JK LAKSHMIPAT UNIVERSITY

STATISTICS WITH PYTHON SPECIALIZATION
AS1402

Project Report

Submitted by:
Mudit Lodha (2021BTechCSE151)

Submitted to:
Dr. Richa Sharma

November 25, 2022

NAAC 'A' Grade Accredited

Abstract

In the modern age, financial services have become a large part of our daily lives. This sector employs a massive section of the workforce. The Indian FinTech industry's market size is estimated to be $50 Bn., and is projected to grow to $150 Bn. by 2025 [1]. This industry is heavily dependent on data analytics and statistics. Hence there can be many tools made to this end, which would assist the industry as a whole or a particular company to achieve their goals in better serving the consumers.

Included in this report is an analysis of a marketing campaign for a Portuguese banking institution [2]. It describes a list of customers with various data and whether or not the campaign was successful. The various variables are analysed to see which ones impact the success of the campaign. Correlation plots are made to see which variables correlate with each other and hypothesis testing is done on the data as well.

Contents

List of Figures

1 Introduction

1.1 About Inferential Statistics

Statistics can be broadly split into two categories, descriptive statistics and inferential statistics. Descriptive statistics helps summarize the characteristics of a dataset, whereas inferential statistics can help to come to conclusions and make predictions based on the data. There are 2 main uses for inferential statistics:

- Making estimates about populations.
- Testing hypotheses to draw conclusions about populations.

There are many tools within infertential statistics. The ones used in this report are:

- Correlation (1.2)
- Regression (1.3)
- Hypothesis Testing (1.4)

Each will be further described.

1.2 Correlation

A correlation is a statistical measure of the relationship between two variables. The measure is best used in variables that demonstrate a linear relationship between each other[3]. The correlation coefficient is a value that indicates the strength of the relationship between variables. The coefficient can take any values from -1 to 1. The interpretations of the values are:

- -1: Perfect negative correlation. The variables tend to move in opposite directions (i.e., when one variable increases, the other variable decreases).
- 0: No correlation. The variables do not have a relationship with each other.
- 1: Perfect positive correlation. The variables tend to move in the same direction (i.e., when one variable increases, the other variable also increases).

To calculate correlation, we use the following formula:

$$r_{xy} = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (x_i - \overline{x})^2 \sum (y_i - \overline{y})^2}}$$

Where:

- r_{xy} – the correlation coefficient of the linear relationship between the variables x and y
- x_i – the values of the x-variable in a sample
- $\overline{x}$ – the mean of the values of the x-variable
- y_i – the values of the y-variable in a sample
- $\overline{y}$ – the mean of the values of the y-variable

1.3 Regression

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables)[4].

In statistics, the logistic model (or logit model) is a statistical model that models the probability of an event taking place by having the log-odds for the event be a linear combination of one or more independent variables.

The logistic function is a sigmoid function, which takes any real input t, and outputs a value between zero and one. This value is taken to be the probability of the outcome. Its equation can be represented by:

$$p(x) = \frac{1}{1 + e^{-t}}$$

Where t can be a single explanatory variable, or a linear combination of multiple explanatory variables. Using the output of this function we can set a threshold where we say that the output was true. For instance, we may say that values above 0.3 are true and below and including 0.3 are false.

1.4 Hypothesis Testing

Hypothesis testing is an act in statistics whereby an analyst tests an assumption regarding a population parameter. The methodology employed by the analyst depends on the nature of the data used and the reason for the analysis[5].

Hypothesis testing is used to assess the plausibility of a hypothesis by using sample data. Such data may come from a larger population, or from a data-generating process. All analysts use a random population sample to test two different hypotheses: the null hypothesis and the alternative hypothesis.

The null hypothesis is usually a hypothesis of equality between population parameters; e.g., a null hypothesis may state that the population mean return is equal to zero. The alternative hypothesis is effectively the opposite of a null hypothesis (e.g., the population mean return is not equal to zero). Thus, they are mutually exclusive, and only one can be true. However, one of the two hypotheses will always be true.

2 Data Collection

2.1 Descriptive analysis

The data used for this analysis comes from a Portuguese bank [2], between 2008 and 2013. This was collected as an effort to study consumer behaviour with regards to a recent campaign that was ran by the bank. The following variables were recorded for the dataset:

- age (numeric)
- job : type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
- marital : marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)

- education (categorical: "unknown", "secondary", "primary", "tertiary")
- default: has credit in default? (binary: "yes", "no")
- balance: average yearly balance, in euros (numeric)
- housing: has housing loan? (binary: "yes", "no")
- loan: has personal loan? (binary: "yes", "no")

The following variables were related with the last contact of the current campaign:

- contact: contact communication type (categorical: "unknown", "telephone", "cellular")
- day: last contact day of the month (numeric)
- month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
- duration: last contact duration, in seconds (numeric)

Other attributes:

- campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
- previous: number of contacts performed before this campaign and for this client (numeric)
- poutcome: outcome of the previous marketing campaign (categorical: "unknown", "other", "failure","success")

Output variable (desired target):

- y - has the client subscribed a term deposit? (binary: "yes","no")

There are no null or missing values in the data. Hence there is no need to deal with that, which is a common part in most statistical analysis. However some data points will need further adjustment during analysis, which will be covered in the following section.

3 Methodology

For this project, the data was first analyzed to see roughly how many of each variable type there were. The results of the initial analysis is described in the following figure:

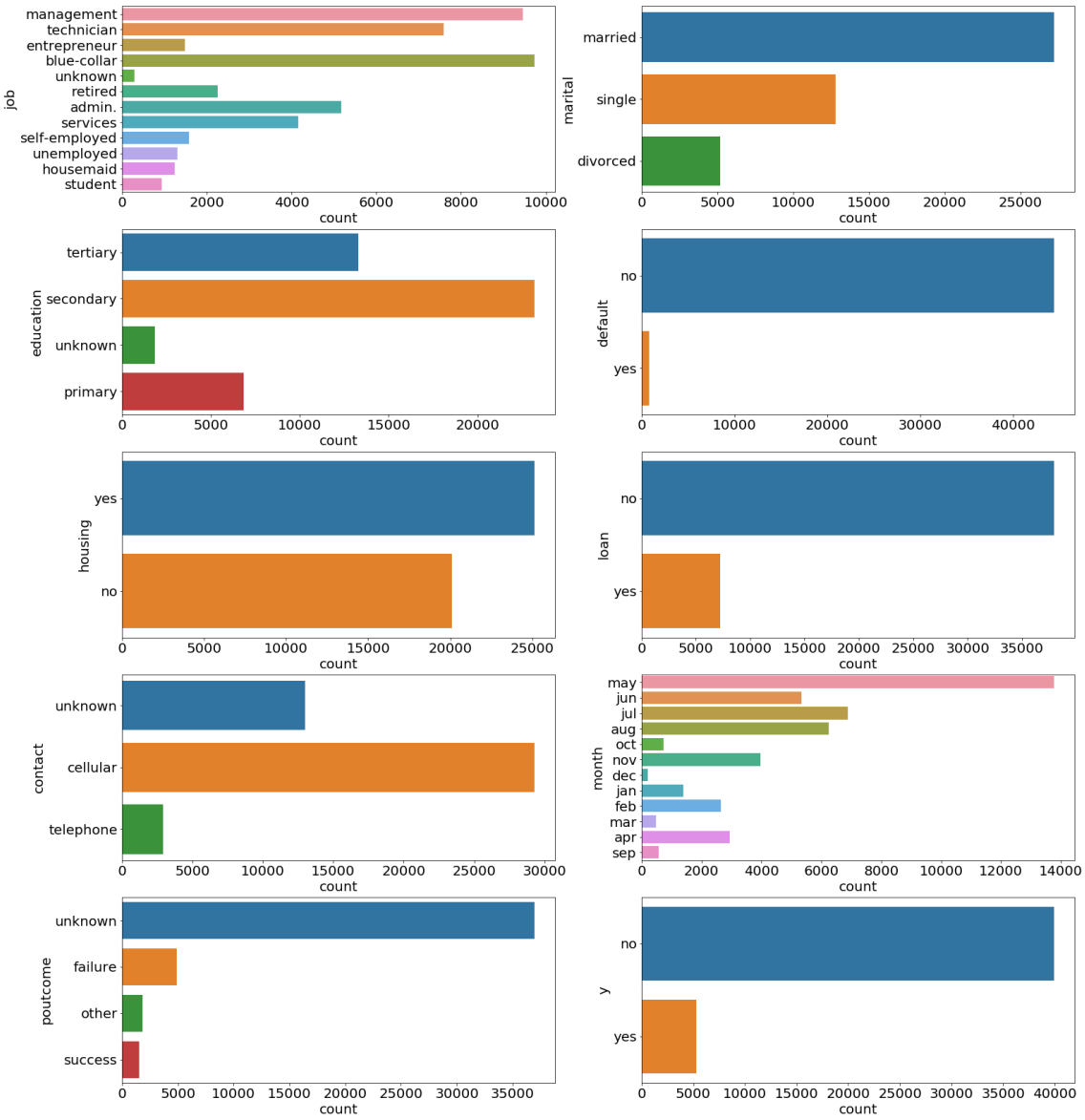


Figure 1: Categorical data analysis

Then the numeric data was analyzed to give us the following information:

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

Figure 2: Numeric data analysis

Hence we can see that the mean age is 41.17, with the standard deviation of 10.57. The median is 39, which is less than the mean, indicating a slight left skew. The youngest person is 19 and the oldest is 87.

The mean balance is 1422.65, with the standard deviation of 3009.64. The median is 444, which is significantly less than the mean, indicating a heavy left skew.

The day represents the day of the month.

The mean duration for last contact is 263.96, with the standard deviation of 3.11. The median is 185, which is less than the mean, indicating a left skew. The shortest duration was 4 and the longest was 3025 seconds.

The mean number of contacts performed during this campaign for this client is 2.79, with the standard deviation of 259.86. The median is 2, which is less than the mean, indicating a left skew. The fewest was 1 and the most was 50.

The mean days since previous contact is 39.77, with the standard deviation of 100.12. The median is -1, however this is due to many values being -1, indicating that the customer was not contacted previously. Hence we need to remove the -1 value to get a more accurate description.

The mean number of contacts performed before this campaign for this client was 0.54, with a standard deviation of 1.69. The median is 0. This data also needs to be further analysed after removing the 0 value, since most of the customers were not contacted previously.

After further filtering the data, to remove the less useful values as described above, we get the following information:

```
              pdays
count    8257.000000
mean      224.577692
std       115.344035
min         1.000000
25%       133.000000
50%       194.000000
75%       327.000000
max       871.000000
Name: pdays, dtype: float64
```

```
             previous
count    8257.000000
mean        3.177546
std         4.560820
min         1.000000
25%         1.000000
50%         2.000000
75%         4.000000
max       275.000000
Name: previous, dtype: float64
```

Figure 3: Further analysis on days since last contact from a previous campaign and contacts performed before this campaign

Then the proportions of successful and unsuccessful campaigns was analyzed for each variable. The results of this can be seen in the appendix.

3.1 Correlation

For performing correlation, python was used. The following were the results:

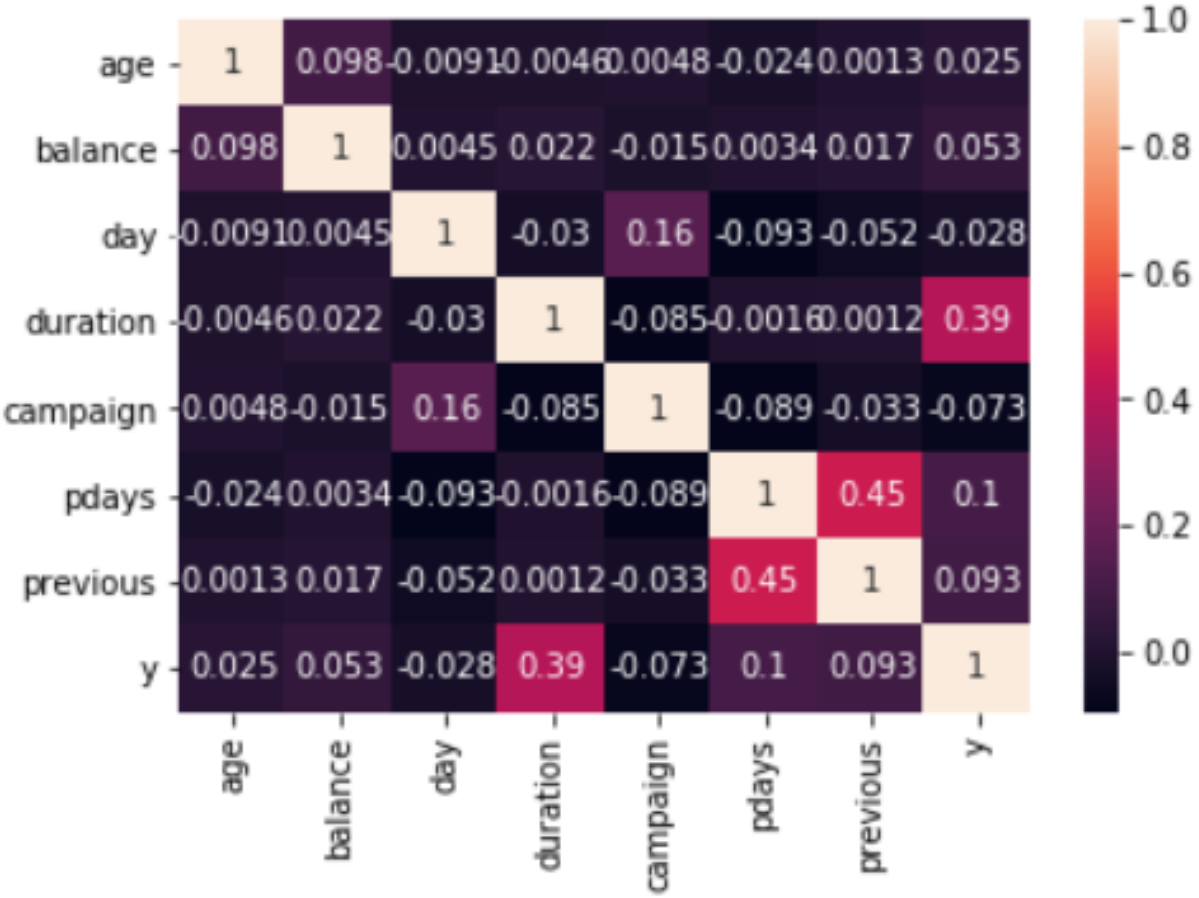


Figure 4: Correlation coefficients between variables

These results will be discussed in the discussion section.

3.2 Logistic Regression

First the data was split into training and test data with and 80/20 split. Then a model was trained on the variables listed belows:

- age
- duration (duration of last contact)
- campaign (number of contacts performed in current campaign)
- previous (number of contacts performed before current campaign)

Following were the results of the model when it was trained:

```
Optimization terminated successfully.
         Current function value: 0.352867
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                36169
Model:                          Logit   Df Residuals:                    36165
Method:                           MLE   Df Model:                            3
Date:                Fri, 25 Nov 2022   Pseudo R-squ.:                 0.02140
Time:                        11:22:23   Log-Likelihood:                -12763.
converged:                       True   LL-Null:                       -13042.
Covariance Type:            nonrobust   LLR p-value:                1.206e-120
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.0667      0.068    -30.516      0.000      -2.199      -1.934
age            0.0065      0.002      4.338      0.000       0.004       0.010
campaign      -0.1275      0.009    -13.688      0.000      -0.146      -0.109
previous       0.1113      0.007     16.783      0.000       0.098       0.124
==============================================================================
```

Figure 5: Logistic regression model training results

Then the model was used on the test data. The threshold was varied and the following table shows the results:

Threshold	Accuracy	Precision	Recall
0.01	13.24%	11.93%	100%
0.05	33.26%	14.82%	98.49%
0.10	75.26%	29.26%	77.89%
0.20	87.10%	45.49%	48.92%
0.30	88.91%	54.46%	34.43%
0.40	89.22%	59.87%	25.12%
0.50	89.36%	66.13%	19.47%

Table 1: Results of testing the model with various thresholds.

However, in a future campaign we can easily influence the ‘duration’ parameter. Keeping this in mind a logistic regression analysis was also computed without this variable.

```
Optimization terminated successfully.
         Current function value: 0.352867
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                36169
Model:                          Logit   Df Residuals:                    36165
Method:                           MLE   Df Model:                            3
Date:                Fri, 25 Nov 2022   Pseudo R-squ.:                 0.02140
Time:                        11:22:23   Log-Likelihood:                -12763.
converged:                       True   LL-Null:                       -13042.
Covariance Type:            nonrobust   LLR p-value:                1.206e-120
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.0667      0.068    -30.516      0.000      -2.199      -1.934
age            0.0065      0.002      4.338      0.000       0.004       0.010
campaign      -0.1275      0.009    -13.688      0.000      -0.146      -0.109
previous       0.1113      0.007     16.783      0.000       0.098       0.124
==============================================================================
```

Figure 6: Logistic regression model without 'duration' training results

Threshold	Accuracy	Precision	Recall
0.01	12.31%	11.82%	100%
0.05	15.24%	12.07%	98.78%
0.10	32.25%	13.08%	84.38%
0.15	84.93%	29.40%	20.13%
0.20	87.43%	29.44%	4.99%

Table 2: Results of testing the model without 'duration' with various thresholds.

3.3 Hypothesis Testing

For this project, we ran the following hypothesis statement:

Students are more likely to result in successful campaigns as compared any other group.

Hence we can make 2 tests with $\alpha = 0.05$:

$$H_0 : \mu_{students} = \mu_{others}, H_a : \mu_{students} > \mu_{others}$$

Where μ represents the proportion of successful campaigns in the group. To conduct this test, first we find the proportion of students that were found to be successful. There were 269 students that were labelled successful, and a total of 938 students in the data set. This gives us a proportion of 28.68%.

The following is the table of all other job groups in the data set:

Job	Successful	Total	Successful(%)
management	1301	9458	13.76%
technician	840	7597	11.06%
entrepreneur	123	1487	8.27%
blue-collar	708	9732	7.27%
retired	516	2264	22.79%
admin.	631	5171	12.20%
services	369	4154	8.88%
self-employed	187	1579	11.84%
unemployed	202	1303	15.5%
housemaid	109	1240	8.79%
unknown	34	288	11.81%

Table 3: Proportion of success across jobs.

Clearly all of these values are below the 28.68% shown in students, however we shall use confidence intervals to be sure of this. Most of these values are already significantly less than the students, other than retired. Hence we shall only run our analysis on them.

The formula for confidence interval is as follows:

$$\overline{X} \pm Z * \frac{\sigma}{\sqrt{n}}$$

Where $\overline{X}$ is the mean proportion, Z is the Z-value, σ is the standard deviation and n is the number of observations.

Hence we can construct confidence intervals for both students and retired persons. Using python, these can be computed. The code is added in the appendix (7).

The results are shown below:

For students the confidence interval was (0.2833, 0.2903)

For retired the confidence interval was (0.2247, 0.2312)

We can see that there is no commonality between them, hence we reject the null hypothesis.

Therefore, we can say with that students are more likely to result in a successful campaign than retired folks.

4 Discussion

4.1 Proportion Analysis

From the proportions shown in the appendix, we can see that some variables clearly affect success while others do not. For instance what the job of the customer is, their education level, whether or not they defaulted, whether they took a housing loan etc, all contribute to whether they will be successfully marketed to.

However 2 of these variables can be influenced by the bank: duration of last contact and number of times contact was established in current campaign; and it is clear that increasing the duration increases the proportion of success. Hence the result of this analysis is that for the consumers whose information suggests they are more likely to succeed, the bank should contact them more frequently and for longer durations.

4.2 Correlation

This result shows that there is only significant correlation in 2 places. Between pdays and previous, and between y and duration. This matches our previous outcome, which suggested that increasing the duration increases the likelihood of success. However this graph is not a strong source of information since our target variable is binary. Correlation is a better tool when the data has more ordered unique values.

4.3 Regression

The results of regression were more promising. We can see in the tables that as we increase the threshold, the accuracy and precision increases, and the recall decreases. In order to understand what these values mean, these values are described here:

Accuracy is the sum total of correct predictions divided by the total predictions.

Precision is the proportion of positive predictions that were correct.

Recall is proportion of positive outcomes that were predicted correctly.

For instance, in regression with duration, at threshold=0.20, 87.10% of predictions were correct, 45.49% of 'success' predictions were actually successful, and of all the customers that were actually successful, 48.92% were correctly predicted.

In the context of a banking system, recall is more important that precision i.e. we want to get a large proportion of the actually successful people in our prediction, even if it means we predict too many and it results in a large number of predictions where they were actually not successful but we predicted so. This is because we can focus on these consumers to further increase their chances by calling them more frequently and for longer durations.

4.4 Hypothesis Testing

The null hypothesis was rejected, which means that students were found to be more likely to succeed than any other group, with $\alpha = 0.05$. This means that if the same population was sampled, and the proportions were calculated, the proportion of students that were successful would be larger than any other group at least 95% of the time. Hence the campaign should be targetted at students.

5 Conclusion

In this report we went over data from a Portuguese Bank, and used various inferential statistical techniques on it, namely correlation, regression and hypothesis testing. Python was used to conduct these calculations, and the code is attached in the appendix, along with various more results that were not discussed for the sake of brevity.

With regards to the bank, a logistic regression model was formed to help them identify the costumers that they need to focus on while the campaign is ongoing, so that they can call them more frequently and for longer durations to increase the number of successful outcomes. It was also shown that students are the most likely to be a successful campaign outcome with a confidence level of 95%.

6 References

[1] Anon., 2022. Financial Services Sector in India | Fintech Industry in India. [online] Available at: <https://www.investindia.gov.in/sector/bfsi-fintech-financial-services> [Accessed 25 November 2022].

[2] Anon., 2022. Bank Marketing Data Set - UCI Machine Learning Repository. [online] Available at: <https://archive.ics.uci.edu/ml/datasets/bank+marketing> [Accessed 25 November 2022].

[3] Anon., 2022. Correlation - Overview, Formula, and Practical Example. [online] Available at: <https://corporatefinanceinstitute.com/resources/data-science/correlation/> [Accessed 25 November 2022].

[4] Anon., 2022. What is Regression? Definition, Calculation, and Example. [online] Available at: <https://www.investopedia.com/terms/r/regression.asp> [Accessed 25 November 2022].

[5] Anon., 2022. Hypothesis Testing Definition. [online] Available at: <https://www.investopedia.com/terms/h [Accessed 25 November 2022].

7 Appendix

In [327]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
pd.set_option('display.max_columns', 30) # set so can see all columns of th
e DataFrame
```

In [328]:
```
# Import the data
df = pd.read_csv("bank/bank-full.csv", sep=";")
```

In [329]:
```
df.head()
```

Out[329]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mont
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	ma
1	44	technician	single	secondary	no	29	yes	no	unknown	5	ma
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	ma
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	ma
4	33	unknown	single	unknown	no	1	no	no	unknown	5	ma

In [330]:
```
n = df.shape[0]
n
```

Out[330]:
```
45211
```

Hence there are 45211 entries in this data.

In [331]:
```
df.isnull().sum()
```

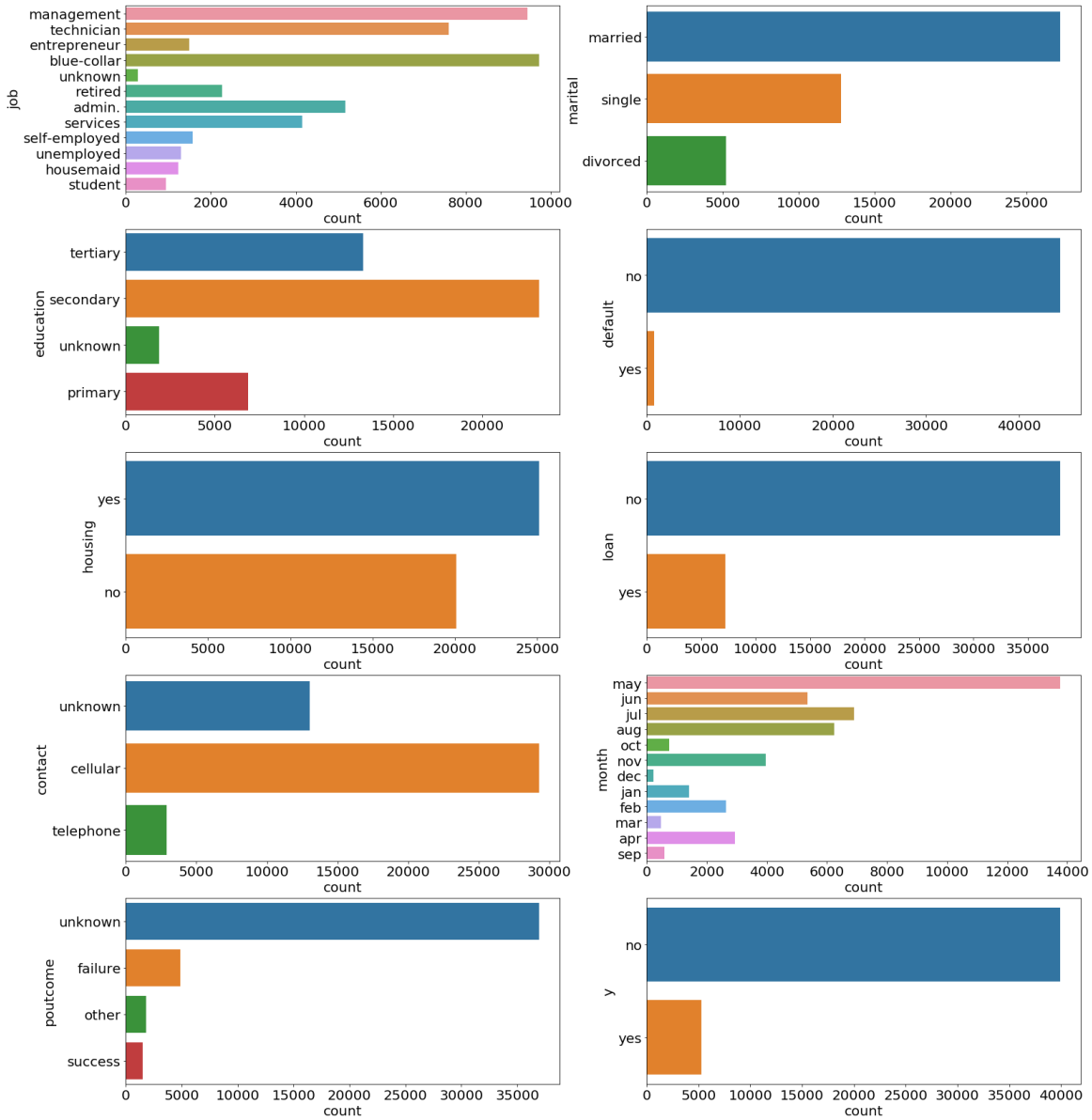
Out[331]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

There are no null vallues in the dataframe.

Non-numeric data

The entries contain the following unique vales in each column. The month value is also omitted since we already know what that will yeild.

In [332]:
```
plt.figure(figsize=(25,35))

categorical_features = [col for col in df.columns if pd.api.types.is_string
_dtype(df[col])]


for index, col in enumerate(categorical_features):
    plt.subplot(6, 2, index+1)
    ax = sns.countplot(y=col, data=df)
    ax.set_xlabel("count", fontsize=20)
    ax.set_ylabel(col, fontsize=20)
    ax.tick_params(labelsize=20)

plt.savefig("Distributions.png")
```



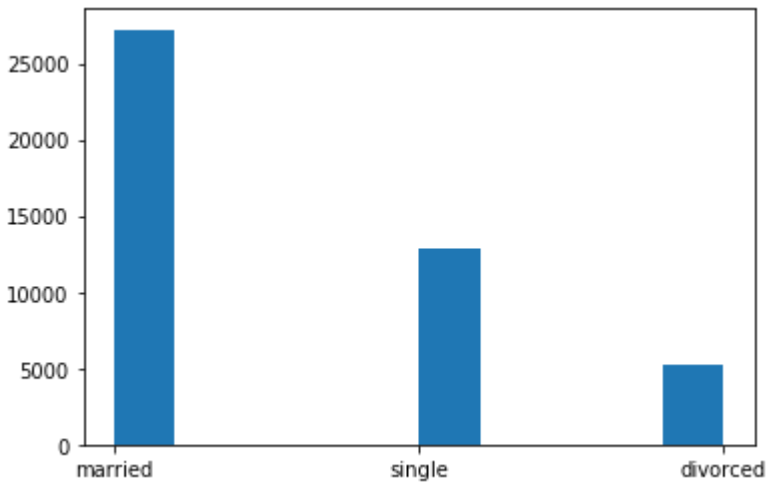
In [333]:
```
df.job.unique()
```

Out[333]:
```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

In [334]:
```
print(df.marital.unique())
plt.hist(df.marital)
```

```
['married' 'single' 'divorced']
```

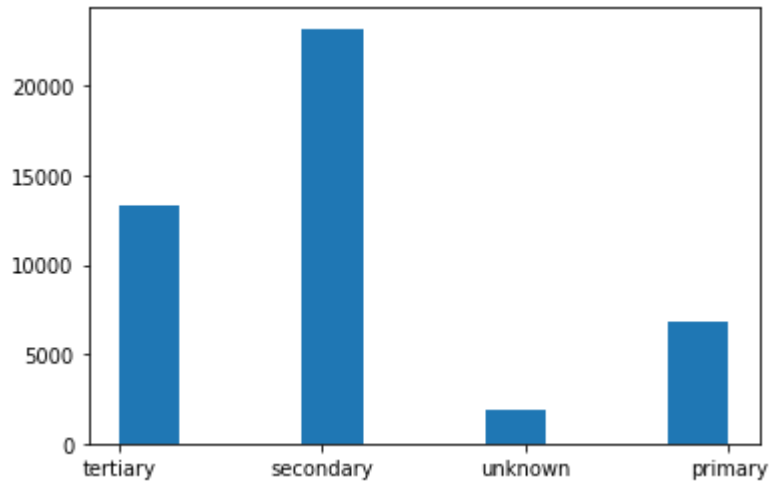
Out[334]:
```
(array([27214.,     0.,     0.,     0.,     0., 12790.,     0.,     0.,
            0.,  5207.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```



In [335]:
```
print(df.education.unique())
plt.hist(df.education)
```

```
['tertiary' 'secondary' 'unknown' 'primary']
```

Out[335]:
```
(array([13301.,     0.,     0., 23202.,     0.,     0.,  1857.,     0.,
            0.,  6851.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <a list of 10 Patch objects>)
```



In [336]:
```
print(df.default.unique())
plt.hist(df.default)
```

```
['no' 'yes']
```

Out[336]:
```
(array([44396.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,   815.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

In [337]:
```
print(df.housing.unique())
plt.hist(df.housing)
```

```
['yes' 'no']
```

Out[337]:
```
(array([25130.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0., 20081.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

In [338]:
```
print(df.loan.unique())
plt.hist(df.loan)
```

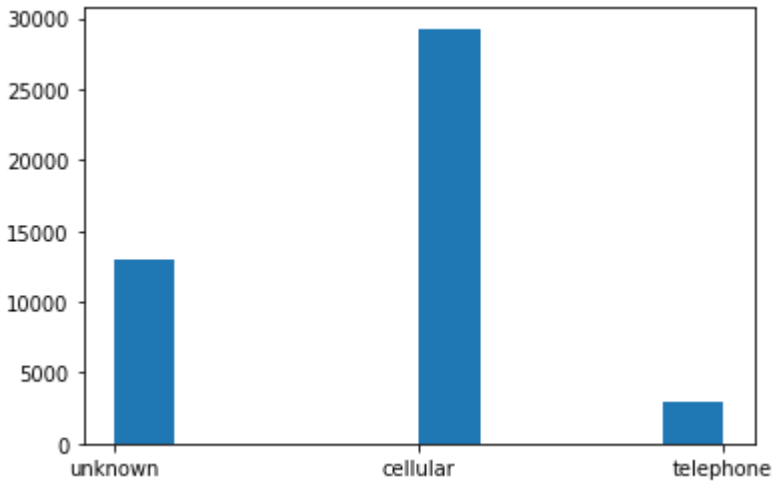
```
['no' 'yes']
```

Out[338]:
```
(array([37967.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,  7244.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

In [339]:
```
print(df.contact.unique())
plt.hist(df.contact)
```

```
['unknown' 'cellular' 'telephone']
```

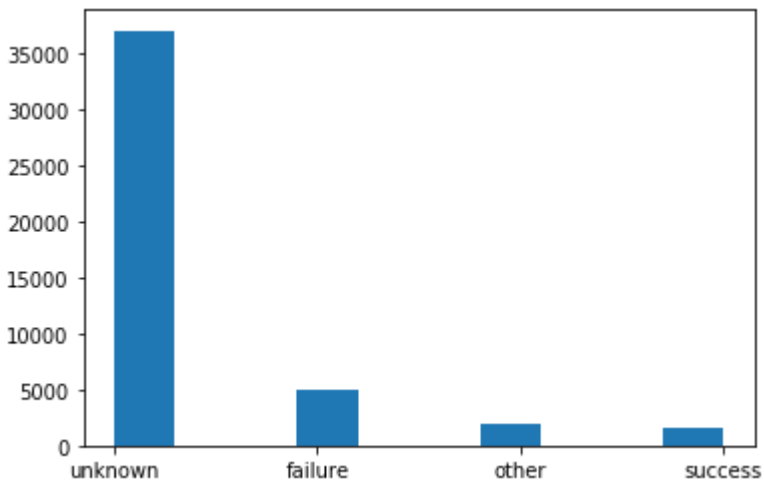
Out[339]:
```
(array([13020.,     0.,     0.,     0.,     0., 29285.,     0.,     0.,
            0.,  2906.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```



In [340]:
```
print(df.poutcome.unique())
plt.hist(df.poutcome)
```

```
['unknown' 'failure' 'other' 'success']
```

Out[340]:
```
(array([36959.,     0.,     0.,  4901.,     0.,     0.,  1840.,     0.,
            0.,  1511.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <a list of 10 Patch objects>)
```



In [341]:
```
print(df.y.unique())
plt.hist(df.y)
```

```
['no' 'yes']
```

Out[341]:
```
(array([39922.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,  5289.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```

In [342]:
```
print(str(len(df[df['y']=='yes'])) + " successful campaigns")
print(str(np.round(100* len( df[df['y']=='yes'])/df.shape[0], 2)) + "% of c
ampaigns were successful")
(100*len(df[df['y']=='no'])/df.shape[0])
```

```
5289 successful campaigns
11.7% of campaigns were successful
```

Out[342]:
```
88.30151954170445
```

Numeric data

Now lets generate some summaries for the numerical data.

In [343]:
```
df[["age", "balance", "day", "duration", "campaign", "pdays", "previous"]].
describe()
```

Out[343]:

	age	balance	day	duration	campaign	pdays
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000

Hence we can see that the mean age is 41.17, with the standard deviation of 10.57. The median is 39, which is less than the mean, indicating a slight left skew. The youngest person is 19 and the oldest is 87.

The mean balance is 1422.65, with the standard deviation of 3009.64. The median is 444, which is significantly less than the mean, indicating a heavy left skew.

The day represents the day of the month.

The mean duration for last contact is 263.96, with the standard deviation of 3.11. The median is 185, which is less than the mean, indicating a left skew. The shortest duration was 4 and the longest was 3025 seconds.

The mean number of contacts performed during this campaign for this client is 2.79, with the standard deviation of 259.86. The median is 2, which is less than the mean, indicating a left skew. The fewest was 1 and the most was 50.

The mean days since previous contact is 39.77, with the standard deviation of 100.12. The median is -1, however this is due to many values being -1, indicating that the customer was not contacted previously. Hence we need to remove the -1 value to get a more accurate description.

The mean number of contacts performed before this campaign for this client was 0.54, with a standard deviation of 1.69. The median is 0. This data also needs to be further analysed after removing the 0 value, since most of the customers were not contacted previously.

Days since previous contact (pdays)

In [344]:
```
pdays = df.pdays
pdays = pdays[pdays != -1]
print("       pdays  ")
print(pdays.describe())
```

```
              pdays
count    8257.000000
mean      224.577692
std       115.344035
min         1.000000
25%       133.000000
50%       194.000000
75%       327.000000
max       871.000000
Name: pdays, dtype: float64
```

Hence we can see after removing the -1 values, that for those customers that were contacted before this campaign the mean for days since previous contact is 224.87. The standard deviation is 117.20. The median is 189, indicating a left skew for the data. The least days was 1 and the most was 871.

Number of contacts performed before this campaign (previous)

In [345]:
```
previous = df.previous
previous = previous[previous != 0]
print("       previous")
print(previous.describe())
```

```
              previous
count    8257.000000
mean        3.177546
std         4.560820
min         1.000000
25%         1.000000
50%         2.000000
75%         4.000000
max       275.000000
Name: previous, dtype: float64
```

Hence we can see after removing the 0 values, that for those customers that were contacted before this campaign the mean for number of contacts before the current campaign is 3.01. The standard deviation is 2.91. The median is 2, indicating a left skew for the data. The least was 1 and the most was 25.

Successful vs unsuccessful campaigns

In [346]:
```
successful = df.drop(df[df.y == 'no'].index)
unsuccessful = df.drop(df[df.y == 'yes'].index)
```

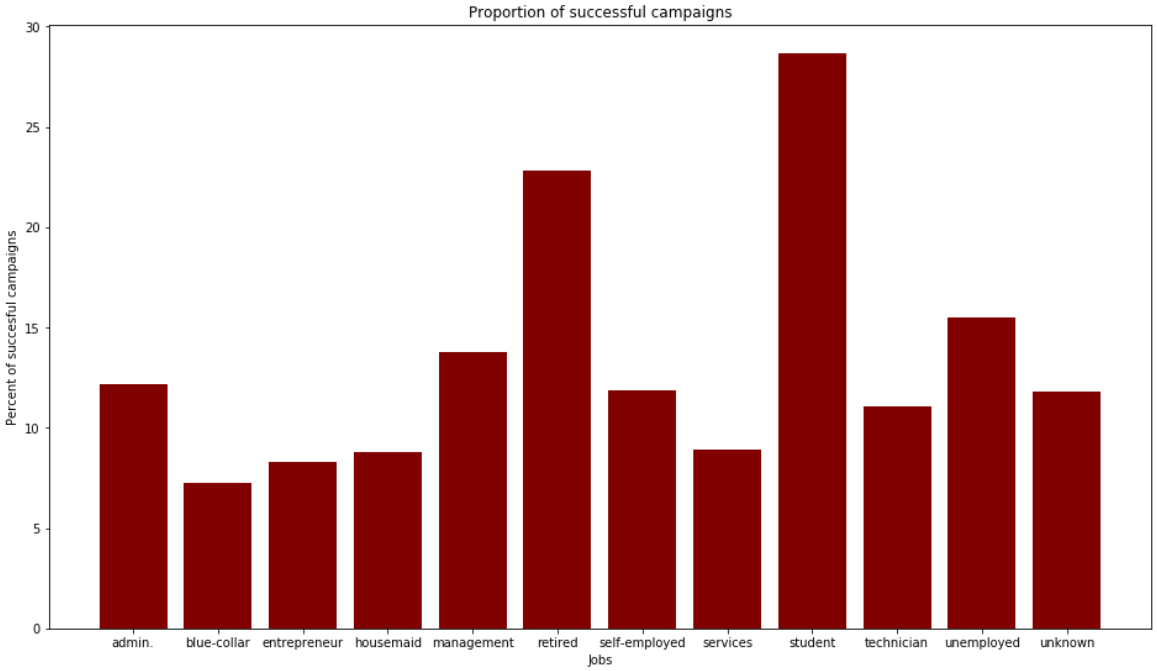
In [347]:
```
print("Successful campaigns = " + str(np.round(len(successful)/len(df)*100,
2)) + "%")
print("Unsuccessful campaigns = " + str(np.round(len(unsuccessful)/len(df)*
100, 2))+ "%")
```

```
Successful campaigns = 11.7%
Unsuccessful campaigns = 88.3%
```

Non-numeric data

Job

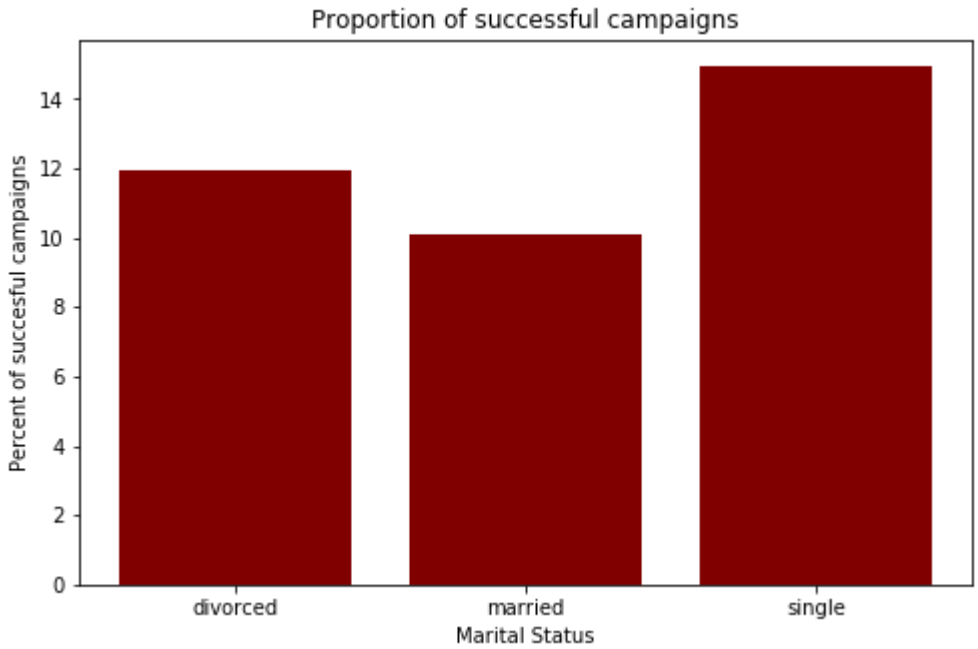
In [348]:
```
successful_job = successful.job
unsuccessful_job = unsuccessful.job
s_unique, s_counts = np.unique(successful_job, return_counts=True)
u_unique, u_counts = np.unique(unsuccessful_job, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))



fig = plt.figure(figsize = (16, 9))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Jobs")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("jobs_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Marital Status

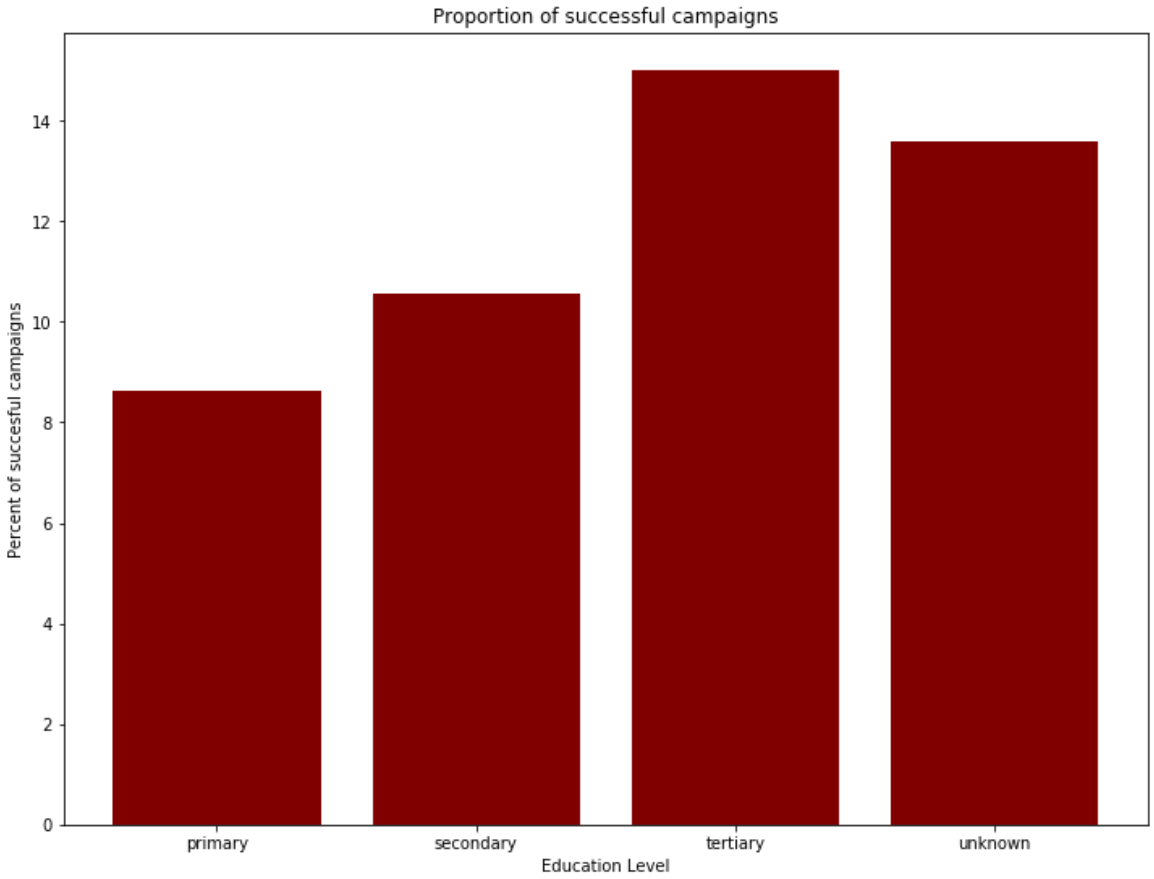
In [349]:
```
s_married = successful.marital
u_married = unsuccessful.marital
s_unique, s_counts = np.unique(s_married, return_counts=True)
u_unique, u_counts = np.unique(u_married, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8, 5))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Marital Status")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("marital_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Education

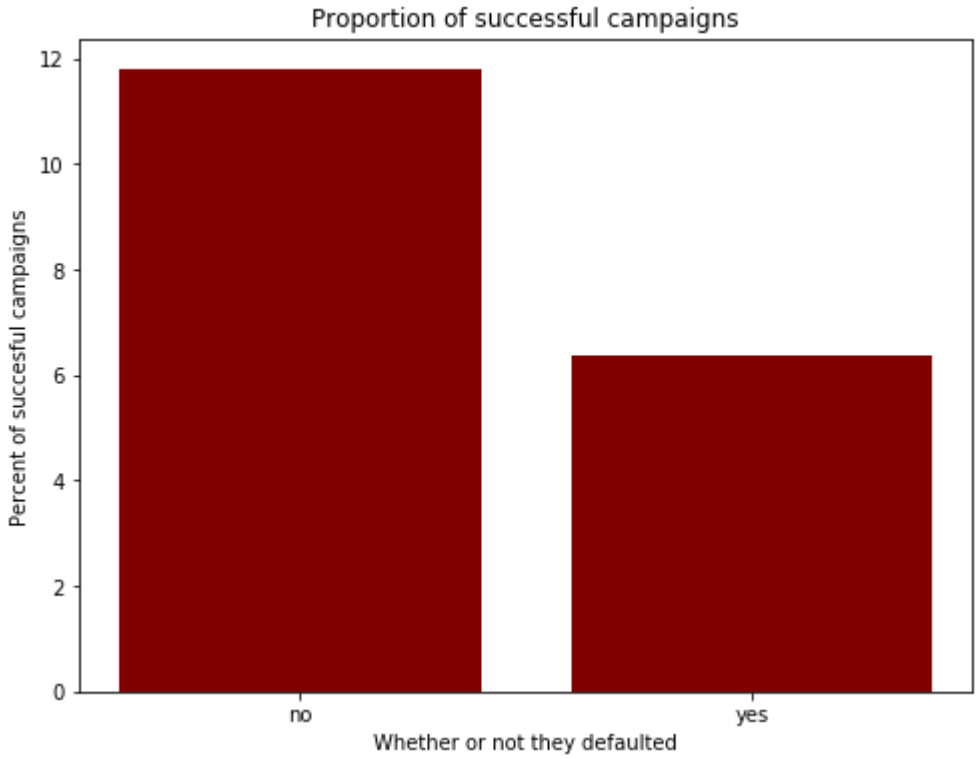
In [350]:
```
s_edu = successful.education
u_edu = unsuccessful.education
s_unique, s_counts = np.unique(s_edu, return_counts=True)
u_unique, u_counts = np.unique(u_edu, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (12, 9))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Education Level")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("edu_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

default

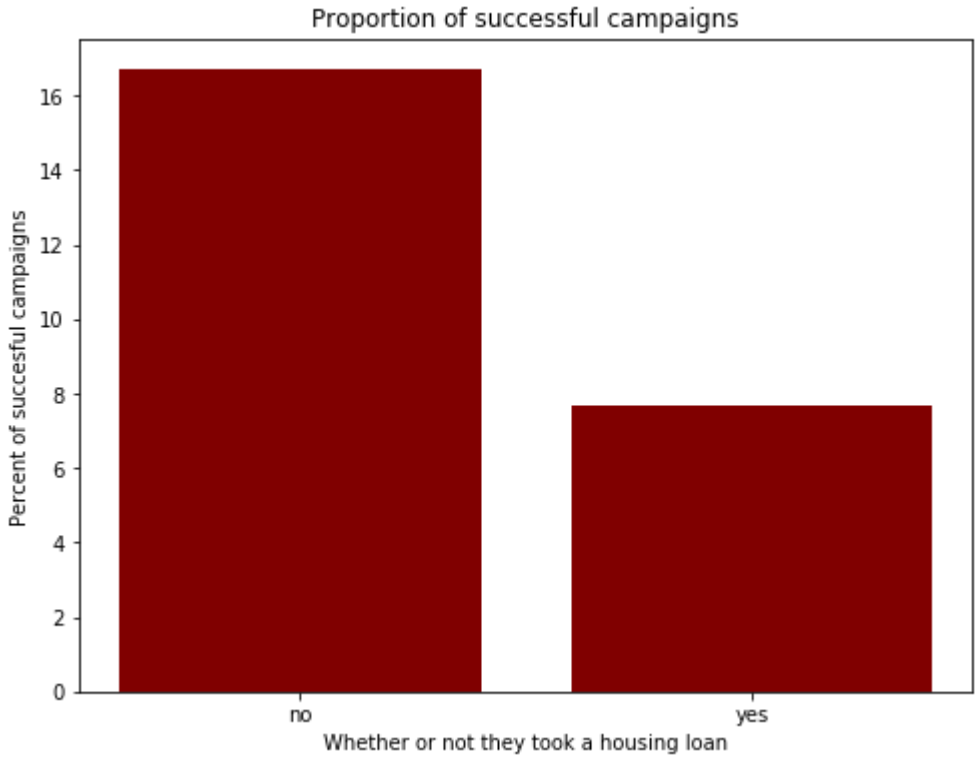
In [351]:
```
s_default = successful.default
u_default = unsuccessful.default
s_unique, s_counts = np.unique(s_default, return_counts=True)
u_unique, u_counts = np.unique(u_default, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8, 6))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Whether or not they defaulted")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("default_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Housing

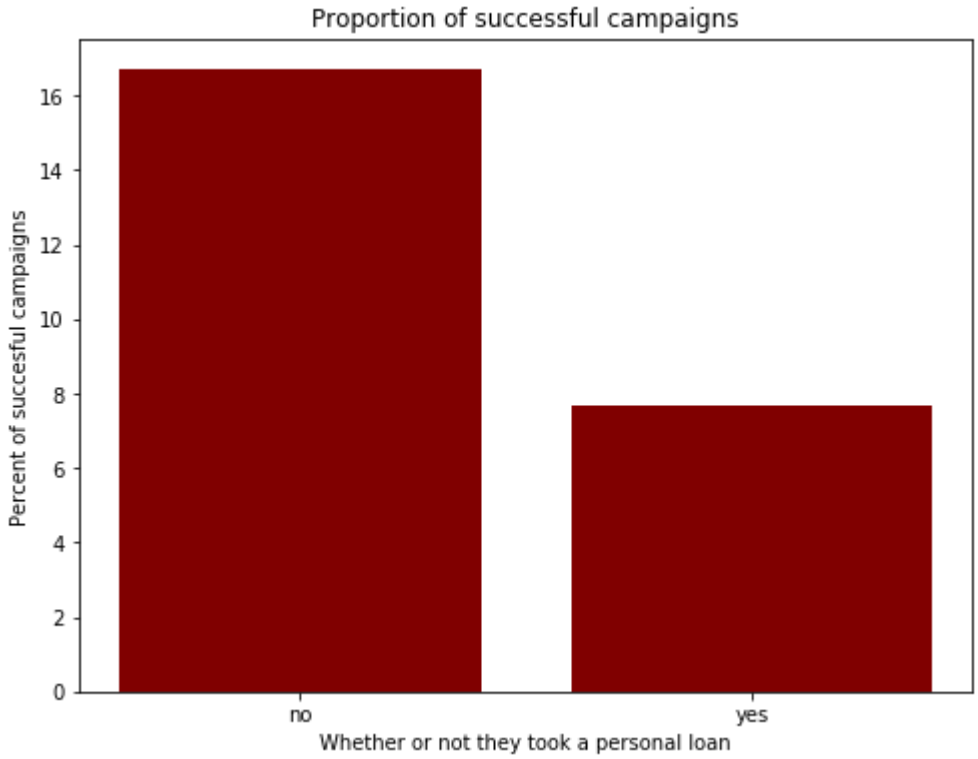
In [352]:
```
s_housing = successful.housing
u_housing = unsuccessful.housing
s_unique, s_counts = np.unique(s_housing, return_counts=True)
u_unique, u_counts = np.unique(u_housing, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Whether or not they took a housing loan")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("housing_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Loan

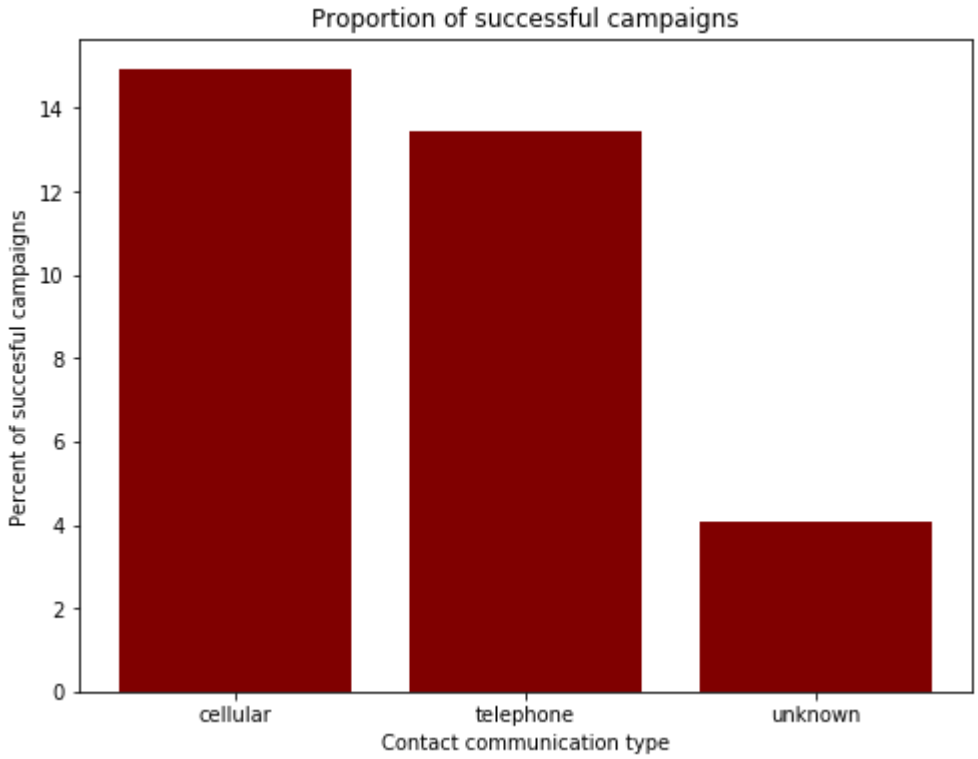
In [353]:
```
s_loan = successful.loan
u_loan = unsuccessful.loan
s_unique, s_counts = np.unique(s_housing, return_counts=True)
u_unique, u_counts = np.unique(u_housing, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Whether or not they took a personal loan")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("loan_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Contact

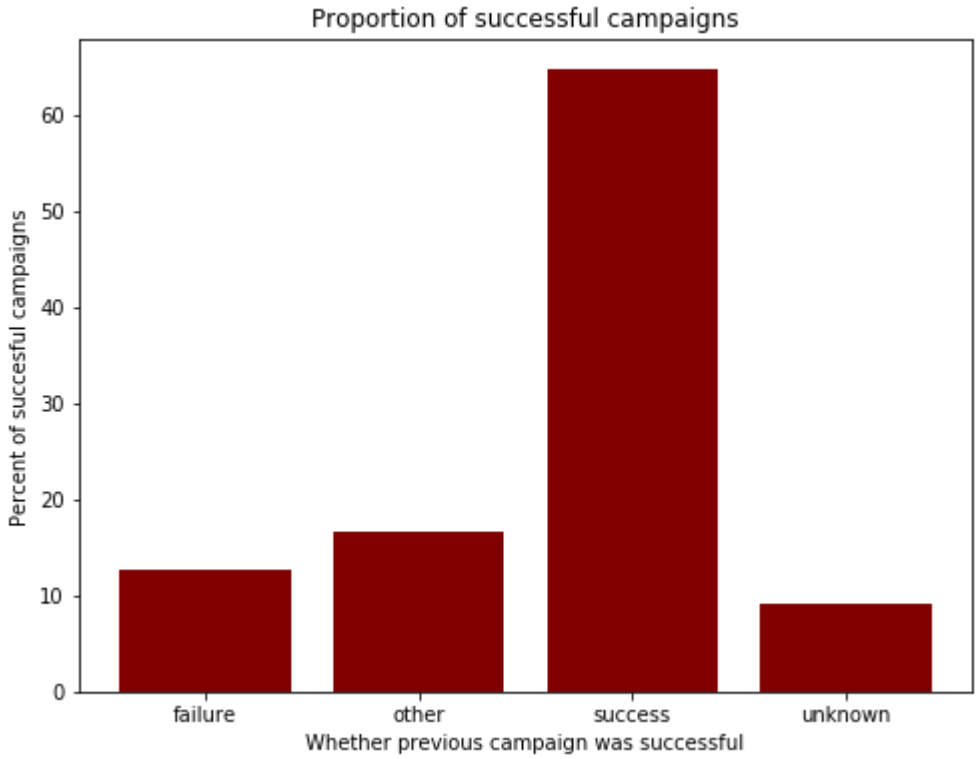
In [354]:
```
s_contact = successful.contact
u_contact = unsuccessful.contact
s_unique, s_counts = np.unique(s_contact, return_counts=True)
u_unique, u_counts = np.unique(u_contact, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Contact communication type")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("contact_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Previous campaign outcome (poutcome)

In [355]:
```
s_pout = successful.poutcome
u_pout = unsuccessful.poutcome
s_unique, s_counts = np.unique(s_pout, return_counts=True)
u_unique, u_counts = np.unique(u_pout, return_counts=True)
proportions = []

for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))


fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(s_unique, proportions, color ='maroon',
        width = 0.8)

plt.xlabel("Whether previous campaign was successful")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("previous_prop.png")
```



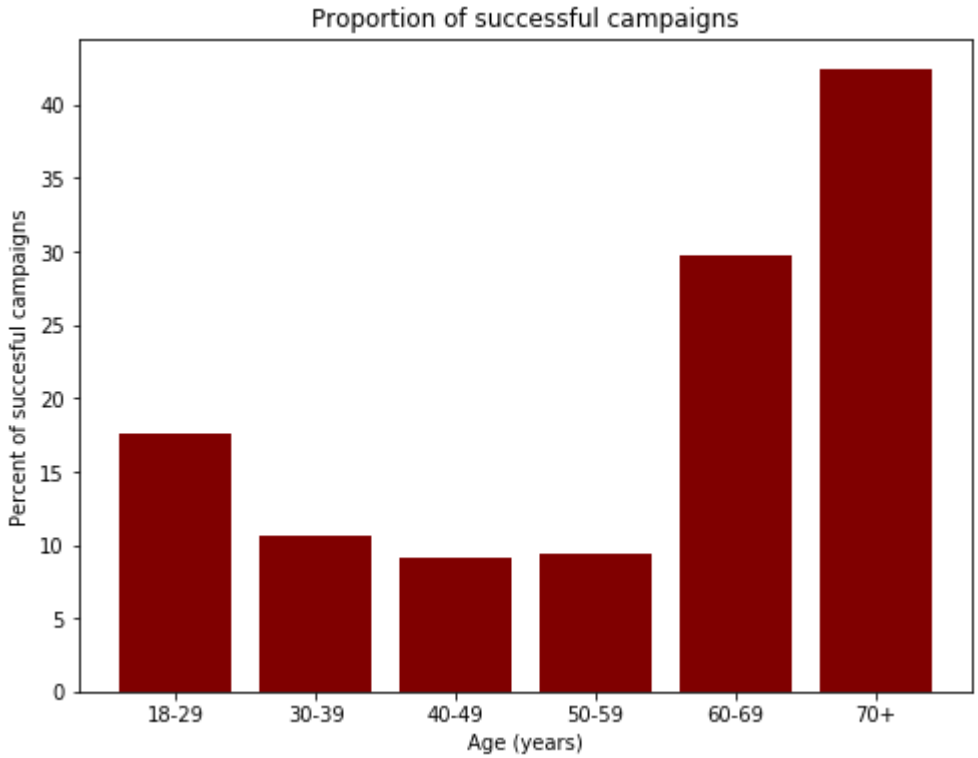

```
<Figure size 432x288 with 0 Axes>
```

Numeric Data

For this data we need to plot histograms with appropriate buckets

Age

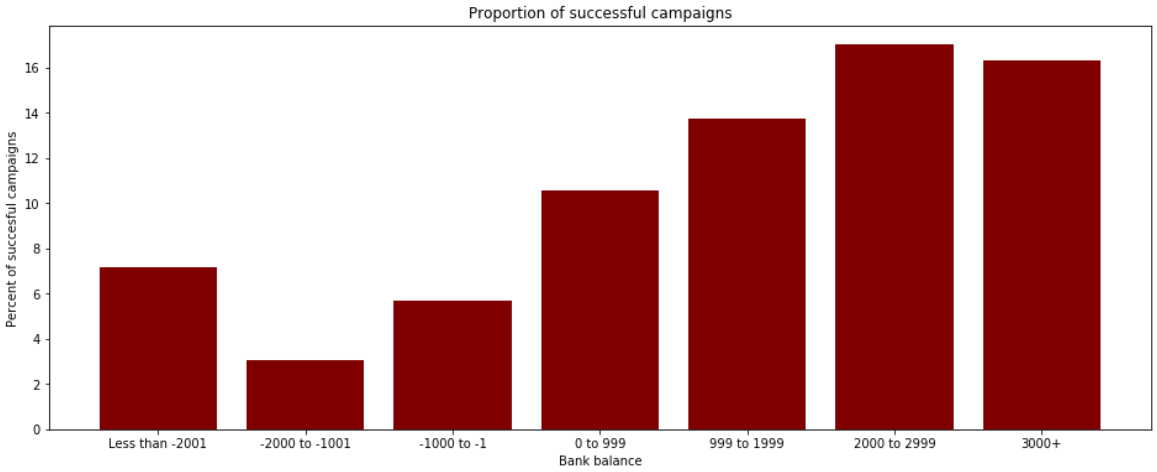
In [356]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0,0,0]
# proportions are 18-29, 30-39, 40-49, 50-59, 60-69, 70+
labels = ['18-29', '30-39', '40-49', '50-59', '60-69', '70+']
for i, row in df.iterrows():
    age = row.age
    success = 1 if row.y == 'yes' else 0
    if 18<=age<=29:
        counts[0][success] += 1
    if 30<=age<=39:
        counts[1][success] += 1
    if 40<=age<=49:
        counts[2][success] += 1
    if 50<=age<=59:
        counts[3][success] += 1
    if 60<=age<=69:
        counts[4][success] += 1
    if age>=70:
        counts[5][success] += 1


for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Age (years)")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("age_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

Balance

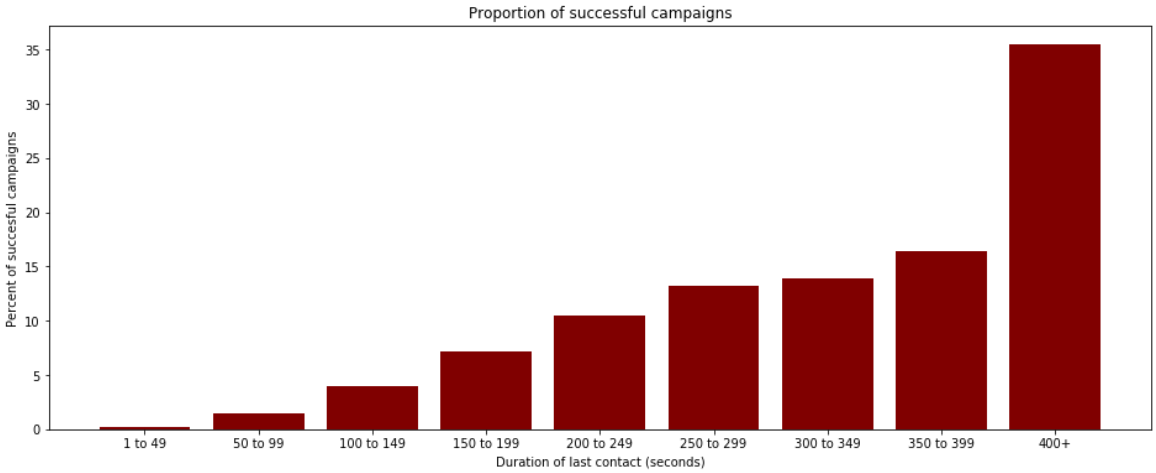
In [357]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0,0,0,0]
# proportions are
"""
-inf to -2000,
-2000 to -1001,
-1000 to -1,
0 to 999 ,
999 to 1999 ,
2000 to 2999,
3000+
"""
labels = ['Less than -2001',  '-2000 to -1001', '-1000 to -1', '0 to 999',
'999 to 1999', '2000 to 2999', '3000+']
for i, row in df.iterrows():
    balance = row.balance
    success = 1 if row.y == 'yes' else 0
    if -99999<=balance<=-2001:
        counts[0][success] += 1
    if -2000<=balance<=-1001:
        counts[1][success] += 1
    if -1000<=balance<=-1:
        counts[2][success] += 1
    if 0<=balance<=999:
        counts[3][success] += 1
    if 1000<=balance<=1999:
        counts[4][success] += 1
    if 2000<=balance<=2999:
        counts[5][success] += 1
    if balance>=3000:
        counts[6][success] += 1


for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (16,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Bank balance")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("balance_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

duration

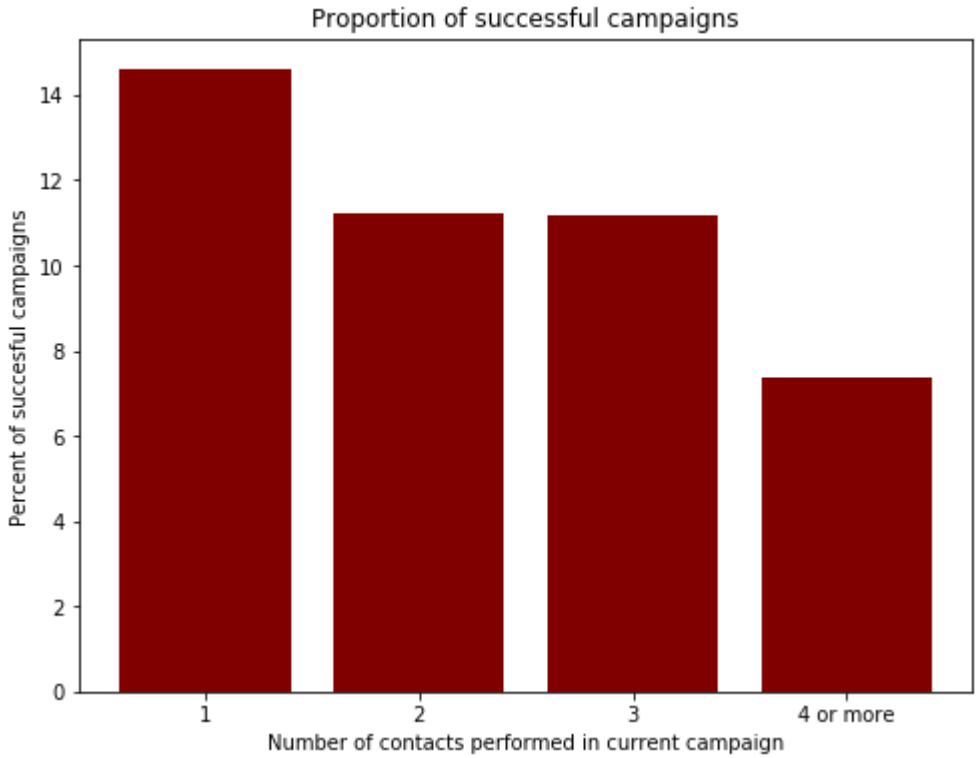
In [358]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0,0,0,0,0,0,0]
# proportions are
"""
1 to 49,
50 to 99,
100 to 149,
150 to 199 ,
200 to 249 ,
250 to 299,
300+
"""
labels = ['1 to 49', '50 to 99', '100 to 149', '150 to 199' ,'200 to 249'
,'250 to 299', '300 to 349', '350 to 399', '400+']
for i, row in df.iterrows():
    duration = row.duration
    success = 1 if row.y == 'yes' else 0
    if 1<=duration<=49:
        counts[0][success] += 1
    if 50<=duration<=99:
        counts[1][success] += 1
    if 100<=duration<=149:
        counts[2][success] += 1
    if 150<=duration<=199:
        counts[3][success] += 1
    if 200<=duration<=249:
        counts[4][success] += 1
    if 250<=duration<=299:
        counts[5][success] += 1
    if 300<=duration<=349:
        counts[6][success] += 1
    if 350<=duration<=399:
        counts[7][success] += 1
    if duration>=400:
        counts[8][success] += 1


for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (16,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Duration of last contact (seconds)")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("duration_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

campaign

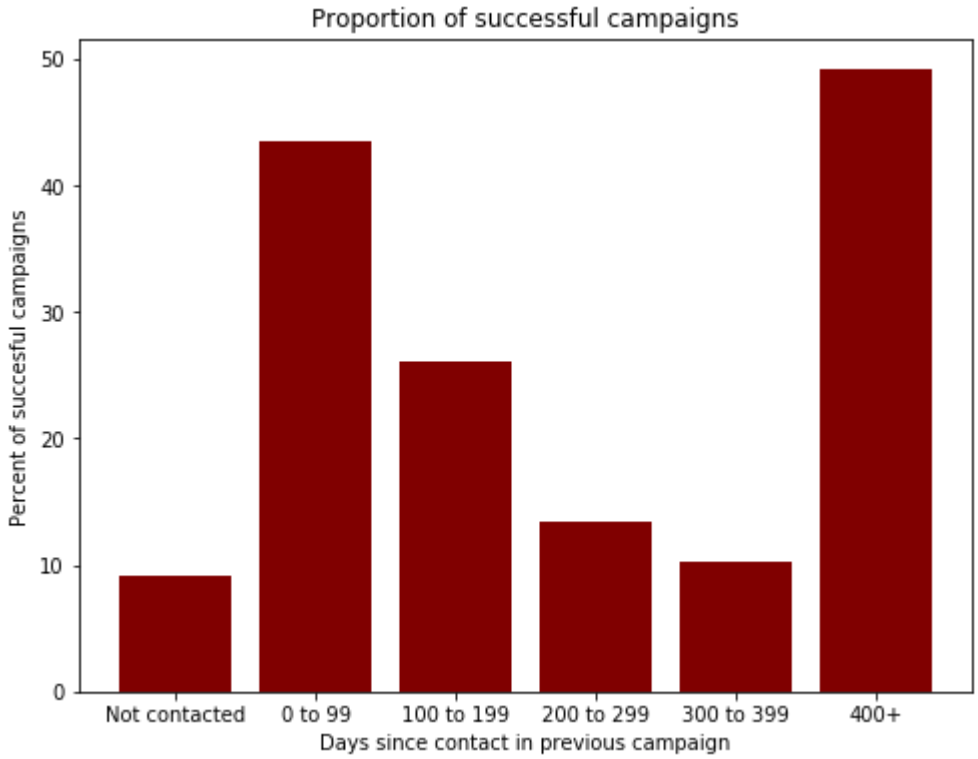
In [359]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0]
# proportions are 1, 2, 3, 3+
labels = ['1', '2', '3', '4 or more']
for i, row in df.iterrows():
    campaign = row.campaign
    success = 1 if row.y == 'yes' else 0
    if campaign==1:
        counts[0][success] += 1
    if campaign==2:
        counts[1][success] += 1
    if campaign==3:
        counts[2][success] += 1
    if campaign>3:
        counts[3][success] += 1

for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Number of contacts performed in current campaign")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("campaign_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

pdays

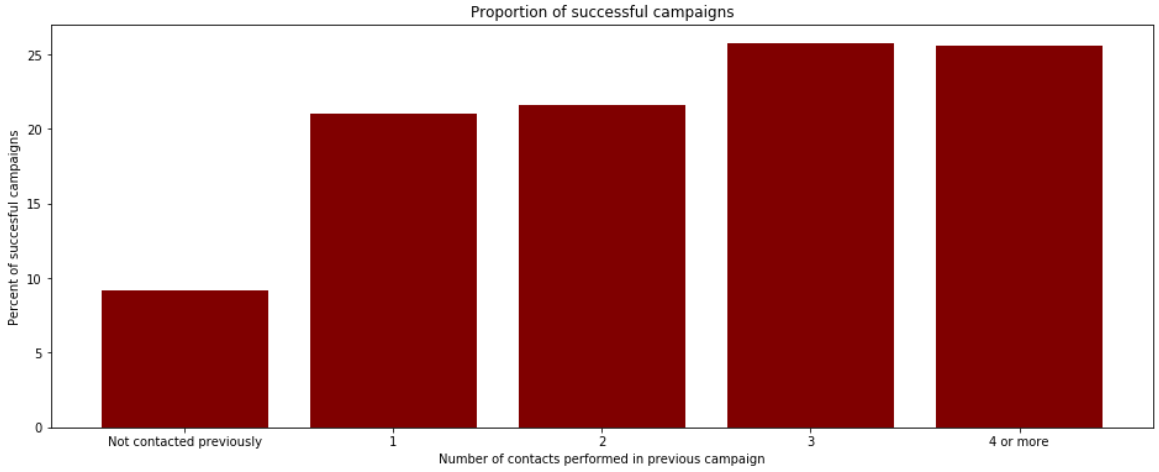
In [360]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0,0,0]
# proportions are not contacted(-1), 0 to 100, 100 to 200, 200 to 300, 300
to 400, 400+
labels = ['Not contacted', '0 to 99', '100 to 199', '200 to 299', '300 to 3
99', '400+']
for i, row in df.iterrows():
    pdays = row.pdays
    success = 1 if row.y == 'yes' else 0
    if pdays==-1:
        counts[0][success] += 1
    if 0<=pdays<=99:
        counts[1][success] += 1
    if 100<=pdays<=199:
        counts[2][success] += 1
    if 200<=pdays<=299:
        counts[3][success] += 1
    if 300<=pdays<=399:
        counts[4][success] += 1
    if pdays>=400:
        counts[5][success] += 1


for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (8,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Days since contact in previous campaign")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("pdays_prop.png")
```




```
<Figure size 432x288 with 0 Axes>
```

previous

In [361]:
```
counts = np.array([(0,0),(0,0),(0,0),(0,0),(0,0)])
proportions = [0,0,0,0,0]
# proportions are 0, 1, 2, 3, 4+
labels = ['Not contacted previously','1', '2', '3', '4 or more']
for i, row in df.iterrows():
    previous = row.previous
    success = 1 if row.y == 'yes' else 0
    if previous ==0:
        counts[0][success] += 1
    if previous==1:
        counts[1][success] += 1
    if previous==2:
        counts[2][success] += 1
    if previous==3:
        counts[3][success] += 1
    if previous>3:
        counts[4][success] += 1

for i in range(len(counts)):
    proportions[i] = 100 * counts[i][1]/(counts[i][0]+counts[i][1])

fig = plt.figure(figsize = (16,6))

# creating the bar plot
plt.bar(labels, proportions, color ='maroon', width = 0.8)

plt.xlabel("Number of contacts performed in previous campaign")
plt.ylabel("Percent of succesful campaigns")
plt.title("Proportion of successful campaigns")
plt.show()
plt.savefig("previous_prop.png")
```

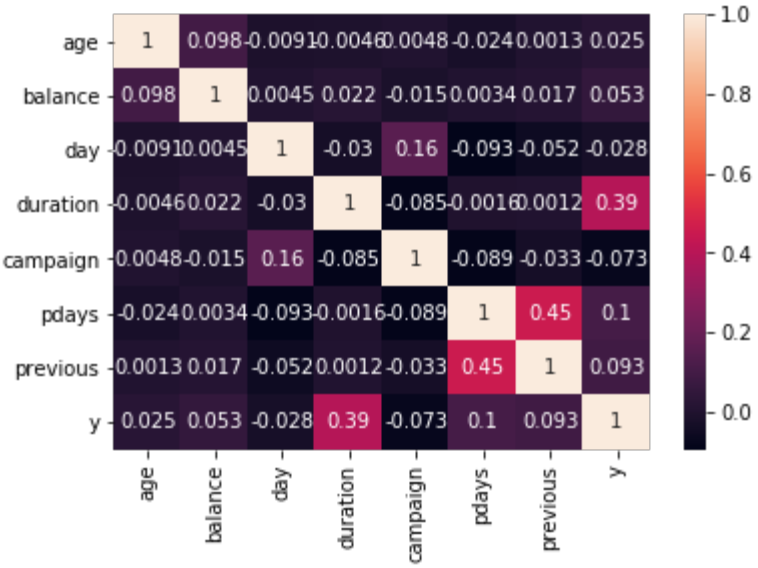



```
<Figure size 432x288 with 0 Axes>
```

Correlation

In [362]:
```
df["y"] = np.where(df["y"] == "yes", 1, 0)
f=df.corr()
f
sns.heatmap(f,annot=True,)
```

Out[362]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ce27f47808>
```



Logistic Regression model

In [363]:
```
# Split dataset into train and test (80/20)
train = df.sample(frac=0.8,random_state=42)
test  = df.drop(train.index)
```

In [364]:
```
# Convert target variable to numeric
y = np.where(train["y"] == 1, 1, 0)
```

In [365]:
```
import statsmodels.api as sm
X = train[["age", "duration", "campaign", "previous"]]
# add constant value for the intercept term
X = sm.add_constant(X)

# define and fit model
full_logistic_regression_model = sm.Logit(y, X)
result = full_logistic_regression_model.fit(maxiter=500)
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.298139
         Iterations 7
                           Logit Regression Results
==============================================================================
===
Dep. Variable:                      y   No. Observations:                36
169
Model:                          Logit   Df Residuals:                    36
164
Method:                           MLE   Df Model:
4
Date:                Fri, 25 Nov 2022   Pseudo R-squ.:                  0.1
732
Time:                        14:30:33   Log-Likelihood:                -107
83.
converged:                       True   LL-Null:                       -130
42.
Covariance Type:            nonrobust   LLR p-value:                    0.
000
==============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const         -3.2933      0.077    -42.771      0.000      -3.444      -3.
142
age            0.0076      0.002      4.677      0.000       0.004       0.
011
duration       0.0035   6.23e-05     56.761      0.000       0.003       0.
004
campaign      -0.1332      0.010    -12.686      0.000      -0.154      -0.
113
previous       0.1320      0.007     18.535      0.000       0.118       0.
146
==============================================================================
===
```

In [366]:
```
predict_params = test[["age", "duration", "campaign", "previous"]]
predict_params = sm.add_constant(predict_params)

predictions = result.predict(predict_params)
```

In [367]:
```
actual = np.where(test["y"] == 1, 1, 0)
threshold = 0.50
tp = 0
tn = 0
fp = 0
fn = 0

predictions = np.array(predictions)
for i in range(len(predictions)):
    if predictions[i]>=threshold and actual[i]==1:
        tp += 1
    if predictions[i]<threshold and actual[i]==0:
        tn += 1
    if predictions[i]>=threshold and actual[i]==0:
        fp += 1
    if predictions[i]<threshold and actual[i]==1:
        fn += 1


accuracy = (tp+tn) / (tp+tn+fp+fn)
precision = tp / (tp+fp)
recall = tp / (tp+fn)
print("Accuracy = " + str(np.round(accuracy*100, 2))+"%")
print("Precision = " + str(np.round(precision*100, 2))+"%")
print("Recall = " + str(np.round(recall*100, 2))+"%")
```

```
Accuracy = 89.36%
Precision = 66.13%
Recall = 19.47%
```

Removing duration

In [368]:
```
X = train[["age", "campaign", "previous"]]
# add constant value for the intercept term
X = sm.add_constant(X)

# define and fit model
full_logistic_regression_model = sm.Logit(y, X)
result = full_logistic_regression_model.fit(maxiter=500)
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.352867
         Iterations 7
                           Logit Regression Results
==============================================================================
===
Dep. Variable:                      y   No. Observations:                36
169
Model:                          Logit   Df Residuals:                    36
165
Method:                           MLE   Df Model:
3
Date:                Fri, 25 Nov 2022   Pseudo R-squ.:                 0.02
140
Time:                        14:30:34   Log-Likelihood:                -127
63.
converged:                       True   LL-Null:                       -130
42.
Covariance Type:            nonrobust   LLR p-value:                1.206e-
120
==============================================================================
===
                 coef    std err          z      P>|z|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const         -2.0667      0.068    -30.516      0.000      -2.199      -1.
934
age            0.0065      0.002      4.338      0.000       0.004       0.
010
campaign      -0.1275      0.009    -13.688      0.000      -0.146      -0.
109
previous       0.1113      0.007     16.783      0.000       0.098       0.
124
==============================================================================
===
```

In [369]:
```
predict_params = test[["age", "campaign", "previous"]]
predict_params = sm.add_constant(predict_params)

predictions = result.predict(predict_params)
```

In [370]:
```
actual = np.where(test["y"] == 1, 1, 0)
threshold = 0.16
tp = 0
tn = 0
fp = 0
fn = 0

predictions = np.array(predictions)
for i in range(len(predictions)):
    if predictions[i]>=threshold and actual[i]==1:
        tp += 1
    if predictions[i]<threshold and actual[i]==0:
        tn += 1
    if predictions[i]>=threshold and actual[i]==0:
        fp += 1
    if predictions[i]<threshold and actual[i]==1:
        fn += 1


accuracy = (tp+tn) / (tp+tn+fp+fn)
precision = tp / (tp+fp)
recall = tp / (tp+fn)
print("Accuracy = " + str(np.round(accuracy*100, 2))+"%")
print("Precision = " + str(np.round(precision*100, 2))+"%")
print("Recall = " + str(np.round(recall*100, 2))+"%")
```

```
Accuracy = 85.78%
Precision = 28.76%
Recall = 14.21%
```

The model has a precision score of 41.1%, which means that when it classifies a customer to have campaigned succesfully it is correct 41.1% of the time.

The recall is 31.25%, which means it correctly classifies only 31.25% of the successful campaigns.

We can adjust the threshold so that the recall is increased. This may decrease the precision, however we can see that in this use case recall is more important. With the current score of 25% precision and 77% recall, it would mean that 1 in every 4 customers that we think is successful would be successful, and 3 in 4 customers who were actually succesful would be classified correctly.

for all the customers we thought were succesful, only 15% actually were

for all the customers that were succesful, we got 60% of them

Hypothesis Testing

In [371]:
```
successful_job = successful.job
unsuccessful_job = unsuccessful.job
s_unique, s_counts = np.unique(successful_job, return_counts=True)
u_unique, u_counts = np.unique(unsuccessful_job, return_counts=True)
proportions = []
n = df.shape[0]
for i in range(len(s_counts)):
    if s_unique[i] != u_unique[i]:
        print("Different order.")
    proportions.append(100*s_counts[i]/(u_counts[i]+s_counts[i]))

students_proportion = s_counts[8]/(u_counts[8]+s_counts[8])
std_dev = np.sqrt(students_proportion*(1-students_proportion))
Z = stats.norm.ppf(1-0.05)

lb = students_proportion - (Z*std_dev/np.sqrt(n))
ub = students_proportion + (Z*std_dev/np.sqrt(n))
```

In [372]:
```
print("For students the confidence interval was - (" + str(np.round(lb,
4))+", "+str(np.round(ub,4))+")")
```

```
For students the confidence interval was - (0.2833, 0.2903)
```

In [373]:
```
retired_proportion = s_counts[5]/(u_counts[5]+s_counts[5])
std_dev = np.sqrt(retired_proportion*(1-retired_proportion))
Z = stats.norm.ppf(1-0.05)

lb = retired_proportion - (Z*std_dev/np.sqrt(n))
ub = retired_proportion + (Z*std_dev/np.sqrt(n))
```

In [374]:
```
print("For retired the confidence interval was - (" + str(np.round(lb,
4))+", "+str(np.round(ub,4))+")")
```

```
For retired the confidence interval was - (0.2247, 0.2312)
```

In [ ]:
```

```