PYTHON FOR DATA ANALYSIS

By Léo CICAL – DIA2 – ESILV A4

THE PROJECT

- The goal of this project is to predict if an online customer will buy sometyhing on the website or not.
- The result is a Flask API which can predict, when you entered the different infos about the customer, if the customer is going to buy something.
- The different steps used are Data Preproccesing, Data Vizualisation, Machine Learning and the creation of a Flask API.

- Source : https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset

THE DATASET

The **Online Shoppers Purchasing Intention Dataset :**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
2	0	0	0	0	1	0	0.2	0.2	0	0	Feb	1	1	1	1	Returning_Visitor	FALSE	FALSE
3	0	0	0	0	2	64	0	0.1	0	0	Feb	2	2	1	2	Returning_Visitor	FALSE	FALSE
4	0	0	0	0	1	0	0.2	0.2	0	0	Feb	4	1	9	3	Returning_Visitor	FALSE	FALSE
5	0	0	0	0	2	2.666666667	0.05	0.14	0	0	Feb	3	2	2	4	Returning_Visitor	FALSE	FALSE

The dataset consists of 10 numerical and 8 categorical attributes.
The 'Revenue' attribute can be used as the class label.

"Administrative", "Administrative Duration", "Informational", "Informational Duration", "Product Related" and "Product Related Duration" represent the number of different types of pages visited by the visitor in that session and total time spent in each of these page categories. The values of these features are derived from the URL information of the pages visited by the user and updated in real time when a user takes an action, e.g. moving from one page to another.
The **"Bounce Rate", "Exit Rate" and "Page Value"** features represent the metrics measured by "Google Analytics" for each page in the e-commerce site.
The value of **"Bounce Rate"** feature for a web page refers to the percentage of visitors who enter the site from that page and then leave ("bounce") without triggering any other requests to the analytics server during that session.
The value of **"Exit Rate"** feature for a specific web page is calculated as for all pageviews to the page, the percentage that were the last in the session.
The **"Page Value"** feature represents the average value for a web page that a user visited before completing an e-commerce transaction.
The **"Special Day"** feature indicates the closeness of the site visiting time to a specific special day (e.g. Mother's Day, Valentine's Day) in which the sessions are more likely to be finalized with transaction. The value of this attribute is determined by considering the dynamics of e-commerce such as the duration between the order date and delivery date. For example, for Valentina's day, this value takes a nonzero value between February 2 and February 12, zero before and after this date unless it is close to another special day, and its maximum value of 1 on February 8.
The dataset also includes **operating system, browser, region, traffic type, visitor type as returning or new visitor,** a Boolean value indicating whether the date of the visit is weekend, and month of the year.

First of all, we import it :

```
data_shop = pd.read_csv("online_shoppers_intention.csv",sep=",")
data_shop
```

Then, we look for some interesting infos about it :

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

	count	mean	std	min	25%	50%	75%	max
Administrative	12330.0	2.315166	3.321784	0.0	0.000000	1.000000	4.000000	27.000000
Administrative_Duration	12330.0	80.818611	176.779107	0.0	0.000000	7.500000	93.256250	3398.750000
Informational	12330.0	0.503569	1.270156	0.0	0.000000	0.000000	0.000000	24.000000
Informational_Duration	12330.0	34.472398	140.749294	0.0	0.000000	0.000000	0.000000	2549.375000
ProductRelated	12330.0	31.731468	44.475503	0.0	7.000000	18.000000	38.000000	705.000000
ProductRelated_Duration	12330.0	1194.746220	1913.669288	0.0	184.137500	598.936905	1464.157213	63973.522230
BounceRates	12330.0	0.022191	0.048488	0.0	0.000000	0.003112	0.016813	0.200000
ExitRates	12330.0	0.043073	0.048597	0.0	0.014286	0.025156	0.050000	0.200000
PageValues	12330.0	5.889258	18.568437	0.0	0.000000	0.000000	0.000000	361.763742
SpecialDay	12330.0	0.061427	0.198917	0.0	0.000000	0.000000	0.000000	1.000000
OperatingSystems	12330.0	2.124006	0.911325	1.0	2.000000	2.000000	3.000000	8.000000
Browser	12330.0	2.357097	1.717277	1.0	2.000000	2.000000	2.000000	13.000000
Region	12330.0	3.147364	2.401591	1.0	1.000000	3.000000	4.000000	9.000000
TrafficType	12330.0	4.069586	4.025169	1.0	2.000000	2.000000	4.000000	20.000000

The target is 'Revenue'. We have 12330 rows, and 18 colmuns.

We have **10** numerical attributes :

- Administrative - Administrative_Duration - Informational - Informational_Duration - ProductRelated - ProductRelated_Duration - BounceRates - ExitRates - PageValues - SpecialDay

We have **8** categorical attributes :

- Month - OperatingSystems - Browser - Region - TrafficType - VisitorType - Weekend - Revenue

DATA PREPROCESSING

```
Entrée [221]: missing_val_count_by_column = (data_shop.isnull().sum())
              print(missing_val_count_by_column[missing_val_count_by_column > 0])

Series([], dtype: int64)
```

There isn't any Missing Values in this Data Set.

We just need to change the bool type of 'Revenue' and 'Weekend' to Int, for certains plots and for the ML part.

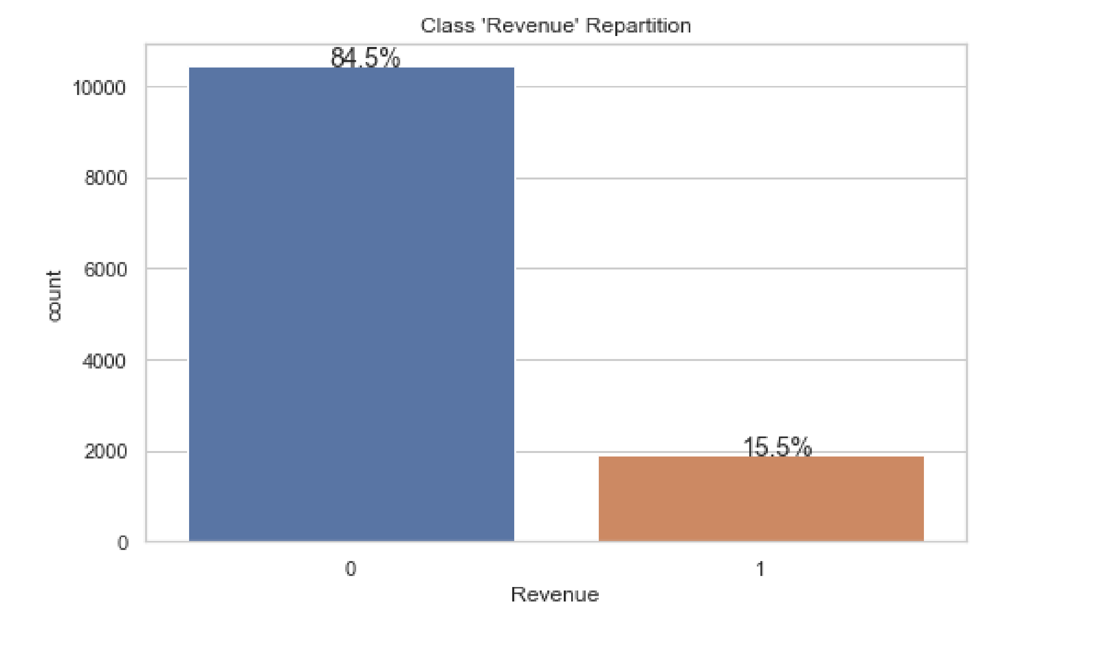
```
data_shop['Revenue'] = data_shop['Revenue']*1
data_shop['Weekend'] = data_shop['Weekend']*1
data_shop
```

For the plots,we also needed to One-Hot Encode Month and Visitor Type, because they we're 'Object' so they weren't include into the correlation matrix.
(The preprocessing for the ML will be explained later).

```
Entrée [236]: data_shop_final = pd.get_dummies(data_shop,columns=['Month','VisitorType'])
              data_shop_final.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 29 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Administrative                 12330 non-null  int64
 1   Administrative_Duration        12330 non-null  float64
 2   Informational                  12330 non-null  int64
 3   Informational_Duration         12330 non-null  float64
 4   ProductRelated                 12330 non-null  int64
 5   ProductRelated_Duration        12330 non-null  float64
 6   BounceRates                    12330 non-null  float64
 7   ExitRates                      12330 non-null  float64
 8   PageValues                     12330 non-null  float64
 9   SpecialDay                     12330 non-null  float64
 10  OperatingSystems               12330 non-null  int64
 11  Browser                        12330 non-null  int64
 12  Region                         12330 non-null  int64
 13  TrafficType                    12330 non-null  int64
 14  Weekend                        12330 non-null  int32
 15  Revenue                        12330 non-null  int32
 16  Month_Aug                      12330 non-null  uint8
 17  Month_Dec                      12330 non-null  uint8
 18  Month_Feb                      12330 non-null  uint8
 19  Month_Jul                      12330 non-null  uint8
 20  Month_June                     12330 non-null  uint8
 21  Month_Mar                      12330 non-null  uint8
 22  Month_May                      12330 non-null  uint8
 23  Month_Nov                      12330 non-null  uint8
 24  Month_Oct                      12330 non-null  uint8
 25  Month_Sep                      12330 non-null  uint8
 26  VisitorType_New_Visitor        12330 non-null  uint8
 27  VisitorType_Other              12330 non-null  uint8
 28  VisitorType_Returning_Visitor  12330 non-null  uint8
dtypes: float64(7), int32(2), int64(7), uint8(13)
memory usage: 1.6 MB
```

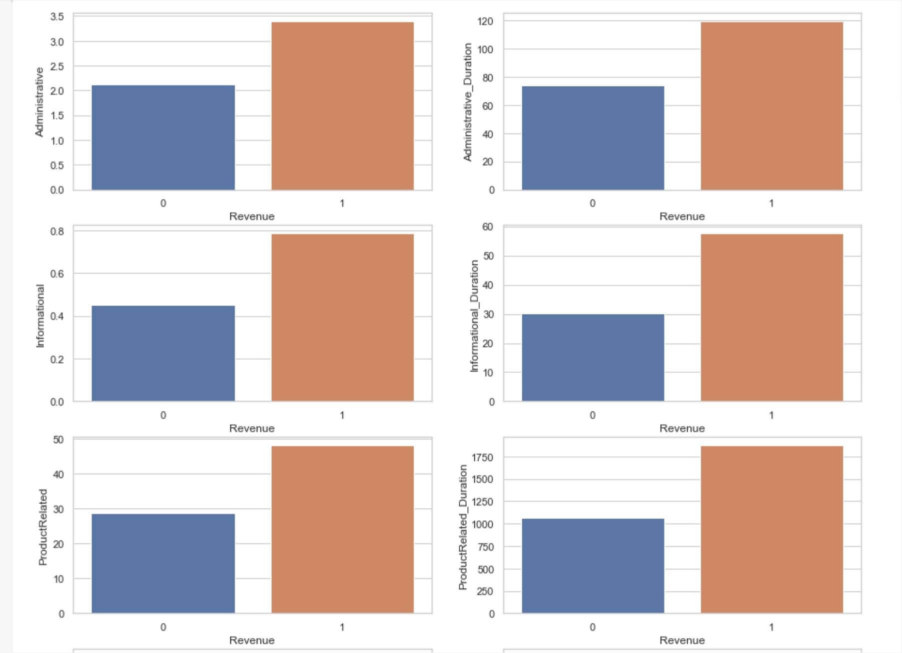
DATA PLOTTING

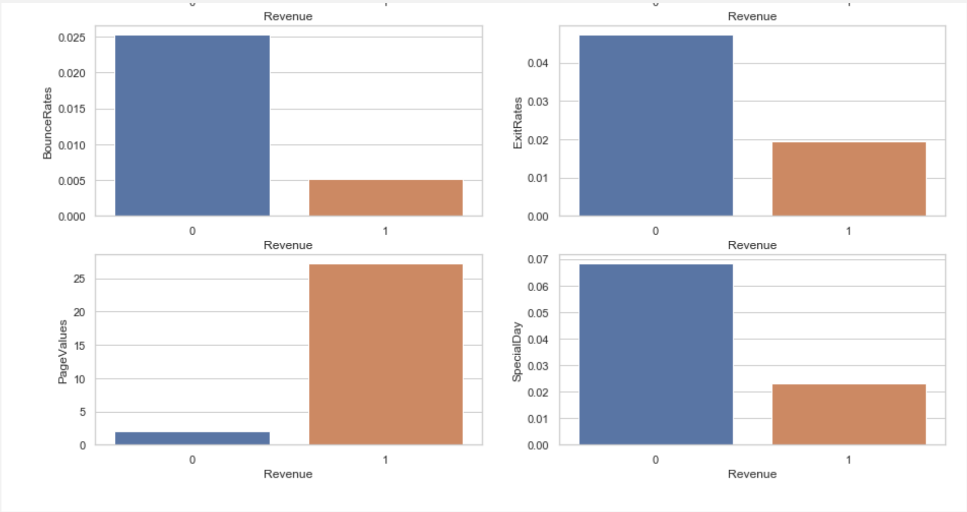
1) CLASS 'REVENUE' REPARTITION (REPARTITION OF THE TARGET VALUES)



We see that in our dataset, 84,5% of the time, the customer didnt't buy something (0 in the column Revenue)

2) MEAN OF THE NUMERICAL ATTRIBUTES DEPENDING ON THE TARGET 'REVENUE

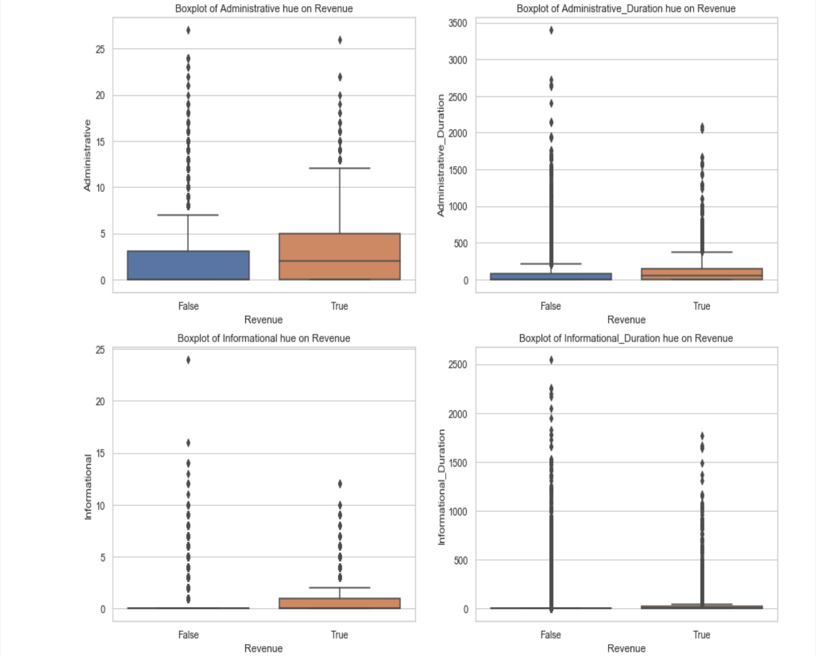


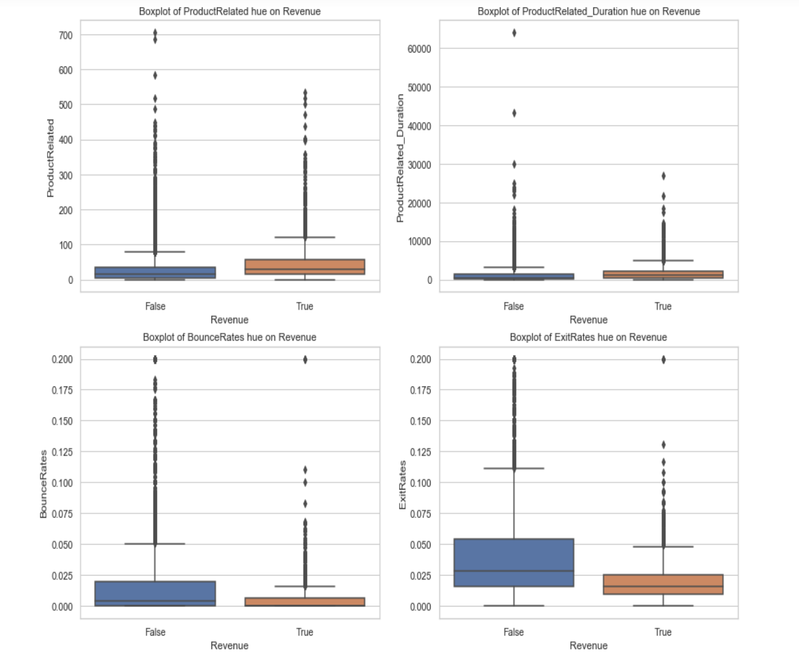


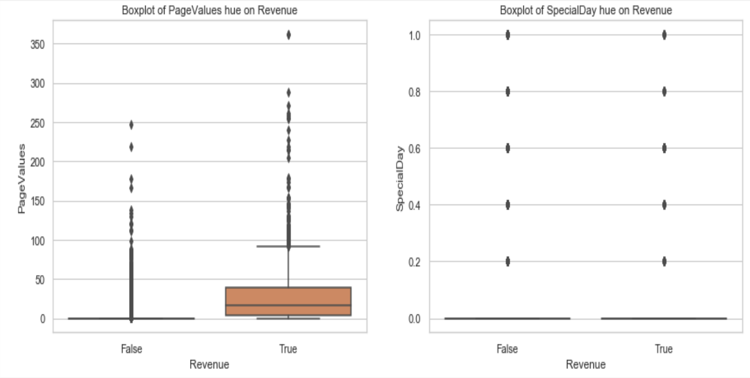
We these plots, we can assume certains things :

- when a customer buy, his **admistriation, informational and product_related are higher.**
- when the **bounces rates and the exit rates are higher**, a customere will probably don't buy.
- when the page values is higher, a customer will propably buy.

3) BOXPLOTS





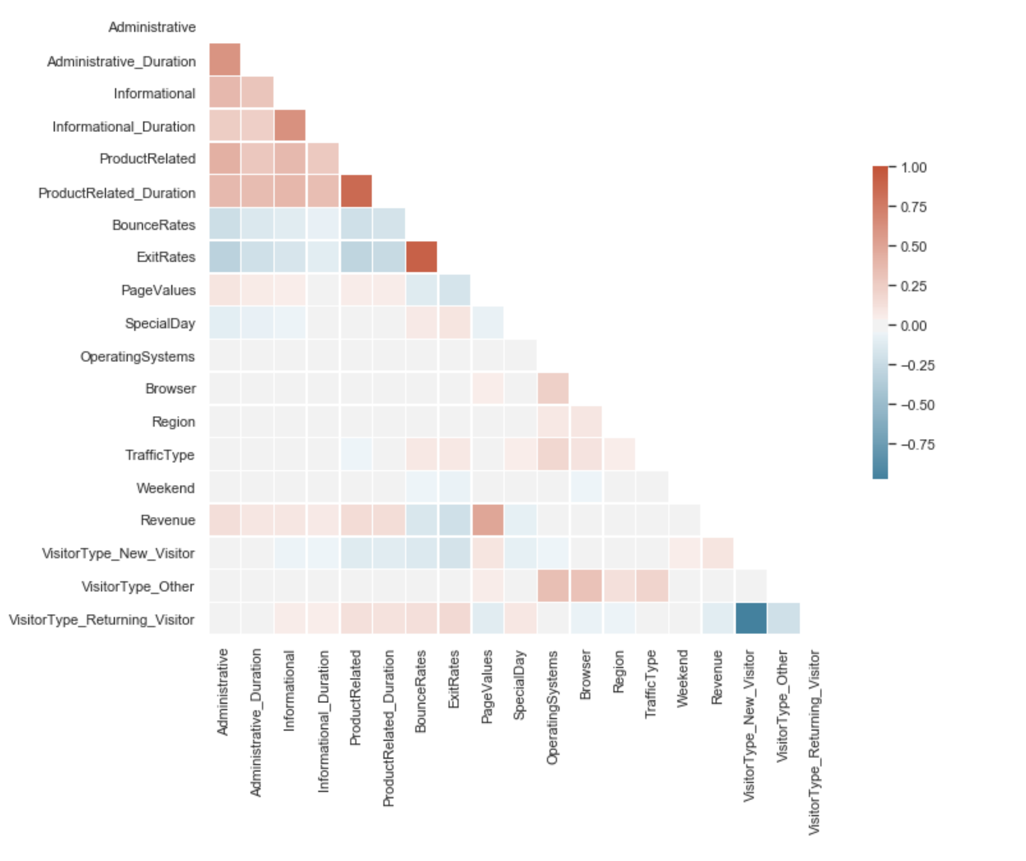


We these boxplots, we can assume certains things :
- the scale are not the same for a lot of attributes, so we're **gonna scaled our dataset.**
- there are a lot of **outliers** in each boxplots.
- the boxplots **confirm what we saw** with precedents plots !

4) CORRELATION MATRIX BETWEEN ALL THE ATTRIBUTES

We're gonna show the correlation between all the attributes (except Month, because there too many months possible, it wasn't very useful to show, so we drop the columns)

```
col_month = ['Month_Feb','Month_Mar','Month_May','Month_June','Month_Jul','Month_Aug','Month_Sep','Month_Oct','Month_Nov',
data_shop_without_month= data_shop_final.drop(columns=col_month)
corr=data_shop_without_month.corr(method='pearson')
corr
```



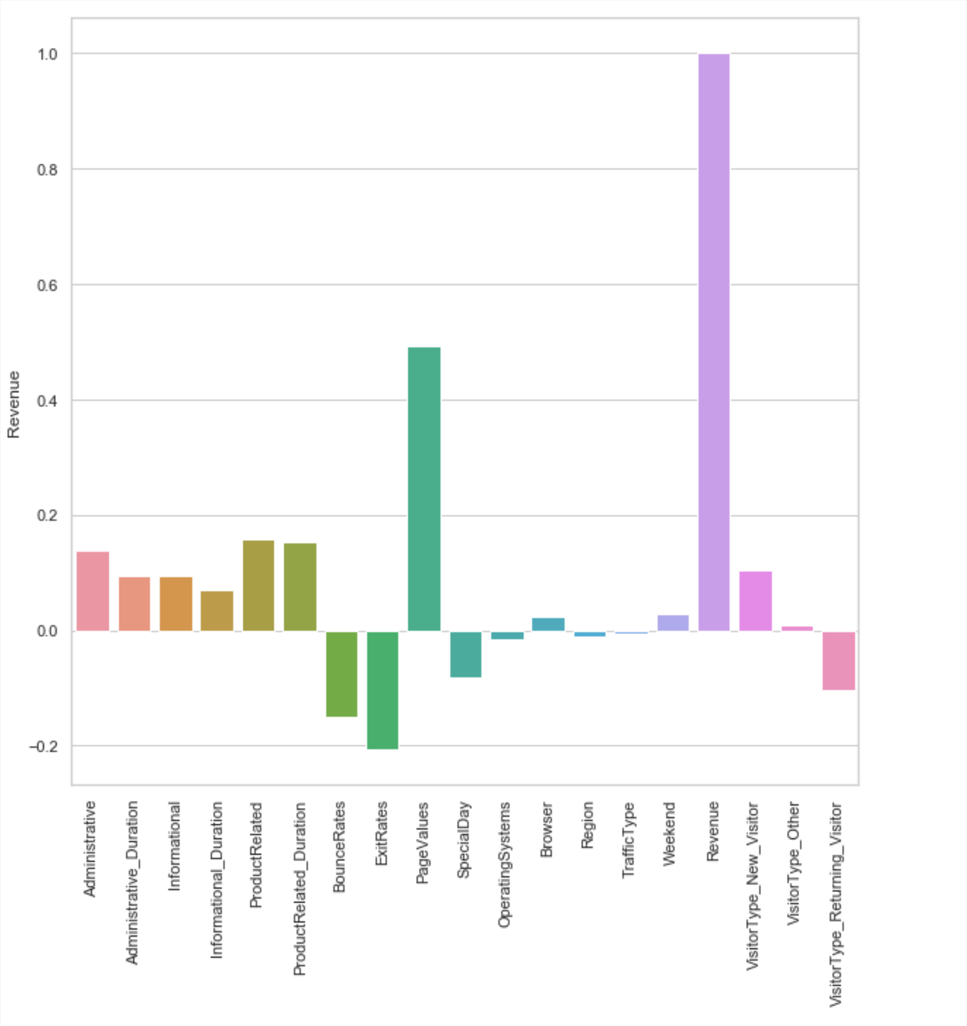
This correlation heatmap is easy to understand : the more the color is **red**, the more the variables are **positively correlated** ; the more the color is **blue**, the more the variables are **negatively correlated**.

We can see that **ProducedRelated_Duration and ProducedRelated** are very positively correlated**; ExitRates and BounceRates too.**

5) CORRELATION BETWEEN THE TARGET '"REVENUE' AND THE OTHER ATTRIBUTES (EXCEPT MONTH)

Now, we're gonna look the link between Revenue (the target) and all the other attributes.

```
Entrée [239]: print(corr.Revenue.sort_values(ascending=False))

Revenue                            1.000000
PageValues                         0.492569
ProductRelated                     0.158538
ProductRelated_Duration            0.152373
Administrative                     0.138917
VisitorType_New_Visitor            0.104136
Informational                      0.095200
Administrative_Duration            0.093587
Informational_Duration             0.070345
Weekend                            0.029295
Browser                            0.023984
VisitorType_Other                  0.007715
TrafficType                       -0.005113
Region                            -0.011595
OperatingSystems                  -0.014668
SpecialDay                        -0.082305
VisitorType_Returning_Visitor     -0.103843
BounceRates                       -0.150673
ExitRates                         -0.207071
Name: Revenue, dtype: float64
```



It seems that Revenue is more correlated with **Pages Values and ProducedRelated.**

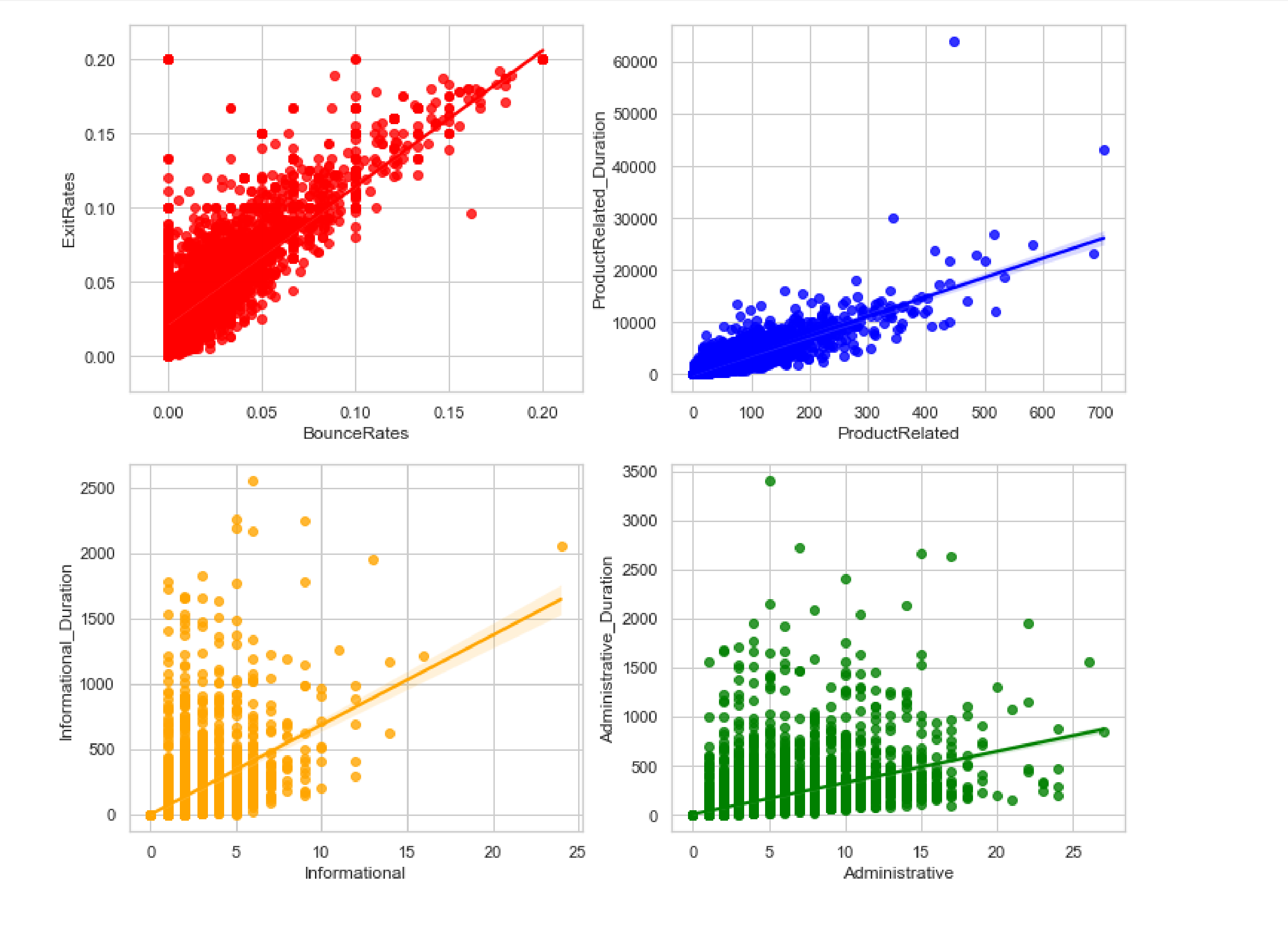
6) BIGGEST CORRELATIONS BETWEEN PAIRS OF ATTRIBUTES

Entrée [241]:

```
attrs = corr.iloc[:-1,:-1] # all except target
# only important correlations and not auto-correlations
threshold = 0.5
important_corrs = (attrs[abs(attrs) > threshold][attrs != 1.0]) \
    .unstack().dropna().to_dict()

unique_important_corrs = pd.DataFrame(
    list(set([(tuple(sorted(key)), important_corrs[key]) \
    for key in important_corrs])), columns=['attribute pair', 'correlation'])
# sorted by absolute value
unique_important_corrs = unique_important_corrs.loc[
    abs(unique_important_corrs['correlation']).argsort()[::-1]]
unique_important_corrs
```

Out[241]:

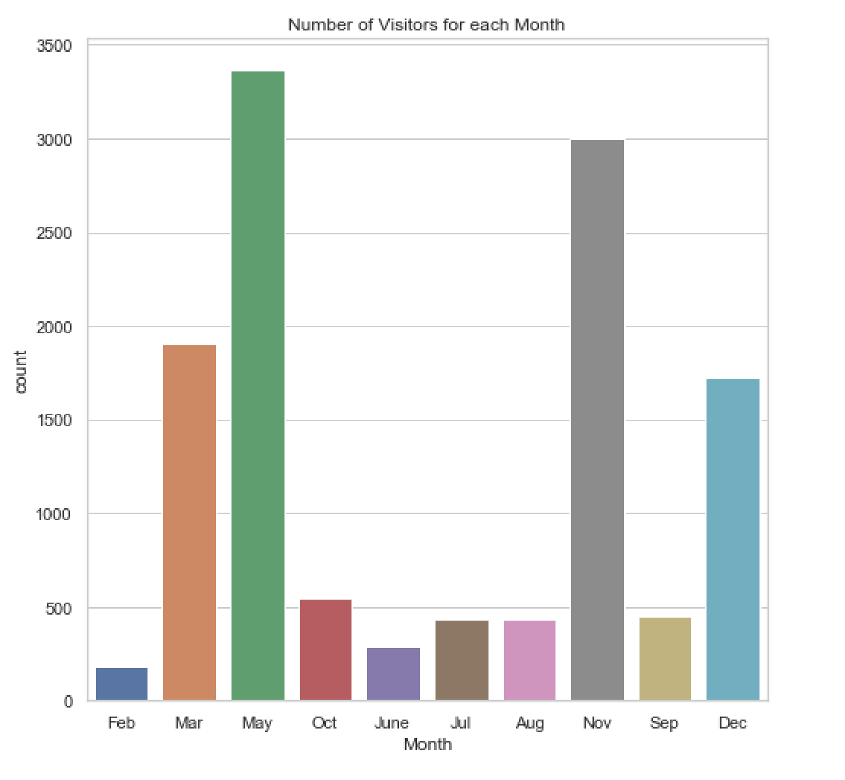
	attribute pair	correlation
3	(BounceRates, ExitRates)	0.913004
2	(ProductRelated, ProductRelated_Duration)	0.860927
1	(Informational, Informational_Duration)	0.618955
0	(Administrative, Administrative_Duration)	0.601583

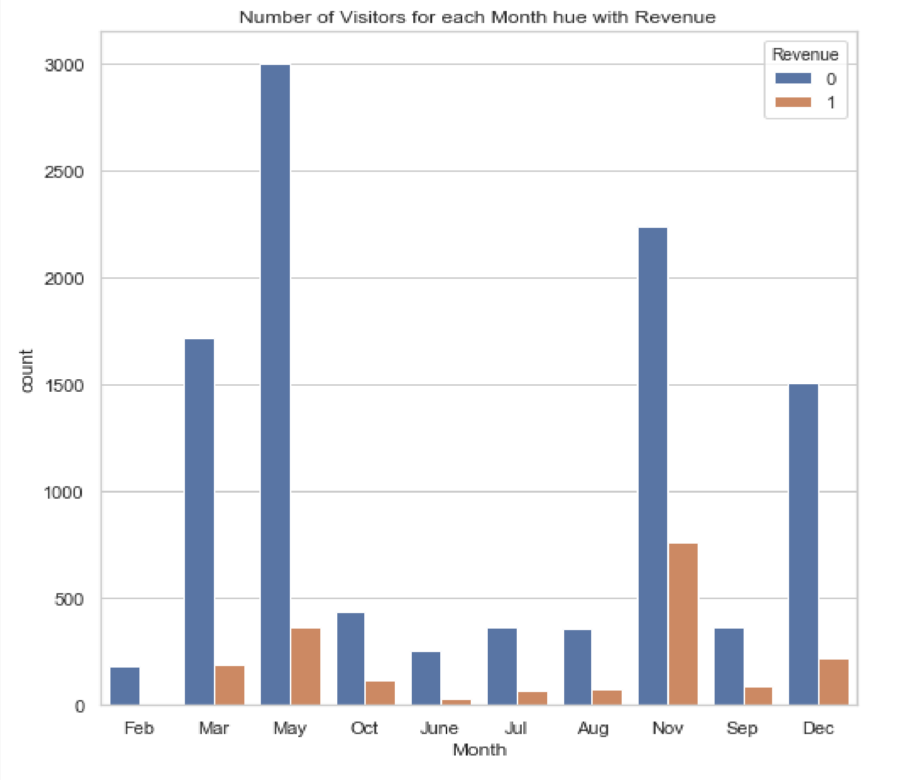


As we said earlier, the biggest correlations are between **BounceRates with ExitRates**, **and ProductRelated with ProductRelated_Duration.** We can cleary see the correlations in the plots !

Correlation between Informational with Informational_Duration and Administrative with Administrative_Duration **exists**, but are less significants.

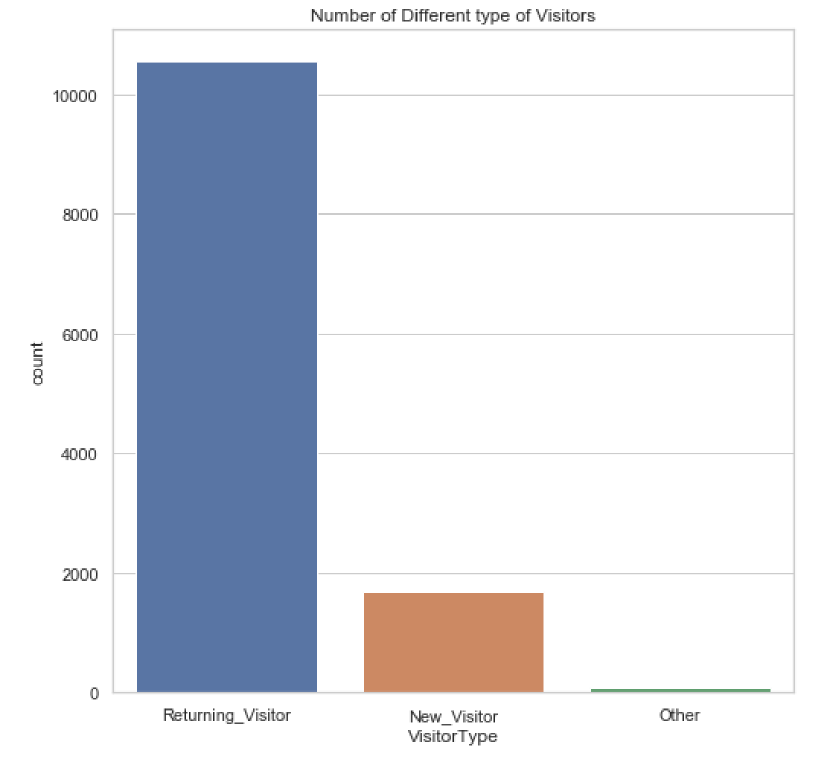
7) REVENUE DEPENDING ON MONTH

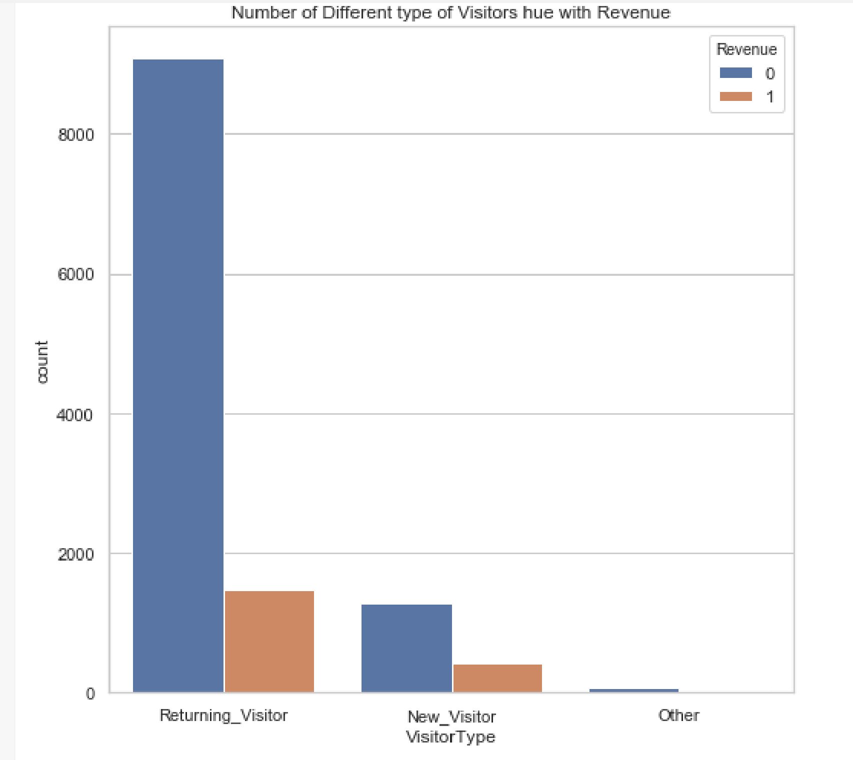




This graph is intersting, because we see that May is the month where there was the biggest number of customer, but it is in November (the second bigger) that there was the biggest number of bought.

8) REVENUE DEPENDING ON VISITOR TYPE





This graph is not very interesting, because we see that the proportions between 0 and 1 in Revenue are quite the same even if the customer is a Returning visitor or a New visitor.

MACHINE LEARNING

DATA PREPROCESSING ML

```
data_shop['Revenue'] = data_shop['Revenue']*1
data_shop['Weekend'] = data_shop['Weekend']*1
data_shop
```

```
data_shop = pd.get_dummies(data_shop,columns=['Month','VisitorType','Browser','OperatingSystems','Region','TrafficType','We
data_shop.info()
```

After processing our Data like we did before, we're gonna scaled it !

```
data_shop_target = data_shop['Revenue']
data_shop_attributes = data_shop.drop(columns=['Revenue'])
scaler = StandardScaler()
data_shop_scaled = StandardScaler().fit_transform(data_shop_attributes)
print(data_shop_scaled)
col_names = list(data_shop.columns)
col_names.pop(15)
print(col_names)
data_shop_final = pd.DataFrame(data=data_shop_scaled, columns=col_names)

data_shop_final
```

And for the study, we're gonna split in in two differents data set:
- one for the training
- one for the testing

```
train_X,test_X,train_y,test_y = train_test_split(data_shop_final, data_shop_target, random_state=88, train_size=0.2)
```

1) GAUSSIAN NB

We create the Gaussian model.

```
model_gauss = GaussianNB()
model_gauss.fit(train_X, train_y)
predi_y_gauss = model_gauss.predict(test_X)
```

The accuracy of our model is very low ; this model is not adapted to our study at all.

```
: accu_gauss = accuracy_score(test_y,predi_y_gauss)
  accu_gauss

0.2663219789132198
```

Confusion Matrix :

	False	True
real_False	1100	7221
real_True	16	1527

2) LOGISTIC REGRESSION

We create the Logistic Regression model.

```
model_regression = LogisticRegression()
model_regression.fit(train_X, train_y)
pred_y = model_regression.predict(test_X)
```

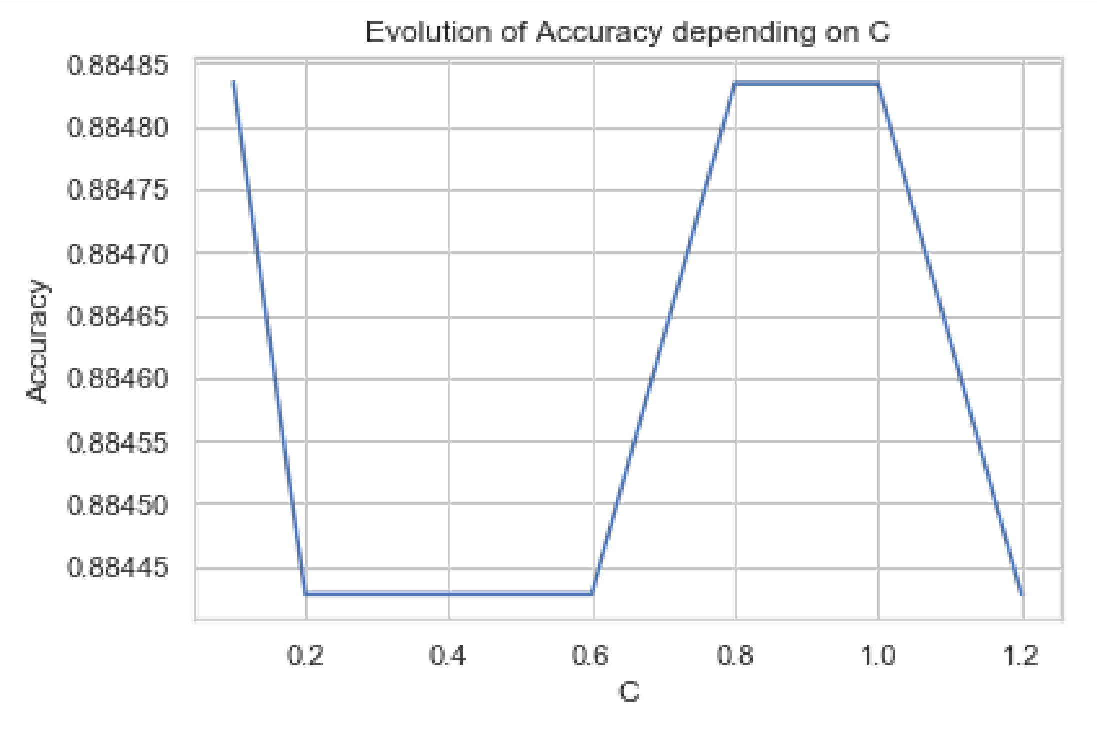
The accuracy of this model is quite good. Let's see with a **GridSearch how we can increase it by tuning the hyperparameters.**

```
: accuracy_score(test_y, pred_y)

0.8788523925385239
```

```
: grid_values = {'C': [0.1,0.2,0.4,0.6,0.8,1,1.2]}
  model_lr_grid = GridSearchCV(model_regression, param_grid=grid_values,scoring='accuracy')
  model_lr_grid

GridSearchCV(cv=None, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 0.2, 0.4, 0.6, 0.8, 1, 1.2]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

We can see that our best parameter **is C = 0.1** ; it gives us an accuracy of 0.885 !

```
: model_lr_grid.fit(train_X, train_y)
  print(model_lr_grid.best_params_)
  print(model_lr_grid.best_score_)

{'C': 0.1}
0.8848346486437657
```



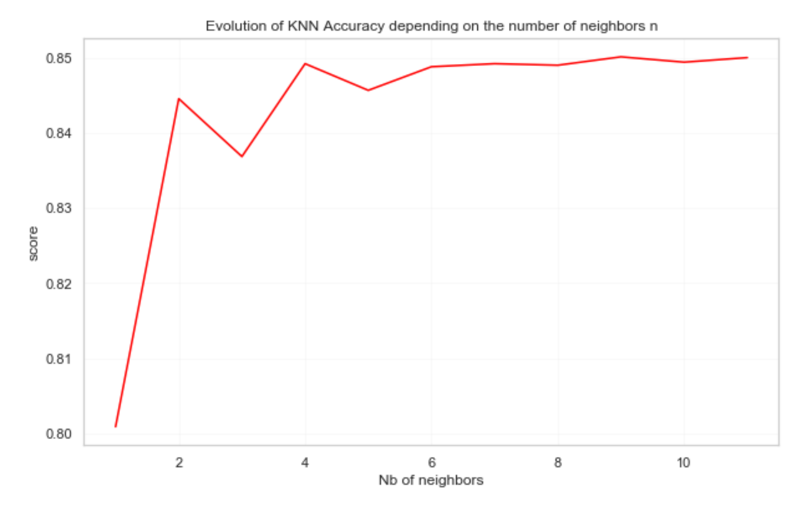
Confusion Matrix with best params :

	False	True
real_False	8160	161
real_True	1038	505

3) KNN METHOD

We create the KNN model.

```
# n_neighbors is the hyperparameter
n_neighbors_list = np.arange(1,12)
scores = []
for n in n_neighbors_list:
    model_knn = KNeighborsClassifier(n_neighbors=n)
    model_knn.fit(train_X, train_y)
    y_model_knn = model_knn.predict(test_X)
    scores.append(accuracy_score(test_y, y_model_knn))
```



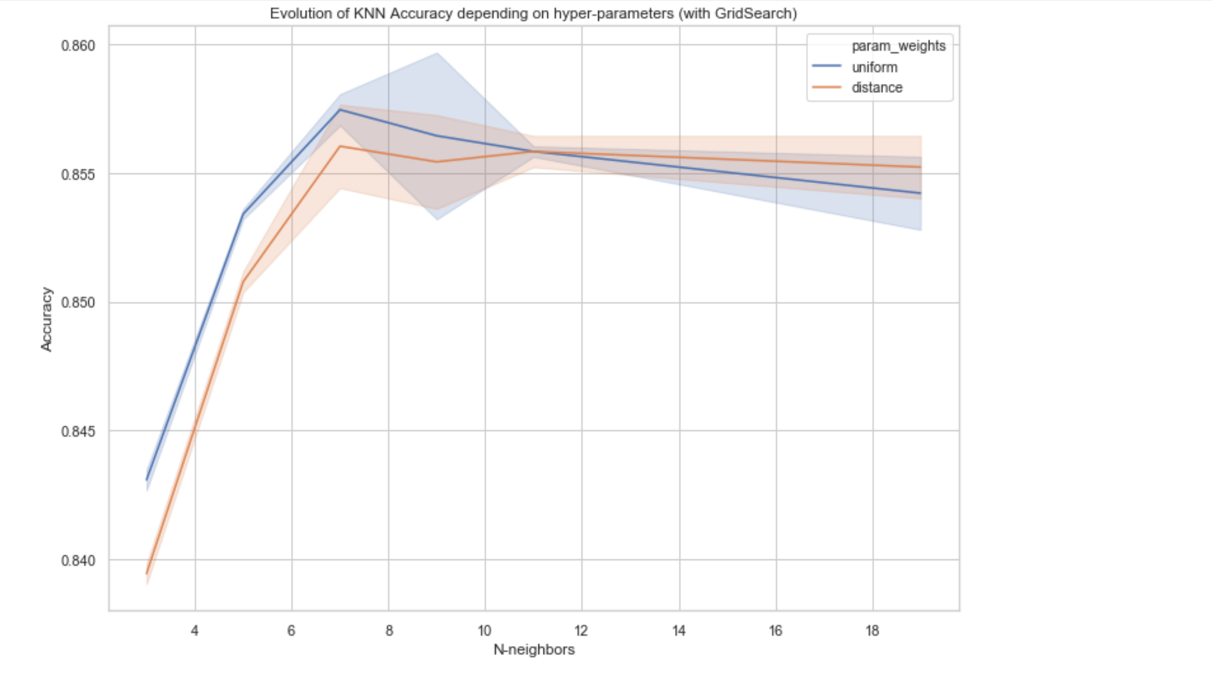
We see that we obtain the best accurcay
with a number of neighbors equal to **9**.

```
Best accuracy : 0.8501622060016221
This max is obtain with nb_neighbors = 9
```

But there are other parameters to tune if we want to increase the accuracy of a KNN model ; let's launch a GridSearch.

```
grid_param_knn = {
    'n_neighbors': [3,5,7,9,11,19],
    'weights':['uniform','distance'],
    'metric':['euclidean','manhattan']
}

gs_knn = GridSearchCV(KNeighborsClassifier(),
                      grid_param_knn,
                      verbose=1,
                      cv=4,
                      n_jobs=-1
                      )

gs_knn_results = gs_knn.fit(train_X,train_y)
```

```
Fitting 4 folds for each of 24 candidates, totalling 96 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=-1)]: Done  38 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Done  96 out of  96 | elapsed:    3.3s finished
```




```
[46]: print('Best accuracy :', gs_knn_results.best_score_)
      print('This max is obtain with these estimators :',gs_knn_results.best_estimator_)

Best accuracy : 0.8596936896167042
This max is obtain with these estimators : KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=9, p=2,
                     weights='uniform')
```

So the best estimators are :
- algorithm='auto'
- leaf_size=30
- metric='manhattan'
- metric_params=None
- n_jobs=None
- n_neighbors=9
- p=2
- weights='uniform'

Confusion Matrix with best params :

	False	True
real_False	8286	35
real_True	1462	81

4) DECISION TREE

We create the Decision Tree model.

```
model_tree = DecisionTreeClassifier(random_state = 88)
model_tree.fit(train_X, train_y)

predi_tree = model_tree.predict(test_X)
```

```
accuracy_score(predi_tree, test_y)
```

```
0.8587793998377939
```

Let's tune the hyperparameters.

```
param_grid_tree = {
    'max_depth': [1,2,3,4,5,6,7,8,9,10],
    'min_samples_leaf': [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.08,0.1]
}

gs_tree = GridSearchCV(estimator = DecisionTreeClassifier(random_state = 88),
                       param_grid = param_grid_tree, scoring = 'accuracy', cv = 7)
```

So the best estimaotor for our Decision Tree are :
- max_depth = 5
- min_samples_leaf = 0.01
- ccp_alpha=0.0,
- class_weight=None,
- criterion='gini’
- max_depth=5
- max_features=None
- max_leaf_nodes=None
- min_impurity_decrease=0.0
- min_impurity_split=None
- min_samples_leaf=0.01
- min_samples_split=2
- min_weight_fraction_leaf=0.0
- presort='deprecated’
- random_state=88
- splitter='best'

```
print(gs_tree_results.best_score_)
print(gs_tree_results.best_params_)
print(gs_tree_results.best_estimator_)
```

```
0.8990218351054045
{'max_depth': 5, 'min_samples_leaf': 0.01}
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=0.01, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=88, splitter='best')
```

Confusion Matrix with best params :

	False	True
real_False	7917	404
real_True	669	874

5) RANDOM FOREST

We create the Random Forest model.

```
model_rf = RandomForestClassifier(random_state = 88)
model_rf.fit(train_X, train_y)

predi_rf = model_rf.predict(test_X)
```

```
accuracy_score(predi_rf, test_y)
```

```
0.8915247364152473
```

Let's tune the hyperparameters.

```
param_grid_rf = {
     'max_features': [0.1,0.2,0.3,0.5,0.8,1,2],
     'min_samples_leaf': [1,2,3, 4, 5,6,7],
     'min_samples_split': [8, 10, 12,15,17,20],
     'n_estimators': [100, 200, 300, 400,500]
}
rf = RandomForestClassifier()
gs_rf = GridSearchCV(estimator = rf, param_grid = param_grid_rf,
                     cv = 4, n_jobs = -1, verbose = 2,scoring='accuracy')
```

```
gs_rf_results = gs_rf.fit(train_X,train_y)
```

```
'itting 4 folds for each of 1470 candidates, totalling 5880 fits
```

```
Parallel(n_jobs=-1)]: Using backend LokyBackend with 6 concurrent workers.
Parallel(n_jobs=-1)]: Done  29 tasks      | elapsed:    4.3s
Parallel(n_jobs=-1)]: Done 150 tasks      | elapsed:   17.8s
Parallel(n_jobs=-1)]: Done 353 tasks      | elapsed:   40.3s
Parallel(n_jobs=-1)]: Done 636 tasks      | elapsed:  1.2min
Parallel(n_jobs=-1)]: Done 1001 tasks      | elapsed:  1.9min
Parallel(n_jobs=-1)]: Done 1446 tasks      | elapsed:  2.9min
Parallel(n_jobs=-1)]: Done 1973 tasks      | elapsed:  4.2min
Parallel(n_jobs=-1)]: Done 2580 tasks      | elapsed:  5.9min
Parallel(n_jobs=-1)]: Done 3269 tasks      | elapsed:  8.3min
Parallel(n_jobs=-1)]: Done 4038 tasks      | elapsed: 12.0min
Parallel(n_jobs=-1)]: Done 4889 tasks      | elapsed: 13.6min
Parallel(n_jobs=-1)]: Done 5820 tasks      | elapsed: 14.9min
Parallel(n_jobs=-1)]: Done 5880 out of 5880 | elapsed: 15.0min finished
```

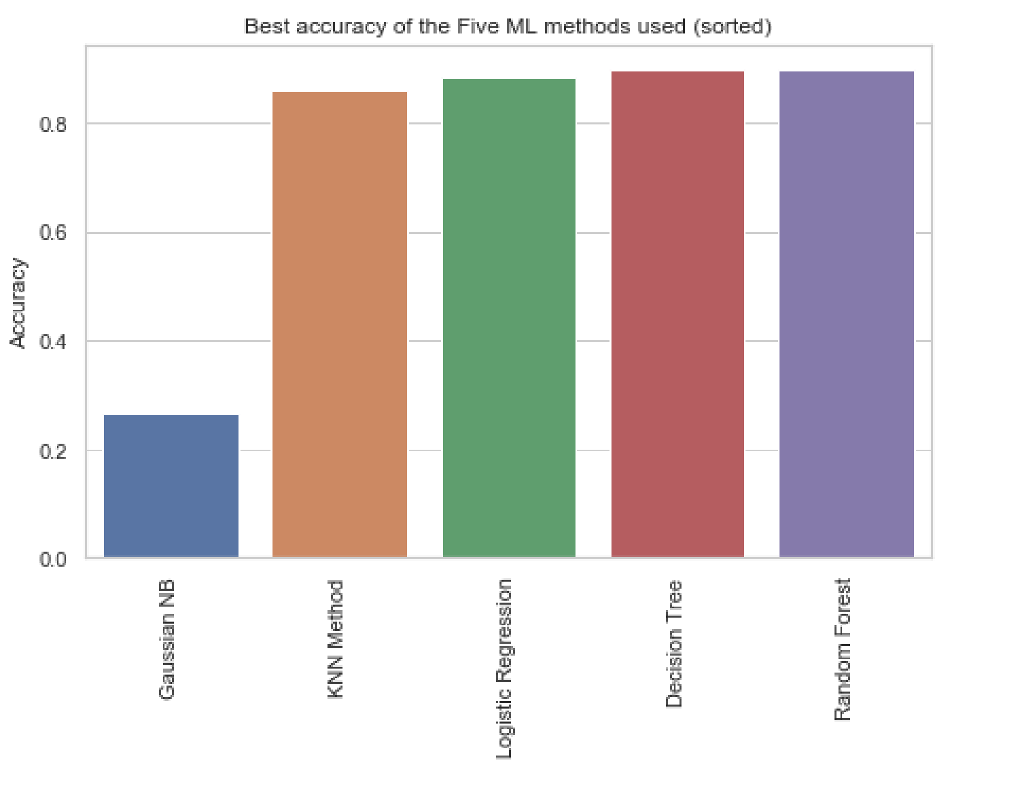
```
: print(gs_rf_results.best_score_)
  print(gs_rf_results.best_params_)

0.8998380569997264
{'max_features': 0.2, 'min_samples_leaf': 3, 'min_samples_split': 10, 'n_estimators': 400}
```

So the best parameters for our Random Forest are :
- max_features = 0.2
- min_samples_leaf = 3
- min_samples_split = 10
- n_estimators = 400

Confusion Matrix with best params :

	False	True
real_False	8119	202
real_True	772	771

BEST ACCURACY



The best model is our **Random Forest model !** We're gonna take him for our Flask API.

```
pickle.dump(best_model_rf, open('rf_customer.pkl','wb'))
```

FLASK API

HOW TO RUN IT

```
leo@leoasus: ~/FlaskApi
leo@leoasus: ~/FlaskApi 204x55
leo@leoasus:~$ cd FlaskApi/
leo@leoasus:~/FlaskApi$ python3 app.py
/home/leo/.local/lib/python3.6/site-packages/sklearn/base.py:334: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 0.22.1 when using version 0.23.2. This might lead to breakin
g code or invalid results. Use at your own risk.
  UserWarning)
/home/leo/.local/lib/python3.6/site-packages/sklearn/base.py:334: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.22.1 when using version 0.23.2. This might lead to breakin
g code or invalid results. Use at your own risk.
  UserWarning)
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
/home/leo/.local/lib/python3.6/site-packages/sklearn/base.py:334: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 0.22.1 when using version 0.23.2. This might lead to breakin
g code or invalid results. Use at your own risk.
  UserWarning)
/home/leo/.local/lib/python3.6/site-packages/sklearn/base.py:334: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.22.1 when using version 0.23.2. This might lead to breakin
g code or invalid results. Use at your own risk.
  UserWarning)
 * Debugger is active!
 * Debugger PIN: 168-608-177
```

Just go to your localhost port 5000 !
(or http://127.0.0.1:5000/)