組別:A

資料來源: 某非營利組織之捐款資料(賀之綸提供)

分工:

data cleaning: B09702113 賀之綸 B10201054 鄭皓中

data visualizing: B09901061 莊政達 B10201051 廖羽翎

data analysis: B10201051 廖羽翎 B09901061 莊政達

making report and presentation: B10201026 許庭瑋

WHAT WE DO

1.data cleaning

2.data visualizing

2.data analysis

- 2-1 Predict whether a person regularly donates
- 2-2 Predict whether a person needs tax deduction

DATA CLEANING:

There are 16 columns in our data, but some of them are not necessary.

ex: 'payment_id' looks like a hash ,'last_status' only has ‘1’ or TRUE

Also ,there are a lot of blanks in our data.

Therefore, we need to invest significant effort in data cleaning.

COLUMNS WE NEED TO CLEAN

'donater_type' : 定期 or not

'uid' : classify person or company

'address' : encode XX縣/市 to int

'date' : YYYY/MM/DD to YYYY and MM

'tax' to 1 or 0

CLEANING:

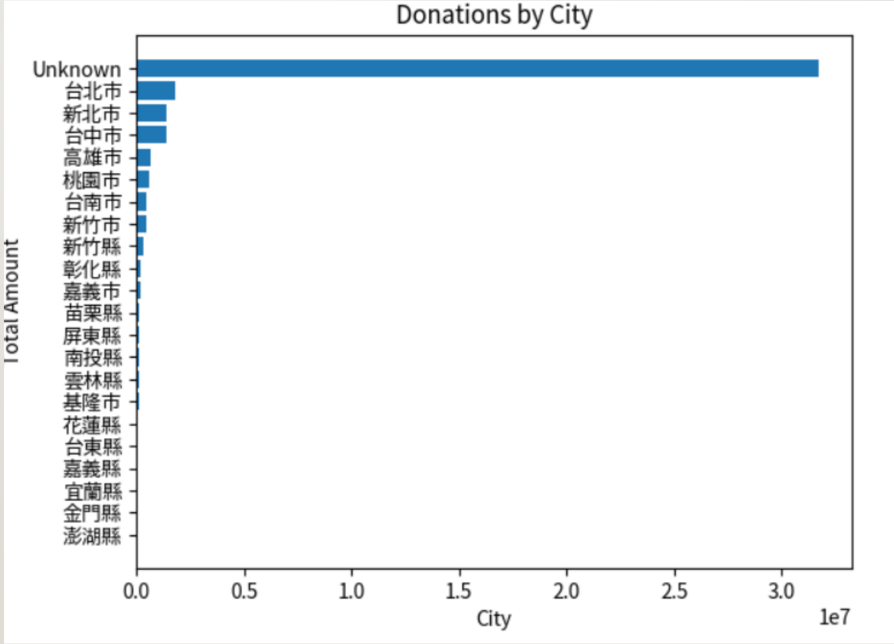
```
def map_donation_type(description):
    if '定期' in description:
        return '1'
    else:
        return '0'

df['donater_type'] = df['payment_type'].apply(lambda x: map_donation_type(x))
```

```
df['date']= pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df.drop(columns=['date'], inplace=True)
```

COLUMNS WITH LOTS OF BLANKS

most of data have no‘city’

so we need to fill in some data.



FILL IN SOME NUMBER TO AVOID NAN

```
valid_cities = data[data['city'] != -1]['city']
city_distribution = valid_cities.value_counts(normalize=True)
# print (city_distribution)

# Create a dictionary to map each donation_id to a city
donation_city_dict = {}
for donation_id in data['donation_id'].unique():
    city = data.loc[data['donation_id'] == donation_id, 'city']
    if all(city == -1):  # If all city values are -1 for this donation_id
        # print('abcdefg')
        # print (city)
        chosen_city = np.random.choice(city_distribution.index, p=city_distribution.values)
        # print(chosen_city)
    else:
        # print ('all city values are not -1')
        # print (city)
        chosen_city = city[city != '-1'].iloc[0]  # Take the first valid city
    donation_city_dict[donation_id] = chosen_city

# print (donation_city_dict)

# Replace '-1' city values in the original DataFrame using the dictionary
data['city'] = data['donation_id'].apply(lambda id: donation_city_dict[id])
```

FIND FEATURES AND COUNT

```
city_list = []
for (key,), group in city_group:
    dict_city = dict()
    dict_city['city'] = key
    dict_city['total_amount'] = group['amount'].sum()
    city_list.append(dict_city)
with open('city_amount.csv', 'w', newline='') as csvfile:
    fieldnames = ['city', 'total_amount']
    writer = csv.DictWriter(csvfile, fieldnames = fieldnames)
    writer.writeheader()
    for i in range(len(city_list)):
        writer.writerow(city_list[i])
```

	city	total_amount
1	-1	31692779
2	01	1796827
3	03	1383597
4	05	481655
5	07	673914
6	11	92289
7	12	433234
8	22	186312
9	31	1399692
10	32	620083
11	33	335883
12	34	37800
13	35	150606
14	37	192311
15	38	124300
16	39	101568
17	40	62348
18	43	147042
19	44	3400
20	45	69432
21	46	64400
22	90	8400

	monthly	total_amount
1	1	4557569
2	2	4352231
3	3	4301392
4	4	4027810
5	5	4239193
6	6	3718236
7	7	1454343
8	8	2628072
9	9	1543392
10	10	2158584
11	11	2985733
12	12	4091317

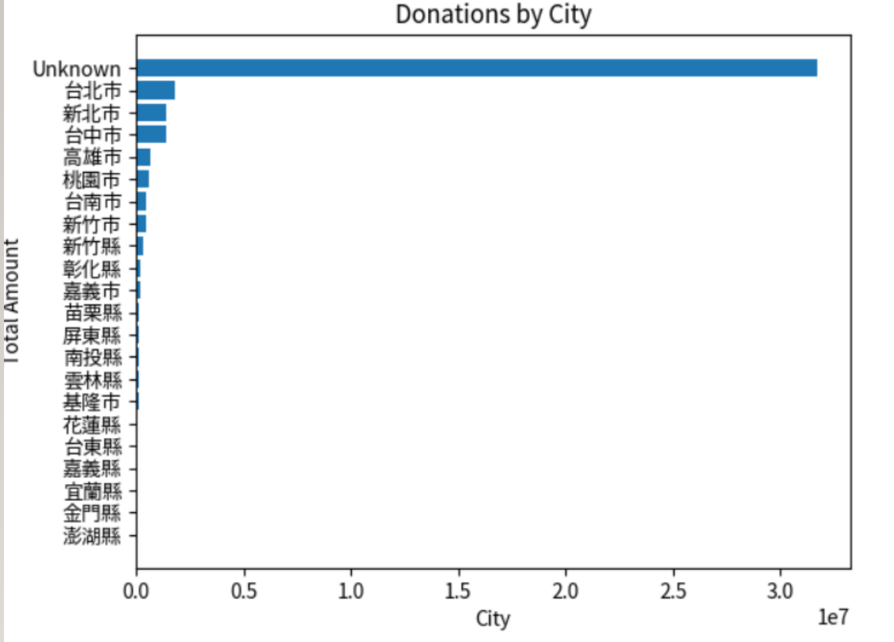
	category	total_amount
1	-1	9868562
2	0	17452720
3	1	11361743
4	2	1374847

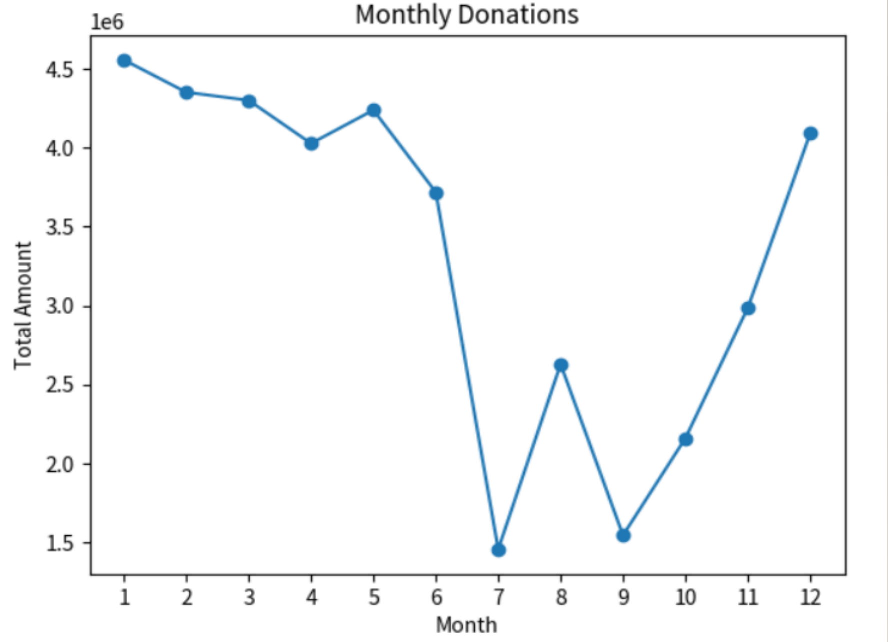
FINALLY WE GET A CLEAN DATA SET

columns:
1.donation_id
2.donater_type
3.category
4.year
5.month
6.amount
7.city
8.tax
9.name

DATA ANALYSIS

draw some graphs and find what we can do





WHAT WE WANT TO DO

1. Predict whether a person regularly donates

2. Predict whether a person needs tax deduction

1.PREDICT WHETHER A PERSON REGULARLY DONATES

methods:

Decision tree , random forest , SVC , XGBoost

determine which method is the best:

find Best ratio Testing Data and Best Parameters

print sklearn.metrics.classification_report

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.44	0.55	1042
1	0.94	0.98	0.96	9023
accuracy			0.92	10065
macro avg	0.82	0.71	0.75	10065
weighted avg	0.91	0.92	0.92	10065

Decision tree

Classification Report:

	precision	recall	f1-score	support
0	0.75	0.32	0.45	1565
1	0.93	0.99	0.96	13533
accuracy			0.92	15098
macro avg	0.84	0.65	0.70	15098
weighted avg	0.91	0.92	0.90	15098

random forest

Classification Report:

	precision	recall	f1-score	suppor
0	0.76	0.19	0.30	540
1	0.91	0.99	0.95	4493
accuracy			0.91	5033
macro avg	0.83	0.59	0.63	5033
weighted avg	0.89	0.91	0.88	5033

SVC

Classification Report:

	precision	recall	f1-score	suppor
0	0.73	0.46	0.56	1042
1	0.94	0.98	0.96	9023
accuracy			0.93	10065
macro avg	0.84	0.72	0.76	10065
weighted avg	0.92	0.93	0.92	10065

XGBoost

2.PREDICT WHETHER A PERSON NEEDS TAX DEDUCTION

methods:

Decision tree , random forest , SVC , XGBoost

determine which is best:

find Best ratio Testing Data and Best Parameters

print sklearn.metrics.classification_report

	precision	recall	f1-score	support
Not for tax deduction	0.77	0.71	0.74	2101
For tax deduction	0.81	0.85	0.83	2932
accuracy			0.79	5033
macro avg	0.79	0.78	0.79	5033
weighted avg	0.79	0.79	0.79	5033

Decision tree

	precision	recall	f1-score	support
Not for tax deduction	0.80	0.67	0.73	2101
For tax deduction	0.79	0.88	0.83	2932
accuracy			0.79	5033
macro avg	0.80	0.77	0.78	5033
weighted avg	0.79	0.79	0.79	5033

random forest

	precision	recall	f1-score	support
Not for tax deduction	0.56	0.37	0.44	2101
For tax deduction	0.64	0.80	0.71	2932
accuracy			0.62	5033
macro avg	0.60	0.58	0.58	5033
weighted avg	0.61	0.62	0.60	5033

SVC

	precision	recall	f1-score	support
Not for tax deduction	0.78	0.74	0.76	2101
For tax deduction	0.82	0.85	0.83	2932
accuracy			0.80	5033
macro avg	0.80	0.79	0.80	5033
weighted avg	0.80	0.80	0.80	5033

XGBoost

Thank you for listening!