Data Science Internship

Week 9: Data Science Project: Bank Marketing (Campaign)

Data Cleansing and Transformation

Group Name: One
Name: Hira Fahim
Email: hirashahidd26@yahoo.com
Country: United Kingdom
Company: Unemployed
Specialization: Data Science
Batch Code: LISUM19
Submission Date: 10th April 2023
Submitted to: Data Glacier

Table of Contents

1. Problem Description

ABC Bank wants to sell its 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).

2. Dataset Information

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

3. Problems in the data

3.1. Null values

```
# total null values in the dataset
d.isnull().sum()
```

```
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 values** in the dataset.

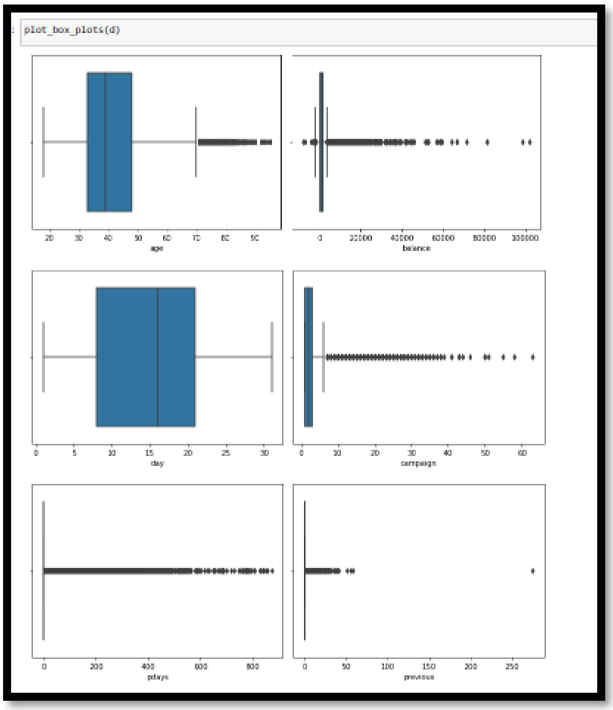
3.2. Outliers Detection

- Description of Numerical column

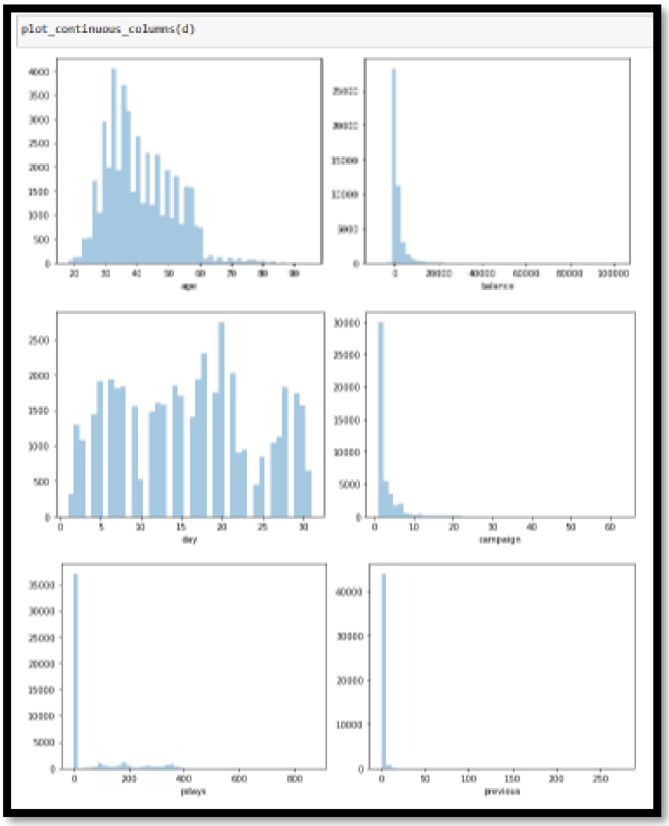
```
# Description of numerical columns
d.describe()
```

	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

➢ Numerical variables’ visualization

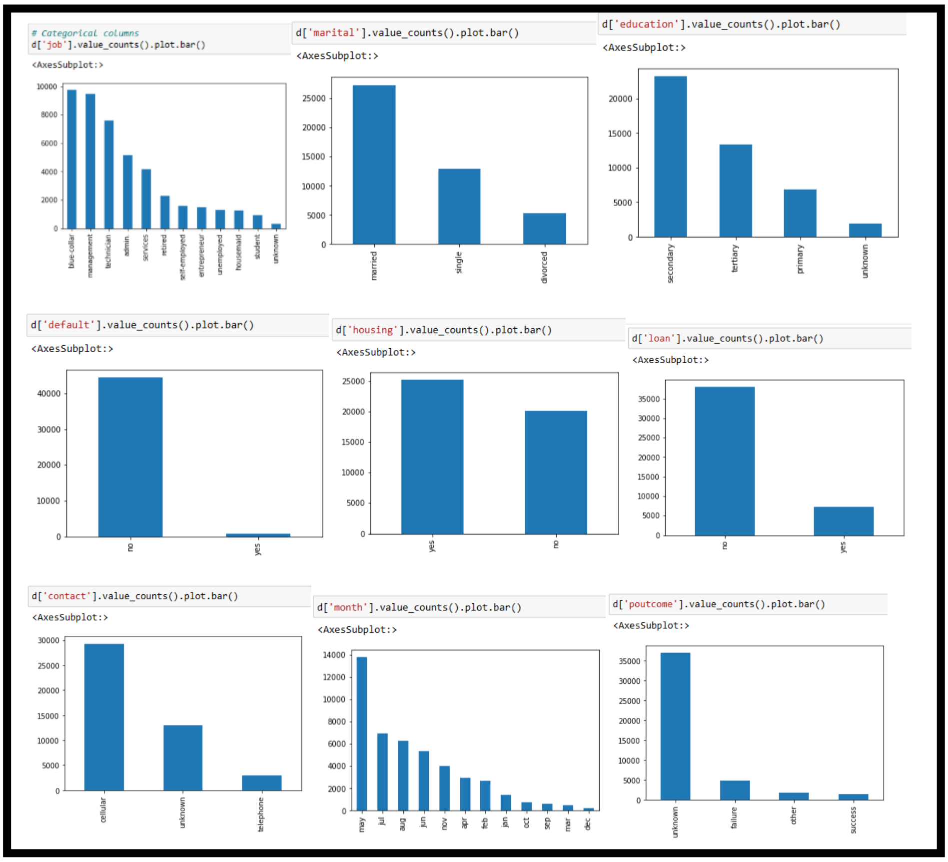


From **description** and **boxplot**, we can see there are outliers in numerical input variables like age, balance, campaign, pdays and previous.



In Histogram, we can see input variables like age, balance, campaign, pdays and previous are **positively skewed**, and we can also see uneven distribution of data in day column.

➢ Categorical data visualization



In **Bar chart** of categorical columns, we see uneven distribution of data in almost all the input categorical columns.

4. Transformation

4.1. Data set information

```
# Datatypes of columns and non-null values
d.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

As per dataset information, the data types of columns are integer and object, but we know object columns are categorical columns so, first we convert the data types of categorical columns into 'category.'

4.2. Convert data type.

```
# change datatype of categorical columns into "category"
d["job"]=d["job"].astype("category")
d["marital"]=d["marital"].astype("category")
d["education"]=d["education"].astype("category")
d["default"]=d["default"].astype("category")
d["housing"]=d["housing"].astype("category")
d["loan"]=d["loan"].astype("category")
d["contact"]=d["contact"].astype("category")
d["month"]=d["month"].astype("category")
d["poutcome"]=d["poutcome"].astype("category")
d["y"]=d["y"].astype("category")
```

```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  category
 2   marital    45211 non-null  category
 3   education  45211 non-null  category
 4   default    45211 non-null  category
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  category
 7   loan       45211 non-null  category
 8   contact    45211 non-null  category
 9   day        45211 non-null  int64
 10  month      45211 non-null  category
 11  campaign   45211 non-null  int64
 12  pdays      45211 non-null  int64
 13  previous   45211 non-null  int64
 14  poutcome   45211 non-null  category
 15  y          45211 non-null  category
dtypes: category(10), int64(6)
memory usage: 2.8 MB
```

Here we see the data types of all categorical column is 'category.'

4.3. Encoding – Label encoding

All machine learning algorithms work with only numerical values so, second transformation that is needed to be done is to convert all categorical columns into numerical columns. Here we use label encoding technique for conversion.

```
from sklearn import preprocessing
```

```
le=preprocessing.LabelEncoder()
d['job']=le.fit_transform(d['job'])
d['marital']=le.fit_transform(d['marital'])
d['education']=le.fit_transform(d['education'])
d['default']=le.fit_transform(d['default'])
d['housing']=le.fit_transform(d['housing'])
d['loan']=le.fit_transform(d['loan'])
d['contact']=le.fit_transform(d['contact'])
d['month']=le.fit_transform(d['month'])
d['poutcome']=le.fit_transform(d['poutcome'])
```

```
d.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome	y
0	58	4	1	2	0	2143	1	0	2	5	8	1	-1	0	3	0
1	44	9	2	1	0	29	1	0	2	5	8	1	-1	0	3	0
2	33	2	1	1	0	2	1	1	2	5	8	1	-1	0	3	0
3	47	1	1	3	0	1506	1	0	2	5	8	1	-1	0	3	0
4	33	11	2	3	0	1	0	0	2	5	8	1	-1	0	3	0

5. Data Cleansing

5.1. Feature Scaling -Normalization

To deal with noises in the data, we need to perform feature scaling and as there are both continuous and discrete columns, we are using **normalization scaling technique** to transform features to be on a similar scale. **This improves the performance and training stability of the model**.

```
In [67]: from numpy import set_printoptions
         from sklearn.preprocessing import MinMaxScaler
```

```
In [68]: d1=d.iloc[:,:-1]
         d1
```

Out[68]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome
0	58	4	1	2	0	2143	1	0	2	5	8	1	-1	0	3
1	44	9	2	1	0	29	1	0	2	5	8	1	-1	0	3
2	33	2	1	1	0	2	1	1	2	5	8	1	-1	0	3
3	47	1	1	3	0	1506	1	0	2	5	8	1	-1	0	3
4	33	11	2	3	0	1	0	0	2	5	8	1	-1	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	51	9	1	2	0	825	0	0	0	17	9	3	-1	0	3
45207	71	5	0	0	0	1729	0	0	0	17	9	2	-1	0	3
45208	72	5	1	1	0	5715	0	0	0	17	9	5	184	3	2
45209	57	1	1	1	0	668	0	0	1	17	9	4	-1	0	3
45210	37	2	1	1	0	2971	0	0	0	17	9	2	188	11	1

45211 rows × 15 columns

```
In [69]: array=d1.values
         scaler=MinMaxScaler(feature_range=(0,1))
         rescaledX=scaler.fit_transform(array)

         set_printoptions(precision=2)
         print(rescaledX[0:5,:])

[[0.52 0.36 0.5  0.67 0.   0.09 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.34 0.82 1.   0.33 0.   0.07 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.19 0.18 0.5  0.33 0.   0.07 1.   1.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.38 0.09 0.5  1.   0.   0.09 1.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]
 [0.19 1.   1.   1.   0.   0.07 0.   0.   1.   0.13 0.73 0.   0.   0.
  1.  ]]
```

In [70]:

```
d2=pd.DataFrame(rescaledX,columns=["age","job","marital","education","default","balance","housing","loan","contact","day",
                                   "month","campaign","pdays","previous","poutcome"])
d2
```

Out[70]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	campaign	pdays	previous	poutcome
0	0.519481	0.363636	0.5	0.666667	0.0	0.092259	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
1	0.337662	0.818182	1.0	0.333333	0.0	0.073067	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
2	0.194805	0.181818	0.5	0.333333	0.0	0.072822	1.0	1.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
3	0.376623	0.090909	0.5	1.000000	0.0	0.086476	1.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
4	0.194805	1.000000	1.0	1.000000	0.0	0.072812	0.0	0.0	1.0	0.133333	0.727273	0.000000	0.000000	0.000000	1.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45206	0.428571	0.818182	0.5	0.666667	0.0	0.080293	0.0	0.0	0.0	0.533333	0.818182	0.032258	0.000000	0.000000	1.000000
45207	0.688312	0.454545	0.0	0.000000	0.0	0.088501	0.0	0.0	0.0	0.533333	0.818182	0.016129	0.000000	0.000000	1.000000
45208	0.701299	0.454545	0.5	0.333333	0.0	0.124689	0.0	0.0	0.0	0.533333	0.818182	0.064516	0.212156	0.010909	0.666667
45209	0.506494	0.090909	0.5	0.333333	0.0	0.078868	0.0	0.0	0.5	0.533333	0.818182	0.048387	0.000000	0.000000	1.000000
45210	0.246753	0.181818	0.5	0.333333	0.0	0.099777	0.0	0.0	0.0	0.533333	0.818182	0.016129	0.216743	0.040000	0.333333

45211 rows × 15 columns