
# Bank Marketing campaign analysis

Conducted by Cao Duc Thanh

# OVERVIEW

1. **Data overview**
2. **Data cleaning**
3. **Exploratory Data Analysis**
4. **Building models**
5. **Conclusion**


# PART 1: DATA OVERVIEW

This dataset is about a bank's marketing campaign, conducted through calling customers and offering **a term deposit product.** If the customer agrees to open a deposit account, the customer's data will be marked as "**yes**", otherwise marked as "**no**".

# OBJECTIVES

- Predicting the future results of marketing campaigns based on available statistics and, accordingly, formulating recommendations for such campaigns in the future.
- Building a profile of a consumer of banking services.

# FEATURE DESCRIPTION

## Features about client data

1. age (numeric)
2. job : type of job (12 types of job)
3. marital : marital status (categorical: divorced, married, single, unknown; note: divorced means divorced or widowed)
4. education (categorical: basic.4y, basic.6y, basic.9y, high.school, illiterate, professional.course, university.degree, unknown)
5. default: has credit in default? (categorical: no, yes, unknown)
6. housing: has housing loan? (categorical: no, yes, unknown)
7. loan: has personal loan? (categorical: no, yes, unknown)

# FEATURE DESCRIPTION

## Features about the closet contact information

8. contact: contact communication type (categorical: cellular, telephone)
9. month: last contact month of year (categorical: jan, feb, mar, ..., nov, dec)
10. day_of_week: last contact day of the week (categorical: mon, tue, wed, thu, fri)
11. duration: last contact duration, in seconds (numeric)

## Other features

12. campaign: number of contacts performed during this campaign and for this client
13. pdays: number of days that passed by after the client was last contacted from a previous campaign
14. previous: number of contacts performed before this campaign and for this client
15. poutcome: outcome of the previous marketing campaign (categorical:  failure, nonexistent,  success)

# FEATURE DESCRIPTION

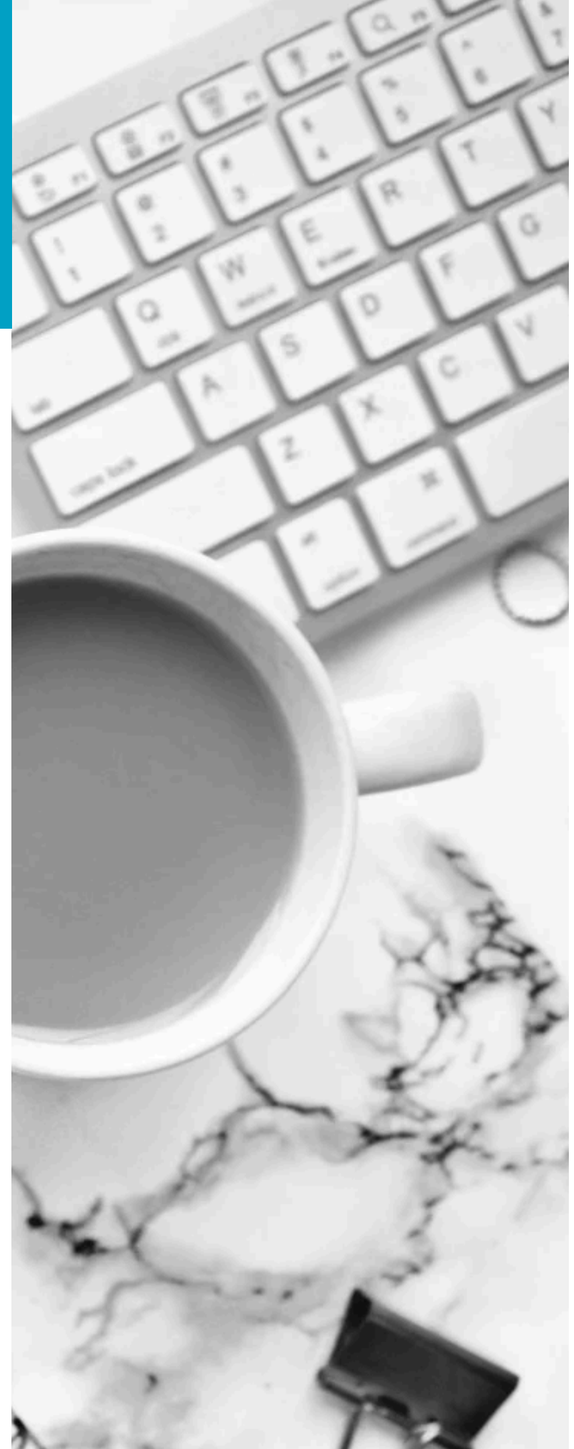
## Contextual features

16. emp.var.rate: employment variation rate - quarterly indicator (numeric)
17. cons.price.idx: consumer price index - monthly indicator (numeric)
18. cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19. euribor3m: euribor 3 month rate - daily indicator (numeric)
20. nr.employed: number of employees - quarterly indicator (numeric)

## Output

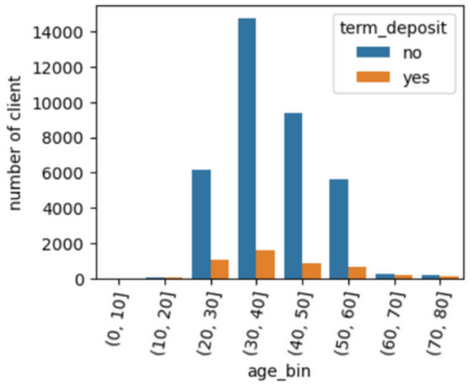
21. y - has the client subscribed a term deposit? (binary: yes, no)

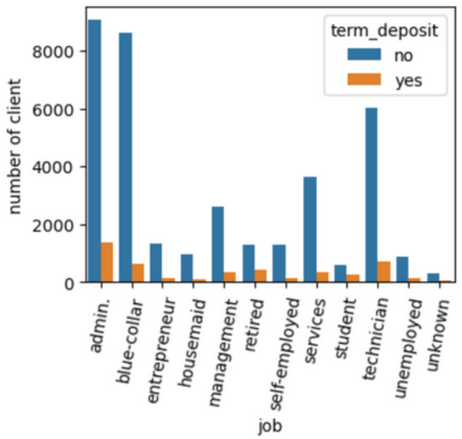
CONDUCTED BY CAO DUC THANH

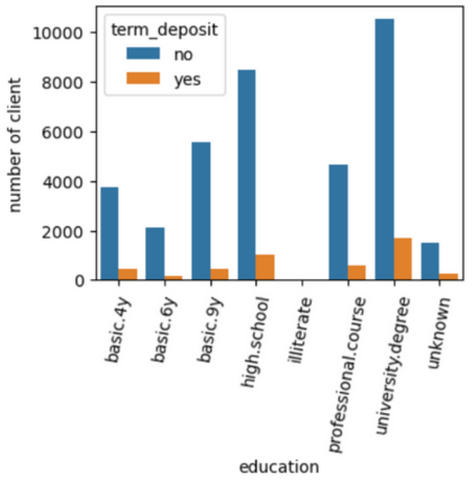


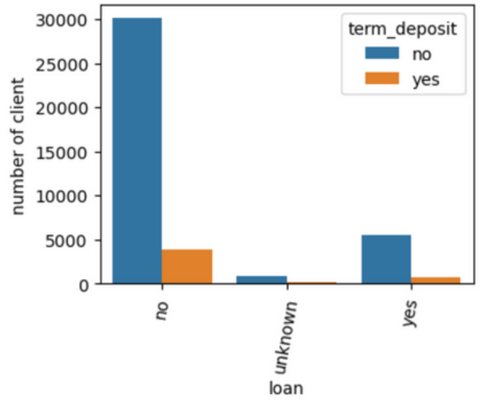
# PART 2: EXPLORATORY DATA ANALYSIS

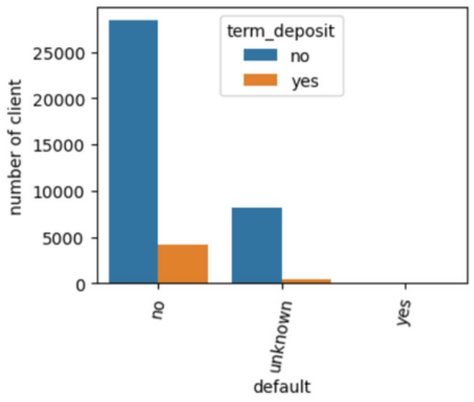
# Distribution of features about client data

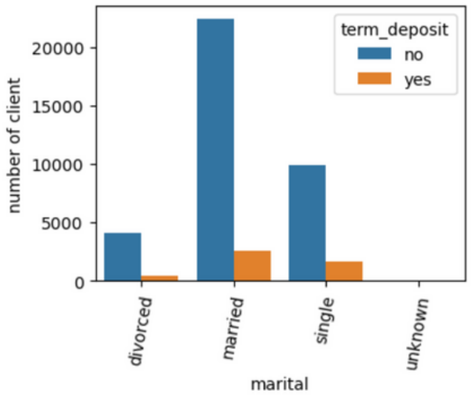


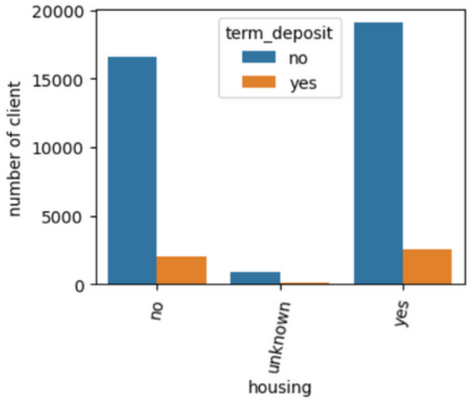



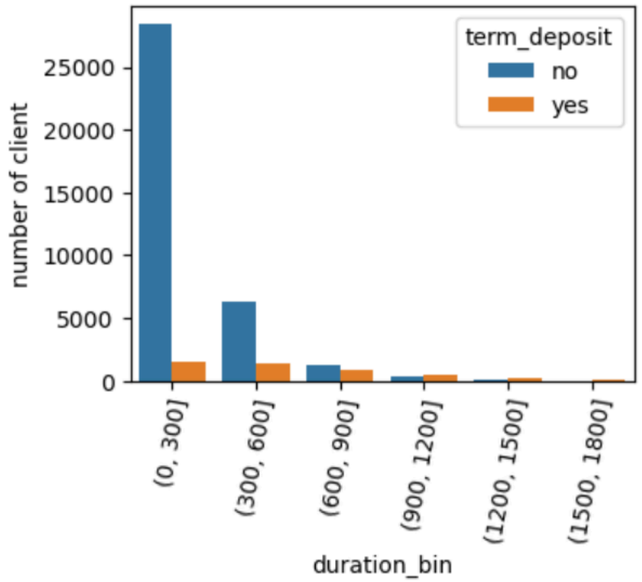



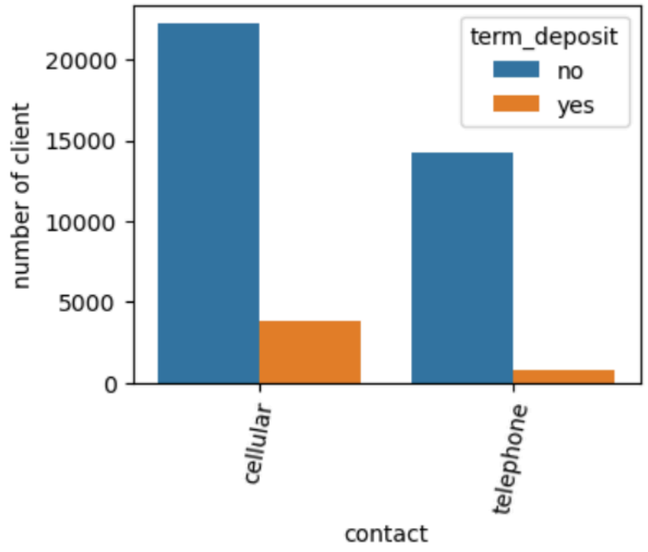



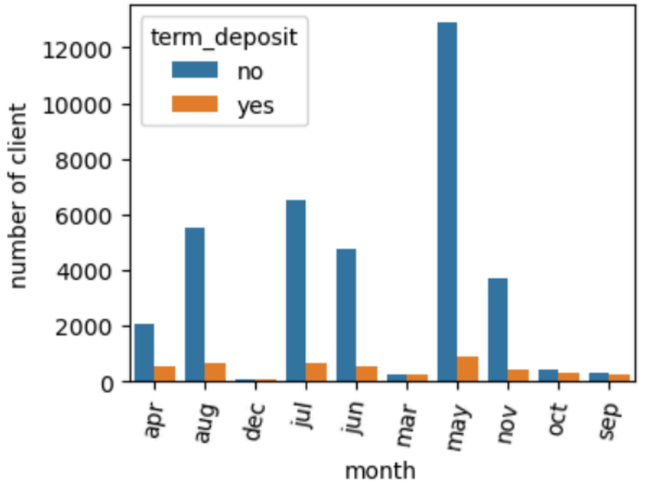



# Distribution of features about the last contact with clients



# QUICK INSIGHTS

The group aged 30-40 had the highest number of agreements, but the group 20-30 had the highest agreement rate.

**Age**

The groups that graduated from high school and university were approached the most frequently and had the highest number of people agreeing.

**Literacy**

Contact time with customers was usually less than 5 minutes. However, the longer the contact, the higher the likelihood of the customer agreeing.

**Duration**

The largest number of customers were approached and agreed to open accounts in May.

**Month**

Most customers had not had a term deposit account yet.

**Default**

CONDUCTED BY CAO DUC THANH


# PART 3: DATA CLEANING AND PREPARATION FOR MODELING

# ENCODE AND NORMALIZE DATA

**1. Create a dataframe with discrete features**

```
# Prepare dataframe with discrete variables
df_discrete = df[['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
                  'month', 'day_of_week', 'poutcome', 'y']]
```

**2. Encode discrete features**

```
# Encode the dataframe with discrete variables
from sklearn.preprocessing import LabelEncoder
df_discrete = df_discrete.apply(LabelEncoder().fit_transform)
```

**3. Create a dataframe with continuous features**

```
# Prepare dataframe with continuous variables
df_con = df[['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
             'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']]
```

**4. Normalize continuous features**

```
# Normalize the continuous variables
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_con)
df_con = pd.DataFrame(scaler.transform(df_con), columns= df_con.columns)
```

# CONCATNATE THE TWO DATAFRAMES

75%

```
# Concat the two dataframes into a single dataframe
df_prepared = pd.concat((df_discrete, df_con), axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

# CLEAN THE DATA

## Count the duplicates

```
# find duplicates:
print(f'the number of duplicated data is: {df_prepared.duplicated().sum()}')
```

✓ 0.0s

```
the number of duplicated data is: 12
```

## Drop the duplicates

```
# drop duplicates:
df_prepared = df_prepared.drop_duplicates(keep= 'first')
print(f'After removing duplicates, there are {df_prepared.shape[0]}
    observations and {df_prepared.shape[1]} features')
```

✓ 0.0s

```
After removing duplicates, there are 41176 observations and 21 features
```

CONDUCTED BY CAO DUC THANH

# PART 4: BUILDING MODELS

# USE MODELS TO EVALUATE AND COMPARE

## Split data into train and test sets

```
# Split data into train and test set:
from sklearn.model_selection import train_test_split
X = df_prepared.drop(columns = 'y')
y = df_prepared['y']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 42)
```

## Models:

1 - Logistic regression

2 - Gaussian Naive Bayes Classifier

3 - Decision tree Classifier

4 - Random forest Classifier

5 - KNeighbor Classifier

# LOGISTIC REGRESSION

**Fit data into the model:**

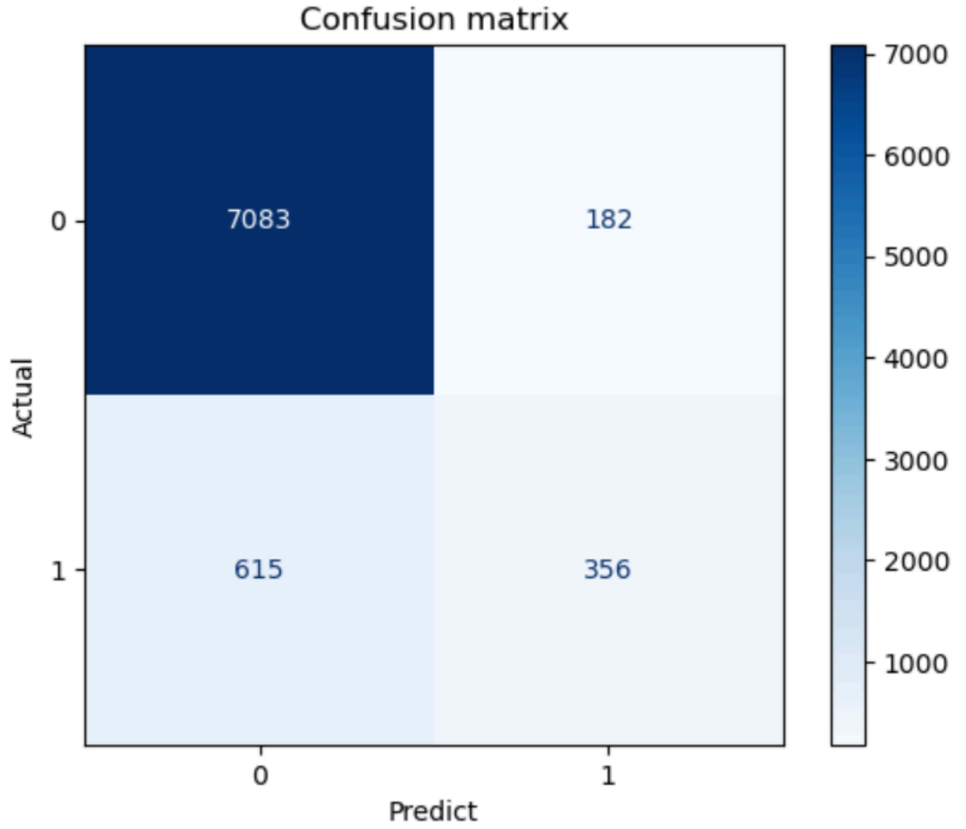
```
# fit model:
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

✓ 0.2s

```
LogisticRegression()
```

```
# predict train set:
y_pred_lr = lr.predict(X_test)
```

✓ 0.0s




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.92 | 0.97 | 0.95 | 7265 |
| 1 | 0.66 | 0.37 | 0.47 | 971 |
| accuracy | | | 0.90 | 8236 |
| macro avg | 0.79 | 0.67 | 0.71 | 8236 |
| weighted avg | 0.89 | 0.90 | 0.89 | 8236 |

# GAUSSIAN NAIVE BAYES CLASSIFIER

**Fit data into the model:**

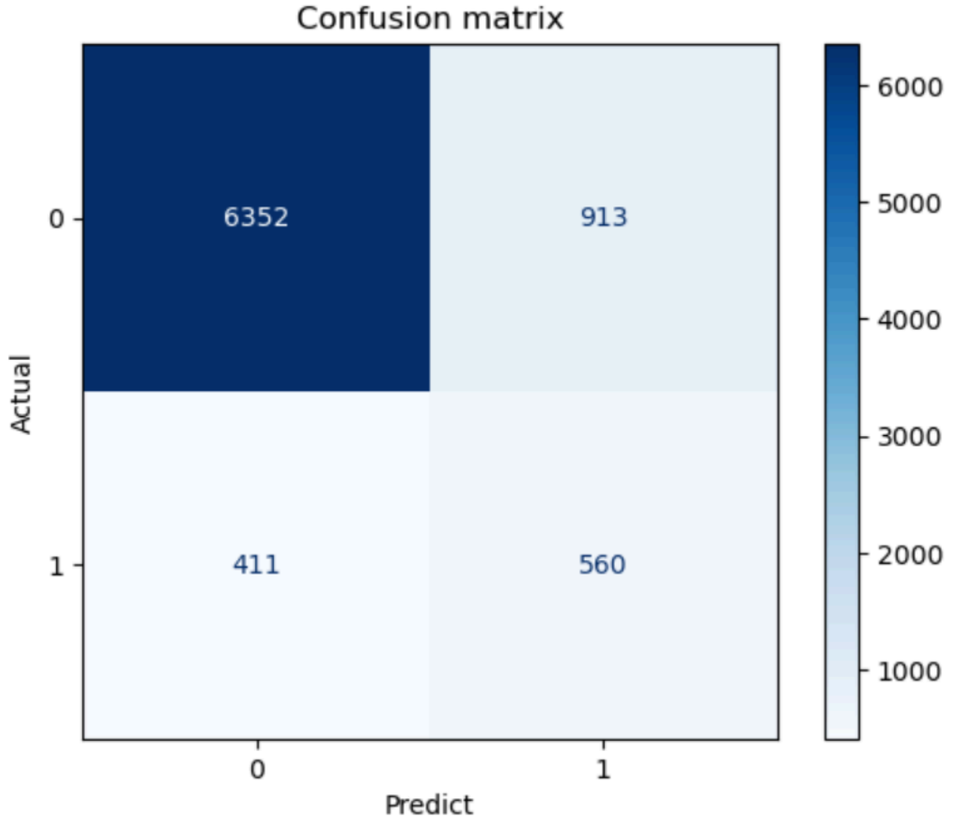
```
# fit model
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

✓ 0.0s

```
GaussianNB()
```

```
# predict test set:
y_pred_gnb = gnb.predict(X_test)
```

✓ 0.0s




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.94 | 0.87 | 0.91 | 7265 |
| 1 | 0.38 | 0.58 | 0.46 | 971 |
| accuracy | | | 0.84 | 8236 |
| macro avg | 0.66 | 0.73 | 0.68 | 8236 |
| weighted avg | 0.87 | 0.84 | 0.85 | 8236 |

# DECISION TREE CLASSIFIER

**Fit data into the model:**

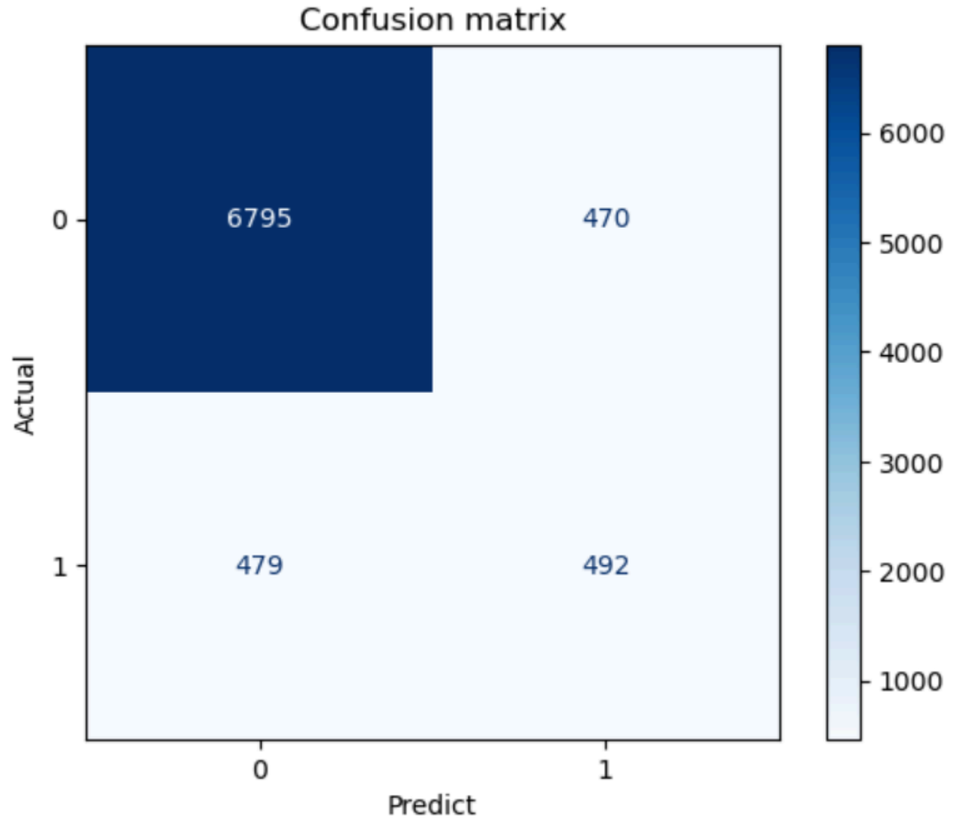
```
# fit model:
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

✓ 0.2s

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# predict test set:
y_pred_dt = dt.predict(X_test)
```

✓ 0.0s




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.93 | 0.94 | 0.93 | 7265 |
| 1 | 0.51 | 0.51 | 0.51 | 971 |
| accuracy | | | 0.88 | 8236 |
| macro avg | 0.72 | 0.72 | 0.72 | 8236 |
| weighted avg | 0.88 | 0.88 | 0.88 | 8236 |

# RANDOM FOREST CLASSIFIER

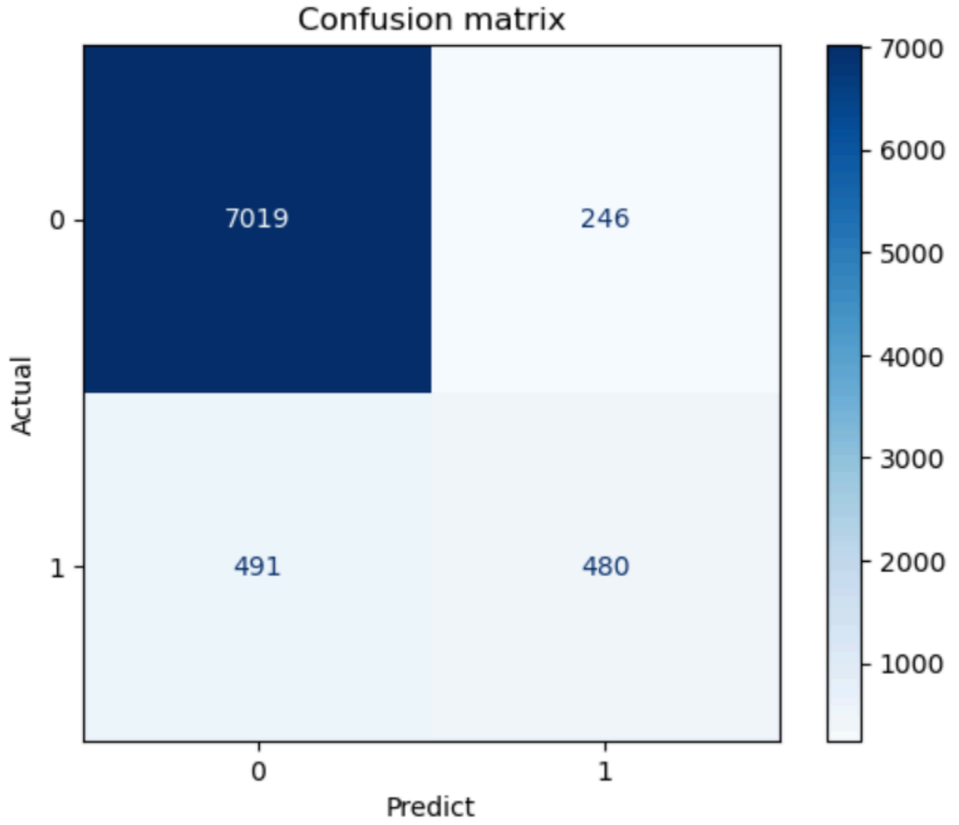
**Find the best random state for the model**

```
accuracy_score_lst = []
for i in range(1,100,1):
    rf = RandomForestClassifier(n_estimators= 50, random_state= i)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    accuracy_score_lst.append(accuracy_score(y_test, y_pred_rf))

max_score = 0
n_max_score = 0
n = 0
while n < len(accuracy_score_lst):
    if accuracy_score_lst[n] >= max_score:
        max_score = accuracy_score_lst[n]
        n_max_score = n
    n+=1
print(f'with random state is: {n_max_score + 1},
      the accuracy score is maximum at: {max_score*100:,.2f}%')
```

✓ 2m 50.7s

```
with random state is: 18, the accuracy score is maximum at: 91.34%
```




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.93 | 0.97 | 0.95 | 7265 |
| 1 | 0.66 | 0.49 | 0.57 | 971 |
| accuracy | | | 0.91 | 8236 |
| macro avg | 0.80 | 0.73 | 0.76 | 8236 |
| weighted avg | 0.90 | 0.91 | 0.90 | 8236 |

# KNEIGHBOR CLASSIFIER

**Fit data into the model:**

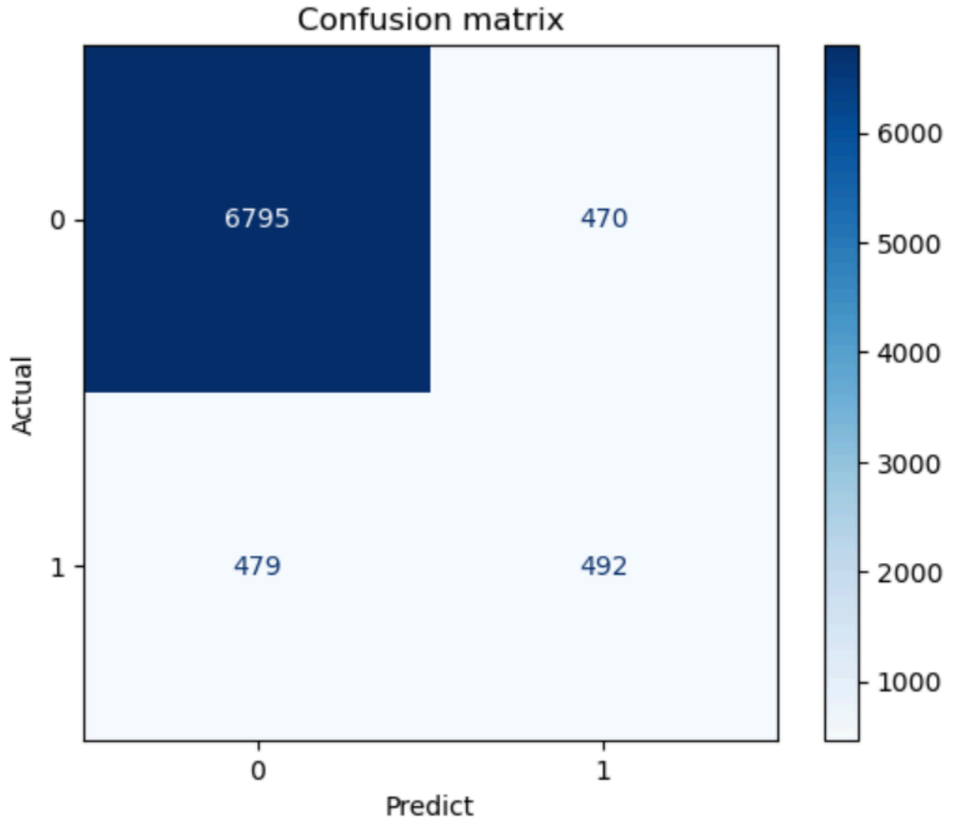
```
# fit model
kn = KNeighborsClassifier(n_neighbors= 10)
kn.fit(X_train, y_train)
```

✓ 0.0s

```
KNeighborsClassifier(n_neighbors=10)
```

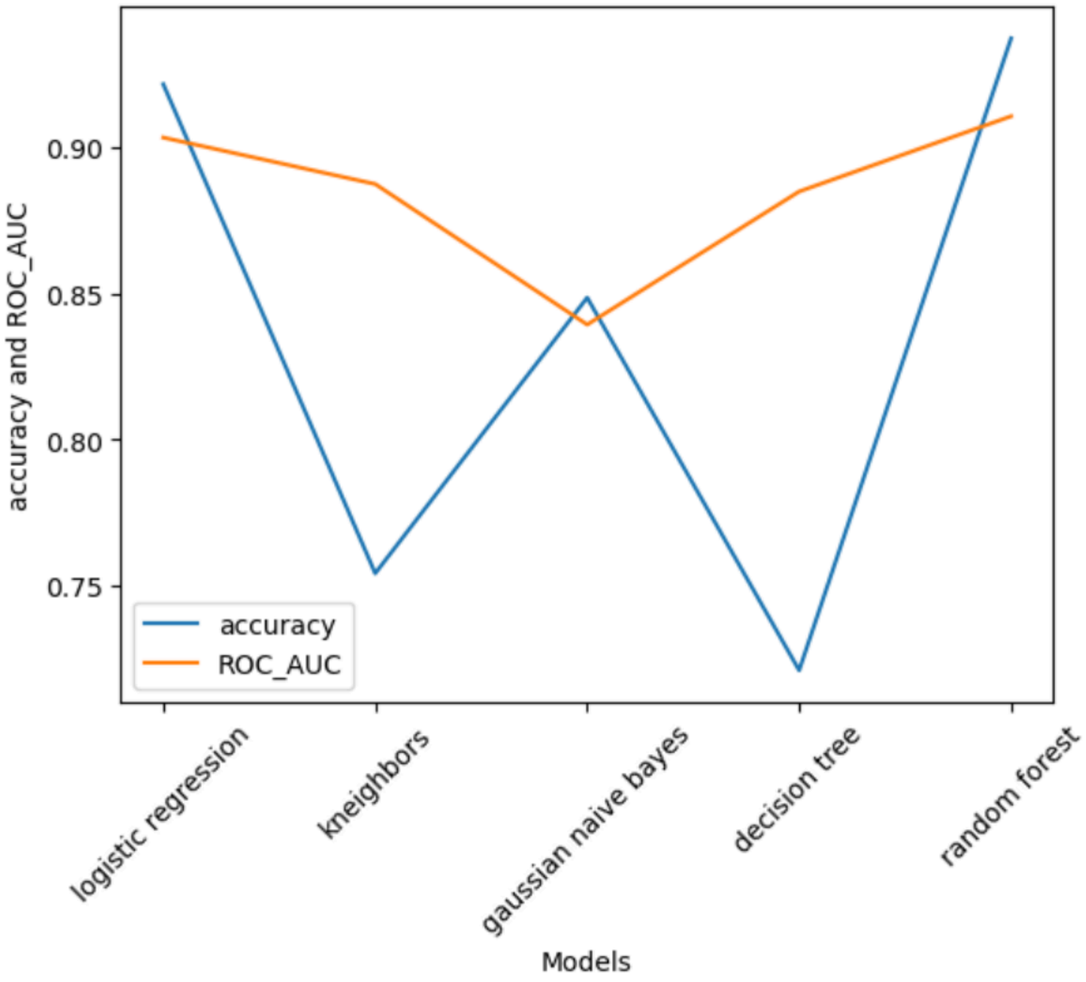
```
# predict test set:
y_pred_kn = dt.predict(X_test)
```

✓ 0.0s



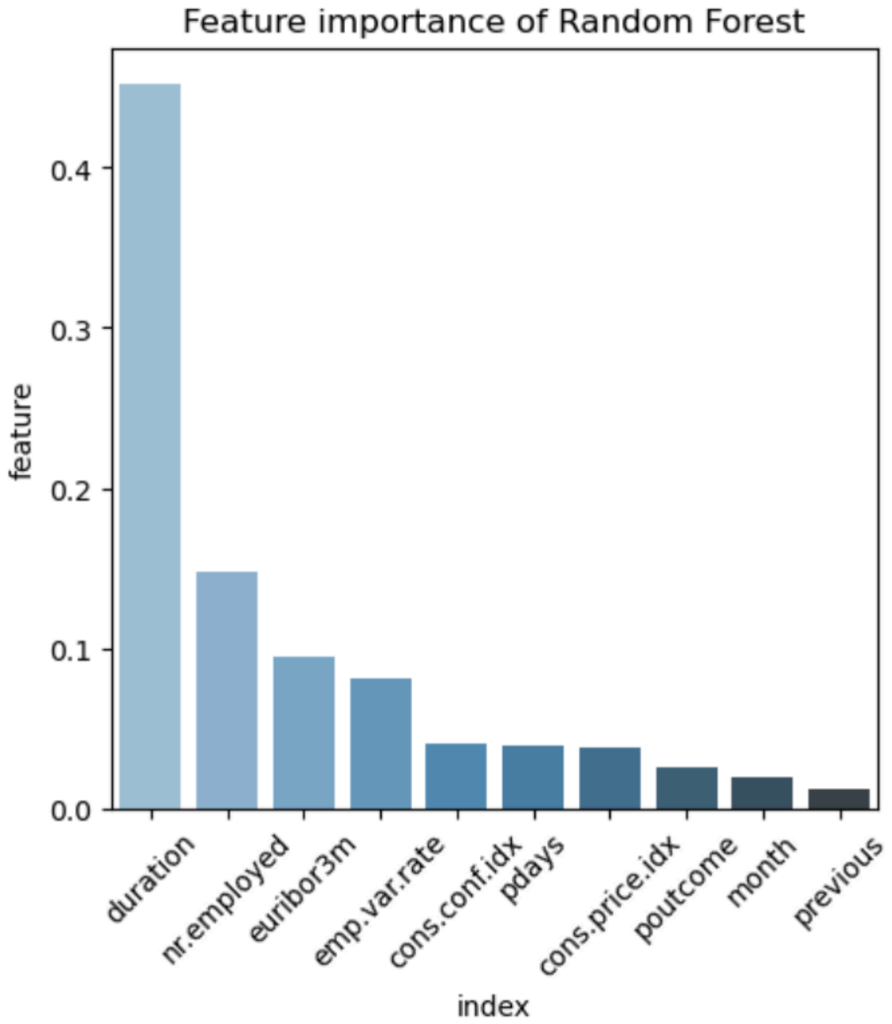

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.99 | 0.94 | 7265 |
| 1 | 0.62 | 0.11 | 0.19 | 971 |
| accuracy | | | 0.89 | 8236 |
| macro avg | 0.76 | 0.55 | 0.57 | 8236 |
| weighted avg | 0.86 | 0.89 | 0.85 | 8236 |

# COMPARE MODELS




It can be seen that the best model is Random forest classifier with the highest accuracy score of 91.05% and the highest AUC score of 93.71%.

# COMPARE THE FEATURES IMPORTANCE




By analyse the Random Forest classifier model, the variable with the highest influence on customer agreements is contact duration (duration), followed by contextual variables such as number of employees, interest rate, credit ratio. employment, consumer confidence, consumer price index.

The influence of the last contacts such as the number of days since the last contact, the results of the previous campaign, the last month of contact, and the number of contacts before the campaign also have a certain impact on customer decisions

# PART 5: RECOMMENDATIONS

### Spending more time for customers

Increasing customer consulting time will increase the likelihood of customer agreements

### Right time’s choice

May is the peak month with a large number of customers

### Regard to the contextual factors

Be aware of the influence of macroeconomic factors every time startong a new campaign

### Increase the quality of services through campaigns

Enhance the quality of consulting and customer care through campaigns to have better results for that future campaigns

Conducted by Cao Duc Thanh

# THANKS FOR WATCHING