EDA_Notebook_Complete

September 19, 2024

0.1 Bank Telemarketing Campaign Case Study.

In this case study you'll be learning Exploratory Data Analytics with the help of a case study on "Bank marketing campaign". This will enable you to understand why EDA is a most important step in the process of Machine Learning.

Problem Statement:

The bank provides financial services/products such as savings accounts, current accounts, debit cards, etc. to its customers. In order to increase its overall revenue, the bank conducts various marketing campaigns for its financial products such as credit cards, term deposits, loans, etc. These campaigns are intended for the bank's existing customers. However, the marketing campaigns need to be cost-efficient so that the bank not only increases their overall revenues but also the total profit. You need to apply your knowledge of EDA on the given dataset to analyse the patterns and provide inferences/solutions for the future marketing campaign.

The bank conducted a telemarketing campaign for one of its financial products 'Term Deposits' to help foster long-term relationships with existing customers. The dataset contains information about all the customers who were contacted during a particular year to open term deposit accounts.

What is the term Deposit?

Term deposits also called fixed deposits, are the cash investments made for a specific time period ranging from 1 month to 5 years for predetermined fixed interest rates. The fixed interest rates offered for term deposits are higher than the regular interest rates for savings accounts. The customers receive the total amount (investment plus the interest) at the end of the maturity period. Also, the money can only be withdrawn at the end of the maturity period. Withdrawing money before that will result in an added penalty associated, and the customer will not receive any interest returns.

Your target is to do end to end EDA on this bank telemarketing campaign data set to infer knowledge that where bank has to put more effort to improve it's positive response rate.

Importing the libraries.

[6]:
```
#import the warnings.
import warnings
warnings.filterwarnings("ignore")
```

[7]:
```
#import the useful libraries.
import pandas as pd, numpy as np
```

```
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

0.2 Session- 2, Data Cleaning

0.2.1 Segment- 2, Data Types

There are multiple types of data types available in the data set. some of them are numerical type and some of categorical type. You are required to get the idea about the data types after reading the data frame.

Following are the some of the types of variables: - **Numeric data type**: banking dataset: salary, balance, duration and age. - **Categorical data type**: banking dataset: education, job, marital, poutcome and month etc. - **Ordinal data type**: banking dataset: Age group. - **Time and date type** - **Coordinates type of data**: latitude and longitude type.

Read in the Data set.

[12]:
```
#read the data set of "bank telemarketing campaign" in inp0.
inp0= pd.read_csv("bank_marketing_updated_v1.csv")
```

[13]:
```
#Print the head of the data frame.
inp0.head()
```

[13]:
```
       banking marketing Unnamed: 1                    Unnamed: 2 Unnamed: 3  \
0  customer id and age.        NaN  Customer salary and balance.        NaN
1             customerid        age                        salary    balance
2                      1         58                        100000       2143
3                      2         44                         60000         29
4                      3         33                        120000          2

                                          Unnamed: 4              Unnamed: 5  \
0  Customer marital status and job with education…                     NaN
1                                            marital                  jobedu
2                                            married     management,tertiary
3                                             single    technician,secondary
4                                            married  entrepreneur,secondary

                                   Unnamed: 6 Unnamed: 7  \
0  particular customer before targeted or not        NaN
1                                    targeted    default
2                                         yes         no
3                                         yes         no
4                                         yes         no

                        Unnamed: 8 Unnamed: 9   Unnamed: 10 Unnamed: 11  \
0  Loan types: loans or housing loans        NaN  Contact type         NaN
1                             housing       loan       contact         day
2                                 yes         no       unknown           5
```

```
3                                yes         no       unknown          5
4                                yes        yes       unknown          5

       Unnamed: 12       Unnamed: 13 Unnamed: 14 Unnamed: 15 Unnamed: 16  \
0  month of contact  duration of call         NaN         NaN         NaN
1             month          duration    campaign       pdays    previous
2         may, 2017           261 sec           1          -1           0
3         may, 2017           151 sec           1          -1           0
4         may, 2017            76 sec           1          -1           0

                  Unnamed: 17                             Unnamed: 18
0  outcome of previous contact  response of customer after call happned
1                     poutcome                                response
2                      unknown                                      no
3                      unknown                                      no
4                      unknown                                      no
```

0.2.2 Segment- 3, Fixing the Rows and Columns

Checklist for fixing rows: - **Delete summary rows**: Total and Subtotal rows - **Delete incorrect rows**: Header row and footer row - **Delete extra rows**: Column number, indicators, Blank rows, Page No.

Checklist for fixing columns: - **Merge columns for creating unique identifiers**, if needed, for example, merge the columns State and City into the column Full address. - **Split columns to get more data**: Split the Address column to get State and City columns to analyse each separately. - **Add column names**: Add column names if missing. - **Rename columns consistently**: Abbreviations, encoded columns. - **Delete columns**: Delete unnecessary columns. - **Align misaligned columns**: The data set may have shifted columns, which you need to align correctly.

Read the file without unnecessary headers.

[17]:
```python
#read the file in inp0 without first two rows as it is of no use.
inp0=pd.read_csv("bank_marketing_updated_v1.csv", skiprows= 2)
```

[18]:
```python
#print the head of the data frame.
inp0.head()
```

[18]:
```
   customerid   age  salary  balance  marital                  jobedu  \
0           1  58.0  100000     2143  married     management,tertiary
1           2  44.0   60000       29   single    technician,secondary
2           3  33.0  120000        2  married  entrepreneur,secondary
3           4  47.0   20000     1506  married     blue-collar,unknown
4           5  33.0       0        1   single         unknown,unknown

  targeted default housing loan  contact  day      month duration  campaign  \
0      yes      no      yes   no  unknown    5  may, 2017   261 sec         1
1      yes      no      yes   no  unknown    5  may, 2017   151 sec         1
```

```
2      yes      no      yes  yes  unknown    5  may, 2017   76 sec          1
3       no      no      yes   no  unknown    5  may, 2017   92 sec          1
4       no      no       no   no  unknown    5  may, 2017  198 sec          1

   pdays  previous poutcome response
0     -1         0  unknown       no
1     -1         0  unknown       no
2     -1         0  unknown       no
3     -1         0  unknown       no
4     -1         0  unknown       no
```

Dropping customer id column.

[20]:
```python
#drop the customer id as it is of no use.
inp0.drop("customerid", axis=1, inplace=True)
inp0.head()
```

[20]:
```
    age  salary  balance  marital                  jobedu targeted default  \
0  58.0  100000     2143  married     management,tertiary      yes      no
1  44.0   60000       29   single    technician,secondary      yes      no
2  33.0  120000        2  married  entrepreneur,secondary      yes      no
3  47.0   20000     1506  married     blue-collar,unknown       no      no
4  33.0       0        1   single         unknown,unknown       no      no

  housing loan  contact  day      month duration  campaign  pdays  previous  \
0     yes   no  unknown    5  may, 2017  261 sec         1     -1         0
1     yes   no  unknown    5  may, 2017  151 sec         1     -1         0
2     yes  yes  unknown    5  may, 2017   76 sec         1     -1         0
3     yes   no  unknown    5  may, 2017   92 sec         1     -1         0
4      no   no  unknown    5  may, 2017  198 sec         1     -1         0

  poutcome response
0  unknown       no
1  unknown       no
2  unknown       no
3  unknown       no
4  unknown       no
```

Dividing "jobedu" column into job and education categories.

[22]:
```python
#Extract job in newly created 'job' column from "jobedu" column.
inp0['job']=inp0.jobedu.apply(lambda x: x.split(",")[0])
inp0.head()
```

[22]:
```
    age  salary  balance  marital                  jobedu targeted default  \
0  58.0  100000     2143  married     management,tertiary      yes      no
1  44.0   60000       29   single    technician,secondary      yes      no
2  33.0  120000        2  married  entrepreneur,secondary      yes      no
```

```
3  47.0   20000      1506  married     blue-collar,unknown       no       no
4  33.0       0         1   single         unknown,unknown       no       no

  housing loan  contact  day      month duration  campaign  pdays  previous  \
0     yes   no  unknown    5  may, 2017  261 sec         1     -1         0
1     yes   no  unknown    5  may, 2017  151 sec         1     -1         0
2     yes  yes  unknown    5  may, 2017   76 sec         1     -1         0
3     yes   no  unknown    5  may, 2017   92 sec         1     -1         0
4      no   no  unknown    5  may, 2017  198 sec         1     -1         0

  poutcome response           job
0  unknown       no    management
1  unknown       no    technician
2  unknown       no  entrepreneur
3  unknown       no   blue-collar
4  unknown       no       unknown
```

[23]:
```python
#Extract education in newly created 'education' column from "jobedu" column.
inp0['education']=inp0.jobedu.apply(lambda x: x.split(",")[1])
inp0.head()
```

[23]:
```
    age  salary  balance  marital                  jobedu targeted default  \
0  58.0  100000     2143  married     management,tertiary      yes      no
1  44.0   60000       29   single    technician,secondary      yes      no
2  33.0  120000        2  married  entrepreneur,secondary      yes      no
3  47.0   20000     1506  married     blue-collar,unknown       no      no
4  33.0       0        1   single         unknown,unknown       no      no

  housing loan  contact  day      month duration  campaign  pdays  previous  \
0     yes   no  unknown    5  may, 2017  261 sec         1     -1         0
1     yes   no  unknown    5  may, 2017  151 sec         1     -1         0
2     yes  yes  unknown    5  may, 2017   76 sec         1     -1         0
3     yes   no  unknown    5  may, 2017   92 sec         1     -1         0
4      no   no  unknown    5  may, 2017  198 sec         1     -1         0

  poutcome response           job  education
0  unknown       no    management   tertiary
1  unknown       no    technician  secondary
2  unknown       no  entrepreneur  secondary
3  unknown       no   blue-collar    unknown
4  unknown       no       unknown    unknown
```

[24]:
```python
#drop the "jobedu" column from the dataframe.
inp0.drop('jobedu',axis= 1, inplace= True)
inp0.head()
```

[24]:
```
    age  salary  balance  marital targeted default housing loan  contact  day  \
0  58.0  100000     2143  married      yes      no     yes   no  unknown    5
1  44.0   60000       29   single      yes      no     yes   no  unknown    5
2  33.0  120000        2  married      yes      no     yes  yes  unknown    5
3  47.0   20000     1506  married       no      no     yes   no  unknown    5
4  33.0       0        1   single       no      no      no   no  unknown    5

      month duration  campaign  pdays  previous poutcome response  \
0  may, 2017  261 sec         1     -1         0  unknown       no
1  may, 2017  151 sec         1     -1         0  unknown       no
2  may, 2017   76 sec         1     -1         0  unknown       no
3  may, 2017   92 sec         1     -1         0  unknown       no
4  may, 2017  198 sec         1     -1         0  unknown       no

            job  education
0    management   tertiary
1    technician  secondary
2  entrepreneur  secondary
3   blue-collar    unknown
4       unknown    unknown
```

Extract the month from column 'month'

[26]:
```python
inp0[inp0.month.apply(lambda x: isinstance(x,float))== True]
```

[26]:
```
        age  salary  balance   marital targeted default housing loan  \
189    31.0  100000        0    single       no      no     yes   no
769    39.0   20000      245   married      yes      no     yes   no
860    33.0   55000      165   married      yes      no      no   no
1267   36.0   50000      114   married      yes      no     yes  yes
1685   34.0   20000      457   married      yes      no     yes   no
1899   49.0   16000      164  divorced      yes      no     yes   no
2433   26.0   60000     3825   married      yes      no     yes   no
2612   38.0   50000      446    single       no      no     yes   no
2747   48.0  120000     2550   married       no      no     yes   no
3556   41.0   20000       59   married      yes      no     yes   no
3890   56.0   55000     4391   married       no      no     yes   no
5311   22.0   20000        0    single      yes      no     yes   no
6265   32.0   50000       13    single      yes      no     yes   no
6396   24.0   70000        0   married      yes      no     yes   no
8433   38.0   60000    12926    single      yes      no     yes   no
8792   24.0   50000      262   married      yes      no     yes   no
10627  45.0   60000      533   married      yes      no     yes   no
11016  46.0   70000      741   married      yes      no      no   no
11284  44.0   16000     1059    single      yes      no      no   no
11394  54.0   60000      415   married      yes      no     yes   no
14502  35.0   70000      819   married      yes      no     yes   no
```

15795	38.0	20000	-41	married	yes	no	yes	no
16023	35.0	60000	328	married	yes	no	yes	no
16850	45.0	55000	25	married	yes	no	no	yes
17568	56.0	70000	0	married	no	no	no	no
18431	42.0	70000	247	single	yes	no	yes	no
18942	49.0	50000	949	married	yes	no	no	no
19118	38.0	50000	1980	married	yes	no	no	no
19769	36.0	100000	162	married	yes	no	yes	no
21777	56.0	16000	605	married	yes	no	no	no
21962	36.0	60000	1044	single	yes	no	yes	no
23897	46.0	20000	123	married	yes	no	no	no
25658	35.0	60000	8647	married	yes	no	no	no
27480	31.0	100000	3283	single	no	no	no	no
28693	26.0	16000	543	married	yes	no	no	no
30740	32.0	100000	2770	single	no	no	no	no
31551	54.0	55000	136	married	yes	no	yes	no
35773	52.0	20000	33	married	no	no	no	no
37194	36.0	20000	1969	married	yes	no	yes	yes
37819	34.0	20000	237	married	yes	no	yes	no
38158	34.0	60000	1317	divorced	no	no	yes	no
39188	30.0	60000	778	single	yes	no	yes	no
41090	35.0	100000	7218	single	no	no	no	no
41434	43.0	100000	13450	married	yes	no	yes	no
41606	25.0	100000	808	single	no	no	no	no
43001	35.0	60000	353	single	no	no	no	no
43021	52.0	100000	4675	married	yes	no	no	no
43323	54.0	70000	0	divorced	yes	no	no	no
44131	27.0	100000	843	single	yes	no	no	no
44732	23.0	4000	508	single	no	no	no	no

	contact	day	month	duration	campaign	pdays	previous	\
189	unknown	5	NaN	562 sec	1	-1	0	
769	unknown	7	NaN	148 sec	3	-1	0	
860	unknown	7	NaN	111 sec	1	-1	0	
1267	unknown	8	NaN	147 sec	1	-1	0	
1685	unknown	9	NaN	266 sec	1	-1	0	
1899	unknown	9	NaN	1080 sec	5	-1	0	
2433	unknown	13	NaN	107 sec	1	-1	0	
2612	unknown	13	NaN	386 sec	1	-1	0	
2747	unknown	14	NaN	175 sec	3	-1	0	
3556	unknown	15	NaN	75 sec	8	-1	0	
3890	unknown	16	NaN	291 sec	1	-1	0	
5311	unknown	23	NaN	816 sec	2	-1	0	
6265	unknown	27	NaN	88 sec	2	-1	0	
6396	unknown	27	NaN	299 sec	1	-1	0	
8433	unknown	3	NaN	280 sec	1	-1	0	
8792	unknown	4	NaN	69 sec	3	-1	0	

10627	unknown	16	NaN	332 sec	2	-1	0
11016	unknown	17	NaN	161 sec	3	-1	0
11284	unknown	18	NaN	2093 sec	1	-1	0
11394	unknown	19	NaN	34 sec	31	-1	0
14502	telephone	14	NaN	1.7 min	14	-1	0
15795	cellular	21	NaN	1.13333333333333 min	10	-1	0
16023	cellular	22	NaN	10.9 min	2	-1	0
16850	cellular	25	NaN	1.91666666666667 min	3	-1	0
17568	cellular	29	NaN	1.38333333333333 min	2	-1	0
18431	cellular	31	NaN	1.9 min	2	-1	0
18942	cellular	4	NaN	1.51666666666667 min	1	-1	0
19118	cellular	5	NaN	2.93333333333333 min	2	-1	0
19769	cellular	8	NaN	1.25 min	2	-1	0
21777	cellular	19	NaN	3.45 min	6	-1	0
21962	cellular	20	NaN	0.25 min	19	-1	0
23897	cellular	29	NaN	2.8 min	2	-1	0
25658	cellular	19	NaN	2.33333333333333 min	2	-1	0
27480	cellular	21	NaN	6.28333333333333 min	1	-1	0
28693	cellular	30	NaN	2.81666666666667 min	3	-1	0
30740	telephone	6	NaN	0.733333333333333 min	9	-1	0
31551	cellular	3	NaN	5.86666666666667 min	1	332	2
35773	telephone	8	NaN	5.01666666666667 min	1	-1	0
37194	cellular	13	NaN	1.45 min	1	-1	0
37819	cellular	14	NaN	1.91666666666667 min	3	-1	0
38158	cellular	15	NaN	3.98333333333333 min	1	-1	0
39188	cellular	18	NaN	0.366666666666667 min	2	346	2
41090	cellular	14	NaN	3.73333333333333 min	3	-1	0
41434	cellular	4	NaN	2.13333333333333 min	1	-1	0
41606	cellular	18	NaN	4.45 min	2	114	2
43001	cellular	11	NaN	5.86666666666667 min	1	183	1
43021	cellular	12	NaN	3.01666666666667 min	3	-1	0
43323	cellular	18	NaN	6.03333333333333 min	1	290	3
44131	cellular	12	NaN	2.05 min	2	185	1
44732	cellular	8	NaN	3.5 min	1	92	1

	poutcome	response	job	education
189	unknown	no	management	tertiary
769	unknown	no	blue-collar	primary
860	unknown	no	retired	secondary
1267	unknown	no	admin.	secondary
1685	unknown	no	blue-collar	secondary
1899	unknown	no	housemaid	primary
2433	unknown	no	technician	tertiary
2612	unknown	no	admin.	unknown
2747	unknown	no	entrepreneur	unknown
3556	unknown	no	blue-collar	secondary
3890	unknown	no	retired	unknown

```
5311    unknown        no    blue-collar  secondary
6265    unknown        no         admin.  secondary
6396    unknown        no       services   tertiary
8433    unknown        no     technician  secondary
8792    unknown        no         admin.  secondary
10627   unknown        no     technician   tertiary
11016   unknown        no       services    primary
11284   unknown       yes      housemaid    primary
11394   unknown        no     technician  secondary
14502   unknown        no       services  secondary
15795   unknown        no    blue-collar    primary
16023   unknown       yes     technician   tertiary
16850   unknown        no        retired    primary
17568   unknown        no       services    unknown
18431   unknown        no       services  secondary
18942   unknown        no         admin.  secondary
19118   unknown        no         admin.   tertiary
19769   unknown        no     management   tertiary
21777   unknown        no      housemaid    primary
21962   unknown        no     technician  secondary
23897   unknown        no    blue-collar    primary
25658   unknown        no  self-employed   tertiary
27480   unknown        no     management   tertiary
28693   unknown        no      housemaid   tertiary
30740   unknown        no     management   tertiary
31551   failure        no        retired    primary
35773   unknown        no    blue-collar    unknown
37194   unknown        no    blue-collar  secondary
37819   unknown        no    blue-collar  secondary
38158   unknown        no     technician   tertiary
39188   failure        no     technician  secondary
41090   unknown        no     management   tertiary
41434   unknown        no     management   tertiary
41606   failure       yes     management   tertiary
43001   success       yes  self-employed   tertiary
43021   unknown       yes     management   tertiary
43323   success       yes       services  secondary
44131   success        no     management  secondary
44732   failure        no        student   tertiary
```

let's check the missing values in month column.

[28]:
```
inp0.isnull().sum()
```

[28]:
```
age           20
salary         0
balance        0
```

```
marital       0
targeted      0
default       0
housing       0
loan          0
contact       0
day           0
month        50
duration      0
campaign      0
pdays         0
previous      0
poutcome      0
response     30
job           0
education     0
dtype: int64
```

0.2.3 Segment- 4, Impute/Remove missing values

Take aways from the lecture on missing values:

- **Set values as missing values**: Identify values that indicate missing data, for example, treat blank strings, "NA", "XX", "999", etc., as missing.
- **Adding is good, exaggerating is bad**: You should try to get information from reliable external sources as much as possible, but if you can't, then it is better to retain missing values rather than exaggerating the existing rows/columns.
- **Delete rows and columns**: Rows can be deleted if the number of missing values is insignificant, as this would not impact the overall analysis results. Columns can be removed if the missing values are quite significant in number.
- **Fill partial missing values using business judgement**: Such values include missing time zone, century, etc. These values can be identified easily.

Types of missing values: - **MCAR**: It stands for Missing completely at random (the reason behind the missing value is not dependent on any other feature). - **MAR**: It stands for Missing at random (the reason behind the missing value may be associated with some other features). - **MNAR**: It stands for Missing not at random (there is a specific reason behind the missing value).

handling missing values in age column.

[32]:
```
#count the missing values in age column.
inp0.age.isnull().sum()
```

[32]: 20

[33]:
```
#pring the shape of dataframe inp0
inp0.shape
```

[33]: (45211, 19)

[34]:
```python
#calculate the percentage of missing values in age column.
float(100.0*20/45211)
```

[34]: 0.04423702196368141

Drop the records with age missing.

[36]:
```python
#drop the records with age missing in inp0 and copy in inp1 dataframe.
inp1=inp0[-inp0.age.isnull()].copy()
inp1.shape
```

[36]: (45191, 19)

handling missing values in month column

[38]:
```python
#count the missing values in month column in inp1.
inp1.month.isnull().sum()
```

[38]: 50

[39]:
```python
#print the percentage of each month in the data frame inp1.
float(100.0*50/45191)
```

[39]: 0.11064149941360005

[40]:
```python
inp1.month.value_counts(normalize = True)
```

[40]:
```
may, 2017    0.304380
jul, 2017    0.152522
aug, 2017    0.138123
jun, 2017    0.118141
nov, 2017    0.087880
apr, 2017    0.064908
feb, 2017    0.058616
jan, 2017    0.031058
oct, 2017    0.016327
sep, 2017    0.012760
mar, 2017    0.010545
dec, 2017    0.004741
Name: month, dtype: float64
```

[41]:
```python
#find the mode of month in inp1
month_mode=inp1.month.mode()[0]
month_mode
```

[41]: 'may, 2017'

[42]:
```
# fill the missing values with mode value of month in inp1.
inp1.month.fillna(month_mode, inplace= True)
inp1.month.value_counts(normalize= True)
```

[42]:
```
may, 2017    0.305149
jul, 2017    0.152353
aug, 2017    0.137970
jun, 2017    0.118010
nov, 2017    0.087783
apr, 2017    0.064836
feb, 2017    0.058551
jan, 2017    0.031024
oct, 2017    0.016309
sep, 2017    0.012746
mar, 2017    0.010533
dec, 2017    0.004735
Name: month, dtype: float64
```

[43]:
```
#let's see the null values in the month column.
inp1.month.isnull().sum()
```

[43]:
```
0
```

handling missing values in response column

[45]:
```
#count the missing values in response column in inp1.
inp1.response.isnull().sum()
```

[45]:
```
30
```

[46]:
```
#calculate the percentage of missing values in response column.
float(100.0*30/45191)
```

[46]:
```
0.06638489964816004
```

Target variable is better of not imputed. - Drop the records with missing values.

[48]:
```
#drop the records with response missings in inp1.
inp1= inp1[~inp1.response.isnull()]
```

[49]:
```
#calculate the missing values in each column of data frame: inp1.
inp1.isnull().sum()
```

[49]:
```
age         0
salary      0
balance     0
marital     0
targeted    0
```

```
default      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
response     0
job          0
education    0
dtype: int64
```

handling pdays column.

[51]:
```
#describe the pdays column of inp1.
inp1.pdays.describe()
```

[51]:
```
count    45161.000000
mean        40.182015
std        100.079372
min         -1.000000
25%         -1.000000
50%         -1.000000
75%         -1.000000
max        871.000000
Name: pdays, dtype: float64
```

-1 indicates the missing values. Missing value does not always be present as null. How to handle it:

Objective is: - you should ignore the missing values in the calculations - simply make it missing - replace -1 with NaN. - all summary statistics- mean, median etc. we will ignore the missing values of pdays.

[53]:
```
#describe the pdays column with considering the -1 values.
inp1.loc[inp1.pdays<0,"pdays"]=np.NaN
inp1.pdays.describe()
```

[53]:
```
count    8246.000000
mean      224.542202
std       115.210792
min         1.000000
25%       133.000000
50%       195.000000
```

```
75%        327.000000
max        871.000000
Name: pdays, dtype: float64
```

0.2.4 Segment- 5, Handling Outliers

Major approaches to the treat outliers:

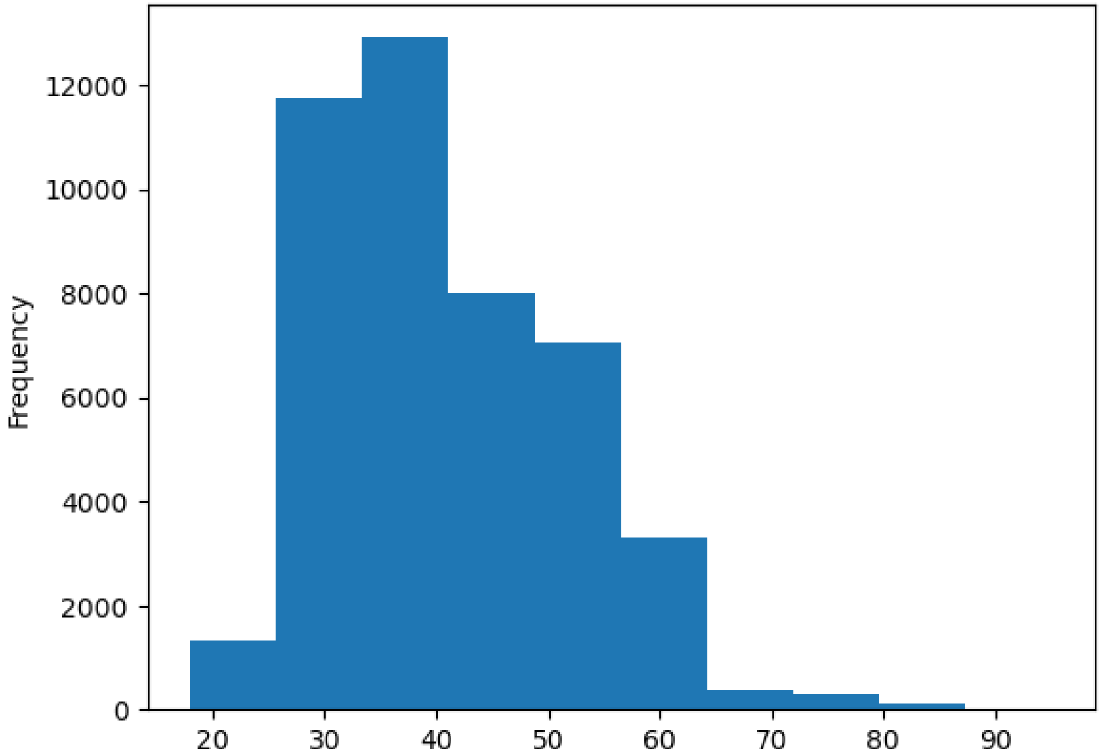
- **Imputation**
- **Deletion of outliers**
- **Binning of values**
- **Cap the outlier**

Age variable

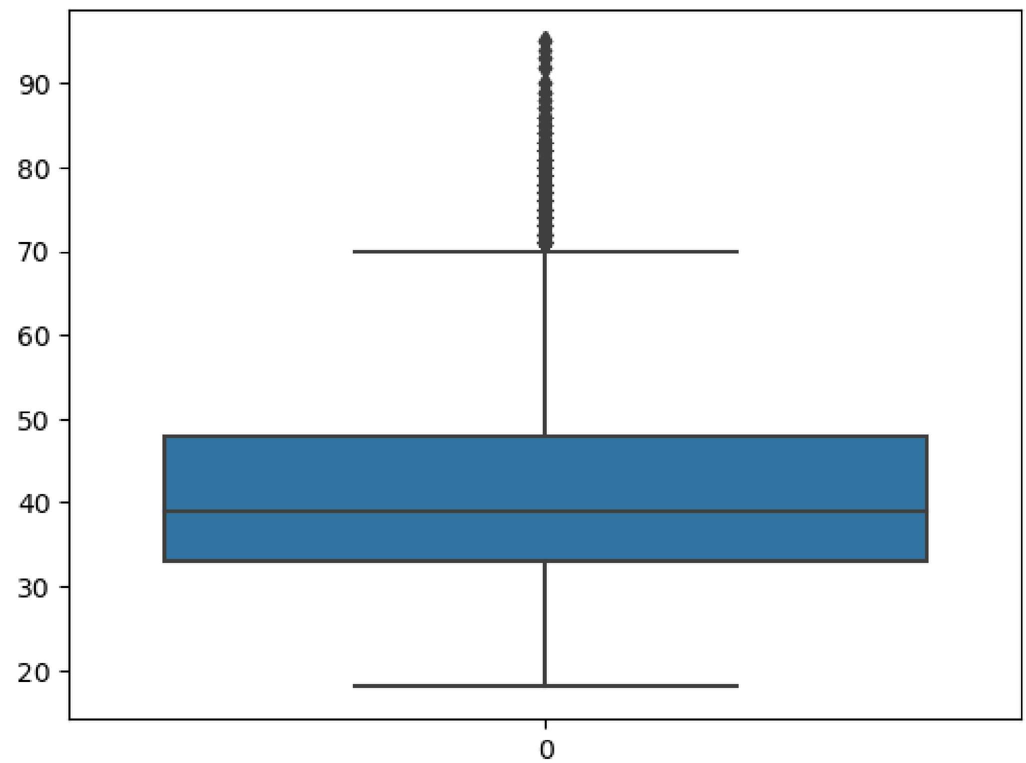
[57]:
```
#describe the age variable in inp1.
inp1.age.describe()
```

[57]:
```
count    45161.000000
mean        40.935763
std         10.618790
min         18.000000
25%         33.000000
50%         39.000000
75%         48.000000
max         95.000000
Name: age, dtype: float64
```

[58]:
```
#plot the histogram of age variable.
inp1.age.plot.hist()
plt.show()
```



[59]:
```
#plot the boxplot of age variable.
sns.boxplot(inp1.age)
plt.show()
```

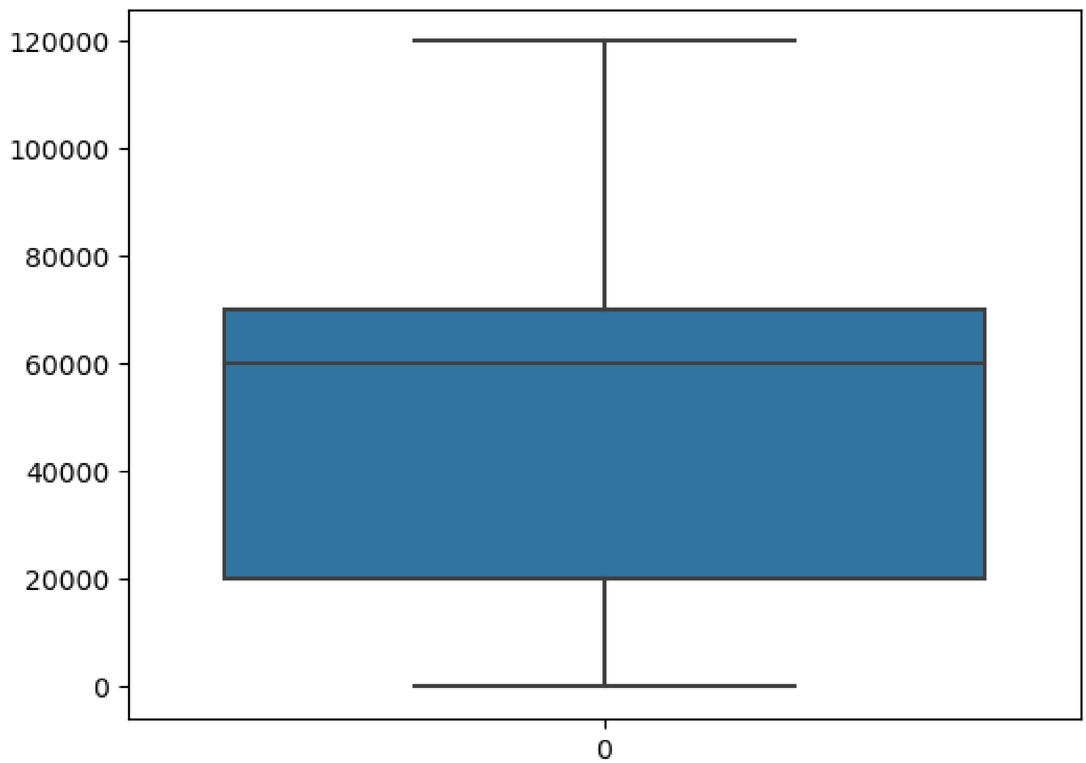


Salary variable

[61]:
```
#describe the salary variable of inp1.
inp1.salary.describe()
```

[61]:
```
count     45161.000000
mean      57004.849317
std       32087.698810
min           0.000000
25%       20000.000000
50%       60000.000000
75%       70000.000000
max      120000.000000
Name: salary, dtype: float64
```

[62]:
```
#plot the boxplot of salary variable.
sns.boxplot(inp1.salary)
plt.show()
```



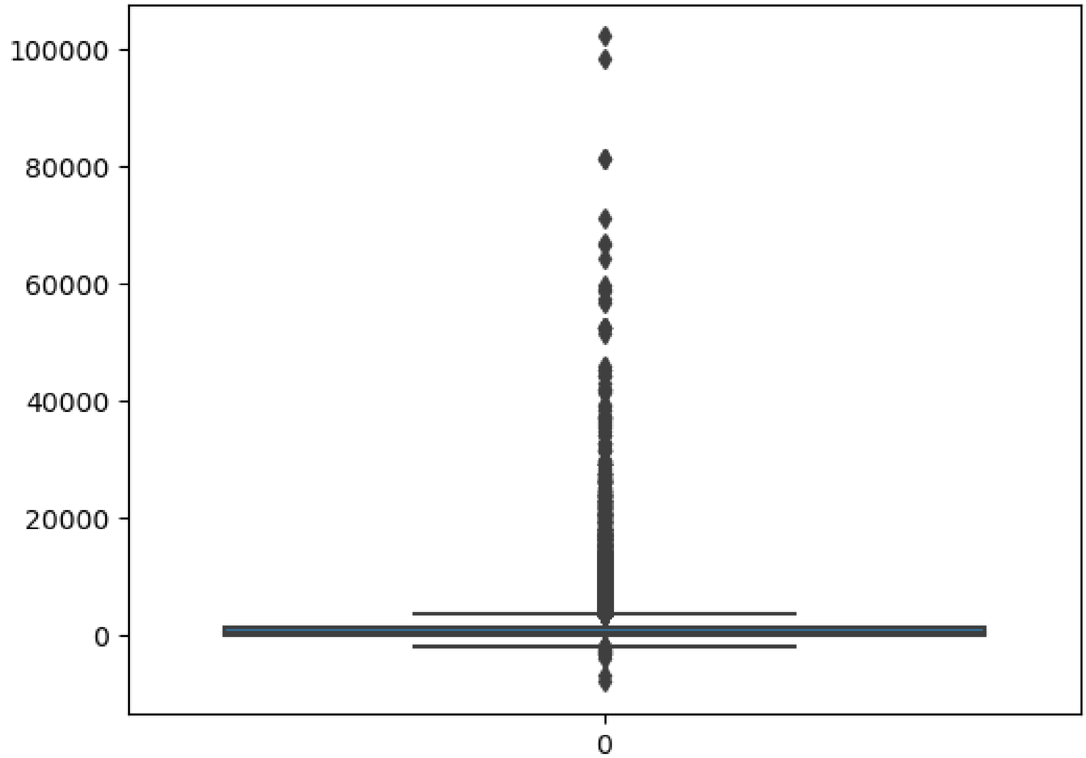
Balance variable

[64]:
```
#describe the balance variable of inp1.
inp1.balance.describe()
```

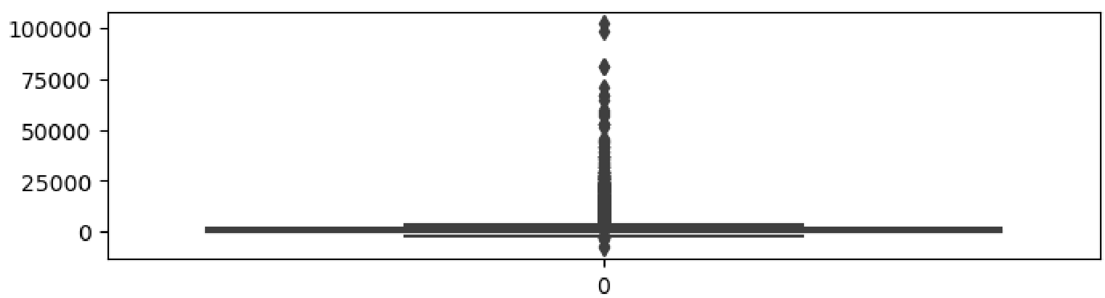
[64]:
```
count     45161.000000
mean       1362.850690
std        3045.939589
min       -8019.000000
25%          72.000000
50%         448.000000
75%        1428.000000
max      102127.000000
Name: balance, dtype: float64
```

[65]:
```
#plot the boxplot of balance variable.

sns.boxplot(inp1.balance)
plt.show()
```



[66]:
```
#plot the boxplot of balance variable after scaling in 8:2.
plt.figure(figsize=[8,2])
sns.boxplot(inp1.balance)
plt.show()
```



[67]:
```
#print the quantile (0.5, 0.7, 0.9, 0.95 and 0.99) of balance variable
inp1.balance.quantile([0.5, 0.7, 0.9, 0.95, 0.99])
```

[67]:
```
0.50      448.0
0.70     1126.0
```

```
0.90     3576.0
0.95     5769.0
0.99    13173.4
Name: balance, dtype: float64
```

[68]:
```
#describe the inp1 dataset for balance variable to be greater than 15000 in
↪inp1.
inp1[inp1.balance>15000].describe()
```

[68]:
```
              age         salary        balance         day    campaign  \
count  351.000000     351.000000     351.000000  351.000000  351.000000
mean    45.341880   70008.547009   24295.780627   16.022792    2.749288
std     12.114333   34378.272805   12128.560693    8.101819    3.036886
min     23.000000       0.000000   15030.000000    1.000000    1.000000
25%     35.000000   50000.000000   17074.000000    9.000000    1.000000
50%     44.000000   60000.000000   20723.000000   18.000000    2.000000
75%     55.000000  100000.000000   26254.000000   21.000000    3.000000
max     84.000000  120000.000000  102127.000000   31.000000   31.000000

            pdays    previous
count   62.000000  351.000000
mean   188.516129    0.555556
std    118.796388    1.784590
min     31.000000    0.000000
25%     96.250000    0.000000
50%    167.500000    0.000000
75%    246.500000    0.000000
max    589.000000   23.000000
```

0.2.5 Segment- 6, Standardising values

Checklist for data standardization exercises: - **Standardise units**: Ensure all observations under one variable are expressed in a common and consistent unit, e.g., convert lbs to kg, miles/hr to km/hr, etc. - **Scale values if required**: Make sure all the observations under one variable have a common scale. - **Standardise precision** for better presentation of data, e.g., change 4.5312341 kg to 4.53 kg. - **Remove extra characters** such as common prefixes/suffixes, leading/trailing/multiple spaces, etc. These are irrelevant to analysis. - **Standardise case**: String variables may take various casing styles, e.g., UPPERCASE, lowercase, Title Case, Sentence case, etc. - **Standardise format**: It is important to standardise the format of other elements such as date, name, etce.g., change 23/10/16 to 2016/10/23, "Modi, Narendra" to "Narendra Modi", etc.

Duration variable

[72]:
```
inp1.duration.head(10)
```

[72]:
```
0    261 sec
1    151 sec
2     76 sec
```

```
3      92 sec
4     198 sec
5     139 sec
6     217 sec
7     380 sec
8      50 sec
9      55 sec
Name: duration, dtype: object
```

[73]:
```
#describe the duration variable of inp1
inp1.duration.describe()
```

[73]:
```
count       45161
unique       2646
top       1.5 min
freq          138
Name: duration, dtype: object
```

[74]:
```
#convert the duration variable into single unit i.e. minutes. and remove the␣
 ↪sec or min prefix.
inp1.duration=inp1.duration.apply(lambda x: float(x.split()[0])/60 if x.
 ↪find("sec")> 0 else float(x.split()[0]) )
```

[75]:
```
#describe the duration variable
inp1.duration.describe()
```

[75]:
```
count    45161.000000
mean         4.302774
std          4.293129
min          0.000000
25%          1.716667
50%          3.000000
75%          5.316667
max         81.966667
Name: duration, dtype: float64
```

0.3 Session- 3, Univariate Analysis

0.3.1 Segment- 2, Categorical unordered univariate analysis

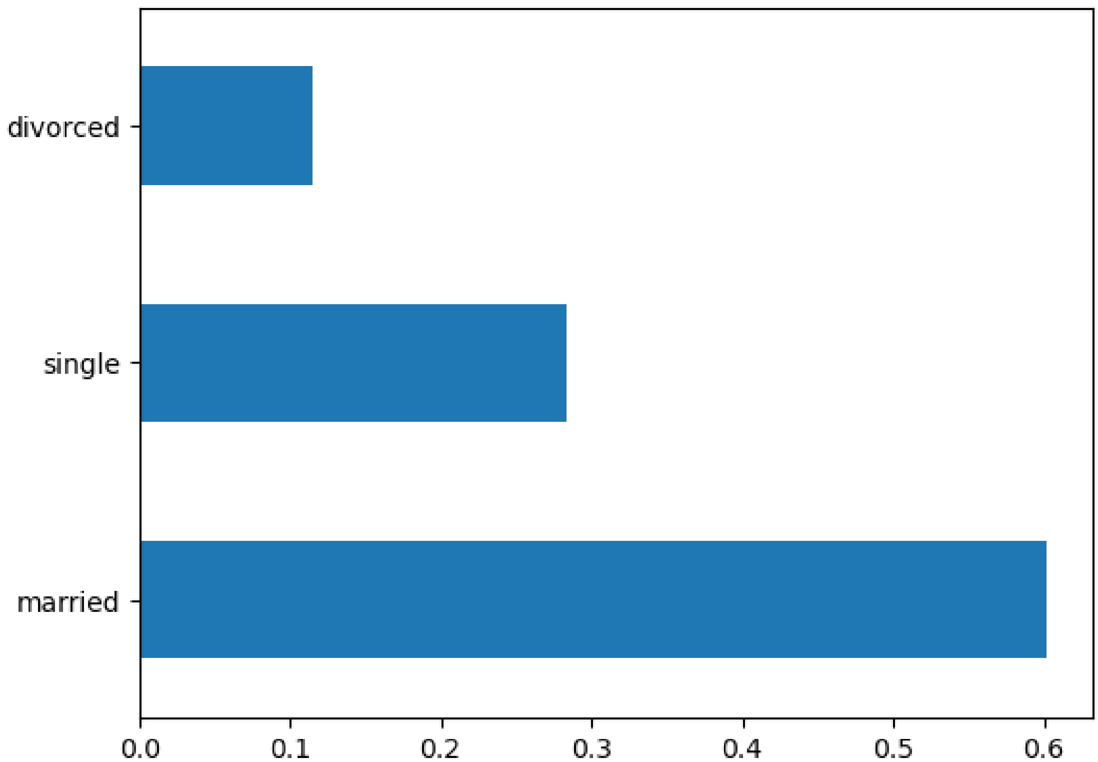
Unordered data do not have the notion of high-low, more-less etc. Example: - Type of loan taken by a person = home, personal, auto etc. - Organisation of a person = Sales, marketing, HR etc. - Job category of persone. - Marital status of any one.

Marital status

[80]:
```
#calculate the percentage of each marital status category.
inp1.marital.value_counts(normalize= True)
```

[80]:
```
married     0.601957
single      0.282943
divorced    0.115099
Name: marital, dtype: float64
```

[81]:
```
#plot the bar graph of percentage marital status categories
inp1.marital.value_counts(normalize= True).plot.barh()
plt.show()
```



Job

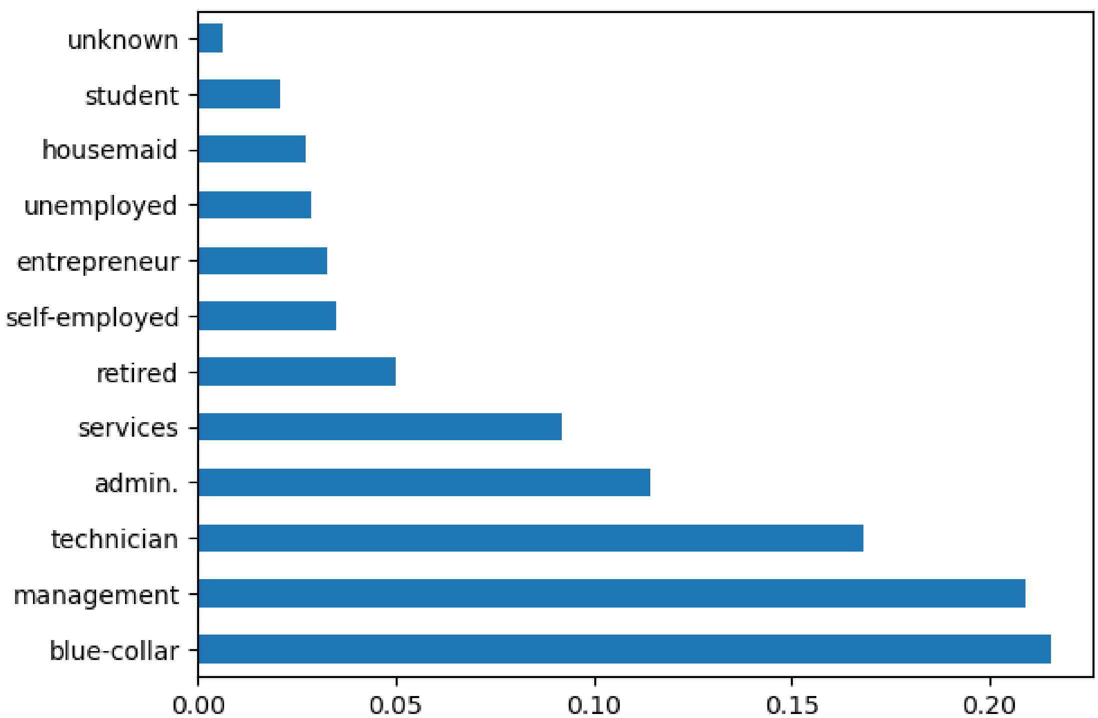
[83]:
```
#calculate the percentage of each job status category.
inp1.job.value_counts(normalize= True)
```

[83]:
```
blue-collar      0.215274
management       0.209273
technician       0.168043
admin.           0.114369
services         0.091849
retired          0.050087
self-employed    0.034853
entrepreneur     0.032860
```

```
unemployed       0.028830
housemaid        0.027413
student          0.020770
unknown          0.006377
Name: job, dtype: float64
```

[84]:
```
#plot the bar graph of percentage job categories
inp1.job.value_counts(normalize= True).plot.barh()
plt.plot()
```

[84]: []



0.3.2 Segment- 3, Categorical ordered univariate analysis

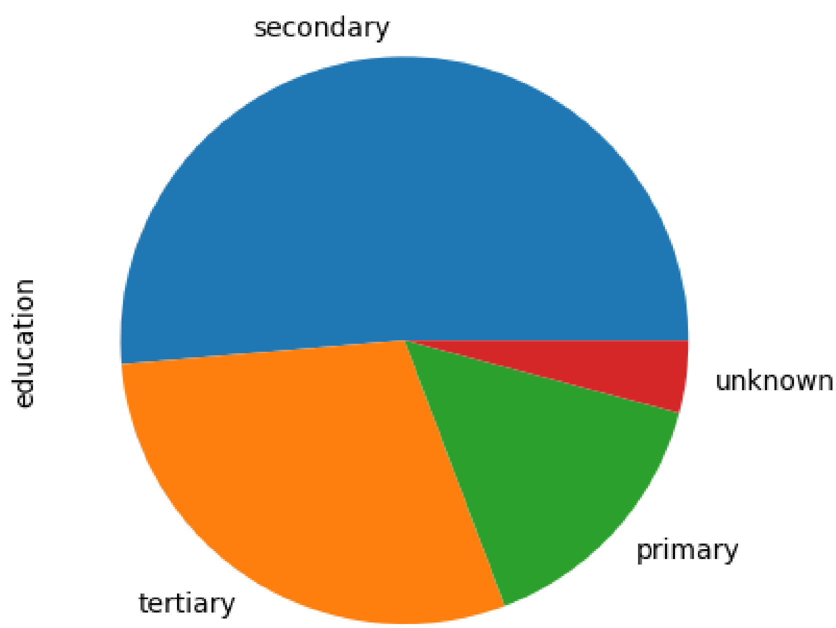
Ordered variables have some kind of ordering. Some examples of bank marketing dataset are: - Age group= <30, 30-40, 40-50 and so on. - Month = Jan-Feb-Mar etc. - Education = primary, secondary and so on.

Education

[88]:
```
#calculate the percentage of each education category.
inp1.education.value_counts(normalize= True)
```

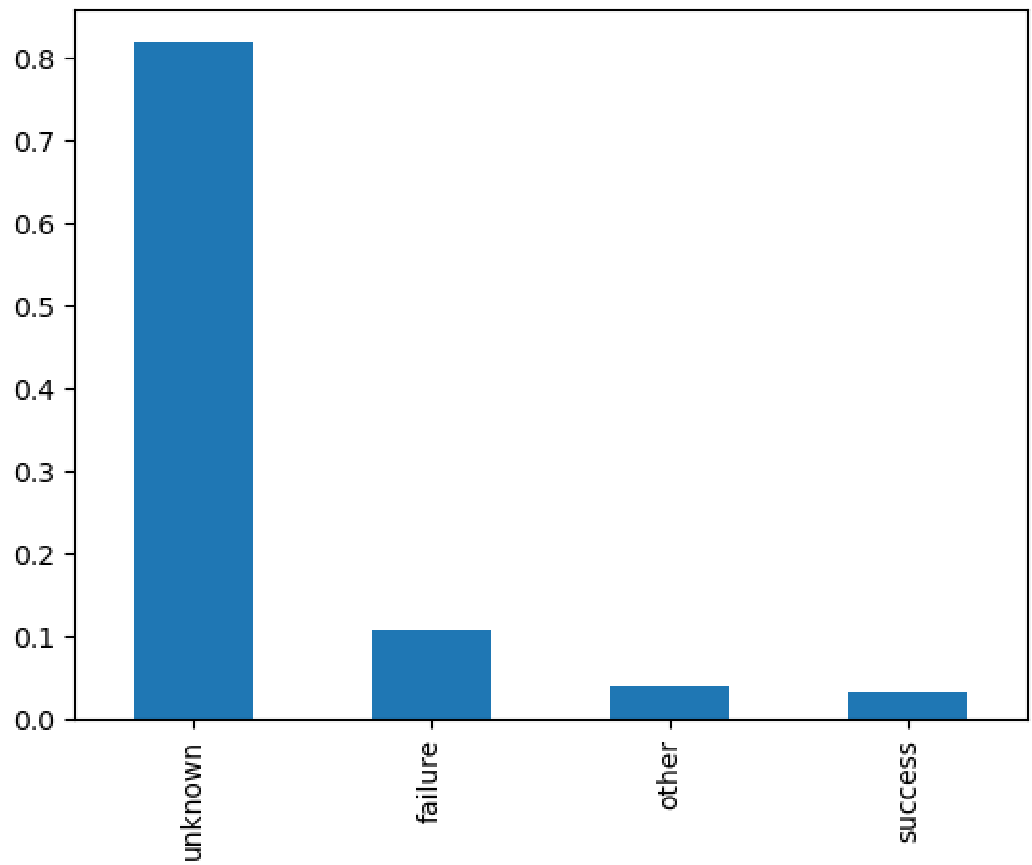
[88]:
```
secondary    0.513275
tertiary     0.294192
primary      0.151436
unknown      0.041097
Name: education, dtype: float64
```

[89]:
```
#plot the pie chart of education categories
inp1.education.value_counts(normalize= True).plot.pie()
plt.show()
```

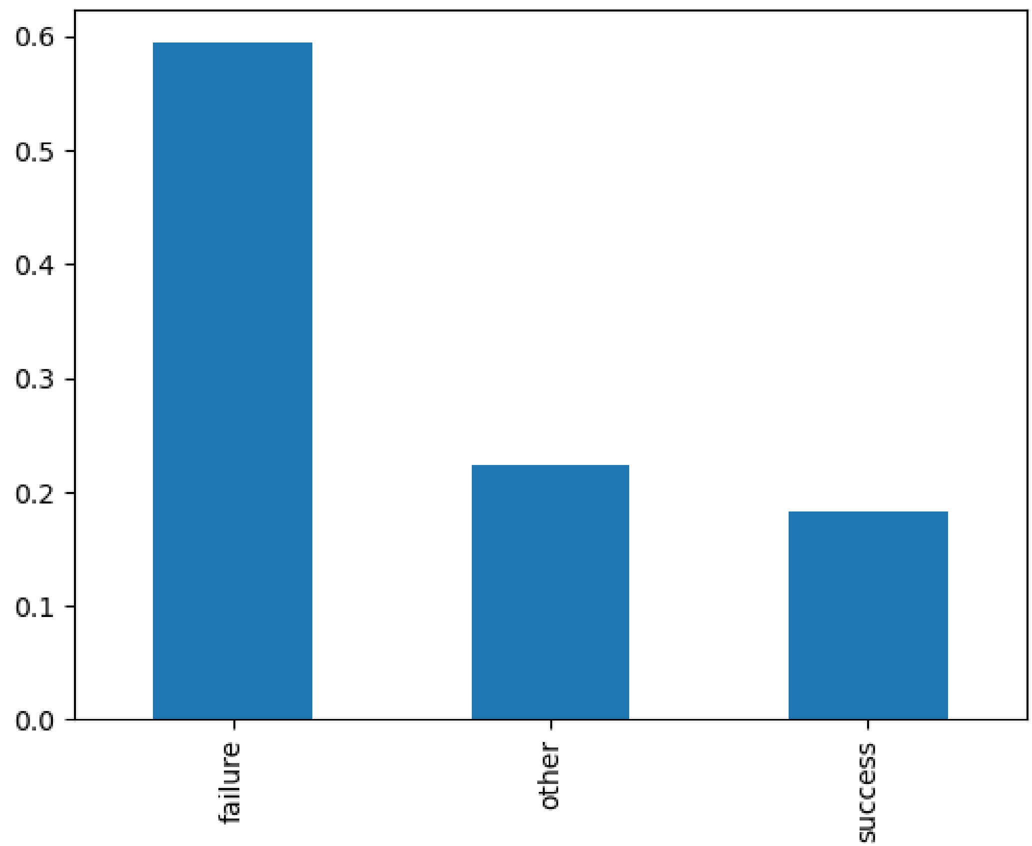


poutcome

[91]:
```
#calculate the percentage of each poutcome category.
inp1.poutcome.value_counts(normalize= True).plot.bar()
plt.show()
```



[92]:
```
inp1[-(inp1.poutcome=="unknown")].poutcome.value_counts(normalize= True).plot.
    ↪bar()
plt.show()
```

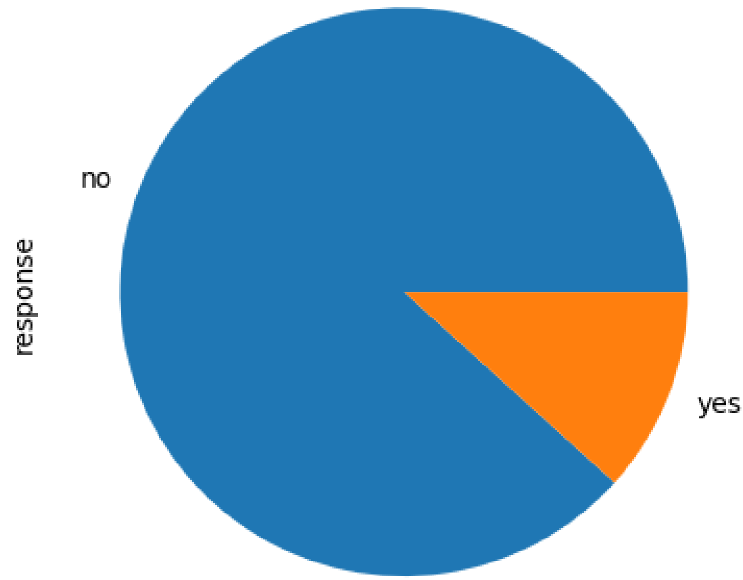


Response the target variable

[94]:
```
#calculate the percentage of each response category.
inp1.response.value_counts(normalize= True)
```

[94]:
```
no     0.882974
yes    0.117026
Name: response, dtype: float64
```

[95]:
```
#plot the pie chart of response categories
inp1.response.value_counts(normalize= True).plot.pie()
plt.show()
```

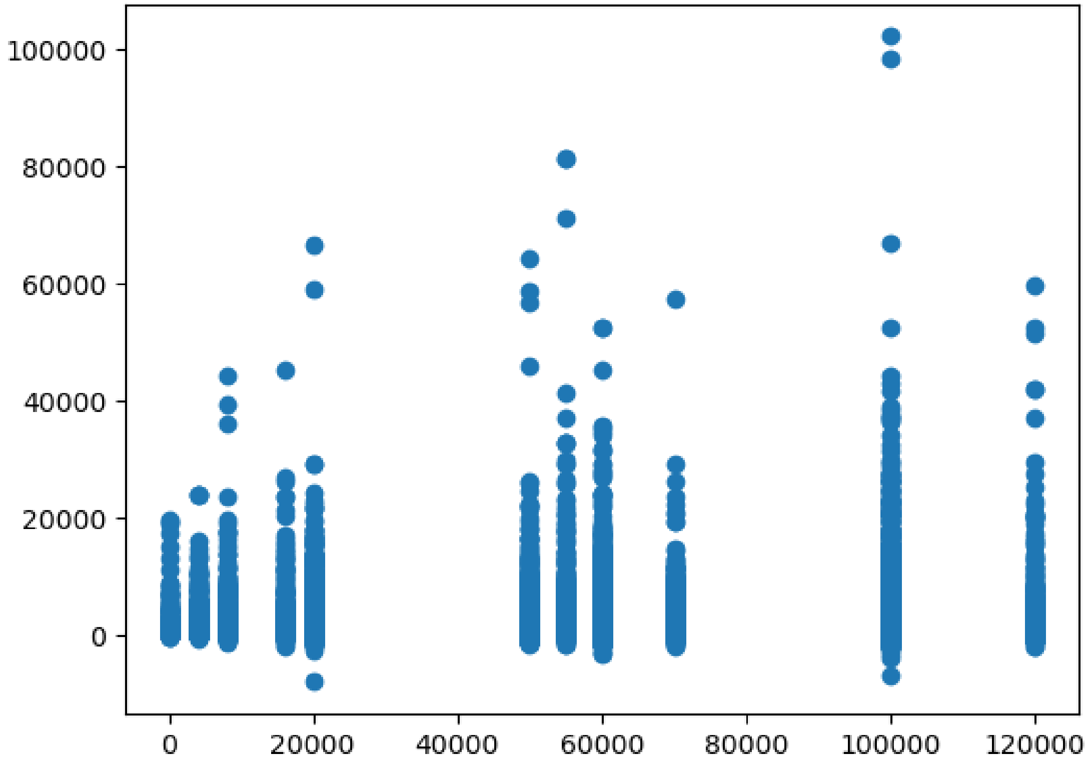


0.4 Session- 4, Bivariate and Multivariate Analysis

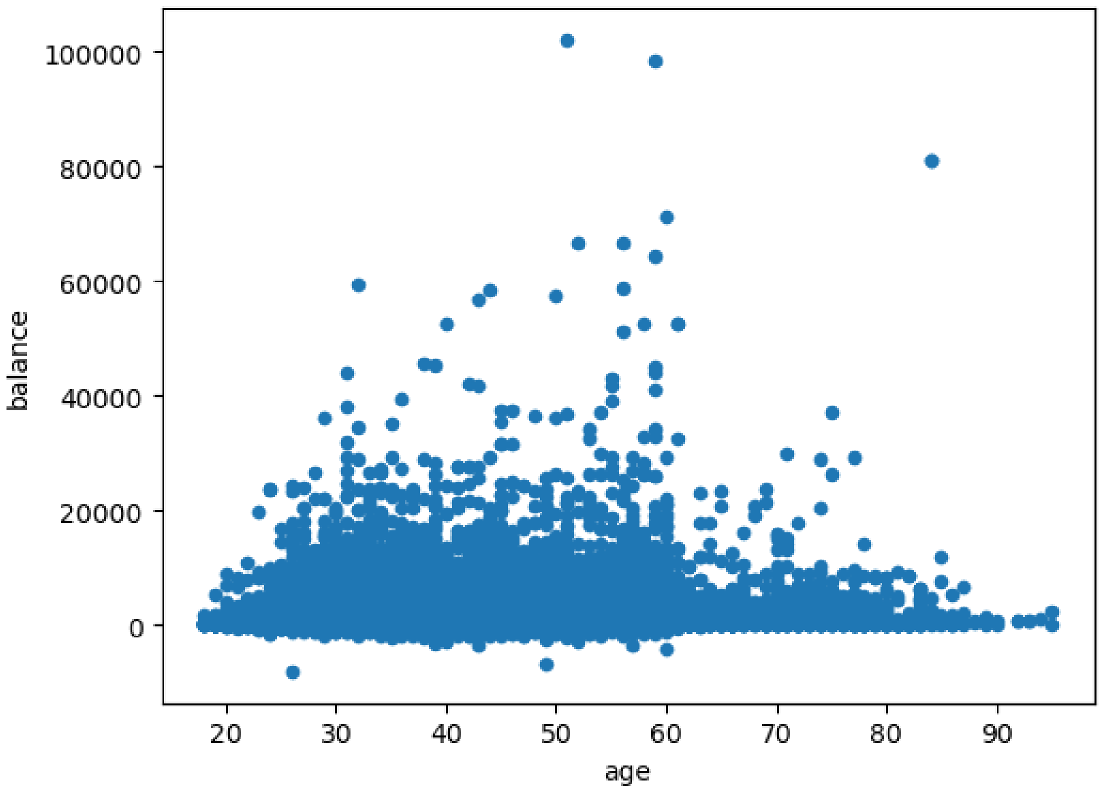
0.4.1 Segment-2, Numeric- numeric analysis

There are three ways to analyse the numeric- numeric data types simultaneously. - **Scatter plot**: describes the pattern that how one variable is varying with other variable. - **Correlation matrix**: to describe the linearity of two numeric variables. - **Pair plot**: group of scatter plots of all numeric variables in the data frame.

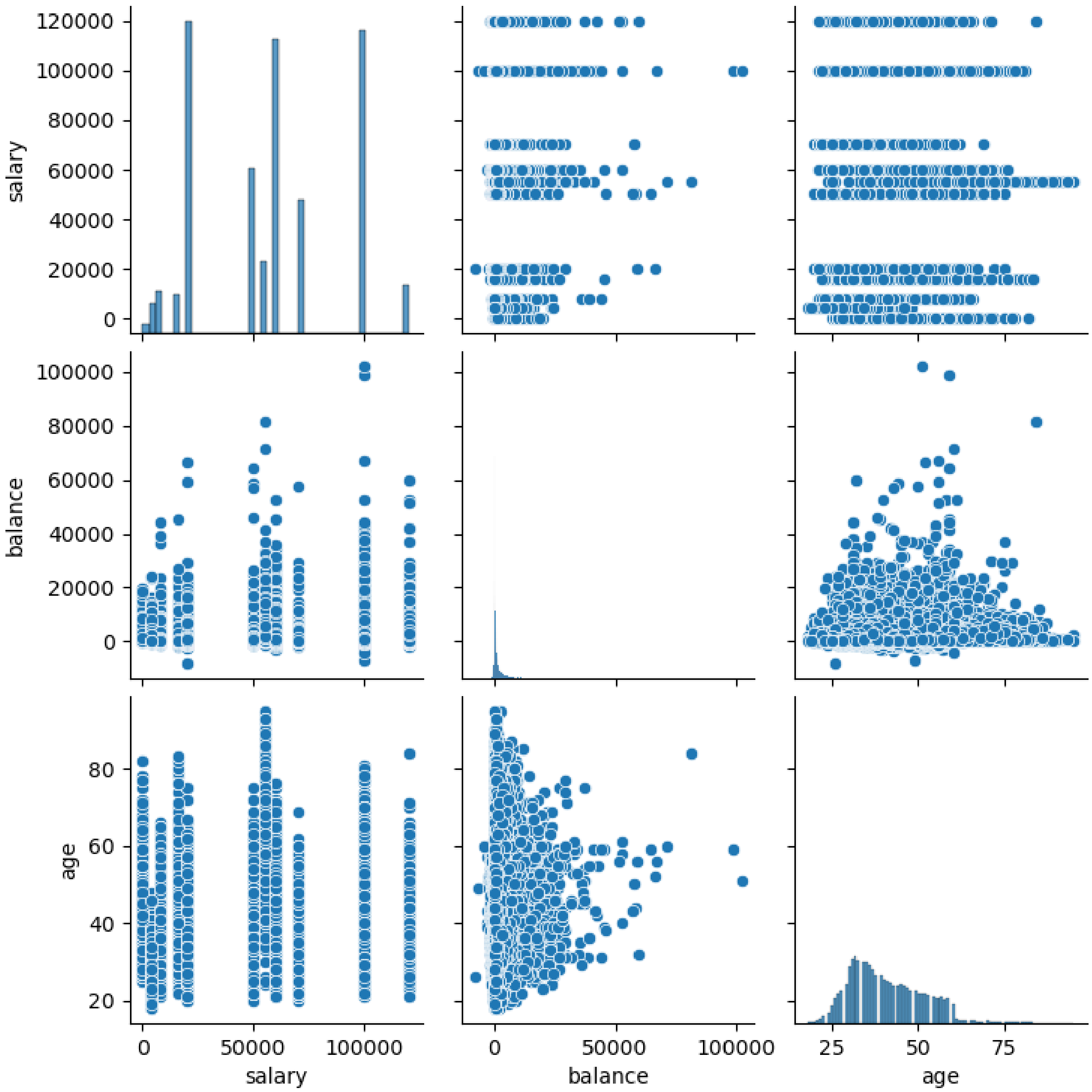
[99]:
```
#plot the scatter plot of balance and salary variable in inp1
plt.scatter(inp1.salary, inp1.balance)
plt.show()
```



[100]:
```
#plot the scatter plot of balance and age variable in inp1
inp1.plot.scatter(x='age', y='balance')
plt.show()
```

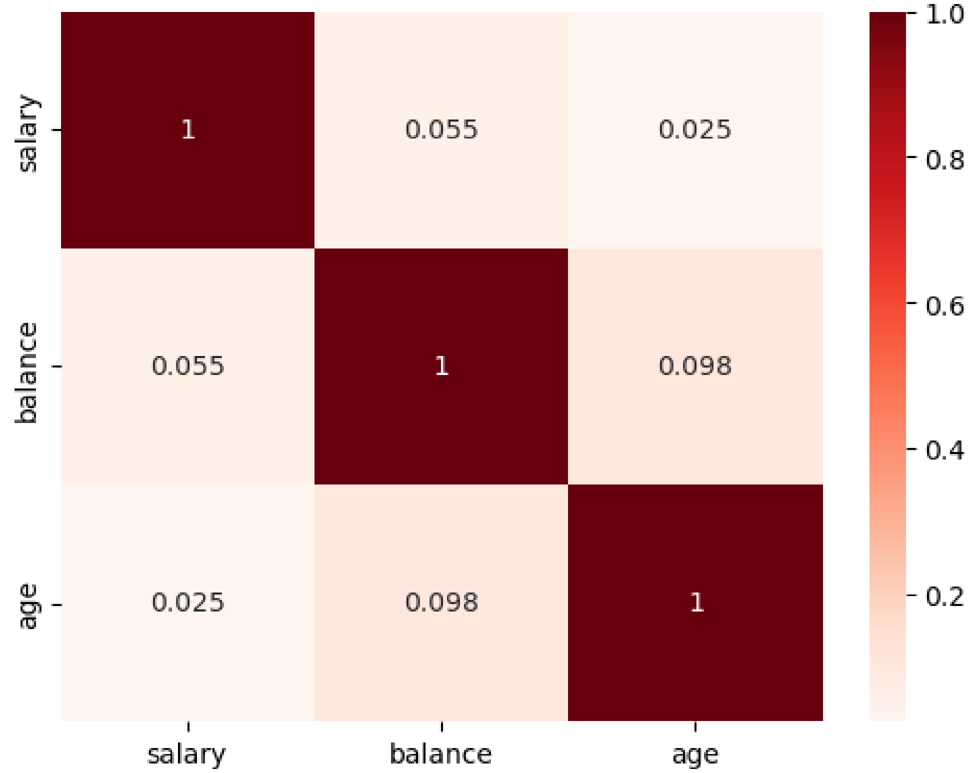


[101]:
```
#plot the pair plot of salary, balance and age in inp1 dataframe.
sns.pairplot(data=inp1, vars=["salary","balance", "age"])
plt.show()
```



Correlation heat map

[103]:
```
#plot the correlation matrix of salary, balance and age in inp1 dataframe.
sns.heatmap( inp1[["salary","balance", "age"]].corr(), annot= True, cmap=␣
↪"Reds")
plt.show()
```



0.4.2 Segment- 4, Numerical categorical variable

Salary vs response

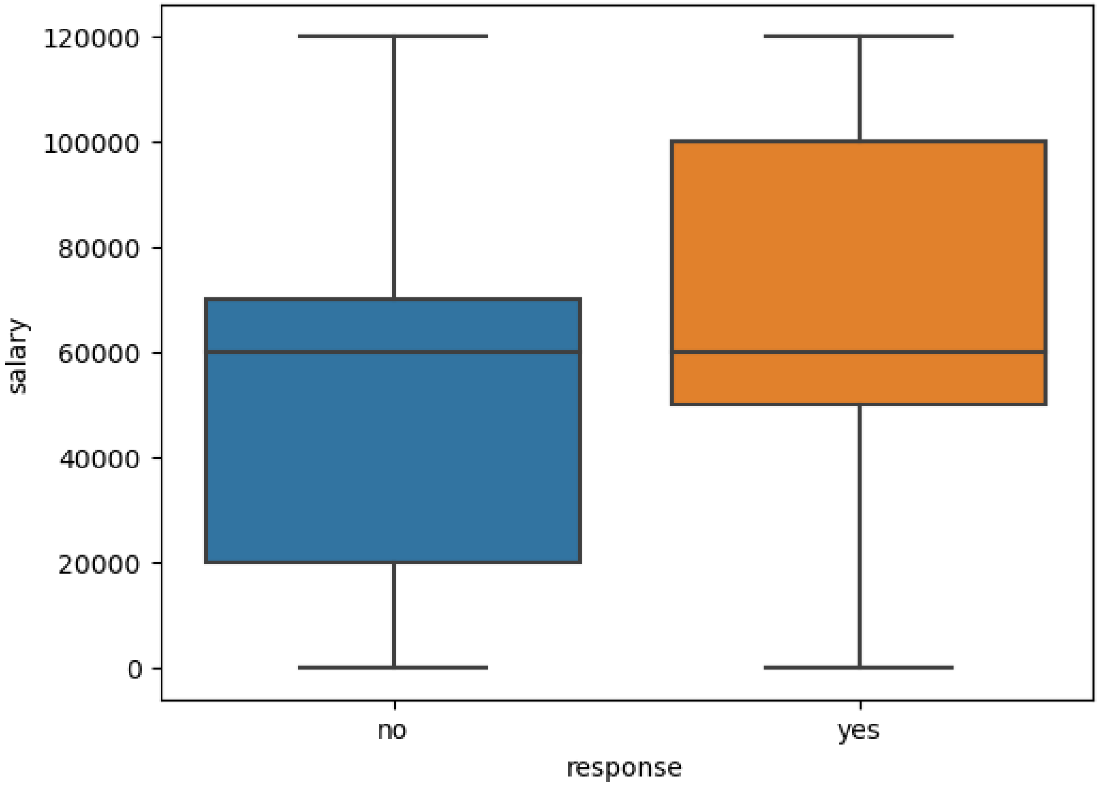
[106]:
```
#groupby the response to find the mean of the salary with response no & yes␣
↪seperatly.
inp1.groupby("response")["salary"].mean()
```

[106]:
```
response
no     56769.510482
yes    58780.510880
Name: salary, dtype: float64
```

[107]:
```
#groupby the response to find the median of the salary with response no & yes␣
↪seperatly.
inp1.groupby("response")["salary"].median()
```

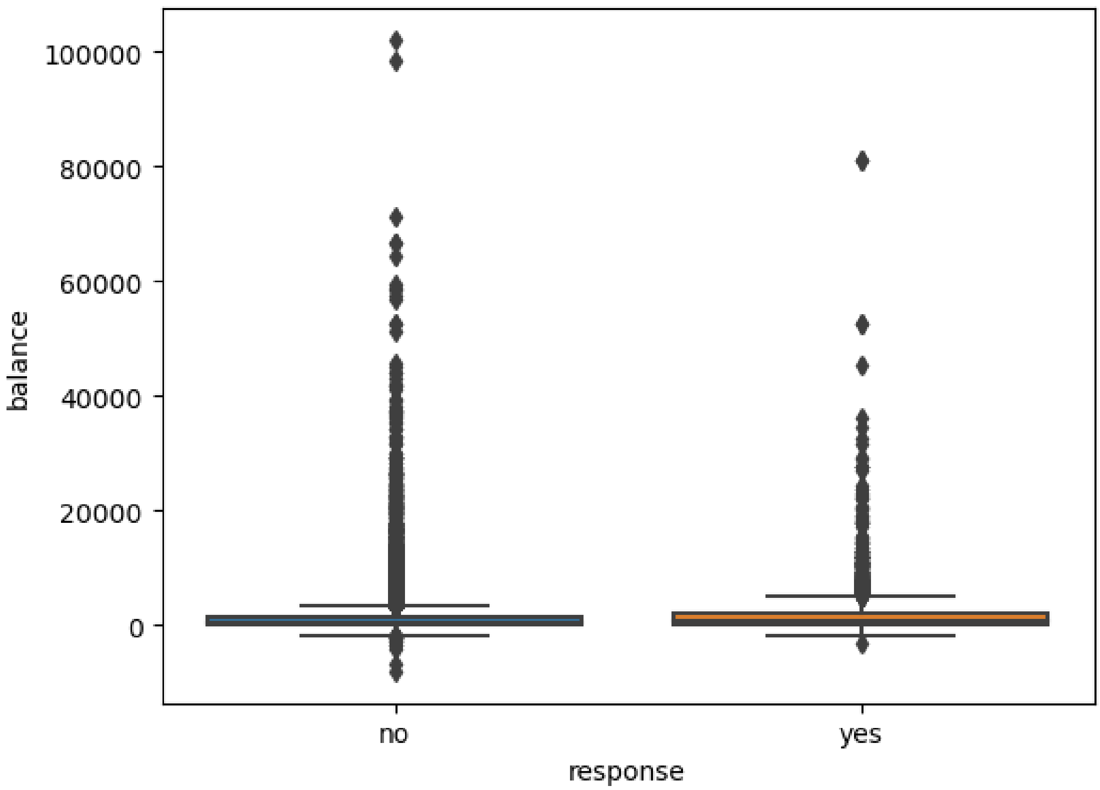
[107]:
```
response
no     60000.0
yes    60000.0
Name: salary, dtype: float64
```

[108]:
```
#plot the box plot of salary for yes & no responses.
sns.boxplot(data=inp1,x="response", y="salary")
plt.show()
```



Balance vs response

[110]:
```
#plot the box plot of balance for yes & no responses.
sns.boxplot(data=inp1,x="response", y="balance")
plt.show()
```



[111]:
```
#groupby the response to find the mean of the balance with response no & yes␣
  ↪seperatly.
inp1.groupby("response")["balance"].mean()
```

[111]:
```
response
no     1304.292281
yes    1804.681362
Name: balance, dtype: float64
```

[112]:
```
#groupby the response to find the median of the balance with response no & yes␣
  ↪seperatly.
inp1.groupby("response")["balance"].median()
```

[112]:
```
response
no     417.0
yes    733.0
Name: balance, dtype: float64
```

75th percentile

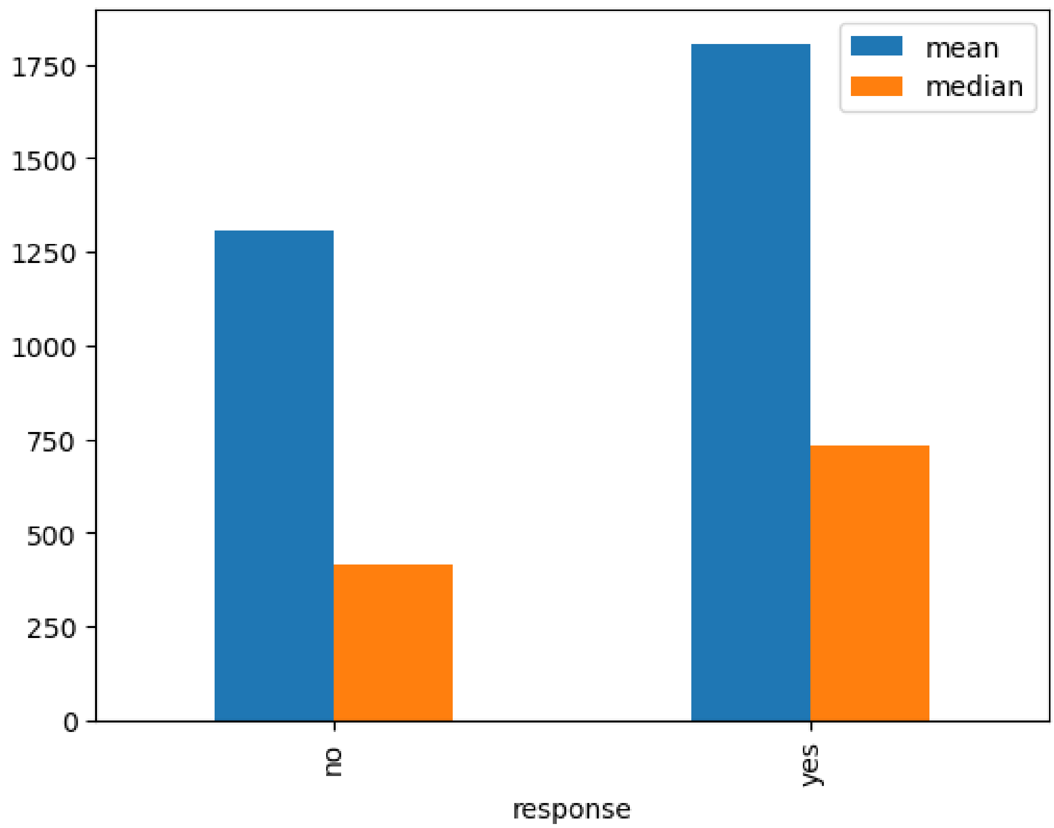
[114]:
```
#function to find the 75th percentile.
def p75(x):
    return np.quantile(x, 0.75)
```

[115]:
```
#calculate the mean, median and 75th percentile of balance with response
inp1.groupby("response")["balance"].aggregate(["mean","median",p75])
```

[115]:

response	mean	median	p75
no	1304.292281	417.0	1345.0
yes	1804.681362	733.0	2159.0

[116]:
```
#plot the bar graph of balance's mean an median with response.
inp1.groupby("response")["balance"].aggregate(["mean","median"]).plot.bar()
plt.show()
```



Education vs salary

[118]:
```
#groupby the education to find the mean of the salary education category.
inp1.groupby("education")["salary"].mean()
```

[118]:
```
education
primary      34232.343910
secondary    49731.449525
tertiary     82880.249887
unknown      46529.633621
Name: salary, dtype: float64
```

[119]:
```
#groupby the education to find the median of the salary for each education␣
↪category.
inp1.groupby("education")["salary"].median()
```

[119]:
```
education
primary       20000.0
secondary     55000.0
tertiary     100000.0
unknown       50000.0
Name: salary, dtype: float64
```

Job vs salary

[121]:
```
#groupby the job to find the mean of the salary for each job category.
inp1.groupby('job')['salary'].mean()
```

[121]:
```
job
admin.            50000.0
blue-collar       20000.0
entrepreneur     120000.0
housemaid         16000.0
management       100000.0
retired           55000.0
self-employed     60000.0
services          70000.0
student            4000.0
technician        60000.0
unemployed         8000.0
unknown               0.0
Name: salary, dtype: float64
```

[122]:
```
inp1.groupby('job')['salary'].median()
```

[122]:
```
job
admin.            50000.0
blue-collar       20000.0
entrepreneur     120000.0
housemaid         16000.0
management       100000.0
retired           55000.0
```

```
self-employed     60000.0
services          70000.0
student            4000.0
technician        60000.0
unemployed         8000.0
unknown               0.0
Name: salary, dtype: float64
```

0.4.3 Segment- 5, Categorical categorical variable

[124]:
```python
#create response_flag of numerical data type where response "yes"= 1, "no"= 0
inp1["response_flag"]=np.where(inp1.response=="yes", 1, 0)
inp1.response.value_counts()
```

[124]:
```
no     39876
yes     5285
Name: response, dtype: int64
```

[125]:
```python
inp1.response.value_counts(normalize= True)
```

[125]:
```
no     0.882974
yes    0.117026
Name: response, dtype: float64
```

[126]:
```python
inp1.response_flag.mean()
```

[126]:
```
0.1170257523084077
```

Education vs response rate

[128]:
```python
#calculate the mean of response_flag with different education categories.
inp1.groupby("education")["response_flag"].mean()
```

[128]:
```
education
primary      0.086416
secondary    0.105608
tertiary     0.150083
unknown      0.135776
Name: response_flag, dtype: float64
```

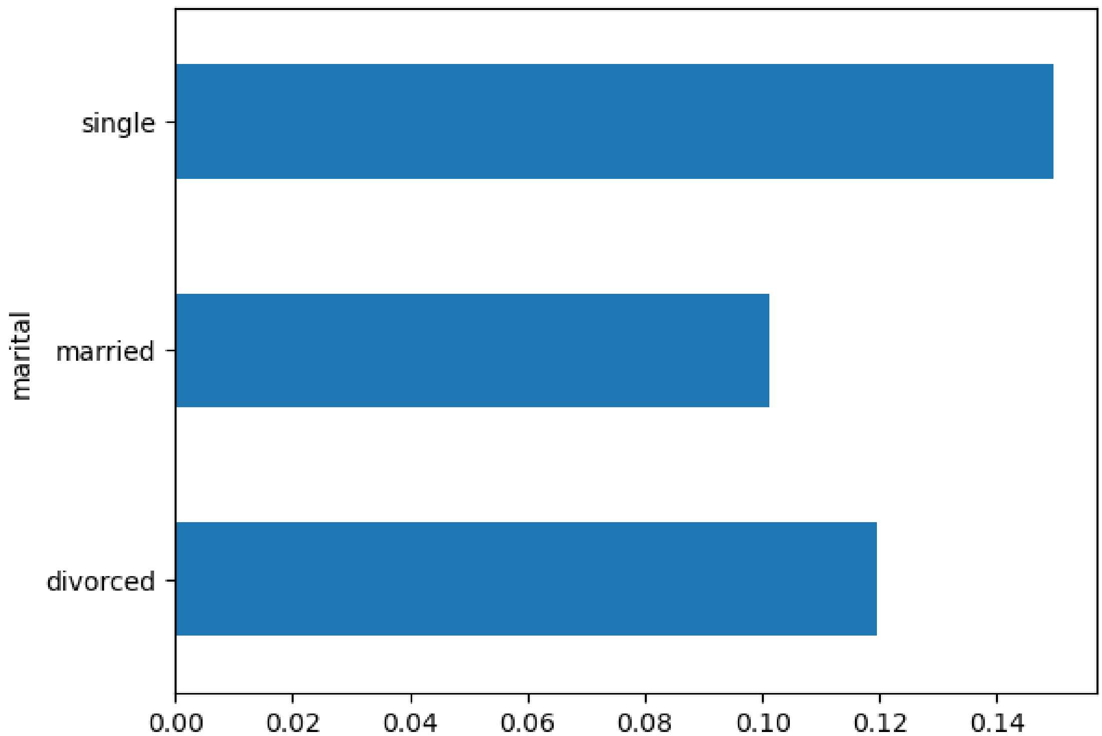
Marital vs response rate

[130]:
```python
#calculate the mean of response_flag with different marital status categories.
inp1.groupby(["marital"])["response_flag"].mean()
```

[130]:
```
marital
divorced    0.119469
```

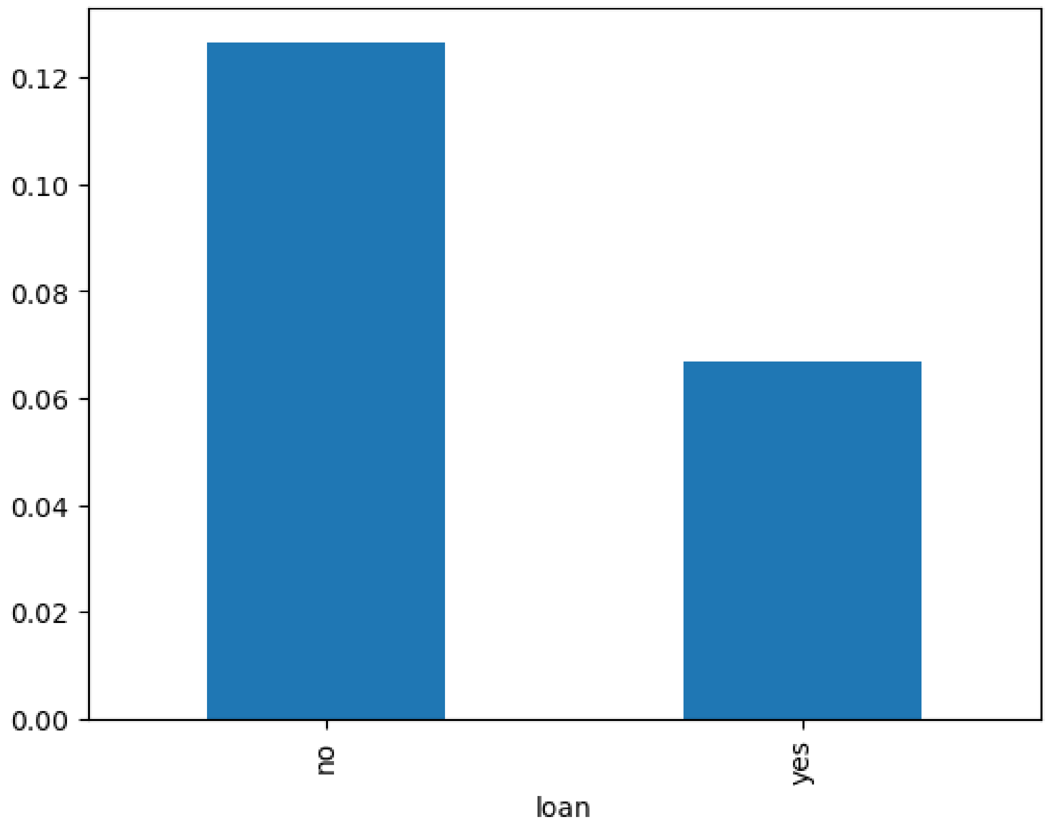
```
married     0.101269
single      0.149554
Name: response_flag, dtype: float64
```

[131]:
```
#plot the bar graph of marital status with average value of response_flag
inp1.groupby(["marital"])["response_flag"].mean().plot.barh()
plt.show()
```



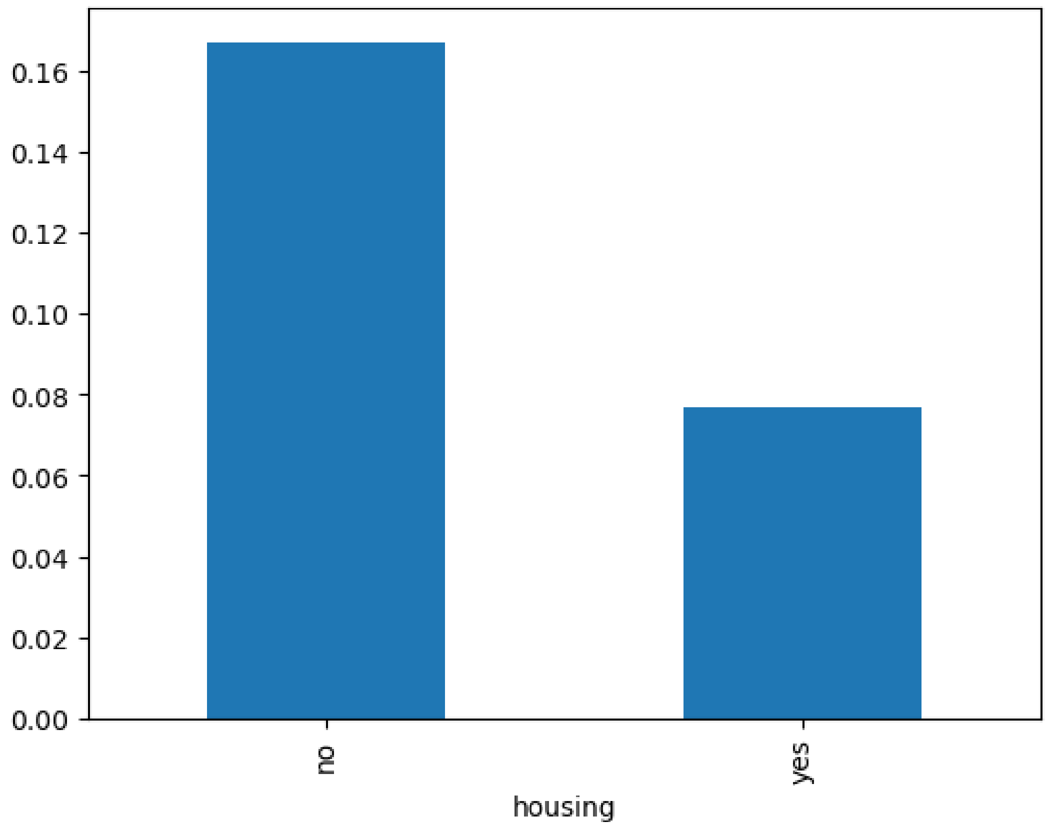
Loans vs response rate

[133]:
```
#plot the bar graph of personal loan status with average value of response_flag
inp1.groupby(["loan"])["response_flag"].mean().plot.bar()
plt.show()
```



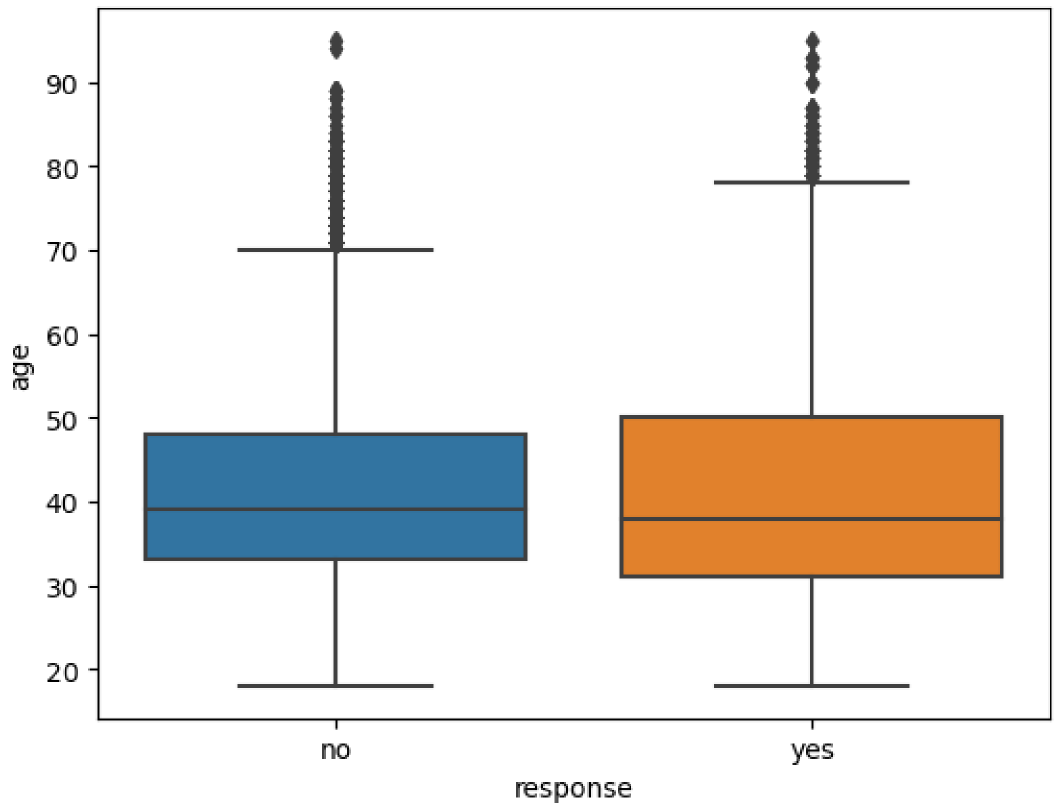
Housing loans vs response rate

[135]:
```
#plot the bar graph of housing loan status with average value of response_flag
inp1.groupby(["housing"])["response_flag"].mean().plot.bar()
plt.show()
```



Age vs response

[137]:
```
#plot the boxplot of age with response_flag
sns.boxplot(data=inp1, x="response",y="age")
plt.show()
```



making buckets from age columns

[139]:
```
#create the buckets of <30, 30-40, 40-50 50-60 and 60+ from age column.
pd.cut(inp1.age[:5],[0, 30, 40, 50, 60, 9999], labels=
 ↪["<30","30-40","40-50","50-60", "60+"])
```

[139]:
```
0    50-60
1    40-50
2    30-40
3    40-50
4    30-40
Name: age, dtype: category
Categories (5, object): ['<30' < '30-40' < '40-50' < '50-60' < '60+']
```

[140]:
```
inp1.age.head()
```

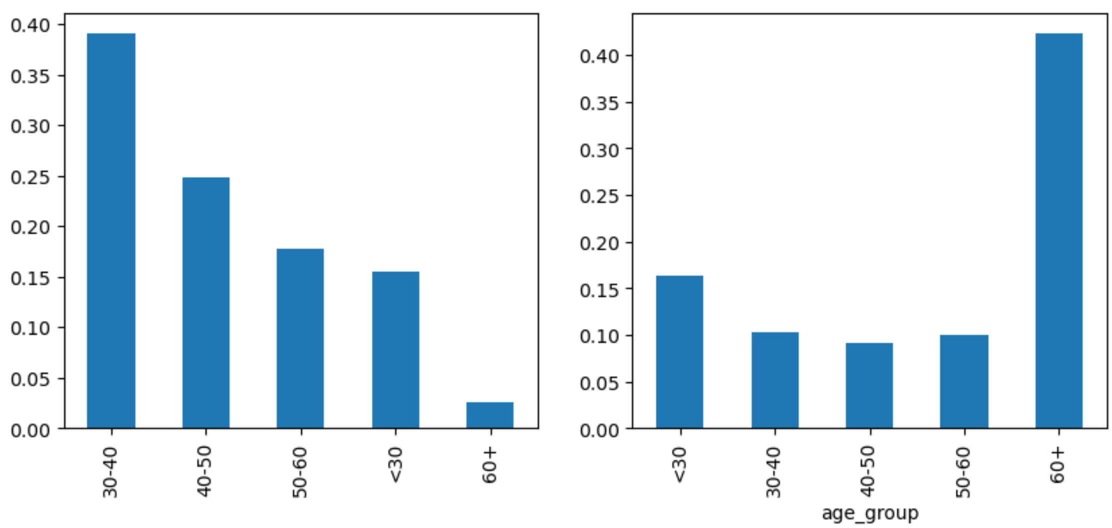
[140]:
```
0    58.0
1    44.0
2    33.0
3    47.0
4    33.0
```

```
Name: age, dtype: float64
```

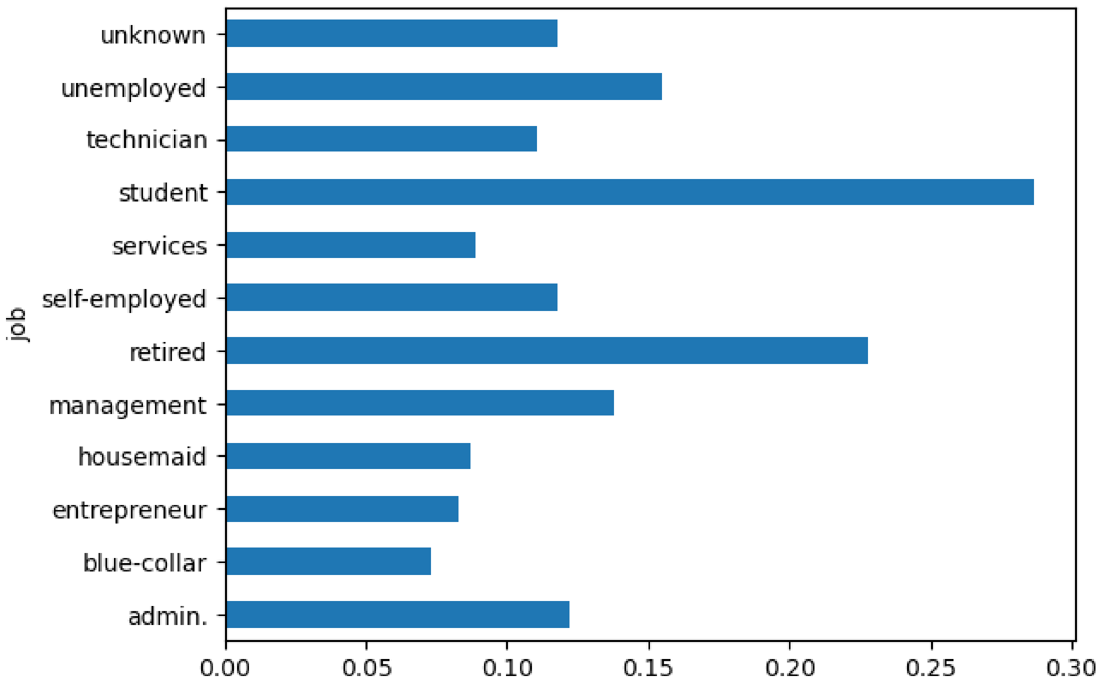
[141]:
```
inp1["age_group"]=pd.cut(inp1.age,[0, 30, 40, 50, 60, 9999], labels=
↪["<30","30-40","40-50","50-60", "60+"])
inp1.age_group.value_counts(normalize= True)
```

[141]:
```
30-40    0.391090
40-50    0.248688
50-60    0.178406
<30      0.155555
60+      0.026262
Name: age_group, dtype: float64
```

[142]:
```
#plot the percentage of each buckets and average values of response_flag in
↪each buckets. plot in subplots.
plt.figure(figsize=[10,4])
plt.subplot(1, 2, 1)
inp1.age_group.value_counts(normalize= True).plot.bar()
plt.subplot(1, 2, 2)
inp1.groupby(['age_group'])['response_flag'].mean().plot.bar()
plt.show()
```



[143]:
```
#plot the bar graph of job categories with response_flag mean value.
inp1.groupby(['job'])['response_flag'].mean().plot.barh()
plt.show()
```



0.4.4 Segment-6, Multivariate analysis

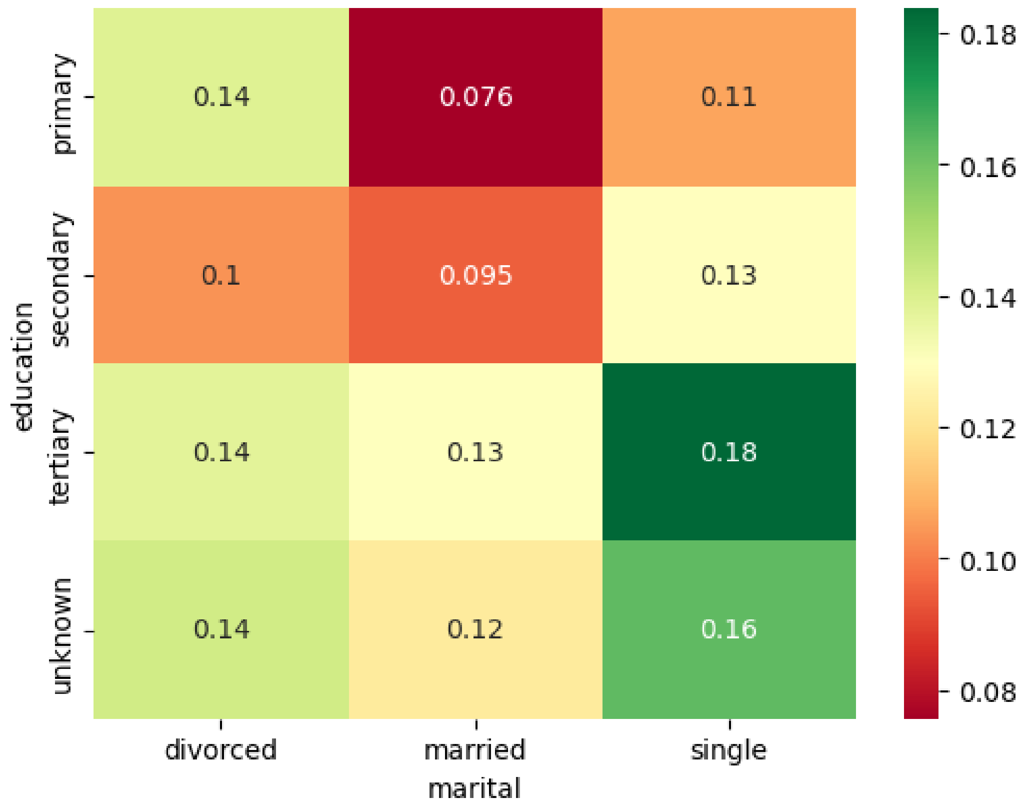
Education vs marital vs response

[146]:
```
res=pd.pivot_table(data=inp1, index="education", columns="marital",␣
↪values="response_flag")
res
```

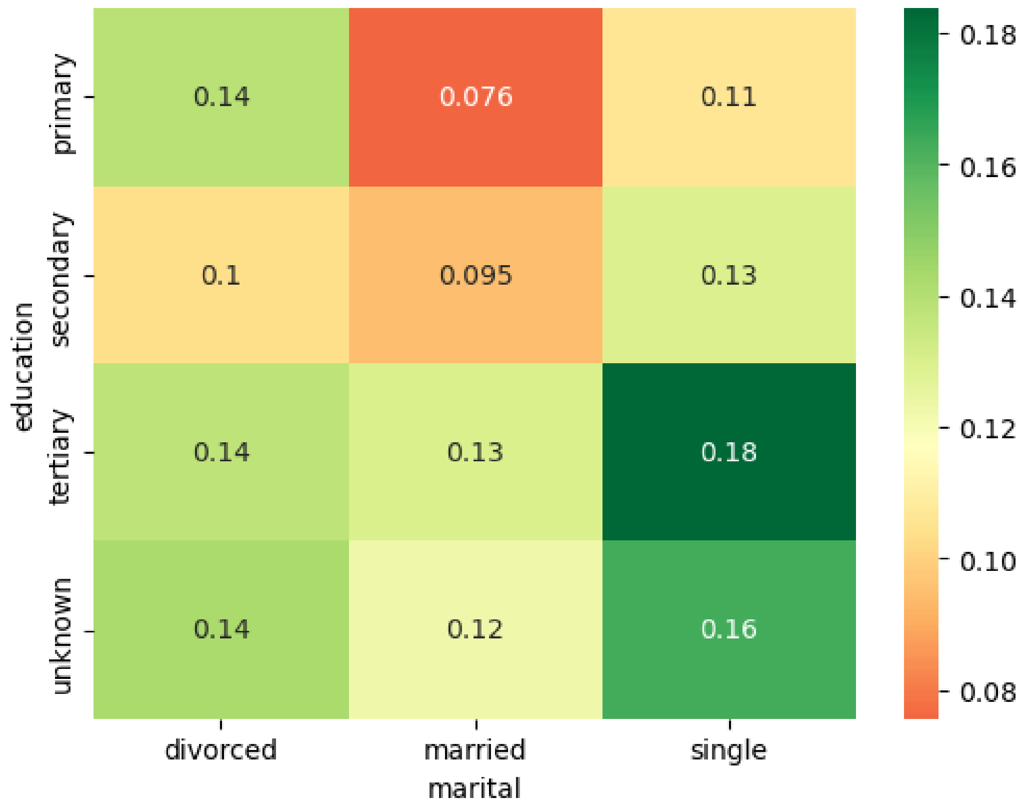
[146]:

marital	divorced	married	single
education			
primary	0.138852	0.075601	0.106808
secondary	0.103559	0.094650	0.129271
tertiary	0.137415	0.129835	0.183737
unknown	0.142012	0.122519	0.162879

[147]:
```
#create heat map of education vs marital vs response_flag
sns.heatmap(res, annot= True, cmap="RdYlGn")
plt.show()
```



[148]:
```
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()
```



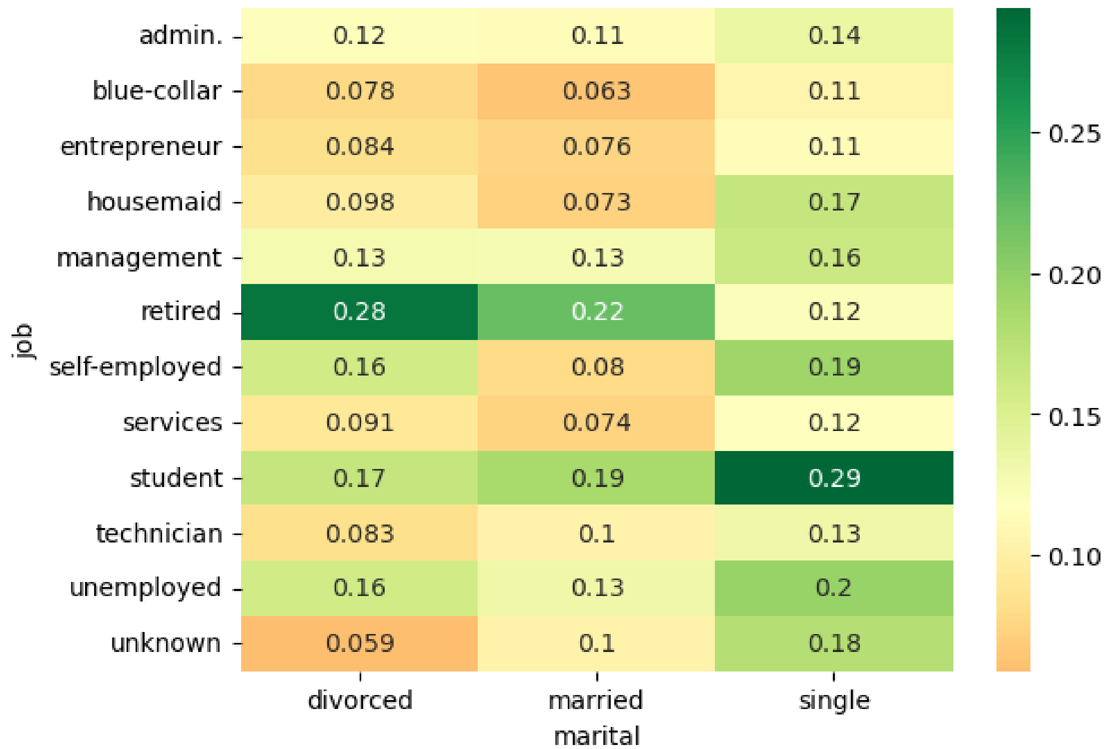
Job vs marital vs response

[150]:
```
res=pd.pivot_table(data=inp1, index="job", columns="marital",
values="response_flag")
res
```

[150]:

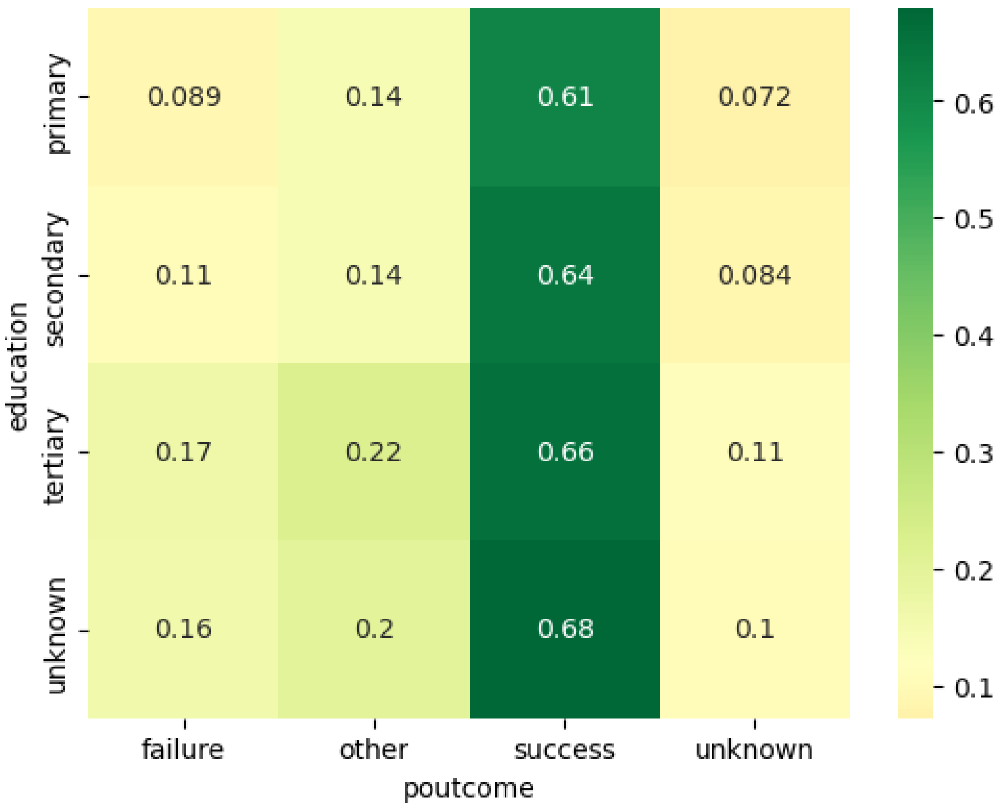
marital	divorced	married	single
job			
admin.	0.120160	0.113383	0.136153
blue-collar	0.077644	0.062778	0.105760
entrepreneur	0.083799	0.075843	0.113924
housemaid	0.097826	0.072527	0.166667
management	0.127928	0.126228	0.162254
retired	0.283688	0.220682	0.120370
self-employed	0.158273	0.079637	0.191874
services	0.091241	0.074105	0.117696
student	0.166667	0.185185	0.293850
technician	0.083243	0.102767	0.132645
unemployed	0.157895	0.132695	0.195000
unknown	0.058824	0.103448	0.176471

[151]:
```
#create the heat map of Job vs marital vs response_flag.
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()
```



Education vs poutcome vs response

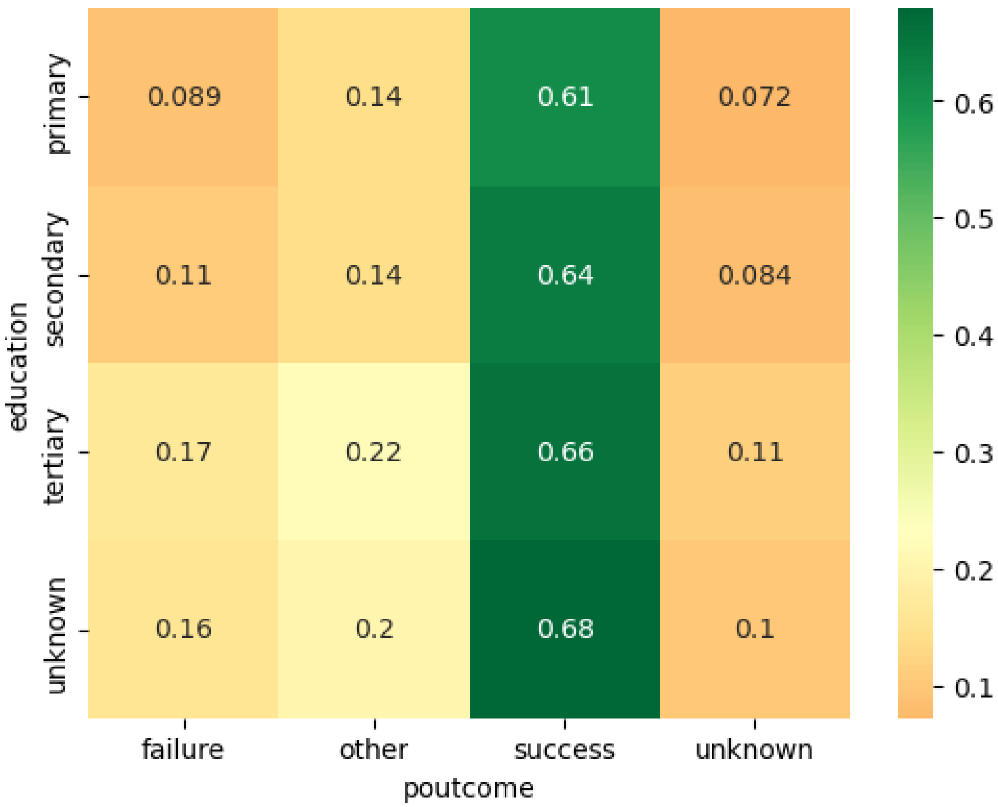
[153]:
```
#create the heat map of education vs poutcome vs response_flag.
res=pd.pivot_table(data=inp1, index="education", columns="poutcome",␣
↪values="response_flag")
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()
```



[154]:
```
inp1[inp1.pdays>0].response_flag.mean()
```

[154]: 0.2307785593014795

[155]:
```
res=pd.pivot_table(data=inp1, index="education", columns="poutcome",␣
    ↪values="response_flag")
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.2308)
plt.show()
```



education
primary
secondary
tertiary
unknown
0.089
0.14
0.61
0.072
0.11
0.14
0.64
0.084
0.17
0.22
0.66
0.11
0.16
0.2
0.68
0.1
failure
other
success
unknown
poutcome
0.6
0.5
0.4
0.3
0.2
0.1