



Using Machine Learning Models

数据科学 –
机器学习模型入门

房贷放款评估
使用 **Logistical Regression**

Nov 2020

Microsoft Reactor | Ryan Chung

```
led by player to  
s.load_image("kg.png")  
(self):  
    initialize Dog object and create Text o  
g, self).__init__(image = Dog.image  
x = games.mouse.x  
bottom = games.sc  
re = games.Text(value = 0, size = 24  
top = 5, right = gam  
reen.add(self.score)  
1 = games.Text(value = 0, size = 24  
top = 5, left = gam
```



Ryan Chung

Instructor / DevelopIntelligence
Founder / MobileDev.TW

@ryanchung403 on WeChat
Ryan@MobileDev.TW





Reactor



developer.microsoft.com/reactor/
@MSFTReactor on Twitter

房贷放款评估

- 评估客户是否符合贷款标准
 - 性别
 - 婚姻状况
 - 教育程度
 - 收入
 - 借贷金额
 - 信用记录

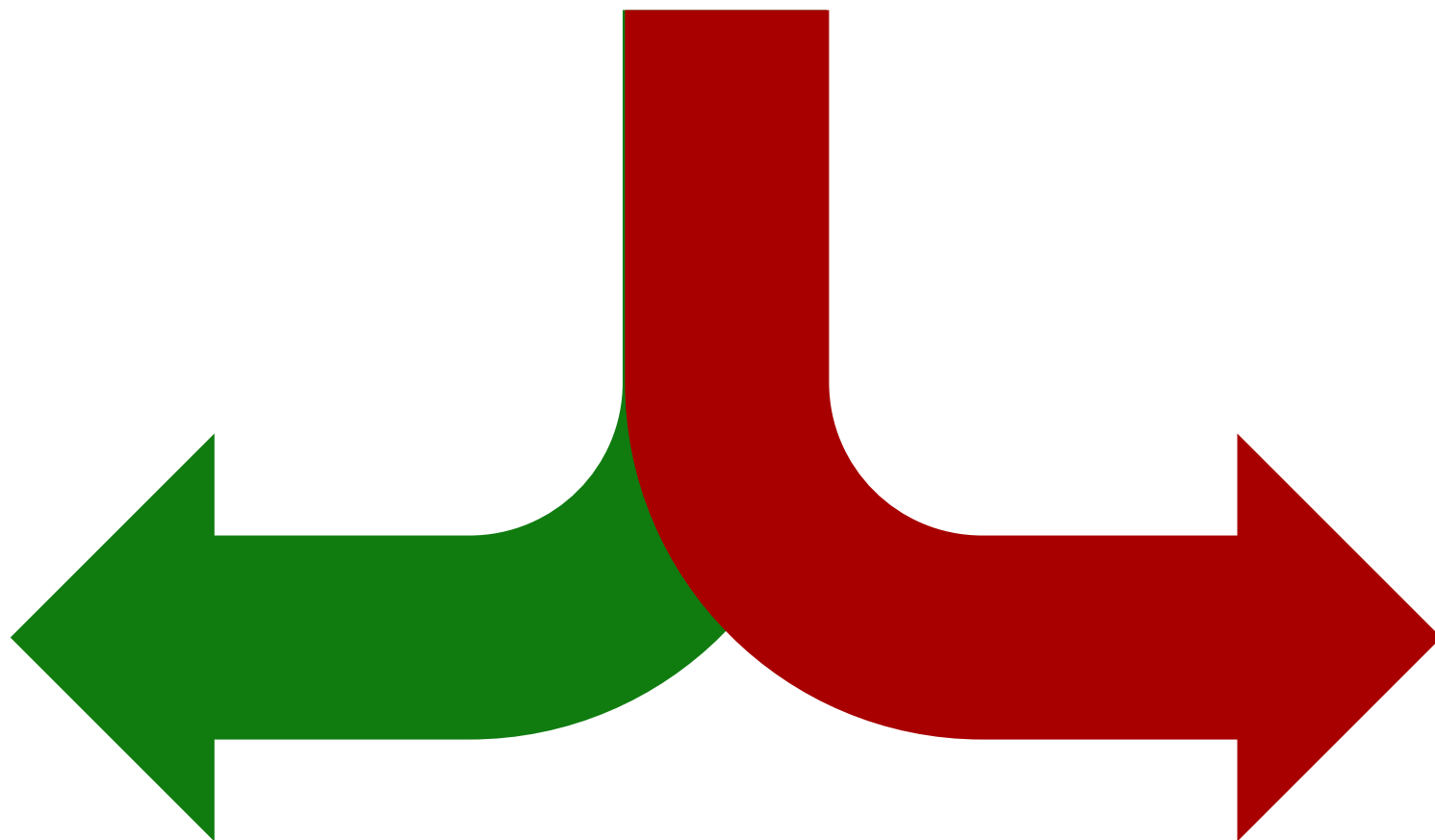


数据说明

变数名称	说明	变数名称	说明
Loan_ID	唯一识别ID	CoapplicantIncome	共同申请人收入
Gender	性别(Male/Female)	LoanAmount	借贷金额(美金千元)
Married	是否已婚 (Y/N)	Loan_Amount_Term	借贷时间(月)
Dependents	家属人数	Credit_History	信用记录(1/0)
Education	教育程度 (Graduate/ Under Graduate)	Property_Area	房产位置 Urban/ Semi Urban/ Rural
Self_Employed	是否为自雇者 (Y/N)	Loan_Status	是否核准借贷 (Y/N)
ApplicantIncome	申请者本人收入		

目标

- 建立机器学习模型，决定是否要核准贷款
 - Yes
 - No



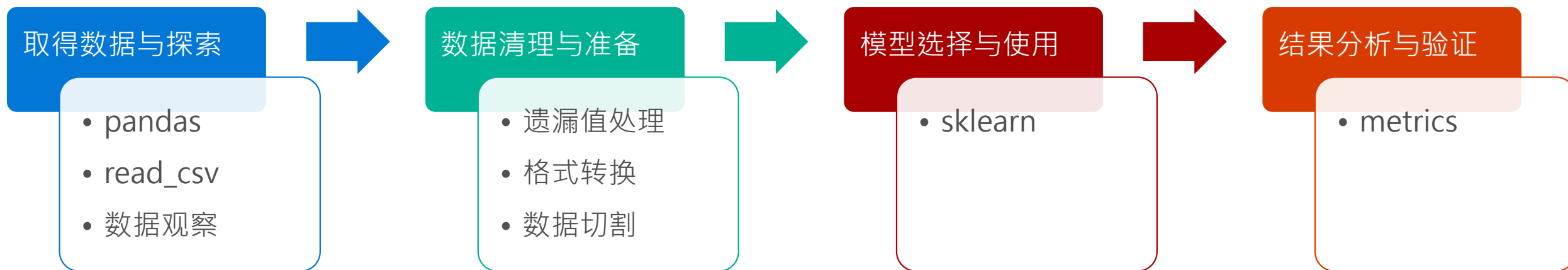
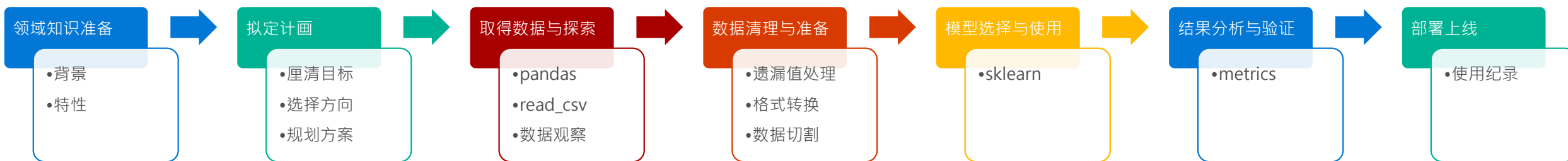
数据科学家的职业道德

- 只搜集必要的、分析需要的数据
- 界定与去除机敏性数据
- 判断错误的备援方案准备

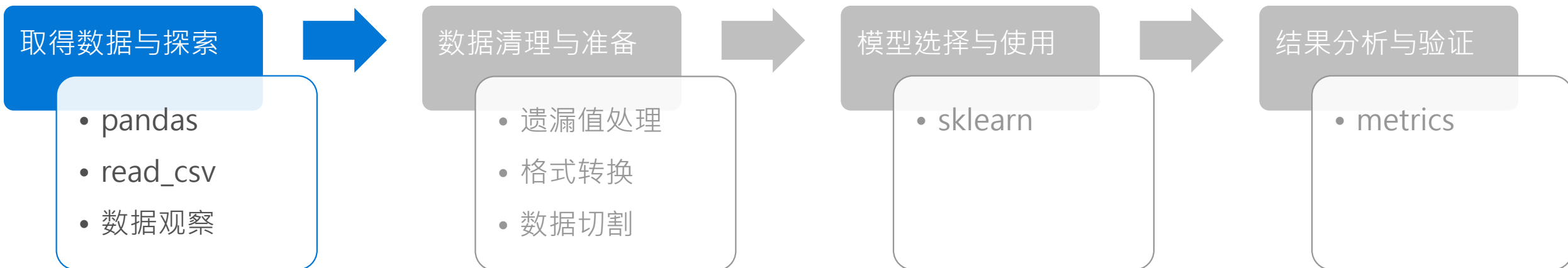
性别与婚姻状态

- 属于个人隐私数据，可考虑去除

数据科学处理流程



数据科学处理流程

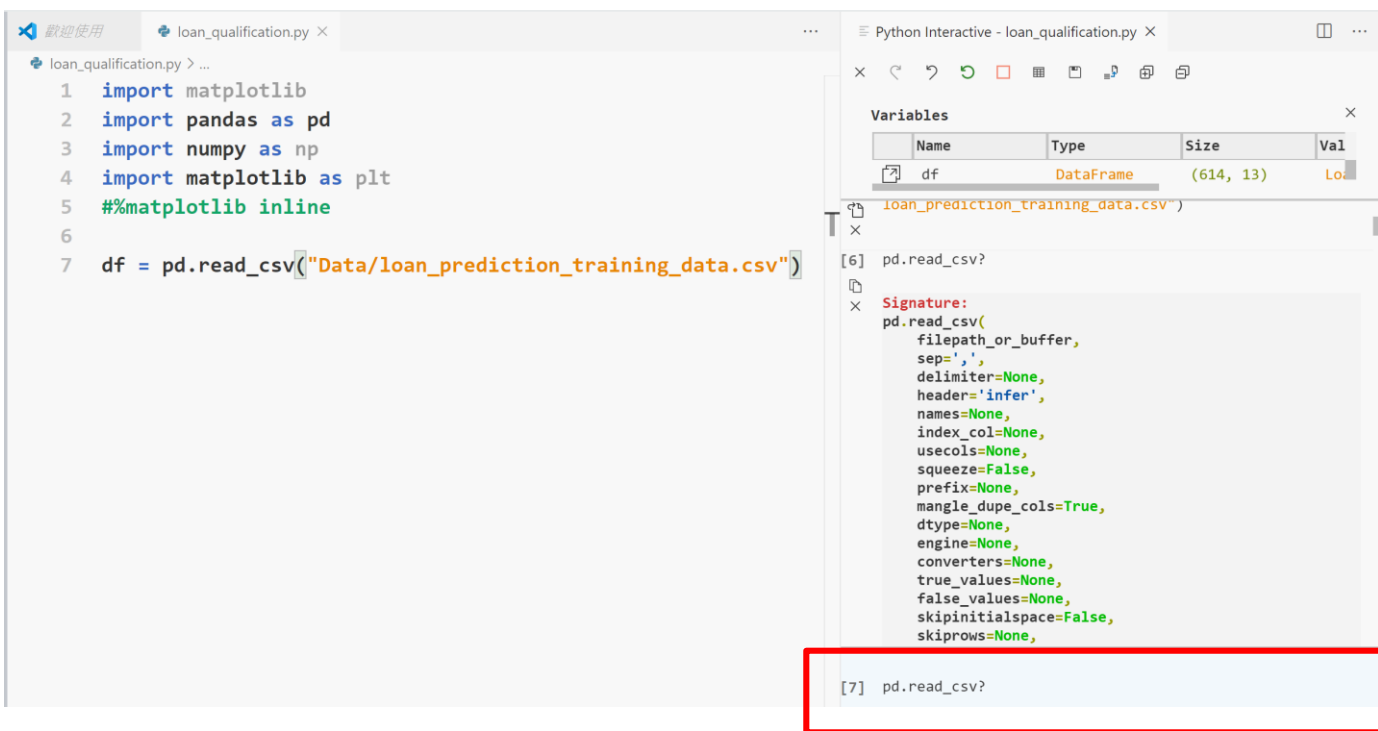


```
import matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#%matplotlib inline
```

```
df = pd.read_csv("Data/loan_prediction_training_data.csv")
```

查阅方法说明

- 在VS Code中，可在右下方区块输入方法名称，最后加上问号，即可获得说明
- 例如：`pd.read_csv?`



```
[6] pd.read_csv?
```

Signature:

```
pd.read_csv(  
    filepath_or_buffer,  
    sep=',',  
    delimiter=None,  
    header='infer',  
    names=None,  
    index_col=None,  
    usecols=None,  
    squeeze=False,  
    prefix=None,  
    mangle_dupe_cols=True,  
    dtype=None,  
    engine=None,  
    converters=None,  
    true_values=None,  
    false_values=None,  
    skipinitialspace=False,  
    skiprows=None,
```

继续探索数据 df.describe()

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

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



	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

Q.从这里可得知
哪几项有缺失值?

继续探索数据 df.describe()

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

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



	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

Q. Credit_History
为1的有几笔?

继续探索数据 df.info()

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
[8] df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID           614 non-null object
Gender            601 non-null object
Married           611 non-null object
Dependents        599 non-null object
Education         614 non-null object
Self_Employed    582 non-null object
ApplicantIncome   614 non-null int64
CoapplicantIncome 614 non-null float64
LoanAmount        592 non-null float64
Loan_Amount_Term  600 non-null float64
Credit_History   564 non-null float64
Property_Area     614 non-null object
Loan_Status       614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

共有614笔资料

去除性别、婚姻数据

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

#剔除Gender, Married栏位与资料

```
df_no_G_M = df.drop(columns=['Gender', 'Married'])
```

#存成csv档

```
df_no_G_M.to_csv('loan_prediction_training_data_no_G_M.csv')
```

[13] df.head()



	Loan_ID	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	0	Graduate	No	6000	0.0	141.0	360.0	1.0

调阅数据

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['Self_Employed'].value_counts()  
df['Property_Area'].value_counts()  
df['Education'].value_counts()
```

```
[22] df['Self_Employed'].value_counts()
```



```
No      500  
Yes      82  
Name: Self_Employed, dtype: int64
```

```
[23] df['Property_Area'].value_counts()
```



```
Semiurban    233  
Urban        202  
Rural        179  
Name: Property_Area, dtype: int64
```

```
[24] df['Education'].value_counts()
```



```
Graduate      480  
Not Graduate  134  
Name: Education, dtype: int64
```

调阅数据- 收入分布情形

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

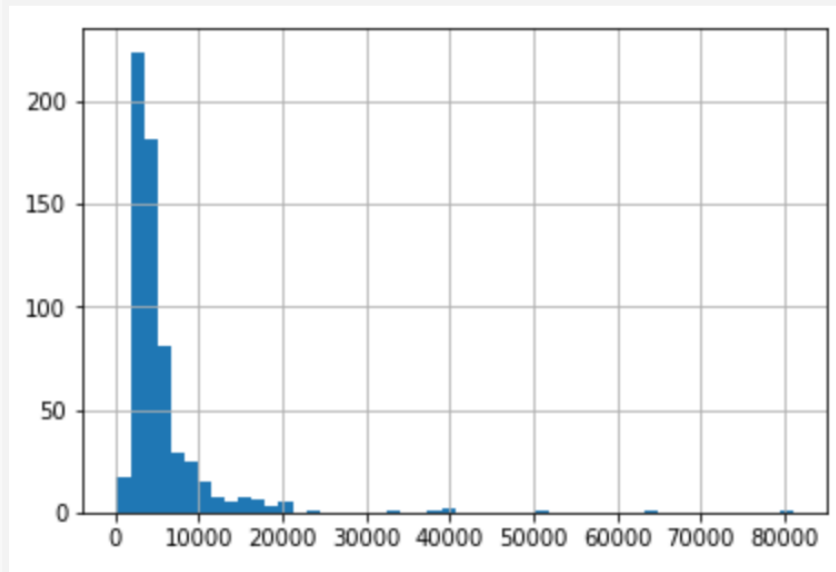
- metrics

```
df['ApplicantIncome'].hist(bins=50)
```

```
[28] df['ApplicantIncome'].hist(bins=50)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x20c2bec9f60>
```



调阅数据- 收入分布情形 – 换一种图试试

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

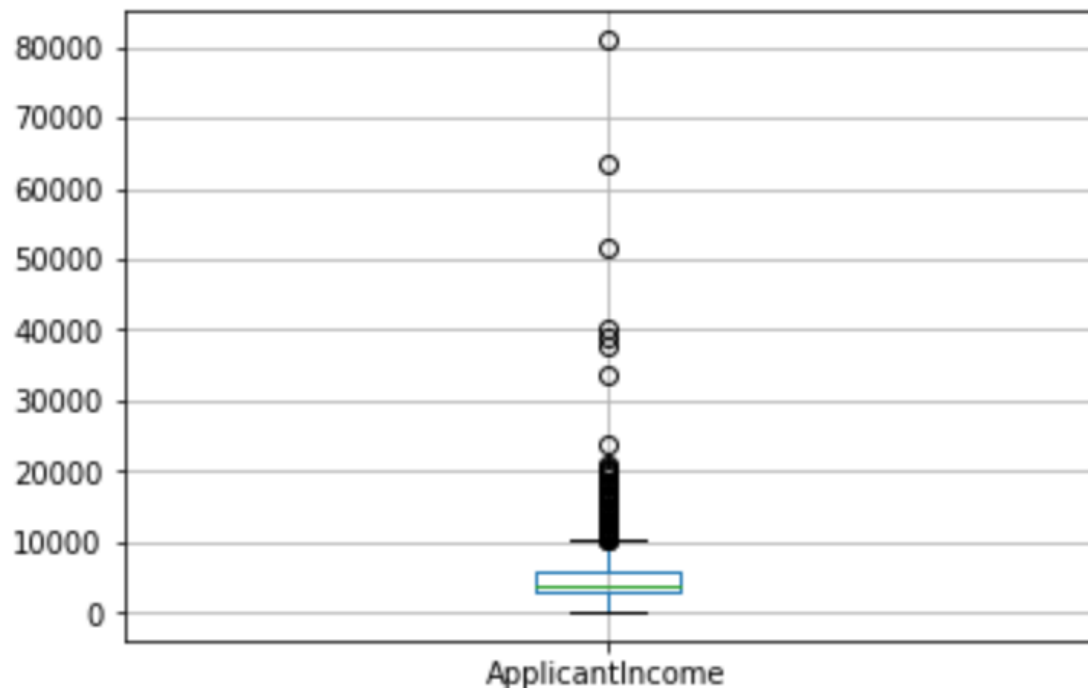
- sklearn

结果分析与验证

- metrics

```
df.boxplot(column='ApplicantIncome')
```

Q. 看来有蛮多收入特别高的人
跟教育程度有没有关联性呢?



调阅数据- 收入分布情形 – 换一种图试试

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

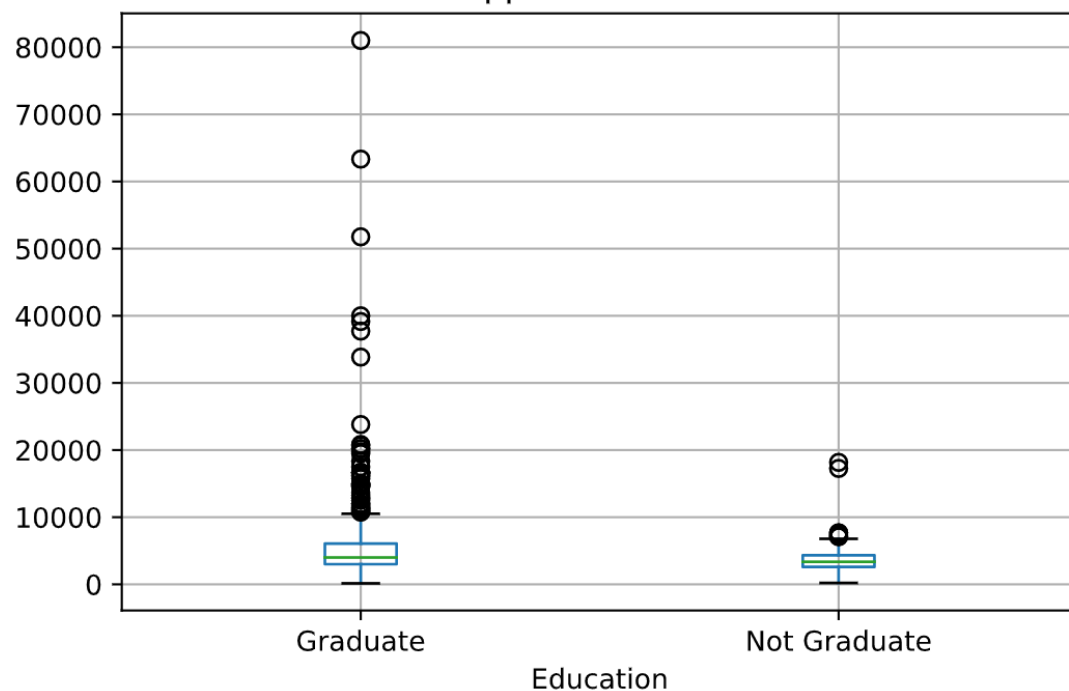
- metrics

```
df.boxplot(column='ApplicantIncome', by = 'Education')
```

Boxplot grouped by Education
ApplicantIncome

Q. 看来有蛮多收入特别高的人
跟教育程度有没有关联性呢?

几乎都在有毕业的那一边!



调阅数据- 借贷金额分布情形

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

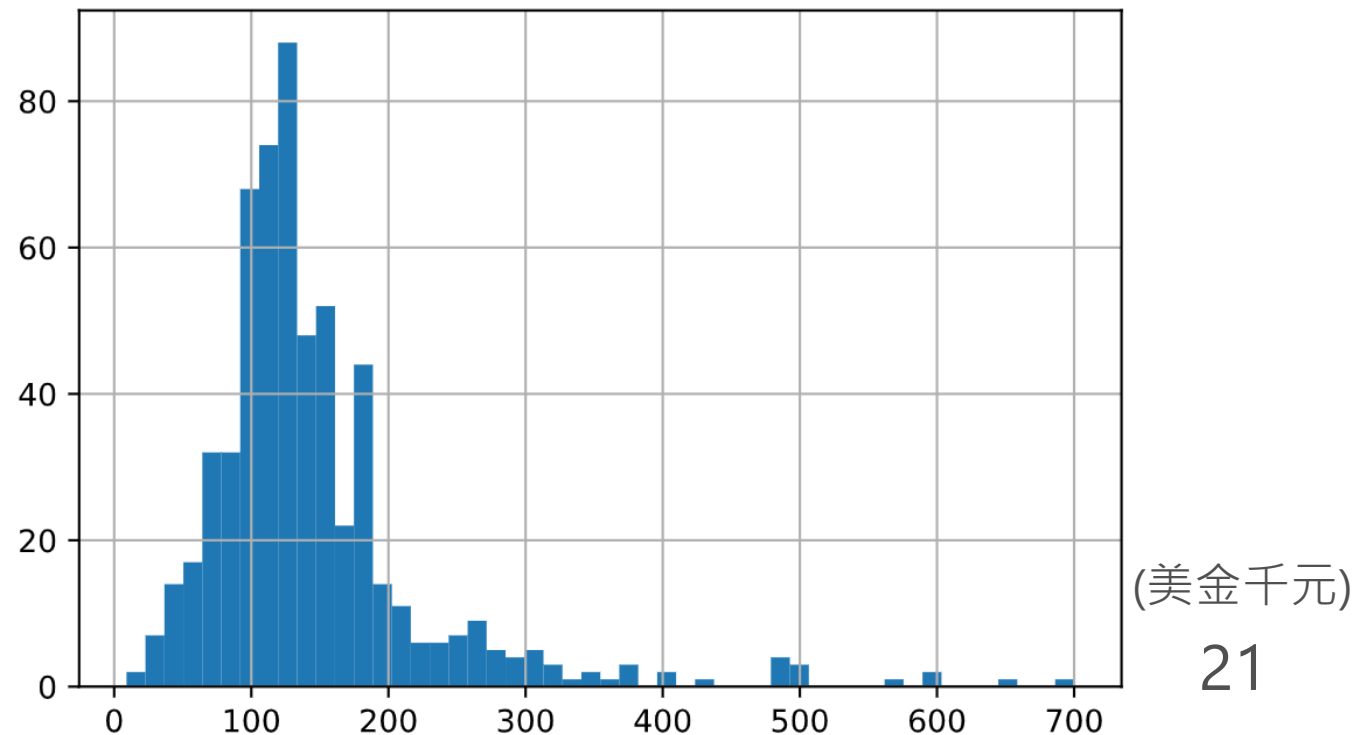
模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['LoanAmount'].hist(bins=50)
```



调阅数据-借贷金额分布情形 – 换一种图

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

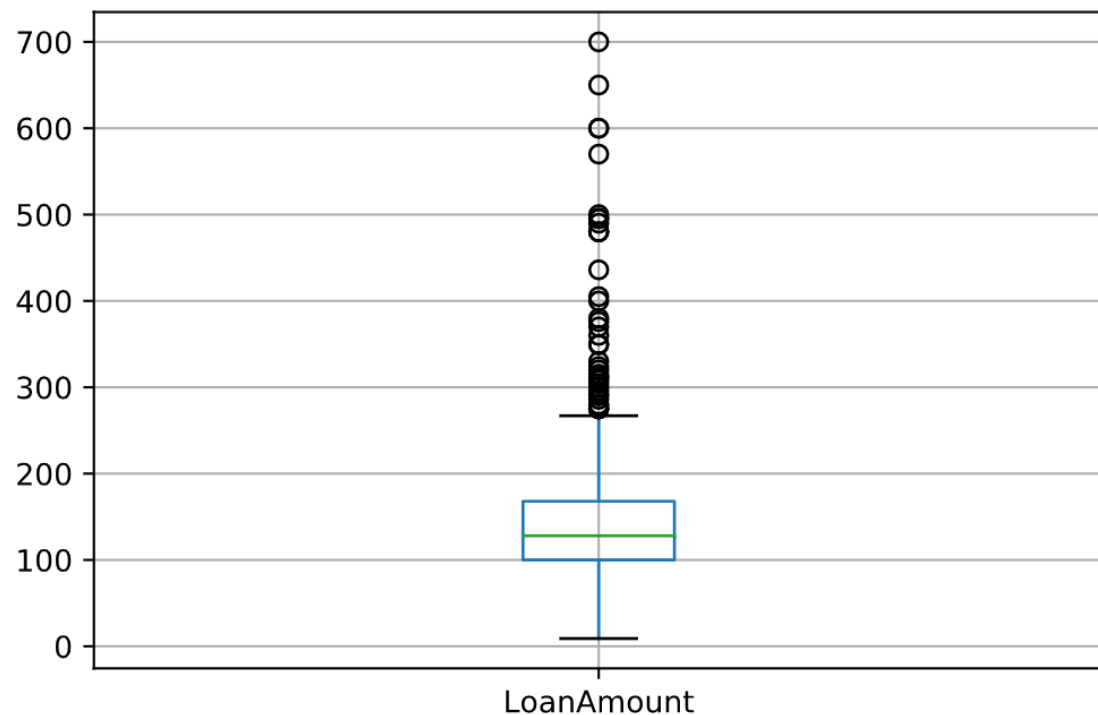
- sklearn

结果分析与验证

- metrics

```
df.boxplot(column='LoanAmount')
```

Q. 也有蛮多借贷金额特别高的人
跟教育程度有没有关联性呢?



调阅数据-借贷金额分布情形 – 换一种图

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

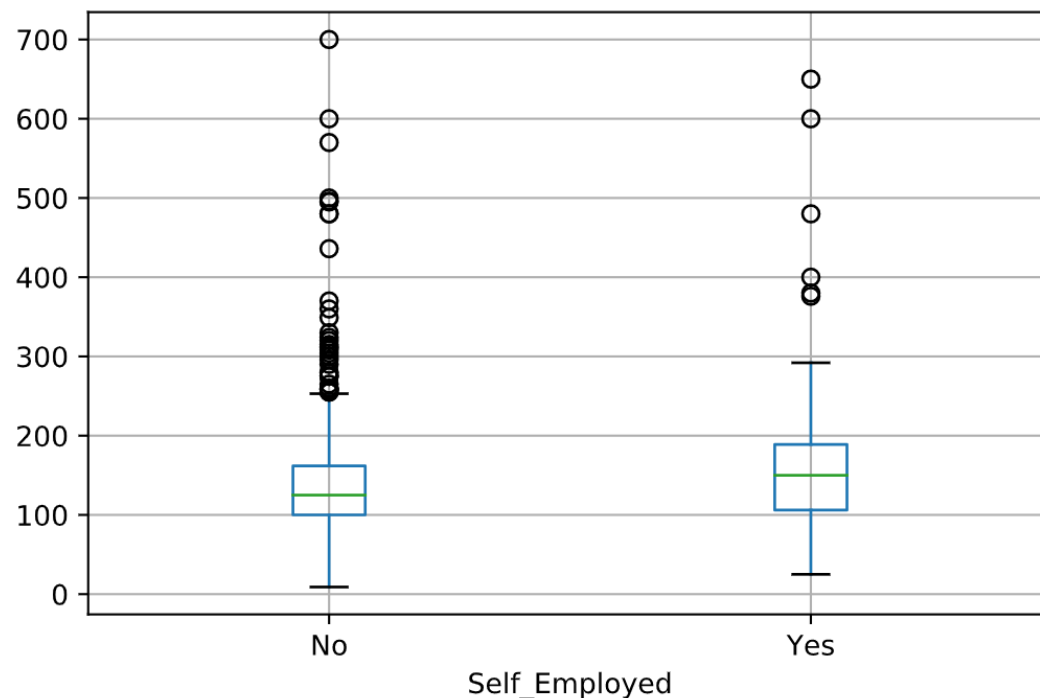
练习

借贷金额高低与是否为自营商的关联

收入高低与是否为自营商的关联

...

Boxplot grouped by Self_Employed
LoanAmount



调阅数据- 信用记录 VS. 借贷状态

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
temp1 = df['Credit_History'].value_counts(ascending=True)
```

```
temp2 = df.pivot_table(values='Loan_Status', index=['Credit_History'], aggfunc=lambda x: x.map({'Y':1, 'N':0}).mean())
```

有信用记录的借贷成功比例高很多！

```
[45] temp1
```

0.0	89
1.0	475

Name: Credit_History, dtype: int64

```
[47] temp2
```

	Loan_Status
Credit_History	
0.0	0.078652
1.0	0.795789

调阅数据- 信用记录 VS. 借贷状态(视觉化)

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

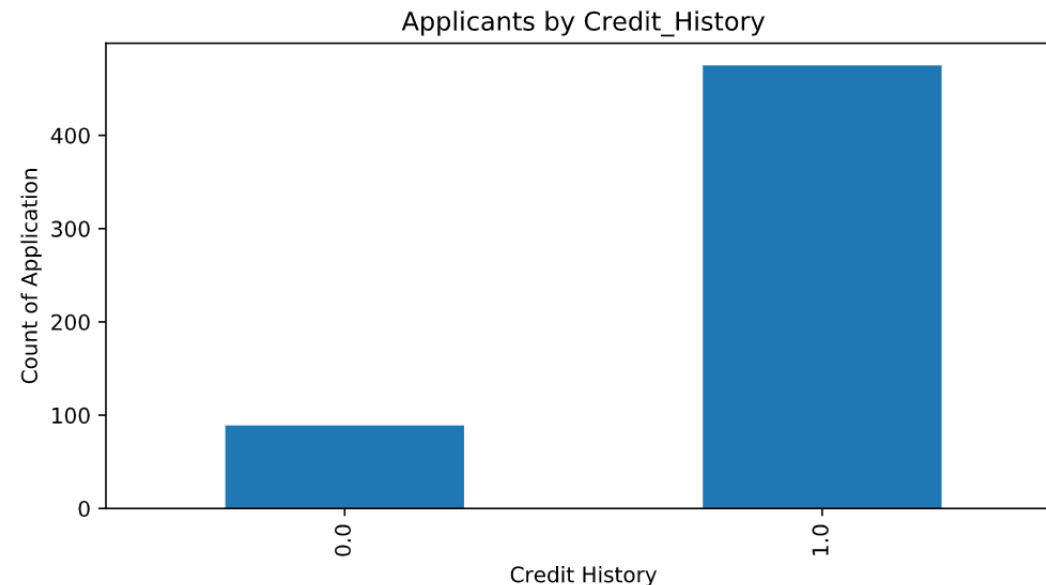
模型选择与使用

- sklearn

结果分析与验证

- metrics

```
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(111)
ax1.set_xlabel('Credit History')
ax1.set_ylabel('Count of Application')
ax1.set_title('Applicants by Credit_History')
temp1.plot(kind = 'bar')
```



调阅数据- 信用记录 VS. 借贷状态(视觉化)

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

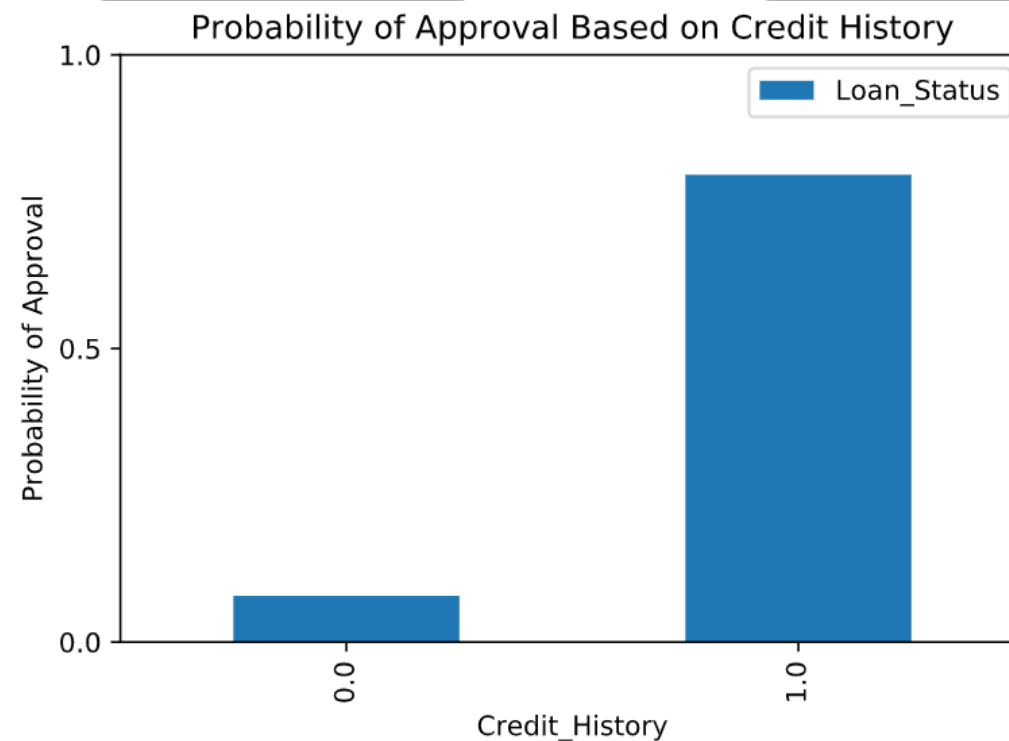
模型选择与使用

- sklearn

结果分析与验证

- metrics

```
temp2.plot(kind = 'bar',yticks=[0,0.5,1],  
           ylabel='Probability of Approval',title=  
           'Probability of Approval Based on Credit  
           History')
```



调阅数据- 房产位置 VS. 借贷状态

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

练习

请用相同方式

观察房产位置与借贷状态是否有关连性

调阅数据- 自雇者 VS. 借贷状态

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

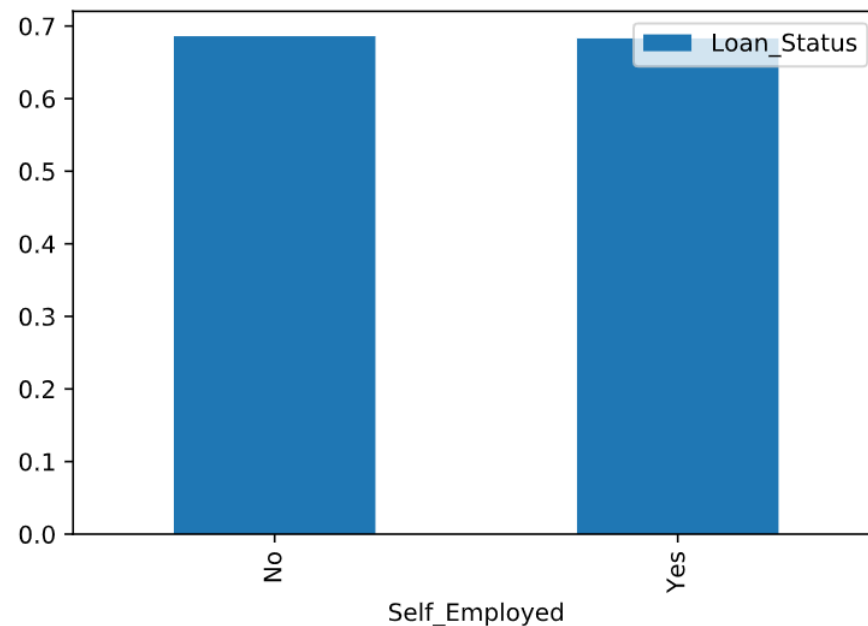
结果分析与验证

- metrics

练习

请用相同方式

观察自雇者与借贷状态是否有关连性



调阅数据- 信用记录 VS. 借贷状态

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

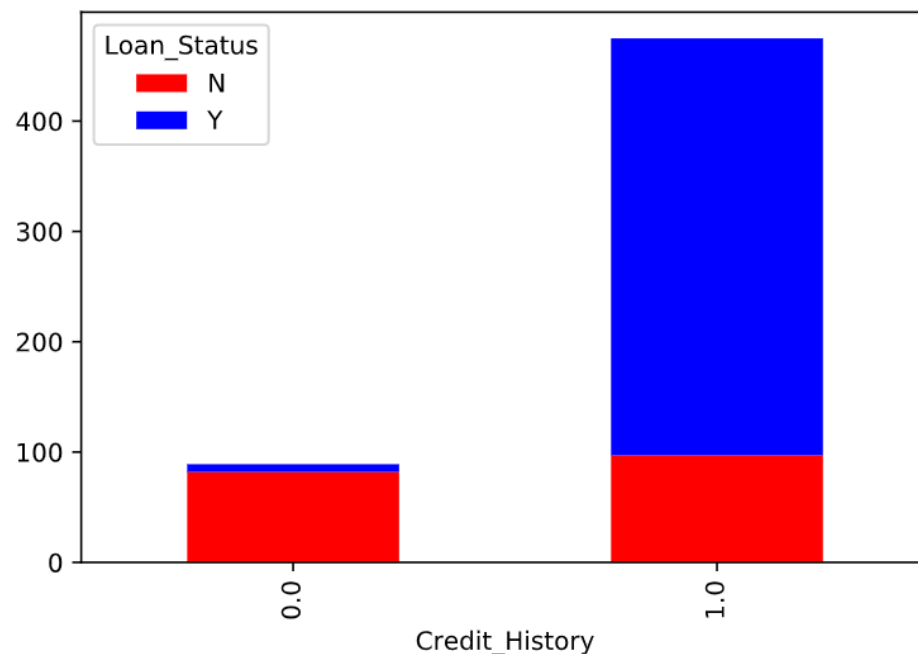
- sklearn

结果分析与验证

- metrics

换一种呈现方式！

```
temp5 = pd.crosstab(df['Credit_History'], df['Loan_Status'])
temp5.plot(kind='bar', stacked=True, color=['red', 'blue'], grid=False)
```



调阅数据- 信用记录/性别 VS. 借贷状态

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

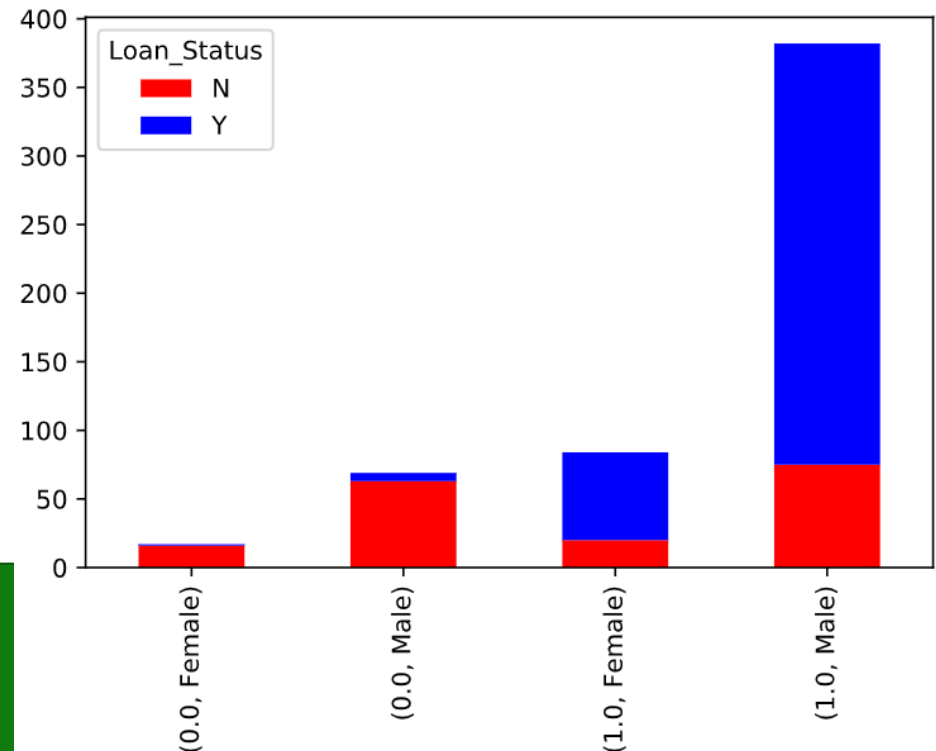
模型选择与使用

- sklearn

结果分析与验证

- metrics

```
temp6 = pd.crosstab([df['Credit_History'], df['Gender']], df['Loan_Status'])
temp6.plot(kind='bar', stacked=True, color=['red', 'blue'])
```



男性&有信用记录的，借贷核准机会最大！
注意：有使用到性别栏位，前面若有删除需重新载入数据

调阅数据- 遗漏值综览

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0
dtype: int64	

遗漏值最多的是：信用记录、是否为自雇者、借贷金额

处理借贷金额的遗漏值 – 使用平均值

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(),  
inplace=True)  
df.apply(lambda x: sum(x.isnull()),axis=0)
```

Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0
dtype:	int64

填补完成后，再次查阅遗漏值

处理借贷金额的遗漏值I – 查看填补值

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(),  
inplace=True)
```

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

```
df['LoanAmount'].value_counts()
```

```
146.412162    22  
120.000000    20  
110.000000    17  
100.000000    15  
160.000000    12  
..  
570.000000     1  
300.000000     1  
376.000000     1  
117.000000     1  
311.000000     1  
Name: LoanAmount, Length: 204, dtype: int64
```

填补完成后，再次查阅遗漏值

处理是否为自雇者的遗漏值 – 使用多数

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['Self_Employed'].value_counts()  
print(500/(500+82))
```

```
df['Self_Employed'].value_counts()
```

No	500
Yes	82

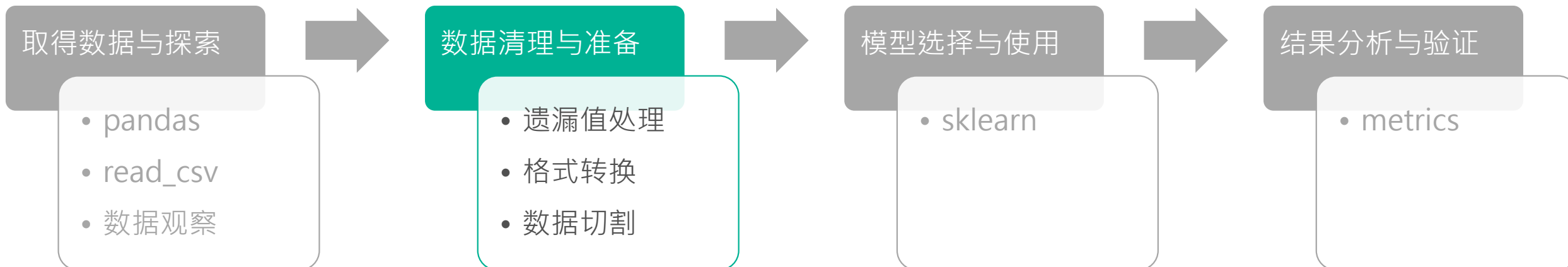
Name: Self_Employed, dtype: int64

```
print(500/(500+82))
```

```
0.8591065292096219
```

非自雇者的比例为85.9 %

处理是否为自雇者的遗漏值 – 使用多数



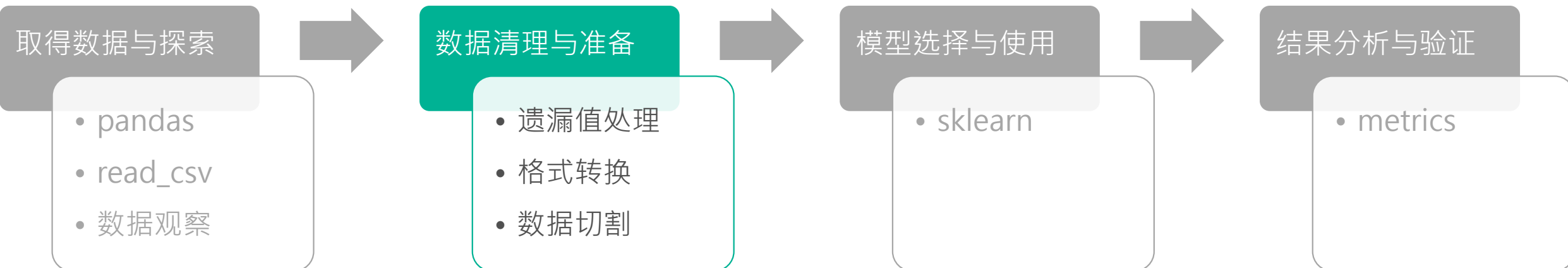
```
df['Self_Employed'].value_counts()  
print(500/(500+82))
```

```
df['Self_Employed'].fillna("No", inplace=True)
```

Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0
dtype:	int64

所以使用No来填补是否为自雇者的遗漏值

处理借贷金额的遗漏值II – 取得个别情况的中位数



```
table =  
df.pivot_table(values='LoanAmount', index='Self_Employed', columns='Education', aggfunc  
=np.median)  
def fage(x):  
    return table.loc[x['Self_Employed'], x['Education']]  
  
df['LoanAmount'].fillna(df.apply(fage, axis=1), inplace=True)
```

依是否毕业、是否为自雇者分成四类，算出个别中位数

使用该中位数来填补

注意：需先确认要用到的**Self_Employed**、**Education**已无遗漏值

注意二：实作时记得先将方法一(用平均值填补借贷金额遗漏值)还原

	Education	Graduate	Not Graduate
Self_Employed			
No		130.0	113.0
Yes		157.5	130.0

借贷金额的观察 – 取对数

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

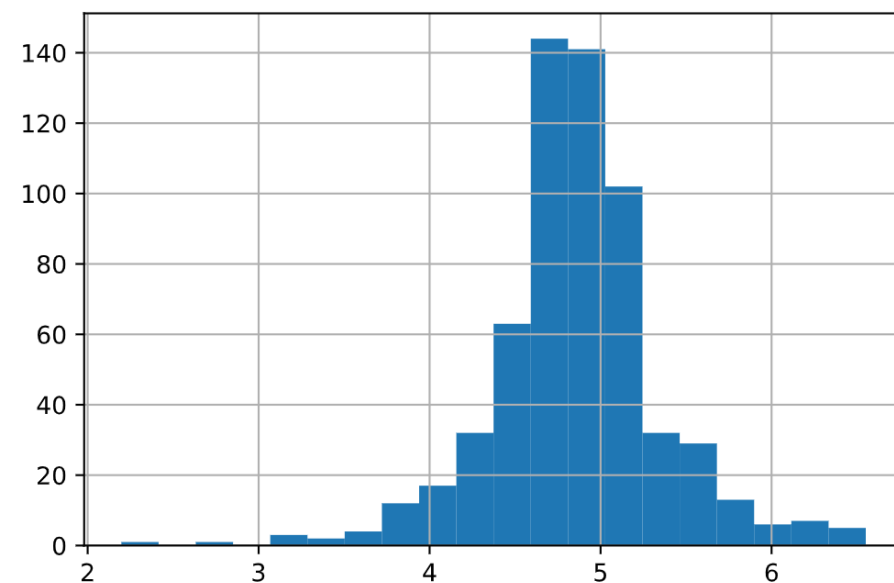
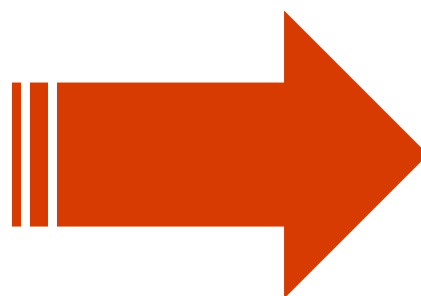
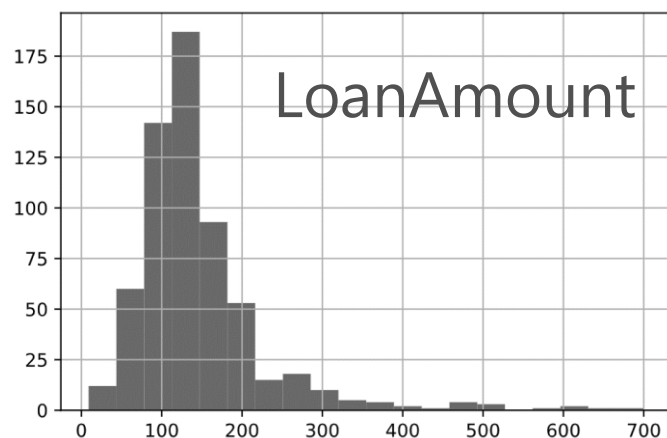
模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['LoanAmount_log'] = np.log(df['LoanAmount'])  
df['LoanAmount_log'].hist(bins=20)
```



透过对数的转换来处理异常值，而非删除

LoanAmount_log

申请者本人收入 + 共同申请者收入

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

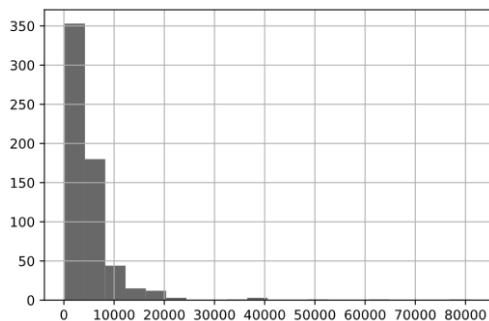
模型选择与使用

- sklearn

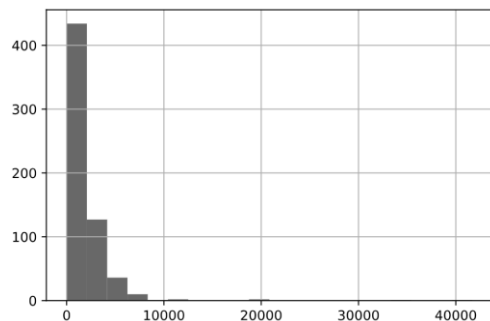
结果分析与验证

- metrics

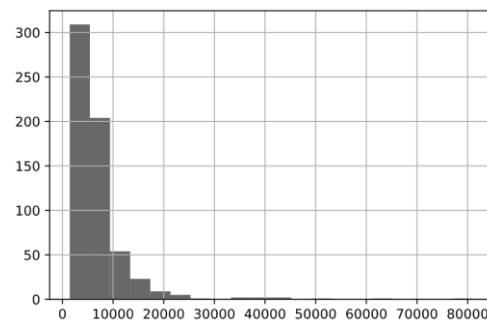
```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']  
df['TotalIncome_log'] = np.log(df['TotalIncome'])  
df['TotalIncome_log'].hist(bins=20)
```



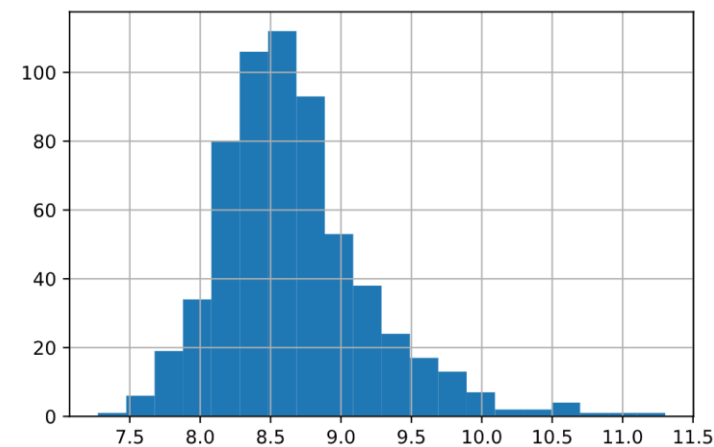
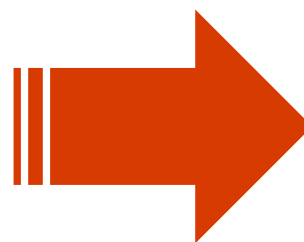
ApplicantIncome



CoapplicantIncome



TotalIncome



TotalIncome_log

透过对数的转换来处理异常值，而非删除

遗漏值填补：剩下的都用最高频率值填补

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

```
df.apply(lambda x: sum(x.isnull()), axis=0)
```

Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0
LoanAmount_log	0
TotalIncome	0
TotalIncome_log	0
dtype:	int64

一个集合的mode就是最常出现的值，可能回传多个所以取第0个



将非数值转换为数值

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

df.dtypes

```
from sklearn.preprocessing import LabelEncoder
```

```
var_mod = ['Gender', 'Married', 'Dependents', 'Education',  
           'Self_Employed', 'Property_Area', 'Loan_Status']
```

```
le = LabelEncoder()
```

```
for i in var_mod:
```

```
    df[i] = le.fit_transform(df[i])
```

```
df.dtypes
```

Loan_ID	object
Gender	object
Married	object
Dependents	object
Education	object
Self_Employed	object
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	object
Loan_Status	object
LoanAmount_log	float64
TotalIncome	float64
TotalIncome_log	float64
dtype:	object

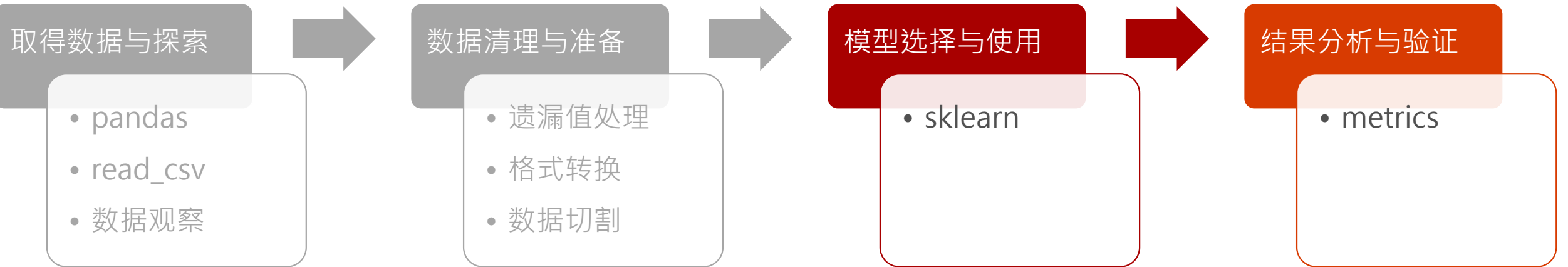


Loan_ID	object
Gender	int32
Married	int32
Dependents	int32
Education	int32
Self_Employed	int32
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	int32
Loan_Status	int32
LoanAmount_log	float64
TotalIncome	float64
TotalIncome_log	float64
dtype:	object

fit_transform() : 回传处理好的数值



使用LogisticRegression



```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

```
def loan_model(model, data, predictors, outcome):
    model.fit(data[predictors], data[outcome])
    predictions = model.predict(data[predictors])
    accuracy = metrics.accuracy_score(predictions, data[outcome])
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
    model.fit(data[predictors], data[outcome])
```

```
outcome_var = 'Loan_Status'
model = LogisticRegression()
predictor_var = ['Credit_History']
loan_model(model, df, predictor_var, outcome_var)
```

先只用信用记录来进行训练

```
[100] outcome_var = 'Loan_Status'...
Accuracy : 80.945%
```

使用DecisionTree试试

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

```
outcome_var = 'Loan_Status'
```

```
model2 = DecisionTreeClassifier()
```

```
predictor_var2 = ['Credit_History']
```

```
loan_model(model2, df, predictor_var2, outcome_var)
```

结果相同

```
[106] outcome_var = 'Loan_Status'...
```



Accuracy : 80.945%

多加几个预测参数试试

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
outcome_var = 'Loan_Status'  
model = LogisticRegression()  
predictor_var = ['Credit_History', 'Gender', 'Married', 'Education']  
loan_model(model, df, predictor_var, outcome_var)
```

结果相同

```
[109] outcome_var = 'Loan_Status'...
```



Accuracy : 80.945%

再换一个模型

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
outcome_var = 'Loan_Status'
```

```
model3 = RandomForestClassifier(n_estimators=10)
```

```
predictor_var3 = ['Credit_History', 'Gender', 'Married', 'Education']
```

```
loan_model(model3, df, predictor_var3, outcome_var)
```

结果相同

```
[118] outcome_var = 'Loan_Status'...
```



Accuracy : 80.945%

再多加几个参数，包含调整过的参数

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
outcome_var = 'Loan_Status'
model2 = DecisionTreeClassifier()
predictor_var2 = ['Gender', 'Married', 'Dependents',
                  'Education', 'Self_Employed', 'Credit_History', 'Property_Area', 'LoanAmount_log']
loan_model(model2, df, predictor_var2, outcome_var)
```

```
[123] outcome_var = 'Loan_Status'...
✖ Accuracy : 98.208%
```

```
outcome_var = 'Loan_Status'
model3 = RandomForestClassifier(n_estimators=10)
predictor_var3 = ['Gender', 'Married', 'Dependents',
                  'Education', 'Self_Employed', 'Credit_History', 'Property_Area', 'LoanAmount_log']
loan_model(model3, df, predictor_var3, outcome_var)
```

```
[124] outcome_var = 'Loan_Status'...
✖ Accuracy : 97.557%
```

终于有了较佳的结果!

将数据分成测试与训练

取得数据与探索

- pandas
- read_csv
- 数据观察

数据清理与准备

- 遗漏值处理
- 格式转换
- 数据切割

模型选择与使用

- sklearn

结果分析与验证

- metrics

```
from sklearn.model_selection import train_test_split
```

```
def loan_modelv2(model, data, predictors, outcome, t_size, rs_number):
```

```
    X = data[predictors]
```

```
    y = data[outcome]
```

```
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=t_size,random_state=rs_number)
```

```
    model.fit(X_train[predictors],y_train)
```

```
    predictions = model.predict(X_test)
```

```
    accuracy = metrics.accuracy_score(predictions, y_test)
```

```
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
```

```
    model.fit(X_train[predictors],y_train)
```

```
outcome_var = 'Loan_Status'
```

```
model = LogisticRegression()
```

```
predictor_var = ['Gender', 'Education', 'Self_Employed', 'Credit_History', 'Property_Area', 'LoanAmount_log']
```

```
loan_modelv2(model,df,predictor_var,outcome_var,0.3,8)
```

```
[48] outcome_var = 'Loan_Status'...
```



Accuracy : 86.486%

虽然低一些，但比较贴近真实状况

小结

- 成熟的模型未必一定能带来最佳成效，数据的筛选与转换有时才是胜出的关键!
- 多了解各种模型的特性与使用时机，多多实验，累积经验
- 特征工程(Feature Engineering)影响力高，让数据更适合当前的模型!





Reactor



developer.microsoft.com/reactor/
@MSFTReactor on Twitter

议程结束 感谢聆听



请记得填写课程回馈问卷
<https://aka.ms/Reactor/Survey>

© 2019 Microsoft Corporation. All rights reserved. The text in this document is available under the Creative Commons Attribution 3.0 License, additional terms may apply. All other content contained in this document (including, without limitation, trademarks, logos, images, etc.) are not included within the Creative Commons license grant. This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it. Some examples are for illustration only and are fictitious. No real association is intended or inferred. Microsoft makes no warranties, express or implied, with respect to the information provided here.