

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:	<i>Bingwei Wang</i>
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input checked="" type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input checked="" type="checkbox"/>
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.	<input checked="" type="checkbox"/>

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 image¹

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, Anaconda 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

¹Python 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

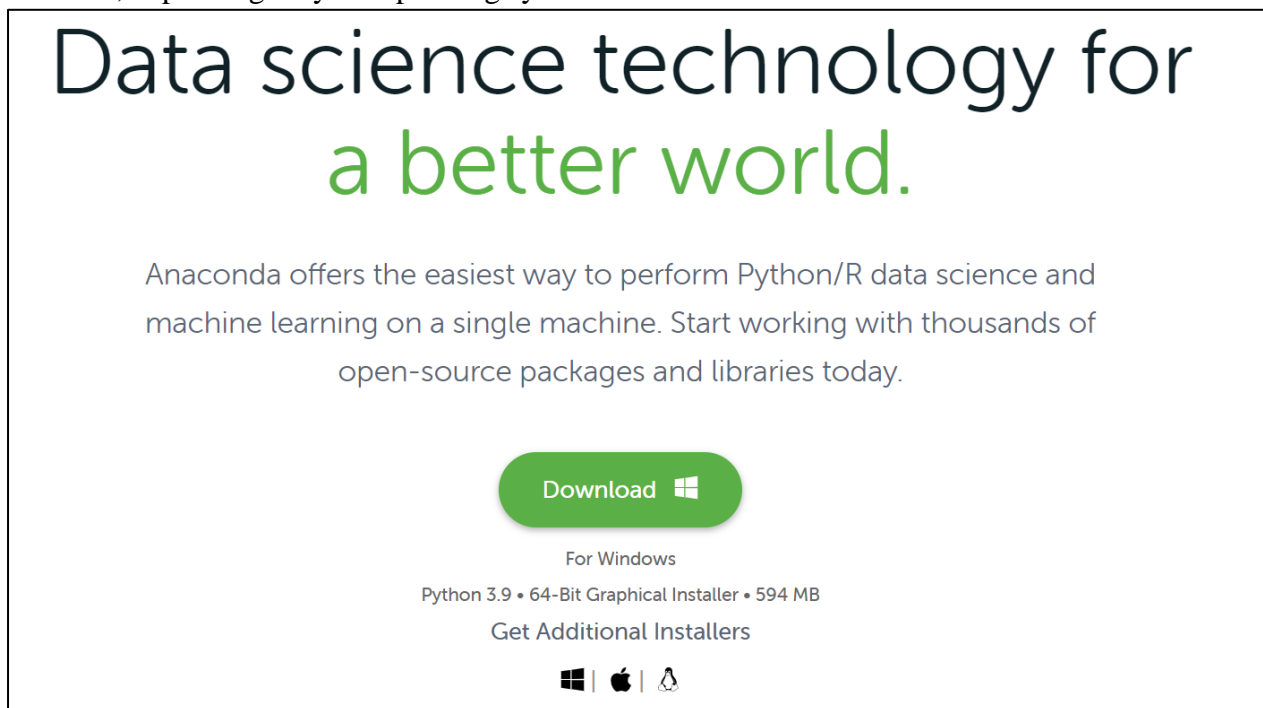


Figure 3: Anaconda download representational image²

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 packages³.

²Anaconda official url: <https://www.anaconda.com>

³Dataset 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

<i>Num.</i>	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 Anaconda, but sometimes there are incompatible versions.

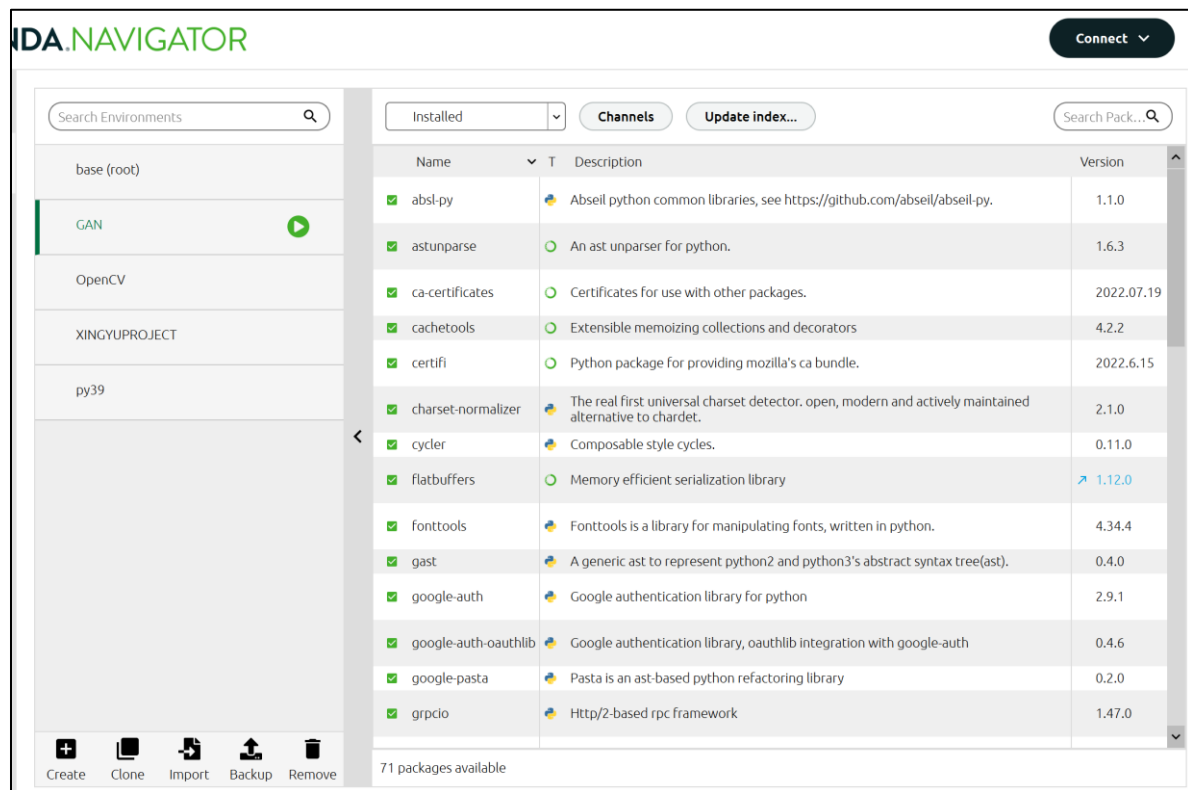


Figure 4. Anaconda Home Page

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.

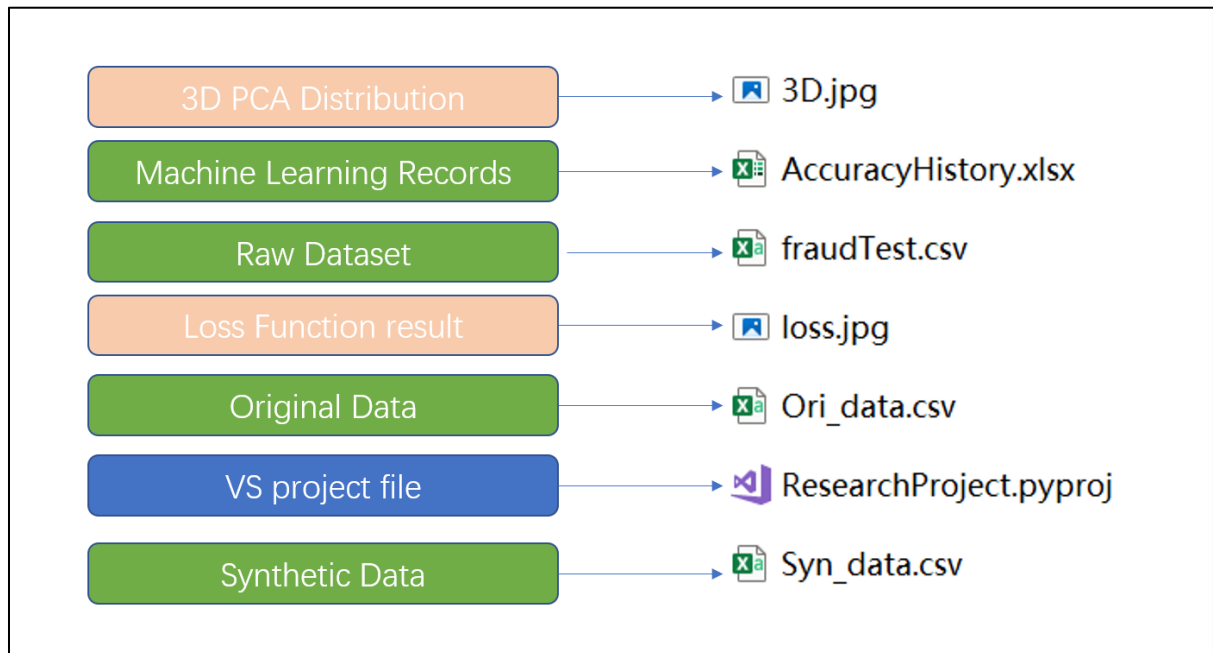


Figure 5. Data and results files in ICT Solution artefact

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

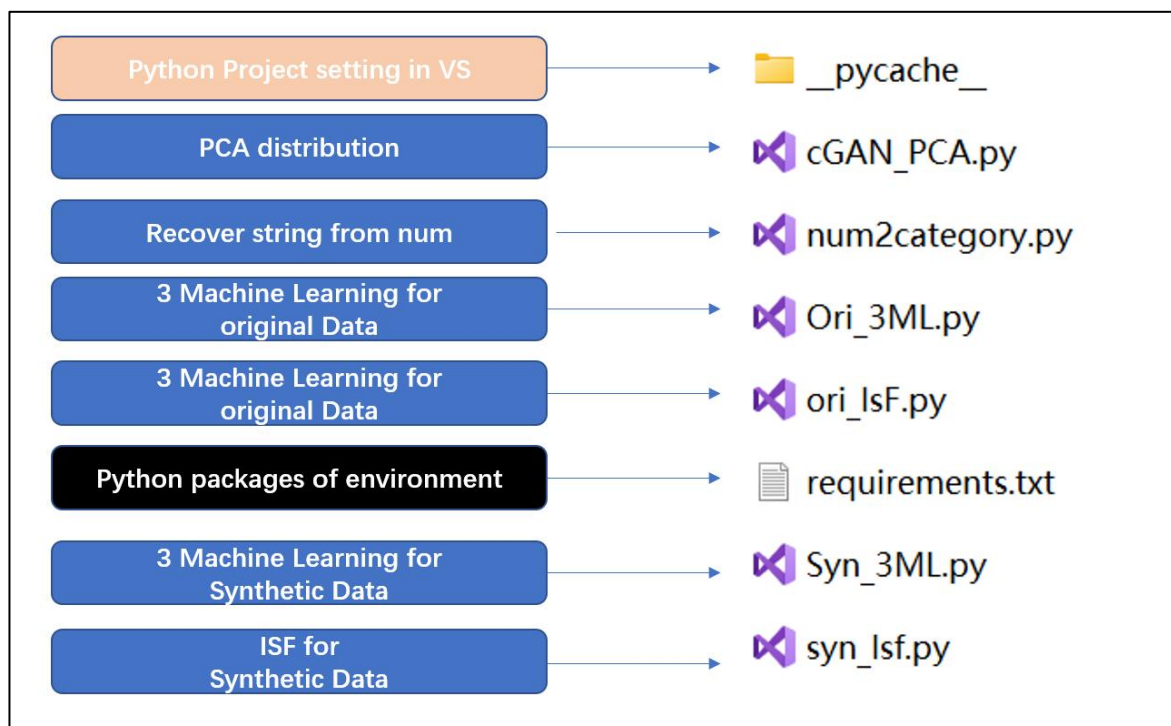


Figure 6. Project code file

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

```
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

```
# 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.

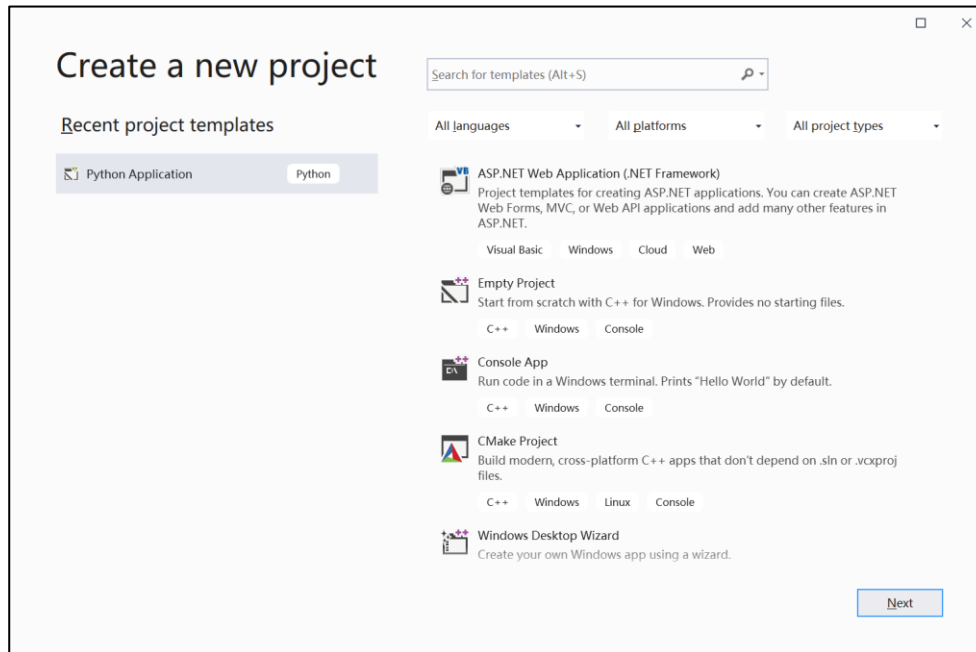


Figure 7. Home Page of Visual Studio

Click Next here:

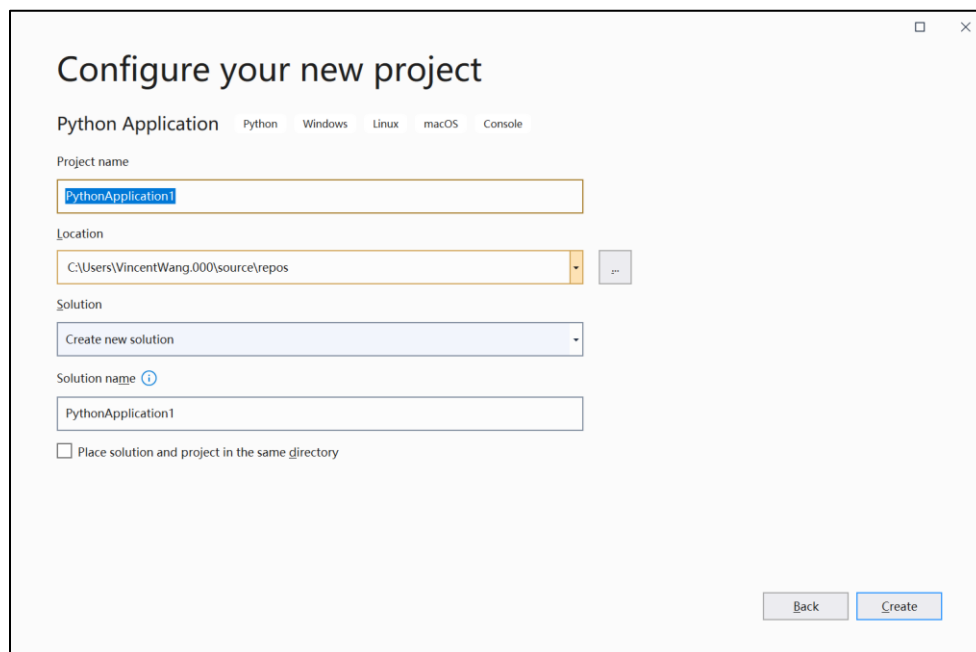


Figure 8. Create a Python Project

Click Next here, as shown in Figure 8.

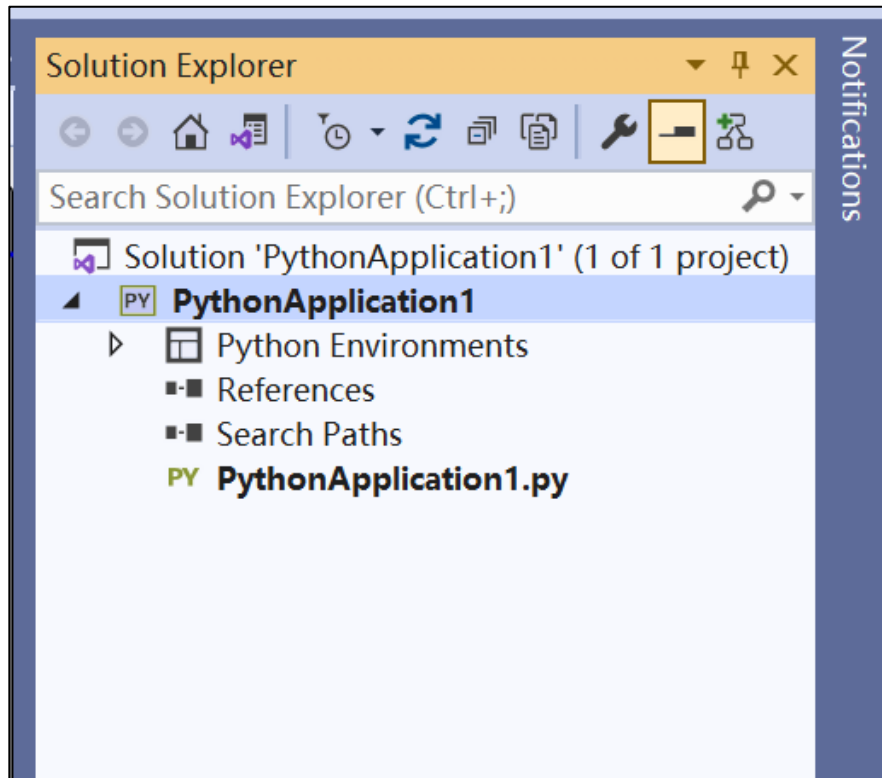


Figure 9. Project solution file list in Visual Studio

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

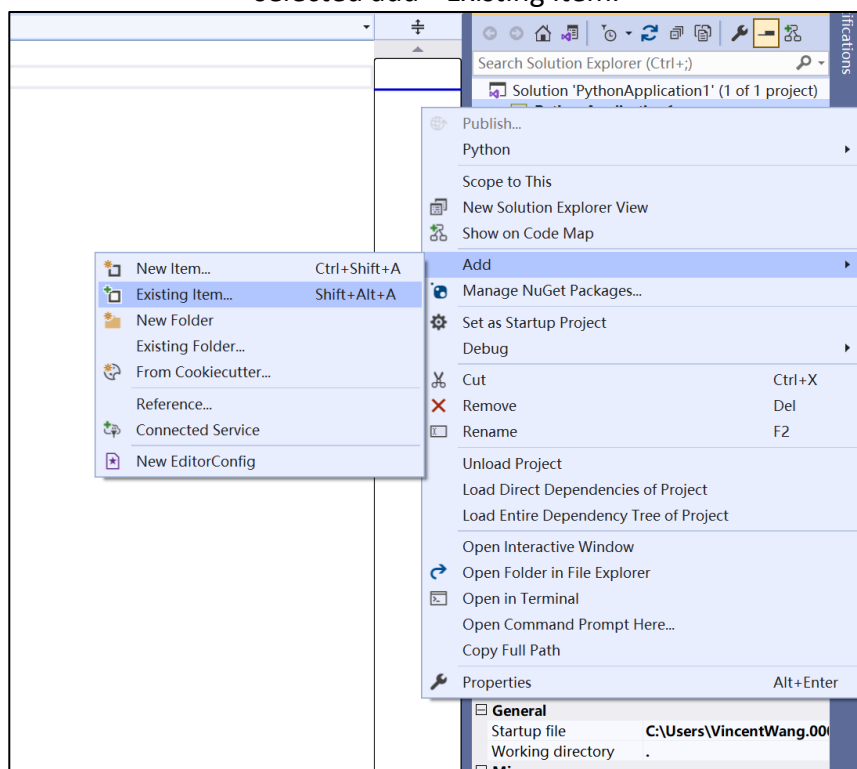


Figure 10. Import items to VS project.

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

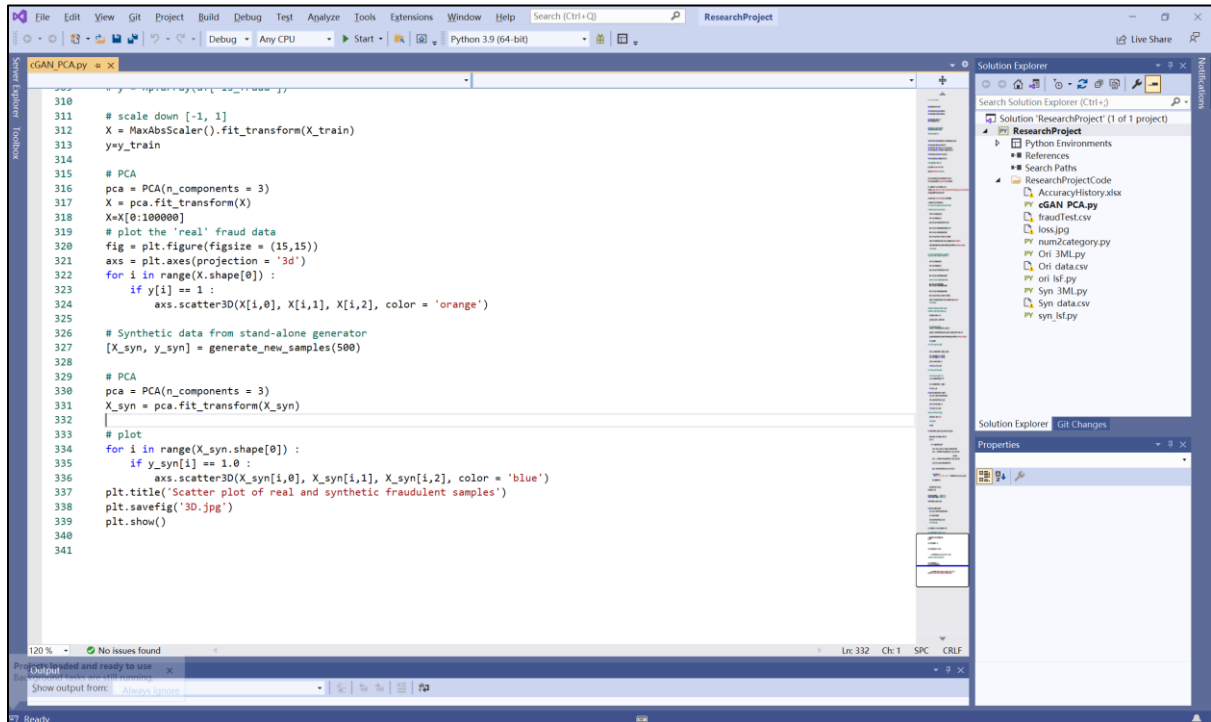


Figure 11. Files list system in VS.

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.

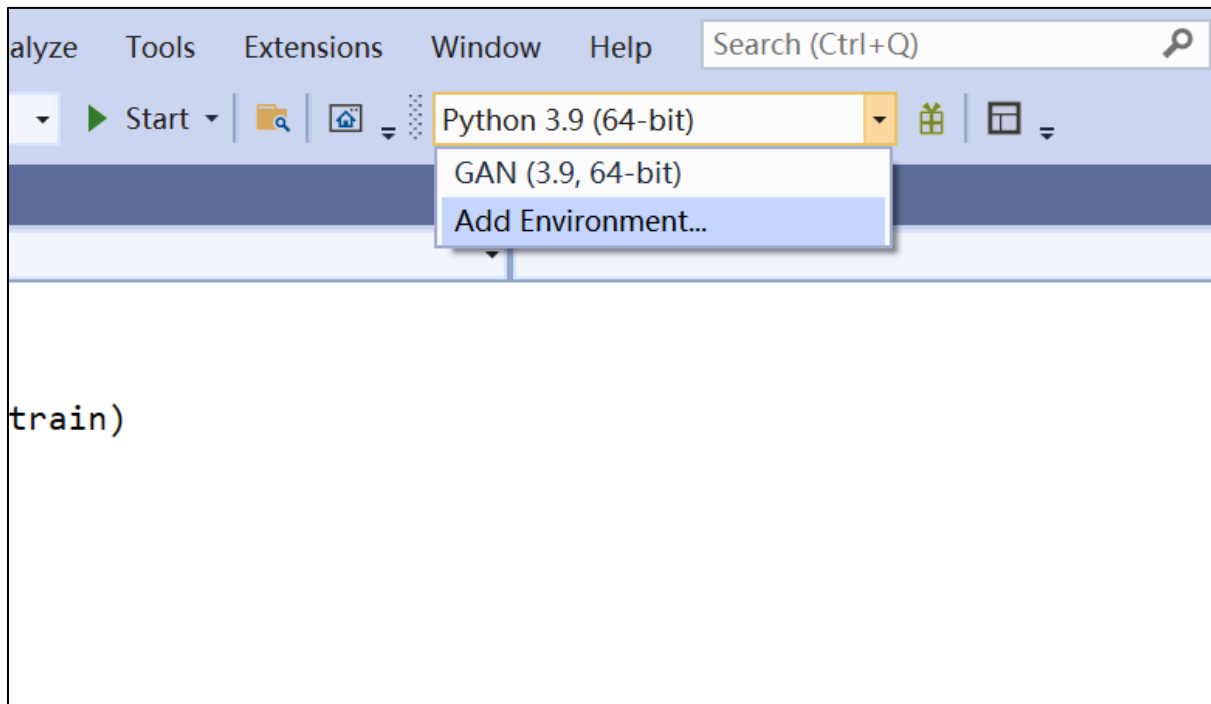


Figure 12. Set environment for python in VS.

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

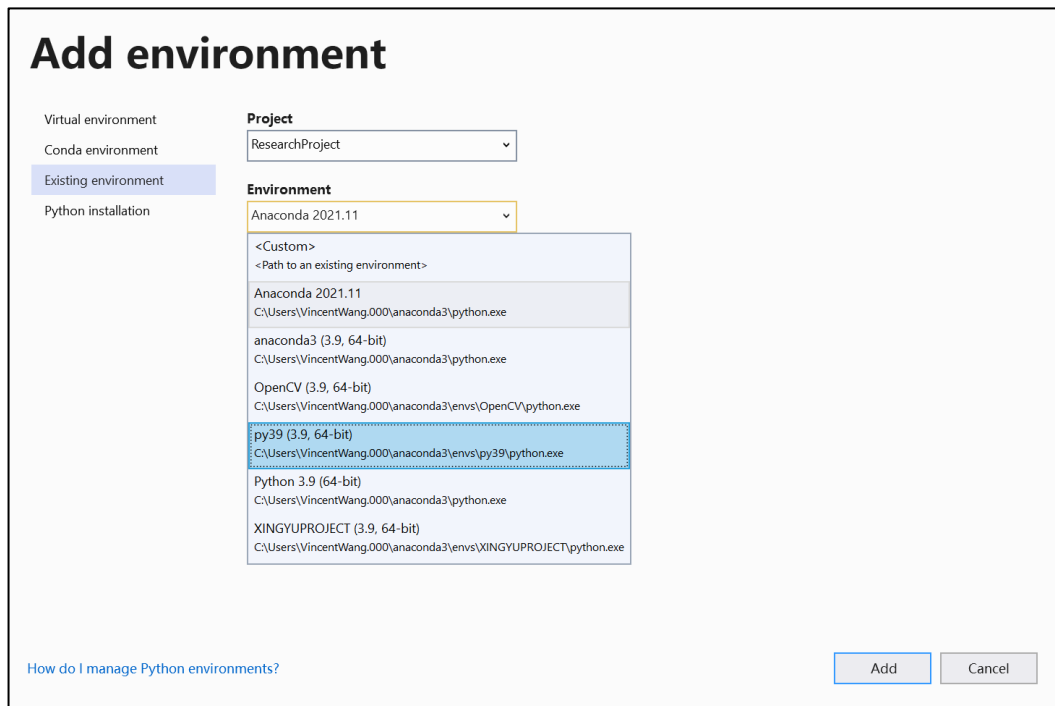


Figure 13. Select an environment created by Anaconda.

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

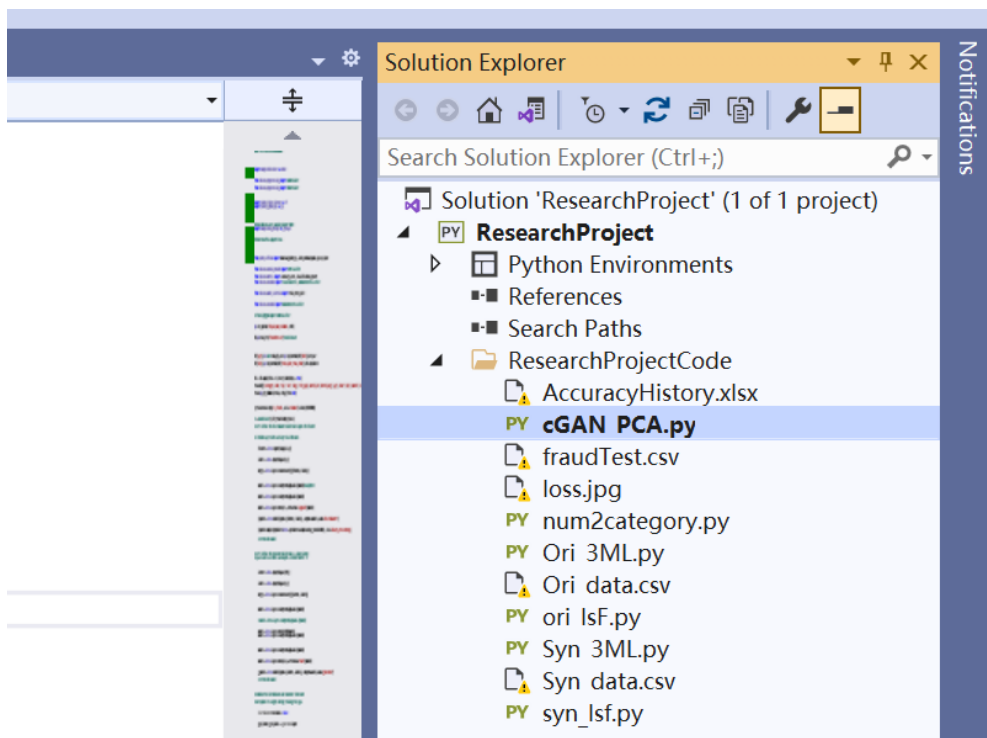


Figure 14. Set a startup file in VS

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.

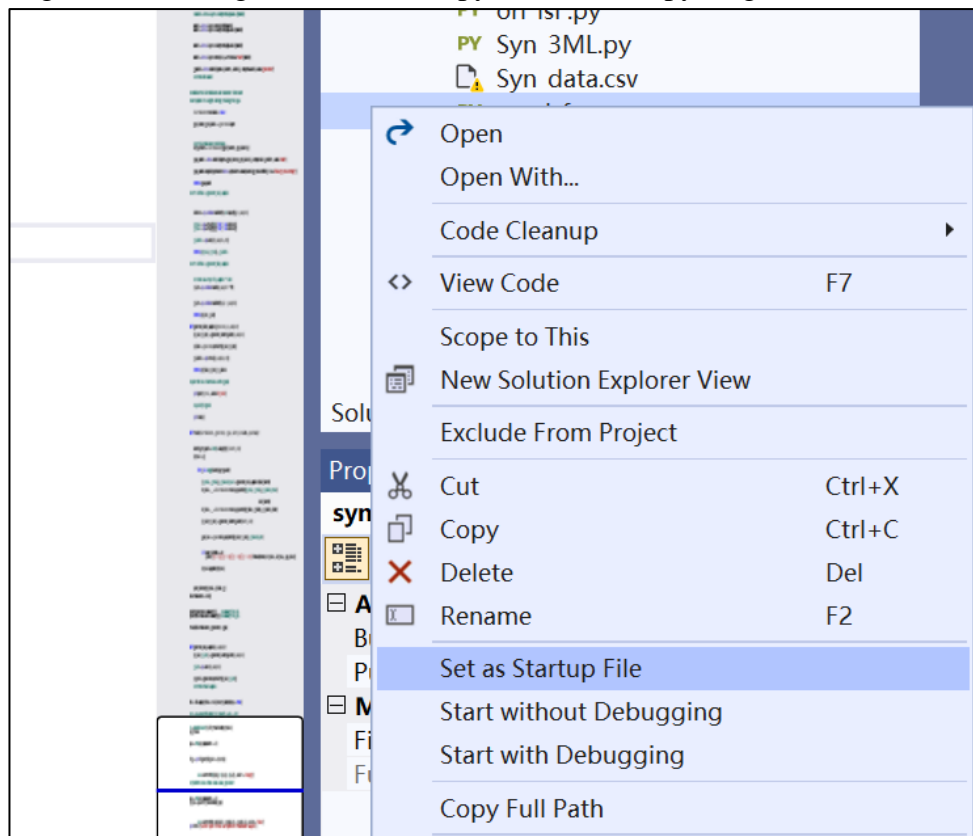


Figure 15. Set a startup file in VS

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

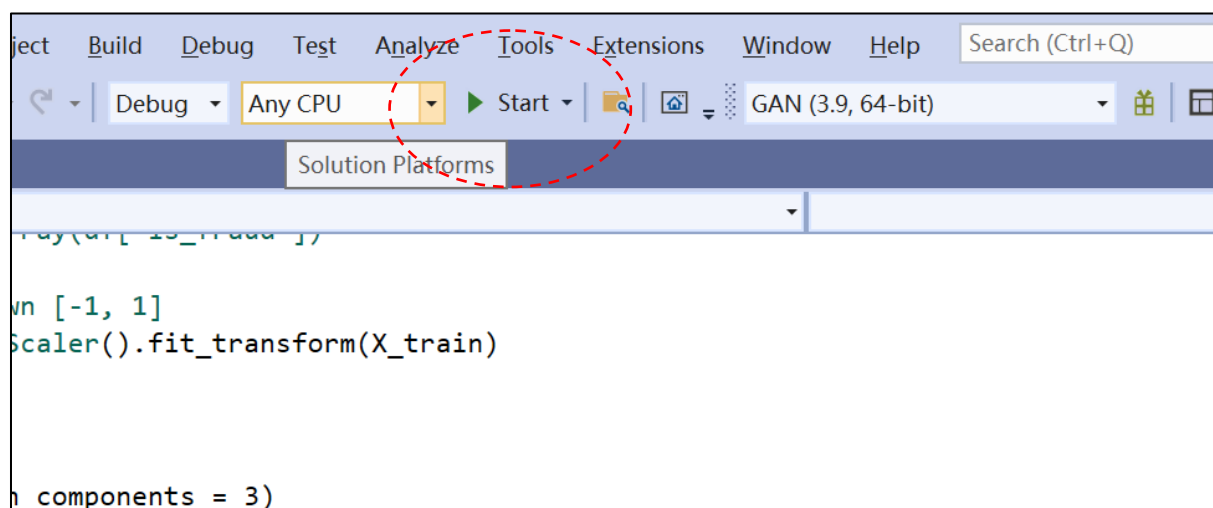


Figure 16. Run a python file in VS.

6. The corresponding result of Smart Synthetic Data project

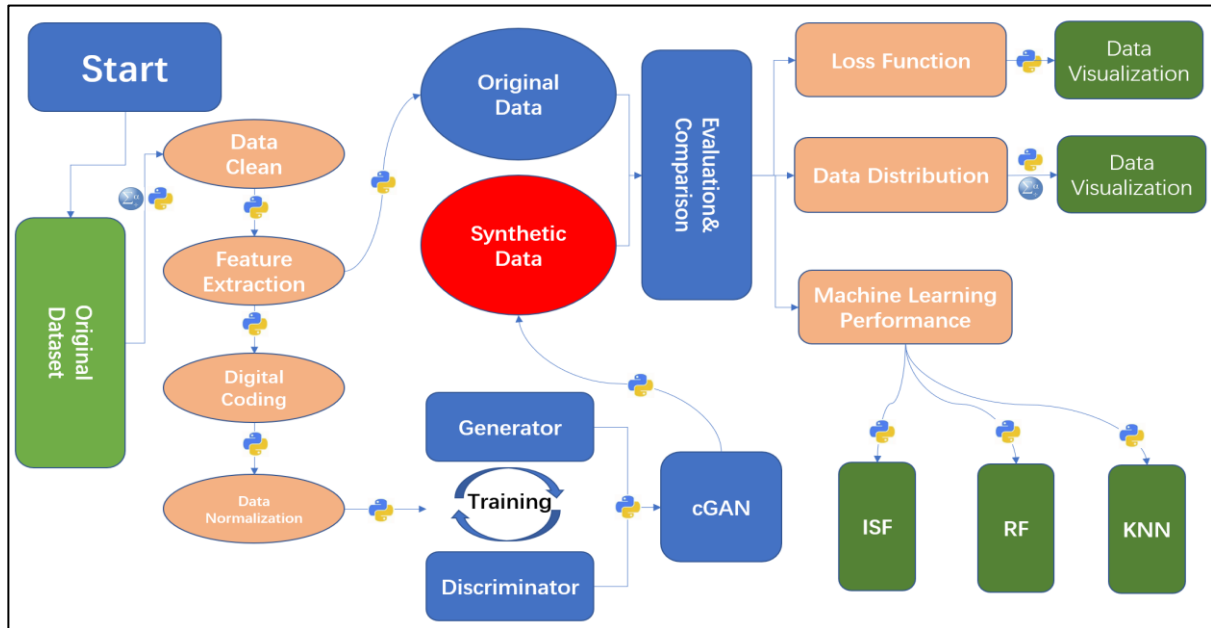


Figure 17. The implementation process flow chat of the whole project

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

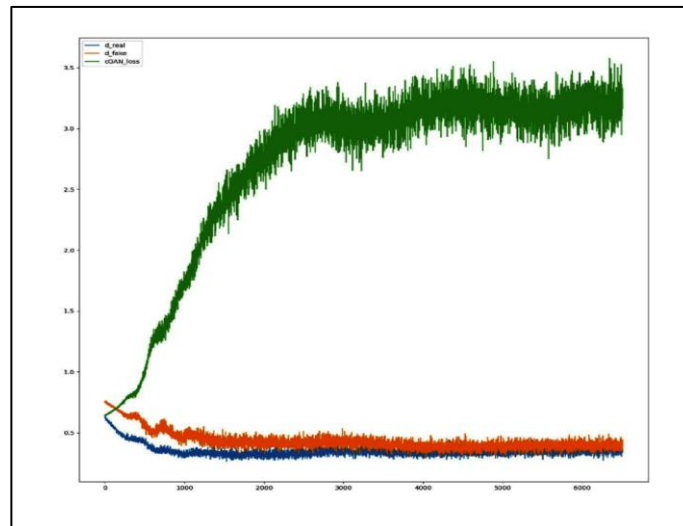


Figure 18. An example of Loss function

2. Analysis for data distribution, as shown in Figure 19.

Run cGAN_PCA.py and analysis from SPSS

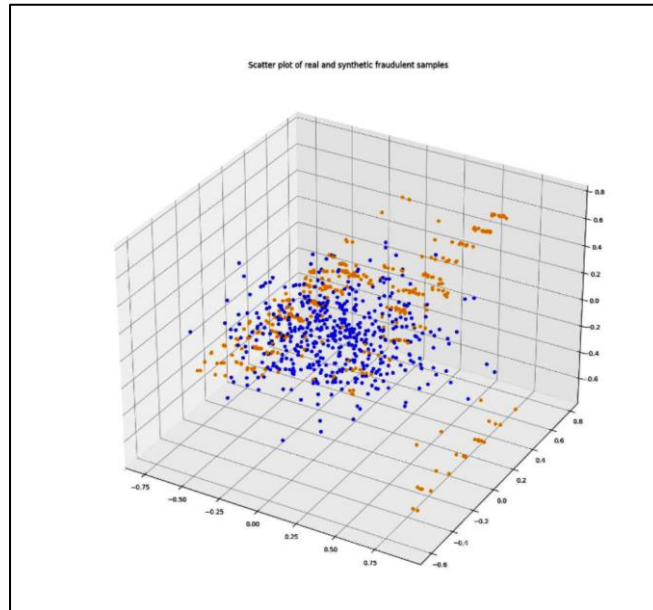


Figure 19. An example of PCA 3D distribution

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.

```
1.  ##### PCA visualization#####
2.  import pandas as pd
3.  import numpy as np
4.  import matplotlib.pyplot as plt
5.  import seaborn as sns
6.  import matplotlib.ticker as mtick
7.  import plotly.express as px
8.  # for scaling
9.  from sklearn.preprocessing import MaxAbsScaler
10. import datetime as dt
11. from sklearn.preprocessing import RobustScaler
12. import seaborn as sns
13. #%matplotlib inline
14. import plotly.graph_objs as go
15. import plotly.figure_factory as ff
16. import plotly.graph_objs as go
17. import plotly
18. import seaborn as sns
19. import plotly.express as px
20. from collections import Counter
21. #from imblearn.over_sampling import SMOTE
22. import matplotlib.gridspec as gridspec
23. import plotly.figure_factory as ff
24. # For GANs
25. #from tensorflow import keras
26. import keras
27. # for PCA
28. from sklearn.decomposition import PCA
29. from plotly import tools
30. from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
31. #init_notebook_mode(connected=True)
32. # MLP model package
33. from sklearn.neural_network import MLPClassifier
34. from sklearn.model_selection import train_test_split
35. from sklearn.metrics import accuracy_score, classification_report
```

```

36. # Isolation Forest and Random Forest packages
37. from sklearn.ensemble import IsolationForest, RandomForestClassifier
38. import gc
39. from datetime import datetime
40. from sklearn.model_selection import train_test_split
41. from sklearn.model_selection import KFold
42. from sklearn.metrics import roc_auc_score
43. from sklearn.ensemble import RandomForestClassifier
44. from sklearn import svm
45. # import lightgbm as lgb
46. # from lightgbm import LGBMClassifier
47. #import xgboost as xgb
48.
49. pd.set_option('display.max_columns', 100)
50.
51. #sns.set(rc = {'figure.figsize':(15,8)})
52. df=pd.read_csv('fraudTest.csv')#read dataset
53. #Data Clean
54. df.isna().sum()# check null data
55. df.drop_duplicates(inplace=True)# drop duplicate
56. df.info()
57. df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
58. df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
59. df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
60. df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
61.
62. # Shuffle the dataframe df
63. df = df.sample(frac = 1).reset_index(drop = True)
64. #Feature extract
65. train=df[['category','amt','zip','lat','long','city_pop','merch_lat',
66. 'merch_long','age','hour','day','month','is_fraud']]
67. #convert category to dummy variables
68. train=pd.get_dummies(train, drop_first=True)
69.
70. y_train=train['is_fraud'].values[0:300000]
71. y=y_train
72. X_train=train.drop("is_fraud", axis='columns').values[0:300000]
73.
74. # scale b/w [-1, 1]
75. X = MaxAbsScaler().fit_transform(X_train)
76.
77. # Let's define the discriminator which takes inputs the feature

```

```

78. # matrix and class vector and predicts the probability of being
79. # fake or real (0 or 1)
80. # 2 Hidden Layer with each layer has 200 nodes
81. def dis():
82.     # define the feature input
83.     feature = keras.Input(shape=(24,))
84.
85.     # define the labels input
86.     labels = keras.Input(shape=(1,))
87.
88.     # merge the two layers
89.     merge = keras.layers.Concatenate()([feature, labels])
90.
91.     # add one hidden layer
92.     model = keras.layers.Dense(200)(merge)
93.     model = keras.layers.LeakyReLU(alpha=0.2)(model)#LeakyReLU
94.
95.     model = keras.layers.Dense(200)(merge)
96.     model = keras.layers.LeakyReLU(alpha=0.2)(model)
97.
98.     # add the output layer
99.     model = keras.layers.Dense(1, activation='sigmoid')(model)
100.
101.     # create a model from the pipeline
102.     d_model = keras.Model(inputs=[feature, labels], outputs=model,
103.         name='discriminator')
104.
105.     # compile the model
106.     d_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002),
107.         loss='binary_crossentropy', metrics=['accuracy'])
108.
109.     # return the model
110.     return d_model
111.
112. # Let's define the generator which takes as input latent
113. # space and class labels and outputs a feature matrix 'X'
114. # generator has 3 hidden layer with 150x200x150 nodes
115. def gen():
116.     # define the latent space
117.     latent = keras.Input(shape=(99,))
118.
119.     # define the label vector

```

```

120. labels = keras.Input(shape=(1,))
121.
122. # merge the two layers
123. merge = keras.layers.Concatenate()([latent, labels])
124.
125. # create 1 hidden layer 1-1000
126. model = keras.layers.Dense(150)(merge)
127. model = keras.layers.LeakyReLU(alpha=0.2)(model)
128.
129. # model = keras.layers.Dense(128)(merge)
130. # model = keras.layers.LeakyReLU(alpha=0.2)(model)
131.
132. # create 2 hidden layer
133. model = keras.layers.Dense(200)(model)
134. model = keras.layers.LeakyReLU(alpha=0.2)(model)
135.
136. # create 3 hidden layer
137. model = keras.layers.Dense(150)(merge)
138. model = keras.layers.LeakyReLU(alpha=0.2)(model)
139.
140. # create an output layer
141. model = keras.layers.Dense(24, activation='tanh')(model)
142.
143. # create a model from the pipeline
144. g_model = keras.Model(inputs=[latent, labels],
145. outputs=model, name='generator')
146.
147. # return the model
148. return g_model
149.
150. # Combine the discriminator and Generator into cGAN
151. # Keep discriminator as non-trainable, so it does
152. # not update its weights during training the cgan.
153. def gan(discriminator, generator):
154.     # Make discriminator as non-trainable
155.     discriminator.trainable = False
156.
157.     # get inputs from the generator
158.     gen_latent, gen_labels = generator.input
159.
160.     # get output from the generator
161.     gen_feature = generator.output

```

```

162.
163.     # get discriminator predictions
164.     dis_predict = discriminator([gen_feature, gen_labels])
165.
166.     # create a model from the pipeline
167.     cgan_model = keras.Model(inputs=[gen_latent, gen_labels],
168.                               outputs=dis_predict, name='cGAN')
169.
170.     # compile the model
171.     cgan_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002),
172.                        loss='binary_crossentropy')
173.
174.     # return the model
175.     return cgan_model
176.
177. # Let's define a generate_real_samples
178. # method to generate random samples as
179. # real data
180. def generate_real_samples(n_samples):
181.     # generate n_samples number of random indices
182.     indices = np.random.randint(0, X.shape[0], n_samples)
183.
184.     # get the newly indexed matrix and vector
185.     X_real = np.array([X[j] for j in indices])
186.     y_real = np.array([y[j] for j in indices])
187.
188.     # classify them as '1' (real)
189.     y_labels = np.ones((n_samples, 1))
190.
191.     # return the computed arrays
192.     return [X_real, y_real], y_labels
193.
194. # Let's define a generate_fake_samples
195. # method to generate random samples as
196. # fake data from latent sapce
197. def generate_latent_point(n_samples):
198.     # create an array of n_samples * late
199.     # -nt_space (= 99)
200.     Z_lat = np.random.randn(n_samples * 99)
201.     X_lat = Z_lat.reshape(n_samples, 99)
202.
203.     # randomly generate the class labels 'y'

```



```

204.     y_lat = np.random.randint(0, 2, n_samples)
205.
206.     # return the points
207.     return [X_lat, y_lat]
208.
209. def generate_fake_samples(generator, n_samples):
210.     # generate n_samples points in latent space
211.     [X_lat, y_lat] = generate_latent_point(n_samples)
212.
213.     # predict on them using the generator
214.     X_fake = generator.predict([X_lat, y_lat])
215.
216.     # classify them as '0' (fake)
217.     y_labels = np.zeros((n_samples, 1))
218.
219.     # return the computed arrays
220.     return [X_fake, y_lat], y_labels
221.
222. # plot the loss functions on the graph
223. def plot_history(d_real, d_fake, g):
224.     fig = plt.figure(figsize=(15, 15))
225.     plt.plot(d_real, label='d_real')
226.     plt.plot(d_fake, label='d_fake')
227.     plt.plot(g, label='cGAN_loss')
228.     # specify legend
229.     plt.legend()
230.     # plot the graph
231.     plt.show()
232.
233. # train the model
234. def train(discriminator, generator, cgan, batch_size=256, epochs=6):
235.
236.     count = 0
237.     half_batch = int(batch_size / 2)
238.     batch_per_epoch = int(X.shape[0] / batch_size)
239.
240.     d_real = []
241.     d_fake = []
242.     g = []
243.     for i in range(epochs):
244.         for j in range(batch_per_epoch):
245.

```

```

246.         # real
247.         [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
248.         # loss on real
249.         d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
250.         # fake
251.         [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
252.                                     half_batch)
253.         # loss on fake
254.         d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)
255.
256.         # Generate latent points
257.         [X_lat, y_lat] = generate_latent_point(batch_size)
258.         # Give them '1' labels
259.         y_labels_lat = np.ones((batch_size, 1))
260.         # calculate generator loss
261.         gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)
262.
263.         # print the results
264.         count += 1
265.         if count % 1000 == 0:
266.             print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'
267.                   .format(count, d1_loss, d2_loss, gan_loss))
268.
269.         # store the losses
270.         d_real.append(d1_loss)
271.         d_fake.append(d2_loss)
272.         g.append(gan_loss)
273.
274.         # plot a summary of loss
275.         plot_history(d_real, d_fake, g)
276.
277. discriminator = dis()
278. generator = gen()
279. cgan = gan(discriminator, generator)
280.
281. print(generator.summary()) # summary for dis
282. print(discriminator.summary()) # summary for gen
283. print(cgan.summary()) # summary for gan
284. # train the cGAN model
285. train(discriminator, generator, cgan)
286.
287. # generate the synthetic fraudulent

```

```

288. # data
289. def generate_new_samples(n_samples):
290.     # get n_samples for latent space
291.     [X_lat, y_lat] = generate_latent_point(n_samples)
292.
293.     # specify the labels
294.     y_lat = np.ones(n_samples)
295.
296.     # predict on generator
297.     X_pred = generator.predict([X_lat, y_lat])
298.
299.     # return fraud samples
300.     return [X_pred, y_lat]
301.
302. # shuffle the dataframe df
303. df = df.sample(frac = 1.0).reset_index(drop = True)
304. # get 'X' and 'y'
305. # X = np.array(df.drop(['is_fraud'], axis = 1))
306. # y = np.array(df['is_fraud'])
307. # scale down [-1, 1]
308. X = MaxAbsScaler().fit_transform(X_train)
309. y=y_train
310. # PCA
311. pca = PCA(n_components = 3)
312. X = pca.fit_transform(X)
313. X=X[0:100000]
314. # plot the 'real' fraud data
315. fig = plt.figure(figsize = (15,15))
316. axs = plt.axes(projection = '3d')
317. for i in range(X.shape[0]) :
318.     if y[i] == 1 :
319.         axs.scatter3D(X[i,0], X[i,1], X[i,2], color = 'orange')
320.
321. # Synthetic data from stand-alone generator
322. [X_syn, y_syn] = generate_new_samples(500)
323.
324. # PCA
325. pca = PCA(n_components = 3)
326. X_syn = pca.fit_transform(X_syn)
327.
328. # plot
329. for i in range(X_syn.shape[0]) :

```

```

330.     if y_syn[i] == 1.0 :
331.         axs.scatter3D(X_syn[i,0], X_syn[i,1], X_syn[i,2], color = 'blue')
332. plt.title('Scatter plot of real and synthetic fraudulent samples')
333. plt.savefig('3D.jpg')
334. 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.

```

1.     ###recover string from num #####
2.     import numpy as np
3.     import pandas as pd
4.
5.     def num2cate(m):
6.         num_list=[]
7.         List1 = ['personal_care', 'health_fitness', 'misc_pos', 'travel', 'kids_pets',
8.                 'shopping_pos', 'food_dining', 'home', 'entertainment', 'shopping_net',
9.                 'misc_net', 'grocery_pos', 'gas_transport', 'grocery_net']
10.        List2 = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
11.        category = dict(zip(List2, List1))
12.        for i in m:
13.            if i<0:
14.                i=0
15.            if i>12:
16.                i=12
17.            a=category[str(round(i))]
18.            num_list.append(a)
19.        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.

```

1.     import pandas as pd
2.     import numpy as np
3.     import matplotlib.pyplot as plt
4.     import seaborn as sns
5.     import matplotlib.ticker as mtick
6.     import plotly.express as px
7.     from sklearn.decomposition import PCA
8.     # for scaling

```

```

9.     from sklearn.preprocessing import MaxAbsScaler
10.    import datetime as dt
11.    import matplotlib.pyplot as plt
12.    from sklearn.preprocessing import RobustScaler
13.    import seaborn as sns
14.    # For GANs
15.    #from tensorflow import keras
16.    import keras
17.    from plotly import tools
18.    import num2category as num2cate
19.    # MLP model package
20.    from sklearn.neural_network import MLPClassifier
21.    from sklearn.model_selection import train_test_split
22.    from sklearn.metrics import accuracy_score, classification_report
23.    # Isolation Forest and Random Forest packages
24.    from sklearn.ensemble import IsolationForest, RandomForestClassifier
25.    from sklearn.neighbors import KNeighborsClassifier
26.    # for scaling
27.    from sklearn.preprocessing import MaxAbsScaler,StandardScaler
28.    import gc
29.    from datetime import datetime
30.    from sklearn.model_selection import train_test_split
31.    from sklearn.model_selection import KFold
32.    from sklearn.metrics import roc_auc_score
33.    from sklearn.ensemble import RandomForestClassifier
34.    from sklearn import svm
35.    # import lightgbm as lgb
36.    # from lightgbm import LGBMClassifier
37.    # import xgboost as xgb
38.
39.    pd.set_option('display.max_columns', 100)
40.
41.    #sns.set(rc = {'figure.figsize':(15,8)})
42.    def get_data():
43.        df=pd.read_csv('fraudTest.csv')
44.        df.isna().sum()
45.        df.drop_duplicates(inplace=True)
46.
47.        df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
48.        df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
49.        df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
50.        df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month

```

```

51.
52.     df = df.sample(frac = 1).reset_index(drop = True)
53.     df=df[['category','amt','zip','lat','long','city_pop',
54.           'merch_lat','merch_long','age','hour','day','month','is_fraud']]
55.     #convert category to dummy variables,
56.     #df=pd.get_dummies(df, drop_first=True)#one-hot
57.     num, index = pd.factorize(df['category'])
58.     df['category'] = num
59.     # Extract from the dataframe, class 1s and 0s
60.     df_1 = df[df['is_fraud'] == 1].sample(frac = 1.0).reset_index(drop = True)
61.     df_2 = df[df['is_fraud'] == 0].sample(frac = 1.0).reset_index(drop = True)
62.     # Split each dataframe to certain fraction
63.     new_df_1, old_df_1 = df_1[: 480].reset_index(drop = True), df_1[480 :].reset_index(drop
= True)
64.     new_df_2, old_df_2 = df_2[: 999].reset_index(drop = True), df_2[999 :].reset_index(drop
= True)
65.
66.     # group them into test and train sets
67.     test, train = pd.concat([new_df_1, new_df_2]), pd.concat([old_df_1, old_df_2])
68.     test, train = test.sample(frac = 1.0).reset_index(drop = True),
69.     train.sample(frac = 1.0).reset_index(drop = True)
70.
71.     X_train = np.array(train.drop(['is_fraud'], axis = 1))
72.     y_train = np.array(train['is_fraud'])
73.
74.     X_test = np.array(test.drop(['is_fraud'], axis = 1))
75.     y_test = np.array(test['is_fraud'])
76.
77.     X_train= StandardScaler().fit_transform(X_train)
78.     X_test = StandardScaler().fit_transform(X_test)
79.
80.     return X_train,y_train,X_test,y_test
81.
82. def MLP(X, y, param):
83.     # train test split
84.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
85.
86.     # hyper-parameters
87.     nodes = param['nodes']
88.     lrate = param['lrate']
89.     toler = param['toler']
90.     batch = param['batch_size']

```

```

91.
92.     # define the model
93.     model = MLPClassifier(hidden_layer_sizes=(nodes,), tol=toler,
94.         batch_size=batch, learning_rate_init=lrate,
95.         verbose=0, random_state=1)
96.
97.     # train the model
98.     model.fit(X_train, y_train)
99.
100.    # predict on the test set
101.    predictions = model.predict(X_test)
102.
103.    # evaluate the accuracy and classification report
104.    print('MLP : ' + str(accuracy_score(y_test, predictions)))
105.    print(classification_report(y_test, predictions))
106.
107.    # assign this to models_dict
108.    models_dict['MLP'] = model
109.
110. def RFR(X, y, split):
111.     # train test split
112.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
113.
114.     # define the model
115.     model = RandomForestClassifier(n_estimators=200, criterion='gini',
116.         min_samples_split=split, random_state=1)
117.
118.     # train the model
119.     model.fit(X_train, y_train)
120.
121.     # predict on the test set
122.     predictions = model.predict(X_test)
123.
124.     # evaluate the accuracy and classification report
125.     print('RFR : ' + str(accuracy_score(y_test, predictions)))
126.     print(classification_report(y_test, predictions))
127.
128.     # assign this to models_dict
129.     models_dict['RFR'] = model
130.
131. def KNN(X, y):
132.     # train test split

```

```

133.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.99, random_state=1)
134.
135.     # define the model
136.     model = KNeighborsClassifier()
137.
138.     # train the model
139.     model.fit(X_train, y_train)
140.
141.     # predict on the test set
142.     predictions = model.predict(X_test)
143.
144.     # evaluate the accuracy and classification report
145.     print('KNN : ' + str(accuracy_score(y_test, predictions)))
146.     print(classification_report(y_test, predictions))
147.
148.     # assign this to models_dict
149.     models_dict['KNN'] = model
150.
151. if __name__ == '__main__':
152.     models_dict = {}
153.
154.     def IsF(X, y, ratio):
155.         # train test split
156.         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
157.
158.         # define the model
159.         model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)
160.
161.         # train the model
162.         model.fit(X_train, y_train)
163.
164.         # predict on the test set
165.         predictions = model.predict(X_test)
166.
167.         # Convert the predictions according to problem profile
168.         predictions[predictions == 1] = 0
169.         predictions[predictions == -1] = 1
170.
171.         # evaluate the accuracy and classification report
172.         print('IsF : ' + str(accuracy_score(y_test, predictions)))
173.         print(classification_report(y_test, predictions))
174.

```



```

175.     # assign this to models_dict
176.     models_dict['IsF'] = model
177.
178.     X_train, y_train, X_test, y_test=get_data()
179.     # For Isolation Forest Model
180.     ratio = float(y_train[y_train == 1].shape[0] / y_train.shape[0])
181.     # For Multi Layer Perceptron
182.     param = {
183.         'nodes': 170,
184.         'lrate': 0.00005,
185.         'toler': 0.00001,
186.         'batch_size': 100
187.     }
188.     # For Random Forest Classifier
189.     split = 2
190.     IsF(X_train, y_train, ratio)
191.     pred_1 = models_dict['IsF'].predict(X_test)
192.     pred_1[pred_1 == 1] = 0
193.     pred_1[pred_1 == -1] = 1
194.     print('IsF : ' + str(accuracy_score(y_test, pred_1)))
195.     print(classification_report(y_test, pred_1))
196.
197.     #print('---MLP---')
198.     #MLP(X_train, y_train, param)
199.     # MLP
200.     # pred_2 = models_dict['MLP'].predict(X_test)
201.     # print('MLP : ' + str(accuracy_score(y_test, pred_2)))
202.     # print(classification_report(y_test, pred_2))
203.
204.     # RF
205.     print('--RF---')
206.     RFR(X_train, y_train, split)
207.     pred_3 = models_dict['RFR'].predict(X_test)
208.     print('RFR : ' + str(accuracy_score(y_test, pred_3)))
209.     print(classification_report(y_test, pred_3))
210.
211.     #KNN
212.     print('--KNN---')
213.     KNN(X_train, y_train)
214.     pred_4 = models_dict['KNN'].predict(X_test)
215.     print('KNN : ' + str(accuracy_score(y_test, pred_4)))
216.     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.

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.  import seaborn as sns
5.  import matplotlib.ticker as mtick
6.  import plotly.express as px
7.  from sklearn.decomposition import PCA
8.  # for scaling
9.  from sklearn.preprocessing import MaxAbsScaler
10. import datetime as dt
11. import pandas as pd
12. import numpy as np
13. import matplotlib.pyplot as plt
14. from sklearn.preprocessing import RobustScaler
15. import seaborn as sns
16. import plotly.graph_objs as go
17. import plotly.figure_factory as ff
18. import plotly.graph_objs as go
19. import plotly
20. import seaborn as sns
21. import plotly.express as px
22. from collections import Counter
23. #from imblearn.over_sampling import SMOTE
24. import matplotlib.gridspec as gridspec
25. import plotly.figure_factory as ff
26. # For GANs
27. #from tensorflow import keras
28. import keras
29. # for PCA
30. from sklearn.decomposition import PCA
31.
32. from plotly import tools
33.
34. # MLP model package
35. from sklearn.neural_network import MLPClassifier
36. from sklearn.model_selection import train_test_split
```

```

37. from sklearn.metrics import accuracy_score, classification_report
38.
39. # Isolation Forest and Random Forest packages
40. from sklearn.ensemble import IsolationForest, RandomForestClassifier
41.
42. # for scaling
43. from sklearn.preprocessing import MaxAbsScaler
44.
45. import gc
46. from datetime import datetime
47. from sklearn.model_selection import train_test_split
48. from sklearn.model_selection import KFold
49. from sklearn.metrics import roc_auc_score
50. from sklearn.ensemble import RandomForestClassifier
51. from sklearn import svm
52. # import lightgbm as lgb
53. # from lightgbm import LGBMClassifier
54. #import xgboost as xgb
55.
56. pd.set_option('display.max_columns', 100)
57.
58. #sns.set(rc = {'figure.figsize':(15,8)})
59. def get_data():
60.     df=pd.read_csv('fraudTest.csv')
61.     df.isna().sum()
62.     df.drop_duplicates(inplace=True)
63.
64.     df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
65.     df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
66.     df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
67.     df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
68.
69.     df = df.sample(frac = 1).reset_index(drop = True)
70.     df=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','
day','month','is_fraud']]
71.     #convert category to dummy variables,
72.     df=pd.get_dummies(df, drop_first=True)#one-hot
73.     # Extract from the dataframe, class 1s and 0s
74.     df_1 = df[df['is_fraud'] == 1].sample(frac = 1.0).reset_index(drop = True)
75.     df_2 = df[df['is_fraud'] == 0].sample(frac = 1.0).reset_index(drop = True)
76.
77.

```

```

78.     # Split each dataframe to certain fraction
79.         new_df_1, old_df_1 = df_1[: 480].reset_index(drop = True),
df_1[480 :].reset_index(drop = True)
80.         new_df_2, old_df_2 = df_2[: 999].reset_index(drop = True),
df_2[999 :].reset_index(drop = True)
81.
82.     # group them into test and train sets
83.     test, train = pd.concat([new_df_1, new_df_2]), pd.concat([old_df_1, old_df_2])
84.     test, train = test.sample(frac = 1.0).reset_index(drop = True), train.sample(frac =
1.0).reset_index(drop = True)
85.
86.     X_train = np.array(train.drop(['is_fraud'], axis = 1))
87.     y_train = np.array(train['is_fraud'])
88.
89.     X_test = np.array(test.drop(['is_fraud'], axis = 1))
90.     y_test = np.array(test['is_fraud'])
91.
92.     X_train= MaxAbsScaler().fit_transform(X_train)
93.     X_test = MaxAbsScaler().fit_transform(X_test)
94.
95.     return X_train,y_train,X_test,y_test
96. if __name__=='__main__':
97.     models_dict = { }
98.
99.     def IsF(X, y, ratio):
100.         # train test split
101.         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1)
102.
103.         # define the model
104.         model = IsolationForest(n_estimators=200, contamination=ratio,
random_state=1)
105.
106.         # train the model
107.         model.fit(X_train, y_train)
108.
109.         # predict on the test set
110.         predictions = model.predict(X_test)
111.
112.         # Convert the predictions according to problem profile
113.         predictions[predictions == 1] = 0
114.         predictions[predictions == -1] = 1

```

```

115.
116.     # evaluate the accuracy and classification report
117.     # print('IsF : ' + str(accuracy_score(y_test, predictions)))
118.     # print(classification_report(y_test, predictions))
119.
120.     # assign this to models_dict
121.     models_dict['IsF'] = model
122.
123. X_train, y_train, X_test, y_test=get_data()
124. # For Isolation Forest Model
125. ratio = float(y_train[y_train == 1].shape[0] / y_train.shape[0])
126.
127. # For Multi Layer Perceptron
128. param = {
129.     'nodes': 170,
130.     'lr': 0.00005,
131.     'toler': 0.00001,
132.     'batch_size': 100
133. }
134.
135. # For Random Forest Classifier
136. split = 2
137. IsF(X_train, y_train, ratio)
138.
139. pred_1 = models_dict['IsF'].predict(X_test)
140. pred_1[pred_1 == 1] = 0
141. pred_1[pred_1 == -1] = 1
142. print('IsF : ' + str(accuracy_score(y_test, pred_1)))
143. 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.

```

1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. import matplotlib.ticker as mtick
6. import plotly.express as px

```

```

7.  from sklearn.decomposition import PCA
8.  # for scaling
9.  from sklearn.preprocessing import MaxAbsScaler
10. import datetime as dt
11. from Ori_3ML import get_data
12. import matplotlib as mpl
13. mpl.use('TkAgg')
14. import matplotlib.pyplot as plt
15. from sklearn.preprocessing import RobustScaler
16. import seaborn as sns
17. #%matplotlib inline
18. import plotly.graph_objs as go
19. import plotly.figure_factory as ff
20. import plotly.graph_objs as go
21. import plotly
22. import seaborn as sns
23. import plotly.express as px
24. from collections import Counter
25. #from imblearn.over_sampling import SMOTE
26. import matplotlib.gridspec as gridspec
27. import plotly.figure_factory as ff
28. # For GANs
29. #from tensorflow import keras
30. import keras
31. # for PCA
32. from sklearn.decomposition import PCA
33. #from ori_IsF import MLP,RFR,KNN
34.
35. from plotly import tools
36.
37. # MLP model package
38. from sklearn.neural_network import MLPClassifier
39. from sklearn.model_selection import train_test_split
40. from sklearn.metrics import accuracy_score, classification_report
41.
42. # Isolation Forest and Random Forest packages
43. from sklearn.ensemble import IsolationForest, RandomForestClassifier
44. from sklearn.neighbors import KNeighborsClassifier
45. # for scaling
46. from sklearn.preprocessing import MaxAbsScaler,MinMaxScaler,StandardScaler
47. from num2category import num2cate
48.

```

```

49. import gc
50. from datetime import datetime
51. from sklearn.model_selection import train_test_split
52. from sklearn.model_selection import KFold
53. from sklearn.metrics import roc_auc_score
54. from sklearn.ensemble import RandomForestClassifier, IsolationForest
55. from sklearn import svm
56. # import lightgbm as lgb
57. # from lightgbm import LGBMClassifier
58. #import xgboost as xgb
59.
60. pd.set_option('display.max_columns', 100)
61.
62. #sns.set(rc = {'figure.figsize':(15,8)})
63. df=pd.read_csv('fraudTest.csv')
64. df.isna().sum()
65. df.drop_duplicates(inplace=True)
66. df.info()
67. df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
68. df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
69. df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
70. df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
71.
72. # Shuffle the dataframe df
73. df = df.sample(frac = 1).reset_index(drop = True)
74. train=df[['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day','month','is_fraud']]
75. train_save=train[0:150000]
76. train_save.to_csv('Ori_data.csv',index=None)
77. #convert category to dummy variables,
78. #train=pd.get_dummies(train, drop_first=True)
79. num,index=pd.factorize(train['category'])
80. train['category']=num
81.
82. y_train=train['is_fraud'].values
83. X_train=train.drop("is_fraud", axis='columns').values
84. y=y_train
85. # scale b/w [-1, 1]
86. # X = MaxAbsScaler().fit_transform(X_train)
87. mm=StandardScaler()
88. X=mm.fit_transform(X_train)
89. # Let's define the discriminator which takes inputs the feature

```

```

90. # matrix and class vector and predicts the probability of being
91. # fake or real (0 or 1)
92. # 2 Hidden Layer with each layer has 200 nodes
93. def dis():
94.     # define the feature input
95.     feature = keras.Input(shape=(12,))
96.
97.     # define the labels input
98.     labels = keras.Input(shape=(1,))
99.
100.    # merge the two layers
101.    merge = keras.layers.Concatenate()([feature, labels])
102.
103.    # add one hidden layer
104.    model = keras.layers.Dense(32)(merge)
105.    model = keras.layers.LeakyReLU(alpha=0.2)(model)
106.
107.    model = keras.layers.Dense(64)(model)
108.    model = keras.layers.LeakyReLU(alpha=0.2)(model)
109.
110.    model = keras.layers.Dense(32)(model)
111.    model = keras.layers.LeakyReLU(alpha=0.2)(model)
112.
113.    # add the output layer
114.    model = keras.layers.Dense(1, activation='sigmoid')(model)
115.
116.    # create a model from the pipeline
117.    d_model = keras.Model(inputs=[feature, labels], outputs=model,
name='discriminator')
118.
119.    # compile the model
120.    d_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rate=0.000
1), loss='binary_crossentropy',
121.                    metrics=['accuracy'])
122.
123.    # return the model
124.    return d_model
125.
126. # Let's define the generator which takes as input latent
127. # space and class labels and outputs a feature matrix 'X'
128. # generator has 3 hidden layer with 150x200x150 nodes
129. def gen():

```



```

130. # define the latent space
131. latent = keras.Input(shape=(99,))
132.
133. # define the label vector
134. labels = keras.Input(shape=(1,))
135.
136. # merge the two layers
137. merge = keras.layers.Concatenate()([latent, labels])
138.
139. # create 1 hidden layer
140. model = keras.layers.Dense(32)(merge)
141. model = keras.layers.LeakyReLU(alpha=0.2)(model)
142.
143. # create 2 hidden layer
144. model = keras.layers.Dense(64)(model)
145. model = keras.layers.LeakyReLU(alpha=0.2)(model)
146.
147. # create 3 hidden layer
148. model = keras.layers.Dense(32)(merge)#merge
149. model = keras.layers.LeakyReLU(alpha=0.2)(model)
150.
151. # create an ouptput layer
152. model = keras.layers.Dense(12, activation='tanh')(model)
153.
154. # create a model from the pipeline
155. g_model = keras.Model(inputs=[latent, labels], outputs=model, name='generator')
156.
157. # return the model
158. return g_model
159.
160. # Combine the discriminator and Generator into cGAN
161. # Keep discriminator as non-trainable, so it does
162. # not update its weights during training the cgan.
163. def gan(discriminator, generator):
164.     # Make discriminator as non-trainable
165.     discriminator.trainable = False
166.
167.     # get inputs from the generator
168.     gen_latent, gen_labels = generator.input
169.
170.     # get output from the generator
171.     gen_feature = generator.output

```

```

172.
173.     # get discriminator predictions
174.     dis_predict = discriminator([gen_feature, gen_labels])
175.
176.     # create a model from the pipeline
177.     cgan_model = keras.Model(inputs=[gen_latent, gen_labels], outputs=dis_predict,
                               name='cGAN')
178.
179.     # compile the model
180.     cgan_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rate=0.0
                               001), loss='binary_crossentropy')
181.
182.     # return the model
183.     return cgan_model
184.
185. # Let's define a generate_real_samples
186. # method to generate random samples as
187. # real data
188. def generate_real_samples(n_samples):
189.     # generate n_samples number of random indices
190.     indices = np.random.randint(0, X.shape[0], n_samples)
191.
192.     # get the newly indexed matrix and vector
193.     X_real = np.array([X[j] for j in indices])
194.     y_real = np.array([y[j] for j in indices])
195.
196.     # classify them as '1' (real)
197.     y_labels = np.ones((n_samples, 1))
198.
199.     # return the computed arrays
200.     return [X_real, y_real], y_labels
201. # Let's define a generate_fake_samples
202. # method to generate random samples as
203. # fake data from latent sapce
204. def generate_latent_point(n_samples):
205.     # create an array of n_samples * late
206.     # -nt_space (= 99)
207.     Z_lat = np.random.randn(n_samples * 99)
208.     X_lat = Z_lat.reshape(n_samples, 99)
209.
210.     # randomly generate the class labels 'y'
211.     y_lat = np.random.randint(0, 2, n_samples)

```

```

212.
213.     # return the points
214.     return [X_lat, y_lat]
215.
216. def generate_fake_samples(generator, n_samples):
217.     # generate n_samples points in latent space
218.     [X_lat, y_lat] = generate_latent_point(n_samples)
219.
220.     # predict on them using the generator
221.     X_fake = generator.predict([X_lat, y_lat])
222.
223.     # classify them as '0' (fake)
224.     y_labels = np.zeros((n_samples, 1))
225.
226.     # return the computed arrays
227.     return [X_fake, y_lat], y_labels
228.
229. # plot the loss functions on the graph
230. def plot_history(d_real, d_fake, g):
231.     fig = plt.figure(figsize=(15, 15))
232.
233.     plt.plot(d_real, label='d_real')
234.     plt.plot(d_fake, label='d_fake')
235.     plt.plot(g, label='cGAN_loss')
236.
237.     # specify legend
238.     plt.legend()
239.
240.     # plot the graph
241.     plt.savefig('loss.jpg')
242.     plt.show()
243.
244. # train the model
245. def train(discriminator, generator, cgan, batch_size=512, epochs=6):
246.
247.     count = 0
248.     half_batch = int(batch_size / 2)
249.     batch_per_epoch = int(X.shape[0] / batch_size)
250.
251.     d_real = []
252.     d_fake = []
253.     g = []

```

```

254.     for i in range(epochs):
255.         for j in range(batch_per_epoch):
256.
257.             # real
258.             [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
259.             # loss on real
260.             d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
261.             # fake
262.             [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
263.                                     half_batch)
264.             # loss on fake
265.             d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)
266.
267.             # Generate latent points
268.             [X_lat, y_lat] = generate_latent_point(batch_size)
269.             # Give them '1' labels
270.             y_labels_lat = np.ones((batch_size, 1))
271.             # calculate generator loss
272.             gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)
273.
274.             # print the results
275.             count += 1
276.             if count % 1000 == 0:
277.                 print('{: > 5} | {: > 5} | {: > 5} | {: > 5}'.format(count, d1_loss, d2_loss,
gan_loss))
278.
279.             # store the losses
280.             d_real.append(d1_loss)
281.             d_fake.append(d2_loss)
282.             g.append(gan_loss)
283.
284.             # plot a summary of loss
285.             plot_history(d_real, d_fake, g)
286.
287. discriminator = dis()
288. generator = gen()
289. cgan = gan(discriminator, generator)
290.
291. print(generator.summary())    # summary for dis
292. print(discriminator.summary()) # summary for gen
293. print(cgan.summary())        # summary for gan
294. # train the cGAN model

```

```

295. train(discriminator, generator, cgan)
296.
297. # generate the synthetic fraudulent
298. # data
299. def generate_new_samples(n_samples):
300.     # get n_samples for latent space
301.     [X_lat, y_lat] = generate_latent_point(n_samples)
302.
303.     # specify the labels
304.     y_lat = np.zeros(n_samples)
305.
306.     # predict on generator
307.     X_pred = generator.predict([X_lat, y_lat])
308.
309.     # return fraud samples
310.     return [X_pred, y_lat]
311.
312. # shuffle the dataframe df
313. df = df.sample(frac = 1.0).reset_index(drop = True)
314.
315. # get 'X' and 'y'
316. # X = np.array(df.drop(['is_fraud'], axis = 1))
317. # y = np.array(df['is_fraud'])
318.
319. # scale down [-1, 1]
320. X = StandardScaler().fit_transform(X_train)
321. y=y_train
322.
323. #####Machine Learning#####
324. models_dict_IsF = { }
325. models_dict_RFR={ }
326. models_dict_KNN={ }
327. def IsF(X, y, ratio):
328.     # train test split
329.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
330. random_state=1)
331.
332.     # define the model
333.     model = IsolationForest(n_estimators=200, contamination=ratio, random_state=1)

```

```

334. # train the model
335. model.fit(X_train, y_train)
336.
337. # predict on the test set
338. predictions = model.predict(X_test)
339.
340. # Convert the predictions according to problem profile
341. predictions[predictions == 1] = 0
342. predictions[predictions == -1] = 1
343.
344. # evaluate the accuracy and classification report
345. print('IsF : ' + str(accuracy_score(y_test, predictions)))
346. print(classification_report(y_test, predictions))
347.
348. # assign this to models_dict
349. models_dict_IsF['IsF'] = model
350. def RFR(X, y, split):
351.     # train test split
352.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
353. random_state=1)
354.
355.     # define the model
356.     model = RandomForestClassifier(n_estimators=200, criterion='gini',
357. min_samples_split=split, random_state=1)
358.
359.     # train the model
360.     model.fit(X_train, y_train)
361.
362.     # predict on the test set
363.     predictions = model.predict(X_test)
364.
365.     # evaluate the accuracy and classification report
366.     print('RFR : ' + str(accuracy_score(y_test, predictions)))
367.     print(classification_report(y_test, predictions))
368.
369.     # assign this to models_dict
370.     models_dict_RFR['RFR'] = model
371. def KNN(X, y):
372.     # train test split
373.     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
374. random_state=1)
375.
376.     # define the model
377.     model = KNeighborsClassifier(n_neighbors=5, metric='euclidean',
378. min_samples_split=split, random_state=1)
379.
380.     # train the model
381.     model.fit(X_train, y_train)
382.
383.     # predict on the test set
384.     predictions = model.predict(X_test)
385.
386.     # evaluate the accuracy and classification report
387.     print('KNN : ' + str(accuracy_score(y_test, predictions)))
388.     print(classification_report(y_test, predictions))
389.
390.     # assign this to models_dict
391.     models_dict_KNN['KNN'] = model

```

```

373. # define the model
374. model = KNeighborsClassifier()
375.
376. # train the model
377. model.fit(X_train, y_train)
378.
379. # predict on the test set
380. predictions = model.predict(X_test)
381.
382. # evaluate the accuracy and classification report
383. print('KNN : ' + str(accuracy_score(y_test, predictions)))
384. print(classification_report(y_test, predictions))
385.
386. # assign this to models_dict
387. models_dict_KNN['KNN'] = model
388. ratio = float(y_train[y_train == 1].shape[0]/y_train.shape[0])
389. # For Multi Layer Perceptron
390. param = {
391.     'nodes' : 170,
392.     'lrates' : 0.00005,
393.     'toler' : 0.00001,
394.     'batch_size' : 100
395. }
396.
397. print("Test performance:")
398.
399. # get the numpy array
400. # X = np.array(train.drop(['is_fraud'], axis = 1))
401. # y = np.array(train['is_fraud'])
402. X=X_train
403. y=y_train
404.
405. # scale [-1, 1]
406. X = StandardScaler().fit_transform(X)
407.
408. #X_gan_test, y_gan_test = X[:1000,:], y[:1000]
409. X_train, y_train, X_test, y_test=get_data()
410. X_gan_test,y_gan_test=X_test,y_test
411. X, y = X[1000:,:], y[1000:]
412. # get synthetic samples
413. [X_syn, y_syn] = generate_new_samples(150000)
414. X_syn=mm.inverse_transform(X_syn)

```

```

415.
416. """
417. # PCA
418. pca = PCA(n_components = 3)
419. X = pca.fit_transform(X)
420. X=X[0:150000]
421. ## plot the 'real' fraud data
422. fig = plt.figure(figsize = (15,15))
423. axs = plt.axes(projection = '3d')
424. for i in range(X.shape[0]) :
425.     if y[i] == 1 :
426.         axs.scatter3D(X[i,0], X[i,1], X[i,2], color = 'orange')
427. # Synthetic data from stand-alone generator
428. #[X_syn, y_syn] = generate_new_samples(1000)
429. ## PCA
430. pca = PCA(n_components = 3)
431. X_syn = pca.fit_transform(X_syn)
432. #
433. ## plot
434. for i in range(X_syn.shape[0]) :
435.     if y_syn[i] == 1.0 :
436.         axs.scatter3D(X_syn[i,0], X_syn[i,1], X_syn[i,2], color = 'blue')
437. plt.title('Scatter plot of real and synthetic fraudulent samples')
438. plt.savefig('3D_2.jpg')
439. plt.show()
440. """
441.
442. #Save data
443. col=['category','amt','zip','lat','long','city_pop','merch_lat','merch_long','age','hour','day',
      'month','is_fraud']
444. Syn_data=np.concatenate((np.array(X_syn),np.array(y_syn.reshape(-1,1))),axis=1)
445. Syn_data=pd.DataFrame(Syn_data,columns=col)
446. Syn_data['category']=num2cate(Syn_data['category'])
447. Syn_data.to_csv('Syn_data.csv',index=None)
448. # reshape into 2d array
449. y, y_syn = y.reshape(-1,1), y_syn.reshape(-1,1)
450. # concatenate all the arrays
451. X_1, X_2 = np.concatenate((X,y), axis = 1), np.concatenate((X_syn,y_syn), axis = 1)
452. # merge the above two arrays
453. X_3 = np.concatenate((X_1, X_2), axis = 0)
454. np.random.shuffle(X_3)
455.

```



```

456. X_gan = np.array(X_3[:, :12])
457. y_gan = np.array(X_3[:, 12])
458. ratio = float(y_gan[y_gan == 1].shape[0]/y_gan.shape[0])
459.
460. print('-----IsF-----')
461. IsF(X_gan, y_gan, ratio)
462. pred_1 = models_dict_IsF['IsF'].predict(X_gan_test)
463. pred_1[pred_1 == 1] = 0
464. pred_1[pred_1 == -1] = 1
465. print('IsF : ' + str(accuracy_score(y_gan_test, pred_1)))
466. print(classification_report(y_gan_test, pred_1))
467.
468. """
469. print('-----MLP-----')
470. MLP(X_gan, y_gan, param)
471. pred_2 = models_dict['MLP'].predict(X_gan_test)
472. print('MLP : ' + str(accuracy_score(y_gan_test, pred_2)))
473. print(classification_report(y_gan_test, pred_2))
474. """
475. print('-----RF-----')
476. RFR(X_gan, y_gan, 2)
477. pred_3 = models_dict_RFR['RFR'].predict(X_gan_test)
478. print('RFR : ' + str(accuracy_score(y_gan_test, pred_3)))
479. print(classification_report(y_gan_test, pred_3))
480.
481. print('-----KNN-----')
482. KNN(X_gan, y_gan)
483. pred_4 = models_dict_KNN['KNN'].predict(X_gan_test)
484. print('KNN : ' + str(accuracy_score(y_gan_test, pred_4)))
485. 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%.

```

1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. import matplotlib.ticker as mtick
6. import plotly.express as px
7. from sklearn.decomposition import PCA

```

```

8.  # for scaling
9.  from sklearn.preprocessing import MaxAbsScaler
10. import datetime as dt
11. from ori_IsF import get_data
12. import pandas as pd
13. import numpy as np
14. import matplotlib as mpl
15. mpl.use('TkAgg')
16. import matplotlib.pyplot as plt
17. from sklearn.preprocessing import RobustScaler
18. import seaborn as sns
19. %matplotlib inline
20. import plotly.graph_objs as go
21. import plotly.figure_factory as ff
22. import plotly.graph_objs as go
23. import plotly
24. import seaborn as sns
25. import plotly.express as px
26. from collections import Counter
27. #from imblearn.over_sampling import SMOTE
28. import matplotlib.gridspec as gridspec
29. import plotly.figure_factory as ff
30. # For GANs
31. #from tensorflow import keras
32. import keras
33. # for PCA
34. from sklearn.decomposition import PCA
35.
36. from plotly import tools
37. #from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
  iplot
38. #init_notebook_mode(connected=True)
39.
40. # MLP model package
41. from sklearn.neural_network import MLPClassifier
42. from sklearn.model_selection import train_test_split
43. from sklearn.metrics import accuracy_score, classification_report
44.
45. # Isolation Forest and Random Forest packages
46. from sklearn.ensemble import IsolationForest, RandomForestClassifier
47.
48. # for scaling

```

```

49. from sklearn.preprocessing import MaxAbsScaler
50.
51. import gc
52. from datetime import datetime
53. from sklearn.model_selection import train_test_split
54. from sklearn.model_selection import KFold
55. from sklearn.metrics import roc_auc_score
56. from sklearn.ensemble import RandomForestClassifier, IsolationForest
57. from sklearn import svm
58. # import lightgbm as lgb
59. # from lightgbm import LGBMClassifier
60. #import xgboost as xgb
61.
62. pd.set_option('display.max_columns', 100)
63.
64. #sns.set(rc = {'figure.figsize':(15,8)})
65. df=pd.read_csv('fraudTest.csv')
66. df.isna().sum()
67. df.drop_duplicates(inplace=True)
68. df.info()
69. df['age']=dt.date.today().year-pd.to_datetime(df['dob']).dt.year
70. df['hour']=pd.to_datetime(df['trans_date_trans_time']).dt.hour
71. df['day']=pd.to_datetime(df['trans_date_trans_time']).dt.dayofweek
72. df['month']=pd.to_datetime(df['trans_date_trans_time']).dt.month
73.
74. # Shuffle the dataframe df
75. df = df.sample(frac = 1).reset_index(drop = True)
76. train=df[['category','amt','zip','lat','long','city_pop','merch_lat','me
rch_long','age','hour','day','month','is_fraud']]
77. #convert category to dummy variables
78. train=pd.get_dummies(train, drop_first=True)
79.
80. y_train=train['is_fraud'].values[0:300000]
81. X_train=train.drop("is_fraud", axis='columns').values[0:300000]
82. y=y_train
83. # scale b/w [-1, 1]
84. X = MaxAbsScaler().fit_transform(X_train)
85.
86. # Let's define the discriminator which takes inputs the feature
87. # matrix and class vector and predicts the probability of being
88. # fake or real (0 or 1)
89. # 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,
112.                           name='discriminator')
113.
114.    # compile the model
115.    d_model.compile(optimizer=keras.optimizers.adam_v2.Adam(learning_rate=0.0002), loss='binary_crossentropy',
116.                   metrics=['accuracy'])
117.
118.    # return the model
119.    return d_model
120.
121. # Let's define the generator which takes as input latent
122. # space and class labels and outputs a feature matrix 'X'
123. # generator has 3 hidden layer with 150x200x150 nodes
124. def gen():
125.     # define the latent space
126.     latent = keras.Input(shape=(99,))
127.
128.     # define the label vector
129.     labels = keras.Input(shape=(1,))
130.
131.     # merge the two layers

```

```

131.     merge = keras.layers.Concatenate()([latent, labels])
132.
133.     # create 1 hidden layer
134.     model = keras.layers.Dense(150)(merge)
135.     model = keras.layers.LeakyReLU(alpha=0.2)(model)
136.
137.     # create 2 hidden layer
138.     model = keras.layers.Dense(200)(model)
139.     model = keras.layers.LeakyReLU(alpha=0.2)(model)
140.
141.     # create 3 hidden layer
142.     model = keras.layers.Dense(150)(merge)
143.     model = keras.layers.LeakyReLU(alpha=0.2)(model)
144.
145.     # create an ouptput layer
146.     model = keras.layers.Dense(24, activation='tanh')(model)
147.
148.     # create a model from the pipeline
149.     g_model = keras.Model(inputs=[latent, labels], outputs=model,
name='generator')
150.
151.     # return the model
152.     return g_model
153.
154. # Combine the discriminator and Generator into cGAN
155. # Keep discriminator as non-trainable, so it does
156. # not update its weights during training the cgan.
157. def gan(discriminator, generator):
158.     # Make discriminator as non-trainable
159.     discriminator.trainable = False
160.
161.     # get inputs from the generator
162.     gen_latent, gen_labels = generator.input
163.
164.     # get output from the generator
165.     gen_feature = generator.output
166.
167.     # get discriminator predictions
168.     dis_predict = discriminator([gen_feature, gen_labels])
169.
170.     # create a model from the pipeline
171.     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

```

```

213.     [X_lat, y_lat] = generate_latent_point(n_samples)
214.
215.     # predict on them using the generator
216.     X_fake = generator.predict([X_lat, y_lat])
217.
218.     # classify them as '0' (fake)
219.     y_labels = np.zeros((n_samples, 1))
220.
221.     # return the computed arrays
222.     return [X_fake, y_lat], y_labels
223.
224. # plot the loss functions on the graph
225. def plot_history(d_real, d_fake, g):
226.     fig = plt.figure(figsize=(15, 15))
227.
228.     plt.plot(d_real, label='d_real')
229.     plt.plot(d_fake, label='d_fake')
230.     plt.plot(g, label='cGAN_loss')
231.
232.     # specify legend
233.     plt.legend()
234.
235.     # plot the graph
236.     plt.savefig('loss.jpg')
237.     #plt.show()
238.
239. # train the model
240. def train(discriminator, generator, cgan, batch_size=512, epochs=6):
241.
242.     count = 0
243.     half_batch = int(batch_size / 2)
244.     batch_per_epoch = int(X.shape[0] / batch_size)
245.
246.     d_real = []
247.     d_fake = []
248.     g = []
249.     for i in range(epochs):
250.         for j in range(batch_per_epoch):
251.
252.             # real
253.             [X_real, y_real], y_labels_real =
generate_real_samples(half_batch)

```

```

254.         # loss on real
255.         d1_loss, _ = discriminator.train_on_batch([X_real, y_real],
y_labels_real)
256.         # fake
257.         [X_fake, y_fake], y_labels_fake =
generate_fake_samples(generator,
258.                                     half
_batch)
259.         # loss on fake
260.         d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake],
y_labels_fake)
261.
262.         # Generate latent points
263.         [X_lat, y_lat] = generate_latent_point(batch_size)
264.         # Give them '1' labels
265.         y_labels_lat = np.ones((batch_size, 1))
266.         # calculate generator loss
267.         gan_loss = cgan.train_on_batch([X_lat, y_lat], y_labels_lat)
268.
269.         # print the results
270.         count += 1
271.         if count % 1000 == 0:
272.             print('{: > 5} | {: > 5} | {: > 5} | {: >
5}'.format(count, d1_loss, d2_loss, gan_loss))
273.
274.         # store the losses
275.         d_real.append(d1_loss)
276.         d_fake.append(d2_loss)
277.         g.append(gan_loss)
278.
279.         # plot a summary of loss
280.         plot_history(d_real, d_fake, g)
281.
282. discriminator = dis()
283. generator = gen()
284. cgan = gan(discriminator, generator)
285.
286. print(generator.summary()) # summary for dis
287. print(discriminator.summary()) # summary for gen
288. print(cgan.summary()) # summary for gan
289. # train the cGAN model
290. train(discriminator, generator, cgan)
291.

```



```

292. # generate the synthetic fraudulent
293. # data
294. def generate_new_samples(n_samples):
295.     # get n_samples for latent space
296.     [X_lat, y_lat] = generate_latent_point(n_samples)
297.
298.     # specify the labels
299.     y_lat = np.ones(n_samples)
300.
301.     # predict on generator
302.     X_pred = generator.predict([X_lat, y_lat])
303.
304.     # return fraud samples
305.     return [X_pred, y_lat]
306.
307. # shuffle the dataframe df
308. df = df.sample(frac = 1.0).reset_index(drop = True)
309.
310. # get 'X' and 'y'
311. # X = np.array(df.drop(['is_fraud'], axis = 1))
312. # y = np.array(df['is_fraud'])
313.
314. # scale down [-1, 1]
315. X = MaxAbsScaler().fit_transform(X_train)
316. y=y_train
317.
318. models_dict = {}
319. def IsF(X, y, ratio):
320.     # train test split
321.     X_train, X_test, y_train, y_test = train_test_split(X, y,
322. test_size=0.3, random_state=1)
322.
323.     # define the model
324.     model = IsolationForest(n_estimators=200, contamination=ratio,
325. random_state=1)
325.
326.     # train the model
327.     model.fit(X_train, y_train)
328.
329.     # predict on the test set
330.     predictions = model.predict(X_test)
331.
332.     # Convert the predictions according to problem profile

```

```

333.     predictions[predictions == 1] = 0
334.     predictions[predictions == -1] = 1
335.
336.     # evaluate the accuracy and classification report
337.     # print('IsF : ' + str(accuracy_score(y_test, predictions)))
338.     # print(classification_report(y_test, predictions))
339.
340.     # assign this to models_dict
341.     models_dict['IsF'] = model
342. ratio = float(y_train[y_train == 1].shape[0]/y_train.shape[0])
343. # For Multi Layer Perceptron
344. param = {
345.     'nodes' : 170,
346.     'lrate' : 0.00005,
347.     'toler' : 0.00001,
348.     'batch_size' : 100
349. }
350.
351. print('Test performance')
352.
353. # get the numpy array
354. # X = np.array(train.drop(['is_fraud'], axis = 1))
355. # y = np.array(train['is_fraud'])
356. X=X_train
357. y=y_train
358.
359. # scale [-1, 1]
360. X = MaxAbsScaler().fit_transform(X)
361.
362. #X_gan_test, y_gan_test = X[:1000,:], y[:1000]
363. X_train, y_train, X_test, y_test=get_data()
364. X_gan_test,y_gan_test=X_test,y_test
365. X, y = X[1000:, :], y[1000:]
366. # get synthetic samples
367. [X_syn, y_syn] = generate_new_samples(150000)
368.
369. Syn_data=np.concatenate((np.array(X_syn),np.array(y_syn.reshape(-
    1,1))),axis=1)
370. Syn_data=pd.DataFrame(Syn_data)
371. Syn_data.to_csv('Syn_data.csv',index=None)
372.
373. # reshape into 2d array

```

```

374. y, y_syn = y.reshape(-1,1), y_syn.reshape(-1,1)
375. # concatenate all the arrays
376. X_1, X_2 = np.concatenate((X,y), axis = 1),
    np.concatenate((X_syn,y_syn), axis = 1)
377. # merge the above two arrays
378. X_3 = np.concatenate((X_1, X_2), axis = 0)
379. np.random.shuffle(X_3)
380.
381. X_gan = np.array(X_3[:, :24])
382. y_gan = np.array(X_3[:, 24])
383. ratio = float(y_gan[y_gan == 1].shape[0]/y_gan.shape[0])
384. IsF(X_gan, y_gan, ratio)
385.
386. pred_1 = models_dict['IsF'].predict(X_gan_test)
387. pred_1[pred_1 == 1] = 0
388. pred_1[pred_1 == -1] = 1
389. print('IsF : ' + str(accuracy_score(y_gan_test, pred_1)))
390. print(classification_report(y_gan_test, pred_1))

```