RMIT International University Vietnam
Assignment Cover Page

Subject Code:	Practical Data Science
Subject Name:	COSC2789
Title of Assignment:	Data Preparation and Exploration
Student name:	Vo Minh Thien An
Student Number:	s3916570
Teachers Name:	Dr. Thuy Nguyen
Number of pages including this one:	19
Word Count:	2171

I declare that in submitting all work for this assessment I have read, understood and agreed to the content and expectations of the Assessment Declaration.

Task 1: Data Preparation

First, import pandas and set the display so that it can show full columns

In [1]:
```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
```

Then import the data and show it, in this report is bank.txt

In [2]:
```
# import data
df = pd.read_csv('bank.txt', delimiter='\t')
# print out data
df
```

Out[2]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	487.0	2.0	999	0	nonexistent
1	39	services	single	high.school	no	no	no	telephone	may	fri	346.0	4.0	999	0	nonexistent
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	227.0	1.0	999	0	nonexistent
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	17.0	3.0	999	0	nonexistent
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	58.0	1.0	999	0	nonexistent
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	53.0	1.0	999	0	nonexistent
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	219.0	1.0	999	0	nonexistent
4116	27	student	single	high.school	no	no	no	cellular	may	mon	64.0	2.0	999	1	failure
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	528.0	1.0	999	0	nonexistent
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	175.0	1.0	999	0	nonexistent

4119 rows × 21 columns

We can see that there are 21 columns in this file, and we have 4119 samples for each category, numbered from 0 to 4118.

Next I'll check the numeric value by using the describe() function

In [3]:
```
#brief check the numeric value
df.describe()
```

Out[3]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	4119.000000	4116.000000	4117.000000	4119.000000	4119.000000	4117.000000	4117.000000	4117.000000	4117.000000	4119.000000
mean	40.162661	256.838678	2.537284	960.422190	0.190337	0.085183	93.579449	-40.502308	3.620728	5166.481695
std	10.621359	254.745327	2.568759	191.922786	0.541788	1.563138	0.579190	4.593059	1.733778	73.667904
min	13.000000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.635000	4963.600000
25%	32.000000	103.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.334000	5099.100000
50%	38.000000	181.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.000000	317.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	140.000000	3643.000000	35.000000	999.000000	6.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

In this step we can see that there are some unnatural things in “age” and “duration”. In the “age” column we have a max value of 140, which can’t happen. In the “duration” column we have a min value of 0, which is weird.

I’ll go with the age first

In [4]:

```python
df.sort_values('age')
```

Out[4]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	e
100	13	management	married	university.degree	unknown	no	yes	cellular	apr	fri	76.0	1.0	999	1	failure	
98	15	management	married	university.degree	no	no	no	cellular	jul	tue	477.0	2.0	999	0	nonexistent	
99	16	admin.	married	unknown	no	no	no	cellular	aug	tue	91.0	1.0	999	0	nonexistent	
477	18	student	single	unknown	no	no	no	cellular	sep	thu	385.0	1.0	3	1	success	
899	18	student	single	unknown	no	yes	yes	telephone	aug	wed	297.0	1.0	999	0	nonexistent	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1215	88	retired	divorced	basic.4y	no	yes	yes	cellular	mar	wed	82.0	2.0	999	0	nonexistent	
1598	90	entrepreneur	married	university.degree	no	yes	no	cellular	aug	mon	145.0	2.0	999	0	nonexistent	
1599	94	services	divorced	high.school	no	no	no	cellular	may	mon	903.0	4.0	999	0	nonexistent	
1500	138	admin.	married	university.degree	no	yes	no	telephone	jun	thu	197.0	1.0	999	0	nonexistent	
1499	140	admin.	married	high.school	unknown	yes	no	telephone	jun	fri	85.0	3.0	999	0	nonexistent	

4119 rows × 21 columns

By using sort_value('age') we can get the data frame from smallest to largest value of age. By this we can see that there are some things that need to be fixed. The first three rows have values that do not match with other columns, and the last two rows show the unusual age. So I decided to use loc() and to_csv() to fix and update it.

In [5]:

```python
#replace 138 and 140 with 38 and 40
#then fix the csv file
df.loc[1500,'age'] = 38
df.loc[1499,'age'] = 40
df.to_csv("bank.csv", index = False)
```

I see that there are three age value which is not match with aother value, it suppose to be a typo. They are value in row number 98, 99 and 100

In [6]:

```python
#replace 13, 16 and 15 with 31, 61 and 51
#then fix the csv file
df.loc[100,'age'] = 31
df.loc[98,'age'] = 51
df.loc[99,'age'] = 61
df.to_csv("bank.csv", index = False)
```

In [7]:

```python
#double check   #age from 30-40 named most
df.sort_values('age')
```

Out[7]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	e
477	18	student	single	unknown	no	no	no	cellular	sep	thu	385.0	1.0	3	1	success	
899	18	student	single	unknown	no	yes	yes	telephone	aug	wed	297.0	1.0	999	0	nonexistent	
1661	18	student	single	unknown	no	yes	no	cellular	may	thu	183.0	1.0	7	2	success	
1887	19	student	single	high.school	unknown	yes	no	cellular	may	tue	338.0	4.0	999	0	nonexistent	
1377	20	student	single	unknown	no	yes	yes	cellular	apr	tue	47.0	2.0	999	0	nonexistent	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
696	86	retired	married	unknown	unknown	yes	yes	cellular	sep	tue	211.0	1.0	7	4	success	
1796	86	retired	married	unknown	unknown	yes	no	cellular	sep	tue	340.0	1.0	999	0	nonexistent	
1215	88	retired	divorced	basic.4y	no	yes	yes	cellular	mar	wed	82.0	2.0	999	0	nonexistent	
1598	90	entrepreneur	married	university.degree	no	yes	no	cellular	aug	mon	145.0	2.0	999	0	nonexistent	
1599	94	services	divorced	high.school	no	no	no	cellular	may	mon	903.0	4.0	999	0	nonexistent	

After that we can see the "age" now look acceptable

Next I will move to "job" column, which can easy have typo so I decided to use value_counts() to check.

```
In [8]: df["job"].value_counts()

Out[8]: admin.           1012
        blue-collar       883
        technician        691
        services          392
        management        324
        retired           166
        self-employed     159
        entrepreneur      147
        unemployed        111
        housemaid         110
        student            82
        unknown            39
        entrepreneurs       1
        bluecollar          1
        servicess           1
        Name: job, dtype: int64
```

Can see that we have three typo in here “entrepreneurs”, “bluecollar” and “servicess”

Then I can use print(str.contains()) to locate the typo, and use loc() to fix, to_csv() to update it.

```
In [9]: #find the typo
        print(df[df['job'].str.contains('entrepreneurs')])
        ...

In [10]: #find the typo
         print(df[df['job'].str.contains('bluecollar')])
         ...

In [11]: #find the typo
         print(df[df['job'].str.contains('servicess')])
         ...

In [12]: #fix the typo
         df.loc[4096,'job'] = 'services'
         df.loc[4068,'job'] = 'blue-collar'
         df.loc[4037,'job'] = 'entrepreneur'
         df.to_csv("bank.csv", index = False)
```

After that, I moved to “marital” and “education”. I don’t see and problem in “marital” so I move straight to “education”

```
In [14]: df["marital"].value_counts()

Out[14]: married     2509
         single      1153
         divorced     446
         unknown       11
         Name: marital, dtype: int64
```

marital value look good

```
In [15]: df["education"].value_counts()

Out[15]: university.degree      1264
         high.school             921
         basic.9y                572
         professional.course     535
         basic.4y                425
         basic.6y                223
         unknown                 167
         basic .6y                 4
         basic0.4y                 2
         basic .4y                 2
         basic .9y                 2
         basic0.6y                 1
         illiterate                1
         Name: education, dtype: int64
```

Can see that there are some typo and extra white space, I’ll use replace() to clear the white space first. After clearing the white space I use the same step as I used when working with the “job” column to clear the rest.

```
In [16]: #clear white space
         df["education"] = df["education"].str.replace(' ', '')
```

```
In [17]: #clear the extra 0
         #find the three typo
         print(df[df['education'].str.contains('basic0.4y')])
         print(df[df['education'].str.contains('basic0.6y')])
```

Can see that the mistake located at column 13, 19 and 20

```
In [18]: #fix the typo
         df.loc[13,'education'] = 'basic.4y'
         df.loc[19,'education'] = 'basic.4y'
         df.loc[20,'education'] = 'basic.6y'
         df.to_csv("bank.csv", index = False)
```

```
In [19]: #double check
         df["education"].value_counts()
```

Then I moved to "default" and "housing", I find it look good in "default" so moving to the "housing" part.

```
In [20]: df["default"].value_counts()

Out[20]: no         3315
         unknown     803
         yes           1
         Name: default, dtype: int64
```

It look clean, no any mistake

```
In [21]: df["housing"].value_counts()

Out[21]: yes        2172
         no         1835
         unknown     105
         Yes           2
         No            2
         na            1
         Name: housing, dtype: int64
```

In the "housing" we can see that there are several typos in it. I'll use str.replace() to fix it. Because by using this I don't need to locate the error but still can work in a large area.

Next is the "loan" column, I saw three values "na", assume that it is a typo of "no" so I use str.replace() to solve it.

```
In [24]: df["loan"].value_counts()

Out[24]: no         3346
         yes         665
         unknown     105
         na            3
         Name: loan, dtype: int64
```

```
In [25]: #solve the typo
         df["loan"] = df["loan"].str.replace('a', 'o')
```

Next to move to “contact” can see that there are some extra white space in here, which can be solve by using str.replace

```
In [27]: df["contact"].value_counts()

Out[27]: cellular       2650
         telephone      1466
          telephone        1
         cellular          1
           cellular        1
         Name: contact, dtype: int64
```

Extra white space

```
In [28]: #clear white space
         df["contact"] = df["contact"].str.replace(' ', '')
```

After finish with the “contact” we move to “month” and “day_of_week “month” value look good otherwise there are some wrong format in ‘day_of_week) so I’ll use str.replace() to fix it

```
In [30]: df["month"].value_counts()

Out[30]: may    1378
         jul     711
         aug     636
         jun     530
         nov     446
         apr     215
         oct      69
         sep      64
         mar      48
         dec      22
         Name: month, dtype: int64
```

```
In [31]: df["day_of_week"].value_counts()

Out[31]: thu       860
         mon       854
         tue       841
         wed       795
         fri       767
         Monday      1
         Friday      1
         Name: day_of_week, dtype: int64
```

In [32]:
```
#solve problem and check again
df["day_of_week"] = df["day_of_week"].str.replace('Monday', 'mon')
df["day_of_week"] = df["day_of_week"].str.replace('Friday', 'fri')
df['day_of_week'].value_counts()
```

Out[32]:
```
thu    860
mon    855
tue    841
wed    795
fri    768
Name: day_of_week, dtype: int64
```

As I mentioned before, there is value “0” in “duration” so I have to check if the column “y” have value “no” .

In [33]:
```
#there are value "0" in duration, which mean the value in column "Y" must be "no"
#there are some nan value in duration
df.sort_values('duration')
```

Out[33]:

loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
no	telephone	may	tue	0.0	4.0	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
no	telephone	may	tue	4.0	4.0	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
no	telephone	nov	wed	5.0	1.0	999	0	nonexistent	-0.1	93.200	-42.0	4.663	5195.8	no
yes	telephone	oct	mon	5.0	1.0	999	0	nonexistent	-1.1	94.601	-49.5	0.953	4963.6	no
no	telephone	sep	mon	5.0	1.0	999	0	nonexistent	-1.1	94.199	-37.5	0.879	4963.6	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
no	telephone	oct	fri	3253.0	1.0	999	0	nonexistent	-0.1	93.798	-40.4	5.045	5195.8	no
no	cellular	jul	thu	3643.0	1.0	999	0	nonexistent	1.4	93.918	-42.7	4.963	5228.1	yes
no	cellular	aug	thu	NaN	7.0	999	0	nonexistent	1.4	93.444	-36.1	4.963	5228.1	no
yes	telephone	jun	mon	NaN	6.0	999	0	nonexistent	1.4	94.465	-41.8	4.960	5228.1	no
no	cellular	may	mon	NaN	1.0	999	0	nonexistent	-1.8	92.893	-46.2	1.244	5099.1	no

By using sort_value() we can make sure that the value “0” is not an error.

After that I go through “campaign”, “pdays”, “previous”, “emp.var.rate”, “cons,price.idx”, “cons.conf.idx”, “euribor3m”, “nr.employed” and “y”. Which looks good.

In [34]:
```
df['campaign'].value_counts()
```
...

In [35]:
```
df['pdays'].value_counts()
```
...

In [36]:
```
df['previous'].value_counts()
```
...

In [37]:
```
df['emp.var.rate'].value_counts()
```
...

In [38]:
```
df['cons.price.idx'].value_counts()
```
...

In [39]:
```
df['cons.conf.idx'].value_counts()
```
...

In [40]:
```
df['euribor3m'].value_counts()
```
...

In [41]:
```
df['nr.employed'].value_counts()
```
...

In [42]:
```
df['y'].value_counts()
```
...

Because of that I keep moving to “poutcome” where I found some typo, and use str.replace() to fix it

```
In [43]: df['poutcome'].value_counts()

Out[43]: nonexistent      3521
         failure           454
         success           142
         nonexistent         1
           nonexistent       1
         Name: poutcome, dtype: int64

In [44]: #clear white space
         df["poutcome"] = df["poutcome"].str.replace(' ', '')

In [45]: #double check
         df['poutcome'].value_counts()
```

Finally is NaN or missing value check, it will help me to know if there are any missing values, and how much is missing. I’ll work on it later in task 3.

```
In [46]: #NaN check
         df.isna().sum()

Out[46]: age               0
         job               0
         marital           0
         education         0
         default           0
         housing           2
         loan              0
         contact           0
         month             0
         day_of_week       0
         duration          3
         campaign          2
         pdays             0
         previous          0
         poutcome          0
         emp.var.rate      2
         cons.price.idx    2
         cons.conf.idx     2
         euribor3m         2
         nr.employed       0
         y                 0
         dtype: int64
```

In conclusion, in this task I have finished cleaning the data by many ways, here are all the work I have done in task 1

#1 In "age" there is typo at row number 98, 99, 100, 1499, 1500. All five mistakes make illogic

#2 In "job" there is typo at row number 4037, 4068, 4096

#3 In "education" which has extra white space and typo. Typo at row number 13, 19, 20

#4 In "housing" is typo which is wrong uppercase and wrong word character and "NaN" value (missing value)

#5 In "loan" is typo which is wrong word character

#6 In "contact" is extra white space

#7 In "day_of_week" is wrong day format

#8 In "poutcome" is extra white space

#9 In "emp.var.rate" is "NaN" value (missing value)

#10 In "cons.conf.idx" is "NaN" value (missing value)

#11 In "cons.price.idx" is "NaN" value (missing value)

#12 In "euribor3m" is "NaN" value (missing value)

#13 In "campaign" is "NaN" value (missing value)

#14 In "duration" is "NaN" value (missing value)

Task 2: Data Exploration

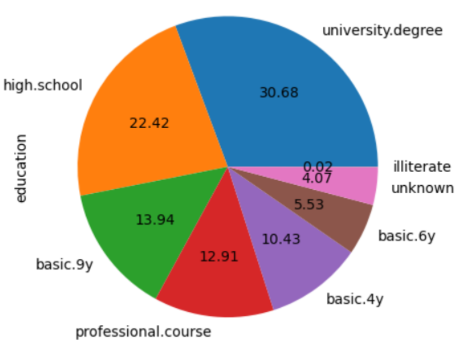
Task 2.1:

First import matplotlib to draw chart

```
In [53]: import matplotlib.pyplot as plt
```

The first chart I choose is pie chart draw based on “education” column

```
In [55]: #draw a pie chart with education level value
         df['education'].value_counts().plot(kind='pie',autopct='%.2f')
Out[55]: <AxesSubplot:ylabel='education'>
```



I choose education level, which is the ordinal value to draw a pie chart because it can show clearly the amount of people who have done high school or higher is much more than people who only did basic study.

And in this pie chart we can see that the amount of people with good education level is higher than the other with low education level

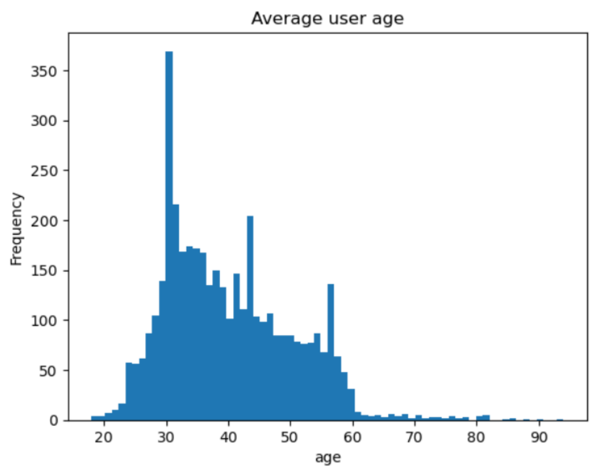
And also, pie chart is the most suitable chart to display this kind of value

This pie chart shows us that the campaign aim mostly to people with high education level, which mean there are more chance they will join the campaign.

The second chart I choose "age" column to draw the bar chart

```
In [56]: #draw bar chart with "age" value
         df['age'].plot(kind='hist',bins=70)
         plt.title('Average user age')
         plt.xlabel('age')

Out[56]: Text(0.5, 0, 'age')
```



I chose age, which is a numerical value to draw a bar chart because it can show the trend/the age that is suitable the most with this campaign. For this kind of value we need a chart that can show us two values, which is bar chart or line chart.

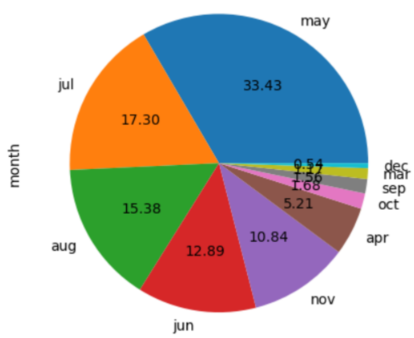
We can see the aim is people from 25 to 55 or 60, this is the age that people want to work more and have more accumulation to prepare themself from retirement age. So they will tend to accept the campaign or the loan to self growth to as high as they could.

People above 60 are the opposite, they want to be safe, enjoy their life. So it will hard to invite them to a new loan or campaign

The third chart I choose is pie chart draw based on value of "month" column

```
In [57]: #draw a pie chart with month level value
         df['month'].value_counts().plot(kind='pie',autopct='%.2f')

Out[57]: <AxesSubplot:ylabel='month'>
```



I choose “month”, which is the nominal value, to draw a pie chart. I choose this category because it can show the difference in the number of people who want to loan by specific time in year. By this we can predict something will happen in around specific time

Can see that the number in may is incredibly high which also lead to june and july, we can assume that there is something happen around this time.

Task 2.2:

first pair

The first pair I choose is education level and default

Hypothesis: The "high education" group have the potential in pay and loan more than the "low level of education" group

While default shows customer's credit in default, the education level shows how you did your study.

If we look at the first pie chart at high education(from basic 9 year to professional) and other is low education. We can see that these two charts have the same patent.

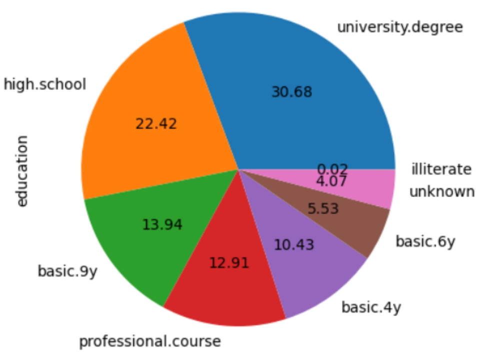
Both high education and no credit in default take nearly 80% of the chart.

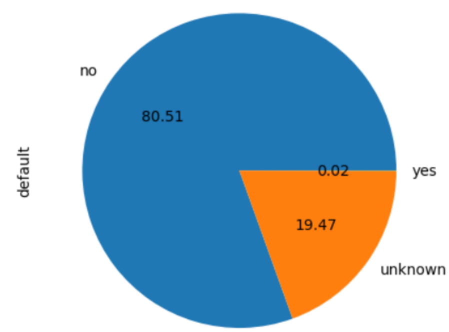
Then we can make a hypothesis that these 80% are easier to pay for their loan and loan back once again than the other 20%, because of that we should aim for this group to make a marketing plan.

For this hypothesis I'll say that because they have a good education level, which makes me believe that they are using this loan for investment.

```
In [59]: plt.figure(0)
         df['education'].value_counts().plot(kind='pie',autopct='%.2f')
         plt.figure(1)
         df['default'].value_counts().plot(kind='pie',autopct='%.2f')
         plt.show

Out[59]: <function matplotlib.pyplot.show(close=None, block=None)>
```





Second pair

In this second pair I choose Job and cons.conf.idx as know as consumer confidence index to draw a boxplot

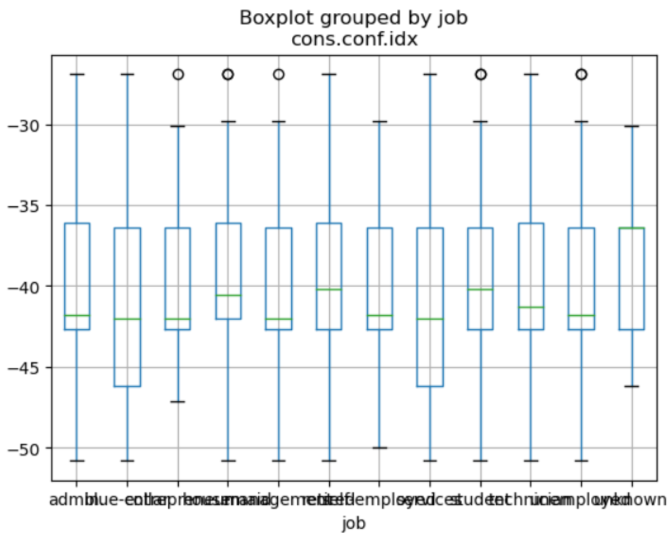
Hypothesis: There can be a pandemic going through.

While consumer confidence index help me to know how customer think about their life now and 6 month later

And it is separated by job, which means this happens in all people with a variety of jobs.

We can see that all the index is negative, which means there is something happening in this time period that makes everybody feel uncomfortable about their life. If you combine this with the consumer price index, which is lower than 100. We can say that mind be a pandemic have undergo through a pandemic which make the increase in the supply and decrease in living condition

```
In [60]: df.dropna().boxplot(column='cons.conf.idx',by='job')
         plt.show()
```



Third pair

In the third pair I choose 'pdays' and 'poutcome' which can show us the success of the previous marketing plan.

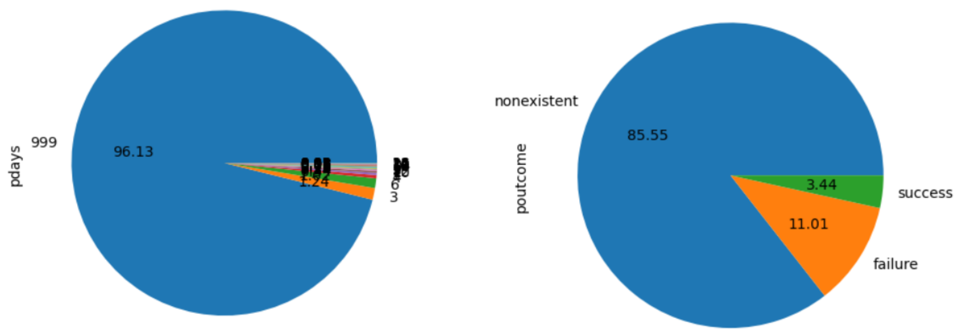
Hypothesis: The previous marketing plan was not successful

These two column show us that the previous marketing campaign did not bring good outcome

We can see the value '999' in 'pdays', which means that the client was not contacted since the last marketing campaign, things go the same in 'poutcome' more than 80% outcome is nonexist, the other 10% is failure.

With these two charts we can assume that the previous marketing plan was not successful, we need to find the reason and fix it if we want to gain success.

```
In [61]: plt.figure(0)
         df['pdays'].value_counts().plot(kind='pie',autopct='%.2f')
         plt.figure(1)
         df['poutcome'].value_counts().plot(kind='pie',autopct='%.2f')
         plt.show
Out[61]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Task 2.3:

First import scatter matrix

```
In [63]: #task 2.3

In [64]: from pandas.plotting import scatter_matrix

In [65]: age = df["age"]
         previous = df["previous"]
         campaign = df["campaign"]
         df_matrix = pd.DataFrame({'age':age, 'previous':previous, 'campaign':campaign})
         df_matrix.head()
Out[65]:
```

	age	previous	campaign
0	30	0	2.0
1	39	0	4.0
2	25	0	1.0
3	38	0	3.0
4	47	0	1.0

Then create three new variables that have values of “age”, “previous” and “campaign” because we can’t make a scatter matrix from a full dataframe.

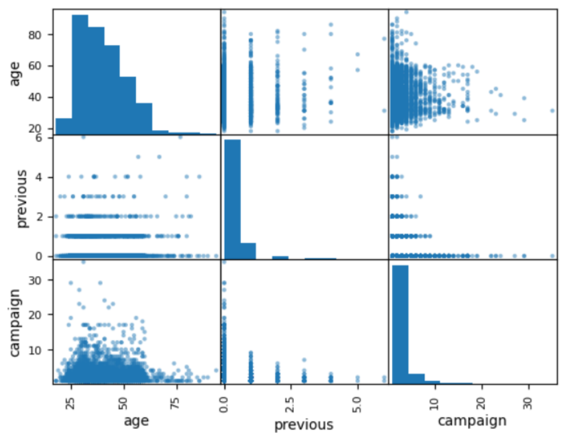
After that, set up the new dataframe called df_matrix and test if it works.

Finally create a scatter matrix by using pd.plotting.scatter_matrix()

We will have the outcome like the picture bellow.

```
In [66]: pd.plotting.scatter_matrix(df_matrix)

Out[66]: array([[<AxesSubplot:xlabel='age', ylabel='age'>,
                 <AxesSubplot:xlabel='previous', ylabel='age'>,
                 <AxesSubplot:xlabel='campaign', ylabel='age'>],
                [<AxesSubplot:xlabel='age', ylabel='previous'>,
                 <AxesSubplot:xlabel='previous', ylabel='previous'>,
                 <AxesSubplot:xlabel='campaign', ylabel='previous'>],
                [<AxesSubplot:xlabel='age', ylabel='campaign'>,
                 <AxesSubplot:xlabel='previous', ylabel='campaign'>,
                 <AxesSubplot:xlabel='campaign', ylabel='campaign'>]], dtype=object)
```



Task3: Dealing with Missing Values and Outliers

We already check the missing value in task one

```
In [46]: #NaN check
         df.isna().sum()

Out[46]: age               0
         job               0
         marital           0
         education         0
         default           0
         housing           2
         loan              0
         contact           0
         month             0
         day_of_week       0
         duration          3
         campaign          2
         pdays             0
         previous          0
         poutcome          0
         emp.var.rate      2
         cons.price.idx    2
         cons.conf.idx     2
         euribor3m         2
         nr.employed       0
         y                 0
         dtype: int64
```

We can see that the missing value does not take a large part in the dataframe So that I'll deal with it in three ways.

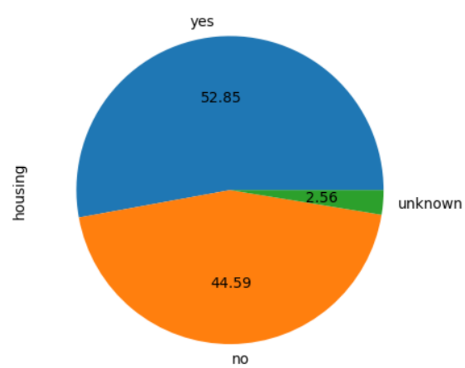
First way is delete any row that have “NaN” value in it

First I created df_delete_NaN.csv as a copy of bank.csv so that anything I changed won't affect the bank.csv file. Then I used dropna(inplace = True) to delete NaN values and make changes in the csv file.

```
In [47]: df_delete_NaN = pd.read_csv(r'bank_delete_NaN.csv')
         df_delete_NaN = df
```

```
In [48]: df_delete_NaN.dropna(inplace = True)
```

```
In [63]: df_delete_NaN['housing'].value_counts().plot(kind='pie',autopct='%.2f')
Out[63]: <AxesSubplot:ylabel='housing'>
```



<pic 3.1.1 pie chart of "housing" column after delete missing value>

By delete NaN row, we can make sure that our data is correct and match with each other, but it will decrease the total number of data we have

Second way is fill any "NaN" value with the value in next row [1]

First I created df_fill_next.csv as a copy of bank.csv so that anything I changed won't affect the bank.csv file. Then I used fillna(method = 'bfill') to fill NaN values with the value in the next row and make changes in the csv file.

```
In [49]: df_fill_next = pd.read_csv(r'bank_fill_next.csv')
         df_fill_next = df
```

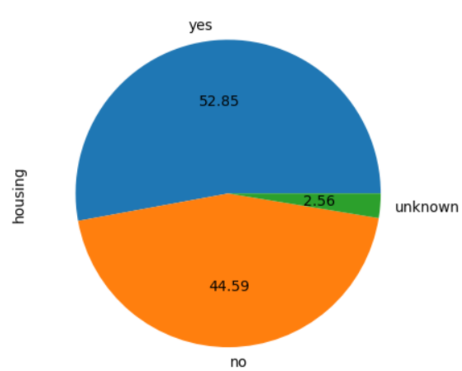
```
In [50]: df_fill_next.fillna(method = 'bfill')
```

Out[50]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	487.0	2.0	999	0	nonexistent
1	39	services	single	high.school	no	no	no	telephone	may	fri	346.0	4.0	999	0	nonexistent
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	227.0	1.0	999	0	nonexistent
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	17.0	3.0	999	0	nonexistent
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	58.0	1.0	999	0	nonexistent
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	53.0	1.0	999	0	nonexistent
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	219.0	1.0	999	0	nonexistent
4116	27	student	single	high.school	no	no	no	cellular	may	mon	64.0	2.0	999	1	failure
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	528.0	1.0	999	0	nonexistent
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	175.0	1.0	999	0	nonexistent

4104 rows × 21 columns

```
In [64]: df_fill_next['housing'].value_counts().plot(kind='pie',autopct='%.2f')
Out[64]: <AxesSubplot:ylabel='housing'>
```



<pic 3.1.2 pie chart of “housing” column after fill missing value with value next row>

Third way is fill any “NaN” value with the value in previous row [1]

First I created df_fill_pre.csv as a copy of bank.csv so that anything I changed won’t affect the bank.csv file. Then I used fillna(method = ‘pad’) to fill NaN values with the value in the previous row and make changes in the csv file.

In [52]:
```
df_fill_pre.fillna(method = 'pad')
```

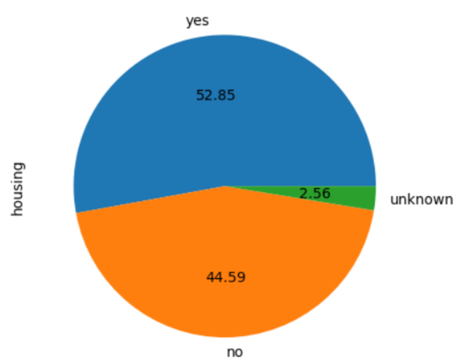
Out[52]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	487.0	2.0	999	0	nonexistent
1	39	services	single	high.school	no	no	no	telephone	may	fri	346.0	4.0	999	0	nonexistent
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	227.0	1.0	999	0	nonexistent
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	17.0	3.0	999	0	nonexistent
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	58.0	1.0	999	0	nonexistent
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	53.0	1.0	999	0	nonexistent
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	219.0	1.0	999	0	nonexistent
4116	27	student	single	high.school	no	no	no	cellular	may	mon	64.0	2.0	999	1	failure
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	528.0	1.0	999	0	nonexistent
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	175.0	1.0	999	0	nonexistent

4104 rows × 21 columns

In [65]:
```
df_fill_pre['housing'].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[65]: <AxesSubplot:ylabel='housing'>



<pic 3.1.3 pie chart of “housing” column after fill missing value with value previous row>

By these two ways we can make sure the data match with each value but it makes some mistakes while we take the value in a different row for another. But on the other hand it can save us the full amount of data collected.

After creating the pie chart of “housing” value after work with it in three different ways we can see that there are not much different between these three, because the missing value take small part in general so it can’t really make a huge change in final result

3.2

- Yes, outliers can affect standard deviation because it can show us unreal value when we draw a chart, if we follow that to make prediction it can cause many problems.[2]
- The outlier may still be valuable but we need to investigate carefully, but most of the time it is considered as a bad data point. [2]

- There are some outliers that represent natural variations in the population, and they should be left as is in your dataset. These are called true outliers. [3]

4. References

[1] “Working with Missing Data in Pandas” last update 08 Jun, 2022. GeeksforGeeks .[online]. available: this link

[2] “7.1.6 What are outliers in the data?” NIST .[online]. available:this link

[3] “When should I remove an outlier from my dataset?” SCRIBBR .[online]. available: this link