop = True)
    df_2 = df[df['is_fraud'] == 0].sample(frac = 1.0).reset_index(drop = True)
    # Split each dataframe to certain fraction
    new_df_1, old_df_1 = df_1[: 480].reset_index(drop = True), df_1[480 :].reset_index(drop = True)
    new_df_2, old_df_2 = df_2[: 999].reset_index(drop = True), df_2[999 :].reset_index(drop = True)

    # group them into test and train sets
    test, train = pd.concat([new_df_1, new_df_2]), pd.concat([old_df_1, old_df_2])
    test, train = test.sample(frac = 1.0).reset_index(drop = True),
     train.sample(frac = 1.0).reset_index(drop = True)

    X_train = np.array(train.drop(['is_fraud'], axis = 1))
    y_train = np.array(train['is_fraud'])

    X_test = np.array(test.drop(['is_fraud'], axis = 1))
    y_test = np.array(test['is_fraud'])

    X_train= StandardScaler().fit_transform(X_train)
    X_test = StandardScaler().fit_transform(X_test)

    return X_train,y_train,X_test,y_test

def MLP(X, y, param):
    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    # hyper-parameters
    nodes = param['nodes']
    lrate = param['lrate']
    toler = param['toler']
    batch = param['batch_size']
```

```

    # define the model
    model = MLPClassifier(hidden_layer_sizes=(nodes,), tol=toler,
    batch_size=batch, learning_rate_init=lrate,
                        verbose=0, random_state=1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('MLP : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['MLP'] = model

def RFR(X, y, split):
    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    # define the model
    model = RandomForestClassifier(n_estimators=200, criterion='gini',
     min_samples_split=split, random_state=1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('RFR : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['RFR'] = model

def KNN(X, y):
    # train test split
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.99, random_state=1)

    # define the model
    model = KNeighborsClassifier()

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('KNN : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['KNN'] = model

if __name__=='__main__':
    models_dict = {}

    def IsF(X, y, ratio):
        # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

        # define the model
        model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)

        # train the model
        model.fit(X_train, y_train)

        # predict on the test set
        predictions = model.predict(X_test)

        # Convert the predictions according to problem profile
        predictions[predictions == 1] = 0
        predictions[predictions == -1] = 1

        # evaluate the accuracy and classification report
        print('IsF : ' + str(accuracy_score(y_test, predictions)))
        print(classification_report(y_test, predictions))

```

```
        # assign this to models_dict
        models_dict['IsF'] = model

    X_train, y_train, X_test, y_test=get_data()
    # For Isolation Forest Model
    ratio = float(y_train[y_train == 1].shape[0] / y_train.shape[0])
    # For Multi Layer Perceptron
    param = {
        'nodes': 170,
        'lrate': 0.00005,
        'toler': 0.00001,
        'batch_size': 100
    }
    # For Random Forest Classifier
    split = 2
    IsF(X_train, y_train, ratio)
    pred_1 = models_dict['IsF'].predict(X_test)
    pred_1[pred_1 == 1] = 0
    pred_1[pred_1 == -1] = 1
    print('IsF : ' + str(accuracy_score(y_test, pred_1)))
    print(classification_report(y_test, pred_1))

    #print('---MLP---')
    #MLP(X_train, y_train, param)
    # MLP
    # pred_2 = models_dict['MLP'].predict(X_test)
    # print('MLP : ' + str(accuracy_score(y_test, pred_2)))
    # print(classification_report(y_test, pred_2))

    # RF
    print('--RF---')
    RFR(X_train, y_train, split)
    pred_3 = models_dict['RFR'].predict(X_test)
    print('RFR : ' + str(accuracy_score(y_test, pred_3)))
    print(classification_report(y_test, pred_3))

    #KNN
    print('--KNN---')
    KNN(X_train, y_train)
    pred_4 = models_dict['KNN'].predict(X_test)
    print('KNN : ' + str(accuracy_score(y_test, pred_4)))
    print(classification_report(y_test, pred_4))
```

7.4 Ori_IsF.py

As the result of ISF generation is not ideal, I optimized ISF separately in this part. At present, the highest accuracy is even improved by 10%. However, due to the problem of training time, this part will be greatly improved in the future.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import plotly.express as px
from sklearn.decomposition import PCA
# for scaling
from sklearn.preprocessing import MaxAbsScaler
import datetime as dt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
import seaborn as sns
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.graph_objs as go
import plotly
import seaborn as sns
import plotly.express as px
from collections import Counter
#from imblearn.over_sampling import SMOTE
import matplotlib.gridspec as gridspec
import plotly.figure_factory as ff
# For GANs
#from tensorflow import keras
import keras
# for PCA
from sklearn.decomposition import PCA

from plotly import tools

# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, classification_report

# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier

# for scaling
from sklearn.preprocessing import MaxAbsScaler

import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
# import lightgbm as lgb
# from lightgbm import LGBMClassifier
#import xgboost as xgb

pd.set_option('display.max_columns', 100)

#sns.set(rc = {'figure.figsize':(15,8)})
def get_data():
    df=pd.read_csv('fraudTest.csv')
    df.isna().sum()
    df.drop_duplicates(inplace=True)

    df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
    df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
    df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
    df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month

    df = df.sample(frac = 1).reset_index(drop = True)
     df=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day','month','is_fraud']]
    #convert category to dummy variables，
    df=pd.get_dummies(df, drop_first=True)#one-hot
    # Extract from the dataframe, class 1s and 0s
    df_1 = df[df['is_fraud'] == 1].sample(frac = 1.0).reset_index(drop = True)
    df_2 = df[df['is_fraud'] == 0].sample(frac = 1.0).reset_index(drop = True)


```

```
    # Split each dataframe to certain fraction
        new_df_1, old_df_1 = df_1[: 480].reset_index(drop = True), df_1[480 :].reset_index(drop = True)
        new_df_2, old_df_2 = df_2[: 999].reset_index(drop = True), df_2[999 :].reset_index(drop = True)

    # group them into test and train sets
    test, train = pd.concat([new_df_1, new_df_2]), pd.concat([old_df_1, old_df_2])
    test, train = test.sample(frac = 1.0).reset_index(drop = True), train.sample(frac = 1.0).reset_index(drop = True)

    X_train = np.array(train.drop(['is_fraud'], axis = 1))
    y_train = np.array(train['is_fraud'])

    X_test = np.array(test.drop(['is_fraud'], axis = 1))
    y_test = np.array(test['is_fraud'])

    X_train= MaxAbsScaler().fit_transform(X_train)
    X_test = MaxAbsScaler().fit_transform(X_test)

    return X_train,y_train,X_test,y_test
if __name__=='__main__':
    models_dict = {}

    def IsF(X, y, ratio):
        # train test split
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

        # define the model
                model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)

        # train the model
        model.fit(X_train, y_train)

        # predict on the test set
        predictions = model.predict(X_test)

        # Convert the predictions according to problem profile
        predictions[predictions == 1] = 0
        predictions[predictions == -1] = 1
```

```

    # evaluate the accuracy and classification report
   # print('IsF : ' + str(accuracy_score(y_test, predictions)))
   # print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['IsF'] = model

X_train, y_train, X_test, y_test=get_data()
# For Isolation Forest Model
ratio = float(y_train[y_train == 1].shape[0] / y_train.shape[0])

# For Multi Layer Perceptron
param = {
    'nodes': 170,
    'lrate': 0.00005,
    'toler': 0.00001,
    'batch_size': 100
}

# For Random Forest Classifier
split = 2
IsF(X_train, y_train, ratio)

pred_1 = models_dict['IsF'].predict(X_test)
pred_1[pred_1 == 1] = 0
pred_1[pred_1 == -1] = 1
print('IsF : ' + str(accuracy_score(y_test, pred_1)))
print(classification_report(y_test, pred_1))
```

7.5 Syn_3ML.py

This part includes two large modules, which are also the core modules of this Project. The first one is cGAN training and generating synthetic data, and the second part is pushing the synthetic data into three machine learning algorithms for prediction and output results.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import plotly.express as px
```

```
from sklearn.decomposition import PCA
# for scaling
from sklearn.preprocessing import MaxAbsScaler
import datetime as dt
from Ori_3ML import get_data
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
import seaborn as sns
#%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.graph_objs as go
import plotly
import seaborn as sns
import plotly.express as px
from collections import Counter
#from imblearn.over_sampling import SMOTE
import matplotlib.gridspec as gridspec
import plotly.figure_factory as ff
# For GANs
#from tensorflow import keras
import keras
# for PCA
from sklearn.decomposition import PCA
#from ori_IsF import MLP,RFR,KNN

from plotly import tools

# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
# for scaling
from sklearn.preprocessing import MaxAbsScaler,MinMaxScaler,StandardScaler
from num2category import num2cate

```

```
import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier,IsolationForest
from sklearn import svm
# import lightgbm as lgb
# from lightgbm import LGBMClassifier
#import xgboost as xgb

pd.set_option('display.max_columns', 100)

#sns.set(rc = {'figure.figsize':(15,8)})
df=pd.read_csv('fraudTest.csv')
df.isna().sum()
df.drop_duplicates(inplace=True)
df.info()
df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month

# Shuffle the dataframe df
df = df.sample(frac = 1).reset_index(drop = True)
train=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day','month','is_fraud']]
train_save=train[0:150000]
train_save.to_csv('Ori_data.csv',index=None)
#convert category to dummy variables，
#train=pd.get_dummies(train, drop_first=True)
num,index=pd.factorize(train['category'])
train['category']=num

y_train=train['is_fraud'].values
X_train=train.drop("is_fraud", axis='columns').values
y=y_train
# scale b/w [-1, 1]
# X = MaxAbsScaler().fit_transform(X_train)
mm=StandardScaler()
X=mm.fit_transform(X_train)
# Let's define the discriminator which takes inputs the feature
```

```
# matrix and class vector and predicts the probability of being
# fake or real (0 or 1)
# 2 Hidden Layer with each layer has 200 nodes
def dis():
    # define the feature input
    feature = keras.Input(shape=(12,))

    # define the labels  input
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([feature, labels])

    # add one hidden layer
    model = keras.layers.Dense(32)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    model = keras.layers.Dense(64)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    model = keras.layers.Dense(32)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # add the output layer
    model = keras.layers.Dense(1, activation='sigmoid')(model)

    # create a model from the pipeline
    d_model = keras.Model(inputs=[feature, labels], outputs=model, name='discriminator')

    # compile the model
    d_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rate=0.0001), loss='binary_crossentropy',
                    metrics=['accuracy'])

    # return the model
    return d_model

# Let's define the generator which takes as input latent
# space and class labels and ouputs a feature matrix 'X'
# generator has 3 hidden layer with 150x200x150 nodes
def gen():
```

```
    # define the latent space
    latent = keras.Input(shape=(99,))

    # define the label vector
    labels = keras.Input(shape=(1,))

    # merge the two layers
    merge = keras.layers.Concatenate()([latent, labels])

    # create 1 hidden layer
    model = keras.layers.Dense(32)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 2 hidden layer
    model = keras.layers.Dense(64)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 3 hidden layer
    model = keras.layers.Dense(32)(merge)#merge
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create an ouptput layer
    model = keras.layers.Dense(12, activation='tanh')(model)

    # create a model from the pipeline
    g_model = keras.Model(inputs=[latent, labels], outputs=model, name='generator')

    # return the model
    return g_model

# Combine the discriminator and Generator into cGAN
# Keep discriminator as non-trainable, so it does
# not update its weights during training the cgan.
def gan(discriminator, generator):
    # Make discriminator as non-trainable
    discriminator.trainable = False

    # get inputs from the generator
    gen_latent, gen_labels = generator.input

    # get output from the generator
    gen_feature = generator.output
```

```
    # get discriminator predictions
    dis_predict = discriminator([gen_feature, gen_labels])

    # create a model from the pipeline
    cgan_model = keras.Model(inputs=[gen_latent, gen_labels], outputs=dis_predict, name='cGAN')

    # compile the model
    cgan_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rate=0.0001), loss='binary_crossentropy')

    # return the model
    return cgan_model

# Let's define a generate_real_samples
# method to generate random samples as
# real data
def generate_real_samples(n_samples):
    # generate n_samples number of random indices
    indices = np.random.randint(0, X.shape[0], n_samples)

    # get the newly indexed matrix and vector
    X_real = np.array([X[j] for j in indices])
    y_real = np.array([y[j] for j in indices])

    # classify them as '1' (real)
    y_labels = np.ones((n_samples, 1))

    # return the computed arrays
    return [X_real, y_real], y_labels
# Let's define a generate_fake_samples
# method to generate random samples as
# fake data from latent sapce
def generate_latent_point(n_samples):
    # create an array of n_samples * late
    # -nt_space (= 99)
    Z_lat = np.random.randn(n_samples * 99)
    X_lat = Z_lat.reshape(n_samples, 99)

    # randomly generate the class labels 'y'
    y_lat = np.random.randint(0, 2, n_samples)
```

```
    # return the points
    return [X_lat, y_lat]

def generate_fake_samples(generator, n_samples):
    # generate n_samples points in latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # predict on them using the generator
    X_fake = generator.predict([X_lat, y_lat])

    # classify them as '0' (fake)
    y_labels = np.zeros((n_samples, 1))

    # return the computed arrays
    return [X_fake, y_lat], y_labels

# plot the loss functions on the graph
def plot_history(d_real, d_fake, g):
    fig = plt.figure(figsize=(15, 15))

    plt.plot(d_real, label='d_real')
    plt.plot(d_fake, label='d_fake')
    plt.plot(g, label='cGAN_loss')

    # specify legend
    plt.legend()

    # plot the graph
    plt.savefig('loss.jpg')
    plt.show()

# train the model
def train(discriminator, generator, cgan, batch_size=512, epochs=6):

    count = 0
    half_batch = int(batch_size / 2)
    batch_per_epoch = int(X.shape[0] / batch_size)

    d_real = []
    d_fake = []
    g = []
```

```
    for i in range(epochs):
        for j in range(batch_per_epoch):

            # real
            [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
            # loss on real
            d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
            # fake
            [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
                                                half_batch)
            # loss on fake
            d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)

            # Generate latent points
            [X_lat, y_lat] = generate_latent_point(batch_size)
            # Give them '1' labels
            y_labels_lat = np.ones((batch_size, 1))
            # calculate generator loss
            gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)

            # print the results
            count += 1
            if count % 1000 == 0:
                print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'.format(count, d1_loss, d2_loss, gan_loss))

            # store the losses
            d_real.append(d1_loss)
            d_fake.append(d2_loss)
            g.append(gan_loss)

    # plot a summary of loss
    plot_history(d_real, d_fake, g)

discriminator = dis()
generator = gen()
cgan = gan(discriminator, generator)

print(generator.summary())     # summary for dis
print(discriminator.summary()) # summary for gen
print(cgan.summary())          # summary for gan
# train the cGAN model
```

```
train(discriminator, generator, cgan)

# generate the synthetic fraudulent
# data
def generate_new_samples(n_samples):
    # get n_samples for latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # specify the labels
    y_lat = np.zeros(n_samples)

    # predict on generator
    X_pred = generator.predict([X_lat, y_lat])

    # return fraud samples
    return [X_pred, y_lat]

# shuffle the dataframe df
df = df.sample(frac = 1.0).reset_index(drop = True)

# get 'X' and 'y'
# X = np.array(df.drop(['is_fraud'], axis = 1))
# y = np.array(df['is_fraud'])

# scale down [-1, 1]
X = StandardScaler().fit_transform(X_train)
y=y_train


###########Machine Learing###############
models_dict_IsF = {}
models_dict_RFR={}
models_dict_KNN={}
def IsF(X, y, ratio):
    # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

    # define the model
    model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)

```

```
    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # Convert the predictions according to problem profile
    predictions[predictions == 1] = 0
    predictions[predictions == -1] = 1

    # evaluate the accuracy and classification report
    print('IsF : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict_IsF['IsF'] = model
def RFR(X, y, split):
    # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

    # define the model
        model = RandomForestClassifier(n_estimators=200, criterion='gini', min_samples_split=split, random_state=1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('RFR : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict_RFR['RFR'] = model
def KNN(X, y):
    # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

```

```
    # define the model
    model = KNeighborsClassifier()

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('KNN : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict_KNN['KNN'] = model
ratio = float(y_train[y_train == 1].shape[0]/y_train.shape[0])
# For Multi Layer Perceptron
param = {
    'nodes' : 170,
    'lrate' : 0.00005,
    'toler' : 0.00001,
    'batch_size' : 100
}

print('Test perfromance:')

# get the numpy array
# X = np.array(train.drop(['is_fraud'], axis = 1))
# y = np.array(train['is_fraud'])
X=X_train
y=y_train

# scale [-1, 1]
X = StandardScaler().fit_transform(X)

#X_gan_test, y_gan_test = X[:1000,:], y[:1000]
X_train, y_train, X_test, y_test=get_data()
X_gan_test,y_gan_test=X_test,y_test
X, y = X[1000:, :], y[1000:]
# get synthetic samples
[X_syn, y_syn] = generate_new_samples(150000)
X_syn=mm.inverse_transform(X_syn)
```

```
"""
# PCA
pca = PCA(n_components = 3)
X = pca.fit_transform(X)
X=X[0:150000]
# # plot the 'real' fraud data
fig = plt.figure(figsize = (15,15))
axs = plt.axes(projection = '3d')
for i in range(X.shape[0]) :
    if y[i] == 1 :
        axs.scatter3D(X[i,0], X[i,1], X[i,2], color = 'orange')
# Synthetic data from stand-alone generator
#[X_syn, y_syn] = generate_new_samples(1000)
# # PCA
pca = PCA(n_components = 3)
X_syn = pca.fit_transform(X_syn)
#
# # plot
for i in range(X_syn.shape[0]) :
    if y_syn[i] == 1.0 :
        axs.scatter3D(X_syn[i,0], X_syn[i,1], X_syn[i,2], color = 'blue')
plt.title('Scatter plot of real and synthetic fraudulent samples')
plt.savefig('3D_2.jpg')
plt.show()
"""

#Save data
col=['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day',
'month','is_fraud']
Syn_data=np.concatenate((np.array(X_syn),np.array(y_syn.reshape(-1,1))),axis=1)
Syn_data=pd.DataFrame(Syn_data,columns=col)
Syn_data['category']=num2cate(Syn_data['category'])
Syn_data.to_csv('Syn_data.csv',index=None)
# reshape into 2d array
y, y_syn = y.reshape(-1,1), y_syn.reshape(-1,1)
# concatenate all the arrays
X_1, X_2 = np.concatenate((X,y), axis = 1), np.concatenate((X_syn,y_syn), axis = 1)
# merge the above two arrays
X_3 = np.concatenate((X_1, X_2), axis = 0)
np.random.shuffle(X_3)

```

```
X_gan = np.array(X_3[:,:12])
y_gan = np.array(X_3[:, 12])
ratio = float(y_gan[y_gan == 1].shape[0]/y_gan.shape[0])

print('-----IsF------')
IsF(X_gan, y_gan, ratio)
pred_1 = models_dict_IsF['IsF'].predict(X_gan_test)
pred_1[pred_1 ==  1] = 0
pred_1[pred_1 == -1] = 1
print('IsF : ' + str(accuracy_score(y_gan_test, pred_1)))
print(classification_report(y_gan_test,pred_1))

"""
print('------MLP--------')
MLP(X_gan, y_gan, param)
pred_2 = models_dict['MLP'].predict(X_gan_test)
print('MLP : ' + str(accuracy_score(y_gan_test, pred_2)))
print(classification_report(y_gan_test,pred_2))
"""
print('-------RF--------')
RFR(X_gan, y_gan, 2)
pred_3 = models_dict_RFR['RFR'].predict(X_gan_test)
print('RFR : ' + str(accuracy_score(y_gan_test, pred_3)))
print(classification_report(y_gan_test,pred_3))

print('-------KNN--------')
KNN(X_gan, y_gan)
pred_4 = models_dict_KNN['KNN'].predict(X_gan_test)
print('KNN : ' + str(accuracy_score(y_gan_test, pred_4)))
print(classification_report(y_gan_test,pred_4))
```

7.6 Syn_Isf.py

This part is also a separate optimization of ISF synthetic data, modifying parameters and coding methods, and finally improving the accuracy by 10%.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import plotly.express as px
from sklearn.decomposition import PCA
```

```
# for scaling
from sklearn.preprocessing import MaxAbsScaler
import datetime as dt
from ori_IsF import get_data
import pandas as pd
import numpy as np
import matplotlib as mpl
mpl.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
import seaborn as sns
#%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.graph_objs as go
import plotly
import seaborn as sns
import plotly.express as px
from collections import Counter
#from imblearn.over_sampling import SMOTE
import matplotlib.gridspec as gridspec
import plotly.figure_factory as ff
# For GANs
#from tensorflow import keras
import keras
# for PCA
from sklearn.decomposition import PCA

from plotly import tools
#from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
#init_notebook_mode(connected=True)

# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier

# for scaling
```

```
from sklearn.preprocessing import MaxAbsScaler

import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier,IsolationForest
from sklearn import svm
# import lightgbm as lgb
# from lightgbm import LGBMClassifier
#import xgboost as xgb

pd.set_option('display.max_columns', 100)

#sns.set(rc = {'figure.figsize':(15,8)})
df=pd.read_csv('fraudTest.csv')
df.isna().sum()
df.drop_duplicates(inplace=True)
df.info()
df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month

# Shuffle the dataframe df
df = df.sample(frac = 1).reset_index(drop = True)
train=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day','month','is_fraud']]
#convert category to dummy variables
train=pd.get_dummies(train, drop_first=True)

y_train=train['is_fraud'].values[0:300000]
X_train=train.drop("is_fraud", axis='columns').values[0:300000]
y=y_train
# scale b/w [-1, 1]
X = MaxAbsScaler().fit_transform(X_train)

# Let's define the discriminator which takes inputs the feature
# matrix and class vector and predicts the probability of being
# fake or real (0 or 1)
# 2 Hidden Layer with each layer has 200 nodes
```

```
90. def dis():
91.     # define the feature input
92.     feature = keras.Input(shape=(24,))
93.
94.     # define the labels  input
95.     labels = keras.Input(shape=(1,))
96.
97.     # merge the two layers
98.     merge = keras.layers.Concatenate()([feature, labels])
99.
100.    # add one hidden layer
101.    model = keras.layers.Dense(200)(merge)
102.    model = keras.layers.LeakyReLU(alpha=0.2)(model)
103.
104.    model = keras.layers.Dense(200)(merge)
105.    model = keras.layers.LeakyReLU(alpha=0.2)(model)
106.
107.    # add the output layer
108.    model = keras.layers.Dense(1, activation='sigmoid')(model)
109.
110.    # create a model from the pipeline
111.      d_model  =  keras.Model(inputs=[feature,  labels],  outputs=model,
    name='discriminator')
112.
113.    # compile the model
114.    d_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rat
    e=0.0002), loss='binary_crossentropy',
115.                    metrics=['accuracy'])
116.
117.    # return the model
118.    return d_model
119.
120. # Let's define the generator which takes as input latent
121. # space and class labels and ouputs a feature matrix 'X'
122. # generator has 3 hidden layer with 150x200x150 nodes
123. def gen():
124.    # define the latent space
125.    latent = keras.Input(shape=(99,))
126.
127.    # define the label vector
128.    labels = keras.Input(shape=(1,))
129.
130.    # merge the two layers
```

```
    merge = keras.layers.Concatenate()([latent, labels])

    # create 1 hidden layer
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 2 hidden layer
    model = keras.layers.Dense(200)(model)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create 3 hidden layer
    model = keras.layers.Dense(150)(merge)
    model = keras.layers.LeakyReLU(alpha=0.2)(model)

    # create an ouptput layer
    model = keras.layers.Dense(24, activation='tanh')(model)

    # create a model from the pipeline
    g_model = keras.Model(inputs=[latent, labels], outputs=model, name='generator')

    # return the model
    return g_model

# Combine the discriminator and Generator into cGAN
# Keep discriminator as non-trainable, so it does
# not update its weights during training the cgan.
def gan(discriminator, generator):
    # Make discriminator as non-trainable
    discriminator.trainable = False

    # get inputs from the generator
    gen_latent, gen_labels = generator.input

    # get output from the generator
    gen_feature = generator.output

    # get discriminator predictions
    dis_predict = discriminator([gen_feature, gen_labels])

    # create a model from the pipeline
    cgan_model = keras.Model(inputs=[gen_latent, gen_labels], outputs=dis_predict, name='cGAN')
```

```
172.
173.     # compile the model
174.     cgan_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_
     rate=0.0002), loss='binary_crossentropy')
175.
176.     # return the model
177.     return cgan_model
178.
179. # Let's define a generate_real_samples
180. # method to generate random samples as
181. # real data
182. def generate_real_samples(n_samples):
183.     # generate n_samples number of random indices
184.     indices = np.random.randint(0, X.shape[0], n_samples)
185.
186.     # get the newly indexed matrix and vector
187.     X_real = np.array([X[j] for j in indices])
188.     y_real = np.array([y[j] for j in indices])
189.
190.     # classify them as '1' (real)
191.     y_labels = np.ones((n_samples, 1))
192.
193.     # return the computed arrays
194.     return [X_real, y_real], y_labels
195.
196. # Let's define a generate_fake_samples
197. # method to generate random samples as
198. # fake data from latent sapce
199. def generate_latent_point(n_samples):
200.     # create an array of n_samples * late
201.     # -nt_space (= 99)
202.     Z_lat = np.random.randn(n_samples * 99)
203.     X_lat = Z_lat.reshape(n_samples, 99)
204.
205.     # randomly generate the class labels 'y'
206.     y_lat = np.random.randint(0, 2, n_samples)
207.
208.     # return the points
209.     return [X_lat, y_lat]
210.
211. def generate_fake_samples(generator, n_samples):
212.     # generate n_samples points in latent space
```

```
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # predict on them using the generator
    X_fake = generator.predict([X_lat, y_lat])

    # classify them as '0' (fake)
    y_labels = np.zeros((n_samples, 1))

    # return the computed arrays
    return [X_fake, y_lat], y_labels

# plot the loss functions on the graph
def plot_history(d_real, d_fake, g):
    fig = plt.figure(figsize=(15, 15))

    plt.plot(d_real, label='d_real')
    plt.plot(d_fake, label='d_fake')
    plt.plot(g, label='cGAN_loss')

    # specify legend
    plt.legend()

    # plot the graph
    plt.savefig('loss.jpg')
    #plt.show()

# train the model
def train(discriminator, generator, cgan, batch_size=512, epochs=6):

    count = 0
    half_batch = int(batch_size / 2)
    batch_per_epoch = int(X.shape[0] / batch_size)

    d_real = []
    d_fake = []
    g = []
    for i in range(epochs):
        for j in range(batch_per_epoch):

            # real
                                [X_real,    y_real],    y_labels_real    =
generate_real_samples(half_batch)
```

```
            # loss on real
            d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
            # fake
                                [X_fake,    y_fake],    y_labels_fake    = generate_fake_samples(generator,
                                                                                                half_batch)
            # loss on fake
            d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)

            # Generate latent points
            [X_lat, y_lat] = generate_latent_point(batch_size)
            # Give them '1' labels
            y_labels_lat = np.ones((batch_size, 1))
            # calculate generator loss
            gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)

            # print the results
            count += 1
            if count % 1000 == 0:
                    print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'.format(count, d1_loss, d2_loss, gan_loss))

            # store the losses
            d_real.append(d1_loss)
            d_fake.append(d2_loss)
            g.append(gan_loss)

    # plot a summary of loss
    plot_history(d_real, d_fake, g)

discriminator = dis()
generator = gen()
cgan = gan(discriminator, generator)

print(generator.summary())     # summary for dis
print(discriminator.summary()) # summary for gen
print(cgan.summary())          # summary for gan
# train the cGAN model
train(discriminator, generator, cgan)

```

```
# generate the synthetic fraudulent
# data
def generate_new_samples(n_samples):
    # get n_samples for latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # specify the labels
    y_lat = np.ones(n_samples)

    # predict on generator
    X_pred = generator.predict([X_lat, y_lat])

    # return fraud samples
    return [X_pred, y_lat]

# shuffle the dataframe df
df = df.sample(frac = 1.0).reset_index(drop = True)

# get 'X' and 'y'
# X = np.array(df.drop(['is_fraud'], axis = 1))
# y = np.array(df['is_fraud'])

# scale down [-1, 1]
X = MaxAbsScaler().fit_transform(X_train)
y=y_train

models_dict = {}
def IsF(X, y, ratio):
    # train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    # define the model
        model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # Convert the predictions according to problem profile
```

```
    predictions[predictions == 1] = 0
    predictions[predictions == -1] = 1

    # evaluate the accuracy and classification report
    # print('IsF : ' + str(accuracy_score(y_test, predictions)))
    # print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['IsF'] = model
ratio = float(y_train[y_train == 1].shape[0]/y_train.shape[0])
# For Multi Layer Perceptron
param = {
    'nodes' : 170,
    'lrate' : 0.00005,
    'toler' : 0.00001,
    'batch_size' : 100
}

print('Test performance')

# get the numpy array
# X = np.array(train.drop(['is_fraud'], axis = 1))
# y = np.array(train['is_fraud'])
X=X_train
y=y_train

# scale [-1, 1]
X = MaxAbsScaler().fit_transform(X)

#X_gan_test, y_gan_test = X[:1000,:], y[:1000]
X_train, y_train, X_test, y_test=get_data()
X_gan_test,y_gan_test=X_test,y_test
X, y = X[1000:, :], y[1000:]
# get synthetic samples
[X_syn, y_syn] = generate_new_samples(150000)

Syn_data=np.concatenate((np.array(X_syn),np.array(y_syn.reshape(-1,1))),axis=1)
Syn_data=pd.DataFrame(Syn_data)
Syn_data.to_csv('Syn_data.csv',index=None)

# reshape into 2d array
```

```
y, y_syn = y.reshape(-1,1), y_syn.reshape(-1,1)
# concatenate all the arrays
X_1, X_2 = np.concatenate((X,y), axis = 1), np.concatenate((X_syn,y_syn), axis = 1)
# merge the above two arrays
X_3 = np.concatenate((X_1, X_2), axis = 0)
np.random.shuffle(X_3)

X_gan = np.array(X_3[:,:24])
y_gan = np.array(X_3[:, 24])
ratio = float(y_gan[y_gan == 1].shape[0]/y_gan.shape[0])
IsF(X_gan, y_gan, ratio)

pred_1 = models_dict['IsF'].predict(X_gan_test)
pred_1[pred_1 ==  1] = 0
pred_1[pred_1 == -1] = 1
print('IsF : ' + str(accuracy_score(y_gan_test, pred_1)))
print(classification_report(y_gan_test,pred_1))
```