导入包、模块

```python
# 基础
import os
import zipfile
import numpy as np
import pandas as pd
# 画图
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import font_manager as fm
from matplotlib import cm
% matplotlib inline
plt.style.use('ggplot')
# 中文图输出
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['STZhongsong']    # 指定默认字体：解决plot不能显示中文问题
mpl.rcParams['axes.unicode_minus'] = False           # 解决保存图像是负号'-'显示为方块的问题
# 数据集归一化
from sklearn import datasets
from sklearn import preprocessing
#切割训练数据和样本数据
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold,cross_val_score
# 模型
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import mean_squared_error
from sklearn.metrics import *
# 导出决策树
import graphviz
import pydotplus
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
```

定义全局函数

```python
# 定义一个路径引用的函数
def file_path(dir_path,dir_name):
    con_path = "D:\\onedrive\\02_work\\01_ScienceResearch\\01_undergraduate_thesis\\01_data\\"
    path = os.path.join(con_path,dir_path,dir_name)
    return path
```

导入原始数据

对2010-2016年经济指标文件解压

```python
# 对2010-2016年经济指标文件解压
def unzip_file(path,zip_name):
    for file in os.listdir(path):
        file_path=os.path.join(path,file)
        if os.path.splitext(file_path)[1]==zip_name:
            fz=zipfile.ZipFile(file_path,'r')
            for zip_file in fz.namelist():
                fz.extract(zip_file,path)
unzip_file("D:\\onedrive\\02_work\\01_ScienceResearch\\01_undergraduate_thesis\\01_data",".zip")
```

读入数据

```python
# 去掉各个变量的标签
pd.read_csv(file_path("01_rawdata","ACS_10_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_10_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_11_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_11_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_12_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_12_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_13_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_13_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_14_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_14_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_15_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_15_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
pd.read_csv(file_path("01_rawdata","ACS_16_5YR_DP02_with_ann.csv"))[1:].to_csv(file_path("02_output","ACS_16_5YR_DP02_with_ann.csv"),encoding="utf-8-sig")
# 读入2010-2016年经济指标数据
ACS_10_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_10_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
ACS_11_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_11_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
ACS_12_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_12_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
ACS_13_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_13_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
ACS_14_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_14_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
```

```
ACS_15_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_15_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
ACS_16_5YR_DP02_with_ann = pd.read_csv(file_path("02_output","ACS_16_5YR_DP02_with_ann.csv"),na_values=["(X)","*****","***","**","-","+","N"])
# # 读入各个地区阿片类使用量数据
MCM_NFLIS_Data=pd.read_excel(file_path("01_rawdata","MCM_NFLIS_Data.xlsx"),sheet_name=1)
# # 读入药物具体分类数据
MCM_NFLIS_Medication=pd.read_csv(file_path("01_rawdata","class_medication.csv"))
# 读入变量标签数据
ACS_10_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_10_5YR_DP02_metadata.csv"),header=None)
ACS_11_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_11_5YR_DP02_metadata.csv"),header=None)
ACS_12_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_12_5YR_DP02_metadata.csv"),header=None)
ACS_13_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_13_5YR_DP02_metadata.csv"),header=None)
ACS_14_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_14_5YR_DP02_metadata.csv"),header=None)
ACS_15_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_15_5YR_DP02_metadata.csv"),header=None)
ACS_16_5YR_DP02_metadata = pd.read_csv(file_path("01_rawdata","ACS_16_5YR_DP02_metadata.csv"),header=None)
```

数据处理

整理ACS_ALL_5YR_DP02数据

```
##  处理无效数据
# 2010
# 删除类型异常的变量（NaN、（x））
typedata = ACS_10_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_10_5YR_DP02_DropNorm = ACS_10_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_10_5YR_DP02_DropColumn=ACS_10_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_10_5YR_DP02_DropColumn.columns[ACS_10_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_10_5YR_DP02_DropColumn[column].mean()
    ACS_10_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

```

```
# 2011
# 删除类型异常的变量（NaN、（x））
typedata = ACS_11_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_11_5YR_DP02_DropNorm = ACS_11_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_11_5YR_DP02_DropColumn=ACS_11_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_11_5YR_DP02_DropColumn.columns[ACS_11_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_11_5YR_DP02_DropColumn[column].mean()
    ACS_11_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 2012
# 删除类型异常的变量（NaN、（x））
typedata = ACS_12_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_12_5YR_DP02_DropNorm = ACS_12_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_12_5YR_DP02_DropColumn=ACS_12_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_12_5YR_DP02_DropColumn.columns[ACS_12_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_12_5YR_DP02_DropColumn[column].mean()
    ACS_12_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 2013
# 删除类型异常的变量（NaN、（x））
typedata = ACS_13_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_13_5YR_DP02_DropNorm = ACS_13_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_13_5YR_DP02_DropColumn=ACS_13_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_13_5YR_DP02_DropColumn.columns[ACS_13_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_13_5YR_DP02_DropColumn[column].mean()
    ACS_13_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 2013
# 删除类型异常的变量（NaN、（x））
typedata = ACS_14_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_14_5YR_DP02_DropNorm = ACS_14_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_14_5YR_DP02_DropColumn=ACS_14_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
```

```
for column in list(ACS_14_5YR_DP02_DropColumn.columns[ACS_14_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_14_5YR_DP02_DropColumn[column].mean()
    ACS_14_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 2015
# 删除类型异常的变量（NaN、（x））
typedata = ACS_15_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_15_5YR_DP02_DropNorm = ACS_15_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_15_5YR_DP02_DropColumn=ACS_15_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_15_5YR_DP02_DropColumn.columns[ACS_15_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_15_5YR_DP02_DropColumn[column].mean()
    ACS_15_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 2016
# 删除类型异常的变量（NaN、（x））
typedata = ACS_16_5YR_DP02_with_ann.dtypes.reset_index()
nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"][2:].tolist()
ACS_16_5YR_DP02_DropNorm = ACS_16_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
# 删除全为空的变量（列）
ACS_16_5YR_DP02_DropColumn=ACS_16_5YR_DP02_DropNorm.dropna(axis=1,how="all")
# 用列均值填补缺失数据
for column in list(ACS_16_5YR_DP02_DropColumn.columns[ACS_16_5YR_DP02_DropColumn.isnull().sum() > 0]):
    mean_val = ACS_16_5YR_DP02_DropColumn[column].mean()
    ACS_16_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)

# 纵向合并2010-2016年的数据到一个数据框中、# 删除第一行数据（变量标签）
ACS_ALL_5YR_DP02=pd.concat([ACS_10_5YR_DP02_DropColumn,
                            ACS_11_5YR_DP02_DropColumn,
                            ACS_12_5YR_DP02_DropColumn,
                            ACS_13_5YR_DP02_DropColumn,
                            ACS_14_5YR_DP02_DropColumn,
                            ACS_15_5YR_DP02_DropColumn,
                            ACS_16_5YR_DP02_DropColumn],axis=0,join="outer",keys=[2010,2011,2012,2013,2014,2015,2016]).reset_index().convert_objects(convert_numeric=True)
# 用列均值填补缺失数据(合并各年份数据之后)
for column in list(ACS_ALL_5YR_DP02.columns[ACS_ALL_5YR_DP02.isnull().sum() > 0]):
    mean_val = ACS_ALL_5YR_DP02[column].mean()
    ACS_ALL_5YR_DP02[column].fillna(mean_val, inplace=True)
# 删除无效的变量(索引，中间产生变量、地理位置),重命名年份变量
ACS_ALL_5YR_DP02_Clear = ACS_ALL_5YR_DP02.ix[:,:-2].drop(["GEO.display-label","level_1","GEO.id"],axis=1).rename(columns={"level_0":"YYYY"})
```

整理MCM_NFLIS_Data数据

```
# 对阿片类药物使用情况数据键值重命名
MCM_NFLIS_Data_Rename=MCM_NFLIS_Data.rename(columns={"FIPS_Combined":"GEO.id2"})
# 删除2017相关数据
MCM_NFLIS_Data_Drop17=MCM_NFLIS_Data_Rename.loc[MCM_NFLIS_Data_Rename["YYYY"] != 2017]
# 匹配药物分类数据
MCM_NFLIS_Class=pd.merge(MCM_NFLIS_Data_Drop17,MCM_NFLIS_Medication,how="left",on=["SubstanceName","YYYY"])
# 删除一些无效变量
MCM_NFLIS_Class_Clear_Drop=MCM_NFLIS_Class.drop(["FIPS_State","FIPS_County","SubstanceName","code"],axis=1)
# 按照中文名药物分类求和
MCM_NFLIS_Class_Clear = MCM_NFLIS_Class_Clear_Drop.groupby(["YYYY","GEO.id2","State","COUNTY","SubstanceClass",
                                                            "SubstanceName_c"])["DrugReports"].sum().reset_index()
```

整理ACS_All_5YR_DP02_metadata数据

```
# 纵向合并2010-2016年的数据到一个数据框中、# 删除第一行数据（变量标签）
ACS_All_5YR_DP02_metadata=pd.concat([ACS_10_5YR_DP02_metadata,
                                     ACS_11_5YR_DP02_metadata,
                                     ACS_12_5YR_DP02_metadata,
                                     ACS_13_5YR_DP02_metadata,
                                     ACS_14_5YR_DP02_metadata,
                                     ACS_15_5YR_DP02_metadata,
                                     ACS_16_5YR_DP02_metadata],axis=0,join="outer",)
# 删除重复值
ACS_All_5YR_DP02_metadata_Dup = ACS_All_5YR_DP02_metadata.drop_duplicates(list(ACS_All_5YR_DP02_metadata.columns)[0],keep="first")
ACS_All_5YR_DP02_metadata_Dup.columns = ["Var","Var_label"]
```

匹配阿片类药物使用情况

```
# 合并阿片类使用情况与相关经济指标
NFLIS_and_ACS_ALL=pd.merge(ACS_ALL_5YR_DP02_Clear,MCM_NFLIS_Class_Clear,how="right",on=["YYYY","GEO.id2"])
```

按照三类药物数据透视

```
# 分类计数
NFLIS_and_ACS_ALL_ClassSum = NFLIS_and_ACS_ALL.groupby(["GEO.id2","State","COUNTY",
 "SubstanceClass","YYYY"])["DrugReports"].sum().reset_index()
# 数据透视表
NFLIS_and_ACS_ALL_Pivot = pd.pivot_table(data=NFLIS_and_ACS_ALL_ClassSum,
                                         index=["GEO.id2","State","SubstanceClass","COUNTY"],
```

```
                                         columns=["YYYY"],values=["DrugReports"])
# 缺失值填补、转置
NFLIS_and_ACS_ALL_Clear = NFLIS_and_ACS_ALL_Pivot[2:].fillna(0).stack().reset_index()
 # 合并
NFLIS_and_ACS_ALL_Out = pd.merge(NFLIS_and_ACS_ALL_Clear,
                                 ACS_ALL_5YR_DP02_Clear,
                                 on=["GEO.id2","YYYY"],how="left")
# 根据药物量分层
NFLIS_and_ACS_ALL_Out["DrugReportsclass"] = np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 5000,"7、5000人以上",
        np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 1000,"6、1000-4999人",
         np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 500,"5、500-999人",
        np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 100,"4、100-499人",
         np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 10,"3、10-99人",
          np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 1,"2、1-9人","1、0人"))))))
```

统计描述

图1：所有类阿片类药物构成饼图

整理数据为直接可用

```
# 提取画图数据"YYYY","SubstanceName","DrugReports","State"
NFLIS_Figure1_Data = MCM_NFLIS_Class_Clear.groupby(["SubstanceName_c"])["DrugReports"].sum().reset_index().sort_values(by="DrugReports",ascending=True)
# 添加列：每种药物的百分占比
NFLIS_Figure1_Data["Percent"] = NFLIS_Figure1_Data["DrugReports"]/(NFLIS_Figure1_Data["DrugReports"].sum())
NFLIS_Figure1_Data["Label"] = NFLIS_Figure1_Data["SubstanceName_c"] +\
                                ' 	' + \
                                NFLIS_Figure1_Data["Percent"].apply(lambda x: format(x, '.2%'))
# 画图所用的数据
Figure1_labels = NFLIS_Figure1_Data["Label"]
Figure1_sizes = NFLIS_Figure1_Data["DrugReports"]
```

饼图

```
# 设置画布和子图
Figure1,axes = plt.subplots(figsize=(20,15),ncols=2)
Figure1_ax1,Figure1_ax2 = axes.ravel()
# 设置参数：颜色盘-colormap；间隙-与labels一一对应，数值越大离中心区越远
explode = [x * 0.00325 for x in range(len(NFLIS_Figure1_Data))]
colors=cm.rainbow(np.arange(len(Figure1_sizes))/len(Figure1_sizes))
# 画饼图：类别太多取消标签labels；每个类别离中心的距离；
```

```
patches,texts =
Figure1_ax1.pie(Figure1_sizes,labels=None,shadow=False,explode=explode,start
angle=0,colors=colors)
# 子图：ax1-饼图、ax2-图例
Figure1_ax1.axis('equal')
Figure1_ax2.axis('off')
Figure1_ax2.legend(patches,Figure1_labels,loc="center left",fontsize="xx-
large")
# 调整大小、读取图片
plt.tight_layout()
Figure1 = plt.gcf()
```

图2：所有类阿片类药物数量条图

整理数据为直接可用

```
# 提取画图数据"YYYY","SubstanceName","DrugReports","State"；排序；
NFLIS_Figure2_Data = MCM_NFLIS_Class_Clear.groupby(["SubstanceName_c"])
["DrugReports"].sum().reset_index().sort_values(by="DrugReports",ascending=Tr
ue)

```

条图

```
# 设置画布
plt.figure(figsize=(16,10))
# 设置参数：颜色盘-colormap
color=cm.rainbow(np.arange(len(NFLIS_Figure2_Data))/len(NFLIS_Figure2_Data))
# 从高到低排列，改变y轴刻度的排列顺序
plt.yticks(np.arange(len(NFLIS_Figure2_Data['SubstanceName_c'])),
NFLIS_Figure2_Data['SubstanceName_c'])
# 水平条图
plt.barh(np.arange(len(NFLIS_Figure2_Data['SubstanceName_c'])),
NFLIS_Figure2_Data['DrugReports'], color=color)
# 坐标轴标签
plt.ylabel("阿片类药物名")
plt.xlabel("报告量")
# 格式整理导出
plt.tight_layout()
Figure2 = plt.gcf()
```

图3：五个州阿片类药物数量热力图

整理数据为直接可用

```
# 提取画图数据"YYYY","SubstanceName","DrugReports","State"
NFLIS_Figure3_Clear1 =
MCM_NFLIS_Class_Clear.groupby(["State","YYYY","SubstanceName_c"])
["DrugReports"].sum().reset_index()
# 提取各个州的数据
NFLIS_Figure3_KY = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
== "KY")]
NFLIS_Figure3_OH = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
== "OH")]
```

```python
NFLIS_Figure3_PA = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"] == "PA")]
NFLIS_Figure3_VA = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"] == "VA")]
NFLIS_Figure3_WV = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"] == "WV")]
# 匹配每种药物（解决某年可能没有某种药）
NFLIS_Figure3_KY_merge = pd.merge(NFLIS_Figure3_KY,MCM_NFLIS_Medication,how = "right",on = ["YYYY","SubstanceName_c"])
NFLIS_Figure3_OH_merge = pd.merge(NFLIS_Figure3_OH,MCM_NFLIS_Medication,how = "right",on = ["YYYY","SubstanceName_c"])
NFLIS_Figure3_PA_merge = pd.merge(NFLIS_Figure3_PA,MCM_NFLIS_Medication,how = "right",on = ["YYYY","SubstanceName_c"])
NFLIS_Figure3_VA_merge = pd.merge(NFLIS_Figure3_VA,MCM_NFLIS_Medication,how = "right",on = ["YYYY","SubstanceName_c"])
NFLIS_Figure3_WV_merge = pd.merge(NFLIS_Figure3_WV,MCM_NFLIS_Medication,how = "right",on = ["YYYY","SubstanceName_c"])
# 将数据转置为dataframe矩阵
NFLIS_Figure3_pivot_KY = NFLIS_Figure3_KY_merge.pivot_table(index = "SubstanceName_c",columns = "YYYY",values = "DrugReports")
NFLIS_Figure3_pivot_OH = NFLIS_Figure3_OH_merge.pivot_table(index = "SubstanceName_c",columns = "YYYY",values = "DrugReports")
NFLIS_Figure3_pivot_PA = NFLIS_Figure3_PA_merge.pivot_table(index = "SubstanceName_c",columns = "YYYY",values = "DrugReports")
NFLIS_Figure3_pivot_VA = NFLIS_Figure3_VA_merge.pivot_table(index = "SubstanceName_c",columns = "YYYY",values = "DrugReports")
NFLIS_Figure3_pivot_WV = NFLIS_Figure3_WV_merge.pivot_table(index = "SubstanceName_c",columns = "YYYY",values = "DrugReports")
```

热力图

```python
# 设置画布大小
f,(Figure3_ax1,Figure3_ax2,Figure3_ax3,Figure3_ax4,Figure3_ax5) = plt.subplots(ncols=5,figsize=(30,10))
# 设置连续调色板cubehelix_palette,as_camp传入matplotlib
cmap=sns.cubehelix_palette(start=1,rot=3,gamma=0.8,as_cmap=True)
# KY州
sns.heatmap(NFLIS_Figure3_pivot_KY,cmap=cmap,linewidths=0.05,ax=Figure3_ax1,cbar=False)
Figure3_ax1.set_title("肯塔基州",fontsize=30)
Figure3_ax1.set_xlabel('')
Figure3_ax1.set_ylabel('阿片类药物名',fontsize=35)
# OH州
sns.heatmap(NFLIS_Figure3_pivot_OH,cmap=cmap,linewidths=0.05,ax=Figure3_ax2,cbar=False)
Figure3_ax2.set_title("俄亥俄州",fontsize=30)
Figure3_ax2.set_xlabel('')
Figure3_ax2.set_ylabel(' ')
Figure3_ax2.set_yticklabels([])
# PA州
sns.heatmap(NFLIS_Figure3_pivot_PA,cmap=cmap,linewidths=0.05,ax=Figure3_ax3,cbar=False)
Figure3_ax3.set_title("宾夕法尼亚州",fontsize=30)
Figure3_ax3.set_xlabel('年份',fontsize=35)
Figure3_ax3.set_ylabel('')
Figure3_ax3.set_yticklabels([])
# VA州
```

```python
sns.heatmap(NFLIS_Figure3_pivot_VA,cmap=cmap,linewidths=0.05,ax=Figure3_ax4,cbar=False)
Figure3_ax4.set_title("弗吉尼亚州",fontsize=30)
Figure3_ax4.set_xlabel('')
Figure3_ax4.set_ylabel('')
Figure3_ax4.set_yticklabels([])
# WV州
sns.heatmap(NFLIS_Figure3_pivot_WV,cmap=cmap,linewidths=0.05,ax=Figure3_ax5,cbar=True)
Figure3_ax5.set_title("西弗吉尼亚州",fontsize=30)
Figure3_ax5.set_xlabel('')
Figure3_ax5.set_ylabel('')
Figure3_ax5.set_yticklabels([])

plt.tight_layout()
Figure3 = plt.gcf()
```

图4：五个州三类阿片药物量折线图

整理数据为直接可用

```python
# 五个州的总量情况分组
NFLIS_Fugure3_Clear1 = MCM_NFLIS_Class_Clear.groupby(["YYYY","SubstanceClass"])["DrugReports"].sum().reset_index()
NFLIS_Fugure3_Class1_all = NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "半合成阿片类药物")]
NFLIS_Fugure3_Class2_all = NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "合成阿片类药物")]
NFLIS_Fugure3_Class3_all = NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "非合成阿片类药物")]
# 五个州的分别情况分组
NFLIS_Fugure3_Class = MCM_NFLIS_Class_Clear.groupby(["YYYY","State","SubstanceClass"])["DrugReports"].sum().reset_index()
NFLIS_Fugure3_Class1 = NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "半合成阿片类药物")]
NFLIS_Fugure3_Class2 = NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "合成阿片类药物")]
NFLIS_Fugure3_Class3 = NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "非合成阿片类药物")]
# 对每个州进行汇合
NFLIS_Figure2_Data_Class1 = NFLIS_Fugure3_Class1.pivot_table(index="YYYY",columns="State",values="DrugReports").reset_index()
NFLIS_Figure2_Data_Class2 = NFLIS_Fugure3_Class2.pivot_table(index="YYYY",columns="State",values="DrugReports").reset_index()
NFLIS_Figure2_Data_Class3 = NFLIS_Fugure3_Class3.pivot_table(index="YYYY",columns="State",values="DrugReports").reset_index()
```

折线图

```
# 创建画布、6个子图
plt.figure(figsize=(15,10))
f4 = plt.figure(figsize=(20,15))
Figure_ax1 = f4.add_subplot(2, 3, 1)
Figure_ax2 = f4.add_subplot(2, 3, 2)
Figure_ax3 = f4.add_subplot(2, 3, 3)
Figure_ax4 = f4.add_subplot(2, 3, 4)
Figure_ax5 = f4.add_subplot(2, 3, 5)
Figure_ax6 = f4.add_subplot(2, 3, 6)

# KY州不同类型药物的折线图
Figure_ax1.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["KY"],label="半合成阿片类药物",linewidth=2)
Figure_ax1.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["KY"],label="合成阿片类药物",linewidth=2)
Figure_ax1.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["KY"],label="非合成阿片类药物",linewidth=2)
Figure_ax1.set_title("肯塔基州")
Figure_ax1.legend(loc=2)
Figure_ax1.grid(axis='x')
 #设置数字标签
for a,b in zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["KY"]):
    Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va= 'bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["KY"]):
    Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va= 'bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["KY"]):
    Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va= 'bottom',fontsize=11)

# OH州不同类型药物的折线图
Figure_ax2.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["OH"],label="半合成阿片类药物",linewidth=2)
Figure_ax2.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["OH"],label="合成阿片类药物",linewidth=2)
Figure_ax2.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["OH"],label="非合成阿片类药物",linewidth=2)
Figure_ax2.set_title("俄亥俄州")
Figure_ax2.legend(loc=2)
Figure_ax2.grid(axis='x')
 #设置数字标签**
for a,b in zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["OH"]):
    Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va= 'bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["OH"]):
    Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va= 'bottom',fontsize=11)
```

```
for a,b in zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["OH"]):
    Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)

# PA州不同类型药物的折线图
Figure_ax3.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["PA"],label="半合成阿片类药物",linewidth=2)
Figure_ax3.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["PA"],label="合成阿片类药物",linewidth=2)
Figure_ax3.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["PA"],label="非合成阿片类药物",linewidth=2)
Figure_ax3.set_title("宾夕法尼亚州")
Figure_ax3.legend(loc=2)
Figure_ax3.grid(axis='x')
 #设置数字标签**
for a,b in zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["PA"]):
    Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["PA"]):
    Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["PA"]):
    Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)

# VA州不同类型药物的折线图
Figure_ax4.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["VA"],label="半合成阿片类药物",linewidth=2)
Figure_ax4.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["VA"],label="合成阿片类药物",linewidth=2)
Figure_ax4.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["VA"],label="非合成阿片类药物",linewidth=2)
Figure_ax4.set_title("弗吉尼亚州")
Figure_ax4.grid(axis="x")
Figure_ax4.legend(loc=2)
for a,b in zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["VA"]):
    Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["VA"]):
    Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)
for a,b in zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["VA"]):
    Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va='bottom',fontsize=11)

# WV州不同类型药物的折线图
Figure_ax5.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["WV"],label="半合成阿片类药物",linewidth=2)
Figure_ax5.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["WV"],label="合成阿片类药物",linewidth=2)
```

```
Figure_ax5.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
["WV"],label="非合成阿片类药物",linewidth=2)
Figure_ax5.set_title("西弗吉尼亚州")
Figure_ax5.legend(loc=2)
Figure_ax5.grid(axis='x')
 #设置数字标签**
for a,b in
zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["WV"]):
    Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
for a,b in
zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["WV"]):
    Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
for a,b in
zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["WV"]):
    Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)

# 5个州总的不同类型药物的折线图
Figure_ax6.plot(NFLIS_Fugure3_Class1_all["YYYY"],NFLIS_Fugure3_Class1_all["
DrugReports"],label="半合成阿片类药物",linewidth=2)
Figure_ax6.plot(NFLIS_Fugure3_Class2_all["YYYY"],NFLIS_Fugure3_Class2_all["
DrugReports"],label="合成阿片类药物",linewidth=2)
Figure_ax6.plot(NFLIS_Fugure3_Class3_all["YYYY"],NFLIS_Fugure3_Class3_all["
DrugReports"],label="非合成阿片类药物",linewidth=2)
Figure_ax6.set_title("总量")
Figure_ax6.legend(loc=2)
Figure_ax6.grid(axis='x')
for a,b in
zip(NFLIS_Fugure3_Class1_all["YYYY"],NFLIS_Fugure3_Class1_all["DrugReports"
]):
    Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
for a,b in
zip(NFLIS_Fugure3_Class2_all["YYYY"],NFLIS_Fugure3_Class2_all["DrugReports"
]):
    Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
for a,b in
zip(NFLIS_Fugure3_Class3_all["YYYY"],NFLIS_Fugure3_Class3_all["DrugReports"
]):
    Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)

plt.tight_layout()
Figure4 = plt.gcf()
```

变量选择

相关系数计算

计算各个年份相关系数

```
# 计算2010年相关系数
df_corr_2010 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2010)].corr().rese
t_index()
df_corr_ext_2010 =
df_corr_2010.loc[(df_corr_2010["index"].str.contains("HC"))]
df_corr_ext_2010_part =
df_corr_ext_2010[["index","DrugReports"]].rename(columns=
{"DrugReports":"2010年相关系数","index":"变量名"})
# 计算2011年相关系数
df_corr_2011 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2011)].corr().rese
t_index()
df_corr_ext_2011 =
df_corr_2011.loc[(df_corr_2011["index"].str.contains("HC"))]
df_corr_ext_2011_part =
df_corr_ext_2011[["index","DrugReports"]].rename(columns=
{"DrugReports":"2011年相关系数","index":"变量名"})
# 计算2012年相关系数
df_corr_2012 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2012)].corr().rese
t_index()
df_corr_ext_2012 =
df_corr_2012.loc[(df_corr_2012["index"].str.contains("HC"))]
df_corr_ext_2012_part =
df_corr_ext_2012[["index","DrugReports"]].rename(columns=
{"DrugReports":"2012年相关系数","index":"变量名"})
# 计算2013年相关系数
df_corr_2013 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2013)].corr().rese
t_index()
df_corr_ext_2013 =
df_corr_2013.loc[(df_corr_2013["index"].str.contains("HC"))]
df_corr_ext_2013_part =
df_corr_ext_2013[["index","DrugReports"]].rename(columns=
{"DrugReports":"2013年相关系数","index":"变量名"})
# 计算2014年相关系数
df_corr_2014 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2014)].corr().rese
t_index()
df_corr_ext_2014 =
df_corr_2014.loc[(df_corr_2014["index"].str.contains("HC"))]
df_corr_ext_2014_part =
df_corr_ext_2014[["index","DrugReports"]].rename(columns=
{"DrugReports":"2014年相关系数","index":"变量名"})
# 计算2015年相关系数
df_corr_2015 =
NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2015)].corr().rese
t_index()
df_corr_ext_2015 =
df_corr_2015.loc[(df_corr_2015["index"].str.contains("HC"))]
df_corr_ext_2015_part =
df_corr_ext_2015[["index","DrugReports"]].rename(columns=
{"DrugReports":"2015年相关系数","index":"变量名"})
# 计算2016年相关系数
```

```
df_corr_2016 = NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2016)].corr().reset_index()
df_corr_ext_2016 = df_corr_2016.loc[(df_corr_2016["index"].str.contains("HC"))]
df_corr_ext_2016_part = df_corr_ext_2016[["index","DrugReports"]].rename(columns={"DrugReports":"2016年相关系数","index":"变量名"})
# 计算全部数据的相关系数
df_corr_all = NFLIS_and_ACS_ALL_Out.corr().reset_index()
df_corr_ext_all = df_corr_all.loc[(df_corr_all["index"].str.contains("HC"))]
df_corr_ext_all_part = df_corr_ext_all[["index","DrugReports"]].rename(columns={"DrugReports":"合计相关系数","index":"变量名"})
```

合并各个年份的相关系数

```
# 合并各个年份的相关系数
df_corr_merge_10_11 = pd.merge(df_corr_ext_2010_part,df_corr_ext_2011_part,on="变量名",how="outer")
df_corr_merge_11_12 = pd.merge(df_corr_merge_10_11,df_corr_ext_2012_part,on="变量名",how="outer")
df_corr_merge_12_13 = pd.merge(df_corr_merge_11_12,df_corr_ext_2013_part,on="变量名",how="outer")
df_corr_merge_13_14 = pd.merge(df_corr_merge_12_13,df_corr_ext_2014_part,on="变量名",how="outer")
df_corr_merge_14_15 = pd.merge(df_corr_merge_13_14,df_corr_ext_2015_part,on="变量名",how="outer")
df_corr_merge_15_16 = pd.merge(df_corr_merge_14_15,df_corr_ext_2016_part,on="变量名",how="outer")
df_corr_merge_all = pd.merge(df_corr_merge_15_16,df_corr_ext_all_part,on="变量名",how="outer")
# 计算平均数
df_corr_merge_all["均值"] = df_corr_merge_all[["2010年相关系数","2011年相关系数","2012年相关系数",
                                               "2013年相关系数","2014年相关系数","2015年相关系数","2016年相关系数"]].mean(axis=1)
# 排序：倒序
All_Corr = df_corr_merge_all.sort_values(by=["均值"],ascending=False).round(4)
```

选择相关系数大于0.5的变量

```
All_Corr_Condi = All_Corr[(abs(All_Corr["2010年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2011年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2012年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2013年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2014年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2015年相关系数"]) >= 0.5)
                          & (abs(All_Corr["2016年相关系数"]) >= 0.5)
```

```
                                            & (abs(All_Corr["合计相关系数"]) >= 0.5)
                                            & (abs(All_Corr["均值"]) >= 0.5)]
connames = []
for conval in NFLIS_and_ACS_ALL_Out.columns.tolist():
    if "HC" not in conval:
        connames.append(conval)
NFLIS_and_ACS_All_Corr_Condi = NFLIS_and_ACS_ALL_Out.ix[:,list(NFLIS_and_ACS_ALL_Out[connames])+list(All_Corr_Condi["变量名"])].dropna()
```

统计推断

归一化

```
data=NFLIS_and_ACS_All_Corr_Condi.ix[:,list(All_Corr_Condi["变量名"])]
NFLIS_and_ACS_All_Condi_Normal_CH = (data - data.mean())/data.std()
# 合并
NFLIS_and_ACS_All_Condi_Normal = pd.concat([NFLIS_and_ACS_All_Corr_Condi.ix[:,list(NFLIS_and_ACS_All_Corr_Condi[connames])],

 NFLIS_and_ACS_All_Condi_Normal_CH],axis=1)
```

训练集与测试集

```
Complex = NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "合成阿片类药物"]
Non_Complex = NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "非合成阿片类药物"]
Semi_Complex = NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "半合成阿片类药物"]

Complex_x_train,Complex_x_test,Complex_y_train,Complex_y_test = train_test_split(Complex.ix[:,list(All_Corr_Condi["变量名"])],

    Complex.ix[:,"DrugReportsclass"],

    test_size=0.3,

    random_state=1234 )
Non_Complex_x_train,Non_Complex_x_test,Non_Complex_y_train,Non_Complex_y_test = train_test_split(Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],

                    Non_Complex.ix[:,"DrugReportsclass"],

                    test_size=0.3,

                    random_state=1234 )
Semi_Complex_x_train,Semi_Complex_x_test,Semi_Complex_y_train,Semi_Complex_y_test = train_test_split(Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
```

```
                         Semi_Complex.ix[:,"DrugReportsclass"],
                         test_size=0.3,
                         random_state=1234 )
```

KNN

```
Complex_KNN = KNeighborsClassifier()
Complex_KNN.fit(Complex_x_train,Complex_y_train)
Complex_KNN_Y_Predict = Complex_KNN.predict(Complex_x_test)
Complex_KNN_train_score = Complex_KNN.score(Complex_x_train,
Complex_y_train)
Complex_KNN_test_score = Complex_KNN.score(Complex_x_test, Complex_y_test)
Non_Complex_KNN = KNeighborsClassifier()
Non_Complex_KNN.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_KNN_Y_Predict = Non_Complex_KNN.predict(Non_Complex_x_test)
Non_Complex_KNN_train_score = Non_Complex_KNN.score(Non_Complex_x_train,
Non_Complex_y_train)
Non_Complex_KNN_test_score = Complex_KNN.score(Non_Complex_x_test,
Non_Complex_y_test)
Semi_Complex_KNN = KNeighborsClassifier()
Semi_Complex_KNN.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_KNN_Y_Predict = Semi_Complex_KNN.predict(Semi_Complex_x_test)
Semi_Complex_KNN_train_score = Semi_Complex_KNN.score(Semi_Complex_x_train,
Semi_Complex_y_train)
Semi_Complex_KNN_test_score = Semi_Complex_KNN.score(Semi_Complex_x_test,
Semi_Complex_y_test)
```

决策树

```
# 决策树
Complex_Decision = DecisionTreeClassifier()
Complex_Decision.fit(Complex_x_train,Complex_y_train)
Complex_Decision_Y_Predict = Complex_Decision.predict(Complex_x_test)
Complex_Decision_train_score = Complex_Decision.score(Complex_x_train,
Complex_y_train)
Complex_Decision_test_score = Complex_Decision.score(Complex_x_test,
Complex_y_test)
Non_Complex_Decision = DecisionTreeClassifier()
Non_Complex_Decision.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_Decision_Y_Predict =
Non_Complex_Decision.predict(Non_Complex_x_test)
Non_Complex_Decision_train_score =
Non_Complex_Decision.score(Non_Complex_x_train, Non_Complex_y_train)
Non_Complex_Decision_test_score = Complex_Decision.score(Non_Complex_x_test,
Non_Complex_y_test)
Semi_Complex_Decision = DecisionTreeClassifier()
Semi_Complex_Decision.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_Decision_Y_Predict =
Semi_Complex_Decision.predict(Semi_Complex_x_test)
Semi_Complex_Decision_train_score =
Semi_Complex_Decision.score(Semi_Complex_x_train, Semi_Complex_y_train)
```

```
Semi_Complex_Decision_test_score = Semi_Complex_Decision.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

随机森林

```
# 随机森林
Complex_RFC = RandomForestClassifier()
Complex_RFC.fit(Complex_x_train,Complex_y_train)
Complex_RFC_Y_Predict = Complex_RFC.predict(Complex_x_test)
Complex_RFC_train_score = Complex_RFC.score(Complex_x_train, Complex_y_train)
Complex_RFC_test_score = Complex_RFC.score(Complex_x_test, Complex_y_test)
Non_Complex_RFC = RandomForestClassifier()
Non_Complex_RFC.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_RFC_Y_Predict = Non_Complex_RFC.predict(Non_Complex_x_test)
Non_Complex_RFC_train_score = Non_Complex_RFC.score(Non_Complex_x_train, Non_Complex_y_train)
Non_Complex_RFC_test_score = Complex_RFC.score(Non_Complex_x_test, Non_Complex_y_test)
Semi_Complex_RFC = RandomForestClassifier()
Semi_Complex_RFC.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_RFC_Y_Predict = Semi_Complex_RFC.predict(Semi_Complex_x_test)
Semi_Complex_RFC_train_score = Semi_Complex_RFC.score(Semi_Complex_x_train, Semi_Complex_y_train)
Semi_Complex_RFC_test_score = Semi_Complex_RFC.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

支持向量机

```
# SVM
Complex_SVM = SVC()
Complex_SVM.fit(Complex_x_train,Complex_y_train)
Complex_SVM_Y_Predict = Complex_SVM.predict(Complex_x_test)
Complex_SVM_train_score = Complex_SVM.score(Complex_x_train, Complex_y_train)
Complex_SVM_test_score = Complex_SVM.score(Complex_x_test, Complex_y_test)
Non_Complex_SVM = SVC()
Non_Complex_SVM.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_SVM_Y_Predict = Non_Complex_SVM.predict(Non_Complex_x_test)
Non_Complex_SVM_train_score = Non_Complex_SVM.score(Non_Complex_x_train, Non_Complex_y_train)
Non_Complex_SVM_test_score = Complex_SVM.score(Non_Complex_x_test, Non_Complex_y_test)
Semi_Complex_SVM = SVC()
Semi_Complex_SVM.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_SVM_Y_Predict = Semi_Complex_SVM.predict(Semi_Complex_x_test)
Semi_Complex_SVM_train_score = Semi_Complex_SVM.score(Semi_Complex_x_train, Semi_Complex_y_train)
Semi_Complex_SVM_test_score = Semi_Complex_SVM.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

神经网络

```python
# 神经网络
Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
Complex_MLP.fit(Complex_x_train,Complex_y_train)
Complex_MLP_Y_Predict = Complex_MLP.predict(Complex_x_test)
Complex_MLP_train_score = Complex_MLP.score(Complex_x_train, Complex_y_train)
Complex_MLP_test_score = Complex_MLP.score(Complex_x_test, Complex_y_test)
Non_Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
Non_Complex_MLP.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_MLP_Y_Predict = Non_Complex_MLP.predict(Non_Complex_x_test)
Non_Complex_MLP_train_score = Non_Complex_MLP.score(Non_Complex_x_train, Non_Complex_y_train)
Non_Complex_MLP_test_score = Complex_MLP.score(Non_Complex_x_test, Non_Complex_y_test)
Semi_Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
Semi_Complex_MLP.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_MLP_Y_Predict = Semi_Complex_MLP.predict(Semi_Complex_x_test)
Semi_Complex_MLP_train_score = Semi_Complex_MLP.score(Semi_Complex_x_train, Semi_Complex_y_train)
Semi_Complex_MLP_test_score = Semi_Complex_MLP.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

线性回归

```python
# 线性回归
Complex_LR = LogisticRegression()
Complex_LR.fit(Complex_x_train,Complex_y_train)
Complex_LR_Y_Predict = Complex_LR.predict(Complex_x_test)
Complex_LR_train_score = Complex_LR.score(Complex_x_train, Complex_y_train)
Complex_LR_test_score = Complex_LR.score(Complex_x_test, Complex_y_test)
Non_Complex_LR = LogisticRegression()
Non_Complex_LR.fit(Non_Complex_x_train,Non_Complex_y_train)
Non_Complex_LR_Y_Predict = Non_Complex_LR.predict(Non_Complex_x_test)
Non_Complex_LR_train_score = Non_Complex_LR.score(Non_Complex_x_train, Non_Complex_y_train)
Non_Complex_LR_test_score = Complex_LR.score(Non_Complex_x_test, Non_Complex_y_test)
Semi_Complex_LR = LogisticRegression()
Semi_Complex_LR.fit(Semi_Complex_x_train,Semi_Complex_y_train)
Semi_Complex_LR_Y_Predict = Semi_Complex_LR.predict(Semi_Complex_x_test)
Semi_Complex_LR_train_score = Semi_Complex_LR.score(Semi_Complex_x_train, Semi_Complex_y_train)
Semi_Complex_LR_test_score = Semi_Complex_LR.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

模型评估

特征重要性

特征重要性计算

```
# 非合成类
Non_Complex_Imp = 100.0*(Non_Complex_RFC.feature_importances_/
                              max(Non_Complex_RFC.feature_importances_))
Non_Complex_Importance = pd.DataFrame(np.array([Non_Complex_x_test.columns,Non_Complex_Imp]).T,
                                      columns=["Var","非合成类重要度"])
Non_Complex_Importance["非合成类重要度"].astype("float")
Non_Complex_Importance_Sort= Non_Complex_Importance.sort_values(by="非合成类重要度",ascending=False)
# 合成类
Complex_Imp = 100.0*(Complex_RFC.feature_importances_/
                              max(Complex_RFC.feature_importances_))
Complex_Importance = pd.DataFrame(np.array([Complex_x_test.columns,Complex_Imp]).T,
                                  columns=["Var","合成类重要度"])
Complex_Importance["合成类重要度"].astype("float")
Complex_Importance_Sort= Complex_Importance.sort_values(by="合成类重要度",ascending=False)
# 半合成类
Semi_Complex_Imp = 100.0*(Semi_Complex_RFC.feature_importances_/
                              max(Semi_Complex_RFC.feature_importances_))
Semi_Complex_Importance = pd.DataFrame(np.array([Semi_Complex_x_test.columns,Semi_Complex_Imp]).T,
                                           columns=["Var","半合成类重要度"])
Semi_Complex_Importance["半合成类重要度"].astype("float")
Semi_Complex_Importance_Sort= Semi_Complex_Importance.sort_values(by="半合成类重要度",ascending=False)
```

变量名匹配

```
Complex_Importance_Rename = pd.merge(Complex_Importance_Sort,
                                     ACS_All_5YR_DP02_metadata_Dup,
                                     on="Var",how="left")
Non_Complex_Importance_Rename = pd.merge(Non_Complex_Importance_Sort,
                                     ACS_All_5YR_DP02_metadata_Dup,
                                     on="Var",how="left")
Semi_Complex_Importance_Rename = pd.merge(Semi_Complex_Importance_Sort,
                                     ACS_All_5YR_DP02_metadata_Dup,
                                     on="Var",how="left")
All_Importance_Rename = pd.concat([Complex_Importance_Rename,
                                 Non_Complex_Importance_Rename,
                                 Semi_Complex_Importance_Rename],
                               axis=1,join="outer")
```

KFold验证

非合成类

```
strKFold = StratifiedKFold(n_splits=10,shuffle=False,random_state=1234)
Non_Complex_KNN_Kfold = cross_val_score(Non_Complex_KNN,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Non_Complex_Decision_Kfold = cross_val_score(Non_Complex_Decision,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Non_Complex_RFC_Kfold = cross_val_score(Non_Complex_RFC,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Non_Complex_SVM_Kfold = cross_val_score(Non_Complex_SVM,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Non_Complex_MLP_Kfold = cross_val_score(Non_Complex_MLP,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Non_Complex_LR_Kfold = cross_val_score(Non_Complex_LR,
                            Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Non_Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
```

合成类

```
strKFold = StratifiedKFold(n_splits=10,shuffle=False,random_state=1234)
Complex_KNN_Kfold = cross_val_score(Complex_KNN,
                            Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Complex_Decision_Kfold = cross_val_score(Complex_Decision,
                            Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Complex_RFC_Kfold = cross_val_score(Complex_RFC,
                            Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
                            cv=strKFold)
Complex_SVM_Kfold = cross_val_score(Complex_SVM,
                            Complex.ix[:,list(All_Corr_Condi["变量名"])],
                            Complex.ix[:,"DrugReportsclass"],
                            scoring='accuracy',
```

```
                             cv=strKFold)
Complex_MLP_Kfold = cross_val_score(Complex_MLP,
                             Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Complex_LR_Kfold = cross_val_score(Complex_LR,
                             Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
```

半合成类

```
strKFold = StratifiedKFold(n_splits=10,shuffle=False,random_state=1234)
Semi_Complex_KNN_Kfold = cross_val_score(Semi_Complex_KNN,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Semi_Complex_Decision_Kfold = cross_val_score(Semi_Complex_Decision,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Semi_Complex_RFC_Kfold = cross_val_score(Semi_Complex_RFC,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Semi_Complex_SVM_Kfold = cross_val_score(Semi_Complex_SVM,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Semi_Complex_MLP_Kfold = cross_val_score(Semi_Complex_MLP,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
Semi_Complex_LR_Kfold = cross_val_score(Semi_Complex_LR,
                             Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],
                             Semi_Complex.ix[:,"DrugReportsclass"],
                             scoring='accuracy',
                             cv=strKFold)
```

Kfold结果值

非合成类

```
Non_Complex_Kfold_Outdata = pd.DataFrame(np.array([Non_Complex_KNN_Kfold,
                                                   Non_Complex_Decision_Kfold,
                                                   Non_Complex_RFC_Kfold,
                                                   Non_Complex_SVM_Kfold,
                                                   Non_Complex_MLP_Kfold,
                                                   Non_Complex_LR_Kfold]).T.round(3),
                                         columns=["KNN","决策树","随机森林","支持向量机","神经网络","线性回归"])
Non_Complex_Kfold_Box = Non_Complex_Kfold_Outdata.stack().reset_index()
Non_Complex_Kfold_Box = Non_Complex_Kfold_Box.rename(columns={"level_1":"各类机器学习算法","0":"Kfold值"})
```

合成类

```
Complex_Kfold_Outdata = pd.DataFrame(np.array([Complex_KNN_Kfold,
                                               Complex_Decision_Kfold,
                                               Complex_RFC_Kfold,
                                               Complex_SVM_Kfold,
                                               Complex_MLP_Kfold,
                                               Complex_LR_Kfold]).T.round(3),
                                         columns=["KNN","决策树","随机森林","支持向量机","神经网络","线性回归"])
Complex_Kfold_Box = Complex_Kfold_Outdata.stack().reset_index()
Complex_Kfold_Box = Complex_Kfold_Box.rename(columns={"level_1":"各类机器学习算法","0":"Kfold值"})
```

半合成类

```
Semi_Complex_Kfold_Outdata = pd.DataFrame(np.array([Semi_Complex_KNN_Kfold,
                                                    Semi_Complex_Decision_Kfold,
                                                    Semi_Complex_RFC_Kfold,
                                                    Semi_Complex_SVM_Kfold,
                                                    Semi_Complex_MLP_Kfold,
                                                    Semi_Complex_LR_Kfold]).T.round(3),
                                         columns=["KNN","决策树","随机森林","支持向量机","神经网络","线性回归"])
Semi_Complex_Kfold_Box = Semi_Complex_Kfold_Outdata.stack().reset_index()
Semi_Complex_Kfold_Box = Semi_Complex_Kfold_Box.rename(columns={"level_1":"各类机器学习算法","0":"Kfold值"})
```

Kfold箱式图

```python
f,(Complex_Box1,Non_Complex_Box2,Semi_Complex_Box3) = plt.subplots(nrows=3,figsize=(15,15))

sns.boxplot(x = "各类机器学习算法", y = Complex_Kfold_Box.ix[:,-1],
            data=Complex_Kfold_Box,ax=Complex_Box1)
Complex_Box1.set_xlabel('')
Complex_Box1.set_ylabel('合成类')

sns.boxplot(x = "各类机器学习算法", y = Non_Complex_Kfold_Box.ix[:,-1],
            data=Non_Complex_Kfold_Box,ax=Non_Complex_Box2)
Non_Complex_Box2.set_xlabel('')
Non_Complex_Box2.set_ylabel('非合成类')

sns.boxplot(x = "各类机器学习算法", y = Semi_Complex_Kfold_Box.ix[:,-1],
            data=Semi_Complex_Kfold_Box,ax=Semi_Complex_Box3)
Semi_Complex_Box3.set_xlabel('')
Semi_Complex_Box3.set_ylabel('半合成类')

plt.tight_layout()
All_Box = plt.gcf()
```

导出结果

数据清洗结果

```python
# 整理后的ACS_ALL
ACS_ALL_5YR_DP02.to_csv(file_path("02_output","ACS_ALL_5YR_DP02.csv"),encoding="utf-8-sig")
# 整理后的MCM_NFLIS
MCM_NFLIS_Class_Clear.to_csv(file_path("02_output","MCM_NFLIS_Class_Clear.csv"),encoding="utf-8-sig")
# 整理后的ACS_All_5YR_DP02_metadata
ACS_All_5YR_DP02_metadata_Dup.to_csv(file_path("02_output","ACS_All_5YR_DP02_metadata_Dup.csv"),encoding="utf-8-sig")
# 按照三类药物数据合并
NFLIS_and_ACS_ALL_Out.to_csv(file_path("02_output","NFLIS_and_ACS_ALL_Out.csv"),encoding="utf-8-sig")
# 相关系数大于0.5的变量
All_Corr_Condi.to_csv(file_path("02_output","All_Corr_Condi.csv"),encoding="utf-8-sig")
# 相关系数大于0.5的变量的社会经济数据表
NFLIS_and_ACS_All_Corr_Condi.to_csv(file_path("02_output","NFLIS_and_ACS_All_Corr_Condi.csv"),encoding="utf-8-sig")
# 归一化后相关系数大于0.5的变量的社会经济数据表
NFLIS_and_ACS_All_Condi_Normal.to_csv(file_path("02_output","NFLIS_and_ACS_All_Condi_Normal.csv"),encoding="utf-8-sig")
```

统计描述结果

```
NFLIS_Figure2_Data_Class1.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class1.csv"),encoding="utf-8-sig")
NFLIS_Figure2_Data_Class2.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class2.csv"),encoding="utf-8-sig")
NFLIS_Figure2_Data_Class3.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class3.csv"),encoding="utf-8-sig")
```

```
Figure1.savefig(file_path("02_output","Figure1_Pie.jpg"),dpi=500)
Figure2.savefig(file_path("02_output","Figure2_Bar.jpg"),dpi=500)
Figure3.savefig(file_path("02_output","Figure3_HeatMap.jpg"),dpi=500)
Figure4.savefig(file_path("02_output","Figure4_Plot.jpg"),dpi=500)

```

模型评估结果

```
# Kfold箱式图
All_Box.savefig(file_path("02_output","All_Box.jpg"),dpi=500)
# 三类药物特征重要度
All_Importance_Rename.to_csv(file_path("02_output","All_Importance_Rename.csv"),encoding="utf-8-sig")
# K折验证
Complex_Kfold_Outdata.to_csv(file_path("02_output","Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
Semi_Complex_Kfold_Outdata.to_csv(file_path("02_output","Semi_Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
Non_Complex_Kfold_Outdata.to_csv(file_path("02_output","Non_Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
```

模型的混淆矩阵

```

print('KNN合成类混淆矩阵为：', confusion_matrix(Complex_y_test, Complex_KNN_Y_Predict), sep='\n')
print('KNN半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test, Semi_Complex_KNN_Y_Predict), sep='\n')
print('KNN非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test, Non_Complex_KNN_Y_Predict), sep='\n')

print('Decision合成类混淆矩阵为：', confusion_matrix(Complex_y_test, Complex_Decision_Y_Predict), sep='\n')
print('Decision半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test, Semi_Complex_Decision_Y_Predict), sep='\n')
print('Decision非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test, Non_Complex_Decision_Y_Predict), sep='\n')

print('RFC合成类混淆矩阵为：', confusion_matrix(Complex_y_test, Complex_RFC_Y_Predict), sep='\n')
print('RFC半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test, Semi_Complex_RFC_Y_Predict), sep='\n')
print('RFC非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test, Non_Complex_RFC_Y_Predict), sep='\n')

```

```
print('SVM合成类混淆矩阵为：', confusion_matrix(Complex_y_test,
Complex_SVM_Y_Predict), sep='\n')
print('SVM半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test,
Semi_Complex_SVM_Y_Predict), sep='\n')
print('SVM非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test,
Non_Complex_SVM_Y_Predict), sep='\n')

print('MLP合成类混淆矩阵为：', confusion_matrix(Complex_y_test,
Complex_MLP_Y_Predict), sep='\n')
print('MLP半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test,
Semi_Complex_MLP_Y_Predict), sep='\n')
print('MLP非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test,
Non_Complex_MLP_Y_Predict), sep='\n')

print('LR合成类混淆矩阵为：', confusion_matrix(Complex_y_test,
Complex_LR_Y_Predict), sep='\n')
print('LR半合成类混淆矩阵为：', confusion_matrix(Semi_Complex_y_test,
Semi_Complex_LR_Y_Predict), sep='\n')
print('LR非合成类混淆矩阵为：', confusion_matrix(Non_Complex_y_test,
Non_Complex_LR_Y_Predict), sep='\n')

```

模型的评估报告

```
print("最近邻法合成阿片类：")
print(classification_report(Complex_KNN_Y_Predict,Complex_y_test))
print("最近邻法非合成阿片类：")
print(classification_report(Non_Complex_KNN_Y_Predict,Non_Complex_y_test))
print("最近邻法半合成阿片类：")
print(classification_report(Semi_Complex_KNN_Y_Predict,Semi_Complex_y_test))

print("决策树合成阿片类：")
print(classification_report(Complex_Decision_Y_Predict,Complex_y_test))
print("决策树非合成阿片类：")
print(classification_report(Non_Complex_Decision_Y_Predict,Non_Complex_y_tes
t))
print("决策树半合成阿片类：")
print(classification_report(Semi_Complex_Decision_Y_Predict,Semi_Complex_y_t
est))

print("随机森林合成阿片类：")
print(classification_report(Complex_RFC_Y_Predict,Complex_y_test))
print("随机森林非合成阿片类：")
print(classification_report(Non_Complex_RFC_Y_Predict,Non_Complex_y_test))
print("随机森林半合成阿片类：")
print(classification_report(Semi_Complex_RFC_Y_Predict,Semi_Complex_y_test))

print("支持向量机合成阿片类：")
print(classification_report(Complex_SVM_Y_Predict,Complex_y_test))
print("支持向量机非合成阿片类：")
print(classification_report(Non_Complex_SVM_Y_Predict,Non_Complex_y_test))
print("支持向量机半合成阿片类：")
print(classification_report(Semi_Complex_SVM_Y_Predict,Semi_Complex_y_test))

print("神经网络合成阿片类：")
print(classification_report(Complex_MLP_Y_Predict,Complex_y_test))
print("神经网络非合成阿片类：")
```

```
print(classification_report(Non_Complex_MLP_Y_Predict,Non_Complex_y_test))
print("神经网络半合成阿片类：")
print(classification_report(Semi_Complex_MLP_Y_Predict,Semi_Complex_y_test))

print("线性回归合成阿片类：")
print(classification_report(Complex_LR_Y_Predict,Complex_y_test))
print("线性回归非合成阿片类：")
print(classification_report(Non_Complex_LR_Y_Predict,Non_Complex_y_test))
print("线性回归半合成阿片类：")
print(classification_report(Semi_Complex_LR_Y_Predict,Semi_Complex_y_test))

```