WEEK 8 DELIVERABLES

Team Member's Details

Group Name: Data Bank

Tevfik SASTIM, talfik22@gmail.com, Turkey, Self- Employed, Data Science

Marelly Escelade, idemany1@gmail.com, Peru, Self- Employed, Data Science

Problem Statement:

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).

We are now working on this problem through the following steps to find best solution for our customer to give the bank ability to better target its customers and personalize the marker plan.

Github Repo Link

Github Repo link: https://github.com/talfik2/Data-Glacier-Data-Bank-Bank-Marketing-Group-Project

Link for Step 2: https://github.com/talfik2/Data-Glacier-Data-Bank-Bank-Marketing-Group-Project/tree/Step-2-Data-Understanding

Data Understanding

Data Set 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, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Attribute Information:

age: (numeric)

job: type of job (categorical)

marital: marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

education: (categorical)

default: has credit in default? (categorical: 'no','yes','unknown')

housing: has housing loan? (categorical: 'no','yes','unknown')

loan: has personal loan? (categorical: 'no','yes','unknown')

balance: deposit amount (numeric)

contact: contact communication type (categorical: 'cellular','telephone')

month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

day: last contact day (numeric)

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)

previous: number of contacts performed before this campaign and for this client (numeric)

poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

deposit: has the client subscribed a term deposit? (binary: 'yes','no')

In [5]:
```
df.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
```

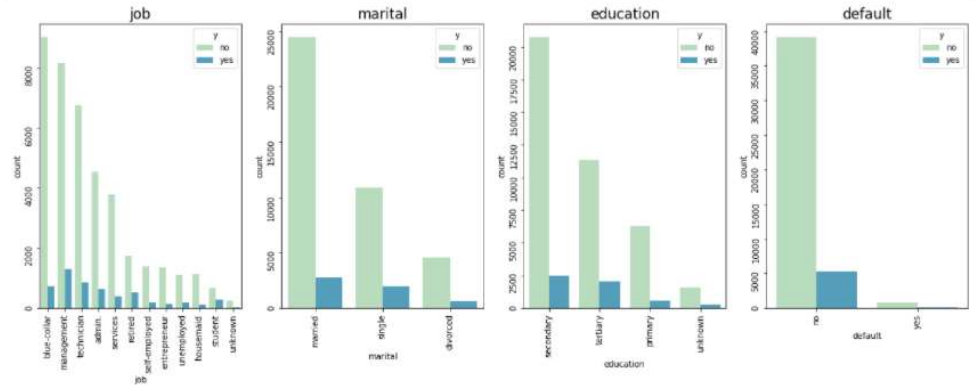
There are no nulls values

In [6]:
```
df.describe()
```

Out[6]:

	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


```
plt.show()
```



Problems of Data

1) Na Values

This data have NA values but not in the shape of np.na. , but as unknown keywords. Thus, first, I replaced them with np.nan,

```
In [5]: # encoding y
obj_df = obj_df.replace({"y":{"no":0,"yes":1}})
# encoding poutcome
obj_df = obj_df.replace({"poutcome":{"unknown":np.nan,"failure":0, "other":1, "success":2}})
# encoding contact
obj_df = obj_df.replace({"contact":{"cellular":0, "unknown":np.nan, "telephone":1}})
# encoding loan
obj_df = obj_df.replace({"loan":{"no":0,"yes":1}})
# encoding housing
obj_df = obj_df.replace({"housing":{"no":0,"yes":1}})
# encoding default
obj_df = obj_df.replace({"default":{"no":0,"yes":1}})
# encoding education
obj_df = obj_df.replace({"education":{"unknown":np.nan,"secondary":0, "tertiary":1, "primary":2}})
# encoding marital
obj_df = obj_df.replace({"marital":{"married":0, "single":1, "divorced":1}})
# encoding job
obj_df = obj_df.replace({"job":{"unknown":np.nan,"blue-collar":0, "management":1, "technician":2,
                                "admin.":3, "services": 4, "retired": 5, "self-employed":6,
                                "entrepreneur": 7, "unemployed": 8, "housemaid": 9, "student":10}})

In [6]: print(obj_df.head(3))
obj_df.dtypes

   job  marital  education  default  housing  loan  contact  poutcome  y
0  1.0        0        1.0        0        1     0      NaN       NaN  0
1  2.0        1        0.0        0        1     0      NaN       NaN  0
2  7.0        0        0.0        0        1     1      NaN       NaN  0
```

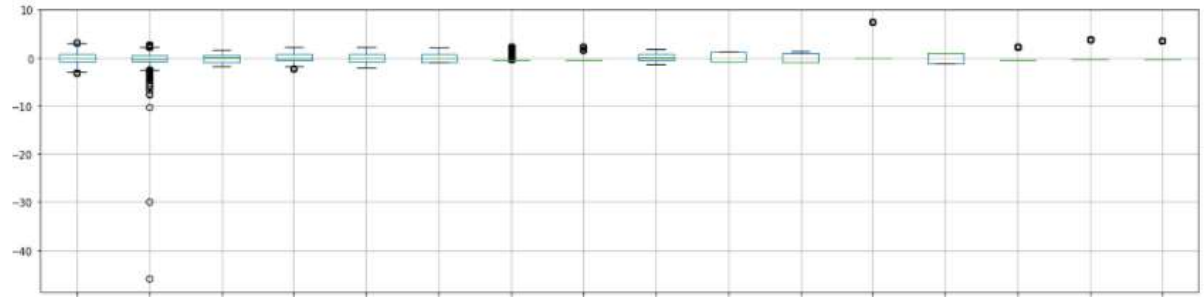

Then, I confronted NA values with sklearn's SimpleImputer as it can replace them with Most Frequent Values of each column iteratively.

```
In [6]: from sklearn.impute import SimpleImputer
imr = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

imr = imr.fit(obj_df)
imputed_data = imr.transform(obj_df)
obj_df = pd.DataFrame(imputed_data)
# As SimpleImputer removes all column names as indexes, I am going to rename them.
obj_df = obj_df.rename(columns = {0:'job' ,1:'marital' ,2:'education' , 3:'default' ,
                                  4:'housing' , 5:'loan' ,6:'contact'
                                  , 7:'poutcome' , 8:'y' })

In [19]: print("First 3 rows of  obj_df: ",obj_df.head(3))
print("Is there any Na values in obj_df:", obj_df.isnull().values.any())

First 3 rows of  obj_df:     job  marital  education  default  housing  loan  contact  poutcome    y
0  1.0      0.0        1.0      0.0      1.0   0.0      0.0       0.0  0.0
1  2.0      1.0        0.0      0.0      1.0   0.0      0.0       0.0  0.0
2  7.0      0.0        0.0      0.0      1.0   1.0      0.0       0.0  0.0
Is there any Na values in obj_df: False
```

2) Outlier Detection & Elimination

There are outliers in this data as it can be seen below. I detected them by sns.boxplot

```
In [40]: raw_transformalized.boxplot(figsize = (20,5))
         plt.show()
```



I eliminated outliers by using cut_off method iteratively as it is more convinient for categorical data.

```
In [32]: y = pd.DataFrame(rawy["y"])
         banking = raw_transformalized.join(y)
```

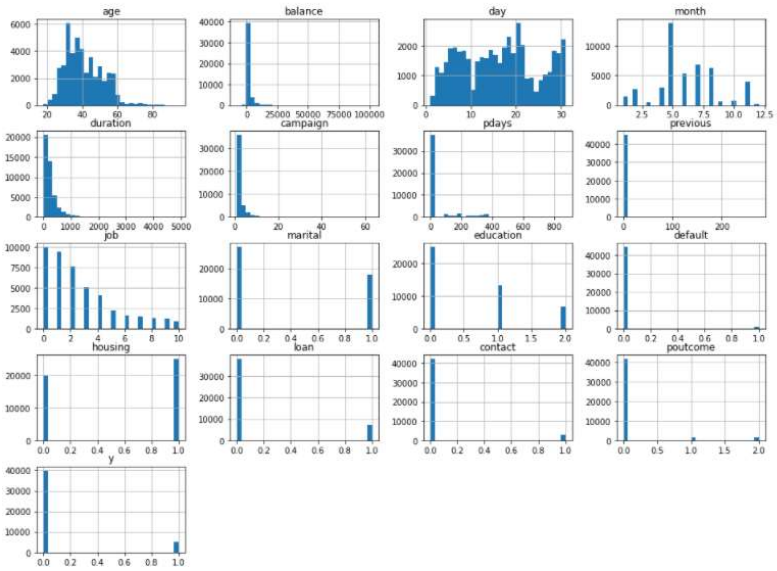
```
In [33]: for i in banking:
             y = banking["y"]
             mean = banking[i].mean()
             std = banking[i].std()
             cut_off = std * 3
             lower, upper = mean - cut_off, mean + cut_off
             if banking[i] is y:
                 continue
             banking = banking[(banking[i] < upper) & (banking[i] > lower)]
```

```
In [34]: banking.shape
Out[34]: (38435, 17)
```

3) Imbalanced Dataset Problem

Our data is highly imbalanced. I examined it as below:

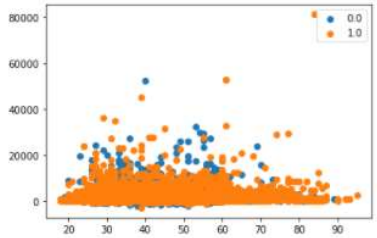
```
[<AxesSubplot:title={'center':'housing'}>,
 <AxesSubplot:title={'center':'loan'}>,
 <AxesSubplot:title={'center':'contact'}>,
 <AxesSubplot:title={'center':'poutcome'}>],
[<AxesSubplot:title={'center':'y'}>, <AxesSubplot:>,
 <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



By imblearn's Random Undersampling

In [175]:
```
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.under_sampling import RandomUnderSampler
from numpy import mean

# Summarized Class Distribution
counter = Counter(rawy["y"])
print("Summarized Class Distribution before Undersampling: ", counter)
# Define the undersampling method
undersample = RandomUnderSampler(random_state = 42)
# transform the dataset
X, y = undersample.fit_resample(rawy.drop(columns = "y"),rawy["y"])
# Summarize the new class distribution
counter = Counter(y)
print("Summarized Class Distribution after Undersampling: ", counter)
# Evaluating Pipeline by plotting it
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    plt.scatter(X.iloc[row_ix, 0], X.iloc[row_ix, 1], label=str(label))
plt.legend()
plt.show()
```

```
Summarized Class Distribution before Undersampling:  Counter({0.0: 39922, 1.0: 5289})
Summarized Class Distribution after Undersampling:  Counter({0.0: 5289, 1.0: 5289})
```



- We can see that that majority class is undersampled to have the same number of examples as the minority class.

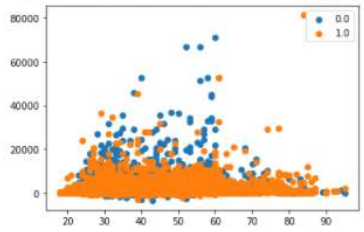
By Near Miss Undersampling(3)

Near Miss(3) Undersampling

- NearMiss-3 selects the closest examples from the majority class for each minority class.

In [173]:
```
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.under_sampling import NearMiss
from numpy import mean

# Summarized Class Distribution
counter = Counter(rawy["y"])
print("Summarized Class Distribution before Undersampling: ", counter)
# Define the undersampling method
undersample = NearMiss(sampling_strategy = 0.3,version = 3, n_neighbors_ver3 = 3)
# transform the dataset
X, y = undersample.fit_resample(rawy.drop(columns = "y"),rawy["y"])
# Summarize the new class distribution
counter = Counter(y)
print("Summarized Class Distribution after Undersampling: ", counter)
# Evaluating Pipeline by plotting it
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    plt.scatter(X.iloc[row_ix, 0], X.iloc[row_ix, 1], label=str(label))
plt.legend()
plt.show()
```

```
Summarized Class Distribution before Undersampling:  Counter({0.0: 39922, 1.0: 5289})
C:\Users\talfi\anaconda3\lib\site-packages\imblearn\under_sampling\_prototype_selection\_nearmiss.py:175: UserWarning: The number of the
samples to be selected is larger than the number of samples available. The balancing ratio cannot be ensure and all samples will be retur
ned.
  warnings.warn(
Summarized Class Distribution after Undersampling:  Counter({0.0: 8695, 1.0: 5289})
```



In this step, feature scaling is also applied to data.

Feature Scaling

- As our data has strict upper & lower bounds, I am going to use Normalization instaad of standardization.
- Standardization also scales the data in it's original shape. This is the other reason for using scaling because our data is categorical data & I am going to apply classification algorithm to our data. In that case, standardization will prevent bias.
- As y is the target variable, I am going to remove y for feature scaling.

In [27]:
```
#removing y
raw = rawy.drop(columns="y")
```

Normalization

In [28]:
```
from sklearn.preprocessing import MinMaxScaler# a.k.a. Normalization
scaler = MinMaxScaler()
scaler.fit(raw)
raw_normalize = scaler.transform(raw)
raw_normalized = pd.DataFrame(raw_normalize)
```

Transformation

- For transformation purposes, I am going to use Log Transformation

In [29]:
```
from sklearn.preprocessing import PowerTransformer
log = PowerTransformer()
log.fit(raw_normalized)
raw_transformalize = log.transform(raw_normalize)
raw_transformalized = pd.DataFrame(raw_transformalize)
```

In [62]:
```
raw_transformalized.head()
```

Out[62]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1.484740	0.918253	-1.333228	-0.444966	0.421659	-1.037276	-0.467895	-0.469277	-0.567670	-0.813212	0.888372	-0.13549	0.893915	-0.436803	-0.262
1	0.429975	-0.752042	-1.333228	-0.444966	-0.338211	-1.037276	-0.467895	-0.469277	0.020550	1.229691	-0.880131	-0.13549	0.893915	-0.436803	-0.262
2	-0.722011	-0.781658	-1.333228	-0.444966	-1.047380	-1.037276	-0.467895	-0.469277	1.506937	-0.813212	-0.880131	-0.13549	0.893915	2.289359	-0.262
3	0.688427	0.528783	-1.333228	-0.444966	-0.880257	-1.037276	-0.467895	-0.469277	-1.356792	-0.813212	-0.880131	-0.13549	0.893915	-0.436803	-0.262
4	-0.722011	-0.782760	-1.333228	-0.444966	0.021445	-1.037276	-0.467895	-0.469277	-1.356792	1.229691	-0.880131	-0.13549	-1.118674	-0.436803	-0.262

- Normalization & Transformation removed the column names. Let's rename them