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，解决银行精准营销的分类问题。

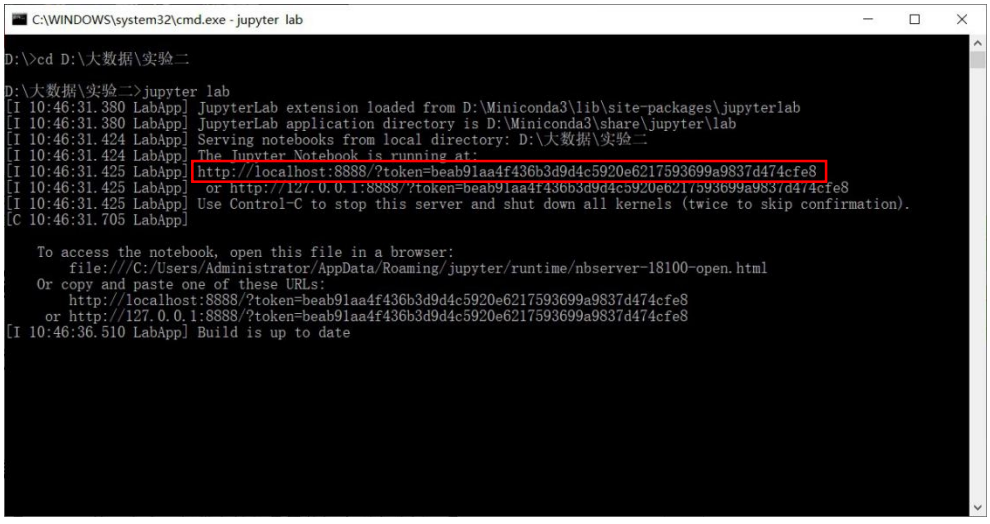
3. 实验环境

- Jupyter
- PyCharm

4. 实验步骤

4.1 启动 jupyter

通过 cmd，进入到实验二的目录下，输入 jupyter lab 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
```

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

（2）选取特定列

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

	A	C
0	1	7
1	2	8
2	3	9

（3）选取特定列行

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

	A	B	C
1	2	5	8
2	3	6	9

（4）选取多行多列

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

	A	C
1	2	8
2	3	9

（5）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

（6）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

4.3.1 读取数据

（1）读取 csv 文件并获取所有的表头

```
import numpy as np
import pandas as pd
df = pd.read_csv("D:/大数据/实验二/bank.csv")
cols = df.columns.values
cols = [col.replace('\"', '') for col in cols[0].split(';')]
print(cols)
```

```
['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'y']
```

属性	描述	类型	缺失值
age	年龄	数值	无
job	工作类型	分类	unknown
marital	婚姻状况	分类	无
education	受教育程度	分类	unknown
default	是否存在拖欠	分类	无
balance	平均年度余额（欧元）	数值	无
housing	是否有住房贷款	分类	无
loan	是否有个人贷款	分类	无
contact	联络方式	分类	unknown
day	最后一次联系是几号	数值	无
month	最后一次联系的月份	分类	无
duration	最后一次联系的持续时间（秒）	数值	无

属性	描述	类型	缺失值
campaign	此营销活动期间和此客户联系的次数，包括最后一次联系	数值	无
pdays	从上一个营销活动最后一次联系客户后经过的天数，-1表示之前未联系过客户	数值	无
previous	此营销活动之前和此客户联系的次数	数值	无
poutcome	上一次营销活动的结果	分类	unknown
y	客户是否将订购银行定期存款	分类	无

（2）获取所有的数据并转为二维数组的形式

```
datas = []
for i in range(len(df)):
    s = df.iloc[i].values[0]
    datas.append([item.replace('\"', '') for item in s.split(';')])
datas = np.array(datas)
print(datas)
```

```
[['30' 'unemployed' 'married' ... '0' 'unknown' 'no']
 ['33' 'services' 'married' ... '4' 'failure' 'no']
 ['35' 'management' 'single' ... '1' 'failure' 'no']
 ...
 ['57' 'technician' 'married' ... '0' 'unknown' 'no']
 ['28' 'blue-collar' 'married' ... '3' 'other' 'no']
 ['44' 'entrepreneur' 'single' ... '7' 'other' 'no']]
```

（3）将数据与表头转化为 DataFrame 格式

```
df_frame = {}
for i in range(len(cols)):
    df_frame[cols[i]] = datas[:,i]
df = pd.DataFrame(df_frame)
print(df)
```

```
      age           job  marital  education default  ... campaign pdays previous poutcome   y
0      30    unemployed  married    primary      no  ...        1    -1        0  unknown  no
1      33      services  married  secondary      no  ...        1   339        4  failure  no
...    ..           ...      ...        ...     ...  ...      ...   ...      ...      ...  ..
4519   28   blue-collar  married  secondary      no  ...        4   211        3    other  no
4520   44  entrepreneur   single   tertiary      no  ...        2   249        7    other  no

[4521 rows x 17 columns]
```

（4）将以下列从字符类型转为整数类型

```
df[['age','balance','duration', 'campaign','pdays','previous']] = df[['age','balance','duration', 'campaign','pdays','previous']].astype(int)
df.dtypes
```

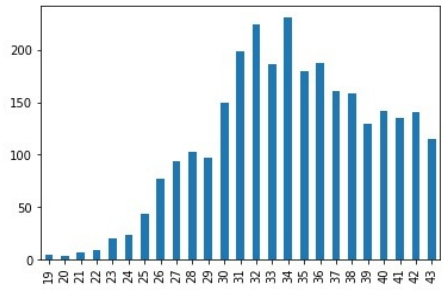
```
age           int32
job          object
marital      object
education    object
default      object
balance       int32
housing      object
loan         object
contact      object
day          object
month        object
duration      int32
campaign      int32
pdays         int32
previous      int32
poutcome     object
y            object
Length: 17, dtype: object
```

（5）数据可视化

绘制柱状图:

```
df['age'].value_counts().sort_index().head(25).plot.bar()
```

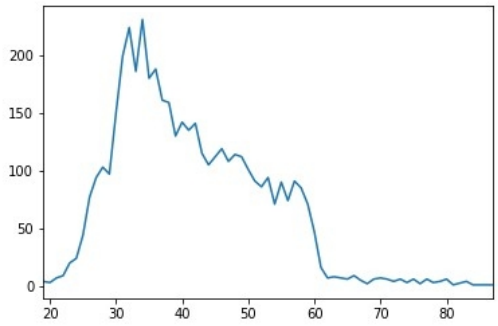
```
<AxesSubplot:>
```



绘制折线图：

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x247287b8be0>
```



4.3.2 数据预处理

（1）筛选重复值并删除

```
df.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
4516    False
4517    False
4518    False
4519    False
4520    False
Length: 4521, dtype: bool
```

```
df.drop_duplicates()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	5	-1	0	unknown	no
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	1	-1	0	unknown	no
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	11	-1	0	unknown	no
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	4	211	3	other	no
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	2	249	7	other	no

（2）将 yes/no 分类映射为 0/1 分类

```
class_mapping = {'no':0, 'yes':1}
df['default'] = df['default'].map(class_mapping)
df['housing'] = df['housing'].map(class_mapping)
df['loan'] = df['loan'].map(class_mapping)
df['y'] = df['y'].map(class_mapping)
```

	y	default	housing	loan
0	no	no	no	no
1	no	no	yes	yes
2	no	no	yes	no
3	no	no	yes	yes
4	no	no	yes	no
...	...	...	...	...
4516	no	no	yes	no
4517	no	yes	yes	yes
4518	no	no	no	no
4519	no	no	no	no
4520	no	no	yes	yes

	y	default	housing	loan
0	0	0	0	0
1	0	0	1	1
2	0	0	1	0
3	0	0	1	1
4	0	0	1	0
...	...	...	...	...
4516	0	0	1	0
4517	0	1	1	1
4518	0	0	0	0
4519	0	0	0	0
4520	0	0	1	1

(3) 将月份和日期转化成距离现在的天数

```
month_dict = {'oct': '10', 'may': '05', 'apr': '04', 'jun': '06', 'feb': '02', 'aug': '08',
              'jan': '01', 'jul': '07', 'nov': '11','sep': '09', 'mar': '03', 'dec': '12'}
df['month'] = df['month'].map(month_dict)
df['date'] = '2019' + '-' + df['month'] + '-' + df['day']
df['date'] = pd.to_datetime(df['date'], format="%Y-%m-%d")
df['date'] = pd.to_datetime('2020-01-01', format="%Y-%m-%d") - df['date']
df['date'] = df['date'].dt.days
print(df[['date']])
```

```
      date
0       74
1      235
2      260
3      212
4      241
...    ...
4516   155
4517   237
4518   135
4519   329
4520   273
```

(4) 删除原有的月份与日期

```
del(df['day'])
del(df['month'])
```

(5) 将字符型分类变量转化为数值型并处理缺失值

```
jobs = df['job'].unique()
job_mapping = {jobs[i]:i for i in range(jobs.shape[0])}
maritals = df['marital'].unique()
marital_mapping = {maritals[i]:i for i in range(maritals.shape[0])}

educations = df['education'].unique()
education_mapping = {educations[i]:i for i in range(educations.shape[0])}

contacts = df['contact'].unique()
contact_mapping = {contacts[i]:i for i in range(contacts.shape[0])}

poutcomes = df['poutcome'].unique()
poutcome_mapping = {poutcomes[i]:i for i in range(poutcomes.shape[0])}

df['marital'] = df['marital'].map(marital_mapping)
df['job'] = df['job'].map(job_mapping)
df['education'] = df['education'].map(education_mapping)
df['contact'] = df['contact'].map(contact_mapping)
df['poutcome'] = df['poutcome'].map(poutcome_mapping)
```

```
      age  job  marital  education  default  balance  housing  loan  contact  \
0      30    0        0          0        0     1787        0     0        0
1      33    1        0          1        0     4789        1     1        0
2      35    2        1          2        0     1350        1     0        0
3      30    2        0          2        0     1476        1     1        1
4      59    3        0          1        0        0        1     0        1
...   ...  ...      ...        ...      ...      ...      ...   ...      ...
4516   33    1        0          1        0     -333        1     0        0
4517   57    4        0          2        1    -3313        1     1        1
4518   57    5        0          1        0      295        0     0        0
4519   28    3        0          1        0     1137        0     0        0
4520   44    6        1          2        0     1136        1     1        0

      duration  campaign  pdays  previous  poutcome  y  date
0           79         1     -1         0         0  0    74
1          220         1    339         4         1  0   235
2          185         1    330         1         1  0   260
3          199         4     -1         0         0  0   212
4          226         1     -1         0         0  0   241
...        ...       ...    ...       ...       ... ..   ...
4516       329         5     -1         0         0  0   155
4517       153         1     -1         0         0  0   237
4518       151        11     -1         0         0  0   135
4519       129         4    211         3         2  0   329
4520       345         2    249         7         2  0   273

[4521 rows x 16 columns]
```

（6）将连续性属性离散化

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。

```
bins = [18, 25, 35, 45, 55, 100]   # 指定年龄的分界点
df['age'] = pd.cut(df['age'], bins, labels=False)
bins = [-np.inf, 4137.1, 11587.2, np.inf]
df['balance'] = pd.cut(df['balance'], bins, labels=False)
print(df[['age','balance']])
```

```
      age  balance
0       1        0
1       1        1
2       1        0
3       1        0
4       4        0
...   ...      ...
4516    1        0
4517    4        0
4518    4        0
4519    1        0
4520    2        0
```

- 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

```
y = df['y']
df.drop(labels=['y'], axis=1,inplace = True)
df.insert(16, 'y', y)
df.to_csv('D:/大数据实验/实验二/after_bank.csv')
```

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**，解决银行精准营销的分类问题。**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） 不允许调用 sklearn 等现成库

（4） 将预处理后的数据自行划分成训练集和测试集，采用交叉验证可作为加分项。

（5） 输入训练数据和相关参数（如树的最大深度等）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训练集和测试集的划分方法，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

5 补充内容：安装 sklearn 并构建决策树

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)

（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/

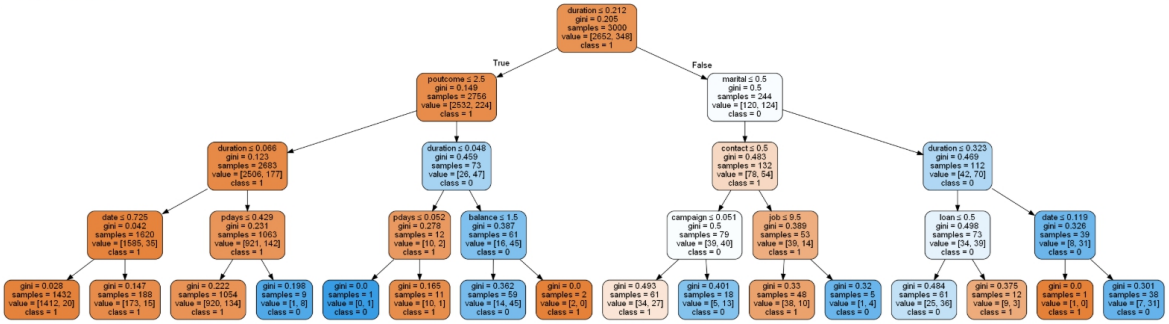
（2）安装 pydotplus

[redacted]

（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 re cognized. Use one of:

可用管理员身份运行 cmd，执行