Data Glacier

Data Science

Project Report : Bank Marketing(Campaign)

Group Name: Project Group 1

Members:

No	Name	Email	Country	Specialization
1	Preeti Verma	vermapreeti.dataanalyst@gmail.com	Canada	Data Science
2	Thanuja Modiboina	thanujayadav953@gmail.com	UK	Data Science
3	Abishek James	abishekjames1998@gmail.com	Ireland	Data Science

Report date: 18-04-2023
Internship Batch: LISUM19
Data intake by: Abishek James
Data intake reviewer: Data Glacier
Data storage location: https://github.com/abishekjames/Data-Glacier-project/tree/main/Week9

Problem Description:

ABC Bank wants to sell it's term deposit product to customers and before launching the product they want to develop a model which help them in understanding whether a particular customer will buy their product or not (based on customer's past interaction with bank or other Financial Institution). This is an application of company's marketing data.

Business Understanding:

The goal is to build Machine Learning model that helps in predicting the outcomes of each customer's marketing campaign and analysing which features have an impact on the outcomes that will help the company to understand how to make the campaign more effective. Additionally, categorizing the customer group that subscribed to the term deposit helps to determine who is more likely to purchase the product in the future, thereby developing more targeted marketing campaigns.

This can be accomplished by using a ML model that shortlists the customers whose possibility of purchasing the product is higher. So that marketing such as telemarketing, SMS or email marketing can concentrate only on those customers. It will save time and resources by doing this.

Data Understanding

The data is related with direct marketing campaigns of a banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed.

Attribute/Features Description:

bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010).

bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.
Dataset have 17 attributes including one dependent attribute and there are 45211 instances/datapoints. So we have 16 predictor/independent attributes and 1 dependent attribute.

- **bank client attributes**:
    - age: age of client (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")
    - education: client highest education (categorical: "unknown", "secondary", "primary", "tertiary")
    - default: has credit in default? (binary/2-categories: "yes", "no")
    - balance: average yearly balance, in euros (numeric)
    - housing: has housing loan? (binary/2-categories: "yes", "no")
    - loan: has personal loan? (binary/2-categories: "yes", "no")
- **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")

Features	Explanations
"Age"	Numeric
"Job"	"Type of job:" Categorical: " Administrative," "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")
"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)
"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)
"Pout come"	"Outcome of the previous marketing campaign" (categorical: "Unknown," "other," "failure," "success")
"Y"	"Has the client subscribed a term deposit?" ("yes" or "no")

Problems in the data

bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.

bank-additional-full.csv contains a total of 41188 rows with 21 columns, out of which there were a total of 12 duplicate rows. The pandas drop_duplicates method is used to drop the duplicate rows

Missing values

```
In [10]: #find percentage of missing values for each column
         missing_values = df.isnull().mean()*100

         missing_values.sum()

Out[10]: 0.0
```

df.isnull().sum() function was utilized to search for missing values in the imported data. As per the observations in the dataset, there is no missing values. Each represents an existing customer that the bank reached via phone calls.

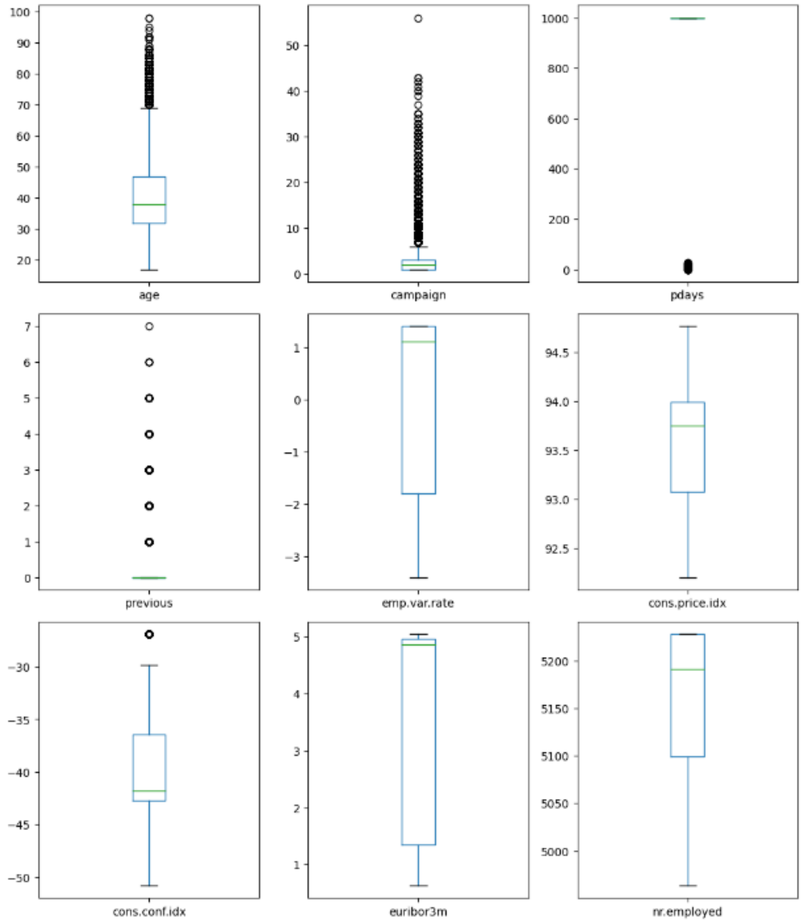
For each observation, the dataset records 16 input variables that stand for both qualitative and quantitative attributes of the customer, such as age, job, housing and personal loan status, account balance, and the number of contacts.

There is no missing value in this dataset. Nevertheless, there are values like "unknown" which are helpless just like missing values.

Outlier Detection

Outliers are datapoints that deviate a lot from the standard dataset. Having outliers in our dataset when training and building a model effects the ultimate accuracy. Therefore, we must find and remove such outliers.

We can only check for outliers in numerical features. Therefore, I have gone through each numerical feature one by one, drawing boxplots to identify outliers and have removed them.
The numerical features I have checked for outliers are 'age', 'day', 'campaign', 'pdays ', and 'previous' which are indicated as data points outside the whiskers of the boxpl ot. After plotting the graph I have found that in 'Emp.var.rate', 'cons.price.idx', 'cons.c onf.idx', 'euribor3m', 'nr.employed' have are no outliers present. So these features ar e dropped.



Approaches to overcome the problem

To have a clear and more accurate sense of the present data I will be displaying general stats.

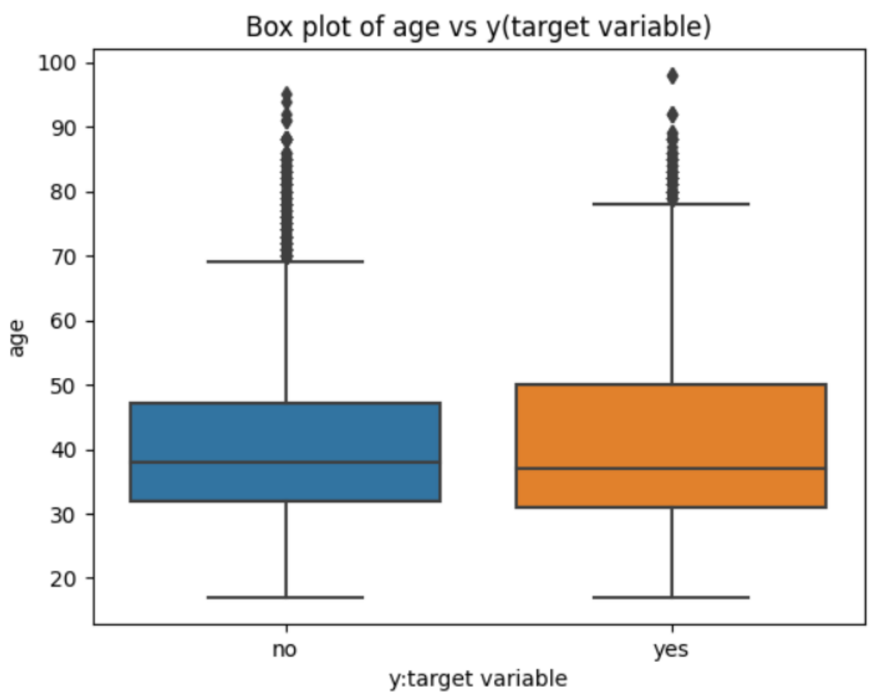
```
In [20]: #checking statistics of outlier features

         df[['age', 'pdays', 'campaign', 'previous']].describe()
```

Out[20]:

	age	pdays	campaign	previous
count	41188.00000	41188.000000	41188.000000	41188.000000
mean	40.02406	962.475454	2.567593	0.172963
std	10.42125	186.910907	2.770014	0.494901
min	17.00000	0.000000	1.000000	0.000000
25%	32.00000	999.000000	1.000000	0.000000
50%	38.00000	999.000000	2.000000	0.000000
75%	47.00000	999.000000	3.000000	0.000000
max	98.00000	999.000000	56.000000	7.000000

Age: the youngest client has 17 years old and the oldest has 98 years with a median of 38 years whereas the average is 40 years old. The distribution is skewed to the left. This possibly indicates the presence of outliers.



Pdays: number of days that passed by after the client was last contacted from a previous campaign. Majority of the clients have the 999 number which indicates most people did not contact nor were contacted by the bank. Those 999 are 'out of range' values. Also, approximately 96% of the rows contain this value, so dropping rows containing 999 is impractical.

```
In [32]: len(df[df['pdays'] ==999])  / len(df) * 100
Out[32]: 96.32174419733903
```

Campaign: "Campaign" contains the number of contracts carried out for this customer during this campaign. According to statistics, the maximum value is 56, which is clearly noise. The figure for 20 or more "campaigns" is about 0.38%. Therefore, it is recommended to substitute the average of the campaign values for these rows

```
In [27]: len(df[df['campaign'] > 20])  / len(df) * 100
Out[27]: 0.3811789841701467

In [29]: df.campaign.describe()
Out[29]: count    41188.000000
         mean         2.567593
         std          2.770014
         min          1.000000
         25%          1.000000
         50%          2.000000
         75%          3.000000
         max         56.000000
         Name: campaign, dtype: float64
```

Previous: Number of contacts performed before this campaign for each client. The vast majority were never been contacted before. The maximum value of 7 does not look like noise, so I decided to ignore this outlier.

```
In [26]: df.previous.describe()

Out[26]: count    41188.000000
         mean         0.172963
         std          0.494901
         min          0.000000
         25%          0.000000
         50%          0.000000
         75%          0.000000
         max          7.000000
         Name: previous, dtype: float64
```

Data processing

- After the data was extracted to achieve the specified goals, the dataset was read into a jupyter notebook using python code because it is flexible in handling large datasets. Anaconda which was also the open-source distribution of python was also used.
- A python feature was being called to replace strings that have space with an underscore, **the isnull().sum()** function was utilized to search for missing values in the imported data.
- The result shows that the data has no missing values, the target variable (y) was renamed to 'signed' for better understanding. with the use of boxplot, outliers were detected from the dataset and this will be properly addressed in the modeling phase. Also, the dataset used for this research work is consistent.
- The target variable(signed) is a binomial classification problem that was categorized into yes or no from the original dataset. As the chosen algorithms work better with numerical values, the yes was encoded with 1 and the no was encoded with 0, and pd.factorized function was used to convert categorical to numerical variables .after carrying out all these functions, resampling technique was carried out due to fact that the dataset is imbalanced.

Data Cleansing and Transformation

1. Deal with missing data

There is no missing value in this dataset. Nevertheless, there are values like “unknown” which are helpless just like missing values. Thus, these ambiguous values are removed from the dataset.

```
In [55]: #changing unknown to null values
         df.replace("unknown", np.nan, inplace=True)

         df.head()
```

Out[55]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.p
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
1	57	services	married	high.school	NaN	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	

5 rows × 21 columns

2. Change column names

```
In [21]: # change column names
         df.rename(columns = {'day_of_week':'day','emp.var.rate':'emp.var','cons.price.idx':'cons.price','euribor3m':'euribor','cons.conf.
```

3. Check potential errors and consistency

```
In [27]: # first, description error: -1 should indicate those that have never been previously contacted
         # second, check consistency with "pdays"
         # Move people who was previously contacted but without exact poutcome("failure" or "success") to "others"
         # Move people who have never been contacted to "unknown"

         df[(df['poutcome']=='others')&(df['pdays']==-1)] # empty dataset
         inconsistent_indices = df[(df['poutcome']=='unknown')&(df['pdays']!=-1)].index
         df.iloc[inconsistent_indices]['poutcome']='other'

         df[(df['pdays']==-1)&(df['previous']!=0)]#empty dataset
         df[(df['pdays']!=-1)&(df['previous']==0)];#empty dataset
```

4. Creating and transforming data

Some changes were made to the column name, units and data types for easier analysis.

Step 1: Change column name: 'y' to 'response'

Step 2: Drop column "contact" which is useless

Step 3: Change the unit of 'duration' from seconds to minutes

Step 4: Change 'month' from words to numbers for easier analysis

In [35]:
```
# Step 1: Change column name: 'y' to 'response'
df.rename(index=str, columns={'y': 'response'}, inplace = True)

def convert(df, new_column, old_column):
    df[new_column] = df[old_column].apply(lambda x: 0 if x == 'no' else 1)
    return df[new_column].value_counts()

convert(df, "response_binary", "response")
```

Out[35]:
```
0    36548
1     4640
Name: response_binary, dtype: int64
```

In [ ]:
```
# Step 2: Drop column "contact" which is useless
dataset5 = dataset4.drop('contact', axis=1)
```

In [67]:
```
# Step 3: Change the unit of 'duration' from seconds to minutes
df['duration'] = df['duration'].apply(lambda n:n/60).round(2)
```

In [38]:
```
# Step 4: Change 'month' from words to numbers for easier analysis
lst = [df]
for column in lst:
    column.loc[column["month"] == "jan", "month_int"] = 1
    column.loc[column["month"] == "feb", "month_int"] = 2
    column.loc[column["month"] == "mar", "month_int"] = 3
    column.loc[column["month"] == "apr", "month_int"] = 4
    column.loc[column["month"] == "may", "month_int"] = 5
    column.loc[column["month"] == "jun", "month_int"] = 6
    column.loc[column["month"] == "jul", "month_int"] = 7
    column.loc[column["month"] == "aug", "month_int"] = 8
    column.loc[column["month"] == "sep", "month_int"] = 9
    column.loc[column["month"] == "oct", "month_int"] = 10
    column.loc[column["month"] == "nov", "month_int"] = 11
    column.loc[column["month"] == "dec", "month_int"] = 12
```

5. Filtering

In [68]:
```
# Step 1: Drop rows that 'duration' < 5s
condition2 = (df['duration']<5/60)
df = df.drop(df[condition2].index, axis = 0, inplace = False)
# Step 2: Drop customer values with 'other' education
condition3 = (df['education'] == 'other')
df = df.drop(df[condition3].index, axis = 0, inplace = False)
```

Handling Outliers

Imputation using mean

In [107]:
```
#The value which is outside the whisker
print(df['age'].quantile(0.95))
```

```
58.0
```

In [108]:
```
#replacing the values which are greater than the 95th percentile
df['age1'] = np.where(df['age'] > 58, 40, df['age'])
df[['age', 'age1']].describe()
```

Out[108]:

	age	age1
count	41188.00000	41188.000000
mean	40.02406	38.992206
std	10.42125	8.858684
min	17.00000	17.000000
25%	32.00000	32.000000
50%	38.00000	38.000000
75%	47.00000	45.000000
max	98.00000	58.000000

Imputation using mode

In [109]:
```
#replacing the values which are greater than the 95th percentile
df['age2'] = np.where(df['age'] > 58, 40.02, df['age'])
df[['age', 'age1','age2']].describe()
```

Out[109]:

	age	age1	age2
count	41188.00000	41188.000000	41188.000000
mean	40.02406	38.992206	38.993011
std	10.42125	8.858684	8.858776
min	17.00000	17.000000	17.000000
25%	32.00000	32.000000	32.000000
50%	38.00000	38.000000	38.000000
75%	47.00000	45.000000	45.000000
max	98.00000	58.000000	58.000000

The statistics of the dataset after median and mean imputation remain roughly the same.