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

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

In [38]:
```python
# 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]:
```python
# 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]:
```python
#The value which is outside the whisker
print(df['age'].quantile(0.95))
```

```
58.0
```

In [108]:
```python
#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.