MARKETING CAMPAIGN

The dataset comes from the UCI Machine Learning repository (File : banking.csv), this db contains records related to a marketing campaign performed by phone calls of a Portuguese Bank.

The goal of this work is to predict whether the client will subscribe (1/0) to a term deposit :
Target Variable Y (No Subscription/Subscription), with a focus on finding the best predictive model.

The dataset provides the bank customers' information. It includes 41,188 rows and 21 columns.

```
(41188, 21)
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous', 'poutcome',
'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed',
'y']
```

```
With data of this kind:
```

```
print(data.shape)
print(list(data.columns))
print(data.dtypes)
```

```
(41188, 21)
['age', 'job', 'marital', 'education', 'd
age                int64
job               object
marital           object
education         object
default           object
housing           object
loan              object
contact           object
month             object
day_of_week       object
duration           int64
campaign           int64
pdays              int64
previous           int64
poutcome          object
emp_var_rate     float64
cons_price_idx   float64
cons_conf_idx    float64
euribor3m        float64
nr_employed      float64
y                  int64
dtype: object
```

```
[6] data.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp_var_rate	cons_p
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	210	1	999	0	nonexistent	1.4	
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	138	1	999	0	nonexistent	-0.1	
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	339	3	6	2	success	-1.7	
3	39	services	married	high.school	no	no	no	cellular	apr	fri	185	2	999	0	nonexistent	-1.8	
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	137	1	3	1	success	-2.9	

This is the Input Variables Description:

```
[[i, list(data[i].unique())] for i in categorical_variables]

[['job',
  ['blue-collar',
   'technician',
   'management',
   'services',
   'retired',
   'admin.',
   'housemaid',
   'unemployed',
   'entrepreneur',
   'self-employed',
   'unknown',
   'student']],
 ['marital', ['married', 'single', 'divorced', 'unknown']],
 ['education',
  ['basic.4y',
   'unknown',
   'university.degree',
   'high.school',
   'basic.9y',
   'professional.course',
   'basic.6y',
   'illiterate']],
 ['default', ['unknown', 'no', 'yes']],
 ['housing', ['yes', 'no', 'unknown']],
 ['loan', ['no', 'yes', 'unknown']],
 ['month',
  ['aug', 'nov', 'jun', 'apr', 'jul', 'may', 'oct', 'mar', 'sep', 'dec']],
 ['day_of_week', ['thu', 'fri', 'tue', 'mon', 'wed']],
 ['previous', [0, 2, 1, 3, 4, 5, 7, 6]],
 ['poutcome', ['nonexistent', 'success', 'failure']]]
```

1. **age** (numeric)
2. **job** : type of job (categorical: "admin", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown")
3. **marital** : marital status (categorical: "divorced", "married", "single", "unknown")
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**: loan? (categorical: "no", "yes", "unknown")
7. **loan**: personal loan? (categorical: "no", "yes", "unknown")
8. **contact**: (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). **Important note**: this feature highly affects the output target because if duration=0 then y='no', but the duration is not known before the call interview. After the call y is known. **So, this feature is not good for a predictive model and I'm not going to take it in account to build a model.**

12. **campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. **pdays**: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. **previous**: number of contacts performed before this campaign and for this client (numeric)
15. **poutcome**: outcome of the previous marketing campaign (categorical: "failure", "nonexistent", "success")
16. **emp.var.rate**: employment variation rate — (numeric)
17. **cons.price.idx**: consumer price index — (numeric)
18. **cons.conf.idx**: consumer confidence index — (numeric)
19. **euribor3m**: euribor 3-month rate — (numeric)
20. **nr.employed**: number of employees — (numeric)

TARGET VARIABLE

The variable we want to predict is Y :

0- No : the customer did not subscribe a term deposit
1- Yes: the customer subscribed a term deposit

```
binary_variables = list(data_uniques[data_uniques['Unique Values'] == 2].index)
binary_variables

['contact', 'y']
```

DATA CLEANING

The data set is overall a good one. Beside the "Y" variable as we'll see, data are consistent, no to be-worried skewed data or null values, I just want to **reduce** the education column categories for a better modelling. So, from :

```
[ ] data['education'].unique()

    array(['basic.4y', 'unknown', 'university.degree', 'high.school',
           'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
          dtype=object)
```

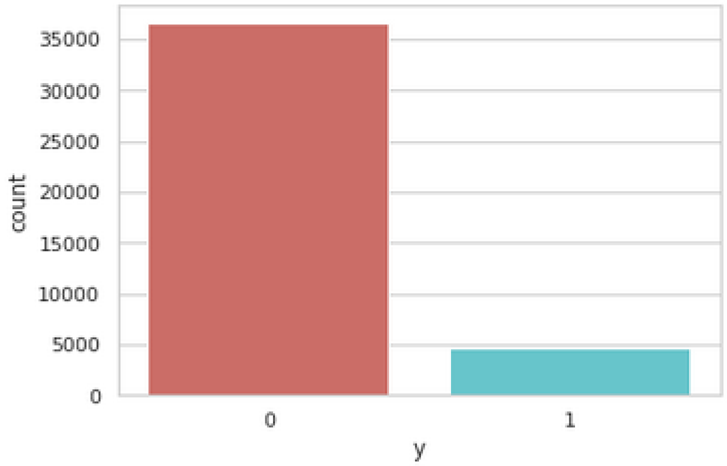
I will regroup the "basic.4y/6y/9y" categories into the "basic" category:

```
[ ] data['education'].unique()

    array(['Basic', 'unknown', 'university.degree', 'high.school',
           'professional.course', 'illiterate'], dtype=object)
```

DATA EXPLORATION

Let's take a look at the Target Variable :

```
[ ]  #Data Exploration
     data['y'].value_counts()

     0    36548
     1     4640
     Name: y, dtype: int64

[ ]  sns.countplot(x='y', data=data, palette= 'hls')
     plt.show()
     plt.savefig('count_plots')
```




```
<Figure size 432x288 with 0 Axes>
```

```
[ ]  #% Sub/No_Sub (target is unbalanced)
     count_no_sub = len(data[data['y']==0])
     count_sub = len(data[data['y']==1])
     pct_of_no_sub = count_no_sub/(count_no_sub+count_sub)
     print("percentage of no subscription is", pct_of_no_sub*100)
     pct_of_sub = count_sub/(count_no_sub+count_sub)
     print("percentage of subscription", pct_of_sub*100)
```

- percentage of no subscription is 88.73458288821988
- percentage of subscription 11.265417111780131

Our class is clearly **Unbalanced** with a ratio of 89:11. Before proceeding with a down\upsample, keep on with further exploration :

```
[ ] data.groupby('y').mean()
```

y	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed
0	39.911185	220.844807	2.633085	984.113878	0.132374	0.248875	93.603757	-40.593097	3.811491	5176.166600
1	40.913147	553.191164	2.051724	792.035560	0.492672	-1.233448	93.354386	-39.789784	2.123135	5095.115991

Relevant Observations:

- The **average age** of customers who subscribed for a term deposit is higher than that of not subscribers.
- The **pdays** (days since the customer was last contacted) is understandably lower for the customers who bought it. The lower the pdays, the better the memory of the last call and hence the better chances of a sale.
- Note that **campaigns** (number of contacts or calls made during the current campaign) are lower for customers who bought the term deposit.

We can calculate categorical means for other categorical variables such as **job, marital status** and **education** to get a better insight of our data.

```
[ ] data.groupby('job').mean()
```

job	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
admin.	38.187296	254.312128	2.623489	954.319229	0.189023	0.015563	93.534054	-40.245433	3.550274	5164.125350	0.129726
blue-collar	39.555760	264.542360	2.558461	985.160363	0.122542	0.248995	93.656656	-41.375816	3.771996	5175.615150	0.068943
entrepreneur	41.723214	263.267857	2.535714	981.267170	0.138736	0.158723	93.605372	-41.283654	3.791120	5176.313530	0.085165
housemaid	45.500000	250.454717	2.639623	960.579245	0.137736	0.433396	93.676576	-39.495283	4.009645	5179.529623	0.100000
management	42.362859	257.058140	2.476060	962.647059	0.185021	-0.012688	93.522755	-40.489466	3.611316	5166.650513	0.112175
retired	62.027326	273.712209	2.476744	897.936047	0.327326	-0.698314	93.430786	-38.573081	2.770066	5122.262151	0.252326
self-employed	39.949331	264.142153	2.660802	976.621393	0.143561	0.094159	93.559982	-40.488107	3.689376	5170.674384	0.104856
services	37.926430	258.398085	2.587805	979.974049	0.154951	0.175359	93.634659	-41.290048	3.699187	5171.600126	0.081381
student	25.894857	283.683429	2.104000	840.217143	0.524571	-1.408000	93.331613	-40.187543	1.884224	5085.939086	0.314286
technician	38.507638	250.232241	2.577339	964.408127	0.153789	0.274566	93.561471	-39.927569	3.820401	5175.648391	0.108260
unemployed	39.733728	249.451677	2.564103	935.316568	0.199211	-0.111736	93.563781	-40.007594	3.466583	5157.156509	0.142012
unknown	45.563636	239.675758	2.648485	938.727273	0.154545	0.357879	93.718942	-38.797879	3.949033	5172.931818	0.112121

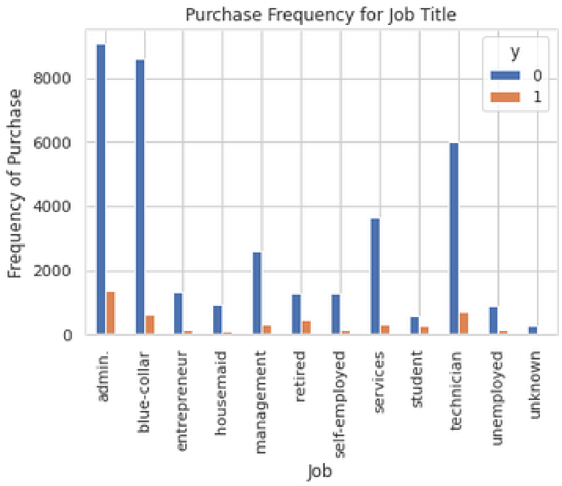
```
[ ] data.groupby('marital').mean()
```

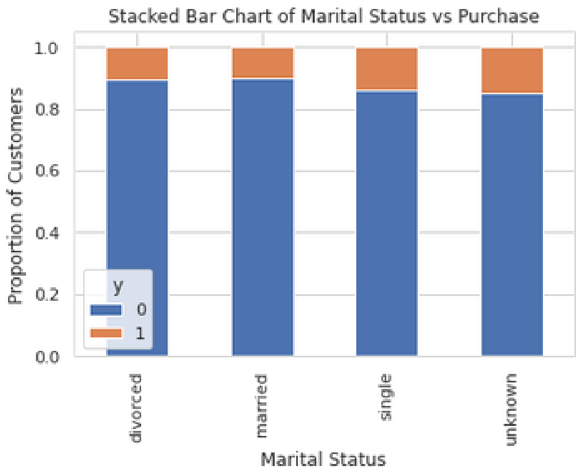
marital	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
divorced	44.899393	253.790330	2.61340	968.639853	0.168690	0.163985	93.606563	-40.707069	3.715603	5170.878643	0.103209
married	42.307165	257.438623	2.57281	967.247673	0.155608	0.183625	93.597367	-40.270659	3.745832	5171.848772	0.101573
single	33.158714	261.524378	2.53380	949.909578	0.211359	-0.167989	93.517300	-40.918698	3.317447	5155.199265	0.140041
unknown	40.275000	312.725000	3.18750	937.100000	0.275000	-0.221250	93.471250	-40.820000	3.313038	5157.393750	0.150000

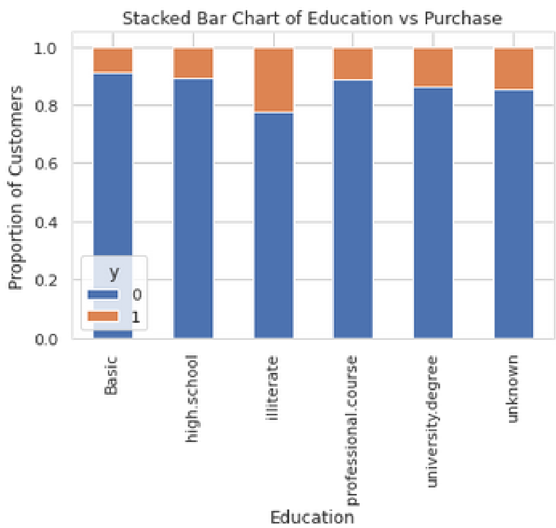
```
[ ] data.groupby('education').mean()
```

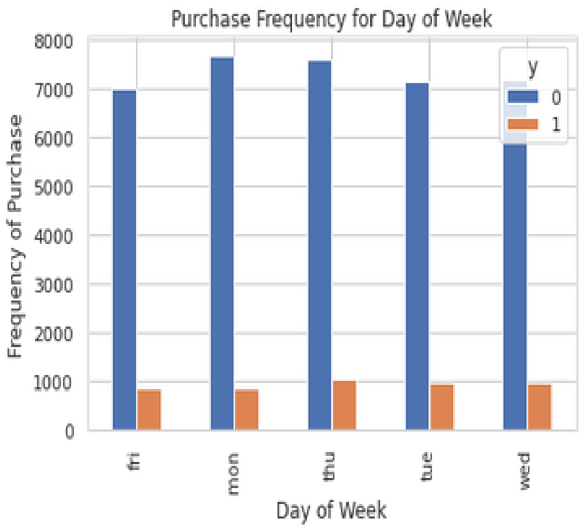
education	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
Basic	42.163910	263.043874	2.559498	974.877967	0.141053	0.191329	93.639933	-40.927595	3.729654	5172.014113	0.087029
high.school	37.998213	260.886810	2.568576	964.358382	0.185917	0.032937	93.584857	-40.940641	3.556157	5164.994735	0.108355
illiterate	48.500000	276.777778	2.277778	943.833333	0.111111	-0.133333	93.317333	-39.950000	3.516556	5171.777778	0.222222
professional.course	40.080107	252.533855	2.586115	960.765974	0.163075	0.173012	93.569864	-40.124108	3.710457	5170.155979	0.113485
university.degree	38.879191	253.223373	2.563527	951.807692	0.192390	-0.028090	93.493466	-39.975805	3.529663	5163.226298	0.137245
unknown	43.481225	262.390526	2.596187	942.830734	0.226459	0.059099	93.658615	-39.877816	3.571098	5159.549509	0.145003

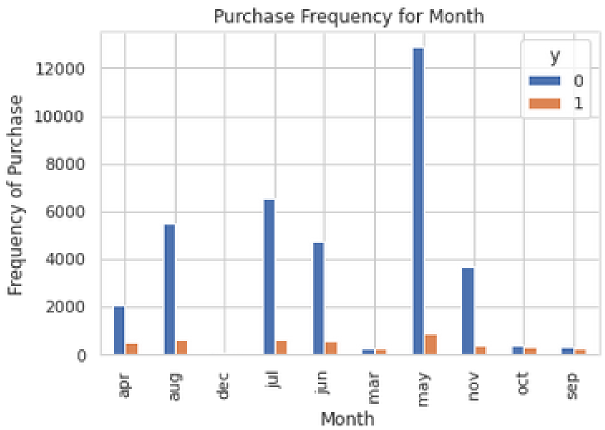
and the plots:

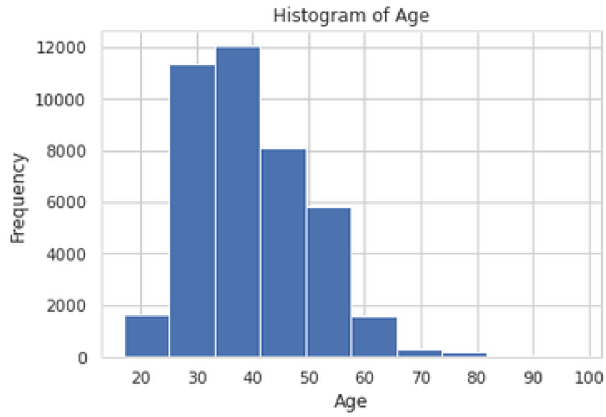


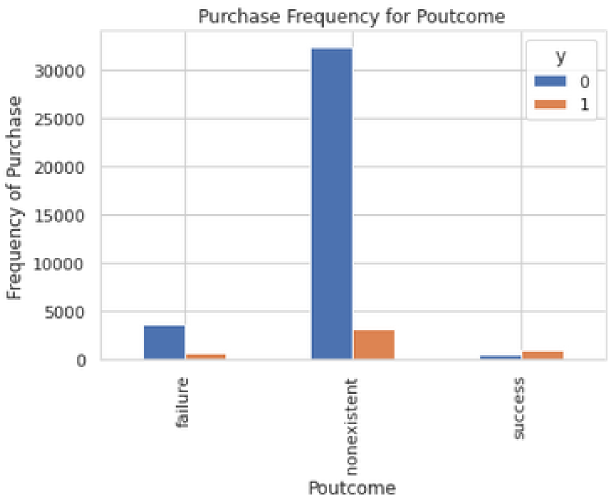












From the Plots we can find the **following insights** :

1) The **job title** can be a good predictor of the outcome variable, since the frequency depends a lot on this feature.
2) The **marital status** seems not influent for the outcome variable.
3) **Education** seems a good predictor of the outcome variable.
4) **Day of week** doesn't seem a good predictor of the outcome.
5) **Month** might be a good predictor of the outcome variable.
6) Most of the customers of the bank in this dataset are in the age range of **30–40**.
7) **Poutcome** (outcome of the previous marketing campaign) seems to be a good predictor of the outcome variable.

Then I will create the Dummy variables and get the final data columns:

```
data_final=data[to_keep]
data_final.columns.values

array(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp_var_rate',
       'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'y',
       'job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student',
       'job_technician', 'job_unemployed', 'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single',
       'marital_unknown', 'education_Basic', 'education_high.school',
       'education_illiterate', 'education_professional.course',
       'education_university.degree', 'education_unknown', 'default_no',
       'default_unknown', 'default_yes', 'housing_no', 'housing_unknown',
       'housing_yes', 'loan_no', 'loan_unknown', 'loan_yes',
       'contact_cellular', 'contact_telephone', 'month_apr', 'month_aug',
       'month_dec', 'month_jul', 'month_jun', 'month_mar', 'month_may',
       'month_nov', 'month_oct', 'month_sep', 'day_of_week_fri',
       'day_of_week_mon', 'day_of_week_thu', 'day_of_week_tue',
       'day_of_week_wed', 'poutcome_failure', 'poutcome_nonexistent',
       'poutcome_success'], dtype=object)
```

Now our dataset is ready to be splitted in training and test set and after that I'll UPSAMPLE with SMOTE method the "Subscription\No Subscription" Target Variable "Y", getting :

- `Number of no subscription in oversampled data 25567`
- `Number of subscription 25567`
- `Proportion of no subscription data in oversampled data is  0.5`
- `Proportion of subscription data in oversampled data is  0.5`

Now we have a perfect balanced class.
I'm going to choose only the most significative Features for sake of simplicity and computation:

```
[ ]  cols=['euribor3m', 'job_blue-collar', 'job_housemaid', 'marital_unknown', 'education_illiterate',
           'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun', 'month_mar',
           'month_may', 'month_nov', 'month_oct', "poutcome_failure", "poutcome_success"]
     X=os_data_X[cols]
     y=os_data_y['y']
```

FITTING THE MODELS

We will use the same training and test splits for all the models.

1) We will first train a **simple Logistic Regression**

2) Then we will fit **gradient boosted tree model** with the following tree numbers (n_estimators = [15, 25, 50, 100, 200, 400]) and evaluate the accuracy on the test data for each of these models and confusion matrix

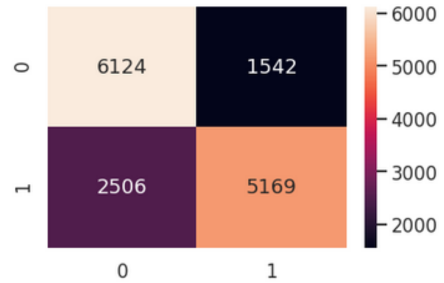
3) Then using a grid search with cross-validation, fit a **new gradient boosted classifie**r with the same list of estimators as in the previous model and I will vary the learning rates (0.1, 0.01, 0.001), the subsampling value (1.0 or 0.5), and the number of maximum features, evaluate the accuracy and the confusion matrix

4) Create an **AdaBoost model** and fit it using grid search, with a range of estimators between 100 and 200, evaluate the accuracy and confusion matrix

5) Using **VotingClassifier**, fit the logistic regression model along with the GratientBoostedClassifier model, again evaluate the accuracy and the confusion matrix.

Logistic Regression

```
[ ] sns.set_context('talk')
    cm = confusion_matrix(y_test, y_pred)
    ax = sns.heatmap(cm, annot=True, fmt='d')
```



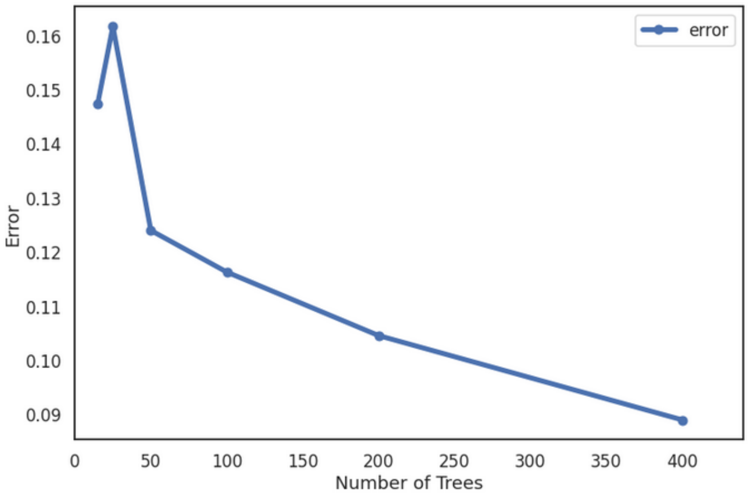

```
[ ] y_pred = LR_L2.predict(X_test)
    print(classification_report(y_pred, y_test))

              precision    recall  f1-score   support

           0       0.80      0.71      0.75      8630
           1       0.67      0.77      0.72      6711

    accuracy                           0.74     15341
   macro avg       0.74      0.74      0.74     15341
weighted avg       0.74      0.74      0.74     15341
```

The result is telling us that we have ***6124+5169*** correct predictions and ***2506+1542*** incorrect predictions.

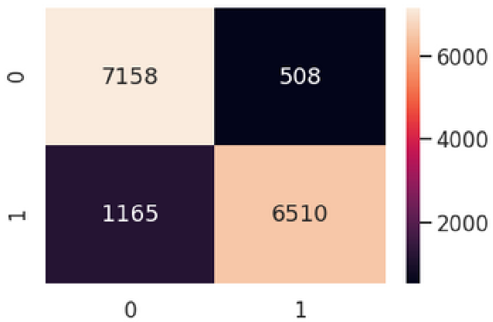
Gradient Boosting

```
Fitting model with 15 trees
Fitting model with 25 trees
Fitting model with 50 trees
Fitting model with 100 trees
Fitting model with 200 trees
Fitting model with 400 trees
```

n_trees	error
15.0	0.147318
25.0	0.161593
50.0	0.123916
100.0	0.116224
200.0	0.104491
400.0	0.088912



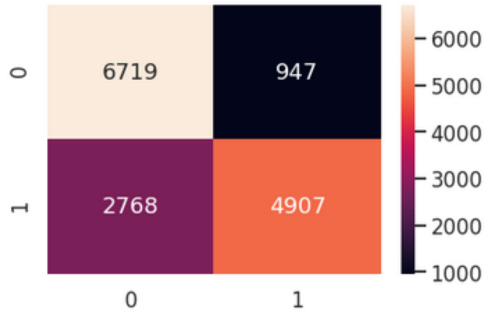

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90      8323
           1       0.85      0.93      0.89      7018

    accuracy                           0.89     15341
   macro avg       0.89      0.89      0.89     15341
weighted avg       0.89      0.89      0.89     15341
```



The result is telling us that we have ***7158+6510*** correct predictions and ***1165+508*** incorrect predictions.

AdaBooster

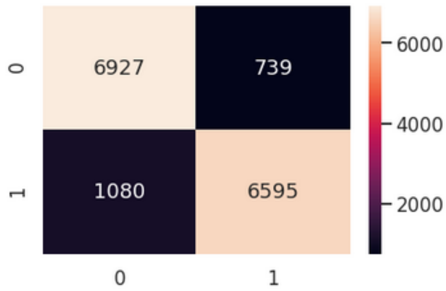
	precision	recall	f1-score	support
0	0.88	0.71	0.78	9487
1	0.64	0.84	0.73	5854
accuracy			0.76	15341
macro avg	0.76	0.77	0.75	15341
weighted avg	0.79	0.76	0.76	15341



The result is telling us that we have ***6719+4907*** correct predictions and ***2768+947*** incorrect predictions.

Voting Classifier

	precision	recall	f1-score	support
0	0.87	0.90	0.88	7666
1	0.90	0.86	0.88	7675
accuracy			0.88	15341
macro avg	0.88	0.88	0.88	15341
weighted avg	0.88	0.88	0.88	15341



The result is telling us that we have ***6927+6595*** correct predictions and ***1080+739*** incorrect predictions.

CONCLUSIONS

Best Classifier is the one with Gradient Boosting, with n_estimators=400 , learning_rate=0.1 max_features=4 and subsample=0.5.

The matrix confusion told us that we have ***7158+6510*** correct predictions and ***1165+508*** incorrect predictions.

With an accuracy of 89% and a macro average of 89%

It did much better than Logistic Regression with a macro average of 74%, maybe because it used a simpler model (less features used)

VotingClassifier performance didn't improve a lot the overall performance.

AdaBoost, in term of performance is very similar to Logistic Regression.

Suggestions for next steps

for lack of computational power, I couldn't test GradientBoosting with more than 400 estimators, it took 2 hours on my PC. As you can see, we didn't reach the plateau error. It reasonable to forecast greater performance improvement around 500-600 estimators, making the GradientBoosting model the best suited for this kind of Dataset.

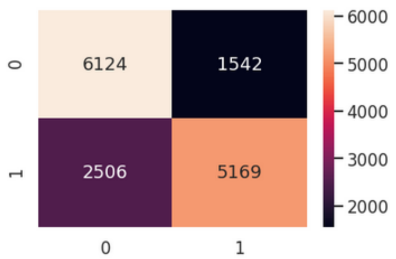
LR

```
[ ] y_pred = LR_L2.predict(X_test)
    print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.80      0.71      0.75      8630
           1       0.67      0.77      0.72      6711

    accuracy                           0.74     15341
   macro avg       0.74      0.74      0.74     15341
weighted avg       0.74      0.74      0.74     15341
```

CM LR

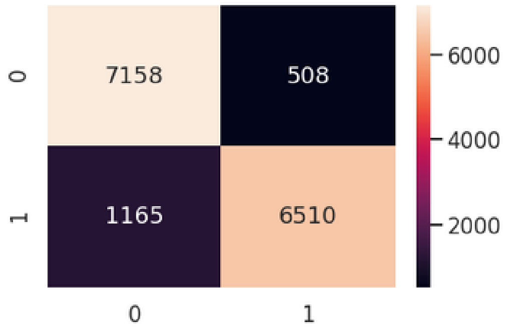
```
[ ] sns.set_context('talk')
    cm = confusion_matrix(y_test, y_pred)
    ax = sns.heatmap(cm, annot=True, fmt='d')
```



GB

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90      8323
           1       0.85      0.93      0.89      7018

    accuracy                           0.89     15341
   macro avg       0.89      0.89      0.89     15341
weighted avg       0.89      0.89      0.89     15341
```

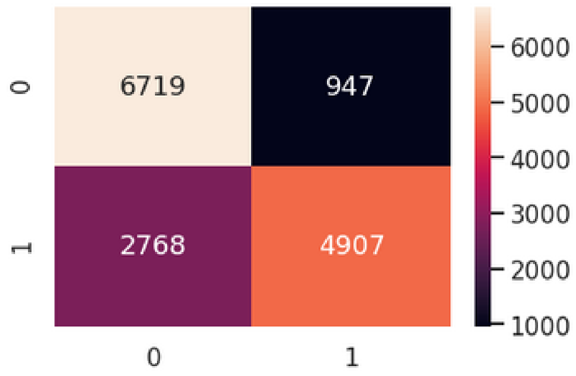
CM GB



AB

	precision	recall	f1-score	support
0	0.88	0.71	0.78	9487
1	0.64	0.84	0.73	5854
accuracy			0.76	15341
macro avg	0.76	0.77	0.75	15341
weighted avg	0.79	0.76	0.76	15341

CM AB



VC

	precision	recall	f1-score	support
0	0.87	0.90	0.88	7666
1	0.90	0.86	0.88	7675
accuracy			0.88	15341
macro avg	0.88	0.88	0.88	15341
weighted avg	0.88	0.88	0.88	15341

CM VC