哈尔滨工业大学（深圳）

大数据实验指导书

实验二 数据理解、数据预处理及决策树的应用

1. 实验目的

1. 学会理解数据并对数据进行预处理；
2. 理解决策树的原理并掌握其构建方法。

2. 实验内容

1. 熟悉 Pandas 的安装和使用，并对数据进行预处理和相关分析；
2. 请编写代码实现一种决策树算法，解决个人年收入的分类问题。

3. 实验环境

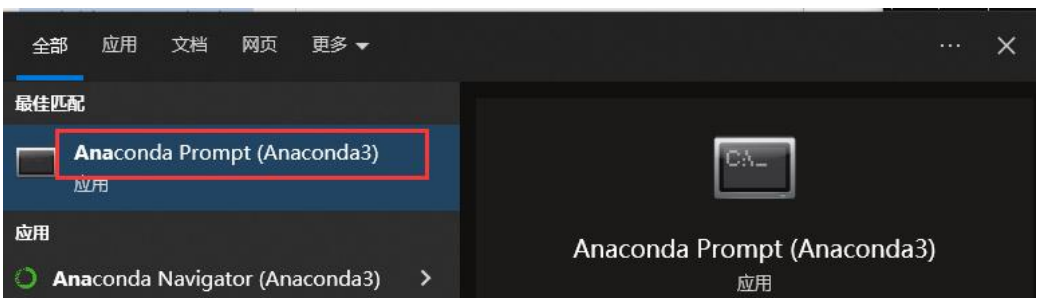
- Jupyter
- PyCharm

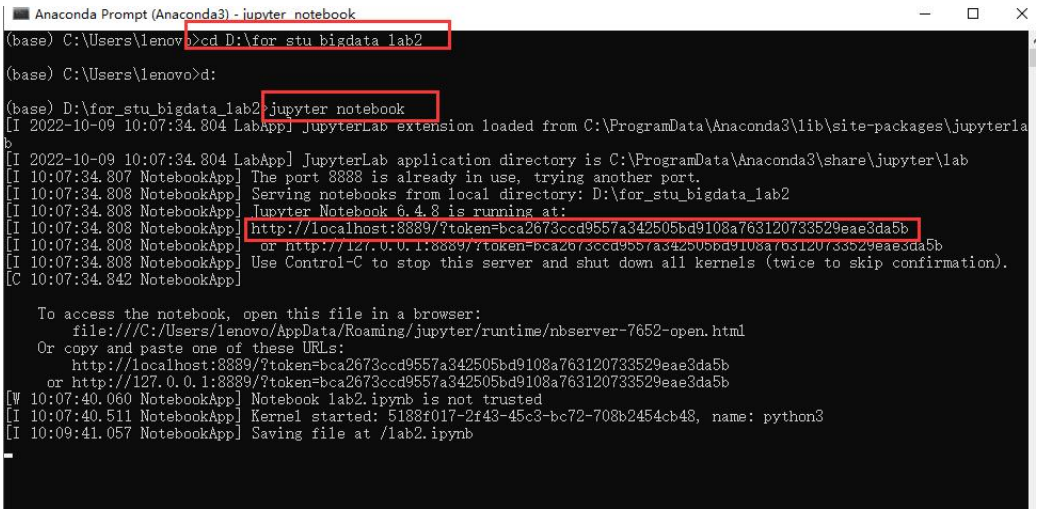
4. 实验步骤

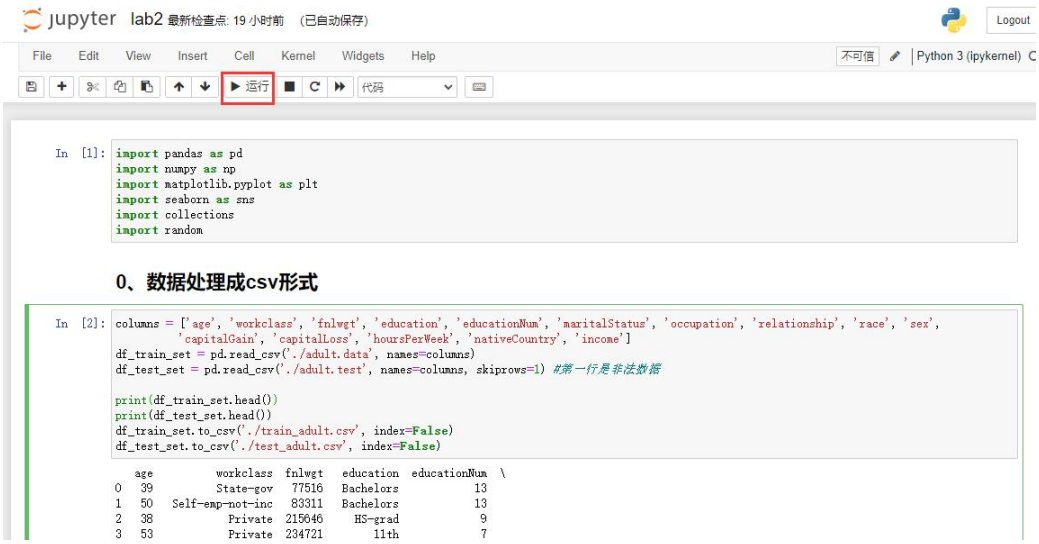
4.1 启动 jupyter notebook

Jupyter Notebook 是一个 Web 应用程序，允许创建和共享包含实时代码、方程、可视化和说明文本的文档。通俗来讲，Jupyter Notebook 是以网页的形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示的程序。

通过 Anaconda Prompt，进入到实验目录下，输入 jupyter notebook 后，会弹出 jupyter 的操作页面，若未弹出则复制红框中的 url 到浏览器中。







4.2 安装 Pandas 库并熟悉其基本操作

Pandas 是 python 第三方库，提供高性能易用数据类型和分析工具。Pandas 基于 numpy 实现，常与 numpy 和 matplotlib 一同使用，更多学习请参考 pandas 中文网：https://www.pypandas.cn/。

4.2.1 安装 Pandas 库

打开 cmd 命令行窗口执行：

```
pip install pandas
```

4.2.2 Pandas 核心数据结构

维数	名称	描述
1	Series	带标签的一维同构数组
2	DataFrame	带标签的，大小可变的，二维异构表格

Series 是带标签的一维数组，可存储整数、浮点数、字符串、Python 对象等类型的数据。轴标签统称为索引。Series 中只允许存储相同的数据类型，这样可以更有效的使用内存，提高运算效率。调用 pd.Series 函数即可创建 Series。

DataFrame 是一个表格型的数据结构，类似于 Excel 或 sql 表，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame 既有行索引也有列索引，它可以被看做由 Series 组成的字典（共用同一个索引），可用多维数组字典、列表字典生成 DataFrame。

4.2.3 Pandas 库基本操作

（1）生成数据

[5]:
```
import numpy as np
import pandas as pd
data = {'A':[1,2,3], 'B': [4,5,6], 'C': [7,8,9]}
df = pd.DataFrame(data)
print(df)
```

```
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
```

（2）计算数据的基本信息

[7]:
```
df.describe()
```

[7]:

	A	B	C
count	3.0	3.0	3.0
mean	2.0	5.0	8.0
std	1.0	1.0	1.0
min	1.0	4.0	7.0
25%	1.5	4.5	7.5
50%	2.0	5.0	8.0
75%	2.5	5.5	8.5
max	3.0	6.0	9.0

(3) 选取特定列

```
df[['A','C']]
```

	A	C
0	1	7
1	2	8
2	3	9

(4) 选取特定列行

```
df.iloc[[1,2]]
```

	A	B	C
1	2	5	8
2	3	6	9

(5) 选取多行多列

```
df.iloc[[1,2],[0,2]]
```

	A	C
1	2	8
2	3	9

(6) 选取 B 列大于等于 5 的数据

```
df[df['B'] >= 5]
```

	A	B	C
1	2	5	8
2	3	6	9

(7) 修改指定列

```
df['B'] = 0
print(df)
```

```
   A  B  C
0  1  0  7
1  2  0  8
2  3  0  9
```

4.3 数据读取及预处理

Adult Data Set 数据集是 Barry Becker 从 1994 年的人口普查数据库中整理筛选得到的，数据收集和整理过程规范统一，数据质量和可信度高，样本量充足且缺失值、异常值较少，能够一定程度上保证模型的质量和可信度。

记录数量：48842(训练集：32561，测试集：16281) 。

属性：14（不包括类别） 。

2 分类任务：预测一个人年收入(income)是否超过 50k 美元 。

4.3.1 将数据处理成 csv 格式，并保存

```
import pandas as pd
columns = ['age', 'workclass', 'fnlwgt', 'education', 'educationNum', 'maritalStatus', 'occupation', 'relationship', 'race', 'sex',
           'capitalGain', 'capitalLoss', 'hoursPerWeek', 'nativeCountry', 'income']
df_train_set = pd.read_csv('./adult.data', names=columns)
df_test_set = pd.read_csv('./adult.test', names=columns, skiprows=1) #第一行是非法数据

print(df_train_set.head())
print(df_test_set.head())
df_train_set.to_csv('./train_adult.csv', index=False)
df_test_set.to_csv('./test_adult.csv', index=False)
```

```
   age          workclass  fnlwgt   education  educationNum  \
0   39          State-gov   77516   Bachelors            13
1   50   Self-emp-not-inc   83311   Bachelors            13
2   38            Private  215646     HS-grad             9
3   53            Private  234721        11th             7
4   28            Private  338409   Bachelors            13

        maritalStatus          occupation    relationship    race      sex  \
0       Never-married        Adm-clerical   Not-in-family   White     Male
1  Married-civ-spouse     Exec-managerial         Husband   White     Male
2            Divorced   Handlers-cleaners   Not-in-family   White     Male
3  Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
4  Married-civ-spouse      Prof-specialty            Wife   Black   Female

   capitalGain  capitalLoss  hoursPerWeek   nativeCountry  income
0         2174            0            40   United-States   <=50K
1            0            0            13   United-States   <=50K
2            0            0            40   United-States   <=50K
3            0            0            40   United-States   <=50K
4            0            0            40            Cuba   <=50K
   age   workclass  fnlwgt      education  educationNum        maritalStatus  \
0   25     Private  226802           11th             7        Never-married
1   38     Private   89814        HS-grad             9   Married-civ-spouse
2   28   Local-gov  336951     Assoc-acdm            12   Married-civ-spouse
3   44     Private  160323   Some-college            10   Married-civ-spouse
4   18           ?  103497   Some-college            10        Never-married

          occupation relationship    race      sex  capitalGain  capitalLoss  \
0   Machine-op-inspct    Own-child   Black     Male            0            0
1     Farming-fishing      Husband   White     Male            0            0
2     Protective-serv      Husband   White     Male            0            0
3   Machine-op-inspct      Husband   Black     Male         7688            0
4                   ?    Own-child   White   Female            0            0
```

4.3.2 读取数据

（1）读取 csv 文件并获取所有的表头和数据信息，转化为 DataFrame 格式

```
import pandas as pd
df_train_set = pd.read_csv('./train_adult.csv')
df_train_set
```

	age	workclass	fnlwgt	education	educationNum	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	native
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Unit
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Unit
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Unit
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Unit
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	Unit
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	Unit
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	Unit
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	Unit
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	Unit

32561 rows × 15 columns

（2）数据属性描述

属性	描述	类型
age	年龄	连续型
workclass	工作类别	离散型
fnlwgt	样本权重	连续型
education	受教育程度	离散型
education-num	受教育时间	连续型
marital-status	婚姻状况	离散型
occupation	职业	离散型

属性	描述	类型
relationship	社会角色	离散型
race	种族	离散型
sex	性别	离散型
capital-gain	资本收益	连续型
capital-loss	资本支出	连续型
hours-per-week	每周工作时间	连续型
native-country	国籍	离散型
income（目标）	年收入	离散型

（3）查看列类型

```
df_test_set.dtypes
```

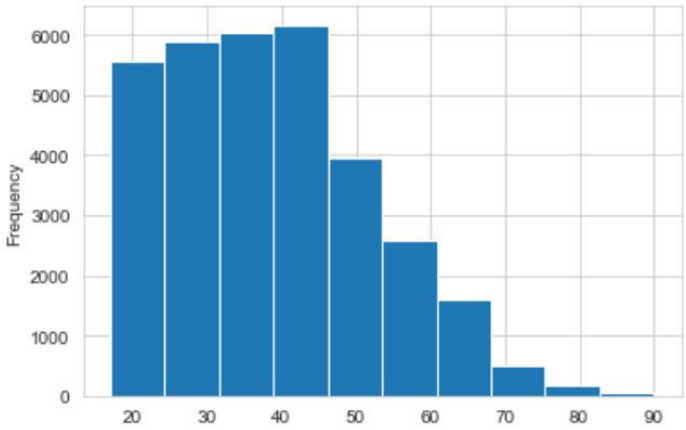
```
age              int64
workclass       object
fnlwgt           int64
education       object
educationNum     int64
maritalStatus   object
occupation      object
relationship    object
race            object
sex             object
capitalGain      int64
capitalLoss      int64
hoursPerWeek     int64
nativeCountry   object
income          object
dtype: object
```

（4）数据可视化

以 age 为例：

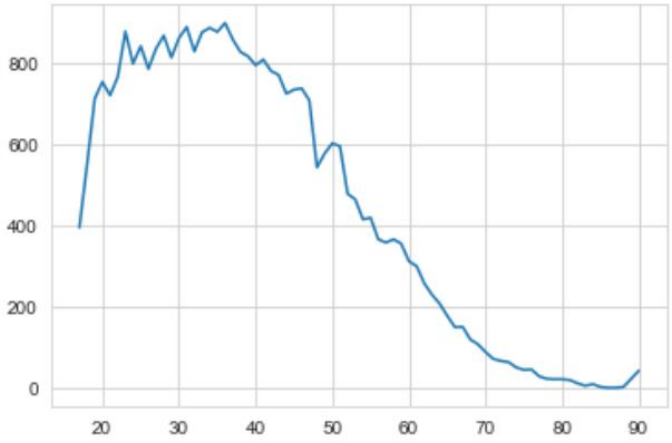
```
df_train_set['age'].plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```




```
df_train_set['age'].value_counts().sort_index().plot.line()
```

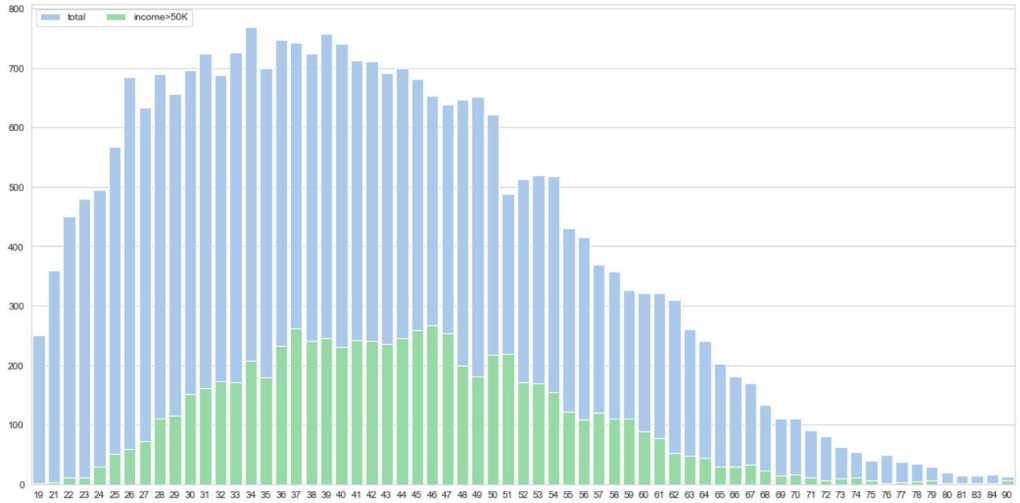
```
<AxesSubplot:>
```




```
# 画出年龄与收入的关系

df_train_set = df_train_set.reset_index(drop=True) #重置索引
df_train_set['age'].isnull() == True
s=df_train_set['age'].value_counts()
k=df_train_set['age'][df_train_set['income']==' >50K'].value_counts()
sns.set_style("whitegrid")
f, ax = plt.subplots(figsize=(18, 9))
sns.set_color_codes("pastel")
sns.barplot(s.index, s.values, label='total', color="b")
sns.barplot(k.index, k.values, label='income>50K', color="g")
ax.legend(ncol=2, loc="upper left", frameon=True)
```

```
<matplotlib.legend.Legend at 0x20b54dbad90>
```



4.3.3 数据预处理

（1） 删除指定的属性

```
# fnlwgt列用处不大，educationNum与education类似
df_train_set.drop(['fnlwgt', 'educationNum'], axis=1, inplace=True)
```

```
df_train_set.columns
```

```
Index(['age', 'workclass', 'education', 'maritalStatus', 'occupation',
       'relationship', 'race', 'sex', 'capitalGain', 'capitalLoss',
       'hoursPerWeek', 'nativeCountry', 'income'],
      dtype='object')
```

（2）重复记录处理

```
df_train_set.drop_duplicates(inplace=True)# 去除重复的行
df_train_set.shape[0]#获取行数
```

```
29096
```

（3）缺失值处理

```
df_train_set[df_train_set.isna().values == True]  # 输出有缺失值的数据行
```

age	workclass	education	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry

```
df_train_set.dropna(inplace=True)#删除包含缺失值的行
df_train_set.shape[0] #获取行数
```

```
29096
```

(4) 异常值处理

以 workclass 属性为例:

筛出含有异常值？的记录

```
df_train_set[df_train_set['workclass'].str.contains(r'\?', regex=True)] # 查找异常值，避免与正则表达式的?冲突需要转义
```

	age	workclass	education	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry	income
27	54	?	Some-college	Married-civ-spouse	?	Husband	Asian-Pac-Islander	Male	0	0	60	South	>50K
61	32	?	7th-8th	Married-spouse-absent	?	Not-in-family	White	Male	0	0	40	?	<=50K
69	25	?	Some-college	Never-married	?	Own-child	White	Male	0	0	40	United-States	<=50K
77	67	?	10th	Married-civ-spouse	?	Husband	White	Male	0	0	2	United-States	<=50K
106	17	?	10th	Never-married	?	Own-child	White	Female	34095	0	32	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32530	35	?	Bachelors	Married-civ-spouse	?	Wife	White	Female	0	0	55	United-States	>50K
32531	30	?	Bachelors	Never-married	?	Not-in-family	Asian-Pac-Islander	Female	0	0	99	United-States	<=50K
32539	71	?	Doctorate	Married-civ-spouse	?	Husband	White	Male	0	0	10	United-States	>50K
32541	41	?	HS-grad	Separated	?	Not-in-family	Black	Female	0	0	32	United-States	<=50K
32542	72	?	HS-grad	Married-civ-spouse	?	Husband	White	Male	0	0	25	United-States	<=50K

取出 workclass 不含有异常值？的记录

```
df_train_set=df_train_set[~df_train_set['workclass'].str.contains(r'\?', regex=True)]
df_train_set
```

	age	workclass	education	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry	income
0	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32554	53	Private	Masters	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	>50K
32555	22	Private	Some-college	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40	United-States	<=50K
32556	27	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32558	58	Private	HS-grad	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32560	52	Self-emp-inc	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

对所有属性去除含有?异常值的记录

```
#删除有异常值的行
new_columns = ['workclass', 'education', 'maritalStatus', 'occupation', 'relationship', 'race', 'sex',
               'nativeCountry', 'income']
for col in new_columns:
        df_train_set = df_train_set[~df_train_set[col].str.contains(r'\?', regex=True)]
df_train_set
```

	age	workclass	education	maritalStatus	occupation	relationship	race	sex	capitalGain	capitalLoss	hoursPerWeek	nativeCountry	income
0	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32554	53	Private	Masters	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	40	United-States	>50K
32555	22	Private	Some-college	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40	United-States	<=50K
32556	27	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32558	58	Private	HS-grad	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32560	52	Self-emp-inc	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

26904 rows × 13 columns

（5）离散型属性处理

以 workclass 属性为例:

查看 workclass 属性的取值个数

```
df_train_set['workclass'].value_counts()
```

```
 Private             19214
 Self-emp-not-inc     2431
 Local-gov            2014
 State-gov            1253
 Self-emp-inc         1049
 Federal-gov           929
 Without-pay            14
Name: workclass, dtype: int64
```

```
df_train_set['workclass'].head() #展示前五条
```

```
0            State-gov
1     Self-emp-not-inc
2              Private
3              Private
4              Private
Name: workclass, dtype: object
```

查看 workclass 属性的 keys 值

```
df_train_set['workclass'].value_counts().keys()
```

```
Index([' Private', ' Self-emp-not-inc', ' Local-gov', ' State-gov',
       ' Self-emp-inc', ' Federal-gov', ' Without-pay'],
      dtype='object')
```

映射为数字

```
workclass_mapping = {' Private': 0, ' Self-emp-not-inc': 1, ' Self-emp-inc': 1, ' Local-gov': 2,
                     ' State-gov': 2, ' Federal-gov': 2, ' Without-pay': 3, ' Never-worked': 3}
df_train_set['workclass'] = df_train_set['workclass'].map(workclass_mapping)
```

```
df_train_set['workclass'].head()
```

```
0    2
1    1
2    0
3    0
4    0
Name: workclass, dtype: int64
```

目标属性 income：

```
df_train_set['income'].value_counts()
```

```
 <=50K    20024
 >50K      6880
Name: income, dtype: int64
```

```
df_train_set['income'].value_counts().keys()
```

```
Index([' <=50K', ' >50K'], dtype='object')
```

```
income_mapping = {' <=50K': 0, ' >50K': 1}
df_train_set['income'] = df_train_set['income'].map(income_mapping)
```

（6）连续型属性处理

决策树对于连续值的属性进行处理的方法主要是将连续性数值属性划分为不同的区间，从而变成离散的数值。常用的离散化策略有分箱法和二分法等。

（a）分箱法

分箱就是把数据按特定的规则进行分组，实现数据的离散化，增强数据稳定性，减少过拟合风险。连续型特征的分箱分为无监督分箱与有监督分箱两类。

- 无监督分箱：不需要提供 Y，仅凭借特征就能实现分箱

等宽分箱: 从最小值到最大值之间，均分为 N 等份。这里只考虑边界，每个等份的实例数量可能不等。

等频分箱: 区间的边界值要经过选择,使得每个区间包含大致相等的实例数量。

以年龄属性为例：

```
df_train_set['age'].max(), df_train_set['age'].min()
```

```
(90, 17)
```

```
df_train_set['age'].head()
```

```
0    39
1    50
2    38
3    53
4    28
Name: age, dtype: int64
```

```
bins = [0, 25, 50, 75, 100] # 分箱区间左开右闭 (0, 25], (25, 50], ...
df_train_set['age'] = pd.cut(df_train_set['age'], bins, labels=False)
```

```
df_train_set['age'].head()
```

```
0    1
1    1
2    1
3    2
4    1
Name: age, dtype: int64
```

- 有监督分箱：需要结合 Y 的值，通过算法实现分箱
  决策树分箱
  卡方分箱

（b）二分法

C4.5 采用二分法对连续属性进行离散化。其基本思想为：对于某个属性出现的连续值从小到大排列，取每两个点的中点进行划分，选取其中信息增益最大的点作为最终划分节点的依据。对于 CART 分类树连续值的处理问题，其思想和 C4.5 是相同的，都是将连续的特征离散化。唯一的区别在于在选择划分点时的度量方式不同，C4.5 使用的是信息增益，则 CART 分类树使用的是基尼系数。

（7）保存数据

```
df_train_set.to_csv('./after_train_adult.csv',index=False)
```

4.4 构建决策树

决策树是一种树形结构的分类器， 树内部的每一个节点代表的是对一个特征的测试，树的分支代表该特征的每一个测试结果，而树的每一个叶子节点代表一个类别。通常根据特征的信息增益或其他指标，构建一棵决策树。

一棵决策树的生成过程主要分为以下 3 个部分:

（1）特征选择：特征选择是指从训练数据中众多的特征中选择一个特征作为当前节点的分裂标准，如何选择特征有着很多不同量化评估标准（信息增益、信息增益率、基尼系数等），从而衍生出不同的决策树算法；

（2）决策树生成： 根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长；

（3）剪枝：决策树容易过拟合，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有预剪枝和后剪枝两种。

此部分为实验的主体部分，请**编码实现某种决策树算法**，解决个人年收入的分类问题。**实验要求如下：**

（1） 编程语言不限

（2） 决策树算法不限， 可选择 ID3、C4.5、C5.0、CART 等

（3） 构建决策树不允许调用 sklearn 等现成库

（4） 训练集和测试集可自行设计，采用剪枝、交叉验证等可作为加分项

（5） 输入训练数据构建决策树

（6） 输入测试数据，使用训练好的决策树进行预测，输出预测结果和准确率

此实验需要提交的内容包括：

1. 实验报告内容包括预处理的过程及结果、构建决策树的基本过程、测试结果和预测准确率。决策树构建过程需描述构建决策树的思想，连续特征离散化的方法、特征选择的方法，剪枝的方法等。

2. 复现实验结果所需文件，包含决策树代码和相应的数据文件。

3. 预测结果文件

已给出部分代码框架（以 python 语言为例，**注意以下函数的参数不一定是这些，下面只是个 demo）：**

（1）计算基尼指数

```
def calc_gini(df):
    """
    计算数据集的基尼指数
    :param df: 数据集
    :return: 基尼指数
    """
```

（2） 按照给定的列划分数据集

```
def split_dataset(df, index, value):
    """
    按照给定的列划分数据集
    :param df: 原始数据集
    :param index: 指定特征的列索引
    :param value: 指定特征的值
    :return: 切分后的数据集
    """
```

（3） 选择最好的特征进行分裂

```
def choose_best_feature_to_split(df):
    """
    选择最好的特征进行分裂
    :param df: 数据集
    :return: best_value:(分裂特征的index, 特征的值), best_df:(分裂后的左右子树数据集), best_gain:(选择该属性分裂的最大信息增益)
    """
```

（4） 构建决策树

```
def build_decision_tree(df, columns, flags):
    """
    构建CART树
    :param df: 数据集
    :param columns: 特征列表
    :param flags: 区分特征是否被完全区分开,初始为全0, 若某个特征被区分开那么flags对应的下标为0
    :return: CART树
    """
```

(5) 决策树的保存

```
def save_decision_tree(cart):
    """
    决策树的存储
    :param cart: 训练好的决策树
    :return: void
    """
    np.save('cart.npy', cart)
```

(6) 决策树的加载

```
def load_decision_tree():
    """
    决策树的加载
    :return: 保存的决策树
    """

    cart = np.load('cart.npy', allow_pickle=True)
    return cart.item()
```

(7) 训练完毕后，一条样本的分类

```
def classify(cart, df_row, columns):
    """
    用训练好的决策树进行分类
    :param cart:决策树模型
    :param df_row: 一条测试样本
    :param columns: 特征列表
    :return: 预测结果
    """
```

(8) 测试集的分类结果

```
def predict(cart, df, columns):
    """
    用训练好的决策树进行分类
    :param cart:决策树模型
    :param df: 所有测试集
    :param columns: 特征列表
    :return: 预测结果
    """
    pred_list = []
    for i in range(len(df)):
        pred_label = classify(cart, df.iloc[i,:], columns)
        if pred_label == -1:
            pred_label = random.randint(0, 1) # 防止classify执行到返回-1,但一般不会执行到返回-1
        pred_list.append(pred_label)
    return pred_list
```

(9) 计算准确率

```
def calc_acc(pred_list, test_list):
    """
    返回预测准确率
    :param pred_list: 预测列表
    :param test_list: 测试列表
    :return: 准确率
    """
    pred = np.array(pred_list)
    test = np.array(test_list)
    acc = np.sum(pred_list == test_list) / len(test_list)
    return acc
```

5 补充内容：安装 sklearn 并构建决策树

此部分仅作补充内容， 本次实验不允许调用 sklearn 库！！！

决策树是一种树形结构的分类器，通过顺序询问分类点的属性决定分类点最终类别。通常根据特征的信息增益或其他指标，构建一棵决策树。

5.1 安装 sklearn

```
D:\大数据\实验二>pip install scikit-learn
```

5.2 读取数据

```
from sklearn.tree import DecisionTreeClassifier as DTC, export_graphviz
# 读取数据
df = pd.read_csv('D:/大数据/实验二/after_bank.csv')
df = df.iloc[:,1:]
cols = list(df.columns.values)
cols.remove('y')
X = df[cols]
y = df[['y']]
```

5.3 划分训练集和测试集

```
# 划分训练集与测试集
X_train = X[:4000]
y_train = y[:4000]
X_test = X[4000:5000]
y_test = y[4000:5000]
```

5.4 训练模型

```
dtc = DTC(criterion='entropy', max_depth=5)  # 基于信息熵
dtc.fit(X_train, y_train)
print('准确率: ', dtc.score(X_test, y_test))
```

```
准确率:  0.8886756238003839
```

5.5 参数调整

构建模型中很重要的一步是**调参**。在 sklearn 中，模型的参数是通过方法参数来决定的，以下给出 sklearn 中，决策树的参数：

```
DecisionTreeClassifier(criterion="gini",
                       splitter="best",
                       max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.,
                       max_features=None,
                       random_state=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.,
                       min_impurity_split=None,
                       class_weight=None,
                       presort=False)
```

通常来说，较为重要的参数有：

criterion：用以设置用信息熵还是基尼系数计算

string, optional (default="gini")

(1)criterion='gini',分裂节点时评价准则是 Gini 指数。

(2)criterion='entropy',分裂节点时的评价指标是信息增益。

splitter：指定分支模式

string, optional (default="best")，指定分裂节点时的策略。

(1)splitter='best',表示选择最优的分裂策略。

(2)splitter='random',表示选择最好的随机切分策略。

max_depth：最大深度，防止过拟合

int or None, optional (default=None)，指定树的最大深度。

如果为 None，表示树的深度不限。直到所有的叶子节点都是纯净的，即叶子节点中所有的样本点都属于同一个类别。或者每个叶子节点包含的样本数小于 min_samples_split。

min_samples_leaf：限定每个节点分枝后子节点至少有多少个数据，否则就不分枝

int, float, optional (default=2)。表示分裂一个内部节点需要的做少样本数。

(1)如果为整数，则 min_samples_split 就是最少样本数。

(2)如果为浮点数(0 到 1 之间)，则每次分裂最少样本数为 ceil(min_samples_split * n_samples)

不同参数对结果的影响：

（1）采用基尼系数替换信息熵对结果的影响：

```
dtc = DTC(criterion='gini', max_depth=5)  # 基于基尼系数
dtc.fit(X_train, y_train)
print('准确率: ', dtc.score(X_test, y_test))
```

```
准确率:  0.8925143953934741
```

（2）采用不同的树的深度，对结果的影响

```
for depth in range(1, 10):
    dtc = DTC(criterion='entropy', max_depth=depth)  # 基于信息熵
    dtc.fit(X_train, y_train)
    print('depth:', depth, '|', '准确率:', dtc.score(X_test, y_test))
```

```
depth: 1 | 准确率: 0.8790786948176583
depth: 2 | 准确率: 0.8733205374280231
depth: 3 | 准确率: 0.8963531669865643
depth: 4 | 准确率: 0.9001919385796545
depth: 5 | 准确率: 0.9001919385796545
depth: 6 | 准确率: 0.8905950095969289
depth: 7 | 准确率: 0.8905950095969289
depth: 8 | 准确率: 0.8867562380038387
depth: 9 | 准确率: 0.8848368522072937
```

5.6 可视化决策树模型

（1）安装 Graphviz

下载 graphviz-install-2.44.1-win64.exe 并安装，下载地址：

https://www2.graphviz.org/Packages/stable/windows/10/cmake/Release/x64/

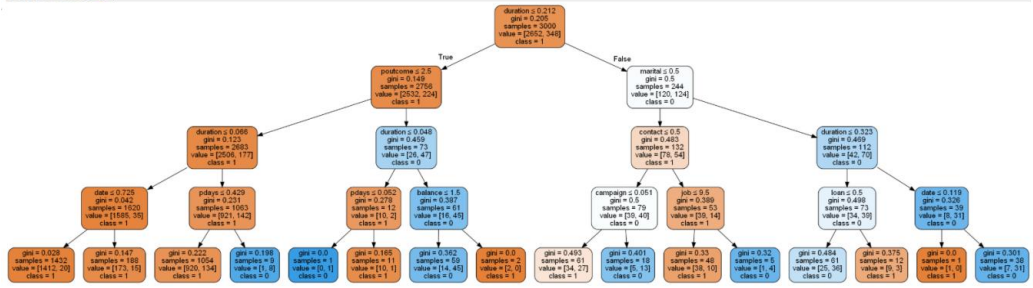
（2）安装 pydotplus

```
pip install pydotplus
```

（3）绘制决策树模型

```
from IPython.display import Image
from sklearn import tree
import pydotplus
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz 2.44.1/bin/'
dot_data = tree.export_graphviz(dtc, out_file=None,
                                feature_names=X_train.columns,
                                class_names=['1', '0'],
                                filled=True, rounded=True,
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png("D:/大数据/实验二/DecisionTree.gif")

Image(graph.create_png())
```



绘制决策树时，如果遇到：
InvocationException: Program terminated with status: 1. stderr follows: Format: " png" not recognized. Use one of:

可用管理员身份运行 cmd，执行

```
C:\Windows\system32>cd c:/

c:\>cd Program Files\Graphviz 2.44.1\bin

c:\Program Files\Graphviz 2.44.1\bin>dot -c

c:\Program Files\Graphviz 2.44.1\bin>
```